International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)
Saratoga Springs, New York, May 3-7, 2009, on CD-ROM, American Nuclear Society, LaGrange Park, IL (2009)

ANALYSIS OF MASSIVELY PARALLEL DISCRETE-ORDINATES TRANSPORT SWEEP ALGORITHMS WITH COLLISIONS

Teresa S. Bailey and Robert D. Falgout
Lawrence Livermore National Laboratory
P.O. Box 808, L-561
Livermore, CA 94551
bailey42@llnl.gov; rfalgout@llnl.gov

ABSTRACT

We present theoretical scaling models for a variety of discrete-ordinates sweep algorithms. In these models, we pay particular attention to the way each algorithm handles collisions. A collision is defined as a processor having multiple angles ready to be swept during one stage of the sweep. The models also take into account how subdomains are assigned to processors and how angles are grouped during the sweep. We describe a data driven algorithm that resolves collisions efficiently during the sweep as well as other algorithms that have been designed to avoid collisions completely. Our models are validated using the ARGES and AMTRAN transport codes. We then use the models to study and predict scaling trends in all of the sweep algorithms.

1. INTRODUCTION

A potential bottleneck when solving Boltzmann transport equations in parallel is the inversion of the streaming operator. The discretized form of this operator is a lower triangular matrix or block lower triangular matrix with small blocks. The solution of these triangular systems by direct methods involves an underlying sequential process that is inherent in the algorithm. Although various overloading techniques have been used to amortize the costs of these lower triangular solves or "sweeps", the practicality of scaling to massively parallel machines with tens of thousands of processors is unclear.

In this paper, we present new theoretical scaling models for several sweep algorithms, paying particular attention to what we refer to as "collisions", which occur when multiple angles have enough information to be swept at once in parallel. The existing literature [1-9] either ignores this important issue or overestimates its effect on the overall performance of the algorithm. In theory, these algorithms have the potential to scale like $O(dP^{1/d} + M)$, where d is the spatial dimension of the problem, M is the number of angles, and P is the number of processors. When M is fairly large, it masks the effect of the P term, whereby delaying the poor asymptotic scaling behavior. This delay may be adequate in some cases to get practical performance, even up to hundreds of thousands of processors. We also examine several processor overloading techniques that can greatly enhance performance, and discuss their advantages and disadvantages.

2. THE BASIC PARALLEL DISCRETE-ORDINATES SWEEP ALGORITHM

We consider the discrete-ordinates angular discretization of the mono-energetic Boltzmann transport equation on a structured spatial mesh of cells. The resulting linear system of equations is solved by applying an iterative method that involves "sweeping" the M discrete angles through

the spatial mesh during each iteration. Because the sweep is dependent on an angular direction, it has a sequential dependence, where the calculation of an angular flux at a given cell depends on the angular flux upstream. These sweeps are essentially wavefronts moving through the spatial mesh. In two dimensions, the sweeps have four blocks of wavefronts starting at each of the four corners of the mesh, with $M/4$ angles in each block. Similarly, in three dimensions the sweeps have eight blocks of wavefronts with $M/8$ angles in each block.

For simplicity, we parallelize only in space, so that each processor contributes to the sweeps of all angles (parallelizing in angle is trivial, and for the purposes of this paper, effectively only reduces the size of M). We assume a 3D spatial mesh of size $N = N_1 \times N_2 \times N_3$ distributed across a logical 3D processor grid of size $P = P_1 \times P_2 \times P_3$. The spatial mesh is divided into blocks of cells called *subdomains* that are arranged in a logical 3D grid of size $D = D_1 \times D_2 \times D_3$ and assigned to processors (hence $D_i \geq P_i$ for $i = 1,2,3$). For simplicity, we assume that each subdomain is the same size. The angles are grouped by quadrant (2D) or octant (3D). For the algorithms considered in this paper, each processor runs the following basic pseudo-code:

While not done sweeping M angles
1) Choose an angle and subdomain that are "ready to sweep" (if possible)
2) Compute the angular flux for that angle on that subdomain
3) Communicate boundary data with neighboring processors

End while

For the purposes of this paper, it is useful to imagine that each step of the above pseudo-code is synchronized to begin at the same time on all processors (in practice, this will most likely not be the case). We call an iteration of the while-loop a *stage* in the sweep algorithm. Note that a processor may not be busy at every stage. For example, during the beginning of a sweep, processors with cells in the middle of the domain do not have sufficient angular flux boundary data to do computations. In other words, there is nothing to do in steps 1 and 2 above. Boundary information eventually arrives in step 3 and these processors are able to contribute to the overall sweep calculation.

To increase efficiency, sweep algorithms often begin sweeping angles from all corners of the domain at the same time. However, this creates the potential for multiple angles to meet on one processor at the same stage during the sweep. We call this situation a *collision*, and note that it leads to a case where there are multiple angles to choose from in step 1. One of the main purposes of this paper is to study the effect of step 1 on the performance of sweep algorithms. As we will see later, it is possible to either dramatically degrade performance through poor choices or achieve optimal performance through good choices. We will also see that it is not necessary to develop algorithms that avoid collisions, in fact, it is better to allow them.

One technique for improving parallel performance is called *overloading*. Here, we define overloading as the ratio of subdomains (D) to processors (P) in our data distribution. Specifically, let ω_i (i=1,2,3) be *overloading factors* in each dimension and let $D_i = \omega_i P_i$. Each processor is assigned the same number of subdomains $\omega = \omega_1 \omega_2 \omega_3 = D/P$. We assign subdomains to processors as follows

$$(p_1, p_2, p_3) = (f_1(d_1), f_2(d_2), f_3(d_3)), \tag{1}$$

where (p_1, p_2, p_3) is a processor, (d_1, d_2, d_3) is a subdomain, and f_i are *distribution functions* for each dimension. Two examples of distribution functions are *blocked* and *round-robin* given by

$$p_i = f_{blocked}(d_i) \equiv \left\lfloor \frac{d_i - 1}{\omega_i} \right\rfloor + 1, \tag{2}$$

$$p_i = f_{robin}(d_i) \equiv \left((d_i - 1) \bmod p_i \right) + 1. \tag{3}$$

Processors also have an integer identifier p corresponding to a simple lexicographical ordering on the logical processor grid. Figure 1 illustrates the subdomain-to-processor assignment in 2D using the blocked and round-robin distributions together.

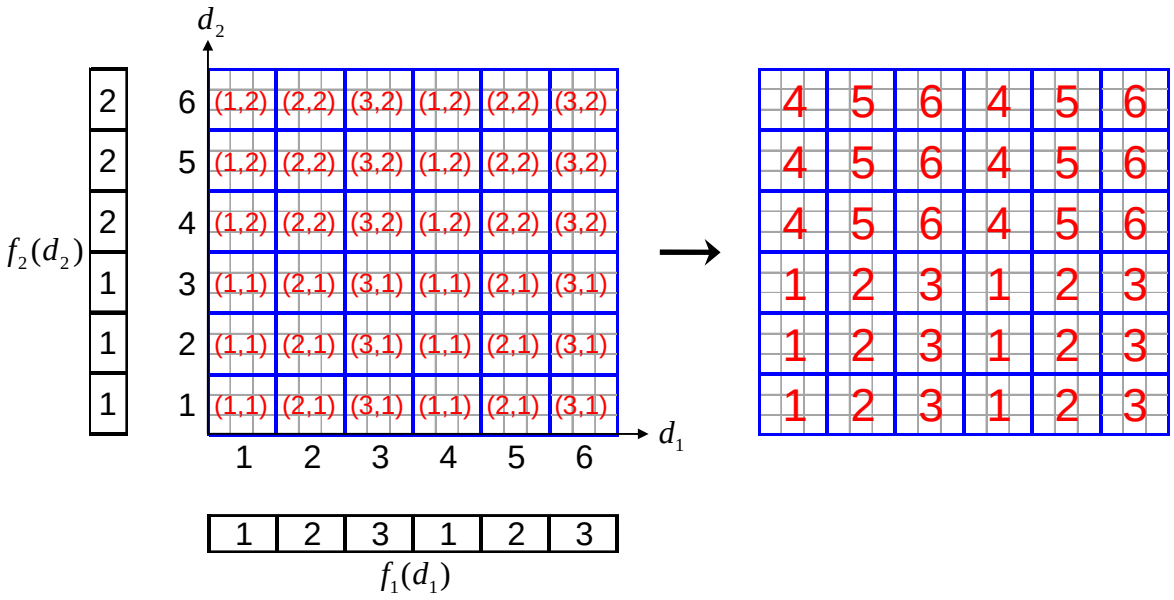

Figure 1: An example distribution of an 18×18 mesh (cells outlined in grey) on a 3×2 processor grid with overloading factors $\omega_1 = 2$ and $\omega_2 = 3$. The subdomain grid is 6×6 (outlined in blue) and each subdomain has 3×3 cells. The functions f_1 and f_2 are round-robin (3) and blocked (2) distribution functions, respectively. These functions determine the processor tuple assignments (p_1,p_2) for the subdomains (in red on the left), which become integer processor numbers (in red on the right).

3. ANALYSIS OF SWEEP ALGORITHMS

To compare the different sweep algorithms, we develop parallel performance models. The total time required to complete one sweep is assumed to have the following general form:

$$T = \begin{pmatrix} Number\ of \\ stages \end{pmatrix} \left(\frac{Computational\ Time}{Stage} + \frac{Communication\ Time}{Stage} + \frac{Latency\ Time}{Stage} \right). \tag{4}$$

The computational time for each stage is the product of the number of cells in a subdomain (Ω), the number of flops per cell-angle calculation (K_γ), the number of angles swept during the stage (m), and a machine dependent measure of the computation time per flop (γ):

$$\frac{Computational\ Time}{Stage} = \gamma m K_\gamma \Omega . \tag{5}$$

The communication time for each stage is the product of some fraction of the number of cells on the boundary of a subdomain (Γ), the number of angles swept during the stage (m), and a machine-dependent measure of the communication time per angular-flux quantity of data (β):

$$\frac{Communication\ Time}{Stage} = \beta m \Gamma . \tag{6}$$

Finally, the latency time is the product of the number of sends during the stage (K_α) and a machine-dependent quantity (α):

$$\frac{Latency\ Time}{Stage} = \alpha K_\alpha . \tag{7}$$

Putting it all together, we can rewrite (4) as

$$T = S\left(\gamma m K_\gamma \Omega + \beta m \Gamma + \alpha K_\alpha\right), \tag{8}$$

where S is the number of stages required to complete one sweep. Using (8), we are able to compare two sweep algorithms if we know the individual parameters. The most difficult of these parameters to determine is the number of stages. The other parameters are either dependent on the spatial decomposition, the angle grouping, or computer performance.

We can derive a lower bound for the number of stages in the basic algorithm. For simplicity, first consider the blocked distribution with no overloading. Each processor must wait a minimum number of stages s given by

$$s = \sum_{i=1}^{d} s_i, \quad 0 \le s_i \le \left\lfloor \frac{P_i + 1}{2} \right\rfloor - 1 . \tag{9}$$

Once a processor has boundary data, it takes a minimum of M stages to compute its part of the sweep and another minimum of s stages for the entire sweep to complete. For the overloaded case, the minimum delay in and out of a processor is still s, but each processor takes a minimum of ωM stages to compute its part of the sweep. Putting this together and maximizing over s_i, we see that the minimum number of stages is

$$S_{\min} = \left(\sum_{i=1}^{d} P_i - 2 + \left(P_i \bmod 2\right) \right) + \omega M . \tag{10}$$

One question of interest is whether there exists an algorithm that actually achieves the minimum in (10). For $\omega = 1$, it appears that the data driven algorithm described in Section 3.1 does

produce the minimal number of stages. However, for $\omega > 1$, it is clear that $S_{\min}$ is too small in general. For example, consider a 1D example with $\omega = 2$, $M = 2$, and P_1 even. Because there are D_1 subdomains that must be computed in sequence, there must be at least D_1 stages. But for $P_1 > 2$, $S_{\min} = P_1 + 2 < 2P_1 = D_1$. This can be remedied by the sharper lower bound

$$S_{\min 2} = \max\left\{S_{\min}, \left(\sum_{i=1}^{d} D_i - 1\right) + 1\right\}, \tag{11}$$

but it is still an open question as to whether this minimum can be attained in an algorithm.

As part of our analysis of the basic sweep algorithm above, we will consider three existing methods: a *data driven* algorithm [1,2,4], the well-known Koch-Baker-Alcouffe algorithm (KBA) [6], and the Compton-Clouse algorithm (CC) [3]. Each of these methods decomposes the domain in the manner described earlier, but with different choices of subdomain and processor grid layouts, and different strategies for angle selection in step 1. We provide a brief description of each algorithm and derive the necessary parameters required in (8). Note that the idea of deriving parallel performance models is not new, and detailed studies have already appeared for both a data driven algorithm and KBA [4,5,7,8]. What's new in this paper is the careful study of the issue of collisions, a new data driven algorithm that achieves the minimum given by (11), and a first attempt to model the CC algorithm.

3.1. The Data Driven Algorithm

We first consider a data driven algorithm similar to those in [1,2,4]. The data for this algorithm is distributed as in Figure 1. Only one angle is swept per stage, so that we have the following parameter values for (8):

$$\Omega_{DD} = \prod_{i=1}^{d} \frac{N_i}{D_i}; \quad \Gamma_{DD} = \Omega_{DD}\left(\sum_{i=1}^{d} \frac{D_i}{N_i}\right); \quad m_{DD} = 1 \ . \tag{12}$$

Sweeps begin in all four quadrants at the same time, and subsequent angles follow the first as soon as possible. As a result, the angles from each quadrant begin to collide once they reach the middle of the domain. At this point, processors can choose any of potentially several angles to sweep in step 1 of the basic algorithm. This choice can affect the total number of stages required to complete all of the sweeps, as we'll see later in Section 4. In this paper, we discuss three methods for choosing the next angle. The first method chooses by rank, where we give each angle a rank based on which corner it started in and choose the angle with the smallest rank during step 1 of the general algorithm. The second method randomly chooses an angle to sweep from the list of angles that are ready. The third method chooses the angle with the longest distance left to travel before exiting the domain. For the third method with $\omega = 1$ and a blocked distribution, we have shown experimentally that the number of stages is given by the minimum model in (10), i.e.,

$$S_{DD} = S_{\min} = \left(\sum_{i=1}^{d} P_i - 2 + (P_i \bmod 2)\right) + M \ . \tag{13}$$

3.2. The Koch-Baker-Alcouffe Algorithm (KBA)

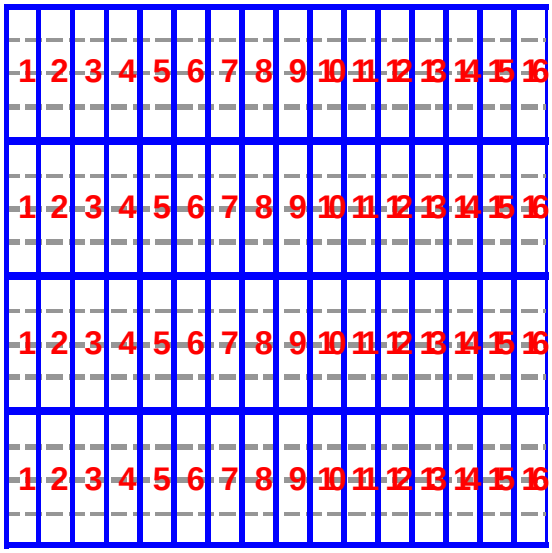

Figure 2: KBA spatial decomposition for a 16×16 cell mesh, a 16x1 processor grid, and a 16×4 subdomain grid. Four subdomains in each column of cells are assigned to each processor.

The processor layout for the KBA algorithm is always defined such that $P_d = 1$ and $P_i = D_i$, $i<d$. The processors are divided in the remaining dimensions as squarely as possible, so that each processor is assigned a column of $\omega = \omega_d = D_d$ subdomains. The distribution function is blocked. This is illustrated in Figure 2 for a 16×16 cell mesh and a 16×4 subdomain grid. As in the data driven case, only one angle is swept per stage, so that we have the following parameter values for (8):

$$\Omega_{KBA} = \prod_{i=1}^{d} \frac{N_i}{D_i}; \quad \Gamma_{KBA} = \Omega_{KBA}\left(\sum_{i=1}^{d-1} \frac{D_i}{N_i}\right); \quad m_{KBA} = 1. \tag{14}$$

For KBA, the decisions in step 1 of the basic algorithm are determined by a global schedule as follows. The sweep begins in the lower left corner, passing information to the neighboring processor after each subdomain is swept. When the sweep for an angle reaches the top of the column of subdomains on a processor, its mirror image angle is swept down the column. When this downward angle reaches the bottom, the next angle in the lower left corner begins in the same manner. The algorithm proceeds until all angles in the left half of the angle set have been swept, then the same procedure occurs for the angles in the right half of the set. Collisions are avoided by beginning the sweep in only one corner. The number of stages for this algorithm is

$$S_{KBA} = 2^{d-1}\left\{\left\lceil \sum_{i=1}^{d-1} P_i \right\rceil - 2^{d-2} + D_d\left(\frac{M}{2^{d-1}}\right)\right\}. \tag{15}$$

To see this in 2D, consider the rightmost processor (e.g., processor 16 in Figure 2). This processor must wait P-1 stages before doing any work. It then requires D_2 stages for each of the $M/2$ angles in the left half angle set. A similar analysis for the right half angle set gives the multiplier of two. A variant of KBA sweeps one angle per corner during each stage. This allows angles in all corners to begin immediately, but increases the amount of work being done and communicated at each stage.

KBA can be highly efficient when the size of the subdomains is small (overloading is high), because it quickly gets all of the processors busy doing work. However, this produces a large number of stages which can cause inefficiency due to increased message latency. In addition, the spatial decomposition of long thin columns of data is not ideal for most other numerical methods found in multi-physics simulation codes, where small surface-to-volume ratios are preferable. Also, the approach of decomposing only in d-1 dimensions limits scalability by requiring that $P \leq \prod_{i=1}^{d-1} N_i$.

3.3. The Compton-Clouse Algorithm (CC)

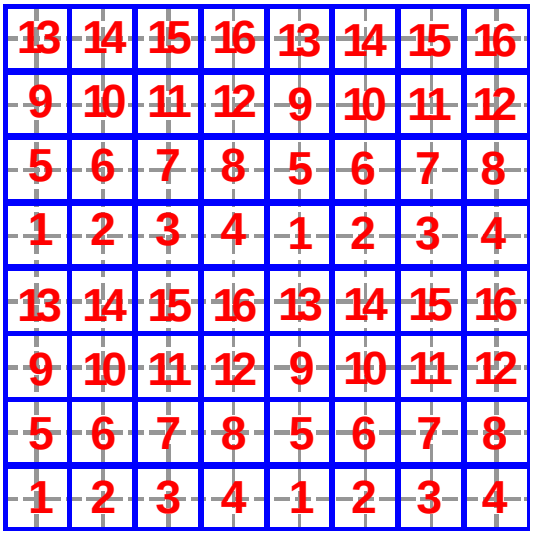

Figure 3: CC spatial decomposition for a 16x16 cell mesh, 4x4 processor grid, and 8x8 subdomain grid. Four subdomains are assigned to each processor.

The CC algorithm is a sweep algorithm designed for adaptive mesh refinement problems. Similar to KBA, CC attempts to get processors busy as quickly as possible by overloading. The subdomains are assigned to processors using the round-robin distribution function. This is illustrated in Figure 3 for $\omega_1 = \omega_2 = 2$ for a 16×16 cell mesh and a 4×4 processor grid. The CC algorithm requires that the overloading factors and number of processors in each dimension be powers of 2 to minimize (and sometimes even avoid) collisions. Furthermore, to allow for better load balance in the AMR setting, the CC spatial decomposition includes an adjacency criterion, which specifies the number of overlapping boundaries for a subdomain in a single dimension. We assume the adjacency criterion is zero in this paper.

For CC, the decision in step 1 of the basic algorithm follows a global schedule. This schedule chooses work based on the distance remaining for an angle to reach the end of its sweep. Angles from each corner are grouped and swept together before communicating with neighboring processors, and all corners are started at the first stage. This leads to the following parameter values for (8):

$$\Omega_{CC} = \prod_{i=1}^{d} \frac{N_i}{D_i}; \quad \Gamma_{CC} = \Omega_{CC} \left(\sum_{i=1}^{d} \frac{D_i}{N_i} \right); \quad m_{CC} = \frac{M}{2^d}. \tag{16}$$

Based on the way the schedule is built, the CC algorithm can be thought of as a special case of the data driven algorithm.

Because the CC algorithm is based on fixed schedules, it is difficult to define a general model for the number of stages. However, we can derive a theoretical minimum based on the amount of subdomain overloading and the adjacency criterion. This minimum is given by either the number of stages required to sweep the entire subdomain grid (first line in (17)), or the number of angle groups multiplied by the number of subdomains on each processor plus the maximum delay to start each processor (second line in (17)), whichever is largest:

$$S_{CC}^{(\min)} = \max \begin{cases} 2^d + \sum_{i=1}^{d} D_i - 2 \\ 2^d \omega - 1 + \sum_{i=1}^{d} \frac{P_i}{2} \end{cases}. \tag{17}$$

It is important to note that the number of stages in (17) is only achieved for a few special cases. When the subdomain overloading is large, the actual number of stages is typically much greater. For all numerical results, we use the actual number of stages for the CC algorithm. We also note that it is possible to develop a schedule where the angles from each corner are further divided into groups. However, we do not yet have an approach for building this schedule in the general setting. We anticipate that if a method was devised to break the angles into groups, it would further enhance the efficiency and scalability of the CC algorithm.

4. PERFORMANCE OF SWEEP ALGORITHMS

In this section, we first validate our models by comparing them to computational results. To find the number of stages to use in the models, we have developed a sweep emulator that determines the number of stages for the data driven algorithm and the CC algorithm. This emulator includes all three methods for choosing the next angle for the data driven case and can apply subdomain overloading in blocked and round robin patterns. It also can vary the quadrature set. We present model validation results for both the data driven and CC algorithms. In the second part of this section, we use the models to predict the behavior of the sweep algorithms as they scale up to large numbers of processors. In these predictions we again use the sweep emulator to generate the number of stages for the model given by (8). The number of stages for KBA can be found deterministically using (15). The predictions from the models are useful for finding scaling trends and to evaluate the effect of different parameter choices in discrete-ordinates sweep

algorithms. In this section, to normalize the results, we typically plot a parameter called "slowdown." Slowdown is defined as the run time for one sweep on one processor configuration divided by the run time for one sweep on a reference processor configuration.

4.1. Preliminary validation of the models using computational results

We have been able to show that computational performance of the CC method and the non-overloaded data driven method closely match the model predictions using a three-dimensional weak scaling test. In this test, N scales proportionally with P, hence the number of cells per processor is constant. Using (8) and the number of stages calculated by the sweep emulator for each of the algorithms, we can estimate how each algorithm will perform on a given machine.

To begin validating the model for the data driven algorithm, we have run a three-dimensional weak scaling problem using Atlas, a 2.4GHz AMD Opteron cluster (8 processors/node) at Lawrence Livermore National Laboratory (LLNL). This weak scaling problem used 125,000 cells per processor and an S_8 quadrature set (M=80). We ran the test problem up to 8000 processors using the Weighted Diamond Difference spatial discretization [10] in ARGES, a radiation transport code developed at LLNL. These runs use the rank approach for angle selection and compare two implementations of the algorithm. The results of the computational tests are plotted along with the model prediction in Figure 4.

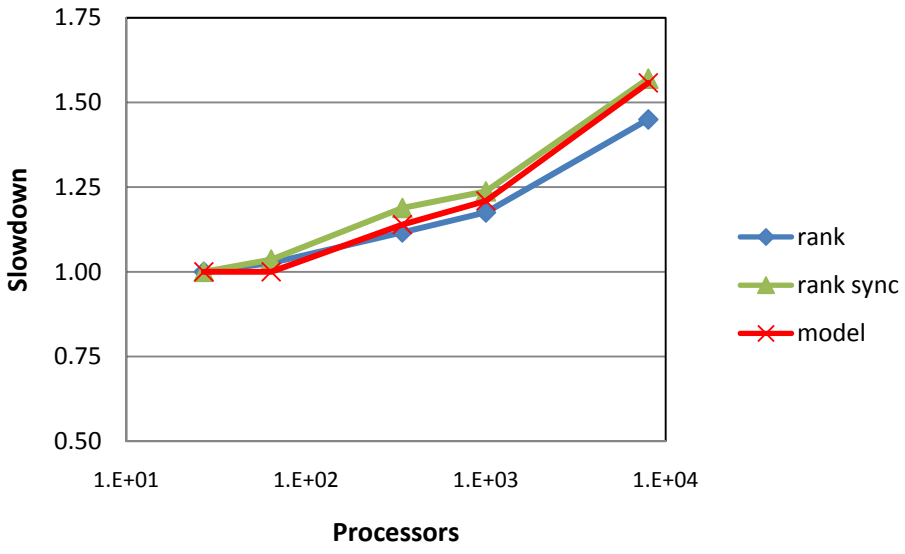

Figure 4: Computational results for the data driven algorithm. This algorithm was implemented in ARGES and run on Atlas. All slowdown curves are normalized to the 27 processor case.

The "rank" results are for an implementation that does not ensure that all boundary data has been received for a given stage. Hence, it is possible to take a different number of stages than predicted. The "rank sync" results are for an implementation that communicates with all 6 neighbors (instead of only 3) so that it behaves as predicted in the synchronous setting. Note that the slowdown calculation in this instance only uses the number of stages from the sweep

emulator. This is possible because the subdomain sizes in this weak scaling problem are always constant, so the terms in the parenthesis of (8) are constant for all processor configurations. We find that the computational results from ARGES match the model predictions well.

To compare the model predictions with computational results for the CC algorithm, we have run a three-dimensional weak scaling problem using AMTRAN, another parallel transport code developed at LLNL. The spatial discretization in AMTRAN is a Petrov-Galerkin continuous finite element method designed for block adaptive mesh refinement [3]. For these problems, we ran with uniformly spaced meshes. The weak scaling problem used 32,768 cells per processor and an S_{10} level symmetric quadrature set (M=120). We ran this problem up to 1024 processors of Hera, a 2.3GHz AMD Opteron linux cluster (16 processors/node) at LLNL. Because the subdomain size was not constant for all processor configurations, we needed machine parameters for the model. The necessary machine dependent data for Hera is

$$\begin{aligned} \alpha &= 3.03\, x\, 10^{-6} \text{ sec} \\ \beta &= 3.64\, x\, 10^{-9} \text{ sec/double} \\ \gamma &= 5.33\, x\, 10^{-9} \text{ sec/flop} \ . \end{aligned} \tag{18}$$

These parameters were obtained by timing standard “ping pong” message-passing tests and vector sums. The computational results of the weak scaling test for the CC algorithm are found in Figure 5. Again, the computation results match the model prediction well.

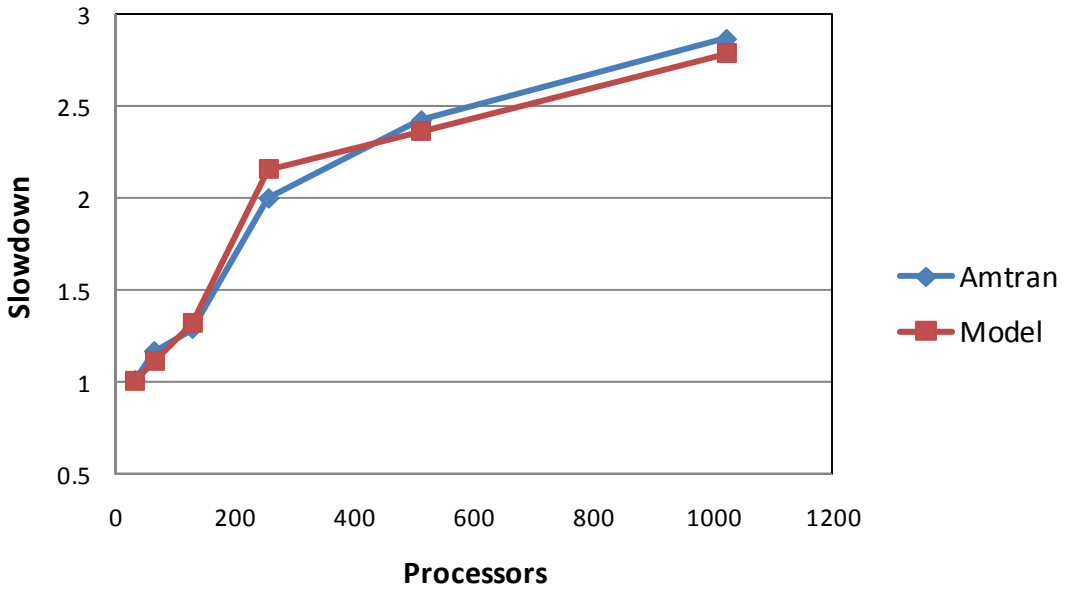

Figure 5: Computational results for the CC algorithm. This algorithm has been implemented in AMTRAN and run on Hera. The slowdown results are normalized to the 32 processor case.

4.2. Model Predictions for sweep algorithm scaling performance

The presented models are extremely valuable tools for studying the scalability of sweep algorithms. The first parameter we study is the different strategies for resolving collisions in the data driven algorithm. In particular, we looked at a random choice as well as the optimal one described in Section 3.1. For these cases we ran the sweep emulator with no overloading, and used an S_{10} quadrature set (M=120). Figure 6 shows that the random choice can produce significantly more stages than the optimal case for large processor counts in 3D. However, we also note that, although the choice of which angle to sweep can produce dramatically different collision patterns, most reasonable choices produce close to the optimal behavior.

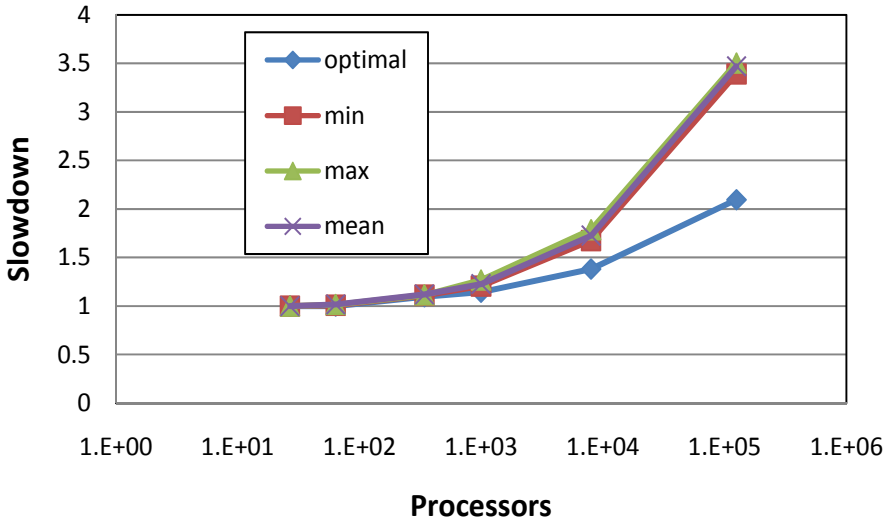

Figure 6: Faced with collisions, the choice of which angle to sweep next at a given stage grows in importance in 3D at large scale. Here we show the min, max, and mean number of stages for a random choice compared with the best case for the data driven algorithm.

We also use our models to study how sweep algorithms will scale as the number of angles is increased. Again, we use the sweep emulator and the non-overloaded data driven algorithm with the optimal-choice method. Figure 7 shows that increased angular discretization accuracy can further improve scaling, assuming that there is enough memory to store the necessary angle data on each processor. This figure also implies that ungrouping angles increases the scalability of a sweep algorithm.

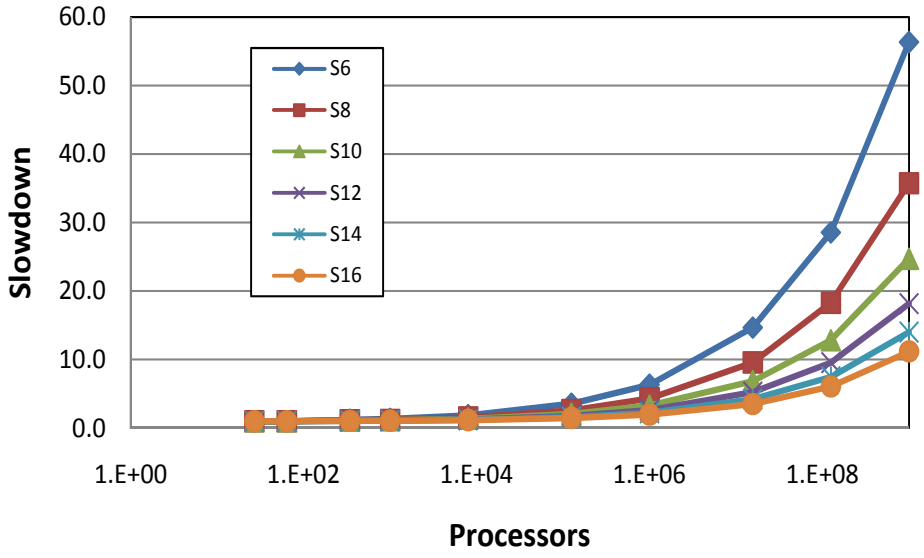

Figure 7: Sweeps can scale acceptably well to huge processor counts in 3D, and increased angular accuracy further improves scaling. Here we show the increase in the number of stages for the data driven algorithm relative to 27 processors.

Figure 6 and Figure 7 show that sweep algorithms can scale acceptably well to huge processor counts in 3D. We can enhance the scalability of an algorithm by running problems with more angles (or un-grouping the angles) and by making good choices in step 1 of the basic algorithm.

We next use the models to compare the behavior of all three sweep algorithms on the weak scaling problem, with 32,768 cells per processor and an S_{10} quadrature set (M=120). In this test, the data driven algorithm (DD) has no overloading, and KBA and CC have overloading factors of 64. Figure 8 shows the results of the models for each algorithm using the machine data for Hera and scaling from 8 to 8192 processors. We plot the time to completion for this case to show that all algorithms complete the sweep in the same order of time. Furthermore, these results demonstrate that all algorithms scale reasonably well in this range of processors. However, CC begins to lose its scaling at larger processor counts because, unlike DD and KBA, it computes all angles from a given corner together before communicating. The effect of this communication delay can be seen by the lack of an M term in (17), as opposed to the models for DD (13) and KBA (15). These terms mask the effect of the processor dependent terms, which effectively delays the poor asymptotic scaling behavior of the models. This effect is particularly pronounced in the case of KBA where the M term is multiplied by D_d because of subdomain overloading. If we can create a schedule for CC that un-groups the angles in each corner, we expect to see better performance for the CC algorithm on this test problem. We believe that it is possible to create this schedule, but have not proven this in practice.

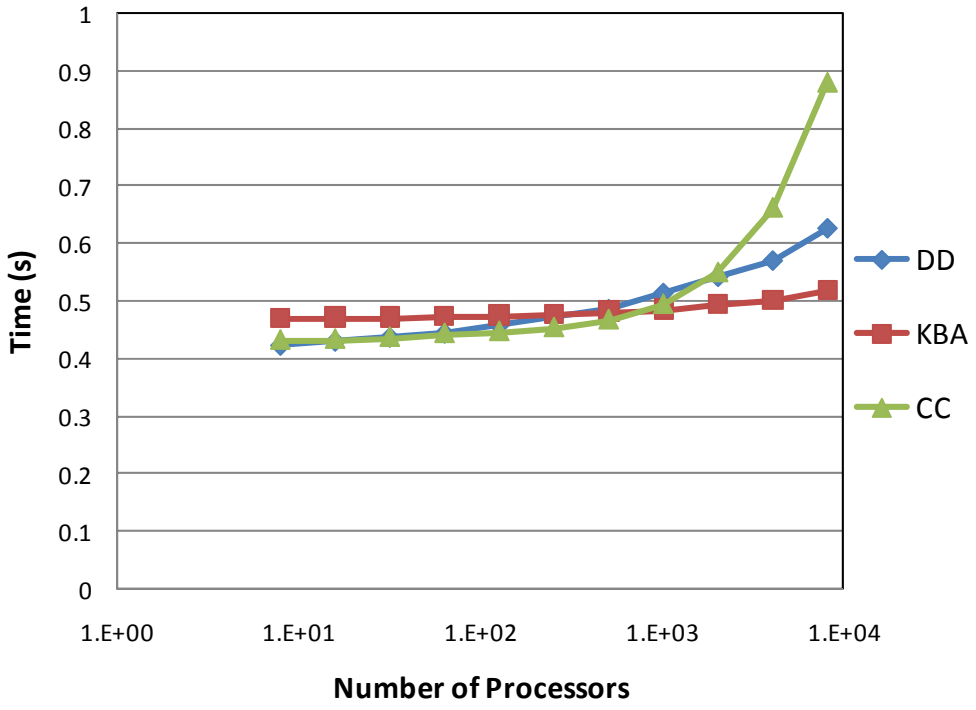

Figure 8: Model results of weak scaling, time to completion

We also use the sweep emulator code and models to study the effect of subdomain overloading on the scaling of the sweep algorithms. For this test, we used the optimal angle choice from the data driven algorithm and applied subdomain overloading of different amounts and patterns. In the figure the number after DD indicates ω for that case. For the first four cases in the figure, round robin overloading was applied and one angle was calculated per stage. The next two cases, called "KBA-like", indicate a blocked overloading pattern with P_3=1 (a distribution similar to KBA) and one angle swept per stage. The final case, the "CC-like" case, indicates a round robin overloading pattern (a distribution similar to CC) with all angles per corner swept per stage. For all cases in the plot, we used the optimal-choice method from the data driven algorithm for step 1 of the general algorithm. The results of this test are found in Figure 9. This plot indicates that subdomain overloading causes all algorithms to scale better. Even when all angles per corner are swept during a stage, subdomain overloading is an effective method to produce a sufficiently scalable algorithm.

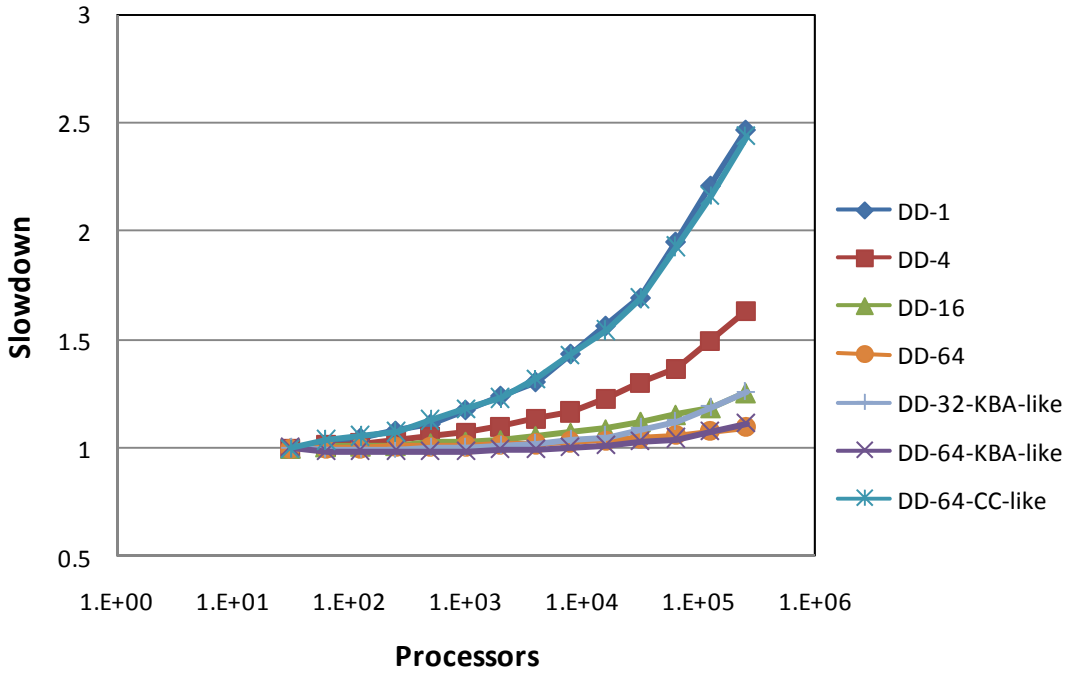

Figure 9: Slowdown for variations in subdomain overloading for the weak scaling problem with an S_{10} quadrature set. In general, as the amount of overloading increases, the algorithm scales better. Also, even if angles are grouped together, a sweep algorithm can scale reasonably well if subdomain overloading is applied.

5. CONCLUSION

We have developed models for analyzing the performance of massively parallel sweep algorithms applied to discrete-ordinates transport. This analysis compares the time to completion and slowdown of each algorithm, and is unique because we have been able to predict the behavior of sweep algorithms that allow collisions. We have applied this analysis to multiple sweep algorithms and compared the models with computational results. We used the models to make predictions about the general trends in performance of discrete-ordinates sweep algorithms. We demonstrated that sweeping small numbers of angles during each stage enhances the scalability of the algorithms. This results in the potential for collisions, but making reasonable choices about which angle to sweep next results in good performance. We also found that subdomain overloading further improves the scaling performance of the algorithms.

The results presented in this paper are preliminary. Much more analysis is required to optimize each of the parameters in the models to produce the best sweeping algorithm. We have not found an optimal pattern for overloading or been able to mathematically show that the furthest distance to travel is truly optimal. However, we have confidence that we can develop discrete-ordinates sweep algorithms with good scaling behavior in the massively parallel setting using the results from our analysis.

ACKNOWLEDGMENTS

The first author would like to thank John Compton for many helpful discussions about the CC algorithm and implementation issues. This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

REFERENCES

1. P. N. Brown, B. Chang, M. R. Dorr, U. R. Hanebutte, and C.S. Woodward, "Performing Three-Dimensional Neutral Particle Transport Calculations on Tera Scale Computers," *Proc. High Performance Computing '99*, April 11-15, San Diego, CA (1999).
2. P. N. Brown, B. Chang, U. R. Hanebutte, and C. S. Woodward, "The Quest for a High-Performance Boltzmann Solver," *Applications of High-Performance Computing in Engineering VI*, H. Power, M. Ingber, C. Brebbia, Eds., WIT Press, Southampton, UK, pp. 91-102 (2000).
3. J. C. Compton and C. J. Clouse, "Tiling Models for Spatial Decomposition in AMTRAN," *Proc. of Joint Russian-American Five-Laboratory Conference on Computational Mathematics/Physics*, Vienna, Austria, June 19-23 (2005).
4. M. R. Dorr and C. H. Still, "Concurrent Source Iteration in the Solution of Three-dimensional, Multigroup, Discrete Ordinates Neutron Transport," *Nucl. Sci. Eng.*, **122**(3), pp. 287-308 (1996).
5. A. Hoisie, O. Lubeck, and H.Wasserman, "Performance and scalability analysis of Teraflop-scale parallel architectures using multidimensional wavefront applications," *Int. J. of High Performance Computing Applications*, **14**(4), pp. 330-346 (2000).
6. K. R. Koch, R. S. Baker, and R. E. Alcouffe, "Solution of the First-Order Form of Three-Dimensional Discrete Ordinates Equations on a Massively Parallel Machine," *Trans. Am. Nucl. Society*, **65**, pp. 198-199 (1992).
7. M. M. Mathis, N. M. Amato, and M. L. Adams, "A general performance model for parallel sweeps on orthogonal grids for particle transport calculations," in *Proc. ACM Int. Conf. Supercomputing (ICS)*, pp. 255-263, Santa Fe, NM (2000).
8. M. M. Mathis and D. J. Kerbyson, "A General Performance Model of Structured and Unstructured Mesh Particle Transport Computations," *J. Supercomputing* , **34**, pp. 181-199, (2005).
9. S. D. Pautz, "An Algorithm for Parallel S_n Sweeps on Unstructured Meshes," *Nucl. Sci. and Eng.* **140**, 111-136 (2002).
10. E. E. Lewis and W. F. Miller, *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, Il (1993).