TAXI Code Overview and Status

Timmie Smith
November 14, 2003
Transport Problem Example

- Oil Well Logging Tool
- Shaft dug at possible well location
- Radioactive sources placed in shaft with monitoring equipment
- Simulation allows for verification of new techniques and tools
Problem Specification

- Given a configuration at time $T$
  - Spatial domain divided into cells
  - Energy range divided into groups
  - Quadrature set chosen for angular integrals
  - Fixed Sources, Boundary Conditions, and Initial Conditions

- Compute particle flux at time $T + \Delta T$
  - Particle flux at time $T + \Delta T$
  - Reaction rates, etc., during time step
TAXI Data Structures

- Spatial domain decomposed into Cells
  - Cells are stored in a grid
  - Cell contains Elements
- Elements are the base spatial data structure
- Material definitions stored as in time-dependent neutronics problems
  - Number densities stored by cell
  - Microscopic cross sections stored by isotope
Aggregation in TAXI

- Angle Sets
  - Angles in AngleSet generate same dependences
- Energy Group Sets
- Cell Sets
  - Code computes aggregation
  - User can specify an aggregation if they want
- Work function operates on GroupSet-AngleSet-CellSet combination
STAPL Overview

- Standard Template Adaptive Parallel Library
  - Superset of C++ STL
  - pContainers – generic distributed containers
  - pAlgorithms – generic parallel algorithms
  - pRange - abstract view of partitioned data
    - binds pContainers and pAlgorithms
    - view of dist. data, random access, data dependences

Example: parallel sorting a distributed vector

```cpp
stapl::pvector<int> pv(100);
... initialize pv ...
stapl::p_sort( pv.get_prange() );
```
The STAPL Programming Environment

User Code

pAlgorithms  pRange  pContainers

pRange

ARMI

Pthreads  OpenMP  MPI  Native
Grid Overview

- ArbitraryPGrid is derived from
  - BaseSpatialGrid
  - pGraph
- Grid is templated on
  - cell type (brick, polyhedral, etc.)
  - element type (whole_cell, corner, ...)
- It contains instantiations of
  - cells (one per graph_vertex)
  - faces (really graph_edges)
- It contains methods for
  - putting outgoing-surface intensities into graph_edges
  - getting incoming-surface intensities from graph_edges
  - both of these are (angleset×groupset) at a time
Cell Overview

- Cell class doesn’t contain much.
  - list of isotopes and their number densities
  - list of elements
  - list of vertex indices
- The “element” concept gives us great generality and flexibility.
- There is one element for each “fundamental” spatial unknown. For example:
  - simple methods with one unknown per cell use the “whole-cell” element
  - methods with one unknown per vertex per cell use the “corner” element
Element Overview

- Elements contain:
  - volumetric sources (scattering and total)
  - the solution (angular moments for generating the scattering source; also the angular intensities in a time-dependent problem)

- Most of the code is unaware of problem geometry or discretization method.
  - It just loops over elements
  - Work function processes the Cells in a STAPL pRange
TAXI Setup

- Energy and Angle objects constructed
- Grid partition decided
  - Specialized STAPL scheduling algorithms used
- Partition information distributed
- Grid constructed in parallel
  - Cells and Elements created and inserted
- pRanges built on grid
  - Specialized STAPL scheduler determines pRanges
  - pRange dependence graphs are directed
TAXI Implementation History

- C++ selected for implementation
  - Wanted to use STL and STAPL
- Prototype implemented with C++ and MPI
  - STAPL components weren’t fully implemented
  - STAPL didn’t support distributed memory systems
- STAPL has advanced
  - ARMI Runtime System provides a consistent base for development
    - Allows STAPL to run on distributed memory systems
  - pContainers provide data distribution and parallel operations
  - pRange supports hierarchical decomposition of data
  - Scheduling components incorporated into STAPL
STAPLing TAXI

- Custom Data Structures
- Distributed 3D vector class
  - User managed distribution
  - 3D indexing (e.g. foo[x][y][z])
  - Stored Cells and CellSets
- CellSet class
  - Explicit list of graph vertex ids (Cells in the CellSet)
  - List of ids provided cell sweep order
STAPLing TAXI

- pGraph stores Cells
  - Each pGraph vertex is a Cell
  - pGraph edges identify neighboring cells
  - Arbitrary grids now possible
  - 2D grids also possible

- pRange used to represent CellSet
  - General representation of dependences between CellSets
  - Multiple execution orders of cells in subrange possible
- Numbers are cellset indices
- Colors indicate processors
Adding a reflecting boundary

angle-set B

angle-set A

angle-set C

angle-set D
A second reflecting boundary
Opposing reflecting boundary
STAPLing TAXI

- Abstracting Communication
  - Explicit MPI calls in driver function replaced with calls to ARMI methods
  - pGraph uses ARMI to transfer information between threads when necessary
- ARMI fences replace MPI barriers
  - Fences required during problem initialization
  - Most fences handled inside STAPL
Effects of STAPLing

- Problem Time to Completion
  - Average 2.16 times slower but improving rapidly
  - Currently 15 global barriers in the ASCI code
  - There is opportunity for improvement
Effects of STAPLing

- Reduced code size
  - Code reduced ~6,000 lines.
    - Currently 13,000 lines and shrinking
  
- Elimination of classes
  - MPI wrappers
  - XYZSpace
  - CellSet

- Code Reorganization
  - Problem class reorganized
  - Eliminated explicit grid distribution
  - Use of pRanges
void driver()
{
    BaseProblem* problem = read_problem(*input_stream, *output_stream,
                                        *error_stream);
    problem->solve();
    problem->report();
}

BaseProblem* read_problem(istream& input_stream, ostream& output_stream,
                           ostream& error_stream){
    ProblemInput* pinput = new ProblemInput(input_stream);
    Hex_cell_creator creator(*pinput);
    BaseEnergyAggregation eggs;
    XMLObject xml_input = pinput->getTagObject("prototype")[0];
    eggs.init_from_input(xml_input, output_stream);
    asci_preprocessing::ASCI.Regular_pRange_Preprocessor* sched =
        build_brickscheduler(creator.dimx(), creator.dimy(), creator.dimz(), eggs,
                             *pinput);
    BaseProblem* bp = new GenericSweepProblem<LogicallyRectangularGrid,
                                                Wt_Diamond_Diff_Method>(pinput, creator, eggs, sched, output_stream,
                                                                             error_stream);
    return bp;
}
Questions?