Efficient Massively Parallel Transport Sweeps

W. Daryl Hawkins[1], Timmie Smith[2], Michael P. Adams[1], Lawrence Rauchwerger[2], Nancy Amato[2], Marvin L. Adams[1]
[1]*Department of Nuclear Engineering;* [2]*Department of Computer Science and Computer Engineering*
Texas A&M University
College Station, TX 77843-3133[*]

INTRODUCTION

The full-domain "sweep," in which all angular fluxes in a problem are calculated given previous-iterate values only for the volumetric source, forms the foundation for many iterative methods that have desirable properties [1]. One important property is that iteration counts do not grow with mesh refinement [1]. The sweep solution on parallel machines is complicated by the dependency of a given cell on its upstream neighbors. A simple task dependence graph (TDG) for a single quadrature direction in a 2D example (Fig.1) illustrates the issue: tasks at a given level of the graph cannot be executed until some tasks finish on the previous level. The KBA algorithm [2] *partitions* the problem by assigning a column of cells to each processor, indicated by the four diagonal task groupings in Fig.1. KBA parallelizes over planes perpendicular to the sweep direction—over the breadth of the TDG. Early and late in a single-direction sweep, some processors are idle, as in stages 1-3 and 9-11 in Fig.1. In this example, parallel efficiency for an *isolated* single-direction sweep could be no better than $8/11 = 0.73$. KBA is much better, because when a processor finishes its tasks for the first direction it begins its tasks for the next direction in the octant-pair with the same sweep ordering. That is, each processor begins a new TDG as soon as it completes its work on the previous TDG, until all directions in the octant-pair finish. This effectively lengthens the "pipe" and increases efficiency. If there were $2M$ directions in the octant pair, then the pipe length is $2M \times 8$ in this example, and the efficiency could be up to $(2M \times 8)/(3 + 2M \times 8)$.

KBA's pipe-fill penalty grows as processor count grows, even if cell count grows proportionally. The width of the TDG grows only as $P^{2/3}$, so traditional KBA eventually runs out of parallelism to exploit. These issues fuel the common belief that sweeps cannot perform well in parallel beyond a few thousand processing elements. One purpose of this summary is to help dispel this belief.

A sweep algorithm is defined by its *partitioning* (dividing the domain among processors), *aggregation* (grouping cells, directions, and energy groups into "tasks"), and *scheduling* (choosing which task to execute if more than one is available). The work presented here follows that of Bailey and Falgout, who theoretically and computationally evaluated the performance of three sweep algorithms including KBA [3]. Their "data-driven" schedule appeared to be *optimal*—executing the sweep in the minimum possible number of stages—for tested partitionings and aggregations, but they were unable to prove this mathematically and they did not attempt to optimize across possible partitionings and aggregations.

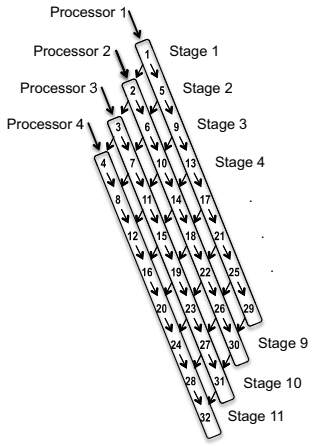

FIG. 1. Example TDG for a single direction's sweep on an XY rectangular grid. Each of four columns of cells is assigned to one processor, and each column is divided into eight tasks. Tasks on a given level of the graph can be executed in parallel.

Here we consider 3D Cartesian grids of $N_x \times N_y \times N_z$ spatial cells and simple $P_x \times P_y \times P_z$ partitioning, and we permit general aggregation of cells, directions, and energy groups. We have found a provably optimal family of scheduling algorithms and we present results from one of these. We exploit our guaranteed minimum stage count to further optimize sweep-execution time by choosing the best possible partitioning and aggregation parameters. Our results show excellent scaling out to 32,768 cores, significantly better than results previously reported for sweeps [4] and in line with the optimistic projections of [3].

PARALLEL SWEEPS

Consider a $P_x \times P_y \times P_z$ processor layout on a $N_x \times N_y \times N_z$ spatial grid, with integer values for all $\{N_u/P_u\}$ for simplicity. Suppose there are M quadrature directions per octant and G energy groups that

[*] dhawkins@tamu.edu; timmie@tamu.edu

can be swept simultaneously. Then each processor must perform $(N_z N_y N_z 8MG)/(P_z P_y P_z)$ cell-direction-group calculations. Aggregate these into tasks, with each task containing $A_x A_y A_z$ cells, A_m directions, and A_g groups. Then each processor must perform $N_{tasks} \equiv (N_z N_y N_z 8MG)/(A_x A_y A_z A_m A_g P_x P_y P_z)$ tasks. At each stage at least one processor computes a task and communicates to downstream neighbors. The complete sweep requires $N_{stages} = N_{tasks} + N_{idle}$ stages, where N_{idle} is the number of idle stages for each processor. Parallel sweep efficiency (serial time per unknown / parallel time per unknown per processor) is therefore approximately

$$\epsilon = \frac{T_{task} N_{tasks}}{[N_{stages}]\,[T_{task} + T_{comm}]} = \frac{1}{\left[1 + \frac{N_{idle}}{N_{tasks}}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]} , \quad (1)$$

where T_{task} is the time to compute one task and T_{comm} is the time to communicate after completing a task. In the second line, the term in the first [ ] is 1+ the pipe-fill penalty and the term in the second [ ] is 1+ the comm penalty. Aggregating for many small tasks (N_{tasks} large) minimizes pipe-fill penalty but increases the comm penalty: latency causes T_{comm}/T_{task} to increase as tasks become smaller. This assumes the most basic comm model, which can be refined to account for architectural realities (hierarchical networks, random variations, dedicated comm hardware, latency-hiding techniques, etc.).

Traditional KBA chooses $P_z = 1$, $A_m = 1$, $G = A_g = 1$, $A_x = N_x/P_x$, $A_y = N_y/P_y$, and $A_z =$ selectable number of z-planes to be aggregated into each task. If $K_z \equiv N_z/A_z$, each processor performs $N_{tasks} = 8MK_z$ tasks. With KBA, $2MK_z$ tasks (two octants) are "launched" from a given corner of the 2D processor layout. For any octant pair the far-corner processor remains idle for the first $P_x + P_y - 2$ stages, so a two-octant sweep completes in $2MK_z + P_x + P_y - 2$ stages. The other two-octant sweeps are similar, so if an octant-pair sweep does not begin until the previous pair's finishes, the full sweep requires $8MK_z + 4(P_x + P_y - 2)$ stages. The KBA parallel efficiency is then

$$\epsilon_{\rm KBA} = \frac{1}{\left[1 + \frac{4(P_x+P_y-2)}{8MK_z}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]} \quad (2)$$

KBA inspires our algorithms, but we do not force $P_z = 1$ or force aggregation factors to have particular values (such as $A_m = 1$), and we allow multiple octants to sweep simultaneously. In contrast to KBA, this requires a scheduling algorithm—a set of rules that tells each processor the order in which to execute tasks when more than one is available. Scheduling algorithms profoundly affect parallel performance, as was noted in [3].

The minimum possible number of stages for given partitioning parameters $\{P_i\}$ and aggregation factors $\{A_j\}$ is $2N_{fill} + N_{tasks}$, where N_{fill} is the minimum number of stages before a sweepfront can reach the center-most processors = number needed to finish a direction's sweep after the center-most processors have finished [3]. If $N_{zp} \equiv N_z/P_z$ and $K_z \equiv N_{zp}/A_z$ and we force $A_x = N_x/P_x$ and $A_y = N_y/P_y$, then

$$N_{fill} = \frac{1}{2}\left[P_x - r_x + P_y - r_y + K_z(P_z - r_z)\right] ,$$
$$N_{idle}^{min} = \left[P_x - r_x + P_y - r_y + K_z(P_z - r_z)\right] , \quad (3)$$

where $r_u \equiv 1(2)$ for P_u odd(even), and

$$N_{stages}^{min} = N_{idle}^{min} + (8MGN_{zp})/(A_m A_g A_z) , \quad (4)$$

Important observation: N_{idle}^{min} is lower for $P_z = 2$ than for the KBA choice of $P_z = 1$, for a given P. In both cases $P_z - r_z = 0$, but with $P_z = 2$, $P_x + P_y$ is lower.

OPTIMAL SWEEPS

It is not obvious that any schedule can achieve the lower bound of Eq. (4), because "collisions" of the $8M$ sweepfronts force processors to delay some fronts by working on others. Bailey and Falgout described a "data-driven" schedule that achieved this in limited testing, but they were unable to prove that it always would.

We have found a family of provably optimal scheduling algorithms: they are guaranteed to execute sweeps in the number of stages given by Eq. (4). (If ties are broken properly, the "data-driven" algorithm of [3] is a provably optimal scheduling algorithm.) We will describe in forthcoming communications the scheduling algorithms we have found, proofs of their optimal executions, and the implementation of one in our PDT code, which is built on the Standard Template Adaptive Parallel Library (STAPL) [5–7]. Here we describe how we have used our optimal *scheduling* algorithm to generate an optimal *sweep* algorithm, and we present weak-scaling results (constant work per core) from 1 to 32,768 cores on two different platforms.

Given an optimal schedule we know exactly how many stages a complete sweep will take, and Eq. (1) becomes

$$\epsilon_{opt} = \frac{1}{\left[1 + \frac{P_x - r_x + P_y - r_y + K_z(P_z - r_z)}{8MGN_{zp}/(A_m A_g A_z)}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]} . \quad (5)$$

Given Eq. (5) we can choose the $\{P_i\}$ and $\{A_j\}$ that maximize efficiency and thus minimize total sweep time. This optimization over $\{P_i\}$ and $\{A_j\}$, coupled with the scheduling algorithm that executes the sweep in N_{stages}^{min} stages, yields what we call an *optimal sweep* algorithm.

It is interesting to compare $\epsilon_{\rm KBA}$ to ϵ_{opt}, especially in the limit of large P (which allows us to ignore the r_u and 2 that appear in N_{idle}). In the large-P limit, with

$P_x + P_y \approx P^{1/2} + P^{1/2}$, Eq. (2) becomes

$$\epsilon_{\mathrm{KBA}} \rightarrow \frac{1}{\left[1 + \frac{4(2P^{1/2})}{8MN_z/A_z}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]} = \frac{1}{\left[1 + \frac{P^{1/2}}{MN_z/A_z^{\mathrm{KBA}}}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]} \quad (6)$$

Now consider ϵ_{opt} with $G = 1$, $P_z = 2$, and $P_x + P_y \approx 2(P/2)^{1/2}$. Because $P_z = 2$, $N_{zp} = N_z/2$. For comparison we aggregate to the same number of tasks as in KBA (which is likely sub-optimal) by setting $A_m = 1$ and $A_z = A_z^{\mathrm{KBA}}/2$. The result is

$$\epsilon_{opt} \rightarrow \frac{1}{\left[1 + \frac{\sqrt{2}P^{1/2}}{8MN_z/A_z^{\mathrm{KBA}}}\right]\left[1 + \frac{T_{comm}}{T_{task}}\right]}. \quad (7)$$

An interesting question is how many more processors the optimal schedule can use with the same efficiency as KBA. We see that with the chosen $\{P_i\}$ and (sub-optimal) $\{A_j\}$, the optimal algorithm produces the same efficiency as KBA with $(8/\sqrt{2})^2 = 32$ times as many processors. That is, even without optimizing the $\{A_j\}$, this scheduling algorithm yields the same efficiency on 132k cores as traditional KBA on 4k cores. Optimizing the $\{A_j\}$ improves this even more.

RESULTS

We consider two weak-scaling test suites introduced by Zerr and Azmy [4], each with one energy group (G=1) and an S_8 level-symmetric quadrature set (M=10). In each suite the number of cells/core is constant (4096 cells/core) as cells and cores increase. The cell and subdomain sizes also remain constant, because the volume of the problem domain is increased in proportion to core count. This kind of scaling may be relevant to high-fidelity reactor analysis, in which larger fractions of the full core can be analyzed as core count increases. The authors explored the performance of their parallel block-Jacobi (PBJ) iteration scheme on these problem suites. (If the spatial mesh were refined instead of the problem domain growing, then blocks would decrease in optical thickness. PBJ iteration counts—and thus solution time—would then grow substantially with core count.) In one suite, cell sizes are roughly 1 mean-free path and the scattering ratio is 0.9. In the other, cell sizes are roughly 10 mean-free paths and the scattering ratio is 0.99 [4].

Figure 2 shows the Zerr-Azmy PBJ solution times for the two suites along with sweep-based solution times from PDT, for core counts ranging from 1 to 32,768. All solution times are normalized to single-core values to focus attention on scaling and to mitigate the difficulty of comparing methods across different codes and different computers. In the h=1 problem, as core count increases

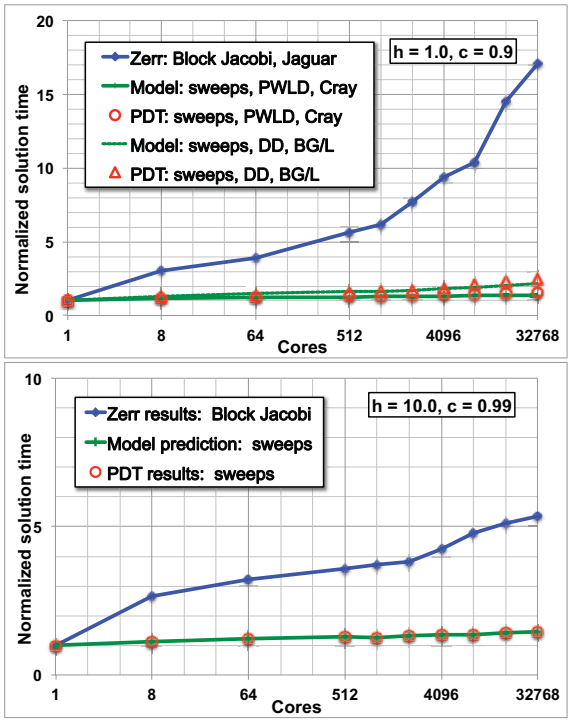

FIG. 2. Solution time normalized to single-core time, for the Zerr-Azmy PBJ method [4] and PDT's source iteration (sweeps), for Zerr-Azmy test problems. Top: 1 cm cell width; scattering ratio 0.9. Bottom: 10 cm cell width; scattering ratio 0.99.

to 32k the PBJ iteration count grows by a factor of 4 and time/iteration grows by approximately a factor of 4, for an overall slowdown factor of 17. In the h=10 problem the PBJ iteration count grows by a factor of 1.25 and time/iteration grows by approximately a factor of 4, for an overall slowdown factor of 5. It is not clear why the PBJ time per iteration increases by a factor of 4; we speculate that further work on the implementation would reduce this factor and improve the PBJ scaling.

PDT's sweep-based slowdown factors were 1.55 ($h = 1$) and 1.47 ($h = 10$) when using the piecewise-linear discontinuous (PWLD) FEM spatial discretization. A factor of 1.3 came from increasing time per sweep; the remaining small increases were from increasing iteration count as the problem domain grew. The PDT PWLD results were obtained on a Cray XE6 at the National Energy Research Scientific Computing Center, while the PBJ results were obtained on the Cray Jaguar machine at Oak Ridge National Lab. We also show PDT results from the h=1 problem using Diamond Differencing (DD) spatial discretization on the (very old) BlueGene/L machine at Lawrence Livermore National Lab. These results exhibit a slowdown factor of 2.5.

Zerr and Azmy showed KBA-sweep-based results from PARTISN on these problems, with slowdown factors ex-

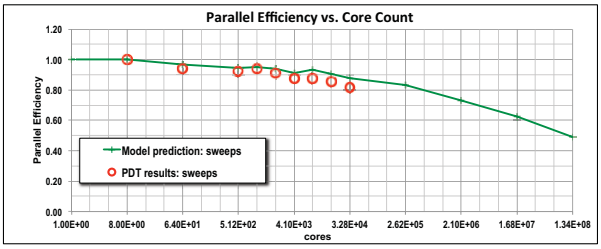

FIG. 3. Model sweep efficiency (solid line) and PDT data (circles) for the $h = 1$ Zerr-Azmy test problem.

ceeding 100 [4]. One reason for these poor results is that KBA is not as efficient as the "optimal sweep" algorithm, but this does not explain the 100× factor. We speculate that a major contributor to the 100× is that default values were used for parameter such as A_z, and these were not good choices for high core counts on these problems. The main point of this summary is that sweeps can be much, much better than previously published results would suggest, including those in [4].

PDT results agree well with the predictions of the model, Eq. (5), where we used $T_{comm} \approx 2(t_{latency} + t_{perbyte}N_{bpm})$. Here $t_{latency}$ and $t_{perbyte}$ are measured machine parameters, N_{bpm} is the number of bytes per message, and T_{task} is taken from the single-core result. Given our experience and understanding, the remaining disagreements suggest that there remain implementation inefficiencies whose removal will further improve the sweep results. The main discrepancy between model and code for PWLD on the Cray is that the code's sweep slows by 1.09× when going from 1 to 8 cores. There is no pipe-fill penalty at 8 cores and little comm cost, so this large slowdown remains a mystery. Beyond 8 cores the results and model agree more closely: the per-sweep slowdown is only 1.22× going from 8 cores to 32,768 cores. In the DD results from the BG/L, the sweep slowdown going from 1 to 8 cores was a remarkably high 1.27×, while the sweep slowdown from 8 to 32,768 was 1.47×.

There is significant interest in efficiently exploiting machines with 10^6 or more processing elements. (LLNL's recently acquired Sequoia computer has 1.6M cores.) Our results and model show that while sweeps degrade in efficiency as processor count grows, they degrade so slowly that factors other than pipe-fill are more likely to dominate performance. Figure 3 shows our model prediction out to 134M cores, along with our data to 32k cores from PWLD on the Cray machine, for the h=1 problem. Optimal sweeps may never achieve 50% efficiency on 134M cores, but it is clear that the pipe-fill penalty—the main criticism of parallel sweeps—is not a show-stopper even out to 10^8 cores. This is a direct consequence of Eq. (5), which is a consequence of optimal sweep scheduling. Our model predictions are consistent with those of [3].

DISCUSSION

Sweeps can be executed efficiently at high core counts. One key is an optimal scheduling algorithm that executes simultaneous multi-octant sweeps with the minimum possible idle time. Another is partitioning and aggregation factors that minimize total sweep time.

The analysis and results in this summary are for 3D Cartesian grids with "brick" cells. We are working on sweeps for AMR-brick grids, for nuclear-reactor grids that resolve pin geometries, and for arbitrary polyhedral-cell grids. For some grids there may be efficiency gains if processors are allowed to "own" non-contiguous cell-sets, an option considered in [3]. We are also working on hybrid parallelism in which compute *nodes* would own spatial subdomains and the cores on each node would share the work of executing the node's tasks.

For completeness we remark that reflecting boundaries introduce direction-to-direction dependencies that decrease available parallelism. This can be addressed either by iterating on the reflected angular fluxes or by accepting reduced parallelism. The best choice will be problem-dependent. Curvilinear coordinates introduce a different kind of direction-to-direction dependency, again reducing available parallelism and ultimately making sweeps somewhat less efficient than in Cartesian coordinates. We are working to quantify this.

ACKNOWLEDGEMENTS

Part of this work was funded under a collaborative research contract from Lawrence Livermore National Security, LLC. Part of this work was performed under the auspices of the Center for Radiative Shock Hydrodynamics at the University of Michigan, which is funded by the DOE NNSA ASC Predictive Science Academic Alliances Program. Part of this work was funded under a collaborative research contract from the Center for Exascale Simulation of Advanced Reactors (CESAR), a DOE ASCR project.

REFERENCES

[1] M. L. ADAMS and E. W. LARSEN, "Fast iterative methods for discrete-ordinates particle transport calculations," *Prog. Nucl. Energy*, **40**, No. 1, pp. 3-159, (2002).

[2] R. S. BAKER and K. R. KOCH, "An Sn Algorithm for the Massively Parallel CM-200 Computer," *Nucl. Sci. Eng.*, **128**, 312 (1998).

[3] T. S. BAILEY and R. D. FALGOUT, "Analysis Of Massively Parallel Discrete-Ordinates Transport Sweep Algorithms With Collisions," *Proc. International Conference on Mathematics, Computational Methods & Reactor Physics*, Saratoga Springs, May 3-7, CDROM (2009).

[4] R. J. ZERR and Y. Y. AZMY, "Solution of the Within-Group Multidimensional Discrete Ordinates Transport

Equations on Massively Parallel Architectures," *Trans. Amer. Nucl. Soc.*, **105**, 429 (2011).
[5] A. Buss, HARSHVARDAN, I. PAPADOPOULOS, O. PEARCE, T. SMITH, G. TANASE, N. THOMAS, X. Xu, M. BIANCO, N. M. AMATO, L. RAUCHWERGER, "STAPL: Standard Template Adaptive Parallel Library," *SYSTOR*, Haifa, Israel, June 4-6, 2010, ACM, pp.1–10, http://doi.acm.org/
[6] G. TANASE, A. BUSS, A. FIDEL, HARSHVARDAN, I. PAPADOPOULOS, O. PEARCE, T. SMITH, N. THOMAS, X. SU, N. MOURAD, J. VU, M. BIANCO, N. M. AMATO, L. RAUCHWERGER, "The STAPL Parallel Container Framework," *Proc. ACM SIGPLAN Symp. Prin. Prac. Par. Prog.* (PPOPP), (2011).
[7] A. BUSS, A. FIDEL, HARSHVARDAN, T. SMITH, G. TANASE, N. THOMAS, X. XU, M. BIANCO, N. M. AMATO, L. RAUCHWERGER, "The STAPL pView," *LCPC*, Houston, October 7-9, (2010).