Literary Review

Tarek Ghaddar
Dr. Jean Ragusa

Nuclear Engineering Department
Texas A&M University
College Station, TX, 77843-3133

The Transport Equation

Equation 1 represents the discrete ordinates (S_n) form of the Boltzmann transport equation, where the angular flux is discretized into a set of energy groups g and angular directions m,

$$\vec{\Omega}_m \cdot \vec{\nabla} \psi_{m,g}^{(l+1)} + \Sigma_{t,g} \psi_{m,g}^{(l+1)} = S_{m,g}^{(l)} + \sum_{g'} \sum_{m'} \omega_m \Sigma_{s,g' \to g,m' \to m} \psi_{m'g'}^{(l)}. \tag{1}$$

The left hand side of the equation represents loss terms: $\vec{\Omega}_m \cdot \vec{\nabla}$ is the leakage operator, and $\Sigma_{(t,g)}$ is the collision operator (absorption and outscatter). The right hand side represents the total source: $S_{m,g}$ represents the external source, $\Sigma_{s,g' \to g,m' \to m}$ represents the inscatter operator, and ω_m is a weighting factor for angular direction m.

The preferred way to solve the S_n form of the transport equation is source iteration. An initial guess is made, which occupies the right hand side of Eq. 1, and the next iteration of $\psi_{m,g}$ is calculated from this. This process is repeated until $\psi_{m,g}$ converges within a user-specified tolerance. The method of solving the S_n transport equation on a domain discretized in space, energy, and angle is the tranpsort sweep.

The Structured Transport Sweep (PDT)

A sweep algorithm is defined by three properties:

- partitioning: dividing the domain among available processors
- aggregation: grouping cells, directions, and energy groups into tasks
- scheduling: choosing which task to execute if more than one is available

If M is the total number of angular directions, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

In a regular grid, we have the number of cells in each Cartesian direction: N_x, N_y, N_z. These cells are aggregated into "cellsets". However, in an unstructured mesh, the number of cells cannot be described as such. In PDT specifically we initially subdivide the domain into subsets, which are just rectangular subdomains. Within each subset, an unstructured mesh is created. This creates a pseudo-regular grid. These subsets become the N_x, N_y, N_z equivalent for an unstructured mesh. The spatial aggregation in a PDT unstructured mesh is done by aggregating subsets into cellsets.

If M is the total number of angular directions, G is the total number of energy groups, and N is the total number of cells, then the total fine grain work units is $8MGN$. The factor of 8 is present as M directions are swept for all 8 octants of the domain. The finest grain work unit is the calculation of a single direction and energy groups unknowns in a single cell, or $\psi_{m,g}$ for a single cell.

Fine grain work units are aggregated into coarser-grained units called *tasks*. A few terms are defined that describe how each variable is aggregated.

- $A_x = \frac{N_x}{P_x}$, where N_x is the number of subsets in x and P_x is the number of processors in x
- $A_y = \frac{N_y}{P_y}$, where N_y is the number of subsets in y and P_y is the number of processors in y
- $N_g = \frac{G}{A_g}$
- $N_m = \frac{M}{A_m}$
- $N_k = \frac{N_z}{P_z A_z}$

It follows that each process owns N_k cell-sets (each of which is A_z planes of $A_x A_y$ cells), $8N_m$ direction-sets, and N_g group-sets for a total of $8N_m N_g N_k$ tasks.

One task contains $A_x A_y A_z$ cells, A_m directions, and A_g groups. Equivalently, a task is the computation of one cellset, one groupset, and one angleset. One task takes a stage to complete. This is particularly important when comparing sweeps to the performance models.

The minimum possible number of stages for given partitioning parameters P_i and A_j is $2N_{\text{fill}} + N_{\text{tasks}}$. N_{fill} is both the minimum number of stages before a sweepfront can reach the center-most processors and the number needed to finish a direction's sweep after the center-most processors have finished. Equations 2, 3, and 4 define N_{fill}, N_{idle}, and N_{tasks}:

$$N_{\text{fill}} = \frac{P_x + \delta_x}{2} - 1 + \frac{P_y + \delta_y}{2} - 1 + N_k(\frac{P_z + \delta_z}{2} - 1) \tag{2}$$

$$N_{\text{idle}} = 2N_{\text{fill}} \tag{3}$$

$$N_{\text{tasks}} = 8N_m N_g N_k \tag{4}$$

where δ_u is 1 for P_u odd, and 0 for P_u even.

Equation 5 approximately defines parallel sweep efficiency. This can be calculated for specific machinery and partitioning parameters by substituting in values calculated using Eqs. 2,3, and 4.

$$\begin{aligned} \varepsilon &= \frac{T_{\text{task}} N_{\text{tasks}}}{[N_{\text{stages}}][T_{\text{task}} + T_{\text{comm}}]} \\ &= \frac{1}{[1 + \frac{N_{\text{idle}}}{N_{\text{tasks}}}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \end{aligned} \tag{5}$$

Equations 6 and 7 show how T_{comm} and T_{task} are calculated:

$$T_{\text{comm}} = M_L T_{\text{latency}} + T_{\text{byte}} N_{\text{bytes}} \tag{6}$$

$$T_{\text{task}} = A_x A_y A_z A_m A_g T_{\text{grind}} \tag{7}$$

where T_{latency} is the message latency time, T_{byte} is the additional time to send one byte of message, N_{bytes} is the total number of bytes of information that a processor must communicate to its downstream neighbors at each stage, and T_{grind} is the time it takes to compute a single cell, direction, and energy group. M_L is a latency parameter that is used to explore performance as a function of increased or decreased latency. If a high value of M_L is necessary for the model to match computational results, improvements should be made in code implementation.

Scheduling utilizes an algorithm to assign priority to tasks. An example of this is the "depth-of-graph" algorithm, which gives priority to the task that has the longest chain of dependencies. In general, a quantity D is defined, which is simply the number of downstream dependents each task has. Then a simple series of logicals determines which task has priority and should be executed first:

1. Tasks with higher D have higher priority,
2. If multiple tasks have the same D, then tasks with $\Omega_x > 0$ have priority,
3. The next tiebreaker is $\Omega_y > 0$,
4. The next tiebreaker is $\Omega_z > 0$,

Equation 8 shows how to calculate the quantity D for an example octant (-x,+y,-z):

$$D(-+-) = (i-1) + (P_y - j) + (k-1) \tag{8}$$

Another optimal scheduling algorithm (not proved) is the "push-to-central" algorithm. This algorithm prioritizes tasks that advance wavefronts to central planes in the processor layout. A few definitions are necessary to understand the push-to-central algorithm:

- $i \, \varepsilon \, (1, P_x) =$ the x index into the processor array, with similar definitions for y and z indices, j and k,
- $X = \frac{P_x + \delta_x}{2}$,
- $Y = \frac{P_y + \delta_y}{2}$,
- $Z = \frac{P_z + \delta_z}{2}$.

With these definitions, The push-to-central algorithm prioritizes the following tasks accordingly:

1. If $i \leq X$, then tasks with $\Omega_x > 0$ have priority, while for $i > X$, tasks with $\Omega_x < 0$ have priority.
2. If multiple ready tasks have the same sign on Ω_x, apply rule 1 to j, Y, Ω_y.
3. If multiple ready tasks have the same sign on Ω_x and Ω_y, apply rule 1 to k, Z, Ω_z.

This schedule pushes tasks toward the $i = X$ central processor plane with top priority, followed by pushing toward the $j = Y$ central processor plane, followed by pushing toward the $k = Z$ central processing plane.[1]

KBA Algorithm

The KBA algorithm traditionally chooses $P_z = 1, A_m = 1, G = A_g = 1, A_x = N_x/P_x, A_y = N_y/P_y$, with A_z being the selectable number of z-planes to be aggregated into each task. With $N_k = N_z/A_z$, each processor performs $N_{\text{tasks}} = 8MN_k$ tasks. With the KBA algorithm, $2MN_k$ are pipelined from a given corner of the 2D processor layout. The far corner processor remains idle for the first $P_x + P_y - 2$ stages, which means that a 2 octant sweep completes in $2MN_K + P_x + P_y - 2$ stages. If an octant-pair sweep does not begin until the previous pair's finishes, the full sweep requires $8MN_k + 4(P_x + P_y - 2)$ stages, which means the KBA parallel efficiency is:

$$\varepsilon_{KBA} = \frac{1}{[1 + \frac{4(P_x+P_y-2)}{8MN_k}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{9}$$

Overloaded Volumetric Partitioning

The overloaded volumetric partitioning proceeds as follows:

1. In a 2D (3D) domain, cellsets are divided into 4 (8) spatial quadrants (octants), with an equal number of cellsets in each SQO (SQO is defined as a spatial quadrant or octant).
2. Assign 1/4 of the processors (1/8) in 3D to each SQO.
3. Choose the individual overload factors $\omega_x, \omega_y, and \omega_z$ and individual processor counts $P_x, P_y, and P_z$, such that $\omega_x \omega_y \omega_z = \omega_r$ and $P_x P_y P_z = P$, with all P_u even. ω_u is defined as the number of cellsets assigned to each P_u.
4. An array of $\omega_x \cdot \omega_y \cdot \omega_z$ "tiles" in each SQO. Each tile is an array of $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ cellsets. These cellsets are mapped one-to-one to the $1/2P_x \cdot 1/2P_y \cdot 1/2P_z$ procesors assigned to the SQO, using the same mapping in each tile.

Each tile has a logically identical layout of cellsets, and each processor owns exactly one cellset in each tile in its SQO, making each processor responsible for ω_r cellsets. Figure 1 shows three different partitioning schemes in 2D.

Figure 1. Three different partitioning schemes in 2D, from left to right: hybrid KBA, volumetric non-overloaded, and volumetric overloaded.

The optimal scheduling algorithm rules are as follows:

1. If $i \leq X$, then tasks with $\Omega_x > 0$ have priority, while for $i > X$, tasks with $\Omega_x < 0$ have priority.
2. If multiple ready tasks have the same sign on Ω_x, apply rule 1 to j, Y, Ω_y.
3. If multiple ready tasks have the same sign on Ω_x and Ω_y, apply rule 1 to k, Z, Ω_z.
4. If multiple tasks are ready in the same octant, then priority goes to the cellset for which the priority octant has greatest downstream depth.
5. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x, then priority goes to the cellset for which the priority octant has greatest downstream depth of graph in y.
6. If multiple ready tasks are in the same octant and have the same downstream depth of graph in x and y, then priority goes to the cellset for which priority octant has greatest depth of graph in z.

This ensures that each SQO orders the octants: the one it can start right away (A), three that have one sign difference from$A$(B, C, and D), three that have two sign differences ($\bar{D}, \bar{C}, \bar{B}$), and one in opposition to its primary ($\bar{A}$).

There are three constraints in order to achieve the optimal stage count. In these constraints, $M = \omega_g \omega_m / 8$, which is the number of tasks per octant per cellset.

1. $M \geq 2(Z-1)$
2. $\omega_z M \geq 2(Y-1)$
3. If $\omega_x > 1$, then $\omega_y \omega_z M \geq X$

These conditions ensure there are no idle time in a variety of situations. At large processor counts, the product $\omega_m \omega_g$ must be large. This means that a weak scaling series refined only in space, but only coarsely refined in angle and energy, will eventually fail the constraints.

The optimal efficiency formula changes slightly from the KBA and hybrid KBA partitioning method in order to account for the overload factors. The only change is in the $\frac{N_{idle}}{N_{tasks}}$ term, as shown in Eq. 10.

$$\varepsilon_{opt} = \frac{1}{[1 + \frac{P_x + P_y + P_z - 6}{\omega_g \omega_m \omega_r}][1 + \frac{T_{\text{comm}}}{T_{\text{task}}}]} \tag{10}$$

The Unstructured Transport Sweep

The development of an algorithm for efficient parallel transport sweeps focuses on obtaining a good sweep ordering. This is called *scheduling*. A new list scheduling algorithm has been constructed for modest levels of parallelism (up to 126 processors).

There are three requirements for a sweep scheduling algorithm to have. First, the algorithm should have low complexity, since millions of individual tasks are swept over in a typical problem. Second, the algorithm should schedule on a set of processors that is small in comparison to the number of tasks in the sweep graph. Last, the algorithm should distribute work in the spatial dimension only, so that there is no need to communicate during the calculation of the scattering source.

Here is the pseudocode for the algorithm:

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

An important part of the algorithm above is the assigning priorities to tasks. Specialized prioritization heuristics generate partition boundary data as rapidly as possible in order to minimize the processor idle time.

Nearly linear speedups were obtained on up to 126 processors. Further work is being done for scaling to thousands of processors.

Cycle Detection

A cycle is a loop in a directed graph. These occur commonly in unstructred meshes. The problem with cycles is that they can cause hang time in the problem, as a processor will wait for a message that might will never come. This means that the computation for one or more elements will never be completed. The solution to this is to "break" any cycles that exist by removing an edge of the task dependence graph (TDG). Old flux information is used on a particular element face in the domain. Most of the time, the edge removed is oriented obliquely with respect to the radiation direction.

Algorithms for for finding cycles are called *cycle detection* algorithms. This must be done efficiently in parallel, both because the task dependence graph is distributed, and because the finite element grid may be deforming every timestep and changing the associated TDG.

Cycle detection utilizes two operations: trim and mark. Trimming identifies and discards elements which are not in cycles. At the beginning of cycle detection, graphs are trimmied in the downwind direction, then the remaining graphs are trimmed in the upwind direction. A pivot vertex is then selected in each graph. Graph vertices are then marked as upwind, downwind, or unmarked. Then, if any vertices are both upwind and downwind, the cycle is these vertices plus the pivot vertex. An edge is removed between 2 cycle vertices, and 4 new graphs are created: a new cycle, the upwind vertices without the cycle, the downwind vertices without the cycle, and a set of unmarked vertices. This recursively continues until all cycles are eliminated.

Provable algorithms for parallel generalized sweep scheduling[2]

Given an unstructured mesh consisting of n cells, k directions, and m processors, the mesh induces a natural graph $G(V,E)$: where cells of the mesh correspond to vertices, and edges between vertices correspond to a shared boundary between two cells. If a direction, i, is present, it inducese a directed graph with an identical vertex set V, and a directed edge from u to v being present if and only if u and v are adjacent in G and the sweep direction requires u to be solved before v. Assuming all cycles are broken, this produces k directed acyclic graphs (DAG), one for each direction i.

An instance of a sweep scheduling (SS) problem is given by a vertex set V (or the cells in the unstructured mesh), k DAGs $G_i(V_i,E_i)$, and m processors. A feasible solution to this problem is a schedule that processes the k DAGs, so that the following constraints are satisfied:

1. The precedence constraints for each DAG $G_i(V_i,E_i)$ must be satisfied. If a directed edge goes from u to v, task (u,i) must be processed before task (v,i) can be started.
2. Each processor can process one task at a time, and a task cannot be pre-empted.

3. Every copy of vertex v must be processed on the same processor for each direction i.

Levels

Given k DAGs $G_i(V_i, E_i)$, levels (or layers) can be formed where in DAG $G_i(V_i, E_i)$, layer $L_{i,j}$ is the set of vertices with no predecessors after vertices $L_{i,1} \cup \cdots \cup L_{i,j-1}$ have been deleted. Figure 2 shows how levels are formed from DAG G_i. It's clear that vertex (7,i) is a leaf (has an out-degree of 0), and vertex (1,i) is a root (has an in-degree of 0).

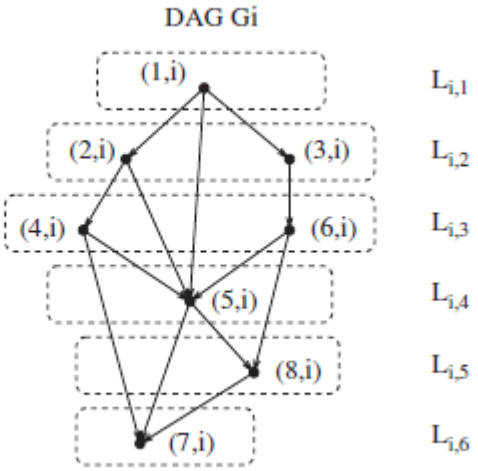

Figure 2. Levels of DAG G_i.

List Scheduling

In list scheduling, a priority may be assigned to each task (recall that a task is defined as the computation of a cell-direction pair). If no priorities are assigned to tasks, then all tasks have the same priority. A task is ready if it has not been processed yet, but all of it's ancestors in the dependence graph have been processed. For a timestep t, $R(t) \subset V \text{x} \{1, \ldots, k\}$ denotes the subset of tasks that are ready. $R_P(t) \subset R(t)$ denote the subset of tasks that are ready and allowed to be processed by processor P. The list scheduling algorithm proceeds such that for each timestep t, it assigns to each processor P the task of highest (or lowest) priority in $R_P(t)$, with ties broken arbitrarily. Processor P is idle at time t if $R_P(t)$ is empty.

Improved random delay algorithm O(log m log log log m)

1. **Preprocessing**: Construct a new set of levels L'_i for each direction i as follows:

- Construct new DAG H, that combines all G_i's, and viewing all copies of (v,i) of vertex v as distinct.
- Run standard list schedulingi algorithm on H with m identical parallel machines, with T being the makespan of this schedule.

2. For all i, choose $X_i \varepsilon \{0,\ldots,k-1\}$ uniformly at random.
3. Form a combined DAG G'' as follows: $\forall r \varepsilon \{1,\ldots,T+k-1\}$, define $L''_r = \bigcup_{\{i:X_i<r\}} L'_{i,r-X_i}$.
4. For each vertex v, choose a processor uniformly at random from $\{1,\ldots,m\}$.
5. Construct a schedule by processing layers $L''_1, L''_2, \ldots$ sequentially in that order.

Sweep Example

Here I will cover an example of a 2D processor layout with different aggregation factors, graphically and mathematically verifying the performance models for different partitioning parameters. It is important to note that in 2D, with P_z and A_z equal to 1, Eqs. 2, and 4 change slightly due to the lack of z dependence. Equation 13 shows the new expressions for N_{fill} and N_{tasks}:

$$N_{\text{fill}} = \frac{P_x + \delta_x}{2} - 1 + \frac{P_y + \delta_y}{2} - 1 \tag{11}$$

$$N_{\text{tasks}} = 4 N_m N_g \tag{12}$$

$$\tag{13}$$

where we see that in N_{tasks} we only multiply N_m by 4 instead of 8, because we now deal with quadrants, not with octants.

References

[1] Michael P. Adams et al. Provably optimal parallel transport sweeps on regular grids. In *International Conference on Mathematics and Computational Methods Applied to Nuclear Science & Engineering*, 2013.

[2] V.S. Anil Kumar et al. Provable algorithms for parallel generalized sweep scheduling. *Journal of Parallel and Distributed Computing*, 66:807–821, 2006.