AN ALGORITHM FOR PARALLEL S$_N$ SWEEPS ON UNSTRUCTURED MESHES

Shawn D. Pautz
Transport Methods Group, CCS-4, MS-D409
Los Alamos National Laboratory
Los Alamos, NM 87545
pautz@lanl.gov

Keywords: radiation transport, parallel processing, sweeps, scheduling

ABSTRACT

We develop a new algorithm for performing parallel S$_n$ sweeps on unstructured meshes. The algorithm uses a low-complexity list ordering heuristic to determine a sweep ordering on any partitioned mesh. For typical problems and with "normal" mesh partitionings we have observed nearly linear speedups on up to 126 processors. This is an important and desirable result, since although analyses of structured meshes indicate that parallel sweeps will not scale with normal partitioning approaches, we do not observe any severe asymptotic degradation in the parallel efficiency with modest (≤ 100) levels of parallelism. This work is a fundamental step in the development of parallel S$_n$ methods.

1 INTRODUCTION

The standard iterative technique for solving discretized transport equations is source iteration, in which one alternates between solving for the local scattering source and inverting the global streaming-plus-collision operator. In the case of discrete ordinates (S$_n$) equations derived from the first-order form of the transport equation the streaming-plus-collision operator is usually directly inverted by the method of "sweeping." During a sweep this operator is locally solved for each spatial cell in the mesh in a specified order for a single direction in the discrete ordinates set. This order is constrained by the interaction of the discrete ordinates set with the spatial mesh; a cell cannot be solved for a particular direction until its "upstream" neighbors have been solved. Because of these constraints the solution order often resembles a plane wave "sweeping" across the mesh along the ordinate direction.

Because of the constraints placed on the sweep ordering, parallelization of the sweep process is difficult. Tasks (cell-angle pairs) assigned to a processor cannot begin until cells upstream from them have been solved; the processor may be idle for some time if upstream tasks are on other processors. The assignment of tasks to processors and the ordering of that work must be carefully coordinated in order to obtain good parallel efficiencies and speedups. Sweep problems are a subset of the class of problems known in computer science as "scheduling" problems, which are difficult to solve in general.

For the special case of sweeps on orthogonal hexahedral meshes the KBA (Koch-Baker-Alcouffe) sweep algorithm has been developed (Koch, 1992, Baker, 1995, Baker, 1997, Baker, 1998). The KBA algorithm uses a special columnar domain decomposition and a particular sweep ordering to obtain very high parallel efficiencies. This algorithm is difficult to generalize to unstructured meshes; it is not obvious what a "columnar" decomposition on an unstructured mesh is or what the corresponding ordering should be.

The purpose of this paper is to develop an algorithm for efficient parallel sweeps on unstructured meshes. Our focus is on obtaining a good sweep ordering; we defer the problem of obtaining a specialized decomposition and use instead a more traditional decomposition. Although this limits the scalability of our algorithm (one needs both the "right" decomposition and a good ordering for scalability) we are able to obtain near-ideal efficiencies and speedups with over 100 processors. Furthermore, our ordering algorithms are designed to work with any decomposition and should be able to exploit favorable properties of specialized partitions as they become available in the future.

The rest of the paper is organized as follows. First we discuss the nature of the sweep process and relate it to the general class of scheduling problems. We also describe the KBA approach for structured meshes. Next we develop an algorithm for parallel sweeps on unstructured meshes. We then present theoretical performance estimates of our algorithm and we compare these estimates to computational results. Finally, we make some conclusions and recommendations for future work.

2 SWEEPS AND THE GENERAL SCHEDULING PROBLEM

As mentioned in the introduction, the problem of efficiently parallelizing sweeps is a subset of the class of scheduling problems. Scheduling concerns itself with the distribution and ordering of tasks among processors, particularly when there are data dependencies between tasks. In this section we will discuss S_n sweeps and show how their parallelization can be described in terms of scheduling problems. We also will discuss the KBA algorithm for scheduling sweeps on structured meshes.

2.1 S$_n$ Sweeps and Scheduling Theory

Discretizations of the Boltzmann transport equation are generally solved by means of source iteration, which we present here for the first-order form of the equation:

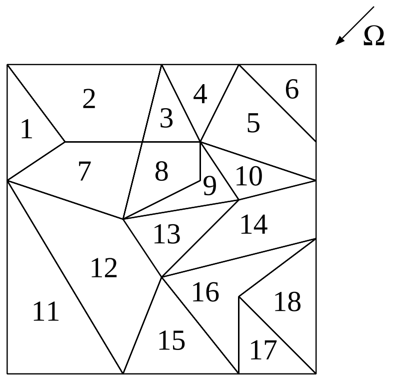
$$[\mathbf{\Omega} \cdot \nabla + \sigma_t]\psi^{(l+1)} = M\Sigma\phi^{(l)} + q, \tag{1a}$$

$$\phi^{(l+1)} = D\psi^{(l+1)}, \tag{1b}$$

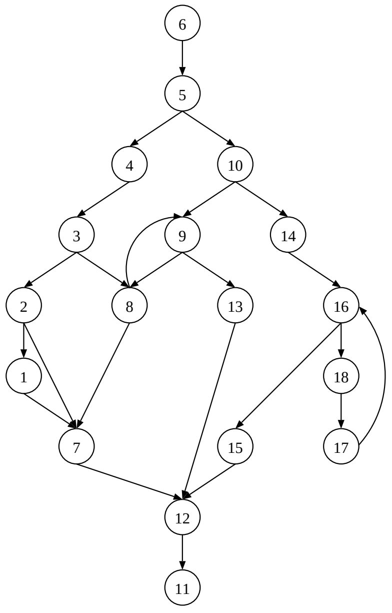
where ψ is the angular flux distribution, ϕ is the set of angular flux moments, l is the iteration index, $\mathbf{\Omega}$ is the direction of particle travel, D, M, and Σ are the discrete-to-moments, moments-to-discrete, and scattering operators, respectively, and σ_t is the total cross section. We assume that Eq. (1) has been discretized in the angular variable by the method of discrete ordinates (S$_n$) and that some spatial discretization method has also been applied. In order to solve Eq. (1a) it is necessary to numerically invert $[\mathbf{\Omega} \cdot \nabla + \sigma_t]$, the streaming-plus-collision operator. Although one could globally invert this operator by iterative methods, with first-order S$_n$ methods we may directly invert this operator by the method of sweeping, as described below.

In Figure 1(a) we depict a small unstructured mesh of triangular and quadrilateral elements; we have also depicted a direction $\mathbf{\Omega}$. In order for the streaming-plus-collision operator to be numerically inverted for $\mathbf{\Omega}$ for a given cell by the method of sweeping, the incoming fluxes for that cell must be known, i.e. the fluxes along cell faces for which $\mathbf{\Omega} \cdot \mathbf{n}$ is negative, where $\mathbf{n}$ is the outward unit normal on the face. These incoming fluxes are determined either from boundary conditions or from "upstream" cells previously solved by the sweep. For example, we cannot solve for $\psi^{(l+1)}(\mathbf{\Omega})$ in cell 5 until we have solved for $\psi^{(l+1)}(\mathbf{\Omega})$ in cell 6, since angular fluxes cross over the upper right face of cell 5 from cell 6. On the other hand, cell 6 can be solved before any other cells, since the only incoming fluxes it has along $\mathbf{\Omega}$ are from external boundaries, which we assume to be known.

The situation described above and depicted in Figure 1(a) induces the dependency graph shown in Figure 1(b). Each vertex in the graph represents the work to be done for direction $\mathbf{\Omega}$ for the corresponding cell, and the directed edges represent the dependencies between the cells. A vertex cannot be solved until all of its immediate predecessors have been solved. There is a dependency graph associated with every direction in the angular quadrature set. Since the angular fluxes in different directions are coupled only during the calculation of the scattering source, the sweep dependency graph for a direction is independent of those for other directions. We note that the graph in Figure 1(b) contains cyclic dependencies, such as between cells 8 and 9. Such cyclic dependencies have already been encountered in serial unstructured codes and have generally been handled by artificially removing one or more dependencies and using information from previous iterations. This procedure may affect the convergence rate of the source iteration process, but it does not affect the converged solution (Wareing, 1999, Wareing, 2000). Throughout the rest of our analysis we will assume that all cyclic dependencies have been removed.



(a) Unstructured mesh.



(b) Dependency graph for **Ω**.

Figure 1: Unstructured mesh and corresponding dependency graph for **Ω**.

For serial codes the ordering of tasks in the sweep is fairly straightforward. For parallel processing the situation is more complicated. We must distribute tasks among processors and order those tasks such that we simultaneously satisfy dependency constraints and efficiently utilize computer resources.

The above problem is a special case of the general class of scheduling problems. The scheduling problem is defined as the assignment of tasks to processors and the assignment of start times of tasks such that dependency constraints are satisfied and such that some objective is met (Gerasoulis, 1992, Kwok, 1999). The objective is usually to minimize the solution time of the graph. The general scheduling problem has been shown to be NP-complete (Garey, 1979), which means that except for special cases we cannot hope to construct an efficient algorithm to calculate an optimal schedule. Instead we must usually resort to heuristics to produce an acceptable, though suboptimal, schedule.

2.2 The KBA Scheduling Algorithm for Orthogonal Hexahedral Meshes

Numerous scheduling algorithms have been developed for special types of graphs. One such algorithm, the KBA algorithm, has been designed to schedule sweeps on orthogonal hexahedral meshes. Although it does not necessarily produce an optimal schedule, it does produce a schedule that yields nearly ideal parallel efficiencies (Koch, 1992, Baker, 1995, Baker, 1997, Baker, 1998).

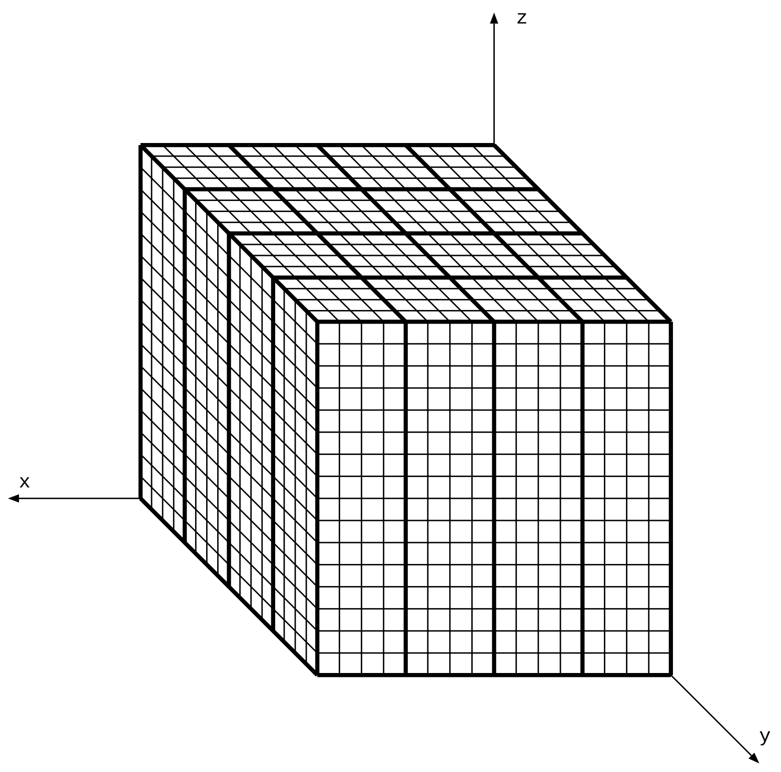


Figure 2: Orthogonal hexahedral mesh with KBA decomposition.

A typical orthogonal hexahedral mesh is depicted in Figure 2; element boundaries are indicated with thin lines. The mesh has I, J, and K elements along the x-, y-, and z-axis, respectively. We assume that I, J, and K are all $O(N)$. The KBA algorithm uses a spatial domain decomposition consisting of $P_x \times P_y \times 1$ domains, as indicated by bold lines in the figure; P_x and P_y are $O(N)$. Each domain has at most $I_C \times J_C \times K$ elements, where $I_C = [I/P_x]$, $J_C = [J/P_y]$, and $[\cdot]$ is the ceiling function. To simplify the rest of our discussion we will assume that $[I/P_x] = I/P_x$ and $[J/P_y] = J/P_y$. Note that KBA does not decompose in angle; the sweep work for any mesh cell for every direction in the quadrature set is assigned to the same processor.

For a given direction KBA orders the work as depicted in Figure 3. First the processor that has been assigned work at the top of the directed graph for that direction (in this case at the top front right corner) solves an $I_C \times J_C \times K_C$ block of elements, where K_C is $O(1)$ and $I_C J_C K_C$ is the block or "chunk" size. The ordering within the block is irrelevant, as long as it satisfies the dependency constraints. The solution of this block is indicated by its removal from the mesh in Figure 3(a). The processor then communicates newly computed partition boundary fluxes to the processors that have been assigned the partitions to the left and to the back of its partition. Note that it does not need to communicate data downward since all of these cells are in its own partition. In the next step, as indicated in Figure 3(b), both the original processor and the two to which it communicated solve another set of blocks and then communicate to the left and back. This continues as shown in the figure until most or all processors are performing computations at each step. Eventually one or more processors, starting with the original one, complete their work for that direction, and when all processors have finished we repeat the process for every other direction in the quadrature set in succession.

In order to describe the quality of the KBA schedules let us first define a few terms. The *parallel efficiency* (η) of a parallel algorithm is defined as the ratio of the serial CPU time and the total CPU time:

$$\eta = \frac{T_s}{PT_p}, \tag{2}$$

where T_s is the serial execution time, T_p is the parallel execution time, and P is the number of processors. The *parallel computational efficiency* (PCE) of an algorithm is the parallel efficiency in the absence of communication costs. The PCE is an upper bound on the parallel efficiency when communication costs are not negligible; a high PCE is a necessary, albeit insufficient, condition for high actual efficiencies. Finally, the *scalability* of an algorithm describes the trends in the efficiency as both the problem size and the number of processors increase; if the efficiency can be kept asymptotically fixed the parallel algorithm is said to scale (Kumar, 1994).

The exact efficiency of the KBA algorithm depends on the communication costs. If communication costs are zero, then the efficiency is given by (Koch, 1992, Baker, 1995, Baker, 1997, Baker, 1998)

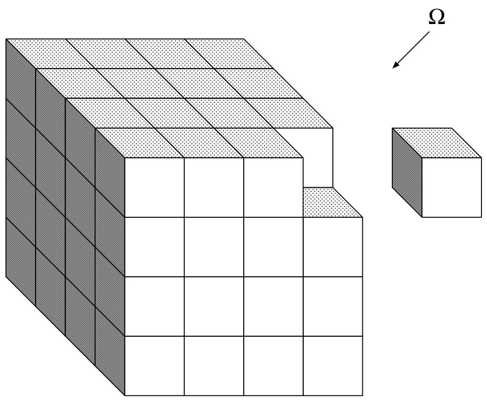
$$PCE = \frac{K}{K + K_C(I/I_C + J/J_C - 2)}. \tag{3}$$

Note that since I, J, and K are $O(N)$ and the other quantities in Eq. (3) are $O(1)$, the PCE is $O(1)$; the KBA scheduling algorithm scales with N. In the case of an asymptotically large cubic mesh with $I_C = J_C = K_C$ the PCE is 33%.

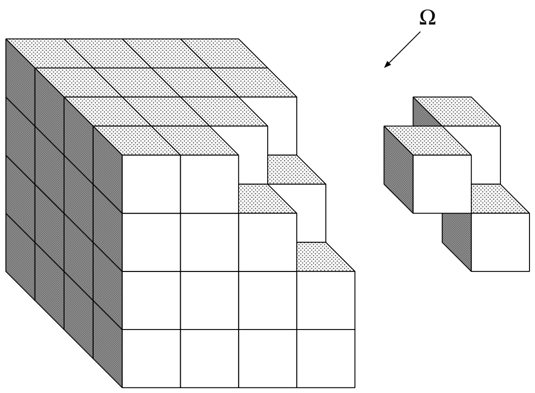
A variant of the simple KBA scheme presented above obtains much higher efficiencies by pipelining the work for several directions. In the ordering that we described previously a processor that had completed its work for a direction would wait for all processors to finish that direction before repeating the process for a new direction. In the alternative scheme when a processor completes all of the work for some direction, it can immediately begin work on the next direction in the same quadrant without waiting. With pipelining we are able to eliminate most of the processor idle time, and the efficiency becomes (Koch, 1992, Baker, 1995, Baker, 1997, Baker, 1998)

$$PCE = \frac{2MK}{2MK + K_C(I/I_C + J/J_C - 2)}, \tag{4}$$

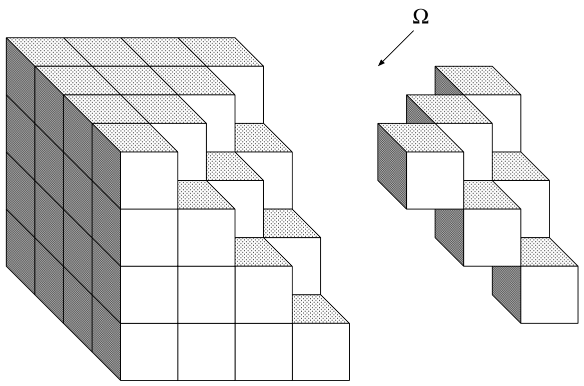
where M is the number of directions in an octant. For an S_8 level-symmetric quadrature on a large cubic mesh with $I_C = J_C = K_C$ the PCE is 91%.



(a) Step 1 of sweep.



(b) Step 2 of sweep.



(c) Step 3 of sweep.

Figure 3: KBA sweep ordering.

We have noted that KBA obtains scalable efficiencies, which can be nearly ideal in the pipelined variant. There are two main factors that contribute to KBA's success: the nature of its decomposition and its particular ordering strategy. Our analyses of structured meshes have shown that it is impossible to obtain a scalable schedule with $O(N^2)$ processors and a $P_x \times P_y \times P_z$ block decomposition unless one of the processor dimensions (say, P_z) is $O(1)$ and the other two are both $O(N)$; KBA satisfies this condition. Furthermore, the KBA ordering exploits the decomposition and the dependency graph in such a way that asymptotically most processors are occupied with useful work most of the time; the result is a scalable scheduling algorithm.

In summary, the parallelization of sweeps is a special case of a scheduling problem. Although optimal schedules are extremely difficult to obtain for general scheduling problems, many suboptimal scheduling algorithms have been developed for a variety of scheduling problems that often yield acceptable solutions. One specialized scheduling algorithm, the KBA algorithm, constructs sweep orderings on orthogonal hexahedral meshes that yield high parallel efficiencies; it relies on a particular spatial decomposition and a special ordering to obtain scalable schedules.

3 DEVELOPMENT OF A PARALLEL UNSTRUCTURED MESH SWEEP ALGORITHM

In this section we develop an algorithm suitable for scheduling parallel sweeps on unstructured meshes. We first show that existing scheduling algorithms are not suitable for this problem. We therefore construct a new list scheduling algorithm that is suitable for modest levels of parallelism. We also introduce several prioritization heuristics for use with the list scheduling algorithm.

3.1 Requirements for a Sweep Scheduling Algorithm

We have identified several properties that we believe a sweep scheduling algorithm should have. First, the algorithm should have low complexity, since we expect the sweep graphs to be quite large; typical problems may contain millions of individual tasks. Second, the algorithm should schedule on a set of processors that is very small in comparison to the number of tasks in the sweep graph. Finally, we would prefer an algorithm that distributes work in the spatial dimension only, since then we will not need to communicate during the calculation of the scattering source.

The above considerations effectively eliminate existing general scheduling algorithms from consideration as sweep schedulers. Such algorithms generally have time-complexity that is too high for sweep problems, they often assume an unreasonably large set of processors in comparison to the graph sizes that we expect, and it is difficult or impossible to impose the constraint of a purely spatial decomposition. We note, however, that the KBA algorithm satisfies the above constraints, although it can only be used with certain meshes. Given its success at achieving high efficiencies in these special cases, however, we would like to determine whether it could be generalized to unstructured meshes. Our goal in the rest of this section is to develop such a generalized sweep algorithm.

3.2 Overview of the New Algorithm

KBA scheduling may be viewed as two separate phases: a partitioning phase and an ordering phase. We will adopt this strategy since it simplifies the problem somewhat. Nevertheless, both subproblems are still challenging for unstructured meshes.

Our analysis of structured meshes shows that a specialized decomposition is necessary to obtain scalable sweep schedules; we assume that a similar specialized partition is needed for unstructured meshes. Unfortunately, the KBA decomposition (and any generalization to unstructured meshes) differs from the partitions produced by established or "traditional" partitioning approaches, which yield decompositions that are three-dimensional in character rather than columnar. Given the difficulty of constructing a specialized partitioning algorithm, we opt to strive for a more modest goal in this study. We will make use of established partitioning algorithms and develop special ordering algorithms that yield relatively high efficiencies on a small or intermediate (≤ 100) number of processors. It is our hope that the relatively small number of processors combined with a pipelining approach will yield relatively high efficiencies for most problems, even though we expect that the lack of a specialized partition will prevent scaling to thousands of processors.

Our approach for the ordering phase is to use a list scheduling heuristic that is a generalization in some sense of the KBA ordering. An outline of this algorithm is given in Figure 4. We first assign priorities to every cell-angle pair according to one of several heuristics we will describe later. We then initialize a priority queue for each processor with tasks that are at the top of the sweep graphs and that have been implicitly assigned to the processor by the partitioning phase. Next we enter an outer loop that repeats until all tasks have been completed. Within this loop we enter an inner loop in which each processor performs up to `maxCellsPerStep` highest-priority tasks (if any are ready to be computed); output data from each of these tasks may enable other tasks on the same processor. Finally all processors send and receive data (partition boundary fluxes) needed or generated by other processors and then the outer loop repeats.

The inner loop of the above algorithm is analogous to the solution of a block in KBA, with `maxCellsPerStep` being the chunk size. Like KBA, we do not explicitly account for communication costs; instead by increasing the value of `maxCellsPerStep` we may reduce communication costs by reducing the number of messages. By doing this, however, we also risk increasing the idle times of processors that are waiting for data.

We note that in Yang and Gerasoulis (1994) it is shown that the complexity of list scheduling with static priorities is $O(v \log(v))$, where v is the number of vertices

```
Assign priorities to every cell-angle pair
Place all initially ready tasks in priority queue
While (uncompleted tasks)
     For i=1,maxCellsPerStep
          Perform task at top of priority queue
          Place new on-processor tasks in queue
     Send new partition boundary data
     Receive new partition boundary data
     Place new tasks in queue
```

Figure 4: Pseudocode for list scheduling of sweeps.

in the graph. This will be the complexity of our overall algorithm if the prioritization heuristics are no more expensive. We believe such complexity is acceptably low, since the cost of each transport sweep is at least $O(v)$ and we generally will perform numerous identical sweeps during a transport calculation.

3.3 Prioritization Heuristics

The determining factor in the performance of the list scheduling algorithm is the assignment of priorities to tasks. The priorities represent the relative importance of completing one available task before some other available task. If the prioritization phase is done poorly we will obtain a low PCE, but if it is done well we may obtain a PCE that is near-optimal for the given decomposition and the selected value of `maxCellsPerStep`.

Although KBA does not make use of list scheduling, we can define a number of list scheduling prioritization heuristics for unstructured meshes that yield the KBA ordering (or a reasonable approximation to it) on structured meshes. Details about the heuristics that we have developed are available in Pautz (2000). In general, these heuristics attempt to generate partition boundary data as rapidly as possible in order to minimize processor idle time. We note that all of our heuristics can be implemented with algorithms with complexity equal to or less than $O(v+e)$, where e is the number of edges (cell faces) in the graph, so we have satisfied our requirement for low-complexity heuristics.

In summary, our sweep scheduling algorithm will initially use a traditional approach for generating a spatial domain decomposition. Although this limits our scalability, we hope to obtain high efficiencies on a modest number of processors. We will order the sweep tasks by means of a list scheduling algorithm. This algorithm will use one of several prioritization schemes we have developed to order the tasks; these prioritization heuristics are low-complexity.

Table 1: Meshes used in sweep studies.

Mesh	Number of elements	Description
adjalum	4097	Simple two-region box for regression testing.
nneut	43012	Neutron well-logging tool and surrounding media.
silc	51963	Computer chip and packaging for radiation shielding.
reac	165530	Reactor pressure vessel and surrounding cavity structures.
contest2	768	Cube divided into approximately equal-sized elements.
contest3	6140	Cube divided into approximately equal-sized elements.
contest4	32546	Cube divided into approximately equal-sized elements.
contest5	168356	Cube divided into approximately equal-sized elements.

4 ANALYSIS OF THE SWEEP SCHEDULING ALGORITHM

In this section we present theoretical performance estimates for our sweep scheduling algorithm. We will construct sweep schedules for several different meshes using the various prioritization heuristics that we have developed. We will evaluate these heuristics based on the PCEs they produce.

We will use tetrahedral meshes in our studies. The meshes we will use are described in Table 1. These meshes vary in size from several hundred elements to over 160,000 elements. They also vary in their structure. We will partition these meshes with Metis©, a "traditional" mesh partitioner that attempts to minimize the edge cut (the number of partition boundary faces) while preserving load balance (Karypis, 1998).

We present the PCEs resulting from list scheduling with different prioritization heuristics as a function of the number of processors in Figures 5 and 6 for the nneut and reac meshes, respectively. For these tests we use the S_8 level-symmetric quadrature and a value of 50 for `maxCellsPerStep`. There are several characteristics common to these and other studies we have conducted. The random heuristic (which uses random numbers for the priorities) results in noticeably worse schedules than the other heuristics, demonstrating the need for (and our ability to develop) intelligent prioritization schemes. The b-level heuristic (a simple heuristic used in some existing scheduling algorithms) also results in PCEs somewhat lower than the others we have developed, although it is still better than the random heuristic. The remaining heuristics (BFDS, DFDS, and DFHDS) yield very similar performance, with DFDS producing slightly better schedules than the other two. In general, for a small number of processors we can obtain schedules yielding nearly 100% efficiency; this efficiency gradually degrades to 80-90% as we increase the number of processors to over 100. Since we are not using specialized decompositions we expect continued degradation of the efficiency as we increase the number of processors further.

Other tests that we do not report here demonstrate that our algorithm is

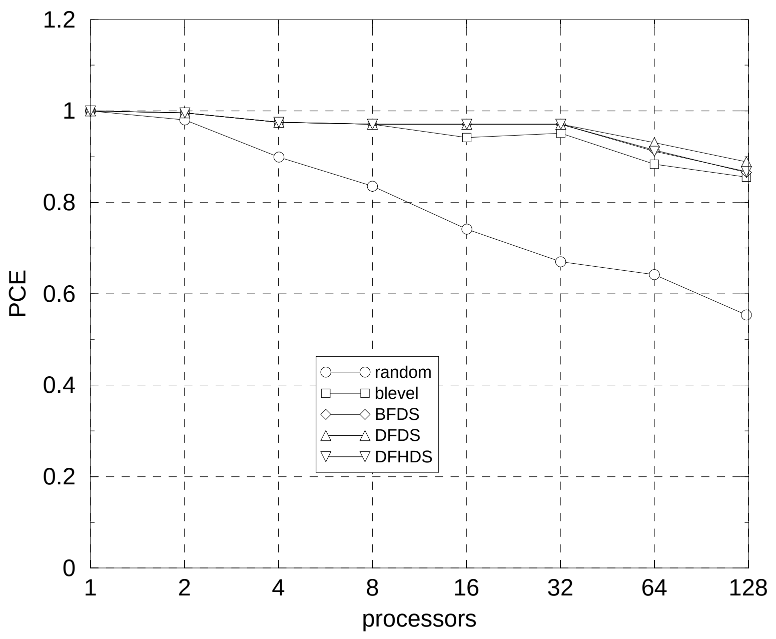


Figure 5: PCE vs. processors for nneut mesh (S_8 quadrature, `maxCellsPerStep` $=$ 50).

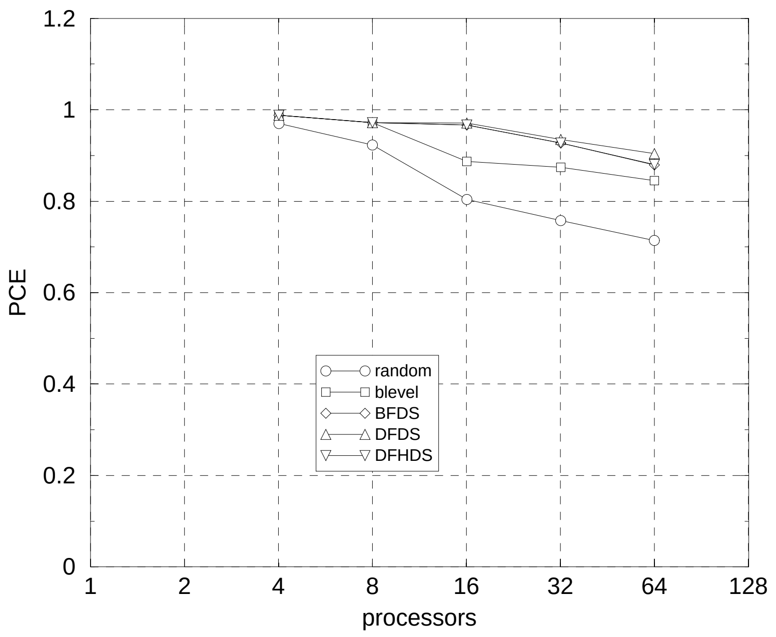


Figure 6: PCE vs. processors for reac mesh (S_8 quadrature, `maxCellsPerStep` $= 50$).

effective at pipelining work; increasing the quadrature order increases the PCE. We also find that we can use rather large values of `maxCellsPerStep` without significantly degrading the theoretical performance of the sweeps.

In summary, our theoretical analyses reveal that our new heuristics (BFDS, DFDS, and DFHDS) should perform quite well, yielding fairly high PCEs on a variety of meshes and with about 100 processors or less. The DFDS scheme consistently performs slightly better than the other schemes. Our algorithm is able to obtain some pipelining effect as quadrature orders are increased. Finally, we may use a wide range of values of `maxCellsPerStep` without significantly impacting the theoretical performance.

5 COMPUTATIONAL RESULTS

In this section we report actual run-time results for a parallel transport code that uses our sweep scheduling algorithm. We describe the major trends and compare them to the theoretical predictions from the previous section. Where the predictions differ from the computational results we offer possible explanations.

We have implemented our sweep scheduling algorithm in a new parallel S_n code, Tycho, which is under development at Los Alamos National Laboratory (LANL). Tycho currently uses linear discontinuous finite element differencing of the first-order form of the transport equation on tetrahedral elements. Our timing studies were conducted on Blue Mountain, a large cluster of SGI Origin2000 (O2K) computational servers at LANL. Each O2K, or "box", consists of 128 250 MHz R10000 processors organized as a non-uniform shared memory machine (Rafiei, 1998). We have elected for this study to share data among processors by means of explicit MPI calls rather than by exploiting the shared memory capability of the machine.

We present the measured parallel efficiencies for calculations on the nneut and reac meshes in Figures 7 and 8, respectively. These calculations use S_8 quadrature and `maxCellsPerStep = 50`; they correspond to the problems examined in Figures 5 and 6. There are general similarities between the PCEs we calculated and the measured efficiencies. The random heuristic yields noticeably lower efficiencies both in theory and in practice. We also observe relatively high efficiencies for a small number of processors and a reduction in efficiencies when much larger numbers of processors are used. There are also some noticeable differences between our theoretical predictions and these computational results. For a small number of processors we observe super-linear scaling effects, especially for large problems. This we attribute to the amount of memory required and the architecture of the O2K; memory allocated by a processor probably has greater locality as the number of processors is increased, since each processor is assigned a smaller part of the problem. For large numbers of processors the measured efficiency curves drop off more rapidly than the predicted efficiencies, probably because communication costs become increasingly severe as we increase the number of processors. Also, we observe that the b-level heuristic appears to yield efficiencies as good as or sometimes even better than our specialized heuristics, despite our predictions that it would produce slightly less optimal schedules. We do not yet understand this behavior, but we note that the variance in the run-time results seems to be larger than the difference between our predictions for these heuristics. Overall, though, it appears that our PCE calculations provide a reasonable guide to the performance characteristics of our parallel sweep scheduling heuristics.

Other timing studies that we do not report here demonstrate the pipelining effect that we predicted in the previous section. Our studies also show that increasing `maxCellsPerStep` can yield marked improvements in the run-time efficiencies, due presumably to reduced communication costs. Only when there is a substantial drop in the PCE as we increase `maxCellsPerStep` do we observe a drop in the run-time efficiency.

The final results we will show are scaling studies in the form of log-log timing results. In Figures 9 and 10 we show the CPU time per source iteration for calculations on "contest" and "irregular" meshes, respectively. We also show the CPU times that we would obtain with perfectly linear speedups. For these calculations we use S_8 quadrature, `maxCellsPerStep = 50`, and DFDS prioritizations. In these plots we observe almost linear speedups on up to 126 processors, but the changing slopes of the curves suggest that we will lose this scaling on even larger numbers of processors. Thus we see that our approach to partitioning and scheduling is sufficient for low or modest parallelism.

In summary, our performance predictions for our various sweep scheduling heuristics roughly correspond to actual run-time results. With our heuristics we can obtain high efficiencies for a small or modest number of processors, but with a larger number of processors the efficiencies drop off. Some of this loss of efficiency is due to the intrinsic nature of the sweep problem, but much of it in the cases we have examined is caused by the non-ideal aspects of our code and computer system. By increasing the value of `maxCellsPerStep` we can improve the run-time efficiencies, but excessive increases in this parameter eventually decrease the efficiencies. In general, our sweep scheduling algorithms may be used to obtain high speedups on about 100 or fewer processors.

6 CONCLUSIONS

We have developed a new algorithm for performing parallel sweeps on unstructured meshes. This algorithm uses a low-complexity list scheduling scheme with one of several prioritization heuristics we have developed to determine the parallel sweep ordering on a spatially decomposed unstructured mesh. With this new scheme in conjunction with traditional mesh decompositions we have obtained nearly linear speedups on up to 126 processors. This is an important step in the development of parallel first-order S_n codes.

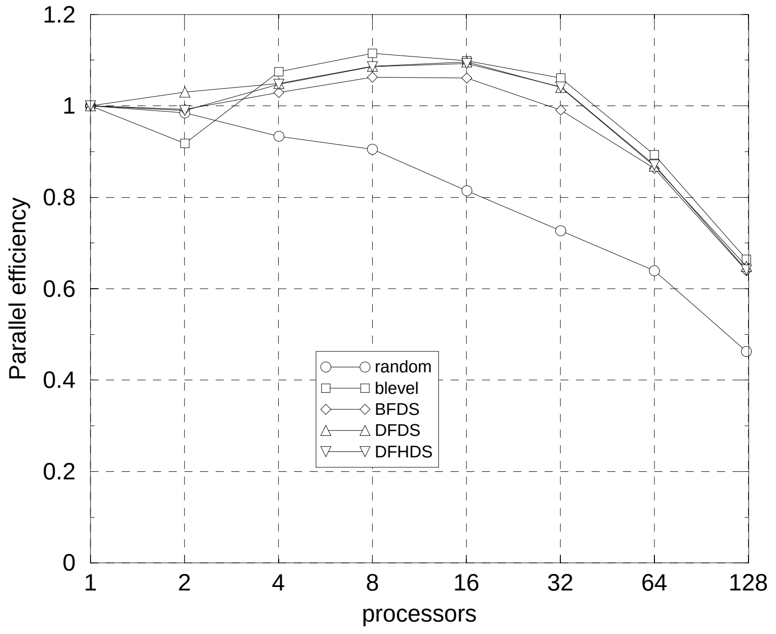


Figure 7: Parallel efficiency vs. processors for nneut mesh (S_8 quadrature, $\texttt{maxCellsPerStep} = 50$).

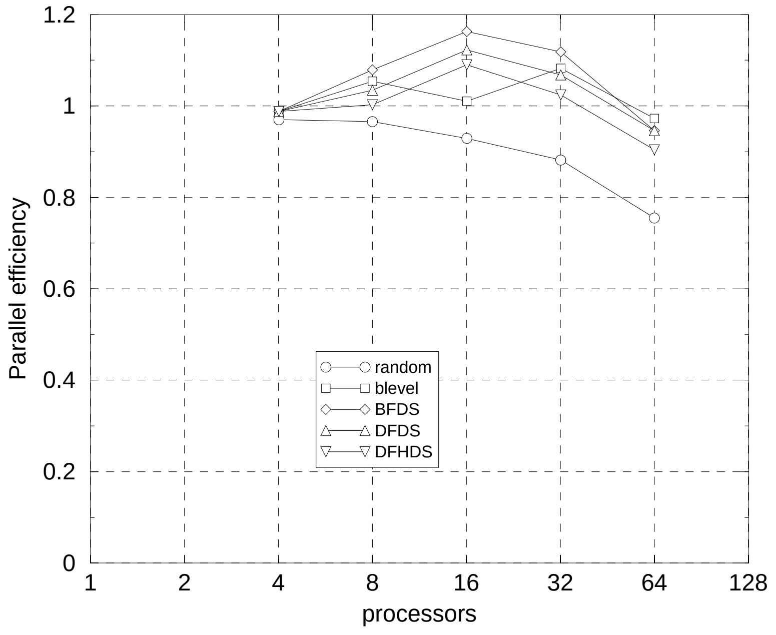


Figure 8: Parallel efficiency vs. processors for reac mesh (S_8 quadrature, $\texttt{maxCellsPerStep} = 50$).

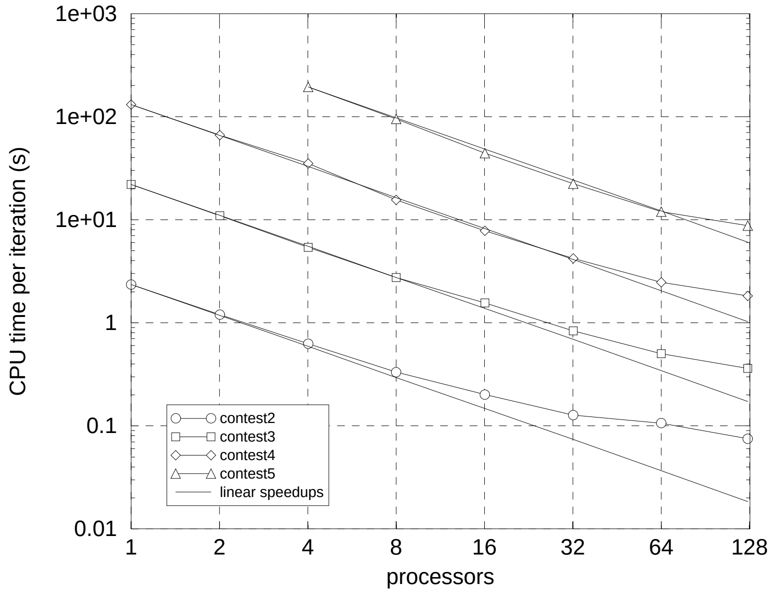


Figure 9: CPU time per iteration vs. processors for contest meshes (S_8 quadrature, `maxCellsPerStep` = 50, DFDS heuristic).

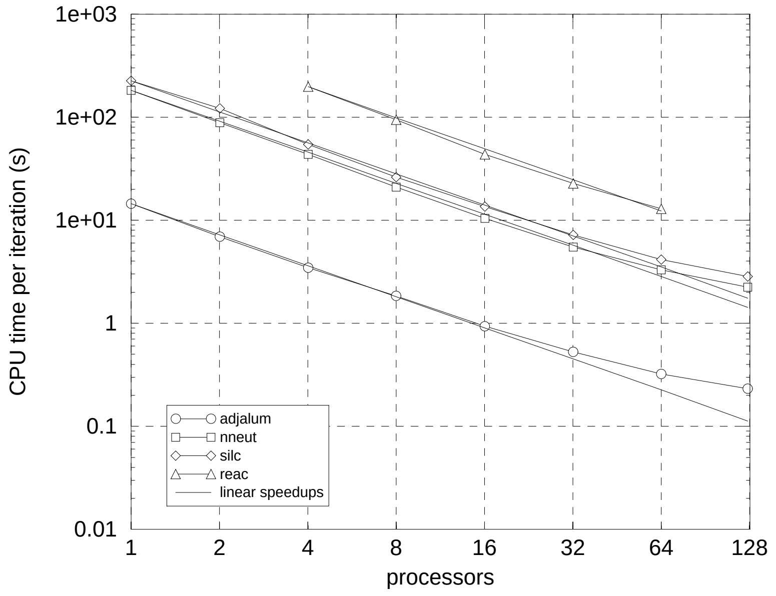


Figure 10: CPU time per iteration vs. processors for irregular meshes (S_8 quadrature, `maxCellsPerStep` = 50, DFDS heuristic).

Our ability to obtain high parallel efficiencies with traditional mesh partitionings is an important result. We have shown that it is impossible to obtain scalable schedules on structured meshes unless the decomposition has a two-dimensional nature; the KBA algorithm has this property. We believe that a similar constraint applies to unstructured meshes. Nevertheless, for typical quadrature orders and for modest levels of parallelism ($\leq$ 100 processors) we can obtain fairly high efficiencies; more processors are apparently needed to observe any severe asymptotic degradation in efficiency.

There are two factors that are key to our algorithm's ability to obtain high theoretical efficiencies for unstructured mesh sweeps. First, we have developed specialized prioritization heuristics that yield much better orderings than a random one; these heuristics attempt to generate and propagate data needed by other processors as rapidly as possible. Second, when the number of quadrature directions is comparable to or greater than the number of partitions we benefit from a pipelining effect. Both of these factors are generalizations of the approach that the KBA algorithm uses to determine the sweep ordering. A third factor, the use of a parameter that delays communications until a "sufficient" amount of computation has been performed, helps to produce run-time efficiencies that are close to the theoretical efficiencies. This too is a generalization of the KBA approach.

There are two main research areas in which we would like to see continuing work. First, we need to develop algorithms that can produce better spatial decompositions. Analysis of structured meshes leads us to believe that decompositions with a two-dimensional or columnar nature will be necessary in order to scale to thousands of processors or more. Second, continued development of prioritization heuristics may produce low-complexity ones that are even better than the ones we have developed. Work with other meshes and better decompositions may yield insights that assist in this process.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by the University of California Los Alamos National Laboratory under contract No. W-7405-Eng-36.

References

[1] Baker, R.S., Alcouffe, R.E., 1997. Parallel 3-D S_N Performance for DANTSYS/MPI on the Cray T3D. *Proc. Joint Int. Conf. Mathematical Methods and Supercomputing for Nuclear Applications*, Saratoga Springs, New York, October 5-9, 1997, Vol. 1, p. 377.

[2] Baker, R.S., Asano, C., Shirley, D.N., 1995. Implementation of the First-Order Form of the Three-Dimensional Discrete Ordinates Equations on a T3D. *Trans. Am. Nucl. Soc.*, **73**, 170.

[3] Baker, R.S., Koch, K.R., 1998. An S_n Algorithm for the Massively Parallel CM-200 Computer. *Nucl. Sci. Eng.*, **128**, 312.

[4] Garey, M.R., Johnson, D.S., 1979. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, California.

[5] Gerasoulis, A., Yang, T., 1992. A Comparison of Clustering Heuristics for Scheduling Directed Acyclic Graphs on Multiprocessors. *J. Parallel and Distributed Computing*, **16**, 276.

[6] Karypis, G., Kumar, V. METIS 4.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System. Technical Report, Department of Computer Science, University of Minnesota, 1998.

[7] Koch, K.R., Baker, R.S., Alcouffe, R.E. A Parallel Algorithm for 3D S_N Transport Sweeps, LA-CP-92-406, Los Alamos National Laboratory, 1992.

[8] Kwok, Y.K., Ahmad, I., 1999. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. *J. Parallel and Distributed Computing*, **59**, 381.

[9] Kumar, V., Grama, A., Gupta, A., Karypis, G., 1994. *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, Redwood City, California.

[10] Pautz, S.D. An Algorithm for Parallel S_n Sweeps on Unstructured Meshes, LA-UR-00-4672, Los Alamos National Laboratory, 2000.

[11] Rafiei, M. Origin2000 Application Development & Optimization, Technical Report O2KAPPL-1.0-6.2/3/4-S-SD-W, Silicon Graphics, Inc., 1998.

[12] Wareing, T.A., McGhee, J.M., Morel, J.E., Pautz, S.D., 1999. Discontinuous Finite Element S_N Methods on 3-D Unstructured Grids. *Proc. Int. Conf. Mathematics and Computations, Reactor Physics and Environmental Analysis in Nuclear Applications*, Madrid, Spain, September 27-30, 1999, Vol. 2, p. 1185.

[13] Wareing, T.A., McGhee, J.M., Morel, J.E., Pautz, S.D., 2000. Discontinuous Finite Element S_N Methods on 3-D Unstructured Grids, *Nucl. Sci. Eng.* (accepted for publication).

[14] Yang, T., Gerasoulis, A., 1994. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Trans. Parallel and Distributed Systems*, **5**, 951.