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. 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

Assuming a fixed right-hand side, solution of the left-hand side can be achieved either by an iterative or 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 :

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

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:

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

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 $\mathrm{k}_B^{(n)}$ of n^{th} order approximation is known inside the block B and is evaluated using the following expression:

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

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 $k^{(n)} \rightarrow K_{\text{eff}}$ 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. If 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:

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

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

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.

II.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 of the flux, in 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 variational method.[6]

The principle of 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 lerge 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 $\mathbf{W}^{\mathbf{g}}$ 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 broadcasted 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 compute the sum K_{eff}, and upadte the fission source related to its energy subgroup which will be used in the next inner iterations.

The iterative process are 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 of cell calculation 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,...)
```

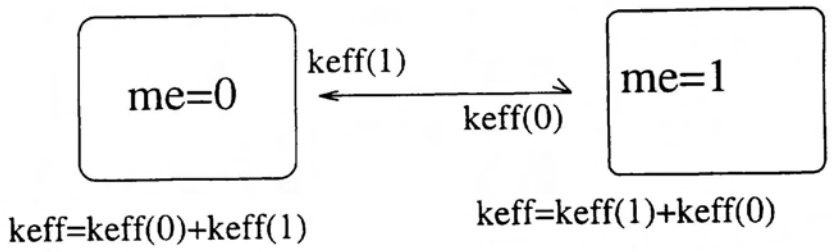

Figure 1: intercommunication of local critical multiplication factor

```
      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 Montreal, limited to 4 processors;
- Sparc 2000 at École Polytechnique de Montreal,limited to 4 processors;
- an 8 processors Sparc 1000 at Centre de Recherches Mathematiques of Montreal University.

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 annulii inside. There are three different material regions corresponding to the fuel, its sheat 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 remark (see Figure 2-5) 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 the ones of IBM SP-2, the figures show that the best speedup is achieved by Sparc 1000 (or 2000). This is explained by the fact that this machine 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 amoung 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 Montreal, who give 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

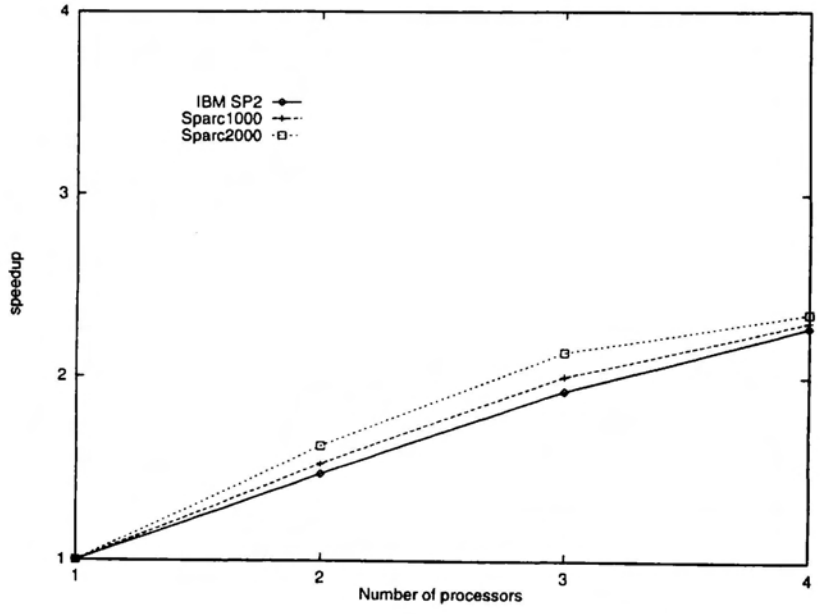

Figure 2: Speedup curves for unaccelerated multigroup parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

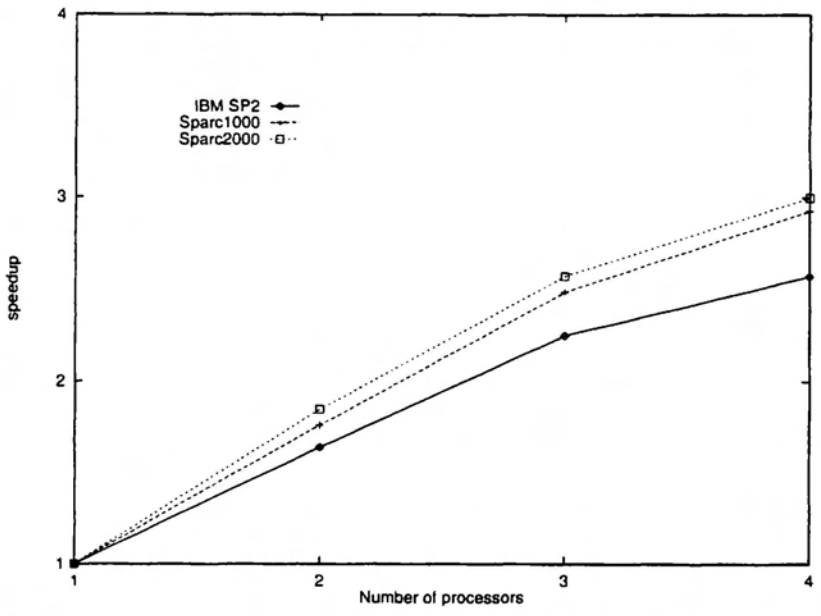

Figure 3: Speedup curves for accelerated multigroup parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

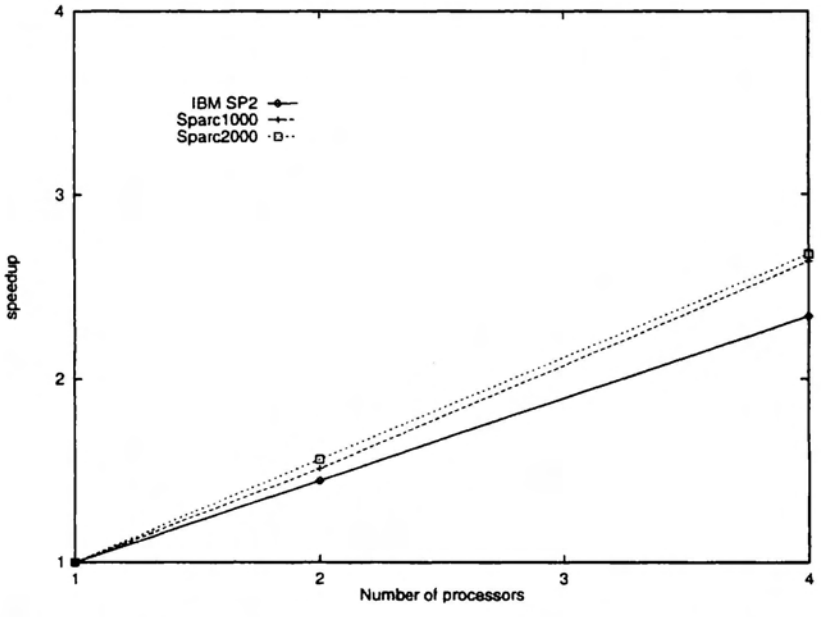

Figure 4: Speedup curves for unaccelerated multiregion parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.

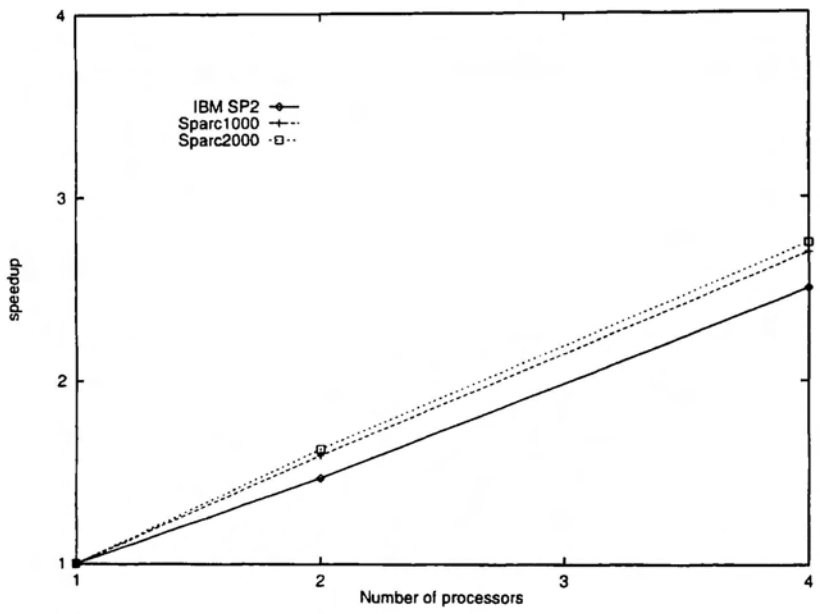

Figure 5: Speedup curves for accelerated multiregion parallel scheme. Performance of various types of processors are compared using 69 condensed groups and 32 regions.