An S_N Algorithm for Modern Architectures

Randal S. Baker

NUCLEAR SCIENCE AND ENGINEERING · VOLUME 185 · 107–116 · JANUARY 2017
© American Nuclear Society
DOI: http://dx.doi.org/10.13182/NSE15-124

An S_N Algorithm for Modern Architectures

Randal S. Baker*

Los Alamos National Laboratory, MS D409, Los Alamos, New Mexico 87545

Received December 7, 2015
Accepted for Publication March 6, 2016

Abstract — *Discrete ordinates transport packages from the Los Alamos National Laboratory are required to perform large computationally intensive time-dependent calculations on massively parallel architectures, where even a single such calculation may need many months to complete. While Koch-Baker-Alcouffe (KBA) methods scale well to very large numbers of compute nodes, we are limited by practical constraints on the number of such nodes we can actually apply to any given calculation. Instead, this paper describes a modified KBA algorithm that allows realization of the reductions in solution time offered by both the current and future architectural changes within a compute node.*

Keywords — *Neutron transport, discrete ordinates, KBA method.*

Note — *Some figures may be in color only in the electronic version.*

I. INTRODUCTION

The solution of the linear Boltzmann transport equation for radiation transport applications poses significant computational challenges due to its inherently seven-dimensional nature: three in space, three in velocity (i.e., two in direction and one in speed), and one in time. The use of the discrete ordinates or S_N algorithm for these solutions was initially not viewed as viable on the early massively parallel architectures of the 1990s due to the apparent sequential nature of the transport sweep that underlay the S_N algorithm. However, the development of the Koch-Baker-Alcouffe (KBA) method[1,2] overcame this difficulty, allowing S_N transport sweeps to be efficiently implemented on these types of massively parallel architectures. This method has subsequently become the basis for several heavily used transport codes, including the transport package PARTISN from the Los Alamos National Laboratory[3] (LANL).

The KBA method, essentially a wave-front method that results in a two-dimensional spatial-domain decomposition for three-dimensional geometries, has successfully provided parallelization for S_N transport sweeps as computing platforms have evolved from the SIMD (single instruction, multiple data) architectures of the early 1990s to today's large clusters of RISC (reduced instruction set computing)–based commercial compute nodes.[4] Indeed, with additional algorithmic modifications and extensions, it forms the basis for hybrid KBA domain-decomposition methods that allow transport sweeps to operate with reasonable parallel efficiency on computing platforms with core counts approaching one million.[5] However, while these KBA-like methods are necessary, they are not sufficient for LANL radiation transport applications for reasons discussed below.

At LANL, we are now transitioning into a computational realm where large three-dimensional time-dependent radiation transport calculations will be performed regularly. A single such calculation may require in excess of 50 million CPU-hours over the course of several months of calendar time, with memory requirements exceeding 50 Tbyte. In our experience, fabric scaling alone (i.e., increasing the node count until the job turnaround time is acceptable) will not be sufficient for these calculations since, as node count increases, power requirements (and, therefore, costs) increase linearly, while the mean time to job interrupt decreases super-linearly. The combination of these two effects limits our practical machine allocations for such calculations to no more than $O(10^3)$ nodes. It is, therefore, essential to meet job turnaround time requirements for our applications that per-node performance is maximized.

*E-mail: rsb@lanl.gov

Transport packages have painlessly benefited from the increase in CPU clock speeds over the last 20 years. However, with these increases now leveling off due to power constraints, improvements in per-node transport package performance must instead be obtained by taking advantage of the still-continuing increases in on-node transistor density. Vendors are using this increased transistor density to design and fabricate CPUs that offer potentially dramatic increases in computational performance, but only if we restructure transport algorithms originally targeted at the far simpler CPU architectures of the mid-1990s. Regardless of the details of the computational platform, e.g., many-core (Xeon Phi) or GPU, the fundamental redesign requirements necessary for increased performance on modern architectures are the same:

1. Maximize the number of independent work units.

2. Minimize the data movement necessary to operate on these work units.

The restructuring of transport packages to meet these requirements will then enable performance gains, regardless of whether we are talking in terms of vector lengths on a Xeon Phi or warps on a GPU, but this restructuring will also necessitate modifications to standard KBA algorithms.

In the remainder of our paper, we first review our standard and modified KBA algorithms for orthogonal meshes, then examine the details of numerical implementation and negative flux fixup for the modified KBA algorithm. We next explain the basis of our computational implementation, including theoretical efficiency, vectorization, threading, and Message Passing Interface (MPI) thread safety. We then compare the performance of our standard and modified KBA algorithms using a weak scaling study, exploring in more detail the impacts of vectorization on our results. We conclude by reiterating our view of the fundamental requirements for computational performance and the importance of satisfying them for LANL applications.

II. KBA ALGORITHMS ON ORTHOGONAL MESHES

The basic premise underlying KBA sweeps is, given an $(n-1)$–dimensional spatial-domain decomposition for n spatial dimensions, to stage the sweep work flow such that the necessary boundary information for the computations of a given domain is available from the upstream spatial domain as needed. The work flow sequencing to enable this is accomplished by breaking the remaining spatial dimension, as well as the additional dimensions in velocity space, into individual tasks that are solved in sequence for all spatial domains, but with a staged offset, as shown in Fig. 1.

For three spatial domains (A, B, and C), and particle flow from left to right, Fig. 1 shows the first two stages (steps) of a KBA sweep. As can also been seen from Fig. 1, spatial domains will sit idle until the wave front reaches them, and similarly, spatial domains will also sit idle once they have completed all of their tasks but downstream domains are still processing theirs. The percentage of stages a domain idles may be minimized by increasing the number of tasks in a work flow, i.e., pipeline length. We use the concept of parallel computational efficiency (PCE) to quantify this effect, where PCE is the ratio of useful work performed in a sweep (i.e., the number of phase-space cells solved), divided by the number of stages required to perform a sweep and the amount of work performed at each stage. Note that PCE is an upper bound on parallel efficiency since it excludes communication costs. In reality, the number of tasks must also be balanced with the compute/communication ratio, which decreases as the task size decreases.

II.A. Standard KBA Algorithm

Consider a $1 \times P_y \times P_z$ processor layout (we choose always to set P_x to one) for a $N_x \times N_y \times N_z$ spatial grid, with M angles per octant and G energy groups. Our standard KBA algorithm sequentially solves a single energy group at a time, with N_k spatial chunks of size N_x/A_x, where A_x is a user-selectable aggregation factor,[4] and $2M$ octant pair angles are used to fill the pipeline. Unlike the KBA algorithm described in Ref. 5, as the corner spatial processors, $(1, 1, P_z)$ and $(1, 1, 1)$, complete their task lists for the octant pairs initiated by the opposing corner spatial domains, $(1, P_y, P_z)$ and $(1, P_y, 1)$, respectively, they initiate their own octant pair sweeps immediately to reduce idle stages. The PCE for this algorithm is then

$$\varepsilon_{KBA0} = \frac{1}{1 + \dfrac{4(P_y - 1) + 2(P_z - 1)}{8MN_k}} . \quad (1)$$

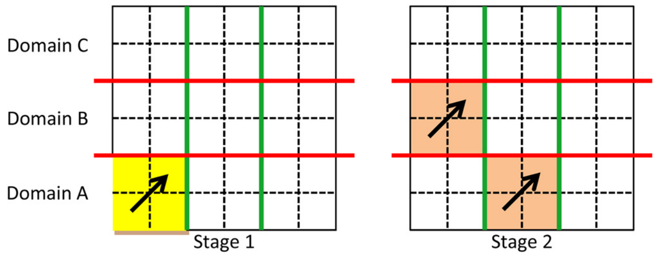

Fig. 1. Conceptual KBA work flow.

While further increases in PCE are available through additional overlapping of sweeps, we have not (yet) chosen to do so to preserve an intrinsic domain decomposition[6] in the presence of left, bottom, or front reflecting boundary conditions.

We also allow mixed domain decomposition, i.e., both energy and spatial-domain decomposition, within our standard KBA algorithm. Here, we use a $1 \times P_y \times P_z \times P_g$ processor layout, where P_g processors are used to decompose the energy groups. PCE is still determined by Eq. (1); the increase in PCE for this approach comes simply from the smaller values of P_y and P_z given a fixed number of processors. However, since we now decompose in energy as well as space, the downscatter term in the Boltzmann transport equation must be lagged during our iteration process, and the convergence criteria must also be appropriately modified. We refer to this treatment of the downscatter term as the point Jacobi in energy treatment, as opposed to the traditional Gauss-Seidel treatment, and will discuss it in greater detail in the next section. We note that we also allow use of the point Jacobi in energy treatment with a spatial-domain–only decomposition, and in that sense the use of mixed domain decomposition still represents an intrinsic decomposition.

II.B. Modified KBA Algorithm

This algorithm was originally developed for Roadrunner,[7] a multilevel heterogeneous computer at LANL, which, in 2008, was the first to achieve a petaflop. Since decommissioned, this system consisted of a standard massively parallel cluster of AMD Opterons combined with IBM Cell Broadband Engine accelerators. The transport solver in the S_N transport package PARTISN from LANL, written in FORTRAN-95, was extensively reworked over 2008 to 2009 to meet the two fundamental requirements for parallel performance identified above. Doing so allowed us to successfully make full use of the cell architecture on the Roadrunner platform, and has subsequently enabled a straightforward transition to many-core architectures such as the Intel Xeon Phi.

To maximize the number of independent work units, we have completely decoupled all phase-space dependencies for angles and energy groups during a transport sweep. That is, during the KBA sweep itself, in Cartesian geometries, the solution for any given angle or energy group no longer depends in any fashion upon any other angle or energy group (in *R-Z* geometries, angles are replaced with polar levels due to the angular derivative in the streaming term). Then, to minimize data movement for this algorithm within our FORTRAN implementation, we have reordered many arrays, such as those used for storage of angular fluxes or flux moments, so that their fastest algorithmically varying index is now leftmost. In other cases, we duplicated storage, such as that used for convergence controls or temporary sweep working arrays, so that they may be accessed simultaneously and independently across energy groups. The resulting increases in storage requirements (~10% to 20% in total application memory usage) from this are acceptable within the context of our time-dependent calculations.

II.B.1. Numerical Implementation

The overarching solution strategy employed with our modified KBA algorithm is identical to that of our standard KBA algorithm when the point Jacobi in energy treatment is used, and thus, results between the two are within machine round-off. However, the details of the actual transport solver implementations are entirely disparate. We illustrate the numerical implementation of our modified KBA transport solver below using, for simplicity, slab geometry with diamond difference in space and Crank-Nicholson in time. The subscripts n, i, g, and m refer to moment, spatial cell (width Δx_i), energy group, and angle, respectively. Flux moments (including scalar) are denoted by ϕ, angular fluxes by ψ, macroscopic cross sections by Σ, and group speeds by v_g. Within a time cycle (superscript n, width Δt^n), we retain our standard iteration strategy of performing inner iterations (superscript l) until the scalar fluxes converge, then embed these iterations within outer iterations (superscript k) that converge the fission, upscatter, and downscatter sources.

The outer iteration k starts by forming, and storing, the source Q to group g due to up- and downscattering from other groups, fission F, and external source S:

$$Q^k_{n,i,g} = \sum_{g' \neq g} \Sigma_{n,g' \to g} \phi^{k-1}_{n,i,g'} + F^{k-1}_{i,g} + S^n_{i,g} \; . \tag{2}$$

The computation of Q is referred to as zero-dimensional since the calculation for each spatial cell is completely independent from all other spatial cells. Furthermore, since we have assumed a point Jacobi in energy treatment for the downscatter term, the calculation of $Q_{i,g}$ is also now not dependent upon any other energy group.

An inner iteration l first adds the self-scattering source to Q, creating the total group source $\tilde{Q}_{i,g}$:

$$\tilde{Q}^{l,k}_{n,i,g} = \Sigma_{n,g \to g} \phi^{l-1,k}_{n,i,g} + Q^k_{n,i,g} \; . \tag{3}$$

Like Eq. (2) above, this is also a zero-dimensional calculation in both space and energy, and the result is stored for subsequent use in the sweep balance equation.

At LANL, we make heavy use of group-dependent quadrature sets, so group indices are used on the direction cosines $\mu_{m,g}$ and spherical harmonics $Y_{m,n,g}$ in the balance equation for cell i (and $\mu_{m,g} > 0$):

$$\begin{aligned}\phi^{l,k}_{n,i,g} &= \sum_m w_{m,g} Y_{m,n,g} \psi^l_{m,i,g}\\ &= \sum_m w_{m,g} Y_{m,n,g} \left\{ \left[\Sigma_{T,i,g} + \frac{2}{\mu_{m,g}\Delta x_i} + \frac{2}{\mathrm{v}_g \Delta t^n} \right]^{-1} \right.\\ &\quad \times \left[\sum_n Y_{m,n,g} \tilde{Q}^{l,k}_{n,i,g} + \frac{2}{\mu_{m,g}\Delta x_i} \psi^l_{m,i-1/2,g} \right.\\ &\quad \left.\left. + \frac{2}{\mathrm{v}_g \Delta t^n} \psi^{n-1/2}_{m,i,g} \right] \right\} ; \quad l = l + 1 \; . \qquad (4)\end{aligned}$$

We first form the time-centered angular source of a cell using the total group source from Eq. (3) above, then add to this the inflows from the upwind spatial cell and the time-edge initial condition. Next, we invert the streaming operator onto this sum to obtain the time-centered cell angular flux $\psi_{m,i,g}$. Finally, the spherical harmonics are applied to this angular flux to produce the scalar flux and flux moments ϕ.

While this equation is decoupled in angle m and energy g from all other cell balance equations, it is of course the spatial dependence on i that requires a sweep. Note that we have chosen to perform the moment-to-discrete and discrete-to-moment spherical harmonic operations explicitly within the cell balance equation, not separately outside of it. Although it is possible to do so, this would incur additional data movement costs for transferring the angular source into, and the angular flux out of, the cell balance solve. Instead, Eq. (4) requires only temporary storage for the time-centered angular source/flux. While the incoming time-edge angular flux still must be brought into the cell balance solve, it has been our experience that this is not a severe impediment, even on machines with accelerators like Roadrunner, as described above.

Once the inner iterations of Eqs. (3) and (4) have converged, we update the fission source using

$$F^k_{i,g} = \frac{1}{4\pi} \sum_{g'} \chi_{g' \to g} \upsilon \Sigma_{F,i,g'} \phi^k_{0,i,g'} ; \quad k = k + 1 \; , \qquad (5)$$

and return to Eq. (2) for the next outer iteration and continue until convergence.

In addition to our standard outer iteration convergence criteria, when using the point Jacobi in energy treatment, we use an additional check on the integral change in the downscatter source:

$$1 - \frac{\sum_g \sum_{g' < g-1} \sum_i \Sigma_{g' \to g,0} \phi^k_{0,i,g'} \Delta x_i}{\sum_g \sum_{g' < g-1} \sum_i \Sigma_{g' \to g,0} \phi^{k-1}_{0,i,g'} \Delta x_i} < \varepsilon \; . \qquad (6)$$

We have not performed a formal stability or convergence analysis on the use of the point Jacobi in energy treatment for the downscatter term in the transport equation. However, our observations to date have been that, as long as the downscatter term does not dominate the self-scatter, upscatter, and fission terms, we converge with only a small increase in iteration count compared to the traditional Gauss-Seidel downscatter treatment.

Storage requirements for our algorithm are dominated by the need, with standard Crank-Nicholson, to retain complete copies of the full angular flux at both the incoming and outgoing time edges. Normally, the incoming time-edge angular flux would then be used with the temporary time-centered angular flux of Eq. (4) to update immediately the time-edge outgoing angular flux in case we subsequently exit this cycle due to convergence of the outer iterations. Instead, we provide the option to allocate only storage for one copy of the time-edge angular flux. This copy is used to store the incoming time-edge angular flux, and is not updated during the solution of Eq. (4) until after cycle convergence. Then, an extra sweep is performed, during which the stored incoming time-edge angular flux is overwritten with the outgoing time-edge flux for use by the next time cycle.

Finally, nothing in the numerical implementation of our modified KBA algorithm precludes the use of standard acceleration techniques for the transport solution. In fact, within PARTISN we allow use of linear diffusion synthetic acceleration (DSA) on the inner source iterations and nonlinear DSA on the outer fission source iterations.[8]

II.B.2. Negative Flux Fixup

When using diamond difference in space or Crank-Nicholson in time, one can, of course, generate negative extrapolated edge fluxes. Our standard KBA algorithms have, thus, always incorporated the option for negative flux fixup. This is a multipass procedure in which, when a negative outgoing edge flux is detected in a phase-space cell, the procedure sets it to zero, recomputes the balance equation with that outgoing term defined as zero, checks again for any negative outgoing edge fluxes, etc. This results in a fixup procedure that can potentially vary for every phase-space cell in a nonlinear manner. While nothing in our modified KBA algorithm necessarily precludes the use of this procedure, to do so would reduce our ability to aggregate instructions (and, thus, data movement) to perhaps the worth of a single phase-space cell at a time.

Instead, suppressing the group index g, we recast the transport balance equation in terms of angularly dependent vectors for $\vec{\psi}$ and functions $\vec{H}$:

$$\frac{1}{v}\left(\frac{\vec{\psi}_i^{n+1/2}\vec{H}_{n+1/2} - \vec{\psi}_i^{n-1/2}}{\Delta t^n}\right) + \vec{\mu}\left(\frac{\vec{\psi}^n_{i+1/2}\vec{H}_{i+1/2} - \vec{\psi}^n_{i-1/2}}{\Delta x_i}\right) + \Sigma_{T,i}\vec{\psi}_i^n = \vec{S}_i^n \,, \quad (7)$$

and

$$\vec{H}_{n+1/2} = \begin{cases} 1, & \vec{\psi}_i^{n+1/2} \geq 0 \\ 0, & \vec{\psi}_i^{n+1/2} < 0 \end{cases} ; \quad \vec{H}_{i+1/2} = \begin{cases} 1, & \vec{\psi}^n_{i+1/2} \geq 0 \\ 0, & \vec{\psi}^n_{i+1/2} < 0 \end{cases} . \quad (8)$$

The transport balance equation in Eq. (7) (and its edge flux extrapolations) is solved iteratively using

$$\vec{H}^{(p)}_{i+1/2} = H\left(\vec{\psi}^{(p)}_{i+1/2}\right)\vec{H}^{(p-1)}_{i+1/2} \;;$$
$$\vec{H}^{(p)}_{n+1/2} = H\left(\vec{\psi}^{n+1/2,(p)}\right)\vec{H}^{(p-1)}_{n+1/2} \,, \quad (9)$$

where

$$p = \text{fixup pass iteration}$$
$$H = \text{Heaviside step function}$$
$$\vec{H}^{(0)}_{n+1/2} = \vec{H}^{(0)}_{i+1/2} = 1 \,.$$

Iteration is terminated when $\Sigma(\vec{H}^{(p)}) = \Sigma(\vec{H}^{(p-1)})$, with p bounded by the number of outgoing cell edges.

This procedure is mathematically consistent with the multipass fixup procedure used in our standard KBA algorithms, yet allows arbitrarily large aggregations in angular data and instructions.

II.C. Computational Implementation for Multicore and Many-Core Platforms

Despite being based on our past work for Roadrunner, nothing in the above discussion on the numerical implementation of our modified KBA dictates what type of architecture it is targeted for. Indeed, we argue that, since the fundamental principles of maximizing independent work units and minimizing data movement are applicable across all currently envisioned computational platforms, focusing redesign efforts at too early a stage on architectural details, such as vectorization or threading, unnecessarily obscures the underlying commonality of these principles. Once the underlying structure to implement these principles has been formed, however, it may then be used as the basis for targeting a specific architecture.

In our case this is Trinity, the next Advanced Technology System for the Accelerated Strategic Computing program at LANL. This system will have two different components, one partition consisting of >9500 Intel Haswell nodes, with two 16-core sockets per node, and a second partition composed of >9500 Intel Xeon Phi (Knights Landing) nodes, with 60+ cores per node. Obtaining good per-node performance on these architectures will require both effective use of the core vector units (length four 64-byte words on Haswell and eight on the Xeon Phi) and (at least on the Xeon Phi) their multithreading capabilities (2 per core on Haswell and 4 per core on the Xeon Phi). While the Haswell architecture will (probably) still permit the use of an MPI everywhere approach, i.e., one MPI rank per core, we, thus, define it as a multicore architecture. This will almost certainly not be the case for the Xeon Phi. Here, the desire to run multiple instruction streams per core to hide memory latency, combined with the increase in memory usage by message buffers as the number of on-node MPI ranks increases, leads us to an alternative MPI + X approach. In this approach, while there will still be at least one, and probably more, MPI ranks used per node, each MPI rank will consist of multiple threads running across multiple cores, and we, therefore, classify it as a many-core architecture.

These requirements have been incorporated into the instantiation of our modified KBA algorithm (hereafter referred to as the KBA1 algorithm, as opposed to our standard KBA0 algorithm described in Sec. II.A), targeted at many- and multicore architectures. The KBA1 implementation requires only ~3000 lines of additional source code within PARTISN (110 000+ lines of source code). The overall strategy of our implementation for these many- and multicore architectures may be summarized as follows:

1. spatial-domain decomposition using MPI and the KBA1 algorithm
2. vectorization over all M angles within an octant (polar levels in R-Z geometries)
3. threading via OpenMP over (at least) energy groups, with a point Jacobi treatment for the downscatter term in Eq. (2).

II.C.1. Parallel Computational Efficiency

While the KBA0 algorithm uses pipelining in angles to increase PCE, this is no longer possible with their use for vectorization in the KBA1 algorithm. From Eq. (1) above, we see that this reduces our PCE, but in practice the removal of angles from the KBA pipeline is counterbalanced by two effects. First, since each stage of our sweep now solves for an entire octant angles M within a spatial chunk N_k, a much smaller spatial aggregation factor A_x (which increases N_k) is necessary to provide task sizes with desirable compute/communication ratios. Second, since we have decoupled all group dependence within the cell balance solve of the KBA1 sweep in Eq. (4) above, we can now choose to pipeline energy groups during our sweeps for an octant pair. When T main-level threads per MPI rank are available, we employ them by solving A_g ($A_g \leq T$) energy groups in parallel during the sweep of an octant pair. With N_g energy group sets, defined by the ceiling function $\lceil G/A_g \rceil$, the KBA1 PCE is then

$$\varepsilon_{KBA1} = \frac{1}{1 + \dfrac{4(P_y - 1) + 2(P_z - 1)}{8N_g N_k}} . \qquad (10)$$

The PCE for the KBA1 algorithm could be further increased by subdividing the M angles of an octant into angular chunks, with these chunks then being used to increase the pipeline length.[5] While these might be beneficial for groups with large values of M, we have not yet chosen to do so as it also decreases vector lengths.

II.C.2. Vectorization Details

Three elements are necessary for effective vectorization within a transport package:

1. vector hardware on the computing platform
2. vectorizable loops and instructions within the transport application software
3. compiler generation of vector instructions for those loops.

Vector hardware has been widely available on high-performance computing systems for well over 5 years, with both increasing vector length and sophistication (i.e., the performance of fused multiply-add operations in a single vector instruction). The software restructuring performed for the KBA1 algorithm to maximize independent work units and minimize data movement readily allows the expression of vectorizable loops within the FORTRAN-95 language. While these loops are vectorized over angle within the cell balance equation, we also vectorize over space and/or moments in zero-dimensional areas of the transport solver.

Any effort spent on vectorization is wasted, however, without a compiler that recognizes vectorizable loops and then generates appropriate vector instructions. Our experience with the Intel compiler has been that it indeed does so for the vast majority of our loops, and without requiring additional use of compiler directives, as long as appropriate compiler options are selected. In our case, we have found these to be "–O3 –align array32byte –fp-model fast –fp-speculation fast –xHost." The combination of these elements has allowed us to obtain significant increases in on-node performance, as will be shown in the results.

II.C.3. Threading Details

We minimize the substantial costs of thread forks and joins by placing thread creation at as high a level as possible. We apply main level threads over energy group sets, where the number of group sets is less than or equal to the number of threads, then let each thread loop asynchronously operate over its assigned energy groups, minimizing thread scheduling costs. We also allow the use of nested threads within the main thread loop when sufficient work exists to justify their creation cost. In this case, each main thread immediately spawns off its nested threads, then uses them as needed to perform lower-level loop parallelism.

II.C.4. MPI Thread Safety

When using threads in conjunction with MPI, one must take into account the thread-safety level of the selected MPI library. The MPI standard defines four different levels of thread safety, but we allow the use of KBA1 transport sweeps and threading with only two of these: MPI_THREAD_MULTIPLE (multiple threads may access MPI without restriction) and MPI_THREAD_SERIALIZED (multiple threads may access MPI, but only in a serial fashion, never concurrently). While the use of MPI_THREAD_MULTIPLE would appear to be optimal from an algorithmic standpoint, our experience with current MPI implementations has shown otherwise. MPI_THREAD_MULTIPLE is not currently available from OpenMPI-based MPI libraries, and while it is offered by MPICH-based MPI libraries, these have fulfilled the necessary thread safety requirements through a global thread lock. Our testing has shown that the subsequent increase in costs for MPI calls due to the use of a global lock easily outweighs any potential algorithmic advantage for our sweeps.

Instead, by default, we select the thread safety level of MPI_THREAD_SERIALIZED during our MPI initialization process. This is not an issue for most of our transport solver when multiple threads are used since it is straightforward, for example, to aggregate convergence checks within a thread pool, then subsequently restrict the MPI broadcast of the results from those checks to only a single thread from the pool. Use of such an aggregation procedure during the cell balance solve of the sweep of Eq. (4) would unnecessarily require the idling of some threads within a pool until the entire pool is ready to transmit its data to downstream processors. Instead, we create a set of A_g thread locks, where each is set as locked except for the first, which is unlocked. During the sweep over cell balance equation solves, a thread computes the angular source for its spatial chunk, then attempts to set its lock and enter a blocking MPI receive for its upstream data. Only the first thread can immediately do so, of course. Once it has completed its MPI receive, it unsets the lock for the next thread in the thread pool, allowing it to proceed with its MPI receive, etc. Similarly, upon completion of the cell balance solve for a spatial chunk, a thread will wait until its lock is available, initiate a nonblocking MPI send of its downstream data, unset the lock for the next pool thread, then move on to its flux moment computations.

We have found these locks, which require only a thread safety level of MPI_THREAD_SERIALIZED, to provide surprisingly good MPI+X scaling with current MPI libraries. We suggest this is at least partly due to the actual overlapping of computations with communications. While the MPI standard permits this with nonblocking operations, current MPI implementations do not actually continue progression after a call returns until another, possibly unrelated MPI function, is entered. The use of multiple threads to (sequentially) initiate a series of nonblocking sends, however, ensures their continued progression even after an initiating thread has resumed computations.

III. RESULTS

The test problem we use for our weak study consists of a 100 × 1 × 1 cm block of uranium oxide with vacuum boundary conditions, 42 energy groups, and P_3 scattering. We use a $720 \times 2\sqrt{c} \times 2\sqrt{c}$ spatial mesh in our study, with c the number of cores, for 2880 spatial cells per core. Our test problem has a group-dependent quadrature set, where M_g ranges from a low of 8 to a maximum of 256, with a mean of ~53, for a total over all groups of 17 792 angles per spatial cell. A time-dependent simulation, using negative flux fixup, is run for five cycles. Only one copy of the time-edge angular fluxes is allocated within memory to minimize storage requirements. These cycles perform a total of 875 inner iterations (i.e., single-group transport sweeps) for all mesh sizes and algorithms, with the point Jacobi in energy treatment used for all algorithms. The use of Gauss-Seidel in energy for the KBA0/P_g = 1 algorithm would reduce its inner iteration count to 690, as shown in Table I, albeit at the expense of forgoing most of the potential performance gains available from vectorization. Note that the relative magnitude of the increase in inner iteration count with the point Jacobi algorithm is problem dependent, and may be substantially larger for other applications.

The PCE from Eqs. (1) and (10) for the standard (KBA0) and modified (KBA1) KBA algorithms, respectively, are shown in Fig. 2. The PCEs for the KBA0 algorithm are for one (P_g = 1) or four (P_g = 4) energy

TABLE I

Storage Requirements, Iteration Counts, and Solution Times for a 720 × 4 × 4 Spatial Mesh Run on Four Intel Haswell Cores Using Four MPI Ranks Over Spatial Domains

KBA algorithm	KBA0	KBA0	KBA1
Energy treatment	Gauss-Seidel	Point Jacobi	Point Jacobi
Storage requirements	1.627 Gbyte	1.685 Gbyte	1.968 Gbyte
Inner iterations	690	875	875
Solution time	56.063 s	73.588 s	20.229 s

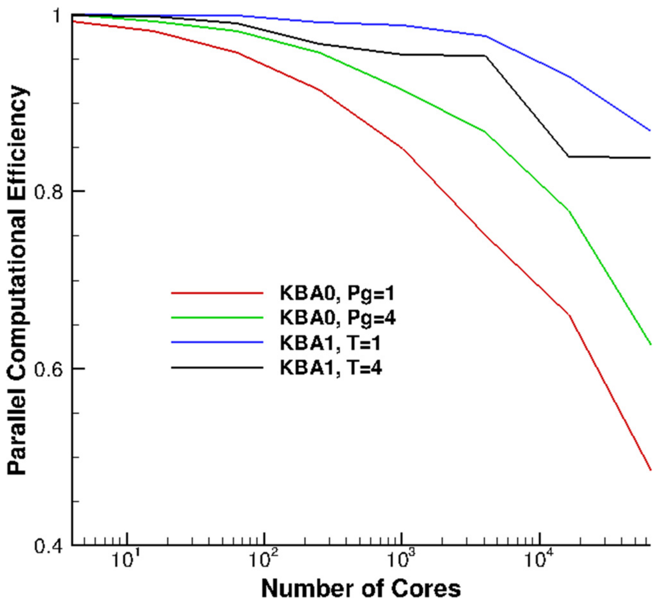

Fig. 2. PCE for weak scaling study.

domains, and for one ($T = 1$) or four ($T = 4$) threads for the KBA1 algorithm. The number of spatial chunks N_k used in the PCE computations is 5 or 6 for the KBA0 algorithm, and 30, 45, or 90 for the KBA1 algorithm. The value of N_k used for a given algorithm per core count was that which, via a parameter study, was found to minimize the actual solution time for that algorithm at that core count. Although our code has an internal algorithm to select this value, it is not (necessarily) always optimal. The larger PCE values for the KBA1 algorithm result from the value of $N_g \times N_k$ used in Eq. (10) being larger than the value of $M \times N_k$ used in Eq. (1) for the KBA0 algorithm, and are indicative of potentially superior scaling.

Weak scaling results were generated on Cielo and Luna, LANL machines with nodes containing two 8-core sockets of AMD Opteron 6100 Magny-Cours (M-C) or Intel Xeon E5-2670 Sandy Bridge (SB) processors, respectively. Although installed in 2010, and now nearing the end of their operational life, the M-C cores have vector units with a 16-byte length (two doubles), while the more recent SB cores have 32-byte vectors. The left ordinate in Fig. 3 is the total problem solution time (lines or dashes). The abscissa is presented in units of thermal design power (TDP), based on 115 W TDP per socket. Power usage and associated job interrupt times are the limiting factors for our calculations, not core counts. While TDP is the maximum design power, and actual power use will be reduced as the CPU idles, it also does not include the power required for subsystems such as disks and network interconnects, nor building cooling. A very rough estimate of this on our part leads us to believe, for Cielo, the actual total power usage will be approximately twice the TDP. Using this estimate, a calculation run on Cielo with 4096

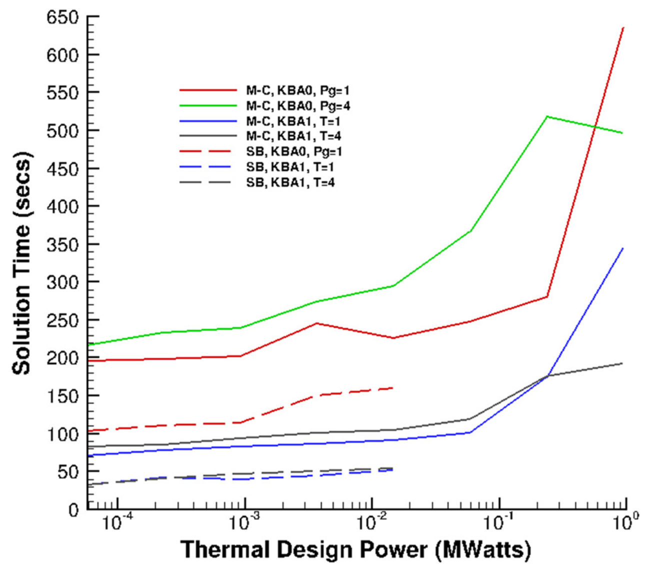

Fig. 3. Weak scaling study.

nodes (our rightmost data point) would actually require 1.8 MW. Additionally, at this size, our experience has shown the mean time to job interrupt will be, at most, a few dozen hours. Checkpoint and restart storage requirements for just the time-edge angular flux in the 4096-node test case, where the spatial mesh is $720 \times 512 \times 512$, are 24.4 Tbyte.

The per-node performance gains with the KBA1 algorithm are evident at the left side of Fig. 3. Here, with only a few nodes in use, the KBA1 algorithm outperforms the KBA0 algorithm by two to three times. While these performance gains diminish somewhat as we scale further, they increase once again as we reach large node counts, where the KBA1 algorithm is able to maintain higher PCE values than the KBA0 algorithm. While both algorithms benefit from reducing the number of spatial domains at large node counts, the KBA0 algorithm does so by decomposing in both space and energy ($P_g = 4$), requiring more MPI communication to synchronize flux moments, for instance. The use of threads ($T = 4$) within the KBA1 algorithm, or MPI+X, eliminates this necessity entirely, as all threads within an MPI rank share a common memory node with equal access costs. At small node counts, the increase in solution times from using MPI+X is either minimal, as seen on the Cielo M-C, or nonexistent, as seen on the Luna SBs. While the results were generated using a group-dependent quadrature set, we have repeated this study using an S_{12} quadrature set ($M = 21$) for all groups, with qualitatively similar results (including the spike in solution time seen in the M-C, KBA0, $P_g = 4$ curve). Finally, as mentioned in Sec. II.A, the PCE for our KBA0 algorithm could be increased at large node counts by, for instance, simultaneously starting sweeps from multiple octant pair corners.[5] Such increases in PCE are also available to the KBA1 algorithm, of course.

The components of the solution time for the KBA1 algorithm on M-C are shown in Fig. 4 for two points, 1 node (16 cores) and 4096 nodes (65 536 cores). The computations performed in Eq. (4), i.e., sweep, clearly comprise both the dominant portion of the total solution time and those which adversely affect scaling.

The performance effects of vectorization for the KBA0 and KBA1 algorithms is examined in Fig. 5. Here, the $720 \times 4 \times 4$ test case is run on single cores of the M-C and SB architectures, and four cores, using 16 threads, on an Intel Xeon Phi (Knights Corner), for similar power requirements. Runs are performed both with vectorization enabled and, via use of the Intel "–no-vec" compiler flag, disabled. The numbers in boxes indicate the vectorization gain (ratio of solution times without and with vectorization) for an algorithm/architecture pair.

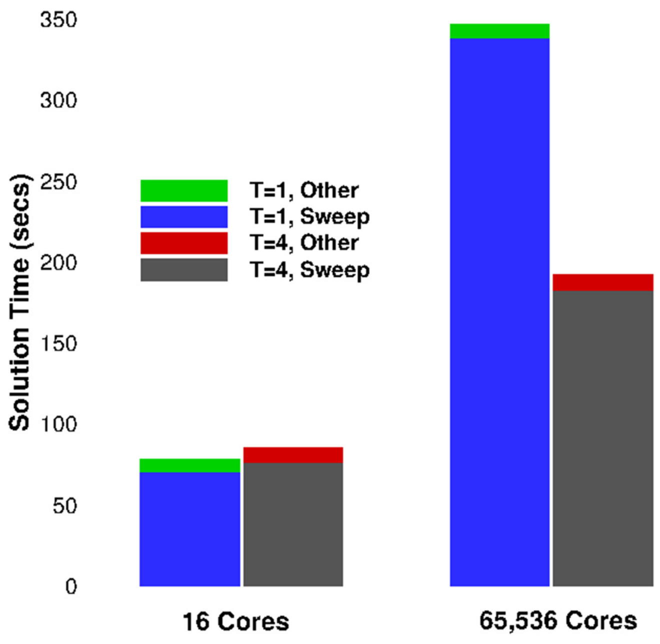

Fig. 4. Solution time components.

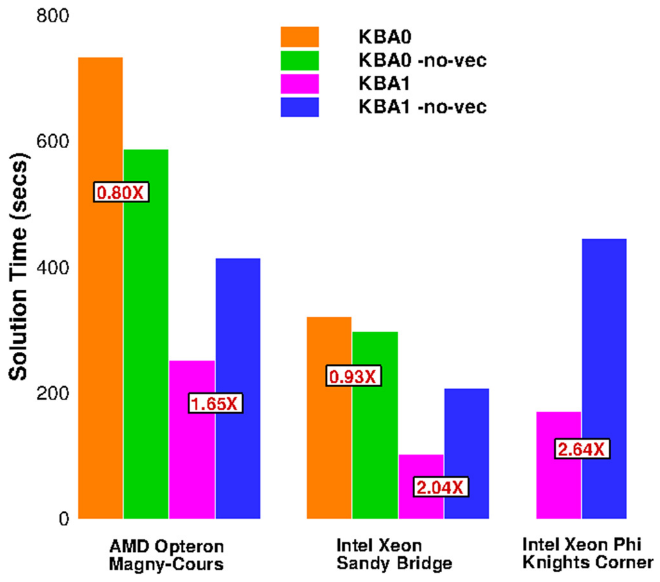

Fig. 5. Vectorization performance.

Disabling vectorization actually increases performance with the KBA0 algorithm, since it exposes insufficient vectorization opportunities to offset the associated increases in loop overhead costs. The KBA1 algorithm, on the other hand, shows substantial benefits from vectorization on all architectures, although always less than the theoretical maximum. Our analysis on Knights Corner has shown that, although our algorithm is indeed vectorized well, i.e., with an average vector length of eight, we are limited by memory bandwidth during the cell balance solution of Eq. (4). This limitation is further aggravated by the use here of all threads as independent energy groups, which induces cache thrashing. We are currently exploring alternative approaches for the cell balance solution that will minimize these effects. Finally, even with all vectorization disabled, we see solution times for the KBA1 algorithm are still less than those for the KBA0 algorithm. This is a result of the design goal of the KBA1 algorithm of minimizing data movement.

IV. CONCLUSIONS

Effective use of current and future computer architectures depends upon maximizing independent work units while minimizing data movement. The modified KBA algorithm discussed here does so, while still preserving the intrinsic domain decomposition that defines KBA sweeps. Building upon the lessons learned from Roadrunner, we have implemented a vectorized MPI+OpenMP version of our algorithm that offers substantial reductions in run time on current multicore platforms and shows promise for sustaining these performance improvements on the many-core platforms of the near future. We believe use of this algorithm will enable us to meet our requirements for job turnaround times while staying within realistic power and mean time to job interrupt constraints.

We also believe our modified KBA algorithm will allow similar performance gains to be obtained on heterogeneous architectures with an accelerator, such as GPUs, both because such an architecture resembles that which our algorithm was originally designed for (i.e., Roadrunner) and, more importantly, because the fundamental principles of maximizing independent work units while minimizing data movement are independent of architectural type.

Acknowledgments

Los Alamos National Laboratory is operated by Los Alamos National Security, LLC, for the United States Department of Energy under contract DE-AC52-06NA25396.

References

1. K. KOCH, R. BAKER, and R. ALCOUFFE, “Solution of the First-Order Form of the Three-Dimensional Discrete Ordinates Equation on a Massively Parallel Machine,” *Trans. Am. Nucl. Soc.*, **65**, 198 (1992).
2. R. S. BAKER and K. R. KOCH, “An S_N Algorithm for the Massively Parallel CM-200 Computer,” *Nucl. Sci. Eng.*, **128**, 312 (1998); http://dx.doi.org/10.13182/NSE98-1.
3. R. E. ALCOUFFE et al., “PARTISN: A Time-Dependent, Neutral Particle Transport Code System,” LA-UR-12-23848, Los Alamos National Laboratory (2012).

4. R. S. BAKER and R. E. ALCOUFFE, “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.

5. M. P. ADAMS et al., “Provably Optimal Parallel Transport Sweeps on Regular Grids,” *Proc. Int. Conf. Mathematics and Computational Methods Applied to Nuclear Science and Engineering*, Sun Valley, Idaho, May 5–9, 2013, American Nuclear Society (2013).

6. Y. Y. AZMY, “Multiprocessing for Neutron Diffusion and Deterministic Transport Methods,” *Prog. Nucl. Energy*, **31**, 317 (1997); http://dx.doi.org/10.1016/S0149-1970(96)00015-7.

7. R. S. BAKER et al., “Solution of the First-Order Form of the Multidimensional Discrete Ordinates Equations on a Two-Level Heterogeneous Processing System,” *Trans. Am. Nucl. Soc.*, **105**, 510 (2011).

8. R. E. ALCOUFFE, “Diffusion Synthetic Acceleration Methods for the Diamond-Differenced Discrete-Ordinates Equations,” *Nucl. Sci. Eng.*, **64**, 344 (1977); http://dx.doi.org/10.13182/NSE77-1.