LA-UR-13-21070

Approved for public release; distribution is unlimited.

Title: SNAP: SN (Discrete Ordinates) Application Proxy - Proxy Description

Author(s): Zerr, Robert J.
Baker, Randal S.

Intended for: Description of developing proxy application, intended for distribution with/without code.

SNAP: SN (Discrete Ordinates) Application Proxy, Rev 1

Proxy Description

Joseph Zerr (rzerr@lanl.gov) and Randal Baker (rsb@lanl.gov)
Los Alamos National Laboratory
Computational Physics and Methods, CCS-2
Updated: 26 March 2013

Introduction

The SNAP (SN Application Proxy) code is written to simulate the computational workload, memory requirements, and communication patterns of a modern neutral particle transport application. It is modeled off the Los Alamos National Laboratory code PARTISN. PARTISN solves the linear Boltzmann transport equation (TE), a governing equation for determining the number of neutral particles (e.g., neutrons and gamma rays) in a multi-dimensional phase space. [1] The phase space is characterized by physical location, $\vec{r}$; the particle's energy, E (or speed, v); the time, t; and the particle's direction of travel, $\widehat{\Omega}$. The unknown of the TE is the particle distribution density, N, or more commonly the angular flux $\psi = vN$. The high dimensionality and complexity of the TE make it a very expensive equation to solve computationally, with unique challenges to achieve excellent parallel performance. In the following sections, details about solving the TE are provided to help elucidate the motivation for the design of the parallel algorithms used in PARTISN and SNAP. The challenges and limitations of current methods are described. The role of SNAP in analyzing architectures and parallel algorithms is documented. Finally, how the findings will relate to PARTISN is discussed.

Solving the Transport Equation (Requirements)

SNAP serves as a simpler code that allows testing of proposed algorithms for parallel solutions of the deterministic TE. The first step in the deterministic approach is to discretize the analytic TE over the phase space variables. The default spatial discretization in PARTISN, and the one therefore modeled in SNAP, is a finite volume approach with auxiliary *diamond difference* relations for closure. The diamond difference approximation relates mesh cell-average quantities to edge-/face-average quantities. The TE is integrated over the energy variable in tens or hundreds of *groups*, with each group's energy range carefully selected to capture important spectral properties of the particle distribution. This multigroup method yields independent tasks for each group, and inter-group coupling—due to common physical processes—is treated iteratively by sharing information about group-to-group sources and recomputing the solution until convergence. The time derivative in the TE is approximated with the Crank-Nicholson finite-difference method. Angular terms related to particle direction are treated with a collocation method that approximates the integrations over the unit sphere by summing solutions along suitably chosen *discrete ordinates* (directions, angles) and their corresponding weights. Discrete ordinates are defined by two angular variables, m and n. The so-called discrete ordinates, S_N, method is a defining characteristic of PARTISN. Codes that employ the S_N method face different computational obstacles than other angular approximations. The differences are reflected at the highest level of the parallel programming models.

The computational challenges to solving the discretized transport equation are immediately apparent. The number of unknowns is incredibly large; a 3-D transport solution vector can have as many as seven dimensions, $\psi(x, y, z, m, n, E, t)$. Fortunately, the rank of the solution array is reduced to six, because a temporal history of the angular flux need not be stored in memory (if at all). However, two copies of the reduced-rank array do need to be stored during execution. If memory is very limiting, this can be reduced to one full copy and one partial copy of the solution array, but at the cost of more operations. Moreover, parallelization over the time domain (e.g., via a prefix algorithm) is unfeasible, because it is useless without some knowledge of time-dependent source functions, which are often unavailable.

The iterative strategy used to solve the time-dependent TE is the source iteration (SI) method. The SI method is composed of a within-group iterative loop nested inside an outer loop that completes the inter-group coupling. While Gauss-Seidel iterations provide better convergence for this outer loop, Jacobi iterations also have acceptable convergence properties. That is, the outer loop can be naturally decomposed over the energy domain with tolerable penalty. The inner loop of the SI method involves converging upon within-group particle scattering interactions. The flux result from the inner loop is used to update group-to-group coupling for outer loop iterations.

The largest portion of execution time for typical SI calculations is spent in the inner loop. The kernel of the inner loop involves a *mesh sweep*: the discrete ordinates transport equation is solved for all spatial cells along all discrete angles. During the mesh sweep for each angle, the code marches over the cells and balances particle sources and losses along that specified direction to compute particle density. The order of the march, including starting and ending points, is determined by the ordinate.

When considering how best to solve the discretized TE in parallel, we consider how to decompose the problem over distributed memory processes, shared memory threads, and vectorized operations. Further, we seek decompositions over the physical domains. As opposed to task-based parallelism, domain decompositions are more natural given the SI method, improving integration with the optimized data layout as determined by the loops.

The nested structure of the SI loops suggests the outer loop, decomposed over energy, may be a good candidate for distributed memory computing. Not surprisingly, energy domain decomposition is an available parallel execution option in PARTISN, and especially useful for 1-D (in space) problems. However, multigroup treatments are typically limited to tens or hundreds of groups. Given the number of available nodes of modern supercomputer designs, limiting the distributed memory decomposition to hundreds or even thousands of independent processes greatly underutilizes the machine and does not achieve the scaling and speedup demanded by the more complicated calculations code developers and users will naturally want to perform.

Angular domain decomposition is not a viable candidate for distributed memory computing. Scaling is again limited by the number of angles, typically in the hundreds to thousands. Furthermore, after the mesh sweep over all angles is complete, reductions of the solution over the angular variable are performed to minimize memory demand and share source information among the groups. Frequent reductions degrade performance.

For massively parallel processing architectures, the most suitable distributed memory decomposition is over the spatial domain. Transport problems can easily be composed of millions, or even billions, of spatial cells. Consequently, memory demand requires that the problem be decomposed over nodes. Spatial chunks of the problem are divided among the processors. Some choice must be made regarding this decomposition, and the resulting solution. The mesh sweep is a numerical wavefront that passes over all cells in the direction of the discrete ordinate being tracked. Information is computed and passed downstream, starting from a known incoming boundary condition for the wave and ending at an opposing boundary where the wave either exits to vacuum or is reflected. This is an inherently serial process. The problem can be decomposed into sub-domains that involve local solves only (e.g., localized mesh sweeps). Global iterations are performed on the information exchanged at sub-domain boundaries. Such parallel block Jacobi (PBJ) algorithms keep all processes busy but have numerical drawbacks, including poor convergence in optically thin systems. On the other hand, parallelized mesh sweeps, a la the Koch-Baker-Alcouffe (KBA) algorithm [2], are an intrinsic decomposition of the serial mesh sweep, retaining the convergence properties of SI. KBA is the transport-specific application of generalized parallel wavefront applications, and is the parallel algorithm used in the test code Sweep3D [3]. A spatial domain of dimension n is decomposed over an $n-1$ process topology. Startup and shutdown penalties create load imbalances that hinder performance, but KBA has been shown to be easily scalable to thousands of processes in the full PARTISN package. Furthermore, mixed domain decomposition—with KBA for spatial domain decomposition and Jacobi iterations for energy domain decomposition—permits excellent scaling out to tens of thousands of distributed memory processes.

To utilize the power of the accelerator architectures of next generation supercomputers, threading must be incorporated into codes. SNAP follows the PARTISN model by threading over a mock energy domain [4]. Nested threading is provided as an option in SNAP, whereby the spatial cells already decomposed and swept via KBA may be further handled concurrently. A secondary level of threads, spawned by group-dependent/primary level threads, perform concurrent operations on cells the lie on the same diagonal line/plane of each MPI process' spatial sub-domain. Such cells are independent of one another, and these "mini-KBA" sweeps provide an additional avenue for concurrency. The angular domain, as described below, is used for vectorization.

KBA efficiency is known to improve when angles are pipelined, which minimizes the effect of startup and shutdown costs. However, because modern processing elements possess excellent vectorization instruction sets, KBA efficiency is slightly sacrificed by abandoning some pipelining and opting instead for angular vectorization of the mesh sweep. Rather than choosing a discrete angle and marching from one spatial cell to the next, SNAP solves the particle balance at each cell for all the angles of a directional octant. (Angles within the same octant have the same starting and ending spatial cell in a mesh sweep.) The operations are vectorized, with different angles serving as the set of data. The mesh sweep proceeds by marching to each cell, completing the same vectorized operations. Pipelining of the angles in an octant is no longer used in the KBA decomposition.

The inner loop solves the within-group equations for group g along each discrete ordinate m within an octant. With diamond difference in space and Crank-Nicholson in time, the 3-D numerical equations can be written in standard operator notation as

$$\begin{aligned} r(g)[\boldsymbol{f}(g) - \boldsymbol{f}^{\boldsymbol{t}}(g)] + \boldsymbol{l}^{\boldsymbol{x}^T} \cdot [\boldsymbol{f}(g) - \boldsymbol{f}^{x}(g)] + \boldsymbol{l}^{\boldsymbol{y}^T} \cdot [\boldsymbol{f}(g) - \boldsymbol{f}^{y}(g)] \\ + \boldsymbol{l}^{\boldsymbol{z}^T} \cdot [\boldsymbol{f}(g) - \boldsymbol{f}^{z}(g)] + \boldsymbol{c}^T(g) \cdot \boldsymbol{f}(g) = \boldsymbol{q}^{\boldsymbol{i}}(g) + \boldsymbol{q}^{\boldsymbol{g}}(g) + \boldsymbol{q}^{s}(g). \end{aligned} \tag{1}$$

The solution (column-) vector is $\boldsymbol{f}$. The length of $\boldsymbol{f}$ is equal to the number of angles, M, by the number of spatial cells (e.g., $I \times J \times K$ for a 3-D problem). Angular vectorization requires that the fastest varying elements of the vectors be over angle. For a given group and time-step size r is a scalar that is computed and stored as a vector of length G (i.e., total number of groups) outside the outer loop calculations. The first left hand side (LHS) term represents the time rate of change of the solution, where $\boldsymbol{f}^{\boldsymbol{t}}$ is the incoming flux from the previous time step. The streaming loss operators—$\boldsymbol{l}^{\boldsymbol{x}}$, $\boldsymbol{l}^{\boldsymbol{y}}$, and $\boldsymbol{l}^{\boldsymbol{z}}$—are vectors of equal length to $\boldsymbol{f}$, and represent particle streaming in the x-, y-, and z-directions respectively. Moreover, the vectors $\boldsymbol{f}^{\boldsymbol{x}}$, $\boldsymbol{f}^{\boldsymbol{y}}$, and $\boldsymbol{f}^{\boldsymbol{z}}$ are known, incoming face fluxes, which are computed during the sweep. The probability of an interaction of any time is represented by $\boldsymbol{c}$; it is angle independent and sized for the number of cells only.

The inhomogeneous source vector $\boldsymbol{q}^{\boldsymbol{i}}$ is a known, fixed quantity. The out-of-group sources vector is computed during the outer loop,

$$\boldsymbol{q}^{\boldsymbol{g}}(g) = \boldsymbol{MSDf} = \sum_{g' \neq g}^{G} \sum_{n'=1}^{N} M(m,n')S(n',i,j,k,g',g) \sum_{m'=1}^{M} D(n',m')f(m',i,j,k,g'), \tag{2}$$

where N is the number of modeled moments to capture scattering anisotropy. $\boldsymbol{q}^{\boldsymbol{s}}$ and $\boldsymbol{q}^{\boldsymbol{g}}$ vectors must be sized for all groups, giving them length of $M \times I \times J \times K \times G$. The $\boldsymbol{D}$ operator maps the discrete angular functions of f to angular moments (integrations over angle). $\boldsymbol{S}$ is a group-to-group scattering operator. $\boldsymbol{M}$ maps the moments back to angularly discrete functions. The within-group scattering source, which is iterated upon during the inner loop, is computed as

$$\boldsymbol{q}^{s}(g) = \boldsymbol{MS_gDf} = \sum_{n=1}^{N} M(m,n')S(n',i,j,k,g,g) \sum_{m'=1}^{M} D(m',n')f(m',i,j,k,g). \tag{3}$$

Because the scattering source is only needed for the within-group equations, it can typically be sized without an additional dimension of G.

The $\boldsymbol{l^a}$ and $\boldsymbol{f^a}$ ($a = x$, y, or z) vectors are never fully computed during the mesh sweep. Moreover, during the mesh sweep, vector multiplications are done only over the angular variable. Loops are performed to cover the entirety of the spatial mesh. That is, at each spatial cell, Eq. 1 is applied as (neglecting the group for brevity)

$$\begin{aligned}\left(r + l^x_{:,i,j,k} + l^y_{:,i,j,k} + l^z_{:,i,j,k} + c_{:,i,j,k}\right) f_{:,i,j,k}{}^{(l+1)} - r f^t_{:,i,j,k} - l^x_{:,i,j,k} f^x_{:,i,j,k} \\ - l^y_{:,i,j,k} f^y_{:,i,j,k} - l^z_{:,i,j,k} f^z_{:,i,j,k} = q^i_{:,i,j,k} + q^g_{:,i,j,k} + q^s_{:,i,j,k}{}^{(l)}.\end{aligned} \tag{4}$$

The subscript ":" is used to indicate a vectorization over discrete angles, which can be done on a per octant basis. All incoming terms are either known from an upwind cell in the sweep, from a boundary condition, or from the previous time step; hence those four terms may be moved to the RHS. The inner iteration index (l) is applied to the solution vector $\boldsymbol{f}$ and the iterated scattering source, $\boldsymbol{q^s}$, which is computed using the previous iterate of $\boldsymbol{f}$.

Diamond difference relations are used to compute the flux at the outgoing faces. For example, the outgoing x-direction face is computed with

$$f^x_{:,i,j,k} \leftarrow 2 f_{:,i,j,k} - f^x_{:,i,j,k}. \tag{5}$$

Analogous expressions can be written for the y- and z-direction faces. The outgoing time edge is also computed with the diamond difference relation, but to save operations, both an ingoing and outgoing copy of the vector are stored separately. Moreover, these vectors must be stored for all G groups.

The sweep proceeds spatially with i as the fastest varying index, followed by j, then k. Because the sweep proceeds in this manner, the $\boldsymbol{l^x}$ and $\boldsymbol{f^x}$ terms need be stored only for an individual upstream cell. Likewise, the $\boldsymbol{l^y}$ and $\boldsymbol{f^y}$ terms need be stored only for the upstream row of I cells. The $\boldsymbol{l^z}$ and $\boldsymbol{f^z}$ terms need be stored only for the upstream $I \times J$ plane of cells.

Lastly, the solution $\boldsymbol{f}$ in neutral particle transport is generally angularly integrated for physical considerations. The so-called scalar flux is computed with the angular quadrature with weights, $\boldsymbol{w}$,

$$p_{i,j,k} = w_{:} \cdot f_{:,i,j,k}. \tag{6}$$

The vector $\boldsymbol{p}$ is stored with length $I \times J \times K$ by the number of groups G. This permits the solution vector $\boldsymbol{f}$ and the incoming flux vectors, $\boldsymbol{f^x}$, $\boldsymbol{f^y}$, and $\boldsymbol{f^z}$, to need to be stored for a single group only, saving a significant amount of memory.

With the use of KBA spatial decomposition, each process is assigned a chunk of the global $I \times J \times K$ system. Furthermore, because KBA decomposes data to an $n - 1$ topology, a 3-D calculation would assign each process approximately $I \times J/P_y \times K/P_z$ spatial cells.

During the outer loop of SI, Eqs. 4–6 are solved within the mesh sweep framework for all groups. Then the out-of-group sources, Eq. 2, are updated before a new set of inner iterations. Therefore, the group-wise components of the outlined operators are decomposed over threads. Data for all the groups is shared, and group-independent portions of the arrays are modified privately by the assigned threads.

SNAP performs the same set calculations with the same operation ordering as the SI scheme in PARTISN. However, the phase-space variables are merely mock representations of those in PARTISN. The data used by SNAP have no physical meaning, but pass through the same series of operations. Moreover, the number of elements in the mock dimensions shall be good approximations for the size of data structures worked upon by PARTISN. This will ensure that scaling testing by SNAP is a realistic measure of optimal PARTISN performance.

Given the selection of the SI method for solving the TE, the above decomposition of the problem provides what is considered the best parallel performance properties. It helps distribute memory burdens to make calculations executable; it preserves favorable numerical (convergence) properties; and it minimizes the communications, improving granularity.

However, other methods cannot be ruled unworkable automatically. The above approaches are a result of the typical problems encountered by PARTISN, the discretization of PARTISN variables (namely diamond difference in space), and experience with hybrid architectures that was partly influenced by the limitations in exploring every possible avenue for parallelization when machines were introduced. SNAP allows more rapid implementation of a different (or novel) algorithm for parallel execution. Testing it with different platforms shall confirm that our preselected algorithms scale well and are suitable for PARTISN on next generation supercomputers; or testing will show weaknesses, drawing attention particularly to the treatment of a specific domain that is not amenable to that architecture.

What can be Tested with SNAP (Analysis)

SNAP is intended to closely approximate the parallel performance of its parent code PARTISN. As a skeleton proxy application, it has not been programmed to replicate any actual neutral particle physics. Therefore, SNAP will be unable to yield any conclusions about the impact that parallel decompositions have on numerical solution algorithms. SNAP will be a tool for examining the scaling of PARTISN parallel decompositions and for quick implementation and testing of other parallel concepts. Analysis of SNAP results shall be made solely on the performance characteristics including distributed memory scaling via MPI; finer-grained, threaded concurrency via OpenMP (and potentially OpenACC in the future); overall memory demand; and data proximity/locality.

SNAP uses the KBA algorithm for parallel decomposition over the distributed memory nodes. The startup/shutdown penalties associated with KBA that create temporary load imbalances are measured informally by the decomposition's parallel computational efficiency (PCE), a ratio of the number of unknowns being solved to the number of operations performed in parallel. Using the same algorithm, SNAP matches the PCE of PARTISN for the same-sized problems. Moreover, the problem parameters that affect PCE that are modifiable in PARTISN are also modifiable in SNAP. The computation to communication ratio in SNAP should be a very good approximation to that of PARTISN. This allows for testing the scaling (strong and weak) relative to increasing MPI processes. Furthermore, MPI options that may improve process layout can also be tested with SNAP.

The effects of threading the energy domain shall also be tested using SNAP. This includes performance relative to the number of available processing elements for threads. The interplay between the individual threads and the MPI process that controls internode communication will be of particular interest. Good scaling will require that communications between threads of different MPI ranks must not deadlock nor hinder performance with restrictive serialized loops at critical points. SNAP is programmed with OpenMP directives to spawn threads and assign work. However, it should be easy to quickly exchange the OpenMP pragmas for OpenACC or some other tool.

SNAP can be analyzed with modern profiling tools. This helps ensure SNAP is mimicking the behavior of the compute intensive kernel in PARTISN. SNAP must match PARTISN's total number of operations, using vectorized loops whenever possible. The profiler will reveal information about cache-hit percentage, will help SNAP coding be organized such that it has similar data locality, and will help achieve a memory bandwidth of approximately one flop per load, similar to PARTISN's

SNAP Code Design

SNAP is a proxy for PARTISN's computational properties. The data layout and decomposition is similar to PARTISN, and the compute-intensive kernel of SNAP has been coded with very similar logic to the sweep kernel and matrix-vector operations of PARTISN. However, SNAP entails no relevant physics of neutral particle transport. Therefore, SNAP results are not reflective of numerical accuracy or the convergence properties of the SI scheme. It is representative of PARTISN performance, including advantages and disadvantages for parallel processing on hybrid architectures. The lack of physically-related computations should not impact co-design efforts.

SNAP Implementation

SNAP is written to the Fortran 90/95 standard. It is composed of several subroutines with explicit, flat interfaces. Distributed memory communications are performed using MPI commands, and threading is achieved with OpenMP pragmas. The implementation of these tools may be modified for improved performance where possible. Additionally, the subroutines of greatest interest are those that perform the compute-intensive sweep operations and matrix-vector multiplications. These include `dim3_sweep`, `inner_src_scat`, and `outer_src_scat`. These subroutines should clearly show the chosen data decompositions across processes and threads. Moreover, the ordering of these subroutines is a direct consequence of the SI scheme. The data dependency that represents neutral particle transport is evident in these subroutines. Lastly, many loops have been written for vectorization, using SIMD instruction sets built into individual processing elements.

The SNAP algorithms and data decomposition have been targeted for optimized performance on distributed memory mesh or torus topologies with MIC nodes using threading over the energy domain. It is expected that significant modification to the algorithms will be necessary for a GPGPU implementation to achieve good speedup and scaling properties. Such modification could mean, in particle transport terms, different TE discretization methods and/or different iterative solution methods (i.e., PBJ).

Testing

SNAP incorporates no actual physics in its available data. The solution it produces must match the numerical equation being solved that is representative of a discretized TE, but absent physical data, the solution has no real-world relevance. Therefore, the method of manufactured solutions can be used to generate an inhomogeneous source from a pre-selected solution by the user. Provided the solution, SNAP will generate the inhomogeneous source, compute the solution with the SI scheme, and compare the result. The difference between the provided and computed solution vectors must be smaller than the SI convergence criterion given as input. This ensures the user that the code is working properly. No code-to-code verification will be available with PARTISN, and no analysis regarding the numerical properties of SNAP's solution algorithm is sought.

Future Plans and Alternative Considerations

Future plans and potentially interesting modifications to SNAP may be made by adding relevant functionality to further match PARTISN tasks and/or modifying the solution algorithm. This includes adding non-linear operations to the mesh sweep that hinders performance but better conforms to common PARTISN applications. Moreover, the solution algorithm, and perhaps the data layout, could be modified and tested for what is expected to be more optimal performance with GPGPU accelerators.

SNAP's numerical solution algorithm currently proxies for PARTISN's linear mesh sweeping algorithm. However, SNAP lacks a non-linear, computationally expensive and disruptive set of operations known as *fixup*, intended to correct unphysical results due to the approximations made to numerically solve the TE. Fixup is an important tool for PARTISN and used frequently enough to have non-negligible effects on performance. Future versions of SNAP will include this functionality.

It is expected that the KBA sweep with typical problem parameters for PARTISN will result in unacceptably poor scaling on systems using GPGPU accelerators. SNAP may perhaps be modified using a PBJ solution algorithm for better performance. Such an effort potentially will have to consider the data decomposition for all phase space variables, given the unique memory and threading properties of such a platform.

Retirement of the SNAP application will be necessary when either the solution algorithms or computing platforms are deemed obsolete. At that time, the SNAP code can be replaced with a proxy that better represents the ASC neutral particle transport capability of the future.

Deployment

The SNAP code and support is available by contacting Joe Zerr of Los Alamos National Laboratory. He can be reached by email at rzerr@lanl.gov or by phone at 505-665-3560.

References

[1] E. E. Lewis and W. F. Miller, Jr., *Computational Methods of Neutron Transport*, American Nuclear Society, La Grange Park, IL, USA (1993).

[2] R.S. Baker and K.R. Koch, "An S_n Algorithm for the Massively Parallel CM-200 Computer," *Nuclear Science and Engineering*, **128**, 312 (1998).

[3] "LANL: CCS-3: Performance and Architecture: Software," Available at http://www.ccs3.lanl.gov/PAL/software.shtml," Last accessed 11/30/2012 (2006).

[4] R. S. Baker et al., "Solution of the First-Order Form of the Multi-Dimensional Discrete Ordinates Equations on a Two-level Heterogeneous Processing System," *Transactions of the ANS*, **105**, 510 (2011).