Université de Montréal

Méthodes Itératives Parallèles
Applications en Neutronique et en Mécanique des Fluides

par

Abdessamad Qaddouri

Département de Physique

Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Physique

Décembre, 1997

Université de Montréal

Faculté des études supérieures

Cette thèse intitulée:

"Méthodes Itératives Parallèles
Applications en Neutronique et en Mécanique des Fluides"

présentée par:

Abdessamad Qaddouri

a été évalué par un jury composé des personnes suivantes:

Monsieur Andrei Malevsky,	président-rapporteur
Monsieur Bernard Goulard,	directeur de recherche
Monsieur Robert Roy,	codirecteur de recherche
Monsieur Guy Marleau,	membre du jury
Monsieur Ben Rouben,	examinateur externe

Thèse acceptée le: 28 Avril 1998

Sommaire

Dans cette thèse, le calcul parallèle est appliqué successivement à la neutronique et à la mécanique des fluides. Dans chacune de ces deux applications, des méthodes itératives sont utilisées pour résoudre le système d'équations algébriques résultant de la discrétisation des équations du problème physique. Dans le problème de neutronique, le calcul des matrices des probabilités de collision (PC) ainsi qu'un schéma itératif multigroupe utilisant une méthode inverse de puissance sont parallélisés. Dans le problème de mécanique des fluides, un code d'éléments finis utilisant un algorithme itératif du type GMRES préconditionné est parallélisé.

Cette thèse est présentée sous forme de six articles suivis d'une conclusion. Les cinq premiers articles traitent des applications en neutronique, articles qui représentent l'évolution de notre travail dans ce domaine. Cette évolution passe par un calcul parallèle des matrices des PC et un algorithme multigroupe parallèle testé sur un problème unidimensionnel (article 1), puis par deux algorithmes parallèles l'un mutirégion l'autre multigroupe, testés sur des problèmes bidimensionnels (articles 2-3). Ces deux premières étapes sont suivies par l'application de deux techniques d'accélération, le rebalancement neutronique et la minimisation du résidu aux deux algorithmes parallèles (article 4). Finalement, on a mis en oeuvre l'algorithme multigroupe et le calcul parallèle des matrices des PC sur un code de production DRAGON où les tests sont plus réalistes et peuvent être tridimensionnels (article 5). Le sixième article (article 6), consacré à l'application à la mécanique des fluides, traite la parallélisation d'un code d'éléments finis FES où le partitionneur de graphe METIS et la librairie PSPARSLIB sont utilisés.

Remerciements

Je veux tout d'abord remercier de façon très particulière mon directeur de recherche, Bernard Goulard, pour ses judicieux conseils qui m'ont été très utiles tout au long de mon doctorat. J'ai pu constater combien il est soucieux de la réussite de ses étudiants, ce qui le rend encore plus apprécié de ces derniers.

Également, j'aimerais profiter de l'occasion pour souligner la bonne collaboration, l'enthousiame et l'ambiance amicale dont mon codirecteur de recherche, Robert Roy, a fait preuve.

Mon ami Michel Mayrand a été mon initiateur au calcul parallèle. Je le remercie de son intérêt soutenu pour ce travail.

Merci à Azzeddine Soulaimani, de l'école de technologie supérieure de Montréal, pour son enthousiasme, ses explications et son souci du détail.

Je tiens à remercier le professeur Yousef Saad, du département d'informatique de l'Univesité de Minnesota, pour sa disponibilité ainsi que pour les connaissances qu'il m'a fait partager avec lui en toute modestie.

Enfin, j'aimerais remercier mes parents pour leur soutien, et cela à tous les niveaux, tout au long de mes études universitaires.

Table des matières

liste des symboles

NOTE: Ici on a une liste des symboles et sigles apparaissant dans l'introduction et la conclusion de cette thèse. Ils sont représentés ainsi que leurs significations respectives, dans l'ordre où ils apparaissent dans le texte.

Symbole ou Sigle	**Définition**
GMRES	Generalized Minimal Residual
PVM	Parallel Virtual Machine
MPI	Message Passing Interface
PC	Probabilités des collisions
CI	Courant d'Interface
Φ^g	Flux des neutrons d'énergie g
P^g	Probabilités des collisions des neutrons d'énergie g
Q^g	Source des neutrons transférés à une énergie g
$\Sigma_s^{g \leftarrow g}$	Section efficace de transfert du groupe g dans lui même
V	Volumes des régions spatiales
ngrp	Nombre total de groupes d'énergie
nreg	Nombre total de régions spatiales
$\Sigma_s^{g \leftarrow g'}$	Section efficace de transfert de l'énergie g' à l'énergie g
S_f^g	Source de fission
Σ_f^g	Section efficace de fission pour des neutrons d'énergie g
χ	spectre d'énergie des neutrons de fission
ν	Nombre moyen des neutrons produits par fission
K_{eff}	Facteur de multiplication effectif
{U}	Vecteur global des degrés de liberté
{R}({U})	Vecteur de sollicitations dépendant de {U}
J({U})	Matrice jacobienne dépendante de {U}
z	Vecteur solution dans le sous-espace K_m
K_m	Sous-espace de Krylov de Dimension m
r_0	Vecteur résidu initial
σ	Scalaire réel assez petit
M	Matrice de préconditionnement
M^{-1}	Matrice inverse de M
U	Matrice triangulaire supérieure
L	Matrice triangulaire inférieure
ILUT	Incomplete LU factorization with Threshold

Introduction

Les besoins croissants en puissance de calcul ont conduit à développer des ordinateurs à architecture parallèle et des algorithmes adaptés pour tirer parti de ces nouvelles architectures. Dans la plupart des applications scientifiques, les méthodes itératives sont utilisées, et on n'est pas surpris de voir ces méthodes au premier rang de celles mises en oeuvre sur des ordinateurs parallèles. En plus de nécessiter un minimum de stockage, ce qui représente leur principal avantage, les méthodes itératives sont faciles à mettre en oeuvre sur des architectures parallèles. Deux approches sont traditionnellement utilisées dans le développement des techniques itératives parallèles. La première fait ressortir le parallélisme présent dans les algorithmes séquentiels standards. Cette approche est relativement simple puisque l'algorithme parallèle n'a pas été changé en profondeur par rapport à son équivalent séquentiel. La seconde approche revient à développer de nouveaux algorithmes où le parallélisme est à son maximum.

Dans la présente thèse, le calcul parallèle est appliqué successivement à la neutronique et à la mécanique des fluides. Dans chacune de ces deux applications, des méthodes itératives sont utilisées pour résoudre le système d'équations algébriques résultant de la discrétisation des équations du problème physique. Dans le problème de neutronique, un schéma iteratif multigroupe utilisant une méthode inverse de puissance [1] est parallélisé. Dans le problème de mécanique des fluides, on applique la version parallèle de l' algorithme itératif GMRES [2] réalisé spécialement pour ce genre d'applications. Les communications entre processeurs ont été réalisées en utilisant PVM [3] pour l'application en neutronique et MPI [4] pour l'application en mécanique des fluides. Pour chacune de ces deux applications, nous allons d'abord résumer brièvement la nature du problème et montrer pourquoi le recours au calcul parallèle s'impose, puis nous procéderons à un survol des différentes étapes accomplies dans le cadre du parallélisme.

Le coeur d'un réacteur nucléaire est généralement composé d'un réseau ordonné de grappes de matériel fissile d'où sont issus les neutrons. En général, l'énergie de fission est extraite par un fluide caloporteur servant à refroidir les grappes. Un réflecteur

est disposé autour du réseau, entre autres, afin de minimiser les fuites des neutrons à l'extérieur du coeur. Différents mécanismes de réactivité, notamment des barres d'arrêt, sont également nécessaires afin de contrôler la réaction en chaîne.

Dans le but d'obtenir les données sur les propriétés physiques nécessaires aux simulations numériques, elles-mêmes essentielles au suivi d'un réacteur nucléaire, il est d'usage de définir une cellule unitaire qui regroupe l'information géométrique et matérielle pertinente au réseau de grappes combustibles. La migration et l'interaction neutron-matière dans ce milieu physique (cellule), est régie par l'équation de transport neutronique [5], qui n'est autre qu'une forme linéaire adaptée de l'équation cinétique des gaz développée par Boltzmann [6]. La résolution de l'équation de transport donne le flux neutronique à l'intérieur de la cellule. Cette distribution de flux est ensuite utilisée afin d'homogénéiser en espace et condenser en énergie les taux de réaction neutronique. Les noyaux lourds des grappes subissent des transmutations causées par l'irradiation du combustible nucléaire. Le calcul d'une cellule regroupe ainsi toute une séquence de calculs de flux, chacune de ces distributions influençant le champ des noyaux de la distribution suivante. Les propriétés nucléaires condensées et homogénéisées peuvent être utilisées dans un calcul de la distribution du flux dans tout le coeur en résolvant, par exemple, l'équation de diffusion [7].

La discrétisation spatiale de la cellule s'effectue en décomposant la géométrie fondamentale en nombre fini de régions de composition matérielle homogène. La discrétisation énergétique est essentiellement régie par le nombre de groupes d'énergie utilisés lors de la définition des librairies de sections efficaces microscopiques, regroupant les données isotopiques pertinentes aux diverses interactions neutroniques. Ces librairies sont elles-mêmes produites en compactant l'information issue d'expériences, où on a exposé différentes cibles à des faisceaux de neutrons.

La résolution du flux s'effectue en deux temps. Dans une première étape, un calcul des probabilités de première collision est effectué en considérant des neutrons d'une certaine énergie migrant dans la cellule. L'équation de transport est alors réduite en un système linéaire où le couplage entre les groupes d'énergie s'effectue par les sections efficaces de transfert.

Typiquement, la plupart des calculs actuels sont effectués avec des nombres de

régions et nombre de groupes d'énergie de l'ordre de la centaine chacun. Ce qui fait qu'un flux neutronique est obtenu en résolvant itérativement un système linéaire comportant quelques dizaines de milliers d'inconnues. Les calculs de cet ordre sont encore réalisables sur les stations de travail actuelles. À terme, il est toutefois prévisible qu'afin de valider les données nucléaires de sections efficaces, il faille accroître substantiellement le nombre de groupes d'énergie, qui passerait alors d'une centaine à quelques dizaines de milliers. Dans ces conditions, il n'est plus possible d'envisager l'exécution du traitement sur un mono-processeur.

Les premières applications du parallélisme en neutronique , réalisées par Turinsky et son équipe [8], ont porté sur la résolution de l'équation de diffusion en partitionnant le domaine spatial en sous-domaines et en départageant ces derniers entre les processeurs. Quant à l'équation de transport, les premières tentatives de parallélisation de sa forme intégrale ont été réalisées par Vujic et son équipe [9, 10] dans le cadre de la méthode des courants d'interface (CI) [11]. Cette méthode est utilisée pour calculer la distribution du flux à l'intérieur des assemblages (dizaines de cellules) où des conditions de continuité des courants aux interfaces des assemblages sont respectées. Dans cette application [9, 10], où le nombre de groupes d'énergie considéré est petit et ne dépasse pas généralement une douzaine, les assemblages sont départagés entre les processeurs et les courants aux interfaces constituent la donnée principale à se communiquer entre processeurs.

Les articles (1-5) sont consacrés à la résolution numérique parallèle de l'équation intégrale de transport. Cette résolution, basée sur une méthode de probabilités de collision (PC) et un formalisme multigroupe, est utilisée pour calculer la distribution du flux dans une cellule unitaire [12] où le nombre de groupes d'énergie peut être assez grand. Après discrétisation spatiale et énergétique, l'équation de transport est représentée par:

$$[\mathbf{V} - \mathbf{P}^{\mathbf{g}}\boldsymbol{\Sigma}_{\mathbf{s}}^{\mathbf{g}\leftarrow\mathbf{g}}]\boldsymbol{\Phi}^{\mathbf{g}} = \mathbf{P}^{\mathbf{g}}\mathbf{Q}^{\mathbf{g}} \qquad ; \mathbf{g} = \mathbf{1}, \mathbf{ngrp} \tag{0.1}$$

Cette équation a pour seul inconnue le vecteur flux intégré $\boldsymbol{\Phi}^{\mathbf{g}}$ relié à tous les résultats recherchés dans la majorité des calculs de transport. La matrice des probabilités de collision d'un groupe d'énergie donné $\mathbf{P}^{\mathbf{g}}$ est symétrique et elle est de

dimension **nreg** $*$ **nreg** où **nreg** est le nombre total de régions spatiales. Les coefficients de la matrice $\mathbf{P^g}$ sont en fait proportionnels aux probabilités de passage du neutron d'une région à une autre. **ngrp** est le nombre total de groupes d'énergie, $\mathbf{\Sigma_s^{g\leftarrow g}}$ est la matrice des sections efficaces de transfert du groupe **g** dans lui même et $\mathbf{Q^g}$ est le terme source des neutrons provenant des groupes d'énergie différents de **g**.

D'après l'équation (0.1), un partitionnement basé sur le groupe d'énergie semble approprié. En effet, les matrices $\mathbf{P^g}$ de même que les flux $\mathbf{\Phi^g}$ sont calculés groupe par groupe. Les calculs des matrices des probabilités de collision sont indépendants d'un groupe à l'autre, et aucune communication entre processeurs n'est nécessaire lors d'un calcul parallèle des $\mathbf{P^g}$. Par contre, le calcul des flux nécessite de la communication puisque le terme source $\mathbf{Q^g}$ contient des termes de transfert par diffusion ou par fission provenant des autres groupes d'énergie. Le terme source $\mathbf{Q^g}$ peut être exprimé explicitement en fonction des flux de la façon suivante:

$$Q_i^g = [\sum_{g'>g} \Sigma_{s,i}^{g\leftarrow g'} \phi_{g'} + \sum_{g'>g} \Sigma_{s,i}^{g'\leftarrow g} \phi_{g'} + S_{f,i}^g] \quad i = 1, nreg \tag{0.2}$$

où $\mathbf{S_f}$ est le vecteur représentant les fissions:

$$S_{f,i}^g = \frac{\chi_i^g}{K_{eff}} \sum_{g'} \nu \Sigma_{fi}^{g'} \phi_i^{g'}. \tag{0.3}$$

En fait, chaque processeur sera responsable d'un sous-ensemble de groupes d'énergie. En supposant des conditions initiales sur la source de fission, chaque processeur contribue à la résolution du système global (Eqs.0.1-0.3). On utilise une méthode de puissance (articles 1-5) composée de deux types d'itérations: des itérations internes pour le calcul de la distribution du flux, et des itérations externes pour le calcul des termes sources. Après les itérations internes, le code parallèle exécute une étape de communication où chaque processeur communique à tous les autres ses valeurs de flux récentes produites par ses itérations internes. Chacun des processeurs, ayant en sa possession toutes les données nécessaires, recalcule son terme source. Les itérations internes sont accélérées par des techniques de rebalancement neutronique et de minimisation du résidu (article 3-5). Cet algorithme itératif multigroupe parallèle

s'arrête quand le facteur de multiplication effectif K_{eff} ainsi que la distribution de flux convergent.

Dans l'algorithme de recomposition de domaine appelé multirégion (voir articles 2-5), les régions de l'espace sont départagées entre les processeurs, chacun d'eux traitant donc un sous-domaine du domaine global. Une région n'appartient qu'à un et un seul sous-domaine, et de ce fait, les sous-domaines ne se chevauchent pas. La solution sur un sous-domaine n'est recalculée que quand le cycle de calcul des solutions sur tous les sous-domaines est terminé. Cependant, au niveau algorithmique, chaque sous-domaine aura besoin de toutes les solutions correspondant à toutes les régions dans tous les autres sous-domaines pour recalculer une nouvelle solution. Ce type de chevauchement algorithmique complet est dû à la méthode adoptée, c'est-à-dire ici la technique des probabilités de collision. Cette méthode permet à un neutron de voyager entre deux régions spatiales géométriquement éloignées. Cela entraîne un couplage entre les solutions de flux sur toutes les régions spatiales et donc tous les sous-domaines.

Quand on partitionne un domaine spatial en sous-domaines dans un traitement parallèle, il est fréquent d'utiliser des représentations de graphe [13]. Certaines restrictions sur le type de partitionnement s'imposent selon le type de problème. Par exemple, dans les techniques d'éléments finis où les calculs sont exécutés élément par élément, il est facile de départager entre les processeurs (sous-domaines) les éléments plutôt que les noeuds. On appelle ce partitionnement un partitionnement basé sur l'élément. Les techniques de volumes finis utilisent plutôt un partitionnement basé sur l'arête, alors que dans les techniques de différences finies, c'est le partitionnement basé sur le noeud qui est adopté en général. Les valeurs aux interfaces peuvent être obtenues en employant des méthodes qui altèrent les calculs entre les sous-domaines. Dans ces méthodes, on résoud à chaque itération un nouveau problème avec des conditions aux interfaces données par les récentes solutions sur les sous-domaines.

En ce qui concerne l'application à la mécanique des fluides, nous avons procédé à la parallélisation d'un code d'éléments finis pour la simulation d'écoulements tridimensionnels avec surface libre pour un fluide incompressible, travail présenté dans notre article 6. Cette technique a été appliquée à la modélisation des écoulements

et du niveau d'eau dans les barrages [14]. Le problème consiste à résoudre, par une méthode d'éléments finis, des équations de conservation (masse et quantité de mouvement) couplées avec une équation décrivant l'évolution de la surface libre. Après discrétisations spatiale et temporelle, le problème revient à résoudre le système non-linéaire suivant:

$$\{R\}(\{U\}) = 0 \tag{0.4}$$

où { R} est un opérateur non linéaire de $\mathbb{R}^N$ dans $\mathbb{R}^N$. $\{U\}$ est le vecteur de l'ensemble des degrés de liberté (d.d.l.).

La méthode de Newton-Raphson utilisée pour trouver une solution du problème non-linéaire de l'Eq.(0.4), consiste à résoudre à chaque pas de temps le système suivant:

$$\mathcal{J}(\{U\}^{n,i-1})\{\delta U\}^{n,i} = -\{R\}(\{U\}^{n,i-1}) \tag{0.5}$$

où $\{U\}^{n,i-1}$ est le vecteur global des degrés de liberté, au pas de temps n et à l'itération $i-1$ de Newton, et $\mathcal{J} = \frac{\partial R}{\partial U}$ est la matrice jacobienne creuse, mais non symétrique.

On utilise l'algorithme GMRES préconditionné pour résoudre le système de l'Eq. (0.5) qui est linéaire en $\{\delta U\}^{n,i}$. GMRES fait partie des techniques itératives les plus utilisées pour résoudre les grands systèmes linéaires creux non symétriques. Ces techniques sont basées sur un processus de projection, orthogonal ou oblique, sur des sous-espaces de Krylov [15]. GMRES consiste à chercher une solution approchée $\{\delta U\}_m = \{\delta U\}_0 + z$ dans l'espace affine $\{\delta U\}_0 + K_m$. $\{\delta U\}_0$ est la solution initiale et K_m est le sous-espace de Krylov de dimension m:

$$K_m = span\{r_0, \mathcal{J}r_0, \mathcal{J}^2 r_0, ..., \mathcal{J}^{m-1} r_0\} \tag{0.6}$$

où $r_0 = -\{R\} - \mathcal{J}\{\delta U\}_0$ est le vecteur résidu initial. En fait, GMRES n'exige que le produit de la matrice jacobienne par un vecteur (et non le calcul explicite de cette matrice). Cela permet d'éviter de stocker la matrice et considérer plutôt son action sur ce vecteur. Pour le système linéaire à résoudre à chaque pas de Newton, on substitue l'action de $\mathcal{J}$ sur un vecteur $\{v\}$ par l'approximation suivante:

$$\mathcal{J}(\{U\}^{n,i-1})\{v\} \approx \frac{\{R\}(\{U\}^{n,i-1} + \sigma\{v\}) - \{R\}(\{U\}^{n,i-1})}{\sigma} \tag{0.7}$$

où σ est un scalaire réel assez petit.

Le système à résoudre par GMRES est d'abord préconditionné. Généralement, le préconditionnement améliore la convergence, et consiste à remplacer le système original par un système équivalent. Par exemple pour un préconditionnement gauche, le système équivalent a la forme suivante:

$$M^{-1}\mathcal{J}(\{U\}^{n,i-1})\{\delta U\}^{n,i} = -M^{-1}\{R\}(\{U\}^{n,i-1}) \tag{0.8}$$

où M est la matrice de préconditionnement. Dans FES, une factorisation incomplète ILUT [16] est appliquée à la matrice M. ILUT ressemble à un LU standard où M est mise sous la forme $M = LU$. Cependant, selon deux paramètres $lfil$ et *droptol* (voir article 6), on ne retient dans ILUT que quelques coefficients par ligne dans L et U.

Nous utilisons cette technique dans le code d'éléments finis FES (article 6) pour simuler un écoulement tridimensionnel avec un surface libre dans le barrage Sainte-Marguerite dans la région de Québec. Ce genre de simulations tend à remplacer les techniques expérimentales dont la réalisation est souvent coûteuse en ressources matérielles. Pour une simulation de ce genre, on a eu besoin d'environ 30 millions de mots réels en mémoire vive. Sur Sun-Entreprise, la simulation de dix pas dans le temps de 0.1 secondes chacun nécessite 3 heures de temps réel, et le nombre d'inconnues est de l'ordre de 110 milles. On voit qu'il s'agit bien d'un problème moyennement grand et une tentative de le paralléliser se justifie pleinement.

La méthode des éléments finis comporte plusieurs parties où l'on peut faire ressortir du parallélisme. D'un point de vue géométrique, le domaine pour lequel on veut résoudre le problème aux dérivées partielles est discrétisé en éléments. Une source de parallélisme est donc de travailler sur une décomposition de ce dernier, c'est-à-dire de faire travailler chaque processeur sur un sous-ensemble des éléments. Du point de vue algorithmique, la méthode des éléments finis fait intervenir la résolution des systèmes d'équations, linéaires ou non, qui compose la quasi-totalité du travail à effectuer. Dans le cas d'équations non-linéaires, les algorithmes de résolution font intervenir une suite de systèmes linéaires à résoudre (Eq. (0.5)). Les algorithmes réalisés spécialement pour résoudre ce genre de système font intervenir des opérations parallélisables. On peut citer par exemple le produit d'une matrice par un vecteur, le produit scalaire

de deux vecteurs et le calcul de la norme d'un vecteur. En pratique on fait appel à des algorithme parallèles disponibles dans des librairies telles que PSPARSLIB [17], PETS [18] ou autres.

Nous avons adopté un partitionnement basé sur l'élément où, après avoir associé un graphe au domaine géométrique (en assimilant les éléments spatiaux et les sommets du graphe), nous avons fait appel à METIS [19] afin de subdiviser d'une façon optimale ce graphe en sous-graphes. METIS minimise le nombre d'arêtes reliant les différents sous-graphes, ce qui en principe minimise le nombre de communications entre les sous-domaines (processeurs).

Une fois les éléments partagés entre les processeurs par l'intermédiaire de METIS. des routines de PSPARSLIB sont utilisées pour réordonner les vecteurs (ou matrices). Chaque vecteur (ou matrice) sur chaque processeur va être réordonné de façon à ce que ses composantes correspondant à des équations (degrés de liberté) non partagées viennent au début suivies des composantes correspondant à des équations partagées. Chaque processeur reconnait, par l'intermédiaire de pointeurs, ses processeurs adjacents et l'endroit où commencent sur tout vecteur (ou matrice) les équations partagées avec un processeur adjacent donné. La librairie PSPARSLIB met donc à notre disposition des outils qui nous permettent quand c'est nécessaire de faire de façon efficace des échanges de données entre les processeurs.

La parallélisation du préconditionneur en approximant la matrice de préconditionnement par ces blocs diagonaux, a déjà été réalisée par Dutto, Habashi et Fortin [20]. La matrice est calculée par un seul processeur qui départage les blocs diagonaux entre les processeurs, et chaque bloc constitue le préconditionneur pour le sous-système traité par le processeur. Ce type de parallélisme du préconditionneur a été appliqué pour un écoulement bidimensionnel d'un fluide compressible [20].

Nous avons appliqué (voir article 6) un préconditionnement par sous-domaine. La matrice de préconditionnement est formée, localement sur chaque processeur, en assemblant seulement les matrices élémentaires correspondant aux éléments appartenant au sous-domaine traîté par ce processeur. Une factorisation ILUT est appliquée localement sur la matrice de préconditionnement. Ce type de préconditionnement local, selon son efficacité, va être très utile pour le résolution de gros problèmes où

la grosseur de la matrice de préconditionnement peut poser un problème d'espace mémoire.

Chaque processeur, forme sa part (sous-système) du système global à résoudre. En fait un sous-système sur un processeur donné va traiter les équations correspondant aux noeuds des éléments appartenant au sous-domaine traité par ce même processeur. Le sous-système est préconditionné et ensuite résolu en utilisant un algorithme parallèle GMRES où les opérations produit scalaire et norme sont parallélisées.

ARTICLE 1

Multigroup Flux Solvers on Parallel Architecture

15^{TH} Annual Conference Canadian Nuclear Society 1994

envoyé en Janvier 1994, accepté en Avril 1994 et publié en Juin 1994

MULTIGROUP FLUX SOLVERS ON PARALLEL ARCHITECTURE

A. Qaddouri [1,2], R. Roy [1], G.Marleau [1], B. Goulard [2] and M. Mayrand [2]

[1] *Institut de Génie Nucléaire, École Polytechnique de Montréal*
Montréal, Québec, Canada H3C 3A7

[2] *Centre de Recherches Mathématiques, Université de Montréal*
Montréal, Québec, Canada H3C 3J7

Abstract — *Using a cluster of processors with a distributed memory, parallelization for multigroup flux computations is investigated. The parallel performance will be studied in cases where the neutron transport equation is discretized using the collision probability technique. Particular techniques pertinent to the two-step process of solving a multigroup linear equation, as well as a fixed source solver, are described. Parallelization is achieved by distributing different energy groups on a set of workstations using PVM. Typical run times will be provided for 1-D CANDU cell test case. Finally, conclusions are drawn for realistic applications of parallel multigroup flux solvers.*

I. INTRODUCTION

Computer simulations play a crucial rôle in the study of large-scale technological systems. The increase in the complexity of such systems and the accuracy demanded of modelling lead to algorithms of ever increasing complexity. It then becomes necessary to investigate new computer technologies in order to keep computation time at reasonable level.

One solution is parallel computing which is a way of reorganizing the structure of computer so that it can do many things simultaneously. Parallel processing has already been applied successfully to several fields of engineering and science (aerodynamics, geophysics, meteorology, etc.). It is commonly accepted that, in coming years, powerful computing machines will consist of many processors connected by high speed message exchange network. Small multi-computers are already quite common in industry research centers and universities. However, many challenges remain with regard to algorithm languages and even standards to evaluate multi-processors performance.

How the multigroup neutron transport equation can be solved for an array of regions as in 2-D cluster cells, as well as 3-D supercells, will be discussed. The multigroup transport equation is generally solved using iteration schemes where the flux solutions of a set of one-group problems are combined via their source terms.

Since the multigroup transport equation using the collision probability method is suitable for parallelization, this paper investigates how the sequential code can be parallelized while minimizing the modifications to the initial code and hardware investment. In this spirit, the parallel multigroup solution

of transport equation using PVM (Parallel Virtual Machine)[1, 2], a public domain software package for parallelization on a network of workstations, is presented. Parallelization over the energy groups as well as other iterative schemes are examined.

Section II presents a detailed description of the parallel environment PVM and comparison with other alternatives. Section III describes the multigroup neutron transport equations. In Section IV, the main steps for the parallelization of the equations are described. Performance results are given in Section V. Finally, conclusions are drawn in the last section.

II. PARALLELISM

In this section, key parameters characterizing parallel computer performance will be reviewed, and will be used as a grid for a brief overview of the various presently available classes of parallel systems. Among them, the PVM software package will be the tool selected for the present parallelization task on DRAGON [3].

Speed up, which is a common measure of the performance gain due to the use of a multi-processor, is defined as the ratio of the time required to complete the job with one processor to the time required to complete the same job with N processors. A good speed up (as close to N as possible) implies that processors experience minimal delays in exchanging information. The approach to the breaking up of a problem in order to run it on a multiprocessor system depends on the architecture of that system. Beside speed up, cost effective computation and skillfull work are also matters of importance when evaluating parallel system.

Among the growing number of new architectures, the following characteristics are a helpful classification guide: shared or distributed memory, coarse or fine grain, SIMD (Single Instruction, Multiple Data) or MIMD (Multiple Instruction, Multiple Data)(see Appendix). The various trade-off between the qualities of multi-computers have led to the following broad classes of parallel computers:

1. Massively parallel processing (fine grained) where thousands of simple, moderately fast, special purpose processors usually operate in single-user mode. For example MasPar in banks of 1,024 processors up to 16,384; Thinking Machines (Connection Machine CM-1 through 5) with typically 2^{16} processors. Expertise is available in large government laboratories and few universities but they are rather expensive.

2. "Mainframe" parallel processing supercomputer, which are popular in universities, laboratories because of their versatility, multi-user possibilities and large software and infrastructure suport. Examples are: Parallel unicomputers based on functional parallelism and pipelining (CRAY 1), Multipurpose MIMD with up to 8 units with vector processors: CRAY X,Y series, NEC -SX series... The market is very competitive as exemplified by the demise of ETA systems of CDC despite a large financial commitment.

3. Multi-processing platforms. The user distributes specific tasks on a few powerful general purpose processors in order to increase performance, particularly in the high-lelvel workstation and graphics domains. They are very popular amongst a highly specialized workforce, such as aeronautical design, satellite imaging personnel... They are easily upgradable and can be used as a powerful node in a heterogeneous architecture as discussed in 4.). Examples are: Apollo/Hewlett-Packard workstations offering 4 CPUs which are RISC chips on a VME bus, Silicon Graphics providing up to 24 processors and promoting a line from a simple deskside graphics workstation to a graphics supercomputer, Sun SPARC station 10, 20 series with 1 to 4 processors.

4. Parallel Virtual Machines. The network heterogeneity already present in large institutions has led to a tool that distributes user problems across diverse architectures. PVM enables a user to define

a heterogeneous networked collection of serial, parallel and vector computers to function as one large computer. Because of the pratical importance of cost-effectiveness and portability between institutions, more than 100 institutions worlwide have already implemented PVM on a variety of CPUs. However, the following limitations must be kept in mind: workstations have to communicate through buses so that when the number of processors increases, bus contention becomes severe and limits the speed of the system; furthermore, the management of load balancing among heterogeneous computers is more complex than it is for a MIMD dedicated machine (with identical processors).

Installation of PVM is easy and does not necessarily require the intervention of the superuser. A plain access or account on different workstations is sufficient for a user's purpose. A parallel code is compiled using the standard compiler (FORTRAN and C) for the workstation and is linked with the PVM library. The executable is stored in a specific user directory where PVM will look for programs to spawn. In a heterogeneous environment, the program is compiled with the various compilers (corresponding to each different CPU) and then stored in a separate directory.

Once an architecture has been set up, two kinds of parallel programming are possible. First, the host-node programming: one processor runs a program which controls and interacts with another code running on the other nodes. The host processor is usually the only one capable of accessing I/O (disk, screen, etc.); communication with other nodes goes by message-passing. Second, in what is called SPMD (Single Program Multiple Data, not to be mistaken with SIMD), the same code is replicated on all nodes, each of them having access to standard I/O. The node identifier is sufficient to produce different behaviour appropriate to different processors or, more often, to divide the global task into smaller ones.

For the present investigation which aims at the parallelization of DRAGON in a cost effective way and with a minimal change from sequential version, a network of Sun SPARK workstations has been selected for hardware. The SPMD programming method, that requires less modification than does the host-nodes choice, has been adopted. In summary, the sequential code undergoes relatively few changes: inclusion of message-passing routines, adaptation of the variables to be transferred by these routines and some load balancing. In the latter part, a special effort is in order to give the same amount of work to each processor in order to avoid that one node, which has been given the longest task, delays other processors. Such a situation would create a sequential type behaviour limiting the parallel performance.

III. MULTIGROUP TRANSPORT THEORY

How the multigroup neutron transport equation can be solved will be discussed here. The DRAGON cell code[3] attempts to produce a consistent solution using the collision probabilities (CP) method to a static problem of the form:

$$\begin{aligned} &\vec{\Omega}\cdot\vec{\nabla}\Phi(\vec{r},\vec{\Omega},E)+\Sigma(\vec{r},E)\Phi(\vec{r},\vec{\Omega},E)= \\ &\int_0^\infty dE'\int_{4\pi}d^2\Omega'\,\Sigma_s(E\leftarrow E',\vec{\Omega}\leftarrow\vec{\Omega}')\,\Phi(\vec{r},\vec{\Omega}',E')+ \\ &\frac{\chi(\vec{r},E)}{4\pi}\int_0^\infty dE'\nu\,\Sigma_f(\vec{r},E')\int_{4\pi}d^2\Omega'\,\Phi(\vec{r},\vec{\Omega}',E')\quad , \end{aligned} \tag{1}$$

where the usual macroscopic cross sections (Σ, Σ_s and Σ_f) are formed by combining the microscopic cross sections of various isotopes.

Here, the standard conditions of most cell calculations [4] will be assumed:

1. the energy spectrum is split into G energy groups;

2. the scattering kernels are assumed to be isotropic, so that all neutron sources are also isotropic;

3. and the cell or supercell boundary conditions are reflexive.

In this paper, the concern is a multigroup and multiregion problem and the implementation of a flux solver strategy on parallel architectures. Instead of directly using the DRAGON system which contains about 100K lines of FORTRAN code, a small application that can accurately treat slab assemblies in a multigroup treatement has been constructed. The idea behind this restriction is to get rid of the memory management of large test cases and to access only the main computational features of a cell code. In order to calculate the CP, we have considered a model that uses specular boundary conditions outside the slab assembly [5]. The one-group integro-differential Eq. (1) may then be written in a simplified integral form:

$$\Phi^g(z^{'},\mu) = \int_{-\infty}^{\infty} \frac{dz}{|\mu|} \Psi^g(z,\mu) H(\mu(z^{'}-z)) e^{-\tau^g(z,z^{'})/|\mu|} \tag{2}$$

where $\Phi^g(z^{'},\mu)$ is the neutron flux in group g at a point $z^{'}$ of the slab, μ represents the cosine of the polar angle, and $H(.)$ is the usual step function. Here, the optical path between z and $z^{'}$ in group g is $\tau^g(z,z^{'})$. Neutron sources in group g and coming from a point z are collected into the $\Psi^g(z,\mu)$ term. Let us now partition the original slab into L zones

$$0 = z_0 < ... < z_{i-1} < z_i < ...z_L = H \tag{3}$$

each zone being defined such that it has homogeneous material properties with total cross-section Σ_i^g and width $V_i = z_i - z_{i-1}$. Assuming reflective boundary conditions, the flux in Eq.(2) will exhibit a $2H$ period in the z-coordinate. In this simplified transport model, the group-dependent reduced and volume-integrated first-flight CP is computed using:

$$P_{ji}^g = \int_{z_{i-1}}^{z_i} dz \int_{z_{j-1}}^{z_j} dz^{'} G^g(z^{'} \leftarrow z) \tag{4}$$

where $G^g(z^{'} \leftarrow z)$ is the symmetric transport kernel assuming an isotropic slab source at z.

Once the CP matrices have been computed in every energy group, the transport equation is discretized into the form of a linear system of equations:

$$V_j \Phi_j^g = \sum_{i=1}^{L} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{\mathrm{k}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \}, \tag{5}$$

with symmetric CP matrices $\mathbf{P^g}$ of order L. In order to solve this eigenvalue problem for k and the fluxes, multigroup iteration schemes are generally implemented using the inverse power method:

$$\mathrm{A}\vec{\phi}^{(n+1)} = V_j \Phi_j^{g(n+1)} - \sum_i P_{ji}^g \{ \sum_{g'} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n+1)} \} =$$

$$\frac{1}{\mathrm{k}^{(n)}} \mathrm{B}\vec{\phi}^{(n)} = \sum_i P_{ji}^g \frac{\chi_i^g}{\mathrm{k}^{(n)}} \{ \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \} \tag{6}$$

where $\vec{\phi}^{(n)}$ and $\mathrm{k}^{(n)}$ are the multigroup flux map and the eigenvalue approximations obtained at iteration n.

In this problem, the order of the matrices is $L \times G$. Generally, both values L and G will be large; this makes the practical inversion of the matrix $\mathbf{A}$ very expensive. For solving at iteration $n+1$, multigroup schemes are therefore also iterative. Now it is important to consider the structure of matrices involved in Eq.(5); the following remarks will influence the way we conceive such an iterative scheme:

i) The $\mathbf{P^g}$ matrices are generally full: they are used to perform the spatial coupling inside the cell.

ii) The scattering matrices $\mathbf{\Sigma_{si}}$ are such that, in the fast groups, there is generally no up-scattering and therefore part of the matrix may be lower diagonal.

In the next subsections, two different approaches to the multigroup solution will be investigated.

III.A. Self-scattering reduction scheme

A usual operation performed before solving the linear transport system on **sequential architecture** is called self-scattering reduction of the collision probability matrices. Using this scheme, all scattering information pertinent to group g is transferred to the left-side of Eq.(5), so that the system is rewritten in the form:

$$V_j \Phi_j^g - \sum_i P_{ji}^g \Sigma_{si}^{g \leftarrow g} \Phi_i^g = \sum_i P_{ji}^g \{ \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{k_{\mathrm{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \}. \tag{7}$$

Assuming a fixed right-hand side, solution of the left-hand side can be achieved either by an iterative or by a direct method. Using direct inversions of these full matrices of order L, we will have to compute the scattering-reduced matrix $\mathbf{W^g}$ for every energy group :

$$\mathbf{W^g} = [\mathbf{V} - \mathbf{P^g \Sigma_s^{g \leftarrow g}}]^{-1} \, \mathbf{P^g} \tag{8}$$

so that the linear system in Eq. (7) can be simplified to:

$$\Phi_j^g = \sum_i W_{ji}^g q_i^g \ , \tag{9}$$

where the new external sources are now coming only from *other* groups and are given by:

$$q_i^g = \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + S_i^g \ , \tag{10}$$

and fission neutrons are still provided by:

$$S_i^g = \frac{\chi_i^g}{k_{\mathrm{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} . \tag{11}$$

The system of Eqs. (9)–(11) is solved by inverse power method assuming a starting value corresponding to one-neutron fission source vector $\vec{S}$. This is the numerical scheme that is currently implemented in

DRAGON.[3] It might be said that it also corresponds to a standard way of solving the multigroup problem when using the integral transport equation. Note that, in every energy group, exactly the same sequence of operations is necessary to compute the $\mathbf{W}^g$ matrix; this is a very nice behavior when going to a very large number of groups. However, the self-scattering reduction is not necessarily the only way of solving the linear transport system on **sequential architecture** as we will see in the next subsection.

IV. Mixed multigroup and multiregion iterative scheme

Another completely different approach which consists of a two-stage iterative process for solving (5), is presently under investigation. Assuming a fixed multigroup source vector $\vec{S}$, the basic form of the linear system is well-known to reactor physics [7]:

$$\begin{pmatrix} \vec{\phi} \\ \vec{q} \end{pmatrix} = \begin{pmatrix} 0 & \mathcal{P} \\ \mathcal{S}_s & 0 \end{pmatrix} \begin{pmatrix} \vec{\phi} \\ \vec{q} \end{pmatrix} + \begin{pmatrix} \vec{0} \\ \vec{S} \end{pmatrix} \tag{12}$$

where matrices $\mathcal{P}$ and $\mathcal{S}_s$ are respectively block-diagonal in space and in energy. We are presently developing an iterative scheme based on a group-region ordering with Chebyshev acceleration. The main problems encountered when trying to parallelize this solver using a two-step iterator are

i)the permutation of unknowns when going from the flux to the emission vector,

ii)the preconditionning of the linear system in order to achieve the best convergence speed.

Such a checker board algorithm that is already known to be well-suited for many parallel applications in diffusion process [8], for group and region accelerations, two dynamically-computed convergence parameters will be needed. When most communications take place only between the neighbours at each step, this scheme is the most efficient.

In the remaining of the paper, implementation of the standard multigroup scheme in the parallel environment and comparative results, will be described.

V. CODE PARALLELIZATION

Let us review some of the steps taken in order to test the code potential for parallelization. The first step was to run the sequential code to find which parts were most CPU time consuming. It was found that, the flux calculations (inner loop) require the most CPU time. The computation of the collision probability matrix is small but becomes non-negligible as the number of regions grows.

The parallel code was written in such a way that it runs for an arbitrary number of processors (which becomes an input); this is what we mean by scalability. A major criteria of good programming resides in making a code as portable as possible: only few changes would be necessary in order to change from a parallel environment to another (ex: from PVM to Express, P4, GENESYS,etc.). Furthermore, debugging consisted of starting with only one processor, making it run successfully and then going further up.

V.A. CP calculation

The CP calculation is carried out in two steps: trajectory tracking and numerical integration. On sequential machine, tracking is done only once for all groups, and CP integration is repeated for each energy group with identical trajectories but different cross-sections.

The parallel algorithm assigns each energy group to a different processor. However, in the first step, tracking is repeated by each processor, even if it gives the same values for all groups. The objective is that there will remain no tracking data communications between groups. In the second step, each processor integrates CP for a set of energy groups. The parallel strategy for computing CP matrices has already been programmed in the TDT code, a subset of APOLLO-2 [9]. The following loops are thus executed:

```
do ig=me+1, ngrp,nproc
   call  pij(...,Pmat(ig))
enddo
```

where **ngrp**, **nproc** and **me**=1,...,**nproc** are the total number of energy groups, the total number of processors and the processor identifier respectively.

V.B. Parallel multigroup solution

In the parallel algorithm of Figure 1 and corresponding to self-scattering reduction, each processor calculates a few inverse matrices associates with a subset of energy ranges. This gives rise to the following loop:

```
do  ig =me+1, ngpr, nproc
    call  calcul(Pmat(ig),...Amat(ig))
    call  invers(Amat(ig),...Wmat(ig))
enddo
```

Assuming an initial one-neutron fission source in the fuel region, an iterative process is started on each processor. At step n of the inverse power method, each processor computes its flux solution and the contribution of its set of energy groups to the total k_{eff} after executing the following loops:

```
do  ig =me+1, ngpr, nproc
        Phi(ig) = Wmat(ig) q(ig)
enddo
Keff(me) =0.
do  ig = me+1, ngrp, nproc
        Keff(me) = Keff(me)+NuSfmat(ig) Phi(ig)
enddo
```

For each processor, the value of the local $k_{\text{eff}}(me)$ and local flux computational results are broadcasted to the other nodes. Each processor computes the sum k_{eff}, and is now able to calculate the source related to its energy subgroup which will be used in the next iteration:

```
do ig = me+1, ngrp, nproc
   q(ig)=0.
   do  ih = 1, ngrp
      q(ig)= q(ig)+ Ssmat(ig,ih ≠ ig) Phi (ih)
      q(ig)= q(ig)+ Chi(ig) NuSfmat(ih) Phi(ih)/ Keff
   enddo
enddo
```

The iterative process is continued until convergence of k_{eff} and of the flux distribution.

V.C. Parallel mixed group-region iterative solution

In attempt to do a new iterative scheme, Eq. (5) has been tranformed using a mixed group-region strategy. Assuming fission sources, at step $(n+1)$ of the inner loop of the multigroup algorithm, the parallel algorithm is split into the following parts:

1. compute $\vec{\phi}^{(n+1)}$ using $\vec{q}^{(n)}$ and CP matrices,
2. evaluate new acceleration parameter for groups; if necessary, region/group permutation
3. compute $\vec{q}^{(n+1)}$ using $\phi^{(n+1)}$ and scattering matrices,
4. evaluate new acceleration parameter for regions, if necessary, group/region permutation.

In steps 2 and 4, permutations are not necessary when using some hypercubic connections [8]. Since a fixed right-hand side and block diagonal matrices are assumed, communications established between nodes in Steps 1 and 3 by associating one processor per group or region respectively, are minimized.

Then, once this inner iterative process has converged, as in the previous section, the source terms which will be used in the next iteration are regenerated and the multigroup process will continue until convergence of K_{eff} and the flux distribution.

VI. RESULTS

VI.A. Cell description

The test case considered here is a 32 regions 32 groups 1D representation of Gentilly-2 cell. First a 69 group cell calculation was performed for the standard 2D CANDU cell [3] using DRAGON. The 32 groups macroscopic cross sections associated with the fuel, coolant and moderator were then obtained by direct condensation of reaction rates, namely

$$\Sigma_M^H = \frac{\sum_{i\in M} V_i \sum_{g\in H} \phi_i^g \Sigma_i^g}{\sum_{i\in M} V_i \sum_{g\in H} \phi_i^g} \tag{13}$$

where i represent the regions associated with material M (fuel, coolant or moderator) and g ($g = 1, 69$) the micro-group associated with the macro-group H ($H = 1, 32$).

The second step involves the generation of an equivalent 1D cell geometry (Figure 2). In order to simplify such a cell we first neglected the structural materials which include the pressure and calandria tubes and the fuel sheath which were explicitly considered in the 2D model. Then the fuel was divided into 7 equal thickness regions (the first one being located at the center of the cell), each being equally spaced. The material located between these fuel regions being filled by the coolant. The thickness of the regions was obtained by ensuring that the fuel volume inside a 1D cell of height 28.575 cm would be equivalent to that of the original 2D cell. Note that the K_∞ computed for the 1D cell is 1.0651 which is slightly lower than that obtained for the standard Gentilly cell (1.1171) even if structural materials are no longer present in the 1D simulation. The reason for this is that the fuel extends to infinity in the 1D cell therefore increasing the neutron absorption in the external fuel region, and leading to a less efficient use of the internal fuel regions.

VI.B. Numerical results

Here are the results using the parallel multigroup algorithm described above:

Table 1: CPU time: Parallelism on the energy		
nproc	32 regions	128 regions
1	179.1s	1578.3s
2	97.8s	841.0s
4	61.8s	453.8s
8	46.2s	247.7s

The speedup curve associated with table 1 above is plotted if Figure 3. Let us recall that the speedup value is the ratio between the CPU time for one processsor and the CPU time for N processors.

It is worth noting that another scheme based on multiregion algorithm has been tried: in Eq. (5) all the entries pertinent to a specific region were moved to the left. Unfortunately, the source terms become related with both regions and groups, and necessitate much more calculation time than their equivalent in multigroup scheme: for one processor, we end up with prohibitive running times for 32 and 64 regions which were 1642 and 9069 seconds respectively.

VI.C. Comments on number of iterations

For the thermal iteration each group was considered independently, that is the iteration process was repeated over each group until convergence was reached. However, this is not the most efficient multigroup iteration scheme. In fact, because there is no up-scattering in the first 13 groups (see Figure 5), a sequential solution would be more efficient in these cases since by treating one group at a time no iteration is required. The only groups which then require the iteration process are lower energy groups corresponding to $g = 14, 32$. Using a parallel solution this means that the solution in group 1 is converged after a single iteration while that in group 13 requires a maximum of 13 iterations to converge. Accordingly, in this case a combination of sequential and parallel treatment would in fact speedup the computation in the case where the number of groups is much larger than the number of processors. For example, if one considers the case of computation which requires 100 iterations for 8 equal-speed processors , in the parallel case, 400 processor cycles (4 processor cycles per iteration) would be required while for sequential-parallel scheme one would start with 13 processors cycles (for the groups with no upscattering) followed by 300 processors cycles for the remaining groups for a total of 313 processors cycle. In the case where 16 out of the 32 groups have no upscattering this is further reduced to 216 cycles. Accordingly, because of the form of the scattering cross section, the self-scattering reduction commonly used in multigroup solvers, and presented above, could be improved only by investigating further the load balance in the slowing-down range.

VII. CONCLUSION

The multigroup scheme provides an attractive alternative to expensive Monte Carlo simulations. Some detailed calculation schemes for solving the multigroup transport equation using a cluster of computers has been presented. All these schemes use the power method for finding the critical multiplication factor in a multigroup iterator. In the coming years, fine-group calculations using thousands of energy groups will be useful to assess transport and equivalence computations on cross-section libraries. The parallel algorithms developed here will provide users with sufficient resources to perform this kind of benchmarking, ,without the usual memory and CPU limitations imposed by problem sizes. It is expected

that results will be improved over the coming years by using well adapted flux rebalancing methods and variational acceleration in order to speedup convergence.

Acknowledgements— This work has been carried out partly with the help of grants from Atomic Energy of Canada Ltd and the Natural Science and Engineering Research Council of Canada.

REFERENCES

[1] G.A. Geist and V.S. Sunderam, "The Evolution of the PVM Concurrent Computing System," Proceeding - 26th IEEE Compcon Symposium, pp. 471-478, San Francisco, February 1993.

[2] A. Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek and V.S. Sunderam, " A User Guide to PVM Parallel Virtual Machine", technical report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[3] R. Roy, G. Marleau, J. Tajmouati and D. Rozon, " Modelling of CANDU Reactivity Control Devices with the Lattice Code DRAGON",*Ann. Nucl. Energy* **21**, 115-132.

[4] R.J.J. Stamm'ler and M.J. Abbate, Methods of steady state reactor physics in nuclear design, Academic Press, New York (1983).

[5] R. Roy, " Anisotropic Scattering for Integral Transport Codes. Part 1. Slab assemblies",*Ann. Nucl. Energy* **17**, 379-388.

[6] G. Marleau and A. Hébert, " Solving the Multrigroup Transport Equation using the Power Iteration Method", 11^{th} Annual Symposium on Simulation of Reactor Dynamics and Plant Control, CMR Kingston, April 21-22, 1985.

[7] A.J. Clayton, " The Program PIP1 for the solution of the Multigroup Transport Equations of the Method of Collision Probabilities", Report AEEEW-R-326, Winfrith, 1964.

[8] R.N. Ibbett and N.P. Topham, Architecture of high performance Computers volume II , Springer-Verlag, New York Inc (1989).

[9] Z. Stankovski, " First Massively Algorithm to be Implemented in Apollo-II Code", private communication to be submitted in ENS-ANS meeting in Israel, Decembre 1993.

APPENDIX

1. In a shared memory architecture, each processor has access to all memory while in a distributed memory architecture, each processor has access only to its own memory and exchange messages with others to have access to their data.

2. The unit of computational work allocated to each processor is a grain. A coarse grain contains many data while a fine grain contains one or few data.

3. The SIMD mode (Single Instruction, Multiple Data) indicates that one current instruction is applied by each of the processors to the data included in their local memory. In the MIMD mode (Multiple Instruction, Multiple Data), each procesor executes its own, separate program.

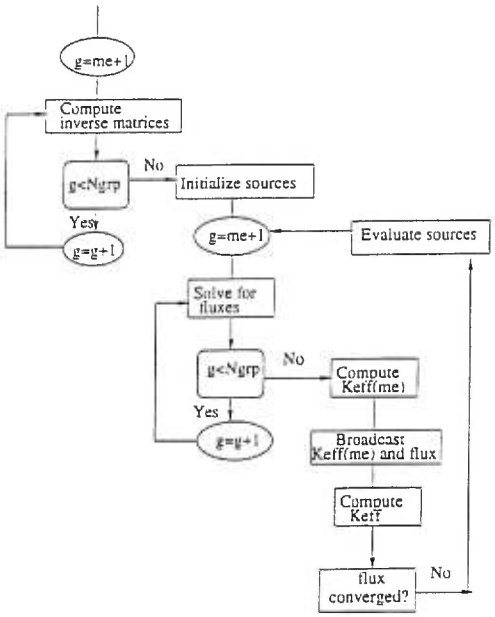

Figure 1: Flow diagram for parallel multigroup strategy

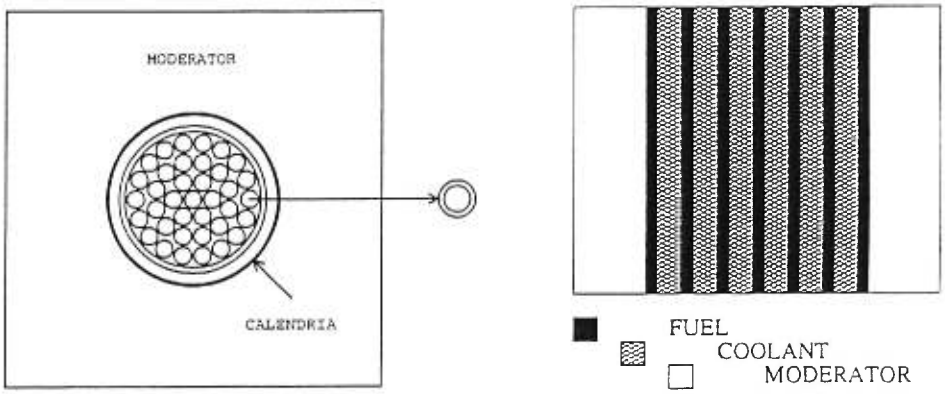

Figure 2: Two-dimensionnal CANDU cluster cell with its slab equivalent model

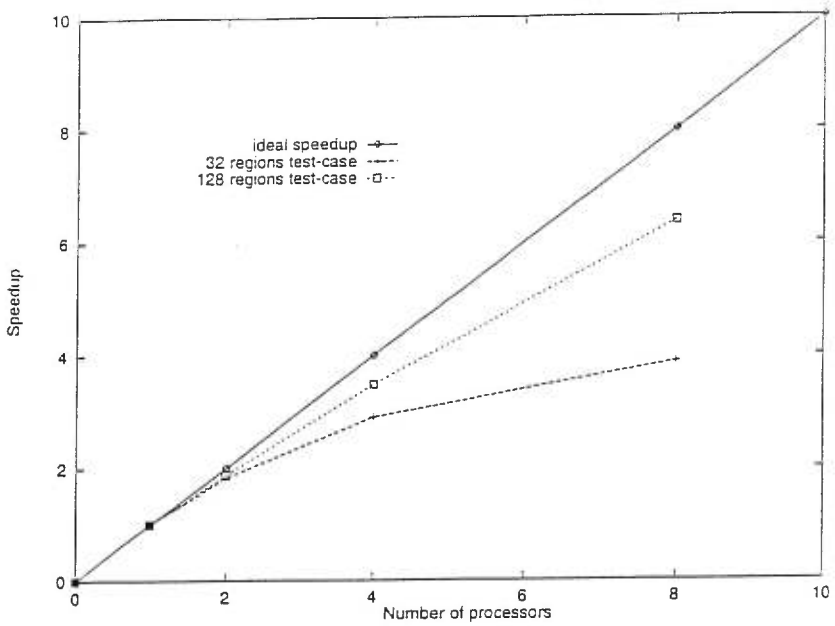

Figure 3: Multigroup parallelization: Speedup curves.

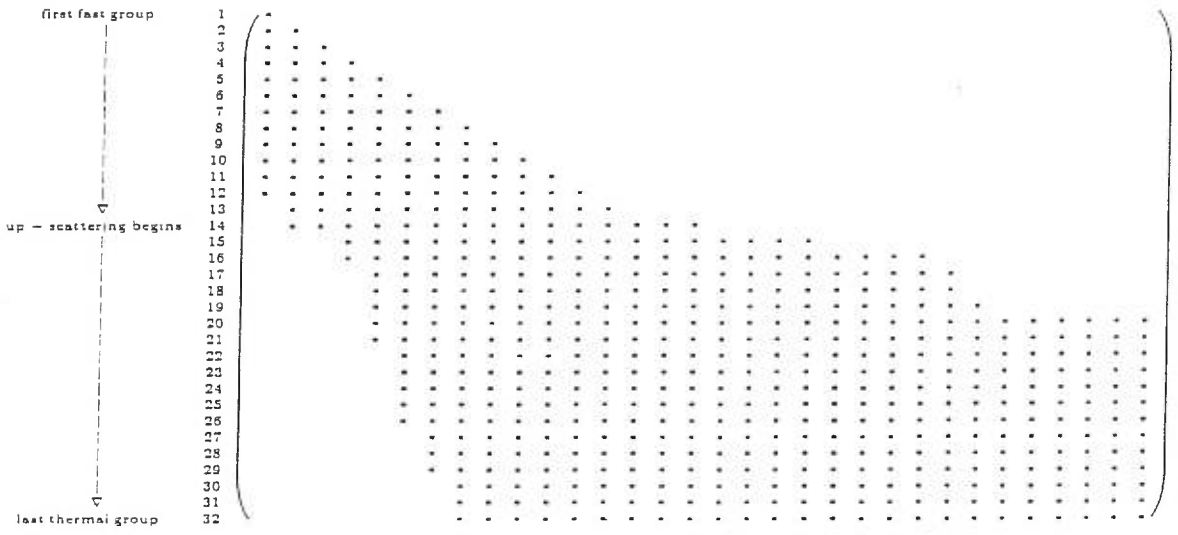

Figure 4: Profile of a typical scattering matrix for the CANDU 32-group model

ARTICLE 2

Multigroup Flux Solvers Using PVM

ANS International Conference
Mathematics and Computations,
Reactor Physics, and Environmental Analyses 1995

envoyé en Septembre 1994, accepté en Fevrier 1995 et publié en Mai 1995

MULTIGROUP FLUX SOLVERS USING PVM

A. Qaddouri[1,2], R. Roy[1], B. Goulard[2]

[1] Institut de Génie nucléaire, École Polytechnique de Montréal
C.P. 6079, Succ.A, Montréal, Québec, H3C 3A7, Canada

[2] Laboratoire de Physique Nucléaire, Université de Montréal,
C.P. 6128, Succ. Centre-Ville, Montréal, Québec, H3C 3J7, Canada
Phone:(514) 343-6111 (4231) Fax:(514) 343-6215
E-mail: Qaddou@lpssub.lps.umontreal.ca

ABSTRACT

Using a cluster of processors with a distributed memory, parallelization for multigroup flux computations is investigated. The parallel performance will be studied in cases where the neutron transport equation is discretized using the collision probability technique. Particular techniques pertinent to the two-step process of solving a multigroup linear equation are described. Parallelization is achieved by distributing either different energy groups or different regions on a set of workstations using PVM. Typical run times will be provided for 1-D CANDU cell test case. Finally, conclusions are drawn for realistic applications of parallel multigroup flux solvers.

I. INTRODUCTION

In the coming years, powerful computing machines will consist of many processors connected by a high speed message exchange network. Small multi-computers are already quite common in industry research centres and universities. However, many challenges remain with regard to algorithm languages and even standards to evaluate multi-processors performance.

Since the multigroup transport equation using the collision probability method is suitable for parallelization, this paper investigates how the sequential code can be parallelized while minimizing the modifications to the initial code and hardware investment. In this spirit, the parallel multigroup solution of the transport equation using PVM (Parallel Virtual Machine) [1,2], a public domain software package for parallelization on a network of workstations, is presented. Parallelization over the energy groups as well as other iterative schemes are examined.

Section II describes the multigroup neutron transport equations. In Section III, the main steps for the parallelization of the equations are described. Performance results are given in Section IV. Finally, conclusions are drawn in the last section.

II. MULTIGROUP TRANSPORT THEORY

Assuming isotropic sources distributed over I regions and G energy groups, the transport equation discretized using the collision probabilities (CP) gives the following linear system of equations:

$$V_j \Phi_j^g = \sum_{i=1}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{\mathrm{k_{eff}}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \} \ , \qquad (1)$$

with symmetric CP matrices $\mathbf{P^g}$ of order I. In order to solve this eigenvalue problem for k_{eff} and compute the fluxes, multigroup iteration schemes are generally implemented using the *inverse power method*:

$$\mathbf{A}\vec{\phi}^{(n+1)} = V_j \Phi_j^{g(n+1)} - \sum_{i=1}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n+1)} \} =$$

$$\frac{1}{k^{(n)}} \mathbf{B}\vec{\phi}^{(n)} = \sum_{i=1}^{I} P_{ji}^g \frac{\chi_i^g}{k^{(n)}} \{ \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \}$$

where $\vec{\phi}^{(n)}$ and $k^{(n)}$ are the multigroup flux and the eigenvalue approximations obtained at iteration n.

In this problem, the order of the matrices is $I \times G$. Generally, both values I and G will be large; this makes the pratical inversion of the matrix A very expensive. For solving at iteration $n+1$, multigroup schemes are therefore iterative. Now it is important to consider the matrix structures involved in Eq. (1); the following remarks will drive the way we conceive such an iterative scheme:

1. The $\mathbf{P^g}$ matrices are generally full: They are used to perform the spatial coupling inside the cell,

2. The scattering matrices $\mathbf{\Sigma_{si}}$ are such that, in the fast groups, there is no up-scattering and therefore part of the matrix is lower diagonal.

In the next subsections, two different approaches to the multigroup solution will be investigated.

II.A Self-scattering reduction scheme (multigroup)

A usual operation performed before solving the linear transport system on **sequential architecture** is called scattering reduction of the collision probability matrices. Using this scheme, all scattering information pertinent to group g is transferred to the left-side of Eq. (1),and the system is rewritten in the form:

$$V_j \Phi_j^g - \sum_{i=1}^{I} P_{ji}^g \Sigma_{si}^{g \leftarrow g} \Phi_i^g = \sum_{i=1}^{I} P_{ji}^g \{ \sum_{g' \neq g}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{k_{\text{eff}}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \} \ , \qquad (1')$$

Assuming a fixed right-hand side, solution of the left- hand side can be achieved either by an iterative or by a direct method. Using direct inversion of these full matrices of order I, we will have to compute the scattering-reduced matrix $\mathbf{W^g}$ for every energy group :

$$\mathbf{W^g} = [\mathbf{V} - \mathbf{P^g \Sigma_s^{g \leftarrow g}}]^{-1} \times \mathbf{P^g}$$

so that the linear system in Eq. (1′) can be simplified to:

$$\Phi_j^g = \sum_{i=1}^{I} W_{ji}^g q_i^g \ . \qquad (2)$$

where the external sources coming from *other* groups are given by:

$$q_i^g = \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + S_i^g \tag{3}$$

with fission neutrons provided by:

$$S_i^g = \frac{\chi_i^g}{\mathrm{k_{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \tag{4}$$

The system of Eqn. (2–4) is solved by the inverse power method assuming a one-neutron fission source vector $\vec{S}$. This is the numerical scheme that is currently implemented in DRAGON [3]. It might be said that it also corresponds to a standard way of solving the multigroup problem when using the integral transport equation [4]. Note that, in every energy group, exactly the same sequence of operations is necessary to compute the $\mathbf{W^g}$ matrix; this is a very nice behaviour when going to a very large number of groups.

However, the self-scattering reduction is not necessarily the only way of solving the linear transport system on **sequential architecture** as we will see in the next subsection.

II.B Spatial recomposition scheme (multiregion)

Another completely different approach which consists of a two-step iterative process is presented.

We split the spatial domain into L non-overlapping set of regions, and reorder the flux and the source vector such that for each set B (block) of regions we have:

$$\Phi_B = \{\Phi_i^g;\ g = 1,2,...,G;\ i \in B;\ B = 1,2,...,L\} \quad \Phi = \oplus_B \Phi_B$$
$$s_B = \{s_i^g;\ g = 1,2,...,G;\ i \in B;\ B = 1,2,...,L\} \quad s = \oplus_B s_B$$

This original scheme of partitioning the spatial domain instead of the energy domain has been attempted by Henkel and Turinsky [5] in order to solve the few-group diffusion equation.

The system of equations (1) is rewritten in the form:

$$\begin{aligned} V_j \Phi_j^g - \sum_{i=1;i\in B}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} \} - \\ \sum_{i=1;i\notin B}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} \} = \sum_{i=1}^{I} P_{ji}^g \{ \frac{\chi_i^g}{\mathrm{k_{eff}}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \} \\ j \in B; \quad B = 1,2,...L \end{aligned} \tag{1''}$$

This system of equations is solved by multiblock iteration scheme using the inverse power method assuming a one-neutron fission source:

$$\begin{aligned} V_j \Phi_j^{g(n+1)} - \sum_{i=1;i\in B}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n+1)} \} - \\ \sum_{i=1;i\notin B}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n+1)} \} = \sum_{i=1}^{I} P_{ji}^g \{ \frac{\chi_i^g}{\mathrm{k}(n)} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \} \\ j \in B; \quad B = 1,2,...L \end{aligned} \tag{5}$$

Collecting coefficients pertinent to each block B, this system of equation can be rewritten as:

$$D_B \Phi_B^{(n+1)} + \sum_{B' \neq B} C_{BB'} \Phi_{B'}^{(n+1)} = s_B^{(n)}; \quad B = 1, 2, ..., L. \tag{6}$$

where D_B is a full matrix, and has the following form:

$$D_B = \begin{pmatrix} d_1 & u_{12} & . & . & . & u_{1G} \\ l_{21} & d_2 & u_{23} & . & . & \\ & . & & & & \\ & . & & & & \\ & . & & & & \\ l_{G1} & . & . & . & & d_G \end{pmatrix} \tag{7}$$

where G is the total number of energy groups. The matrices l_{ij} express the downscattering from energy group j to group i and the matrices u_{ij} express the upscattering from energy group j to group i.

For each block B, we have a fixed source problem, and an iterative method is used to calculate the fluxes. At each step, the following block subsystem has to be solved :

$$D_B \Phi_B = - \sum_{B' \neq B} C_{BB'} \Phi_{B'} + s_B = s'_B; \quad B = 1, 2, ..., L. \tag{8}$$

and the fluxes in the regions of B for each energy group are:

$$\phi_B^{g\,(k+1)} = d_g^{-1} \left[s'_B - \sum_{g' < g} l_{gg'} \phi_B^{g'\,(k+1)} - \sum_{g' > g} u_{gg'} \phi_B^{g\,(k)} \right]; \quad g' \text{ and } g = 1, 2, 3, ..., G. \tag{9}$$

where $\phi_B^{g\,(k+1)}$ and $\phi_B^{g\,(k)}$ are respectively the $(k+1)^{th}$ and k^{th} order approximation to the fluxes in the block B in g^{th} energy group. This level of iteration provides a convergence for fluxes even in energy groups concerned by upscattering. This level of iteration constitutes the inner iterations, and we expect the convergence of the block fluxes $\phi_B^{(k)} \rightarrow \Phi_B$. Since not all the regions are included in each block B, an intersting property of matrices d_g is that their spectral radius is much smaller than the ones of matrices involved in the self-scattering scheme.

First, the calculation of $s_B^{'(n)}; \quad B = 1, 2, ..., L$ is done. A new iterative strategy (an outer iteration) is inserted in the process to take into account the flux-dependent sources. Only a part of the effective multiplication factor $\mathrm{k}_B^{(n)}$ of n^{th} order of approximation for the block B is evaluated using the following expression:

$$\mathrm{k}_B^{(n)} = \sum_{i \in B} \sum_{g}^{G} \chi_i^g \nu \Sigma_{f_i}^g \phi_i^g \tag{10}$$

The total multiplication factor is then calculated as:

$$\mathrm{k}^{(n)} = \sum_{B} \mathrm{k}_B^{(n)}$$

The sources $s_B^{'(n)}; \; B = 1, 2, ..., L$ are then reevaluated using the multiplication factor and the fluxes thus computed:

$$s_B^{'(n)} = \sum_{i=1; i \notin B}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n)} \} + \sum_{i=1}^{I} P_{ji}^g \{ \frac{\chi_i^g}{\mathrm{k}^n} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \} \tag{11}$$

$$j \in B; \quad B = 1, 2, ...L$$

and these sources can be used as input in the next inner iteration process, until the convergence of $\mathrm{k}_{\mathrm{eff}}$.

III. CODE PARALLELIZATION

Let us review some of the steps taken in order to test the code potential for parallelization. The first step was to run the sequential code to find which parts were most CPU time consuming. It was found that the flux calculations (inner loop) require the most CPU time. The computation of the collision probability matrix is small but becomes non-negligible as the number of regions grow.

The parallel code was written in such a way that it runs for an arbitrary number of processors (which becomes an input); this is what we mean by scalability. A major criterion of good programming resides in making a code as portable as possible: only few changes would be necessary in order to change from a parallel environment to another (ex: from PVM to Express, P4, GENESYS,etc.). Furthermore, debugging consists of working with only one processor, making it run successfully and then going further up.

III.A Parallel multigroup solution

The CP calculation is carried out in two steps: trajectory tracking and numerical integration. On a sequential machine, tracking is done only once for all groups, and CP integration is repeated for each energy group with identical trajectories but different cross sections.

The parallel algorithm corresponding to self-scattering reduction assigns each energy group to a different processor [6]. However, in the first step, tracking is repeated by each processor, even if it gives the same values for all groups. The objective is that there will remain no tracking data communications between groups. In the second step, each processor integrates CP for a set of energy groups. This parallel strategy for computing CP matrices has already been programmed in the TDT code, a subset of APOLLO-2 [7]. The following loops are thus executed:

```
call pvmfmytid(mytid)
call pvmfparent(tids(0))
do i=1, nproc
  if(mytid.eq.tids(i)) me=i
enddo
do ig=me+1, ngrp,nproc
  call PIJ(...,Pmat(ig))
enddo
```

where **ngrp**, **nproc** and **me=1,...,nproc** are the total number of energy groups, the total number of processors and the processor identifier respectively.

Each processor calculates a few inverse matrices associated with a subset of energy ranges. This gives rise to the following loop:

```
do ig=me+1, ngpr, nproc
   call calcul(Amat(ig),...)
   call invers(Amat(ig),...Wmat(ig))
enddo
```

Assuming an initial one-neutron fission source in the fuel region, an iterative process is started on each processor. At each step (n) of the inverse power method, each processor computes its flux solution and the contribution of its set of energy groups to the total K_{eff} after executing the following loops:

```
do ig=me+1, ngpr, nproc
  Φ(ig) = Wmat(ig) q(ig)
enddo
```

$$\mathbf{K_{eff}(me) = 0.0}$$
$$\mathbf{do\ ig = me + 1, ngrp, nproc}$$
$$\mathbf{K_{eff}(me) = K_{eff}(me) + \nu\Sigma_f mat(ig)\Phi(ig)}$$
$$\mathbf{enddo}$$

For each processor, the value of the local $K_{\text{eff}}(me)$ is broadcast to the other nodes in order to compute the sum. This gives the same value of K_{eff} on each processor. After, the processors exchange their respective local flux computational results. At this stage, each processor is able to calculate the source related to its energy subgroup which will be used in the next iteration $(n+1)$:

$$\mathbf{do\ ig = me + 1, ngrp, nproc}$$
$$\mathbf{q(ig) = 0.0}$$
$$\mathbf{do\ ih = 1, ngrp}$$
$$\mathbf{q(ig) = q(ig) + \Sigma_S mat(ig, ih \neq ig)\Phi(ih \neq ig)} + \frac{\chi^{ig}}{\mathbf{K_{eff}}}\left\{\nu\Sigma_f \mathbf{mat(ih)}\Phi(\mathbf{ih})\right\}$$
$$\mathbf{enddo}$$
$$\mathbf{enddo}$$

The iterative process continues until convergence of K_{eff} and of the flux distribution.

III.B Parallel multiregion iterative solution

The parallel algorithm assigns each block of regions to a different processor. The CP integration is repeated by each processor, even if it gives the same values. The CP matrices are distributed among the processors so that each of them can calculate its matrices D_B and $C_{BB'}$ $(B' \neq B)$, and perform the resolution of Eq.8 for blocks B by inner iterations. Each processor calculates the contribution of its Blocks to the total multiplication factor (Eq.10).

The computation of the sources s'_B in Eq.11 requires the interprocessor communication by the algorithm at each outer iteration. As in the previous parallel algorithm, each processor communicates to other nodes its computed fluxes and its own contribution to total $\mathrm{K_{eff}}$. Then each processor calculates its sources s'_B:

$$\mathbf{do\ iB = (me)\ NBlk, (me + 1)\ NBlk}$$
$$\mathbf{s'(iB) = 0.0}$$
$$\mathbf{do\ jB = 1, NBlk}$$
$$\mathbf{s'(iB) = s'(iB) + C(jB \neq iB, iB)\Phi(jB)}$$
$$\mathbf{enddo}$$
$$\mathbf{s'(iB) = s'(iB) + s(iB)}$$
$$\mathbf{enddo}$$

which will serve as input in the next inner iterations, and where **me** and **NBlk** are the processor identifier and the total number of blocks in the processor **me** respectively.

Theoretically, we could balance the work of each processor by using the cross-section values of materials located in each block; the high scattering regions are generally slower to converge.

The iterative process continues until convergence of K_{eff} and of the flux distribution on each processor.

IV. RESULTS

IV.A Cell description

The test case considered here is a 32-region, 32-group 1D representation of a Gentilly-2 cell. First a 69 group cell calculation was performed for the standard 2D CANDU cell [3] using DRAGON. The 32 groups macroscopic cross sections associated with the fuel, coolant and moderator were then obtained by direct condensation of reaction rates, namely

$$\Sigma_M^H = \frac{\sum_{i\in M} V_i \sum_{g\in H} \phi_i^g \Sigma_i^g}{\sum_{i\in M} V_i \sum_{g\in H} \phi_i^g}$$

where i represents the regions associated with material M (fuel coolant or moderator) and g ($g = 1, 69$) the micro-group associated with the macro-group H ($H = 1, 32$).

The second step involves the generation of an equivalent 1D cell geometry. In order to simplify such a cell we first neglected the structural materials which include the pressure and calandria tubes and the fuel sheath which were explicitly considered in the 2D model. Then the fuel was divided into 7 equal thickness regions (the first one being located at the center of the cell), each being equally spaced. The material located between these fuel regions is filled by the coolant. The thickness of the regions was obtained by ensuring that the fuel volume inside a 1D cell of height 28.575 cm would be equivalent to that of the original 2D cell. Note that the K_∞ computed for the 1D cell is 1.0651, which is slightly lower than that obtained for the standard Gentilly cell (1.1171) even if structural materials are no longer present in the 1D simulation. The reason for this is that the fuel extends to infinity in the 1D cell, therefore increasing the neutron absorption in the external fuel region, and leading to a less efficient use of the internal fuel regions.

IV.B Numerical results

Here are the results using the parallel multigroup and parallel multiregion algorithms described above. The number of processors is indicated by nproc:

Table 1: CPU time:32 energy groups and 32 regions

nproc	multigroup	multiregion
1	179.1s	430.35s
2	97.8s	224.7s
4	61.8s	135.8s
8	46.2s	80.9s

The speedup curve associated with table 1 above, is plotted in Figure 1. Let us recall that the speedup value is the ratio between the CPU time for one procesor and the CPU time for N processors. It is interesting to see that for the same problem the speed up associated with the spatial recomposition scheme (multiregion) is better than self scattering reduction scheme (multigroup).

Figure 2 represents the speedup curves in the parallel multigroup algorithm for 32 and 128 regions. We observe that when the number of region increases, the multigroup speedup curve approaches the ideal speedup curve. The same results were observed in the case of multiregion scheme.

V. CONCLUSION

The multigroup and multiregion schemes provide an attractive alternative to expensive Monte Carlo simulations. Numerical results using calculation schemes for solving the multigroup transport equation using a cluster of computers have been presented. All these schemes use the power method for finding the critical multiplication factor in a multigroup iterator. In the coming years, fine-group calculations using thousands of regions will be useful to assess transport and equivalence computations on cross-section libraries over complex geometries. The parallel algorithms developed here will provide users with sufficient resources to perform this kind of benchmarking, without the usual memory and CPU limitations imposed by problem size. It is expected that preliminary results stated here will be improved, over the coming years, by using new flux rebalancing methods adapted to the cluster environment in order to speedup convergence.

Acknowledgements— This work has been carried out partly with the help of grants from Hydro-Québec, Atomic Energy of Canada Ltd and the Natural Science and Engineering Research Council of Canada. We would like to thank also J. Cloutier and J.L. Martineau of the Computer Science Department of Montréal University, who gave us access to their SparcStation network. Specials thanks to ours colleagues G.Marleau and M.Mayrand for their help in accomplishing this work.

REFERENCES

[1] G.A. Geist and V.S. Sunderam, *The Evolution of the PVM Concurrent Computing* System, Proceeding – 26th IEEE Compcon Symposium, pp. 471-478, San Francisco, February 1993.

[2] A. Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek and V.S. Sunderam, *A Users' Guide to PVM Parallel Virtual Machine* , technical report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[3] R. Roy, G. Marleau, J. Tajmouati, and D. Rozon, *Modelling of CANDU Reactivity Control Devices with the Lattice Code DRAGON* , Ann. nucl. Energy **21**, 115–132.

[4] B. Sanchez and N.J. McCormick, *A Review of Neutron Transport Approximations,* Nucl.Sci.Eng., 80, 481-535 (1982).

[5] C.S. Henkel and P.J. Turinsky, *Solution of the Few-Group Diffusion Equation on a Distributed Memory Multiprocessor*, Top. mtg. on Advances in Math., Comp. and Reactor Physics, Pittsburgh, USA, March 1992.

[6] A. Qaddouri, R. Roy, G. Marleau, B. Goulard, and M. Mayrand, *Multigroup Flux Solvers on Parallel Architecture*, CNA/CNS Ann. conference , Montreal, June 1994.

[7] Z. Stankovski, *First Massively Algorithm to be Implemented in Apollo-II Code* , Int. Conf. on Reactor Physics and Reactor Comp., Telaviv, Israel, January 1994.

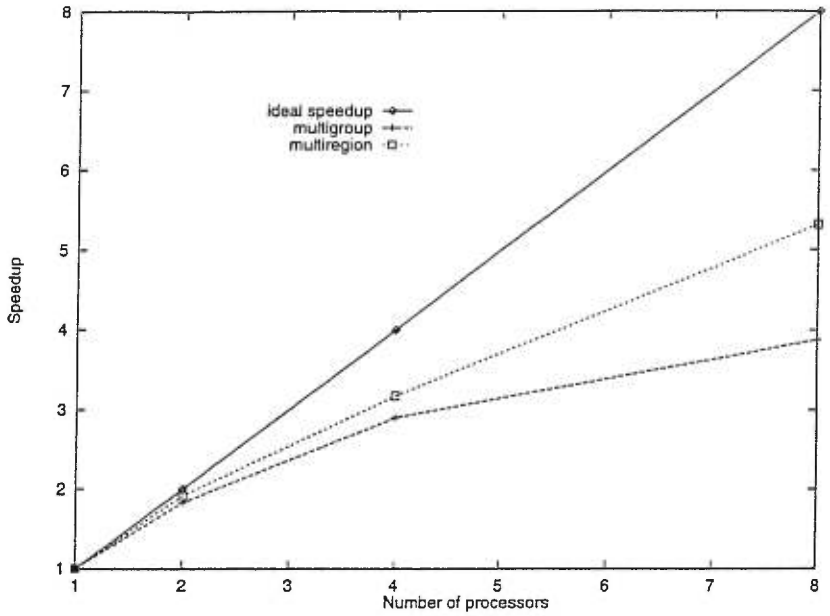

Figure 1: Comparative Speedups for both iterative schemes on the same test problem.

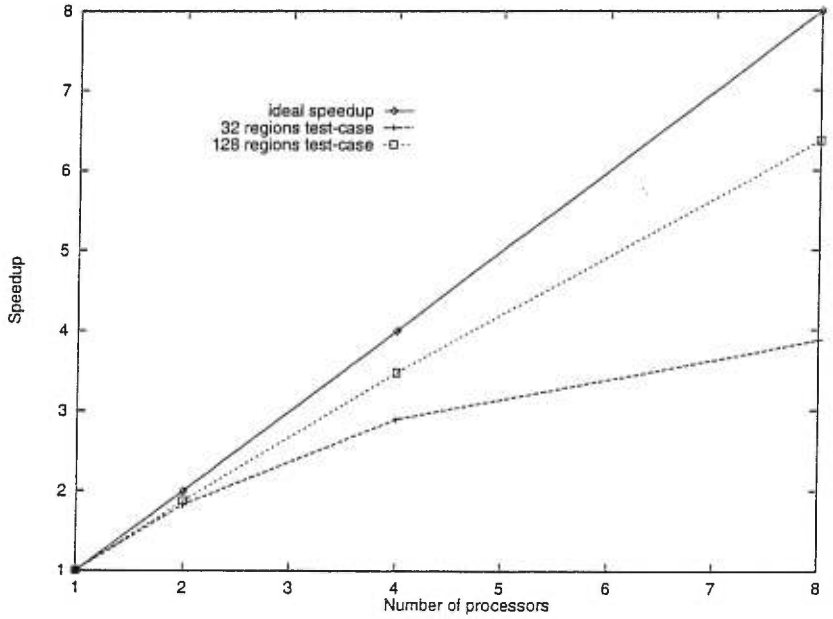

Figure 2: Speedup variation with an increasing number of regions for the multigroup iterative scheme.

ARTICLE 3

Collision Probability Calculation and Multigroup Flux Solvers Using PVM

Nucl. Sci. Eng. 1996

envoyé en Septembre 1995, accepté Janvier 1996 et publié en Octobre 1996

COLLISION PROBABILITY CALCULATION AND MULTIGROUP FLUX SOLVERS USING PVM

A. Qaddouri [1,2], R. Roy [1], M. Mayrand [2] and B. Goulard [2]

[1] *Institut de Génie Nucléaire, École Polytechnique de Montréal*

Montréal, Québec, Canada H3C 3A7

and

[2] *Centre de Recherches Mathématiques, Université de Montréal*

Montréal, Québec, Canada H3C 3J7

Collision probability (CP) evaluation and flux computation share the most time consumed in the applications based on the linearized time-independent transport equation. In this paper, parallelization for CP calculation and multigroup flux computation are investigated. Particular techniques pertinent to the two-step energy/space iterative process of solving a multigroup transport equation are described. The parallel performance is studied in cases where the cyclic tracking technique is used to integrate CP. Parallelization is achieved by distributing either different energy groups or different regions on set of processors. These algorithms were tested on 4 processors IBM SP-2, 8 processors SPARC 1000 as well as networks of workstations using the public domain PVM library. Typical run times are provided for unit cell calculations.

I. INTRODUCTION

In the DRAGON code,[1] collision probability (CP) methods for computing the neutron flux within a lattice cell as a function of space and energy have been developed on the basis of a representation of neutron transport in terms of first-flight probabilities. This approach has the considerable advantage that the transport and collision mechanics parts of the problem are completely dissociated from the group-to-group energy transfer terms; the form of the final linearized equations which are solved numerically is not dependent on the type of geometry.

The numerical integration of CP formulas is based on a ray tracing procedure that selects discrete angles and generates a finite set of neutron paths. One way to input the macroscopic mixture cross sections is from a standard GOXS format file.[2] These macroscopic cross sections are stored by type, each record containing the information pertaining to every material required in the transport calculation. The cross sections associated with a given mixture can be updated, and an other output GOXS macroscopic cross section file can be generated.

The CP calculation for unit cell geometries is done, in each energy group, by multi-integration over the neutron trajectories traversing the domain. A geometry with many regions can require a large number of trajectories. The number of unknowns in the flux solver increases fast if we also consider many energy groups.

Since the multigroup transport equation using CP methods is suitable for parallelization, this paper investigates how the sequential code can be parallelized while minimizing the modifications to the initial code and hardware investment. In this spirit, the parallel multigroup solution of the transport equation using PVM[3,4] on IBM SP2, SPARC 1000 as

well as networks of workstations, is presented. The rather straight-forward parallelization over the energy groups as well as other more sophisticated iterative schemes are examined.

The purpose of this paper is to study the performance of these iterative schemes for obtaining a converged flux map. The form of linear system resulting after space/energy discretization is the same for one-, two- or three-dimensional geometries. Without any loss of generality, the multigroup standard version of the code DRAGON, previously used to solve the transport equation on infinite lattices of cells, was taken as the basis of our tracking file generator. We thus start our calculations using two input files:

1) The tracking file resuming geometric data in the sequential format described in Appendix of Ref. 1;

2) The GOXS file describing the nuclear properties.

In Sec. II, we present the multigroup equations, where the first subsection is devoted to the cyclic tracking procedure.[5] The other subsections concern the two schemes used here to solve the multigroup neutron transport equation. In Sec. III, the main steps for the parallelization of the equations are described. Performance results are given in Sec. IV. Finally, conclusions are drawn in the last section.

II. MULTIGROUP TRANSPORT METHODS

The DRAGON cell code attempts to produce a consistent solution using the collision probabilities (CP) method to a static problem of the form:

$$\vec{\Omega}\cdot\vec{\nabla}\Phi(\vec{r},\vec{\Omega},E)+\Sigma(\vec{r},E)\Phi(\vec{r},\vec{\Omega},E)=$$

$$\int_0^\infty dE'\int_{4\pi}d^2\Omega'\,\Sigma_s(E\leftarrow E',\vec{\Omega}\leftarrow\vec{\Omega}')\,\Phi(\vec{r},\vec{\Omega}',E')+ \qquad (1)$$

$$\frac{\chi(\vec{r},E)}{4\pi}\int_0^\infty dE'\nu\,\Sigma_f(\vec{r},E')\int_{4\pi}d^2\Omega'\,\Phi(\vec{r},\vec{\Omega}',E')\quad ,$$

where the usual macroscopic cross sections (Σ, Σ_s and Σ_f) are formed by combining the microscopic cross sections of various isotopes.

In order to calculate the CP,[5] we have considered a model that uses specular boundary conditions outside the domain,[5] which is partitioned into I zones, each zone being defined such that it has homogeneous material properties.

II.A. Cyclic tracking procedure and CP evaluation

We thus consider a 2-D square geometry made up of I homogeneous regions with associated specular reflexive boundary conditions at its four external surfaces. Such a cell can be represented by an infinite assembly where each region in the initial cell is reproduced an infinite number of times. This unfolding takes directly into account the contribution to the CP arising when a neutron, after crossing an external surface, encounters a new cell. Assuming that the neutron angular direction is described by $(\mu = cos\theta, \varphi)$, where φ is the angle in the 2-D cell plane, we recall that the first-flight collision probability from region j to i for energy group g is then given by:[1,5]

$$P_{ij}^g = \int_0^{2\pi} \frac{d\varphi}{2\pi} \int_{\perp_\varphi} dl \int_0^1 \frac{d\mu}{\sqrt{(1-\mu^2)}} F_{ij}^g(\varphi, l, \mu). \tag{2}$$

In the above, the function $F_{ij}^g(\varphi, l, \mu)$ which includes the boundary conditions, represents the attenuation of the neutron source from region j to region i in the energy group g. For fully reflected unit cell calculations, the expression for F_{ij}^g is:

$$F_{ij}^g(\varphi, l, \mu) = (1 - \ exp[-\frac{\tau_c^g}{\sqrt{(1-\mu^2)}}])^{-1} \int_{t_i}^{t_{i+1}} dt \int_{t_j}^{t_{j+1}} dt' \ exp[\frac{-(\Sigma_i^g(t_{i+1}-t)+\tau_{ij}^g+\Sigma_j^g(t'-t_j))}{\sqrt{(1-\mu^2)}}], \tag{3}$$

where τ_{ij}^g and τ_c^g are the optical paths respectively between regions i and j and associated with one track cycle.[1] The integrations over dt and dt' are performed analytically with different formulas in the cases where the regions are voided or not.[5]

For square unit cells, angular quadratures are defined using sets of angles φ_k which generate periodic tracks and are given by:

$$\varphi_k = \arctan\, k/(n-k), \tag{4}$$

where for a given integer n, k varies from 0 to n. To define the integration meshes, the user specifies the order of the angular quadrature (that is the value of n) and a minimal track separation d in tracks/cm. This track separation is used to scan the unit cell along the dl line element normal to φ in Eq. (2).

We use this cyclic tracking procedure to generate tracks all over the geometric domain. Each track is composed of couples (j, L_j) identifying each region j that is crossed, and the length $L_j = t_{j+1} - t_j$ of this crossing. The tracking file also contains all the data pertinent to the geometry. Using the data stored on the tracking file, the elements P_{ij}^g can now be evaluated for each energy group, with the same trajectories, by changing only the total cross sections.

II.B. Remarks on the linearized multigroup system

Assuming G energy groups and I regions and once the CP matrices have been computed in every energy group, the transport equation is discretized into the form of a linear system of equations:

$$V_j \Phi_j^g = \sum_{i=1}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{\mathrm{k}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \}, \tag{5}$$

with symmetric CP matrices $\mathbf{P^g}$ of order I. Note that P_{ji}^g is an element of the group g matrix, representing the first-flight reduced probability from region i to region j which has been multiplied by the volume of region j; the CP reciprocity relations assess the symmetry of $\mathbf{P^g}$ matrices. In order to avoid cumbersome notations, we will use the shorter forms $\sum_{g'}$ for $\sum_{g'=1}^{G}$ and $\sum_i$ for $\sum_{i=1}^{I}$ in the rest of the paper. In order to solve this eigenvalue problem for K_{eff} and compute the fluxes, multigroup iteration schemes are generally implemented using the inverse power method:

$$\mathbf{A}\vec{\phi}^{(n+1)} = V_j \Phi_j^{g(n+1)} - \sum_i P_{ji}^g \{\sum_{g'} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n+1)}\} =$$
$$\frac{1}{\mathrm{k}^{(n)}} \mathbf{B}\vec{\phi}^{(n)} = \sum_i P_{ji}^g \frac{\chi_i^g}{\mathrm{k}^{(n)}} \{\sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)}\} \tag{6}$$

where $\vec{\phi}^{(n)}$ and $\mathrm{k}^{(n)}$ are the multigroup flux map and the eigenvalue approximations obtained at iteration n.

In this problem, the order of the matrices is $I \times G$. Generally, both values I and G will be large; this makes the practical inversion of the matrix $\mathbf{A}$ very expensive. For solving at iteration $n+1$, multigroup schemes are therefore also iterative. Now it is important to consider the structure of matrices involved in Eq. (5); the following remarks will influence the way we conceive such an iterative scheme:

i) The $\mathbf{P^g}$ matrices are generally full: they are used to perform the spatial coupling inside the cell. However, if their symmetry is used, the memory (required in each energy group) is reduced from I^2 to $I(I+1)/2$ words.

ii) The scattering matrices $\mathbf{\Sigma_{si}}$ are such that, in the fast groups, there is generally no up-scattering and therefore part of the matrix may be lower diagonal.

In the next subsections, two different approaches to the multigroup solution will be investigated.

II.C. Self-scattering reduction scheme (multigroup)

A usual operation performed before solving the linear transport system on **sequential architecture** is called self-scattering reduction of the collision probability matrices. Using this scheme, all scattering information pertinent to group g is transferred to the left-side of Eq.(5), so that the system is rewritten in the form:

$$V_j \Phi_j^g - \sum_i P_{ji}^g \Sigma_{si}^{g \leftarrow g} \Phi_i^g = \sum_i P_{ji}^g \{ \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \}. \tag{7}$$

Assuming a fixed right-hand side, solution of the left-hand side can be achieved either by an iterative or by a direct method. Using direct inversions of these full matrices of order I, we will have to compute the scattering-reduced matrix $\mathbf{W^g}$ for every energy group :

$$\mathbf{W^g} = [\mathbf{V} - \mathbf{P^g} \mathbf{\Sigma_s^{g \leftarrow g}}]^{-1} \, \mathbf{P^g} \tag{8}$$

so that the linear system in Eq. (7) can be simplified to:

$$\Phi_j^g = \sum_i W_{ji}^g q_i^g \ , \tag{9}$$

where the new external sources are now coming only from *other* groups and are given by:

$$q_i^g = \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + S_i^g \ , \tag{10}$$

and fission neutrons are still provided by:

$$S_i^g = \frac{\chi_i^g}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'}. \tag{11}$$

The system of Eqs. (9)–(11) is solved by the inverse power method assuming a starting value corresponding to a fission source vector $\vec{S}$. This is the numerical scheme corresponding to a standard way of solving the multigroup problem when using the integral transport equation.[6] Note that, in every energy group, exactly the same sequence of operations is necessary to compute the $\mathbf{W^g}$ matrix; this is a very nice behavior when going to a very large number of groups.

However, the self-scattering reduction is not necessarilly the only way of solving the linear transport system on **sequential architecture** as we will see in the next subsection.

II.D. Spatial recomposition scheme (multiregion)

Another completely different approach which consists of a two-step iterative process is presented.

We partition the spatial domain into a L non-overlapping set of regions, and reorder the flux and the source vector so that, for each set B (blocks $B = 1, 2, ..., L$) of regions, we reorder unknowns to form block-dependent and global flux and source vectors:

$$\begin{aligned} \vec{\phi}_B = &\ \{\Phi_i^g;\ g = 1, 2, ..., G;\ i \in B\}, \quad \vec{\phi} = \oplus_B \vec{\phi}_B, \\ \vec{s}_B = &\ \{s_i^g;\ g = 1, 2, ..., G;\ i \in B\}, \quad \vec{s} = \oplus_B \vec{s}_B. \end{aligned} \tag{12}$$

where $\oplus_B$ represents a direct sum used to concatenate all the unknowns together. This original scheme of partitioning the spatial domain instead of the energy domain has been developed by Henkel and Turinsky in order to solve the few-group diffusion equation.[7]

The original system of Eq.(5) is rewritten in the form:

$$[\ V_j\Phi_j^g - \sum_{i\in B} P_{ji}^g\{\sum_{g'} \Sigma_{si}^{g\leftarrow g'}\Phi_i^{g'}\}\ ] \ - \ \sum_{i\notin B} P_{ji}^g\{\sum_{g'} \Sigma_{si}^{g\leftarrow g'}\Phi_i^{g'}\}$$

$$= \sum_i P_{ji}^g \{ \frac{\lambda_i^g}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \} \qquad (13)$$

$$j \in B; \quad B = 1, 2, ...L.$$

The updated unknowns from block $B' \neq B$ will be unavailable to the processor responsible for block B, so the usual inverse power iterator will be slightly changed. Collecting coefficients pertinent to each block B, this system of equations will now be iteratively solved as:

$$\mathcal{D}_B \Phi_B^{(n+1)} + \sum_{B' \neq B} \mathcal{C}_{BB'} \Phi_{B'}^{(n)} = s_B^{(n)}; \quad B = 1, 2, ..., L. \qquad (14)$$

where we have used matrix form and where $\mathcal{D}_B$ is a one-block multigroup matrix which can be split in the following form:

$$\mathcal{D}_B = \begin{pmatrix} d_B^1 & u_B^{12} & . & . & . & u_B^{1G} \\ l_B^{21} & d_B^2 & . & . & . & u_B^{2G} \\ . & . & & & & \\ . & & . & & & \\ . & & & . & & \\ l_B^{G1} & l_B^{G2} & . & . & & d_B^G \end{pmatrix} = \mathbf{D}_B + \mathbf{L}_B + \mathbf{U}_B \qquad (15)$$

The submatrices l_B^{ij} and u_B^{ij} express respectively the down- and up-scattering contributions in block B from energy group j to group i.

For each block B, we have a fixed source problem, and another (inner) iterative method is once more used to calculate the fluxes. At each step, the following block subsystem has to be solved :

$$\mathcal{D}_B \vec{\phi}_B = - \sum_{B' \neq B} \mathcal{C}_{BB'} \vec{\phi}_{B'} + \vec{s}_B = \vec{s}'_B; \quad B = 1, 2, ..., L \quad . \qquad (16)$$

and the fluxes in the regions of B (for all energy groups) are:

$$\vec{\phi}_B^{(k+1)} = \mathbf{D}_B^{-1}[\ \vec{s}_B\,' - \mathbf{L}_B\vec{\phi}_B^{(k+1)} - \mathbf{U}_B\vec{\phi}_B^{(k)}] \tag{17}$$

where $\vec{\phi}_B^{(k+1)}$ and $\vec{\phi}_B^{(k)}$ are respectively the $(k+1)^{th}$ and k^{th} order approximation to the B-fluxes. This level of iteration provides a convergence for fluxes even in energy groups concerned by up-scattering. This level of iteration now constitutes the inner iterations, and we expect the convergence of all the block fluxes $\vec{\phi}_B^{(k)} \rightarrow \vec{\phi}_B$, to yield a new flux map. Since not all the regions are included in each block B, an interesting property of matrices d_B^g is that their spectral radius is much smaller than that of matrices involved in the self-scattering scheme. The multiregion scheme is thus applied to subparts of the geometric domains, and inner iterations act as if the transport problem with fixed sources has to be solved only for each subpart. All these properties can help to improve block convergence, especially in the case of high-scattering regions.

Let us now briefly describe the interprocessor communication necessary to couple the blocks. Our new iterative strategy (for outer iterations) is inserted in the process to take into account the flux-dependent sources. At each outer iteration, only a part of the effective multiplication factor $\mathrm{k}_B^{(n)}$ of n^{th} order of approximation is known inside the block B and is evaluated using the following expression:

$$\mathrm{k}_B^{(n)} = \sum_{i\in B}\sum_{g'} \nu\,\Sigma_{f_i}^{g'}\Phi_i^{g'}. \tag{18}$$

After flux communications between the blocks, the total multiplication factor is then calculated as:

$$\mathrm{k}^{(n)} = \sum_B \mathrm{k}_B^{(n)}, \tag{19}$$

and the sources $\vec{s}_B^{\,'(n)}$; $B = 1, 2, ..., L$ are then reevaluated using multiplication factor and the fluxes thus computed and communicated:

$$s_j^{'(n)} = \sum_{i \notin B} P_{ji}^g \{ \sum_{g'} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'(n)} \} \quad + \quad \sum_i P_{ji}^g \{ \frac{\chi_i^g}{\mathrm{k}^n} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \}$$

$$j \in B; \quad B = 1, 2, ...L. \tag{20}$$

and these sources can be used as input in the next inner iteration process to give a new flux map $\Phi_i^{g(n+1)}$. With this multiregion scheme, at each outer iteration, fission sources are reevaluated as well as contributions of sources coming from other blocks. The inner-outer iterations then proceeds until the convergence of $\mathrm{k}^{(n)} \rightarrow K_{\mathrm{eff}}$ and of the flux distribution.

III. CODE PARALLELIZATION

Let us review some of the steps taken in order to test the multigroup version of DRAGON code potential for parallelization. The first step was to run the sequential code to find which parts were most CPU time consuming. It was found that the flux computation (inner loop) and the calculations of the collision probability matrices, require the most CPU time. This CPU time becomes non-negligible as the number of regions and energy groups grows.

A parallel subset of the code was written in such a way that it can run for an arbitrary number of processors (which becomes an input). This subset of DRAGON code (referred to as DPV-01) contains only the routines necessary to evaluate CP from the DRAGON tracking files and the GOXS macroscopic cross section files. In DPV-01, the flux solver was entirely redesigned, and all attributes useful for single machine operation (such as acceleration and coarse-mesh rebalancing) were discarded. A major criterion of good

programming resides in making a code as portable as possible: only few changes would be necessary in order to change from a parallel environment to another. Furthermore, debugging consists of working with only one processor, making it run successfully and then going further up. This explains our choice of using the public domain PVM available to all kinds of workstations.

III.A. Parallel multigroup solution

The parallel algorithm corresponding to self-scattering reduction assigns each energy group to a different processor.[8] Each processor integrates CP for a set of energy groups. The parallel strategy for computing CP matrices has already been programmed in the TDT code, a subset of APOLLO-2.[9] The loop of CP calculations is done by the **pij** routine:

```
do ig=me+1, ngrp,nproc
   call  pij(...,Pmat(ig))
enddo
```

where **ngrp**, **nproc** and **me**=1,...,**nproc** are the total number of energy groups, the total number of processors and the processor identifier respectively. The processor identifiers are located using the usual PVM routines:[4]

```
call pvmfmytid(mytid)
call pvmfparent(tids(0))
do  i=1, nproc
    if(mytid.eq.tids(i)) me = i
```

```
enddo
```

Each processor then calculates self-scattered matrices associated with its subset of energy ranges. Using the routine **calcul** that will form the left-hand matrices of Eq. (7) and the routine **invers** that will compute the matrix $\mathbf{W}^g$ of Eq. (8), the self-scattering parallel algorithm gives rise to the following loop:

```
do  ig =me+1, ngpr, nproc
    call  calcul(Pmat(ig),...Amat(ig))
    call  invers(Amat(ig),...Wmat(ig))
enddo
```

Assuming an initial one-neutron fission source in the fuel region, an iterative process is started on each processor. At step n of the inverse power method, each processor computes its flux solution and the contribution of its set of energy groups to the total K_{eff} after executing the following loops:

```
do  ig =me+1, ngpr, nproc
        Phi(ig) = Wmat(ig) q(ig)
enddo
Keff(me) =0.
do  ig = me+1, ngrp, nproc
        Keff(me) = Keff(me)+NuSfmat(ig) Phi(ig)
enddo
```

For each processor, the value of the local $K_{\text{eff}}(me)$ and local flux computational results are broadcast to the other nodes. Each processor computes the sum K_{eff}, and is now

able to calculate the source related to its energy subgroup which will be used in the next iteration:

```
do ig = me+1, ngrp, nproc
   q(ig)=0.
   do  ih = 1, ngrp
      q(ig)= q(ig)+ Ssmat(ig,ih ≠ ig) Phi (ih)
      q(ig)= q(ig)+ Chi(ig) NuSfmat(ih) Phi(ih)/ Keff
   enddo
enddo
```

The iterative process is continued until convergence of K_{eff} and of the flux distribution.

III.B. Parallel multiregion iterative solution

The multiregion parallel algorithm assigns each block of regions to a different processor.[10] The CP integration was repeated by each processor, even if it gives the same values. The CP matrices are distributed among the processors so that each of them can calculate its matrices $\mathcal{D}_B$ and $\mathcal{C}_{BB'}$ $(B' \neq B)$, and perform the resolution of Eq.(17) for blocks B (inner iterations). Each processor calculates the contribution of its block to total multiplication factor (see Eq.(18)). The following loop is thus executed:

```
Keff(me)=0.0
do  iB =(me)  Nblk+1, (me+1)  Nblk
      Keff(me)=Keff(me)+NuSfmat(iB) Phi(iB)
enddo
```

where **me** and **Nblk** are the processor identifier and the total number of blocks in the processor **me** respectively. The computation of the sources $\vec{s}_B{}'$ in Eq. (20) requires the interprocessor communication by the algorithm at each outer iteration. As in the previous parallel algorithm, each processor communicates to others nodes its computed fluxes and its own contribution to total K_{eff}. Then each processor calculates its sources $\vec{s}_B{}'$:

```
do iB = (me) Nblk+1, (me+1) Nblk
    sprim(iB) =0.0
    do jB = 1, Nblk
      sprim(iB) =sprim(iB) - C(iB, jB ≠ iB) Phi(jB)
    enddo
    sprim(iB) =sprim(iB) + s(iB)
enddo
```

which will serve as input in the next inner iterations. The iterative process continues until convergence of K_{eff} and of the flux distribution on each processor.

Theoretically, we could balance the work of each processor after scrutinizing the cross section values of materials located in each block; the high-scattering regions are generally slower to converge.

IV. APPLICATIONS

The parallel multigroup and multiregion schemes described above were tested on several types of computers including:

- the IBM SP-2 at École Polytechnique de Montreal, limited to 4 processors;

- a network of 8 Sun Sparc 2 workstations;
- a network of up to 33 IBM RISC/6000-355 computers;
- an 8 processors Sparc 1000 at Centre de Recherches Mathematiques of Montreal University.

In order to isolate the behavior of our parallel iterative schemes, we recall that no acceleration or coarse-mesh rebalancing was done during the following numerical simulations.

IV.A. Test problems

The two iterative schemes were used to integrate probabilities and compute fluxes for a unit cell geometry and the physical data corresponding with one of the Mosteller benchmarks.[12] The unit cell geometry is a simple 2-D square with three concentric annulii inside(see Figure 1). The three indices indicate regions corresponding to the fuel (1), its sheath (2) surrounded by light-water coolant (3).

Starting with a 69-group microscopic cross-section library, we ran the DRAGON code to produce the multigroup fluxes with the unit cell discretized into 4, 8, 16 and 32 regions. When we increase the number of regions, we need to have more tracks. Using the 16-region discretization, we used the angular quadrature of Eq. 4 with $n = 7$ and $d = 10$ tracks/cm, while using the 32-region discretization, we changed the minimal track separation to $d = 20$ increasing the size of the tracking file by a factor of about 2.5.

We then used these 69-group fluxes to condense the various reaction rates and produce macroscopic cross sections with fewer energy groups (33, 18 ... 4 groups). This ensures

that all the numerical simulations will have a similar neutronic behavior.

IV.B. Numerical results and discussion

Let us recall that the speedup value is the ratio between the CPU time required for one processor and the CPU time required for N processors to solve the same problem. Thus the best speedup, called ideal linear speedup, that one can expect is equal to the number of processors. In practice, communications will slow the program and the speedup decreases. This decrease depends on the extent of communications. In order to compare the two schemes (multigroup, multiregion) performance, we give in this paper the speedups using 4 processors of different types of computer.

The test problems were first examined for the multigroup parallel scheme. Table I contains the CPU times obtained for 33 energy groups and 32 regions by using 2, 3 and 4 processors. Table II shows the CPU times for 69 energy groups and 32 regions. The speedup curves associated with Tables I and II are plotted in Figure 2 and Figure 3 respectively. In general, the speedup for the multigroup scheme is almost linear up to four processors. Even if the CPU's of the Sparc 1000 are slower than IBM SP-2, those two figures show that the best speedup is achieved by the Sparc 1000. This is explained by the fact that these processors use a shared memory architecture, and so, message passing is faster and more efficient than Internet communication or IBM SP-2 fast communication links, which is second best.

We have also used 8 processors on a network of IBM RISC/6000-355 computers to represent the variation of speedup for multigroup scheme, with 69 energy groups, when the number of regions increases from 16 to 32 (see Figure 4). We note that the speedup

is almost universally an increasing function of the size of the input problem.[11] Figure 4 shows what we know as the Amdhal effect, which states that if there is not enough work to be done by the number of processors available, then a parallel algorithm would show constrained speedup.

The test problems were also examined for multiregion parallel scheme. We have in Table III the CPU times obtained for 33 energy groups and 32 regions on 1, 2, and 4 processors of different type of computers. Figure 5 represents the speedup in this case. The speedups are still almost linear up to four processors, and are better than the ones obtained for the multigroup scheme (see Figure 6). Figure 7 represents the speedup curves in the parallel multiregion algorithm with 32 regions for 18 and 33 energy groups. We observe that when the number of energy groups (size of the input problem) increases, as in the multigroup scheme, the speedup curve approaches the ideal speedup curve.

The high multiregion/multigroup CPU ratio is not surprising in view of the higher number of the floating point operations involved in the multiregion algorithm. Nevertheless, because for each block of regions all energy groups are treated together, usual methods like neutron rebalancing can be implemented easily to accelerate the multiregion algorithm.

V. CONCLUSION

Numerical results using parallel iterative schemes for solving the multigroup transport equation have been presented. All these schemes use a two-step iterative process (with inner-outer iterations) for finding the critical multiplication factor in a multigroup iterator. In the coming years, fine-group calculations using thousands of regions could be useful

to assess transport and equivalence computations on cross-section libraries over complex geometries. The parallel algorithms developed here will provide users with sufficient resources to perform this kind of benchmarking, without the usual memory and CPU limitations imposed by problem size. The parallel flux solvers using collision probabilities should also provide an attractive alternative to expensive Monte Carlo simulations. It is expected that preliminary results stated here will be improved, over the coming years, by introducing acceleration techniques adapted to the parallel environment in order to speedup convergence.

ACKNOWLEDGMENTS

This work has been carried out partly with the help of grants from Atomic Energy of Canada Ltd and the Natural Science and Engineering Research Council of Canada.

REFERENCES

1. R. Roy, G. Marleau, J. Tajmouati, and D. Rozon, "Modelling of CANDU Reactivity Control Devices with the Lattice Code DRAGON," *Ann. Nucl. Energy* **21**, 115 (1994).

2. G. Marleau, A. Hébert, and R. Roy "DRAGON: A collision Probability Transport Code for cell and Multicell Calculations," Report IGE-100, École Polytechnique de Montreal, Canada (1990).

3. G.A. Geist and V.S. Sunderam, "The Evolution of the PVM Concurrent Computing System," *Proc. 26th IEEE Compcon Symposium*, 471-478, San Francisco USA (1993).

4. A. Beguelin, J.J. Dongarra, G.A. Geist, R. Manchek and V.S. Sunderam, "A Users' Guide to PVM Parallel Virtual Machine, " Report ORNL/TM-11826, Oak Ridge National Laboratory (1991).

5. R. Roy, G. Marleau, A. Hébert, and D. Rozon "A Cyclic Tracking Procedure for Collision Probability Calculations in 2D Lattices," *Int. Top. Mtg. on Advances in Mathematics, Computations and Reactor Physics,* Pittsburgh USA (1991).

6. R. Sanchez and N.J. McCormick, "A Review of Neutron Transport Approximations," *Nucl. Sci. Eng.* **80**, 481 (1982).

7. C.S. Henkel and P.J. Turinsky, "Solution of the Few-Group Diffusion Equation on a Distributed Memory Multiprocessor," *Top. Mtg. on Advances in Reactor Physics,* Charleston USA (1992).

8. A. Qaddouri, R. Roy, G. Marleau, B. Goulard, and M. Mayrand, "Multigroup Flux Solvers on Parallel Architecture,", *Fifteenth CNS Annual Conference*, Montreal Canada (1994).

9. Z. Stankovski, "First Massively Algorithm to be Implemented in Apollo-II Code ", *Int. Conf. on Reactor Physics and Reactor Comp.*, Tel-Aviv, Israel (1994).

10. A. Qaddouri, R. Roy, and B. Goulard, "Multigroup Flux Solvers Using PVM ", *Int. Conf. on Mathematics and Comp. Reactor Phys. and Env. Analyses,* Portland, Oregon USA (1995).

11. S.E. Goodman and S.T. Hedetniemi *Introduction the Design and Analysis of Algorithms,* McGraw-Hill, New York (1977).

12. R.D. Mosteller and *al.*, "Benchmark Calculations for the Doppler Coefficient of Reactivity", *Nucl. Sci. Eng.* **107**, 265 (1991).

Table I. CPU times for the Multigroup Parallel Scheme

(using 33 condensed groups and 32 regions)

Number-processors	IBM-RISC	IBM-SP2	SPARC-2	SPARC-1000
1	126 s	69 s	455 s	302 s
2	72 s	38 s	270 s	156 s
3	52 s	28 s	190 s	116 s
4	41 s	22 s	152 s	89 s

Table II. CPU times for the Multigroup Parallel Scheme

(using 69 energy groups and 32 regions)

Number-processors	IBM-RISC	IBM-SP2	SPARC-2	SPARC-1000
1	320 s	150 s	1524 s	864 s
2	177 s	79 s	876 s	440 s
3	128 s	58 s	617 s	295 s
4	100 s	45 s	493 s	230 s

Table III. CPU times for the Multiregion Parallel Scheme

(using 33 condensed groups and 32 regions)

Number-processors	IBM-RISC	IBM-SP2	SPARC-2	SPARC-1000
1	1104 s	407 s	3637 s	3580 s
2	601 s	216 s	2090 s	1908 s
4	340 s	124 s	1173 s	1013 s

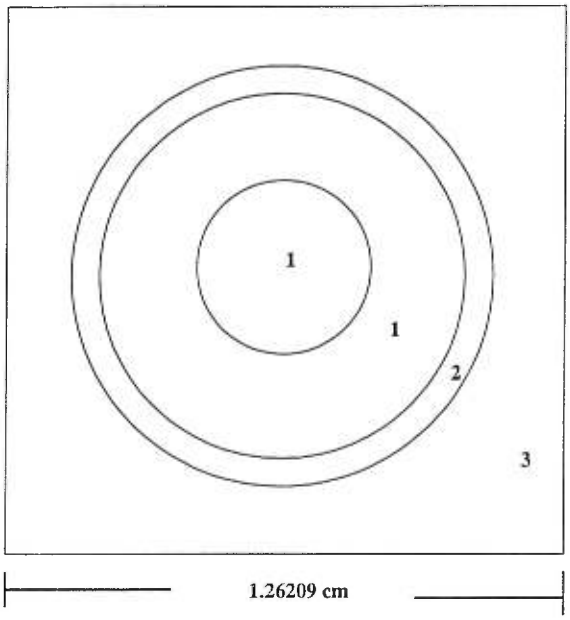

Figure 1. Unit cell geometry used for the test problems. The domain is discretized into 4 regions.

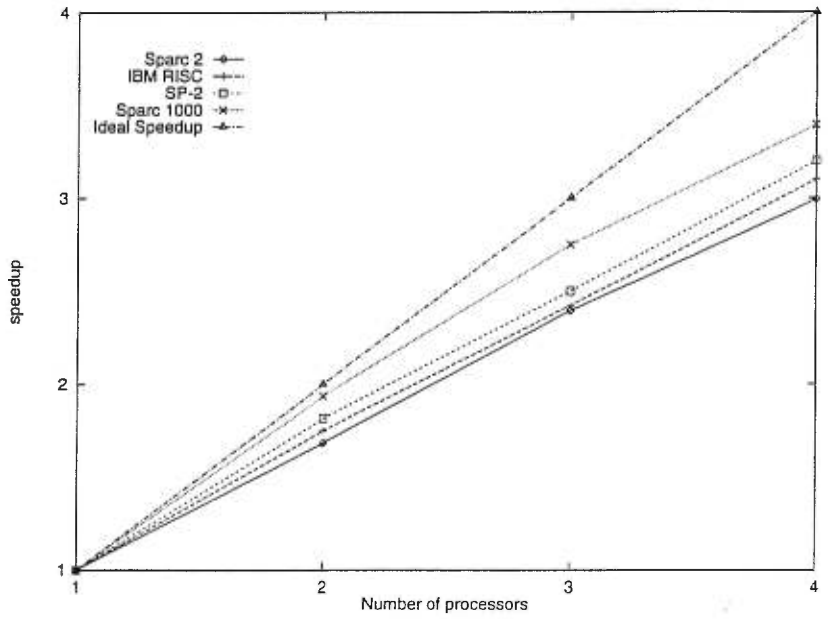

Figure 2. Speedup curves for multigroup parallel scheme. Performance of various types of processors are compared using 33 condensed groups and 32 regions.

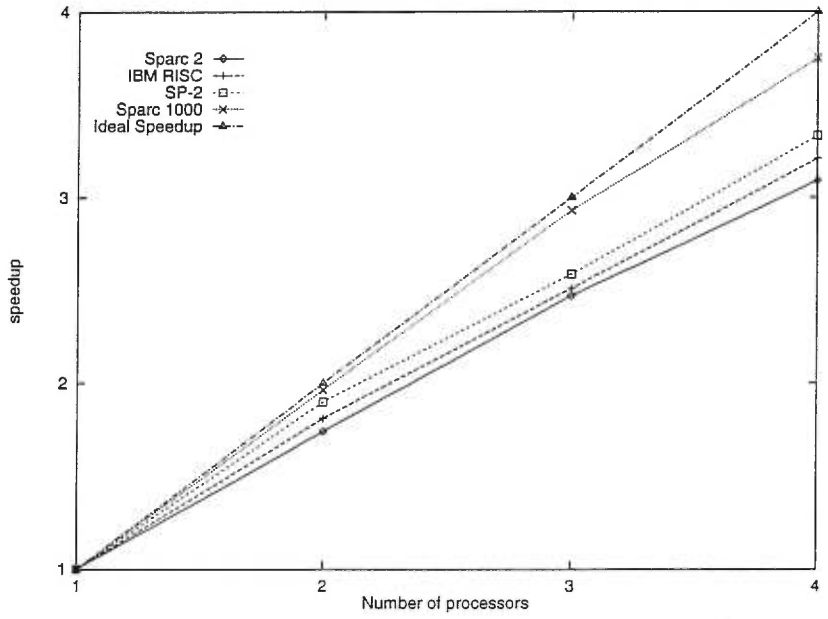

Figure 3. Speedup curves for multigroup parallel scheme. Performance of various types of processors are compared using 69 energy groups and 32 regions.

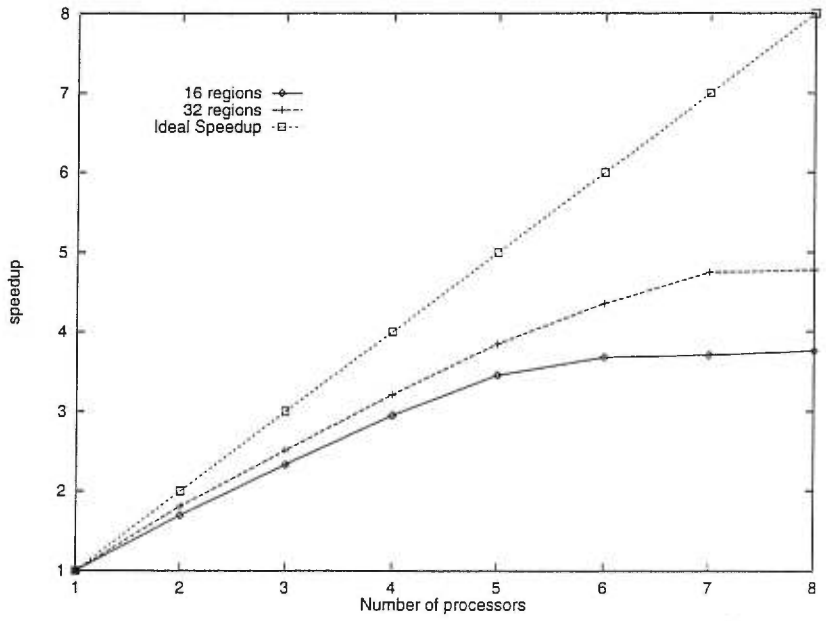

Figure 4. Speedup variation with increasing number of regions for the multigroup parallel scheme. The 69-group problem was run on the RISC network.

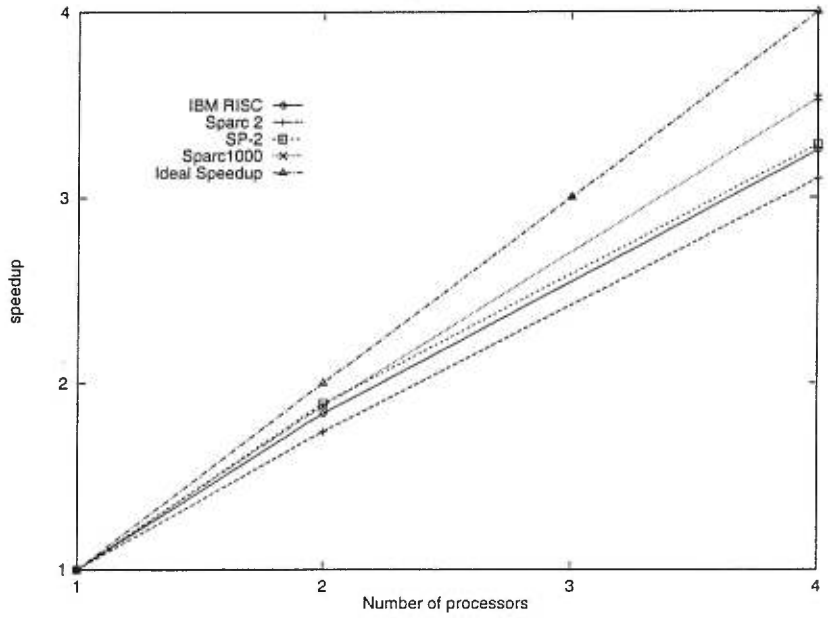

Figure 5. Speedup curves for multiregion parallel scheme. Performance of various types of processors are compared using 33 condensed groups and 32 regions.

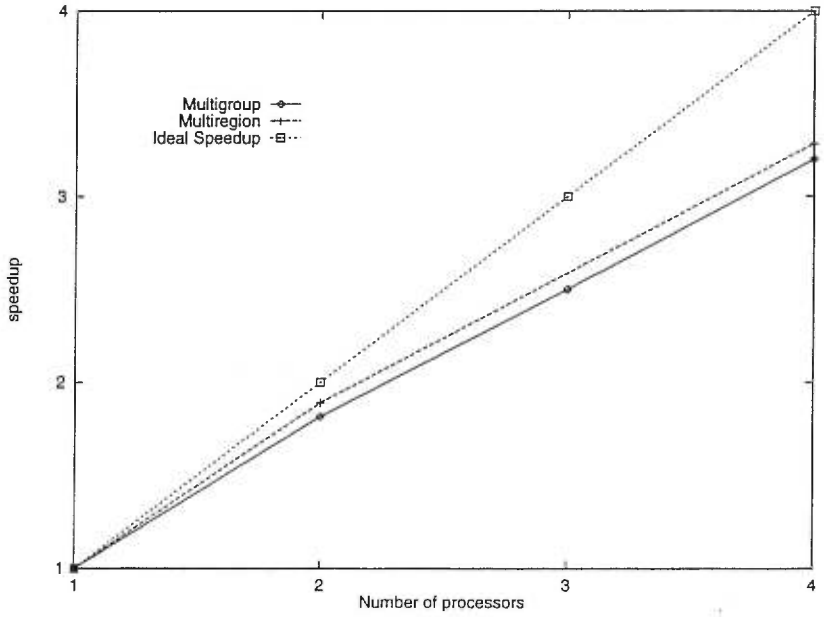

Figure 6. Comparative speedups for both iterative schemes on SP-2. 33 condensed groups and 32 regions were used.

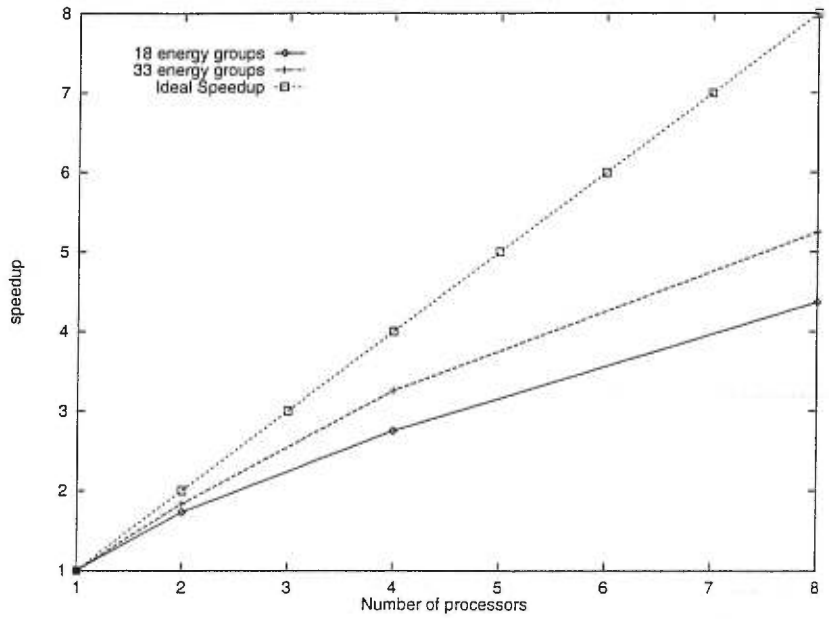

Figure 7. Speedup variation with increasing number of energy groups for the multiregion parallel scheme. The 32-region problem was run on the RISC network.

ARTICLE 4

Parallel Acceleration and Rebalancing Schemes for Solving Transport Problems Using PVM

CNS Fifth International Conference on Simulation Methods in Nuclear Engineering 1996

envoyé en Janvier 96, accepté en Juin 1996 et publié en Septembre 1996

PARALLEL ACCELERATION AND REBALANCING SCHEMES FOR SOLVING TRANSPORT PROBLEMS USING PVM

A. QADDOURI [1,2], R. ROY [1] AND B. GOULARD [2]

[1] *Institut de Génie Nucléaire, École Polytechnique de Montréal*
Montréal, Québec, Canada H3C 3A7

[2] *Centre de Recherches Mathématiques, Université de Montréal*
Montréal, Québec, Canada H3C 3J7
Email: qaddou@hans.crm.umontreal.ca

Abstract — *In this paper, parallelization for CP calculation and multigroup flux computation are presented. Implementation of acceleration and neutron rebalancing strategies is also investigated. Particular techniques pertinent to the two-step energy/space iterative process of solving a multigroup transport equation are described. The parallel performance is studied in cases where the cyclic tracking technique is used to integrate CP. Parallelization is achieved by distributing either different energy groups or different regions on set of processors. These algorithms were tested on 4 processors IBM SP-2, 4 processors SPARC 2000 as well as 8 processors SPARC 1000 using the public domain PVM library. Typical run times are provided for unit cell calculations.*

I. INTRODUCTION

Most Canadian lattice cell codes attempt to solve the transport equation using first-flight collision probabilities. The first step in doing such calculations consists in performing numerical integration (ray tracing) on the cell or supercell geometry. The CPU times can become prohibitive for multigroup evaluation of these probabilities. The standard multigroup flux solver is generally a two-step iterative process using the inverse power method.[1] Fortunately, such algorithms can be parallelized efficiently.

In this paper, we will show how the collision probability (CP) methods are suitable for parallelization. Some particular parallel techniques, pertinent to the energy/space

iterative process of solving a multigroup transport equation were already presented.[1, 2, 3] The speedups observed in such parallel simulations enable consistent calculations using several machines, with limited communication times. However, there still remains the problem of decreasing the global CPU times of the simulations. Now, we intend to focus on acceleration and neutron rebalancing strategies. The motivation is to increase the parallel algorithm performance sufficiently to enable analysis of very large problems (large spatial domain and many energy groups).

This paper thus investigates how acceleration and rebalancing strategy can be implemented to the previous parallel schemes in order to increase their global performance for obtaining a converged flux map. The performance is studied in cases where the cyclic tracking technique is used to integrate CP. These schemes were tested on 4 processors IBM SP-2, 4 processors SPARC 2000 as well as on 8 processors SPARC 1000 using the public domain PVM library. Further typical run times will be provided for some standard cell calculations.

The paper is organized as follows. In Section II we give a detailed description of two differents approaches used to solve multigroup transport equation and the neutron rebalancing and variational accelerations techniques. In Section III the main steps for the parallelization of the equations are discussed, also some communications routines from the parallel environment PVM are briefly commented. Performance results are given in Section IV. Finally, conclusions are drawn in the last section.

II. NUMERICAL METHODS FOR SOLVING MULTIGROUP TRANSPORT EQUATION

In the first two subsections two differents approaches to the multigroup transport equation are investigated. These two methods involve two iteration processes (an inner iteration to treat the flux and an outer iteration for sources and multiplication factor calculations). Two acceleration methods that are used to improve the convergence speed of the inner iteration are the object of the two last subsections.

II.A. Self-scattering reduction scheme (multigroup)

A usual operation performed before solving the linear transport system on **sequential architecture** is called self-scattering reduction of the collision probability matrices. Using this scheme, all scattering information pertinent to group g is transferred to the left-side of the equation, so that the system to be solved is in the form:[1]

$$V_j\Phi_j^g - \sum_i P_{ji}^g \Sigma_{si}^{g\leftarrow g}\Phi_i^g = \sum_i P_{ji}^g \{ \sum_{g'\neq g} \Sigma_{si}^{g\leftarrow g'}\Phi_i^{g'} + \frac{\chi_i^g}{K_{\text{eff}}} \sum_{g'} \nu\Sigma_{f_i}^{g'}\Phi_i^{g'} \}, \qquad (1)$$

where P_{ji}^g is an element of the group g matrix, representing the first-flight reduced probability from region i to region j which has been multiplied by the volume V_j of region j; the CP reciprocity relations assess the symmetry of $\mathbf{P^g}$ matrices. The usual macroscopic cross sections (Σ_s and Σ_f) are formed by combining the microscopic cross sections of various isotopes

order I, we will have to compute the scattering-reduced matrix $\mathbf{W^g}$ for every energy group :

$$\mathbf{W^g} = [\mathbf{V} - \mathbf{P^g}\boldsymbol{\Sigma}_\mathbf{s}^{\mathbf{g}\leftarrow\mathbf{g}}]^{-1}\,\mathbf{P^g}, \tag{2}$$

so that the linear system in Eq. (1) can be simplified to:

$$\Phi_j^g = \sum_i W_{ji}^g q_i^g \ , \tag{3}$$

where the new external sources are now coming only from *other* groups and are given by:

$$q_i^g = \sum_{g' \neq g} \Sigma_{si}^{g\leftarrow g'} \Phi_i^{g'} + S_i^g \ , \tag{4}$$

and fission neutrons are still provided by:

$$S_i^g = \frac{\chi_i^g}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} . \tag{5}$$

The system of Eqs. (3)–(5) is solved by inverse power method assuming a starting value corresponding to a fission source vector $\vec{S}$. This is the numerical scheme corresponding to a standard way of solving the multigroup problem when using the integral transport equation.[4].

However, the self-scattering reduction is not necessarily the only way of solving the linear transport system on **sequential architecture** as we will see in the next subsection.

II.B. Spatial recomposition scheme (multiregion)

Another completely different approach which consists of a two-step iterative process is presented. This original scheme of partitioning the spatial domain instead of the energy domain has been initiated by Henkel and Turinsky in order to solve the few-group diffusion equation.[5].

We partition the spatial domain into L non-overlapping set of regions, and reorder the flux and the source vector so that, for each set B (blocks $B = 1, 2, ..., L$) of regions, we reorder unknowns to form block-dependent and global flux and source vectors:

$$\begin{aligned} \vec{\phi}_B &= \{\Phi_i^g;\ g = 1, 2, ..., G;\ i \in B\}, \quad \vec{\phi} = \oplus_B \vec{\phi}_B, \\ \vec{s}_B &= \{s_i^g;\ g = 1, 2, ..., G;\ i \in B\}, \quad \vec{s} = \oplus_B \vec{s}_B. \end{aligned} \tag{6}$$

where $\oplus_B$ represents a direct sum used to concatenate all the unknowns together.

The system to be solved is rewritten in the form:

$$\begin{aligned} [\ V_j\Phi_j^g - \sum_{i\in B} P_{ji}^g \{\sum_{g'} \Sigma_{si}^{g\leftarrow g'} \Phi_i^{g'}\}\ ] \ - \ & \sum_{i\notin B} P_{ji}^g \{\sum_{g'} \Sigma_{si}^{g\leftarrow g'} \Phi_i^{g'}\} \\ = \ & \sum_i P_{ji}^g \{\frac{\chi_i^g}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'}\} \\ & j \in B; \quad B = 1, 2, ...L. \end{aligned} \tag{7}$$

The updated unknowns from block $B' \neq B$ are unavailable to the processor responsible for block B, so the usual inverse power iterator have to be slightly changed. Collecting coefficients pertinent to each block B, this system of equations is now iteratively solved as:

$$\mathcal{D}_B \Phi_B^{(n+1)} + \sum_{B' \neq B} \mathcal{C}_{BB'} \Phi_{B'}^{(n)} = s_B^{(n)}; \quad B = 1, 2, ..., L, \tag{8}$$

where we have used matrix form and where $\mathcal{D}_B$ is a one-block multigroup matrix which can be split in the following form:

$$\mathcal{D}_B = \begin{pmatrix} d_B^1 & u_B^{12} & . & . & . & u_B^{1G} \\ l_B^{21} & d_B^2 & . & . & . & u_B^{2G} \\ . & & . & & & \\ & . & & . & & \\ & & & & . & \\ l_B^{G1} & l_B^{G2} & . & . & & d_B^G \end{pmatrix} = \mathbf{D}_B + \mathbf{L}_B + \mathbf{U}_B. \tag{9}$$

The submatrices l_B^{ij} and u_B^{ij} express respectively the down- and up-scattering contributions in block B from energy group j to group i.

For each block B, we have a fixed source problem, and another (inner) iterative method is once more used to calculate the fluxes. At each step, the following block subsystem has to be solved :

$$\mathcal{D}_B \vec{\phi}_B = - \sum_{B' \neq B} \mathcal{C}_{BB'} \vec{\phi}_{B'} + \vec{s}_B = \vec{s}'_B; \quad B = 1, 2, ..., L. \tag{10}$$

and the fluxes in the regions of B (for all energy groups) are:

$$\vec{\phi}_B^{(k+1)} = \mathbf{D}_B^{-1} [\, \vec{s}'_B - \mathbf{L}_B \vec{\phi}_B^{(k+1)} - \mathbf{U}_B \vec{\phi}_B^{(k)}]. \tag{11}$$

where $\vec{\phi}_B^{(k+1)}$ and $\vec{\phi}_B^{(k)}$ are respectively the $(k+1)^{th}$ and k^{th} order approximation to the B-fluxes. This level of iteration provides a convergence for fluxes even in energy groups concerned by up-scattering. Since not all the regions are included in each block B, an interesting property of matrices d_B^g is that their spectral radius is much smaller than the ones of matrices involved in the self-scattering scheme. The multiregion scheme is thus applied to subparts of the geometric domains, and Eq. (11) acts as if the transport problem with fixed sources has to be solved only for each subpart. All these properties can help to improve blockconvergence, especially in the case of high-scattering regions.

At each flux calculation in Eq. (11) each processor communicates to others their local computed flux. The term of scattering in $\vec{s}'_B$ is reevaluated for including the contributions of sources coming from other blocks. This level of iteration now constitutes the inner iterations, and we expect the convergence of all the block fluxes $\vec{\phi}_B^{(k)} \rightarrow \vec{\phi}_B$, to yield a new flux map.

Our new iterative strategy (for outer iterations) is inserted in the process to take into account the flux-dependent fission sources. At each outer iteration, only a part of the effective multiplication factor $k_B^{(n)}$ of n^{th} order approximation is known inside the block B and is evaluated using the following expression:

$$k_B^{(n)} = \sum_{i \in B} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'}. \tag{12}$$

After communications of local computed k_B between the blocks, the total multiplication factor is then calculated as:

$$k^{(n)} = \sum_B k_B^{(n)}, \tag{13}$$

and the fission sources $\vec{s}_B^{(n)}$; $B = 1, 2, ..., L$ are then reevaluated using multiplication factor and the fluxes thus computed and communicated:

$$s_j^{(n)} = \sum_i P_{ji}^g \{ \frac{\chi_i^g}{k^{(n)}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'(n)} \}$$

$$j \in B; \quad B = 1, 2, ...L. \tag{14}$$

and these sources can be used as input in the next inner iteration process to give a new flux map $\Phi_i^{g(n+1)}$. The inner-outer iterations then proceed until the convergence of $k^{(n)} \to K_{\text{eff}}$ and of the flux distribution.

II.C. Rebalancing scheme

Rebalancing is a popular scheme applied to production codes in order to accelerate inner iterations. This scheme is introduced to deal with the problem of neutron transfers between various groups due to scattering. We rewrite Eq.(10) (or Eq.(3)) in following matrix form:

$$\Phi^{g(k)} = H^g \{ [\sum_{g>g'} \Sigma_s^{g \leftarrow g'} \Phi^{g'(k)} + \sum_{g<g'} \Sigma_s^{g \leftarrow g'} \Phi^{g'(k-1)}] + S^g \}, \tag{15}$$

where H^g is the explicit linear operator for the group solver. We observe that in both previous flux solvers schemes, fluxes become unbalanced because up-scattered neutrons are not taken directly into account during the inner iteration.

The objective of the flux rebalancing method is to enhance the rate of convergence by imposing neutron conservation after every iteration, and thus to force the equality:

$$\sum_i \{ \Sigma_{t,i}^g V_i \Phi_i^g - \sum_{g'} \Sigma_{si}^{g \leftarrow g'} V_i \Phi_i^g \} = \sum_i \chi_i^g F_i^g V_i. \tag{16}$$

where Σ_t is the total cross section, and F_i^g is given by:

$$F_i^g = \frac{1}{K_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'}. \tag{17}$$

We compute for each unconverged energy group an average rebalancing factor α^g such that the rebalanced flux $\bar{\Phi}_i^g$ defined as:

$$\bar{\Phi}_i^g = \alpha^g \Phi_i^g, \tag{18}$$

satisfies exactly Eq. (16). We obtain the following system:

$$\sum_i \{ [\Sigma_{t,i}^g - \Sigma_{si}^{g \leftarrow g}] V_i \Phi_i^g \alpha^g - \sum_{g'(unconverged) \neq g} \Sigma_{si}^{g \leftarrow g'} V_i \Phi_i^{g'} \alpha^{g'} \} \tag{19}$$

$$= \sum_i \{ \chi_i^g F_i^g V_i + \sum_{g'(converged) \neq g} \Sigma_{si}^{g \leftarrow g'} V_i \Phi_i^{g'} \}.$$

This latter system is solved after each inner iteration. The computed α^g are used to update rebalanced fluxes $\bar{\Phi}^g$ for the next inner iteration.

II.D. Variational relaxation scheme

The next acceleration method we will introduce is a variational relaxation method. Assuming a fixed source S obtained from an outer iteration, in general we can rewrite the equation of the flux, in the inner iteration, in the following matrix form:

$$\Phi^{(n+1)} = H\Phi^{(n)} + S. \tag{20}$$

The relaxation method, that we will use, consists in using an extrapolation formula as:

$$\Phi^{(n+1)} = \Phi^{(n)} + \mu R^{(n+1)}, \tag{21}$$

where μ and R are the acceleration parameter and the residual respectively. In ordinary over-relaxation, this acceleration parameter is fixed to a value greater than one for all inner iterations. Here, we will introduce a variable acceleration parameter $\mu^{(n)}$ which can be computed efficiently after each iteration using a variational method.[6]

The principle of the variational method is, at each iteration , to take in Eq. (21) the value of μ that minimizes $||R^{(n+1)}||$, where

$$R^{(n+1)} = R^{(n)} + \mu[\ H\ R^{(n)} - R^{(n)}\ ]. \tag{22}$$

Suppose that we estimate the residual using an L_2-norm

$$\begin{aligned} ||R^{(n+1)}||^2 &= < R^{(n+1)}, R^{(n+1)} > = < R^{(n)}, R^{(n)} > + 2\mu < R^{(n)}, HR^{(n)} - R^{(n)} > \\ &+ \mu^2 < HR^{(n)} - R^{(n)}, HR^{(n)} - R^{(n)} >, \end{aligned} \tag{23}$$

the condition of minimizing is $\frac{d}{d\mu}||R^{(n+1)}||^2 = 0$.

At each iteration n, the following optimum value of μ is obtained:

$$\mu^{(n)} = -\frac{< R^{(n)}, HR^{(n)} - R^{(n)} >}{||HR^{(n)} - R^{(n)}||^2}. \tag{24}$$

To illustrate this acceleration process, suppose that we have performed three inner iterations without acceleration:

$$\begin{aligned} \Psi_1 &\leftarrow \Phi^{(n)}, \\ \Psi_2 &= H\Psi_1 + S, \\ \Psi_3 &= H\Psi_2 + S, \end{aligned}$$

then, we can define:

$$\begin{aligned} e_1 &= \Psi_2 - \Psi_1 = H\Psi_1 + S - \Psi_1 = R^{(n)} \\ e_2 &= \Psi_3 - \Psi_2 = H\Psi_2 + S - \Psi_2. \end{aligned}$$

It can be easily shown that:

$$\mu^{(n)} = -\frac{< e_1, e_2 - e_1 >}{< e_2 - e_1, e_2 - e_1 >}. \tag{25}$$

Inner convergence implies that, for n large enough, $|e_2| < |e_1|$ with $sign(e_1) = sign(e_2)$ leading to $\mu^{(n)} > 0$ (often $\mu^{(n)} >> 1$). At each inner iteration, we can thus decide to

apply the acceleration factor or not. Note also that the CPU overhead associated with acceleration is minimal because, using the accelerated flux computed as:

$$\tilde{\Phi}^{(n)} = \mu^{(n)}\Psi_2 + (1-\mu^{(n)})\Psi_1, \tag{26}$$

one can also obtained the next unaccelerated estimate as:

$$\Phi^{(n+1)} = \mu^{(n)}\Psi_3 + (1-\mu^{(n)})\Psi_2, \tag{27}$$

without doing a new inner iteration. The whole acceleration process is very efficient because it always keeps three last iterates of the flux on a stack, and decides if the last two are accelerated or not.

III. CODE PARALLELIZATION

A parallel subset of the DRAGON [7] code was written in such a way that it can run for an arbitrary number of processors (which becomes an input). This subset (referred to as DPV-01) contains only the routines necessary to evaluate CP from the DRAGON tracking files and the GOXS macroscopic cross section files. In DPV-01, the flux solver was entirely redesigned. A major criterion of good programming resides in making a code as portable as possible: only few changes would be necessary in order to change from a parallel environment to another. Furthermore, debugging consists of working with only one processor, making it run successfully and then going further up. This explains our choice of using the public domain PVM available for all kinds of workstations.

III.A. Parallel multigroup solution

The parallel algorithm corresponding to self-scattering reduction scheme (II.A.) assigns each energy group to a different processor.[2] Each processor integrates CP for a set of energy groups. This parallel strategy for computing CP matrices has also been programmed in the TDT code, a subset of APOLLO-2.[8] The loop of CP calculations is done by the **pij** routine:

```
do ig=me+1, ngrp,nproc
   call  pij(...,Pmat(ig))
enddo
```

where **ngrp**, **nproc** and **me**=1,...,**nproc** are the total number of energy groups, the total number of processors and the processor identifier respectively.

Each processor then calculates self-scattered matrices associated with its subset of energy ranges. Using the routine **calcul** that will form the left-hand matrices of Eq. (1) and the routine **invers** that will compute the matrix $\mathbf{W}^g$ of Eq. (2), the self-scattering parallel algorithm gives rise to the following loop:

```
do  ig =me+1, ngpr, nproc
    call  calcul(Pmat(ig),...Amat(ig))
    call  invers(Amat(ig),...Wmat(ig))
enddo
```

Assuming an initial flux map, an outer iterative process is started. A fission source is calculated by each processor on its corresponding set of energy groups. An inner iterative process is started on each processor.

At step k of the inner iterations, each processor computes its local flux solution:

```
do  ig =me+1, ngpr, nproc
        Phi(ig) = Wmat(ig) q(ig)
enddo
```

These fluxes are broadcast to the other nodes. Each processor then rebalances all the unconverged energy groups by performing the resolution of the system in Eq. (18). Rebalancing is followed by calling acceleration routine:

```
do  me =0, nproc-1
    call  FLUBAL(me,Phi,...Phibal)
    Phi=Phibal
    call  FLUACC(me,Phi,...Phiacc)
    Phi=Phiacc
enddo
```

The inner iterations are continued until convergence of flux distribution.

Then the outer iteration continues, and each processor calculates the contribution of its set of energy groups to the total K_{eff} after executing the following loops:

```
Keff(me) =0.
do  ig me+1, ngrp, nproc
      Keff(me) = Keff(me)+NuSfmat(ig) Phi(ig)
enddo
```

For each processor, the value of the local $K_{\text{eff}}(me)$ is broadcasted to the other nodes. Each processor computes the sum K_{eff}, and updates the fission source related to its energy subgroup which will be used in the next inner iterations.

The iterative process is continued until convergence of K_{eff} and of the flux distribution.

III.B. Parallel multiregion iterative solution

The multiregion parallel algorithm assigns each block of regions to a different processor.[1] The CP integration is repeated by each processor, even if it gives the same values. This CPU overhead is the price to pay if we do not want to classify the tracks by block; this classification is theoretically possible but, with the cyclic tracking method which uses specular reflexion, most cell calculations are done using very long tracks, that will intersect many regions, and thus many blocks. This explains why we compute the same complete probability matrices by each processor.

The CP matrices are distributed among the processors so that each of them can calculate its matrices $\mathcal{D}_B$ and $\mathcal{C}_{BB'}$ $(B' \neq B)$.

Assuming an initial flux map, an outer iteration starts in each processor. Initial fission terms in $\vec{s}_B{}'$ sources are calculated and used as input in the inner iterations. At each inner iteration each processor calculates its local block flux distribution by performing the resolution of eq.(10) for blocks B (inner iterations). After intercommunication of fluxes, rebalancing and acceleration are then done in a sequential way by each processor. The inner iteration continues until convergence of global flux distribution with the initial fission terms in $\vec{s}_B{}'$.

Then, each processor calculates the contribution of its block to total multiplication factor (see Eq.(12)). The following loop is thus executed:

```
Keff(me)=0.0
do  iB =(me)  Nblk+1, (me+1)  Nblk
      Keff(me)=Keff(me)+NuSfmat(iB) Phi(iB)
enddo
```

where **me** and **Nblk** are the processor identifier and the total number of blocks in the processor **me** respectively. As in the previous parallel algorithm, each processor communicates to others nodes its own contribution to total K_{eff}, and reevaluates the fission terms in the sources $\vec{s}_B{}'$ in Eq. (14) :

```
do iB  = (me)  Nblk+1, (me+1)  Nblk
       s(iB) =0.0
       do jB = 1, Nblk
         s(iB) =s(iB) + P(iB,JB) XSCHI(jB) NuSfmat(jB) Phi(jB)
       enddo
enddo
```

which will serve as input in the next inner iterations. The iterative process continues until convergence of K_{eff} and of the global flux distribution on each processor.

III.C. Communication between processors

The processors use standard routines of PVM to communicate data one to onother. To illustrate this, we give here, a brief description of a subset of code which provide intercommunication of fluxes for parallel multigroup scheme:

```
ngrpro = 0
do  ig =(me+1),ngrp,nproc
       igg=ig
       call pvmfpack(...,igg,...)
       call pvmfpack(...,Phi(...,igg),...)
       ngrpro= ngrpro+1
enddo
do II = 1,nproc-1
       call pvmfsend(tids(mod(me+II,nproc)),2,...)
enddo
do II = 1,nproc-1
       call pvmfrecv(-1,2,...)
```

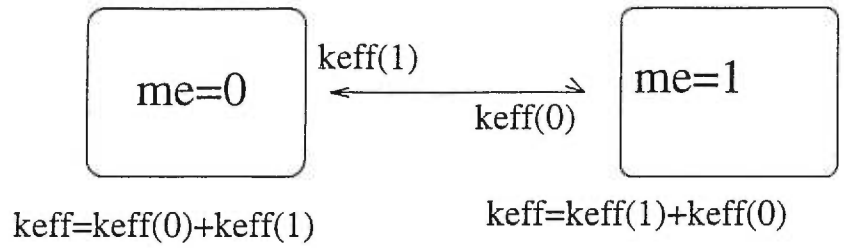

Figure 1: intercommunication of local critical multiplication factor

```
do IJ =1,ngrpro
call pvmfunpack(...,igg,...)
call pvmfunpack(...,Phi(...,igg),nr,...)
enddo
enddo
```

where **nr** and **ngrpro** are the total number of spatial regions and the number of energy groups in the the processor **me** respectively. The routines **pvmfpack** pack the data into the active send buffer and the routines **pvmfunpack** unpack the data from the active receive buffer. The routine **pvmfsend** sends a message to other processor, **pvmfsend** has the identifier of the recipient processor and the message type as variables. The variable -1 in the routine **pvmfrecv** means that messages coming from any other processor can be accepted, and the second variable 2 is the message type.

As for fluxes each processor communicates to others nodes its local computed $K_{\text{eff}}(me)$ (see Figure 1), and each processor calculates the sum K_{eff}.

IV. APPLICATIONS

The parallel multigroup and multiregion schemes described above were tested on several types of computers including:

- the IBM SP-2 at École Polytechnique de Montréal, limited to 4 processors;
- Sparc 2000 at École Polytechnique de Montréal,limited to 4 processors;
- an 8 processors Sparc 1000 at Centre de Recherches Mathématiques of université de Montréal.

IV.A. Test problems

The two parallel iterative schemes were used to integrate probabilities and compute fluxes for a unit cell geometry and the physical data corresponding with one of the Mosteller benchmarks.[9] The unit cell geometry is a 2-D square with three concentric annuli inside. There are three different material regions corresponding to the fuel, its sheath surrounded by light-water coolant. This cell problem is further divided to obtain 32 zones.

IV.B. Numerical results and discussion

For the variational acceleration, we alternate between sets of non-accelerated and accelerated iterations. The reason for this is to compensate for instabilities arising due to non-fondamental modes excited by this variational acceleration technique. In fact after numerical testing, it was found that one should alternate between 3 non-accelerated and 3 accelerated iterations ((3,3) acceleration scheme).

The test problems were examined for both multigroup and multiregion parallel schemes. In order to compare parallel schemes and hardware, we choose to present a 69 groups problem. Table 1 contains the CPU times obtained by using multigroup parallel scheme for 1,2, 3 and 4 processors (with and without (rebalancing+variational acceleration)). Table 2 shows the CPU for 1, 2 and 4 processors (with and without (rebalancing+variational acceleration)) by using multiregion parallel scheme. As one can see, the CPU time necessary for converging the solution of the transport equation, in both schemes, is reduced by using the acceleration and neutron rebalancing techniques. As can be seen, the CPU times observed for both methods using the IBM SP2 with 4 processors are extremely competitive with the best available tools for solving this kind of transport problems.

In Table 3 we present the number of iterations necessary for converging in both methods. We remark that the use of acceleration and neutron rebalancing dramatically decreases the number of iterations needed for convergence, and consequently reduces the communication between processors.

Let us recall that the speedup value is the ratio between the CPU time required for one processor and the CPU time required for N processors to solve the same problem. Thus the best speedup, called ideal linear speedup, that one can expect is equal to the number of processors. In practice, communications are slowing the program and the speedup decreases. This decrease depends on the extent of communications.

We note (see Figure 6-9) that the use of acceleration and neutron rebalancing techniques in both methods improve the speedups. The reason for this is that in the accelerated schemes the computing time between two successive communications of fluxes between processsors is increased by the time needed for acceleration and neutron rebalancing techniques. The ratio between communication time and computation time decreases and then the speedup increases.[10]

Even if the CPU's of Sparc 1000 (or 2000) are slower than those of IBM SP-2, the figures show that the best speedup is achieved by Sparc 1000 (or 2000). This is explained by the fact these processors use a shared memory architecture, and so, message passing is faster and more efficient than IBM SP-2 fast communication links.

The high multiregion/multigroup CPU ratio, displayed by comparison of Table 1 and Table 2, is not surprising in view of the higher number of floating point operations involved in the multiregion algorithm. However, this scheme can be useful in distributing among several processors the amount of memory required for transport calculations involving a very large number of regions.

V. CONCLUSION

Numerical results using parallel iterative schemes for solving the multigroup transport equation have been presented. All these schemes use a two-step iterative process (with inner-outer iterations) for finding the critical multiplication factor in a multigroup iterator.

In the coming years, fine-group calculations using thousands of regions could be useful to assess transport and equivalence computations on cross-section libraries over complex geometries. The parallel algorithms developed here will provide users with sufficient resources to perform this kind of benchmarking, without the usual memory and CPU limitations imposed by problem size.

Acknowledgements— This work has been carried out partly with the help of grants from Atomic Energy of Canada Ltd and the Natural Science and Engineering Research Council of Canada. The authors would like to thank Vincent Cavuoti, from École Polytechnique de Montréal, who gave us access to IBM SP2 and Sparc 2000

REFERENCES

[1] A. Qaddouri, R. Roy, M. Mayrand, and B. Goulard, "Collision Probability Calculation and Multigroup Flux Solvers Using PVM ",*Nucl. Sci. Eng.*, **123**, 392 (1996)

[2] A. Qaddouri, R. Roy, and B. Goulard, "Multigroup Flux Solvers Using PVM ", *Proc.Int. Conf. on Mathematics and Computations, Reactor Physics and Environmental Analyses*, Portland, Oregon, April 30-May 4, 1995, American Nuclear Society (1995).

[3] E. Fuentes and P.J. Turinsky, "Parallel Implementation of Integral Transport Methods," *Nucl. Sci. Eng.*, **121**, 277 (1995).

[4] R. Sanchez and N.J. McCormick, "A Review of Neutron Transport Approximations," *Nucl. Sci. Eng.* **80**, 481 (1982).

[5] C.S. Henkel and P.J. Turinsky, "Solution of the Few-Group Diffusion Equation on a Distributed Memory Multiprocessor," *Top. Mtg. on Advances in Reactor Physics*, Charleston USA (1992).

[6] M. Livolant, "Méthodes itératives simples pour la résolution de problèmes linéaires," Report SPM-706, CEN Saclay (1968).

[7] G. Marleau, A. Hébert, and R. Roy "DRAGON: A Collision Probability Transport Code for cell and Multicell Calculations," Report IGE-100, École Polytechnique de Montreal, Canada (1990).

[8] Z. Stankovski, "A Massively Parallel Algorithm for the Collision Probability Calculations in Apollo-II Code Using the PVM Library", *Proc.Int. Conf. on Mathematics and Computations, Reactor Physics and Environmental Analyses. Analyses*, Portland, Oregon, April 30-May 4, 1995, American Nuclear Society (1995).

[9] R.D. Mosteller and *al.*, "Benchmark Calculations for the Doppler Coefficient of Reactivity", *Nucl. Sci. Eng.* **107**, 265 (1991).

[10] S.E. Goodman and S.T. Hedetniemi *Introduction the Design and Analysis of Algorithms*, McGraw-Hill, New York (1977).

Table 1: CPU times for the Multigroup Parallel Scheme
without/with rebalancing and acceleration
(using 69 energy groups and 32 regions)

Number-processors	SPARC-2000	IBM-SP2	SPARC-1000
1	94 s/72 s	25 s/18 s	691 s/445 s
2	58 s/39 s	17 s/11 s	453 s/253 s
3	44 s/28 s	13 s/ 8 s	345 s/179 s
4	40 s/24 s	11 s/ 7 s	300 s/152 s

Table 2: CPU times for the Multiregion Parallel Scheme
without/with rebalancing and acceleration
(using 69 energy groups and 32 regions)

Number-processors	SPARC-2000	IBM-SP2	SPARC-1000
1	265 s/88 s	75 s/25 s	3800 s/1257 s
2	170 s/54 s	52 s/17 s	2520 s/788 s
4	99 s/32 s	32 s/10 s	1440 s/466 s

Table 3: number of iterations for both Parallel Schemes
without/with acceleration and rebalancing
(using 69 energy groups and 32 regions)

	parallel multigroup algorithm	parallel multiregion algorithm
inner iterations	263/17	217/36
outer iterations	10/3	8/5

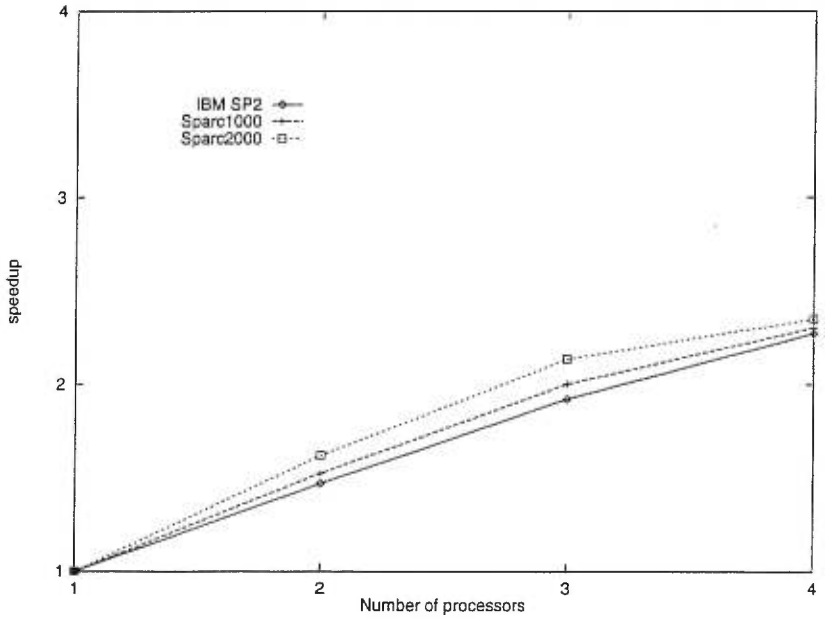

Figure 2: Speedup curves for unaccelerated multigroup parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

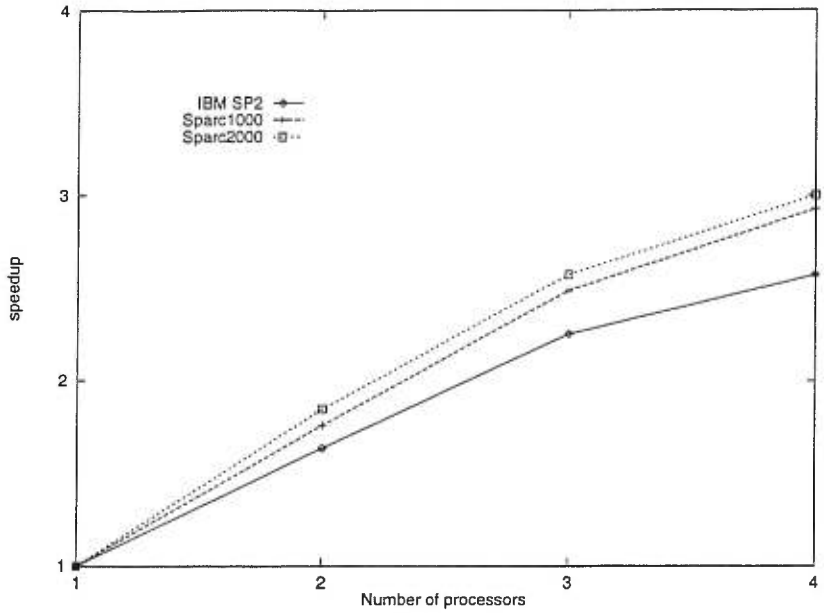

Figure 3: Speedup curves for accelerated multigroup parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

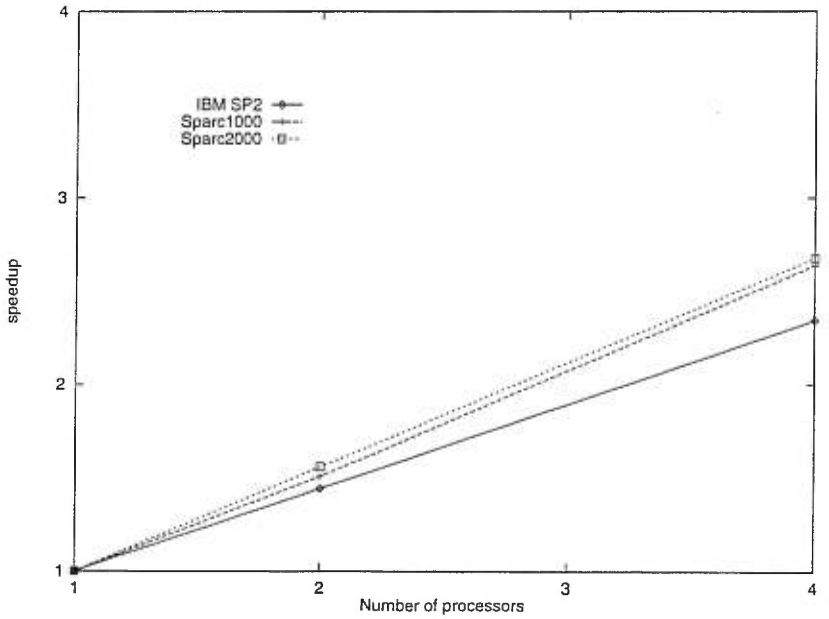

Figure 4: Speedup curves for unaccelerated multiregion parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

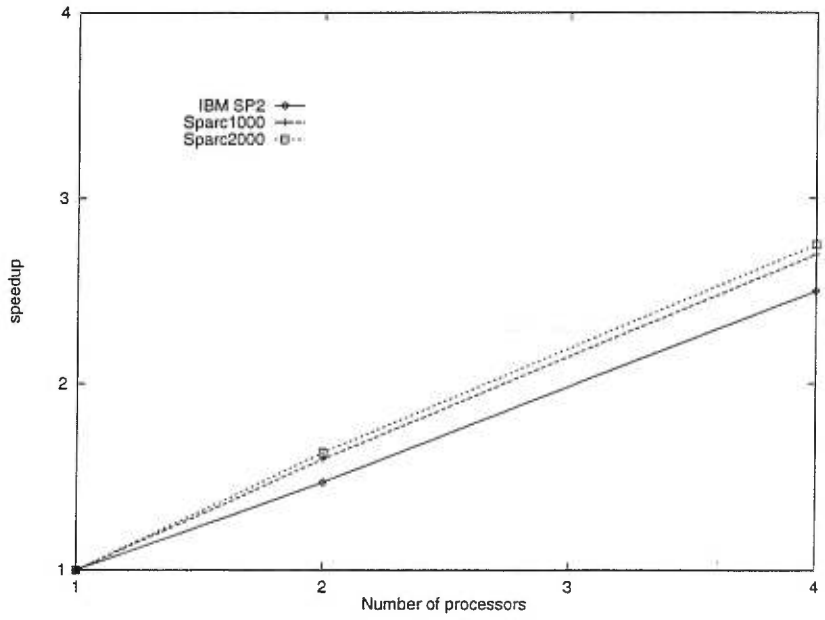

Figure 5: Speedup curves for accelerated multiregion parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

ARTICLE 5

Parallel Algorithms for Solving Transport Problems in Cell and Supercell Codes

International Conference on Mathematical Methods and Supercomputing for Nuclear Applications 1997

envoyé en Janvier 1997, accepté en Mai 1997 et publié en Octobre1997

PARALLEL ALGORITHMS FOR SOLVING TRANSPORT PROBLEMS IN CELL AND SUPERCELL CODES

R. Roy [1], A. Qaddouri [1,2] and B. Goulard [2]

[1] *Institut de Génie Nucléaire, École Polytechnique de Montréal*
Montréal, Québec, Canada H3C 3A7
e-mail: roy@meca.polymtl.ca

[2] *Centre de Recherches Mathématiques, Université de Montréal*
Montréal, Québec, Canada H3C 3J7

Abstract – Some parallel techniques pertinent to the iterative process for solving multigroup transport problems using the collision probability technique are described. The most natural way to implement parallelism for calculating the collision probabilities is by distributing the set of energy groups among several processors. A simple parallel multigroup flux solution is achieved by exchanging the partial flux values produced by all processors until convergence. Sustained by acceleration and rebalancing strategies to minimize the number of iterations, this multigroup parallel scheme was implemented in the new EXCELL module of DRAGON for calculating three-dimensional problems. Using the PVM software, typical run times are provided for such problems. Speedups obtained with the previous multigroup scheme are also compared for unit cell calculations with another scheme where we distribute different regions on the set of available processors.

I. INTRODUCTION

Solving the neutron transport equation on complex geometries is still one of the most time-consuming process for modern computer resources. Since the early stages of reactor physics, the most powerful computers were used for solving such problems in laboratories all over the world, but code developers are still limited by the memory size and the disk space available on a single machine. Most lattice codes attempt to solve the transport equation using first-flight collision probabilities. The first step in doing such calculations consists of performing numerical integration (ray tracing) on the cell or supercell geometry. The collision probability technique then leads to full matrices of coefficients coupling the group-dependent contributions of sources in every region to the flux in others. Unfortunately, the CPU times can become prohibitive for evaluating these probabilities in a multigroup context. Moreover, the active memory needed for storing these full group-dependent matrices can limit the use of such a technique for solving large problems.

In this paper, we will show how the collision probability (CP) methods can be efficiently parallelized. Some particular parallel techniques, pertinent to the energy/space iterative process of solving a multigroup transport equation were already presented.[1, 2, 3] The speedups observed in such parallel simulations enable consistent calculations using several machines, with limited communication times. However, there still remains the problem of decreasing the global real times of the simulations. Once all the CP matrices are computed, the multigroup flux solver is generally a two-step iterative process using the inverse power method. This paper also describes how acceleration and rebalancing strategy can be implemented in the

previous parallel schemes in order to increase their global performance for obtaining a converged flux map. The motivation is to increase the parallel algorithm performance sufficiently to enable analysis of very large problems (large spatial domain and many energy groups).

The paper is organized as follows. In Section II, we describe the modular environment used in the lattice code DRAGON,[4, 5] attention will be given to the new improvements in the 3D collision probability module EXCELL for doing a calculation on a single machine. General tools were defined to allow users to dispatch work on any subpart of their calculation flow-chart to other processors; these tools are also presented in Section II which serve as a general introduction to parallel techniques. In Section III, we give a description of two different approaches used to solve multigroup transport equation, as well as neutron rebalancing and variational acceleration techniques. In Section IV, the main steps for the parallelization of the equations are discussed, communication routines from the parallel environment PVM are also presented. Discussion of performance results are given in Section V.

II. THE DRAGON SOFTWARE (EXCELL MODULE) AND ITS ENVIRONMENT

The current calculation sequence in DRAGON usually contains one or more calls to a collection of modules such as:[5]

- The `GEO` module, to define the geometry of the domain.
- The `LIB` module, to access microscopic data on cross-section libraries (currently, three multigroup libraries are available in Canada : an older 69-group Winfrith library and two 89-group libraries based on ENDF/B-V and ENDF/B-VI data).
- The `SHI` module, to perform self-shielding calculations.
- The `EXCELT` module, for the ray tracing in 2D or 3D geometries.
- The `ASM` module, to calculate the collision probability matrices.
- The `FLU` module, a general tool that takes CP matrices packaged in a well-defined format and performs inner and outer iterations necessary for obtaining the critical value (k-effective or critical buckling search) and the converged multigroup flux map.
- The `EDI` module, to condense and homogenize nuclear properties into few-group macroscopic cross sections and to produce diffusion coefficients for subsequent core calculations.

A significant advantage of DRAGON for analyzing 3D effects is that it can solve the multigroup transport equation for the most realistic 2D cell model, as well as for 3D supercell models.

II.A. Features of the new EXCELL module of DRAGON

The previous approach restricted the use of 3D calculations to domains with coarse regions, because finer meshing would require too much disk space to store Gigs of tracking data. The new `EXCELL` module[6] integrates the work done in both `EXCELT` and `ASM` modules and processes the multigroup CP calculations into steps:

- First, trajectories are computed at the same time as angle-dependent factors are kept for normalizing the segment lengths. recalculated and normalized to ensure volume conservation.

- Collision, escape and transmission probabilities are evaluated and CP matrices processed for a certain number of groups. This step is redone until all energy groups are processed.

Using these features, the user has no further restriction on the number of tracks necessary to solve a very fine 3D transport problem. In case of lack of memory, he can also process each group one right after the other and collect the CP matrices on a direct access file. The kinds of operations needed on the CP matrices are now summarized:

- normalization of matrix coefficients to restore conservation relations: various normalization techniques are available in the DRAGON code.
- integration of boundary conditions: this is done by factoring out the symmetric transmission part of the probability matrices, including the albedo coupling between inner and outer external currents.
- elimination of the within-group scattering effects: this option is used to speed up the convergence of the multigroup flux solver.

Another interesting feature of the EXCELL module is that the innermost loops are generally on the number of groups for processing all CP information. The symmetric CP arrays are arranged in such a way that the leftmost subscript is the group index. In this way, we achieve stride 1 processing of all these innermost loops where the group index is always increased by 1. Then, each time an operation is needed on these matrices, we can benefit from the various automatic vectorization opportunities in most high-level computers. Most optimizing preprocessors can thus be used to scan the code and substitute carefully tuned vector library routines to improve the performance of these operations. Using this new EXCELL module, instead of the previous combination of EXCELT/ASM modules, results in computations which are more than twice faster on **sequential architectures**.[6]

II.B. Tools for distributing calculations on several processors

The DRAGON input uses the CLE-2000 language as a toolbox to allow user-friendly calculations in the main input stream or in procedures using reversed polish notation. The CLE-2000 language also allows loops, conditional testing and has macro-processor capabilities. Moreover, the DRAGON collection of modules is supported by a generalized driver that is responsible for managing the objects produced by each module.[7] The simplest module calls have the form:

```
{input objects} := MOD: {output objects} :: {data input for module MOD} ;
```

Objects appearing on both sides are assumed to exist and are modified by the module MOD. The objects can reside in memory in the form of hierarchical linked lists or on a direct access file.

Some tools have been added to the generalized driver to enable users to dispatch module or procedure calls on several processors and recover the objects produced anywhere. For example, suppose that we have access to two processors with distributed memory, the user can initialize a parallel task by calling the INITP module as follows:

```
(* example of use for INITP module *)
LINKED_LIST P0 P1 SETUP SF ;
MODULE      INITP: SENDP: RECVP: SOLVP: END: ;
INTEGER     me ;
CHARACTER   task := 'DRAGON-SCRIPT' ;
me SETUP :=   INITP: task :: NPROCS 2 ;
```

On the first processor, the user will have `me = 0`, on the second one `me = 1`. The object `SETUP` contains informations about the processors id's; it will be used to transfer data between processors.

Suppose now that the user wants to create an object `P0` on processor 0, an object `P1` on processor 1 and send it back to processor 0 for further calculations. The input would be:

```
IF me 0 = THEN
   P0 := ... ;                          (* create complete object 0 *)
   P1 := RECVP: SETUP :: FROM 1 ;       (* receive object 1 from processor 1 *)
   ...                                  (* continue calculation with both objects *)
ELSE
   P1 := ... ;                          (* create complete object 1 *)
   SENDP: P1 SETUP :: TO 0 ;            (* send object 1 to processor 0 *)
ENDIF ;
```

In the above example, processor 0 is the master and processor 1 is a slave used to create an object. The hierarchical content of `P1` is sent using the `SENDP` module and recovered on the other processor by the `RECVP` module. Both modules were encoded to transfer arbitrary objects; the user has the responsibility to balance the work asked on each processor, so that the message passing routines are not idle.

Another useful parallel application is the completion of an object:

```
IF me 0 = THEN
   P0 := ... ;                          (* create part of incomplete object 0 *)
   P0 := RECVP: P0 SETUP :: FROM 1 ;    (* receive the other part from processor 1 *)
   ...                                  (* continue calculation with complete object *)
ELSE
   P0 := ... ;                          (* create another part of incomplete object 0 *)
   SENDP: P0 SETUP :: TO 0 ;            (* send this part to processor 0 *)
ENDIF ;
```

In the above example, processor 1 is again a slave, but it is used to perform an incomplete calculation. The `P0` object in processor 0 is then modified to allow for the missing part.

Finally, using a true parallel solver, the user could create incomplete objects on both processors and used the parallel solver to perform calculations on `P0` without sharing the object subparts. In this configuration, we have:

```
IF me 0 = THEN
   P0 := ... ;                          (* create part of incomplete object 0 *)
ELSE
   P0 := ... ;                          (* create another part of incomplete object 0 *)
ENDIF ;
SF := SOLVP: P0 SETUP ... ;             (* use parallel module SOLVP *)
```

The modules `INITP`, `SENDP` and `RECVP` are useful when using a network of equivalent workstations or a dedicated parallel machine with distributed memory. The user can dispatch work to be performed by various sequential modules on several processors, while recovering the output objects. One application could be to couple neutronic calculations with thermal-hydraulics feedback effects.

III. PARALLELIZATION FOR MULTIGROUP TRANSPORT PROBLEMS

In usual applications of the DRAGON cell code, the user wants to produce a consistent solution to a multigroup problem of the form (G energy groups, I regions):

$$V_j \Phi_j^g = \sum_{i=1}^{I} P_{ji}^g \{ \sum_{g'=1}^{G} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + \frac{\chi_i^g}{k_{\text{eff}}} \sum_{g'=1}^{G} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \} \ , \tag{1}$$

with symmetric CP matrices $\mathbf{P}^g$ of order I. Note that P_{ji}^g is an element of the group g matrix, representing the first-flight reduced probability from region i to region j which has been multiplied by the volume of region j. The usual macroscopic cross sections (Σ_s and Σ_f) are formed by combining the microscopic cross sections of various isotopes. In Eq. (1), we have to find the critical eigenvalue and the corresponding multigroup flux eigenvector. Another possibility would be to find a critical buckling vector, but the iterative process would be somehow similar.

In the first two subsections, the two different approaches to the multigroup transport equation are investigated. These methods involve a two-step iteration processes (an inner iteration to treat the flux and an outer iteration for sources and multiplication factor calculations). Two methods that are used to improve the convergence speed of the inner iteration are the object of the two last subsections.

III.A. Self-scattering reduction (multigroup parallel scheme)

The $\mathbf{P}^g$ matrices are computed using only the total (or transport-corrected) macroscopic cross section of group g, plus the tracking data generated by passing lines in the spatial domain. If groups have been distributed on the set of processors, each processor only needs the tracking data and its cross sections to compute its $\mathbf{P}^g$ matrices. Provided that the tracking data is computed on every processor, no communication between processors is needed.

Now, suppose that the group self-scattering cross section $\Sigma_s^{g \leftarrow g}$ is available to the processor responsible for generating the matrix $\mathbf{P}^g$. To accelerate convergence of the iterative process for solving this system, we transfer this within-group scattering part to the left-side part of each one-group system. It is possible to simplify the linear system of Eq. (1) and to produce group-dependent equations of the form:

$$\Phi_j^g = \sum_{i=1}^{I} W_{ji}^g q_i^g \ , \tag{2}$$

where the new external sources are now coming only from *other* groups and are given by:

$$q_i^g = \sum_{g' \neq g} \Sigma_{si}^{g \leftarrow g'} \Phi_i^{g'} + S_i^g \ , \tag{3}$$

and fission neutrons are still provided by:

$$S_i^g = \frac{\chi_i^g}{k_{\text{eff}}} \sum_{g'} \nu \Sigma_{f_i}^{g'} \Phi_i^{g'} \ . \tag{4}$$

In the new EXCELL module, the transformation of symmetric matrix $\mathbf{P}^g$ into the new matrix $\mathbf{W}^g$ is performed simultaneously for all groups of the processor using a vectorised LDL^T decomposition. Note that, in every energy group, exactly the same sequence of operations is necessary to compute the $\mathbf{P}^g$ and transform it into $\mathbf{W}^g$ matrices; this is a very nice behavior when going to a very large number of groups. Moreover, there are logical switches for domains with void regions, furthermore tests are the same for groups. This scheme will give a nice balance of work between processors if the number of groups is similar on each processor.

III.B. Spatial recomposition (multiregion parallel scheme)

The previous self-scattering reduction is not the only way of solving the linear transport system of Eq. (1). An original scheme of partitioning the spatial domain instead of the energy domain has been developed by Henkel and Turinsky in order to solve the few-group diffusion equation.[8] In this scheme, if we have access to N processors, we partition the spatial domain into N non-overlapping set of regions, and reorder the flux unknowns and sources:

$$\begin{aligned} \vec{\phi}_B = &\quad \{\Phi_i^g;\, g=1,2,...,G;\, i\in B\}, \quad \vec{\phi} = \oplus_B \vec{\phi}_B, \\ \vec{s}_B = &\quad \{s_i^g;\, g=1,2,...,G;\, i\in B\}, \quad \vec{s} = \oplus_B \vec{s}_B. \end{aligned} \tag{5}$$

where $\oplus_B$ represents a direct sum used to concatenate all the unknowns together. This scheme was used in the context of collision probability methods for 2-D cell calculations.

The original system of Eq.(1) could be rewritten in the form of multigroup sub-matrices of $\mathbf{P}^g$. If we want to perform self-scattering reduction of matrices $\mathbf{P}^g$, the amount of data transfer between processors is too large. If each processor perform the collision probability integration for all groups, the resulting coefficients in $\mathbf{W^g}$ matrices are distributed among the processors so that each of them can calculate only the probabilities of staying in regions of block B and going to another block B'.

The updated unknowns from block $B' \neq B$ will be unavailable to the processor responsible for block B, so the usual inverse power iterator is slightly changed. For each block B, we now have a fixed source problem. The multiregion scheme is thus applied to subparts of the geometric domains, and inner iterations act as if the transport problem with fixed sources has to be solved only for each subpart. With this multiregion scheme, at each outer iteration, fission sources are reevaluated as well as contributions of sources coming from other blocks. The inner-outer iterations then proceed until the convergence of $\mathrm{k}^{(n)} \rightarrow k_{\mathrm{eff}}$ and of the flux distribution.

III.C. Rebalancing scheme

Rebalancing is a popular scheme applied to production codes in order to accelerate inner iterations. This scheme is introduced to deal with the problem of neutron transfers between various groups due to scattering.[9] The objective of the flux rebalancing method is to enhance the rate of convergence by imposing neutron conservation after every iteration, and thus to force the equality:

$$\sum_i \{\Sigma_{t,i}^g V_i \Phi_i^g - \sum_{g'} \Sigma_{si}^{g\leftarrow g'} V_i \Phi_i^{g'}\} = \sum_i S_i^g V_i \,. \tag{6}$$

where $\Sigma_{t,i}^g$ is the total (or transport-corrected) cross section in region i and group g.

We compute for each unconverged energy group an average rebalancing factor α^g such that the rebalanced flux $\bar{\Phi}_i^g$ defined as:

$$\bar{\Phi}_i^g = \alpha^g \Phi_i^g \,, \tag{7}$$

exactly satisfies Eq. (6). We obtain the following system:

$$\begin{aligned} \sum_i \{[\Sigma_{t,i}^g - \Sigma_{si}^{g\leftarrow g}] V_i \Phi_i^g \alpha^g \;&- \sum_{g'(unconverged)\neq g} \Sigma_{si}^{g\leftarrow g'} V_i \Phi_i^{g'} \alpha^{g'}\} \\ &= \sum_i \{S_i^g V_i + \sum_{g'(converged)\neq g} \Sigma_{si}^{g\leftarrow g'} V_i \Phi_i^{g'}\} \,. \end{aligned} \tag{8}$$

This later system is resolved after each inner iteration. The computed α^g are used to update rebalanced fluxes $\bar{\Phi}^g$ for the next inner iteration.

III.D. Variational relaxation scheme

The next acceleration method we will introduce is a variational relaxation method. Assuming fixed source S obtained from an outer iteration, in general we can rewrite the equation for the flux, in an inner iteration, in the following matrix form:

$$\Phi^{(n+1)} = \mathbf{T}\Phi^{(n)} + S \ . \tag{9}$$

We will now introduce a relaxation parameter $\mu^{(n)}$ which can be computed efficiently after each iteration using a variational method.[9] To illustrate this acceleration process, suppose that we have performed three inner iterations without acceleration:

$$\begin{aligned} \Psi_1 &\leftarrow \Phi^{(n)} \ , \\ \Psi_2 &= \mathbf{T}\Psi_1 + S \ , \\ \Psi_3 &= \mathbf{T}\Psi_2 + S \ , \end{aligned}$$

then, we can define errors:

$$\begin{aligned} e_1 &= \Psi_2 - \Psi_1 = \mathbf{T}\Psi_1 + S - \Psi_1 \ , \\ e_2 &= \Psi_3 - \Psi_2 = \mathbf{T}\Psi_2 + S - \Psi_2 \ . \end{aligned}$$

It can be easily shown that the value:

$$\mu^{(n)} = -\frac{< e_1, e_2 - e_1 >}{< e_2 - e_1, e_2 - e_1 >} \ . \tag{10}$$

will minimize the mean square residual. Inner convergence implies that, for n large enough, $|e_2| < |e_1|$ with $sign(e_1) = sign(e_2)$ leading to $\mu^{(n)} > 0$ (often $\mu^{(n)} >> 1$). At each inner iteration, we can thus decide to apply the acceleration factor or not. Note also that the CPU overhead associated with acceleration is minimal because, using the accelerated flux computed as:

$$\tilde{\Phi}^{(n)} = \mu^{(n)}\Psi_2 + (1 - \mu^{(n)})\Psi_1 \ , \tag{11}$$

one can also obtained the next unaccelerated estimate as:

$$\Phi^{(n+1)} = \mu^{(n)}\Psi_3 + (1 - \mu^{(n)})\Psi_2 \ , \tag{12}$$

without doing a new inner iteration. The whole acceleration process is very efficient because it always keeps the three last iterates of the flux on a stack, and decides if the last two are accelerated or not.

For the variational acceleration, we usually alternate between sets of non-accelerated and accelerated iterations. The reason for this is to compensate for instabilities arising due to non-fundamental modes excited by this variational acceleration technique. In fact, it was found after numerical testing that one should alternate between 3 non-accelerated and 3 accelerated iterations ((3,3) acceleration scheme).

IV. MULTIGROUP PARALLELIZATION OF THE EXCELL MODULE

A parallel version of the module EXCELL was written in such a way that it can run for an arbitrary number of processors, and the process is initiated using the INITP module. The user has only to assign the set of groups he wants to put on each processor after testing its me value. As explained earlier, the tracking data is first generated on each processor. All the attributes of the EXCELL module on a single processor are kept, except that only the given groups are treated. A mask is included in each incomplete

collision probability object to describe the groups that were computed on this processor. The group-dependent collision probability matrices $\mathbf{W}^g$ are stored under different records with names concatenated with their group index. At this point, it would be possible to send all the incomplete objects in one processor and solve for the flux there.

However, a parallel flux solver module FLG was also designed, while keeping all attributes useful for single machine operation (such as acceleration and rebalancing described earlier). The solver was made as portable as possible: only a few changes would be necessary in order to change from a given parallel environment to another. The module FLG first checks for consistency between incomplete objects from all processors. When the superposition of all masks gives the whole energy domain expected, the flux solution can proceed. Each processor knows all of the multigroup cross section data, so that the calculation of fission and scattering sources is possible. Once this consistency check is done, the message passing part is restricted to one routine.

We chose the public domain PVM[10] available to all kinds of workstations (version 3) and PVMe on our IBM-SP2. Suppose that we have the flux vector FLUX(NREG,NGRP), the logical mask MASKGR(NGRP) for groups in a processor and the processor identifications TIDS(NPROC). After each inner iteration, multigroup fluxes are exchanged in a routine by a simple loop such as:

```
DO IG= 1, NGRP
   IF( MASKGR(IG) )THEN
      CALL PVMFINITSEND( 0, INFO )
      CALL PVMFPACK    ( 3, IG, 1, 1, INFO )
      CALL PVMFPACK    ( 4, FLUX(1,IG), NREG, 1, INFO )
      CALL PVMFMCAST   ( NPROC, TIDS, 4, INFO )
   ELSE
      CALL PVMFRECV    ( -1, 4, INFO )
      CALL PVMFUNPACK  ( 3, JG, 1, 1, INFO )
      CALL PVMFUNPACK  ( 4, FLUX(1,JG), NREG, 1, INFO )
   ENDIF
ENDDO
```

Moreover, the multiregion flux exchange would also be simple in PVM; there would be a region mask vector and fluxes would be exchanged after stride 2 packing and unpacking.

V. PARALLEL PERFORMANCE RESULTS AND DISCUSSION

The parallel multigroup and multiregion schemes described above were tested on several types of computers including:

- the 4-processor IBM SP-2 at École Polytechnique;
- the 4-processor Sparc 2000 at École Polytechnique;
- the 8-processor Sparc 1000 at Centre de Recherches Mathématiques.

A parallel subset of the DRAGON [5] code was written in such a way that it can run for an arbitrary number of processors (which becomes an input) with either distributed or shared memory. In this extension of DPV-01,[3] the flux solver was enhanced to include the above rebalancing and acceleration techniques. The two parallel iterative schemes were used to integrate probabilities and compute fluxes for a unit cell geometry and the physical data corresponding with one of the Mosteller benchmarks.[11] The

unit cell geometry is a 2-D square with three concentric annuli inside. There are three different material regions corresponding to the fuel, its sheath surrounded by light-water coolant. This cell problem was further divided to obtain 32 zones. In order to compare parallel schemes and hardware, we choose to present a 69-group problem. Table 1 contains the total number of inner iterations to reach a relative precision of 10^{-5} on k_{eff} and flux distribution and the speedups observed when using 4 processors (with rebalancing+variational acceleration). We can observe in this table that using 4 processors decreases the CPU times by a factor of around 3. Note that the 4-processor CPU times for the multigroup scheme are around 7 seconds for IBM SP2, 24 seconds for SPARC 2000 and 152 seconds for SPARC 1000. The CPU times for the multiregion schemes are 50% higher on the SP2. If the real time is the important factor, the CPU times observed for both methods are extremely competitive with the best available tools for solving this kind of unit cell problem. The best speedup is achieved by Sparc machines (shared-memory), where message passing is faster than IBM SP-2 fast communication links.

Table 1: Comparison between the Multigroup and Multiregion Parallel Scheme
(same unit cell problem using 69 groups, 32 regions, 4 processors)

	Number of flux exchanges	Speedup IBM SP2	Speedup SPARC 2000	Speedup SPARC 1000
multigroup	17	2.6	3.0	2.9
multiregion	36	2.5	2.8	2.7

Numerical tests were also done using the EXCELL module optimized for sequential architectures.[6] The benefits of stride 1 multigroup operations when calculating and transforming the collision probability matrices will still remain when distributing the energy groups on the processors. The 3D geometry is composed of two-half CANDU bundles with end regions located at the center of the assembly; isotropic reflection is assumed on external boundaries. One bundle contains fresh fuel while the other contains irradiated fuel; the problem is intended to simulate the on-power refueling of a CANDU channel. CANDU 37-element bundles are composed of 4 rings of fuel pins (with 1, 6, 12 and 18 fuel rods respectively). Each fuel pin is sheathed with Zircalloy and has an end cap region. The end caps and plates to support the cluster assembly form the end region containing sheathing material and heavy-water coolant. The thermal flux will peak inside the end region, so that its maximal value will be reached between bundles. Figure 1 gives an idea of the thermal flux behavior when looking at each fuel ring region in the axial direction.

Table 2: Multigroup 3D EXCELL Parallel Scheme on IBM SP2
(same supercell problem using 69 groups, 416 regions)

Modules used	Ref. real time (1 processor)	Time (2 processors)	Time (3 processors)	Time (4 processors)
EXCELL	49:42 m	41:57 m	36:55 m	31:36 m
EXCELL/FLG	49:45 m	34:58 m	28:05 m	23:40 m

The domain contains 416 regions and calculation of the flux peaking effect was done using a 69-group library. The 69-group calculations were distributed from 1 up to the 4 processors available on our IBM SP2. Fortunately, the memory is sufficient to run this problem on a single processor. Using the parallel version of the EXCELL module, we recall that the tracking (with $\approx 500K$ tracks renormalized) is done on each processor, so that there is a basic penalty when using several processors. Moreover, flux convergence of this problem is slow because the peaking is restricted to a tiny region inside the assembly. The FLG parallel module will have to exchange multigroup values more than a hundred times, increasing the communications between processors. Table 2 gives the CPU times when using either only the EXCELL module (all group CP matrices transfer to one processor) or the multigroup parallel solver combining EXCELL/FLG modules. In this last case, the speedup for 4 processors is about 2.1, less than in 2D unit cell problems. However, we still conclude that this scheme will be useful to assess fine-mesh transport calculations, without the usual memory and CPU limitations imposed by problem size.

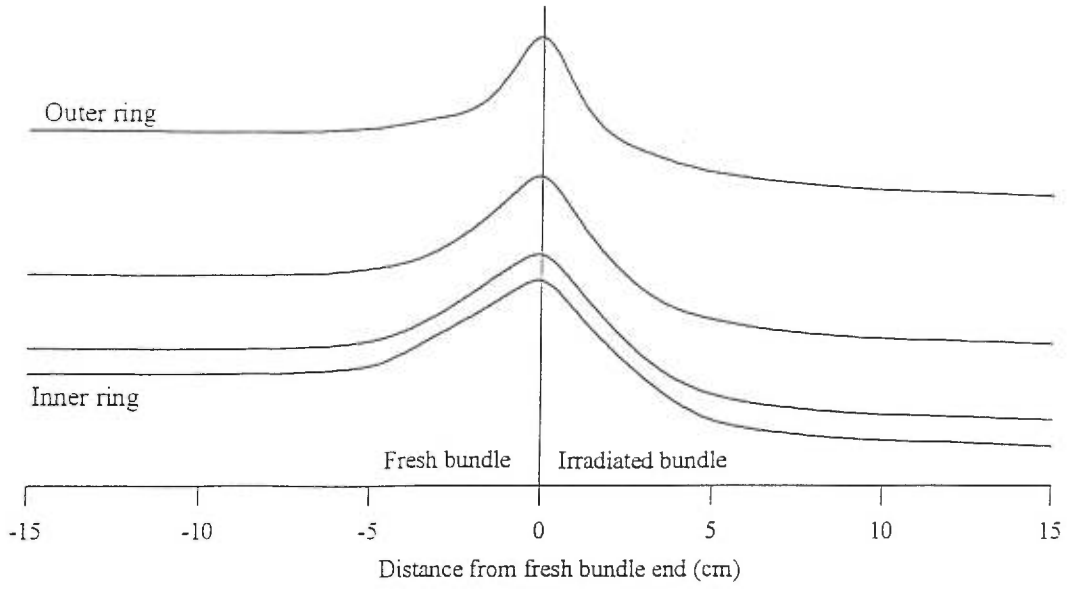

Figure 1. Thermal flux behavior at the junction of two CANDU fuel bundles.

Acknowledgments – This work has been carried out partly with the help of a grant from the Natural Science and Engineering Research Council of Canada. The authors would like to thank Vincent Cavuoti, from École Polytechnique, for his support in doing IBM SP2 and Sparc 2000 calculations.

REFERENCES

[1] A. Qaddouri, R. Roy, G. Marleau, B. Goulard and M. Mayrand, "Multigroup Flux Solvers on Parallel Architecture,", *Fifteenth CNS Annual Conference*, Montreal Canada (1994).

[2] A. Qaddouri, R. Roy, and B. Goulard, "Multigroup Flux Solvers Using PVM ", *Int. Conf. on Mathematics and Comp. Reactor Phys. and Env. Analyses*, Portland, Oregon USA (1995).

[3] A. Qaddouri, R. Roy, M. Mayrand, and B. Goulard, "Collision Probability Calculation and Multigroup Flux Solvers Using PVM ",*Nucl. Sci. Eng.*, **123**, 392-402 (1996)

[4] R. Roy, G. Marleau, J. Tajmouati, and D. Rozon, "Modelling of CANDU Reactivity Control Devices with the Lattice Code DRAGON," *Ann. Nucl. Energy* **21**, 115-132 (1994).

[5] G. Marleau, A. Hébert and R. Roy, "A User's Guide for DRAGON", Report IGE-174 Rev. 1, École Polytechnique de Montreal, Canada (1996).

[6] R. Roy, "Collision Porbability Solvers: Applications to 3D Supercell Calculations and Experience with Some Parallel Algorithms (DRAGON-EXCELL)", *Seminar on 3D Deterministic Radiation Transport Computer Programs*, Nuclear Energy Agency, OECD, Paris (1997).

[7] A. Hébert and R. Roy, "A Programmer's Guide for the GAN Generalized Driver – Fortran 77 version", Report IGE-158, École Polytechnique de Montreal, Canada (1994).

[8] C.S. Henkel and P.J. Turinsky, "Solution of the Few-Group Diffusion Equation on a Distributed Memory Multiprocessor," *Top. Mtg. on Advances in Reactor Physics*, Charleston USA (1992).

[9] A. Qaddouri, R. Roy and B. Goulard, "Parallel Acceleration and Rebalancing Schemes for Solving Transport Problems Using PVM,", *Fifth Int. Conf. on Simulation Methods in Nuclear Engineering*, Montreal (1996).

[10] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, "PVM 3 User's Guide and Reference Manual, " Report ORNL/TM-12187, Oak Ridge National Laboratory (1994).

[11] R.D. Mosteller and *al.*, "Benchmark Calculations for the Doppler Coefficient of Reactivity", *Nucl. Sci. Eng.* **107**, 265 (1991).

ARTICLE 6

Parallélisation d'un Code d'Éléments Finis
pour le Calcul d'Écoulements Tridimensionnels

Centre de Recheches Mathématiques de Montréal 1997

publié en novembre 1997

PARALLÉLISATION D'UN CODE D'ÉLÉMENTS FINIS POUR LE CALCUL D'ÉCOULEMENTS TRIDIMENSIONNELS

A. Qaddouri [1*], A. Soulaimani [2] and Y. Saad [3]

[1] *Centre de Recherches Mathématiques, Université de Montréal*
Montréal, Québec, Canada H3C 3J7

[2] *Ecole de Technologie Supérieure, Département de génie mécanique*
1100 Notre-Dame Ouest,Montréal, Québec, Canada H3C 1K3

[3] *Computer Science Departement, University of Minnesota*
4-192 EE/CSci Building, 200 Union Street SE., Minneapolis, MN 55455

Abstract — *The aim in this paper is to parallelize a finite element code to simulate incompressible 3D flows coupled to a free surface. In this code, the solution is accelerated at each time step by a preconditionned nonlinear GMRES and the preconditionner is based on ILUT factorization of the jacobian matrix. In this work we suggest an element-based partitioning of the spatial domain into non overlapping subdomains by using METIS package. We use PSPARSLIB to form a local data structure to represent matrices and vectors. The parallelization of the preconditionner is achieved by considering only the contribution of local elements to jacobian matrix in each processor, and applied ILUT to this local matrix. The parallel code was tested on SUN-Enterprise and on SGI-ONYX2, and some numerical results are presented.*

Résumé — *Le but de ce travail est la parallélisation d'un code d'éléments finis pour la simulation d'écoulements tridimensionnels avec surface libre pour un fluide incompressible. Dans ce code, à chaque pas de temps la solution est accélérée en utilisant un algorithme non linéaire de type GMRES préconditionné. Le préconditionnement est basé sur une factorisation ILUT de la matrice jacobienne. Nous proposons une décomposition du domaine spatial en sous-domaines non chevauchés en utilisant l'interface METIS. La librairie PSPARSLIB est utilisée pour former une structure locale des données afin de représenter les matrices et les vecteurs. Le parallélisme du préconditionneur consiste à former sur chaque processeur la matrice jacobienne correspondant aux éléments appartenant au sous-domaine traité par le processeur. Le code parallèle a été testé sur un SUN-Enterprise et un SGI-ONYX2, et des résultats sur la convergence sont présentés.*

I. INTRODUCTION

Un problème d'hydrodynamique en présence d'une surface libre (ou frontière libre) est un problème aux limites, qui consiste à résoudre des équations aux dérivées partielles (équations de conservation) sur un domaine dont certaines parties (les frontières libres) sont inconnues et doivent être déterminées.

Le problème résolu dans le programme parallélisé est celui des équations tridimensionnelles de Navier-Stokes des fluides incompressibles, en présence d'une surface libre. Le problème consiste à résoudre des équations de conservation (masse, quantité de mouvement et énergie) couplées à une équation de transport définissant l'évolution de la surface libre. Les équations sont obtenues dans une description cinématique eulérienne-lagrangienne mixte.[1] La méthode des éléments finis dite stabilisée est utilisée pour approcher la solution.

La résolution du système d'équations discrètes est accélérée à chaque pas de temps par un algorithme itératif de type GMRES[2] préconditionné. L'efficacité des algorithmes itératifs est reliée à l'utilisation d'un bon préconditionneur. Dans le présent travail, le préconditionnement repose sur une factorisation incomplète ILUT [3] de la matrice jacobienne elle-même. Cette dernière est recalculée, lorsque c'est nécessaire, après un certain nombre de pas de temps. Le préconditionneur basé sur une factorisation incomplète de la matrice jacobienne, est facile à implémenter et donne une bonne accélération au solveur itératif.[4] Dans le code séquentiel le calcul explicite de la matrice jacobienne n'est utilisé que pour le préconditionnement.[1]

La librairie PSPARSLIB[5] est utilisée dans ce travail, pour simplifier l'implémentation du solveur iteratif parallèle, ainsi que la formation de la structure locale des données. Le parallélisme utilisé est un parallélisme de données où les éléments sont départagés entre les processeurs d'une façon optimale en utilisant l'interface METIS.[6] Par la suite, chaque processeur forme sa part locale du système global à résoudre. Chaque processeur forme la matrice jacobienne correspondant aux éléments du sous-domaine traité par le processeur, et une factorisation de type ILUT est appliquée à cette matrice. L'algorithme GMRES parallèle est utilisé pour résoudre le système global.

Dans la section II, nous présentons les équations de Navier-Stokes d'un fluide incompressible ainsi que l'équation de la surface libre. La discrétisation spatiale et temporelle utilisée dans ce travail est brièvement présentée. La section III présente une description détaillée de la méthode de résolution. Nous passons en revue l'algorithme GMRES ainsi que le préconditionnement du type ILUT. La section IV constitue une brêve description du code séquentiel. Dans la section V, nous décrivons la librairie PSPARSLIB et nous discutons du concept de partition de graphe et de l'utilisation de METIS pour produire des sous-domaines spatiaux non chevauchés. La structuration des données dans PSPARSLIB et une façon de l'utiliser pour le calcul des résidus est décrite puis discutée en detail. Finalement, la factorisation ILUT locale utilisée dans ce travail et les types de communications dans un algorithme de type GMRES parallèle sont présentés. La section VI est consacrée aux résultats numériques, et la conclusion suivra en section VII.

II. LOIS DE CONSERVATION ET ÉQUATION DE LA SURFACE LIBRE

Nous résumons dans ce qui suit les relations fondamentales obtenues en appliquant la description cinématique eulérienne-lagrangienne mixte.

Soit Ω un référentiel connu dans lequel sont écrites les équations de conservation. Le domaine $\lambda_t(\Omega)$ est sa configuration à l'instant t. En utilisant la méthode des élément finis, le domaine Ω est en fait concrétisé par un maillage initial (la dernière configuration existante) et $\lambda_t(\Omega)$ est sa configuration à

l'instant t qui reste à déterminer.

Les coordonnées d'un point de Ω sont notées par X dans un repère cartésien fixe. Nous notons par $\underline{\mathbf{c}}(\mathbf{X},t)$ et $\underline{\mathbf{u}}(\mathbf{X},t)$ respectivement la vitesse du référentiel et du corps matériel au point X. La vitesse relative est la différence: $\underline{\mathbf{u}}(\mathbf{X},t) - \underline{\mathbf{c}}(\mathbf{X},t)$.

- L'équation de conservation de la masse s'écrit:

$$\nabla \cdot \underline{\mathbf{u}} = 0 \tag{1}$$

- L'équation de conservation de la quantité de mouvement s'écrit:

$$\rho[\frac{\partial \underline{\mathbf{u}}}{\partial t} + (\underline{\mathbf{u}} - \underline{\mathbf{c}}) \cdot \nabla \underline{\mathbf{u}}] + \nabla p - \nabla \cdot \underline{\underline{\tau}} = \rho \underline{\mathbf{g}} \tag{2}$$

où ρ est la masse volumique, p la pression, $\underline{\underline{\tau}} = \mu(\nabla \underline{\mathbf{u}} + (\nabla \underline{\mathbf{u}})^t)$ les contraintes visqueuses. μ est la viscosité laminaire et turbulente et $\underline{\mathbf{g}}$ est la gravité. Les dérivées spatialles dans les équations (1-2) sont faites par rapport aux coordonnées $X \in \lambda_t(\Omega)$.

La frontière libre est une surface quasi-horizontale paramétrisée par une fonction de hauteur. L'évolution de celle-ci est régie par une équation de transport traduisant le fait que c'est une surface matérielle: en suivant le mouvement de la frontière libre, le débit est nul.

Notons par $h(X,t)$ la cote verticale de la position de la surface libre (S.L). La fonction h(X,t) vérifie l'équation de transport (en trois dimensions) suivante:

$$\frac{\partial h}{\partial t} + u_1 \frac{\partial h}{\partial x_1} + u_2 \frac{\partial h}{\partial x_2} - u_3 = 0 \tag{3}$$

où les u_i sont les composantes de la vitesse.

II.A. Formulation variationnelle et approximation par éléments finis

Nous cherchons des solutions faibles aux équations de conservation écrites selon une description eulérienne-lagrangienne mixte. La formulation variationnelle s'écrit comme suit

$$\begin{aligned}
&\int_{\lambda_t(\Omega)} \underline{\mathbf{w}} \cdot [\frac{\partial \underline{\mathbf{u}}}{\partial t} + (\underline{\mathbf{u}} - \underline{\mathbf{c}}) \cdot \nabla \underline{\mathbf{u}} + \frac{1}{\rho} \nabla p - \underline{\mathbf{g}}] \mathrm{d}\Omega + \int_{\lambda_t(\Omega)} \nabla \underline{\mathbf{w}} \cdot \underline{\underline{\tau}} \mathrm{d}\Omega \\
&- \int_{\Gamma} \underline{\mathbf{w}} \cdot (\underline{\underline{\tau}} \cdot \underline{\mathbf{n}}) \mathrm{d}\Gamma + \int_{\lambda_t(\Omega)} [(\underline{\mathbf{u}} - \underline{\mathbf{c}}) \cdot \nabla \underline{\mathbf{w}} + \nabla q] \beta [(\underline{\mathbf{u}} - \underline{\mathbf{c}}) \cdot \nabla \underline{\mathbf{u}} - \frac{1}{\rho} \nabla p - \underline{\mathbf{g}}] \\
&+ \int_{\lambda_t(\Omega)} q \nabla \cdot \underline{\mathbf{u}} \mathrm{d}\Omega + \int_{\Gamma_{S.L}} \delta h [\frac{\partial h}{\partial t} + u_1 \frac{\partial h}{\partial x_1} + u_2 \frac{\partial h}{\partial x_2} - u_3] \mathrm{d}\Omega + \int_{\lambda_t(\Omega)} \nabla \delta c_3 \cdot \nabla c_3 \mathrm{d}\Omega = 0
\end{aligned} \tag{4}$$

où $\underline{\mathbf{w}}$, q, δh et δc_3 sont les fonctions de pondération. β est un coefficient de stabilisation.[1] $\Gamma_{S.L}$ qui représente la surface libre, est la partie mobile de la frontière Γ.

Le domaine $\lambda_t(\Omega)$ est décomposé en éléments tétraédriques. L'élément tétraédrique est le mieux adapté, aussi bien sur le plan numérique que physique, à la résolution des problèmes hydrodynamiques en présence d'une surface libre. C'est un élément qui offre beaucoup de flexibilité pour modéliser des

contours de forme géométrique complexe. Sur chaque élément tétraédrique, les composantes de la vitesse, la pression et la hauteur sont approchées par des polynômes linéaires continus.

Pour le maillage de la peau Γ (entrée de l'écoulement, solide, surface libre et sortie de l'écoulement), on utilise des éléments triangulaires (faces du tétraédre). Ainsi les composantes de la vitesse, la pression et la hauteur restent approchées par des polynômes linéaires continus.

Pour ce qui est des conditions limites, on a:

- À l'entrée de l'écoulement, le profil de la vitesse $\underline{\mathbf{u}}$ est supposé connu et on impose la hauteur: $h = h_o$.
- À la sortie de l'écoulement, on impose la vitesse $\underline{\mathbf{u}}$, lorsqu'elle est connue.
- À la paroi solide, on a: $\underline{\mathbf{u}} \cdot \underline{\mathbf{n}} = 0$ où $\mathbf{n}$ est le vecteur unitaire normal à la paroi. La contrainte de cisaillement est égale à $c_f u^2$ où c_f est un coefficient de frottement.
- À la surface libre $\underline{\underline{\tau}} \cdot \underline{\mathbf{n}} = p \cdot \underline{\mathbf{n}}$.

On utilise un shéma implicite pour discrétiser la dérivée temporelle de toute quantité $\nu(*,t)$. L'usager peut choisir entre les deux schémas suivants:

1. Un schéma d'Euler implicite du premier ordre:

$$\frac{\partial \nu(*,t)}{\partial t} \approx \frac{\nu(*,t+\triangle t) - \nu(*,t)}{\triangle t} \tag{5}$$

2. Un schéma implicite du second ordre:

$$\frac{\partial \nu(*,t)}{\partial t} \approx \frac{3\nu(*,t+\triangle t) - 4\nu(*,t) + \nu(*,t-\triangle t)}{2\triangle t} \tag{6}$$

$\triangle t$ étant le pas de temps.

La discrétisation spatiale et temporelle réduit la formulation de l'Eq.(4) à un système non linéaire de dimension finie N qui peut être représenté par:[1]

$$\{\mathbf{R}\}(\{\mathbf{U}\}) = 0 \tag{7}$$

où $\{\mathbf{R}\}$ est un opérateur non linéaire de $\mathbb{R}^N$ dans $\mathbb{R}^N$. $\{\mathbf{U}\}$ est le vecteur de l'ensemble des degrés de liberté (d.d.l.)

III. Méthode de résolution

La méthode de Newton-Raphson pour trouver une solution du problème non linéaire de l'Eq.(7), consiste à résoudre à chaque pas de temps le système des équations algébriques suivant:

$$\mathcal{J}(\{U\}^{n,i-1})\{\delta U\}^{n,i} = -\{R\}(\{U\}^{n,i-1}) \tag{8}$$

où $\{U\}^{n,i-1}$ est le vecteur global des degrés de liberté, au pas de temps n et à l'itération $i-1$ de Newton. $\{R\}$ est le vecteur résidu global, et $\mathcal{J} = \frac{\partial R}{\partial U}$ est la matrice jacobienne. $\mathcal{J}$ est une matrice creuse non symétrique.

L'algorithme de résolution est basé sur une méthode de Newton couplée avec une discrétisation temporelle implicite. Il peut se résumer comme suit:

Algorithme de Newton:

1. étant donné une solution initiale $\{\mathbf{U}\}^0$ Faire ,
2. Pour n= 1,2,... Faire:
3. $\{\mathbf{U}\}^{\mathbf{n,0}} = \{\mathbf{U}\}^{\mathbf{n-1}}$
4. Pour i=1,..., Faire
5. Calculer $\mathcal{J}(\{\mathbf{U}\}^{\mathbf{n,i-1}})$ et $\{R\}(\{\mathbf{U}\}^{\mathbf{n,i-1}})$,
6. Résoudre le système linéaire (8)
7. mettre à jour $\{\mathbf{U}\}^{\mathbf{n,i}}$: $\{\mathbf{U}\}^{\mathbf{n,i}} = \{\mathbf{U}\}^{\mathbf{n,i-1}} + \{\delta\mathbf{U}\}^{\mathbf{n,i}}$
8. mettre à jour les coordonnées
9. Si le critère de convergence est satisfait Aller à 11,
10. Fin ,
11. Fin

Le système donné par l'équation (8) est linéaire en $\{\delta\mathbf{U}\}^{\mathbf{n,i}}$. Ce système est résolu par une méthode itérative dans un sous-espace de Krylov de dimension m inférieure à la dimension de la matrice jacobienne $\mathcal{J}$ ($m < N$). L' algorithme précédent devient un algorithme approché de Newton. Pour résoudre le système de l'équation (8) on utilise l'algorithme de type GMRES préconditionné.

III.A. Algorithme GMRES

L'algorithme GMRES a été introduit par Saad-Schultz.[2] Étant donné une solution initiale $\{\delta\mathbf{U}\}_0$ du système (8), l'algorithme GMRES consiste à obtenir une solution approchée $\{\delta\mathbf{U}\}_m = \{\delta\mathbf{U}\}_0 + z$ dans l'espace affine $\{\delta\mathbf{U}\}_0 + K_m$ de dimension m. K_m est appelé sous-espace de Krylov associé à la matrice $\mathcal{J}$ et au vecteur $r_0 = -\{\mathbf{R}\} - \mathcal{J}\{\delta\mathbf{U}\}_0$. La solution approchée minimise la norme du vecteur $-\{\mathbf{R}\} - \mathcal{J}\{\delta\mathbf{U}\}$. Le sous-espace de Krylov est donné par:

$$K_m = span\{r_0, \mathcal{J}r_0, \mathcal{J}^2 r_0, ..., \mathcal{J}^{m-1} r_0\} \tag{9}$$

Dans GMRES on construit une base orthonormée [$v_1, v_2, ..., v_m$] du sous-espace de Krylov K_m en utilisant un processus de Gram-Schmidt connu sous le nom de méthode d'Arnoldi.

Algorithme d'Arnoldi:

1. Calculer $v_1 = r_0/||r_0||_2$.
2. Pour k= 1,2,..,m Faire:
3. Calculer $w_k = \mathcal{J}v_k$

4. Pour i=1,...,k Faire
5. $h_{ik} = (w_k, v_i)$
6. $w_k = w_k - h_{ik} v_i$
7. Fin
8. $h_{k+1,k} = ||w_k||_2$. If $h_{k+1,k} = 0$ STOP.
9. $v_{k+1} = w_k / h_{k+1,k}$.

Cet algorithme s'arrête si $h_{k+1,k} = 0$, c.a.d, quand le degré du polynome minimal pour r_0 est égal à k.

Si on dénote par V_m la matrice $n \times m$ dont les colonnes sont les vecteurs $v_1, ..., v_m$,on a la relation suivante:

$$\mathcal{J} V_m = V_{m+1} \bar{H}_m. \tag{10}$$

La solution approchée dans $\{\delta \mathbf{U}\}_0 + K_m$ peut être exprimée dans la base [$v_1, v_2, ..., v_m$] de la façon suivante:

$$\{\delta \mathbf{U}\}_m = \{\delta \mathbf{U}\}_0 + V_m y. \tag{11}$$

On définit:

$$f(y) = || - \{\mathbf{R}\} - \mathcal{J}(\{\delta \mathbf{U}\}_0 + V_m y)||_2 = ||r_0 - \mathcal{J} V_m y||_2. \tag{12}$$

$f(y)$ peut être exprimée en fonction de y et de $\beta = ||r_0||_2$ comme:

$$f(y) = ||\beta e_1 - \bar{H}_m y||_2 \tag{13}$$

On cherche alors $y_m \in \mathbb{R}^n$ qui minimise $f(y)$ sur $\mathbb{R}^n$. La solution approchée s'ecrit finalement $\{\delta \mathbf{U}\}_0 + V_m y_m$.

Algorithme GMRES:

- **I. Départ:** choisir une solution initiale et calculer r_0, β, v_1
- **II. boucle d'Arnoldi**
- Pour k= 1,2,..,m Faire:
- Calculer $w_k = \mathcal{J} v_k$
- Pour k= 1,2,..,m Faire:
- Calculer $w_k = \mathcal{J} v_k$
- Pour i=1,...,k-1 Faire
- $h_{ik} = (w_k, v_i)$

- $w_k = w_k - h_{ik}v_i$
- Fin
- $h_{k+1,k} = ||w_k||_2$. Si $h_{k+1,k} = 0$ Aller à **III** .
- $v_{k+1} = w_k/h_{k+1,k}$.
- **III. Former la solution**
- Calculer y_m et $\{\delta \mathbf{U}\}_0 + V_m y_m$ où V_m =$[v_1, v_2, ..., v_m]$

III.B. Préconditionnement ILUT

Le succès des méthodes itératives comme GMRES est dû en partie au préconditionnement du système à résoudre. Le préconditionnement consiste à remplacer le système original (8) par un système équivalent. Par exemple pour um préconditionnement à gauche on va résoudre:

$$M^{-1}\mathcal{J}\{\delta \mathbf{U}\} = -M^{-1}\{R\} \tag{14}$$

où M est une matrice creuse donnée. M est appelée matrice de préconditionnement. Le système équivalent est alors résolu par l'algorithme GMRES. On n'a pas à calculer explicitement la matrice $\tilde{\mathcal{J}} = M^{-1}\mathcal{J}$, on calcule plutôt le vecteur $b = \mathcal{J}\{\delta \mathbf{U}\}$ résultant de l'opération multiplication de la matrice par le vecteur. On résoud alors le système linéaire $Mx = b$ pour obtenir $M^{-1}\mathcal{J}\{\delta \mathbf{U}\}$. Le système linéaire $Mx = b$ est résolu en utilisant une factorisation incomplète ILUT de la matrice de préconditionnement M.[3]

Comme dans la factorisation LU standard, M est mise sous la forme $M = LU$ où L et U sont des matrices creuses triangulaires. L et U ont respectivement la même structure que la partie triangulaire supérieure et inférieure de la matrice jacobienne $\mathcal{J}$. Cependant, dans la factorisation ILUT on ne retient que quelques termes par ligne dans les matrices L et U. Le nombre de termes dépend de deux paramètres:

1. Un nombre entier qu'on notera **lfil**. C'est le nombre des plus grands termes qu'on garde par ligne dans L et U en plus des du terme diagonal. **lfil** aide à contrôler l'usage de l'espace memoire.
2. Un nombre réel qu'on notera **droptol**. Pour chaque ligne le produit de sa norme par **droptol** constitue un coeficient de tolerence où tous les termes de la ligne plus petits que ce dernier sont considérés comme nuls.

Quand la taille du problème est grande, le temps de calcul des matrices L et U peut être assez grand. Pour la majorité des problèmes en mécanique des fluides on se contente de les mettre à jour (via M) après quelques pas de temps plutôt qu'à chaque pas de temps. Dans le programme séquentiel la matrice de préconditionnement M est recalculée, par exemple, à chaque dix pas de temps.

Algorithme Newton-GMRES préconditionné

1. étant donné une solution initiale $\{\mathbf{U}\}^0$ Faire ,
2. Pour n= 1,2,... Faire:
3. $\{\mathbf{U}\}^{\mathbf{n},\mathbf{0}} = \{\mathbf{U}\}^{\mathbf{n-1}}$
4. Calculer $\mathcal{J}$ et sa factorisation ILUT si nécessaire
5. Pour i=1,...max-iteration, Faire

6. Calculer $\tilde{\mathcal{J}}(\{\mathbf{U}\}^{\mathbf{n,i-1}})$ et $M^{-1}\{R\}(\{\mathbf{U}\}^{\mathbf{n,i-1}})$,
7. Résoudre le système linéaire (14)
8. mettre à jour $\{\mathbf{U}\}^{\mathbf{n,i}}$, $\{\mathbf{U}\}^{\mathbf{n,i}} = \{\mathbf{U}\}^{\mathbf{n,i-1}} + \{\delta\mathbf{U}\}^{\mathbf{n,i}}$
9. mettre à jour les coordonnées
10. Si le critère de convergence est satisfait aller à 12,
11. Fin ,
12. Fin

Comme on l'a déjà mentionné, c'est le vecteur résultant de l'operation de multiplication de $\mathcal{J}^{n,i-1}$ par un vecteur $\{v\}$ qui est requis dans l'algorithme de GMRES. Cette multiplication n'est pas faite explicitement. Le vecteur résultant est calculé en utilisant l'approximation suivante:

$$\mathcal{J}(\{\mathbf{U}\}^{\mathbf{n,i-1}})\{\mathbf{v}\} \approx \frac{\{\mathbf{R}\}(\{\mathbf{U}\}^{\mathbf{n,i-1}} + \sigma\{\mathbf{v}\}) - \{\mathbf{R}\}(\{\mathbf{U}\}^{\mathbf{n,i-1}})}{\sigma} \tag{15}$$

où σ est un scalaire assez petit donné par :$\sigma = \epsilon(\|\{\mathbf{U}\}^{n,i-1}\| + \epsilon)$, avec $\epsilon = 10^{-6}$.

IV. Description de la structure du programme séquentiel

Le programme séquentiel d'élements finis **FES**[7] est destiné à la résolution des problèmes d'hydrodynamique avec transfert de chaleur et en présence d'une surface libre. Ce programme est écrit en Fortran77. Nous avons implementé dans FES l'allocation dynamique en utilisant un appel externe de la fonction malloc du langage C.

Comme tout programme d'éléments finis, les calculs dans FES se font en quatres étapes successives: structuration des données, calcul par éléments finis, résolution numérique et impression des résultats. Nous pouvons résumer ces étapes comme suit

- Lecture, vérification et organisation des données décrivant le maillage (noeuds, éléments, etc.), des paramètres connus liés à des éléments ou des noeuds (propriétés élémentaires ou nodales) et des conditions aux limites.
- Construction des matrices et vecteurs élémentaires, puis assemblage de ceux-ci pour former la matrice globale et le vecteur global. Plus précisement , pour chaque élément cela revient à:
  1. extraire les informations liées à cet élément
  2. construire la matrice et le vecteur élémentaire
  3. assembler dans une matrice globale et un vecteur global
- résolution numérique du système des équations après prise en compte des conditions aux limites,
- impression des résultats après calcul éventuel de variables additionnelles.

V. Parallélisation, concept et outils

Les objectifs principaux du parallélisme sont l'utilisation rationnelle de l'espace mémoire et la diminution du temps de calcul. La réalisation de ces objectifs permettera alors de s'attaquer à la résolution des problèmes de grande taille impossibles à résoudre sur des uni-processeurs .

Dans le code FES , on remarque que la plus grande partie de l'espace mémoire alloué par la structure de données est utilisée pour représenter la matrice de préconditionnement (voir table 1) . On remarque aussi que la majorité du temps d'exécution passe dans le calcul des résidus et la factorisation incomplète de la matrice de préconditionnement.

Comme on a pu le remarquer dans la description de FES, le calcul et l'assemblage de la matrice et du residu se font élément par élément. Ceci nous suggère une subdvision de l'ensemble des éléments en sous-ensembles de tailles plus petites. La boucle sur les éléments pourra ainsi être exécutée en parallèle.

Chacun des éléments autour d'un noeud donnera une contribution à la valeur associée à ce dernier lors de l'assemblage du résidu (ou la matrice). Lors de la partition de l'ensemble des éléments en sous ensembles, certains éléments adjacents (ayant des noeuds en commun) vont se voir affectés à différents sous ensembles. On se voit donc dans l'obligation d'utiliser ou construire un outil pour reconnaitre si un noeud appartient à un seul sous ensemble ou s'il est partagé entre deux ou plusieurs sous ensembles d'éléments (voir Figure 1). Cet outil (quelques routines de PSPARSLIB par exemple) est indispensable pour un calcul en parallèle du résidu ou de la matrice de préconditionnement.

V.A. Description de la librairie PSPARSLIB

PSPARSLIB est une librairie parallèle portable de solveurs itératifs pour résoudre des systèmes linéaires creux.[5]

Soit le système linéaire suivant:

$$Ax = b \tag{16}$$

où A est une grande matrice creuse, qui peut être régulièrement structurée ou non, b est le vecteur source et x est le vecteur des inconnues.

Dans PSPARSLIB le système (16) est premièrement partagé entre les processeurs. Le partage est fait soit en se basant sur une connaissance physique du problème soit en utilisant un partitionneur de graphe.[8] Le système est ensuite résolu avec une méthode de sous-espaces de Krylov préconditionnée.

PSPARSLIB présente des utilitaires pratiques pour faciliter le développement et l'implémentation des solveurs itératifs parallèles. Parmi le contenu de PSPARSLIB on peut citer:

- Quelques partitionneurs de graphe.
- Des routines pour former la structure locale des données en préparation à une eventuelle résolution du système linéaire. On a essentiellement des routines pour former et ordonner les matrices locales.
- Des accélérateurs de Krylov incluant entre autres GMRES.
- Quelques préconditionneurs incluant entre autres ILUT locale.

La librairie PSPARSLIB utilise des routines de communication du MPI (Message-Passing Interface).[9] La portabilité de PSPARSLIB dépend de celle de MPI. PSPARSLIB a été testée sur plusieurs plateformes telles que CRAY-T3D, le CRAY-T3E, LE IBM SP2, un réseau de RS600, et un réseau de SGI.

La plupart des routines de PSPARSLIB sont écrites en Fortran avec quelques modules en C. L'allocation dynamique est assurée par un appel externe de la fonction malloc du langage C.

V.B. Décomposition du domaine

La partition de graphe s'applique à la méthode des éléments finis afin de subdiviser un domaine discrétisé en sous-domaines. Le problème consiste à partitionner les éléments en parts plus au moins égales tout en minimisant le nombre de faces reliant les éléments dans les différents sous-domaines.

V.B..1 Concept de la partition de graphe

Un graphe (V, E) est composé d'un ensemble V de nœuds appelés sommets et d'un ensemble E d'arêtes reliant les sommets entre eux. L'arête E_{ij} décrit la connectivité entre les sommets V_i et V_j. Nous considérons le cas simple où une arête relie seulement deux sommets.

Subdiviser de façon optimale un domaine en sous domaines, revient donc à subdiviser le graphe (V, E) en sous-graphes $(V, E)_k$, contenant chacun un nombre fini de sommets et d'arêtes. Le nombre de noeuds locaux $(V, *)_k$ doit être plus ou moins égal dans tous les sous-ensembles, et le nombre d'arêtes E_{ij} tel que les sommets V_i et V_j appartiennent à des sous-ensembles différents doit être minimisé. De ce fait, les processeurs auront la même charge de travail (load balancing of the work), et la communication entre les processeurs sera minimisée.[8]

Nous appelons une partition de l'ensemble de sommets $(V, *)$, l'ensemble $(V, *)_1, (V, *)_2, ..., (V, *)_k$, des sous-ensembles des noeuds tel que:

$$(V, *)_l \subseteq (V, *) \ , \quad \cup_{l=1,k}(V, *)_l = (V, *)$$

Lorsque tous les sous ensembles $(V, *)_l$ ne sont pas disjoints deux à deux, nous parlerons d'une partition chevauchée (ovelapping partition). Dans le cas contraire, la partition est dite propre (proper partition).

Il existe deux grandes classes d'algorithmes utilisées pour la partition de graphe : l'approche géométrique et les techniques spectrales. L'approche géométrique utilise le maillage physique, elle nécessite donc la connaissance des coordonnées des points du maillage afin de faire un partitionnement adéquat.

La bissection spectrale se définit comme une technique qui exploite les propriétés du laplacien du graphe. À un graphe (V, E) donné, on associe une matrice laplacienne L creuse et définie comme suit:

$$l_{ij} = \begin{cases} -1 & \text{si } E_{ij} \in \text{E et } i \neq j \\ deg(V_i) & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

où $deg(V_i)$ est le degré du sommet V_i. Le degré d'un sommet V_i est le nombre de sommets V_j tels que $E_{ij} \in E$. Cette matrice L est symétrique, et on peut montrer qu'elle est semi-définie négative et que zéro est sa plus petite valeur propre. À la seconde plus petite valeur propre correspond un vecteur propre appelé le vecteur de Fiedler.[8] Ce vecteur est tel que les signes de ses composantes divise le domaine, correspondant au graphe en question, en deux parts plus ou moins egales. L'algorihme RSB (Recursive

Spectral Bissection)[8] consiste à partager les composantes du vecteur de Fiedler en assignant les noeuds d'un certain signe à un premier sous-domaine et les noeuds de l'autre signe au second sous-domaine. Les deux sous-domaines résultants sont partionnés à leur tour chacun en deux sous-domaines, jusqu'à ce qu'on obtienne le nombre de sous-domaines voulu.

V.B..2 considération basée sur les éléments

Un partitionnement adéquat peut être réalisé en utilisant des partitionneurs de graphe très performants comme RSB et METIS. Dans nôtre cas on a utilisé METIS.

L'utilisation de METIS nécessite la préparation d'un fichier. Ce dernier est sous forme d'une succession de lignes et dans chaque ligne $i+1$ figurent les numéros des éléments qui ont au moins un noeud en commun avec l'élément i. La première ligne contient le nombre total des éléments suivi du nombre total des arêtes.

Dans notre situation, la correspondance entre le domaine spatial et un graphe (V,E) va se faire en identifiant les sommets du graphe aux numéros des éléments. Un sommet V_i est relié à un autre sommet V_j par une arête E_{ij}, si les éléments correspondant à ces deux sommets ont au moins un noeud commun. Il faut noter que seuls les éléments de volumes sont considérés lors de cette correspondance domaine-graphe.

Par exemple, si la ligne 2 du fichier préparé pour METIS contient 12 chiffres (numéros), cela veut dire que l'élément numéro 1 possède 12 éléments partageant chacun avec lui au moins un noeud. On peut dire aussi, que du sommet 1 partent 12 arêtes qui le connectent à 12 sommets différents. Si on note $NART$ le nombre total des numéros figurants sur toutes les lignes, alors le nombre total des arêtes sera simplement $NART/2$. $NART$ est évidemment un multiple de 2.

Une fois le fichier en question préparé, METIS est très simple à utliser et il nous fournit un fichier dont chaque ligne contient un seul entier. Si $ndom$ est le nombre total de sous domaines voulus par l'usager, alors la ligne i contiendra un entier $nsdmn$ tel que $(0 \leq nsdmn \leq ndom - 1)$. Celà veut dire que l'élément i est contenu dans le sous-domaine (processeur) identifié par l'entier $nsdmn$. On fait correspondre ensuite les éléments de surface aux éléments de volumes correspondants. Le fichier résultant de METIS sera augmenté par l'information concernant les éléments de surface.

Cette étape, appelée prétraitement, est faite à l'extérieur du code en question. Le fichier produit par cette étape sera un fichier de données pour le code d'éléments finis.

V.C. structuration des données locales et calcul du Résidu

Deux stratégies peuvent être adoptées, lors de la préparation des données locales pour les solveurs itératifs de PSPARSLIB. Dans la première, le partitionnement est fait à l'extérieur du code (étape de prétraitement), et chaque processeur crée sa part locale du système à résoudre. Cette première approche s'adapte très bien à la méthode des éléments finis. Dans la deuxièmme stratégie, un processeur lit entièrement le système à résoudre et le distribue alors entre les processeurs. Le système (entier) linéaire creux et 'distribué' est alors résolu par une méthode de sous-espaces de Krylov préconditionnée.[5]

C'est la première stratégie qui est adoptée dans le présent travail. Un partitionnement adéquat du domaine, est fait en utilisant le partitionneur de graphe METIS. La librairie PSPARSLIB n'a besoin que du tableau (mapping) qui resssort du partitionneur. Ce tableau sera introduit comme paramètre dans des routines de PSPARSLIB afin de préparer la structure des données locales pour le solveur itératif.

Nous allons présenter une façon d'utiliser quelques routines de PSPARSLIB dans le but de calculer

un résidu global 'distribué' sur plusieurs processeurs en assemblant des résidus locaux. Chaque résidu local est présent sur un processeur.

V.C..1 Représentation des données locales:

Les éléments de volumes sont départagés entre les processeurs en utilisant METIS. Ensuite, une interface est utilisée pour faire correspondre les éléments de peau (de surface) aux éléments de volume correspondants. Le tableau *list* résultant de cette procédure de partitionnement est de dimension $NELT$ où $NELT$ est le nombre total des éléments. *list* prend des valeurs positives supérieures ou égales à zéro. Par exemple 'list(1)=myproc' signifie que l'élément 1 est contenu dans le processeur (sous domaine) identifié par l'entier myproc.

En utilisant le vecteur *list* chaque processeur construit un vecteur *maploc* de dimension *nloc*. Les valeurs de *maploc* sont les équations locales en numérotation globale. Ces équations correspondent aux noeuds des éléments contenus dans le processeur identifié par l'entier *myproc*. Comme on s'y attendait, certaines équations sont partagées entre plusieurs processeurs. On parle ainsi d'équation partagée et non partagée. On empile (gather) dans un ordre croissant d'identificateurs de processeurs, les vecteurs locaux *maploc* dans un vecteur *mapglob* de dimension *nsum*.

Soient *numproc* et *pointer* respectivement le nombre de processeurs (sous-domaines) et un tableau de pointeurs de dimension $numproc + 1$ tel que $pointeur(numproc + 1) = nsum + 1$. Si par exemple $pointeur(2) = 50$, cela signifie que pour le deuxième processeur l'emplacement de ses équations dans *mapglob* commence à la position 50. L'appel suivant aux routines **nod2dom** et **getjamap** de PSPARSLIB:

```
Call nod2dom (nsum,numproc,mapglob,pointeur,mapneuproc,...)
Call getjamap (nloc,maploc,mapneuproc,map,...)
```

produit un vecteur *map* à valeurs entières tel que $map(i) = nmap$. Si l'entier *nmap* est supérieur ou égal à zéro, cela signifie que l'équation i est non partagée et qu'elle est contenue dans le processeur *nmap*. Si *nmap* est inférieur à zéro, alors $-nmap$ contient la position, dans le vecteur *map*, où commence la liste des processeurs auxquels l'équation i appartient. Une fois obtenu le tableau *map*, on peut libérer l'espace mémoire du tableau *mapglob* qui n'est d'aucune utilité pour la suite des opérations.

L'information pour trouver les processeurs adjacents à un processeur *myproc* donné est donc contenue maintenant dans le vecteur *map*. Deux processeurs sont adjacents s'ils se partagent au moins une équation. Pour un processeur *myproc* donné, l'équation i est non partagée si $map(i) = myproc$. Si $map(i)$ est différent de *myproc* alors $-map(i)$ contient l'adresse de la liste chainée des processeurs se partageant l'équation i. La routine **bdry** de PSPARSLIB, basée sur ce principe, est utilisée pour trouver le nombre $NPROC$ et la liste $PROC$ des processeurs adjacents d'un processeur *myproc* donné. La routine **bdry** produit aussi la liste ix des équations contenues dans *myproc* et qui sont partagées avec les processeurs adjacents. Ces équations sont fournies dans un ordre correspondant à un tableau de pointeurs *ipr*. Si $ipr(j) = 50$ alors les équations contenues dans myproc et en même temps dans le procésseur j commencent à la position 50 dans le vecteur ix. Le nombre de ces équations est $ipr(j + 1) - ipr(j)$.

La routine **bdry** produit aussi un vecteur *riord* de dimension $2 * nloc$. Les premières *nloc* valeurs de *riord* sont équivalentes à *maploc* réordonné: les premières $nbnd - 1$ valeurs sont les équations non partagées et les valeurs restantes sont des équations partagées dans un ordre correspendant à la liste $PROC$. Ces équations sont équivalentes à la liste ix.

```
Call bdry (nloc,numproc,myproc,ia,map,riord, ...,nbnd,nproc,ix,ipr,proc,...)
Call setup (nloc,nbnd,...,nproc,proc,ix,ipr,...,jb,...,iord,riord,...)
```

Pour qu'un processeur *myproc* possède une structure de données locales complète, il lui reste à savoir dans quel ordre les équations partagées vont lui être envoyées par ses processeurs adjacents. Aussi, il doit associer à la numérotation globale d'origine une numérotation locale. Ainsi la routine **setup** de SPARSLIB produit un vecteur *iord* équivalent à *riord* mais en numérotation locale. De plus la routine **setup** donne une numérotation locale à la liste ix.

Donc, après l'appel à **setup**, chaque processeur *myproc* devra avoir les informations suivantes, qui définissent sa structure locale de données:

1. **nproc:** Le nombre de processeurs adjacents, c.a.d, les processeurs avec lesquels le procésseur *myproc* partage au moins une équation.
2. **Proc(1:2*nproc)**: les premières **nproc** valeurs sont les identificateurs des processeurs adjacents. Les dernières **nproc** valeurs sont les nombres des équations partagées avec chacun des processeurs adjacents.
3. **ix:** la liste des équations contenues dans *myproc* et qui sont partagées avec des processeurs adjacents. La liste est organisée processeur par processeur , dans le même ordre que **Proc**, afin que l'échange de données se réalise facilement en utilisant une liste de pointeurs.
4. **ipr:** pointeurs dans la liste **ix** des débuts des équations partagées avec chaçun des processeurs adjacents.

   Nous allons utiliser ces informations pour former une structure locale de données. Cette structure est employée pour représenter les vecteurs locaux (résidu, solution, ...) et matrices locales (matrice jacobienne).

V.C..2 Calcul du Résidu :

Dans le but de calculer le résidu en utilisant un vecteur global distribué, nous aurons besoin de communiquer certaines composantes du vecteur résidu correspondant à des équations partagées entre des processeurs adjacents. Nous noterons $VRES_{loc}$ le résidu local, c.à.d le résidu correspondant à des équations contenues dans le processeur *myproc*.

Nous avons mentionné lors de la description du code séquentiel FES que le calcul du résidu et son assemblage sont exécutés élément par élément. La boucle de calcul du résidu va se faire sur $NELLOC$, où $NELLOC$ est le nombre d'éléments contenus dans le processeur *myproc*. Calculées localement, certaines composantes du vecteur $VRES_{loc}$ correspondant à des noeuds d'interfaces, ne seront affectées que par des valeurs partielles intermédiaires. Et une communication est nécessaire afin de calculer un résidu réel final. Le calcul de $VRES_{loc}$ sur chaque processeur, peut être représenté par le fragment du code suivant:

Pour elem = 1,$NELLOC$ Faire
Call EXTR(extraire les informations de elem)
Call RESIDU (construire le vecteur elementaire de elem)
Call ASSBL(Assembler dans le vecteur $VRES_{loc}$)
Fin

Notons que $VRES_{loc}$ est ordonné de la même foçon que *riord*. Après l'assemblage, nous échangeons des résultats intermédiaires entre les différents $VRES_{loc}$ situés sur les différents processeurs et nous calculons un résidu final. Ces opérations peuvent être représentées par le fragment de code suivant:

Call MSG-bdx-sendi (echanger des composantes entre les $VRES_{loc}$, utiliser un tableau jb)

```
Pouri = nbnd,nloc Faire
    sum=VRES_loc(i)
    len1=1
    Pour ip = 1, nproc Faire
      len2=len1+ PROC(nproc+ip)
      Pour j = len1, len2-1 Faire
        Si (riord(i) .eq. jb (j)) Alors Faire
          sum=sum+ VRES_loc(nloc+j)
          goto 20
        FinSi
      Fin
20    len1=len2
    Fin
    VRES_loc(i)=sum
Fin
```

V.C..3 Préconditionnement de type factorisation incomplète ILUT par sous-domaine

Afin de tirer avantage du grand nombre de coefficients nuls dans la matrice de préconditionnement M, cette dernière est stockée sous un format appelé CSR (compressed sparse row). La matrice M aura une nouvelle structure de données composée de trois tableaux qui ont les fonctions suivantes:

1. Un tableau AA contenant des valeurs réelles M_{ij} non nulles stockées ligne par ligne dans l'ordre croissant des lignes. La longueur du tableau AA est NNZ.
2. Un tableau JA à valeures entières, contenant les indices des colonnes des coefficients M_{ij} non nuls.
3. Un tableau IA à valeurs entières, contenant les pointeurs de débuts des lignes dans les tableaux AA et JA

Considérons par exemple la matrice suivante:

$$M = \begin{pmatrix} 12. & 0. & 0. & 11. & 0. \\ 10. & 9. & 0. & 8. & 0. \\ 7. & 0. & 6. & 5. & 4. \\ 0. & 0. & 3. & 2. & 0. \\ 0. & 0. & 0. & 0. & 1. \end{pmatrix} \tag{17}$$

Cette matrice M, peut être stockée comme suit:

```
AA = 12.  11.  10.  9.  8.  7.  6.  5.  4.  3.  2.  1.
JA = 1  4  1  2  4  1  3  4  5  3  4  5
IA = 1  3  6  10  12  13
```

Comme nous le savons, le calcul de la matrice et son assemblage sont réalisés élément par élément. Pour ses éléments locaux, chaque processeur va faire l'assemblage pour obtenir une matrice locale qu'on notera M_{loc}.

Dans le code séquentiel FES, on lit un fichier contenant les données NNZ, le tableau JA et le tableau IA. Dans le programme parallèle, on va construire des tableaux locaux IA_{loc} et JA_{loc} en se basant sur la

connaisance des équations locales dans chacun des processeurs. IA_{loc} et JA_{loc} sont utilisés par la suite pour assembler M_{loc}. Le préconditionnement ILUT est localement appliqué sur M_{loc}. L'assemblage et le préconditionnement de M_{loc} sont représentés par le fragement du code suivant:

Pour elem = 1,$NELLOC$ Faire
Call EXTR(extraire les informations de elem)
Call CALMAT (construire la matrice elementaire de elem)
Call ASSBL(Assembler dans la matrice locale M_{loc})
Fin
Call ILUT (factorisation incomplete de M_{loc})

On peut remarquer que le calcul de M_{loc} ne nécessite aucune communication entre les processeurs.

V.C..4 Communications dans l'algorithme de GMRES

L'orthogonalisation dans GMRES d'un vecteur Mv_i par rapport aux vecteurs v calculés auparavant, est accomplie via un processus de Gram-Schmidt modifié. Ce dernier est basé simplement sur les deux opérations suivantes:

- Calculer $\alpha = (y, v)$.
- Calculer $z = y - \alpha v$.

Dans la première opération on calcule un produit scalaire de deux vecteurs y et v, et dans la deuxième on affecte le vecteur z du résultat de l'operation $y - \alpha v$.

Les deux opérations orthogonalisent un vecteur z par rapport à un autre vecteur v de norme unitaire. On peut remarquer que ces deux opérations peuvent s'éxecuter en parallèle par les processeurs, et cela sur des données locales différentes.

L'opération produit scalaire utilise toutes les composantes des vecteurs y et v pour calculer un scalaire α, lequel est demandé par tous les processeurs pour calculer z. Mentionnons ici que les vecteurs y et v sont distribués de la même façon entre les processeurs. Alors le produit scalaire se traduira en *numproc* produits sacalaires (chacun sur un processeur) suivis d'une communication. Spécifiquement, si *nloc* est la dimension locale des vecteurs locaux y_{loc} et v_{loc} pour le processeur *myproc*, nous pouvons représenter le calcul du produit scalaire par le fragment de code suivant:

$tloc$ = DDOT($nloc$, x_{loc},1,y_{loc},1)
Call MPI-allreduce(α_{loc}, α,....)

La fonction DDOT calcul le produit scalaire de deux vecteurs x_{loc} et y_{loc} de dimension *nloc* chacun. La routine MPI-allreduce somme tous les scalaires α_{loc} à partir de chaque processeur et met le résultat global dans une variable t sur chaque processeur.

La deuxième opération pour le calcul du vecteur z, ne nécessitera pas de communication entre les processeurs. Ayant le même saclaire t, les processeurs font la même opération sur les vecteurs locaux x_{loc} et y_{loc}, et le résultat sera le vecteur z " distribué ".

À côté des opérations produit scalaire et SAXSPY, GMRES utilise le calcul de la norme d'un vecteur. En s'inspirant du calcul du produit scalaire, on peut calculer la norme d'un vecteur v de la façon suivante:

$normloc$ = DDOT($nloc$, v_{loc},1,v_{loc},1)
Call MPI-allreduce($normloc, norm$,....)
$norm$ = SQRT($norm$)

où SQRT est la fonction racine carrée.

VI. Résultats numériques

Les calculs sont exécutés sur un Sun-Enterprise et un SGI-ONYX2. Le cas test est une modélisation numérique d'un écoulement à surface libre du barrage Sainte-Marguerite.[7] Le domaine de calcul est constitué de 142 462 éléments tétraédriques (éléments de volumes **3D**), de 17 326 éléments triangulaires (éléments de surfaces **2D**) et de 28 528 noeuds. Chaque noeud du domaine a 5 degrés de libérté et le nombre total d'équations est égale à 109448 . Le pas de temps est de 0.1 secondes. En ce qui concerne les paramètres du solveur, on a une dimension 8 pour le sous-espace de Krylov , un maximum de 4 iterations de Newton par pas de temps, et le critère de convergence utilisé est de 10^{-6}.

Sur toutes les Figures de convergence en fonction du temps, on a représenté le logarithme neperien du résidu au lieu du logarithme décimal. La raison c'est qu'en logarithme décimal, les différences lors des comparaisons des résultats restent difficiles à percevoir. Seules les Figures(2-4) sont associées avec des calculs où le schéma implicite de premier ordre (voir Eq.5) pour disrétiser les dérivées temporelles a été adopté. Toutes les autres Figures sont associées àvec des calculs utilisant un schéma du second ordre dans le temps(voir Eq.6). Aussi, c'est parceque les résultats correspondant à un schéma du premier ordre présentent des oscillations pour $lfil = 5$ et $lfil = 10$, que nous avons considéré dans tous les tests un $lfil \geq 15$.

La Figure 2 indique la convergence de la solution en fonction du temps pour le code séquentiel (lfil=5), et le code paralèlle dans le cas $lfil = 15$ pour différents nombres de processeurs. Nous avons représenté 100 pas de temps. Le pas du temps adopté est le même (0.1 secondes) que dans le programme séquentiel. La matrice jacobienne, comme dans le cas séquentiel, est recalculée à chaque dix pas de temps. Le résidu initial est égale à environ 712. Les courbes correspondant à l'utilisation de 4 et 8 processeurs donnent des résultats satisfaisants comparés à ceux donnés par le code séquentiel. Au lieu d'une stagnation dans le temps autour d'un résidu égal à environ 6, on a une stagnation autour d'un résidu égal à environ 8. L'allure des deux courbes est semblable à celle du code séquentiel. Nous avons remarqué dans le code parallèle et pour différents nombre de processeurs (2-8) une convergence dans le temps de la surface libre comparable à celle dans le code séquentiel.

La Figure 3 indique la convergence de la solution pour le code séquentiel (lfil=5) et le code paralèlle dans le cas $lfil = 25$ pour différent nombres de processeurs. Nous remarquons une amélioration très nette de la convergence spécialement pour le cas correspondant à l'utilisation de 2 processeurs.

En augmentant $lfil$ à 35 on remarque d'après la Figure 4, que les courbes correspondant à l'utilisation de 2, 4 et 8 processeurs s'approchent encore plus de la courbe correspondant au cas séquentiel. aussi, les résultats de convergence correspondant à 8 processeurs deviennent meilleurs que ceux de 4 processeurs. Cela montre que la décomposition du domaine n'affectent pas la convergence dans la mesure où on prend un $lfil$ suffisant. On peut dire que l'augmentation de $lfil$ améliore le préconditionneur, et par conséquent la convergence. Cependant, plus $lfil$ augmente et plus cela nécessité d'espace mémoire (voir tableaux 1-3). En pratique on utilisera un $lfil$ qui assure la robustesse de la résolution.

En utilisant un schéma implicite de second ordre pour disrétiser les dérivées temporelles (voir Eq.6), on obtient les courbes de convergence correspondant à l'utilisation de 2, 4 et 8 processeurs. Plus on augmente $lfil$ et plus ces courbes s'approchent de la courbe du séquentiel spécialement après les dix premiers pas dans le temps (voir Figures 5-7). La Figure 8 montre que la convergence devient meilleure

(plus rapide) que dans le cas séquentiel si on utilise 2 processeurs et un $lfil = 35$. Ce genre de situation où le parallélisme du préconditoinneur rend sous certaines conditions la convergence plus rapide que le cas séquentiel a aussi été rapporté dans la référence [10]. Il s'agissait du cas bidimensionnel concernant un fluide compressible et un partage spécial entre les processeurs des blocks de la matrice de préconditionnement.

Les Figures 9 et 10 montrent que le calcul de la matrice de préconditionnement à chaque 5 pas de temps plutôt qu à chaque dix pas n'influence pas beaucoup la convergence de la solution en fonction du temps.

La Figure 11 représente les résutats du tableau 4. Il s'agit du temps total d'exécution de dix pas dans le temps en fonction du nombre de processeurs. Le temps correspondant à 1 processeur est en fait le temps requis pour exécuter 10 pas dans le temps en utilisant le code séquentiel.

On peut dire que l'algorithme parallèle GMRES préconditionné avec une factorisation ILUT par sous-domaine (ILUT locale), possède donc des performances très prometteuses pour constituer une méthode de résolution des systèmes non symétriques de grande taille.

VII. CONCLUSION

Nous avons presenté dans ce papier un préconditionnement parallèle de type ILUT par sous-domaine (ILUT locale). Le domaine spatial est partitionné en sous-domaines en utilisant l'interface METIS. La matrice jacobienne est construite localement sur chaque processeur en ne considérant que les éléments traités par ce processeur. Une factorisation de type ILUT est appliquée sur cette matrice dans le but de résoudre par la méthode des éléments finis, des équations tridimentionnelles de Navier-Stokes pour un fluide incompressible couplées à une équation d'une surface libre. Nous avons constaté d'après les résultats numériques, que pour un nombre de processeurs modéré (2 à 8) le programme parallèle donne un bonne convergence de la solution en fonction du temps. Aussi, la convergence s'améliore en augmentant la valeur de $lfil$ et peut être sous certaines conditions plus rapide que la convergence du code séquentiel.

Dans le prochain travail, Nous allons utiliser des partitions chevauchées des éléments spatiaux. Ce genre de partitions dont lesquelles le degré de chevauchement (c.a.d le nombre des éléments communs á différents sous-domaines) peut varié, améliore l'efficacité des préconditionneurs locaux, et par conséquent la convergence de la solution.

REFERENCES

[1] A. Soulaimani and Y. Saad, "An Arbitray Lagrangian-Eulerian Finite Element Method For Solving Three-Dimensional Free Surface Flows," *accepté dans Computer Methods in Applied Mechanics ans Engineering* (1997).

[2] Y. Saad and M.H. Schutz, "GMRES: A Generalized Minimal Residual algorithm for solving nonsymetric linear systems," SIAM J. Sci. Statist. Comput. 7 (1986) 856-869.

[3] Y. Saad, "ILUT: a dual threshold incomplete ILUT factorization," Numerical Linear Algebra with applications, 1:387-402, 1994

[4] P. Chin, E.F. D'Azevedo, P.A. Forsyth and W.P. Tang, "Preconditioned conjugate gradient methods for incompressible Navier-Stokes equations" INT J. Numer. Methods Fluids 15 (1992) 273-295.

[5] Y. Saad and A. Malevsky, "PSPARSLIB: A portable library of distributed memory sparse iterative solvers," in Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, st. Petersburg, sept. 1995.

[6] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregu-

lar graphs," Technical Report TR 95-035, Departement of Computer Science, University of Minnesota,1995.

[7] A. Soulaimani and N. EL Kadri , "FES: Une méthode d'éléments finis pour des problèmes d'Hydrodynamique avec transfert de chaleur et en présence d'une surface libre" Rapport technique, Département de génie mécanique, Ecole de technologie superieure, Montréal, 1997.

[8] Y. Saad, Iterative Methods for Sparse Linear Systems , International Thomson Publishing Inc 1996

[9] W. Gropp, E. Lusk and A. Skjellum, Usinp MPI: Portable Parallel Programming with the Message-Passing Interface, MIT Press, Cambridge, MA, 1994

[10] L. C. Dutto, W. G. Habashi, M. Fortin "Parallelizable block diagonal preconditioners for the compressible Navier-Stokes equations" , Comput. Methods Appl. Mech. Engrg. 117 (1994) 15-47.

Nombre-processeurs	NNZ-matrice jacobienne	mots réel total
1	5968718	28681886
2	3052383	18381173
4	1478166	9689697
8	819279	5927854

Table 1: Espace memoire maximal alloué par processeur, lfil=15 sauf pour 1 processeur (le cas 1 processeur est pour lfil=5)

Nombre-processeurs	NNZ-matrice jacobienne	mots réel total
1	5968718	28681886
2	3052383	20111843
4	1478166	10538937
8	819279	6388084

Table 2: Espace memoire maximal alloué par processeur, lfil=25 sauf pour 1 processeur (le cas 1 processeur est pour lfil=5)

Nombre-processeurs	NNZ-matrice jacobienne	mots réel total
1	5968718	28681886
2	3052383	18381173
4	1478166	11388177
8	819279	6848314

Table 3: Espace memoire maximal alloué par processeur, lfil=35 sauf pour 1 processeur (le cas 1 processeur est pour lfil=5)

Nombre-processeurs	Temps Sun-Entreprise	Temps SGI-ONYX2
1	10305 sec	3540 sec
2	5312 sec	2015 sec
4	2822 sec	1150 sec
8	1523 sec	851 sec

Table 4: Temps pour exécuter 10 pas dans le temps avec un seul calcul de la matrice (le cas 1 processeur correspond au code séquentiel)

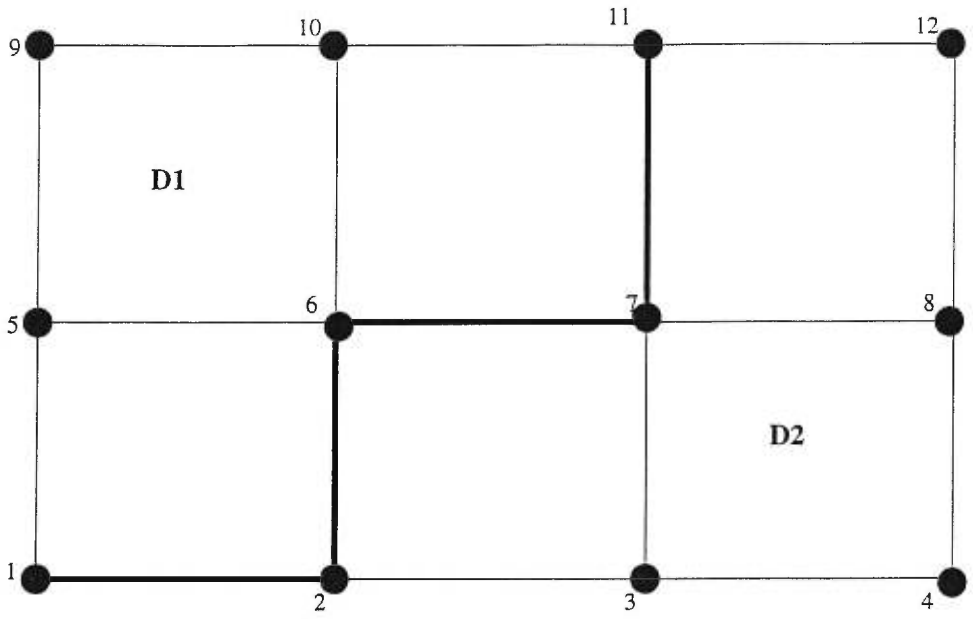

Figure 1: Décomposition basée sur l' élément d'un domaine en 2 sous-domaines

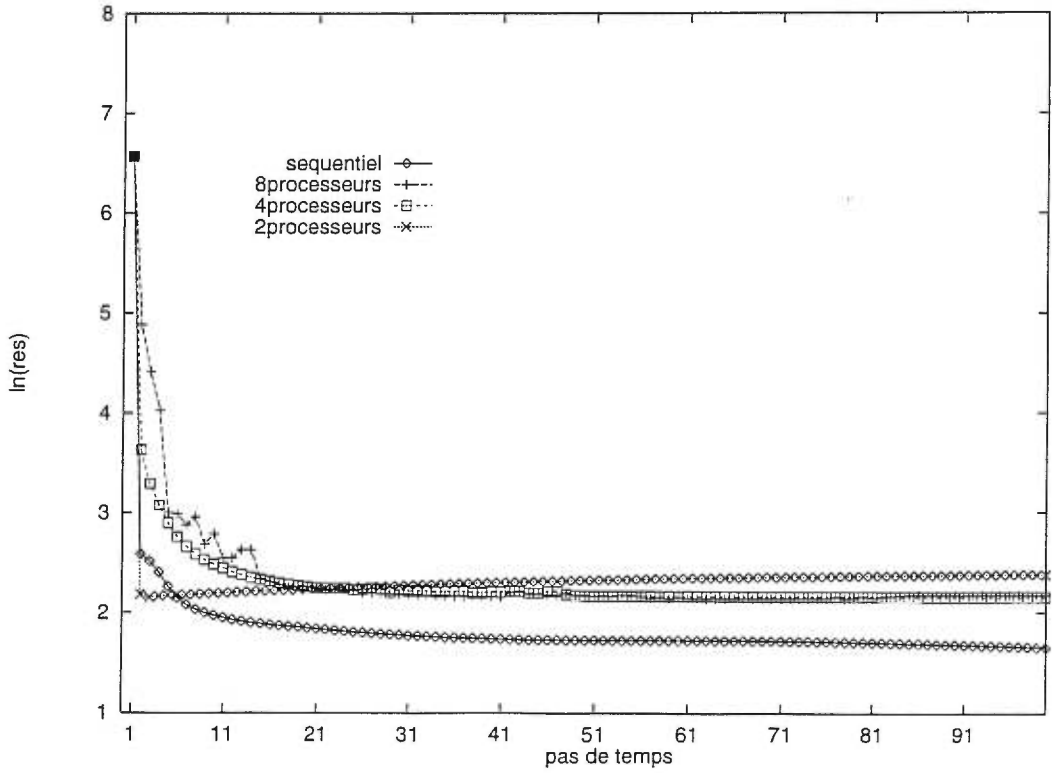

Figure 2: Convergence en fonction du temps pour un lfil=15 (premier ordre)

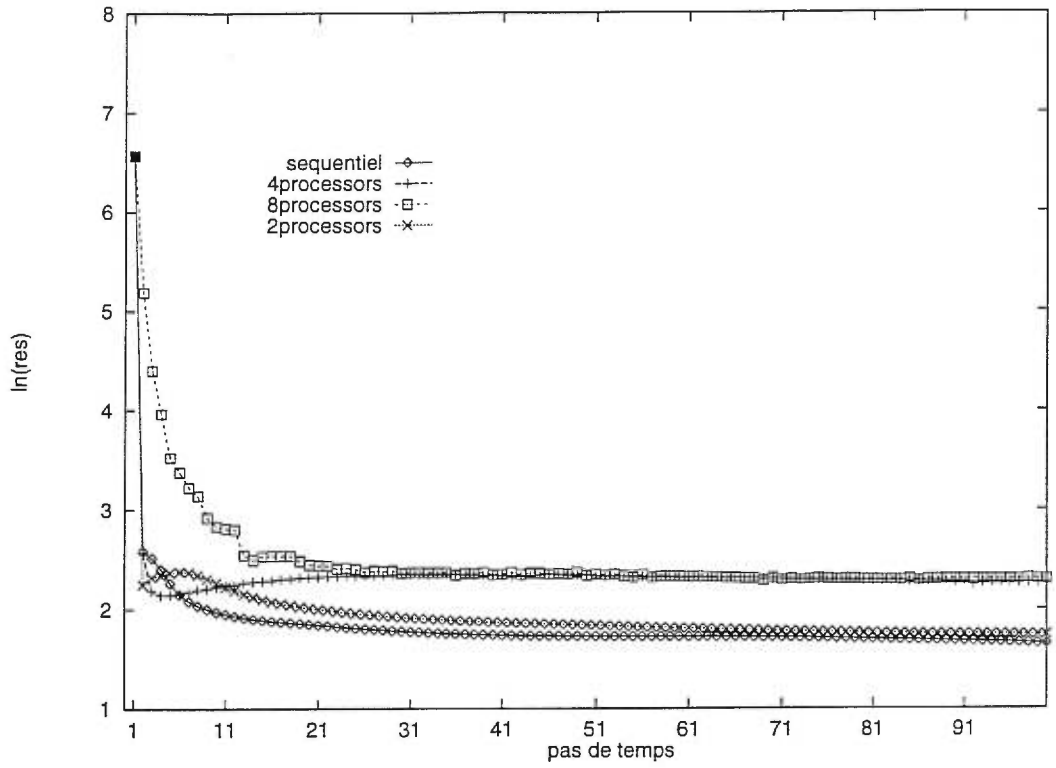

Figure 3: Convergence en fonction du temps pour un lfil=25 (premier ordre)

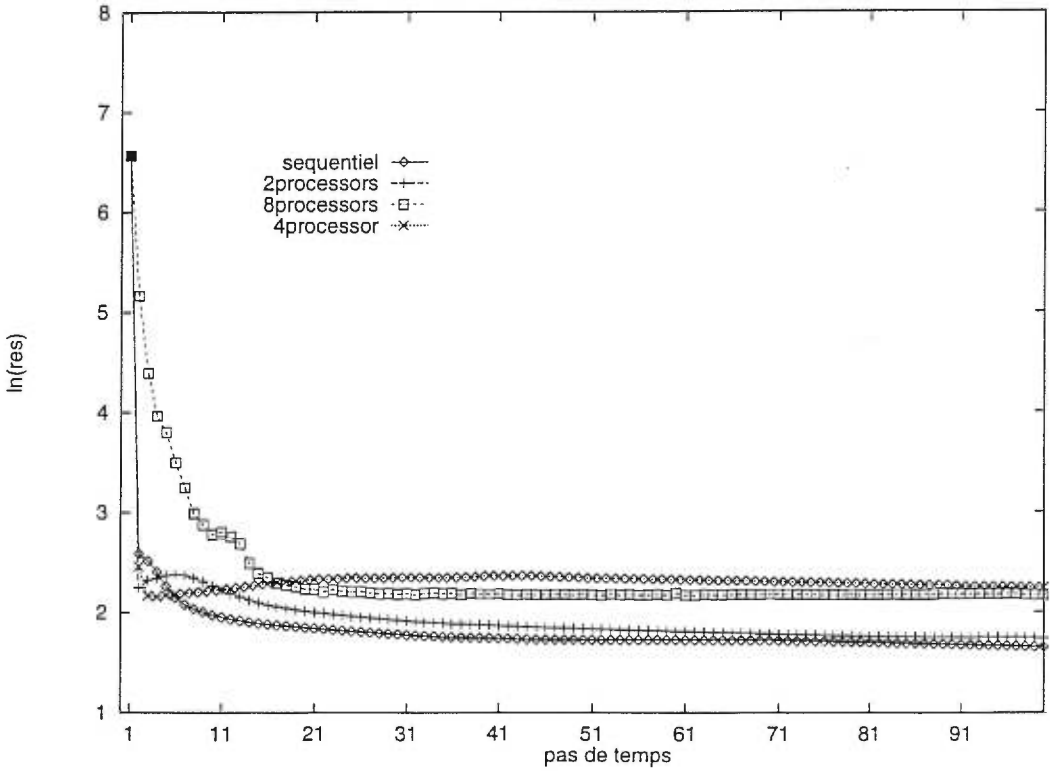

Figure 4: Convergence en fonction du temps pour un lfil=35 (premier ordre)

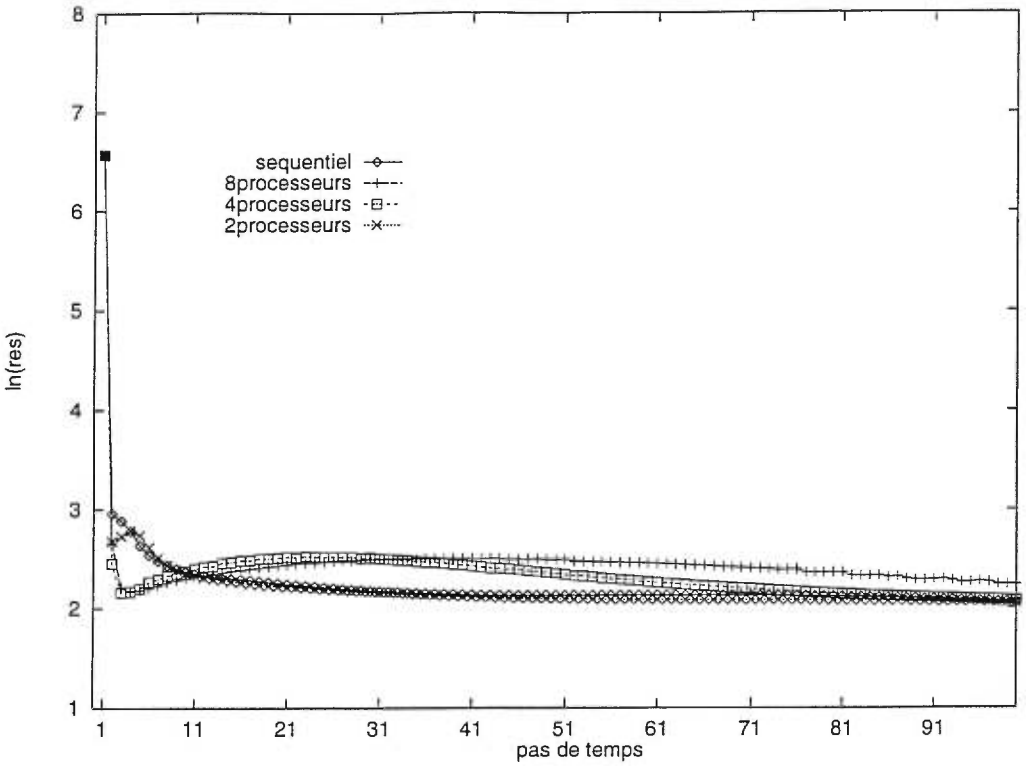

Figure 5: Convergence en fonction du temps pour un lfil=15 (second ordre)

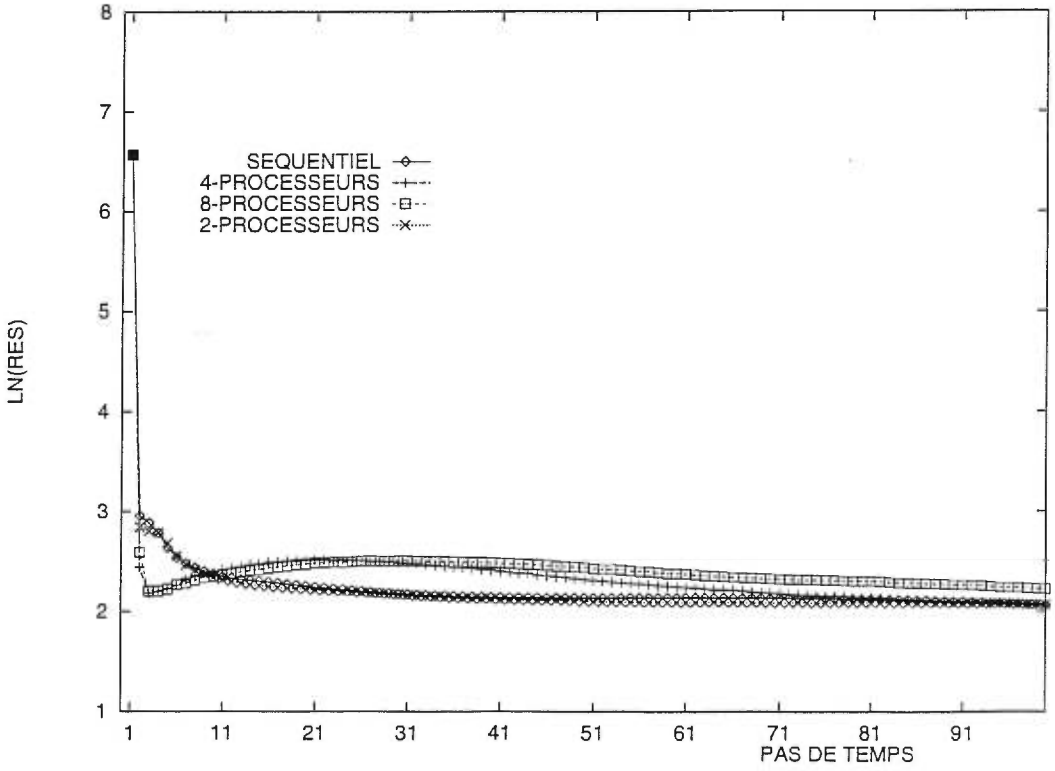

Figure 6: Convergence en fonction du temps pour un lfil=25 (second ordre)

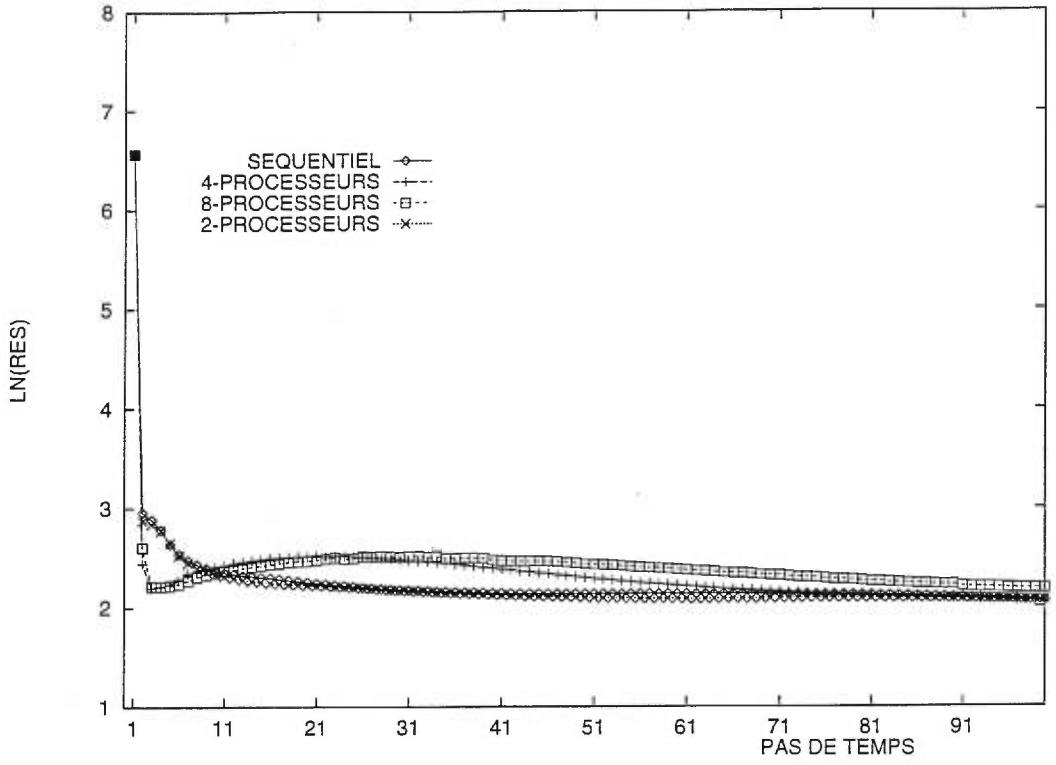

Figure 7: Convergence en fonction du temps pour un lfil=35 (second ordre)

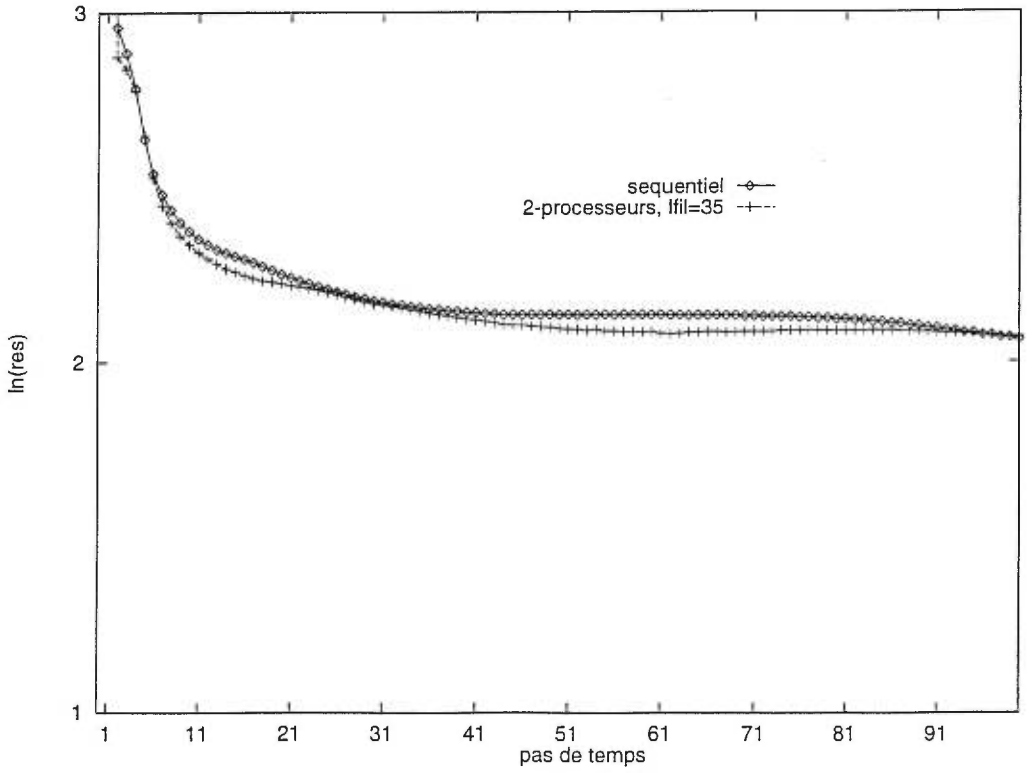

Figure 8: Comparaison de convergence entre éxecution séquentiel et parallèle (second ordre)

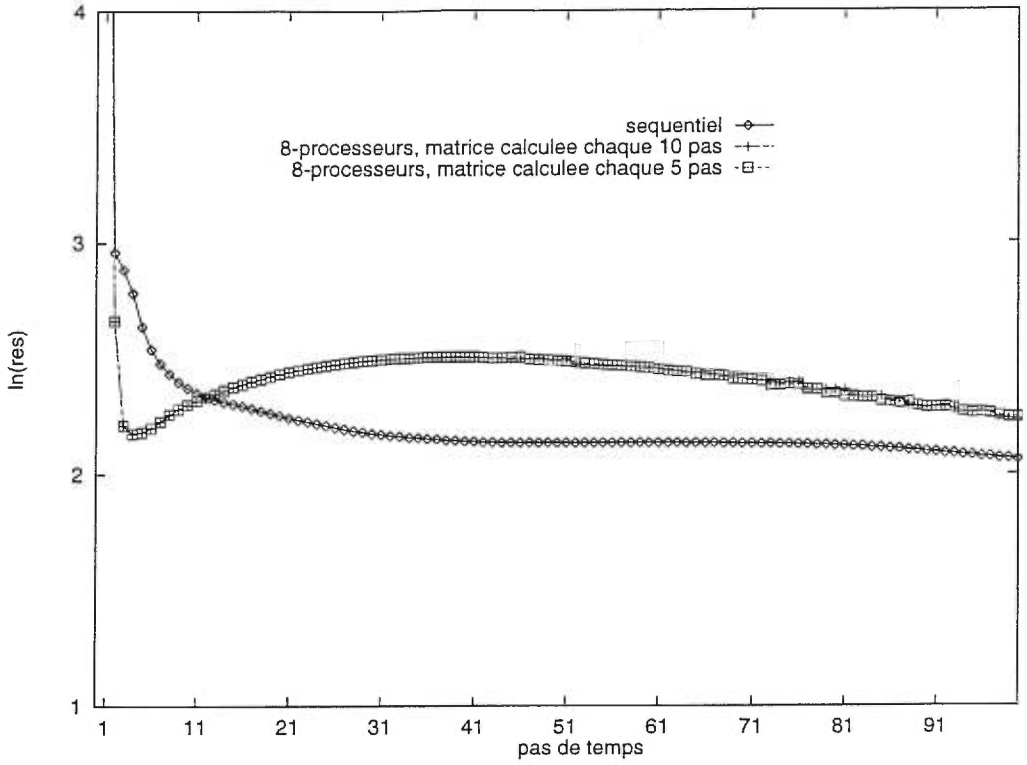

Figure 9: Convergence avec un calcul plus fréquent de la matrice (second ordre)

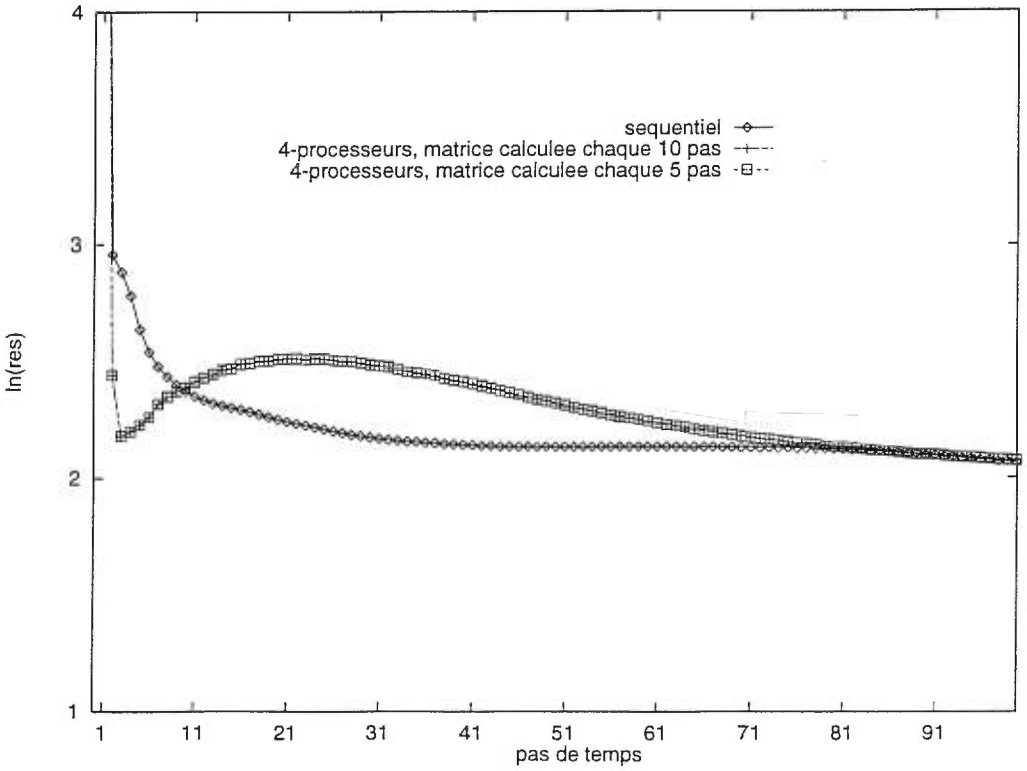

Figure 10: Convergence avec un calcul plus fréquent de la matrice (second ordre)

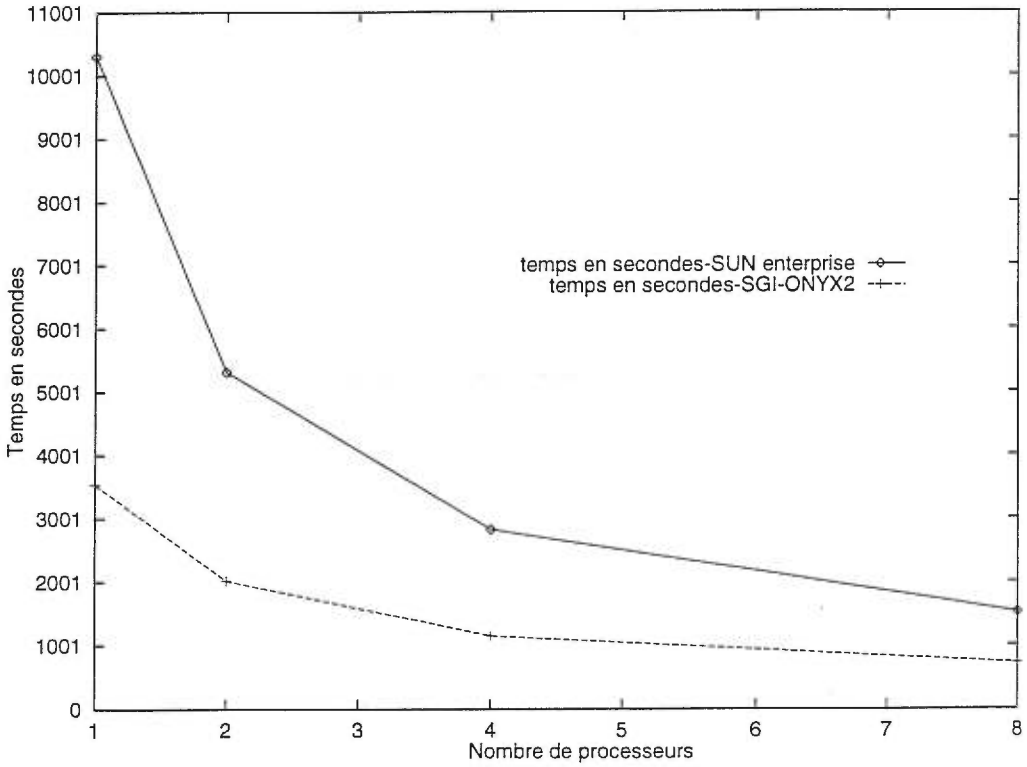

Figure 11: Temps en secondes correspondant à 10 pas dans le temps avec un seul calcul de la matrice

Conclusion

Cette thèse a été présentée sous forme de six articles. Les premiers articles (1-5) traitent des applications du calcul parallèle itératif en neutronique, articles qui représentent l'évolution de notre travail dans ce domaine. Le dernier article (6) est, lui aussi une application du calcul parallèle itératif au domaine de la mécanique des fluides.

Conséquence du choix d'un tel format, des répétitions se sont nécessairement glissées dans la succession des articles, aussi je vais reprendre brièvement l'évolution des six articles, suivront alors quelques perspectives quant aux calculs parallèles en neutronique.

Le premier article constitue l'introduction du parallélisme aux calculs de transport basées sur une méthode de probabilités de collision (PC) et utilisant un formalisme multigroupe. Ce calcul est indispensable pour obtenir des données sur les propriétés physiques nécessaires aux simulations numériques, elles mêmes essentielles pour assurer le suivi d'un réacteur nucléaire. De fait, ce premier article constitue une des toutes premières tentatives dans ce domaine pour paralléliser le solveur de l'équation de transport basée sur des CP. Un partitionnement (multigroupe) basé sur les groupes d'énergie a permis d'exécuter, sans recours à des communications, le calcul parallèle des matrices des probabilités de collision. Ces dernières restent en effet indépendantes d'un groupe d'énergie à un autre. Par contre les calculs du flux et du facteur de multiplication effectif qui se font par un processus itératif, ont nécessité des communications des flux récemment calculés. En effet, le terme source contient des termes de couplage entre les différents groupes d'énergie.

Le deuxième article est consacré au développement d'un nouveau solveur multirégion parallèle basé sur un partitionnement des zones spatiales. Le domaine spatial étant subdivisé en sous-domaines, un processus itératif avec des communications des flux récemment calculés entre les sous-domaines, permet de retrouver une distribution du flux et un facteur de multiplication effectif égaux à ceux du cas multigroupe.

Nous avons alors passé à des comparaisons de performances des deux algorithmes multirégion et multigroupe sur un calcul de cellule unidimensionnelle et un autre de cellule bidimensionnelle (Mosteller benchmark), sujet du troisième article.

Dans le quatrième article, la performance des deux algorithmes parallèles présentés a été augmentée par deux techniques d'accélérations, le rebalancement neutronique et la minimisation du résidu. Finalement l'objet principal du cinquième article fut l'implémentation, sur un code de production DRAGON [21], du calcul parallèle des matrices des probabilités de collision et de l'algorithme itératif parallèl multigroupe. Cette implémentation, nous a permis d'évaluer les performances du calcul parallèle sur des tests tridimensionnels très réalistes. L'ensemble des cinqs premiers articles est caractérisé par l'utilisation de calculs itératifs qu'on a parallélisé et dont on a testé les performances.

Pour amplifier la portée de nos travaux, nous avons choisi d'appliquer le calcul itératif parallèle à la mécanique des fluides. Pour ce faire, nous avons collaboré avec le Professeur Azzeddine Soulaimani de l'École de Technologie Superieure de Montréal, qui a mis à notre disposition son code séquentiel d'élément finis FES (Finite Element System) pour la simulation d'un écoulement tridimensionnel avec surface libre. Pour la mise en oeuvre de la parallélisation du code FES, nous avons eu la chance de collaborer avec le Professeur Yousef Saad de l'Université de Minessota, une autorité dans le domaine des méthodes itératives parallèles. Le sixième article de la présente thèse provient de cette double collaboration. Notre choix d'appliquer le calcul itératif parallèle à un problème différent (mécanique des fluides) a donné une extension significative à nos travaux tout en les maintenant dans un cadre unificateur.

Les quelques lignes qui suivent, résument le sixième article où la méthode des éléments finis est appliquée à des équations tridimensionnelles de Navier-Stokes afin de simuler un écoulement d'un fluide incompressible en présence d'une surface libre. Le système algébrique non linéaire qui résulte de la discrétisation spatiale et temporelle est résolu par une méthode dite de Newton-GMRES. À chaque pas de temps et pour une itération de Newton, un système linéaire est résolu par l'algorithme GMRES. Pour paralléliser FES, le domaine spatial est décomposé en sous-domaines non chevauchés par le biais du programme METIS, l'algorithme GMRES préconditionné et

parallèle est utilisé pour résoudre le système linéaire. Le préconditionnement est de type block-Jacobi parallèle. La matrice de préconditionnement est formée localement en n'assemblant que les éléments faisant partie du sous-domaine, et une factorisation incomplète ILUT est appliqué localement à cette matrice. Les résultats obtenus sont satisfaisants et comparables à ceux obtenus par la version séquentielle du code. En plus, des gains en temps de calcul et en espace mémoire, ont été obtenus par la parallélisation du code. D'autres types de préconditionneurs peuvent être implémentés très facilement dans la structure actuelle de notre code parallèle.

Le développement croissant des architectures parallèles ouvre de nouvelles perspectives pour accélérer encore plus les calculs requis en physique des réacteurs. En ce qui a trait aux problèmes classiques de neutronique statique, trois types d'applications déterministes courantes à la modélisation des coeurs des réacteurs nucléaires devraient s'amplifier dans les prochaines années:

1. le ralentissement à plusieurs miliers de groupes d'énergie, utile pour le traitement fin des résonances;

2. la résolution de l'équation de transport multigroupe, que ce soit pour des problèmes de blindage ou pour la génération des propriétés condensées pour un calcul de réacteur;

3. la résolution de l'équation de diffusion multigroupe, qui sert généralement pour déterminer le flux de neutrons dans les études de suivi de coeur.

Pour le traitement de ces problèmes, la géométrie fondamentale étudiée est, comme on l'a vu dans les premier et deuxième articles, décomposée en I régions avec un spectre énérgétique qui contient G groupes. Pour paralléliser la résolution des grands systèmes doublement couplés ainsi obtenus, un premier schéma consiste naturellement à distribuer les groupes d'énergies sur les différents processeurs disponibles et à gérer les transfert énergétiques par échange de messages entre les processeurs. Cette approche devient plus performante à mesure que le nombre de groupes d'énergie augmente.

Lorsque le couplage spatial est faible, les matrices des problèmes à résoudre sont généralemet creuses. Les méthodes basée sur la décomposition de domaine, conviennent alors mieux pour paralléliser ce genre de problèmes. Les zones spatiales

sont regroupées sous forme de sous-domaines attribué chacun à un processeur. Certaines zones peuvent faire partie de plus d'un sous-domaine (processeur), entrainant ainsi un chevauchement des sous-domaines. À cause du faible couplage spatial, un processeur n'aura à communiquer qu'avec ses processeurs voisins.

Les problèmes courants de la neutronique sont certainement des candidats de choix pour les calculateurs les plus puissants, tant par le nombre important d'inconnues à traiter que par le nombre d'opérations en point flottant à réaliser. Le fait que ces problèmes soient parallélisables, ouvre une nouvelle perpective à la neutronique pour accélérer encore les calculs et traiter des problèmes envisagés avec des approximations plus générales. Le présent travail s'inscrit dans le cadre de cette mouvance.

BIBLIOGRAPHIE

[1] Marleau G and Hébert A (1985) *Solving the Multigroup Transport Equation using the Power Iteration Method* 11^{th} Annual Symposium on Simulation of Reactor Dynamics and plant Control, CMR Kingston, April 21-22

[2] Saad Y and Schultz M.H (1986) *GMRES: A Generalized Minimal Residual algorithm for solving nonsymetric linear systems* SIAM J. Sci. Statist. Comput. **5** pp 856–869

[3] Beguelin A, Dongarra J.J, Geist G.A, Manchek R and Sunderam V.S (1991) *A Users' Guide to PVM Parallel Virtual Machine* technical report **ORNL/TM-11826**, Oak Ridge National Laboratory, July

[4] Gropp W, Lusk E and Skjellum A (1994) *Using MPI: Portable Parallel Programming with the Message-Passing Interface* MIT Press, Cambridge, MA

[5] Sanchez R and McCormick N.J (1982) *A Review of Neutron Transport Approximation* Nucl. Sci. Eng. **80**, pp 481–535

[6] Bussac J and Reuss P (1978) *Traité de Neutronique* Hermann Press, Paris

[7] Hébert A (1983) *Neutronique* notes de cours, Institut de génie énergétique, École Polytechnique de Montréal

[8] Henkel C.S and Turinsky P.J (1992) *Solution of the Few-Group Diffusion Equation on Distributed Memory Multiprocessor* Top.mtg. on Advances in Math., Comp. and Reactor Physics, Pittsburgh, USA, Mars

[9] Wilson W, Vujic J and Gu A (1993) *Parallel Multiple-Assembly Calculations in GTRAN2/M* Transactions of the American Nuclear Society Winter Meeting, San Francisco, CA, November 14-18

[10] Slater S and Vujic J (1994) *P4 Parallelization of General Geometry Ray Tracing in Computational Physics* Proc. High Performance Computing, San Diego, April 10-15

[11] Marleau G and Hébert A (1986) *Interface Current Method for Cluster Geometry* Nucl. Sci. Eng. **92**, 240

[12] Roy R, Hébert A and Marleau G (1989) Nucl. Sci. Eng. **101**, 217

[13] Goehring T and Saad Y (1994) *Heuristic algorithms for automatic graph partitionnig* Technical Report UMSI 94-29, University of Minnesota Supercomputer Institute, Minneapolis, MN

[14] Soulaimani A and El Kadri N (1997) *FES: Une méthode d'éléments finis pour des problèmes d'Hydrodynamique avec transfert de chaleur et en présence d'une surface libre* Rapport Technique ÉTS, École de Technologie Superieure Département de génie mécanique, Montréal, Qc

[15] Saad Y (1996) *Iterative Methods for Sparse Linear Systems* , International Thomson Publishing Inc

[16] Saad Y (1994) *ILUT: a dual threshold incomplete ILU factorization*

Numerical Linear Algebra with Applications, 1:387-402

[17] Saad Y and Malevsky A (1995) *PSPARSLIB: A portable library of distributed memory sparse iterative solvers* in Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, st. Petersburg, sept

[18] Balay S, Curfman McInnes L, Gropp W.D and Smith B.F (1995) *PETSc 2.0 Users Manuel* Technical Report ANL-95/11, Argonne National Lab, Argonne, IL

[19] Karypis G and Kumar V (1995) *A fast and high quality multilevel scheme for partitioning irregular graphs* Technical Report TR 95-035, Departement of Computer Science, University of Minnesota,MN

[20] Dutto L.C, Habashi W.G and Fortin M (1994) *Parallelizable block diagonal preconditioners for the compressible Navier-Stokes equations* Comput. Methods Appl. Mech. Engrg. 117 15-47.

[21] Marleau G, Hébert A and Roy R (1985) *DRAGON: A Collision Probability Transport Code for Cell and Multicell Calculations* Report IGE-100, École Polytechnique de Montréal.