FIRST MASSIVELY PARALLEL ALGORITHM TO BE IMPLEMENTED IN APOLLO-II CODE

Zarko Stankovski

Commissariat à l'Energie Atomique
Service d'Etudes de Réacteurs et de Mathématiques Appliquées
Département de Mécanique et Technologie
Centre d'Etudes de Saclay - 91191 Gif sur Yvette - France
E-mail: stan@soleil.serma.cea.fr

ABSTRACT

The collision probability (CP) method [1] in neutron transport, as applied to arbitrary 2D XY geometries, like the TDT [2] module in APOLLO-II [3], is very time consuming. Consequently RZ or 3D extensions became prohibitive. Fortunately, this method is very suitable for parallelization. Massively parallel computer architectures, especially MIMD machines, bring a new breath to this method.

In this paper we present a CM5 [4] implementation of the CP method. Parallelization is applied to the energy groups, using the CMMD message passing library. In our case we use 32 processors for the standard 99-group APOLLIB-II library.

The real advantage of this algorithm will appear in the calculation of the future fine multigroup library (about 8000 groups) of the SAPHYR project with a massively parallel computer (to the order of hundreds of processors).

Introduction

The TDT (TDT for Two Dimensional Transport) method calculates the collision (escape and transmission) probabilities for a single motif or several ones (called macros) coupled by interface currents. The boundary condition for every surface can be exact (such as specular reflexion) or the neighbouring surfaces can be cupled by the interface currents.

The calculation of ij's for 2D geometries is done, for each energy group, by a double integration on R and Φ over the neutron trajectories traversing the basic motif. For each trajectory one must calculate the contribution to the Pij matrix of every traversed region i in the basic domain with every traversed region j within the optical distance $\tau \leq \tau_{max}$.

The quadrature formula, number of trajectories and their maximum length, depend on the geometrical complexity and the cross sections of the problem treated. A geometry with many regions with very complex forms requires a great number of trajectories: thousands or tens of thousands. The transparency of the fast energy groups needs very long trajectories (in length, and consequently in number of intersections): to the order of hundreds intersections. The interface current aproximation can significantly reduce the length of the trajectories, but a finer quadrature formula is necessary in order to achieve sufficient accuracy for the escape and transmission probabilities.

One can imagine several parallelization techniques, each one with a different degree of complexity. Massively parallel architectures allow simultaneous calculations on many processors, but need to find the equilibrium between different computational parameters by minimizing: the quantity of communications, repeted calculations, repeted memory storage and unoccupied processors.

It is also important to develop a parallel algorithm independent (as much as possible today) of the computer architecture, and especially independent of the number of processors. For production codes, like APOLLO-II, it is absolutely necessary to have a unique source for a sequential and a parallel computer.

In practice it is not possible to have rigorously the same source code for several computers used. The solution is the use of a pre-processor, which will activate necessary program lines. In our case lines that should be used only by the CM5 computer contains the "!CM5" mark in the firs four columns.

Parallelization algorithm

The most adapted programmation model for the necessery MIMD treatment of our problem is "message passing". The fact that there is no standard massage passing library is an acceptable sacrifice in term of portability requirement the number of library calls is limited and in any case different message passing libraries are similar. A distributed memory model is supposed.

For our first parallel algorithm, we choose to parallelize the energy group treatment. In the classical, sequential, approach the task is splited into two steps: trajectory tracking and CP evaluation. Tracking is done only once, and CP calculation is repeated for each energy group, with the same trajectories, only the cross-sections change. The optical length of the trajectories depends on the cross-sections, that is why one has to track very long trajectories, in order to "have" enough optical length for the fast groups. Because of the amount of data, the trajectories are stored on disk.

The parallel algorithm assigns each energy group to a different processor. Trajectory tracking is repeted by each processor for each group

Repeated tracking is not penalizing, otherwise one could either execute tracking on one processor, while all others are unoccupied, or to parallelize the tracking, which is a hard job. Our aproach has two other adventages. The first one is a dramatical decrease of communications: the same geometrical data are broadcast to all processors, and the corresponding total cross-sections are send to every processor; there is no tracking data communications. The second advantage is that there is no overtracking - only the necessary optical length of the trajectories is calculated for each group.

The CM5 machine used for this work has NPROC=32 processors, and the standard APOLLIB-II library has NG=99 groups: NG>NPROC and NG/NPROC is not an integer. In our exemple, if nothing more is done, we will execute, for each processor, a loop like:

```
      MYPROC = 0 ;                    NPROC = 1
!CM5  MYPROC = CMMD_self_adress();    NPROC = CMMD_partition_size()
      do ig = MYPROC+1, NG, NPROC
        call PIJ(ig, ...)                                          (A)
      end do
```

where MYPROC is the node identifier: MYPROC $\in$ (0, NPROC-1). Hence, all processors will execute the same loop, for different parametres; processor number 0 for ig = 1, NPROC+1, 2*NPROC+1..., number 1 for ig = 2, NPROC+2, 2*NPROC+2..., etc. Note that the same code works for sequential computer also.

The Pij matrix calculation time is different for each group. Thus matrices will not be written on disk in order, that is why we should use direct access.

Suppose that the calculation time is the same for every group. In this case, in 3 cycles the loop wil evaluate 3*32=96 Pij matrices. In the fourth cycle the 3 last matrices will be evaluated, while 29 processors are unoccupied. The global occupation ratio is .77, a poor performance. In practice the calculation time for each group is different, the occupation ratio will be better or worse.

For the realistic cases high energy groups are more transparents and need larger calculational times, which means that the firsts processors will be more occupated. A slight developpement of the energy groups loop:

```
MG = NG/NPROC*NPROC

do ig = MYPROC+1, MG, NPROC
  call PIJ(ig, ...)                                                (B)
end do

if(MYPROC.ge.NPROC-(NG-MG))then
  ig = NG + 1 - (NPROC - MYPROC)
  call PIJ(ig, ...)
end if
```

allows a better distribution of the calculations - the lasts groups will be calculated on the lasts processors. Note that the same code still works also for a sequential computer.

In order to achieve a better occupation ratio we propose to change the order of Pij matrix calculations.

Define the transparency of the motif containing N regions, as:

$$t = \frac{\sum_I V_i}{\sum_I \Sigma_{ti} V_i}, \quad i=1,N$$

where Σ_{ti} and V_i are the total cross section and the volume of region i.

Suppose that the calculational time for each energy group is nearly proportional to its optical transparency, which is not a bad assumption in case of specular reflexion as boundary condition. If we distribute the energy groups to be calculated over the processors in such a way that the sum of transparancies of the groups of each processor are nearly equal, we can expect calculation times to be nearly equal. Now we will execute a loop like:

```
      PROC = 1
!CM5  PROC = ORDER(T)
      do ig = NG
        if(PROC(ig).eq.MYPROC+1)call PIJ(ig, ...)          (C)
      end do
```

where the function ORDER uses group transparencies (array T) to load array PROC, so that

$$\sum_{i \in PROC(P)} T(i) \simeq Const, \quad P = 1, NPROC$$

It is clear that the greater the ratio NG/NPROC is, the more efficient the algorithm is. For NG<NPROC, one should look for a different parallelization technique.

Numerical results

Actually we do not intend to compile the entire APOLLO-2 (more than 140 000 instructions) on CM5. Thanks to the modularity, we install only the TDT module on CM5, preceding work is done on workstation, the total cross-sections are tubed to TDT. The calculated collision probabilities are stored on disk only to test processors to disk comunications, not for use.

A slight restructuration of the code was necessary in order to have the possibility to do tracking once and store data or to repeat tracking on each processor for each energy group.

TDT is written in standard FORTRAN 77, some FORTRAN 90 (and CM FORTRAN) adaptations, like dynamic allocation, were necessary. Calls to standard FORTRAN 90 intrinsic functions were introduced. The programmation was simplified by introduction of array instructions.

Two different problems are treated: a RBMK fuel channel and a RSM (Tight Lattice) hexagonal PWR assembly.

The RBMK assembly, Figure 1, with a side of 25 cm. is a graphite block with 18 fuel rods embedded in a 8,6 cm radius fuel channel. The motif was divided in 58 regions and 2 macros - one in the fuel channel, another in the graphite. CPU times, per processor are shown in Figure 2.

The RSM assembly, Figure 3, is composed of a central water-hole cell, a tube-guide cell, a fertile cell and 13 typical fuel cells. The motif was divided in 94 regions and 8 macros - macros limites crossing the cell centers. CPU times, per processor are shown in Figure 4.

Two different tipe of calculations were carried out for both problems: a global Pij calculation with single motif and specular reflexion (with algorithms A, B and C) and an interface current calculation in which the macros are coupled by interface currents (with algorithms A, and B). All the calculations were done with repeated tracking (**RT**, on the left side of the figures) and with only one tracking (**OT**, on the right side) for all groups. For the **OT** calculations, the lower rectangles represent the tracking times. The times are in seconds, but their absolute values are not significant: the operating system and the compiler of the CM5 computer used are not yet stabilized and times change with every release.

The speed up and the occupation ratio were calculated and are given in the tables. Ttr is the tracking time for **OT**. Ttot is the Pij calculational time for all groups, including the tracking time for **RT**. Tmax is the Pij calculational time for the maximum occupied processor, non including tracking time in the **OT** case. The **OT** speed-up is defined as (Ttr+Ttot)/(Ttr+Tmax). The sequentional/parallel speed-up, the more important one, is defined as (Ttr+Ttot)/Tmax where the numerator is from **OT** and the denominator from **RT**: it shows the speed-up of parallel computer versus sequentional one.

A rigorous quantification of the processor to disk communication was not possible with the actual CM5 configuration. Comparing only the CPU times and noticing that for the specular reflexion case the trajectories were not long enough on **OT** for the first three groups, and with the communications in mind, we recommend the **RT** technique.

For the interface current approximation the calculational times are the same for all groups, so none of the proposed algorithms is prefered.

For the specular reflexion, the advantage of algorithme C is evident, especially for the RSM case, where the calculational times are very group dependent.

The main result of this study is that the speed-up between sequentional and parallel computer, for specular reflexion boundary conditions, is **18** and **14**, respectively, for the **RBMK** and the **RSM** assemblies.

Conclusions

This presentations shows the advantages and the limitations of the algorithm proposed, concerning production code. The use of the CM5 computer, in the hostless mode needs a small parallelization work, but the entire code has to be checked in order to be adapted to CM Fortran.

The next step - use of an actual production code - needs another aproach. It seems to us that the host-node mode will be more adapted. The code should run on a host computer and send to the nodes, for parallel processing, only the paralellized code fragments. There are two advantages to this aproach: a) It is not necessary to adapt the entier code and b) it will be easier to do a portable code working at the others platforms, like IBM's SP1 and the couple CRAY 90 - T3D.

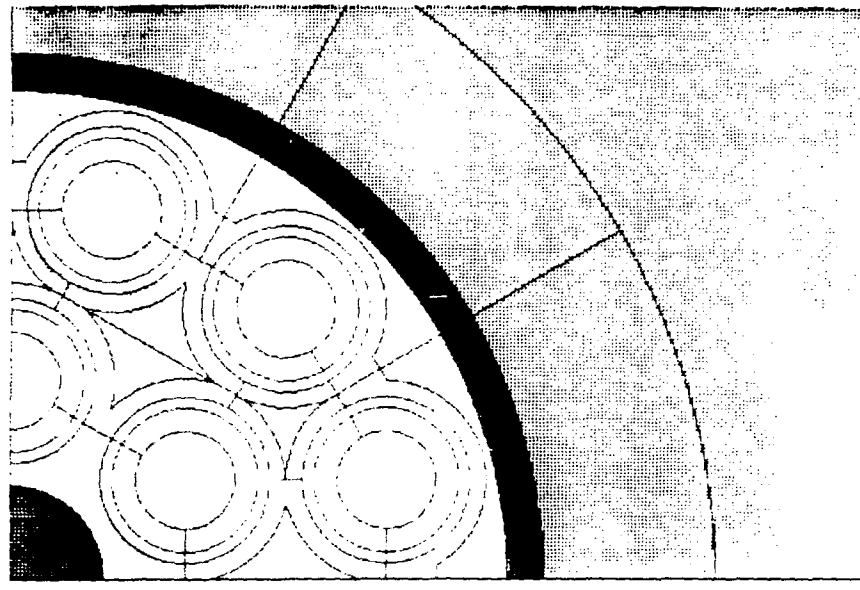


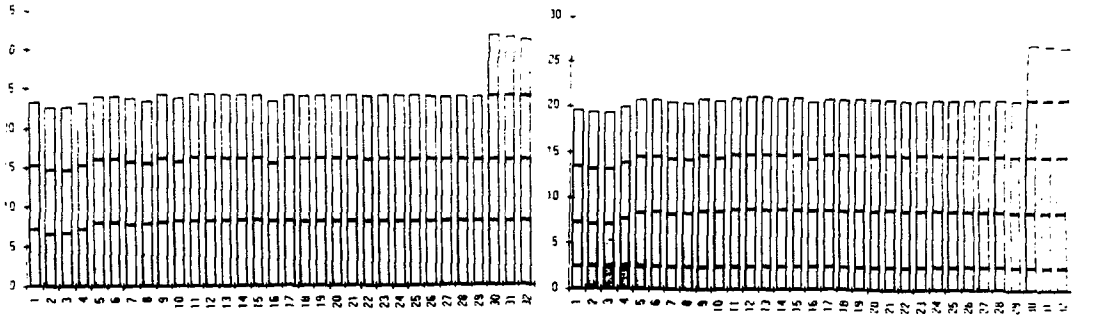
Figure 1. RBMK assembly.

Table 1. RBMK results for single motif and two macros coupled t interface currents

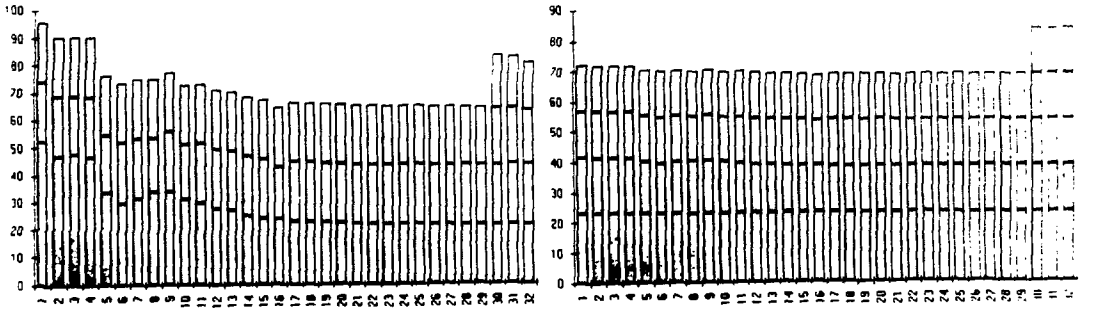
Specular reflexion	repeated tracking			only one tracking		
Algorithm	A	B	C	A	B	C
(Ttr+) Ttot	2334,0			23,0+1531		
Tmax	115,3	95,7	37,5	63,9	60,5	61,0
speed-up	20,2	24,4	26,7	17,9	18,6	18,5
occ. ratio	,63	,76	,83	,82	,85	,84
seq/par speed-up	13,5	16,2	17,8			

Interface currents	repeated tracking			only one tracking		
Algorithm	A	B	C	A	B	C
(Ttr+) Ttot	788,0			2,4+605,0		
Tmax	31,2	32,0		23,3	24,7	
speed-up	25,3	24,6		23,6	22,4	
occ. ratio	,79	,77		,83	,79	
seq/par speed-up	19,5	19,0				

a) Two macros, interface current approximation.



b) Single motif, specular reflexion, algorithm B



c) Single motif, specular reflexion, algorithm C

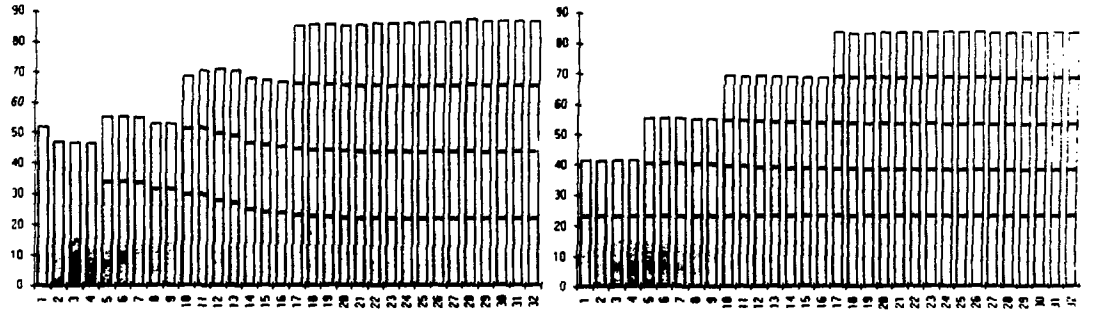


Figure 2. RBMK results. CPU times per processor. Repeated tracking for each group on left side. Only one tracking for all groups on right side, the lower rectangles represent the tracking times.

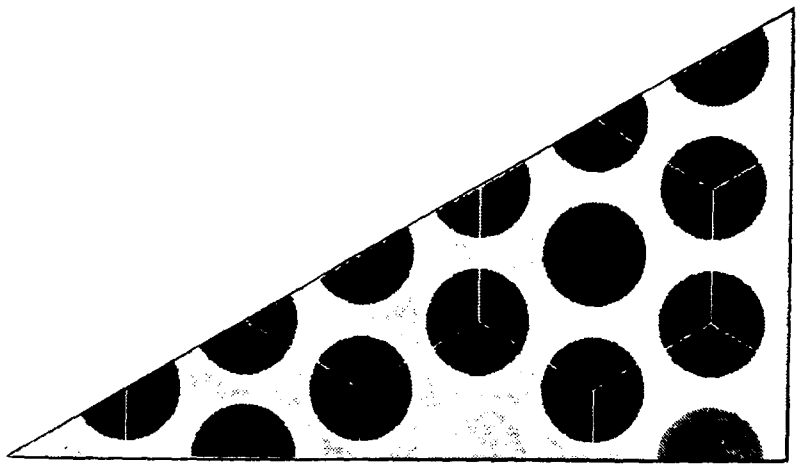


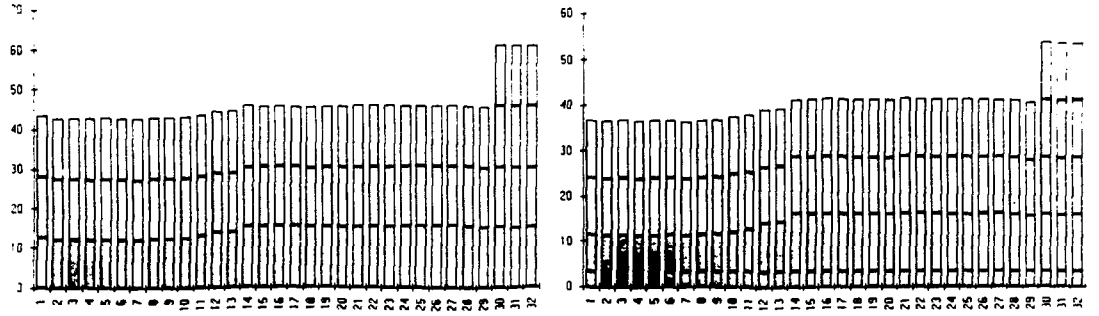
Figure 3. RSM assembly.

Table 2. RSM results for single motif and eight macros coupled by interface currents

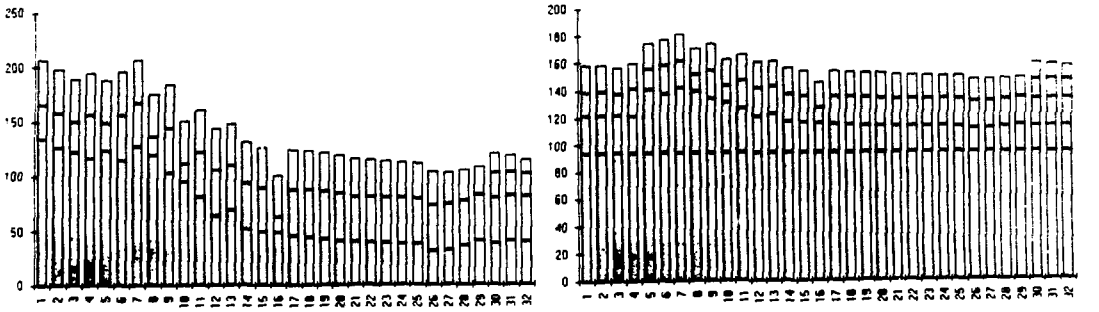
Specular reflexion	repeated tracking			only one tracking		
Algorithm	A	B	C	A	B	C
(Ttr+) Ttot	4536,0			93,9+2022,0		
Tmax	224,8	208,2	155,9	86,7	86,7	82,4
speed-up	20,2	21,8	29,1	11,7	11,7	12,0
occ. ratio	,63	,68	,91	,87	,87	,89
seq/par speed-up	9,4	10,2	13,6			

Interface currents	repeated tracking			only one tracking		
Algorithm	A	B	C	A	B	C
(Ttr+) Ttot	1487,0			3,4+1193,0		
Tmax	59,0	61,8		46,0	50,5	
speed-up	25,2	24,1		24,2	22,2	
occ. ratio	,79	,75		,82	,75	
seq/par speed-up	20,3	19,4				

a) Eight macros, interface current approximation.



b) Single motif, specular reflexion, algorithm B



c) Single motif, specular reflexion, algorithm C

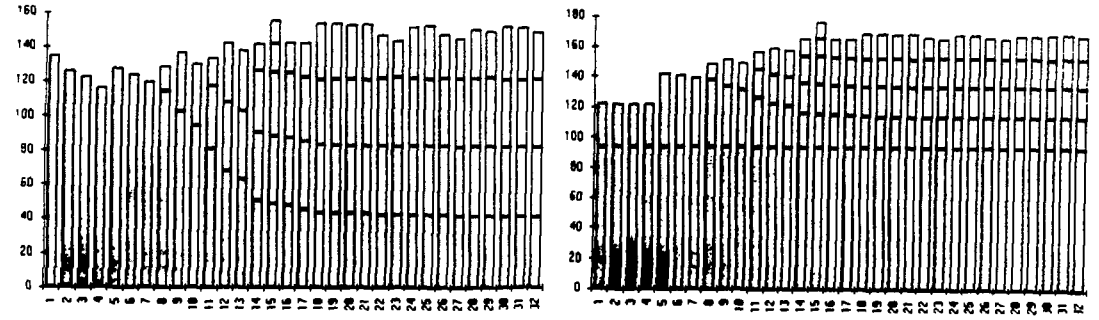


Figure 4. RSM results. CPU times per processor. Repeated tracking for each group on left side. Only one tracking for all groups on right side, the lower rectangles represent the tracking times.

Table 3. RSM results. Single motif, repeated tracking for each group, algorithm C. Speed-up for different parallel processors partition.

Nombre of processosrs	Tmax	speed-up	occupation ratio
32	156	29,08	,909
16	307	14,79	,924
8	588	7,71	,964
4	1138	3,98	,996

Acknowledgments

I would like to thank Richard for his help with TDT, and A. Pirklbauer from TMC for his helpfull advices in using CM5 machine.

References

1. R. SANCHEZ and N.J. McCORMICK, "A Review of Neutron Transport Approximations," *Nuc. Sci. Eng.* **80**, 481 (1981).

2. R. SANCHEZ and Z. STANKOVSKI, "SILENE & TDT: A Code for Collision Probability Calculations in XY Geometries," *Proc. 1993 ANS Annual Mitting,* June 20-24, 1993, San Diego, California.

3. R. SANCHEZ, J. MONDOT, Z. STANKOVSKI, A. COSSIC and I. ZMIJAREVIC, "APOLLO-II: A User-Oriented, Portable, Modular Code for Multigroup Transport Assembly Calculations," *Nuc. Sci. Eng.* **100**, 352 (1988).

4. CM5 - Technical summary, November 1993, Thinking Machines Corporation, Cambridge, MA.