Parallelization of PANDA Discrete Ordinates Code Using Spatial Decomposition

Philippe Humbert[*]
Commissariat à l'Energie Atomique, 91680 Bruyères le Châtel, France

Abstract

We present the parallel method, based on spatial domain decomposition, implemented in the 2D and 3D versions of the discrete Ordinates code PANDA. The spatial mesh is orthogonal and the spatial domain decomposition is Cartesian. For 3D problems a 3D Cartesian domain topology is created and the parallel method is based on a domain diagonal plane ordered sweep algorithm. The parallel efficiency of the method is improved by directions and octants pipelining. The implementation of the algorithm is straightforward using MPI blocking point to point communications. The efficiency of the method is illustrated by an application to the 3D-Ext C5G7 benchmark of the OECD/NEA.

KEYWORDS: Neutron Transport, Discrete Ordinates Method, Parallel Computing, PANDA code

1. Introduction

Solving the 3D neutron transport equations is a very expensive task in terms of both execution time and memory size. This is the reason why several methods have been proposed to solve the linear Boltzmann equation on 3D meshes in parallel. These methods involve domain decomposition over phase space, i.e., energy, angle or space [1].

In the present paper we address the parallelization of PANDA, a standard discrete ordinates code which solves the first order form of the transport equation on Cartesian meshes.

A first parallel version of the code was based on energy domain decomposition. This method is easy to implement using OPENMP or MPI and provides good speedup on a moderate number of processors for limited size problems. However the major drawback is that the problem size is limited because the spatial mesh must be duplicated on every processor. For these reasons a second method based on spatial domain decomposition has been implemented.

For orthogonal meshes, it was demonstrated by the KBA method [2] that efficient parallelization can be based on a diagonal plane sweep algorithm. Hence, we have programmed in PANDA the parallel diagonal sweep algorithm in the domain space using the MPI message passing library with 3D Cartesian topology of processors for 3D problems and 2D Cartesian topology of processors for 2D problems. The parallel method implemented in PANDA differs

[*] Corresponding author, Tel. 33(1)69-26-53-17, Fax. 33(1)69-26-70-96, E-mail: philippe-p.humbert@cea.fr

from the KBA method which is based on a 2D columnar domain decomposition of the 3D geometry.

In the following, we will recall the diagonal plane sweep algorithm, discuss its parallel efficiency and describe its implementation in PANDA. In the last part, we present computational results obtained on the 3D-Ext C5G7 Benchmark of the OECD/NEA [3] with TERA supercomputer at CEA/Bruyère-le-Châtel.

2. Parallel Algorithm

2.1 Parallel Diagonal Plane Sweep Algorithm with 3D decomposition

We consider a 3-D Cartesian decomposition of an orthogonal hexahedral mesh. Each domain is associated with one processor, the total number of processors is $N_P=P_X*P_Y*P_Z$ where P_X, P_Y, P_Z are the number of processors along *X, Y, Z* axis.

In the S_n method the transport equation is solved using the source iteration technique. A single iteration is composed of two parts: source calculation using previously calculated flux and flux calculation by streaming plus collision operator inversion.

$$\left[\vec{\Omega}\cdot\vec{\nabla}+\sigma\right]\varphi^{(l+1)}=S\left(\varphi^{(l)}\right) \tag{1}$$

Parallelization of source calculation using any kind of spatial decomposition is straightforward because the calculation is local and no communications are required.

In most discrete ordinates codes the streaming plus collision operator is efficiently inverted in a single iteration when the cells are processed in a specific order using an ordered sweep algorithm. The parallelization of the sweep is more involved than source calculation because of domain dependencies. For a given direction, the angular flux calculation in a spatial domain can start only when the incoming fluxes on the domain boundary are known. Communications of domain boundary angular fluxes are required. Each domain receives the incoming boundary fluxes from the incoming neighbors or boundary limit condition and sends the outgoing boundary fluxes to the outgoing neighbors.

The dependencies between domains are identical to the dependencies between cells, so the same type of ordered sweep algorithms can be employed at the domain and cell levels. Two types of sweep algorithms are used on orthogonal regular meshes. In the first one, the cells are processed successively along x, y and z axis. This method cannot be utilized for parallelization purpose because it is recursive and results in a completely sequential algorithm. The second method is the diagonal plane sweep method. A sweep consists of a sequence of steps. The cells in each step lie on a diagonal plane. These cells are independent and can be solved simultaneously in parallel. In the past, the diagonal plane sweep algorithm was successfully used for vectorization purpose.

It is worth of noting that the parallel diagonal plane ordered sweep algorithm does not need to be explicitly coded because it results implicitly from the blocking send and receive communications. The efficiency of the algorithm is also automatically improved due to a "pipelining" effect when the order in which directions are processed is such that all directions in one octant are computed before the next adjacent octant is considered.

This method is different from the KBA method which uses a 2D decomposition (P_Z=1). Another difference is that the angular fluxes in KBA method are not communicated only at the beginning and at the end of the domain processing but for each line or chunk of lines of the domain. This results in a greater number of short messages.

The presented algorithm has the advantage to use a 3D decomposition, a smaller number of greater size messages and its implementation is extremely simple because it needs essentially only one send and one receive message to transform a sequential code to a parallel one. On the other hand, because the inherent parallelism of the diagonal plane sweep algorithm is 2D the 3D decomposition method does not scale contrary to the KBA method. It means that the asymptotic parallel efficiency tends to zero for an infinite number of processors. However, we will see that, for typical applications, good speedups can be obtained with up to hundreds of processors using our 3D decomposition method.

2.2 Parallel efficiency of the ordered sweep algorithm with pipelining

The parallel efficiency is the speedup per unit processor.

$$PE = \frac{1}{N}\frac{T(1)}{(N)} \tag{2}$$

The maximum parallel efficiency of a sweep algorithm is the ratio of the number of steps in an ideal parallel algorithm to the number of steps in the actual sweep algorithm. This is an upper limit because it does not take into account the inter-processors communication time.

For a $N_P = P^3$ spatial decomposition, there are $3P-2$ steps in a diagonal sweep, so the parallel efficiency is:

$$PE_1 = \frac{1}{3P-2} \tag{3}$$

This equation shows that the intrinsic parallel efficiency of the diagonal plane sweep algorithm with 3D decomposition is not scalable. Although this method is not well adapted to massively parallel computation it can be improved using directions and octants pipelining.

In the S_n method there are $N_O = \frac{n(n+2)}{8} = \frac{N_D}{8}$ directions in each octant. When a processor has processed one direction it can proceed with the next direction in the current octant without waiting. This pipelining of directions within an octant increases the parallel efficiency of the sweep algorithm

$$PE_2 = \frac{N_O}{3(P-1)+N_O} \tag{4}$$

Furthermore, when a processor has processed all the directions on the current octant it can proceed with the next octant without waiting. This pipelining of octants increases again the parallel efficiency.

$$PE_3 = \frac{N_D}{10(P-1)+N_D} \tag{5}$$

However, reflective boundary conditions require octants to be processed in a given order that reduce pipelining or degrade iterative performance.

Parallel efficiency results presented in table 1 show that a theoretical high level of parallelism can be achieved with hundreds of processors when the number of directions is much greater than the number of processors on a given axis $(N_D > 10(P-1))$. When this condition is not satisfied and for massively parallel computing applications a scalable algorithm like the KBA method is more appropriate.

Table 1: Maximum parallel efficiency results of the 3D domain decomposition diagonal plane sweeping algorithm. Three cases are considered, no pipelining (PE_1), pipelining of directions (PE_2) and of both directions and octants pipelining (PE_3).

Number of Processors=P^3	Method	S_8 (80 directions)	S_{16} (288 directions)	S_{24} (624 directions)
$64=4^3$	PE_1	10%	10%	10%
$64=4^3$	PE_2	52.63%	80%	89.66%
$64=4^3$	PE_3	72.73%	90.57%	95.41%
$512=8^3$	PE_1	4.54%	4.54%	4.54%
$512=8^3$	PE_2	32.26%	63.15%	78.79%
$512=8^3$	PE_3	53.33%	80.45%	89.91%

3. Algorithm Implementation with MPI

The 3-D parallel diagonal sweep algorithm with 3-D spatial domain decomposition was implemented in PANDA discrete ordinate code using the MPI message passing interface. Starting from the sequential version of the code, only minor modifications are necessary to implement the parallel version.

The updates include first, the creation of the virtual Cartesian topology of processors and mesh partitioning, second, the local communication of boundary angular fluxes between neighbors and third the global communication for the flux norm used to quantify the convergence of the iterative algorithm.

Tools for creating a virtual topology are provided by the MPI library. For a given total number of processors a spatial decomposition along X, Y and Z axis is given by the *MPI_DIMS_CREATE()* function. The virtual Cartesian topology is generated using the *MPI_CART_CREATE()* function which creates a new communicator with the specified topology. The *MPI_CART_GET()* function returns the current process coordinates in the grid and the neighbors ranks are given by the *MPI_CART_SHIFT()* function. Using these tools the total orthogonal hexahedral mesh is partitioned into a Cartesian grid of domains. When possible each domain has the same size in order to optimize the load balancing.

The other modifications of the sequential algorithm concern communications. The algorithmic structure remains unchanged. The parallel version of the code is obtained by just inserting point to point and global communication function calls into the sequential algorithm.

The streaming plus collision operator inversion operator for angular fluxes calculation needs communication of the boundary angular fluxes between neighbor's processors. The point to point communications are implemented using basic MPI blocking send, *MPI_SEND()* and receive, *MPI_RECV()* functions. Within a domain, at the cell level we do not use the diagonal sweep but rather a simple z-level plane sweep. The parallel diagonal plane ordered sweep algorithm for domains is not explicitly coded; it results implicitly from the blocking communications. In the same way, pipeline processing results from the sequential ordering of directions and octants and parallel processing without collective process synchronization (barriers).

There are no modifications of the source calculation algorithm because it is local and the parallel version does not need any communications.

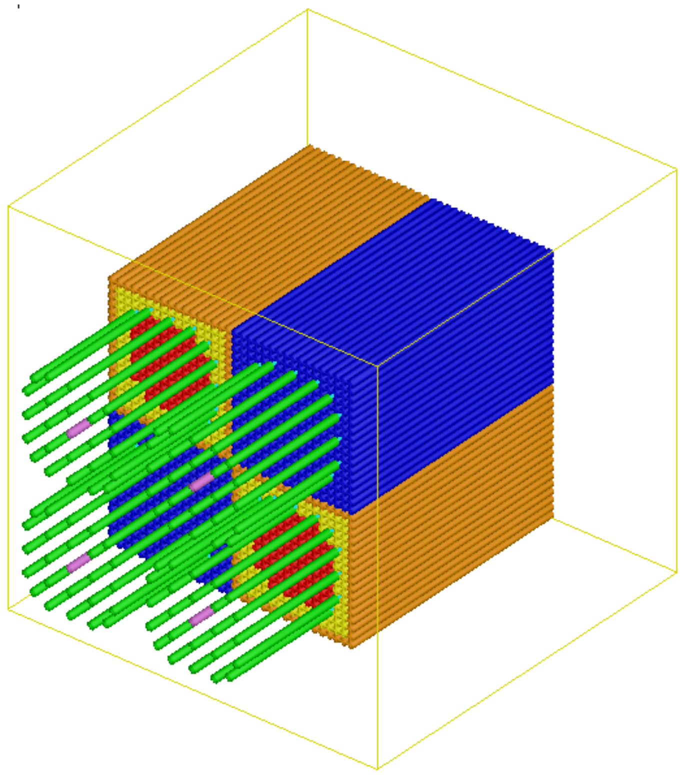
Eigenvalue and neutron fluxes are computed using iterative algorithms. The convergence test depends on the global fission source whose calculation requires a global reduction operation implemented by a *MPI_ALLREDUCE()* function.

4. Computational Results

4.1 The benchmark

In order to test the performances of this new parallel version of PANDA on realistic 3D problems we considered the C5G7 3D-Ext benchmark proposed by the OECD/NEA [3]. This benchmark is a quarter symmetric MOX/UO2 core surrounded by a light water moderator. It consists of an array of 34x34 pins gathered in two MOX assemblies and two UO2 assemblies. Three problems are defined corresponding to different levels of control rod insertion. Here we have treated the case for which the control rods are not inserted in the core (unrodded case). An illustration of the benchmark geometrical modeling is presented in figure 1. The problem is to calculate the effective multiplication factor eigenvalue.

Figure 1: PANDA geometrical modeling of the UO2/MOX reactor benchmark (unrodded 3-D Ext C5G7). The colors represent the different materials.



The calculations were run on CEA Tera supercomputer, a Compaq alphaserver with 2560 DEC alpha EV68 1Ghz processors. We performed two series of calculations versus processor number, the first one is a fixed total size problem and the second one is a fixed size per processor problem. For all computations the number of directions for an S_{16} angular quadrature is equal to 288 and the number of energy groups is equal to 7.

4.2 Fixed Size Problem

The X-Y plane is perpendicular to the fuel pins, and the Z axis is parallel to the pins. The spatial discretization is uniform along each axis with NX=NY=512 and NZ=32. The total number of cells is equal to 8 388 608.

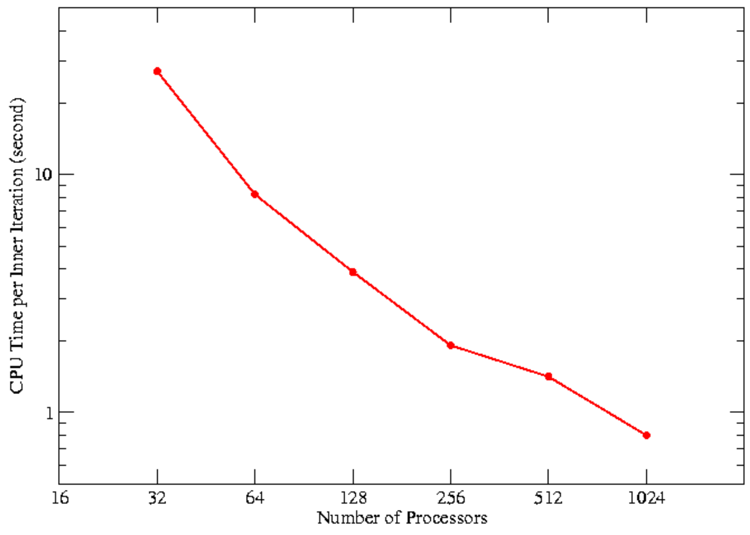
In order to fit in memory, the problem was run on a minimum number of 32 processors due to the 4 Giga bytes memory limit per processor. The computational results are given in term of CPU time per inner iteration versus processor number. They are detailed in table 2 and plotted in figure 2.

Table 2: Fixed size problem computational results. The relative speed up is taken between two successive calculations where the processor number is doubled.

Number of Processors	Spatial Decomposition $P_X*P_Y*P_Z$	CPU time per Inner Iteration (second)	Relative Speed Up
32	4*4*2	27.14	-
64	8*4*2	8.23	3.30
128	8*4*4	3.89	2.12
256	8*8*4	1.91	2.04
512	8*8*8	1.41	1.35
1024	16*8*8	0.80	1.76

The relative speed up is equal to the ration of the CPU time between two successive calculations. Because the processor number is doubled, the relative speed up should be less than two, however relative speed up greater than two are observed in table 1, this may result from a more efficient memory access (cache management) due to domain size reduction. The reduction in CPU time is very important between the 32 and 64 processor cases with a relative speed up equal to 3.30.

Figure 2: Total CPU time per inner iteration versus processor number for a fixed size problem.



4.2 Fixed Size per Processor Problem

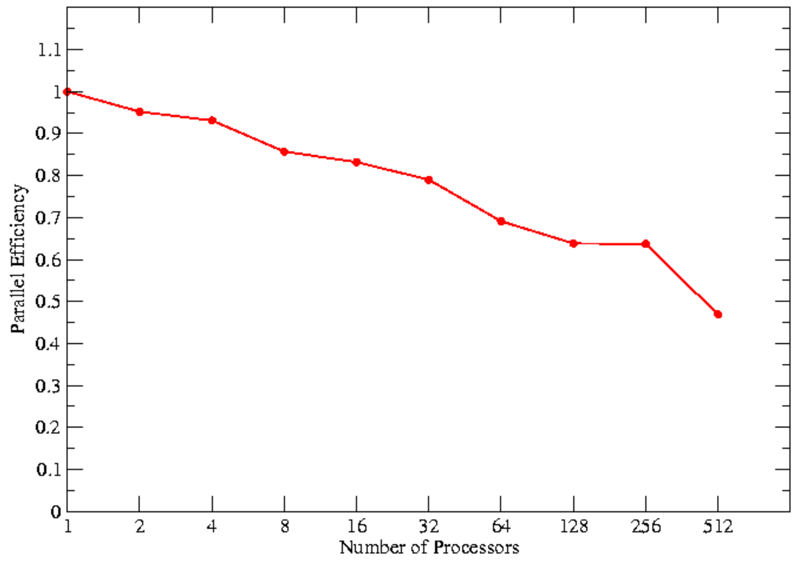
In order to remain insensitive to the memory management effect we have also considered a series of calculation for a fixed size per processor problem. The domain size associated to each processor is characterized by NX=NY=64 and NZ=4. The number of cells per processor is equal to 16384.

We have reported in table 3 the CPU time per inner iteration. The parallel efficiency plotted in figure 2 is equal to the ratio of the parallel (N processors) CPU time to the sequential (1 processor) CPU time. The parallel efficiency decreases with processor number but remains near 50% for 512 processors.

Table 3: Fixed size per processor problem computational results.

Number of Processors	Spatial Decomposition $P_X*P_Y*P_Z$	CPU time per Inner Iteration (second)	Parallel efficiency
1	1*1*1	0.662	1.000
2	2*1*1	0.695	0.952
4	2*2*1	0.711	0.931
8	2*2*2	0.739	0.857
16	4*2*2	0.796	0.832
32	4*4*2	0.838	0.790
64	8*4*2	0.958	0.691
128	8*4*4	1.037	0.638
256	8*8*4	1.039	0.637
512	8*8*8	1.412	0.469

Figure 2: Parallel efficiency versus processor number for a fixed size per processor problem.



6. Conclusion

In order to treat larger problems with reduced execution time a new 3-D parallel version of PANDA discrete ordinates code has been developed using a 3-D domain decomposition method.

The streaming plus collision operator inversion is performed using a diagonal plane sweep of the Cartesian domain topology and processor efficiency is improved by pipelining octants and angles within an octant. It is an intrinsic algorithm so eigenvalue results and iteration numbers are independent of the processor number.

The implementation of the algorithm is straightforward using the MPI library because it results implicitly from the blocking communications of domain boundary angular fluxes.

Although a 3-D decomposition on a structured mesh is not scalable we observe high efficiencies with hundred of processors on a typical 3-D benchmark problem.

References

1) B.L. Kirk, E. Sartori, Luis Garcia de Viedma, " An Overview of the Activities of the OECD/NEA Task Force on Adapting Computer Codes in Nuclear Applications to Parallel Architectures", M&C'1997, Saratoga, USA (1997).
2) R.S. Baker, K.R. Koch, "An SN Algorithm fort he massively parallel CM-200 Computer", *Nucl. Sci. Eng.*, **128**, 312 (1998).
3) M.A. Smith, E.E. Lewis, Byung-Chan Na, "Benchmark on Deterministic Transport Without Spatial Homogenisation. MOX Fuel Assembly 3-D Extension Case", OECD/NEA report, NEA/NSC/DOC(2005)16, (2005)