Progress in Nuclear Energy, Vol. 26, No. 3, pp. 147-162, 1991.
Printed in Great Britain. All rights reserved.

0149-1970/91 $0·00 + ·50
© 1992 Pergamon Press Ltd

VECTORIZATION AND PARALLELIZATION OF A PRODUCTION REACTOR ASSEMBLY CODE

JASMINA L. VUJIC* and WILLIAM R. MARTIN

Laboratory for Scientific Computation and the Department of Nuclear Engineering, The University of Michigan, Ann Arbor, MI 48109, U.S.A.

Abstract - In order to efficiently use new features of supercomputers, production codes, usually written 10 - 20 years ago, must be tailored for modern computer architectures. We have chosen to optimize the CPM-2 code, a production reactor assembly code based on the collision probability transport method. Substantional speedups in the execution times were obtained with the parallel/vector version of the CPM-2 code. In addition, we have developed a new transfer probability method, which removes some of the modelling limitations of the collision probability method encoded in the CPM-2 code, and can fully utilize parallel/vector architecture of a multiprocessor IBM 3090.

1. INTRODUCTION

Most of the software which the nuclear industry uses routinely is written 10 to 20 years ago. It was developed for serial mainframe computers with much less power and memory. With the present generation of supercomputers it is possible to achieve substantial improvements in performance by using large real and virtual memory, and vector/parallel processing. In order to efficiently use what is available, computer software must be tailored for modern computer architectures. There are two ways to achieve this goal. One can start from the beginning and develop new methods and numerical algorithms that can fully utilize advanced computer architectures. This way is very attractive for method and software developers, but can be very expensive. No one in nuclear industry is willing to forget about the existing software, with many man-years incorporated into it, and finance the development of the new software whenever there is some breakthrough in computer industry. The old software has been simply transferred to each new generation of computers, with minimum changes needed to make it useable. In the most cases no effort has been made to optimize the old codes for the new architectures. Consequently, modern supercomputers have not been used efficiently, and improvements in performance over the previous computer generations are barely noticeable. We have taken somewhat different approach in order to accomplish the following two goals:

• Transfer a production code to a particular supercomputer, and make all necessary changes in the existing numerical algorithms to optimize them for a given advanced architecture. This also includes usage of all available tools for efficient vector and parallel processing.

• Extend the existing theoretical methods by removing the modelling limitations which were mostly consequences of the limited amount of memory and low speed of the previous generation of computers. Develop accurate and cost effective parallel/vector algorithms by taking advantage of the large virtual memory in addition to vector and parallel processing on a multiprocessor IBM 3090.

Transport theory codes are typically applied to assembly-level calculations but in principle could be used in place of the multigroup diffusion (MGD) codes for global reactor calculations. With the Boltzmann transport equation as their theoretical basis, these codes are very accurate, but time-consuming. As a consequence, MGD codes are still the main methodology for reactor design and analysis, even in applications where diffusion theory has little mathematical or physical basis. However, advances in nuclear engineering often demand the use of more accurate physical models and it is becoming increasingly important to have accurate and efficient computational algorithms

* Present Address: Argonne National Laboratory, Engineering Physics Division, Argonne, IL 60439, U.S.A.

for solving global transport problems, such as in the design of advanced reactor cores. A specific example is the CPM-2 code (CPM-2, 1987), a multigroup collision probability code based on the integral transport equation, that solves the 2-D neutron transport equation for light water (BWR and PWR) reactor assemblies. Due to exorbitant demands on central processor unit (CPU) time, this very accurate code is mostly used for benchmarking only.

In the first part of this paper we present an efficient parallel-vector transport algorithm based on the collision probability (CP) transport method employed in the CPM-2 code, which can fully utilize advanced IBM 3090 architecture. An order of magnitude reduction in CPU time and a factor of 20 reduction in wallclock time are obtained with this modified vector-parallel version of CPM-2.

Our new transfer probability method is presented in the second part of this paper. Two major limitations, infinite lattice geometry and isotropic scattering, which inhibited the use of CPM-2 for global reactor calculations, were removed. The method is based on the integral transport equation and use of collision, escape and transmission probabilities, together with linearly anisotropic scattering, for assembly level calculations. In order to compare the efficiency of these two methods in utilizing the advanced IBM 3090 architecture, we will only present results for isotropic scattering.

2. VECTORIZATON AND PARALLELIZATION OF THE CPM-2 CODE

The neutron transport equation is a linearized form of the more general Boltzmann equation from gas dynamics. Knowing that the analytical solution of this equation can be found only for several idealized problems, many methods have been developed to find approximate numerical solutions. These methods are usually divided into two broad groups: stochastic (Monte Carlo) and deterministic. In Monte Carlo methods, the history of each particle is followed (i.e., simulated on a computer). In the case of the deterministic methods, approximate equations are derived from the transport equation, and the resulting system of algebraic equations is solved on a computer. Depending on the form of the neutron transport equation chosen as a starting point, the deterministic methods can be divided into three large groups (Sanchez, 1982):

- those based on the integro-differential form,
- those based on the surface-integral form, and
- those based on the integral form of the neutron transport equation.

The major idea behind the integral transport methods is to integrate out the angular dependence and to solve the transport equation for the scalar neutron flux directly. The method is very accurate and relatively simple to apply, if isotropic scattering can be assumed.

The integral transport equation is based on a global neutron balance in a given direction, leading to a strong coupling of all regions, which is opposite of the case with the integro-differential transport equation which is based on a local neutron balance, leading to a coupling between the neighboring regions in space only. Although the treatment of the spatial variables in 2-D or 3-D in the integral transport equation leads to the discretized matrices that are full (dense), integral transport methods can treat very complicated geometries, usually treated by Monte Carlo methods only. The strong spatial coupling and large, dense matrices, which put strong demands on computer memory and CPU time in the early years of applications of these methods, are not prohibitive any more for today's very fast supercomputers with large virtual memory.

Derivation of the integral transport equations for the volumetric flux and outgoing current for two-dimensional geometries can be found elsewhere (Lewis, 1984). To obtain the discretized integral transport equations, the area A and boundary L are partitioned into NR subareas and NB subboundaries, respectively, such that

$$A = \sum_{i=1}^{NR} A_i, \qquad L = \sum_{\alpha=1}^{NB} L_\alpha.$$

giving, in the multigroup form

$$A_i \Sigma_{t,g,i} \Phi_{g,i} = \sum_{i'} A_{i'} Q_{g,i'} P_g(A_i \leftarrow A_{i'}) + \sum_{\alpha'} L_{\alpha'} J^{in}_{g,\alpha'} P_g(A_i \leftarrow L_{\alpha'}). \tag{1}$$

where

$P_g(A_i \leftarrow A_{i'})$- is the volume-to-volume collision probability,
$P_g(A_i \leftarrow L_{\alpha'})$- is the surface to volume collision probability,
$P_g(L_\alpha \leftarrow A_{i'})$- is the escape probability,
$P_g(L_\alpha \leftarrow L_{\alpha'})$- is the transmission probability,

and the spatially averaged quantities are defined as follows

$$\Phi_{g,i} = \frac{1}{A_i} \int_{A_i} dA\, \Phi_g(\vec{r}),$$

$$Q_{g,i} = \frac{1}{A_i} \int_{A_i} dA' \int_{4\pi} d\vec{\Omega}\, Q_g(\vec{r}',\vec{\Omega}), \qquad \Sigma_{t,g,i} = \frac{1}{A_i \Phi_{g,i}} \int_{A_i} dA\, \Sigma_{t,g}(\vec{r})\, \Phi_g(\vec{r}),$$

$$J^{in}_{g,\alpha'} = \frac{1}{L_{\alpha'}} \int_{L_{\alpha'}} dL' \int_{\vec{n}'\bullet\vec{\Omega}<0} d\vec{\Omega}\, J^{in}_g(\vec{r}_{L'},\vec{\Omega}), \qquad J^{out}_{g,\alpha} = \frac{1}{L_\alpha} \int_{L_\alpha} dL \int_{\vec{n}\bullet\vec{\Omega}>0} d\vec{\Omega}\, J^{out}_g(\vec{r}_L,\vec{\Omega}).$$

The total neutron source is given as

$$Q_g(\vec{r},\vec{\Omega}) = \sum_{g'} S_{g'\to g}(\vec{r},\vec{\Omega}) + \frac{1}{k}\chi_g \sum_{g'=1}^{G} F_{g'}(\vec{r},\vec{\Omega}),$$

and k is the effective multiplication factor. The general isotropic fission neutron source is given as

$$F_g(\vec{r},\vec{\Omega}) = \frac{1}{4\pi}\, \nu\Sigma_{f,g'}(\vec{r})\, \Phi_{g'}(\vec{r}),$$

and the scattering neutron source as

$$S_{g'\to g}(\vec{r},\vec{\Omega}) = \int_{4\pi} d\vec{\Omega}'\, \Sigma_{s,g'\to g}(\vec{r},\mu_0)\, \Phi_g(\vec{r},\vec{\Omega}'), \qquad \mu_0 = \vec{\Omega} \bullet \vec{\Omega}'.$$

The most time-consuming part of the entire collision/transfer probability formalism is the evaluation of the coefficients of the collision, escape, and transmission probability matrices. The calculation time increases rapidly with the increase in the number of energy and space meshes. The usefulness of the collision/transfer probability method is closely related to the efficiency with which the collision, escape and transmission probability matrices are numerically calculated.

2.1. The simplified collision probability method

The simplified collision probability method employed in CPM-2 is based on the integral form of the Boltzmann transport equation assuming infinite lattice geometry, isotropic scattering, and steady state. In this case, the discretized set of equations in 2-D (Eqs. (1) and (2)) collapses into one equation:

$$A_i\, \Sigma_{t,g,i}\, \Phi_{g,i} = \sum_{i'} A_{i'}\, P^*_g(A_i \leftarrow A_{i'})\, Q_{g,i'}, \tag{3}$$

where $P^*_g(A_i \leftarrow A_{i'})$ is the modified collision probability matrix. The reflective boundary conditions must be built into the collision probabilities, by reflecting the neutron path several times on the boundaries (Fig. 1) until the neutrons have travelled up to 10 mean free paths.

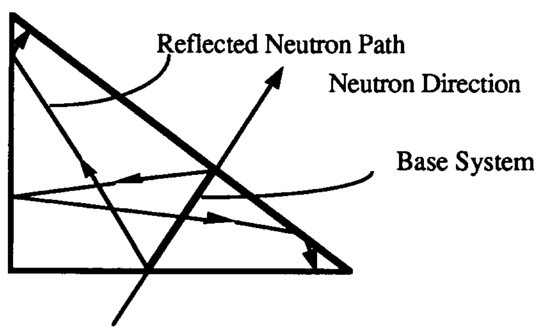


Fig. 1 Neutron path reflections on boundaries

In order to calculate the CP matrices numerically, the region of interest is covered by the set of equidistant parallel lines (Fig. 2) at a number of equally spaced angles. The track intercepts and the corresponding region numbers are calculated for each line in the base system, as well as for the reflected lines. Using the track intercept data, the optical lengths are calculated and contributions to the CP matrix are added from each array of intercepts and for each energy group. The accuracy of the collision probabilities can be increased by increasing the number of integration directions, decreasing the distance between parallel integration lines (thus increasing their number), increasing

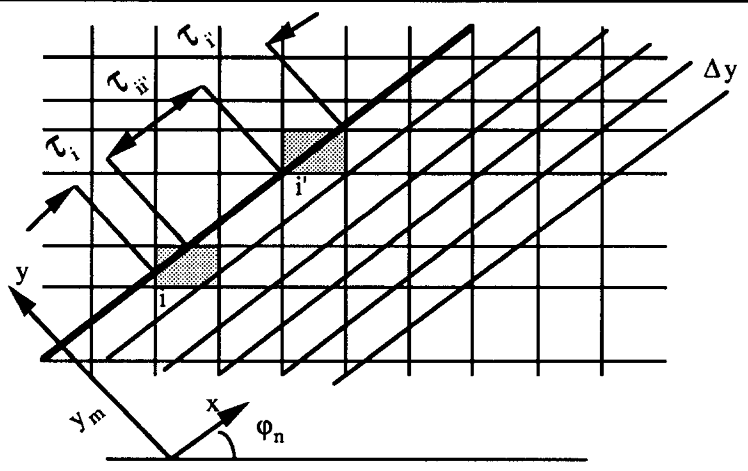


Fig. 2 Calculation of collision probabilities

the optical distance the neutrons are followed after reflections, and increasing the number of meshes per pin cell. Requests for higher accuracy in the CP matrix calculation, result in a rapid increase in time per burnup step (Table 1).

TABLE 1 Timinig results for one burnup step and different levels of accuracy in the CP matrix calculation

	Default		Medium		High	
	CPU sec	[%]	CPU sec	[%]	CPU sec	[%]
CPXY	17.58	41.35	497.00	87.37	2123.25	97.60
FLUXY	13.66	32.13	61.84	10.87	43.58	2.00
CPM-2	42.51	100.00	568.83	100.00	2175.37	100.00
CPXY+ FLUXY		73.48		98.24		99.60

2.2. *Basic concepts of the IBM 3090 VF architecture*

As with any other sophisticated tool, and we can say with certainty that computers have become indispensable "tools" in science and engineering, one must know enough about the "tool" to achieve high performance. Each computing system is different and requires different "fine tuning" for high performance. Today's compilers are not sophisticated enough to perform all kinds of optimizations and "fine tuning" automatically, so that the algorithm developer's and programmer's hand will be inevitable for a long period in the future. Without going into a detailed explanation of the IBM 3090-600E VF architecture, we will present its major features and focus our attention on those parts where programmer knowledge and skills can have a large influence on code performance: vector facility, cache, and virtual memory.

The IBM 3090 was announced in 1985 as the most powerful machine in the IBM production line. It introduced two important extensions to the IBM System/370-XA architecture: the vector facility and the expanded storage (Tucker, 1986). The cycle time on the IBM 3090 base machine was 18.5 ns, giving the peak megaflop rate of 108 MFLOPS per processor. The maximum configuration allows up to 6 processors with 6 vector facilities. In our research we have been using the following two IBM 3090 models: Model 600E and Model 600J, with some basic data given in Table 2.

TABLE 2 Basic data for the IBM 3090 Model 600E and Model 600J

	Model 600E	Model 600J
Peak rate	696 MFLOPS	827.6 MFLOPS
Cycle time	17.2 ns	14.5 ns
Real memory	256 Mbytes	512 Mbytes
High speed cache	64 Kbytes per processor	256 Kbytes per processor
Vector length	128	256

The vector facility is a fully integrated addition to the IBM 3090 central processor, that can overlap the processing of multiple array elements resulting in faster execution than purely scalar processing (Tucker, 1986). It consists of 8 double word (64 bit) vector registers, or equivalently, 16 single word (32 bit) vector registers. Each vector register can contain up to 256 elements. When full, a vector register can provide two elements to be read out and one element to be written in, every cycle.

The Extended Architecture (XA) with 31-bit addressing allows up to 2 Gbytes of virtual memory addressing. The IBM 3090 memory hierarchy consists of 5 memory levels: peripheral storage (disks, tapes), expanded storage, main memory or central storage, high speed buffer or cache, sixteen 32-bit or eight 64-bit vector registers. Going from the peripheral storage level to the vector register level, the data access time becomes faster and faster, so that the transfer of data between the high speed buffer (cache), registers and execution unit proceed at a rate of one per cycle.

A high speed buffer or cache is fast, but expensive memory placed between the fast CPU registers and the large slow main memory. When data are requested by CPU, the hardware will check the cache first. If the data are there, they will be delivered to the CPU at a rate of 1 double word per cycle. If data are not in the cache, the hardware requests them to be transferred from the main memory to the cache. When data are loaded into the cache, the least recently used data are displaced (copied back to the main memory - only if changed). It is desirable to fill the cache with contiguous data. Non-contiguous data will effectively have access to only a subset of cache locations, if the data are computing for the same cache locations. Although the retention of data in cache is not directly controlled by user, it can be managed by proper coding of FORTRAN programs.

2.3. Local and global restructuring of the CPM-2 code

The CPM-2 code has been written in a good programmer's practice of the 70's, for optimal scalar performance. It is highly modularized, computations are spread over many small subroutines, and modules communicate through disk inputs/outputs (I/Os) due to small memory available at the time CPM-2 was written. The initial timing performances were measured for the two test cases: BWR assembly (8x8 pin cells) and PWR assembly (17x17 pin cells), in vector and scalar mode. Three levels in accuracy of the CP matrix calculation (default, medium, high) were used. The execution-time profile for the BWR case, for 27 burnup steps, with the default accuracy in the CP matrix calculation, is given in Fig. 3.

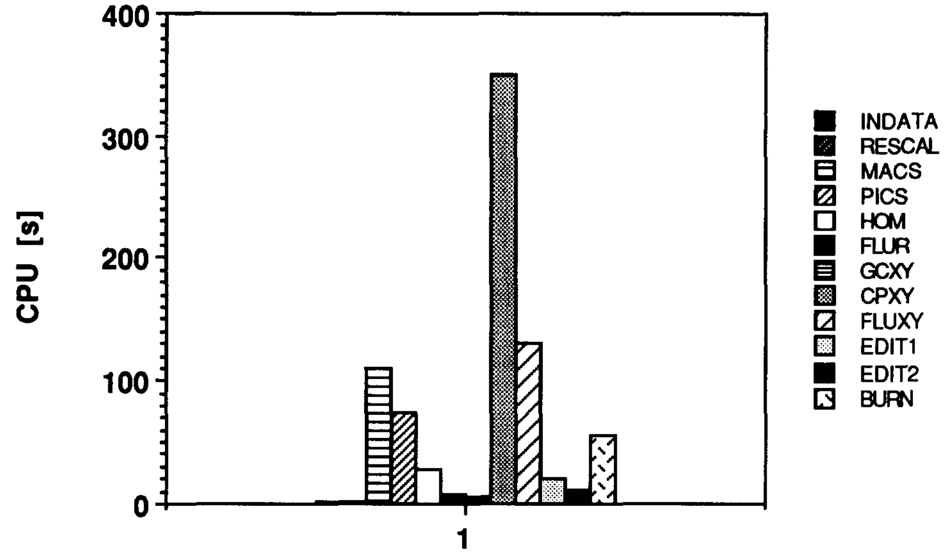


Fig. 3 The execution-time profile for the CPM-2 modules

As can be seen from Fig. 3, the principal computational tasks in CPM-2 are the calculation of the CP matrix (CPXY module), and the solution of the resulting discretized eigenvalue equations (FLUXY module). The CPXY and FLUXY modules were then modified for vector/parallel performance, including conversion to FORTRAN-77 and local restructuring to remove inhibitors to vectorization. However, the gains in performance were modest, with approximately 50% increases in each segment. This experience demonstrates that in many cases, an old code cannot be efficiently optimized for new vector/parallel architectures with only local changes to the algorithm. In order to take full advantage of the vector/parallel architecture global restructuring is necessary, and this is the subject of the next section.

Several features of the original CPM-2 code impaired its performance on a vector processor: high modularity of the code; many small subroutines to handle parts of the calculations; intermediate results kept in scalar variables, thus requiring all intermediate result to be immediately written into files or tapes, and then read in when they were needed by a different module; modules communicate through I/O statements; poor memory reference pattern, where the substantional amount of indirect addressing and stride greater than 1 prevented efficient use of vector registers and cache.

Efficient global restructuring requires a firm understanding of the theory behind the code and the numerical methods used. Examples of global restructuring techniques (Soll, 1986) include: data rearrangement, subroutine merging, creating vector instead of scalar variables, increasing vector lengths, taking advantage of virtual memory, changing COMMONs and modifying the numerical algorithm. To illustrate the work on global restructuring methods, the techniques which made a major impact on performance of CPM-2 are summarized below:

Subroutine Merging. Many small subroutines were eliminated, and substituted by more efficient in-line coding.

Vector Data Structuring. The general concept in creating a data structure for vectorization is to establish a data ordering that is consistent with the addressing order of the primary computations. This requires the scalar variables in which the intermediate computational results are stored to be expanded to vector variables where all results of particular calculations can be stored. The amount of storage required by the program increases due to the data restructuring and scalar expansions. Use of 31-bit addressing in VM/XA extends the virtual address space from 16 Mbytes to 2 Gbytes, therefore satisfying even the most extreme storage requirements. The track intercept data were originally kept in scalar or one-dimensional vector variables, so that the data from one array of intercepts had to be immediately written into a logical unit, which totaled approximately 30,000 WRITEs per calculation (per each energy group and time step). In the subroutine CPC, where the CP matrix is calculated, these data had to be read in (30,000 READs) for each energy. To remedy this, two- or three-dimensional arrays were created for these variables, so that all track intercept data from all arrays of intercepts could be stored. All WRITEs and READs are thus eliminated, data handling became more efficient, and the basis for parallelization of the CP matrix calculation was created.

Virtual Memory Utilization. To save storage in the original CPM-2 code, a one-dimensional array was used to store only the lower triangular part of the CP matrix, thus requiring indirect addressing. Since there are large number of energy groups and spatial meshes for two-dimensional applications, the CP array was partitioned into several arrays, which made the solution of the eigenvalue equations extremely complicated, and even gave inconsistent results.

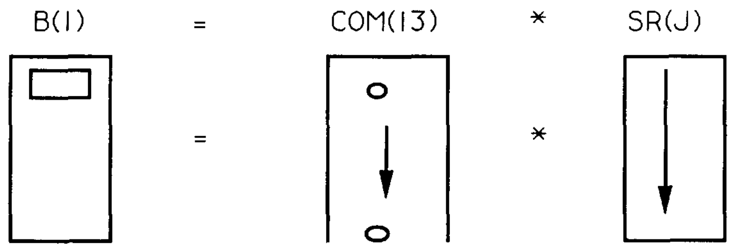
In our new algorithm, we again take advantage of the large virtual memory on the IBM 3090 and use large arrays to store and transfer data among the code segments, removing the major bottleneck of reading/writing to main memory. All collision probabilities are kept in one large three-dimensional array which simplified the data manipulation considerably, and allowed use of the same numerical algorithm regardless of the number of meshes. Due to the reciprocity relation among the collision probabilities, the CP matrix is symmetric in its two leading dimensions, allowing it to be accessed always column-wise in matrix-matrix or matrix-vector operations. Figure 4 shows how complicated the matrix-vector multiplication had to be in the original coding, due to a fact that only the lower triangular part of the CP matrix was stored in a one-dimensional array. Different, even more complicated coding was used when the CP matrix had more that 32,131 elements. Figure 4 also shows the simple and efficient new coding, where the entire CP matrix is stored in the one three-dimensional array. This results in the total eliminating of indirect addressing.

Modified Numerical Method. For segment FLUXY, the subroutine SOLV1, which is responsible for the inner iterations in the solution scheme for the coupled systems of linear equations, was the most time-consuming. In the original (scalar) version of the SOR method in CPM-2, the new values of Φ_i (Eq. 4) are used as soon as they are calculated, by storing the new and old values in the same vector,

$$\Phi_i^{(m+1)} = \Phi_i^{(m)} + \omega(b_i - \sum_{i'=1}^{i-1} a_{ii'}\Phi_{i'}^{(m+1)} - \sum_{i'=i}^{NRG} a_{ii'}\Phi_{i'}^{(m)}) / a_{ii}. \quad (4)$$

This is a clever way of coding for serial machines, but it introduces recurrences on vector machines. Due to the lack of memory, the elements of the matrix A were not stored in the original CPM-2 code, but recalculated for each iteration. In the new coding, the elements of A are precalculated and stored. In the matrix-vector multiplications, the elements of A are fetched by

The CP Matrix is Stored in 1-D Array



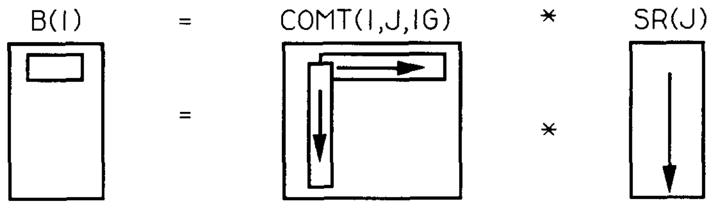
The CP Matrix has less than 32131 elements

```
   COMMON /ASE005/ COM (32131)
   . . .
   NELEM = NEL (1)
   READ (LU5) COM
   DO 20 I = 1, NRG
   SUM = 0.
     DO 10 J = 1, NRG
     I1 = MAX0 (I,J)
     I2 = MIN0 (I,J)
     I3 = IADR (I1) + I2
     SUM = SUM + COM (I3) * SR (J)
10   CONTINUE
   B (I) = SUM
20 CONTINUE
```

The CP matrix has more than 32131 elements

```
   COMMON /ASE005/ COM (32131)
   . . .
   NKLAR = 0
   REWIND LU4
   DO 10 I = 1, NRG
   B (I) = 0.
10 CONTINUE
   DO 40 N = 1, NREC
   MINRAD = MINROW(N)
   MAXRAD = MAXROW (N)
   NELEM = NEL (N)
   READ (LU5) COM
   WRITE (LU4) COM
     DO 30 I = MINRAD, MAXRAD
       DO 20 J = 1, I
       K = IADR (I) + J - NKLAR
       B (I) = B (I) + COM (K) * SR (J)
       IF (I.EQ.J) GO TO 20
       B (J) = B (J) + COM (K) * SR (J)
20      CONTINUE
30   CONTINUE
   NKLAR = NKLAR + NELEM
40 CONTINUE
```

The CP Matrix is Stored in 3-D Array



```
   COMMON /C1/ COMT (500,500,12)
   . . .
   DO 20 I = 1, NRG
   SUM = 0.
     DO 10 J = 1, NRG
     SUM = SUM + COMT (J,I,IG) * SR (J)
10 CONTINUE
   B (I) = SUM
20 CONTINUE
```

Fig. 4 Matrix-vector multiplication for original and new storing of the collision probability matrix

column, with stride 1, and the inner DO loops are vectorized, producing a decrease in the CPU time.

Parallelization. The new programming features of Parallel FORTRAN (IBM, 1988) available on the IBM 3090 were used to perform the CP matrix calculations for different energy groups on up to six processors simultaneously, reducing the wallclock time, or turnaround time for CPM-2. We have used the PARALLEL TASKS construct, which is a subroutine-level parallelism. In this case, parallelism is coarse grained, i.e. each processor has a large amount of work to do. Creating the three-dimensional array for the CP matrix, where the third dimension was energy group, not only removed a need for many I/Os in CPC and the FLUXY segment and simplified data access, but also allowed parts of this matrix for a given energy group to be PRIVATE for each parallel task (Fig. 5). Although the parallel tasks are permitted to have parallel I/Os (IBM, 1988), they have to wait in line, because there is only one "real" processor doing I/O. The best performance, in our experience, is obtained if there are no I/Os in parallel tasks.

Because of the 16 Mbyte memory limit for the root code, all static arrays and scalar variables, as well as multiple copies of parallel tasks, all shared and private variables which are used in the parallel tasks are allocated at run time through dynamic COMMON. The calculation of the CP matrix for one energy group was assigned to one parallel task. By careful planning in restructuring of the algorithm, all parallel tasks are doing exactly the same amount of work, so that load balancing is perfect. In spite of these numerous changes (vectorization and parallelization), it was always assured that the results (effective multiplication factor, flux and power distributions, group constants) were EXACTLY the same for the original CPM-2 code and the parallel-vector (modified) code.

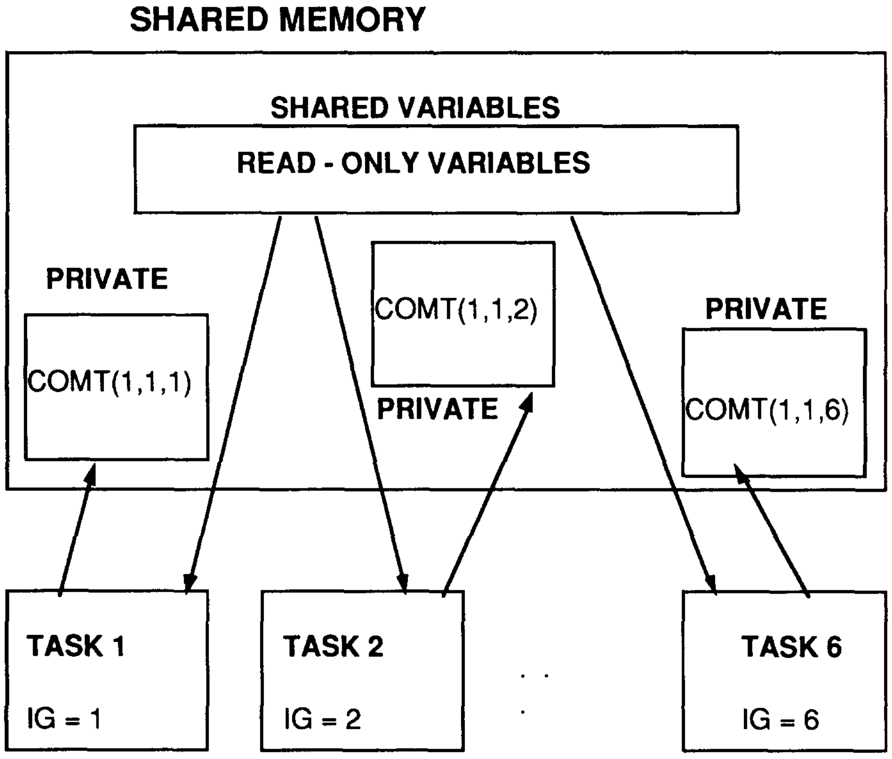


Fig. 5 Private vs shared memory in collision probability matrix calculation on several processors

3. A NEW PARALLEL-VECTOR TRANSFER PROBABILITY METHOD

The major limitation in the CPM-2 model is the assumption of infinite lattice geometry. With this approximation, no "real" boundary conditions can be treated, and no more than one assembly can be handled at a time. Reflective boundary conditions must be built into the collision probabilities (Eq. 3), since the volume integral is over the entire lattice, while summation is taken only over part of the assembly. The CP matrix is large and dense, which must be stored during the calculation, requiring a large memory. This created tremendous problems for the previous generation of computing systems, and therefore limited the use of the "exact collision probability formalism" to infinite lattice geometries (Lewis, 1977). In addition, the original CPM-2 code was very time-consuming and expensive to use.

Based on the collision/transfer probability formalism presented at the beginning of this paper we have developed a new parallel/vector transport algorithm which is optimized for the multiprocessor IBM 3090 (Vujic, 1989). The following equations have been derived from Eqs. (1) and (2):

$$\underline{\Phi}_g = \underline{\underline{CP}}_g \, \underline{AQ}_g + 4 \, \underline{\underline{SP}}_g \, \underline{J}_g^{in}, \tag{5}$$

$$\underline{LJ}_g^{out} = \underline{\underline{EP}}_g \, \underline{AQ}_g + 4 \, \underline{\underline{TP}}_g \, \underline{J}_g^{in}, \tag{6}$$

where the elements of $\underline{\underline{CP}}_g$, $\underline{\underline{SP}}_g$, $\underline{\underline{EP}}_g$, and $\underline{\underline{TP}}_g$, are the volume-to-volume collision probabilities, the surface-to-volume collision probabilities, the escape probabilities and the transmission probabilities, respectively, given as

$$(CP_g)_{i\leftarrow i'} = \frac{I_{g,i\leftarrow i'}}{A_i \Sigma_{t,g,i} \, A_{i'} \Sigma_{t,g,i'}}, \qquad (SP_g)_{i\leftarrow \alpha'} = \frac{I_{g,i\leftarrow \alpha'}}{A_i \Sigma_{t,g,i}},$$

$$(EP_g)_{\alpha'\leftarrow i} = \frac{I_{g,\alpha'\leftarrow i}}{A_i \Sigma_{t,g,i}}, \qquad (TP_g)_{\alpha\leftarrow \alpha'} = I_{g,\alpha\leftarrow \alpha'}.$$

The collision, escape, and transmission probabilities all contain similar "kernels", given as:

$$I_{g,m\leftarrow n} = \int \frac{d\varphi}{2\pi} \int dy \; \zeta(Z_m) \, \zeta(Z_n) \, K_{mn}(\tau_g), \tag{7}$$

where

$$K_{mn}(\tau_g) = \begin{cases} Ki_3(\tau_{g,ii'}) - Ki_3(\tau_{g,ii'} + \tau_{g,i}) - Ki_3(\tau_{g,ii'} + \tau_{g,i'}) + & \\ + Ki_3(\tau_{g,ii'} + \tau_{g,i} + \tau_{g,i'}), & \textit{for } m = i \textit{ and } n = i', \\ Ki_3(\tau_{g,\alpha i'}) - Ki_3(\tau_{g,\alpha i'} + \tau_{g,i}), & \textit{for } m = \alpha \textit{ and } n = i', \\ Ki_3(\tau_{g,\alpha\alpha'}), & \textit{for } m = \alpha \textit{ and } n = \alpha', \end{cases}$$

$$\zeta(Z_p) = \begin{cases} \zeta(A_i), & p = i, \\ \zeta(L_\alpha), & p = \alpha. \end{cases}$$

and where the Bickley-Nayler functions of order n, $Ki_n(\tau)$, are defined elsewhere (Bickley, 1935). It was possible to create an algorithm where contributions to all "kernels" from one array of intercepts are calculated at the same time (Fig. 6). In order to reduce the number of the collision, escape and transmission probabilities which relate different flux and current moments, several reciprocity and conservation relations were derived.

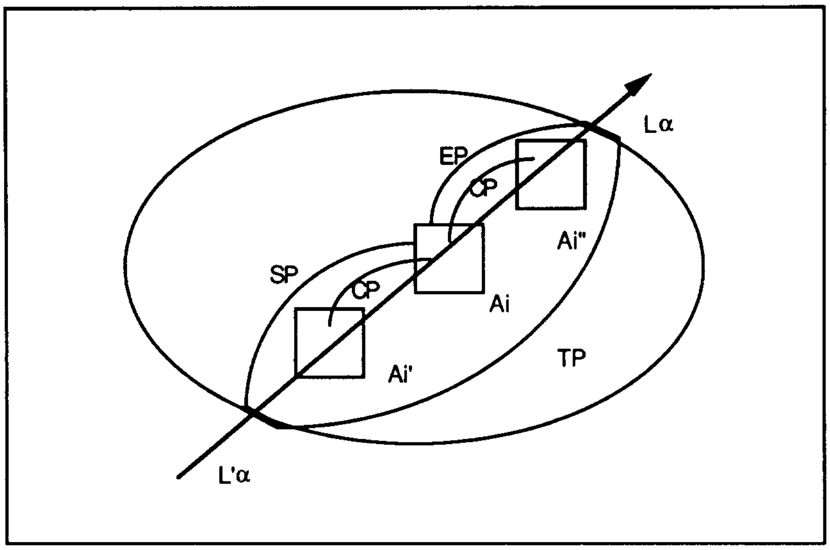


Fig. 6 Calculation of contributions to all transfer probability matrices from one array of intercepts

The transfer probabilities are independent for different energy groups, so that their calculation can proceed in parallel on several processors (Fig. 7). Because the "real" boundary conditions are used, the track intercepts need to be calculated for the base system only (i.e. there is no need any more to have numerous reflections of the neutron path in the boundaries). This considerably reduces the time spent to calculate the transfer probability matrices. The multigroup eigenvalue system of equations is solved using an iterative scheme (Fig. 8). For the outer iterations, the power method has been used. In the case of inner iterations, the vectorized SOR method (explained in the first part of this paper) has been used for the flux equations (Eq. (5)), and the vectorized ESSL routines DGEMV, DGEF and DGES (IBM, 1989) for the current equations (Eq. (6)). No acceleration technique has been used.

4. COMPUTATIONAL RESULTS

Table 3 summarizes the timing results measured on the IBM 3090-600J VF multiprocessor system in non-dedicated mode (VS FORTRAN V2.3 compiler and Parallel FORTRAN compiler have been used) for:

- Original collision probability algorithm - the original CPM-2 code in scalar mode,
- Modified collision probability algorithm - the vectorized and parallelized original CPM-2 algorithm in scalar mode, and in vector/parallel mode on one, three, and six virtual processors ,
- New transfer probability algorithm - the new vector/parallel algorithm (for isotropic scattering) in scalar mode, and in vector/parallel mode on one, three, and six virtual processors.

The number of processors we are requesting for our job (virtual processors) is usually less or equal to the number of real processors that will be, at run time, assigned to our job by the operating system, depending on how the system is busy at that time. Our results were obtained for a standard PWR assembly (17x17 pin cells) with high accuracy in the transfer matrix calculation and for one burnup step. Two-dimensional assembly calculations are performed with a six energy-group structure, with 3x3 micro-meshes per pin cell. The results with the new transport algorithm are in a good agreement with the original results (for example, the difference in effective multiplication factor is less than 0.1%).

It can be seen that the modified CPM-2 code is an order of magnitude faster than the original code, with overall reduction in wallclock time up to a factor of 22. The largest increase in performance was obtained with the FLUXY module, where the factor was 180 (wallclock time)

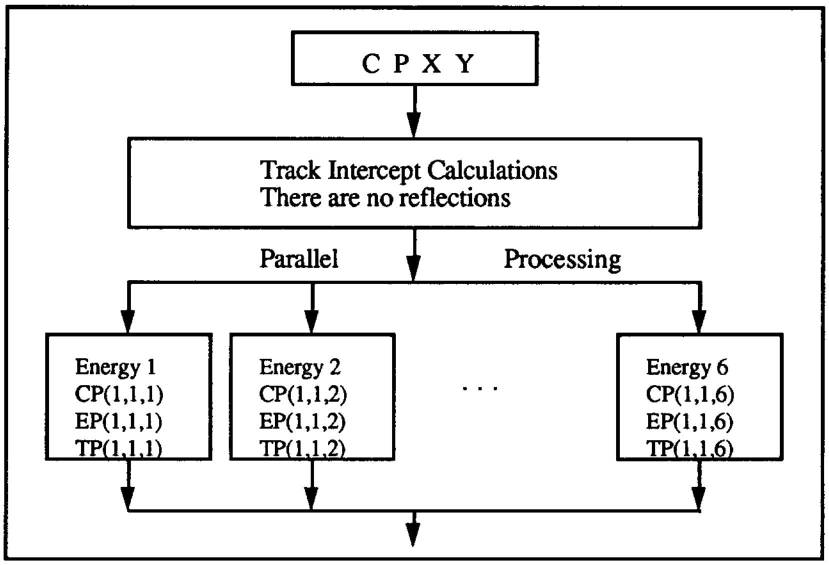
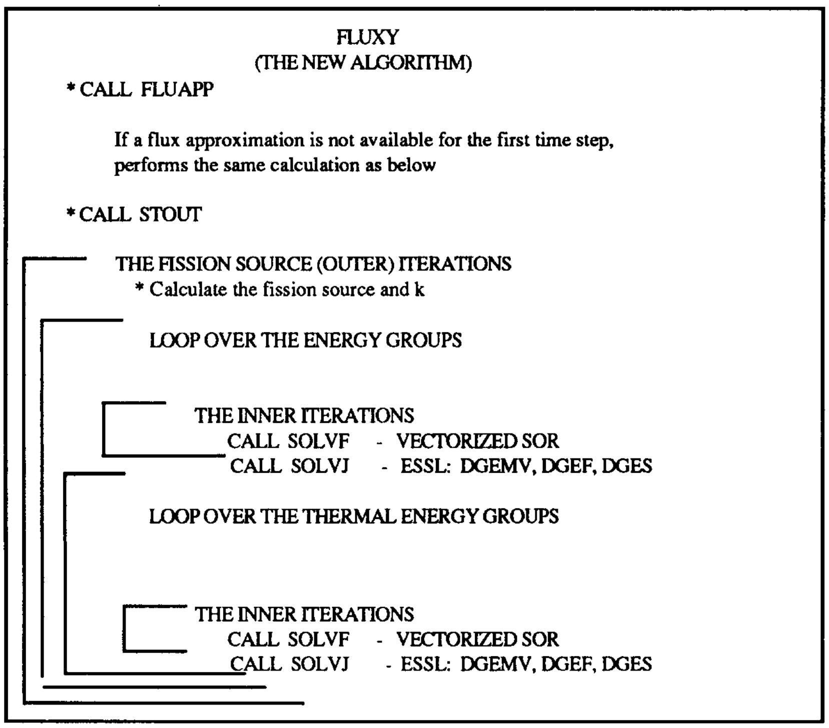


Fig. 7 Calculation of transfer probability matrices in parallel



```
                    FLUXY
            (THE NEW ALGORITHM)
 * CALL FLUAPP

     If a flux approximation is not available for the first time step,
     performs the same calculation as below

 * CALL STOUT

     THE FISSION SOURCE (OUTER) ITERATIONS
       * Calculate the fission source and k

        LOOP OVER THE ENERGY GROUPS

            THE INNER ITERATIONS
               CALL SOLVF   - VECTORIZED SOR
               CALL SOLVJ   - ESSL: DGEMV, DGEF, DGES

        LOOP OVER THE THERMAL ENERGY GROUPS

            THE INNER ITERATIONS
               CALL SOLVF   - VECTORIZED SOR
               CALL SOLVJ   - ESSL: DGEMV, DGEF, DGES
```

Fig. 8 Flow diagram for the eigenvalue problem

and 12 (CPU time). The FLUXY module was 90% vectorized, giving the scalar-to-vector job speedup of 2.2. The ESSL routines were not used in this version of the code. The CPU time in the CP matrix calculations was reduced 9 times, whereas the wallclock time on three processors was reduced for a factor of 37. The CP matrix calculation was efficiently parallelized, giving the serial-to-parallel speedup of 2.69 on three virtual processor for a parallel efficiency 90%. In this case we can be sure that three real processors were assigned to our job. The serial-to-parallel speedup, with six virtual processors requested, was 2.48 for a parallel efficiency 41%, if we assume that six real processors were assigned to our job.

Using the new vector/parallel transport algorithm for the case of isotropic scattering, DP_0 incoming flux approximation, and reflective boundary conditions, the CPU time was reduced by a factor of 44, with a corresponding reduction in wallclock time by a factor of 51. The reduction factor in the CPU time for the transfer probability matrices calculation was 120, whereas the factor for the wallclock time was 880. For the FLUXY module we have obtained vectorization speedup of 2.45 which is much larger than in the case of the modified CPM-2 code. Since the algorithm to calculate the collision, escape and transmission matrices is highly parallelized, the serial-to-parallel speedup on three processors increased to 2.98, which corresponds to a parallelization efficiency of 99.3%. The serial-to-parallel speedup, with six virtual processors requested, increased to 4.81, for a parallel efficiency of 80.2%, if we assume that six real processors were assigned to our job. Similar results were obtained for the BWR assembly. The serial-to-parallel speedups vs number of virtual processors for the modified and new vector/parallel algorithm are given in Fig. 9.

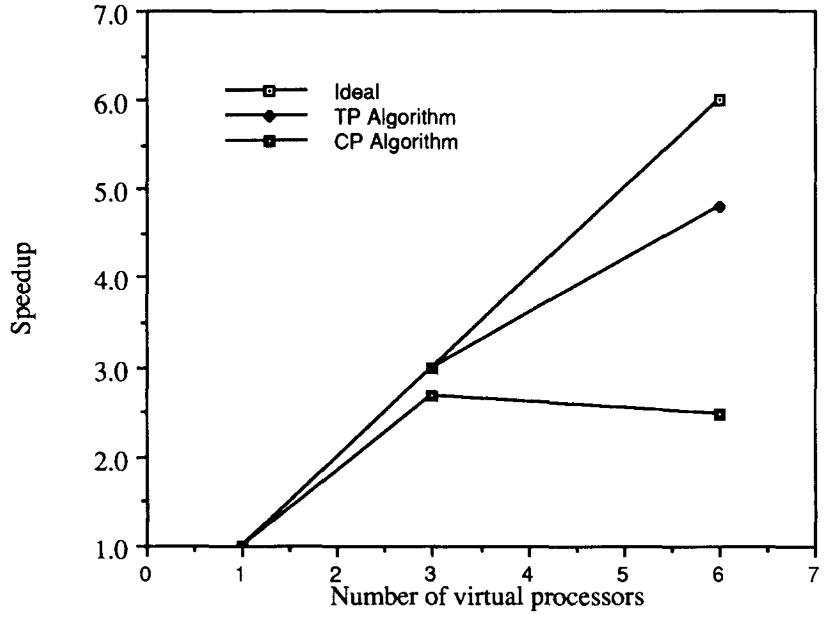


Fig. 9 Serial-to parallel speedups on the IBM 3090-600J

Table 4 gives the corresponding results on the IBM 3090 Model 600E, on one and three processors. It can be seen that the reduction in CPU time by going from Model 600E to Model 600J for the original CPM-2 was 18%, for the modified vector/parallel CPM-2, 22%, and for the new vector/parallel transfer probability algorithm, 30%. This means that the new algorithm can take much larger advantage of the faster Model 600J.

TABLE 3 Timing Results for the Original, Modified and New CPM-2 Algorithm on the IBM 3090-600J (PWR 17x17, high accuracy, 3x3 meshes per pin cell, 6 energy groups, one burnup step)

Performances for the NEW vector/parallel algorithm (TP - Matrix)	TP Matrix		FLUXY		CPM-2	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
Vector/parallel mode, 6 processors	14.63 (116.6)	2.99 (877.1)	17.31 (2.0)	19.04 (31.8)	41.02* (43.2)	66.61* (50.9)
Vector/parallel mode, 3 processors	14.25 (119.7)	4.83 (543.0)	17.30 (2.0)	18.49 (32.7)	40.59* (43.7)	67.80* (50.0)
Vector/parallel mode, 1 processor	14.17 (120.3)	14.37 (182.5)	17.22 (2.0)	18.10 (33.4)	40.39 (43.9)	76.93 (44.0)
Scalar mode	14.97 (113.9)	15.42 (170.07)	42.19 (0.8)	44.24 (13.7)	66.52 (26.6)	104.63 (32.4)
Performances for MODIFIED CPM-2 algorithm	CP Matrix		FLUXY		CPM-2	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
Vector/parallel mode, 6 processors	193.72 (8.8)	77.17 (34.0)	2.94 (11.9)	3.59 (168.4)	238.22* (7.4)	158.76*(21.3)
Vector/parallel mode, 3 processors	188.58 (9.0)	71.37 (36.8)	2.95 (11.9)	3.54 (170.8)	233.03* (7.6)	152.52*(22.2)
Vector/parallel mode, 1 processor	188.56 (9.0)	191.69 (13.7)	2.98 (11.7)	3.38 (178.9)	233.02 (7.6)	272.68 (12.4)
Scalar Mode	189.73 (9.0)	193.87 (13.5)	6.52 (5.4)	6.95 (87.0)	233.91 (7.6)	273.89 (12.4)
Performances for ORIGINAL algorithm in scalar mode	CP Matrix		FLUXY		CPM-2	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
	1705.28	2622.53	34.97	604.51	1771.80	3387.36

Numbers in parentheses are speedups with respect to the original CPM-2 code in scalar mode.
*Corrected for anomalous increase in timing for the serial portion of the code

TABLE 4 Timing Results for the Original, Modified and New CPM-2 Algorithm on the IBM 3090 -600E (PWR 17x17, high accuracy, 3x3 meshes per pin cell, 6 energy groups, one burnup step)

Performances for the NEW vector/parallel algorithm (TP - Matrix)	TP Matrix		FLUXY		CPM-2	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
Vector/parallel mode, 3 processors	22.50 (92.0)	7.61 (391.7)	23.43 (1.9)	24.53 (24.5)	58.14* (37.4)	78.51* (48.0)
Vector/parallel mode, 1 processor	22.40 (92.4)	22.52 (132.4)	22.77 (1.9)	23.45 (25.6)	57.28 (38.0)	94.82 (39.7)
Scalar mode	22.72 (91.1)	22.84 (130.5)	52.93 (0.8)	53.60 (11.2)	87.33 (24.9)	122.68 (30.7)
Performances for MODIFIED CPM-2 algorithm	**CP Matrix**		**FLUXY**		**CPM-2**	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
Vector/parallel mode, 3 processors	244.59 (8.5)	90.54 (32.9)	3.66 (11.9)	4.13 (145.6)	298.41* (7.3)	187.87*(20.1)
Vector/parallel mode, 1 processor	243.60 (8.5)	244.49 (12.2)	3.55 (12.3)	3.86 (155.7)	297.15 (7.3)	341.42 (11.0)
Scalar Mode	246.27 (8.4)	247.65 (12.0)	8.49 (5.1)	8.90 (67.5)	300.19 (7.2)	336.63 (11.2)
Performances for ORIGINAL algorithm in scalar mode	**CP Matrix**		**FLUXY**		**CPM-2**	
	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)	CPU (sec)	Wall (sec)
	2069.50	2980.67	43.58	601.13	2175.37	3766.82

Numbers in parentheses are speedups with respect to the original CPM-2 code in scalar mode.
*Corrected for anomalous increase in timing for the serial portion of the code

4. SUMMARY

Three major goals have been accomplished: 1) decrease in the computational effort, by optimizing the production reactor assembly code for vector and parallel processing on the IBM 3090 supercomputer; 2) improvement of the transport model to remove infinite lattice geometry limitation, that is implicit in the CPM-2 method, and include linearly anisotropic scattering and P_1 half-space expansion in the angular flux at assembly boundaries, and 3) efficient vectorization and parallelization of the new method. Our work was motivated by both the success and accuracy of the collision probability method in solving neutron transport problems, and its excessive demands for computing time. For the optimized CPM-2 code with a new parallel-vector transfer probability algorithm, the total CPU time was reduced by a factor of 44, with the corresponding reduction in wallclock time by a factor of 51. The transfer probability calculation is two orders of magnitude faster than the collision probability matrix calculation in the original code, with a factor of 880 reduction in wallclock time. Substantial accuracy of the new method was tested on standard PWR and BWR assemblies and compares favorably with results of the original CPM-2 code as well as the TWODANT code (Alcouffe, 1984).

Acknowledgment - This work was supported by the Department of Energy, Contract No. DE-AC02-89ER 40513, and IBM Corporation (Kingston Laboratory). We would like to thank the Cornell National Supercomputer Facility (CNSF) for providing time on their IBM 3090-600E and assisting us with IBM Parallel FORTRAN.

REFERENCES

Alcouffe R. E., Brinkley F. W., Marr D. R. and O'Dell R. D. (1984) Rep. LA-10049-m.
Bickley W. C. and Nayler J. (1935) *Phil. Mag.* **20**, 343.
CPM-2 (1987) EPRI RP1252-9.
IBM (1988) IBM-SC23-0431-0.
IBM (1989) IBM-SC23-0184.
Lewis E. E. (1977) *Nucl. Sci. Eng.* ***64***, 279.
Lewis E. E. and Miller W. F. Jr. (1984) *Computational Methods of Neutron Transport*, John Wiley&Sons.
Sanchez R. and McCormic N.J. (1982) *Nucl. Sci. Eng.* **80**, 481.
Soll D. B. (1986) "Vectorization and Vector Migration Techniques," IBM Technical Bulletin.
Tucker S. G. (1986) *IBM Systems Journal* **25**, 4.
Vujic J.L. (1989) "A Vectorized and Parallelized Assembly Transport Method for Nuclear Reactor Core Analysis," Ph.D. Thesis, University of Michigan, Ann Arbor, MI.