
Marvin L. Adams (PI)  
Texas A&M University Nuclear Engineering  
mladams@tamu.edu

Collaborators  
Nancy Amato, TAMU Comp. Sci.  
Lawrence Rauchwerger, TAMU Comp. Sci.  
Paul Nelson, TAMU Nuclear Engr. & Comp. Sci.  
Bjarne Stroustrup (indirectly), TAMU Comp. Sci.  
Lots & lots of students

LabFest VI, May 10-11, 2005
We are mostly on track to accomplish what we proposed.

<table>
<thead>
<tr>
<th>Task</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAPL Infrastructure</td>
<td>Significant progress</td>
</tr>
<tr>
<td><strong>Transport Support Library</strong></td>
<td>Getting started</td>
</tr>
<tr>
<td>Adaptive Spatial</td>
<td>2D (RZ) underway (with R. Lazarov)</td>
</tr>
<tr>
<td>Adaptive Angular</td>
<td>2D (XY rect.) complete; RZ and XYZ next</td>
</tr>
<tr>
<td>Iterative Methods</td>
<td>SI / TSA’s / GMRES’s / BiCG complete</td>
</tr>
<tr>
<td>Partitioning and Scheduling</td>
<td>Ready to test on unstructured grids</td>
</tr>
<tr>
<td>Cycle Breaking</td>
<td>Ready to test on unstructured grids</td>
</tr>
<tr>
<td>Perf. Modeling &amp; Monitoring</td>
<td>Getting started</td>
</tr>
<tr>
<td>Adaptive Algorithm Selection</td>
<td>Getting started</td>
</tr>
<tr>
<td>Time dependence</td>
<td>Loops complete; ready for “real” math</td>
</tr>
<tr>
<td>Get students involved in ASC</td>
<td>5 NE + ~8 CS currently involved</td>
</tr>
</tbody>
</table>
We’re still happy with our approach to parallel transport...

• **Flexible** algorithm permits adaptive optimization.

• **STAPL** provides attractive infrastructure.
  - parallel containers (pGraphs, pVectors, pArrays)
    - handle communication “automatically”
    - pgraph can represent any grid
  - parallel executor calls “work functions,” respecting “dependence graphs”

• **Modular** code divides & conquers. Examples:
  - **Partition**: put subdomains on processors and build pgraph of grid
    (many options for this, including KBA-like)
  - **Aggregate**: cellsets, anglesets, groupsets
  - **Schedule**: determine sweep order for each unique angleset
    (break cycles if necessary; break other dependencies if desirable)
  - **Execute**: iterate on groupset-to-groupset scattering; for each groupset:
    - solve in parallel all (cellset,angleset) pairs (respecting scheduled dependencies)
    - iterate on within-groupset scattering
  - **Sweepchunk**: solve problem for given phase-space “chunk” = cellset × angleset × groupset. (Called by executor; operates on single processor.)
... but we are now working toward much greater flexibility and generality.

- More about this later!
- Basic idea: enable implementation of huge variety of discretization methods and iterative techniques with very minimal coding.

**Minimal Coding to implement new methods**

<table>
<thead>
<tr>
<th>Compositional Infrastructure</th>
<th>mostly assembling existing pieces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transport Support Library</td>
<td>loops, sweeps, I/O, ...</td>
</tr>
<tr>
<td>STAPL</td>
<td>transport-specific</td>
</tr>
<tr>
<td></td>
<td>generic</td>
</tr>
</tbody>
</table>
Transport problems are tremendously varied. So are possible parallel machines.

- Code has parameters and options that permit optimization for given problem and machine
  - **Partitioning**: KBA, Hybrid, METIS-type, etc.
  - **Aggregation**: cellsets, anglesets, groupsets
  - **Iterative schemes**
    - From block-Jacobi to pure sweep via scheduling decisions
    - various TSA preconditioners
    - GMRES (two options) and BiCGSTAB
  - **Discretizations**: Weighted Diamond, Various Corner-Balance Methods (soon), PWLD (soon), BLD and TLD (soon), some coupled with AMR (soon)

- Will soon get even more flexible.

- The strategy is to build “smart” decision-making into the code
  - *if problem is simple enough, models can estimate optimum parameters*
  - *For any problem, will iteratively refine parameters (one variation per time step, e.g.) to try to approach optimum*
  - *Combining simple models and iterative refinement looks attractive*
We made some decisions and adopted some guidelines.

- We use C++.
- 1st priority is “good” (maintainable, portable, etc.) code.
- We do what we can to be lean and fast given these constraints. 
  *We worry about this a lot. Tim Smith will show some results. We still have improvements to make. We are mostly pleased and quite optimistic.*
- We use general-purpose routines where possible. (BLAS)
- We expect compilers to become smarter, but we try to be smart ourselves.
- We are building the code to be an easy-to-use transport-methods testbed.
  ⇒ *I’m particularly excited about this!* 🎉
  ⇒ *This summer’s re-organization will be a testbed breakthrough, I think.*
  ⇒ *Code also serves to test parts of STAPL on a real application.*
Present code embodies some assumptions, limiting what we can easily test.

- Every problem has a spatial grid.
- Spatial domain is decomposed.
- For now, there is 1-to-1 mapping between processors and spatial subdomains. (This could be changed.)
- For now, the same angular discretization is applied across the entire spatial domain. We must relax this to allow locally adaptive quadrature sets.
- Code is built now for sweeping, but with infrastructure to handle other solution methodologies (such as those arising with elliptic-type operators).

These assumptions will be removed soon!
A simple example illustrates the basic approach of the current code.

Example problem:
2 spatial dimensions, 8 processors, 8x4 = 32 cell-sets, 8x4x14=448 cells, 12 angles, 4 angle-sets, 7 energy groups, 3 group-sets, same directions and angle-sets for each group-set.
This generates four Directed Acyclic Graphs (DAGs) for each groupset.

- Numbers are cellset indices
- Colors indicate processors
The optimal schedule, honoring dependencies, completes the sweep in 18 steps.

- 18 steps for 16 chunks/processor ⇒ par. effic. = 88% [w/o comm.]
- Opt. schedule for KBA partition has 22 steps ⇒ par. effic. = 73%
- If cellset or angleset size is halved, get 34 steps for 32 chunks/proc ⇒ 94%
- One cellset per sub-domain [old AMTRAN] ⇒ 67%
Adding a reflecting boundary to previous example just changes the DAGs.

angle-set B

angle-set A

angle-set C

angle-set D

Little else changes. Executor respects the dependencies and executes in parallel. (Reduced parallelism $\Rightarrow 88\% \rightarrow 73\%$)
Adding a second reflecting boundary (not opposite the 1\textsuperscript{st}) changes the DAGs some more.

To the executor, it’s just another DAG. Less parallelism, but still no cycles.
Adding an *opposing* reflecting boundary introduces cycles. Our strategy is to break them and sweep.

Lots of iterations if particles have lots of back-and-forth reflections (unless acceleration is really good). (Ask Zika!)
For reduced communication we could try Block Jacobi. Again, this just changes the DAGs.

- Lots of parallelism!
- Lots of iterations if particles cross sub-domain boundaries lots of times (unless preconditioner is really good).
Paul Nowak combines Block Jacobi with a sweeping, in a sense.

- Sub-domains **don’t wait** for new incident intensities (like Block-Jacobi).
- Sub-domains **do use latest** available incident intensities (like Block Gauss-Seidel).
- Heuristic rules attempt to order the anglesets for each sub-domain to maximize the probability of having new incident intensities (Sweep-like).
- Nowak has one cellset per sub-domain. *(Might profitably change this?)*
- **We could generalize by replacing rigid dependencies with priority values.** STAPL executor will eventually be able to accept this.
About sweeps and cycles ... 

- Cell cycles (mutually dependent sets of cells) exist on 3D unstructured grids.
- Cellset cycles can exist even when cell cycles don’t.
- Sub-domain cycles can exist even without cell or cellset cycles.
- **We must break all cycles or alter solution strategy.** We choose to break them, and we do so by using previous-iteration intensities.

- **Start at sub-domain level (local calculations).**
  - Determine “upstream” domain for each angleset
  - Mark all incoming cellset boundaries on that sub-domain to use old info for all angles in angleset

- Repeat at cellset level, then at cell level (local calculations).
- This is implemented in the code but not yet tested.
The transport solution is in “sweepchunk.”

- Ex.: Weighted Diamond on brick cells. Sweepchunk solves a (cellset, angleset, groupset) portion of the problem:
  
  loop over cells in the cellset (in the order given by the scheduler); for each cell:
  - get geometry info
  - loop over incident cell_faces; for each one:
    - invoke graph_edge method “get_psiface”
  - loop over angles in the angleset; for each angle:
    - do geometry arithmetic
    - loop over groups in the groupset; for each one:
      - solve for cell-avg and cell-exiting intensities
      - accumulate cell-avg angular-flux moments
  - loop over exiting faces; for each one:
    - invoke graph_edge method “set_psiface” to put exiting intensities into graph_edge.

- “Corner” discretizations use graph_edge methods that know element_face – to – element_face mapping on each cell_face.
Adding time dependence is straightforward. After this we can test adaptation ideas.

- Add loop over time steps outside of existing solver.
  ⇒ This is done and tested.
  ⇒ Now working on code to transition properly from one time-step to the next. Not difficult.

- Challenging part is adaptation. We plan to refine the following as time steps march on:
  ⇒ partitioning and scheduling
    • optimum can’t be found in realistic time
    • use heuristics initially, then modify as time goes on
  ⇒ spatial grid (adapt to resolve solution)
  ⇒ spatial discretization (sort of like p-refinement)
  ⇒ aggregations
  ⇒ iterative preconditioners and solvers
  ⇒ etc.

- Later, Nathan will present example of how to do this.
Initially, things will adapt only between time steps or outer iterations.

- Simplifies solution logic for adaptive time-dependent or eigenvalue problems.

- For example, could permit one level of coarsening or refining per step/iteration.
  
  *This means the grid is fixed for a time step’s (or outer iteration’s) calculation – much simpler than allowing change during the step.*

- We could permit small perturbations of partitioning or scheduling each time step.

- If adaptive in angle, change quadrature sets only between time steps or outer iterations.
Students are involved!

- Lots of students currently involved in the code:
  - 5 NE grad students at present (Bailey, Chang, Hawkins, Maslowski, H. Stone)
  - 5 CS grad students immersed in the code (Smith, Thomas, Tanase, Onica, An)
  - several additional CS grad students working on STAPL
  - undergrads also involved (Ness, Fagan, Jarrell)

- Several of these have spent time at labs:
  - Chang, Hawkins, Maslowski, Smith, Mathis
Still working on “near-term” list.

- Near-term efforts (months):
  - Get new “STAPL” version running smoothly
  - Get xy WD running
  - Implement infrastructure for xyz corner elements and polyhedral grids
  - Test mixed-mode communication
  - Re-organize code, separating transport-support library
  - Implement infrastructure for 2D corner elements and polygonal grids
  - Get “real” xyz UCB running
  - Get rz BLD running on rectangles
  - Get xy BLD running on rectangles
  - Get xy PWLD running on rectangles and coded for polygons
  - Get xy UCB running on rectangles and coded for polygons
  - Get rz UCB running on rectangles and coded for polygons
  - Get rz BLD running on user-defined AMR grid
  - Get xyz PWLD running on for polyhedra
  - Get xyz TLD running on bricks
  - Finish time-dependent coding.
Need to get going on the longer-term list.

- Get time-dependent problems running with *programmed grid motion* (not real mesh-moving physics)
  - *must efficiently alter partitioning and scheduling* (DAGs, cycle-breaking, load balance, parallel efficiency of sweep, ...)
  - *sweepchunks must be aware of time dependence*

- Adaptation of everything
  - *Partitioning and scheduling* (even if grid doesn’t move)
  - *Aggregation, groupsets, anglesets, and cellsets*
  - *Also includes grid if doing AMR*
  - *Also includes quadrature set if doing adaptive quadrature sets*
  - *Also may include parameters in spatial discretization or even different choices of spatial discretization* (e.g., *characteristic methods in streaming-dominated regions*)
My self-assessment is mixed.

• We’re doing some things well
  ⇒ *developing promising algorithms and strategies*
  ⇒ *developing extremely general approach to transport*
  ⇒ *developing STAPL*
  ⇒ *creating flexible, adaptive algorithms and implementations*
  ⇒ *integrating the efforts of lots of developers on one code (!)*
  ⇒ *getting students involved and excited about ASCI-type lab problems*
  ⇒ *leveraging (using other funding sources)*
  ✓ *Communication/collaboration with labs*

• We need to improve in other areas
  ⇒ *SQA (getting better)*
  ⇒ *Project planning*
  ⇒ *Faster work (but funding slow-down has hampered us)*