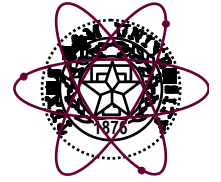


Toward a computational-transport testbed: *Work in Progress.*

**Lawrence Rauchwerger, Nancy Amato,
Tim Smith, Marvin L. Adams
Texas A&M University**

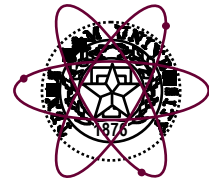
**LabFest VI
May 10-11, 2005**

Our testbed will have three key layers.



- **Generic Library (STAPL)**
 - ⇒ *containers (data structures)*
 - ⇒ *basic algorithms (sorting, graph operations, partitioning, BLAS, ...)*
 - ⇒ *pRange (connects algorithms to containers; expresses duality of data and task parallelism)*
- **“Domain-specific” Library (routines to support transport computations)**
 - ⇒ *quadrature sets*
 - ⇒ *efficient interfaces to existing libraries (like Trilinos)*
 - ⇒ *partitioning and scheduling tailored to transport (e.g., KBA)*
 - ⇒ *transport-specific containers (e.g., pGrid, cells, elements, geometry)*
 - ⇒ *I/O support*
- **Compositional Infrastructure (support for building programs from library)**

First step: Cast transport problems in generic form. Avoid restrictive assumptions.

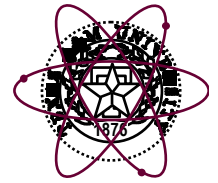


- The latter part is difficult!
- We don't want to rule out classes of problems. Our generic time-dependent equation is:

$$\frac{1}{v} \frac{\partial \psi}{\partial t} + \underline{\Omega} \cdot \underline{\nabla} \psi + \sum_{p=E, \theta, \gamma} \frac{\partial}{\partial p} \left[\left(\frac{dp}{dt} \right)_{acc} \psi \right] = \left(\frac{\partial \psi}{\partial t} \right)_{collisions} + S$$

- This permits self-interaction (as in gases) and acceleration (as with fields or fluid-frame equations), for example.
- Must allow operators and sources to depend on state variables determined by other physics.

We have identified six kinds of transport problems.



- Let \mathcal{L} represent loss (streaming + collision) and \mathcal{P} production (from fission for neutrons; zero for most other particles). Then

- **Coupled time-dep:**

$$\left\{ \frac{1}{v} \frac{\partial}{\partial t} + \mathcal{L}(t, \mathbf{s}) - \mathcal{P}(t, \mathbf{s}) \right\} \Psi(t) = Q(t, \mathbf{s})$$

- **Time-dep:**

$$\left\{ \frac{1}{v} \frac{\partial}{\partial t} + \mathcal{L}(t) - \mathcal{P}(t) \right\} \Psi(t) = Q(t)$$

- **α -eigenvalue:**

$$\{ \alpha / v + \mathcal{L} - \mathcal{P} \} \Psi = 0$$

- **k -eigenvalue:**

$$\mathcal{L} \Psi = \frac{1}{k} \mathcal{P} \Psi$$

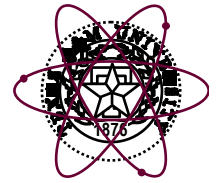
- **Critical search:**

$$\{ \mathcal{L}(p) - \mathcal{P}(p) \} \Psi = 0$$

- **Steady-state:**

$$\{ \mathcal{L} - \mathcal{P} \} \Psi = Q$$

Discretization: *operators* → **matrices** and **FUNCTIONS** → *vectors*.



- Matrix elements may depend on solution – our algebraic system can be nonlinear.

- **Coupled** time-dep:

$$\mathbf{A}(s)\psi = q(s)$$

- Time-dep:

$$\mathbf{A}\psi = q$$

- α -eigenvalue:

$$\{\mathbf{L} - \mathbf{P}\}\psi = -\frac{\alpha}{v}\psi$$

- k -eigenvalue:

$$\mathbf{L}\psi = \frac{1}{k}\mathbf{P}\psi$$

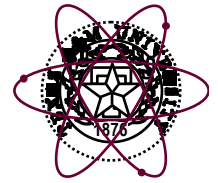
- Critical search:

$$\{\mathbf{L}(p) - \mathbf{P}(p)\}\psi = 0$$

- Steady-state:

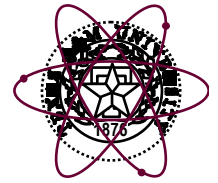
$$\{\mathbf{L} - \mathbf{P}\}\psi = q$$

There are infinite options for discretization of L and P (and A).



- **Spatial:**
 - ⇒ *First-order form: Finite Volume, Finite Element, Characteristic, ...*
 - ⇒ *2nd-order forms: Finite Volume, Finite Element, ...*
 - ⇒ *Integral forms: Collision Probabilities, Quadratures, ...*
 - **Energy: Multigroup, Function Expansion, Quadrature**
 - **Direction: Quadrature, Function Expansion, SPn**
 - **Time:**
 - ⇒ *with time steps: Finite Difference, Finite Element, Characteristic, ...*
 - ⇒ *something new?*
- Just a short list of examples!***
- **One mission is to identify and create routines to support a variety of interesting discretization methods.**
 - **Another is to make it easy for someone else to create routines to support *different* discretization methods.**

Discretization produces algebraic system. Solution technique solves the system.

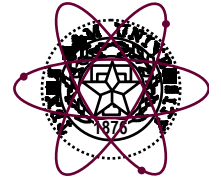


- Infinite options for solution technique. Examples include LAPACK, Gaussian elimination, iterative methods, etc.
 - ⇒ Will often nest iterations
 - ⇒ May use different techniques at different levels
- We seek to express a very general iterative approach that will admit essentially anything. Consider the system:

$$\mathbf{A}\psi = q$$

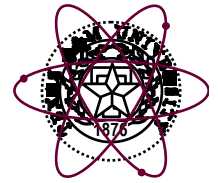
- We make two choices:
 - ⇒ *splitting*: $\mathbf{A} = \mathbf{A}_L - \mathbf{A}_R$, such that we can compute $[\mathbf{A}_L^{-1}\mathbf{A}]u$
 - ⇒ *solver*: algorithm for finding ψ from series of $[\mathbf{A}_L^{-1}\mathbf{A}]u$ calculations
- **Code should allow implementation of any splitting (preconditioning) and solver with minimal coding effort.**

We find common support needed by many discretization methods.



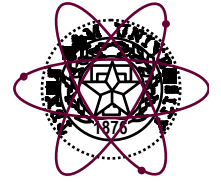
- **Many spatial discretization methods:**
 - ⇒ *are cell-based*
 - ⇒ *have one or more unknowns per cell*
 - ⇒ *model transport only across cell faces*
- **Many discretized collision operators:**
 - ⇒ *will use angular moments of the solution*
 - ⇒ *will build angular sources from moments*
- **As we design supporting routines, we try not to rule out other discretization methods.**
 - ⇒ *must allow easy extension to vertex-based methods, for example, or long-characteristics methods*

We find common support needed by many preconditioners and solvers.



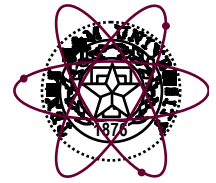
- **Preconditioning steps can include**
 - ⇒ *sweeps*
 - ⇒ *solution of smaller systems (nested iterations)*
 - ⇒ *one-cell one-energy solution*
 - ⇒ *one-cell all-energy solution*
 - ⇒ *block-Jacobi-type operations*
 - ⇒ *...*
- **Solvers often need**
 - ⇒ *scratch arrays*
 - ⇒ *inner products with various definitions (not just usual “dot”)*
 - ⇒ *BLAS-type operations*
- **We aim to provide support for all of this and to make it easy to add support for other options.**

As we explore, we are building a web site.

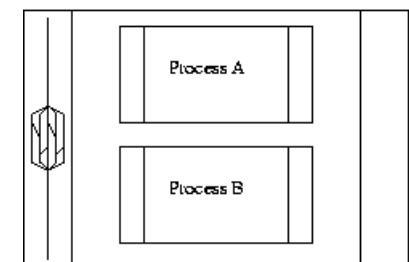
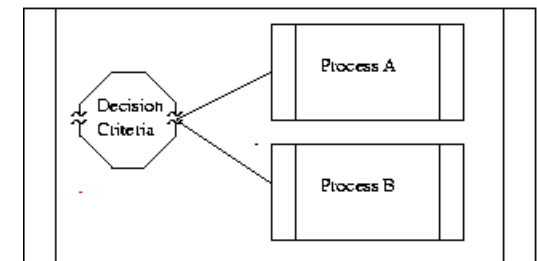
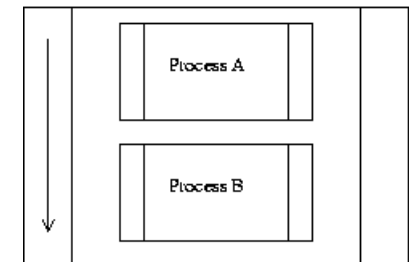
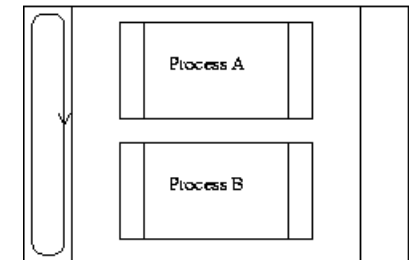


-
- I'll show you some pages if you promise to remember that this is an early stage of W.I.P.!

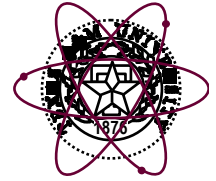
“Compositional infrastructure” enables program building, modification, etc.



- Calls to I/O routines
- Compositional code for basic execution constructs
 - ⇒ *Four identified so far: Loop, Sequence, Case, Coupled Set*
 - ⇒ *More on this later*
- Calls to performance-monitoring tools
- Calls to performance models
- Interface with tools that select algorithms or parameters
 - ⇒ *Nathan Thomas will give examples*
- Automatic code generation from user input
 - ⇒ *User may want to test something the developer didn't imagine*
 - ⇒ *graphical input will eventually be an option*



“Domain-Specific” library builds on STAPL and supports transport computations.



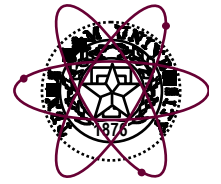
- **Quadrature library**

- ⇒ *can build variety of quadrature sets (weights, cosines, differencing parameters) for various geometries (XYZ, XY, RZ, slab, spherical, ...)*
- ⇒ *contains “moment-to-discrete” and “discrete-to-moment” operators needed for sources*
- ⇒ *builds on STL vectors*
- ⇒ *easily extensible*

- **Transport-specific containers**

- ⇒ *pGrid: transport-specific pGraph. For cell-based transport:*
 - *graph_vertex = cell*
 - *graph_edge allows cell-to-cell communication*
- ⇒ *cell:*
 - *contains nuclide number densities and list of elements*
 - *different types supported (tets, triangles, polygons, rectangles, polyhedra, ...)*
 - *easily extensible*
- ⇒ *element: contains sources and solution variables*
- ⇒ *extensible (for example, to vertex-based schemes like CFEMs)*

“Domain-Specific” library (cont.)



- **Partitioning and scheduling tailored to transport**
 - ⇒ *KBA, Hybrid, ...*
 - ⇒ *Heuristic schedules for sweeping arbitrary grids*
 - ⇒ *Algorithms for “breaking” dependencies*
 - decision-making algorithms
 - implementation coding
- **Efficient interfaces to existing libraries (like Trilinos)**
- **I/O support**
 - ⇒ *GUI to describe transport problems*
 - ⇒ *Routines to process transport input*
 - ⇒ *Support for various transport outputs*
 - ⇒ *easy extensibility*

Standard Template Adaptive Parallel Library is a superset of STL.

