Joint International Topical Meeting on Mathematics & Computation and Supercomputing in Nuclear Applications (M&C + SNA 2007)
Monterey, California, April 15-19, 2007, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2007)

A LOAD BALANCING STRATEGY FOR THE RADIATION TRANSPORT CODE EVENT

Aliva Pattnaik and Cassiano R. E. de Oliveira
Nuclear and Radiological Engineering Program,
George W. Woodruff School of Mechanical Engineering,
Georgia Institute of Technology
Atlanta, GA 30332-0405
aliva@gatech.edu; c.oliveira@gatech.edu

ABSTRACT

Uneven assignment of the computational load among processors causes a significant portion of the total solution time being spent on processor synchronization. To address this inefficiency, we propose two different guiding parameters: one for determining whether the computational load is evenly distributed among the processors, and another for determining whether improvement in the speed up can be achieved. These two parameters are computed from a fast, auxiliary diffusion calculation, and, based on their values, a load balancing technique is applied to repartition the mesh. This strategy has been implemented in the finite element-spherical harmonics radiation transport code EVENT and the speed up achieved for higher-order angular approximations before and after load redistribution is compared for the three dimensional C5G7Mox [4] reactor problem. A significant improvement on speed up is achieved with higher number processors after load redistribution.

Key Words: Load balancing, Radiation transport, High performance computing, Finite Elements, Spherical Harmonics

1. INTRODUCTION

The general purpose radiation transport code EVENT [1] solves the self-adjoint, second-order form of the radiation transport equation. This form of transport equation provides a natural framework for finite element spherical harmonics discretizations and yields a linear system of equations for each energy group of the form [2]:

$$\boldsymbol{A}\,\boldsymbol{\Psi} = \mathbf{b} \tag{1}$$

where $\boldsymbol{A}$ is a NM x NM matrix, $\boldsymbol{\Psi}$ the NM x 1 vector of space-moment unknowns, with N the number of nodes in the finite element mesh, and M the number of spherical harmonics moments in the expansion of the angular flux. The matrix $\boldsymbol{A}$ is symmetric, positive-definite and, due to the compact support of the finite element basis, sparse.

Solution of equation (1) is obtained by the iterative moment-by-moment preconditioned conjugate gradient (MBM-PCG) method [3]. The matrix is partitioned the into MxM submatrices of size NxN :

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & .. & \boldsymbol{A}_{1M} \\ \boldsymbol{A}_{12}^{T} & \boldsymbol{A}_{22} & .. & .. \\ .. & .. & .. & \boldsymbol{A}_{M-1M} \\ \boldsymbol{A}_{1M}^{T} & .. & \boldsymbol{A}_{M-1M}^{T} & \boldsymbol{A}_{MM} \end{pmatrix} \tag{2}$$

where the $\boldsymbol{A}_{ij}$ possess the diffusion matrix stencil. Only the diagonal angular sub-matrices are explicitly assembled. Operations involving the off-diagonal matrices are performed using the tensorial product nature of these matrices. The MBM preconditioning step requires the explicit inversion of the diagonal block matrices $\boldsymbol{A}_{ij}$ and this represents a considerable part of the solution effort.

For large problems parallel solution is necessary. This is accomplished through the use of a multi-block explicit domain decomposition procedure. In this procedure the NxN block matrix $\boldsymbol{A}_{ii}$ is further partitioned as follows:

$$\boldsymbol{A}_{ii} = \begin{pmatrix} \boldsymbol{M}_{11} & \boldsymbol{M}_{12} & .. & \boldsymbol{M}_{1p} \\ \boldsymbol{M}_{21} & \boldsymbol{M}_{22} & .. & .. \\ .. & .. & .. & \boldsymbol{M}_{p-1p} \\ \boldsymbol{M}_{p1} & .. & \boldsymbol{M}_{pp-1} & \boldsymbol{M}_{pp} \end{pmatrix} \tag{3}$$

where p is the number of processors. The diagonal block matrices $\boldsymbol{M}_{ii}$ are the finite element matrices strictly contained within each processor domain. The off-diagonal matrices $\boldsymbol{M}_{ij}$ contain the connections between the boundary nodes shared by different processors. The multi-block explicit procedure chooses the preconditioner matrix as:

$$\boldsymbol{M} = \begin{pmatrix} \boldsymbol{M}_{11} & 0 & \ldots & 0 \\ 0 & \boldsymbol{M}_{22} & .. & .. \\ .. & .. & .. & 0 \\ 0 & .. & 0 & \boldsymbol{M}_{pp} \end{pmatrix} \tag{4}$$

The preconditioner above involves explicit inversion of the matrix $\boldsymbol{M}_{ii}$ which can still require a considerable effort for a relatively small number of processors. An increase in the number of processors reduces this effort but at the expense of increased communication between them. Also, if the computational work for each processor is not equal, then the cost of synchronizing the processors increases with an increase in number of processors.

The performance of the above parallel solution strategy in a parallel environment depends on an even distribution of computational load across the processors and the minimum number of halo nodes. Even if the number of spatial nodes assigned to each processor is kept almost equal in each of the processors, computational load may still widely vary across processors due to the governing physics. To solve this imbalance in the computational load, we have proposed a load balancing technique which is described next.

2. Load Balancing Approach

We are interested here in whether the computational load is balanced ie. each processor is assigned the same amount of work, because distribution of load affects synchronization time and in turn, the total execution time. Synchronization of a group of processor dictates that all processors must wait at the point of synchronization until all of the processors in the group arrive at that point. The synchronization time is the difference between the time when the first processor arrives and the time when the last processor arrives. Thus the time to synchronize among a group of processors is determined by the fastest and slowest processor of the group. When load is not balanced, it is more likely that a processor with a lighter load arrives at the synchronization point earlier than a processor with a heavier load, and thus increasing the synchronization time.

2.1 Detection of Load Imbalance

To determine whether load is balanced, we propose an indicator whose value represents the magnitude of the load imbalance. The indicator is computed by the computing standard deviation of percentage of waiting times of all processors. The percentage of waiting time of a processor refers to the percentage of the total time that the processor has to wait for other processors over all synchronization points. The intuition here is if the standard deviation is high, it means there are some processors who have to wait significantly less than other processors, a possible indication that those processors who wait more have a lighter load than those processors who wait less. In contrast, if the standard deviation is less, it means all processors have to wait almost equal amount of time; which will be the case when all processors have almost equal amount of load. However, a significant standard deviation alone does not necessarily indicate an opportunity for optimization through load rebalancing. Load rebalancing can benefit only if in the original problem, the processors spend a significant portion of the time waiting.

We propose that if the load is imbalanced (as determined from the standard deviation), there is a scope for improving the parallel performance by load rebalancing if the second indicator i.e. average percentage of waiting time over all processors, is a significant portion of the total running time. The total % of waiting time is calculated as follows:

$$\text{Average percentage of waiting time} = \frac{\sum_{i=1}^{P} (WaitTime)_i}{\sum_{i=1}^{P} (CPU\ Time)_i} * 100$$

where P is the number of processors, *WaitTime* is the recorded waiting time for each processor, and *CPU Time* is the total running time of the problem

2.2. Load Rebalancing Strategy

In redistribution, a new mesh partition is created with the goal that each new partition corresponds to an equal computational load. For load-rebalancing, first the problem is solved for

diffusion case (P_1 approximation), which requires significant less solution time. Waiting times for each processor is recorded in the diffusion run and a weight for each processor is computed as follows from the recorded waiting times:

$$(weight)_i = \frac{MaxWaitTime \ - \ (WaitTime)_i}{MaxWaitTime \ - \ MinWaitTime}$$

where, $(WaitTime)_i$, and$(weight)_i$ represent the recorded waiting time in diffusion run and computed weight for i^{th} processor respectively. *MinWaitTime* and *MaxWaitTime* represent the minimum and maximum of the recorded waiting times of all processors respectively. $(weight)_i$ as calculated above takes a value between 0 and 1, and is calculated such that weights for processors are inversely proportional to their waiting times. The intuition is, if a processor has more load, it waits for less time and thus gets a higher weight.

1. Run EVENT for diffusion case and record waiting times for each processor.
2. Compute an weight $(weight)_i$ for each processor i as follows:

$$(weight)_i = \frac{MaxWaitTime - (WaitTime)_i}{MaxWaitTime - MinWaitTime}$$

   $(WaitTime)_i$ - waiting time of i^{th} processor

   $MinWaitTime$ - minimum of the waiting times over all processors

   $MaxWaitTime$ - maximum of the waiting times over all processors
3. Each node is given the weight of the processor that it is assigned to.
4. The spatial domain is repartitioned with the goal to keep sum of the weights of the nodes same for all domains.
5. EVENT is then run for higher moments.

Figure 1. Load Balancing Algorithm

After a weight for each processor is calculated, the mesh is repartitioned using METIS [4]. METIS can partition a spatial mesh in such a way that each sub-domain will have almost equal number of total nodal weight. In other words, METIS can partition a mesh where each node has been assigned a weight, such that total weights for all nodes in each partition are kept equal as much as possible. Finally, the repartitioned mesh is used for solving for higher-order approximations. The algorithm for the load balancing strategy is given in Figure 1.

3 NUMERICAL RESULTS

To evaluate the effectiveness of the proposed load rebalancing technique, the three-dimensional version of the C5G7Mox [5] benchmark problem was solved in parallel using a finite element mesh consisting of 1288600 elements and 235968 nodes. The solution runs were performed in a 17-node cluster, with each node consisting of 2 1.8GHz dual-core 64-bit Opteron processors and 8GB of RAM.

The total CPU time was collected for three different angular approximations: diffusion (P_1), P_3, and P_5 using 8, 16, 32, and 64 processors. Table I shows the standard deviation of percentage of waiting times among the processors for the various cases. Table II shows the average percentage of waiting time for the different orders of angular approximation and number of processors. It can be seen from Table I and Table II that the value of both guiding parameters, namely standard deviation of percentage of waiting times and average percentage of waiting times, were high for all approximations for the 64 processors case. These cases were thus deemed potential candidates for optimization through load balancing.

Table I. Standard deviation in percentage of waiting time of processors

Number of Processors	P_1	P_3	P_5
8	7.3%	8.9%	9.1%
16	5.3%	5.9%	8.3%
32	3.8%	6.8%	7.8%
64	6.47%	4.8%	5.9%

Table II. Total percentage of waiting time for angular approximations

Number of Processors	P_1	P_3	P_5
8	13%	14%	15%
16	16%	18%	18.6%
32	24%	20%	21%
64	72%	56%	43%

The load balancing technique previously described was applied to the numerical problem for the P_3 and P_5 approximation solutions on 64 processors. Figure 2 and Figure 3 show the speed-up achieved for the two problems before and after load-rebalancing. In each of two cases, significantly higher speed-up is achieved after load-rebalancing.

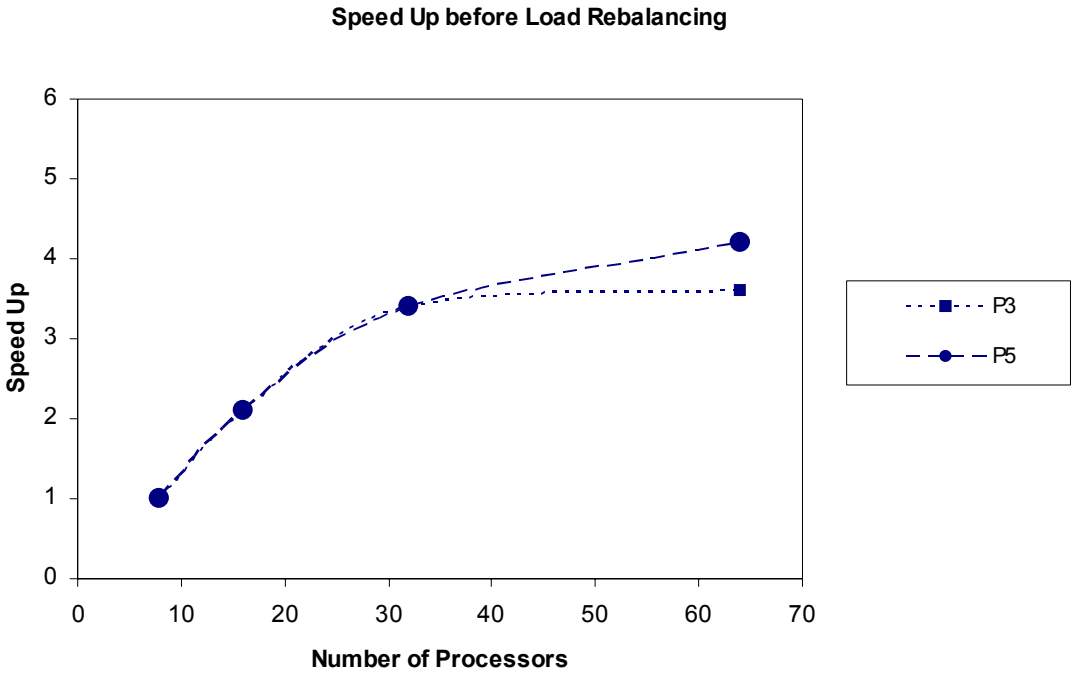

Figure 2. Speed Up before load balancing

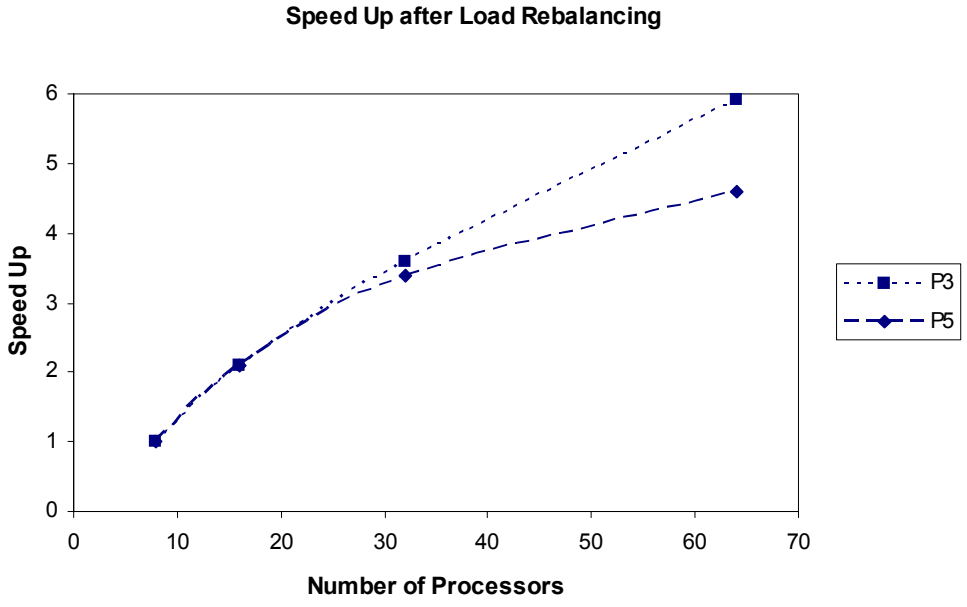

Figure 3. Speed Up after Load Rebalancing

4. CONCLUSIONS

We have proposed two indicators that identify when the computational load is not evenly distributed among the processors and thus load-rebalancing may be required. For cases where the opportunity for such improvement exists, we have proposed a load-rebalancing technique that computes weights for each domain from a fast diffusion computation and then uses the weight to re-partition the mesh with the goal of evenly distributing the load. Empirical results showed significant improvement in speed-up through the proposed load-rebalancing technique. In future, along with the percentage of waiting time, we would like to assign additional weights to each node based on the adjoint neutron flux. Also, this suggested load balancing strategy will be tested for other reactor benchmark problems. Finally consideration will be given to extending the analysis to the case of space-angle adaptivity.

ACKNOWLEDGMENTS

This work is supported by the US Department of Energy through the Nuclear Engineering Education Research Program (NEER) Award DE-PS07-03ID14540.

REFERENCES

1. C. R. E. De Oliveira, A. J. H. Goddard, “EVENT - A Multidimensional Finite Element-Spherical Harmonics Radiation Transport Code”, (Proceedings of the OECD International Seminar on 3D Deterministic Radiation Transport Code*s*, Paris, December 01-02, 1996).
2. C. R. E. De Oliveira, "An Arbitrary Geometry Finite Element Method for Multigroup Neutron Transport with Anisotropic Scattering", *Prog. in Nuclear Energy*, 18, 227 (1986).
3. C. R. E. De Oliveira, C. C. Pain, A. J. H. Goddard, “Parallel Domain Decomposition Methods for large-Scale Finite Element Transport modeling”, Proceedings of the ANS International Conference on Mathematics and Computations, Reactor Physics and Environmental Analysis, Portland, OR, (1995).
4. METIS - A Software Package for Partitioning Unstructured Graphs, Meshes, and Computing Fill-Reducing Orders of Sparse Matrices, Version 4.0: www.scs.fsu.edu/~burkardt/pdf/metis.pdf
5. Benchmark on deterministic transport calculations without spatial homogenization, NEA/NSC/DOC, (2003)16.