UNIVERSITÉ DE MONTRÉAL

MODÉLISATION DU CALCUL DISTRIBUÉ POUR DES PROBLÈMES DE TRANSPORT DE NEUTRONS À GRANDE ÉCHELLE

MOHAMED DAHMANI
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
MAI 2005

Your file Votre référence
ISBN: 978-0-494-16989-6
Our file Notre référence
ISBN: 978-0-494-16989-6

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée:

MODÉLISATION DU CALCUL DISTRIBUÉ POUR DES PROBLÈMES DE TRANSPORT DE NEUTRONS À GRANDE ÉCHELLE

présentée par: DAHMANI Mohamed
en vue de l'obtention du diplôme de: Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de:

M. PIERRE Samuel, Ph.D., président
M. ROY Robert, Ph.D., membre et directeur de recherche
M. KOCLAS Jean, Ph.D., membre et codirecteur de recherche
M. BERTRAND François, Ph.D., membre
M. AZMY Yousry Y., Ph.D., membre

À ma femme Sara

À ma fille Meriam

REMERCIEMENTS

Je tiens d'abord à remercier tout particulièrement et à exprimer ma reconnaissance à mon directeur de recherche professeur Robert Roy pour ses encouragements, son soutien et sa disponibilité. Je le remercie aussi pour les nombreuses discussions que nous avons eues et qui m'ont été très utiles.

j'aimerais remercier également mon codirecteur de recherche professeur Jean Koclas pour sa bonne collaboration et pour l'interêt qu'il a manifesté à l'égard de ce travail.

Je remercie professeur Samuel Pierre d'avoir accépté de présider le jury de cette thèse et pour ces nombreux conseils concernant la rédaction de ce document.

Je remercie également professeur François Bertrand et professeur Yousry Azmy de l'université de Penn State d'avoir acceptés à faire partie du jury.

Merci à Benoît Morin et Louis-Alexandre Leclaire pour leur soutien technique.

Enfin, mes remerciements vont également à tous les professeurs, chercheurs et étudiants de l'institut de génie nucléaire à l'École Polytechnique de Montréal.

RÉSUMÉ

La conception des nouvelles générations de réacteurs nucléaires, utilisant des représentations toujours plus fines exige des codes de simulation pouvant s'adapter à l'augmentation de la taille et de la complexité des systèmes à traiter et par le développement d'algorithmes de résolution plus rapides. La nécessité de résoudre des problèmes de transport de neutrons de tailles de plus en plus importantes est devenue une réalité incontournable pour l'ingénieur nucléaire.

L'objectif principal de cette thèse est la modélisation du calcul réparti dans un contexte de résolution de problèmes de transport de neutrons à grande échelle. Nous proposons ainsi une nouvelle approche informatique pour résoudre ces problèmes afin d'améliorer substantiellement les capacités de traitement de la chaîne de calcul de réacteurs.

Dans le cadre des applications utilisant des techniques de calcul de haute performance et plus particulièrement le calcul parallèle, les études de performances sont indispensables. Elles doivent permettre de prévoir l'évolution de la performance d'une application donnée sur des machines parallèles données. L'analyse en terme de l'évolutivité permet aussi de prévoir les performances du système parallèle sur des machines massivement parallèles (composée de milliers de processeurs) en utilisant des tests sur des machines avec seulement quelques dizaines de processeurs. En effet, l'étude analytique de l'évolutivité du système parallèle peut nous aider à identifier les paramètres importants du problèmes : la combinaison algorithme-architecture, l'effet d'augmenter le nombre de processeurs sur la performance en particulier le nombre optimal de processeurs à utiliser pour résoudre le problème, l'accélération maximale et enfin l'impact des paramètres physiques propres à chaque machine. Une grande partie des travaux de cette thèse se consacre à cette étude. Notre modèle d'ailleurs a été testé pour analyser l'évolutivité de l'algorithme parallèle utilisé par le solveur sur deux machines parallèles différentes en variant la taille du problème.

Pour éliminer des problèmes liés aux applications à grande échelle, un nouveau solveur nommé *MCG* a été conçu. Ce solveur, basé sur la méthode des caractéristiques en 3D avec des sections efficaces de diffusion isotropes et linéairement anisotropes, utilise une nouvelle méthodologie de calcul. Les lignes d'intégration ne sont plus alors stockées dans des fichiers séquentiels, mais sont générées au fur et à mesure que le calcul se déroule. Quoique moins rapide, ce solveur a l'avantage de n'avoir aucune limitation d'entrée/sortie.

Cette thèse est présentée sous forme de quatre articles. Après l'introduction du modèle analytique de performances, celui-ci est utilisé pour étudier essentiellement l'impact des paramètres réseau sur le temps de communications et l'évolutivité de l'application parallèle. La méthode d'accélération *GMRES* préconditionnée est aussi présentée. Dans l'annexe, nous avons regroupé d'autres publications complémentaires présentés dans différentes conférences internationales. Ces articles traitent de la parallélisation du solveur, l'étude des performances (sans utiliser le modèle analytique) en terme d'accélération sur différentes machines avec différents réseaux d'interconnexion, les techniques d'équilibrage de charges et les techniques utilisées pour stocker et accéder aux données liées aux lignes d'intégration. L'introduction du nouveau solveur des caractéristiques *MCG* et le traitement de l'anisotropie linéaire sont aussi abordés. L'ensemble représente un apport significatif à la conception et l'étude d'algorithmes parallèles pour le transport neutronique moderne.

ABSTRACT

The design of new generations of nuclear reactors will involve fine representations of the theoretical models. Advanced computational methods capable to solve large scale problems treating large and complex systems are required. Therefore, the solution of challenging large scale neutron transport problems is becoming more and more pressing in nuclear engineering applications.

The main objective of this thesis is the modelization of the distributed neutron transport calculation in the context of large scale problems. A performance model is used for scalability analysis. A new computational approach is then proposed in order to considerably enhance the processing efficiency of the reactor calculation procedure.

For the applications using high performance computing techniques and especially parallel computing, performance studies are crucial. These should predict the performance evolution for a given application on a given parallel machines. Scalability analysis may be used to predict the performance of a parallel algorithm and a parallel architecture for a large number of processors from the known performance on fewer processors. An analytical study of the scalability of the parallel system can help in identifying the best algorithm-architecture combinations, the effect of an increasing number of processors on performance, the optimal number of processors to use for solving the problem and the impact of the machine parameters. A large part of this thesis deals with such study. Our model was tested to analyze the scalability of the parallel algorithm used by the solver on two different machines by varying the problem size.

In order to get rid of some problems related to the large scale applications, a new solver called *MCG* was developed. This solver, based on 3D characteristics method with isotropic and linearly anisotropic scattering cross sections, uses a new computational methodology. The huge capacity of storage needed in large problems and the related I/O queries needed by the characteristics solver are replaced by on-the-fly recalculation of tracks at each iteration step. Even though this solver requires more CPU resources, large 3D problems are now no longer I/O-bound.

This thesis is composed of four articles. After the introduction of the analytical performance model, this is used to study essentially the impact of network parameters on the communication time and the scalability of the parallel application. The preconditioned *GMRES* acceleration method is also presented. In the annex, we gather five other publications presented in different international conferences. These articles treat the parallelization of the solver, performance studies (without using the model) in terms of speedup on different machines with different interconnection networks, load balancing techniques and techniques used to store and to access to the tracking lines. The introduction of *MCG* solve and the extension of the solver to take into account the anisotropic effects are also discussed. The collection of these works represents significant contribution to the design and the analysis of the parallel algorithms in modern neutron transport applications.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ANNEXES

CHAPITRE 1

INTRODUCTION GÉNÉRALE

Avec l'évolution rapide des technologies avancées liées aux moyens de calcul et au traitement de l'information, le calcul scientifique (*computational science*) a un impact de plus en plus important sur l'ingénierie, la physique, la chimie et même les sciences économiques et sociales. Le but du calcul scientifique est de développer et analyser des modèles mathématiques sophistiqués pour simuler et contrôler des processus complexes, tout en optimisant l'utilisation de la puissance de calcul afin de résoudre des problèmes de grande taille. Ceci nécessite des outils théoriques et algorithmiques liés à differentes disciplines : équations aux dérivées partielles, développement de logiciels, architectures des ordinateurs, parallélisation et calcul distribué. Ainsi, cette approche (*multidisciplinaire*) englobe plusieurs domaines de recherche fondamentaux et appliqués.

Une partie importante de ces applications demandent de très gros volumes de calculs. Les exigences des scientifiques ont toujours été supérieures aux possibilités fournies par les ordinateurs du moment. Chaque nouvelle génération de machine a permis des avancées significatives, mais celles-ci se sont heurtées, à leur tour, aux nouvelles limites. Les avancements dans ces différents domaines scientifiques ont souvent été liés à la puissance de calcul disponible, et ce n'était pas un hasard si les meilleures équipes étaient implantées à proximité des super-calculateurs. La disponibilité de la puissance maximum est indispensable, non seulement pour calculer des modèles réalistes mais aussi pour mener ces simulations en un temps raisonnable. La météorologie serait incapable de nous fournir des prévisions au jour le jour sans ces machines. Pour obtenir en un temps raisonnable une précision de calcul respectable, il faut donc des ordinateurs très rapides avec de grandes mémoires pour stocker les données lors du calcul. Depuis environ cinquante ans, la vitesse des ordinateurs séquentiels a augmenté d'environ un facteur d'un million, mais ils sont toujours trop lents pour beaucoup d'applications scientifiques actuelles. Le calcul de haute performance, en particulier le traitement parallèle, peut alors apporter une solution à ce problème. Le calcul parallèle utilise la devise 'partager et conquérir' en faisant travailler simultanément un certain nombre de processeurs. Cela ne va pas sans poser des problèmes, en particulier le problème classique dans l'organisation du travail collectif : la coordination des efforts réalisés par chacune des parties. Pour un ordinateur parallèle, il faut que les processus puissent communiquer rapidement leurs données et/ou résultats. Ce type d'opérations s'effectue par l'intermédiaire d'un réseau d'interconnexions.

1.1 Définitions et concepts de base

Introduisons, maintenant, brièvement notre application. La conception d'un réacteur nucléaire à fission est basée sur le principe de la réaction en chaîne contrôlée: la fission des noyaux lourds du combustible étant provoquée par l'absorption des neutrons issus des fissions précédentes. Ces fissions sont accompagnées d'un dégagement important d'énergie qui est déposée dans le combustible. Dans les centrales nucléaires de puissance, cette énergie est extraite du réacteur par un fluide caloporteur qui la transporte au générateur de vapeur. La vapeur ainsi créée est ensuite livrée à la turbine pour entraîner un turboalternateur en vue de la production d'électricité.
Un réacteur nucléaire est en fonctionnement normal lorsqu'il est au voisinage de la criticité. L'état critique provient de l'équilibre qui existe entre deux processus dynamiques : la création et l'élimination des neutrons. Un léger écart entre le taux d'élimination et celui du production peut mener à une variation importante de la population de neutrons (et donc du taux de fission). Le calcul de réacteur nucléaire est basé sur la *théorie de transport* de neutrons. Le début de la théorie de transport remonte à plus d'un siècle quand Boltzmann l'a formulée dans le cadre d'étude de

la cinétique des gas (Boltzmann, 1912). Cette théorie décrit mathématiquement le mouvement de particules dans un milieu physique donné. Ces particules peuvent être des neutrons, des ions, des électrons, des molécules ou des quanta (photons ou phonons). Les milieux en question peuvent être aussi variés qu'un coeur de réacteur nucléaire, une étoile, ou encore un plasma thermonucléaire. En effet, le but de la théorie de transport est de déterminer la distribution des particules dans le milieu en tenant compte de leur mouvement et de leur interaction avec les noyaux atomiques du milieu. La modélisation mathématique de ces phénomènes est gouvernée par l'équation de transport appelée aussi *l'équation de Boltzmann.* La quantité à déterminer est la densité de particules dans l'espace des phases. Cette densité est définie par :

$$n(\vec{r}, \hat{\Omega}, E, t)\ d^3r\ d^2\Omega\ dE \tag{1.1}$$

où $n(\vec{r}, \hat{\Omega}, E, t)$ est la densité angulaire qui exprime le nombre de particules se trouvant dans le volume d^3r autour de $\vec{r}$ voyageant dans un cône de base $d^2\Omega$ dans la direction $\hat{\Omega}$ avec des énergies comprises entre E et $E+dE$ au temps t. Dans sa forme générale, et si on considère que les particules sont des neutrons dans un réacteur, l'équation de Boltzmann peut être formulée de la façon suivante :

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\Phi(\vec{r}, \hat{\Omega}, E, t) + \hat{\Omega}\cdot\vec{\nabla}\Phi(\vec{r}, \hat{\Omega}, E, t) + \Sigma_t(\vec{r}, E)\Phi(\vec{r}, \hat{\Omega}, E, t) &= \\ \int_0^\infty dE' \int_{4\pi} d^2\Omega'\ \Sigma_s(\vec{r}, \hat{\Omega} \leftarrow \hat{\Omega}', E \leftarrow E')\Phi(\vec{r}, \hat{\Omega}', E', t)& \\ +\chi(\vec{r}, E)\ \int_0^\infty dE'\ \nu\Sigma_f(\vec{r}, E') \int_{4\pi} d^2\Omega'\ \Phi(\vec{r}, \hat{\Omega}', E', t)&\end{aligned} \tag{1.2}$$

Σ_t, Σ_s, et Σ_f sont respectivement la section efficace totale, la section efficace de diffusion et la section efficace de fission. Ces quantités sont directement reliées à la probabilité qu'une réaction nucléaire (absorption, fission, diffusion élastique, diffusion inélastique...) ait lieu. ν est le nombre secondaire de neutrons générés par la fission, χ est le spectre en énergie des neutrons émis par la fission. v est la vitesse des neutrons. Le flux angulaire $\Phi(\vec{r}, \hat{\Omega}, E, t)$ est lié à la densité angulaire par :

$$\Phi(\vec{r}, \hat{\Omega}, E, t) = vn(\vec{r}, \hat{\Omega}, E, t) \tag{1.3}$$

L'équation (1.2) est une équation intégro-différentielle avec comme inconnu le flux angulaire.

La complexité de cette équation nous force à utiliser des méthodes numériques. En général, ces méthodes introduisent des approximations de telle sorte que l'équation intégro-différentielle soit transformée en un système d'équations algébriques que l'on peut résoudre sur ordinateur. Les méthodes numériques les plus utilisées sont: la méthode S_N (Carlson, 1968; Lewis, 1984), la méthode des probabilités de collision (Roy, 2003; Sanchez, 1999) et la méthode des caractéristiques (Askew, 1972; Halsall, 1998). C'est cette dernière méthode qui est utilisée dans ce travail car elle ne nécessite aucun traitement matriciel et par conséquent une capacité de stockage réduite comparée à la méthode des probabilités de collision.

La grande majorité de calculs numériques en transport utilisent l'équation stationnaire (*indépendante du temps*). Le domaine énergétique est discrétisé, selon le formalisme *multi-groupe*, en G intervalles appelées groupes d'énergie. L'opérateur de diffusion dépend généralement de l'angle de déviation $\mu_0 = \hat{\Omega}.\hat{\Omega}'$ du neutron après collision. Pour un calcul de bilan, souvent on ajuste ν pour avoir une solution indépendante du temps. On remplace alors ν par ν/K_{eff} introduisant ainsi le problème multi-groupe aux valeurs propres suivant :

$$\begin{aligned} \left(\hat{\Omega}\cdot\vec{\nabla} + \Sigma^g(\vec{r})\right)\Phi^g(\vec{r},\hat{\Omega}) &= \\ \frac{\chi^g}{K_{\text{eff}}}\sum_{g'=1}^{G}\nu\Sigma_f^{g'}(\vec{r})\int_{4\pi} d^2\Omega'\;\Phi^{g'}(\vec{r},\hat{\Omega}') + \sum_{g'=1}^{G}\int_{4\pi} d^2\Omega'\;\Sigma_s^{g\leftarrow g'}(\vec{r},\hat{\Omega}.\hat{\Omega}')\;\Phi^{g'}(\vec{r},\hat{\Omega}') & \end{aligned} \tag{1.4}$$

où g est l'indice sur les groupes d'énergie et K_{eff} est le facteur de multiplication effectif. Le réacteur est critique si $K_{\text{eff}} = 1$. La valeur propre dominante et le vecteur propre correspondant sont déterminés, d'habitude pour les problèmes de criticité, en utilisant les méthodes itératives classiques comme la méthode des puissances. Finalement, à l'équation (1.4) on rajoute des conditions aux frontières adéquates pour fermer le système.

1.1.1 Méthode des caractéristiques

En général, la méthode des *caractéristiques* a pour but de résoudre une équation différentielle aux dérivées partielles en utilisant des courbes dans l'espace 3D de telle sorte que ces équations sont transformées en équations différentielles ordinaires. En effet, toute courbe dans l'espace peut être exprimée sous forme paramétrique $\vec{r} = \vec{r}(s)$ où le paramètre s mesure la distance sur la courbe. La courbe commence à n'importe quel point $\vec{p}$. Supposons que la solution de l'équation est connue partout sur la courbe, un autre choix du point de départ $\vec{p}$ aboutit à une autre courbe et les valeurs des inconnues sont alors déterminées sur cette courbe et ainsi de suite jusqu'à la construction de la solution finale.

Dans le cas de l'équation de transport de neutrons, la méthode des caractéristiques est utilisée pour transformer *l'opérateur de transport* en un opérateur différentiel exact que nous appelons *l'opérateur des caractéristiques.* Les courbes, dans ce cas, sont des lignes droites représentant les trajectoires des neutrons dans le domaine et s mesure la distance locale sur ces lignes. Ces lignes sont appelées des *lignes d'intégration* ou les *caractéristiques.*

1.2 Élément de la problématique

L'équation de transport dépend donc de l'espace, de l'énergie des neutrons, de l'angle solide et du temps. Cette dépendance ajoutée à la complexité géométrique et matérielle des réacteurs de puissance interdit que l'on puisse obtenir en un seul calcul de bilan tous les flux neutroniques requis pour la détermination de la distribution de puissance. L'approche utilisée consiste à représenter le réacteur comme un assemblage de '*cellules unitaires*' disposées sur un réseau régulier rectangulaire ou hexagonal. Les cellules unitaires formant le réacteur sont supposées géométriquement identiques et se différencient seulement par un ensemble de paramètres locaux tels que la composition matérielle et le niveau de burnup. Dans la pratique, on résout l'équation de transport sur une ou plusieurs cellules. La distribution du flux, ainsi obtenue, est utilisée pour homogénéiser en espace et condenser en énergie (*calcul de cellule*). Les propriétés nucléaires condensées et homogénéisées peuvent être ensuite utilisées dans le calcul de la distribution du flux dans tout le coeur à l'aide d'un modèle approché comme celui de la diffusion qui est une approximation classique de transport (*calcul de réacteur ou de coeur*). (Glasstone, 1991; Lewis, 1984)

L'évolution actuelle au niveau de la conception tend vers la construction de réacteurs nucléaires toujours plus puissants. Cette croissance de la puissance engendre une plus grande instabilité du coeur du réacteur. Il devient donc essentiel de perfectionner la simulation du comportement des réacteurs nucléaires et ceci à trois niveaux:

- la précision issue de l'équation de la diffusion n'est plus suffisante. Les simulations doivent traiter les approximations du transport plus générales;
- l'amélioration de la simulation porte aussi sur la précision des méthodes numériques utilisées. Les géométries sont de plus en plus complexes et les maillage de calcul sont de plus en plus fins;
- le dernier progrès porte sur la réduction du temps de calcul nécessaire à la

simulation d'un coeur de réacteur.

Ces trois changements se traduisent au niveau des codes de simulation par une augmentation de la taille et de la complexité des systèmes à traiter et par le développement d'algorithmes de résolution plus rapides. En effet, un calcul de transport typique, avec des nombres de régions et de groupes d'énergie de l'ordre de la centaine chacun, génère un système comportant quelques dizaines de milliers d'inconnues. Afin de valider les données nucléaires de sections efficaces, il faudrait accroître le nombre de groupes d'énergie, qui passerait alors d'une centaine à quelques dizaines de milliers. Pour un domaine spatial relativement large (comme un assemblage), il faut également compter des milliers de régions. Avec ces quantités de données, le calcul sur un ordinateur séquentiel est pratiquement impossible. Dans ce cas, pour résoudre des problèmes de grande taille générant un nombre extrêmement grand de données, l'utilisation des techniques de *calcul de haute performance* est nécessaire. Les connaissances algorithmiques du calcul parallèle et réparti, nous conduisent à identifier les problèmes à résoudre en trois catégories:

1. *informatiques* :
   (a) Gestion d'énormes bases de données générées par la procédure d'intégration '*ray tracing*' : problèmes de stockage de données et les problèmes d'entrée/-sortie 'E/S' dans l'environnement parallèle;
   (b) Équilibrage de charges : répartition des données et des tâches sur différents processeurs. Ces problèmes deviennent plus sérieux, surtout quand on utilise des machines avec des architectures hétérogènes;
   (c) Gestion des communications et synchronisation entre les processus;
   (d) évolution de la performance de l'algorithme parallèle en fonction de la performances des machines utilisées en terme de puissance de calcul et de la vitesse de communication (performance de réseaux d'interconnexion);

   (e) évolutivité (scalability) : comment maintenir la performance de l'algorithme parallèle au maximum en terme d'efficacité ou d'accélération (speed up) quand on augmente considérablement la taille de la machine (nombre de processeurs) et/ou la taille du problème.

2. *numériques* :

   (a) développement et implémentation de méthodes d'accélération adaptées aux problèmes de grande taille qui présentent souvent une forte hétérogéinété spatiale avec des discontinuités matérielles;

   (b) optimisation du solveur.

3. *validation* : avoir accès à des benchmarks et des solutions avec d'autres codes afin de servir de références.

1.3 Travaux antérieurs: Revue de la littérature

Parallélisme en transport des neutrons

Concernant les applications qui se basent sur la résolution de l'équation de transport dédiées aux calculs de réacteurs, nous ne trouvons, dans la littérature, aucun travail traitant le calcul du coeur entier ou de grandes parties du coeur en 3D. Cependant, des travaux sur des calculs de coeur de réacteurs à eau légère en 2D ont été entrepris récemment.
L'application du parallélisme en transport neutronique a commencé depuis une dizaine d'années. Dans le cadre de ces applications, on peut citer les travaux (Wilson, 1993; Staler, 1994) qui utilisent la méthode des courants d'interface pour calculer la distribution de flux à l'intérieur des assemblages. Les assemblages sont départagés entre différents processeurs, et les courants aux interfaces constituent la donnée principale à communiquer entre les processeurs. La parallélisation du problème de

transport était surtout basée sur la décomposition du domaine : soit le domaine spatial (Qaddouri, 1996), angulaire ou énergétique (Qaddouri, 1995). Dans le cadre de calcul de coeur entier ou de grandes parties du coeur (multi-assemblage), quelques travaux ont été réalisés en utilisant la méthode des caractéristiques. Le premier utilise une décomposition spatiale pour résoudre un problème multi-assemblage où les chemins des neutrons sont directement connectés quand ils traversent différents assemblages (Kosaka, 1999). L'accélération obtenue sur Sun Entreprise4000 est de 5.3 sur 8 processeurs. Ceci est dû au couplage fort entre les assemblages. Le deuxième est basé sur une technique de décomposition par angle, les directions correspondant à chaque angle sont distribuées sur différents processeurs (Lee, 2000). Les accélérations obtenues sont de l'ordre de 3.7 pour 4 processeurs et 6.8 pour 8 processeurs.

Des techniques basées sur la méthode d'homogénéisation ont été récemment utilisées pour le calcul du coeur en 2D. Il est possible de représenter le problème original (configuration hétérogène) avec une formulation de différences finies utilisant des données homogénéisées et condensées incluant aussi les coefficients de diffusion pour générer la même solution moyenne. Ces méthodes imposent la conservation des courants aux surfaces et les taux de réactions volumiques (Joo, 2002; Cho, 2002; Smith, 2002). Dans (Lee, 2000), les auteurs ont utilisé la méthode de Coarse Mesh Rebalancing 'CMR' et une parallélisation par angle pour un calcul de coeur en 2D avec la méthode des caractéristiques. Les résultats numériques montrent une réduction importante du nombre d'itérations et du temps de calcul. En appliquant les deux méthodes ensemble, on a atteint des accélérations de 20 à 60. La procédure d'intégration originale a été aussi modifiée dans ce travail. Dans les articles (Cho, 2002; Smith, 2002), on utilise la méthode d'accélération Coarse Mesh Finite Difference 'CMFD' pour le calcul du coeur de réacteur LWR [1] en 2D avec la méthode des caractéristiques. Des accélérations de 100 à 300 ont été atteintes sur un seul processeur.

On peut citer aussi quelques travaux portant sur le calcul parallèle en décomposition

[1]Light Water Reactor

du domaine spatial et énergétique et utilisant la méthode de probabilités de collision (Qaddouri, 1996). Et d'autres utilisant la méthode S_N avec une décomposition du domaine spatiale et/ou angulaire (Azmy, 1993; Sjoden, 1997; Fisher, 2003).
Tous ces travaux ont été restreints aux calculs 2D, ceci est principalement dû au fait que les réacteurs PWR [2] peuvent être représentés avec une bonne précision avec les modèles 2D. Cependant, pour les réacteurs CANDU, où les mécanismes de contrôle sont perpendiculaires aux grappes du combustible, une représentation de la géométrie du coeur en 3D est nécessaire. En plus, aucun travail concernant l'évolutivité de ces codes parallèles n'a été publié.

Méthodes d'accélération numériques en transport

Pour avoir une solution en un temps CPU raisonnable, le développement des méthodes d'accélération numériques au niveau des solveurs ont pris une place importante dans le développement des codes de calcul de transport. Dans ce paragraphe, une brève revue des méthodes d'accélération pour les solveurs basés sur la méthode des caractéristiques va être faite. Dans le code des caractéristiques CACTUS (Halsall, 1998), la méthode de rebalancement des groupes d'énergie a été utilisée pour accélérer la solution. Il s'agit de résoudre le problème homogène à la fin de chaque itération et de mettre à jour le flux selon la solution homogène. Les problèmes spatialement larges ne bénéficient pas beaucoup de cette méthode car la dépendance spatiale de la solution du flux converge très doucement. Zika et Adams (Zika, 2000) utilisent une méthode d'accélération synthétique en transport. Comme pour d'autres méthodes synthétiques, l'idée est de résoudre une approximation plus grossière de l'équation de transport pour accélérer la solution la plus exacte de l'équation de transport. Ces méthodes permettent une bonne réduction du temps de calcul. La méthode itérative de rebalancement utilisant des dépendances angulaires a été étudiée par Hong et Cho (Hong, 1999). Mais cette technique a été restreinte

[2]Pressurized Water Reactor

aux discrétisations sous forme de triangles équilatéraux. Sanchez et Chetaine ont développé une méthode synthétique adaptée à leur code de caractéristiques en deux dimensions (Sanchez, 1999). Cependant, les équations d'accélération synthétiques sont non-symétriques et la taille du système devient très grande.
Récemment, des méthodes d'accélération, basées sur l'utilisation des sous-espaces de Krylov (Saad, 1996), ont été developpées pour les problèmes de transport. Ces méthodes sont aussi utilisées comme des solveurs intégrés pour les méthodes de résolution S_N. (Patton, 2002; Warsa, 2003; Hanshaw, 2003)

La nature fortement hétérogène des problèmes de réacteur à grande échelle rend difficile l'implémentation de ces méthodes d'accélération classiques en transport avec la méthode des caractéristiques. Même si ces techniques sont largement utilisées dans les codes de transport, elles sont inefficaces pour les problèmes de grande taille comportant des discontinuités matérielles. Cependant, il faut penser à développer des méthodes d'accélération adaptées à ce genre de problèmes. Des travaux dans ce sens ont été réalisés pour accélérer le calcul de transport pour le coeur complet du réacteur en 2D. (Smith, 2002)

Étude de performances : l'évolutivité des systèmes parallèles

Dans le cadre des applications utilisant des techniques de calcul de haute performance et plus particulièrement le calcul parallèle, les études et tests de performances sont indispensables. Elles doivent permettre de prévoir l'évolution de la performance d'une application donnée sur des machines parallèles données. Cependant, il y a plusieurs manières d'approcher ces études. Durant les dernières années, plusieurs travaux portant sur des études de *l'évolutivité* ou *scalability* des systèmes parallèles ont été entreprises par différents groupes. Ces approches cherchent à quantifier, par l'introduction de métriques, l'évolution de la performance quand la taille du problème et/ou la taille de la machine parallèle (nombre de processeurs) changent.

L'étude de l'évolutivité se base, en général, sur :

- l'étude de la *complexité de l'algorithme parallèle* : la modélisation du temps d'exécution séquentiel et parallèle et la modélisation du temps de communications;
- la *modélisation de la machine parallèle* : calcul des caractéristiques physiques intrinsèques de la machine et calcul des caractéristiques du réseau d'interconnexion.

L'étude analytique de l'évolutivité du système parallèle peut nous aider à identifier les paramètres importants du problèmes : la combinaison algorithme-architecture, l'effet d'augmenter le nombre de processeurs sur la performance en particulier le nombre optimal de processeurs à utiliser pour résoudre le problème, l'accélération maximale et enfin l'impact des paramètres physiques propres à chaque machine.
Dans l'article (Kumar, 1994), les auteurs Kumar et Gupta démontrent la nécessité d'introduire des métriques pour évaluer l'évolutivité des algorithmes et architectures parallèles. Ils ont presenté une revue des métriques existantes et la relation entre elles. L'impact de la dépendance des facteurs technologiques comme la puissance de calcul et la vitesse de communications sur l'évolutivité est aussi discuté.
Les auteurs Marinescu et Rice (Marinescu, 1994) ont proposé un nouveau modèle pour l'accéleration. Ce modèle est basé sur trois paramètres représentant trois sources de surcoût (overhead) dans les calculs parallèles : le nombre d'événements communicant, la fraction séquentielle, le travail répété et le taux d'exécution des instructions. Si les fonctions représentant ces trois paramètres peuvent être obtenues, alors ce modèle peut être utilisé pour prévoir l'accélération maximale et l'évolutivité du système parallèle.
La latence du réseau d'interconnexion, le délai dû aux communications entre processeurs, et l'accès mémoire à travers le réseau sont les sources majeures de surcoûts. Dans l'article (Zhang, 1994), les auteurs ont introduit une métrique expérimentale

utilisant la latence du réseau pour évaluer l'évolutivité du système parallèle. Cette métrique est une extension de la fonction d'isoefficacité (Kumar, 1994) et de la fonction d'isospeed (Sun, 1994). Cependant, la métrique basée sur la latence peut être utilisée pour comparer l'évolutivité des différentes architectures parallèles.
Les auteurs, dans l'article (Sivasubramaniam, 1994), discutent du fait que toutes les métriques proposées sont efficaces pour prévoir la performance analytiquement mais elles ne spécifient pas les sources d'inefficacité dans la combinaison algorithme-architecture. L'article présente une méthode pour isoler la composante séquentielle, les charges non-balancées, la latence du réseau et la contention impliquée dans ces intéractions. Ces composantes sont incluses dans une fonction de surcoût. Cette fonction est utilisée alors pour étudier l'évolutivité du système parallèle.

À côté de ces études analytiques sur l'évolutivité, d'autres travaux ont été réalisés portant sur l'évolutivité des architectures parallèles du point de vue matériel et de la conception. L'évolutivité des architectures à mémoire partagée dépend du réseau d'interconnexion et du protocole de cohérence de cache. Dans l'article (Krishnamoorthy, 1994), on décrit l'architecture à mémoire partagée utilisant un réseau d'interconnexion sous forme de Fat Tree. On présente aussi le protocole de cohérence de cache pour ce réseau. Les auteurs décrivent la représentation des algorithmes pour analyser l'évolutivité des architectures à mémoire partagée en termes d'opérations de bases réalisées par une architecture pour supporter le paradigme de mémoire partagée. Parmi d'autres travaux traitant les mêmes aspects, on peut citer (Saavedra, 1994).
Une architecture parallèle évolutive doit être complétée par un système logiciel capable d'augmenter la performance de l'application quand le nombre de processeurs augmente. Le système logiciel doit aussi faciliter la portabilité des programmes à travers les plateformes du calcul parallèle tout en maintenant leur évolutivité. Un algorithme qui est optimal sur une architecture donnée avec un certain nombre de processeurs peut ne pas être optimal si le nombre de processeurs ou d'autres

paramètres de la machine changent. Ceci implique que l'algorithme doit être adapté pour différentes machines parallèles. Les travaux (Panwar, 1994; Dongarra, 1994) traitent de cet aspect. Dans l'article (Vetter, 2003), les auteurs décrivent les caractéristiques de communications dans les applications scientifiques à grande échelle utilisant le système logiciel MPI (Message Passing Interface) (Groop, 1994).
Un autre aspect qui peut influencer l'évolutivité d'un algorithme parallèle est l'équilibrage de charges. Dans plusieurs applications, ce n'est pas possible d'avoir une distribution du travail statique sur les processeurs quand le travail n'est pas bien structuré et il n'est pas alors possible de déterminer la charge de calcul associée à une partie du travail. Dans ce cadre, on peut citer (Dutt, 1994).

En conclusion, l'étude de l'évolutivité n'est pas une tâche facile du fait que plusieurs paramètres, souvent reliés, peuvent entrer en jeu. Ces paramètres sont liés soit au système logiciel utilisé par l'application comme la bibliothèque de passage de messages ou l'algorithme d'équilibrage de charges, soit à la machine parallèle comme la puissance de calcul, le protocole de cohérence de cache, la latence du réseau... La majorité de ces travaux se concentrent sur une partie de ces paramètres au détriment des autres. Selon l'application, la machine parallèle en question et les paramètres dont on tient compte, ces études peuvent être plus ou moins efficaces. Alors, le choix de l'une ou de l'autre de ces études, selon les paramètres à considérer, est crucial.
En plus, presque dans tous ces travaux on utilise pour les expériences des benchmarks se basant sur des algorithmes simples et très optimisés comme les algorithmes de transformée de Fourrier FFT, quelques méthodes de résolution de systèmes linéaires comme Gauss-Seidel, factorisation de Cholesky... Dans les applications scientifiques, où plusieurs développeurs participent à l'élaboration d'énormes codes de calcul, les algorithmes utilisés ne sont pas forcement très optimisés et sont plus compliqués. Par conséquent, ces mêmes études peuvent s'avérer moins efficaces pour ces applications à cause des difficultés au niveau de la modélisation.
En ce qui concerne les travaux reliés au génie nucléaire, nous n'avons trouvé, dans la

littérature, aucun travail traitant l'analyse de l'évolutivité des applications parallèles en se basant sur des modèles informatiques.

1.4 Objectifs de recherche

L'objectif principal de cette thèse est de proposer une nouvelle approche informatique pour résoudre les problèmes de transport de neutrons à grande échelle afin d'améliorer la chaîne de calcul de réacteurs sur différentes machines parallèles avec différentes architectures. Nous avons conçu alors un modèle analytique pour le calcul distribué. Un modèle de performance est ensuite utilisé pour l'étude de l'évolutivité de l'algorithme parallèle.

Depuis l'apparition des premiers réacteurs nucléaires, les efforts de recherche se sont concentrés sur le développement de nouvelles méthodes numériques pour résoudre l'équation de transport et pour accélérer ces solveurs. Maintenant que ces méthodes ont atteint leur maturité, les limitations au niveau des codes de neutronique traitant des représentations plus détaillées des phénomènes physiques qui se déroulent dans un réacteur (engendrant alors des problèmes de grande taille) proviennent essentiellement des limitations au niveau informatique comme la puissance de calcul, les capacités de stockage, optimisation des algorithmes, des E/S...

Dans cette thèse, nous traitons surtout ces aspects informatiques liés aux calculs de réacteurs.
Plus spécifiquement, nous visons les objectifs suivants :

1. *Informatique* :

   (a) études des performances de l'application parallèle en termes de : choix de bibliothèques de passage de messages, les caractéristiques du réseau d'interconnexion, équilibrage de charges et d'autres caractéristiques de la machine parallèle;

(b) la portabilité du code parallèle;

(c) parallélisation des E/S et utilisation d'une base de données pour éviter les problèmes de stockage et les goulots d'étranglement;

(d) extension du modèle à l'étude de l'évolutivité de l'application sur des machines spécifiques;

(e) généralisation du modèle pour tenir compte des architectures hybrides.

2. *Développements numériques* :

   (a) adaptation et implémentation d'une méthode d'accélération basée sur la méthode *GMRES* préconditionnée utilisant les sous espaces de Krylov;

   (b) extension du solveur pour tenir compte des effets de l'anisotropie.

3. *Conception d'un nouveau solveur* :

   (a) afin d'éviter les problèmes liés à l'utilisation et la gestion d'énormes quantités de données générées par la procédure de création des lignes d'intégration, nous avons conçu un nouveau solveur *MCG* (Méthode des Caractéristiques Généralisée) basé sur la méthode des caracteristiques. Ce solveur utilise une nouvelle méthodologie de calcul. Les lignes d'intégration ne sont plus maintenant stockées dans de gros fichiers séquentiels, mais sont générées au fur et à mesure qu'on en a besoin.

1.5 Esquisse méthodologique

Dans cette section, nous allons présenter les démarches suivies pour atteindre ces objectifs :

- *Informatique* : pour atteindre l'objectif (a), après avoir implémenté les deux bibliothèques de passage de messages PVM et MPI au niveau du solveur, nous allons fait des tests de performances en terme d'accélération (speedup). Des comparaisons entre les deux implantations, pour les mêmes cas tests, seront faites pour conclure quant à la plus performante. Notre modèle permettra d'expliquer ces différences. Le modèle sera aussi utilisé pour étudier l'influence des paramètres réseaux sur le temps de communications. Des comparaisons seront faites entre les résultats prévus par le modèle et ceux obtenus par des tests expérimentaux en utilisant une variété de réseaux. Des tests sur l'équilibrage de charges vont être aussi réalisés pour connaitre la technique la plus balancée.
  Pour tester la portabilité de notre code parallèle et atteindre alors l'objectif (b), nous allons exécuter le code sur différentes architectures avec différentes implémentations de bibliothèques de passage de messages MPI; les résultats montrent que le code est complétement portable.
  L'objectif (c), sera atteint en proposant deux approches : la première consiste à paralléliser les E/S au niveau de la lecture de fichier de lignes d'intégration. Pour ce faire, nous avons utilisé les fonctions de MPI-IO (l'implémentation MPI-2). La deuxième approche consiste à utiliser une base de données pour stocker et accéder aux données concernant les lignes d'intégration. Une comparaison entre les deux approches a été faite pour montrer les limitations de chacune.
  Pour l'ojectif (d), nous allons se baser sur le modèle pour étudier le comportement de notre algorithme parallèle vis à vis de l'augmentation de la taille du problème et du nombre de processeurs. Des tests d'évolutivitéseront alors réalisés en faisant changer la taille du problème (de quelques dizaines de régions jusqu'à quelques milliers de régions) et le nombre de processeurs (de 2 jusqu'à 64). Ces tests vont aussi servi à valider le modèle. Nous allons ensuite généralisé le modèle pour tenir compte des modes de communications dans les

architectures hybrides, des tests de validation seront réalisés pour atteindre l'objectif (e);

- *Développements numériques* : pour atteindre l'objectif (a), nous allons d'abord dériver les équations des caractéristiques sous forme algébrique en utilisant l'équivalence avec la méthode des probabilités de collision. Une fois que nous aboutirons à un système algébrique linéaire, nous utiliserons la méthode *GMRES* préconditionnée pour accélérer le solveur. Le choix de cette méthode de résolution est dû surtout à sa bonne efficacité pour résoudre les systèmes linéaires en un nombre réduit d'itérations. Des tests traitant des problèmes hétérogènes avec des discontinuités matérielles vont être réaliser. L'objectif (b), est atteint en considérant un développement linéaire des sections efficaces de diffusion en polynômes de Legendre intégré dans les équations des caractéristiques. Nous allons faire des tests pour comparer les solutions isotrope et anisotrope.

- *Conception d'un nouveau solveur* : Pour éliminer quelques problèmes qui perdurent, lors de la résolution de gros problèmes, comme les limitations dues aux capacités de stockage, nous allons concevoir un nouveau solveur des caractéristiques basé sur une nouvelle méthodologie. Pour atteindre cette objectif, un effort important de programmation a été fourni. Nous allons exécuter plusieurs tests de validation du solveur pour comparer avec des résultats de références.

1.6 Principales contributions de la thèse

Les principales contributions de cette thèse sont :

- la modélisation du calcul de transport réparti pour des problèmes de grandes tailles en 3D qui comporte : la modélisation du temps de calcul, en se basant sur le calcul atomique (calcul sur une ligne d'intégration), nous a permis de bien distribuer la charge sur les processeurs en utilisant des techniques d'equilibrage de charges adéquates. La modélisation du temps de communications, en tenant compte des paramètres des réseaux comme la latence et la bande passante, nous a permis d'expliquer l'influence de ces paramètres sur le temps de communications;
- la modélisation des performances et l'introduction d'une métrique pour mesurer l'évolutivité du solveur parallèle. Des problèmes de taille de plus en plus grande ont été résolus;
- généralisation du modèle pour tenir compte des architectures hybrides;
- une des contributions importante de cette thèse est aussi la conception du nouveau solveur dédié à la résolution des problèmes de grande taille. Ce solveur pourrait servir dans le futur dans l'industrie liée au génie nucléaire.

L'aspect multidisciplinaire du sujet de cette thèse, nous a conduit à aborder plusieurs aspects reliés à notre application. Cependant, nous pouvons séparer les contributions générales de cette thèse en deux parties :

- contributions en génie nucléaire : De point de vue méthodologique, nous avons introduit une nouvelle approche pour résoudre des problèmes de transport liés au calculs de réacteurs. Cette approche consiste à utiliser les méthodes de calcul de haute performance en se basant sur des modèle informatiques pour construire des solveurs capables de traiter des problèmes de plus en plus grands en un temps d'exécution raisonnable. Ceci permettra aux ingénieurs nucléaires et aux scientifiques d'utiliser le calcul de transport plus fréquemment pour évaluer les différents paramètres intervenant lors de la conception des réacteurs. De tels calculs de transport, avec les méthodes et les ressources informatiques actuellement utilisées, prendraient un temps énorme, et ne peuvent en aucun cas être assimilés à des calculs de routine. Ceci permettera aussi de renforcer l'utilisation des technologies de calcul de haute performance en industrie nucléaire.

- contributions en génie informatique : Du point de vue académique, ce travail est un exemple de multidisciplinarité qui rassemble plusieurs domaines comme le génie nucléaire, l'analyse numérique et le génie informatique.
  Le modèle que nous avons conçu, surtout la modélisation de temps de communications et la modélisation des machines parallèles utilisées, peut servir de base pour d'autres problèmes scientifiques et d'ingénierie qui utilisent des modèles mathématiques se basant sur des équations aux dérivées partielles tels qu'en dynamique des fluides, thermo-hydraulique, météorologie...

Des tests de validation du modèle montrent une bonne concordance entre les résultats prévus par le modèle et ceux obtenus par des expérimentations. Les tests d'évolutivité ont montré que nous pouvons maintenir l'efficacité maximale constante si nous arrivons à trouver une bonne combinaison entre la taille du problème et la taille de la machine. Nous avons atteint des accélérations de 7.5 sur 8 processeurs et 23 sur 32 processeurs. Des problèmes de taille allant de quelques dizaines de régions à des milliers de régions ont été résolus.
Les résultats de validation du nouveau solveur utilisant des tests standard ont montré de très bonnes concordances avec les résultats de références issus des autres solveurs de transport.

1.7 Organisation de la thèse: Revue des articles

Cette thèse est présentée sous forme de quatre articles, trois sont accéptés dans des revues avec comité de lecture et un article est soumis à une revue avec comité de lecture aussi. En annexe, nous regrouperons cinq autres articles presentés dans différentes conférences internationales. L'ensemble de ces articles exposent les différents développements décrits dans la section précédente. Dans cette section, nous allons faire une revue de ces articles et présenter l'organisation de ce document.

1.7.1 Les articles constituant les chapitres de la thèse

Chapitre 2 : Cette article est consacré surtout à l'étude de l'influence des caractéristiques des réseaux d'interconnexion sur les performances de notre application. Nous avons considéré une grande variété de réseaux existant sur le marché comme Ethernet, Myrinet, Infiniband et autres. Nous avons alors mesuré le temps de communication utilisant des tailles de messages différentes. Nous avons constaté que le temps de communication dépend fortement des caractéristiques des réseaux, surtout de la latence.

Chapitre 3 : Ici, après avoir expliquer les aspects théoriques de l'application, nous avons introduit le modèle analytique de l'algorithme parallèle. Il s'agit d'une étude de complexité de cet algorithme en terme de modélisation générale du temps de communication et du temps de calcul. Nous avons commencé par une étude théoriques utilisant des machines standards comme *CRAY/T3D* et *IBM/SP2* dont on connait les caractéristiques. Ces études se sont portées sur deux implémentations différentes des bibliothèques de passage de messages : PVM et MPI. Une comparaison utilisant le même modèle a été alors faite pour montrer les écarts de performances. La deuxième étude, utilisant uniquement l'implementation MPI, consistait à l'étude de performances sur des machines parallèles auxquelles on a accès. Des accélérations et des efficacités sont alors montrées et comparées.

Chapitre 4 : Nous nous sommes concentrés, dans cet article, sur l'application de notre modèle à l'étude de l'évolutivité de l'algorithme parallèle. Il s'agit de répondre à la question suivante : comment la performance (en terme d'efficacité) varie quand on fait varier la taille du problème et/ou le nombre de processeurs de la machine parallèle? Ce genre d'étude est important pour prévoir les performances d'une application sur un très grand nombre de processeurs en utilisant un nombre limité de processeurs. Pour nos tests, nous avons utilisé deux machines paralleles: un cluster de *SMP* et un cluster de *NUMA*.
Les résultats sur la validation de notre modèle montrent une bonne concordance entre les resultats prévus par le modèle et ceux donnés par les expérimentations. L'analyse d'évolutivité, utilisant la notion de la fonction d'isoefficacité, nous a conduit à la conclusion suivante : notre application est scalable si nous arrivons à avoir une bonne combinaison entre la taille du problème et la taille de la machine (nombre de processeurs).
Nous avons aussi généralisé le modèle pour tenir compte des architectures hybrides.

Chapitre 5 : Dans cet article, nous avons adapté et implémenté une nouvelle méthode d'accélération numérique. Il s'agit de la méthode *GMRES* utilisant une projection de sous espace de Krylov. La méthode *SCR* est aussi utilisée comme precoditionnement pour *GMRES*. De bonnes accélérations ont été observées surtout pour des problèmes comportant de fortes hétérogéneités spatiales.

1.7.2 Les publications en annexe

Publication 1 : Dans cette article, nous présentons un aperçu général sur : (a) les méthodes d'accélération utilisées dans le solveur des caractéristiques *MCI* notamment la méthode de Self Collision Rebalancing (*SCR*) et la méthode de fusionnement des lignes d'intégration (TMT). (b) la parallélisation du solveur, en passant en revue les différentes techniques d'équilibrage des charges statiques utilisées. Enfin, des résultats sur l'accélération (speedup) de l'implémentation PVM sont donnés avec différents niveaux de fusionnement des lignes et sur deux différentes grappes de calcul.

Publication 2 : Cette article traite en détails des différentes techniques et algorithmes d'équilibrage de charges pour montrer la technique la plus balancée. Une introduction à la nécéssité de la parallélisation des E/S et à l'utilisation de base de données est aussi discutée. Des résultats comparatifs de ces différentes techniques montrent que la méthode d'équilibrage de charges *STRD*, utilisant un algorithme en round-robin pour distribuer les lignes d'intégration, est plus balancée que les deux autres *SPLT* et *ANGL*.

Publication 3 : Nous nous intéressons, dans cette article, à un aspect très important dans les calculs de réacteurs impliquant des problèmes de grandes tailles. Il s'agit de la gestion d'énormes quantités de données liées aux lignes d'intégration. Deux sortes de problèmes peuvent alors survenir : le stockage de ces données et l'accès à ces données. Deux solutions ont été proposées : la première est la parallélisation des E/S utilisant MPI-IO, dans ce cas les lignes d'intégration sont stockées dans des fichiers séquentiels binaires. La deuxième solution consiste à stocker les données dans une base de données centralisée et à utiliser des accès parallèles et concurents. Dans la partie des résultats numériques, nous comparons ces deux approches et nous aboutissons à des conclusions soulignant les limitations de chacune de ces approches.

Publication 4 : Pour les problèmes de grande taille, l'utilisation de fichiers pour stocker les lignes d'intégration reste toujours le problème majeur à cause du dépassement de la capacité des machines. Dans cet article, nous proposons une solution qui consiste à éliminer l'utilisation de ces fichiers. Nous concevons alors un nouveau solveur des caractéristiques *MCG* basé sur une autre approche, les lignes d'intégration sont générés au fur et à mesure que le calcul se déroule. Ce solveur nécessite beaucoup de ressources de calcul mais aucun stockage de lignes. Une comparaison des différents solveurs de transport, pour résoudre les problèmes de grande taille, est aussi présentée.

Publication 5 : Finalement, dans cet article, nous traitons un problème physique lié aux phénomènes de diffusion des neutrons dans un réacteur. En effet, un traitement linéaire de la dépendance angulaire des sections efficaces de diffusion et de l'opérateur de diffusion a été utilisé pour tenir compte de l'anisotropie. Des résultats comparatifs entre les solutions anisotrope et isotrope montrent que dans certains cas des écarts importants peuvent être observés à la frontière du domaine.

Dans la partie annexes, nous présentons aussi un aperçu sur le calcul de haute performance, le calcul des réacteurs, l'utilisation du module *MCG* et enfin un exemple de fichier d'entrée utilisant le module *MCG*.

CHAPITRE 2

PERFORMANCE EVALUATION FOR NEUTRON TRANSPORT APPLICATION USING MESSAGE PASSING

Auteurs: M. Dahmani, B. Morin and R. Roy

Revue: International Journal of High Performance Computing and Networking

Abstract

In this paper, recent advances in parallel software development for solving neutron transport problems are presented. Following neutron paths along the characteristics of the system, the transport equation is solved to obtain the scalar flux per region and energy group. Due to the excessive number of tracks in the demanding context of 3D large-scale calculations, the parallelization of the solver is considered in order to obtain fast iterative solution. Different load balancing strategies are used for the distribution of the tracks along processors. These strategies are based on the calculation load implied by each track length. The performance of the MPI implementation, using different parallel machines, is analyzed for realistic applications and numerical results are shown. A comparative study of different networks existing on the market and the influence of their parameters on total communication time is also presented.

2.1 Introduction

The goal of the neutron transport theory is to determine the distribution of the neutrons in a certain domain (for example: nuclear reactor). In the most general situations, this distribution is a function of three position variables, two angular variables, the neutron energy and the time. With several fundamental physical assumptions, conservation of neutrons is expressed, in term of their distribution, by the Boltzmann transport equation. The interactions between the neutrons and the atomic nuclei in the domain is expressed by the cross sections defined as a certain probability that a given interaction held. These cross sections are considered as a data for the reactor calculations. The solution of the transport equation in its general form is mathematically impossible. Therefore, several deterministic methods were developed in order to solve the transport equation in particular situations. The most popular methods used are: collision probability, the method of discrete ordinates and the method of characteristics which is the method of our interest in this paper. Different academic and industrial codes based on these methods were developed in many countries all over the world. The recent advances in computer capability in term of memory size and processor speed, on one hand, and the development of the high performance computing methods on the other hand allow investigating the boundaries of the old models in many scientific and engineering applications. Like other applications, the advanced reactor core designs require the advanced computational methods in order to achieve high accuracy in reasonable computational time. In order to solve such a large-scale problems with a minimum amount of CPU time, the parallelization of our solver, based on characteristics method (MCI), is now considered. This parallelization is done by distributing the group of tracking lines on several processors. In this paper, we are interested to evaluate the performance of the parallel program, used in our solver, to solve the 3D transport problems. The message passing implementation using MPI is also presented. Several numerical tests are done to analyze the performance of the algorithm using MPI implementation.

CANDU reactor-physics simulations generally consist of three stages (Roy, 2004):

- Calculation of lattice properties for the *bare* CANDU cell, which includes fuel bundle, coolant, pressure tube, gas gap and calandria tube, and the appropriate amount of moderator, but which excludes any interstitial reactivity device;

- Calculation of *incremental* cross sections of reactivity devices, which represent the effect of such devices, and which are added to the basic lattice cross sections in a modelled volume around the reactivity device;
- Modelling of the entire reactor core in three dimensions, and calculation of the core-wide flux and power distributions.

This paper focuses on simulations for the second stage where extensive 3D simulations are done in order to correctly represent the reactivity devices. The movement of these devices influence the neutron population and accurate nuclear properties are essential for safety analysis and core follow-up. Comparisons with site measurements have shown that numerical simulations of the operating CANDUs gives accurate results. However, new Canadian reactor core designs are now evolving from the original CANDU design, and these new design features are more demanding on HPC resources.

An outline of the paper now follows. We begin, in section 2, by presenting a brief review of the neutron transport theory and the method of characteristics used to solve the transport equation. Section 3 is devoted to the presentation of the parallel characteristics solver. In section 4, message passing implementation using the MPI standard and an optimized Linux kernel is described. Then, tests and numerical results pertinent to the field of Canadian nuclear engineering will be shown. Finally, we draw some conclusions.

2.2 Theoretical background

2.2.1 Transport equation

The time-independent transport equation can be written as follows (Roy, 2003):

$$\begin{aligned}\hat{\Omega}\cdot\vec{\nabla}\Phi(\vec{r},\hat{\Omega},E)+\Sigma_t(\vec{r},E)\Phi(\vec{r},\hat{\Omega},E) &= \\ \int_0^{\infty} dE' \int_{4\pi} d^2\Omega'\ \Sigma_s(\vec{r},\hat{\Omega}\leftarrow\hat{\Omega}',E\leftarrow E')\Phi(\vec{r},\hat{\Omega}',E') & \\ +\chi(\vec{r},E)\int_0^{\infty} dE'\ \frac{\nu}{K_{\text{eff}}}\Sigma_f(\vec{r},E')\int_{4\pi} d^2\Omega'\ \Phi(\vec{r},\hat{\Omega}',E'); & \end{aligned} \tag{2.1}$$

where

- $\vec{r}$ is a spatial point in the domain D;
- $\hat{\Omega}$ is the solid angle;
- E is the neutron energy;
- $\Phi(\vec{r},\hat{\Omega},E)$ is the angular flux;
- Σ_t is the total cross section;
- Σ_s is the scattering cross section;
- Σ_f is the fission cross section;
- ν is the secondary number of neutrons generated by the fission reactions;
- K_{eff} is the effective multiplication factor;
- χ is the energy-dependent neutron spectra.

After the energy domain discretization using a multi-group formalism, the multi-group isotropic transport equation to be solved to obtain the flux from an isotropic source is

$$\left(\hat{\Omega} \cdot \vec{\nabla} + \Sigma^g(\vec{r})\right) \Phi^g(\vec{r}, \hat{\Omega}) = Q^g(\vec{r}); \tag{2.2}$$

where

- g is the energy group;
- $\Sigma^g(\vec{r})$ is the total cross section in group g;
- $Q^g(\vec{r})$ is the source in group g.

The source in Eq.(2.2) is composed of two terms, the isotropic fission and scattering sources:

$$\begin{aligned} Q^g(\vec{r}) &= \frac{\chi^g}{4\pi\, K_{\text{eff}}} \sum_{g'} \nu\, \Sigma_f^{g'}(\vec{r}) \phi^{g'}(\vec{r}) \\ &+ \frac{1}{4\pi} \sum_{g'} \Sigma_s^{g \leftarrow g'}(\vec{r}) \phi^{g'}(\vec{r}); \end{aligned} \tag{2.3}$$

where $\phi^g(\vec{r})$ is the scalar flux in group g.

2.2.2 Solution of the transport equation with characteristics method

The main idea behind the characteristics method is to solve the differential form of the transport equation following the straight lines (characteristics or tracking lines) (Askew, 1972). The neutron trajectories will be followed in the local coordinates system where an observer is traveling in the neutron direction. The basic transport operator is then transformed into a total differential operator. The local multi-group characteristics equation is then given by

$$\left(\frac{d}{ds} + \Sigma^g(\vec{r} + s\hat{\Omega})\right) \Phi^g(\vec{r} + s\hat{\Omega}, \hat{\Omega}) = Q^g(\vec{r} + s\hat{\Omega}); \tag{2.4}$$

where s stands for the variable along the characteristics line.

The spatial and the angular domains are sampled using a ray tracing procedure as described in (Dahmani, 2002). The Υ domain is first covered by choosing a quadrature set of solid angles, composed of discretized directions and their corresponding weights $(\hat{\Omega}_i, \omega_i)$. Then, the plane $\Pi_{\hat{\Omega}_i}$ perpendicular to any selected direction i, is split into a Cartesian grid meshing and the starting points $\vec{p}_{i,n}$ of each characteristics are found. For each discretized direction i, a whole set of tracks $\vec{T}_{i,n}$ will be generated. When traveling across different regions, the neutron beam following the characteristics crosses $K_{i,n}$ segments numbered by index k; the segment lengths are labeled L_k and the region numbers are N_k. Assume that the segment k crosses region j, then the local relationship between the incoming and outgoing angular flux is given by

$${}^{\text{out}}\phi_j^g(k) = {}^{\text{in}}\phi_j^g(k) + \left[\frac{Q_j^g}{\Sigma_j^g} - {}^{\text{in}}\phi_j^g(k)\right] E(\Sigma_j^g L_k); \tag{2.5}$$

where $E(x) = 1 - e^{-x}$.

The average scalar flux in region j is calculated using

$$\Phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \frac{1}{\Sigma_j^g V_j} \sum_i \omega_i \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g; \tag{2.6}$$

where $\Delta^g_{i,n,k} = {}^{\text{in}}\phi^g_j(k) - {}^{\text{out}}\phi^g_j(k)$ is the flux difference on segment k and $\pi_{i,n}$ is the weight associated with track $\vec{T}_{i,n}$.

From this equation, we see that the iteration sweep over all characteristics involves the sequential evaluation of outgoing fluxes of Eq. (2.5) on every single track and the adding of flux differences $\Delta^g_{i,n,k}$ for each track $\vec{T}_{i,n}$ in Eq.(2.6).

2.3 Parallel characteristics solver

2.3.1 Finest granularity level

In a parallel characteristics solver, the basic computation unit is focused on repeating the same sequence of calculations on each tracking line. The characteristics sweep on one single line represents the atomic level for the solver from which we can construct the overall solution for each region, each angle, and each energy group. As we have seen before, the group variable can easily be treated using data parallelism. We now define the finest granularity level of operations that can be performed without any communication from the point of view of task parallelism. At this stage, the required local data is a collection of segment lengths L_k and the region numbers encountered along the line. The nuclear data needed are the local total cross section Σ^g_j and the scattering cross section from one group to itself $(\Sigma^{g \leftarrow g}_s)_j$. No scattering matrices are necessary; this approach is important to preserve certain linearity in calculation and to ensure independence between the calculations performed on different tracks. The global solution is then the combination of these atomic level solutions.

In real world applications, the number of tracks is much greater than the number of processors available. The parallelization of the solver is then done by grouping the tracks and each slave processor takes control of one of these groups. At each iteration step, a reduce/broadcast operation is activated to recover the partial contribution to the angular fluxes; each processor has its own copy of the flux and the source arrays with a consistent unknown ordering. The scalar flux is then reconstructed on every single processor before the iteration procedure starts (multicasting communication process) .

2.3.2 Distribution of the tracks and load balancing

We now study the mechanism by which the atomic tasks are assigned to run on the different physical processors. The term *process* refers to a processing or computing agent that performs tasks (Grama, 2003); this abstract entity is more convenient to study task-dependency and task-interaction without having to adhere to a rigorously defined parallel architecture. Processes can then run on physical processors. The allocation of tasks to processes is done by a *mapping:*

$$\begin{aligned} M : \mathcal{T} &\rightarrow \mathcal{P} \\ (i,n) &\mapsto p \end{aligned}$$

from $\mathcal{T}$ the domain of characteristics onto $\mathcal{P}$ the range of processes. As explained above, the tracking lines $\vec{T}_{i,n}$ are identified by their direction i and their starting point n. A global numbering of tracks has no particular use, so we will assume here that tracks are numbered from 1 to N_T in the order they are generated. This global numbering scheme $l(i,n)$ will be assumed when needed in the following.

Since the calculation time for a track linearly depends on its number of segments, the process calculation load defined as

$$L(p) = \sum_{(i,n)\in M^{-1}(p)} t_T(i,n); \tag{2.7}$$

is not necessarily uniform even if we put the same number of tracks on each process. An evenly number of segments distributed on each processor will be the major criteria to build our load balancing algorithm. A good mapping will take into account the total calculation load of each batch of tracks. The options currently implemented to distribute tracks in the MCI module are:

SPLT Distribution of beams of tracks: in this option, we simply subdivide the tracks in batches of $\frac{N_T}{N_P}$ tracks, as these are generated by the EXCELT module. The first process will work on the first $\frac{N_T}{N_P}$ generated, the second process takes the next $\frac{N_T}{N_P}$ and so on. For domains that contain a lot of small regions, one of these track batches may contain larger number of segments than other ones.

STRD Round-robin distribution: the tracks are distributed in the sequential order of processes coming back to the first when all processors have been given a batch. It is a cyclic mapping of all tracks to a single processor, such that $p \equiv l(i,n) \bmod N_P$. The first process takes the tracks numbered $1, 1+N_P, 1+2N_P, \ldots$, the second process $2, 2+N_P, 2+2N_P, \ldots$, etc. Using this option, it is statistically unexpected to have two batches of tracks having a substantial difference in the average number of segments;

ANGL Distribution on angles: all the tracks having the same direction are grouped together. So, each process takes the tracks with the same direction, so the mapping is $p \equiv i$ or $M(i,n) = i$. Because the number of tracks generated in one direction is not necessarily the same as in other directions, this option may lead to imbalance the load. This option also limits the number of processes to be a multiple of the number of angles.

Several tests were done to show that the STRD option is more balanced than the two other options. Each processor is loaded almost evenly, and the average number of segments on each processor is almost the same (Dahmani, 2003a).

2.3.3 Parallel solution of the transport equation

Once a static load balancing scheme is chosen, there exists a mapping $p = M(i, n)$ that gives for any characteristics $\vec{T}_{i,n}$ the process p where computation is performed. Two computational approaches for distributing the groups of tracks over the processors can be considered. The first approach is that one processor (*master*) distributes the tracks to other processors (*slaves*); the second approach is that every processor generates tracks by calling the EXCELT module and takes its own batch of tracks. In our implementation, we do not use the first approach because when the master processor is working to generate the tracks, the slaves remain in wait status. In addition to that, because the size of the tracking file is generally quite large, the communication times become non negligible. With the second approach, we can avoid to have the processors in wait and also to reduce the communications.

At the solver level, each processor takes control of batches of tracks to accumulate contributions to average angular fluxes; a partial sum of flux differences is accumulated by region and energy group.

$$\Delta_p \Phi_j^g = \frac{1}{\Sigma_j^g V_j} \sum_{(i,n)\in M^{-1}(p)} \omega_i \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g . \tag{2.8}$$

At each inner iteration, the processors communicate their results to all other processes by using a reduce/broadcast procedure. This total reduction operation is necessary to add all the contributions together in order to construct the scalar flux by region and energy group from the partial sums of Eq. (2.8).

$$\Phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \sum_P \Delta_p \Phi_j^g . \tag{2.9}$$

Each process, working on its own group of tracks, executes the following iterative scheme:

1. Guess new fission sources and incoming current (outer loop);
   (a) Guess scattering sources (inner loop);
   (b) Compute the flux for each energy group:
      i. compute local solution for each characteristics line;
      ii. perform a reductive sum of all contributions to flux;
      iii. apply global inner acceleration technique.
   (c) If flux map are not converged, go to (a).
2. Apply global outer acceleration techniques;
3. Compute the K_{eff} for next neutron generation;
4. If not converged, go to 1.

After the reduction sum operation, at the end of the inner loop, all processes have the same copy of the flux.

2.4 Implementation of the message passing

In this section, we give details about the operating system and the implementation of the parallel characteristics solver.

2.4.1 MPI Implementation

To perform the communications between processes participating in calculation, we use a Message Passing Interface (MPI) (Groop, 1994). All MPI communications require a communicator and the MPI processes can only communicate if they share the communicator. The standard MPI includes a whole set of routines that can be used for collective communications. When implementing the MCI solver, we used the variant of the reduce operations where the result is returned to all processes in the communicator. On that case, the standard specification requires that all processes participating in these operations receive identical results using a single Fortran call:

```
      call MPI_ALLREDUCE(totF,partF,Nrs*Ng,MPI_DOUBLE_PRECISION,MPI_SUM,
     $       MPI_COMM_WORLD, ierr)
```

equivalent to the following C instruction:

```
MPI_Allreduce( partF, totF, Ng*Nrs, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD )
```

Note that these all-reduce operations can be transformed into a one-node reduce, followed by a broadcast of the resulting values. However, a better performance is obtained if we take into account the possibility to send several synchronous point-to-point messages on the local switched network. If we consider a perfect butterfly network we can minimize the number of synchronized steps that are needed to add the contribution to the scalar flux (Grama, 2003).

2.4.2 Optimizing computations with Adelie/SSI toolkit

In order to minimize the computational time, we used an optimized Linux kernel. This optimization is performed using Adelie/SSI (Adelie, 2004), a clustering toolkit over a Gentoo GNU/Linux base distribution (Gentoo, 2004). This allows the maximum level of performance to be reached without having to sacrifice the ease of system administration.

The Gentoo GNU/Linux distribution incorporates the concept of FreeBSD ports collection, named portage tree under Gentoo. This feature allows the complete system to be recompiled seamlessly without effort, permitting the highest level of optimization for a specific processor to be used. Gentoo also do away with the traditional System V init process in favor of a much more flexible named levels system. This system permits an almost infinite number of run-levels to be created and used.

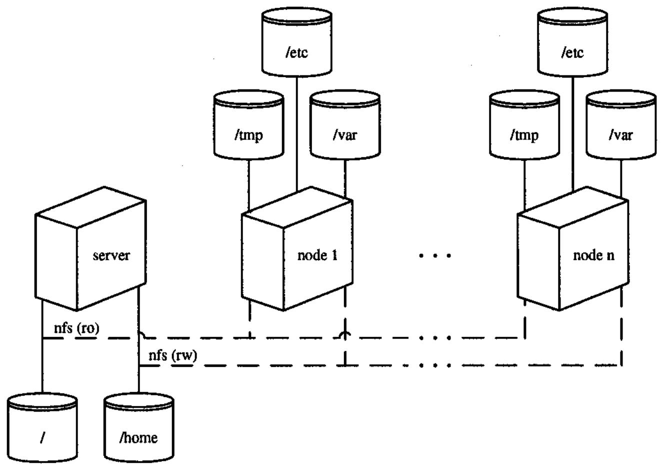


Figure 2.1 Adelie/SSI based cluster

Adelie/SSI, or Adelie for Single System Image, is an initiative of the Adelie Linux Project. This enhancement sits on top of a Gentoo system and allows a complete cluster to be run from a single copy of the system image. This image resides on the server disks and is shared between both the server and the disk-less nodes. Administration is done on this single image, while different run-levels allow customizing of the boot and init process for individual or groups of nodes. Since the nodes do not have disks nor uses space on server disks for writing data, the integrity of a node can not be compromise from the node itself.

Adelie/SSI based cluster is composed of a server containing some form of storage system and a number of disk-less nodes (Fig. 2.1). The operating system image resides on the server and is exported to the nodes via a network file system. In order to have multiples nodes running from the same system image, some area of this image must be local to each node. These areas (`/etc`, `/tmp` and `/var`) are cloned initially from the system image into a memory based file system. After some modifications, they are then locally binded to the main system image. In this manner, it is possible to have a read-only system image which no node can corrupt.

2.5 Numerical results

All the results presented in this section are obtained for CANDU-6 3D supercell problems. A supercell is composed by two horizontal bundles containing the Uranium oxide fuel, and one vertical absorber rod. The transport equation is usually solved by critical buckling search with first order leakage treatment $B1$. Homogenization and condensation processes are then made with the resulting multigroup fluxes. Nuclear properties are condensed to a limited number of energy groups; only 2 groups are generally sufficient for good on-power followup in CANDU-6 reactor cores, keeping the small up-scattering effect from the thermal to fast group. For standard tests, we use 3 discrete directions and 89 energy groups. In all test cases presented here, we use the STRD option for the load balancing of the tracking file.

2.5.1 Validation of parallel program

The parallel program was run on several processors (from 1 up to 16 processors) and the results were compared with those given by the sequential solver. The relative error for a calculated value, presented in Table I, is defined as follows

$$\frac{\Delta\Sigma_{par} - \Delta\Sigma_{seq}}{\Delta\Sigma_{seq}} \times 100\%. \tag{2.10}$$

Results show that the relative errors, between the results given by the sequential

Tableau 2.1 Validation of the parallel algorithm on several processors

Number of processors	1	2	4	8	16
Incr. cross-section	Relative errors (%)				
$\Delta\Sigma_t^1$	0.0124	0.0124	0.0124	0.0083	0.0083
$\Delta\Sigma_a^1$	0.0211	0.0361	0.0273	0.0237	0.0458
$\Delta\Sigma_{s0}^{1\leftarrow 1}$	0.0145	0.0137	0.0072	0.0012	0.0087
$\Delta\Sigma_{s0}^{1\leftarrow 2}$	0.0103	0.0084	-0.0466	-0.0068	0.0143
$\Delta\Sigma_t^2$	-0.0177	-0.0088	-0.0266	-0.0266	-0.0266
$\Delta\Sigma_a^2$	-0.0068	-0.0123	-0.0062	-0.0053	-0.0037
$\Delta\Sigma_{s0}^{2\leftarrow 1}$	-0.0094	-0.0251	0.0482	-0.0110	0.0180
$\Delta\Sigma_{s0}^{2\leftarrow 2}$	-0.0798	0.0110	-0.1037	0.1297	-0.0891

code and the parallel one, are very small. The little discrepancies between the results are due to floating point accuracy. These differences can be explained by the fact that, when performing their partial addition on each processor, the summation order (adding first the biggest values or the smallest ones) can slightly affect the results by neglecting some small contributions.

2.5.2 Speedup

To evaluate the performance of our parallel program, we use the speedup S and the efficiency $\mathcal{E}$ as global metrics. The speedup is defined as usual by the ratio of the serial execution time to the parallel execution time

$$S = \frac{T_s}{T_p}; \tag{2.11}$$

and the efficiency of the system

$$\mathcal{E} = \frac{S}{N_p}. \tag{2.12}$$

As stated above, the sequential program is highly optimized and generates a single tracking file. The parallel program generates local tracking files (in the `/tmp` directory. The number of characteristics lines is linearly decreasing with the number of processors.

For our tests, we use the following machines:

- a Beowulf cluster of 16 disk-less nodes equipped with AMD Athlon 1.4 GHz processor and 1 GB memory connected by a fast Ethernet switched network. The message passing library installed is the LAM MPI implementation;
- a SMP cluster composed of 8 QuadXeon multiprocessors, each of these 8 nodes has 4 processors sharing a 4 GB memory. Nodes are connected with a Myrinet switch. Two MPI implementations are available on this machine: LAM and MPICH;
- a SGI shared memory machine with 16 IP35 Processors (500 MHz) sharing 16 GB memory connected by a Gigabit Ethernet switched network. The MPI implementation used is SGI proprietary version of MPI.
- a NUMA cluster composed of 32 nodes, each node has two AMD Opteron processors (2 GHz) sharing 5 GB memory. Nodes are connected with Myrinet Fiber. The MPI implementation used is MPICH.

In Table II, we show the speedups obtained with different parallel machines. The differences in the speedups are mainly due to the communication performances of each machine and to the MPI implementation used. This shows also that our parallel code is portable on different combination of architecture/implementation. Note that both clusters use Adelie/SSI.

Tableau 2.2 Speedups using different parallel systems

Environment			Number of processors			
Architecture	OS	implem	2	4	8	16
Beowulf ($P = 16$)	Linux	LAM	1.95	3.73	6.58	12.11
SMP cluster ($P = 32$)	Linux	MPICH	1.98	3.92	7.93	15.29
NUMA ($P = 16$)	IRIX	SGI MPI	1.98	3.87	7.68	13.85
NUMA cluster ($P = 64$)	Linux	MPICH	1.97	3.76	7.54	13.22

2.5.3 The influence of the network characteristics on the communication time

For the same machine (SMP cluster), we study the influence of the network bandwidth on the performance. In Fig. 3.4, the speedups for two different implementations using respectively Fast Ethernet ($100Mb/s$) and Myrinet (over $1Gb/s$) switched networks are reported. The speedup curves shows that the speedups have been improved by using the Myrinet switch, especially when the number of processors becomes greater than 16 where the network traffic increases.

In order to extract the influence of the network on our application and to see if our parallel solver is optimal, we have studied more closely network characteristics. Table 2.3 presents measured network performance for various low latency interconnect. The numbers in bold were obtained using the MPI Performance Test version 1.3a

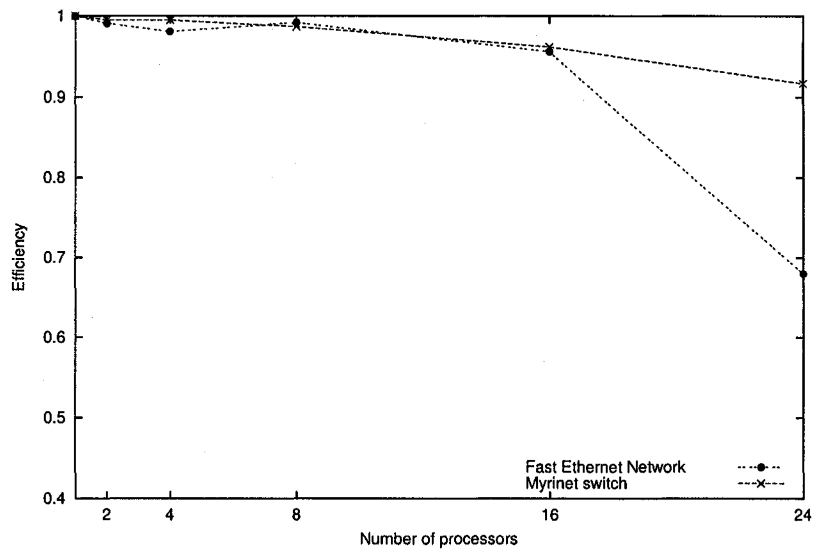


Figure 2.2 The efficiency for the Fast Ethernet and Myrinet switched networks

benchmark (Groop, 1999) using two different clusters. The first is equiped with both Gigabit Ethernet and Myrinet Fiber, the second with Fast Ethernet and Myrinet Lan. These measures are completed with results obtained by different authors (Liu, 2003; Seifert, 2000; Baker, 2001). The results from (Seifert, 2000) were not used for VIA performances since the authors state that the latency measurements were unordinarly high due to the software implementation used.

If we consider a perfect butterfly network, and if we send a message with a size $N_g \times N_r$, the communication time at each iteration would be

$$T_{com} = \lceil \log N_p \rceil \left[t_s + N_g N_r t_w \right] \tag{2.13}$$

where N_p is the number of processors used for the computation, t_s is the message startup time for the data transfert also called the MPI latency, and t_w is the per-word transfert time and which is inversely proportional to the bandwidth.

For a fixed number of processors, the total communication time for our application is then a function of the network parameters

$$T_{com} = T_{com}(t_s, t_w) \tag{2.14}$$

In Fig. 2.3 and 2.4, we present the communication time versus the number of processors for those different networks presented in Table 2.3. In the first Figure, we present the results for a short message. The curves show that networks with a smallest per-word transfert time provide the best communication times. The best communication times here are better for the Infiniband, QsNet and the Myrinet(3) respectively. The same behavior is seen for a longer message (Fig. 2.4). In general, $N_g N_r t_w >> t_s$; the communication time is then practically influenced by the per-word transfert time (or the bandwidth). The curves in these Figures could be classified according to the per-word transfert time corresponding to each network. If we consider the machines with similar processors resources connected with those different networks, the best speedups expected will be those corresponding to the networks with the smallest per-word transfert time according to the famous Amdhal's law. Note that the choice of the machine connecting network should also be a compromise between the network bandwidth and the network cost.

Tableau 2.3 Comparison of different networks parameters for a fixed message size

Network	Theoritical bandwidth (Gb/s)	Measured latency (μs)	Measured transfert time per-word (ns)
Ethernet:			
Fast	**0.1**	**94.60**	**690.87**
Gigabit	**1.0**	**72.02**	**160.38**
Infiniband:			
MT23108 HCA (Liu, 2003)	10.000	6.800	9.506
Myrinet:			
(1)M2L-PCI64A-2	**1.28**	**16.06**	**82.95**
(2)M3F-PCI64B-2	**2.0**	**13.39**	**37.56**
(3)M3F-PCIXD-2 (Liu, 2003)	2.0	6.7	36.45
QsNet:			
Elan3 (Liu, 2003)	2.72	4.6	26.04
SCI:			
D310 (Seifert, 2000)	3.2	8.0	112.73
VIA :			
cLAN1000 (Baker, 2001)	1.25	9.0	81.58

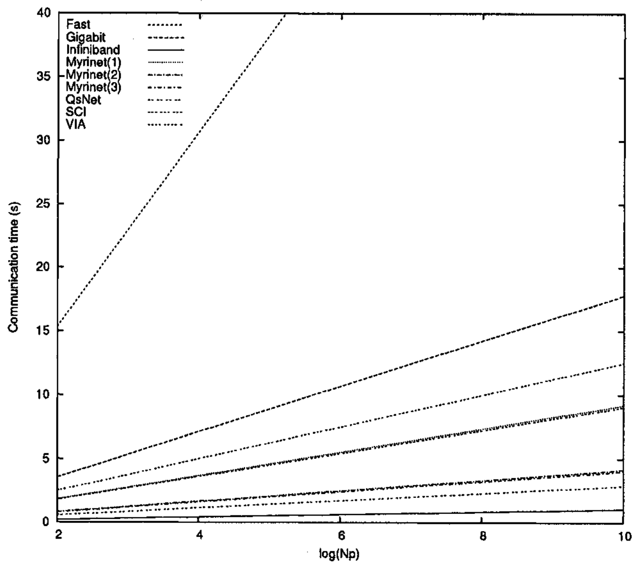


Figure 2.3 Communication time for different networks and with various number of processors (message size=0.46 MB)

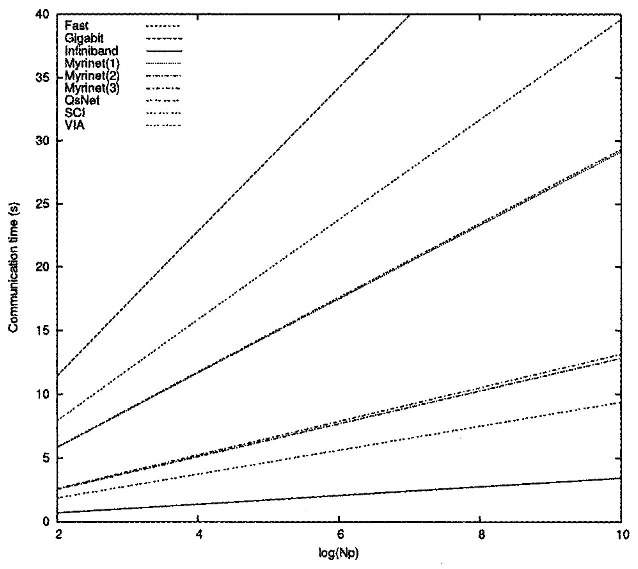


Figure 2.4 Communication time for different networks and with various number of processors (message size=1.28 MB)

2.6 Conclusion

The objective of this work was to evaluate the performance of the parallel program used to solve three-dimensional neutron transport problems. The message passing implementation using MPI, the particular setup using an optimized Linux kernel and the benefits of different load balancing techniques were presented. Speedups obtained yet are encouraging. It was also shown that the parallel code is portable on different types of parallel machines. This will allow us to implement our application on a small heterogeneous systems or using a computer grid.
We expect that our application will be scalable when we increase the dimensionality of the problem according with the increase of the number of processors. Therefore, we are currently working on improving the scalability potential of Adelie/SSI based systems. The goal is to support a massive ad hoc cluster easily, by federating a large number of workstations and clusters together. In order to acomplish this, we are developing a mecanism to rapidely and with no local modification convert a workstation into an Adelie/SSI node. Already existing Adelie/SSI servers and nodes will be integrated in the federated cluster resource tree. There is no need for a common architecture among nodes as heteregenous hardware will be supported.

The next step to performance analysis would be to analytically modelize the communication/calculation ratio in order to be able to predict the specific performance on different kind of architectures. In the coming years, it is expected that large-scale transport calculations will increase the accuracy of nuclear reactor simulations, with the direct effect of safer reactor design and operation. To perform these simulations, we will need scalable linear algorithms that are able to run on several thousands of nodes.

CHAPITRE 3

PARALLEL SOLVER BASED ON THE THREE DIMENSIONAL CHARACTERISTICS METHOD: DESIGN AND PERFORMANCE ANALYSIS

Auteurs: M. Dahmani and R. Roy

Revue: Nuclear Science and Engineering

Abstract

This paper presents the recent advances in parallel software development for solving 3D neutron transport problems using the characteristics method. The characteristics method solves the transport equation by collecting local angular fluxes along neutron paths. In order to be able to solve large 3D transport problems in a reasonable time-frame, the characteristics solver needs to be accelerated. After applying adequately numerical acceleration techniques, the only issue is to parallelize the solver. The parallelization of this solver is based on distributing a group of tracks, generated by ray tracing procedure, on several processors. Different distributing schemes and load balancing techniques based on a calculation load model are presented. A message-passing model is used to communicate the local solutions between processes participating in solving a problem. Both analytical model of this parallel algorithm and performance analysis are presented and illustrated by several examples.

3.1 Introduction

The method of characteristics (MOC) is now commonly used for solving the transport equation in exact geometry because of its good performance for large problems compared to the collision probability (CP) method. Unlike the CP method, the MOC permits the elimination of the CP matrices which have the dimensionality of the size of the square of the number of regions, and still provides solutions as accurate as the CP. Assuming that there are no interactions between neutrons in the medium and the neutrons travels along straight trajectories, the MOC is based on solving the differential form of the transport equation by following the characteristics of the system (tracking lines) which simulate these neutron paths (Askew, 1972). The scalar flux per region and energy group is constructed by collecting all mean angular fluxes in terms of the entering angular flux and the source inside the region. The 3D characteristics solver MCI was developed to treat supercell problems with isotropic boundary conditions. This technology was found accurate with respect to several transport problems specific to CANDU reactors, where reactivity devices are perpendicular to the fuel channels (Wu, 2003a). It has been shown that for relatively small problems with a limited number of regions, the CPU times observed with the sequential MCI solver are similar to ones of the EXCELL module of DRAGON (Marleau, 1997) (which is based on CP method) and with almost the same number of outer iterations. Nevertheless, for large problems the MCI solution is slowed down by the current convergence due to flux/current initialization (flat flux). To accelerate the solution of the MCI, several numerical acceleration methods were tested. The adapted Self-Collision Probability (SCR) method, based on an equivalence theorem between the CP and MOC methods, was shown to give good performance results in high-scattering cases(Wu, 2003b). The SCR technique uses the collision probabilities from one region to itself in order to rebalance the energy distribution of the scalar flux for each region separately. The Track Merging Technique (TMT) can also be used for numerical acceleration. The TMT is designed to reduce the number of tracking lines by merging the two neighboring tracking lines crossing the same regions in the same order into one line associated with the sum of the weights of each of them (Wu, 2003b). Those two techniques are numerically useful, but they still have their limitations. Another complementary way to solve the problem is to explore the computational aspects. For cases with a large number of regions, even when using both TMT and SCR techniques, the iterative solver based on characteristics method can still become very expensive. In order to solve such a large-scale problems with a minimum amount of CPU time, the parallelization of the MCI solver is now considered. This parallelization is based on a distribution of a group of tracking lines on several processors.

Recently, the parallel computers got faster and more accessible economically. Great advances have been made in the CPU and the communication performances. Accordingly, the scientists are increasingly using parallel computers to solve problems that require large amount of computing effort. Day-to-day simulations of advanced reactor design are now done by using these new parallel algorithms. The parallelization methodologies are based on splitting the computational work, necessary to solve the problem, on small parts that are associated to each process(Leopold, 2001). Each process is assigned to one or more processors. To construct the final solution, the processes usually need to communicate some of their intermediate results. Because communications activities are slow compared to computation (often by orders of magnitude), the performance is better when communications are reduced to a minimum, The idea is that the computation effort must dominate the communications. Two standard parallel paradigms are commonly used:

Task parallelism a program is broken down by tasks, and parallelism is achieved by assigning tasks to different processors. In that case, the parallelism is explicit and the message-passing model is used to exchange data between processors. The most widely used software interfaces are the PVM (Parallel Virtual Machine) (Geist, 1994) and MPI (Message Passing Interface) (Groop, 1994).

Data parallelism each processor performs the same work on a unique segment of the data. Either message passing libraries such as MPI (with implementations tuned for shared memory multiprocessors) or higher-level languages such as HPF (or OpenMP directives) can be used for coding with this model. For the second option, the execution of the parallel regions in the code are automatically detected by the compiler.

The parallelization based on CP method obtained after distributing the energy variable in multi-group solver, as well as the usual spatial domain decomposition methods, have already been developed a few years ago (Qaddouri, 1996). Because all group-dependent CP matrices share the same geometric data, data parallelism can easily be applied for computing CP just by varying the total cross sections. Domain decomposition techniques have been also applied to large-scale transport calculations using the method of characteristics. The spatial decomposition of multi-assembly problems where neutron paths are directly connected when crossing assemblies exhibit limited speedup (5.3 on 8 processors) on shared memory Sun Enterprise4000 because of the tight coupling between the assemblies (Kosaka, 1999). The angular decomposition technique, where directions are distributed among a set of processors has also been tested (Lee, 2000). Speedup obtained using this angular decomposition is of the order of 3.7 for 4 processors, and 6.8 for 8 processors. Other related works were done in this context, we can cite (Azmy, 1993; Sjoden, 1997). Almost all these techniques were restricted to two-dimensional problems. In this paper, we now focus on a flexible algorithm to perform 3D state-of-the-art computations.

The paper is organized as follows: in section 2, we present a brief review of the 3D MOC principles and we will derive the discretized characteristics equations. In section 3, the analytical model of our parallel algorithm is described and analysed using different performance metrics and load balancing techniques. Section 4 is devoted to message passing implementations issues pertinent to characteristics transport solver. Section 5 shows numerical tests performed with the CANDU supercells. A scalability study based on efficiency response when varying the problem size and/or the machine size is discussed. In the last section, we draw some conclusions and discuss future work needed to obtain loosely coupled scalable distributed transport solver.

3.2 The 3D characteristics formalism

The multi-group isotropic transport equation to be solved to obtain the flux from an isotropic source is:

$$B^g(\vec{r},\hat{\Omega})\Phi^g(\vec{r},\hat{\Omega}) = Q^g(\vec{r}) \tag{3.1}$$

where:

- $\vec{r}$ is a spatial point in the domain D;
- $\hat{\Omega}$ is the solid angle;
- g is the energy group;
- $B^g(\vec{r},\hat{\Omega}) = \hat{\Omega}\cdot\vec{\nabla} + \Sigma^g(\vec{r})$ is the transport operator in group g;
- $\Sigma^g(\vec{r})$ is the total cross section in group g;
- $Q^g(\vec{r})$ is the source in group g.

The source in Eq.(3.1) is composed of two terms, the isotropic fission and scattering sources:

$$Q^g(\vec{r}) = \frac{\chi^g}{4\pi\, K_{\text{eff}}} \sum_{g'} \nu\, \Sigma_f^{g'}(\vec{r}) \phi^{g'}(\vec{r}) + \frac{1}{4\pi} \sum_{g'} \Sigma_s^{g \leftarrow g'}(\vec{r}) \phi^{g'}(\vec{r}) \tag{3.2}$$

where $\phi^g(\vec{r})$ is the scalar flux in group g.

The main idea behind the characteristics method is to solve the differential form of the transport equation following the straight lines (characteristics or tracking lines). The neutron trajectories will be followed in the local coordinates system where an observer is traveling in the neutron direction. The basic transport operator is transformed into a total differential operator:

$$B^g(\vec{r}, \hat{\Omega}) \to C^g(s) = \frac{d}{ds} + \Sigma^g(\vec{r} + s\hat{\Omega}), \tag{3.3}$$

where $C^g(s)$ is the characteristics transport operator in group g, s stands for the variable along the characteristics line. The local multi-group characteristics equation is then given by:

$$C^g(s) \Phi^g(\vec{r} + s\hat{\Omega}, \hat{\Omega}) = Q^g(\vec{r} + s\hat{\Omega}) \tag{3.4}$$

The average scalar flux in region j with volume V_j and in energy group g is obtained by integrating over volume and direction:

$$V_j \Phi_j^g \;=\; \int_{V_j} d^3r \int_{4\pi} d^2\Omega\, \Phi^g(\vec{r}, \hat{\Omega})$$

$$= \int_{\Upsilon} d^4T \int_{-\infty}^{+\infty} dt\, \chi_{V_j}(\vec{T},t)\ \Phi^g(\vec{p}+t\hat{\Omega},\hat{\Omega}) \tag{3.5}$$

A characteristics line $\vec{T}$ is determined by its orientation $\hat{\Omega}$ along with a reference starting point $\vec{p}$ for the line. Variable t refers to the local coordinates on the tracking line, and the function $\chi_{V_j}(\vec{T},t)$ is defined as 1 if the point $\vec{p}+t\hat{\Omega}$ on the line $\vec{T}(\hat{\Omega},\vec{p})$ in the region V_j, and 0 otherwise. The d^4T element can now be decomposed into a solid angle element $d^2\Omega$ and a corresponding plane element d^2p.

3.2.1 Discretization of characteristics equations

The spatial and the angular domains are sampled using a ray tracing procedure as described in (Dahmani, 2002). The Υ domain is first covered by choosing a quadrature set of solid angles, composed of discretized directions and their corresponding weights $(\hat{\Omega}_i,\omega_i)$. We generally use the Equal Weight Quadrature (EQ_N) to generate the angular directions (Carlson, 1971). Then, the plane $\Pi_{\hat{\Omega}_i}$ perpendicular to any selected direction i, is split into a Cartesian grid meshing and the starting points $\vec{p}_{i,n}$ of each characteristics are found. For each discretized direction i, a whole set of tracks $\vec{T}_{i,n}$ will be generated. When travelling across different regions, the neutron beam following the characteristics crosses $K_{i,n}$ segments numbered by index k; the segment lengths are labeled L_k and the region numbers are N_k. Assume that the segment k crosses region j, then the local relationship between the incoming and outgoing angular flux is given by:

$$^{\text{out}}\phi_j^g(k) = {}^{\text{in}}\phi_j^g(k) + \left[\frac{Q_j^g}{\Sigma_j^g} - {}^{\text{in}}\phi_j^g(k)\right] E_1(\Sigma_j^g L_k) \tag{3.6}$$

where $E_1(x) = 1 - e^{-x}$. Assuming that the track lengths have been normalized to ensure volume conservation, the average angular flux per region is given by:

$$\Sigma_j^g V_j \bar{\Phi}_j^g(\hat{\Omega}_i) = V_j Q_j^g + \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g \tag{3.7}$$

where $\Delta_{i,n,k}^g = {}^{\text{in}}\phi_j^g(k) - {}^{\text{out}}\phi_j^g(k)$ is the flux difference on segment k and $\pi_{i,n}$ is the weight associated with track $\vec{T}_{i,n}$. In Eqn (3.6–3.7), the same calculation is done for every energy group. Automatic data parallelization can be performed for the group variable, or groups can be dispatched on different tasks by compiler directives. This can be particularly useful in shared memory machines.

The average scalar flux in region j is computed by integrating Eq. (3.7) over all directions:

$$\Phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \frac{1}{\Sigma_j^g V_j} \sum_i \omega_i \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g \tag{3.8}$$

From this equation, we see that the iteration sweep over all characteristics involves the sequential evaluation of outgoing fluxes of Eq. (3.6) on every single track and the adding of flux differences $\Delta_{i,n,k}^g$ for each track $\vec{T}_{i,n}$ in Eq.(3.8).

3.3 Analytical model for the MCI parallel algorithm

3.3.1 Parallel algorithm and performance metrics

In this section, we show how the tracks are distributed in order to balance the charge on each processor and to avoid the idle time that can lead to the decline of the performance. The static load balancing techniques are based on calculation load estimations on each processor. First, we provide some metrics on which our performance analysis of the load balancing schemes will be based on.

3.3.1.1 Atomic level

In a parallel characteristics solver, the basic computation unit is focused on repeating the same sequence of calculations on each tracking line. The characteristics sweep on one single line represents the atomic level for the solver from which we can construct the overall solution for each region, each angle, and each energy group. As we have seen before, the group variable can easily be treated using data parallelism. We now define the finest granularity level of operations that can be performed without any communication from the point of view of task parallelism. At this stage, the required local data is a collection of segment lengths L_k and the region numbers encountered along the line. The nuclear data needed are the local total cross section Σ_j^g and the scattering cross section from one group to itself $(\Sigma_s^{g\leftarrow g})_j$. No scattering matrices are necessary; this approach is important to preserve certain linearity in calculation and to ensure independence between the calculations performed on different tracks. The global solution is then the combination of these atomic level solutions.

In real world applications, the number of tracks is much greater than the number of processors available. The parallelization of the solver is then done by grouping the tracks and each slave processor takes control of one of these groups. At each iteration step, a reduce/broadcast operation is activated to recover the partial contribution to the angular fluxes; each processor has its own copy of the flux and the source arrays with a consistent unknown ordering. The scalar flux is then reconstructed on every single processor before the iteration procedure starts (multicasting communication process).

3.3.1.2 Calculation load and problem size

The calculation load on one processor is defined as the time required by the processor in order to complete the computation. In our case, this time can be related to the number of floating point operations (*flops*) necessary for computing all local angular fluxes on a single track containing K segments. The one-group atomic-level time t_T is given by:

$$t_T = 2t_\times + 2t_+ + K\left(t_e + 5t_\times + 7t_+\right), \tag{3.9}$$

where t_+, $t_\times$ and t_e are the time required for an addition, a multiplication and an exponential evaluation respectively.

The total serial time T_S spent by a single processor working on N_T tracks and N_G groups is:

$$\begin{aligned} T_S &= N_G N_T \left[2t_\times + 2t_+ + \bar{K}_T\left(t_e + 5t_\times + 7t_+\right)\right] \\ &\sim N_G N_T \left[4 + \bar{K}_T\left(\bar{\tau}_e + 12\right)\right] t_{op}; \end{aligned} \tag{3.10}$$

where $\bar{K}_T$ is the mean number of segments for the N_T tracks, and t_{op} is the time necessary for one flop (this is a machine-dependent constant). The variable $\bar{\tau}_e$ stands for the mean number of flops for exponential evaluation. In our solver, two ways for evaluating the exponentials are possible: the first one is to use the exact exponential function, the second one is to use interpolation tables based on exponential function values pre-calculated in given interval. The value of $\bar{\tau}_e$ is clearly different for the two methods.

The size of the problem W is defined as the measure of the number of basic operations needed to solve the problem. Here, we can assume that W is proportional to serial time needed to solve the problem:

$$T_S = W t_{op}. \tag{3.11}$$

The size of the problem is nearly proportional to $N_G N_T \bar{K}_T$. In order to achieve high performance computing on parallel machines, the calculation load must be distributed fairly among several processors. This implies that the atomic-level load of each track must be taken into account before statically dispatching a group of tracks to a processor.

3.3.1.3 Communication overheads

In addition to computation time, the processor spends part of its time to communicate with other processors that participate in the calculation. The communication time depends on the number of data words exchanged at each step of the algorithm. Here, we consider the most general case where we exchange parts of the flux vector of dimension $N_R N_G$. In that case, the communication time is:

$$T_{comm} = A(N_P)\left[t_s + N_R N_G t_w\right] \tag{3.12}$$

where:

- t_s is the startup time for data transfer, it is the time required to handle a message at the sending processor. This delay is occurred once for a single message transfer;
- t_w is the per-word transfer time, inversely proportional to the available bandwidth between nodes;
- $A(N_P)$ is a communicator function of the number of processors. It takes into account the communication paths between the processors, depending on the

network topology and the message passing implementation.

If we use N_P processors to do the parallel computation and we assume that all processors have the same load, we can write the relationship between the serial time T_S and the parallel time T_P as:

$$T_P = \frac{T_S}{N_P} + T_{comm} = \frac{T_S}{N_P} + A(N_P)\left[t_s + N_R N_G t_w\right] \tag{3.13}$$

3.3.1.4 Speedup and efficiency

The parallel speedup S is the most common performance metric used in the scientific applications. It is defined as the ratio of the serial execution time to the parallel execution time:

$$S = \frac{T_S}{T_P} = \frac{N_P}{1 + \frac{N_P T_{comm}}{T_S}} \tag{3.14}$$

This relation shows that the speedup tightly depends on the communication overheads. The ideal situation is reached when $T_S \gg T_{comm}$ or $T_{comm} = 0$, in these cases $S \to N_P$, the speedup is then the same than the number of processors used.

The efficiency of the parallel system is evaluated as the ratio of the speedup to the number of processors:

$$E = \frac{S}{N_P} = \frac{1}{1 + \frac{N_P T_{comm}}{T_S}} \tag{3.15}$$

3.3.2 Load balancing techniques

We now study the mechanism by which the atomic tasks are assigned to run on the different physical processors. The term *process* refers to a processing or computing agent that performs tasks (Grama, 2003); this abstract entity is more convenient to study task-dependency and task-interaction without having to adhere to a rigorously defined parallel architecture. Processes can then run on physical processors. The allocation of tasks to processes is done by a *mapping:*

$$\begin{aligned} M : T &\rightarrow P \\ (i, n) &\mapsto p \end{aligned}$$

from T the domain of characteristics onto P the range of processes. As explained above, the tracking lines $\vec{T}_{i,n}$ are identified by their direction i and their starting point n. A global numbering of tracks has no particular use, so we will assume here that tracks are numbered from 1 to N_T in the order they are generated in the sequential module EXCELT of DRAGON: that is firstly by direction and secondly by ΔX and then ΔY incrementation along the perpendicular plane coordinates (Roy, 1994). This global numbering scheme $l(i, n)$ will be assumed when needed in the following.

Since the calculation time for a track linearly depends on its number of segments, the process calculation load defined as

$$L(p) = \sum_{(i,n) \in M^{-1}(p)} t_T(i, n) \tag{3.16}$$

is not necessarily uniform even if we put the same number of tracks on each process. An evenly number of segments distributed on each processor will be the major criteria to build our load balancing algorithm. A good mapping will take into account the total calculation load of each batch of tracks. The options currently implemented to distribute tracks in the MCI module are:

SPLT option Distribution of beams of tracks: in this option, we simply subdivide the tracks in batches of $\frac{N_T}{N_P}$ tracks, as these are generated by the EXCELT module. The first process will work on the first $\frac{N_T}{N_P}$ generated, the second process takes the next $\frac{N_T}{N_P}$ and so on. For domains that contain a lot of small regions, one of these track batches may contain larger number of segments than other ones.

STRD option Round-robin distribution: the tracks are distributed in the sequential order of processes coming back to the first when all processors have been given a batch. It is a cyclic mapping of all tracks to a single processor, such that $p \equiv l(i,n) \bmod N_P$. The first process takes the tracks numbered $1, 1+N_P, 1+2N_P, \ldots$, the second process $2, 2+N_P, 2+2N_P, \ldots$, etc. Using this option, it is statistically unexpected to have two batches of tracks having a substantial difference in the average number of segments;

ANGL option Distribution on angles: all the tracks having the same direction are grouped together. So, each process takes the tracks with the same direction, so the mapping is $p \equiv i$ or $M(i,n) = i$. Because the number of tracks generated in one direction is not necessarily the same as in other directions, this option may lead to imbalance the load. This option also limits the number of processes to be a multiple of the number of angles.

Several tests were done to show that the STRD option is more balanced than the two other options. Each processor is loaded almost evenly, and the average number of segments on each processor is almost the same (Dahmani, 2003a; Dahmani, 2002).

3.4 Implementation issues using message exchange

3.4.1 Parallel solution of the transport equation

Once a static load balancing scheme is chosen, there exists a mapping $p = M(i,n)$ that gives for any characteristics $\vec{T}_{i,n}$ the process p where computation is performed.

Two computational approaches for distributing the groups of tracks over the processors can be considered. The first approach is that one processor (*master*) distributes the tracks to other processors (*slaves*); the second approach is that every processor generates tracks by calling the EXCELT module and takes its own batch of tracks. In our implementation, we do not use the first approach because when the master processor is working to generate the tracks, the slaves remain in wait status. In addition to that, because the size of the tracking file is generally quite large, the communication times become non negligible. With the second approach, we can avoid to have the processors in wait and also to reduce the communications.

At the solver level, each processor takes control of batches of tracks to accumulate contributions to average angular fluxes; a partial sum of flux differences is accumulated by region and energy group.

$$\Delta_p \Phi_j^g = \frac{1}{\Sigma_j^g V_j} \sum_{(i,n)\in M^{-1}(p)} \omega_i \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g \tag{3.17}$$

At each inner iteration, the processors communicate their results to all other processes by using a reduce/broadcast procedure. This total reduction operation is necessary to add all the contributions together in order to construct the scalar flux by region and energy group from the partial sums of Eq. (3.17).

$$\Phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \sum_P \Delta_p \Phi_j^g \tag{3.18}$$

Each process, working on its own group of tracks, executes the following iterative scheme:

1. Guess new fission sources and incoming current (outer loop);
   (a) Guess scattering sources (inner loop);
   (b) Compute the flux for each energy group:
      i. compute local solution for each characteristics line;
      ii. perform a reductive sum of all contributions to flux;
      iii. apply global inner acceleration technique.
   (c) If flux map are not converged, go to (a).
2. Apply global outer acceleration techniques;
3. Compute the K_{eff} for next neutron generation;
4. If not converged, go to 1.

After the reduction sum operation, at the end of the inner loop, all processes have the same copy of the flux.

3.4.2 Message-Passing Implementations

To perform the communications between processes participating in calculation, we use message passing. It consists of the use of precompiled libraries at high level programming. Two different libraries corresponding to two different implementations are used. To evaluate the communicator function $A(N_P)$ for the two implementations, we made these assumptions:

- all processors are directly connected (switched networks);

- the machines are homogeneous: all the processors have similar power;
- we have a perfect load balancing: $\forall p \bullet L(p) = T_S/N_P$.

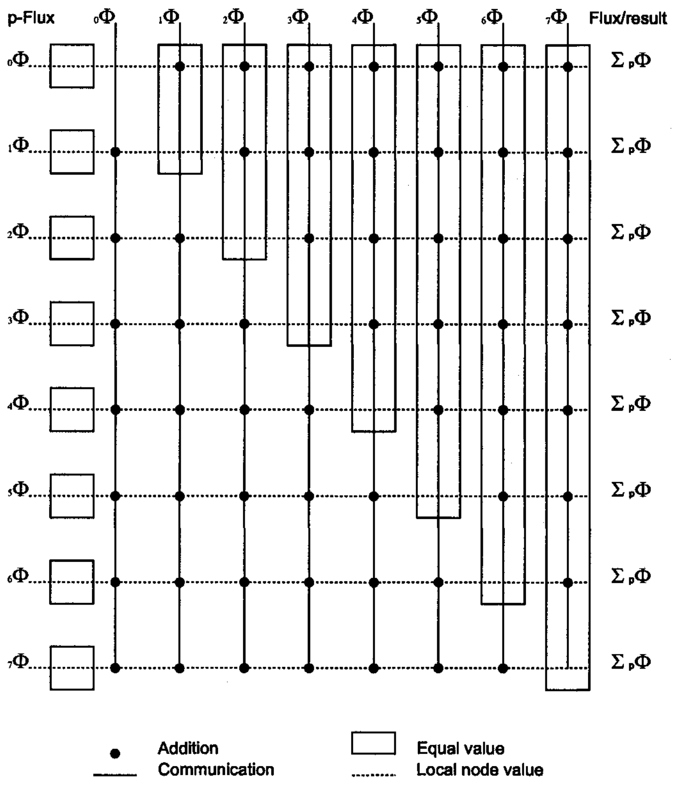


Figure 3.1 The communication process using the multicast operation

3.4.2.1 PVM implementation

The processors participating in calculation build a virtual machine where the communications are managed. In the original PVM implementation of the MCI solver, we used a SPMD-style of programming where each process participates on a peer-to-peer basis to the flux calculation. The routine `pvm_mcast` multicasts a message stored in the active send buffer to `nproc` tasks specified in the `tids` array. The message is not sent to the caller even if it is listed in the array of `tids`. The content of the message can be distinguished by msgtag. Each receiving processes calls pvm_recv to receive their parts of the flux.

The routine pvm_mcast is asynchronous and is based on a minimum spanning tree algorithm between the pvmd daemons. The message is passed to the pvmd daemons containing the specified task identifiers, which in turn distribute the message to their local tasks without further network traffic. The PVM implementation corresponds to the step (*ii*) in the previous iterative scheme, and the MCI/PVM code is the following:

```
      msgtag= 12;
      call pvmfinitsend( PVMRAW, info )
!
!              pack my part of the scalar flux array "partF"
      call pvmfpack( REAL8, partF, Nr*Ng, 1, info);
!
!              mcast to all "tids" (message not sent to the caller!)
      call pvmfmcast( nproc, tids, msgtag, info );

!     initialize scalar flux array "totF" with my part
      do j= 1, Nr*Ng
         totF(j)= partF(j)
      enddo
!
      do i=1, nproc-1
!
!                 receive from any other
         call pvmfrecv( -1, msgtag, info )
         call pvmfunpack( REAL8, partF, Nr*Ng, 1, info)
!
!        add to scalar flux array
         do j= 1, Nr*Ng
            totF(j)= totF(j) + partF(j)
         enddo
      enddo
```

As shown in Fig. 3.1, this multicast communication operation is completed in N_P

steps. The overhead due to communications is driven by the communicator function:

$$A(N_P) = N_P(N_P - 1); \tag{3.19}$$

and the efficiency for the parallel program with PVM implementation as given by Eq. 3.15 can be approximated by:

$$E = \frac{1}{1 + \frac{N_P^2(N_P-1)[t_s+N_RN_Gt_w]}{N_GN_T\left[4+\bar{K}_T(\bar{\tau}_e+12)\right]t_{op}}} \sim \frac{1}{1 + \frac{N_P^2(N_P-1)[t_s+N_RN_Gt_w]}{N_GN_T\left[4+32\bar{K}_T\right]}}. \tag{3.20}$$

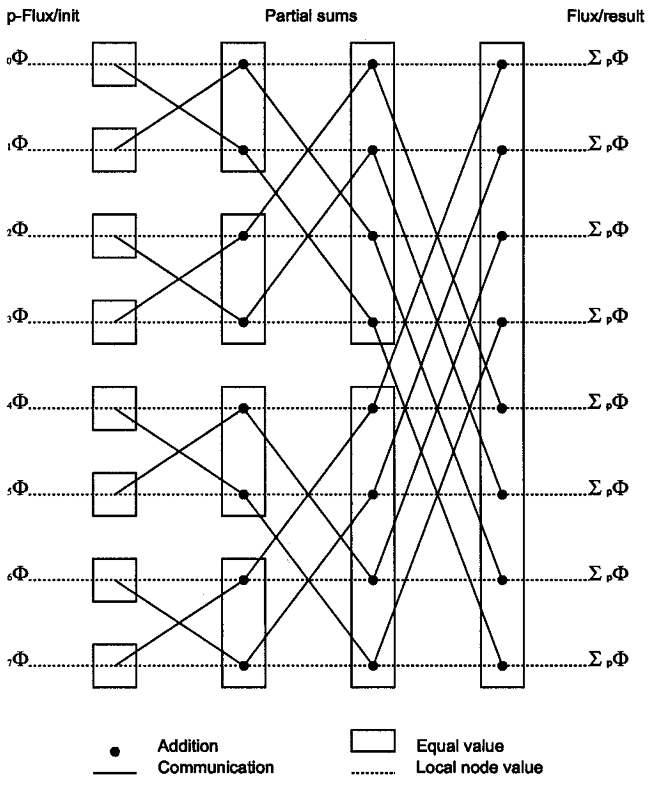


Figure 3.2 The communication process for all-reduce on a butterfly network

3.4.2.2 MPI implementation

All MPI communications require a communicator and the MPI processes can only communicate if they share the communicator. The standard MPI includes a whole set of routines that can be used for collective communications. When implementing the MCI solver, we used the variant of the reduce operations where the result is returned to all processes in the communicator. On that case, the standard specification requires that all processes participating in these operations receive identical results.

The whole set of instructions associated to message passing in MCI/PVM simplifies into a single Fortran call:

```
      call MPI_ALLREDUCE(totF,partF,Nrs*Ng,MPI_DOUBLE_PRECISION,MPI_SUM,
     $     MPI_COMM_WORLD, ierr)
```

equivalent to the following C instruction:

```
MPI_Allreduce( partF, totF, Ng*Nrs, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD )
```

Note that these all-reduce operations can be transformed into a one-node reduce, followed by a broadcast of the resulting values. However, a better performance is obtained if we take into account the possibility to send several synchronous point-to-point messages on the local switched network.

If we consider a perfect butterfly network as shown in Fig. 3.2, we can minimize the number of synchronized steps that are needed to add the contribution to the scalar flux; in that case, the function A is given by (Grama, 2003):

$$A(N_P) = log(N_P); \tag{3.21}$$

and the total communication time:

$$T_{comm} = log(N_P)\,[t_s + N_R N_G t_w]\,. \tag{3.22}$$

The efficiency for the MCI/MPI implementation is then:

$$E = \frac{1}{1 + \frac{N_P log(N_P)[t_s + N_R N_G t_w]}{N_G N_T \left[4 + \bar{K}_T(\bar{r}_e + 12)\right] t_{op}}} \sim \frac{1}{1 + \frac{N_P log(N_P)[t_s + N_R N_G t_w]}{N_G N_T \left[4 + 32\bar{K}_T\right]}}. \qquad (3.23)$$

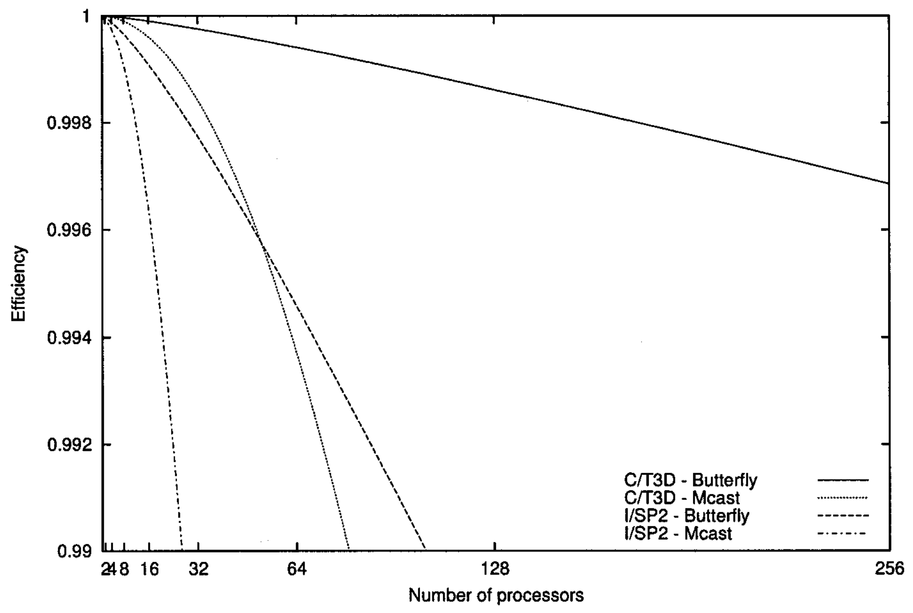


Figure 3.3 The MCI parallel model applied to CRAY/T3D and IBM SP2 computers

The efficiency curves for the two implementations are presented in Fig.3.3 when using data of two common parallel machines: CRAY/T3D and IBM SP2. The characteristics of these computers and the parameters used in the calculations are described in (Wilkinson, 1999). The curves shows that the efficiencies are much better for the MPI implementation on a butterfly network because the number of the communication steps to do the reductive sum is largely reduced comparing to the PVM multicast operations (3 versus 7 for 8 processors) (see Figures 3.1 and 3.2).

In Table 3.1, we show results of the parallel model using the exact exponential function and the table of the interpolated exponential values. These results are obtained by using 32 and 64 processors. In the all cases, the efficiency decreases

Tableau 3.1 Efficiency obtained using exact or interpolated exponentials

	$N_p = 32$		$N_p = 64$	
Computer/Network	Exact exp()	Interpolated values	Exact exp()	Interpolated values
C/T3D - Mcast	95.33%	91.64%	71.52%	57.43%
C/T3D - Butterfly	99.98%	99.96%	99.96%	99.92%
I/SP2 - Mcast	68.94%	54.38%	21.45%	12.79%
I/SP2 - Butterfly	99.84%	99.70%	99.62%	99.30%

when using the interpolated values. This can be explained by the fact that the computation time is reduced and the communication time becomes more important. The differences are bigger when we use the I/SP2 machine because already the communication time parameters (t_s and t_w) are very large than those of C/T3D machine.

3.4.2.3 Comparison between the two implementations

In order to verify the accuracy of our analytical model, the same calculations were performed, using a distributed memory Beowulf cluster (Beowl) with 16 processors for the PVM and MPI versions of the MCI solver. The 3D problem treats a CANDU-6 supercell composed by two bundles containing the Uranium oxide fuel, and one vertical absorber rod. An EQ_4 angular quadrature is used. The dimensions are 89 energy groups, 46 regions and 275262 tracks were generated after using a density of 10 tracks/cm^2.

In Table 3.2, we compare the speedups obtained with the two implementations in two situations: with and without merging lines. The results shows that the MPI implementation is more efficient than the PVM one, especially when the number of processors is greater than 8. The speedups with MPI are better than those of PVM

Tableau 3.2 Speedup for PVM and MPI implementations

		Number of processors				
Implementation	Merging	2	4	8	12	16
PVM	No TMT	1.94	3.68	6.53	8.68	9.60
MPI	No TMT	1.95	3.73	6.58	8.65	12.11
PVM	TMT1	1.89	3.45	5.47	6.42	6.37
MPI	TMT1	1.89	3.43	5.43	6.70	9.72

for both cases because of the performance of the MPI message passing routines. With PVM, we begin to lose a performance after 8 processors because the communication time becomes important. For both implementations, when using the track merging option TMT1, (Wu, 2003b) the speedups are less than those without merging the lines. This is due to the fact that the number of tracks becomes insufficient; the computation time is comparable to the communication time. The metrics defined in our analytical model show a similar behavior, but we will now concentrate on realistic numerical results and analyze the performance of the parallel solver with different problem sizes.

3.5 Numerical results and discussion

All the results presented in this section are obtained for CANDU-6 3D supercell problems. A supercell is composed by two horizontal bundles containing the Uranium oxide fuel, and one vertical absorber rod. The transport equation is usually solved by critical buckling search with first order leakage treatment $B1$. Homogenization and condensation processes are then made with the resulting multigroup fluxes. Nuclear properties are condensed to a limited number of energy groups; only 2 groups are generally sufficient for good on-power followup in CANDU-6 reactor cores, keeping the small up-scattering effect from the thermal to fast group. For standard tests, we use an EQ_4 quadrature and 89 energy groups. The spatial mesh-

Tableau 3.3 Validation of the parallel algorithm on several processors

Number of processors	1	2	4	8	16
Incr. cross-section	Relative errors (%)				
$\Delta\Sigma_t^1$	0.0124	0.0124	0.0124	0.0083	0.0083
$\Delta\Sigma_a^1$	0.0211	0.0361	0.0273	0.0237	0.0458
$\Delta\Sigma_{s0}^{1\leftarrow 1}$	0.0145	0.0137	0.0072	0.0012	0.0087
$\Delta\Sigma_{s0}^{1\leftarrow 2}$	0.0103	0.0084	-0.0466	-0.0068	0.0143
$\Delta\Sigma_t^2$	-0.0177	-0.0088	-0.0266	-0.0266	-0.0266
$\Delta\Sigma_a^2$	-0.0068	-0.0123	-0.0062	-0.0053	-0.0037
$\Delta\Sigma_{s0}^{2\leftarrow 1}$	-0.0094	-0.0251	0.0482	-0.0110	0.0180
$\Delta\Sigma_{s0}^{2\leftarrow 2}$	-0.0798	0.0110	-0.1037	0.1297	-0.0891

ing and the density of tracks will be changed as needed. In cases presented here, we use the MPI implementation and the STRD option for the load balancing.

3.5.1 Validation of the parallel solver

The parallel program was run on several processors (from 1 up to 16 processors) and the results were compared with those given by EXCELL module. The relative error for a calculated value, presented in Table 3.3, is defined as follows:

$$\frac{\Delta\Sigma_{cal} - \Delta\Sigma_{ref}}{\Delta\Sigma_{ref}} \times 100\% \tag{3.24}$$

Results show that errors have the same order of magnitude. However, some little discrepancies between results have been detected due to floating point accuracy. These differences can be explained by the fact that, when performing their partial addition on each processor, the summation order (adding first the biggest values or the smallest ones) can slightly affect the results by neglecting some small contributions.

Tableau 3.4 Speedups using different MPI parallel systems

			Number of processors			
Machine	OS	implemetation	2	4	8	16
Beowulf cluster ($P = 16$)	Linux	LAM	1.95	3.73	6.58	12.11
Cluster of SMPs ($P = 32$)	Linux	MPICH	1.98	3.92	7.93	15.29
NUMA ($P = 16$)	IRIX	SGI MPI	1.98	3.87	7.68	13.85
Cluster of NUMAs ($P = 64$)	Linux	MPICH	1.97	3.76	7.54	13.22

3.5.2 The portability of the parallel program

Our parallel program is designed to run on different parallel systems: machine architecture, operating system and MPI implementation can vary as long as these are compliant with the MPI standard. In Table 4.1, we show the speedups obtained with three different combinations of machine architectures, operating systems, network bandwidth and MPI implementations:

- a Beowulf cluster of 16 disk-less nodes equipped with AMD Athlon 1.4 GHz processor and 1 GB memory connected by a fast Ethernet switched network. The message passing library installed is the LAM MPI implementation;
- a cluster of SMPs composed of 8 QuadXeon multiprocessors, each of these 8 nodes has 4 processors sharing a 4 GB memory. Nodes are connected with a Myrinet switch. Two MPI implementations are available on this machine: LAM and MPICH;
- a SGI shared memory machine with 16 IP35 Processors (500 MHz) sharing 16 GB memory connected by a Gigabit Ethernet switched network. The MPI implementation used is SGI proprietary version of MPI;
- a cluster of NUMAs composed of 32 nodes, each node has two AMD Opteron processors (2 GHz) sharing 5 GB memory. Nodes are connected with Myrinet Fiber. The MPI implementation used is MPICH.

The differences in the speedups are mainly due to the communication performances of each machine and to the MPI implementation used. It is also due to the imbalance between the computation power and the network speed.

3.5.3 Performance analysis

In this subsection, we present some results of the performance tests that we have done. All these tests were executed on the cluster of SMPs with MPI implementation.

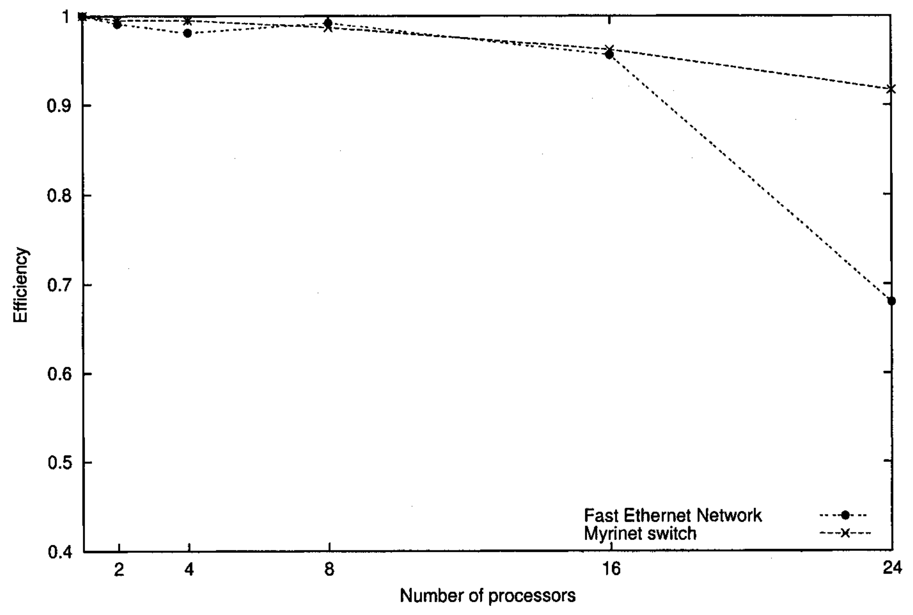


Figure 3.4 The efficiency for the Fast Ethernet and Myrinet switched networks

3.5.3.1 Network bandwidth

We begin by testing the influence of the speed of the network on the performance of our parallel program. We use then two different networks: Fast Ethernet $(100Mb/s)$

and Myrinet switch (over $1Gb/s$). The efficiency curves in Fig. 3.4, shows that the efficiencies have been improved by using the Myrinet switch, especially when the number of processors become greater than 16 where the network traffic increases.

3.5.3.2 Scalability

Scalability analysis is aimed at answering to the following question: How is it possible to change the problem size and/or the machine size (N_P) to keep the efficiency fixed at its maximum value ? For a given parallel computer, the efficiency function depend on the number of energy groups, the number of regions, the number of tracks and the number of processors:

$$E = E(N_G, N_R, N_T, N_P); \tag{3.25}$$

while the number of tracks depend on the number of regions and the density of tracks D_T:

$$N_T = N_T(N_R, D_T). \tag{3.26}$$

We study the behavior of the efficiency when varying the parameters of the problem. In these tests, the number of energy groups kept constant (still equal to 89). If we fix the number of regions N_R and the number of tracks N_T and we vary the number of processors, the efficiency function variations depend essentially on the variation of the function $N_P A(N_P)$ as:

$$E \sim \frac{1}{1 + \frac{N_P A(N_P) N_R N_G \left[\frac{t_s}{N_R N_G} + t_w\right]}{N_G N_T 32 K_T}} \tag{3.27}$$

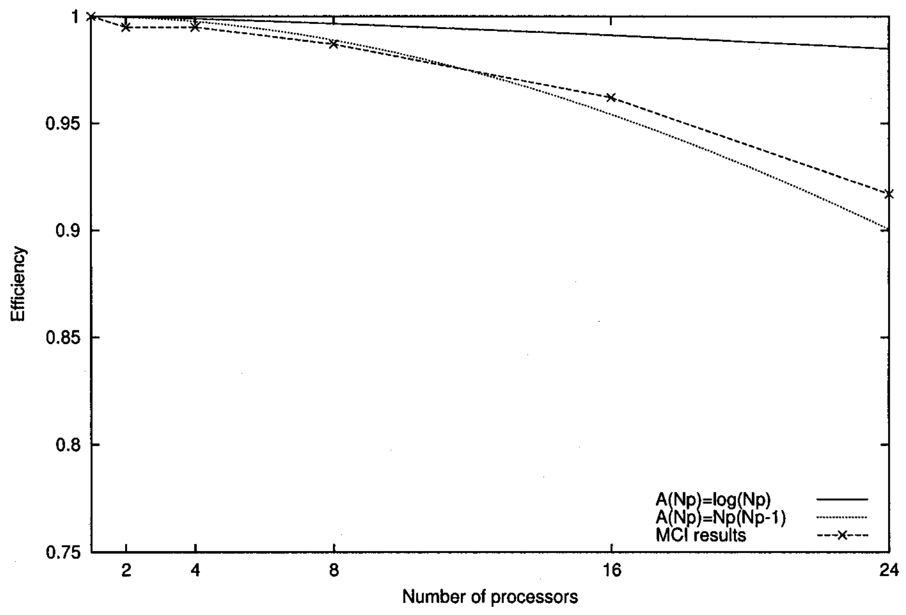


Figure 3.5 The efficiency using different models

For large messages ($N_R N_G \gg t_s$), the communication time is dominated by the network bandwidth. The efficiency can be written as follows:

$$E \sim \frac{1}{1 + N_P A(N_P) \frac{N_R t_w}{32 N_T K_T}} \tag{3.28}$$

For $N_P > 16$, the efficiency linearly decreases. Thus, for a fixed problem size, the peak performance is reached in this interval where the speedups are at their maximum (Fig.3.5).

Now, let us vary the number of regions by choosing different mesh sizes (46, 368 and 560 regions). Obviously, the number of tracks and the mean number of segments also varies and so the ratio $\frac{N_R}{N_T K_T}$. This means that the computational time and the communication time (the size of the message) change at the same time. Thus this ratio slightly varies as we can see in Table 3.5. For $N_P = 8, 16$, the efficiency for

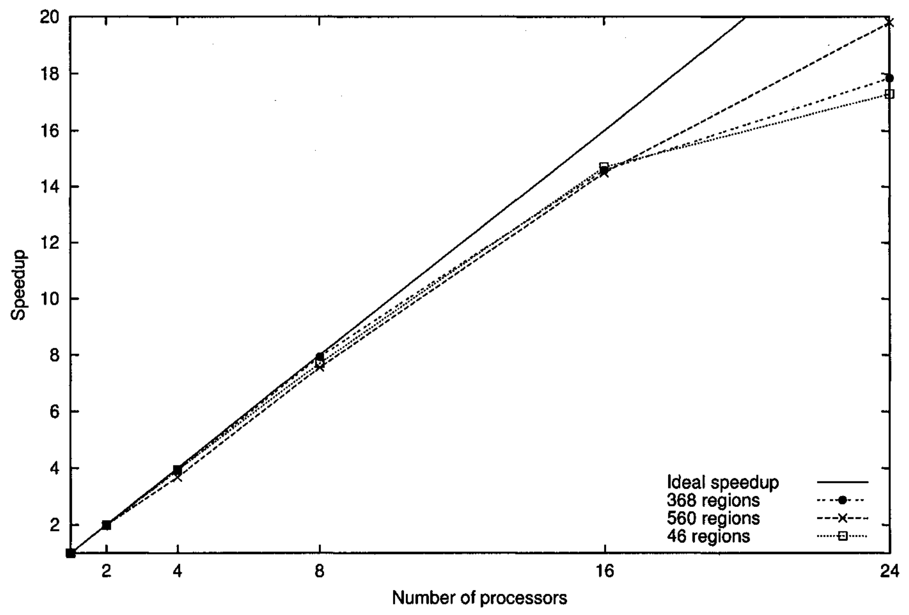


Figure 3.6 The speedups with different mesh sizes

the three cases varies also slowly. For $N_P = 24$, differences are much larger between the two first cases and the third one. We can remark also that, for the case where $N_R = 560$, the efficiency is more stable than in other cases. The ratio between the efficiency for $N_P = 8$ and $N_P = 24$ is 1.15 as compared to 1.33 which is the same ratio for other cases. The speedups for these three cases are presented in Fig.3.6.

For a fixed number of regions, we now increase the number of tracks by varying the user-input value for the density of tracks (tracks/cm^2). Computation time increases

Tableau 3.5 Efficiency for different mesh sizes

			Number of processors		
Mesh size	Number of tracks	$N_R \div N_T$	8	16	24
$N_R = 46$	$N_T = 275262$	0.016 %	0.962	0.918	0.720
$N_R = 368$	$N_T = 1101048$	0.033 %	0.991	0.955	0.741
$N_R = 560$	$N_T = 1646825$	0.034 %	0.943	0.910	0.820

Tableau 3.6 Efficiency for different number of tracks (N_R=368)

		Number of processors		
Number of tracks	$N_R \div N_T$	8	16	24
$N_T = 439004$	0.083 %	0.937	0.896	0.783
$N_T = 1101048$	0.033 %	0.991	0.955	0.741
$N_T = 2207552$	0.016 %	0.951	0.908	0.687

when increasing the number of tracks. However, a nice property of our parallel code is that the size of messages is kept constant because the number of regions is constant. The communication time essentially depends on the variation of the function $A(N_P)$. The ratio $\frac{N_R}{N_T K_T}$ considerably changes in this case, and so the efficiency, as one can see from results in Table 3.6. The speedups curves for this case are also shown in Fig. 3.7. This nice behavior implies that an accurate large-scale 3D characteristics solver could use a multigrid method with different mesh sizes in a rather independent way of the communication overhead costs.

3.6 Conclusion

An analytical model of our parallel algorithm used in the context of a 3D characteristics solver has been presented. The message passing issues, the scalability analysis based on the efficiency response when we change the size of the problem and/or the size of the parallel computer were discussed. Our results show that the efficiency of the system is very sensitive to the ratio of the number of regions to the number of tracks in the domain. However, a good combination between the number of regions, the number of tracks and the number of processors used to solve a given problem can be used to preserve the global efficiency of the parallel system constant at its maximum value. It was also shown that our parallel code is portable on different type of parallel machines with different OS and MPI implementations.

For very large problems, since tracking lines are often stored in sequential files and the solver needs to access these data, we are faced with some I/O and memory bounds. In order for parallel computers to be fully usable in solving such large-scale problems, the I/O performance must also be scalable and balanced with respect to the CPU and communication performance. Thus, in order to be able to solve larger 3D transport problems, the implementation of parallel I/O or the elimination of the tracking file needed in the solver will be considered in our future work.

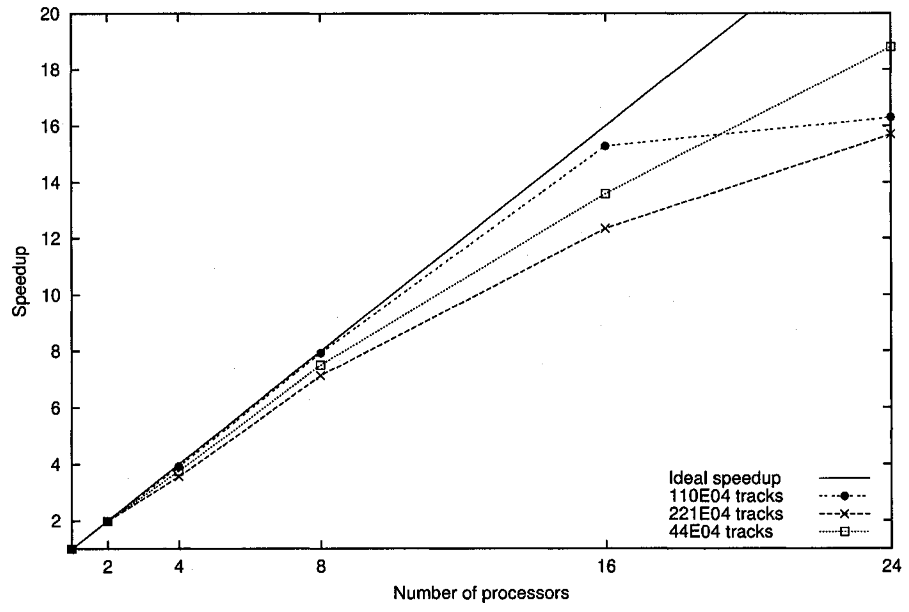


Figure 3.7 The speedups with various number of tracks

CHAPITRE 4

SCALABILITY MODELING FOR DETERMINISTIC PARTICLE TRANSPORT SOLVERS

Auteurs: M. Dahmani and R. Roy

Revue: International Journal of High Performance Computing and Applications

Summary

In this paper, we propose a new parallel solver for the large-scale 3D neutron transport problems used in nuclear reactor simulations. Modern large-memory computers have made possible direct application of transport methods to large-scale computational models. However, many numerical acceleration techniques common to lattice transport codes are not applicable to heterogeneous geometry, especially high-dominance ratio when eigenvalue problems are solved. Consequently, large heterogeneous reactor problems have remained computationally intensive and impractical for routine engineering applications.

Based on the characteristics method, this new code solves the transport equation by following neutron tracks. Due to the excessive number of these tracks in the demanding context of 3D large-scale calculations, the parallelization of the solver is the only way to obtain a fast solution. The parallelization is based on distributing a group of tracks, generated by a common ray tracing procedure, on several processors. An analytical model for the communication/computation ratio is presented in order to predict the specific performance on different kind of architectures. A scalability analysis based on the isoefficiency function is performed on our parallel code when we increase the size of the problem and the number of processors. Tests

are done to validate the analytical model by comparing the results to those given by the empirical tests. Results show that our parallel code is scalable and portable.

4.1 Introduction

The *transport theory* focuses on the mathematical description of the motion of particles through a host medium. The 'particles' of interest might be neutrons, ions, electrons, molecules or quanta (photons or phonons). The 'host medium' might consist of the materials comprising a nuclear reactor core, a stellar atmosphere, or even a burning thermonuclear plasma. The goal of the transport theory is thus to determine the distribution of particles in the medium, taking into account their motion and interaction with atomic nuclei in the medium. The derivation of the neutron transport equation uses several physical assumptions based on the neutron observed behavior when traveling in matter. Moreover, if we further assume that there are no external sources and that the neutrons are generated in multiplying media, the time-dependent transport equation can be expressed mathematically as (Duderstadt, 1983; Lewis, 1984)

$$\begin{aligned}\frac{1}{v}\frac{\partial}{\partial t}\Phi(\vec{r},\hat{\Omega},E,t) + \hat{\Omega}\cdot\vec{\nabla}\Phi(\vec{r},\hat{\Omega},E,t) + \Sigma_t(\vec{r},E)\Phi(\vec{r},\hat{\Omega},E,t) &= \\ \int_0^\infty dE' \int_{4\pi} d^2\Omega'\ \Sigma_s(\vec{r},\hat{\Omega}\leftarrow\hat{\Omega}',E\leftarrow E')\Phi(\vec{r},\hat{\Omega}',E',t) & \\ +\chi(\vec{r},E)\ \int_0^\infty dE'\ \nu\Sigma_f(\vec{r},E') \int_{4\pi} d^2\Omega'\ \Phi(\vec{r},\hat{\Omega}',E',t), & \qquad (4.1)\end{aligned}$$

where v and $\hat{\Omega}$ are the neutron speed and its solid angle direction. Σ_t, Σ_s, and Σ_f are known as the total cross section, scattering cross section, and fission cross section respectively. Those quantities are directly related to the probability that a given interaction holds and are considered as a nuclear data for the reactor calculations. ν is the secondary number of neutrons generated by the fission reaction, and χ is the energy-dependent neutron spectrum. Equation (4.1) is the integrodifferential form of the transport equation for the unknown dependent angular flux variable $\Phi(\vec{r}, \hat{\Omega}, E, t)$.

The complexity of this equation describing the neutron transport processes in heterogeneous forces one to implement numerical methods for its solution. Such methods usually seek to introduce approximations that convert the integrodifferential (or integral) form of the transport equation into a system of algebraic equations that is more amenable to solution by a computer. Concurrently, increasingly sophisticated methods have been developed over the past decades, associated with the rapidly increasing computational power of computers. It is these methods, incorporated into general-purpose computer codes, that one must turn to for the solution of multiregion, multidimensional transport problems that are most often encountered in the analysis of nuclear reactors, radiation shielding, and other applications. These methods can be classified into two categories: the stochastic methods based on the probabilistic approach (Monte Carlo method) and the deterministic methods based on conventional numerical methods. The most widely used solvers in deterministic neutron transport codes are based on the Discrete Ordinate method (S_N), the Collision Probability method (CP), or the Method of Characteristics (MOC). Here, we shortly describe these three methods:

- The S_N approach is the most direct procedure in which the dependent variable is replaced by a discrete set of values at specific points in the domain. The derivative and integrals appearing in the transport equation must also be replaced by a corresponding discrete representation using finite difference and numerical integration. In this way, one arrives at a set of algebraic equations for the discrete representation of the dependent variables. (Carlson, 1968;

Lewis, 1984)

- The CP method is based on the integral form of the transport equation. After computing the collision, escape, and transmission probabilities, we also end up with an algebraic system of equations. (Roy, 2003; Sanchez, 1982)
- In the method of characteristics, the differential form of transport equation is solved by following the characteristics of the system (tracking lines) which simulate the neutron paths. Unlike the previous two methods, the MOC does not end up with an algebraic system. (Askew, 1972; Halsall, 1998)

The vast majority of numerical transport calculations are carried out using a form of the equation in which time dependence is not treated explicitly. The energy domain is also discretized on G discrete intervals called, energy groups according to the multi-group formalism. The scattering operator generally depends on the angular deflection $\mu_0 = \hat{\Omega}.\hat{\Omega}'$ resulting after a neutron collision. The K-eigenvalue form of the criticality problem is formulated by assuming that ν can be adjusted to obtain a time-independent solution. Hence, ν is replaced by ν/K_{eff} leading to a *multi-group eigenvalue problem*

$$\left(\hat{\Omega}\cdot\vec{\nabla} + \Sigma^g(\vec{r})\right)\Phi^g(\vec{r},\hat{\Omega}) = \frac{\chi^g}{K_{\text{eff}}}\sum_{g'=1}^{G}\nu\Sigma_f^{g'}(\vec{r})\int_{4\pi} d^2\Omega'\,\Phi^{g'}(\vec{r},\hat{\Omega}') + \sum_{g'=1}^{G}\int_{4\pi} d^2\Omega'\,\Sigma_s^{g\leftarrow g'}(\vec{r},\hat{\Omega}.\hat{\Omega}')\,\Phi^{g'}(\vec{r},\hat{\Omega}'); \quad (4.2)$$

where g is the energy group index. The solution of K-eigenvalue problem is usually required for the analysis of fission-based systems such as nuclear reactors. The prime interest is to evaluate the fundamental mode eigenvalue (K_{eff}) and the associated shape of neutron flux. The K_{eff} is called the effective multiplication factor. The dominant K-eigenvalue(s) and corresponding eigenfunction(s) for criticality problems are often computed with classical iterative methods such as the power iteration method. To obtain a closed system, some boundary condition equations are derived to take into account the neutron behavior at the external boundary of the domain.

In reactor calculation, lattice codes are used to calculate the spatial and energy distribution of neutrons in a multiplying media (reactor). It takes, as input, a multi-group library of isotopic nuclear data and a geometric description of the reactor lattice, and solves the transport equation over a specified region of the reactor lattice. The smallest reactor component of interest is often a unit cell (for example, a CANDU [1] cluster surrounded by the pressurized tube and D_2O moderator), a fuel assembly (for PWRs [2] and BWRs [3]) or, depending on the availability of computer resources, a part of the core. The calculated flux can be used to produce a set of macroscopic cross sections homogenized over chosen subregions and a broad energy group structure. Those sets of cross sections are then used as nuclear properties for the input for various codes simulating the time-dependent neutron behavior over parts or the whole reactor. The calculated flux can be also used for reaction rate calculation or in fuel depletion calculations. (Glasstone, 1991) However, there is a loss of accuracy when using this two-step process. This is due to the spatial homogenization and energy condensation needed to produce nuclear property database.

Using a fine transport calculation in unit cell and a coarser approximation in whole reactor was necessary because of the limited computer resources (in terms of memory size and CPU power) for large-scale transport problems. Modern large-memory computers have made possible direct application of lattice transport methods to large-scale computational models. Concurrently, the recent advances in computer capability in terms of memory size and processor speed, on one hand, and the development of high performance computing methods on the other hand allow investigating the boundaries of the old models in many scientific and engineering applications.

[1]CANada Deuterium Uranium reactor.
[2]Pressurized Water Reactor.
[3]Boiling Water Reactor.

Like other applications, the advanced reactor core designs require advanced computational methods in order to achieve high accuracy in reasonable computational time. In order to solve such large-scale problems, the parallelization of our solver, based on anisotropic characteristics method, has been performed. This parallelization is based on the distribution of the group of tracking lines on several processors.

In order to solve the large-scale reactor problems using our parallel algorithm, it is important to analyze the variations of the performance according to the changes in the parallel system (formed by the parallel algorithm and the parallel machine) in terms of problem size or machine size (number of processors). This type of analysis is known as the scalability analysis. The scalability can be defined as the system ability to maintain as best the speedup when increasing the number of processors. It permits for a given problem to determine an algorithm-machine combination in order to obtain the best performance. Evaluation of scalability can also predict the performance of large problems on large systems based on the performance of small problems on small systems. There are many scalability metrics cited in the literature, we can cite:

- The *isoefficiency function* (Grama, 2003; Kumar, 1994): relates the problem size to the number of processors required to maintain a system's efficiency, and it can be used to determine the scalability with respect to the number of processors, their speed, and the communication bandwidth of the interconnection network;
- The *isospeed function* (Sun, 1994): based on the speed definition which is the work [4] divided by the number of processors. The algorithm-machine combination is then scalable if this metric remains constant while increasing the number of processors, provided the problem size can be increased with the system size;

[4]In many applications, the work is measured as the number of floating point operations performed.

- The *Latency metric* (Zhang, 1994): uses the network latency to measure and to evaluate the scalability.

Zhang, Yan, and He have proved mathematically the equivalence between the three metrics by giving the analytical relationship among those metrics (Zhang, 1994). In this paper, we will present the analytical model on which our parallel algorithm is based on. The isoefficiency function will be used for the scalability analysis. Several numerical tests are done to analyze the performance of the algorithm and to validate the model.

In section 2, we present a brief overview of the method of characteristics used to solve the transport equation for reactor calculations. The analytical model for the parallel algorithm and the isoefficiency modeling are presented in section 3. The section 4 is dedicated to the MPI implementation. Tests and numerical results are shown in section 5.

4.2 Background and related work

4.2.1 Neutron transport solvers

Assuming isotropic source and scattering operator, the multi-group linear transport equation to be solved to obtain the angular flux

$$\left(\hat{\Omega} \cdot \vec{\nabla} + \Sigma^g(\vec{r})\right) \Phi^g(\vec{r}, \hat{\Omega}) = Q^g(\vec{r}); \tag{4.3}$$

where $Q^g(\vec{r})$ is the source in group g.

The source in Equation (4.3) is composed of two terms, the isotropic fission and scattering sources

$$Q^g(\vec{r}) = \frac{\chi^g}{4\pi\, K_{\text{eff}}} \sum_{g'} \nu\, \Sigma_f^{g'}(\vec{r}) \phi^{g'}(\vec{r}) + \frac{1}{4\pi} \sum_{g'} \Sigma_s^{g\leftarrow g'}(\vec{r}) \phi^{g'}(\vec{r}); \tag{4.4}$$

where $\phi^g(\vec{r})$ is the scalar flux in group g defined by

$$\phi^g(\vec{r}) = \int_{4\pi} d^2\Omega\ \Phi^g(\vec{r}, \hat{\Omega}). \tag{4.5}$$

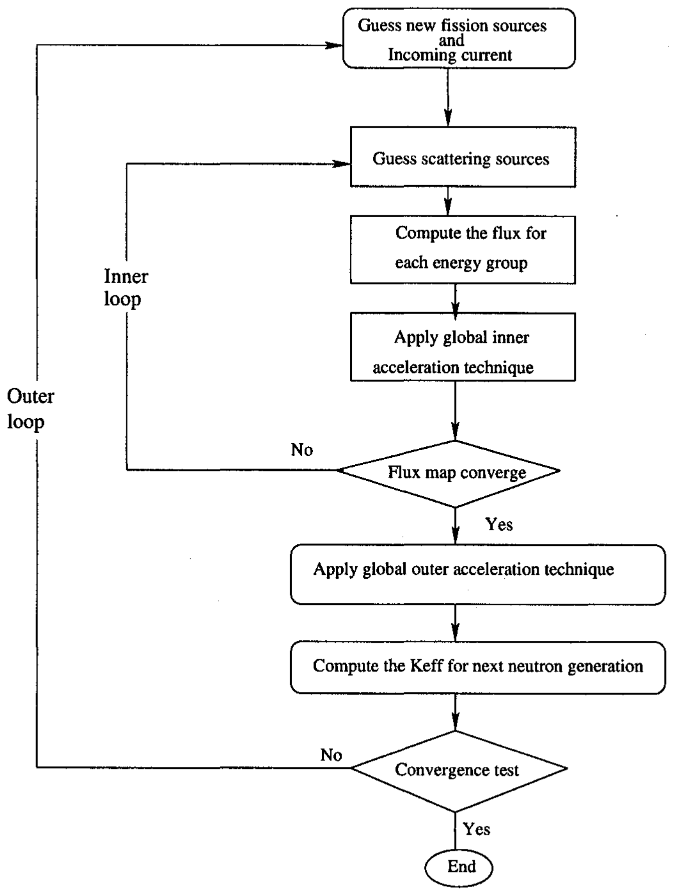

Guess new fission sources and Incoming current
Guess scattering sources
Compute the flux for each energy group
Apply global inner acceleration technique
Inner loop
Outer loop
No
Flux map converge
Yes
Apply global outer acceleration technique
Compute the Keff for next neutron generation
No
Convergence test
Yes
End

Figure 4.1 Iterative transport solvers

The solution over a single group proceeds by assuming that all sources depend upon fluxes in other groups; the within-group fission source is assumed to be known, thus only the within-group self-scattering source depends upon the fluxes in the group. Solution of this group equation is obtained by iteration on the magnitude of the system-wide self-scattering source. Solution proceeds to the next lower energy group until convergence is reached and the solution is obtained for all groups. Once this solution is obtained, an *inner iteration* is done. At this point, a solution over the entire system has been obtained; but now the sources that were assumed to be known (e.g., fission and upscattering sources) must be re-evaluated. This procedure introduces another level of iterations, called *outer iteration.* In each of these iteration levels described, it is necessary to decide when a solution has been obtained, i.e., that satisfies a convergence criterion. In Figure 4.1, we show a typical *inner-outer* iteration procedure used in transport solvers. The power iteration method is often used to solve this kind of problem.

4.2.2 Ray tracing

Neutron paths, between nuclear interactions, are straight lines. We generally neglect the neutron/neutron interactions. Stochastic solvers attempt to generate neutron histories using the following approximations: a neutron will collide at a certain point following the attenuation probability law in materials, and scattering direction after this collision is also randomly determined. In deterministic characteristics solvers, the ray tracing is quite different from what it is in stochastic solvers or computer graphics applications. In these solvers, no secondary ray is followed; however, the primary rays must cover all angular direction in the geometric domain. A characteristics line $\vec{T}$ (tracking line) is defined by its orientation (solid angle $\hat{\Omega}$) along with a reference starting point $\vec{p}$ for the line. The domain is first covered by choosing a quadrature set of solid angles, composed of discretized directions and their corresponding weights $(\hat{\Omega}_i, \omega_i)$. Then, the plane $\Pi_{\hat{\Omega}_i}$ perpendicular to any selected direction i, is split into a Cartesian grid of n elements and the starting points $\vec{p}_{i,n}$ of each characteristics are found. For each discretized direction i, a whole set of tracks $\vec{T}_{i,n}$ will be generated. For a chosen $\vec{T}(\hat{\Omega}, \vec{p})$, the collection of segment lengths L_k and numbers of regions N_k for each region encountered along the line must be calculated (and can eventually be recorded in sequential binary files). In order to accurately cover a 3D domain with reflected boundary conditions, we need $\sim 10^3 - 10^6$ such tracks depending on the spatial mesh dimensions.

4.2.3 Solution of transport equation with characteristics method

The main idea behind the method of characteristics (MOC) is to solve the differential form of the transport equation following the straight lines (characteristics or tracking lines). The neutron trajectories will be tracked in the local coordinates system where an observer is traveling in the neutron direction. (Askew, 1972) The basic transport operator is then transformed into a total differential operator. The local multi-group characteristics equation is then given by

$$\left(\frac{d}{ds} + \Sigma^g(\vec{r} + s\hat{\Omega})\right) \Phi^g(\vec{r} + s\hat{\Omega}, \hat{\Omega}) = Q^g(\vec{r} + s\hat{\Omega}); \tag{4.6}$$

where s stands for the variable along the characteristics line. Assuming that the segment k crosses region j, the local relationship between the incoming and outgoing angular fluxes is given by

$$^{\text{out}}\Phi_j^g(k) = \,^{\text{in}}\Phi_j^g(k) + \left[\frac{Q_j^g}{\Sigma_j^g} - \,^{\text{in}}\Phi_j^g(k)\right] E(\Sigma_j^g L_k); \tag{4.7}$$

where $E(x) = 1 - e^{-x}$.

The average scalar flux in region j is calculated using

$$\phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \frac{1}{\Sigma_j^g V_j} \sum_i \omega_i \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g; \tag{4.8}$$

where $\Delta^g_{i,n,k} = {}^{\text{in}}\Phi^g_j(k) - {}^{\text{out}}\Phi^g_j(k)$ is the flux difference on segment k and $\pi_{i,n}$ is the weight associated with track $\vec{T}_{i,n}$. The usual Kronecker symbol δ is used here to ensure that each contribution is associated with the correct region flux component.

From this equation, we see that the iteration sweep over all characteristics involves the sequential evaluation of outgoing fluxes of Equation (4.7) on every single track and the addition of flux differences $\Delta^g_{i,n,k}$ for each track $\vec{T}_{i,n}$ in Equation (4.8).

4.2.4 Definition of the problem

The transport calculation for large systems is very expensive in terms of CPU time and computation resources. Therefore, the sequential run time increases at least linearly with problem size (number of regions for fixed number of energy groups). Typical large-scale problems can take weeks. (Dahmani, 2003a) In addition to that, the tracking data must be stored in a sequential binary file which size can reach hundreds of GB. Thus the storage resources can become prohibitive in addition to the related I/O operations which are often bottleneck for such applications. (Dahmani, 2003b) Moreover, conventional transport acceleration techniques, such as Diffusion Synthetic Acceleration (DSA) (Adams, 2002) are difficult, if not applicable, for MOC equations. Transport Synthetic Acceleration (TSA) methods have also been applied recently to MOC acceleration (Zika, 2000), but these are still inefficient for large-scale problems. In order to overcome such limitations, an alternative to improve the performance of the transport solvers is to use the high performance computing methods.

4.2.5 Parallelization of transport equation: Related work

Recently, the scientists working in nuclear engineering field have increasingly used parallel computers to solve problems that require large amounts of computing resources. Simulations involving advanced reactor design are now done using these new parallel algorithms.

The parallelization of the transport equation, based on Collision Probability method, obtained after distributing the energy variable in multi-group solver, as well as the usual spatial decomposition methods, have already been developed a few years ago (Qaddouri, 1996). Domain decomposition techniques have also been explored for large-scale transport calculations using the method of characteristics. The spatial decomposition of multi-assembly problems where neutron paths are directly connected when crossing assemblies exhibit limited speedup (5.3 on 8 processors) on shared memory Sun Enterprise4000 because of the tight coupling between the assemblies (Kosaka, 1999). The angular decomposition technique, where directions are distributed among a set of processors, has also been tested (Lee, 2000). Speedup obtained using this angular decomposition is of the order of 3.7 for 4 processors, and 6.8 for 8 processors. Other works were done in this context (Azmy, 1993; Sjoden, 1997). In these works, based on S_N method, the domain decomposition techniques are used for the spatial and/or the angular variables.

Almost all these techniques were restricted to two-dimensional problems; this is mainly due to the fact that PWR reactors can be accurately represented using 2D models. However, in CANDU reactors as in many newer designs, the geometry of the core is three-dimensional, with control mechanisms perpendicular to the fuel channels.

In the present work, we analyze the computational solutions for 3D large-scale transport calculation using the method of characteristics. We conceive an analytical model for the communication/computation ratio in order to be able to predict the specific performance on different kinds of architecture. The speedups obtained for up to 32 processors on different parallel machines are encouraging. Scalability analysis based on the isoefficiency function is applied to our parallel code when we increase the size of the problem and the number of processors.

4.3 Analytical model for the MCI parallel algorithm

4.3.1 Parallel algorithm and distribution of tasks

In this section, we show how the tracks are distributed in order to balance the charge on each processor and to avoid idle time that can decrease the overall performance. Our static load balancing techniques are based on calculation load estimations for each processor. First, we provide some metrics useful for our performance analysis.

4.3.1.1 Finest granularity level

In a parallel characteristics solver, the basic computation unit is focused on repeating the same sequence of calculations for each tracking line. The characteristics sweep on one single line represents the atomic operation for the solver from which we can construct the overall solution for each region, each angle, and each energy group. As we have seen before, the group variable can easily be treated using *data parallelism.* We now define the finest granularity level of operations that can be performed without any communication from the point of view of *task parallelism.* At this stage, the required local data is a collection of segment lengths L_k and the region numbers encountered along the line. The nuclear data needed are the local total cross section Σ_j^g and the scattering cross section from one group to itself $(\Sigma_s^{g\leftarrow g})_j$. No scattering matrices are necessary; this approach is important to preserve certain linearity in calculation and to ensure independence between the calculations performed on different tracks. The global solution is then the combination of these huge amounts of these atomic solutions.

The solution on a single track:

For a given track $\vec{T}$ (formed by K segments), we define the crossing points in local coordinates: $s_0, s_1, ..., s_K$ (Figure 4.2). Assuming a known flux Φ_0 entering the domain, for each segment of the track, the Equation (4.6) is solved. The angular flux at the end of each segment is recursively given by

$$\Phi_k = \Phi_{k-1} e^{-\tau_k} + Q_{N_k} \frac{(1 - e^{-\tau_k})}{\Sigma_{N_k}} \tag{4.9}$$

where Σ_{N_k} and Q_{N_k} are the local total cross section and isotropic source in the region N_k. τ_k is the total optical path when crossing the region N_k defined by

$$\tau_k = \int_{s_{k-1}}^{s_k} dt \Sigma(\vec{r} + t\hat{\Omega}). \tag{4.10}$$

Equation 4.9 represents the exponential attenuation of the neutrons when crossing different regions along the track.

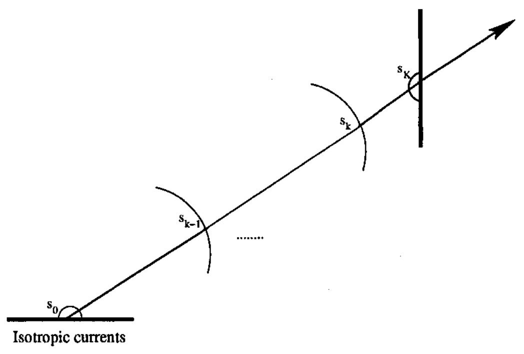


Figure 4.2 The neutron attenuation law across a track.

In practice, the number of tracks is much greater than the number of processors available. The parallelization of the solver is then done by grouping the tracks and each slave processor takes control of one of these groups. At each iteration step, a reduce/broadcast operation is used to recover every partial contribution to the angular fluxes; each processor has its own copy of the flux and the source array with a consistent unknown ordering. The scalar flux is then reconstructed on every single processor before the iteration procedure starts.

4.3.1.2 Calculation load and problem size

The *calculation load* on one processor is defined as the run time required by the processor in order to complete the computation. In our case, this time can be related to the number of floating point operations (*flop*) necessary for computing all local angular fluxes on a single track containing $\mathcal{K}_T$ segments. The atomic one-group time t_T is given by

$$t_T = 2t_\times + 2t_+ + \mathcal{K}_T\left(t_e + 5t_\times + 7t_+\right); \tag{4.11}$$

where t_+, $t_\times$ and t_e are the time required for an addition, a multiplication and an exponential evaluation respectively. Two options are available : either exponential calls, or the use of linear tables for interpolating exponential values (Dahmani, 2002)

The total serial time T_S spent by a single processor working on N_T tracks and N_G groups is then

$$\begin{aligned} T_S &= N_G N_T \left[2t_\times + 2t_+ + \sum_{T=1}^{N_T} \mathcal{K}_T\left(t_e + 5t_\times + 7t_+\right)\right] \\ &\sim N_G N_T \left[4 + \overline{\mathcal{K}}_T\left(\overline{\tau}_e + 12\right)\right] t_{op}; \end{aligned} \tag{4.12}$$

where $\overline{\mathcal{K}}_T$ is the mean number of segments for the N_T tracks, and t_{op} is the mean time necessary for one flop (this is a machine-dependent constant). The variable $\bar{\tau}_e$ stands for the mean number of flop for exponential evaluation.

The size of the problem W is defined as the measure of the number of basic operations needed to solve the problem. Here, we can assume that W is proportional to serial time needed to solve the problem

$$T_S = W t_{op}. \tag{4.13}$$

In the following, we assume $t_{op} = 1$. The size of the problem is nearly proportional to $N_G N_T \overline{\mathcal{K}}_T$. In order to achieve high performance on parallel machines, the calculation load must be distributed *fairly* among several processors. This implies that the atomic-level load of each track must be taken into account before statistically dispatching a group of tracks to a processor.

4.3.1.3 Fair distribution of the tracks

We now study the mechanism by which the atomic tasks are assigned to run on different physical processors. In order to keep our model general enough to cover most parallel architectures, we will decouple the running '*processes*' from the physical processors (Leopold, 2001). The allocation of tasks to processes is done by a *mapping:*

$$\begin{aligned} M : \mathcal{T} &\rightarrow \mathcal{P} \\ (i,n) &\mapsto p \end{aligned}$$

This mapping goes from $\mathcal{T}$, the domain of characteristics onto, $\mathcal{P}$, the range of processes. As explained above, the tracking lines $\vec{T}_{i,n}$ are identified by direction i and starting point n. A global numbering of tracks has no particular use, so we will assume that tracks are numbered from 1 to N_T in the order they are generated. This global numbering scheme $l(i,n)$ will be assumed when needed in the following.

Since the calculation time for a track linearly depends on its number of segments, the process calculation load defined as

$$L(p) = \sum_{(i,n)\in M^{-1}(p)} t_T(i,n); \tag{4.14}$$

is not necessarily uniform, even if we put the same number of tracks on each process. An evenly distributed number of segments on each processor will be the major criteria to build our load balancing algorithm. A good mapping should take into account the total calculation load of each batch of tracks. The options currently implemented to distribute tracks are called: SPLT, STRD and ANGL (Dahmani, 2004; Dahmani, 2003a).

Several tests were done that show the STRD option is better balanced than the two others. This technique is based on a round-robin distribution of tracks. It is a cyclic mapping of all tracks to a single processor, such that $p \equiv l(i,n) \bmod P$. The first process takes the tracks numbered $1, 1+P, 1+2P, \ldots$, the second process $2, 2+P, 2+2P, \ldots$, etc. Using this option, it is statistically unexpected to have two batches of tracks having a substantial difference in their average number of segments. (Dahmani, 2002)

4.3.1.4 Parallel algorithm

At the solver level, each processor takes control of batches of tracks to accumulate contributions to average angular fluxes; a partial sum of flux differences is accumulated by region and energy group in a flux scalar array. At each inner iteration, the processors communicate their results to all other processes by using a reduce/broadcast procedure. In Figure 4.3, we show how the partial sums PS_p are collected from all processors. These sums are defined by

$$PS_p = \Delta_p \phi_j^g = \frac{1}{\Sigma_j^g V_j} \sum_{(i,n) \in M^{-1}(p)} \omega_i \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g; \tag{4.15}$$

where ϕ_j^g is the flux for region j and energy group g. After the reduction operation, at the end of the inner loop, all processes have the same copy of the scalar flux

$$\phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \sum_{p=0}^{P} \Delta_p \phi_j^g. \tag{4.16}$$

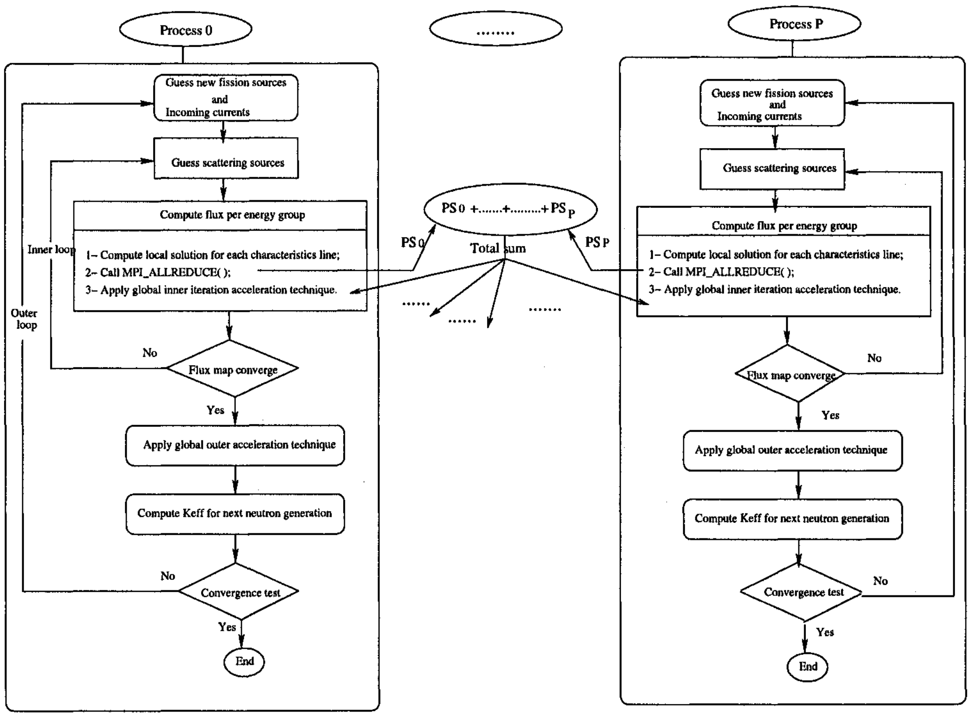

Process 0
....
Process P
Guess new fission sources and Incoming currents
Guess scattering sources
Compute flux per energy group
1– Compute local solution for each characteristics line;
2– Call MPI_ALLREDUCE( );
3– Apply global inner iteration acceleration technique.
Inner loop
Outer loop
No
Flux map converge
Yes
Apply global outer acceleration technique
Compute Keff for next neutron generation
No
Convergence test
Yes
End
PS 0 +.......+.........+ PS P
PS 0
PS P
Total sum
......
......
....

Figure 4.3 Parallel iterative scheme

4.3.2 Modeling the overhead function

The overhead function T_0 is defined as the total time collectively spent by all processors in addition to the one required by the sequential algorithm for solving the same problem on single processor. The major sources of overhead, in a well-parallelized code, are the interprocessor communications and the load imbalance. If we assume that the tasks in our parallel program are evenly distributed, then the overhead function will be dominated by the communications overhead. The communication time depends on the amount of data exchanged at each step of the algorithm. Here, we consider the most general case where we exchange parts of the current/flux vector of dimension $(N_R + N_S + 1) \times N_G$, where N_R and N_S stand for the number of regions and the number of external surfaces respectively. In the following, we set $N_{RS} = N_R + N_S + 1$. In that case, we have

$$T_0 = \alpha(P)t_s + [t_h + N_{RS}N_G t_w]\,\beta(P); \tag{4.17}$$

where

- t_s is the startup time for data transfer, it is the time required to handle a message at the sending processor. This delay is occurred once for a single message transfer;
- t_w is the per-word transfer time, inversely proportional to the available bandwidth between nodes;
- t_h is the per-hop time, time taken by the header of the message to travel between two directly-connected processors in the network. In general this time is very small. In the following, we take $t_h = 0$;
- α and β are functions of the number of processors. They take into account the communication paths between the processors, depending on the network

topology and the message passing implementation.

If we use P processors to do the parallel computation and we assume that all processors have the same load, we can write the relationship between the serial time T_S and the parallel time T_P as

$$T_0 = PT_P - T_S \tag{4.18}$$

4.3.3 Modeling the isoefficiency function

Using the Equation (4.13), the parallel execution time T_P can be expressed as function of the problem size, overhead function, and the number of processors as

$$T_P = \frac{W + T_0(W,P)}{P}. \tag{4.19}$$

The parallel speedup S is defined as the ratio of the serial execution time to the parallel execution time

$$S = \frac{T_S}{T_P} = \frac{WP}{W + T_0(W,P)}. \tag{4.20}$$

The efficiency of the parallel system is evaluated as the ratio of the speedup to the number of processors

$$\mathcal{E} = \frac{S}{P} = \frac{1}{1 + \frac{T_0(W,P)}{W}}. \tag{4.21}$$

For scalable parallel systems, efficiency can be maintained at fixed value ($0 < E \leq 1$) if the ratio T_0/W is maintained at a constant value.

$$\frac{T_0(W,P)}{W} = \frac{1-E}{E}; \tag{4.22}$$

Let $K = E/(1-E)$ be the parameter depending on the efficiency to be fixed constant. Equation (4.22) can then be written as

$$W = KT_0(W,P). \tag{4.23}$$

This equation leads to the relationship between the problem size and the number of processors. It defines the isoefficiency function of the parallel system. For our application, in the general case where no specific parallel architecture is chosen, the isoefficiency function, $W(P)$, is expressed as

$$W(P) = K\left[\alpha(P)t_s + N_{RS}N_G t_w \beta(P)\right]. \tag{4.24}$$

This function is useful to determine the rate at which the problem size must increase with respect to the number of processors to keep the efficiency fixed.

4.4 Parallelization methodology

4.4.1 MPI Implementation

To perform the communications between processes participating in calculation, we use the Message Passing Interface (MPI) (Groop, 1994). In the following, for analyzing our model, we make the following assumptions:

- all processors are directly connected (switched networks);

- the nodes are homogeneous: all the processors have similar power;
- we have a perfect load balance: $\forall p \bullet L(p) = T_S/P$.

All MPI communications require a communicator and the MPI processes can only communicate if they share the communicator. The MPI standard includes a whole set of routines that can be used for collective communications. When implementing our solver, we use the variant of the reduce operations where the result is returned to all processes in the communicator. On that case, the standard specification requires that all processes participating in these operations receive identical results using a single function call:

```
MPI_Allreduce( partF , totF , Ng*Nrs , MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD )
```

Note that these all-reduce operations can be transformed into a one-node reduce, followed by a broadcast of the resulting values. However, a better performance is obtained if we take into account the possibility to send several synchronous point-to-point messages on the local switched network.

If we consider a perfect butterfly network [5], we can minimize the number of synchronized steps that are needed to add the contribution to the scalar flux. The MPI_Allreduce routine is implemented in the last version of MPICH (1.2.5) using an optimized recursive doubling algorithm where the processes exchange data with each other in $log(P)$ steps.

For a power-of-two number of processors, each process gets the final results in $log(P)$ steps. This communication time is

[5] interconnection network composed of $log(P)$ levels and $Plog(P)$ switches.

$$t_{com} = log(P)t_s + N_{RS}N_G t_w log(P); \tag{4.25}$$

This interprocessor communication contributes as $P \times t_{com}$ to the overhead function. Therefore, the total overhead function when using P processors is given by

$$T_0 = Plog(P)t_s + N_{RS}N_G t_w Plog(P); \tag{4.26}$$

And the efficiency for the MCI/MPI implementation is then

$$\mathcal{E} = \frac{1}{1 + \frac{Plog(P)[t_s + N_{RS}N_G t_w]}{N_G N_T\left[4 + \overline{\mathcal{K}}_T(\bar{\tau}_e + 12)\right]}}. \tag{4.27}$$

4.4.1.1 Generalization of the model

In all these equations involving the communication time, we consider implicitly that the machines used are composed of nodes with only one processor. However, for machines containing nodes with several processors, the communication overhead function must take into account these two mechanisms used for the communication: the shared memory when processors belong to the same node and the network communication when the processors belong to the different nodes. Let us assume that there are m processors per node. If the recursive doubling algorithm for communications is applied, the first $log(m)$ communication steps will not involve message passing between nodes. Thus, the communication overhead function becomes

$$\begin{aligned} T_0 = P\left(log(P) - log(m)\right)\left[t_{s,net} + N_{RS}N_G t_{w,net}\right] \; + \\ Plog(m)\left[t_{s,sh} + N_{RS}N_G t_{w,sh}\right]; \end{aligned} \tag{4.28}$$

where $t_{s,net}$ and $t_{w,net}$ are the startup time and the per-word time for the message passing network communication. $t_{s,sh}$ and $t_{w,sh}$ are the startup time and the per-word time for shared memory communication. In the following, for the seek of simplicity, we generally continue use the Equation (4.26). However, numerical results are provided for the corrected Equation (4.28) in the case of hybrid machines.

4.4.2 Isoefficiency analysis

From Equation (4.24), we can write

$$W(P) = KPlog(P)\left[t_s + N_{RS}N_Gt_w\right]; \tag{4.29}$$

The isoefficiency function varies asymptotically as $\Theta(Plog(P))$. Thus if the number of processors is increased from P to P', then the problem size must be increased by a factor of $(P'log(P'))/(Plog(P))$ to maintain the same efficiency as on P processors. We can do the same thing by using Equation (4.27), the parameter K is given by

$$K = \frac{N_GN_T\left[4 + \overline{\mathcal{K}}_T\left(\bar{\tau}_e + 12\right)\right]}{Plog(P)\left[t_s + N_{RS}N_Gt_w\right]}. \tag{4.30}$$

Since $t_s << N_{RS}N_Gt_w$, the dominating term in the overhead function is then $Plog(P)N_{RS}N_Gt_w$. Then K can be approximated by

$$K \sim \frac{\mathcal{D}}{Plog(P)}; \tag{4.31}$$

where the factor $\mathcal{D} = \left(N_T\overline{\mathcal{K}}_T\left(\bar{\tau}_e + 12\right)\right) / \left(N_{RS}t_w\right)$ depends only on the dimensionality of the problem.

In order to maintain K constant when increasing the number of processors by a factor of P'/P, $\mathcal{D}'/\mathcal{D}$ must be equal to $(P'log(P'))/(Plog(P))$. Actually, the number of tracks depends on the number of regions and d_T the density of tracks (tracks/cm^2); and the mean number of segments depends on the number of regions: $N_T = N_T(N_R, d_T)$ and $\overline{\mathcal{K}}_T = \overline{\mathcal{K}}_T(N_R, N_T)$.

The variation of the problem size affects the overhead function by varying the size of the message. For a fixed problem size, the efficiency varies asymptotically as $\Theta(\frac{1}{1+\frac{Plog(P)}{\mathcal{D}}})$. To keep the efficiency at acceptable values, it is necessary to find the best trade-off between the problem size and machine size (number of processors). In other words, the algorithm is scalable **if** the factor $\mathcal{D}$ increases by a factor of $(P'log(P'))/(Plog(P))$ when increasing the number of processors from P to P' leading to an increase of the speed up by a factor of P'/P.

4.5 Results and discussion

All the results presented in this section are obtained for CANDU-6 3D supercell problems. A supercell is composed by two horizontal bundles containing the Uranium oxide fuel, and one vertical absorber rod (see Figure III.3). The transport equation is usually solved by critical buckling search with first order leakage treatment. (Roy, 1994) For most of our tests, we use 24 discrete directions and 89 energy groups. The spatial meshing and the track density can be changed as needed to ensure accuracy of the multigroup solution. In all cases presented here, we use the STRD option for the load balancing.

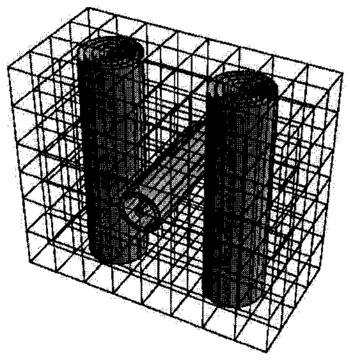


Figure 4.4 CANDU-6 supercell

4.5.1 Program portability

Our parallel program is designed to run on different parallel systems: machine architecture, operating system and MPI implementation which can vary as long as these are compliant with the MPI standard. In Table 4.1, we show the speedup obtained with three different parallel systems.

Tableau 4.1 Speedups using different MPI parallel systems

			Number of processors			
Machine	OS	implemetation	2	4	8	16
Beowulf cluster ($P = 16$)	Linux	LAM	1.95	3.73	6.58	12.11
Cluster of SMPs ($P = 32$)	Linux	MPICH	1.98	3.92	7.93	15.29
NUMA ($P = 16$)	IRIX	SGI MPI	1.98	3.87	7.68	13.85
Cluster of NUMAs ($P = 64$)	Linux	MPICH	1.97	3.76	7.54	13.22

4.5.2 Performance analysis

In the following subsections, we present our performance analysis tests for two machines: a cluster of SMP computer (*Charybde*) with 32 processors and a cluster of NUMAs (*Hydra*) with 64 processors, both using the MPICH implementation of MPI.

4.5.2.1 Performance tests

In order to validate the performance of the model, empirical data was gathered to complete the machine specific aspects of the model.

The actual floating point performance of the system is evaluated using the "flops.c" benchmark developed by Al Aburto (Aburto). We choose the MFLOPS[4] result, which does not involve any floating point divisions (only the addition and the multiplication are performed), to calculate the time per flop t_{op} because it reflects more accurately the distribution of floating point operations for our application. The evaluation of the average number of flops per exponential τ_e is done by using a simple benchmark. Random numbers were generated and their exponential evaluation timed. This evaluation makes sense because the total cross section varies from one energy group to the other and is not predictable along a characteristic. An average time, τ_e, was derived from theses results, which was converted to flops using t_{op} (Table 4.2). For modern NUMA nodes, pipelining makes it difficult to count the calculation load in terms of counting the exact number of flops. However, we shall see that this model still gives a crude estimate of the load.

These data were used to calculate both the startup time for data transfer ($t_{s,net}$ and $t_{s,sh}$) and the per-word transfer time ($t_{w,net}$ and $t_{w,sh}$). Communication timing was collected using the MPI benchmark suite (Table 4.3).

Tableau 4.2 Empirical data for two machines

Machine		(μs)	(flops)
Charybde	t_{op}	0.00445	1
	τ_e	0.4225	94.885
Hydra	t_{op}	0.000827	1
	τ_e	0.028	34.5

Tableau 4.3 Communication timing for two machines

Machine	$t_{s,net}(\mu s)$	$t_{s,sh}(\mu s)$	$t_{w,net}(\mu s)$	$t_{w,sh}(\mu s)$
Charybde	17.1	2.11	0.08	0.079
Hydra	13.3	0.6	0.0612	0.01

4.5.2.2 Model validation

Numerical tests done for various problems [6] are summarized in the following table:

Tableau 4.4 Problem size parameters for different tests

Test	N_R	N_S	N_{RS}	d_T (track/cm^2)	N_T
T_1	46	35	82	10	275262
T_2	368	280	649	10	1101048
T_3	560	376	937	10	1646825
T_4	944	568	1513	4	1078084
T_5	1136	664	1801	4	1293560
T_6	1712	952	2665	4	1928506
T_7	2480	1336	3817	4	6965508

Efficiencies obtained by the model are compared to those given after running the parallel code. In Figures 4.5 and 4.6, we show the results obtained on machine *Charybde* for two different problem sizes (T_3 and T_4). The curves show a good agreement between the experimental data and the analytical model. Small divergences are observed for $P = 32$ when all the cluster nodes are used. This can be

[6]The problem corresponding to the test T_1 is shown in Figure 4.4

explained by the fact that node 01 is also the server for the cluster.

In Figures 4.7 and 4.8, results on *Hydra* using over 64 processors are reported. The results show small discrepancies between the calculated and measured efficiencies. This could be due to the fact that for this specific machine the communication overhead is not very well estimated or the machine communication parameters measured (especially t_w) are not accurate enough.

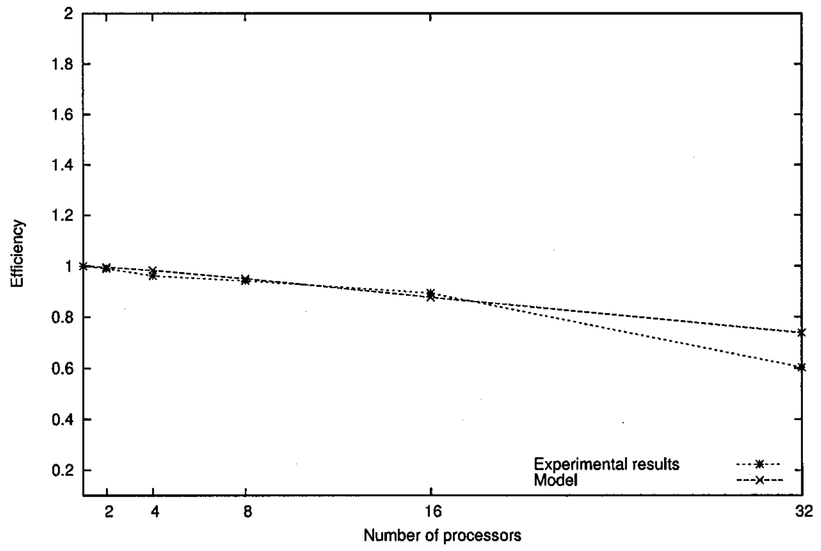


Figure 4.5 The measured and predicted efficiency on *Charybde* for (T_3 with $N_{RS} =$ 937)

In our model, the overhead function T_0 grows faster than $\Theta(P)$. Following an argument explained in (Gupta, 1993), from Equation (4.19), the term W/P keeps decreasing with increasing P, while the term T_0/P is increasing. Therefore, the overall T_p will first decrease and then increase with increasing P. It means that $\exists\ P_0$, for fixed problem size W, that minimize T_p as

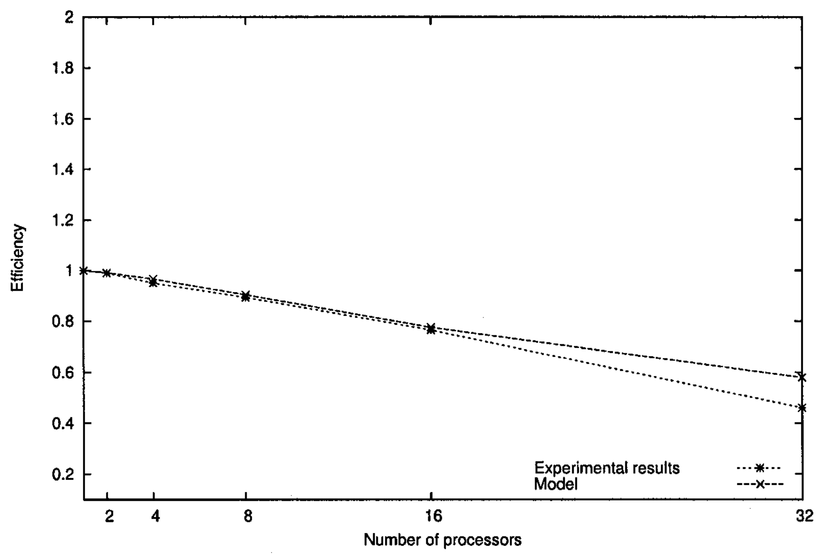


Figure 4.6 The measured and predicted efficiency on *Charybde* for (T_4 with $N_{RS} = 1513$)

$$\frac{d}{dP}\left(\frac{W + T_0}{P}\right) = 0; \tag{4.32}$$

In Figures 4.9 and 4.10, the T_p and T_0 functions are presented for two test cases (T_3 and T_4). It shows that T_p slightly varies when varying the problem size while the differences in T_0 are more important because of the increase of the message size.

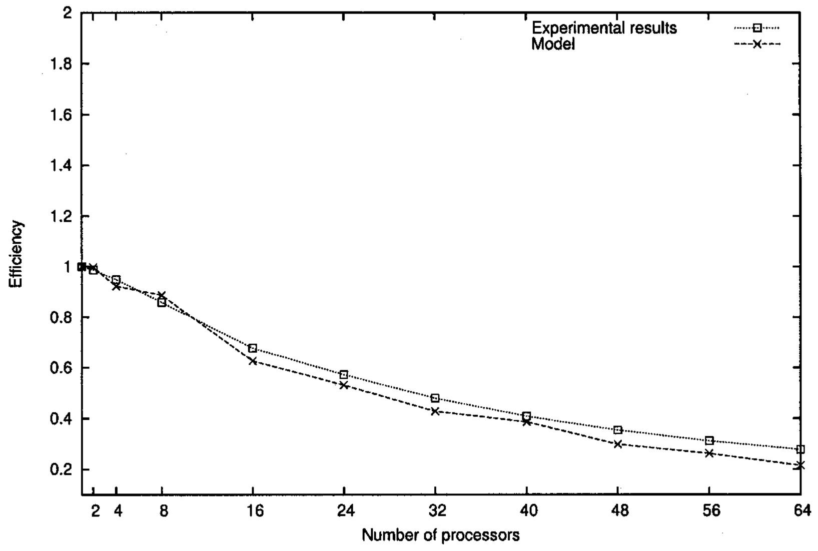


Figure 4.7 The measured and predicted efficiency on *Hydra* for (T_4 with $N_{RS} = 1513$)

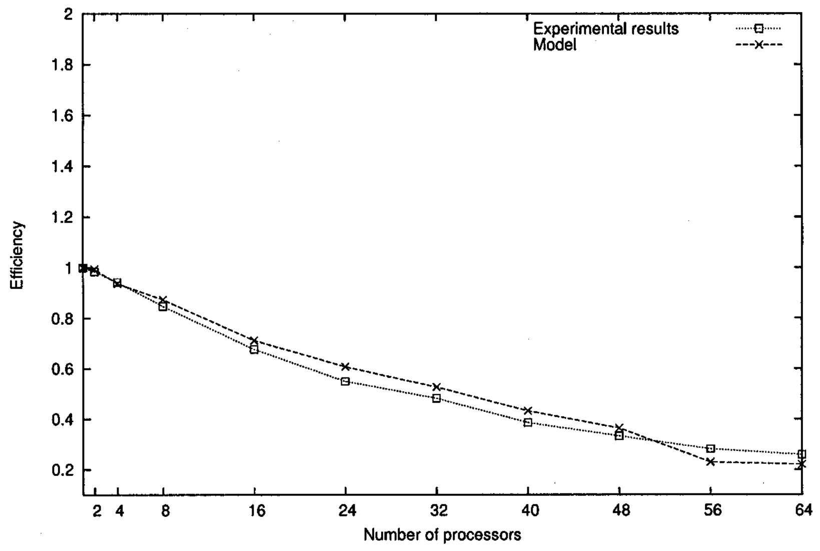


Figure 4.8 The measured and predicted efficiency on *Hydra* for (T_6 with $N_{RS} = 2665$)

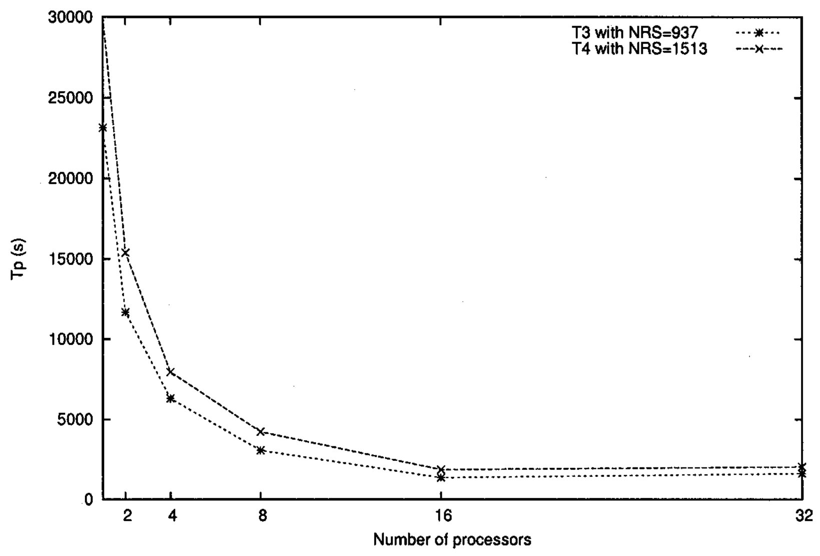


Figure 4.9 Parallel time T_P using different problem sizes

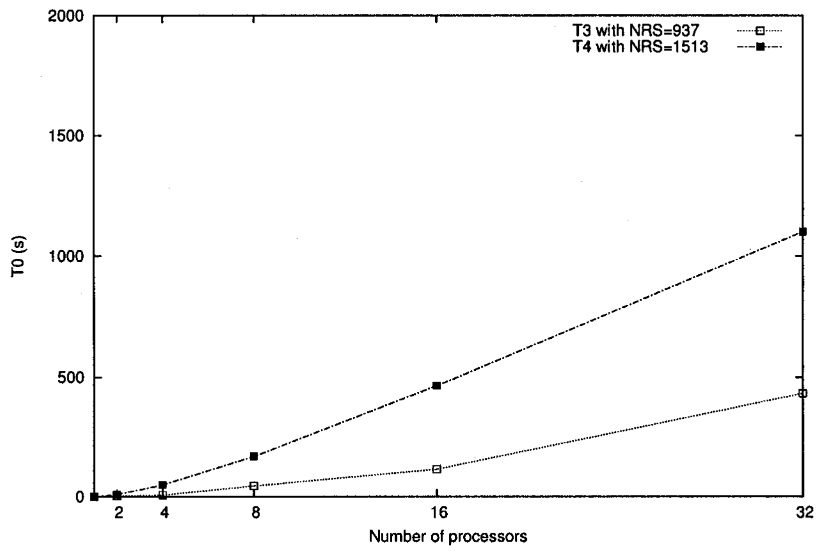


Figure 4.10 The overhead function T_0 for different problem sizes

4.5.2.3 Scalability

We now study the behavior of the efficiency when varying some main parameters of the problem. All the following results are obtained using *Charybde* machine. For a given parallel machine, the efficiency function depends on the number of energy groups, the number of regions and surfaces, the number of tracks and the number of processors

$$\mathcal{E} = \mathcal{E}(N_G, N_{RS}, N_T, P) \equiv \mathcal{E}(\mathcal{D}, P); \tag{4.33}$$

where $\mathcal{D} = \left(N_T \overline{\mathcal{K}}_T \left(\bar{\tau}_e + 12\right)\right) / \left(N_{RS} t_w\right)$ was defined as the dimensionality of the problem.

In the following tests, the number of energy groups is kept constant (still equal to 89). If we fix $\mathcal{D}$ and we vary the number of processors, the efficiency function varies asymptotically as

$$\mathcal{E} \sim \frac{1}{1 + \frac{Plog(P)}{\mathcal{D}}}. \tag{4.34}$$

If $\mathcal{D} \to \infty$ then $\mathcal{E} \to 1$. For $P > 16$, the efficiency decreases faster. Thus, for a fixed problem size, the peak performance is reached in this interval where the parallel run time is minimum.

Now, let us vary the number of regions by choosing different mesh sizes (46, 368, 560, 944, 1136, 1712 and 2480 regions). Obviously, the number of tracks and the mean number of segments also vary and so does the factor $\mathcal{D}$. This means that the computational time and the communication time (the size of the message) change at the same time. The comparison between the measured and predicted values of the efficiency are reported in Tables 4.5 and 4.6. The results are very close except for

$P = 32$ for reasons explained above. The small discrepancies are also due to the load imbalances (even if the tracks are distributed evenly over all processors, the number of segments per track is not automatically the same) and the synchronizations that we did not take into account in our analytical model. However, the results show the dependence of the efficiency on the $\mathcal{D}$ factor. The efficiency is very sensitive to the variation of $\mathcal{D}$ especially for $P \geq 8$ in all cases.

For T_1 and T_5 of the Tables 4.5 and 4.6, the efficiency values are very close. Those values correspond to almost the same $\mathcal{D}$ factor, though N_{RS} and N_T are different. For $P \geq 8$, the efficiency values are more stable as $\mathcal{D}$ increases.

Tableau 4.5 Predicted efficiency by varying the problem size (values in bold are similar to Table 4.6)

Problem size parameters				Number of processors				
Test	N_{RS}	N_T	$\mathcal{D}$	2	4	8	16	32
T_1	82	275262	49222	**0.992**	**0.971**	**0.916**	**0.801**	**0.616**
T_2	649	1101048	75540	0.995	0.983	0.950	0.876	0.738
T_3	937	1646825	95647	0.996	0.986	0.958	0.895	0.773
T_4	1513	1078084	45903	0.991	0.966	0.903	0.775	0.578
T_5	1801	1293560	49747	**0.992**	**0.971**	**0.916**	**0.802**	**0.617**
T_6	2665	1928506	55720	0.993	0.971	0.917	0.806	0.625
T_7	3817	6965508	140514	0.997	0.987	0.962	0.906	0.794

Tableau 4.6 Measured efficiencies by varying the problem size (values in bold are similar to Table 4.5)

Problem size parameters				Number of processors				
Test	N_{RS}	N_T	$\mathcal{D}$	2	4	8	16	32
T_1	82	275262	49222	**0.992**	**0.974**	**0.913**	**0.802**	**0.634**
T_2	649	1101048	75540	0.989	0.97	0.932	0.851	0.603
T_3	937	1646825	95647	0.989	0.962	0.942	0.893	0.61
T_4	1513	1078084	45903	0.988	0.951	0.8927	0.765	0.46
T_5	1801	1293560	49747	**0.991**	**0.972**	**0.908**	**0.804**	**0.602**
T_6	2665	1928506	55720	0.991	0.964	0.897	0.801	0.607
T_7	3817	6965508	140514	0.994	0.975	0.946	0.892	0.688

For a fixed number of regions, we now increase the number of tracks by varying the user-input value for the density of tracks d_T. Computation time increases when increasing the number of tracks. However, a nice property of our parallel code is that the size of messages is kept constant because N_{RS} is constant. According to the isoefficiency analysis (see section 4.3), to maintain K constant when increasing number of processors from P to P', $\mathcal{D}$ must be increased by the factor of $(P'log(P'))/(Plog(P))$ which is equivalent to increase the ratio N'_T/N_T by the same factor because N_{RS}, $\bar{\mathcal{K}}_T$, $\bar{\tau}_e$ and t_w remain constant. In other words, since, for fixed N_{RS}, N_T depends only on the density of tracks d_T, we should have

$$\frac{N'_T}{N_T} = \frac{d'_T}{d_T} = \frac{P'log(P')}{Plog(P)}; \tag{4.35}$$

or

$$d'_T = d_T \frac{P'log(P')}{Plog(P)}. \tag{4.36}$$

To test this behavior, we did some experimental tests. For fixed $N_{RS} = 82$, we vary the density of tracks d_T in such a way that the ratio to the new value respects Equation (4.36). Thus, we increase d_T by a factor of 4 for $P = 4$ and by 2.5 for $P = 16$. The results in Table 4.7 show that the efficiency values are improved in both cases and the efficiency is kept almost constant (the first two values and the last two values).

Tableau 4.7 Efficiency by varying d_T

d_T	10	40	10	25
P	2	4	8	16
$\mathcal{E}$	0.992	0.984	0.913	0.892

Following the same reasoning, in order to keep the efficiency constant by changing the machine size from 2 to 32 processors the density of tracks must be multiplied by 60. Note that, since the tracks are stored in sequential binary files, increasing the density of tracks is equivalent to increase the file size by almost the same factor. Therefore, it is practically impossible to scale our application on such a large number of processors by varying only the density of tracks because of the machine storage capacity (I/O bound). In all cases, the increase of d_T generally improves the computational precision, but it does not increase the physical dimensionality of the problem. In large-scale 3D problems, the ideal way to scale is to find a good combination when varying the global dimensionality of the problem, $\mathcal{D}$, in order to keep the efficiency constant when varying the number of processors.

4.6 Conclusion

Nuclear engineering has introduced many industrial strength applications, namely in order to follow the neutron behavior in reactor cores. The parallel strategy described in this paper allows reactor physicists to accurately solve neutron transport problems over large-scale domains (whole reactors or large parts of them). In the coming years, day-to-day runs for core follow-up may need the direct solution over large 3D problems to get accurate results in a reasonable amount of CPU time.

In this paper, we have introduced a computational method to solve such problems. An analytical model for the communication/computation processes, on which our parallel algorithm is based, has been presented. The validation of the model using empirical data shows its good accuracy. Using this model, we have extended our study to the scalability tests. Based on the system efficiency responses, we have done several tests by increasing the dimensionality of the problem and/or the number of processors. Results show that by choosing an adequate combination of the problem size parameters, we can keep the efficiency almost constant. These tests, done using an cluster of SMPs with 32 processors, show that the algorithm is conditionally scalable.

The method of characteristics has been shown to provide a linear and scalable approach to solve large-scale transport problems. Moreover, this work could be extended in the context of heterogeneous architectures and for a very large number of processors. However, in that case, the static load balancing should be revised to take into account the CPU resources. For very large 3D problems, since tracks are stored in sequential files and the solver needs to access these data, we are faced with some I/O and memory bounds. In order for parallel computers to be fully usable in solving such large-scale problems, the I/O performance must also be scalable and balanced with respect to the CPU and communication performance. Thus, the implementation of parallel I/O or the elimination of the tracking file needed in the solver shall be considered in future works.

Acknowledgments

We gratefully thank Benoît Morin and Louis-Alexandre Leclaire who ran different standard benchmarks in order to get the machines specific performance parameters. The anonymous reviewers also helped improve the quality of the paper. This work has been carried out partly with the help of a grant from the Natural Science and Engineering Research Council of Canada.

CHAPITRE 5

AN EFFICIENT PRECONDITIONING TECHNIQUE USING KRYLOV SUBSPACE METHODS FOR 3D CHARACTERISTICS SOLVERS

Auteurs: M. Dahmani, R. Le Tellier, R. Roy and A. Hébert
Revue: Annals of Nuclear Energy

Abstract

The Generalized Minimal RESidual (GMRES) method, using a Krylov subspace projection, is adapted and implemented to accelerate a 3D iterative transport solver based on the characteristics method. Another acceleration technique called the Self-Collision Rebalancing technique (SCR) can also be used to accelerate the solution or as a left preconditioner for GMRES. The GMRES method is usually used to solve a linear algebraic system ($Ax = b$). It uses $\mathcal{K}(r^{(o)}, A)$ as projection subspace and $A\mathcal{K}(r^{(o)}, A)$ for the orthogonalization of the residual. This paper compares the performance of these two combined methods on various problems.
To implement the GMRES iterative method, the characteristics equations are derived in linear algebra formalism by using the equivalence between the method of characteristics and the method of collision probability to end up with a linear algebraic system involving fluxes and currents. Numerical results show good performance of the GMRES technique especially for the cases presenting large material heterogeneity with a scattering ratio close to 1. Similarly, the SCR preconditioning sligthly increases the GMRES efficiency.

5.1 Introduction

The method of characteristics (MOC) solves the differential form of the neutron transport equation by following the characteristics of the system (tracking lines) which simulate the neutron paths (Askew, 1972). The scalar flux per region and energy group is constructed by collecting all mean angular fluxes in terms of the entering angular flux and the source inside the region. The sequential 3D characteristics solver MCI was developed to treat supercell problems with isotropic boundary conditions. This technology was found accurate with respect to several transport problems specific to CANDU reactors, where reactivity devices are perpendicular to the fuel channels (Wu, 2003a). It has been shown that for relatively small problems with a limited number of regions, the CPU times observed with the sequential MCI solver are similar to ones of the EXCELL module of DRAGON (Marleau, 1997) (which is based on Collision Probability (CP) method) and with almost the same number of outer iterations. Nevertheless, for large problems the MCI solution is slowed down by the current convergence due to flux/current initialization (flat flux). However, several acceleration techniques have been developed which are particular to the method of characteristics. CACTUS has used the same algorithm for many years: the energy group rebalancing algorithm (Halsall, 1998). This algorithm consists in solving a homogeneous problem at the end of every iteration and rescaling the characteristics flux according to the homogeneous solution. Larger spatial problems benefit less from the energy rebalancing of the homogeneous calculation because the spatial dependence of the flux solution converges much more slowly. Zika and Adams employ a Transport Synthetic Acceleration (TSA) in their long characteristics code (Zika, 2000). Like others synthetics methods, the idea is to solve a low-order approximation which is represented using a simplified transport operator in order to accelerate the high-order approximation which is the original transport operator. The low-order equation used in TSA is a modified transport problem in which the scattering cross section is artificially reduced. The low-order equation is also solved by a long characteristics method but using a cruder angular quadrature and a coarser ray spacing than the high order problem. These two simplifications significantly reduce the number of unknowns in the low order equations and result in considerable savings in computational cost.

Unfortunately, for several categories of problems restriction and prolongation operations must be defined in order to map between the high and low order boundary angular grids. The Angular Dependent Rebalance (ADR) iteration method was studied when applying to the Step Characteristic (SC) scheme for transport problems in equilateral triangular meshes where the unknowns are surface-averaged and the volume-averaged quantities (Hong, 1999) The rebalancing factor is only defined on the edges of the triangular meshes and its angular dependency is assumed to be uniform for each sextant. However, it would be very difficult to generate the ADR method for an arbitrary geometry and the accuracy of the SC scheme is not very good. In the environment of the interface-current code TDT (part of APOLLO2 (Loubière,1999)), Sanchez and Chetaine have developed a characteristics method for unstructured two-dimensional geometries (Sanchez, 1999). Based on piecewise uniform and isotropic approximations for cell entering and exiting fluxes they have developed a synthetic acceleration technique for their characteristics method. However, the synthetic acceleration equations are nonsymmetrical and the size of the system of equations is very large. There are two methods for the inner iteration acceleration in the MCCG code (Method of Characteristics in Complicated Geometry) (Suslov, 1994): the most of the standard acceleration techniques cannot be directly used (namely, diffusion synthetic acceleration cannot be directly used due to their restricted geometrical application) or were not found efficient enough (Wu, 1999).

To accelerate the solution of the MCI, several numerical acceleration methods were tested. The adapted Self-Collision Rebalancing (SCR) method was shown to give good performance results (Wu, 2003b). This technique is based on the equivalence theorem between the CP and MOC methods, The SCR technique uses the collision probabilities from one region to itself in order to rebalance the energy distribution of the scalar flux for each region separately. The Track Merging Technique (TMT) can also be used for numerical acceleration. The TMT is designed to reduce the number of tracking lines by merging the two neighboring tracking lines crossing the same regions in the same order into one line associated with the sum of the weights of each of them (Wu, 2003b).

Recently, remarkable progresses has been made in solving linear systems with sparse non-symmetric matrices using algebraic iterative methods because of the introduction of methods based on Krylov subspaces (Saad, 1986). Consequently, the use of these methods has been increasingly applied in many different fields. In neutron transport applications, many researchers have been using these methods either as integrated solvers or to accelerate the solution (Patton, 2002; Hanshaw, 2003; Warsa, 2003). In all these works, the Krylov-based subspaces methods are combined with the S_N method. Unlike the S_N method, the solution of the characteristics equations does not end up with a solution of a direct linear algebraic system which is not trivial for the implementation of these techniques. Thus, the derivation of the characteristics equations in operator form is needed. This is done by using the equivalence between the characteristics method and the CP method. The solution of the characteristics equations is then obtained by solving a linear algebraic system involving fluxes and currents.

In this paper, we investigate the application of the GMRES method, using a Krylov subspace, as an accelerator for 3D sequential characteristics solver. The SCR technique is also used either for the acceleration or as a left preconditioner for the system. The remainder of the paper is organized as follows. In section 2, we present a 3D characteristics formalism with brief review of the SCR technique. In section 3, a brief overview of the methods based on Krylov subspaces used to solve the linear algebraic systems, GMRES iterative method and the computational schemes used for the implementation are discussed. Section 4 is dedicated to the derivation of the characteristics equations in operator form and the implementation of the GMRES method to accelerate the solution. We will show also how the `SCR` can be used as left preconditioner for the `GMRES` system. Finally, numerical results showing the performance of these acceleration techniques will be reported.

5.2 The 3D characteristics formalism

The multi-group isotropic transport equation to be solved to obtain the flux from an isotropic source is

$$B^g(\vec{r},\hat{\Omega})\Phi^g(\vec{r},\hat{\Omega}) = Q^g(\vec{r}); \tag{5.1}$$

where

- $\vec{r}$ is a spatial point in the domain D;
- $\hat{\Omega}$ is the solid angle;
- g is the energy group;
- $B^g(\vec{r},\hat{\Omega}) = \hat{\Omega}\cdot\vec{\nabla} + \Sigma^g(\vec{r})$ is the transport operator in group g;
- $\Sigma^g(\vec{r})$ is the total cross section in group g;
- $Q^g(\vec{r})$ is the isotropic source in group g;
- g is the group index with G the total number of groups.

The source in Eq.(5.1) is composed of two terms, the isotropic scattering and fission sources:

$$Q^g(\vec{r}) = S^g(\vec{r}) + F^g(\vec{r}); \tag{5.2}$$

where

$$F^g(\vec{r}) = \frac{\chi^g}{4\pi\, K_{\text{eff}}} \sum_{g'} \nu\, \Sigma_f^{g'}(\vec{r})\phi^{g'}(\vec{r}); \tag{5.3}$$

$$S^g(\vec{r}) = \frac{1}{4\pi} \sum_{g'} \Sigma_s^{g\leftarrow g'}(\vec{r})\phi^{g'}(\vec{r}); \tag{5.4}$$

and $\phi^g(\vec{r})$ is the scalar flux in group g.

The characteristics method solves the differential form of the transport equation following the straight lines (characteristics or tracking lines). The neutron trajectories will be followed in the local coordinates system where an observer is traveling in the neutron direction. Along these lines, the basic transport operator is transformed into a total differential operator

$$B^g(\vec{r}, \hat{\Omega}) \to C^g(s) = \frac{d}{ds} + \Sigma^g(\vec{r} + s\hat{\Omega}); \tag{5.5}$$

where $C^g(s)$ is the characteristics transport operator in group g and s stands for the variable along the characteristics line. The local multi-group characteristics equation is then given by

$$C^g(s)\Phi^g(\vec{r} + s\hat{\Omega}, \hat{\Omega}) = Q^g(\vec{r} + s\hat{\Omega}). \tag{5.6}$$

The average scalar flux in region j with volume V_j and in energy group g is obtained by integrating over volume and directions

$$V_j \Phi_j^g \; = \; \int_{V_j} d^3r \int_{4\pi} d^2\Omega\; \Phi^g(\vec{r}, \hat{\Omega})$$

$$= \int_{\Upsilon} d^4T \int_{-\infty}^{+\infty} dt\, \chi_{V_j}(\vec{T},t)\, \Phi^g(\vec{p}+t\hat{\Omega},\hat{\Omega}). \tag{5.7}$$

A characteristics line $\vec{T}$ is determined by its orientation $\hat{\Omega}$ along with a reference starting point $\vec{p}$ for the line. Variable t refers to the local coordinates on the tracking line, and the function $\chi_{V_j}(\vec{T},t)$ is defined as 1 if the point $\vec{p}+t\hat{\Omega}$ on the line $\vec{T}(\hat{\Omega},\vec{p})$ is in the region V_j, and 0 otherwise. The d^4T element can now be decomposed into a solid angle element $d^2\Omega$ and a corresponding plane element d^2p.

5.2.1 Discretization of characteristics equations

The spatial and the angular domains are sampled using a ray tracing procedure as described in (Dahmani, 2002). The Υ domain is first covered by choosing a quadrature set of solid angles, composed of discretized directions and their corresponding weights $(\hat{\Omega}_i, \omega_i)$. We generally use the Equal Weight Quadrature (EQ_N) to generate the angular directions (Carlson, 1971). Then, the plane $\Pi_{\hat{\Omega}_i}$ perpendicular to any selected direction i, is split into a Cartesian grid meshing and the starting points $\vec{p}_{i,n}$ of each characteristics are found. For each discretized direction i, a whole set of tracks $\vec{T}_{i,n}$ will be generated. When travelling across different regions, the neutron beam following the characteristics crosses $K_{i,n}$ segments numbered by index k; the segment lengths are labeled L_k and the region numbers are N_k. Assuming that the segment k crosses region j, then the local relationship between the incoming and outgoing angular flux is given by

$$^{\text{out}}\phi_j^g(k) = {}^{\text{in}}\phi_j^g(k) + \left[\frac{Q_j^g}{\Sigma_j^g} - {}^{\text{in}}\phi_j^g(k)\right] E(\tau_{jk}^g); \tag{5.8}$$

where $\tau_{jk}^g = \Sigma_{tj}^{gL_k}$ is the neutron optical path and $E(x) = 1 - e^{-x}$. Assuming that the track lengths have been normalized to ensure volume conservation, the average angular flux per region is given by

$$\Sigma_j^g V_j \bar{\Phi}_j^g(\hat{\Omega}_i) = V_j Q_j^g + \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g; \tag{5.9}$$

where $\Delta_{i,n,k}^g = {}^{\text{in}}\phi_j^g(k) - {}^{\text{out}}\phi_j^g(k)$ is the flux difference on segment k and $\pi_{i,n}$ is the weight associated with track $\vec{T}_{i,n}$. In Eqn (5.8–5.9), the same calculation is done for each energy group.

The average scalar flux in region j is computed by integrating Eq. (5.9) over all directions:

$$\Phi_j^g = \frac{Q_j^g}{\Sigma_j^g} + \frac{1}{\Sigma_j^g V_j} \sum_i \omega_i \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \Delta_{i,n,k}^g. \tag{5.10}$$

From this equation, we see that the iteration sweep over all characteristics involves the sequential evaluation of outgoing fluxes of Eq. (5.8) on every single track and the adding of flux differences $\Delta_{i,n,k}^g$ for each track $\vec{T}_{i,n}$ in Eq. (5.10).

At the boundary of the external surfaces, the outgoing current is calculated using the angular flux computed on the segment K crossing the surface α

$$J_\alpha^{+,g} = \sum_i \omega_i \sum_n \pi_{i,n} \chi_{\alpha,K} \left(\hat{\Omega}_i \cdot \hat{N}_\alpha\right) {}^{\text{out}}\phi_j^g(K), \quad \hat{\Omega}_i \cdot \hat{N}_\alpha > 0; \tag{5.11}$$

and $\chi_{\alpha,K}$ is defined as

$$\chi_{\alpha,K} = \begin{cases} 1, if\ \vec{r} \in S_\alpha \\ \\ 0, otherwise. \end{cases} \tag{5.12}$$

The isotropic reflection condition with an albedo is then applied as follows

$$J_{\alpha}^{-,g} = \beta_{\alpha} J_{\alpha}^{+,g}; \tag{5.13}$$

where β_{α} is the albedo factor on surface S_{α}.

5.2.2 SCR formalism

In the following, the group index will be dropped unless necessary. From the Eq. (5.7), one can calculate the region integrated scalar flux by using the segment integrated angular flux as

$$V_j \Phi_j = \int_{\Upsilon} d^4T \sum_k \delta_{jN_k} L_k \bar{\phi}_j(k); \tag{5.14}$$

where

$$L_k \bar{\phi}_j(k) = \int_0^{L_k} dt \Phi(\vec{p} + t\hat{\Omega}, \hat{\Omega}). \tag{5.15}$$

The integration of the Eq. (5.15) leads to the following equation

$$\bar{\phi}_j(k) = {}^{\mathrm{in}}\phi_j(k) \frac{E(\tau_{jk})}{\tau_{jk}} + \frac{Q_j}{4\pi\Sigma_{tj}}(1 - \frac{E(\tau_{jk})}{\tau_{jk}}); \tag{5.16}$$

By adding all the angular flux contributions, the region scalar flux can be calculated as

$$\begin{aligned}\Phi_j &= \frac{1}{\Sigma_{tj}V_j}\int_\Upsilon d^4T\left\{\sum_k \delta_{j\,N_k}\,{}^{\mathrm{in}}\phi_j(k)E(\tau_{jk})\right\}\\ &+ \frac{Q_j}{4\pi\Sigma_{tj}V_j}\int_\Upsilon d^4T\left\{\sum_k \delta_{j\,N_k}\,L_k(1-\frac{E(\tau_{jk})}{\tau_{jk}})\right\};\end{aligned} \tag{5.17}$$

The outgoing current at the crossing surface S_α is then given by

$$\begin{aligned}J_\alpha^+ &= \int_\Upsilon d^4T\,\chi_{\alpha,K}\,{}^{\mathrm{in}}\phi_{j_\alpha}(K)\,(1-E(\tau_{j_\alpha K}))\\ &+ \frac{Q_j}{4\pi\Sigma_{tj_\alpha}}\int_\Upsilon d^4T\,\chi_{\alpha,K}E(\tau_{j_\alpha K}).\end{aligned} \tag{5.18}$$

The equations above can be rewritten as

$$\Phi_j = {}^{\mathrm{in}}\Phi_j + p_{jj}Q_j \tag{5.19}$$

$$J_\alpha^+ = {}^{\mathrm{in}}J_\alpha^+ + p_{j_\alpha\alpha}V_{j_\alpha}Q_{j_\alpha}; \tag{5.20}$$

where the index j_α indicates the last region encountered before the sub-surface S_α. The $p_{jj} = \frac{1}{4\pi\Sigma_{tj}V_j}\int_\Upsilon d^4T\left\{\sum_k \delta_{j\,N_k}\,L_k(1-\frac{E(\tau_{jk})}{\tau_{jk}})\right\}$ is the reduced collision probability from region j to itself (reduced self-collision probability) and $p_{j_\alpha\alpha} = \frac{1}{4\pi\Sigma_{tj_\alpha}V_{j_\alpha}}\int_\Upsilon d^4T\,\chi_{\alpha,K}E(\tau_{j_\alpha K})$ is the leakage probability from a convex region j_α to the surface α. Both are expressed with the characteristics formalism.

The use of the SCR to accelerate the MCI solver is based then on splitting the total sources in two terms, the self-scattering term and the fission with the scattering of all other groups as: $Q_j^g = \Sigma_{sj}^{g\leftarrow g}\Phi_j^g + F_{sj}^g$. The Eqn.5.19 and 5.20 can be rewritten then as

$$\begin{aligned}(1 - p_{jj}\Sigma_{sj})\Phi_j &= {}^{\text{in}}\Phi_j + p_{jj}\, F_{sj} && (5.21)\\ J_\alpha^+ &= {}^{\text{in}}J_\alpha^+ + p_{j_\alpha\alpha}V_{j_\alpha}\Sigma_{s\,j_\alpha}\Phi_{j_\alpha} + p_{j_\alpha\alpha}V_{j_\alpha}F_{sj_\alpha}. && (5.22)\end{aligned}$$

More details on the derivation of the SCR method and its use to accelerate the characteristics solver are fully presented in (Wu, 2003b).

5.3 Krylov-based iterative methods

The objective of these methods is to solve the linear system of the form

$$Ax = b \qquad (5.23)$$

performing a projection of the solution onto a Krylov subspace and orthogonalizing the residual with respect to another subspace. The different methods of this class arise from the choices of these subspaces and the orthogonalization process. Among these methods, we can distinguish: (Meurant, 1999; Saad, 1996)

1. Methods based on the Arnoldi orthogonalization process which is a modified version of the well-known Graam Schmidt method to construct an orthonormal basis of the subspace $\mathcal{K}_m(\vec{r}_{(o)}, PH)$. Among these algorithms, we can cite GMRES, GCR and ORTHOMIN.

2. Methods based on the Lanzcos bi-orthogonalization process which constructs a bi-orthogonal basis of the subspaces $\mathcal{K}_m(\vec{r}_{(o)}, PH)$ and $\mathcal{K}_m(\vec{r}_{(o)}, (PH)^{\mathrm{T}})$.

This technique is an extension of the Lanzcos orthogonalization process for symmetric system in the general non-symmetric case. It is used for instance in BCG, BiCGSTAB and QMR algorithms.

Considering two subspaces generated by N vectors $\mathcal{K} = \text{span}\{v_1, v_2, ..., v_N\}$ and $\mathcal{L} = \text{span}\{w_1, w_2, ..., w_N\}$, the projection-orthogonalization process can be written as

$$x^{(n+1)} = x^{(n)} + \text{ an element of } \mathcal{K}; \tag{5.24}$$

where $x^{(n+1)}$ is the result of the n^{th} iteration with $x^{(n)}$ as input subject to

$$r^{(n+1)} = \left\| x^{(n+1)} - x^{(n)} \right\| \perp \mathcal{L}. \tag{5.25}$$

5.3.1 GMRES method

As highlighted in many works, each specific problem has its optimal method; one can not expect to find a method that would be the most efficient in every configurations. In our case, the system is non-symmetric and the matrix A is not directly available; the choice is then restricted. A successful approach was to symmetrize the transport operator (Santandrea, 2002) and to use a Krylov technique specific to symmetric operator. However, with such a technique, many interesting preconditioners are useless because of the symmetrization issues. In these conditions, we tried one of the most popular method for non-symmetric systems which has been shown to perform well in several computational fields. The GMRES method is easy to implement and its restarted version, permits to control the storage amount. This method uses $\mathcal{K}(r^{(o)}, A)$ as projection subspace and $A\mathcal{K}(r^{(o)}, A)$ for the orthogonalization. This orthogonalization is equivalent to minimize the 2-norm of the residual $\left\| b - Ax^{(n+1)} \right\|_2$ at each step. The restarted version GMRES(m) used for our implementation limits the dimension of the Krylov subspace $\mathcal{K}(r^{(o)}, A)$ to m. The orthonormal basis is constructed using the modified Graam Schmidt method with Arnoldi step.

The new estimate and its residual are obtained by the QR decomposition of the Hessenberg matrix obtained during the orthonormalization process. This decomposition is updated at each iteration applying Givens rotations to the Hessenberg matrix. A brief summary of the iterative scheme is given in Fig. 5.1. Details on this algorithm can be found in the original paper (Saad, 1986).

5.4 GMRES implementation

5.4.1 Characteristics equations in operator form

5.4.1.1 Inner Iterations

The usual characteristics solution does not involve directly linear algebra operations. However, by using its equivalence with the CP method, we can derive the characteristics equations in operator formalism. The equivalence theorem between the two methods is demonstrated in (Wu, 2003b) where the collision probabilities are written in characteristics formalism. We start from the system

$$\begin{aligned} V_j\Phi_j &= \sum_\alpha J_\alpha^- p_{\alpha j} + \sum_i Q_i V_i p_{ij} \\ J_\gamma^+ &= \sum_\alpha J_\alpha^- p_{\alpha\gamma} + \sum_i Q_i V_i p_{i\gamma}; \end{aligned} \tag{5.26}$$

where the p_{ij}, $p_{\alpha j}$, $p_{\alpha\gamma}$, and $p_{i\gamma}$ are the reduced collision probabilities. By using the sources decomposition described above, the system (5.26) becomes

$$(V_j\Phi_j - \sum_i \Sigma_{si} V_i p_{ij} \Phi_i) - \sum_\alpha J_\alpha^- p_{\alpha j} = \sum_i V_i p_{ij} F_{si}$$

initialization

choose initial guess $x^{(o)}$ and calculate the first vector of the orthonormal basis of $\mathcal{K}(r^{(o)}, A)$ $v_1 = \frac{r^{(o)}}{\left\|r^{(o)}\right\|_2}$

iterations: for $j = 1, \ldots, m$ while $(\left\|r^{(j)}\right\|_2 > \varepsilon \left\|x^{(j)}\right\|_2)$ do:

Computing the Ax

Arnoldi process to update the orthonormal basis

$$\begin{aligned} h_{ij} &= \langle Av_j, v_i\rangle \text{ for } i \in [1,j] \text{ (scalar product)} \\ \hat{v}_{j+1} &= Av_j - \sum_{i=1}^{j} h_{ij} v_i \\ h_{j+1\,i} &= \|\hat{v}_{j+1}\|_2 \\ v_{j+1} &= \frac{\hat{v}_{j+1}}{h_{j+1\,i}} \end{aligned}$$

Updating the QR decomposition of the upper Hessenberg matrix $H^{(j)}$

$$H^{(j)} = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1j} & \left\|r^{(o)}\right\|_2 \\ h_{21} & h_{22} & \cdots & h_{2j} & 0 \\ & \ddots & \ddots & \vdots & \vdots \\ 0 & & h_{j\,j-1} & h_{j\,j} & \vdots \\ 0 & \cdots & \cdots & h_{j+1\,j} & 0 \end{bmatrix} = Q_j R_j$$

where Q_j is an orthonormal matrix and R_j is an upper triangular matrix of dimensions $((j+1) \times (j+1))$

Calculation of $\left\|r^{(j)}\right\|_2$ and $x^{(j)}$ based on the R_j calculation

convergence reached ?

$\frac{\left\|r^{(j)}\right\|_2}{\left\|x^{(j)}\right\|_2}$ less than convergence criterion ?

if not reached, go back to “initialization step” with $r^{(o)} \longleftarrow r^{(j)}$ to compute v_1

Figure 5.1 The GMRES(m) iterative scheme

$$(\beta_\gamma J_\gamma^- - \sum_\alpha p_{\alpha\gamma} J_\alpha^-) - \sum_i V_i p_{i\gamma} \Phi_i \;=\; \sum_i F_{si} V_i p_{i\gamma}. \tag{5.27}$$

This linear system represents the inner iterations for the characteristics method for each energy group. We introduce these notations:

- $\vec{\Phi} = \left[\begin{array}{c} (\Phi_i)_{i\in[1,N_R]} \\ (J_\alpha^-)_{\alpha\in[1,N_S]} \end{array}\right]$: vector of $(N_R + N_S)$ elements, composed of region scalar fluxes and incoming currents. N_R and N_S are respectively the number of regions and the number of surfaces;
- $\vec{F_s} = (F_{si})_{i\in[1,N_R]}$: vector of N_R elements containing the region sources composed of the fission and the out-of-group scattering sources.

The system (5.27) can be written in condensed form as

$$A\vec{\Phi} = B\vec{F_s}; \tag{5.28}$$

where

- $A = \left[\begin{array}{c|c} A_{\Phi\Phi} & A_{\Phi J} \\ \hline A_{J\Phi} & A_{JJ} \end{array}\right]$ is an $(N_R+N_S)\times(N_R+N_S)$ matrix. Its blocks are defined as follows:
  - $A_{\Phi\Phi} = (V_i\delta_{ij} - \Sigma_{sj}V_jp_{ji})_{()0pt2i\in[1,N_R]j\in[1,N_R]}$
  - $A_{\Phi J} = (p_{\gamma i})_{()0pt2i\in[1,N_R]\gamma\in[1,N_S]}$
  - $A_{J\Phi} = (\Sigma_{sj}V_jp_{j\alpha})_{()0pt2\alpha\in[1,N_S]j\in[1,N_R]}$
  - $A_{JJ} = (\beta_\alpha\delta_{\alpha\gamma} - p_{\alpha\gamma})_{()0pt2\alpha\in[1,N_S]\gamma\in[1,N_S]}$
- $B = \left[\begin{array}{c} B_{\Phi F} \\ \hline B_{JF} \end{array}\right]$ is an $(N_R + N_S) \times N_R$ matrix and where
  - $B_{\Phi F} = (V_jp_{ji})_{()0pt2i\in[1,N_R]j\in[1,N_R]}$

- $B_{JF} = (V_j p_{j\alpha})_{()0pt2\alpha\in[1,N_S]j\in[1,N_R]}$.

If we introduce an iteration index, (n), we can consider standard iterations of the method of characteristics

$$\begin{aligned}\vec{\Phi}^{(1)} &= PB\vec{F}_s \\ \vec{\Phi}^{(n+1)} &= \vec{\Phi}^{(1)} + (I_{N_R+N_S} - PA)\vec{\Phi}^{(n)};\end{aligned} \tag{5.29}$$

where

- $I_{N_R+N_S}$ is the $(N_R + N_S) \times (N_R + N_S)$ identity matrix
- P is a $(N_R + N_S) \times (N_R + N_S)$ left preconditioning matrix approaching A^{-1}.

This iterative procedure converges if one can find a norm definition such that $||I - PA|| < 1$. We can express these iterations in practical way in terms of the residual at the n^{th} iteration as

$$P(B\vec{F}_s - A\vec{\Phi}^{(n)}) = \vec{\Phi}^{(n+1)} - \vec{\Phi}^{(n)}. \tag{5.30}$$

This algorithm takes $\vec{F}_s$ and $\vec{\Phi}^{(n)}$ as inputs and delivers $\vec{\Phi}^{(n+1)}$ in output.

Using this linear algebra formulation, the solution of the system at the inner iterations of the characteristics solver can be done by using any linear algebraic methods especially those based on Krylov subspaces. In this context, the `GMRES` method has been chosen as the accelerator of the characteristics solver.

5.4.1.2 Multigroup iterations

For each group $g \in [1, G]$, the inner iterations in multigroup formulation are written as

$$A^g \vec{\Phi}^g = B^g \vec{F}_s^g \tag{5.31}$$

The multigroup iterations can also be viewed as a linear system by extracting the fission source F_i^g from the region source F_{si}^g and constructing a system for the multi-group flux. If we define:

$$\begin{array}{rcl} \vec{F}^g &=& (F_i^g)_{i\in[1,N_R]} \\ \vec{\Phi} &=& \begin{bmatrix} \vec{\Phi}^1 \\ \vdots \\ \vec{\Phi}^G \end{bmatrix} \text{ and } \vec{F} = \begin{bmatrix} \vec{F}^1 \\ \vdots \\ \vec{F}^G \end{bmatrix} \\ B &=& \mathrm{diag}(B^g)_{g\in[1,G]} \\ A &=& \mathrm{diag}(A^g)_{g\in[1,G]} \\ \Sigma_s^{g\leftarrow g'} &=& diag(\Sigma_{si}^{g\leftarrow g'})_{i\in[1,N_R]} \end{array}$$

- $\Sigma_s^* = \begin{bmatrix} O_{N_R\times(N_R+N_S)} & \Sigma_s^{1\leftarrow 2}\, O_{N_R\times N_S} & \cdots & \Sigma_s^{1\leftarrow G}\, O_{N_R\times N_S} \\ \Sigma_s^{2\leftarrow 1}\, O_{N_R\times N_S} & O_{N_R\times(N_R+N_S)} & \cdots & \Sigma_s^{2\leftarrow G}\, O_{N_R\times N_S} \\ \vdots & & \ddots & \vdots \\ \Sigma_s^{G\leftarrow 1}\, O_{N_R\times N_S} & \cdots & \Sigma_s^{G\leftarrow G-1}\, O_{N_R\times N_S} & O_{N_R\times(N_R+N_S)} \end{bmatrix}$;

where $O_{N_R\times N_S}$ is the $N_R \times N_S$ zero matrix, we can then write the system as

$$A\vec{\Phi} = B\left(\sum_s^* \vec{\Phi} + \vec{F}\right); \tag{5.32}$$

and the free iterations take this form

$$B\vec{F} - (A - B\sum_s^*)\vec{\Phi}^{(n)} = \vec{\Phi}^{(n+1)} - \vec{\Phi}^{(n)}. \tag{5.33}$$

5.4.2 SCR as a left preconditioner

5.4.2.1 Inner Iterations

The SCR technique has been investigated as a left preconditioner of the original system. Indeed, the SCR acceleration equations for the inner iterations can be written as

$$\Phi_j^{(n+1)} = \frac{\Phi_j^{(n+\frac{1}{2})} - p_{jj}\Sigma_{sj}\Phi_j^{(n)}}{1 - p_{jj}\Sigma_{sj}} \tag{5.34}$$

$$J_\alpha^{+(n+1)} = J_\alpha^{+(n+\frac{1}{2})} + p_{j_\alpha\alpha}V_{j_\alpha}\Sigma_{s\,j_\alpha}(\Phi_{j_\alpha}^{(n+1)} - \Phi_{j_\alpha}^{(n)}); \tag{5.35}$$

where the index $(n+\frac{1}{2})$ corresponds to the result of a free iteration with fluxes and currents indexed (n) and $(n+1)$ to the final result of this accelerated iteration. This system can be rewritten in operator form as

$$\vec{\Phi}^{(n+1)} - \vec{\Phi}^{(n)} = P(\vec{\Phi}^{(n+\frac{1}{2})} - \vec{\Phi}^{(n)}) \tag{5.36}$$

where

[h!]

Thus, the SCR technique stands for a left preconditioner of the linear system (5.28). The system to solve is now

$$PA\vec{\Phi} = PB\vec{F}_s. \tag{5.37}$$

$$\begin{array}{rcl}
P & = & \left[\begin{array}{c|c} (I_{N_R} - P_R\sum_s)^{-1} & O_{(N_R \times N_S)} \\ \hline P_{SV}E\sum_s(I_{N_R} - P_R\sum_s)^{-1} & I_{N_S} \end{array}\right] \\
P_R & = & diag(p_{jj})_{j\in[1,N_R]} \\
\sum_s & = & diag(\Sigma_{sj})_{j\in[1,N_R]} \\
P_{SV} & = & diag(p_{j_\alpha \alpha}V_{j_\alpha})_{\alpha\in[1,N_S]} \\
E & = & (E_{\alpha j})_{()0pt2\alpha\in[1,N_S]j\in[1,N_R]} \text{ such as } E_{\alpha j} = \begin{cases} 1 & if surface S_\alpha is in region j \\ 0 & otherwise \end{cases}
\end{array}$$

5.4.2.2 Global SCR for Multigroup Iterations

In the same way, we can extend the linear algebra formalism for SCR in a rebalancing method for the multigroup system. If we introduce, [h!] we can write SCR as a left

$$\begin{array}{rcl}
\sum_s & = & \left(\begin{bmatrix} \sum_s^{g\leftarrow h} & O_{N_R\times N_S} \\ O_{N_S\times N_R} & O_{N_S\times N_S} \end{bmatrix}\right)_{()0pt2g\in[1,G]h\in[1,G]} \\
P_{SV} & = & \text{diag}\left(\begin{bmatrix} O_{N_R\times N_R} & O_{N_R\times N_S} \\ O_{N_S\times N_R} & P_{SV}^g \end{bmatrix}\right)_{g\in[1,G]} \\
P_R & = & \text{diag}\left(\begin{bmatrix} P_R^g & O_{N_R\times N_S} \\ O_{N_S\times N_R} & O_{N_S\times N_S} \end{bmatrix}\right)_{g\in[1,G]} \\
\tilde{E} & = & \text{diag}\left(\begin{bmatrix} O_{N_R\times N_R} & O_{N_R\times N_S} \\ E & O_{N_S\times N_S} \end{bmatrix}\right)_{g\in[1,G]} \\
I_\Phi & = & \text{diag}\left(\begin{bmatrix} I_{N_R} & O_{N_R\times N_S} \\ O_{N_S\times N_R} & O_{N_S\times N_S} \end{bmatrix}\right)_{g\in[1,G]} \\
I_J & = & \text{diag}\left(\begin{bmatrix} O_{N_R\times N_R} & O_{N_R\times N_S} \\ O_{N_S\times N_R} & I_{N_S} \end{bmatrix}\right)_{g\in[1,G]} \\
C & = & I_\Phi - P_R\sum_s
\end{array}$$

preconditioner of the multigroup system (5.33)

$$\vec{\Phi}^{(n+1)} - \vec{\Phi}^{(n)} = \left(I_J + (I_{N_R+N_S} - P_{SV}\tilde{E})\sum_s C^{-1}\right)(\vec{\Phi}^{(n+\frac{1}{2})} - \vec{\Phi}^{(n)}) \qquad (5.38)$$

5.4.3 Computational schemes

To implement the `GMRES` iterator in our solver, we use two computational schemes.

- In scheme 1, the `GMRES` is implemented at the inner loop level, here is to say that `GMRES` carries out the convergence for each group concerning the within group scattering, `GMRES` replaces the standard scattering source iterations. For each tracking sweep, all the non-converged groups are calculated at the same time.

- In scheme 2, we have suppressed the inner loop as it is done in the standard MCI solver and `GMRES` procedure is used for the multigroup system. `GMRES` converges on the total scattering source and takes the place of the standard multigroup iteration scheme. This second scheme was tested because in common flux calculation, the inner loop is not needed and increases the computational time. However, when the method of characteristics is used for a self-shielding subgroup model, this inner loop is essential because groups are not correlated. The Fig. 5.2 shows in detail those two computational schemes.

5.4.4 Convergence Criteria

5.4.4.1 Scheme 1

In this scheme, the `GMRES` iterator is implemented at the inner iteration level according to the MCI standard implementation. The number of the multigroup iterations allowed to reach a convergence is defined as

$$N_{th} = \min(it_{out}, N_{thmax}); \tag{5.39}$$

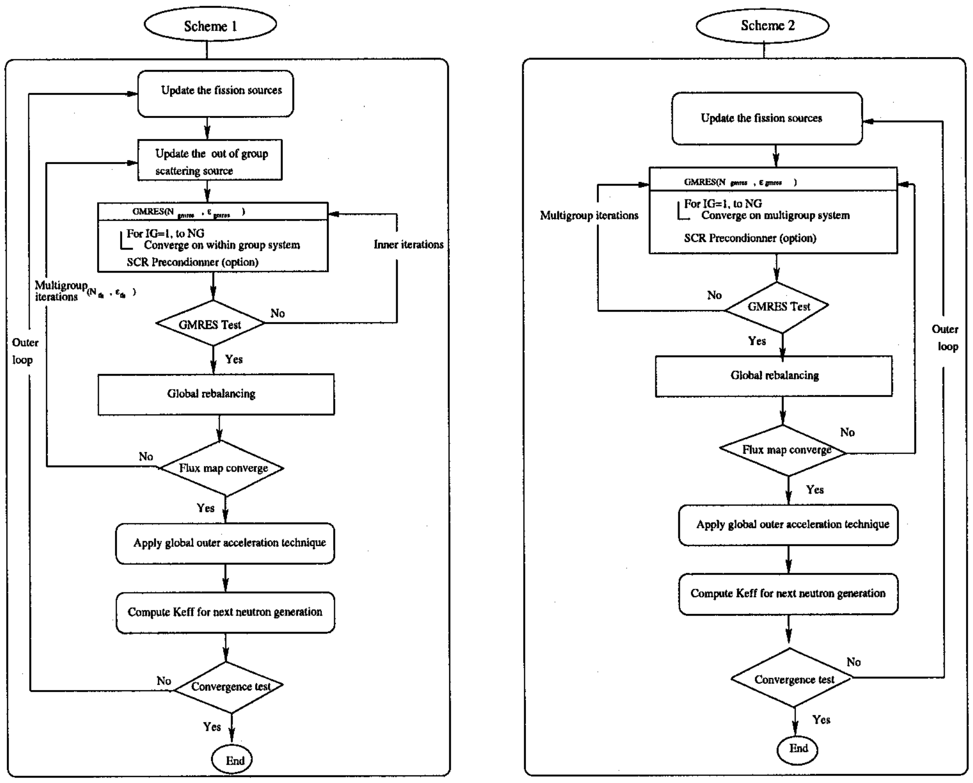

Scheme 1
Update the fission sources
Update the out of group scattering source
GMRES(N_{gmres} , ε_{gmres})
For IG=1, to NG
Converge on within group system
SCR Precondionner (option)
Inner iterations
Multigroup iterations (N_{th} , ε_{th})
GMRES Test
No
Yes
Outer loop
Global rebalancing
No
Flux map converge
Yes
Apply global outer acceleration technique
Compute Keff for next neutron generation
No
Convergence test
Yes
End
Scheme 2
Update the fission sources
GMRES(N_{gmres} , ε_{gmres})
For IG=1, to NG
Converge on multigroup system
SCR Precondionner (option)
Multigroup iterations
No
GMRES Test
Yes
Outer loop
Global rebalancing
No
Flux map converge
Yes
Apply global outer acceleration technique
Compute Keff for next neutron generation
No
Convergence test
Yes
End

Figure 5.2 The computational schemes using the GMRES Iterator

where it_{out} is the number of the outer iteration and N_{thmax} is the maximum number of the multigroup iterations. The precision is a given constant

$$\epsilon_{th} = \epsilon_{th0}. \tag{5.40}$$

For the GMRES loop, the number of iterations N_{gmres} and the precision ϵ_{gmres} are set constant. There is no loop needed for the SCR.

5.4.4.2 Scheme 2

Here, a lot of changes needs to be done for the optimization. For the GMRES loop, we have

$$N_{gmres} = \begin{cases} 1 & if\ it_{out} = 1\ or\ 2 \\ N_{gmresmax} & otherwise \end{cases} \tag{5.41}$$

The precision is given by

$$\epsilon_{gmres} = \max(\gamma, \epsilon_{gmres0}); \tag{5.42}$$

where γ is defined as

$$\gamma = \begin{cases} 0.1E_k & if\ E_k > \epsilon_k \\ 0.01E_f & if\ E_k \leq \epsilon_k \end{cases} \tag{5.43}$$

E_k and E_f are the errors on the K_{eff} and the flux respectively and ϵ_k is the precision on K_{eff}. E_k and ϵ_k are replaced by E_B and ϵ_B if a calculation with buckling search is performed. Here, E_B and ϵ_B are the error on a buckling value and the corresponding error respectively.

Between the outer iterations (power iteration like) and the `GMRES`, the method of contraction on the expected precision of the `GMRES` is used (Bouras, 2000a). Moreover, between `GMRES` and `SCR` iterations, we use the relaxation method (Bouras, 2000b). The number of iterations N_{scr} is set constant and the precision is defined as

$$\epsilon_{scr} = \begin{cases} \epsilon_{scr0} & if\ it_{gmres} = 1 \\ 0.1 \max\left(\epsilon_{scr0}, \min(1, \frac{\epsilon_{scr0}}{\min(E_{gmres}, 1)}\right) & otherwise \end{cases} \quad (5.44)$$

where E_{gmres} is the precision on the `GMRES` iterations.

5.5 Numerical results

5.5.1 CANDU 3D-Supercells

The numerical results are obtained for a CANDU-6 3D supercell problems. The supercells are composed of two horizontal bundles containing the Uranium oxide fuel, and one vertical absorber rod (Fig. 5.3). The transport equation is then solved by critical buckling search with first order leakage treatment $B1$. Homogenization and condensation processes are then made with the resulting fluxes. The properties are condensed to 2 energy groups keeping the small up-scattering effect from the thermal to fast group. For all tests, we use an EQ_4 quadrature and 89 energy groups. The spatial meshing and the density of tracks may be changed as needed.

For each case, two types of calculation were performed, using the `GMRES` implementations corresponding to scheme 1 and 2. the results in Table 5.1 show the number of iterations for different techniques when using the scheme 1 for the `GMRES`. Using only the `SCR` acceleration gives the best performance. The `GMRES` takes much more

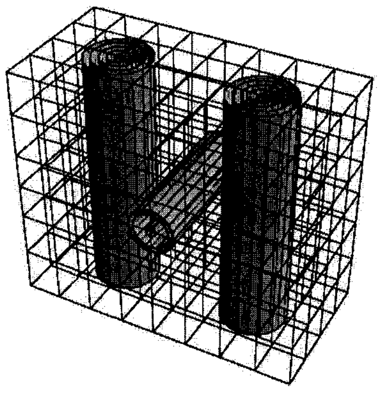


Figure 5.3 CANDU-6 supercell

iterations to converge on flux calculation; even more than the free iterations when using the scheme 1 because of the indroduction of the inner iterations. In this case `GMRES` is badly inefficient. When the scheme 2 (where the `GMRES` takes care of the multigroup calculation) is used, the number of iterations is considerably reduced but the technique still remains inefficient (Table 5.2). These difficulties come from a limitation imposed by the use of `GMRES`. When a group has converged, one can not omit this group in the next iteration because the linear system has to be kept untouched; this fact compromises the efficiency of `GMRES` in these tests.

5.5.2 PWR Assemblies

To evaluate the performance of these techniques, numerical tests based on 9×9 assemblies were performed. The assemblies are composed of three different types of

Tableau 5.1 Number of iterations for CANDU reactor's adjuster calculation (Scheme 1)

Adjuster type	GMRES	GMRES+ SCR	SCR	Free iterations
BCAINT	175	118	24	35
BCAOUT	168	115	23	39
BCBINT	168	120	25	37
BCCINT	168	120	23	35
BCCOUT	169	114	23	39
BCDINT	166	118	24	35

Tableau 5.2 Number of iterations for CANDU reactor's adjuster calculation (Scheme 2)

Adjuster type	GMRES	GMRES+ SCR	SCR	Free iterations
BCAINT	37	45	24	35
BCAOUT	37	45	23	39
BCBINT	38	46	25	37
BCCINT	36	45	23	35
BCCOUT	37	46	23	39
BCDINT	37	45	24	35

cells (Fig. 5.4). In order to generate a corresponding 3D tests, the third dimension of the benchmark is extended from 0 to 1 *cm* with isotropic reflection at the boundary. To reproduce the difficulties highlighted in previous works with diffusion synthetic acceleration techniques (Warsa, 2003; Warsa, 2003; Le Tellier, 2004), large material discontinuities and scattering ratio ($c = \Sigma_s/\Sigma_t$) close to 1, using monoenergetic cross sections, have been chosen. These calculations have not directly a physical meaning but were performed to highlight the capability of such a method in very heterogeneous situations. Fixed source calculations have been then performed.

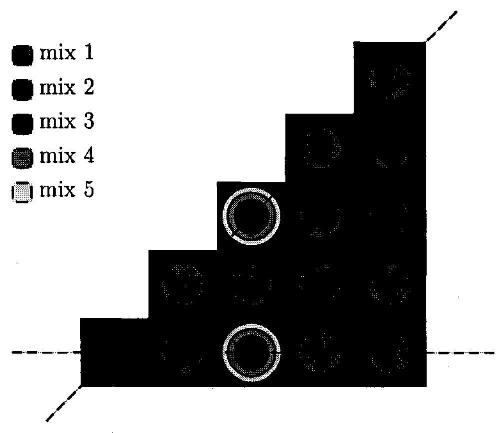


Figure 5.4 The geometry of the assemblies

5.5.2.1 Results using a fixed scattering ratio

We begin these tests with a fixed scattering ratio (c = 0.9) using different cross section configurations as reported in Table 5.3. In these tests, we use 20 angles and 40 tracks/cm^2 for density of tracks.The maximum dimension of the Krylov subspace is set to 20.

Tableau 5.3 Cross sections of the 9 × 9 assemblies for $c = 0.9$

	Case1	Case2	Case3	Case4
$\Sigma_{t1}(cm^{-1})$	1.0	10.0	0.1	10.0
$\Sigma_{t2}(cm^{-1})$	0.1	0.01	0.01	100.0
$\Sigma_{t3}(cm^{-1})$	10.0	100.0	0.001	1000.0
$\Sigma_{t4}(cm^{-1})$	10.0	100.0	0.001	1000.0
$\Sigma_{t5}(cm^{-1})$	10.0	100.0	0.001	1000.0

The results in Table 5.4 show that even with a fixed scattering ratio, the number of iterations for the all techniques varies considerably with respect to the degree and the order of the material heterogeneity in cells. In all cases, the `GMRES` technique reduce considerably the number of iterations comparing to the `SCR` preconditioned and free iterations techniques. In these cases, the `SCR` technique does not perfom well. Almost the same number of iterations than with the free iterations. The GMRES preconditioned with SCR produces good results, the smallest number of iterations. The CPU time spent with all of these techniques follow the number of iterations. The minimun of the CPU time corresponds to the GMRES preconditioned by SCR (Table 5.5).

Tableau 5.4 Number of iterations using different acceleration techniques

Method	GMRES	GMRES+ SCR	SCR	Free iterations
Case1	17	15	63	87
Case2	16	13	27	27
Case3	11	12	386	389
Case4	16	9	47	47

Tableau 5.5 CPU time with different acceleration techniques

Method	GMRES	GMRES+ SCR	SCR	Free iterations
Case1	102	87	356	501
Case2	100	79	158	163
Case3	72	76	2226	2252
Case4	105	60	287	285

We also present, in Fig. 5.5, the convergence rate of the MCI solver when using different acceleration techniques.

5.5.2.2 Increasing the scattering ratio

Now we vary the value of the scattering ratio c. We use three tests with respectivly $c = 0.9$, $c = 0.999$ and $c = 0.99999$. The number of iterations with the GMRES technique grows as $c \to 1$. By using the SCR as preconditioner, the number of these iterations is reduced. SCR does not reach convergence for the two last tests when c is close to 1. The same behavior has been observed with the DSA method (Le Tellier, 2004). The results of these tests are reported in Table 5.6.

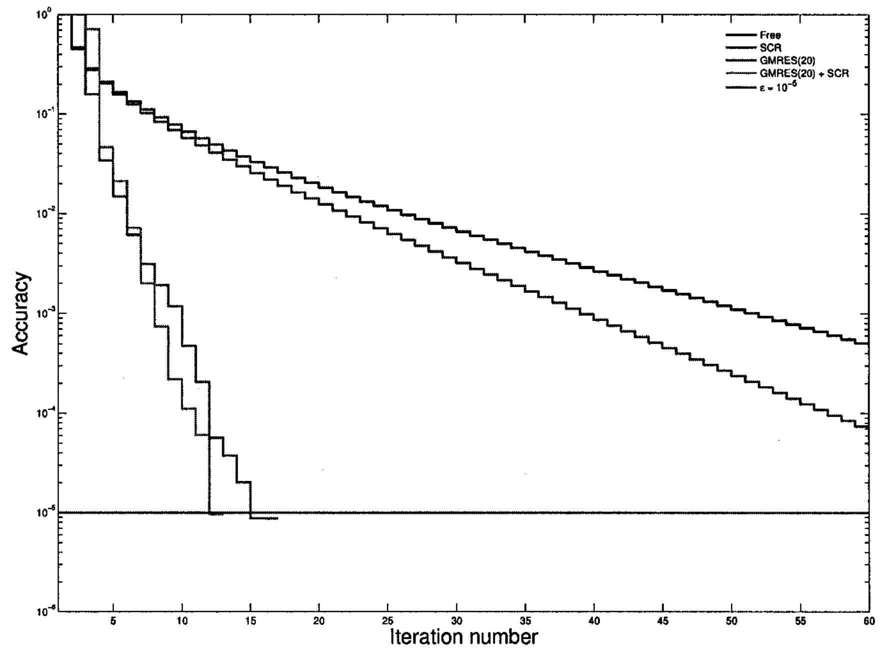


Figure 5.5 Convergence of the MCI solver with different acceleration techniques

In order to test the behavior of these techniques according to the domain refinement, we first vary the number of angles for a fixed ratio $c = 0.9$ and density of tracks. For all these tests, the number of iterations does not vary. The CPU time varies slightly (Table 5.7). The same statement can be done when we vary the density of tracks for a fixed number of angles. However, the CPU time varies linearly. The results are shown in Table 5.8.

Tableau 5.6 Number of iterations and (CPU time in second) for different scattering ratio

Method	GMRES	GMRES+ SCR	SCR
$c = 0.9$	17 (102)	15 (87)	63 (356)
$c = 0.999$	28 (173)	19 (118)	-
$c = 0.99999$	55 (332)	19 (116)	-

Tableau 5.7 Number of iterations and (CPU time in second) by varying the number of angles

Method	GMRES	GMRES+ SCR	SCR
8	17 (30)	15 (26)	66 (104)
20	17 (125)	15 (118)	63 (363)
32	17 (131)	15 (124)	66 (391)
48	17 (143)	15 (130)	66 (406)

At the end, we vary the maximum dimension of Krylov subspace (m). If the dimension is not enough to construct the corresponding basis, the GMRES method needs a lot of iterations to reach the convergence. For $m \geq 15$, the number of iterations becomes stable and equal to 17. Thus, there is no need to further increase the value of the dimension (Table 5.9).

Tableau 5.8 Number of iterations and (CPU time in second) for different density of tracks (tracks/cm^2)

Method	GMRES	GMRES+ SCR	SCR
10	17 (26)	15 (24)	63 (93)
20	17 (55)	15 (47)	63 (195)
40	17 (104)	15 (94)	63 (363)
60	17 (155)	15 (140)	66 (554)

Tableau 5.9 Number of iterations and (CPU time in second) by varying the Krylov subspace dimension

Method	GMRES
10	141 (836)
12	19 (155)
15	17 (110)
20	17 (102)

5.6 Summary and Conclusions

The implementation of the GMRES Krylov subspace iterative method in our sequential 3D characteristics solver introduces a new approach to the solution of the characteristics system by using the linear algebraic formalism. Recently, several works have been presented on the application of the Krylov subspace methods to the numerical solution of the neutron transport equation. The results show that the efficiency of the methods, in terms of acceleration, varies with the nature of problems. For each specific problem, one can find an optimal technique to accelerate effciently the solution.
In our numerical experiments, tests on realistic CANDU 3D supercells and some 3D extensions of PWR assemblies were performed. In the first category of tests, the GMRES method is inefficient. It needs a lot of iterations to converge on flux. The SCR, as confirmed in previous work, gives good performance for these kind of problems. In the second category of problems, the GMRES method and GMRES preconditioned with SCR produce best results. SCR alone is less efficient.

In conclusion, for problems presenting high material discontinuities, a GMRES method can significantly improve the efficiency of the characteristics calculations for which the SCR technique will be ineffective. In these cases, according to our results, the SCR can be used as an efficient preconditioner for GMRES method. The number of iterations is considerably reduced when the GMRES system is preconditioned with SCR. However, for the CANDU type calculations, the SCR technique still remains the most efficient. The number of iterations is substantially reduced and the technique is not memory intensive, requiring only new arrays of the size of multigroup flux. For these relatively small cases, the memory requirement and the storage capacity needed by the GMRES calculations were acceptable. However, it is expected that for large problems, large capacity of storage will be needed in order to store the vectors of the orthonormal basis and more computational effort to do the dot product. In

CANDU type of calculations, the GMRES method is inefficient and its use is not recommended.

In this context, we plane to extend this study for large 3D calculations, including problems with upscattering and anisotropic scattering. In order to reduce the computational time and to distribute the resources in GMRES calculations for large problems, the parallelization and the implementation of the parallel GMRES method in our 3D parallel characteristics solver will be also considered in future work.

CHAPITRE 6

DISCUSSION GÉNÉRALE ET CONCLUSIONS

Dans ce travail, une nouvelle approche aux calculs de transport de neutrons impliqués dans la chaîne de calculs de réacteurs a été introduite. Cette approche est basée sur l'utilisation d'un modèle informatique que nous avons conçu. Des problèmes de taille de plus en plus grande ont été résolus.
Afin de prévoir les performances de notre code sur des machines massivement parallèles, une grande partie des travaux de cette thèse a été consacrée à l'étude de l'évolutivité. Nous avons ainsi trouvé une relation analytique entre la taille du problème, le nombre de processeurs et l'efficacité du système. Cette relation permet de trouver le meilleur compromis entre la taille du problème et le nombre de processeurs afin de maintenir l'efficacité constante.
Basé sur une nouvelle méthodologie de calcul, le solveur *MCG* a aussi été développé pour remédier aux problèmes liés au stockage des lignes d'intégration dans le contexte de problèmes de grande taille.
Dans la section suivante, nous allons faire une synthèse de ces travaux.

6.1 Discussion générale

Dans les articles constituant les chapitres 2, 3, 4 et 5, dans un premier temps, le modèle analytique a été dérivé dans le cas général, pour n'importe quelle architecture. Puis, les résultats pour deux machines parallèles ont été analysés : un cluster de *SMP* (avec 32 processeurs) et un cluster de *NUMA* (avec 64 processeurs). Le modèle a ensuite été généralisé pour tenir compte des architectures hybrides. Nous avons utilisé le modèle pour expliquer l'impact des paramètres réseau sur le temps de communication (et donc sur la performance), l'impact des paramètres propres de chaque machine (puissance de calcul) sur le rapport calcul/communication et l'influence de l'augmentation du nombre de processeurs sur la performance. Les premiers tests portent sur l'utilisation d'une variété de réseaux d'interconnexion comme Fast Ethernet, Myrinet, Infiniband... Les temps de communication les plus faibles ont été obtenus avec des réseaux ayant les temps de transfert par mot les plus petits, la latence n'ayant presque pas d'effet sur le temps de communication. Ces comportements ont été expliqués par le modèle en comparant des résultats analytiques à ceux obtenus par des expériences. Des tests similaires ont été faits avec des machines différentes utilisant des réseaux semblables pour tirer des conclusions concernant l'influence de la puissance de calcul ou exactement le rapport calcul/-communication sur la performance.

Le modèle est aussi utilisé pour l'étude de l'évolutivité de l'algorithme parallèle dans le but de prévoir les performances de l'application sur un très grand nombre de processeurs. Les résultats sur la validation de notre modèle montrent une bonne concordance entre les résultats prévus par le modèle et ceux donnés par les expérimentations. L'analyse d'évolutivité nous a conduit à la conclusion suivante : notre application est évolutive ('scalable') si nous arrivons à avoir une meilleure combinaison entre la taille du problème et la taille de la machine parallèle (nombre de processeurs).

Une méthode d'accélération *GMRES* utilisant une projection de sous-espaces de Krylov a également été adaptée et implementée. La méthode *SCR* peut aussi être utilisée comme précoditionnement pour *GMRES*. De bonnes accélérations ont été observées surtout pour des problèmes comportant de fortes hétérogéneités spatiales.

Dans les publications en annexe, nous avons présenté un aperçu général des méthodes d'accélération numériques utilisées dans le solveur des caractéristiques. La parallélisation du solveur est introduite, en passant en revue les différentes techniques d'équilibrage de charges statiques utilisées.
Des résultats comparatifs entre différentes techniques d'équilibrage de charges ont montré que $STRD$ est plus balancée que les deux autres : $SPLT$ et $ANGL$.
Quant à la gestion des données liées aux lignes d'intégration, nous avons proposé deux solutions : la première est la parallélisation des E/S utilisant MPI-IO, dans ce cas les lignes d'intégration sont stockées dans des fichiers séquentiels. La deuxième solution consiste à stocker les données dans une base de données centralisée et à utiliser des accès parallèles et concurents. Avec une variété de tests, nous avons montré les limitations de chacune de ces approches.
Nous exposons aussi d'autres développements. Dans le contexte de problèmes de grande taille, l'utilisation de fichiers pour stocker les lignes d'intégration reste toujours le problème majeur à cause du dépassement de la capacité des machines. Pour éliminer ces problèmes, nous avons proposé une solution qui consiste à éliminer l'utilisation de ces fichiers. Il s'agit d'un nouveau solveur des caractéristiques MCG qui est basé sur une autre approche, les lignes d'intégration sont alors générées au fur et à mesure que le calcul se déroule. Ce solveur nécessite beaucoup de ressources de calcul mais aucun stockage de lignes. Nous avons fait aussi une comparaison entre les différents solveurs de transport dans le cadre des problèmes de grande taille. Les tests de validation du solveur ont montré de très bonnes concordances avec les résultats de référence issus d'autres solveurs.
Finalement, nous avons considéré un traitement linéaire de la dépendance angulaire des sections efficaces de diffusion et de l'opérateur de diffusion pour tenir compte de l'anisotropie. Des résultats comparatifs entre la solution anisotrope et isotrope montrent que dans certains cas des écarts importants peuvent être observés à la frontière du domaine.

6.2 Recommendations pour les travaux futurs

Bien que nous ayons montré que le code est évolutif, le suivi de ces études est nécessaire pour valider expérimentalement le modèle analytique sur des machines massivement parallèles comportant un très grand nombre de processeurs.
L'un des inconvénients des solveurs caractéristiques est la convergence lente qui est due à un démarrage du processus itératif en utilisant un flux constant partout. Un nombre important d'itérations est donc nécessaire avant que la convergence ne soit atteinte. Une des façons de contourner ce problème est d'utiliser un flux non uniforme qui peut être obtenu en utilisant une approximation de l'équation de transport (ou l'équation de diffusion) sur un maillage grossier. Ce processus peut servir alors pour accélérer le solveur et diminuer le nombre d'itérations.
La parallélisation du solveur *MCG* sera aussi importante pour pouvoir accélérer la solution en tirant profit du calcul parallèle. Un modèle de programmation hybride serait ainsi le plus approprié.
Au niveau de la validation et des tests, une collaboration avec d'autres équipes ou le développement de nouveaux benchmarks seront nécessaires.

Dans un futur proche, le développement de nouveaux solveurs de transport basés sur des algorithmes évolués utilisant des méthodes du calcul de haute performance et la révolution actuelle au niveau de l'informatique parallèle vont nous permettre de réaliser des calculs de transport à la place des calculs approximatifs de diffusion pour résoudre les problèmes de coeur de réacteur. On pourrait alors se permettre l'application de modèles théoriques toujours plus fins et plus précis.

RÉFÉRENCES

ADAMS, M. L., and LARSEN, E. W. 2002. *Fast Iterative Methods for Discrete-Ordinates Particle Transport Calculations.* Progress in Nuclear Energy, Vol. 40. No. 1 pp3-159.

AL ABURTO : http://gwyn.tux.org/ mayer/linux/bmark.html

ALMASI, G. and GOTTLIEB, A. 1994. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Compagny Inc, 2nd*edition.*

ASKEW, J. R. 1972. *A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries*, Report AEEW-M 1108, United Kingdom Atomic Energy Establishment,Winfrith.

AZMY, Y. Y. 1993. *An Efficient Communication Scheme for Solving the S_N Equations on Message-Passing Multiprocessors. Trans. Am. Nucl. Soc.*, **69**, 203.

BAKER, M., FARRELL, H., ONG, H. and SCOTT, S. L. 2001. *VIA Communication Performance on a Gigabit Ethernet Cluster*, Proceedings of the 7th International Euro-Par Conference on Parallel Processing, Manchester, United Kingdom, August 2001.

BOLTZMANN, L. 1912. *Vorlesungen über Gastheorie. 2nd edition.*

BOURAS, A., FRAYSSÉ, V., 2000a. *A relaxation strategy for inexact matrix-vector products for Krylov methods.* CERFACS TR/PA/00/15, European Centre for Research and Advanced Training in Scientific Computing, Toulouse, France.

BOURAS, A., FRAYSSÉ, V., 2000b. *A relaxation strategy for Arnoldi method in eigenproblems.* CERFACS TR/PA/00/16, European Centre for Research and Advanced Training in Scientific Computing, Toulouse, France.

BRAUNL, T., 1993. *Parallel Programming: An Introduction.* Prentice Hall.

BUTLER, R. M. and LUSK, E. L. 1994. *monitors, Messages, and Clusters: The p4 Parallel Programming System.* Parallel Computing, vol. 20,$N^0 4, pp.547 - 564$.

CALKIN, R., HEMPEL, R., HOPPE, H. C. and WYPIOR, P. 1994. *Portable programming with the PARMACS message-passing library.* Parallel Computing, vol. 20,$N^0 4, pp.615 - 632$.

CARLSON, B., 1971. *Table of Equal Weight Quadrature EQ_N Over the Unit Sphere.* Technical Report LA-4734, Los Alamos National Laborstory.

CARLSON, B. G. and LATHROP, K. D. 1968. *Transport Theory-Discrete Ordinates-, in Computing Methods in Reactor Physics.* Greenspan, H., Kelber, C. N. and Okrent, D. Gordon and Breach Science New York.

CHO, J. Y., JOO, H. G., PARK, S. Y., and ZEE, S. Q., 2002. *Consistent Group Collapsing Scheme for Multi-Group MOC Calculation.* Proc. PHYSOR-2002.

COSNARD, M. and TRYSTRAN, D. 1993. *Algorithmes et Architectures Parallèles.* InterEdition.

Cyberlogic, *Adelie Linux for Single System Image Cluster FAQ.* [Online 2004]. Available: http://www.adelielinux.com/en/ssi/faq.html.

DAHMANI., M., MORIN, B., and ROY, R. 2004. *Performance Evaluation for Neutron Transport Application Using Message Passing.* Proc. of the 18th Annual International Symposium on High Performance Computing Systems and Applications, Winnepeg, Manitoba, Canada, May 16-19.

DAHMANI, M., ROY, R. and KOCLAS, J. 2003b. *Data Management in Parallel Transport Calculation.* International Conference on Supercomputing in Nuclear Applications SNA'2003, Paris, France, September 22-24.

DAHMANI, M., WU, G. J., ROY, R. and KOCLAS, J. 2002. *Development and Parallelization of the Three-Dimentional Characteristics Solver MCI of DRAGON*, Proc. of PHYSOR2002, Seoul, Korea, October 7-10, 2002.

DAHMANI, M., ROY, R. and KOCLAS, J. 2003a. *Parallel Distribution of Tracking for 3D Neutron Transport Calculation*, ANS Conf. on Nuclear Mathematical and Computational Sciences, on CD-ROM, Gatlinburg, Tennessee, April 6-11, 2003.

DONGARRA, J., VANDEGEIJN, R. A., WALKER, D., W. 1994. *Scalability Issues Affecting the Design of a Dense Linear Algebra Library.* Journal of Parallel and Distributed Computing, 22, 523-537.

DUDERTADT, J. J., LEWIS, E. E. and BARDOS, C. 1983. *Neutron Transport Equation.* Editions Eyrolles, Paris.

DUTT, S., MAHAPATRA, N. R., 1994. *Scalable Load Balancing Strategies for Parallel A* Algorithms.* Journal of Parallel and Distributed Computing, 22, 488-505.

FISHER, J. W. and AZMY, Y. Y. 2003. *Parallel Performance of the Angular Versus Spatial Domain Decomposition for Discrete Ordinates Transport Methods.* Proc. Super Computing in Nuclear Applications SNA03, Paris, September 23-26.

FOSTER, I. 1994. *Designing and Building Parallel Programs.* Addison-Wesley Publishing Compagny.

FOX, G. C., WILLIAMS, R. D. and MESSINA, P. C. 1994. *Parallel Computing Works.* Morgan Kaufmann Publishers, Inc.

FLOWER, J. and KOLAWA, A. 1994. *Express is not a message passing system: Current and future directions in Express.* Parallel Computing, vol. 20,$N^0 4, pp.597-614$.

Gentoo Technologies, Inc., *Gentoo Linux – About Gentoo Linux.* [Online 2004]. Available: http://www.gentoo.org/main/en/about.xml.

GEIST, G. A. and SUNDERAM, V. S. 1994. *PVM: Parallel Virtual Machine- A users Guide and Tutorial for Networking Parallel Computing.* The MIT Press.

GENGLER, M. UBÉDA, S. and DESPREZ, F. 1996. *initiation au parallélisme-Concepts, architectures et algorithmes.* Edition Masson.

GLASSTONE, S. and SESONSKE, A. 1991. *Nuclear Reactor Engineering.* 3rd ed., Malabar, Floride, Krieger Publishing Company.

GRAMA, A., GUPTA, A., KARYPSIS, G. and KUMAR, V. 2003. *Introduction to Parallel Computing*, Second Edition, Addison-Wesley.

GRAMA, A., GUPTA, A., KARYPIS, G. and KUMAR, V. 1993. *Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures.* IEEE Parallel Distributed Technology, Volume **1**, 12-21.

GROOP, W., and LUSK, E. 1999. *Reproducible measurements of MPI performance characteristics*, Technical Report ANL/MCS-P755-0699, Argonne National Labratory, Argonne, IL, USA, June 1999.

GROOP, W., LUSK, E., SKJELLUM, A. 1994. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, Cambridge.

GROOP, W. and LUSK, E. 1998. *installation Guide to MPICH, a Portable Implementation of MPI* Argonne National Laboratory, http://www.mcs.anl.gov/mpi/mpiinstall/paper.html.

GUPTA, A. and KUMAR, V. 1993. *Performance Properties of Large Scale Parallel Systems.* Journal of Parallel and Distributed Computing, **19**, 234-244.

GUSTAFSON, J. L. 1988. *Reevaluating Amdahl's Law.* Communications of ACM, vol. 31, pp. 532-533.

HALSALL, M.J., 1998. *WIMS8 Speed with accuracy.* Int. Conf. Physics of Nuclear Science and Technology, Long island, New York.

HANSHAW, H.L., LARSEN, E.W., 2003. *The explicit slope S_N discretization method.* Int. Conf. on Nuclear Mathematical and Computational Sciences, on CD-ROM, Gatlinburg, Tennessee.

HOARE, C. A. R. 1978. *Communicating Sequential Processes.* Communications of ACM, vol.2, N^{0}8.

HONG, S.G., CHO, N.Z., 1999. *Angular depend rebalancing (ADR) iteration for discrete-ordinate transport problems in equilateral triangle meshes.* Int. Conf. mathematics and Computation, Reactor Physics and Environmental Analysisi in Nuclear Applications, Madrid.

JOO, H. G., CHO, J. Y., KIM, H. Y., ZEE, Q. and CHANG, M. H., 2002. *Dynamic Implementation of the Equivalence Theory in the Heterogeneous Whole Core Transport Calculation.* Proc. PHYSOR-2002.

KOSAKA, S., SAJI, E. 1999. *The Characteristics Transport Calculation for a Multi-Assembly System Using Path Linking Technique.* Proc. ANS Conference on Mathematics and Computation.

KRISHNAMOORTHY, S., CHOUDHARY, A. N., 1994. *A Scalable Distributed Shared Memory Architecture.* Journal of Parallel and Distributed Computing, 22, 547-554.

KUMAR, V. and GUPTA, A. 1994. *Analyzing Scalability of Parallel Algorithms and Architectures.* Journal of Parallel and Distributed Computing, 22, 379-391.

KUMAR, V., GRAMA, GUPTA, A., KARYPIS, G. 1994. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Compagny Inc.

LEE, G. S., CHO, N. Z. and HONG, S. G. 2000. *Acceleration and Parallelization of the Method of Characteristics for Lattice and Whole-Core Heterogeneous Calculation.* Proc. of PHYSOR-2000 Conference.

LEOPOLD, C. 2001. *Parallel and Distributed Computing: a Survey of Models, Paradigms, and Approaches.* Wiley, New York.

LE TELLIER, R., HÉBERT, A., 2004. *Application of the DSA preconditioned GMRES formalism to the method of characteristics-First results.* PHYSOR-2004 Conference, Chicago, Illinois.

LEWIS, E. E. and MILLER, W. F. 1984. *Computational Methods of Neutron Transport.* WileySons, New York.

LIU. J., et al. 2003. *Performance Comparison of MPI Implementations over Infiniband, Myrinet and Quadrics*, Proceedings of the ACM/IEEE SC2003 Conference, Phoenix, Arizona, November 2003.

LOUBIÈRE, S., SANCHEZ, R., COSTE, M., HÉBERT, A., STANKOVSKI, Z., VAN DER GUCHT, C. and ZMIJAREVIC, I. *APOLLO2,Twelve Years Later*, paper presented at the *Int. Conf. on Mathematics and Computation, Reactor Physics and Environmental Analysis in Nuclear Applications*, Madrid, Spain.

MARINESCU, D. C. and RICE, J. R. 1994. *On the Scalability of Asynchronous Parallel Computations.* Journal of Parallel and Distributed Computing, 22, 538-546.

MARLEAU, G., HÉBERT, A., and ROY, R., 1997. *A User's Guide for DRAGON.* Report IGE-174 Rev. 3, École Polytechnique de Montréal.

MCBRAYAN, O. A., 1994. *An Overview of Message Passing Environnements.* Parallel Computing, vol. 20,N^{0}4, $pp.417 - 444$.

MEURANT, G., 1999. *Computer solution of large linear systems.* Editor: J.L. Lions, Paris.

PATTON, B.W. and HOLLOWAY, J.P., 2002. *Application of preconditioned GMRES to the numerical solution of the neutron transport equation. Ann. Nucl. Energy* **29**, 109-136.

PANWAR, R. and AGHA, G. 1994. *A Methodology for Programming Scalable Architectures.* Journal of Parallel and Distributed Computing, 22, 479-487.

QADDOURI, A., ROY, R. and GOULARD, B. 1995. *Multigroup Flux Solvers Using PVM.* ANS Int. Conf. on Mathematics and Computation.

QADDOURI, A., ROY, R., MAYRAND, M. and GOULARD, B. 1996. *Collision Probability Calculation and Multigroup Flux Solvers Using PVM. Nucl. Sc. Eng.*, **123**, 392.

ROY, R., KOCLAS, J., SHEN, W., JENKINS, D.A., ALTIPARMAKOV, and ROUBEN, B. 2004. *Reactor Core Simulations in Canada*, Proc. of PHYSOR-2004, Chicago, April 25-27, 2004.

ROY, R. 2003 *Théorie des probabilités de collision et méthode des caractéristiques*, IGE-235, Institut de génie nucléaire, École Polytechnique de Montréal.

ROY, R., MARLEAU, G., TAJMOUATI, J., ROZON, D., 1994. *Modelling of CANDU reactivity control devices with the lattice code DRAGON. Ann. Nucl. Energy* **21**, 115-132.

SAAD, Y. and SHULTZ, M.H., 1986. *GMRES: A Generalized Minimal RESidual Algorithm for solving nonsymmetric linear systems.* SIAM J. Sci. Stat. Comput., **7** 856-869.

SAAD, Y., 1996. *Iterative methods for sparse linear systems.* PWS Publishing Company, Boston.

SAAVEDRA, R. H., MAO, W. and HWANG, K. 1994. *Performance and Optimization of Data Prefetching Strategies in Scalable Multiprocessors.* Journal of Parallel and Distributed Computing, 22, 427-448.

SANCHEZ, R. and CHETAINE, A., 1999. *Synthetic acceleration for 2D characteristics method in non regular meshes.* Int. Conf. Mathematics and Computation, Reactor Physics and Environmental Analysisi in Nuclear Applications, Madrid.

SANCHEZ, R. and McCORMICK, N. J. 1982. *A review of Neutron Transport Approximations. Nucl. Sc. Eng.*, **80**, 481.

SANTANDREA, S., SANCHEZ, R., 2002. *Acceleration techniques for the characteristics method in unstructured meshes. Ann. Nucl. Energy* **29**, 323-352.

SEIFERT, F., BALKANSKI, D. AND REHM, W. 2000. *Comparing MPI Performance of SCI and VIA*, Conference Proceeding of SCI-EUROPE 200, Munich, Germany, August 2000.

SIVASUBRAMANIAM, A., SINGLA, A., RAMACHANDRAN, U. and VENKATESWARAN, H. 1994. *A Simulation-Based Scalability Study of Parallel Systems.* Journal of Parallel and Distributed Computing, 22, 411-426.

SJODEN, G. E. AND HAGHIGHAT, A. 1997. *PENTRAN-A 3-D Cartisian Parallel S_N Code with Angular, Energy, and Spatial Decomposition.* Proc. Joint Int. Conf. Mathematical Methods and Supercomputing for Nuclear Applications, Saratoga Springs, New York, October 5-6.

SMITH, K. S. and RHODES, J. D. 2002. *Full-Core, 2-D, LWR Core Calculation with CASMO-4E.* Proc. of PHYSOR-2002, Seoul, Korea, October 7-10.

STALER, S. and VUJIC, J., 1994. *P4 Parallelization of General Geometry Ray racing in Computational Physics.* Proc. High Performance Computing, San Fransisco, Ca.

SUN, X. H. and ROVER, D. T. 1994. *Scalability of Parallel Algorithm-Machine Combinations. IEEE Transaction on Parallel and Distributed Systems*, Vol. 5, No. 6.

SUNDERAM, V. S., GEIST, G. A., DONGARRA, J. and MANCHEK, R. 1994. *The PVM Concurrent Computing System: Evolution, Experiences, and Trends.* Parallel Computing, vol. 20,N^{0}4,$pp.531-546$.

SUSLOV, I.R., 1994. *Solution of transport equation in 2- and 3-dimensional irregular geometry by the method of characteristics.* Int. Conf. mathematics and Computation, Reactor Physics and Environmental Analysisi in Nuclear Applications, Madrid.

TUCKER, L. W. and MAINWARING, A. 1994. *CMMD: Active Message on CM-5.* Parallel Computing, vol. 20,$N^0 4, pp.481 - 496$.

VETTER, J. S. and MUELLER, F. 2003. *Communication characteristics of large-scale scientific applications for contemporary cluster architectures.* Journal of Parallel and Distributed Computing, 63, 853-865.

WARSA, J.S., WAREING, T.A. and MOREL, J.E. 2003. *Krylov iterative methods applied to multidimensional S_N Calculations in the presence of material discontinuities.* Int. Conf. Nuclear Mathematical and computational Sciences, Gatlinburg, Tennessee.

WARSA, J.S., WAREING, T.A., MOREL, J.E., 2003. *Krylov iterative methods applied to multidimensional S_N Calculations in the presence of material discontinuities.* Int. Conf. Nuclear Mathematical and computational Sciences, Gatlinburg, Tennessee.

WARSA, J.S., WAREING, T.A., MOREL, J.E., 2003. *On the degraded effectivness of Diffusion Sythetic Acceleration for multidimensional S_N Calculations in the presence of material discontinuities.* Int. Conf. Nuclear Mathematical and computational Sciences, Gatlinburg, Tennessee.

WALKER, D, W. 1994. *The design of a standard message passing interface for distributed memory concurrent computers.* Parallel Computing, vol. 20,$N^0 4, pp.657-$674.

WILKINSON, B., ALLEN, M., 1999. *Parallel Programming – Techniques and Applications Using Network of Workstations and Parallel Computers.* Prentice Hall, Upper Saddle River, New Jersey.

WILSON, W., VUJIC, J. and GU, A. 1993. *Parallel Multiple-Assembly Calculation in GTRAN2/M.* Transaction of the American Nuclear Society Winter Meeting, San Fransisco, Ca.

WILSON, G. V. 1995. *Pratical Parallel Programming.* The MIT Press.

WU, G.J., ROY, R., 2003a. *A New Characteristics Algorithm for 3D Transport Calculations. Ann. Nucl. Energy* **30**, 1-16.

WU, G.J., ROY, R., 2003b. *Acceleration Techniques for Trajectory-based Deterministic 3D Transport Solvers. Ann. Nucl. Energy* **30**, 567-583.

WU, G.J., ROY, R., 1999. *Self-collision rebalancing technique for the MCI characteristics solver.* Int, twentieth Annual Conf. Canadian Nuclear Society, Montreal.

ZHANG, X., YAN, Y. and HE, K. 1994. *Latency Metric: An Experimental Method for Measuring and Evaluating Parallel Program and Architecture Scalability.* Journal of Parallel and Distributed Computing, 22, 392-410.

ZIKA, M. R. and ADAMS, M.L. 2000. *Transport Synthetic acceleration for long-characteristics assembly-level problems.* Nucl. Sci. Eng. **134**, 135-158.

ANNEXE I

PUBLICATIONS COMPLÉMENTAIRES

Publication 1

Development and Parallelization of the Three-Dimension Characteristics Solver MCI of DRAGON

PHYSOR-2002: International Conference on the New Frontiers of Nuclear Technology : Reactor Physics, Safety and High-Performance Computing

Publié en octobre 2002

PHYSOR 2002, Seoul, Korea, October 7-10, 2002

DEVELOPMENT AND PARALLELIZATION OF THE THREE-DIMENSIONAL CHARACTERISTICS SOLVER MCI OF DRAGON

M. Dahmani, G. J. Wu*, R. Roy and J. Koclas
Nuclear Engineering Institute
École Polytechnique de Montréal
P.O.Box 6079, Station Centre-Ville
Montreal, H3C 3A7 CANADA
{Mohamed.Dahmani , Robert.Roy, Jean.Koclas} @polymtl.ca

ABSTRACT

In this paper, recent advances in parallel software development for solving neutron transport problems are presented. The method of characteristics is based on the resolution of the differential transport equation following the tracking lines in order to collect the local angular flux components. Due to the excessive number of tracks in the demanding context of 3D large-scale calculations, reliable acceleration techniques are developed in order to obtain a faster iterative solver, especially for problems with high-scattering ratio. The load balancing strategies include a global round-robin distribution of tracks, an approach where we forecast the calculation load implied by each track length and a macro-band decomposition where tracks crossing the same regions are grouped together. The performance of the PVM and MPI implementations in the characteristics solver is analyzed for realistic applications.

1. INTRODUCTION

DRAGON[1] is a lattice cell code which uses the Collision Probability (CP) method for solving the neutron transport equation in arbitrary geometrical domain. A new solver (called MCI) based on the method of characteristics (MOC) was developed as an alternative technique for 3D domains. For large problems, the MOC technique is very promising because it does not generate huge CP matrices, and still provides solutions as accurate as the CP method for general 3D geometries. However, the MCI iterative solver can converge slowly, especially in the presence of high-scattering media. In order to minimize the number of iterations, a Self-Collision Rebalancing (SCR) method was designed as an acceleration method for the MCI sequential solver[2]; this method uses first-collision probability to recover from the local self-scattering effect. Another new technique, the Track Merging Technique (TMT), was also designed to reduce the number of tracking lines. Using this technique, two tracking lines crossing the same regions in the same order are merged together. It can be easily implemented inside the ray tracing routine that computes the tracks. The TMT is generally very efficient because more than half of tracking lines can be merged together without loosing any accuracy.

* G.J.Wu can now be reached at E-mail address: GuangJunWu@netscape.net

Based on these ideas, a sequential module MCI was developed in DRAGON to treat supercell 3D problems with isotropic boundary conditions. This technology was found accurate with respect to several standard transport problems specific to CANDU reactors, where reactivity devices are perpendicular to the fuel channels [2]. Using this sequential module, most of regular supercells have been solved in approximately the same number of outer iterations as in the CP method, even with the high-scattering ratio due to the heavy water moderator.

Nevertheless, it is still far from possible to solve the transport equation over the complex geometry of a whole power reactor core without various approximations. However, the exponential growth in capacity and power of computer resources enables the nuclear engineering community to improve their computational models and to explore the limits of older models. Nowadays, the three-dimensional effects for various critical configurations can be simulated using new parallel algorithms. Here, we will present one of the most flexible and well-organized algorithms known to perform such 3D state-of-the-art computations in a distributed environment.

The simplified static transport equation for obtaining a flux ϕ from a source Q is given by:

$$\hat{\Omega}.\vec{\nabla}\Phi(\vec{r},\hat{\Omega},E)+\Sigma_t(\vec{r},E)\Phi(\vec{r},\hat{\Omega},E)=Q(\vec{r},\hat{\Omega},E), \qquad (1)$$

where $\vec{r}$ is a spatial point in the domain D, $\hat{\Omega}$ is a solid angle and E is the energy. The parallelization based on CP method obtained after distributing the energy variable in multi-group solver, as well as the usual spatial domain decomposition methods, have already been developed a few years ago [3], and the spectrum of applications for similar techniques has been extended to other related fields such as radiography [4]. The aim of this paper is to explain the newest approaches based on the characteristics formulation of the transport equation (1) and to introduce the new parallel computational techniques implemented to extend the MCI sequential solver.

Recently, domain decomposition techniques have been applied to large-scale parallel transport calculations using the method of characteristics. The spatial decomposition of multi-assembly problems where neutron paths are directly linked together when crossing assemblies exhibit limited speed up (5.3 on 8 processors) on shared memory Sun Entreprise4000 because of tight coupling between the assemblies [5]. The angular decomposition technique, where directions are distributed among a set of processors, has also been tested [6]. Speed up obtained using this angular decomposition is of the order of 3.7 for 4 processors, and 6.8 for 8 processors. These techniques were restricted to two-dimensional problems, but they give the trend that we are looking for when we apply our characteristics method to three-dimensional geometries.

The paper is organized as follows. In section 2, we present a brief review of MOC principles, and the particular 3D implementation of these principles in our solver. Our iterative MOC scheme is also described with the related acceleration techniques that can be applied in the 3D context. In section 3, we present different options for the implementation of a general MCI parallel solver. Section 4 is composed of numerical tests, most of these appearing in CANDU supercells, and the parallel performance for various parallel options is studied. In the last section, we conclude by a discussion on prospective future work in order to obtain a large-scale distributed transport solver in the coming years.

2. THE 3D CHARACTERISTICS SOLVER MCI

2.1 MOC PRINCIPLES AND SPECIFIC 3D IMPLEMENTATION

The main idea of the method is to solve the differential form of the Boltzmann equation following the tracking lines (also called "characteristics"). Assuming a finite domain V split into homogeneous regions, each having a volume V_j, the average (one-group) flux Φ_j is given by:

$$V_j\Phi_j = \int_{V_j} d^3r \int_{4\pi} d^2\Omega\Phi(\vec{r},\hat{\Omega}) = \int_{\Gamma} d^4T \int_{-\infty}^{\infty} dt\,\chi_{V_j}(\vec{T},t)\Phi(\vec{p}+t\hat{\Omega},\hat{\Omega}) \qquad (2)$$

A characteristics line $\vec{T}$ is determined by its orientation $\hat{\Omega}$ along with a reference starting point $\vec{p}$ for the line. The variable t refers to the local coordinates on the tracking line and the function $\chi_{V_j}(\vec{T},t)$ is defined as 1 if the point $\vec{p}+t\hat{\Omega}$ on the line $\vec{T}$ in the region V_j, and 0 otherwise. Assume that a line $\vec{T}$ crosses K regions before reaching the external boundary, and define the crossing points $r_k = \vec{p}+t_k\hat{\Omega}$ ordered in the neutron traveling direction. The d^4T element is then decomposed into a solid angle element $d^2\Omega$ a corresponding plane element d^2p. The Γ domain is covered by a quadrature set of solid angles and by scanning the plane $\pi_{\hat{\Omega}}$ perpendicular to the selected direction $\hat{\Omega}$ for the starting point $\vec{p}$. The differential transport equation for angular flux that has to be solved on each segment k is now:

$$\left(\frac{d}{ds}+\Sigma_{N_k}\right)\Phi(\vec{r}_{k-1}+s\hat{\Omega},\hat{\Omega}) = \frac{Q_{N_k}}{4\pi};\ s\in[0,L_k] \qquad (3)$$

For the chosen line $T(\hat{\Omega},\vec{p})$, the collection of segment lengths L_k and region numbers N_k for each region encountered along the line must be calculated (and can eventually be recorded in sequential binary files). Assuming isotropic input currents at the external boundary and isotropic sources, a recursive solution can be found from the following simple attenuation equation:

$$\phi_k = \phi_{k-1}e^{-\tau_k} + \frac{q_k}{\sigma_k}(1-e^{-\tau_k}), \qquad (4)$$

where local sources are given by $q_k = Q_{N_k}/4\pi$, the local total cross sections are $\sigma_k=\Sigma_{N_k}$ and total optical paths when crossing the region N_k are $\tau_k=\Sigma_{N_k}\times L_k$. Reciprocity relations allow us to solve concurrently for direction $\hat{\Omega}$ and for its reverse direction $-\hat{\Omega}$ using the same line; however, the input currents are not the same at both ends. The scalar flux in region j can be recovered by a reductive operation over all lines:

$$\Phi_j=\frac{1}{4\pi\Sigma_j V_j}\sum_T \omega_T \sum_k \delta_{jN_k}\Delta\phi_k+\frac{Q_j}{\Sigma_j}, \qquad (5)$$

where δ is the Kronecker symbol and $\Delta\phi_k=\phi_k-\phi_{k-1}$ is the local flux difference.

Equations (2-5) are the basis of all characteristics solvers that will be used here under the following multi-group iterative scheme:

1. Guess new fission sources and incoming currents (outer loop).
2. Guess scattering sources (inner loop).
3. Compute the flux map for each energy group:
   (a) compute local solutions for each characteristics line;
   (b) apply local acceleration techniques;
   (c) perform a reductive sum of all contributions to the flux moments;
   (d) apply global inner acceleration techniques.
4. If flux map are not converged, go to 2.
5. Apply global outer acceleration techniques.
6. Compute critical factors for next neutron generation.
7. If not converged, go to 1.

This iterative scheme exhibits good performance only if consistent acceleration techniques are used. We will now present the current state of work on these acceleration techniques pertinent to the MCI module in DRAGON.

2.2 SELF-COLLISION REBALANCING TECHNIQUE

As we can see from the iterative scheme described above, we apply at Step 3.(b) any local acceleration techniques to reduce the number of inner iterations. Several acceleration techniques are developed and used for MOC [7]. In our characteristics solver, we use the Self-Collision Rebalancing (SCR) technique [8]. This technique is based on the equivalence between the collision probabilities method and the method of characteristics. The SCR uses the self-collision probabilities (first-flight collision probabilities from one region to itself) in order to rebalance the energy distribution of the scalar flux for each region separately.

2.3 TRACK MERGING TECHNIQUE

In the process of generating tracking lines for a large domain, two successive lines can cross exactly the same regions with the same direction. Segment lengths can be slightly different: for a reference length L, one track may have length $L+\varepsilon$ while another has $L-\varepsilon$. It was shown that the two tracks could be merged together with $O(\varepsilon^2)$-order of error on the local angular flux. The resulting line takes the averaged segment lengths and additional weights of the original lines as its properties [8].

Another step in attempting to group together characteristics lines having the same behavior is the macro-band grouping [7]. In this concept, we identify tracks by the region numbers that they are crossing. This classification encompasses two different attributes: the spatial regions that are crossed plus the set of directions that allow this crossing sequence. Once the finite set of these attributes has been identified, it is possible to define an almost perfectly vectorized process for each macro-band. This approach could be useful in the context of SMP processors, providing another (fine-grain) level of parallelism.

3. PARALLELIZATION IN THE MCI SOLVER

For cases with a large number of regions, even when using both TMT and SCR techniques, the iterative solver based on characteristics method can become very expensive. In order to solve such a large-scale problems with minimum amount of time, we use a parallel approach in the MCI solver.

Parallelization of transport solvers is generally based on domain decomposition; however, this can include spatial, angular or energy decomposition. In characteristics methods context, the basic computation unit is focused on repeating the same sequence of calculations on each tracking line. Obviously, this is the finest granularity level of operations that can be performed without any communication. The idea is to distribute the tracking lines on several different processors. The number of lines is much larger than the number of processors available. Therefore, the lines are grouped and each processor takes control of a group. Several strategies are used to group the lines in order to load balance the parallel calculation. In the following, we assume that N_T represents the number of tracks and that N_p is the number of processors.

3.1 STATISTICAL LOAD BALANCING

These options are statistical in the way that they rely on the uniform distribution of number segments when grouping tracks. Three of these options have been developed:

- **ANGL**: all tracks of the same direction are grouped together; the number of directions must be equal to the number of processors;
- **SPLT**: subsets of N_T / N_p tracks respect the order in which these would be sequentially generated;
- **STRD**: each tracks is given in a round-robin fashion to each processor, so that track i is on processor $i \bmod p$.

Different tests were performed with these options, and it was shown that the **STRD** option is the most efficient in statistically preserving the load balance [9].

3.2 DETERMINISTIC LOAD BALANCING

Only one option was encoded using a deterministic approach based on the calculation load of each single track:

- **MCRB**: the number of segments of each track is taken into account as a weighting factor for distributing tracks; each processor receives an equivalent total weight based on the calculation load.

This option represents the most uniform load balancing. The tests done using this option show speedups nearly linear and similar to the statistical **STRD** option [9]. The **MCRB** option is still under investigation and, in the context of distributed grid computing, it could help to synchronize the reduction operations.

3.3 PERFORMANCE OF MESSAGE-PASSING ALGORITHMS

After using one of the options described above, the total number of tracks N_T is distributed on the processors. At each iteration step, a multicasting communication process is used to recover the partial contribution to the angular fluxes; each processor has its own copy of the flux and the source arrays with the same consistent ordering for unknowns (regions and energies). The scalar flux is then reconstructed on every single processor before the iteration procedure starts (multicasting communication process). The original MCI parallel solver was designed and coded in such way it runs on an arbitrary number of processors using the parallel virtual machine (PVM) environment [10]. Recently, an MPI version of the MCI parallel solver was also implemented to improve the message passing performance.

3.3.1 PVM IMPLEMENTATION

The angular flux is scattered by one given processor to all others processors belonging to the virtual machine. A process begins to construct their subpart of the flux and then scatters its results to all other processes using a **pvmfmcast** subroutine call. At this level, all processes are synchronized. After that, every process will receive (**pvmfrecv**) the results from the others and accumulate their contributions (loop on other processors). The **pvmfrecv** subroutine is called $N_p - 1$ times for each process, this non-collective operation can increase the communication time and can therefore downgrade the parallel performance.

3.3.2 MPI IMPLEMENTATION

Recently, the MPI version was implemented in the MCI solver to take advantage of the growth of the MPI library regarding to the collective communication subroutines and of its extended portability [11]. The angular flux is then scattered to all processors by using the **Mpi_Bcast** subroutine that performs the send and receive in one synchronized step. When constructing the flux, we have used the **Mpi_Allreduce** to perform the collective communication between the process, making the reduction sum on all contributions at the same time. The reduction sum is then stored in the buffer and a copy is scattered to all processes (see Fig. 1). In the following, we will compare both implementations.

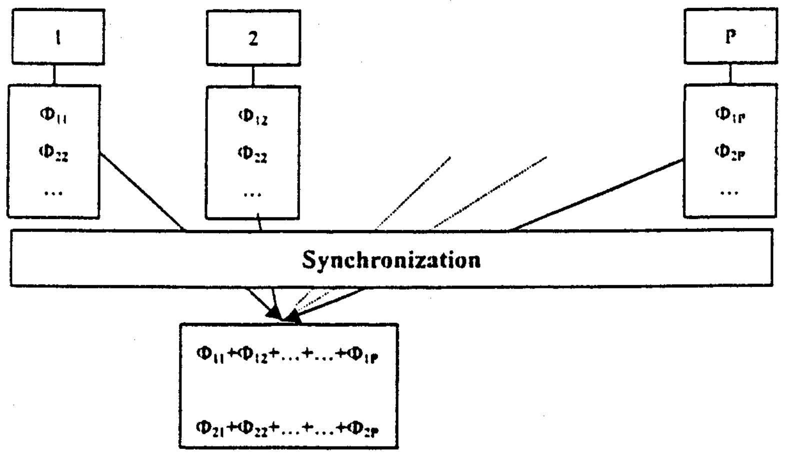

1
2
P
Φ_{11}
Φ_{22}
Φ_{12}
Φ_{22}
Φ_{1P}
Φ_{2P}
Synchronization
$\Phi_{11}+\Phi_{12}+\ldots+\ldots+\Phi_{1P}$
$\Phi_{21}+\Phi_{22}+\ldots+\ldots+\Phi_{2P}$

Figure 1. Collective communication by reduction sum in MPI implementation.

4. NUMERICAL RESULTS

Tests were done for CANDU6 absorber rod supercell calculations in 3D geometry [2]. Two horizontal fuel bundles and one vertical rod compose the domain. Tests were performed on two Beowulf clusters located at the Center for Research in Computation and its Applications (CERCA) [12]:

- **Beowulf 1** is a cluster of 16 nodes equipped with AMD Athlon 1.4 GHz processor and 1 GB memory connected by a Fast Ethernet switched network;
- **Beowulf 2** is a small SMP composed of 8 QuadXeon multiprocessors, each of these 8 nodes has 4 processors sharing a 4GB memory. Nodes are connected with a 1 Gb/s Myrinet switch.

All the tests were done using the STRD option with SCR acceleration and with three different levels of track merging:

- **No TMT**: no track merging at all;
- **TMT-1**: one line can be merged to the next one as the ray tracing is performed, thus this is equivalent to merging lines on a unique dimension;
- **TMT-2**: more than the previous option, lines are sorted and merging is done on both dimensions of the perpendicular plane.

In Table I, we vary the density of tracks (DT) (tracks/cm^2) and show the CPU time spent by the PVM implementation of the MCI solver for different number of processors on Beowulf 1. The CPU time decreases with the number of processors, but this decreases is little lower for the TMT-1 option and even lower for TMT-2. This is due to the insufficient number of lines of TMT-2, so the communication time increases faster than computation time as we can see in Figure 2 which represents the speedup variation with the number of processors. In Table II, we show the speedup variation of the PVM implementation on both clusters. The speedup on the SMP cluster is almost linear and the closest to the ideal speedup because the communications are faster. In Figure 3, we compare the respective speedup curves of the MPI and PVM implementations with and without merging lines. The MPI speedup curve is more linear than the PVM one for both cases; this is due to the increased performance of the message-passing MPI collective subroutines. With the PVM implementation, we begin to lose performance after 8 processors when the communication time becomes important; this shows that the MPI implementation is more scalable.

CONCLUSIONS

We have presented here the recent developments done in the characteristics solver MCI. To obtain a fast iterative solver, self-collision rebalancing and track merging techniques were used. Results have shown that this characteristics solver can offer fast and accurate solutions when implemented with efficient parallel computation algorithms. The decrease in the computational time strongly depends on the number of tracks in the domain. When this number is significantly larger than the number of regions, the computing time is dominant and the parallelism is efficient.

Next step would be to introduce modular ray tracking schemes based on pattern recognition for the macro-bands. Calculation along characteristics is so simple that mobile agents could be defined to compute on very limited view of the whole-core system, allowing these solvers to evolve on grid computers. In order to further improve the parallel performance and to allow for the solver to become more portable to the SMP architectures clusters, I/O operations could also be performed on the tracking file using MPI-2.

ACKNOWLEDGEMENTS

This work has been carried out partly with the help of a grant from the Natural Science and Engineering Research Council of Canada. M. Dahmani would like to thank the Beowulf administrators (S. Fourmanoit, P. Hamelin and M. Béland) at CERCA for their help.

REFERENCES

1. G. Marleau, A. Hebert and R. Roy, "A User's Guide for DRAGON", Technical Report **IGE-174 Rev. 3.**, École Polytechnique de Montréal (1997).
   A copy of this report is available at **http://www2.polymtl.ca/nucl**
2. G. J. Wu and R. Roy, "A New Characteristics Algorithm for 3D Transport Calculation", to appear in *Annals of Nuclear Energy* (2002).
3. A. Qaddouri, R. Roy, M. Mayrand, B. Goulard, "Collision Probability Calculation and Multigroup Flux Solvers Using PVM", *Nucl. Sci. Eng.*, **123**, 392 (1996).
4. F. Inanc, B. Vasiliu, D. Turner, "Parallel Implementation of the Integral Transport Equation-Based Radiography Simulation Code", *Nucl. Sci. Eng.*, **137**, 173 (2001).
5. S. Kosaka and E. Saji, "The Characteristics Transport Calculation for a Multi-Assembly System using Neutron Path linking Technique", *Proc. Mathematics and computation ANS M&C 99*, September (1999).
6. G. S. Lee, N. Z. Cho, S. G. Hong, "Acceleration and Parallelization of the Method of Characteristics for Lattice and Whole-Core Heterogeneous Calculation", *Proc. of PHYSOR 2000*, May (2000).
7. G. J. Wu, "Développement d'une méthode des caractéristiques tridimensionnelle et application aux calculs de supercellules d'un réacteur CANDU", *Ph.D. dissertation*, Ecole Polytechnique de Montréal (2001).
8. G. J. Wu and R. Roy, "Self Collision Rebalancing Technique for the MCI Characteristics Solver", *20th Annual Conference of Canadian Nuclear Society*, Toronto (2000).
9. G. J. Wu and R. Roy, "Parallelization of Characteristics solver for 3D neutron transport", *Lecture Notes in Computer Sciences*, **2131**, 344 (2002)
10. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, "PVM3 User's Guide and reference Manual", Report **ORNL/TM-12187**, Oak Ridge National Laboratory (1994).
11. W. Groop, E. Lusk, A. Skjellum, "Using MPI: Portable Parallel Programming with the Message-Passing Interface", MIT Press, Cambridge (1994).
12. For informations see: http://www.cerca.umontreal.ca

Table I. CPU time for the PVM implementation of MCI vs. the number of processors on Beowulf 1 when varying the tracking density and the track merging.

		Number of processors					
DT	Track merging	1	2	4	8	12	16
		MCI CPU time (s)					
2.5	No TMT	70.96	37.38	19.68	11.81	9.93	9.4
2.5	TMT-1	36.28	19.77	10.90	7.13	6.55	6.51
2.5	TMT-2	10.39	6.14	4.36	3.9	4.34	4.5
4	No TMT	114.49	58.92	31.11	17.53	13.66	12.3
4	TMT-1	54.14	28.31	15.67	9.88	8.43	8.28
4	TMT-2	12.29	7.27	4.76	4.25	5.24	5.12
10	No TMT	254.43	132.54	65.95	35.3	25.19	21.52
10	TMT-1	86.61	45.53	23.62	13.43	10.63	9.86
10	TMT-2	13.78	8.16	5.38	4.46	4.91	5.4

Table II. Speedup variations on both clusters for the PVM implementation of MCI (results are for No TMT or TMT-1 and a fixed density DT = 4).

		Number of processors					
Beowulf	Track merging	1	2	4	8	12	16
		Speedups					
1 (REF)	No TMT	1	1.94	3.68	6.53	8.68	9.60
2 (SMP)	No TMT	1	1.98	3.88	7.48	10.69	13.60
1 (REF)	TMT-1	1	1.89	3.45	5.47	6.42	6.37
2 (SMP)	TMT-1	1	1.96	3.78	6.99	9.38	12.28

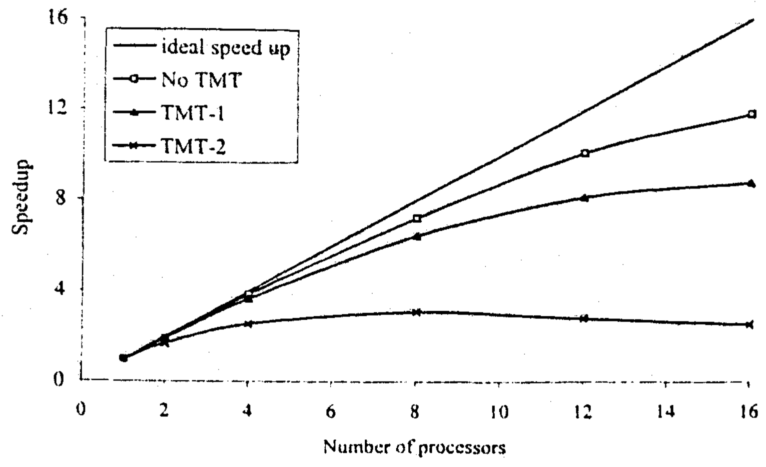


Figure 2. Speedup for different track merging options (PVM implementation)

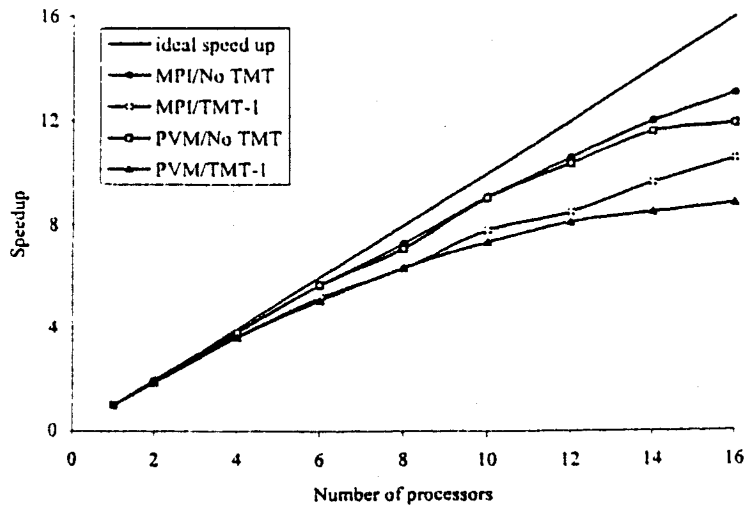


Figure 3. Speedups for PVM and MPI implementation with and without TMT (DT=10)

Publication 2

Parallel Distribution of Tracking for 3D Neutron Transport Calculation

International Conference on Mathematical and Computational Sciences: A Century In Review-A Century Anew

Publié en avril 2003

Nuclear Mathematical and Computational Sciences: A Century in review, A Century Anew
Gatlinburg, Tennessee, April 6-11, 2003, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2003)

PARALLEL DISTRIBUTION OF TRACKING FOR 3D NEUTRON TRANSPORT CALCULATION

M. Dahmani, R. Roy and J. Koclas
Nuclear Engineering Institute
École Polytechnique de Montréal
P.O. Box 6079, Station Centre-Ville
Montréal H3C 3A7, CANADA
{Mohamed.Dahmani, Robert.Roy, Jean.Koclas}@~~poly~~mtl.ca

ABSTRACT

In this paper, we present the recent advances in parallel software development for solving neutron transport problems in context of the 3D characteristics method. The method of characteristics is based on the resolution of the differential transport equation following the tracking lines to collect the local angular flux components. The parallelization of this solver is based on distributing a group of tracks, generated by the ray tracing procedure, on several processors. Different distributing and load balancing techniques are presented. Due to the excessive number of the tracks in the demanding context of 3D large-scale calculations, two different strategies for data management for these large amounts of data are implemented. The first strategy uses the regular files and the second one uses the database. To access the data, we use either the regular sequential mode or the MPI-IO as API.

Key Words: Neutron transport, characteristics method, parallel IO.

1. INTRODUCTION

The method of characteristics (MOC) has been largely used to solve the transport equation because of its good performance for large problems regarding to the Collision Probability (CP) method. Unlike the CP method, the MOC does not generate huge matrices, and still provides solutions as accurate as the CP method. Assuming that there are no interactions between neutrons in the medium and the neutrons traveling take a straight trajectories, the characteristics method are based on the resolution of the differential transport equation following the tracking lines to collect the local angular flux components. The solution of the transport equation using MOC can then be split into:

- Geometry preparation and tracking generation;
- Integration of differential transport equation using the generated tracking lines and then computation of the scalar flux.

In the context of the 3D large-scale calculations, the number of tracking lines can become prohibitive. Therefore, the data management for these tracks is a real challenge: at same time, we want to reduce the CPU time and to avoid the problems with the I/O operations which is often the bottleneck for such applications.

In recent years, the parallel computers get faster and more accessible economically. The great advances have been made in the CPU and the communication performances. Accordingly, the scientists are increasingly use parallel computers to solve problems that require a large amount of computing effort. Most of these applications need also to perform I/O operations, such as reading data and writing the results. In general, the I/O speed is slower than the CPU and communication speed. In order for parallel computers to be fully usable in solving large-scale problems, the I/O performance must be scalable and balanced with respect to the CPU and communication performance of the system. The studies have shown that an inappropriate Application Program Interface (API) is often the cause of poor I/O performance in applications [1].

Like in many scientific applications, the nuclear engineering applications, especially the neutron transport calculation for large domains, generate and use a large amount of data. The static transport equation must be solved to obtain the flux from a source Q :

$$\hat{\Omega} \cdot \vec{\nabla} \Phi(\vec{r}, \hat{\Omega}, E) + \Sigma_t(\vec{r}, E) \Phi(\vec{r}, \hat{\Omega}, E) = Q(\vec{r}, \hat{\Omega}, E) \qquad (1)$$

where $\vec{r}$ is a spatial point in the domain D, $\hat{\Omega}$ is the solid angle and E is the energy.

In this paper, we will present the developments made to optimize the data storage of the tracks in context of the 3D characteristics formulation. To reduce the number of these tracks in the domain, Track Merging Technique (TMT) was implemented. Using this technique, two tracking lines crossing the same regions in the same order are merged together. The TMT is generally very efficient because more than half of tracking lines are merged together without loosing accuracy. It can also be easily implemented inside the ray tracing routine that computes the tracks. In order to access the tracking data stored in the files, we need to do the I/O operations. We will also present here the developments made to perform such operations. An MPI-IO implementation has been used as an API, so the processors belonging to the same communicator can access the data in the same file collectively. Another alternative and complementary method is to use a database to store and access the information about the tracking lines. Once the tracking lines are generated and stored, we have to answer to the question: how to distribute these tracking lines among different processors on parallel machines without loosing a performance? We will explain the load balancing strategies, including a global round-robin distribution of tracks where we forecast the calculation load implied by each track length and also a macro-band decomposition where tracks crossing the same regions are grouped together.

The paper is organized as follows: In section 2, we represent the brief review of the 3D MOC principles, the tracking generation procedure, and the resolution of the transport equation on one tracking line. Section 3 is dedicated to the strategies we use to distribute and to manage the data associated with the tracks; track merging and load balancing techniques will be explained. Section 4 shows numerical tests performed with the CANDU supercells. In the last section, we conclude by discussion on prospective future work in order to obtain a loosely coupled large-scale distributed transport solver in coming years.

2. THE 3D CHARACTERISTICS FORMULATION

Assuming a finite domain V split into homogeneous regions, each having a volume V_j, the average (one-group) flux Φ_j is given by:

$$V_j\Phi_j = \int_{V_j} d^3r \int_{4\pi} d^2\Omega\, \Phi(\vec{r},\hat{\Omega}) \tag{2}$$

The main idea of the method is to solve the differential form of the Boltzmann equation following the tracking lines (also called "characteristics"). To involve the tracking lines, we change the reference using the local coordinates, equation (1) becomes:

$$V_j\Phi_j = \int_{\Gamma} d^4T \int_{-\infty}^{\infty} dt\, \chi_{V_j}(\vec{T},t)\Phi(\vec{p}+t\hat{\Omega},\hat{\Omega}) \tag{3}$$

A characteristics line $\vec{T}$ is determined by its orientation $\hat{\Omega}$ along with a reference starting point $\vec{p}$ for the line. The variable t refers to the local coordinates on the tracking line and the function $\chi_j(\vec{T},t)$ is defined as 1 if the point $\vec{p}+t\hat{\Omega}$ on the line $\vec{T}$ in the region V_j, and 0 otherwise.

2.1. Tracking Generation Procedure

The d^4T element is decomposed into a solid angle element $d^2\Omega$ and a corresponding plane element d^2p. The Γ domain is covered by a quadrature set of solid angles and by scanning the plane π_{Ω} perpendicular to the select direction $\hat{\Omega}$ for the starting point $\vec{p}$. The quadrature used in our code **DRAGON** [2] is the Equal Weight Quadrature EQ_N. The directions in EQ_N are choosing to be axially invariant.
Let $\hat{\Omega}$ be one of the values of angular quadrature, the set of the planar quadrature to the direction $\hat{\Omega}$ is a set of the equidistant and parallel lines to $\hat{\Omega}$: this is the characteristics lines or tracking lines. [3]
Assume that a line $\vec{T}$ crosses K regions before reaching the external boundary, and define the crossing points $r_k=\vec{p}+t_k\hat{\Omega}$ ordered in the neutron traveling direction. We obtain then K segments $[r_{k-1},r_k]$ and we define length of segment L_k as:

$$L_K=(\vec{r}_k-\vec{r}_{k-1}).\hat{\Omega}, \quad k=1,2,\ldots..,K \tag{4}$$

and N_k is the region where the segment k is located.
For the opposite direction $\hat{\Omega}'=-\hat{\Omega}$, we use the same lines but in the opposite way:

$$\vec{r}_k'=\vec{r}_{K-k}, \quad k=1,2,\ldots,K$$

$$L_k = L_{K+1-k}, \qquad k=1,2,\ldots,K\,.$$

This will allow us to decrease the number of tracking sweep.
The procedure is to define the geometry, to split the domain into regions, and generate the lines according to techniques explained above. To solve the differential transport equation, we have to associate to each region its nuclear data (cross sections).

2.2. The 3D characteristics Solver MCI

2.2.1. Solution on a single track

For a given track line $\vec{T}$, we define the crossing point $\vec{r}_0, \vec{r}_1, \ldots\ldots, \vec{r}_K$. For each segment of line, we must calculate the integrated angular flux:

$$L_k \bar{\phi}_k(\vec{T}) = \int_0^{L_k} dt\, \Phi(\vec{r}_{k-1} + t\hat{\Omega}, \hat{\Omega}) \tag{5}$$

The angular flux $\Phi(\vec{r}_{k-1}+t\hat{\Omega},\hat{\Omega})$ is obtained by solving the differential transport equation on each segment k :

$$\left(\frac{d}{ds} + \Sigma_{N_k}\right)\Phi(\vec{r}_{k-1} + s\hat{\Omega}, \hat{\Omega}) = \frac{Q_{N_k}}{4\pi}; \quad s \in [0, L_k] \tag{6}$$

Assuming isotropic input current at the external boundary and isotropic sources, a recursion is based on the following simple attenuation equation:

$$\phi_k = \phi_{k-1} e^{-\tau_k} + \frac{q_k}{\sigma_k}(1 - e^{-\tau_k}), \tag{7}$$

where local sources are $q_k = \frac{Q_{N_k}}{4\pi}$, local total cross section are $\sigma_k = \Sigma_{N_k}$ and total optical path when crossing the region N_k are $\tau_k = \Sigma_{N_k} \times L_k$. Reciprocity relations allow us to solve concurrently for direction $\hat{\Omega}$ and for the inverse direction $-\hat{\Omega}$ using the same line; however, the input currents are not the same at both ends.

At this stage, the required local data is a collection of segment lengths L_k and numbers N_k for each region encountered along the line. The nuclear data needed are the local total cross section $\sigma_k = \Sigma_{N_k}$ and the scattering cross section from one group to it self $(\Sigma s^{g\to g})_{N_k}$. No scattering matrices are necessary; this approach is important to preserve certain linearity in calculation and to preserve independency between the calculations performed on different tracks. This represents the atomic level for our application from which we can construct the overall solution for each region, each angle, and each energy group.

2.2.2. The MCI solver

Using equation (3), we get the integrated flux:

$$V_j \Phi_j = \int_{\Gamma} d^4 T \sum_k \delta_{jN_k} L_k \bar{\phi}_k (\vec{T}) \tag{8}$$

where δ is the Kronecker symbol, and where k runs over all integers. All the characteristics lines are accepted, but only the contributions of segments crossing the same region j are added together.
To preserve the exact volumes, the characteristics lines have to be normalized. This is achieved by evaluating for each angle a numerical approximation V_j for volumes using segment lengths. We define then the angular volume for each direction $\hat{\Omega}_i$:

$$V_j(\hat{\Omega}_i) = \sum_n p_{i,n} \sum_k \delta_{jN_k} L_k$$

integrating over all angles, we obtain :

$$V_j = \frac{1}{4\pi} \sum_i \omega_i V_j(\hat{\Omega}_i)$$

where the $p_{i,n}$ is the weight associated with the line $\vec{T}_{i,n}$ having a point $\vec{p}_{i,n}$ center of the mesh as origin in the perpendicular plane π_{Ω_i} ; and where ω_i is the weight corresponding to the direction $\hat{\Omega}_i$ in the quadrature. The normalization is done by multiplying L_k by the factor $\frac{V_j}{V_j(\hat{\Omega}_i)}$. The scalar flux in region j can be recovered by a reductive addition:

$$\Phi_j = -\frac{1}{4\pi \Sigma_j V_j} \sum_i \omega_r \sum_k \delta_{jN_k} \Delta\phi_k + \frac{Q_j}{\Sigma_j}, \tag{9}$$

where δ is the Kronecker symbol and $\Delta\phi_k = \phi_k - \phi_{k-1}$.

The basic computation unit is focused on repeating the same sequence of calculations on each tracking line. Obviously, this is the finest granularity level of operations that can be performed without any communication. The number of lines is obviously much larger than the number of processors available. Therefore, the lines are grouped and each processor takes control of a group. At each iteration step, multicasting is used to recover the partial contribution to the angular fluxes; each processor has its own copy of the flux and the source arrays with a consistent unknown ordering. The scalar flux is then reconstructed on every single processor before the iteration procedure starts (multicasting communication process). In our MCI parallel solver, we use the message-passing interface MPI to communicate between the processes [4]. To reduce the number of iterations the self-collision rebalancing acceleration technique is used [5].

3. LOAD BALANCING AND DISTRIBUTION OF THE TRACKING

Since the time spent in calculation on one track depends linearly on the number of segments [3], the number of tracking distributed on each processor will be the major criteria to build our load balancing algorithm.

3.1. Track Merging Technique

In process of generating the tracking lines for a large domain, two successive lines cross exactly the same regions with the same direction. Segment lengths can be slightly different: for a reference length L, one track may have length $L+\varepsilon$ while the other has $L-\varepsilon$. It was shown that the two tracks can be merged together with $O(\varepsilon^2)$-order of error on the local angular flux. The resulting line takes the averaged segments length and additional weights of the original lines as its properties. Two levels of merging are applicable:

- TMT-1: one line can be merged to the next one as the ray tracing is performed, thus this equivalent to merging lines on a unique dimension;
- TMT-2: lines are stored and the reduction is done on both dimension of the perpendicular plane.

3.2. Load balancing strategies

3.2.1. Statistical load balancing

- Distribution on a beam (SPLT option): in this option, we subdivide the number of track on packs of $\frac{N_L}{N_p}$ tracks generated by EXCELT module. The first processor will work on the first $\frac{N_L}{N_p}$ generated, the second processor takes the next $\frac{N_L}{N_p}$ and so on. Figure 1 shows the distribution of the tracks using this option considering one direction with its associated perpendicular plane for four processors.
  For the domains that contain a lot of small regions, if we don't choose the right number of processors, one of the track groups may have a large number of segments than other ones.

- Round-robin distribution (STRD option): the tracks are distributed using constant intervals. However, the first processor take the tracks number $1, 1+N_p, 1+2N_p,...$ the second processor $2, 2+N_p, 2+2N_p,...$ In general, the processor i take the tracks $i+1 \mod N_p$.
  Using this option, it's statistically unexpected to have two groups of tracks having great difference in the average number of segments. We represent this option of distribution of tracks over four processors in Figure 1.

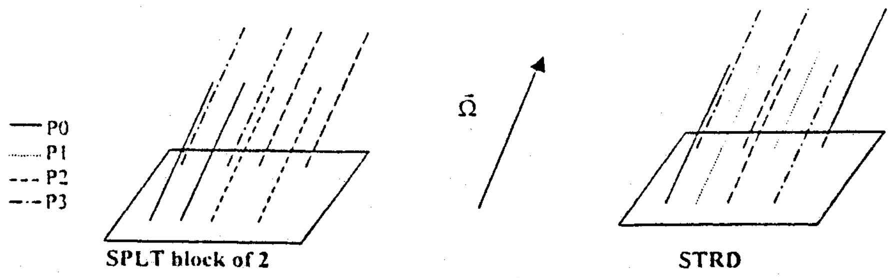


Figure 1. Distribution of the tracks on 4 processors for SPLT and STRD options.

- Distribution on angles (ANGL option): all the tracks having the same direction are grouped together. In this option, the number of processors has to be equal to the number of angles. So, each processor takes the tracks with the same direction.
  The disadvantage of this option is the limitation in the number of processors, for example, if we have four processors and five directions, one of the processors will be doubly loaded. For two directions, two of the four processors will don't be used (see Figure 2. for this option).

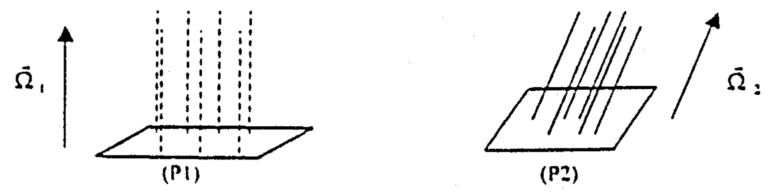


Figure 2. Distribution of the tracks with 2 directions and 2 processors for ANGL options.

3.2.2. Deterministic load balancing

The only one option was coded using a deterministic approach based on the calculation load of each single track is the macro-band (MCRB option). The macro-band can be defined as combination of a section of a geometric volume V_M and an angle $\hat{\Omega}_M$ that satisfied the following condition: all tracks $\vec{T}$ in $\hat{\Omega}$ direction are considered either they don't cross the volume V_M or they cross always the same regions in same order no matter where they traverse the macro-band.
In this option, the number of each track is taken into account as a weighting factor for distributing tracks; each processor receives an equivalent total weight based on the calculation load. This option is still under investigation.

Once the one of these options is chose, two approaches for distributing the groups of tracks over the processors can be considered. The first approach is that one processor (master) distributes the tracks on other processors (slaves); the second approach is that every single processor generates all tracks by calling the EXCELT module and then takes its own group of tracks allocated after calling the TCHTRK module. In our calculation, we don't use the first approach because when the master processor is working to generate the tracks, the slaves remain in wait status. In addition to that, the size of the tracking file is generally very large so that the time of communication is non negligible. With the second approach we can avoid to have the processors in wait and also the communications.

3.3. Performance of the I/O operations

To generate tracks, to merge them, and to distribute them over processors, we need to do the I/O operations to access and to store the data (read/write). The speed and performance of the I/O operations depend essentially on File System (FS) and the Application Program Interface (API). Usually the tracks are stored in sequential binary file created by the EXCELT module. The sequential access in file with conventional APIs is very expensive when the size of the file is large as in our applications. In parallel calculation, this is also often the bottleneck. To eliminate such encountered problems and to improve a performance, we introduce two strategies to manage the data:

- using the files to store the data via a FS. To access the data, we use the sequential mode and MPI-IO as API;
- using a database to store the data; a high level interface integrating MPI-IO is used to access the data.

In the following, we will describe both of these strategies.

3.3.1. MPI-IO implementation

To avoid the concurrency access problems that we faced especially when we run our applications on SMP parallel machine, and to improve, in general, the performance for the read/write operations on files, we have implemented the MPI-IO for a file I/O. MPI-IO implementation, ROMIO, a part of the MPI-2 standard [6], is an interface designed specifically for portable high-performance parallel I/O. MPI-IO allows the users to specify a noncontiguous access pattern and read or write all the data with a single I/O function call. It allows also performing collective I/O requests of group of processes.
In order to achieve a good performance, we use the collective read/write functions. We also use the explicit collective pointer with offsets to locate the data directly in file, this allow us to reduce the access time.

3.3.2. Database implementation

For large problems, the size of the tracking file can reaches hundreds of Megabytes to Gigabytes. Thus, storing and managing this large amount of data is very difficult and it stresses the computer resources. We use the database to store the data as an alternative. A high level interface integrating MPI-IO is used to store the data in and to retrieve the data from the database. A database tables are used for storing: the mean table contain the name of the tracking file, application options and the dimension of the problem like the number of tracks, the spatial dimension, and number of regions... On all our applications, we use MySQL as database server [7].

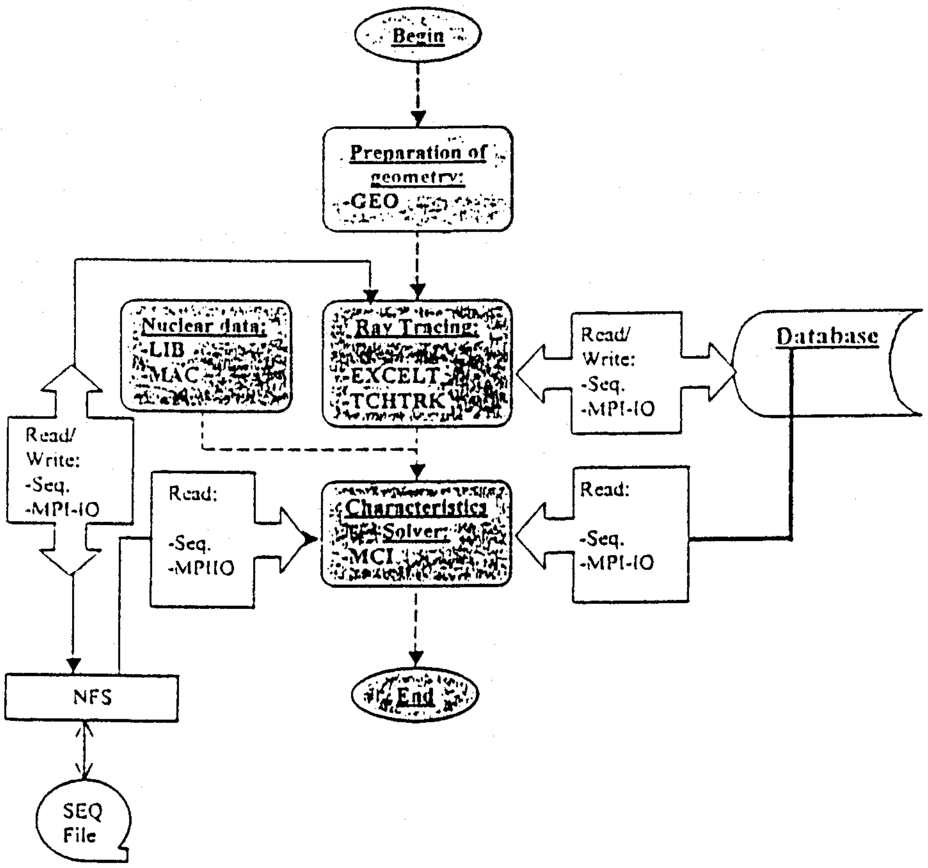


Figure 3. The transport calculation scheme using either a file or a database.

In Figure 3., we represent the transport calculation scheme using the DRAGON code with the two strategies described above. The first step is to prepare geometry of the problem, to define meshes using the GEO module. This geometry is passed to the ray tracing generator and a tracking file is created by the EXCELT module. At this step, we have to choose either to use regular files or the database to manage the data:

a) using files: in this case, every processor will have a copy of the tracking file by calling the EXCELT module and will be used by the TCHTRK module to do the merging using the TMT techniques and then each processes will take a group of tracking allocated to him according to one of the load-balancing options. At the end, all the a file containing the tracking data is created by each processes; this will be the file used by the characteristics solver MCI to solve the transport equation on each processor and then the partial results will be communicated between a processes using the message passing as we have explained above. The read/write operations use either the sequential mode or the MPI-IO interface.

b) using the database: one process will create the database and stores all the tracking data generated by the EXCELT module in the database using the database tables. After that, the merging operations will be performed using the operations on the database tables; and then using one of the load-balancing options, we create for each process a table containing its group of data and the number of processor in the communicator. Once this is done, each process will access its group of data, as client of the database (server), from the MCI module via the read/write high level interface which use the MPI-IO.

4. NUMERICAL RESULTS

Tests were done for CANDU6 adjuster rod supercell calculations in 3D geometry. Two horizontal fuel bundles (a bundle is a cluster grouped of rods containing Uranium oxide) and one vertical absorber rod at center of the assembly compose the domain. There is 1/8 symmetry and angular quadrature EQ_4 is used. Dimensions are $N_G=89$ (number of energy groups), $N_R=48$ (number of regions).
Tests were essentially performed on a two Beowulf clusters located at Center for Research on Computation and its Applications CERCA [8]:

- **Beowulf 1** is cluster of 16 nodes equipped with AMD Athlon 1.4 GHz processor and 1 GB memory connected by a fast Ethernet switched network;
- **Beowulf 2** is small SMP composed of 8 QuadXeon multiprocessors; each of these 8 nodes has 4 processors sharing a 4 GB memory. Nodes are connected with a 1 Gb/s Myrinet switch.

Our tests are done for two different densities of tracks (DT) 4.0 and 10.0 (tracks/cm^2). However, in some cases, with using the merging operations we can loose performance [3]. It is clearly shown, for especially the TMT-2 option in Figure 4, that the performance is so poor because of the increasing time of the communications over the computations. This is due to the insufficient number of the tracks in the domain. For large scale problems, where the number of tracks may be excessive, the TMT options will be more efficient. In Table I, II, and III, we present the distribution load (in Bytes) on each processor for three load-balancing options that we described above, respectively: SPLT, STRD, and ANGL. The STRD option is most balanced one, each of processors is loaded almost equally than others. With SPLT option, we see some differences of the size of load on each processor. The ANGL option shows large differences in load sizes, in addition to the limitation on the number of processors. This can be due to the interaction of the cosines directions with the cylinders main axis; in fact, for some angles, the various regions are more likely to be crossed by tracks. The limit case would be a solid angle in the same direction

as the cylinder axis, in which case the cylindrical regions would be crossed only once. The results for the MCRB option are shown in Table IV. In this option, an equivalent number of segments are distributed on each processor. Considering the load sizes on each processor, the MCRB option can be set between STRD and SPLT options. No TMT option is allowed in this case. In the following tests, we will therefore use only the STRD option.
In Figure 5, we represent the speed up curves corresponding to two different calculations: the first one is done on SMP machine (Beowulf 2) using the MPI-IO and the second one are done on the Beowulf 1. We can see that the performance is better on the Beowulf 2 with MPI-IO because of the speed of the network and the performance of the collective access to the files allowed by the MPI-IO. In the past, we had some concurrently access problems when running the applications on Beowulf 2 with the sequential mode access. All the results shown were performed with the FS option.

5. CONCLUSION

We have presented in this paper the parallel software development implemented in our 3D characteristics solver. Different techniques for distributing and merging the tracks are used. To manage the data associated with tracks, we can use either regular files or a database to store and access the data. MPI-IO was also implemented to allow collective file access. The use of the database allows us to access a set of data in same time and to manipulate data as arrays, this introduces another new approach to solve the problem.

Next step would be to subdivide the problem on sets where a minimum of nuclear data is locally collected: using a solver on groups of tracks. These mobile agents will do the most of work almost independently one of another. To attempt to solve the problem of the whole-core, we will distribute these agents on a heterogeneous grid of computers.

ACKNOWLEDGMENTS

This work has been carried out partly with the help of a grant from the Natural Science and Engineering Research Council of Canada.

REFERENCES

1. R. Thakur, W. Gropp, and E. Lusk "Data Sieving and Collective I/O in ROMIO," *Proc. Of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pp. 182-189 (1999).
2. G.Marleau, A Hebert and R. Roy, "A User's Guide for DRAGON", *Technical Report* IGE-174 Rev. 3., École Polytechnique de Montréal, December (1997). Available at http://www2.polymtl.ca/nucl
3. G. J. Wu, "Développement d'une methode des caracteristiques tridimensionelle et application aux calculs de supercellules d'un reacteur CANDU," *Ph.D. dissertation*, Ecole Polytechnique de Montreal (2001).
4. M. Dahmani, G. J. Wu, R. Roy, and J. Koclas, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON," *Proc. Of PHYSOR 2002*, October 7-10, (2002).

5. G. J. Wu, and R. Roy, "Self Collision Rebalancing Technique for the MCI Characteristics Solver," *20th Annual Conference of Canadian Nuclear Society*, Toronto (2000).
6. Message Passing Interface Forum. "MPI-2: Extensions to the Message-Passing Interface," http://www.mpi-forum.org/docs/docs.html (1997).
7. MySQL Reference Manual. http://www.mysql.com, (1999).
8. For information see: http://www.cerca.umontreal.ca

Table I. The distribution of the tracks on processors for the SPLT option (in Bytes)

	Number of processors			
Merging	2	4	8	
TMT-1		1783116	878376	793920
	3463004	1679908	904760	979960
	3764920	1991080	811704	987744
		1773860	868224	1003356
TMT-2		221288	105676	94900
	413060	191792	115492	118372
	475092	261576	94004	141196
		213536	97716	120916

Table II. The distribution of the tracks on processors for the STRD option (in Bytes)

	Number of processors			
Merging	2	4	8	
TMT-1		1806456	902884	903592
	3613224	1807756	903872	903940
	3614700	1806945	904344	903152
		1806788	903656	902640
TMT-2		222616	111152	111260
	444436	222056	110816	111484
	443716	221680	110848	110852
		221840	110760	111100

Table III. The distribution of the tracks on processors for the ANGL option (in Bytes)

	Number of angles and processors	
Merging	3	6
TMT-1	2271152 2244576 2712216	2117432 2243744 2676188 2107908 2048536 2698076
TMT-2	291028 220160 376984	262560 242168 434628 169020 199200 364824

Table IV. The distribution of the tracks on processors for the macro-band option (MCRB) (in Bytes)

Number of processors				
1	2	4	8	
11812268	5435376 6377848	2587036 2846704 3158712 3222684	1376168 1211824 1491400 1356260	1398244 1761424 1448072 1775568

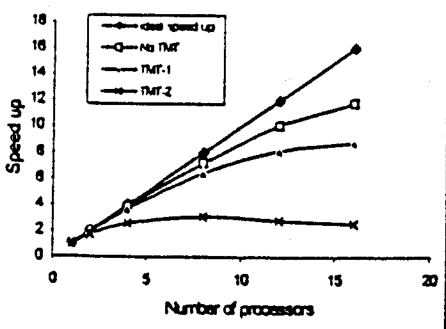


Figure 4. Speed up with and without TMT (DT=10)

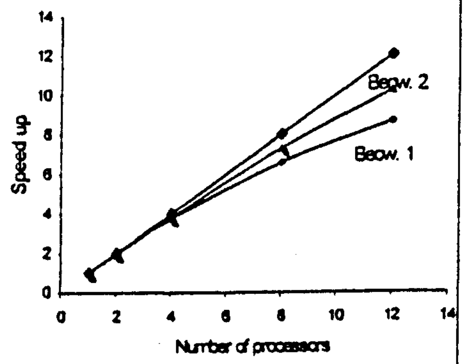


Figure 5. Speed up on two Beowulfs using files without TMT

Publication 3

Data Management in Parallel Transport Calculation

International Conference on Super Computing in Nuclear Applications

Publié en septembre 2003

DATA MANAGEMENT IN PARALLEL TRANSPORT CALCULATION

M. Dahmani, J. Chavez-Guzman, R. Roy and J. Koclas

Nuclear Engineering Institute
École Polytechnique de Montréal
P.O. Box 6079, Station Centre-Ville
Montréal H3C 3A7, CANADA
{Mohamed.Dahmani, Jaime.Chavez-Guzman, Robert.Roy, Jean.Koclas}@polymtl.ca

Abstract

Advances in parallel software development for managing the tracking data needed to solve a 3D neutron transport problems will be presented. Tracking data is needed by the characteristics solver to integrate the differential transport equation along the straight lines known as characteristics or tracks. These tracks are generated by ray tracing methods. The data related to the tracks present the major part of the tracking data which are usually stored in files. The parallelization of the solver is based on the distribution of a group of tracks on several processors. Due to the excessive number of tracks in the demanding context of 3D large-scale calculations, two different strategies for data management are implemented. The first strategy uses the regular files and the second one uses a database in parallel environment. To access to the data in files, we use either a regular sequential mode or the MPI-IO as API. Different database implementations are tested in order to achieve the optimal one for our data structure.

Introduction

In recent years, the parallel computers get faster and more accessible economically. The great advances have been made in the CPU and the communication performances. Accordingly, the scientists are increasingly use parallel computers to solve problems that require a large amount of computing effort. Most of these applications need also to perform I/O operations, such as reading data and writing the results. In general, the I/O speed is slower than the CPU and communication speed. In order for parallel computers to be fully usable in solving large-scale problems, the I/O performance must be scalable and balanced with respect to the CPU and communication performance of the system. The studies have shown that an inappropriate Application Program Interface (API) is often the cause of poor I/O performance in applications [1].

Like in many scientific applications, the nuclear engineering applications, especially the neutron transport calculation for large domains, generate and use a large amount of data. The static transport equation must be solved to obtain the flux from a source Q :

$$\hat{\Omega}\cdot\vec{\nabla}\Phi(\vec{r},\hat{\Omega},E)+\Sigma_t(\vec{r},E)\,\Phi(\vec{r},\hat{\Omega},E)=Q(\vec{r},\hat{\Omega},E) \tag{1}$$

where $\vec{r}$ is a spatial point in the domain D, $\hat{\Omega}$ is the solid angle and E is the energy.

To solve this equation, we use the 3D characteristics method (MOC) [2] [3]. Using this method, the transport equation is solved along the straight lines (tracks) using a discrete number of angles. The lines simulate the neutron paths along which the differential operator of the transport equation reduces to a total differential. Considering an isotropic source, the differential transport equation is given by:

$$\frac{d}{ds}\Phi(\vec{r}+s\hat{\Omega},\hat{\Omega})+\Sigma(\vec{r}+s\hat{\Omega})\,\Phi(\vec{r}+s\hat{\Omega},\hat{\Omega})=\frac{Q(\vec{r}+s\hat{\Omega})}{4\pi} \tag{2}$$

where s is the variable along the line. Using the flat-flux assumption, the geometrical domain is subdivided into regions where the material properties are assumed to be uniform. The outgoing flux along each segment of a track is then calculated by integrating the differential transport equation given the incoming flux [3]. In deterministic transport codes based on the characteristics method, the solver needs to access the data related to the tracks. These data, generated by the ray tracing procedure, are usually stored in files. In context of 3D large-scale calculations, the number of tracking lines can become prohibitive. Therefore, data management for these tracks is a real challenge. CPU time must be reduced, and bottleneck I/O operations must be avoided.

In this paper, we will present the last developments made to optimize the data storage and to perform the I/O operations in parallel environment. Some of these developments were introduced in the previous work [4]. An MPI-IO implementation has been used as an API, so all the process can access collectively the data in same file. In order to reduce the storage size and the time to access to the data, we use the database to store the information about the tracks.

Ray tracing: Generation of the tracks

To generate tracks, we use the ray tracing procedure in the **EXCELT** module of **DRAGON** code [5]. The angular domain is covered by a quadrature set of solid angles; by scanning the plane $\pi_{\hat{\Omega}}$ perpendicular to the select direction $\hat{\Omega}$ for the starting point $\vec{p}$, we generate the tracks for this direction (Fig. 1). The local variables are used to integrate the transport equation and then to compute the angular flux on each region [6].

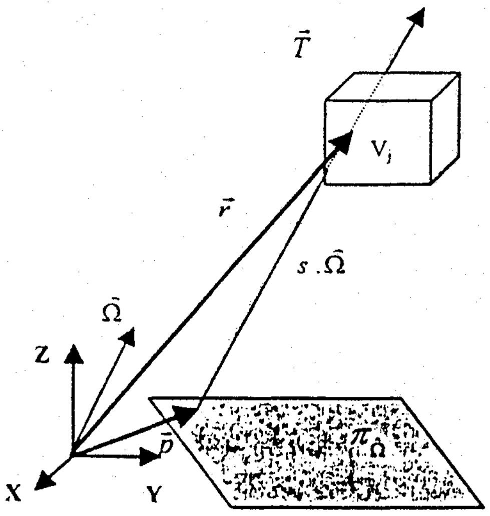


Figure 1. Generation of one track

For sequential applications, the data are stored in sequential binary file created by the EXCELT module. The tracking file looks like (Fig. 2):

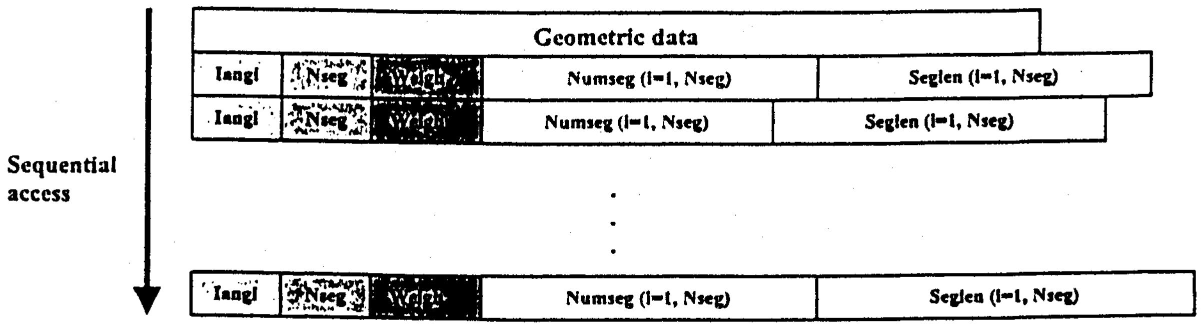


Figure 2. The sequential file

where:
Iangl: Angle number of the track,
Nseg: Number of segments of the track,
Weigh: Weight associated with the track,
Numseg: Array of surface and region numbers crossed by the track, this array contain *Nseg* elements,

Seglen: Array with spatial information, it contains *Nseg* elements.
We have to store these data for **each track**, these is the biggest part of the file. The top part of the file contains the geometric information [7].

In following sections, we will discuss the various issues accessing the tracks:

- Using files: each process will have its copy of the global tracking file. After using one of the TMT options [4] [8], we use the load balancing strategies to distribute a data. Then each process will have its own tracking file with its own data. We can use either the sequential mode or the MPI-IO as API.
- Using database: one process will creates a DB on a server node, and all other processes participate to store a data after choosing one of TMT options. Each data is tagged by the processor number. The distribution of data, using a load balancing strategies, is done into DB by updating the tables by changing the processor number corresponding to each data. Thus, a centralized DB is constructed and each process has its own data.

Parallel calculation using sequential files

For large 3D problems, the number of tracks becomes excessive. The parallelization of our code is based on distributing a set of tracks on several processors [4] [6]. The solution of the transport equation on one single track defines the finest granularity level: the smallest CPU-feasible unit.

Atomic level

At this level, we need only the data for one track with the geometric data corresponding to only the regions crossed by the track (Fig. 3).

Geometric data				
Iang	Nseg	Weigh	Numseg (I=1, Nseg)	Seglen (I=1, Nseg)

Figure 3. The tracking data at the atomic level

In addition to tracking data, we need also the local nuclear data: total cross section and the scattering cross section from the group to itself for the crossed regions. These data are used by the solver to solve the transport equation on each segment along the track.

Groups of tracks

In real problems, the number of tracks is very large (hundred of thousands to hundred of millions). In parallel environment, the problem can be treated as group of atomic level problems connected together. The global data are the sum of all the atomic level data. Therefore, the solution of the problem is the combination of the solutions of these entire atomic level problems.

To distribute a group of tracks on each process using sequential files, we distinguish two cases:

• Using distributed memory architecture: each process will have its own copy of the global tracking file. After applying one of the load balancing techniques available [4], each process creates its own tracking file which is constituted by a fraction of global data (Fig. 4).

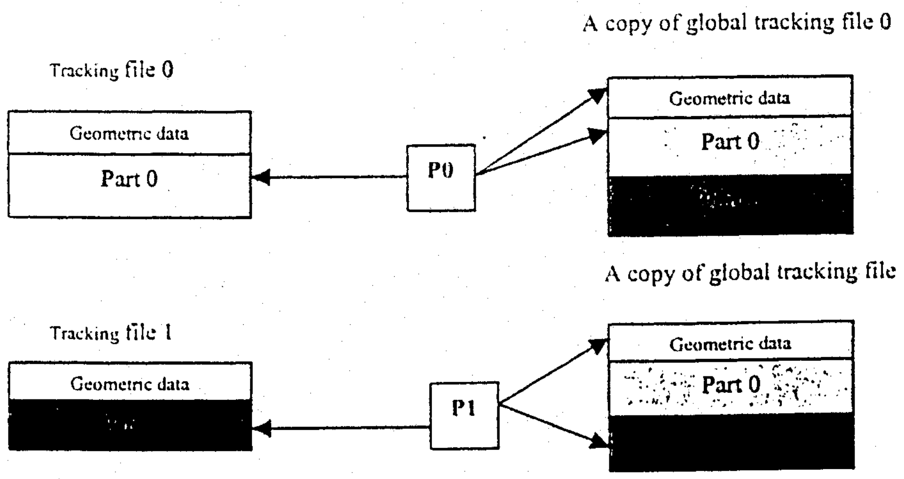


Figure 4. Case 1: Distributions of the sequential tracking file on different processes

• Using shared memory architecture: a group of processes shares the same global tracking file. Each process will take its part of data from this file. All the processes get the same copy of the geometric data (Fig. 5).

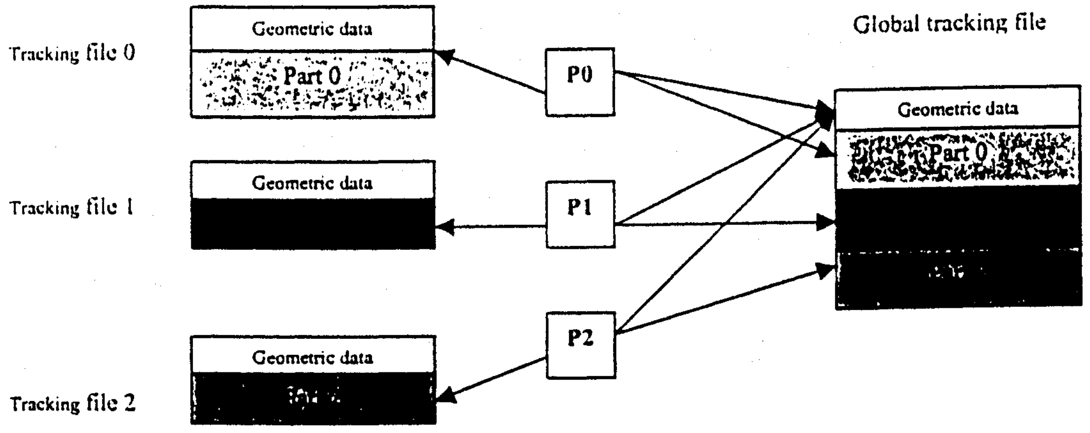


Figure 5. Case 2: Distributions of the sequential tracking file on different processes

The access to the tracking data by each solver of each process, at each inner iteration, is done by looping over the number of tracks.

The sequential access with conventional mode is very expansive when the size of file is large like in large 3D problems. In parallel calculation this is also the bottleneck; in addition to the concurrency problems encountered when several processes access to the same data simultaneously. To eliminate such problems and to improve performance, two strategies are used to manage a data. The first one is to use the parallel I/O techniques to access to the data in files; the second one is to use a database to store and to access to the data.

Parallel IO implementation

To avoid the concurrently access problems that we have faced specially when we run our applications on SMP parallel machine; and to improve, in general, the performance for the read/write operations on files, we have implemented the MPI-IO for a file I/O. MPI-IO implementation, ROMIO, a part of the MPI-2 standard [9], is an interface designed specifically for portable high-performance parallel I/O. MPI-IO allows the users to specify a noncontiguous access pattern and read or write all the data with a single I/O function call. It allows also performing collective I/O requests of group of processes. The speed and performance of the I/O operations depend essentially on File System (FS) and the Application Program Interface (API).

In order to achieve a good performance, we use the collective operations: the processes synchronize their access to the file to read or write. We also use the explicit collective pointer with offsets to locate the data directly in the file, this collective operation allows us to reduce the access time comparing to the sequential access which requires to read all the previous stored data (Fig. 6). We don't need to use the file view or the derived data because the data are contiguously stored in the file.

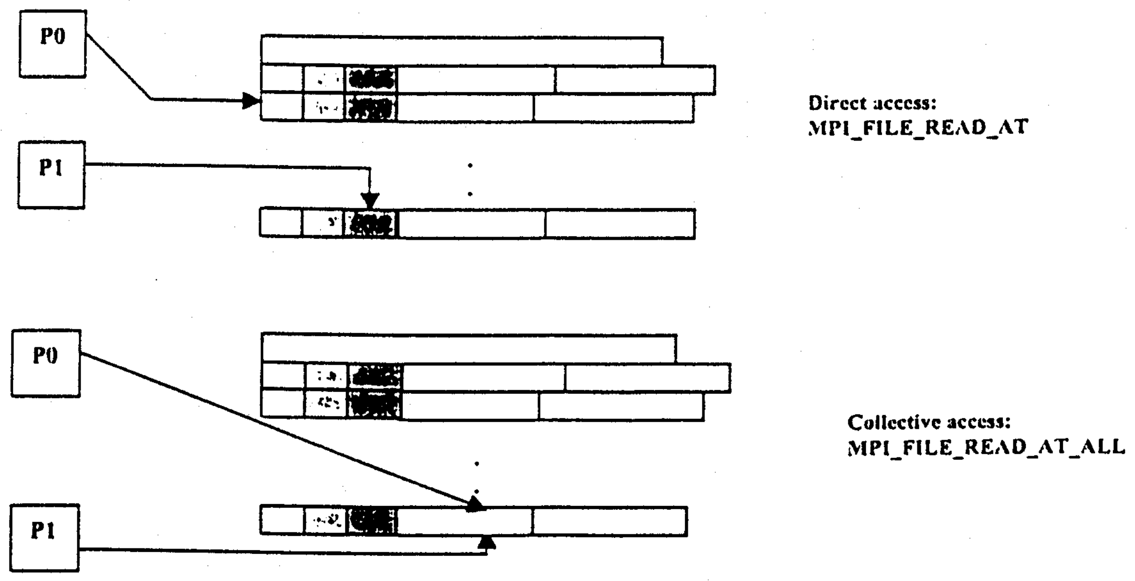


Figure 6. Direct and collective access using MPI-IO with two processes

The advantages of this implementation are: elimination of the concurrency problems, performing the collective access of a group of processes, access to noncontiguous data and it can support different file systems (NFS, PVFS ...). However, there are some issues: the use of MPI-IO increases the overheads due to the communications, this affect the global performance. This is explained by the immaturity of the MPI-IO implementations in term of performance.

For large 3D applications, the size of the tracking files can reaches hundreds of Megabytes to hundreds of Gigabytes. Thus, storing and managing this large amount of data is a difficult task which needs a lot of computing resources. In addition to the memory and the I/O bounds due to the manipulation of these large files, these affect also the scalability: the variation of performance when we increase the size of problem and/or the size of machine (number of processors). One solution of this problem is to use a database (DB) to store data in stead of files.

Database implementation

Our choice to use a database was imposed by our needs in term of storing and managing the tracking data for large-scale 3D problems in parallel environment, and the solutions offered by a database approach. We have been especially attracted by these advantages:

- Multiple users (process) access
- Concurrently access to a shared data
- Centralized DB manage all changes occurred and preserve the consistency of a data
- The speed and the ease to store and to retrieve the data

However, the use of a DB itself does not guarantee satisfactory results in terms of storage size and access time to the data. The design of the DB adapted with a given data structure has been optimized. In our application, we have tested different ways to design the DB in order to achieve the more optimized one. In the next paragraph, we will explain the different methods used to organize the data.

Data organization and distribution

We use the client/server mode: database is created on a server (master) node and all other nodes are considered as clients. In a DB approach, the data are stored in tables, each table has a name and more often related to other tables. Once the DB is created, all the processes running on different nodes participate to fill a data into a DB in order to avoid the idle time. The data stored are labeled by the number of processor (*Me*) associated with the process. To organize the data, we have introduced three approaches corresponding to three different implementations. For all three implementations, the geometric data are stored in separate tables: the principal table, called *Datageom* contains the general geometric information such as number of regions, number of directions .... The *Datageom* table is connected to seven tables containing other arrays (Fig. 7).

Implementation 1

The tracking data are stored in two tables: the first table (*Datatrk*), contains *Iang*, *Nseg*, *Weigh* and the number of processor (*Me*). Each *Datatrk* table is connected, via the track number (*trknum*), to another table containing two arrays of *Nseg* dimension: *Numseg* and *Seglen*, this table is called *Arrtrk* (Fig. 7).

Implementation 2

In this implementation, the tables containing the tracking data in implementation 1 (*Datatrk* and *Arrtrk*) are merged together into one table with the processor number (Fig. 7).

Implementation 3 (Hybrid)

This implementation is somewhere a combination of the two previous implementations. Tracks having the same number of segments (*Nseg*) are stored in the same table. We have then to create the tables: *Datatrk_3* to *Datatrk_Mxseg*, where *3* and *Mxseg* are the minimum and maximum number of segments respectively (Fig. 7).

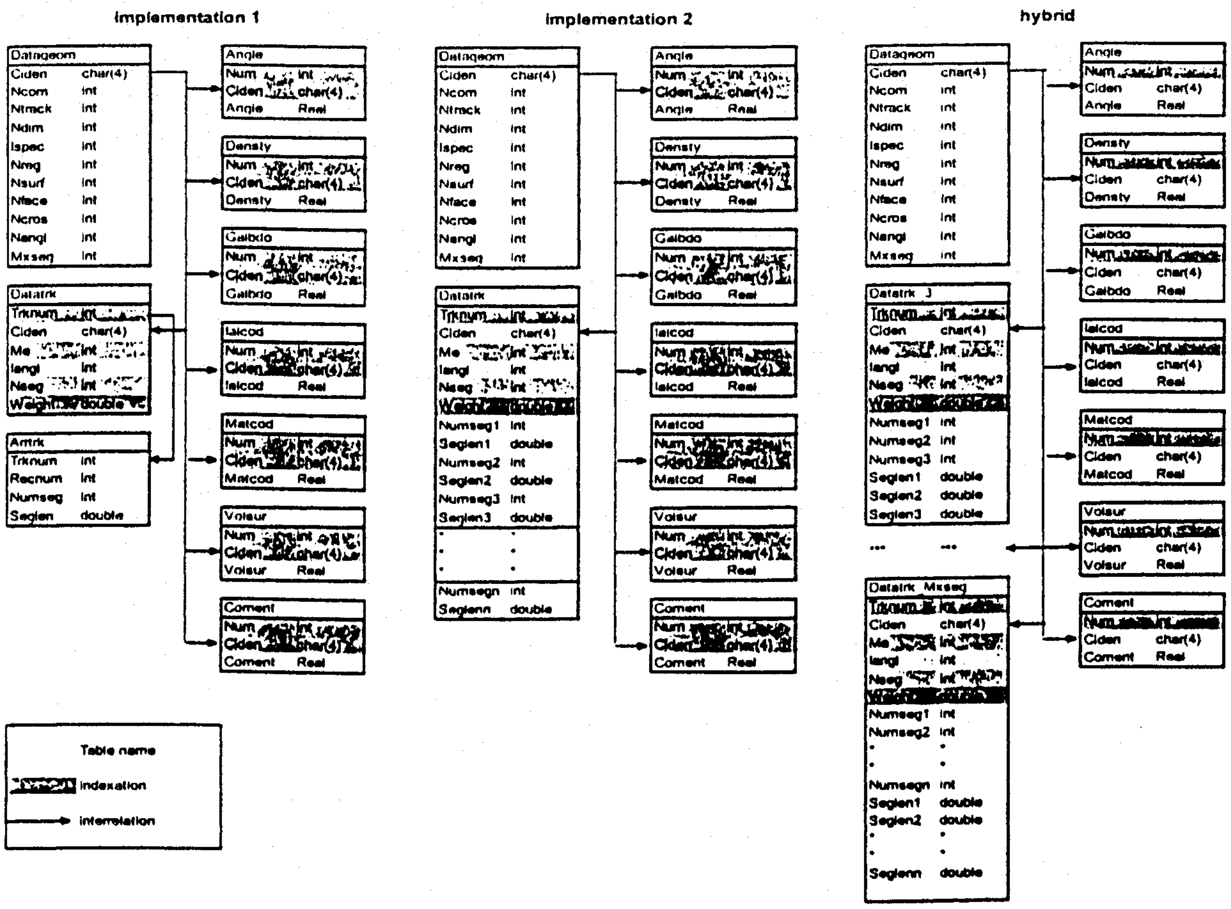


Figure 7. Three database implementations

Comparison between the three implementations

In the implementation 1, as we can see in Table 1, the time to write into a database is relatively large because for each track data, we have to create and write in two tables. In the implementation 2, the time to write is considerably reduced because the two tables are already merged. However, it takes more disk space than in other implementations: to construct the track table (*Datatrk*), we have to allocate a table with a size at least equal to the maximum number of segments. This generates a lot of unused void spaces corresponding to the data with small number of segments. In the hybrid implementation, the size is largely reduced because we have eliminated these spaces by storing the tracks with the same number of segments together. The time to write is almost the same as in the implementation 2 (Table 1). The hybrid implementation seems the optimal one for our data structure.

	Implementation 1	Implementation 2	Hybrid
Time to write (s)	500	120	115
Size (Kb)	50578	105425	29309

Table 1. Time to write into DB and the size of DB

Distribution of the data

The distribution of a data is done into a DB using the load balancing strategies available in our code [4]. It consists to update the processor number (Me) corresponding to a data which are stored in arbitrary order by the processes (Table 2). In this way, we let the possibility to more than one instance of the same process to participate in filling a data into DB and to permit the use of the heterogeneous machines with different processor speeds.

Me	Iangl	Nseg	Weigh	Numseg	Seglen
2	1	3	•	•	•
2	1	3	•	•	•
2	1	3	•	•	•
0	1	3	•	•	•
0	1	3	•	•	•
0	1	3	•	•	•
1	1	3	•	•	•
1	1	3	•	•	•
1	1	3	•	•	•
3	1	3	•	•	•
3	1	3	•	•	•
3	1	3	•	•	•

After using STRD $\longrightarrow$

Me	Iangl	Nseg	Weigh	Numseg	Seglen
0	1	3	•	•	•
0	1	3	•	•	•
0	1	3	•	•	•
1	1	3	•	•	•
1	1	3	•	•	•
1	1	3	•	•	•
2	1	3	•	•	•
2	1	3	•	•	•
2	1	3	•	•	•
3	1	3	•	•	•
3	1	3	•	•	•
3	1	3	•	•	•

Table 2. Example of storing and distributing the data using four processors

Access to the data

To access to the data, two approaches are considered:

• The data are put in temporary tables in each local node memory. These tables are managed by the same centralized system where each process has its own data. In this approach, for large amount of data, the local memory may become insufficient.

• The processes access their own data directly in database via a network. The access time in this approach is slightly greater than in the first approach but it decreases almost linearly with increasing number of processors in the two cases (Fig. 8).

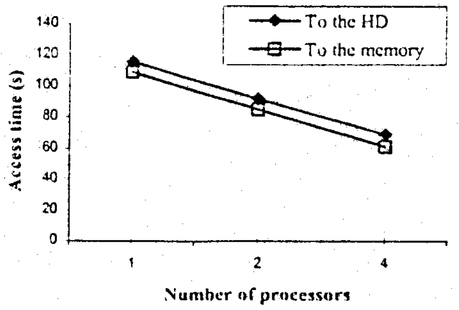


Figure 8. Access time to memory and to hard disk

A choice of managing system

A lot of choices are possible; we separate these choices on two categories:
- Open-source solutions
- Other possibilities: ORACLE, distributed database ...

Since our code is open-source, we have opted for the first choice. Therefore, we use the MySQL managing system as a database server [10]. This system uses the SQL (Structured Query Language) language which is the industrial standard created to manipulate a data. The MySQL is the most popular database on Linux; it is also available on all operating systems. The main advantage of this DB is its speed and a minimum amount of resources needed. With MySQL, the manipulation of the data is easy and transparent. Since our code sources are written in FORTRAN, we implemented an interface to connect the MySQL API (written in C language) to the code.

Distributed transport calculation scheme

In this paragraph, we will present the general transport calculation scheme in parallel environment using files or database approach.

After the preparation of geometry (module **GEO**), the tracks are generated by the ray tracing method with **EXCELT** module. The tracking data are then stored in file or into a database.

Once the ray tracing procedure and the storage of the data are finished, the next step is to solve the differential transport equation and to compute the flux. To do so, each solver of each process needs to access to its own tracking data at each inner iteration:
- Using files: each solver access to its own tracking file. The tracking sweep is done by looping on number of tracks at each inner iteration. These operations are very expansive for large files.

- Using database: each solver access directly to its own data using the processor number to identify a data. The data can retrieved as blocks or arrays and it can be stored in temporary vectors to reduce the frequency of access.

The solver needs also to access to the nuclear data: cross sections per region and group. At the end of each inner iteration, the processes broadcast their results to other processes (One-to-All communications) and use the reductive operations to construct the global flux solution per region and group.

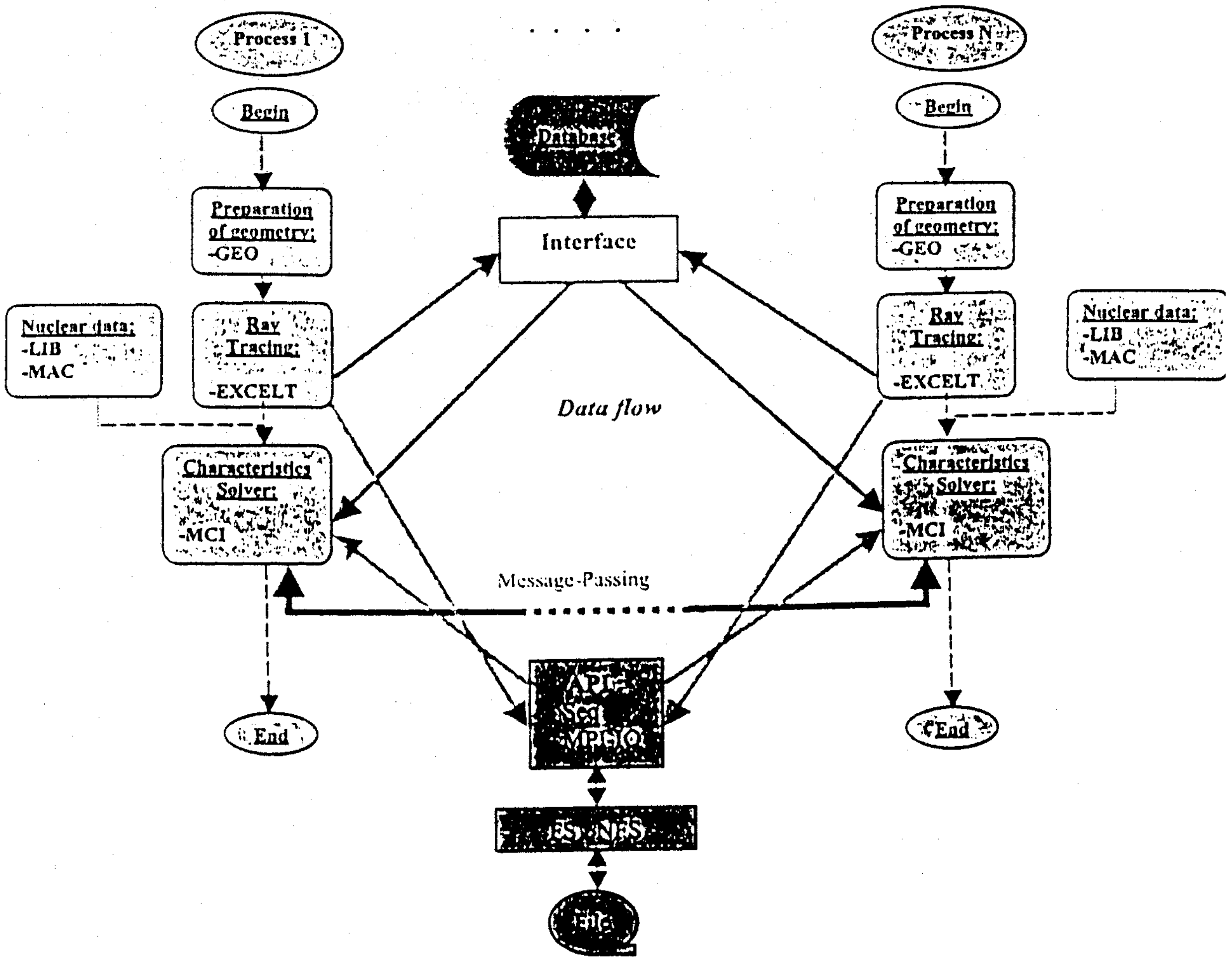


Figure 9. Parallel transport calculation scheme

Numerical results

Tests were done for CANDU6 absorber rod supercell calculations in 3D geometry. Two horizontal fuel bundles and one vertical absorber rod at center of the assembly compose the domain. We use the EQ_4 angular quadrature and 10 lines/cm^2 tracking density on the perpendicular plane. Tests were essentially performed on a two Beowulf clusters:

- **Beowulf 1** is cluster of 16 diskless nodes equipped with AMD Athlon 1.4 GHz processor and 1 GB memory connected by a fast Ethernet switched network;
- **Beowulf 2** is an SMP composed of 8 QuadXeon multiprocessors; each of these 8 nodes has 4 processors sharing a 4 GB memory. Nodes are connected with a 1 Gb/s Myrinet switch.

In Figure 10, we represent the time taken by the processes to read the tracking file using sequential mode and MPI-IO. The time is larger for the sequential mode because of the sequential access; we have to read all the previous data to get the one we want. However, when increasing the number of processors, the time decreases considerably comparing to MPI-IO because the amount of data decreases after the distribution of data on several processes and no communications are involved. For the MPI-IO, the communication overheads increase with the number of processors so the time curve gets flat. Hence, the performance for the sequential mode is better as we can see in Figure 11 which presents the speedup curves.

Figure 12 shows the time using sequential mode and MPI-IO for different number of regions. For the sequential mode, the time decreases by a factor of about 4 corresponding to the factor increasing the number of tracks when increasing the number of regions from 46 to 386 (Table 3). For MPI-IO, we see small changes in time when we increase the number of regions.

Number of tracks	69801	362100	275262	1101048
Time to read using sequential mode (s)	0.285	1.5261	1.0721	4.641
Time to read using MPI-IO (s)	0.0761	0.1362	0.126	0.147

Table 3. Time to read using sequential and MPI-IO for different number of tracks

Figure 13 represents the speedup curves corresponding to two different calculations; the first one is done on Beowulf 1 and the second one on Beowulf 2. The speedup is much better on Beowulf 2 because, not only the speed of network is higher, but also all processes running on processors belonging to the same node share the same tracking file so the communication overheads decrease. In the past, we had some concurrently access problems when running the applications on Beowulf 2 with the sequential mode access, these problems have disappeared with the use of MPI-IO. We have tested also different MPI-IO routines to read tracking file: MPI_FILE_READ_AT, MPI_FILE_READ_AT_ALL, and the nonblocking read MPI_FILE_IREAD_AT. The results are presented in Figure 14. The read operation using a collective pointer, on distributed memory architecture, takes more time because the processes read in different files. In this case, the synchronization of the processes contributes to increase uselessly the overheads due to communications. With the two others routines, the time to read is almost the same.

Figure 15 shows the time spent to store data using a database for different number of regions. The time to write into DB is considerably reduced by using several processors. Comparing the three options, the time spent to read a data using DB is between the sequential and MPI-IO (Figure 16). We have also considered the scalability of DB reading with respect of the number of tracks. In Table 4, we report the results using a single processor.

Number of tracks	(1) 275262	(2) 1101048	(3) 2207552	(4) 4422956
Time to read using DB (s)	0.54	2.16	5.78	8.26

Table 4. Time to read using DB for different number of tracks

The time to read, when using one processor, varies almost linearly with the number of tracks. When increasing the number of processors, the time to read decreases until a certain limit where the time values remain almost constant. At this point, we begin to loose performance because the access operations become dominated by the processes queries to a DB. The access by several processes to a same table into a DB simultaneously contributes also to cause these delays. The delay time is directly related to the centralized managing system and the network speed. The amount of data affects slightly the performance of reading (Fig. 17). The knowledge of that point where the performance begin to decrease considerably is important to prevent the number of processes which will share the same server in order to achieve the optimal performance for the distributed 3D large-scale transport calculations.

Summary

1- Sequential mode: the time to read tracking file is relatively large, a sequential access in file increase the time to access to a data. However, in parallel environment, the time to read decreases considerably with increasing number of processors because no communications between a process are involved. The time to read is proportional to the number of tracks.

2- MPI-IO: the use of a collective or individual file pointer to locate a data reduces the time to access to the data in file comparing to the sequential access. However, for relatively small amount of a data, communication and synchronization overheads may dominate access time. This clearly affects the global performance. Therefore, in this case, it is not necessary to use MPI-IO. The use of a collective access approach permits to eliminate the concurrency problems.

In general, the use of files generates these major problems:

- Size of files: due to the duplicate of the tracking file on all processors, especially on distributed memory architecture, the size of the file can exceed the capacity of the machine or slow down the execution of the process.
- Frequency of access to a file: in addition to the time spent to access to the data, the frequent access by the solver increases the global computing time.
- Scalability with I/O bounds: how the system handle the I/O operations when we increase the size of problem (increasing a data) or a number of processors. In this context, the test with other file systems like PVFS will be useful.

3- Database approach: using a client/server mode, data are stored in a centralized system. There is no need to duplicate data on all processors. This contributes to reduce the storage size. The participation of all processes to store data permits to considerably reduce time to write and the idle time. The database approach handles the muti-process access to a data. This eliminates the concurrency problems. The use of tables makes a data access easier and transparent. The identification of the data by the processor number facilitates the distribution of a data among processes. The manipulation of blocks of a data as arrays reduces the frequency to access to a data by a solver. The database approach permit also the use of some algebraic operations on data like sum, logarithm, exponential ... However, the main issue of the database implementation in parallel environment is the scalability at great costs: for 3D large-scale problems, we have to increase the number of servers.

Conclusion

We have presented here the last software developments made in the parallel version of DRAGON code to manage a data. After the ray tracing procedure, the data are stored either in files or in database. The use of MPI-IO to perform the collective access to the data in files allows us to eliminate concurrency problems encountered when running on an SMP machine. For 3D large-scale calculations, the use of a database to manage data is more suitable because the storage size and the centralized management system which handle the parallel access to the distributed data by different processes. The database also allows us to reduce the access time and to manipulate data as arrays which reduces the frequency of access by the solver. The use of a database introduces a new approach to solve the distributed large-scale transport problem on heterogeneous system of computers where all the processes or a part of them will be associated to the database.

References

1. R. Thakur, W. Gropp, and E. Lusk "Data Sieving and Collective I/O in ROMIO," *Proc. Of the 7th Symposium on the Frontiers of Massively Parallel Computation*, pp. 182-189 (1999).
2. J. R. Askew, "A characteristics Formulation of the Neutron Transport Equation in Complicated Geometries" *Report AEEW-M 1108*, United Kingdom Atomic Energy Establishment, Winfrith, (1972).
3. G. J. Wu and R. Roy, "A New Characteristics Algorithm for 3D Transport Calculations", *Ann. Nucl. Energy*, 30, 1-16, (2003).
4. M. Dahmani, R. Roy, and J. Koclas, "Parallel Distribution of Tracking for 3D Neutron Transport Calculation", *M&C 2003*, April 6-11, Gatlinburg, TN, (2003)
5. G. Marleau, A Hebert and R. Roy, "A User's Guide for DRAGON", *Technical Report* IGE-174 Rev.3., École Polytechnique de Montréal, December (1997). Available at: http://www2.polymtl.ca/nucl
6. M. Dahmani, G. J. Wu, R. Roy, and J. Koclas, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON," *Proc. Of PHYSOR 2002*, October 7-10, (2002).
7. R. Roy, G. Marleau, J. Tajmouati, and D. Rozon, "Modeling of CANDU Reactivity Control Devices with the Lattice Code DRAGON," *Ann. Nucl. Energy*, 21, (1994).
8. G. J. Wu and R. Roy, "Acceleration Techniques for Trajectory-Based Deterministic 3D Transport Solvers", *Ann. Nucl. Energy*, 30, 567-583, (2003).
9. Message Passing Interface Forum. "MPI-2: Extensions to the Message-Passing Interface," http://www.mpi-forum.org/docs/docs.html (1997)
10. MySQL Reference Manual. http://www.mysql.com, (1999).

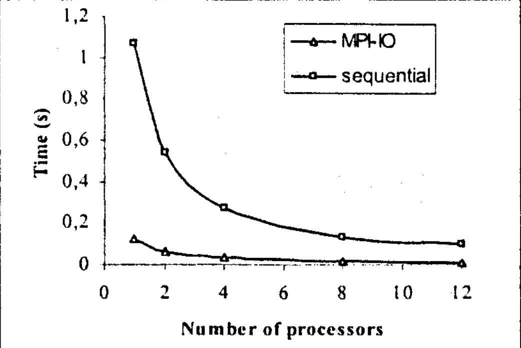


Figure 10. Time to read using sequential mode and MPI-IO

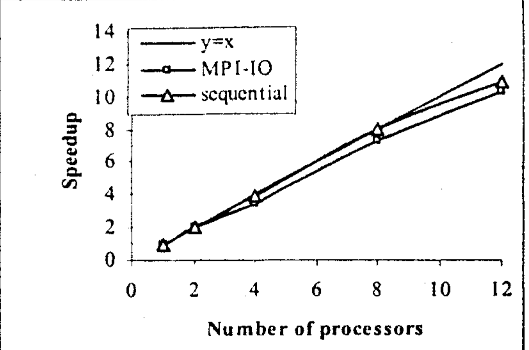


Figure 11. Speedup using sequential mode and MPI-IO

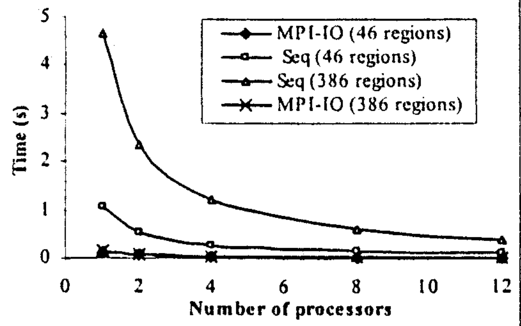


Figure 12. Time to read using sequential and MPI-IO with different number of regions

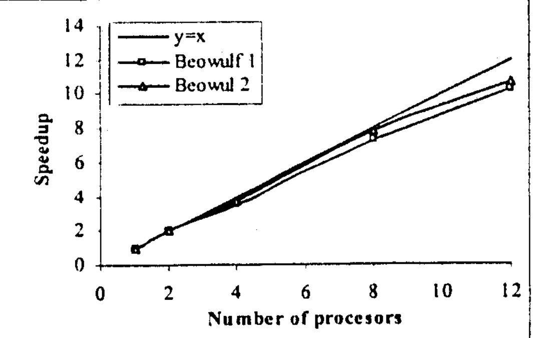


Figure 13. Speedup on two different archetectures

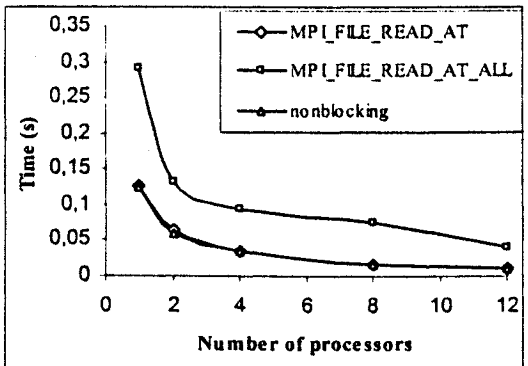


Figure 14. Time to read using three different MPI-IO subroutines

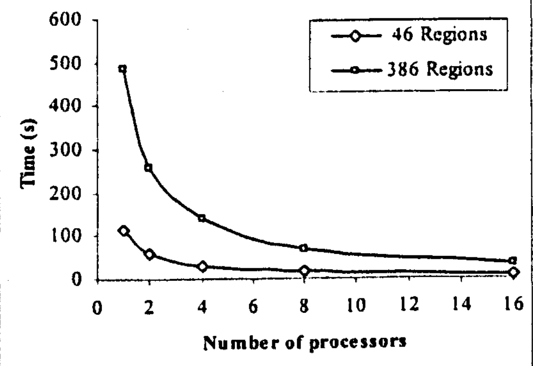


Figure 15. Time to write into DB for different number of regions

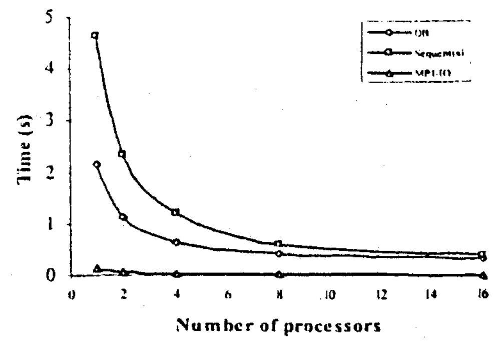


Figure 16. Time to read for the three options

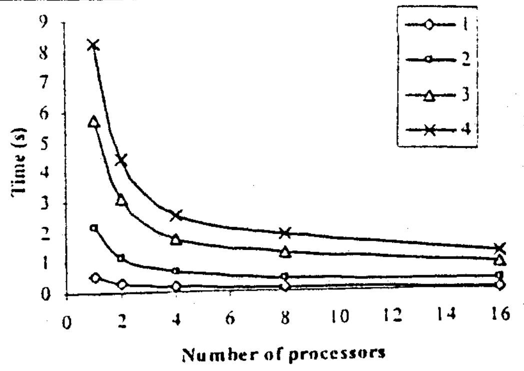


Figure 17. Time to read using DB for different number of tracks

Publication 4

New Computational methodology for Large 3D Neutron Transport Problems

PHYSOR-2004: The Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments

Publié en avril 2004

PHYSOR 2004 -The Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments
Chicago, Illinois, April 25-29, on CD-ROM, American Nuclear Society, Lagrange Park, IL. (2004)

New Computational Methodology for Large 3D Neutron Transport Problems

M. Dahmani*, R. Roy and J. Koclas
Nuclear Engineering Institute, École Polytechnique de Montréal, P.O.Box 6079, Station CV, Montréal, Québec, Canada H3C 3A7

We present a new computational methodology, based on 3D characteristics method, dedicated to solve very large 3D problems without spatial homogenization. In order to eliminate the input/output problems occurring when solving these large problems, we set up a new computing scheme that requires more CPU resources than the usual one, based on sweeps over large tracking files. The huge capacity of storage needed in some problems and the related I/O queries needed by the characteristics solver are replaced by on-the-fly recalculation of tracks at each iteration step. Using this technique, large 3D problems are no longer I/O-bound, and distributed CPU resources can be efficiently used.

KEYWORDS: Neutron transport, characteristics method, iterative linear solver, large-scale problems.

1. Introduction

In the last three decades, it has been practically observed that the CPU power almost doubled every 18 months. This is known as Moore's law. [1] However, the I/O speed increased at slower pace than the CPU speed and the capacity of storage does not follow linearly the CPU speed. According to these computational considerations and taking into account the limitations in term of I/O and memory bounds that are faced when trying to solve large 3D problems; we decided to investigate a computational methodology that requires more CPU power and less storage resources.

This paper aims to explain the new 3D characteristics technique dedicated to solve very large 3D problems (a part or a whole core) without spatial homogenization. In order to eliminate the input/output problems occurring when solving these large problems, we define a new computing scheme that requires more CPU resources than the usual one, based on sweeps over large tracking files. The huge capacity of storage needed in some problems and the related I/O queries needed by the characteristics solver are replaced by on-the-fly recalculation of tracks at each iteration step. Using this technique, large 3D problems are no longer I/O-bound, and distributed CPU resources can be efficiently used.

We begin by describing briefly, in section 2, different computational methodologies used in the DRAGON [2] code to solve a 3D neutron transport problems. In section 3 we present the new computational methodology that we have developed in order to solve the 3D large problems. In the last section, numerical results are shown. The validation of the new solver and the comparison between the two computational methodologies based on characteristics method are presented.

* Corresponding author: Fax: 514-340-4192, Email: mohamed.dahmani@polymtl.ca

2. Transport Calculation Schemes

2.1 Collision Probability Method

The usual deterministic method used in DRAGON for solving the 3D neutron transport equation in supercell geometry is the collision probability (CP) technique. In this context, the discretization of the spatial domain is done by using the ray tracing method [2] to construct the CP matrices (CPM). Two different methodologies can be used. In the following, we will describe these two different ways to perform the transport calculation and discuss their limitations for treating the large 3D problems.

2.1.1 EXCELT-ASM Scheme

The integration lines, required to cover the 3D Cartesian domain, are generated by the *EXCELT* module using the ray tracing method and then stored in a sequential binary file. The tracking file (TF) size increases with increasing number of regions, number of angles and density of tracks. The *ASM* module accesses then to the TF in order to compute the collision, escape and transmission probabilities for each energy group (corresponding to region-to-region, region-to-surface and surface-to-surface respectively). These probability matrices are then algebraically reduced by eliminating the currents from the multigroup system (Fig.1). The module *FLU* is then used to compute the solution to an eigenvalue problem corresponding to a set of group-dependent system matrices. In this representation, we need to store both the tracking file and the collision probability matrices.

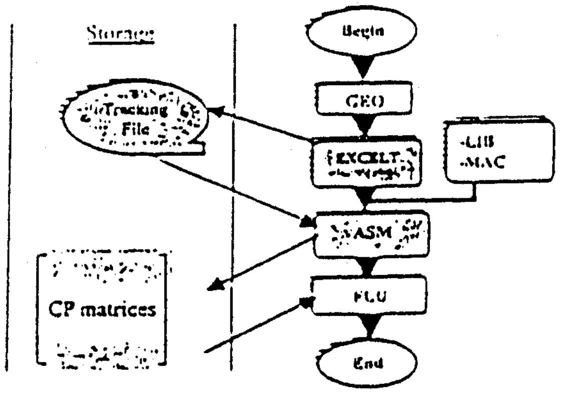


Fig. 1 EXCELT-ASM computational scheme

2.1.2 EXCELL Scheme

The ray tracing procedure and the evaluation of the CP matrices is done in the same module *EXCELL*. The tracking file does not have to be stored. This method is faster than the previous one but it still needs to store the huge CP matrices [3] (Fig. 2).

In CP method, the storage requirements increase as the square of the number of regions. There is no more limitation on the size of the TF, and thus the problem is no longer I/O-bound; however, the memory resources could be easily exceeded, so that the problem is now memory-bound.

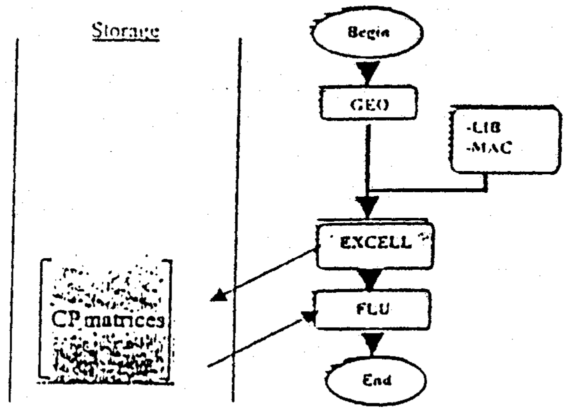


Fig. 2 EXCELL computational scheme

2.2 Characteristics Method

The 3D characteristics method was developed as an alternative for solving large problems. The long-characteristics MOC is based on solving the differential form of the transport equation by following the characteristics of the system (tracking lines) to simulate neutron paths [4]. The scalar flux is constructed by collecting all mean angular fluxes in terms of the entering angular flux and the source of the region. Unlike the CP method, the MOC permits the elimination of the CP matrices, which have the dimensionality of the size of the square of the number of regions. We have shown that such linear MOC solvers can give solutions equivalent with the quadratic CP ones. [5]

2.2.1 EXCELT-MCI Scheme

The same ray tracing method than in CP in EXCELT module can be used to create the tracking file TF. The tracking sweep is done by Looping on number of tracks at each inner iteration. A 3D MOC sequential solver has been implemented as the *MCI* module in DRAGON code. At each loop, the solver needs to access the TF to read tracking data (Fig.3). These I/O operations can be very expensive for large files, and the only possible alternative is to parallelize the I/O operations. [6]

Further, because there are no linear algebra treatments involving the matrices, the memory needed to perform 3D calculation is greatly reduced. However, the major problem that is encountered for large problems, in addition to the fact that the MCI solution can converge slowly, is the lack of space to store the tracking data [7]. In some cases, we have to restrict the number of tracks (less angles or smaller density of tracks) and the transport solution may not have the required accurate.

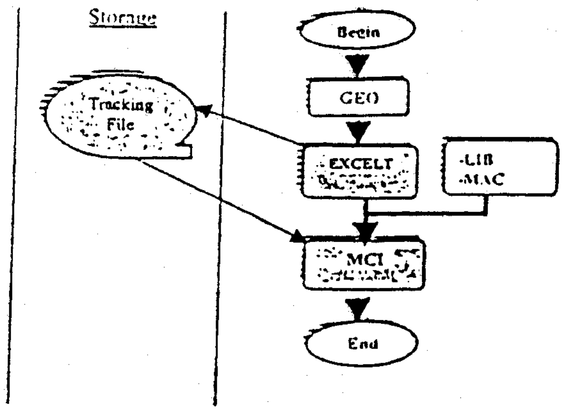


Fig. 3 EXCELT-MCI computational scheme

2.2.1.1 Parallelization

At the solver level, the smallest operation possible is composed by an I/O operation to get one tracking data from the TF and a CPU operation on this single track to compute the angular flux. The global computational operations are the combination of these atomic operations.
To reduce the computation time, the parallelization of the solver was performed by distributing the tracks on several processors. A broadcast operation is activated to communicate the local solution between processes [5, 8]. The parallelization of the I/O operations was also implemented. [6]

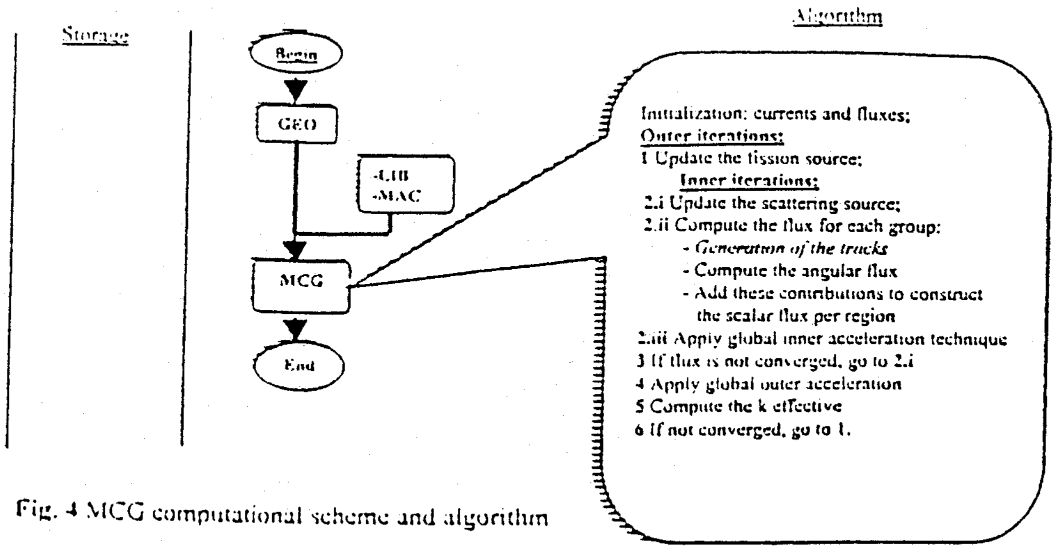
3. A New Methodology

The idea is to take advantage of the best features of the above methods in order to be able to solve large 3D problems. Intuitively, the ideal choice would be a linear characteristics method without I/O bounds. The new MCG technique is an alternative to the MCI technique. It is based on the elimination of the use of the tracking file to store the data. The MCG inner solver is based on this process:

1- Generate a track;
2- Calculate the angular flux on this track;
3- Add this contribution to compute the scalar flux;
4- Destroy the track and go to 1.

This new solver was fully implemented and results were compared with the usual MCI calculations (see Fig. 4).

The solver generates tracks, as needed, and then the angular flux is computed along the track to contribute in the scalar flux calculation. In this way, the storage of tracks is not necessary. Like in MCI, in order to reach a convergence of the solution, more iterations than in CP method are required. This is due to the current convergence and to the flux initialization (flat flux).



```
Initialization: currents and fluxes;
Outer iterations:
1 Update the fission source:
      Inner iterations:
 2.i Update the scattering source;
 2.ii Compute the flux for each group:
      - Generation of the tracks
      - Compute the angular flux
      - Add these contributions to construct
        the scalar flux per region
 2.iii Apply global inner acceleration technique
3 If flux is not converged, go to 2.i
4 Apply global outer acceleration
5 Compute the k effective
6 If not converged, go to 1.
```

Fig. 4 MCG computational scheme and algorithm

3.2 Summary

In Table 1 we summarize the characteristics of the four computational methodologies described above. For the large-scale problems, each of these methodologies has computational limitations. For the new methodology that we have developed, the only hypothetical limitation would be the CPU bounds.

Table 1 The summary of the four computational schemes

Computational scheme	Method	TF	Storage	Parallelization	Limitations
EXCELT-ASM	CP	yes	TF CPM	no	Memory and I/O bounds
EXCELL	CP	no	CPM	no	Memory bounds
EXCELT-MCI	MOC	yes	TF	Message Passing : PVM, MPI, MPI-IO	Memory and I/O bounds
MCG	MOC	no	no	no	CPU bounds!

4. Numerical Results

We will present some results used to validate the MCG module. The tests were done for CANDU-6 supercells with different configurations as described in [9]. Transport equation is then solved by critical buckling search with order leakage treatment B_1. For standard tests, we use an EQ_4 angular quadrature, an isotropic reflection option, and a 89 energy groups.

The results given by the module *EXCELL* with HELI normalization option (which gives almost the same results than the reference used in [8]) are considered as references, we use then the relative error for the calculated value of the incremental cross-sections. In Table 2-3, we can see that the results given by the *MCG* module are very close to those given by *EXCELT-ASM* (based on CP method). The *MCG* gives better results than *EXCELT-MCI* because it is based on the same tracking procedure than in *EXCELL* and use the same variable precision. In Table 4, we report the total CPU time used for these two simulations. The MCG takes more CPU time than all other methodologies. The ratio between the total CPU time spent by *EXCELT-ASM* and *EXCELL* is 3.6, it is only 1.9 between *MCG* and *EXCELT-MCI* for these cases.

Table 2 Results for BCAINT Case

Incremental cross-sections	MCG Methodology	MCG Error (%)	EXCELT-MCI Error (%)	EXCELT-ASM Error (%)
$\Delta\Sigma_1^1$	7.1641925E-04	4.33E-03	4.16E-03	0
$\Delta\Sigma_2^1$	1.3236834E-05	5.94E-02	5.10E-02	9.6842E-03
$\Delta\Sigma_1^{1-1}$	6.3281296E-04	2.45E-03	4.95E-03	-1.0572E-03
$\Delta\Sigma_1^{1-2}$	1.5222793E-07	2.20E-02	4.26E-02	0.019696
$\Delta\Sigma_1^2$	3.3613721E-04	-6.30E-02	-7.98E-02	0.017723
$\Delta\Sigma_2^2$	2.6223146E-04	-1.11E-02	1.07E-02	0.022467
$\Delta\Sigma_1^{2-1}$	7.0351729E-05	-9.72E-03	2.20E-02	0.024411
$\Delta\Sigma_1^{2-2}$	7.3685341E-05	-3.26E-01	-4.38E-01	-0.033187

Table 3 Results for BCAOUT Case

Incremental cross-sections	MCG Methodology	MCG Error (%)	EXCELT-MCI Error (%)	EXCELT-ASM Error (%)
$\Delta\Sigma_1^1$	5.8347536E-04	9.40E-06	-5.10E-05	0.
$\Delta\Sigma_2^1$	1.0809423E-05	-0.00011	0.00033	9.69E-05
$\Delta\Sigma_1^{1-1}$	5.1470812E-04	1.28E-05	-3.11E-05	-6.2E-06
$\Delta\Sigma_1^{1-2}$	1.2468421E-07	-6.18E-05	0.00016	0.00019
$\Delta\Sigma_1^2$	2.7318534E-04	0.00039	-0.00076	0.
$\Delta\Sigma_2^2$	2.1568943E-04	3.56E-05	-8.95E-05	0.00021
$\Delta\Sigma_1^{2-1}$	5.7983421E-05	0.00024	-0.0003	0.00025
$\Delta\Sigma_1^{2-2}$	5.7289320E-05	0.00032	-0.0029	-0.0010

Table 4 The total CPU time Spent for the four methodologies

Methodology	EXCELL	EXCELT-ASM	EXCELT-MCI	MCG
Total CPU (s)	48	176	858	1682

In Fig. 5, we plot the size of the tracking file in MB versus the number of regions; we can see that the size of TF, and therefore the capacity of storage, varies linearly with the number of regions. By varying the number of regions from 46 to 2480, the disk space needed increase by a factor 34. For this test case we expect, for 4500 regions, that we will need approximately 1.3 GB of disk space. In Fig. 6, the CPU time spent by the two solvers is shown when varying the number of regions. The CPU time is almost linear for the both solvers. The MCG solver takes, obviously, more CPU time than MCI solver. Both solvers reach the convergence almost at the same number of iterations.

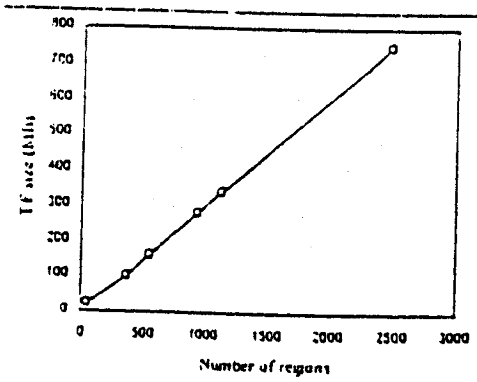


Fig. 5 Tracking file size with varying number of regions.

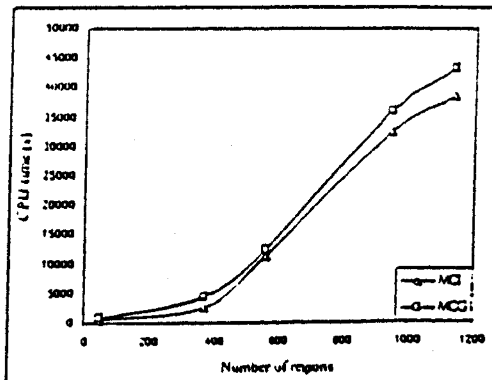


Fig. 6 CPU time for MCI and MCG solvers for different number of regions

5. Conclusion

We have presented a new technique that we have developed in order to solve the large 3D problems. This computational methodology is no longer I/O-bound, comparing to other methodologies that are both CPU and memory consuming. It was a compromise between the need of the huge storage resources and utilization of more CPU power. The computing time is greater than with MCI methodology but eliminating the tracking file reduces drastically the general storage resources needed. However, with the parallelization of the MCG solver, the use of an adequate flux/current initialization and the application of an appropriate acceleration technique, the MCG methodology is the most adapted one for solving large 3D transport problems with a reasonable computing time.

Acknowledgements

This work has been carried out partly with the help of grants from the Natural Science and Engineering Research Council of Canada.

References

1) http://www.intel.com/silicon/mooreslaw.htm
2) G. Marleau, A Hebert and R. Roy, "A User's Guide for DRAGON", Report IGE-174 Rev.3., École Polytechnique de Montréal, December (1997). (see also http://www2.polymtl.ca/nucl)
3) R. Roy, A Hebert and G. Marleau, "A Transport method for Treating Three-Dimensional Lattice of Heterogeneous Cells," *Nucl. Sci. Eng.*, **101**, pp. 227-225 (1989).
4) J. R. Askew, "A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries," Report AEEW-M 1108, Atomic Energy Establishment, Winfrith (1972).
5) G. J. Wu and R. Roy, "A New Characteristics Algorithm for 3D Transport Calculations," *Ann. Nucl. Energy*, **30**, pp. 1-16 (2003).
6) M. Dahmani, J. Chavez-Guzman, R. Roy and J. Koclas, "Data Management in Parallel Transport Calculation," Proc. Super Computing in Nuclear Applications, Paris, September 23-26, 2003.
7) M. Dahmani, G. J. Wu, R. Roy and J. Koclas, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON," Proc. PHYSOR-2002, Seoul, Korea, October 7-10, 2002.
8) M. Dahmani, R. Roy and J. Koclas, "Parallel Distribution of Tracking for 3D Neutron Transport Calculation, ANS Conf. M&C 2003, Gatlinburg, Tennessee, April 6-11, 2003.
9) R. Roy, G. Marleau, J. Tajmouati and D. Rozon, "Modeling of CANDU Reactivity Control Devices with the Lattice Code DRAGON," *Ann. Nucl. Energy*, 21, pp. 115-132 (1994).

Publication 5

3D Characteristics Method with Linearly Anisotropic Scattering

PHYSOR-2004: The Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments

Publié en avril 2004

PHYSOR 2004 - the Physics of Fuel Cycles and Advanced Nuclear Systems: Global Developments
Chicago, Illinois, April 25-29, 2004, on CD-ROM, American Nuclear Society, Lagrange Park, IL. (2004)

3D Characteristics Method with Linearly Anisotropic Scattering

M. Dahmani[1], R. Roy and J. Koclas
Nuclear Engineering Institute, École Polytechnique de Montréal
P.O.BOX 6079, Station Centre-ville, Montréal, Québec, Canada H3C 3A7

The objective of this work is to extend the characteristics method to take into account anisotropic scattering effects in 3D geometries. Only the linearly anisotropic scattering cross sections and sources treatment will be considered.
The method of characteristics is based on computing the angular flux following the tracking lines. Therefore, the implementation of anisotropic sources is relatively simpler than in collision probability method. However, the calculation of the anisotropic sources requires large amount of memory in order to store the angular flux per energy group, region, direction and track. Comparisons between the isotropic and anisotropic fluxes for different meshing and number of discretized angles will be shown in the results section.

KEYWORDS: Characteristics method, anisotropic, scattering.

1. Introduction

The recent advances in computer capability in term of memory size and processor speed, on one hand, and the development of high performance computing methods on the other hand allow investigating the boundaries of the old models in many scientific and engineering applications. Like other applications, the advanced reactor core designs require advanced computational methods in order to achieve high accuracy in reasonable computational time. To assure this accuracy, in advanced reactor core analysis methods based on transport equation solution using the deterministic transport codes, the approximations used must be reviewed to take into account of:

- Space: the 3D geometry configuration with great degree of heterogeneity;
- Energy: to enlarge by refining the discritization of the domain to treat well the resonance regions;
- Angle: to allow the use of the large number of discrete angles to eliminate in part the ray effects and to treat the anisotropic effects.

[1]Corresponding author: E-mail: mohamed.dahmani@polymtl.ca, Fax: 514-340-4192

In this paper, we present the extension of the 3D characteristics method to take into account the anisotropic effects. We will limit our analysis to the linearly anisotropic scattering cross sections and sources. The anisotropic treatment has been presented in several papers for 2D geometries using the Collision Probabilities method [1-3] and Method of Characteristics. [4,5]

Since MOC is based on computing the angular flux on tracks, the implementation of the anisotropic sources is relatively simpler than in CP. However, the calculation of the anisotropic sources requires large amount of memory in order to store the angular flux per energy group, region, direction and track. This implementation is done in the 3D Characteristics solver [6], a part of DRAGON code. [7] The comparison between the isotropic and anisotropic fluxes for different meshing and number of discretized angles will be shown in the results section.

The ramainder of this paper is organized as follows. In section 2 we derive the characteristics equations using the linearly anisotropic scattering. The discretized characteristics equations are also presented in this context. In section 3 we describe the calculation methodology and the algorithm used. Finally numerical results comparing the isotropic and anisotropic fluxes will be shown.

2. Characteristics Equation with Anisotropic Scattering

The multigroup transport equation to be solved to obtain the flux from a source Q is:

$$B^g(\vec{r},\hat{\Omega})\Phi^g(\vec{r},\hat{\Omega}) = Q^g(\vec{r},\hat{\Omega}) \tag{1}$$

where:

- $\vec{r}$ is a spatial point in the domain D;
- $\hat{\Omega}$ is the solid angle;
- g is the energy group;
- $B^g(\vec{r},\hat{\Omega}) = \hat{\Omega}\cdot\vec{\nabla} + \Sigma^g(\vec{r})$ is the transport operator in group g;
- $\Sigma^g(\vec{r})$ is the total cross section in group g;
- $Q^g(\vec{r},\hat{\Omega})$ is the anisotropic source in group g.

The source in Eq.(1) is composed of two terms, the fission and scattering sources:

$$Q^g(\vec{r},\hat{\Omega}) = F^g(\vec{r}) + S^g(\vec{r},\hat{\Omega}) \tag{2}$$

The isotropic fission source is given by:

$$F^g(\vec{r}) = \frac{\chi^g}{4\pi\, K_{\text{eff}}}\sum_{g'} \nu\,\Sigma_f^{g'}(\vec{r})\phi^{g'}(\vec{r}) \tag{3}$$

where $\phi^{g'}(\vec{r})$ is the scalar flux in group g.

Using the first order Legendre polynomial expansion on angular flux and scattering cross section, the linear anisotropic scattering source is obtained by:

$$S^g(\vec{r},\hat{\Omega}) = S_0^g(\vec{r}) + S_1^g(\vec{r},\hat{\Omega}) \tag{4}$$

where:

$$S_0^g(\vec{r}) = \frac{1}{4\pi}\sum_{g'}\Sigma_{s0}^{g\leftarrow g'}(\vec{r})\phi^{g'}(\vec{r}) \tag{5}$$

is the isotropic scattering source. The anisotropic scattering source term is given by:

$$S_1^g(\vec{r},\hat{\Omega}) = \frac{3}{4\pi}\sum_{g'}\Sigma_{s1}^{g\leftarrow g'}(\vec{r})\int_{4\pi} d^2\Omega'\ \hat{\Omega}.\hat{\Omega}'\Phi^{g'}(\vec{r},\hat{\Omega}') \tag{6}$$

The main idea behind the characteristics method is to solve the differential form of the transport equation following the straight lines (characteristics or tracking lines). The neutron trajectories will be followed in the local coordinates system where an observer is traveling in the neutron direction. The basic transport operator is transformed into a total differential operator:

$$B^g(\vec{r},\hat{\Omega}) \rightarrow C^g(s) = \frac{d}{ds} + \Sigma^g(\vec{r}+s\hat{\Omega}), \tag{7}$$

where: $C^g(s)$ is the characteristics operator in group g, s is the variable along a line. The multi-group characteristics equation is then given by:

$$C^g(s)\Phi^g(\vec{r}+s\hat{\Omega}) = Q^g(\vec{r}+s\hat{\Omega},\hat{\Omega}). \tag{8}$$

The average scalar flux in region j with volume equal to V_j and in group g is obtained by integrating over volume and directions:

$$\begin{aligned} V_j\Phi_j^g &= \int_{V_j} d^3r \int_{4\pi} d^2\Omega\ \Phi^g(\vec{r},\hat{\Omega}) \\ &= \int_{\Upsilon} d^4T \int_{-\infty}^{+\infty} dt\ \chi_{V_j}(\vec{T},t)\ \Phi^g(\vec{p}+t\hat{\Omega},\hat{\Omega}). \end{aligned} \tag{9}$$

A characteristics line $\vec{T}$ is determined by its orientation $\hat{\Omega}$ along with a reference starting point $\vec{p}$ for the line. Variable t refers to the local coordinates on the tracking line, and the function $\chi_{V_j}(\vec{T},t)$ is defined as 1 if the point $\vec{p}+t\hat{\Omega}$ on the line $\vec{T}(\hat{\Omega},\vec{p})$ in the region V_j, and 0 otherwise. The d^4T element is then decomposed into a solid angle element $d^2\hat{\Omega}$ and a corresponding plane element d^2p which constitute the Υ domain.

2.1 Discretization of Characteristics Equations

The spatial and the angular domains are sampled using a ray tracing procedure as described in [6]. The Υ domain is first covered by choosing a quadrature set of solid angles, composed of discretized directions and their corresponding weights $(\hat{\Omega}_i,\omega_i)$. We generally use the Equal Weight Quadrature (EQ_N) to generate the angular directions.[9] Then, the plane $\Pi_{\hat{\Omega}_i}$ perpendicular to any selected direction i, is split into a Cartesian grid meshing and the starting points

$\vec{p}_{i,n}$ of each characteristics are found. For each discretized direction i, a whole set of tracks $\vec{T}_{i,n}$ will be generated. When travelling across different regions, the neutron beam following the characteristics crosses $K_{i,n}$ segments numbered by index k; the segment lengths are labeled $L_{i,n,k}$ and the region numbers are N_k. The multigroup discretized equations are then derived from Eq. 8 as:

$$\frac{d}{ds_{j,i,n,k}}\Phi^g_{j,i,n,k} + \Sigma^g_j \Phi^g_{j,i,n,k} = F^g_j + S^g_{0,j} + S^g_{1,j,i,n,k} \tag{10}$$

The anisotropic source is obtained from the discretization of Eq. 6:

$$S^g_{1,j,i,n,k} = \frac{3}{4\pi}\sum_{g'} \Sigma^{g\leftarrow g'}_{s1,j} \sum_m \omega_m \left(\hat{\Omega}_i.\hat{\Omega}_m\right) \Phi^g_{j,m,n,k} \tag{11}$$

where ω_m are the angular weights in the quadrature and the solid angle vector is given by:

$$\hat{\Omega}_i \equiv \begin{bmatrix} \sqrt{1-\mu_i^2}\cos\varphi_i \\ \sqrt{1-\mu_i^2}\sin\varphi_i \\ \mu_i = \cos\theta_i \end{bmatrix} \equiv \begin{bmatrix} \cos\gamma_i \\ \cos\lambda_i \\ \mu_i = \cos\theta_i \end{bmatrix} \tag{12}$$

where $\cos\gamma_i$, $\cos\lambda_i$, μ_i are the director cosines on x, y and z axis respectively. Using the relationship between these two angles representations, we obtain:

$$\begin{cases} \Theta_{i,m}(\mu,\varphi) = \mu_i\mu_m + \sqrt{1-\mu_i^2}\sqrt{1-\mu_m^2}\cos(\varphi_i - \varphi_m) = \Theta_{i,m}(\mu,\gamma,\lambda); \\ \varphi_i = Arctan\left(\frac{\cos\lambda_i}{\cos\gamma_i}\right); \\ S^g_{1,j,i,n,k} = \frac{3}{4\pi}\sum_{g'}\Sigma^{g'}_{s1,j}\sum_m \omega_m \Theta_{i,m}(\mu,\gamma,\lambda)\Phi^g_{j,m,n,k}. \end{cases} \tag{13}$$

Assume that the segment k crosses region j, then the local relationship between the incoming and outgoing angular flux is given by integrating the Eq. 10 along the track:

$$^{\text{out}}\phi^g_{j,i,n,k} = {}^{\text{in}}\phi^g_{j,i,n,k} + \left[\frac{\bar{Q}^g_j}{\Sigma^g_j} + \frac{S^g_{1,j,i,n,k}}{\Sigma^g_j} - {}^{\text{in}}\phi^g_{j,i,n,k}\right] E(\tau^g_{j,i,n,k}) \tag{14}$$

where:
$^{\text{out}}\phi^g_{j,i,n,k} = \phi^g_{j,i,n,k}(L_{i,n,k})$;
$^{\text{in}}\phi^g_{j,i,n,k} = \phi^g_{j,i,n,k}(0)$;
$\tau^g_{j,i,n,k} = \Sigma^g_j L_{i,n,k}$, is the neutron optical path;
$E(x) = 1 - e^{-x}$.

$\bar{Q}^g_j$ is composed of fission and isotropic scattering sources. We then integrate Eq. 14 along the segment to get the segment average angular flux. Assuming that the track lengths have been normalized to ensure volume conservation, the average angular flux per region is given by:

$$\Sigma^g_j V_j \bar{\Phi}^g_{j,i} = V_j \bar{Q}^g_j + \sum_{nls} \pi_{i,n} \sum_k \delta_{jN_k}\left[S^g_{1,j,i,n,k} L_{i,n,k} + \Delta^g_{j,i,n,k}\right] \tag{15}$$

where $\Delta^g_{j,i,n,k} = {}^{\text{in}}\phi^g_{j,i,n,k} - {}^{\text{out}}\phi^g_{j,i,n,k}$ is the flux difference on segment k and $\pi_{i,n}$ is the weight associated with track $\vec{T}_{i,n}$.

The average scalar flux in region j and energy group g is computed by integrating Eq. (15) over all directions to obtain:

$$\phi_j^g = \frac{\tilde{Q}_j^g}{\Sigma_j^g} + \frac{1}{\Sigma_j^g V_j} \sum_i \omega_i \sum_n \pi_{i,n} \sum_k \delta_{jN_k} \left[S_{1,j,i,n,k}^g L_{i,n,k} + \Delta_{j,i,n,k}^g \right] \tag{16}$$

From this equation, we see that the iteration sweep over all characteristics involves the sequential evaluation of outgoing fluxes of Eq.(14) on every single track and the adding of flux differences $\Delta_{j,i,n,k}^g$ for each track $\vec{T}_{i,n}$ in Eq.(16).

2.2 Boundary Conditions

At the boundary of the external surfaces, the outgoing current is calculated using the angular flux computed on the segment K crossing the surface α:

$$J_\alpha^+ = \sum_i \omega_i \sum_n \pi_{i,n} \chi_{\alpha,K} \hat{\Omega}_i \cdot \hat{N}_\alpha \, {}^{\text{out}}\phi_{j,i,n,K}, \quad \hat{\Omega}_i \cdot \hat{N}_\alpha > 0 \tag{17}$$

and $\chi_{\alpha,K}$ is defined as:

$$\chi_{\alpha,K} = \begin{cases} 1, if\ \vec{r} \in S_\alpha \\ 0, otherwise. \end{cases} \tag{18}$$

The current is composed of two terms:

$$J_\alpha^+ = {}^{\text{iso}}J_\alpha^+ + {}^{\text{aniso}}J_\alpha^+ \tag{19}$$

where:

$${}^{\text{iso}}J_\alpha^+ = \sum_i \omega_i \sum_n \pi_{i,n} \chi_{\alpha,K} \hat{\Omega}_i \cdot \hat{N}_\alpha \left[{}^{\text{in}}\phi_{j,i,n,K} + \left(\frac{\tilde{Q}_j}{\Sigma_j} - {}^{\text{in}}\phi_{j,i,n,K} \right) E(\tau_{j,i,n,K}) \right] \tag{20}$$

is the isotropic part of the current ($S_1 = 0$) and

$${}^{\text{aniso}}J_\alpha^+ = \sum_i \omega_i \sum_n \pi_{i,n} \chi_{\alpha,K} \hat{\Omega}_i \cdot \hat{N}_\alpha \left(\frac{S_{1,j,i,n,K}}{\Sigma_j} E(\tau_{j,i,n,K}) \right) \tag{21}$$

is the anisotropic part. We apply then the quasi-isotropic reflection with albedo. The reflected neutrons are entering at the same position on surfaces but in the opposite directions:

$$J_\alpha^- = \beta_\alpha J_\alpha^+ \tag{22}$$

3. Calculation Methodology and Implementation

In order to compute the anisotropic sources, using the Eq. 13, we must store the angular flux per energy group, region and direction. Therefore, two characteristics sweeps are necessary:

1. compute and store the angular flux per energy group, region and direction;
2. compute the anisotropic scattering source.

This two steps are implemented at the inner iteration procedure. Consequently, the convergence takes more CPU time than without anisotropic scattering sources and the memory size needed is also very large. The global computational scheme is described in the next subsection.

3.1 Computational scheme

initialisation: currents and fluxes.

1. Guess new fission sources and incoming current (**outer loop**);
   - (a) Guess scattering sources (**inner loop**);
   - (b) Compute the flux for each energy group:
     - i. *First characteristics sweep*: compute and store the angular flux per energy group, region, direction;
     - ii. *Second characteristics sweep*: compute the anisotropic scattering source;
     - iii. compute the outgoing current at the boundary surfaces and apply the boundary conditions;
     - iv. compute the scalar flux.
   - (c) apply global inner acceleration technique.
   - (d) If flux map are not converged, go to (a).
2. Apply global outer acceleration techniques;
3. Compute the K_{eff} for next neutron generation;
4. If not converged, go to 1.

4. Numerical results

The 3D problem we treat here is the extension of the 2D Lathrop one group problem described in [4]. The geometry of the problem is a $[0,2] \times [0,2] \times [0,2]$ centimeter cube with a fixed source located in $[0,1] \times [0,1] \times [0,1]$ cube. The total and isotropic scattering cross-sections are $\Sigma_t = 0.75\ cm^{-1}$ and $\Sigma_{s0} = 0.5\ cm^{-1}$ respectively. For the anisotropic problem, we add a forward-scattering cross-section $\Sigma_{s1} = 0.5\ cm^{-1}$ everywhere. Reflections are assumed as boundary conditions on all faces. Our analysis is based on the comparison between the isotropic and the anisotropic solutions. To do so, two different metrics are used:

$$E_{max} = \max_j \left(\frac{| \phi_j^{aniso} - \phi_j^{iso} |}{\phi_j^{iso}} \right) \times 100; \tag{23}$$

and

Table 1: Maximum percentage error for different spatial splitting

Meshing	$E_{max}(\%)$
2×2 × 2	12.69
4×4 × 4	17.99
8×8 × 8	18.47

Table 2: CPU time for the two solutions

Solution	Isotropic	Anisotropic
CPU Time (s)	59	1630
Number of iterations	13	13

$$E_j = \frac{\mid \phi_j^{aniso} - \phi_j^{iso} \mid}{\phi_j^{iso}} \times 100; \tag{24}$$

where E_{max} and E_j are the maximum percentage error on all regions and the percentage error for a given region respectively.
ϕ_j^{aniso} and ϕ_j^{iso} are the anisotropic and isotropic integrated flux in region j respectively.
The Equal Weight Quadrature EQ_N [9] is used to discretize the angular variable, the number of angles is given by:

$$Nang = \frac{N(N+2)}{8} \tag{25}$$

We use different meshing by splitting the domain on equivolumetric zones, different number of angles by varing the values of N in the quadrature and different density of tracks (DT). In Table 1, we show the maximum errors for different meshing. We use 8 angles and a density of tracks equal to 20 tracks/cm^2. It shows that the max error varies slightly with the spatial meshing. However, the max error on fluxes reaches about 18%.
We have fixed the values of x and y at $0.25cm$ respectively and then we present the values of the scalar flux, with and without anisotropic scattering, corresponding to different z-planes. the percent errors between the two fluxes is important in the first four z-planes where the external source is located Fig. 1. The anisotropic scalar flux map is also shown in Fig. 2. We observed that the highest errors are concentrated in the corners where the neutrons are reflected on two surfaces. In Fig. 3-4, we report the pecent errors for the z-planes 1 and 8 respectively. In Table 2 we show the CPU time spend by the isotropic and anisotropic solutions. The convergence is reached at the same number of iterations for both solutions.

5. Conclusion

The extension of the characteristics solver to include the linearly anisotropic treatment of the scattering sources, in 3D geometries, has been presented. The results, comparing the isotropic and anisotropic fluxes, show that neglecting the anisotropic effects in scattering sources may induce significant errors on flux calculation especially on the regions near the reflected boundaries.

Acknowledgements

This work has been carried out partly with the help of grants from the Natural Science and Engineering Research Council of Canada.

References

1) R. Roy, "Anisotropic Scattering for Integral Transport Codes. Part2: Cyclic Tracking and its Application to XY Lattices", Ann. Nucl. Energy, **18**, pp. 511-524 (1991).
2) J. L. Vujic and W. R. Martin, "Two-Dimensional Collision Probability Algorithm with Anisotropic Scattering for Vector and Parallel Processing", Proc. Of PHYSOR-90, Marseille, France, April 23-27, (1990).
3) R. Roy, "Anisotropic Scattering for Integral Transport Codes. Part1: Slab Assemblies", Ann. Nucl. Energy. **17**, pp. 379-388 (1990).
4) R. Roy, "The Cyclic Characteristics Method with Anisotropic Scattering", Proc. Conf. ANS-MathComp'99 , Madrid (1999).
5) T. Postma and J. Vujic, "The Method of Characteristics in General Geometry with Anisotropic Scattering", M&C Conference, Madrid, (1999).
6) M. Dahmani, G. J. Wu, R. Roy and J. Koclas, "Development and Parallelization of the Three-Dimensional Characteristics Solver MCI of DRAGON", Proc. PHYSOR 2002, Seoul, Korea, October 7-10, (2002).
7) G. Marleau, A Hebert and R. Roy, "A User's Guide for DRAGON", Technical Report IGE-174 Rev.3., École Polytechnique de Montréal, December (1997).
Available at: http://www2.polymtl.ca/nucl
8) J. R. Askew, "A Characteristics Formulation of the Neutron Transport Equation in Complicated Geometries", AEEW-M 1108 (1972).
9) B. Carlson, "Table of Equal Weight Quadrature EQ_N Over the Unit Sphere", Technical Report LA-4734, Los Alamos National Laboratory, (1971).

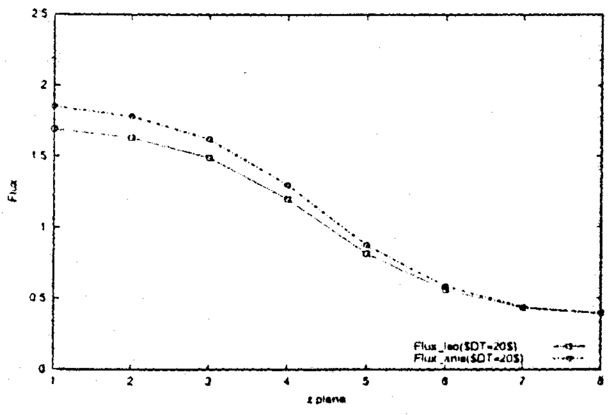


Figure 1: Isotropic and anisotropic flux for ($x = 0.25cm; y = 0.25cm$)

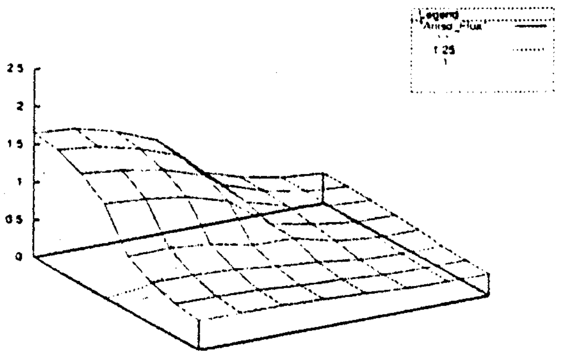


Figure 2: The Anisotropic scalar flux in plane $z = 1$ for $(8 \times 8 \times 8)$ mesh

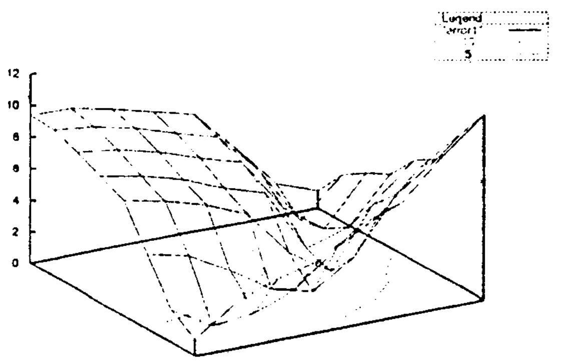


Figure 3: Percent error per region in plane $z = 1$ ($8 \times 8 \times 8$ mesh)

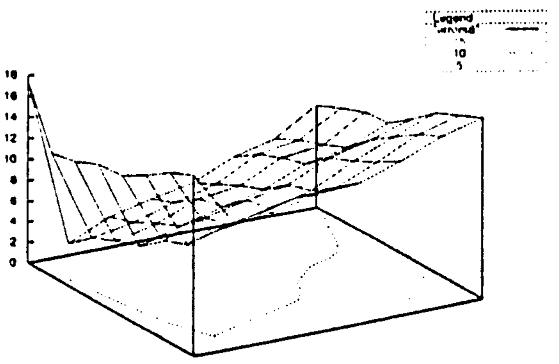


Figure 4: Percent error per region in plane $z = 8$ ($8 \times 8 \times 8$ mesh)

ANNEXE II

UN APERÇU SUR LE CALCUL PARALLÈLE

II.1 À propos du calcul de haute performance

Le *Calcul de Haute Performance* (*CHP*) peut être défini comme la technologie utilisée pour résoudre les problèmes scientifiques qui nécessitent d'énormes puissances de calcul et des accès et traitements intensifs sur de grandes quantités de données. La réduction du temps de calcul pour ce genre d'applications est le but ultime du CHP.
Les applications du CHP peuvent être classées en deux catégories:

1. *Les applications séquentielles*: La méthode traditionnelle d'exécuter une application est de l'exécuter en série (les instructions ou les tâches sont exécutées l'une après l'autres) en utilisant seulement un seul processeur. Ces applications sont simples à développer et sont les plus difficiles à optimiser. En effet, en exécution séquentielle, il est impératif d'essayer d'exécuter chaque instruction ou tâche aussi rapidement que possible. Pour avoir une bonne performance pour les applications séquentielles de CHP, la prise en considération de la puissance du processeur et de la mémoire est primordiale. En plus de cela, l'optimisation de l'application au niveau des sources du code et au niveau du compilateur aide beaucoup à atteindre la performance optimale.

2. *Les applications parallèles*: De nos jours, la grande majorité de stratégies de programmation en CHP utilisent le parallèlisme. Le parallèlisme peut être appliqué soit en utilisant des multi-processeurs d'un seul ordinateur ou en utilisant un ensemble de processeurs impliquant plusieurs ordinateurs (*Grappes*).

Ces ordinateurs s'appelent alors les noeuds de la grappe. Quand le parallèlisme est utilisé pour exécuter plusieurs instances de l'application en travaillant sur differentes données sur differents noeuds de la grappe, ceci peut être vu comme l'exécution de plusieurs programmes séquentiels sur plusieurs noeuds qui travaillent ensemble sur le même problème. Par conséquent, les techniques d'optimisation des applications séquentielles sur chaque noeud peuvent être utilisées pour atteindre la performance optimale. Pour atteindre la solution finale, souvent les processeurs ou les processus ont besoin de communiquer. Selon la fréquence des communications, les applications parallèles peuvent être divisées en applications fortement couplées et applications lâchement couplées (voir Figure II.1).

- **Applications fortement couplées**: Ce sont des applications où les processus communiquent plus fréquemment. Pour ces applications, des performances interessantes sont atteintes sur des systèmes d'architectures comme des SMP (Symmetric Multiprocessing) ou MPP (Massively Parallel Multiprocessor) utilisant des réseaux d'interconnexion les plus rapides.
- **Applications lâchement couplées**: Ces applications requièrent un minimum de communications entre processus. Le modèle de communication typique pour ce genre d'application est celui de maître-esclave. De bonne performances peuvent être atteintes sur des grappes de calcul composées de plusieurs ordinateurs reliés par un réseau Ethernet standard, sur des réseaux de stations de travail (NOW) ou grilles.

Dans la suite de ce chapitre, nous allons revenir plus en détail sur le calcul parallèle à travers trois axes majeurs: l'architecture, la programmation et les performances.

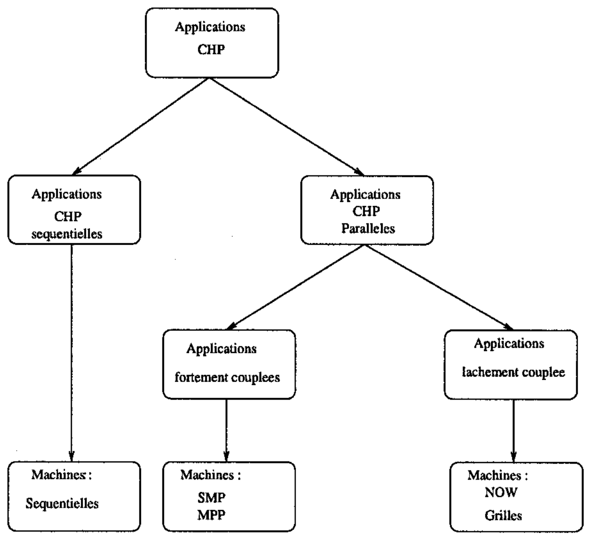

Applications CHP
Applications CHP sequentielles
Applications CHP Paralleles
Applications fortement couplees
Applications lachement couplee
Machines : Sequentielles
Machines : SMP MPP
Machines : NOW Grilles

Figure II.1 Organisation du calcul de haute performance

II.2 Présentation du calcul parallèle

Une des définitions classiques d'un ordinateur parallèle est : un ordinateur parallèle est un ensemble de processeurs capables de coopérer à l'exécution d'une tâche (Foster, 1994). Cette définition s'adapte à une grande variété de machines qu'elles soient des super-ordinateurs comportant quelques centaines ou milliers de processeurs, des ordinateurs vectoriels ou plus simplement des réseaux de station de travail. Néanmoins, toutes ces architectures tendent vers le même but : permettre l'implémentation d'applications nécessitant des ressources informatiques importantes.

II.2.1 Architectures parallèles

Une machine séquentielle de Von Neumann est constituée d'une unité de calcul, d'une unité de mémoire et d'une unité de contrôle qui gère les relations entre la mémoire et l'unité de calcul.

Une machine parallèle consiste en la réplication de tous ou partie d'une machine séquentielle (l'unité de calcul étant toujours répliquée). Une machine parallèle est, alors, constituée de plusieurs unités de calcul, d'une ou plusieurs unités de contrôle, d'une ou plusieurs unités de mémoire et d'un réseau assurant les communications entre toutes ces unités.

II.2.1.1 Architecture des mémoires

Une machine de type SIMD (Single Instruction stream Multiple Data stream) ou MIMD (Multiple Instruction stream Multiple Data stream) est composée de plusieurs unités de calcul, les *processeurs*, plusieurs unités de mêmoire, les *bancs de mêmoire* et d'un réseau d'interconnexion. La mêmoire peut être:

- *partagée* par tous les processeurs. Le réseau d'interconnexion (*bus*) permet aux processeurs d'accèder à n'importe quel élément de la mêmoire par le biais d'une adresse. L'espace d'*adressage* est *unique* pour tous les processeurs. Cependant, on est rapidement limité par l'apparition de conflits qui se produisent lorsque plusieurs processeurs accèdent à la méme zone de mêmoire, même si on multiplie les bancs pour augmenter le nombre de chemins d'accès simultanés. Ceci conduit à une dégradation des performances. On a rarement construit des machines à plus de 64 processeurs, si ce n'est qu'en utilisant des protocoles de *cohérence de cache* avec des accès mémoire *non-uniformes* ;
- *distribuée* entre tous les processeurs. Chaque processeur dispose de sa propre mémoire et n'accède qu'aux données qui y sont stockées. L'espace d'*adressage* est *local* à chaque processeur. Le réseau de connexion, dit *réseau de communication*, assure l'acheminement des données non locales au processeur qui en a besoin.

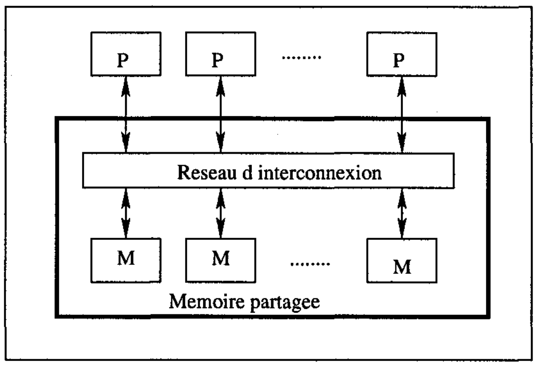


Figure II.2 Organisation d'une machine à mémoire partagée

L'efficacité des chemins de communication est essentielle au bon fonctionnement des machines. Dans les premières générations, on a multiplié le nombre de processeurs dotés de petites mémoire. Ceci s'est traduit par la nécessité de communications intenses qui ralentissaient considérablement les calculs. On a tendance maintenant, en conséquence, à préférer des architectures basées sur des processeurs puissants disponibles pour les stations de travail, des mémoires de bonne capacité et un nombre plus raisonnable de processeurs, rarement supérieur à 512;

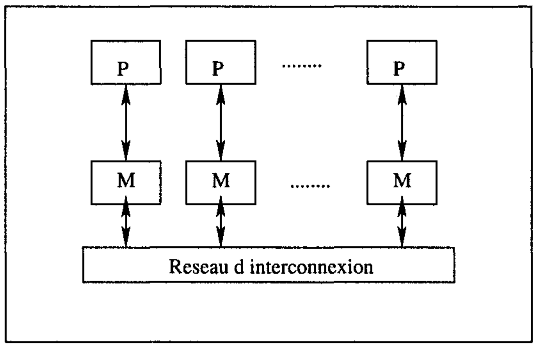


Figure II.3 Organisation d'une machine à mémoire distribuée

- *virtuellement partagée.* Une couche de logiciels permet de gérer une mémoire physiquement distribuée comme une mémoire partagée. Cependant, cette souplesse apparente peut conduire à une mauvaise distribution des données qui multiplie les échanges donc les attentes des processeurs.

II.2.1.2 Architecture des réseaux de communications

Dans une machine parallèle, pour assurer le transfert de données, les processeurs peuvent être connectés de deux facons:

1. par des *bus* (réseau d'interconnexion). L'échange d'information est réalisé par le biais de la mémoire commune;
2. par des *réseaux de connexion.* Les processeurs sont connectés par des réseaux externes via des *switchs.* Pour qu'un processeur accède rapidement à ses voisins, des *topologies* de communication ont été réalisées. Comme exemple, on peut citer les topologies en hypercube, tore, etoile...
   La performance de ces réseaux est mesurée principalement par des paramètres comme:
   - *le diamètre*: c'est la distance maximum entre deux processeurs quelconques dans le réseau. Cette distance détermine le temps de communication, les réseaux avec un faible diamètre sont les meilleurs;
   - *la connectivité*: elle mesure la multiplicité des chemins entre deux processeurs donnés. Un réseau avec une grande connectivité est meilleur;
   - *le coût*: plusieurs critères sont utilisés pour évaluer le coût d'un réseau. On peut le définir en terme de nombre de liens de communication ou le nombre de câbles nécessaires pour construire le réseau.

   Selon ces considérations, le choix d'une topologie bien définie peut améliorer la performance et le coût du réseau selon que ce réseau est complètement connecté ou incomplètement connecté.

- les *réseaux complètement connectés* offrent deux types de topologies physique: point-à-point et topologie en bus. Ce type de topologie ne constitue pas un bon rapport entre la connexion et la contention. En plus, leur réalisation peut s'avérer coûteuse à cause des nombreuses connexions qu'il faut réaliser;
- les *réseaux incomplètement connectés* englobent deux autres types de topologies: hypercube et anneau. Ces topologies offrent un rapport entre la connexion et la contention équilibrée et nécessitent moins de connexions.

Dans quelques applications parallèles, le choix d'une topologie physique peut produire un gain énorme en performance en facilitant les communications entre processeurs en établissant une équivalence entre la topologie physique et la topologie logique qui gouverne le cheminement des messages entre processeurs d'une machine à passage de messages.

3. par la combinaison de ces deux modes de communications. On rencontre ce genre d'architecture sur les plus récentes machines qui sont constituées par des machines à mémoire partagée connectées par un réseau externe. Par exemple des clusters de *SMP* (Figure II.4).

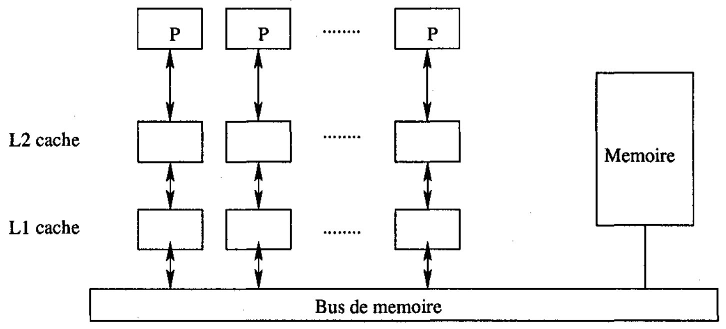


Figure II.4 Organisation d'une machine SMP

II.2.2 Programmation parallèle

Il existe plusieurs modèles de Programmation parallèle (Gengler, 1996). Le *parallèlisme de données* et le *parallèlisme de tâches* comptent parmi les modèles les plus utilisés. Nous décrirons dans la section qui suit ces deux modèles.

II.2.2.1 Les modèles de Programmation parallèle

Le parallèlisme de données

C'est un modèle tout indiqué lorsqu'une application parallèle manipule intensivement des structures *régulières* de données comme des grilles. Les données sont distribuées sur les processeurs. Cette distribution engendre une répartition du travail. Chaque processeur exécute simultanément les mêmes instructions sur la partition de données qui lui est attribuée.
Ce modèle est très bien adapté aux machines de type SIMD dont la caractéristique principale réside dans une exécution synchrone d'instructions sur des flots de données. Pour une machine de type MIMD, le parallèlisme de données va consister à coordonner le travail des processeurs par une resynchronisation entre deux calculs en parallèles.
Afin d'obtenir une application correcte, il faut respecter les dépendances des données lors de leur partition sur les processeurs. La distribution de données est construite de manière à limiter l'utilisation par un processeur de données qui ne lui sont pas attribuées. Assez souvent, un *graphe de dépendance des données* est utilisé.

Le parallèlisme de tâches

Ce modèle de programmation consiste à faire exécuter simultanément par les processeurs des opérations différentes, *les tâches.* Il est fondamentalement asynchrone et ne peut s'envisager que sur des architecture de type MIMD.
L'application parallèle est découpée en un certain nombre de tâches et un *graphe de dépendance des tâches* est construit. Ce graphe indique l'ordre suivant lequel les tâches doivent s'exécuter et les tâches pouvant s'exécuter simultanément. À partir de ces informations, les tâches sont réparties sur différents processeurs de la machine parallèle.
Lors du *découpage* des tâches, un équilibre est recherché entre la minimisation des dépendances et une distribution maximale du travail. Le *placement* des tâches sur les processeurs doit permettre un bon *equilibrage de charge* (du travail) afin que des processeurs ne soient pas inactifs par manque de travail ou de données issues d'autres tâches. De plus, selon l'architecture de la machine parallèle, le lancement d'une tâche peut prendre plus ou moins de temps.
Un modèle très utilisé dans le parallèlisme de tâches est le modèle *maître-esclave.* Une tâche principale, le *maître*, contrôle la répartition du travail sur les autres tâches, les *esclaves*, notamment en assurant la communication des données et des résultats entre les tâches.

II.2.2.2 Communication par passage de messages

Placons-nous dans le cadre des machines parallèles MIMD à mémoire distribuée. Chaque processeur dispose d'une mémoire privée et tout accès par ce processeur à une donnée non locale se fait en adressant une requête au processeur détenant la donnée en question.
La *programmation par processus communicant* est la technique utilisée dans ce contexte. Nous allons expliquer dans le paragraphe suivant cette technique, mais avant de procéder, définissons d'abord la notion de processus.
Un *processus* est un ensemble d'instructions qui s'exécutent séquentiellement. Un

programme parallèle est composé de processus travaillant à la résolution d'un problème. Ce travail en commun nécessite des *communications* entre les processus.
Une communication peut être *point-à-point* losque seulement deux processus entre en communication ou *collective* lorsqu'elle s'applique sur un groupe de processus. Une procedure de communications sera dite *bloquante* lorsque les ressources spécifiées lors de son appel ne peuvent être réutilisées qu'après l'achèvement de la communication. En particulier, l'exécution est en attente du retour de la communication. Une procedure de communications sera dite *non-bloquante* lorsqu'elle se termine avant que l'opération de communication associée soit finie.
Un premier modèle plus strict a été développé en 1978 par Hoare (Hoare, 1978) introduisant le principe de communication entre processus. Il s'agit de *CSP* ou *Communicating Sequential Processes.* Les communications entre processus sont synchrones et bloquantes et utilisent le principe de *rendez-vous* (Almasi, 1994; Braunl, 1993). La programmation par processus communicant est plus souple et autorise des communications asynchrones et non-bloquantes.
Plusieurs implémentations de cette technique de programmation sont possibles. Chaque variante correspond à une manière d'attribuer les processus aux processeurs. L'attribution peut être dynamique ou statique, comprendre une mono-programmation (un processus par processeur) ou une multi-programmation (n processus par processeur). Une attribution dynamique posera le problème de la régulation des charges et le suivi de la charge de travail au cours du déroulement du programme. Une attribution statique imposera de chercher le meilleur placement possible des différents processus. Nous distinguerons alors deux structures de programme:

- la structure *SPMD* ou *single program multiple data* où tous les processus exécutent le même code;
- la structure *MPMD* ou *multiple program multiple data* où les processus exécutent des codes différents.

La structure du programme ne reflète pas le modèle de programmation utilisé, parallèlisme de données ou parallèlisme de tâches, mais uniquement la manière dont le programmeur a écrit le code.

II.3 Mesures des performances

La mesure des performances est essentielle lors de la conception et de l'implantation d'un algorithme parallèle. Elle permet de mesurer l'impact de la programmation sur une machine parallèle donnée d'une application, de repérer les goulets d'étranglements dans cette application et de prendre les mesures adéquates afin d'y remédier.
Pour ce faire, plusieurs métriques sont utilisées pour quantifier les performances. Dans la suite, nous allons décrire et expliquer ces métriques.

II.3.1 Métriques

Il faut savoir quelle référence servira à mesurer les performances d'une application parallèle. Il existe de nombreuses métriques possibles. En voici une liste succincte : le temps d'exécution, l'accéleration (Speed up) et l'efficacité parallèle, l'évolutivité (Scalability), la place mémoire nécessaire, le taux d'E/S au niveau de la mémoire, le débit dans le réseau entre processeurs.

Ce sont les principales références à partir desquelles on jugera de la performance d'une application parallèle. D'autres critères entrent également en ligne de compte comme dans tout développement de programme informatique: les coûts de conception, d'implémentation, de vérification, de maintenance, les besoins et les coûts en matériel informatique et en logiciel...
Le choix d'une métrique va dépendre des orientations choisies pendant la concep-

tion et la mise en oeuvre de l'application parallèle : minimiser le temps de travail, disposer d'un code le plus rapidement possible, avoir une efficacité maximum...
En outre, selon le point de vue de l'utilisateur, les jugements portés sur les performances d'une application parallèle peuvent être différents. Ainsi, l'efficacité mesure l'utilisation des ressources de la machine parallèle par un programme. Du point de vue du théoricien, une efficacité de 25% est décevante tandis que du point de vue d'un industriel, le problème est résolu à un coût raisonnable et donc une efficacité de 25% est convenable.
En général, les performances d'un algorithme parallèle dépendent:

- du matériel (Hardware):
  - puissance de calcul des processeurs,
  - temps d'accès à la mémoire,
  - capacité de transfert de données à travers le réseau;
- des logiciel utilisés (Software):
  - compilateur,
  - bibliothèques éventuelles,
  - le codage de l'application;

Trois métriques sont principalement utilisées:

- la puissance du calcul;
- le temps d'exécution;
- l'accéleration ou l'efficacité obtenue par la programmation parallèle.

II.3.1.1 Puissance de calcul

La puissance de calcul correspond à la mesure du nombre d'opérations flottantes par unité de temps. Elle est donnée en *Floating-Point Per Second* ou *Flops.* Elle se fait par rapport à une référence commune. Mais de nombreuses instructions ne sont pas des opérations flottantes comme les boucles, les instructions conditionnelles, les accès mémoire et les accès cache.
En générale, le constructeur fournit une *puissance crête*, puissance maximale atteinte par la machine. En pratique, il est possible d'espérer d'obtenir entre 1 % et 50 % de cette puissance.
Cette mesure des performances est peu utilisée dans le domaine de l'informatique parallèle. Elle ne permet pas de rendre véritablement compte de l'effet que la programmation parallèle à sur une application.

II.3.1.2 Temps d'exécution

Considérons le temps d'exécution en tant que métrique possible pour la mesure des performances d'une application parallèle.
La définition du *temps d'exécution* sur un ordinateur séquentiel est assez simple. Il s'agit du temps écoulé entre le début et la fin de l'exécution.
Mais sur une machine parallèle, la définition du temps d'exécution pose un problème. En effet, plusieurs applications tournent en même temps et se partagent les ressources de la machine. Dans ce cas, pour une même application et des données initiales identiques, plusieurs exécutions peuvent donner des temps très différents selon l'état d'occupation de la machine.
Il est donc nécessaire de bien définir le temps effectivement mesuré. Il en existe deux types:

- le *temps d'horloge* qui correspond à un chronométrage du temps écoulé. Il peut varié énormément suivant l'occupation de la machine;

- le *temps CPU* qui est le temps pendant lequel la machine effectue réellement les instructions du programme. Ce temps est mesuré à partir de l'unité de contrôle du processeur de la machine.

Ces temps peuvent être très différents et le choix d'un temps de référence dépendra de l'orientation choisie pendant le développement du code. Dans le cas du temps d'horloge, des moyennes devront être établies à partir d'un grand nombre d'exécutions afin de pouvoir obtenir des mesures significatives.
Dans le cas des machines parallèles, il y a, en outre pour un choix de mesure de temps, trois possibilités:

- une mesure sur un processeur donné;
- le maximum sur tous les processeurs;
- une moyenne sur tous les processeurs.

Toutefois, le temps d'exécution n'est pas la métrique la plus pratique pour mesurer l'impact de la programmation parallèle sur une application. Ce temps dépend trop fortement de la taille du problème. Considérons une application parallèle donnée utilisant une distribution des données. Le temps d'exécution décroit lorsque le nombre de processeurs utilisé croît, et ce, jusqu'à une borne inférieure à partir de laquelle le temps stagne, voir augmente. Pour un même nombre de processeurs utilisés, un problème important aura un temps d'exécution supérieur à celui d'un problème plus petit, mais il utilisera mieux les ressources de la machine parallèle. En particulier, si un problème peu important a atteint sa limite inférieure de temps pour un nombre donné de processeurs, un problème plus gros peut encore abaisser son temps d'exécution en utilisant plus de processeurs.

II.3.1.3 Accélération et efficacité

Supposons choisie une manière de mesurer le temps d'exécution d'une application parallèle. Deux quantités, *l'accélération* et *l'efficacité* sont obtenues après normalisation par un temps d'exécution de référence. Nous allons définir ces notions, expliquer ce qu'elles représentent physiquement et terminer en présentant quelques propriétés.

Définition 1 *Accélération et efficacité parallèles*
L'accélération S d'une application parallèle est définie comme le rapport entre le temps T_P d'exécution sur P processeurs et un temps de référence T_S sur un processeur.

$$S = \frac{T_S}{T_P} \tag{II.1}$$

L'efficacité $\mathcal{E}$ d'une application parallèle est le rapport entre l'accélération de cette application et le nombre de processeurs P.

$$\mathcal{E} = \frac{S}{P} \tag{II.2}$$

Mais il reste le problème du choix du temps de référence T_S. Une définition stricte de l'accélération implique que ce temps soit celui du meilleur algorithme séquentiel correspondant au problème traité dans l'application parallèle. Une définition plus pratique stipule que ce temps de référence soit le temps d'exécution de l'application sur un seul processeur. Une troisième approche, employée lorsque l'application parallèle est obtenue par parallélisation d'une application séquentielle, utilise le temps d'exécution de l'application séquentielle sur un processeur comme temps de référence.

II.3.1.4 Comportements d'une application parallèle

Pratiquement, l'accélération mesure la faculté de l'application à se paralléliser. L'efficacité évalue l'exploitation par cette application parallèle des ressources de la machine parallèle de test. Il est possible de donner des limites à l'accélération et l'efficacité.

Propriété 1

$$1 \leq S \leq P \qquad \forall P \tag{II.3}$$

D'après la propriété ci-dessus, l'accélération est au plus linéaire et au moins supérieure à un. Cependant, en pratique, il est possible d'observer des accélérations *super-linéaire* ou inférieure à un.

Nous allons donner deux exemples extrêmes de situations où l'accélération ne respecte pas l'inégalité II.3, un effet de cache (Foster, 1994) et une surcharge du réseau de communications.

Considérons, par exemple, une parallélisation sur une machine MIMD à mémoire distribuée par distribution des données. Sur la plupart des machines parallèles, chaque processeur dispose d'une mémoire cache petite et d'accès rapide et d'une mémoire plus lente. Quand le problème est exécuté sur un seul processeur, il occupe beaucoup d'espace mémoire et le transfert de donnée de la mémoire au cache est répété plusieurs fois ceci peut alors induire des délais. Plus le nombre de processeurs augmente, plus la taille du problème (décomposé) a tendance à s'ajuster au cache (ou à la mémoire) liée à chaque processeur individuellement, le delai de communication entre la mémoire et le processeur diminue. Des études traitant des problèmes basés sur la décomposition du domaine exécutés sur des SGI Origine2000 ont montré qu'on faisant varier la taille du problème, l'accélération super-linéaire a été observée dans tous les cas quand on atteint un certain nombre de processeurs (nombre de

processeurs critique) pour lequel correspond un ajustement entier du problème en cache L2. Ce phénomène est lié donc directement à l'effet de cache ou de hiérarchie de la mémoire qui peuvent pénaliser le programme séquentiel.
Dans l'autre cas, supposons que tous les processeurs communiquent au même instant des petites quantités de données et que la réception des données soit indispensable pour que le programme puisse continuer. Le réseau peut alors se trouver en surcharge et les communications sont alors bloquées en cours de passage sur le réseau le temps que le logiciel de gestion du réseau mettent en place une stratégie de désengorgement. Il est possible d'observer alors des accélérations inférieures à un.
De même, on peut aboutir à la propriété suivante pour l'efficacité :

Propriété 2

$$\frac{1}{P} \leq \mathcal{E} \leq 1 \qquad \forall P \tag{II.4}$$

Une autre manière d'étudier l'efficacité et l'accélération est de décomposer le temps d'exécution d'un programme parallèle en trois parties:

$$T_P = T_P^{cal} + t_P^{com} + t_P^{ina} \tag{II.5}$$

- T_P^{cal} est le temps passé dans les calculs par les P processeurs;
- t_P^{com} est le temps passé dans les communications par les P processeurs;
- t_P^{ina} est le temps d'inactivité des processeurs. Il provient des attentes de réceptions et de synchronisations des processeurs.

Les temps de communications et d'inactivité n'existent que pour un programme parallèle. Elles sont regroupées dans une composante qu'on note T_o, nous avons alors :

$$T_P = T_P^{cal} + T_o \tag{II.6}$$

Supposons que le travail soit réparti aussi équitablement que possible entre différents processeurs, alors le temps de calcul sur P processeurs est égal au rapport entre le temps de calcul sur 1 processeur, soit le temps séquentiel, et le nombre P de processeurs:

$$T_P^{cal} = \frac{T_S}{P} \tag{II.7}$$

En reportant les deux relations (II.6) et (II.7) dans les expressions de l'accélération et l'efficacité, nous obtenons:

$$\begin{aligned} S &= \frac{P}{1 + P\frac{T_o}{T_S}} = \frac{P}{1 + P\mathcal{R}_\mathcal{P}} \\ \mathcal{E} &= \frac{1}{1 + P\frac{T_o}{T_S}} = \frac{1}{1 + P\mathcal{R}_\mathcal{P}} \end{aligned} \tag{II.8}$$

avec $\mathcal{R}_\mathcal{P} = \frac{T_o}{T_S}$. Le comportement de la quantité $P\mathcal{R}_\mathcal{P}$ détermine le comportement de l'algorithme parallèle.

Si cette quantité est nulle, l'accélération est linéaire. Si cette quantité est strictement positive, l'accélération sera sous-linéaire.

En 1967, Amdahl formule dans une loi (Foster, 1994; Cosnard, 1993) une limite asymptotique à l'accélération réalisable dans une application parallèle.

Loi d'Amdahl : *Tout programme parallèlisable se compose d'une partie séquentielle f_s et d'une partie parallèlisable f_p. La partie séquentielle correspond à la création de tâches, au traitement des scalaires, à la structure de contrôle et aux accès mémoire.*

L'accélération est alors limitée asymptotiquement par l'inverse de la fraction du temps d'exécution total nécessaire à l'exécution de la partie séquentielle du pro-

gramme

$$S \leq \frac{f_s + f_p}{f_s} \qquad \text{(II.9)}$$

Démonstration

Le programme se compose d'une partie séquentielle f_s et d'une partie parallèlisable f_p. Alors, le temps d'exécution séquentiel vaut $t_s + t_p$ et le temps d'exécution sur P processeur vaut $t_s + \frac{t_p}{P}$. L'accélération tend alors asymptotiquement, lorsque P tend vers l'infini, vers la quantité suivante:

$$S = \frac{t_s + t_p}{t_s + \frac{t_p}{P}} \sim \frac{t_s + t_p}{t_s} \quad quand \quad P \rightarrow \infty \qquad \text{(II.10)}$$

D'après cette loi, la fraction séquentielle va limiter l'accélération que l'on peut espérer obtenir et donc le nombre de processeurs sur lequel il est utile de travailler. Cette loi a beaucoup influencé les premiers temps du parallèlisme. L'informatique parallèle ne semblait alors utile que pour des applications bien précises et pour des machines parallèles avec peu de processeurs.

Dans les années 80, le développement du calcul parallèle massif a remis en question les conclusion de la loi d'Amdahl. De nombreuses expérimentations numériques sur des ordinateurs massivement parallèles ont montré la viabilité du parallèlisme massif.

Il y a de nombreuse raisons au fait que les conclusions de la loi d'Amdahl soient inadéquates dans la plupart des cas. La loi d'Amdahl correspond à une conception séquentielle de la programmation parallèle. Elle ne prend pas en compte le fait que la parallèlisation d'un programme va engendrer du travail suplémentaire pour les processeurs, en particulier des communications chez les machines à passage de messages et des accès mémoire chez les machines à mémoire partagée.

De plus, la loi d'Amdahl ne prend pas en compte un des buts principaux du calcul parallèle, travailler sur des machines parallèles pour pouvoir traiter des problèmes de plus en plus importants. Cette loi correspond à une taille constante du problème traité quelque soit le nombre de processeurs utilisés et la place mémoire disponible. Cette remarque fut à la base du travail de Gustafson (Gustafson, 1988; Wilson, 1995) qui a réévalué le comportement de l'accélération en 1988.

Loi de Gustafson : *Si la taille de la fraction parallèle d'un problème croît suffisamment, alors il est possible d'obtenir n'importe quelle valeur d'accélération quelque soit le nombre de processeur utilisé.*

Démonstration

Le temps d'exécution de la partie parallèle t_p du problème dépend non seulement du nombre de processeur utilisé P mais aussi de la taille du problème W. L'accélération obtenue vaut alors:

$$S = \frac{t_s + t_p(W, 1)}{t_s + t_p(W, P)} \tag{II.11}$$

La fraction parallèle $t_p(W, P)$ décroît avec le nombre de processeurs P mais croît strictement avec la taille du problème. Donc, pour obtenir une valeur particulière de l'accélération pour un nombre donné de processeurs, il suffit d'augmenter suffisamment la taille du problème.

La loi de Gustafson donne une meilleure approche du comportement de l'accélération par rapport à la loi d'Amdahl. Mais elle correspond toujours à un comportement idéal. En pratique, il est possible d'avoir une fraction séquentielle du programme croissante avec le nombre de processeur même si le temps d'exécution de la partie parallèle décroît.

Une autre approche consiste à penser en terme d'éfficacité et d'évolutivité en se demandant comment la taille du problème doit être augmentée afin de maintenir l'efficacité constante à une valeur donnée lorsque le nombre de processeurs augmente. Dans cette approche, on essaye de trouver une relation analytique reliant l'efficacité à la taille du problème et à d'autres paramètres de la machine (entre autre le nombre de processeurs). Ainsi, on définit le *système parallèle* composé de l'algorithme parallèle et de la machine parallèle. On parle alors de l'évolutivité du système parallèle qui consiste à trouver la meilleure combinaison algorithme-machine parallèle pour avoir une efficacité optimale et constante.

Placement de tâches et équilibrage de charges

Un des paramètres qui peut influencer la pérformance d'une application donnée est *l'équilibrage de charges* et *le placement de tâches.* L'équilibrage de charges est la manière avec laquelle on répartit les charges sur les différentes tâches constituant les processus parallèles dans le but de maximiser les performances. Le placement de tâches consiste à répartir les tâches d'une application sur les processeurs d'une machine parallèle de facon à optimiser une fonction de coût.
Un algorithme peut se définir par son graphe de précédence. Il s'agit d'un graphe valué orienté $A(T, D)$ dont les sommets T représentent les tâches élémentaires de l'application et les arcs D correspondent aux dépendances de données entre tâches. Le poids w_T d'une tâche indique sa charge de calcul, et le poids d'un arc w_D de quantité données à transmettre entre les tâches.
La machine parallèle est aussi représentée par un graphe $M(P, L)$, qui cette fois n'est ni valué, ni orienté si les ressources sont homogènes. Les sommets P sont les processeurs, et les arêtes L les liens de communications.
Le placement est alors une fonction $\mathcal{P}$ de T dans P qui, pour être efficace, doit minimiser une fonction de coût.

ANNEXE III

CALCULS DE RÉACTEURS

III.1 La chaîne de calcul de transport

Les calculs de transport sont basés sur des données nucléaires qui sont originellement stockées dans des bibliothèques de données nucléaires sous format *ENDF* (Evaluated Nuclear Data File). Ces données sont collectées à partir des expériences de physique nucléaire ou en utilisant des modèles théoriques. Ces données, ainsi stockées, ne peuvent pas être directement utilisées par des code de transport déterministes, qui eux, utilisent des sections efficaces multigroupes. Un traitement supplémentaire est donc nécessaire pour produire des section efficaces dédiées aux calculs multigroupes. Ceci est réalisé par des codes de traitement des sections efficaces, comme *NJOY* et *SAMMY.* Des librairies de sections efficaces isotopiques multigroupes sont alors générées pour être utilisées directement comme bases de données par des codes de transport.

Une fois que les données nucléaires sont disponibles et la géometrie du problème à traiter est définie, l'effort de calcul est concentré au niveau du solveur afin d'obtenir la solution du flux. Par conséquent, la méthode de résolution et la conception générale du solveur vont être décisives du point de vue des limitations du solveur en termes de temps de calcul, de taille du problème et de ressources informatiques. Nous avons distingué deux approches vis-à-vis de la conception des solveurs :

- *l'approche classique* : des solveurs utilisant des techniques de programmation séquentiels sont élaborés. Des techniques d'accélération numériques sont alors nécessaires pour accélérer la solution. Avec ces solveurs, on est limité à utiliser des machines séquentielles. Dans le contexte de problèmes de grande taille, ces solveurs présentent des limitations liées surtout à la puissance de calcul et à la capacité de stockage.

- *l'approche moderne* : des solveurs utilisant des techniques de programmation parallèle sont conçus. Ces codes peuvent être alors exécutés sur une variété de machines parallèles comme les grappes de calcul, les machines massivement parallèles, des machines hybrides... Dans ce cas d'autres complexités sont introduites vis-à-vis de la performance :
- études de performances en terme d'accélération, d'efficacité, d'évolutivité...;
- choix du réseau d'interconnexion;
- choix de l'architecture de la machine.

Les solveur séquentiels peuvent être parallélisés pour faire parti de la catégorie des solveurs parallèles.

En fin de compte, pour un même problème, on devrait avoir des solutions semblables avec les trois approches de résolution de l'équation de transport de neutrons, mais les temps d'exécution peuvent varier considérablement d'une approche à l'autre. (Figure III.1)

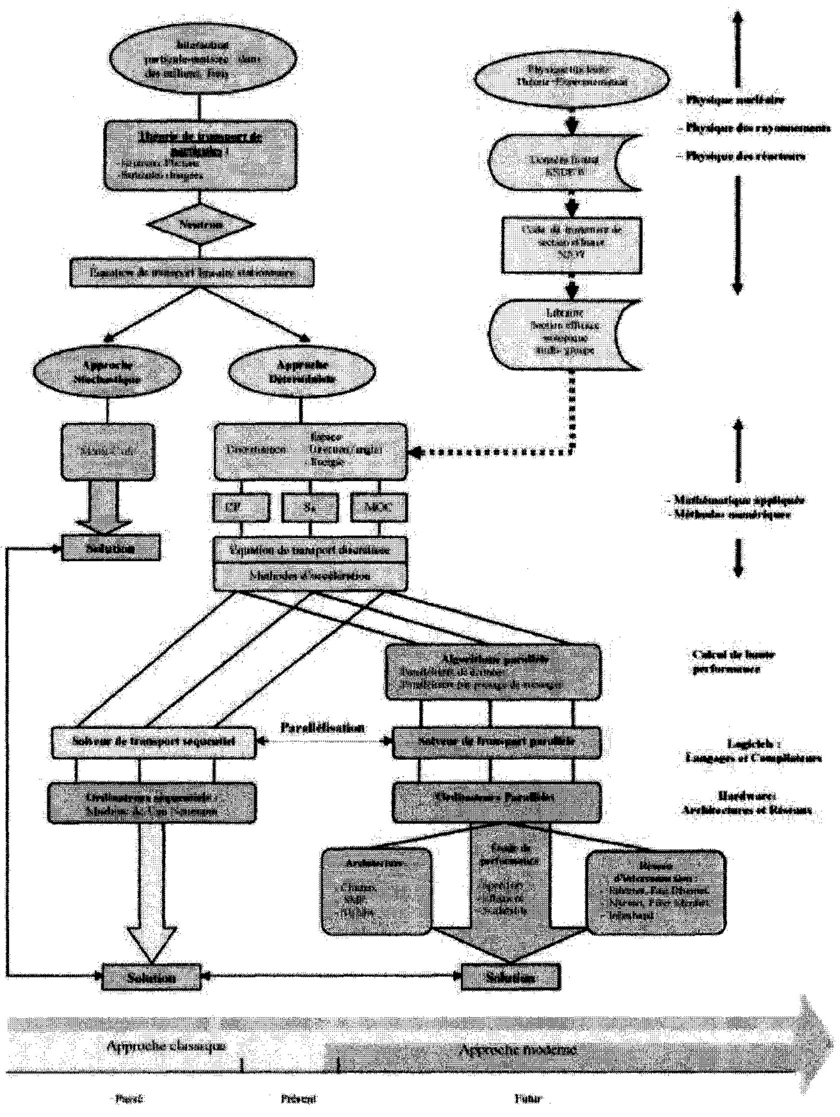

Interaction particule-matière dans des milieux finis
Théorie de transport de particules :
- Neutrons, Photons
- Particules chargées
Neutrons
Équation de transport linéaire stationnaire
Approche Stochastique
Approche Déterministe
Monte Carlo
Solution
Discrétisation
Espace
Direction (angle)
Énergie
CP
S_N
MOC
Équation de transport discrétisée
Méthodes d'accélération
Physique nucléaire
Données brutes ENDF/B
Code de traitement de sections efficaces NJOY
Librairie Sections efficaces isotopique multi-groupe
- Physique nucléaire
- Physique des rayonnements
- Physique des réacteurs
- Mathématique appliquée
- Méthodes numériques
Algorithme parallèle
Parallélisme de données
Parallélisme par passage de message
Calcul de haute performance
Solveur de transport séquentiel
Parallélisation
Solveur de transport parallèle
Logiciels : Langages et Compilateurs
Ordinateurs séquentiels : Modèle de Von Neumann
Ordinateurs Parallèles
Hardware: Architectures et Réseaux
Architecture
- Clusters
- SMP
- Hybrides
Étude de performance
Réseau d'interconnexion :
- Ethernet, Fast Ethernet
- Myrinet, Fiber
- Infiniband
Solution
Solution
Approche classique
Approche moderne
Passé
Présent
Futur

Figure III.1 Schéma de calcul de transport

III.2 Organisation d'un calcul de réacteur

Le calcul neutronique de réacteur comporte deux étapes. La première étape consiste en un calcul fin en espace et en énergie appelé un *calcul de réseau* (*lattice calculation*). Ce calcul, pour la majorité des réacteurs, se fait sur un assemblage de combustible entouré de son modérateur. Les operations de condensation en énergie et d'homogénéisation en espace, permettent d'obtenir les propriétés nucléaires (les sections efficaces) utilisées par la deuxième étape du calcul qui est le calcul du *réacteur* entier. Le calcul de réacteur se fait dans le cadre de l'approximation de diffusion. Des couplages thermohydrauliques sont aussi utilisés à ce niveau pour tenir compte des paramètres thermohydrauliques comme la pression, la température, les concentrations... (Voir Figure III.2).

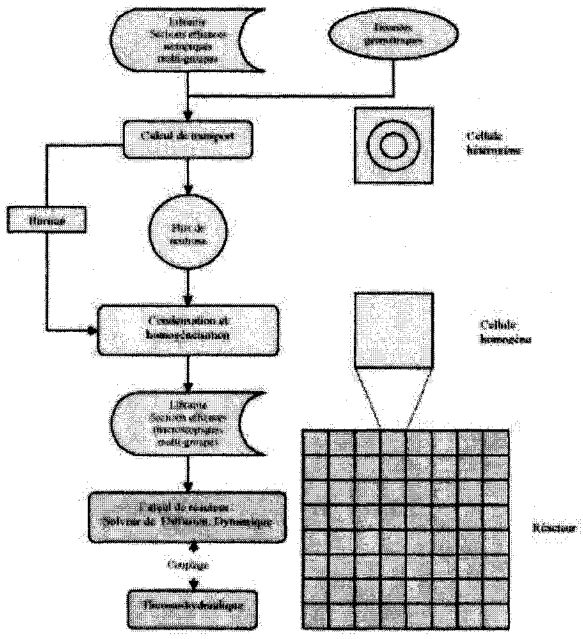


Figure III.2 Schéma simplifié de calcul de réacteur

III.2.1 Caractéristiques des réacteurs CANDU

Les réacteurs CANDU (CANadian Deuterium Uranium) se caractérisent par:

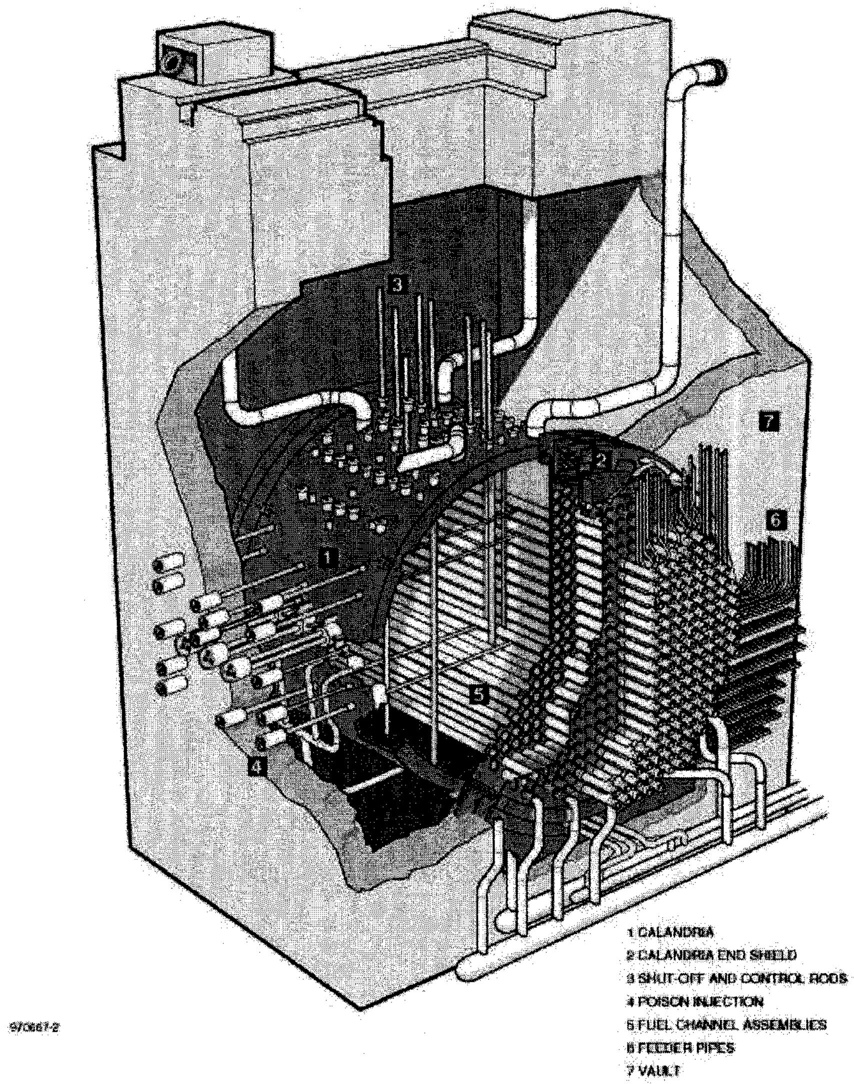
- l'utilisation de leau lourde à la fois comme modérateur et caloporteur;
- une structure à tube de force qui sépare le modérateur froid du caloporteur chaud;
- un combustible à uranium naturel, renouvelé pendant le fonctionnement du réacteur.

Le réacteur CANDU est constitué par une grande cuve cylindrique, appelée calandre, reposant sur sa face latérale. Plusieurs centaines de tubes (dépendant de la puissance du réacteur, entre 380 et 480) traversent la calandre d'une face à l'autre. La calandre est remplie d'eau lourde qui sert de modérateur; la pression et la température de ce dernier sont relativement faibles. D'autres tubes, appelés tubes de forces sont insérés de manière concentrique à l'intérieur des tubes de calandre. Le combustible, sous forme de grappes, est logé dans les tubes de force et son refroidissement est assuré par le caloporteur d'eau lourde qui circule autour de la grappe. L'espacement entre le tube de force et le tube de calandre est rempli par un gaz. Cet espacement sert d'isolant thermique entre le caloporteur et le modérateur. L'ensemble du modérateur et du combustible nucléaire est appelé coeur du réacteur (Figure III.3)[1]. Le combustible nucléaire utilisé dans les réacteurs CANDU actuels est l'uranium naturel. Par exemple, une grappe de combustible des réacteurs CANDU-600, d'une longueur de 50 cm, comporte 37 crayons. Les principales composantes d'un crayon de combustible sont le combustible de dioxyde d'uranium (UO_2) sous forme de pastilles et la gaine en alliage de zirconium (Zircaloy). Le zirconium est utilisé à cause de ses propriétés nucléaires avantageuses (absorbe peu les neutrons). Un crayon de

[1]http://canteach.candu.org/

combustible peut contenir jusqu'à 20 pastilles de UO_2. Les réacteurs de puissance sont fortement hétérogènes. Il existe cependant une certaine périodicité dans la géométrie, en particulier au niveau des éléments du combustible. Cette périodicité nous permet généralement d'introduire la notion de la cellule unitaire.

Dans les reacteurs CANDU, le calcul fin en transport comporte deux étapes : un calcul de *cellule unitaire* en 2D et un calcul de *supercellule* en 3D pour tenir compte des effets des mécanismes de contrôle qui sont perpendiculaires aux grappes de combustible.



CANDU 6 Reactor Assembly

Figure III.3 Le réacteur CANDU-6

III.2.1.1 Calcul de cellule

Les cellules unitaires des réacteurs CANDU sont définies à partir du réseau des canaux de combustible. Elles contiennent chacune une seule grappe de combustible entourée de son modérateur (Figure III.4)[2]. Les cellules unitaires sont caractérisées par des paramètres locaux tels que la densité de puissance, la température du combustible et la densité du caloporteur. Le code de cellule permet d'évaluer en théorie de transport les sections efficaces macroscopiques par matériaux et par région, en fonction de l'énergie des neutrons dans le combustible, ainsi que les taux de réaction dans le combustible.

III.2.1.2 Calcul de supercellule

Pour tenir compte des mécanismes de réactivité qui sont des barres verticales pour les réacteurs CANDU, un calcul de cellule en 3D est nécessaire. Ce genre de calcul s'appelle calcul de supercellule (Figure III.5)[3]. Le calcul de supercellule est un processus complexe. Plusieurs approches ont été utilisées pour de tels calculs. On retrouve principalement les méthodes des probabilités de collision, des courants d'interfaces et dernièrement la méthode des caractéristiques.

[2]Guide d'utilisateur DRAGON: http://www.polymtl.ca/nucleaire/DRAGON/index.php
[3]Guide d'utilisateur DRAGON: http://www.polymtl.ca/nucleaire/DRAGON/index.php

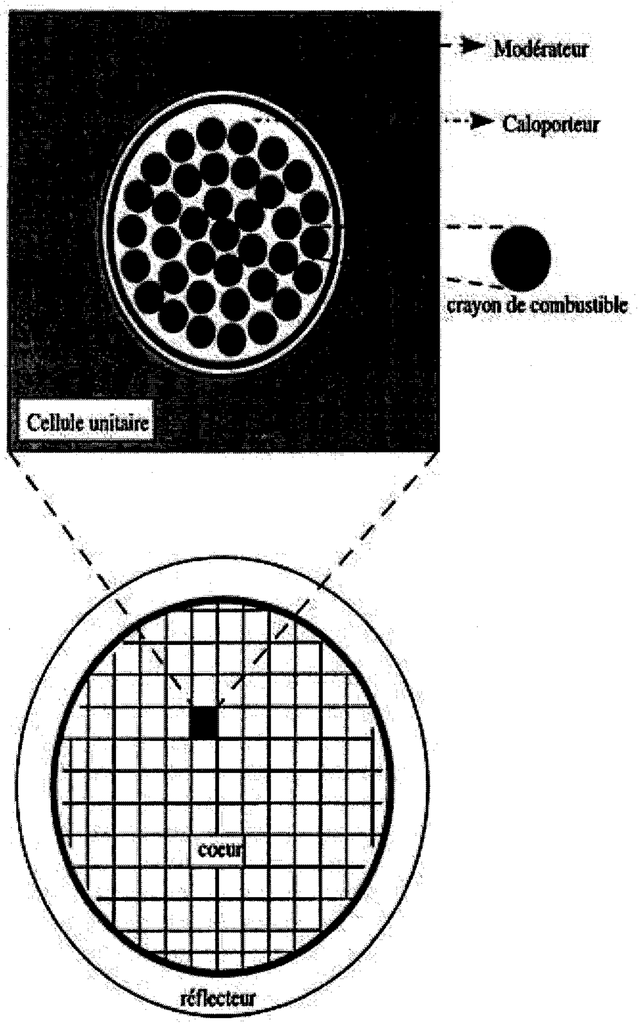


Figure III.4 Une cellule unitaire

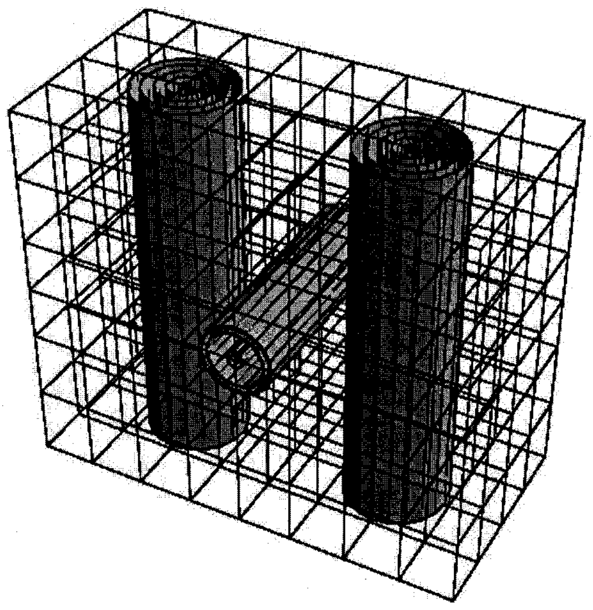


Figure III.5 Une supercellule

ANNEXE IV

MODULE MCG : MODE D'UTILISATION

The MCG: module

The MCG: module can be used to solve the transport equation using the method of characteristics in 3-D geometries. The MCG: module is used to perform the work of both the EXCELT: and the MCI: modules. The MCG: module does not keep the tracking file. The calling specifications are:

Structure **(MCG:)**

TRKNAM FLUNAM := MCG: [*FLUNAM*] *GEONAM LIBNAM* : : **(descmcg)** **(desctrack)**

where

FLUNAM
: `character*12` name of the FLUXUNK data structure containing the solution. If *FLUNAM* appears on the RHS, the solution previously stored in *FLUNAM* is used to initialize the new iterative process; otherwise, a uniform unknown vector is used.

GEONAM
: `character*12` name of the GEOMETRY data structure.

LIBNAM
: `character*12` name of the MACROLIB or MICROLIB data structure that contains the macroscopic cross sections .

TRKNAM
: `character*12` name of the TRACKING data structure containing the tracking .

(descmcg)
: structure containing the input data to this module.

(desctrack)
: structure containing the general tracking data

1 Data input for module MCG:

Structure **(descmcg)**

[EDIT *iprint*]
TYPE { S | K | B {B0 | B1 } [SIGS | PN L] }
[THER [*maxthr*] [*epsthr*]]
[EXTE [*maxout*] [*epsout*]]
[NOBA]
[ACCE *nlibre naccel*]
[SCR maxscr]
[ETAB { ON | OFF}]
[ITLM]

where

EDIT
keyword used to modify the print level *iprint*.

iprint
index used to control the printing of this module. The amount of output produced by this tracking module will vary substantially depending on the print level specified.

TYPE
keyword to specify the type of flux flux calculation to be performed.

S
keyword to specify that a fixed source problem is to be treated. Such problem can also include fission source contributions.

K
keyword to specify that a fission source eigenvalue problem is to be treated. The eigenvalue is then the effective multiplication factor with a fixed buckling.

B
keyword to specify that a fission source eigenvalue problem is to be treated. The eigenvalue in this case is the critical buckling with a fixed effective multiplication factor.

THER
keyword to specify that the control parameters for the thermal iterations are to be modified.

maxthr
maximum number of thermal iterations. The fixed default value is 2**ngroup*-1 or 4**ngroup*-1.

epsthr
convergence criterion for the thermal iterations. The fixed default value is $5.0*10^{-5}$.

EXTE
keyword to specify that the control parameters for the external iteration are to be modified.

maxout
maximum number of external iterations. The fixed default value for a case with no leakage model is $2*n_f$-1 where n_f is the number of regions containing fuel. The fixed default value for a case with a leakage model is $10*n_f$-1.

epsout
convergence criterion for the external iterations. The fixed default value is $5.0*10^{-5}$.

NOBA
keyword used to specify that the flux rebalancing option is to be turned on or off in the thermal iteration. By default (floating default) the flux rebalancing option is initially activated.

ACCE
keyword used to modify the variational acceleration parameters. This option is active by default (floating default) with *nlibre*=3 free iterations followed by *naccel*=3 accelerated iterations.

nlibre
number of free iterations per cycle of *nlibre+naccel* iterations.

naccel
number of accelerated iterations per cycle of *nlibre+naccel* iterations. Variational acceleration may be deactivated by using *naccel*=0..

SCR
keyword used to modify the number of iterations used in the Self-Collision Rebalancing procedure.

maxscr
the number of iterations used in the Self-Collision Rebalancing procedure. The default value is *maxscr=5.*

ETAB
keyword to specify that the option of using exponential tables is to be turned ON or OFF.

ITLM
keyword to specify that the effective number of thermal iterations in the nth outer iteration will not exceed the maximum of n and *maxthr*

Structure **(desctrack)**

```
[ MAXR maxreg ]
[ TRAK { TISO [ { EQW | GAUS } ] nangl dens [ CORN pcorn ] [
SYMM isymm ]
```

MAXR
keyword which permits the maximum number of regions to be considered during a DRAGON run to be specified.

maxreg
maximum dimensions of the problem to be considered. The default value is set to the number of regions previously computed by the GEO: module. However this value is generally insufficient if symmetries or mesh splitting are specified.

TRAK
keyword to specify the tracking parameters to be used.

TISO
keyword to specify that isotropic tracking parameters will be supplied. This is the default tracking option for cluster geometries.

EQW
keyword to specify the use of equal weight quadrature. This option is valid only if an hexagonal geometry is considered.

GAUS
keyword to specify the use of the Gauss-Legendre or the Gauss-Jacobi quadrature. This option is valid only if an hexagonal geometry is considered.

TSPC
keyword to specify that specular tracking parameters will be supplied. This option is invalid if an hexagonal geometry is considered.

MEDI
keyword to specify that instead of selecting the angles located at the end of each angular interval, the angles located in the middle of these intervals are selected. This is particularly useful if one wants to avoid tracking angles that are parallel to the *X*- or *Y*-axis as its is the case when the external region of a CARCEL geometry is voided.

nangl
angular quadrature parameter. For applications involving 3-D cells, the choices are *nangl*=2, 4, 8, 10, 12, 14 or 16; these angular quadratures EQ_n present a rotational symmetry about the three cartesian axes. For 2-D isotropic applications, any value of *nangl* may be used, equidistant angles will be selected. For 2-D specular applications the input value must be of the form *p*+1 where *p* is a prime number (for example *p*=7,11, etc.); the choice of *nangl* = 8, 12, 14, 18, 20, 24, or 30 are allowed. For cluster type geometries the default value is *nangl*= 10 for isotropic cases and *nangl*=12 for specular cases.

dens

real value representing the density of the integration lines (in cm^{-1} for 2-D cases and cm^{-2} for 3-D cases). This choice of density along the plan perpendicular to each angle depends on the geometry of the cell to be analyzed. If there are zones of very small volume, a high line density is essential.

CORN

keyword to specify that the input of the parameters used to treat the corners for the isotropic integration.

pcorn

maximum distance (cm) between a line and the intersection of $n > 1$ external surfaces where track redistributing will take place. Track redistributing will take place if a line comes close to the intersection of $n > 1$ external surfaces. In this case the line will be replicated n times, each of these lines being associated with a different external surface, while its weight is reduced by a factor of $1/n$. This allows for a better distribution of tracks which are relatively close to n external surfaces. By default, there is no treatment of the corners and *pcorn*=0.0.

SYMM

keyword to specify that the geometry has a rotation symmetry.

isymm

integer value describing the rotation symmetry of the geometry. The fixed default of this parameter is 1.

ANNEXE V

FICHIER D'ENTRÉE UTILISANT LE MODULE MCG

```
*DECK ModelExtMCI.x2m
*----
*  Nom            : ModelExtMCG.c2m
*  Type           : Fichier d'entree DRAGON
*  Usage          : Analyse du Benchmark C5G7 Extended
*                   en utilisant MCG
*  Auteur         : G. Marleau
*  Date           : 2003/09/03
*  Modifie        : M. Dahmani 12/07/04
*  Procedure requise  :
*    UNRODDED    : macrolib pour cas UNRODDED
*    RODDEDA     : macrolib pour cas RODDEDA
*    RODDEDB     : macrolib pour cas RODDEDB
*  Fichiers requis
*    flxU        : flux CP pour UNRODDED
*    flxA        : flux CP pour RODDEDA
*    flxB        : flux CP pour RODDEDB
*
*
*----
*  Procedures et Modules
*----
PROCEDURE    UNRODDED RODDEDA  RODDEDB  ;
MODULE       GEO: MCG: DELETE: END: ;
*----
LINKED_LIST  Geometry TRACK MACRO   ;
XSM_FILE     SYS FLUX               ;
SEQ_ASCII    flxU flxA flxB
             flxUmcg flxAmcg flxBmcg  ;
*
REAL      DenTra := 10.0 ;
INTEGER   AngTra := 4 ;
INTEGER     Nsplit NregTot := 1 1000    ;
*
STRING    Itlm ;
EVALUATE Itlm := "ITLM" ;
**STRING   Prll := "STRD" ; !"SPLT", "ANGL", "STRD" , "MCRB"
INTEGER  Merg ;
EVALUATE Merg  := 0 ;
STRING   Etab  ;
EVALUATE Etab := "OFF" ;
STRING   Jacc ;
EVALUATE Jacc  := "JACC"  ;




*----
* Geometry
* 3-D Cartesian Model ( 35 * 35 * 4 )
*  Mixtures required  (reference to tables in proposal)
*
*  Mix     UNRODDED  RODDEDA   RODDEDB
*  1       2a        2a        2a
*  2       2b        2b        2b
*  3       2c        2c        2c
*  4       2d        2d        2d
*  5       2e        2e        2e
*  6       2f        2f        2f
*  7       2f        2f        1
*  8       2f        1         1
*  9       1         1         1
* 10       2g        2g        2g
```

```
*----
Geometry := GEO: :: CAR3D 35 35 4
 X- DIAG X+ VOID Y+ DIAG Y- SSYM Z- SSYM Z+ VOID
*----
* Plane 1
*----
CELL
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
      C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
         CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
            C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
               CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                  C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                        CF C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CF C4 C4 CG C4 C4 CG C3 C2 CMX
                           C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                              C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                                 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                                    C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
                                       CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
                                          C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
                                             C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
                                                C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
                                                   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                      C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                         C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 CMX
                                                            CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 CMX
                                                               C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                  CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                        C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                           CF C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                              C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                    CG C1 C1 CG C1 C1 CMX
                                                                                       C1 C1 C1 C1 C1 CMX
                                                                                          CG C1 C1 C1 CMX
                                                                                             C1 C1 C1 CMX
                                                                                                C1 C1 CMX
                                                                                                   C1 CMX
                                                                                                     CMXY

*----
* Plane 2
*----
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
      C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
         CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
            C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
               CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                  C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                        CF C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CF C4 C4 CG C4 C4 CG C3 C2 CMX
                           C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                              C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                                 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                                    C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
                                       CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
                                          C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
                                             C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
                                                C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
                                                   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                      C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                         C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 CMX
                                                            CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 CMX
                                                               C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                  CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                        C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                           CF C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                              C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                    CG C1 C1 CG C1 C1 CMX
                                                                                       C1 C1 C1 C1 C1 CMX
                                                                                          CG C1 C1 C1 CMX
                                                                                             C1 C1 C1 CMX
                                                                                                C1 C1 CMX
                                                                                                   C1 CMX
                                                                                                     CMXY
```

```
*----
* Plane 3
*----
C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
      C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
         CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
            C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
               CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                  C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                        CF C1 C1 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CF C4 C4 CG C4 C4 CG C3 C2 CMX
                           C1 C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                              C1 C1 C1 C1 C1 C1 C1 C2 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C2 CMX
                                 CG C1 C1 CG C1 C1 C2 C3 CG C4 C4 CG C4 C4 CG C4 C4 CG C4 C4 CG C3 C2 CMX
                                    C1 C1 C1 C1 C1 C2 C3 C3 C3 C4 C4 C4 C4 C4 C4 C4 C4 C4 C3 C3 C3 C2 CMX
                                       CG C1 C1 C1 C2 C3 C3 CG C3 C4 C4 C4 C4 C4 C4 C4 C3 CG C3 C3 C2 CMX
                                          C1 C1 C1 C2 C3 C3 C3 C3 CG C3 C3 CG C3 C3 CG C3 C3 C3 C3 C2 CMX
                                             C1 C1 C2 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C3 C2 CMX
                                                C1 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 C2 CMX
                                                   C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                      C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                         C1 C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 C1 C1 C1 CMX
                                                            CG C1 C1 C1 C1 C1 C1 C1 C1 C1 CG C1 C1 C1 CMX
                                                               C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                  CG C1 C1 CG C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                     C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                        C1 C1 C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                           CF C1 C1 CG C1 C1 CG C1 C1 CMX
                                                                              C1 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                 C1 C1 C1 C1 C1 C1 C1 CMX
                                                                                    CG C1 C1 CG C1 C1 CMX
                                                                                       C1 C1 C1 C1 C1 CMX
                                                                                          CG C1 C1 C1 CMX
                                                                                             C1 C1 C1 CMX
                                                                                                C1 C1 CMX
                                                                                                   C1 CMX
                                                                                                     CMXY
*----
* Plane 4
*----
Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
   Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
      Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cm Cmx
         Cc Cm Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cmx
            Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
               Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cmx
                  Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                     Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                        Cf Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cc Cm Cm Cc Cm Cm Cf Cm Cm Cc Cm Cm Cc Cm Cm Cmx
                           Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                              Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                 Cc Cm Cm Cc Cm Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cmx
                                    Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                       Cc Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cmx
                                          Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cm Cmx
                                             Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                   Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                      Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                         Cm Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cm Cm Cm Cmx
                                                            Cc Cm Cm Cm Cm Cm Cm Cm Cm Cm Cc Cm Cm Cm Cmx
                                                               Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                                  Cc Cm Cm Cc Cm Cm Cc Cm Cm Cc Cm Cm Cmx
                                                                     Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                                        Cm Cm Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                                           Cf Cm Cm Cc Cm Cm Cc Cm Cm Cmx
                                                                              Cm Cm Cm Cm Cm Cm Cm Cm Cmx
                                                                                 Cm Cm Cm Cm Cm Cm Cm Cmx
                                                                                    Cc Cm Cm Cc Cm Cm Cmx
                                                                                       Cm Cm Cm Cm Cm Cmx
                                                                                          Cc Cm Cm Cm Cmx
                                                                                             Cm Cm Cm Cmx
                                                                                                Cm Cm Cmx
                                                                                                   Cm Cmx
                                                                                                     Cmxy

*----
*  Fuel cells:
*  C1 is UO2 Fuel
*  C2, C3, C4 are MOX Fuel
*---
  ::: C1 := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 1 10 ;
  ::: C2 := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 2 10 ;
  ::: C3 := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 3 10 ;
  ::: C4 := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 4 10 ;
*----
* Fission chamber cells:
* CF is fission chamber for plane 1, 2, 3
* Cf is fission chamber for plane 4
*----
  ::: CF := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 5 10 ;
  ::: Cf := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 21.42
    RADIUS 0.0 0.54  MIX 5 10 ;
*----
* Control and guide tube cells
* Plane 1: CG in all control rod cell
* Plane 2: CG in all control rod cell except
*          in central UO2 assembly which contains CC
* Plane 3: CG only in exterenal UO2 assembly
*          central UO2 assembly which contains CD
*          MOX assemble contain CC
* Plane 4: Cc in all control cell location
* CG is always Guide tube  (Mix 6)
* CC is Guide tube for Unrodded and RoddedA
*    and control rod for RoddedB (Mix 7)
* CD is Guide tube for Unrodded
*    and control rod for RoddedA and RoddedB (Mix 8)
* Cc is control rod for all in plane 4 (Mix 9)
*
*----
  ::: CG := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 6 10 ;
  ::: CC := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 7 10 ;
  ::: CD := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 14.28
    RADIUS 0.0 0.54  MIX 8 10 ;
  ::: Cc := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 21.42
    RADIUS 0.0 0.54  MIX 9 10 ;
*----
* Moderator cells
*----
  ::: Cm  := GEO: CARCELZ 1 1
    MESHX 0.0 1.26 MESHY 0.0 1.26 MESHZ 0.0 21.42
    RADIUS 0.0 0.54  MIX 10 10 ;
  ::: CMX := GEO: CAR3D 1 1 1
    MESHX 0.0 21.42 MESHY 0.0 1.26 MESHZ 0.0 14.28
      MIX 10 ;
  ::: CMXY := GEO: CAR3D 1 1 1
    MESHX 0.0 21.42 MESHY 0.0 21.42 MESHZ 0.0 14.28
      MIX 10 ;
  ::: Cmx := GEO: CAR3D 1 1 1
    MESHX 0.0 21.42 MESHY 0.0 1.26 MESHZ 0.0 21.42
      MIX 10 ;
  ::: Cmxy := GEO: CAR3D 1 1 1
    MESHX 0.0 21.42 MESHY 0.0 21.42 MESHZ 0.0 21.42
      MIX 10 ;
  ;

*----
* Get Macroscopic XS
* UNRODDED Configuration
*----
MACRO := UNRODDED ;
**FLUX := flxU ;
TRACK FLUX := MCG: Geometry MACRO ::
    TYPE K
    ETAB <<Etab>>
**    THER 200 0.0005 EXTE 200 0.0001
    CURR DIRT <<Itlm>>
    MAXR 4900
    TRAK TISO <<AngTra>> <<DenTra>>  ;

flxUmcg := FLUX ;
TRACK MACRO FLUX := DELETE: TRACK MACRO FLUX ;
**
*----
* Import Macroscopic XS
* RODDED A Configuration:
*----
MACRO := RODDEDA ;
FLUX := flxA ;
TRACK FLUX := MCG: FLUX Geometry MACRO ::
    TYPE K
    ETAB <<Etab>>
**    THER 200 0.0005 EXTE 200 0.0001
    <<Itlm>>
    MAXR 4900
    TRAK TISO <<AngTra>> <<DenTra>>  ;

flxAmcg := FLUX ;
TRACK MACRO FLUX := DELETE: TRACK MACRO FLUX ;
END: ;
*----
* Import Macroscopic XS
* RODDED B Configuration:
*----
MACRO := RODDEDB ;
FLUX := flxB ;
TRACK FLUX := MCG: FLUX Geometry MACRO ::
    TYPE K
    ETAB <<Etab>>
**    THER 200 0.0005 EXTE 200 0.0001
    <<Itlm>>
    MAXR 4900
    TRAK TISO <<AngTra>> <<DenTra>>  ;

flxBmcg := FLUX ;
TRACK MACRO FLUX := DELETE: TRACK MACRO FLUX ;
*
Geometry := DELETE:  Geometry ;
END: ;
QUIT "LIST" .
```