

Particle Transport Code Overview

Silvius Rus

Texas A&M University

<http://www.cs.tamu.edu/research/parasol/asci>

Presentation Structure

- Infrastructure
 - Generality
 - Hierarchical design
- Performance
 - Parallel execution
 - Implementation issues

Problem Specification

- Given configuration at time T :
 - Spatial discretization: **Grid**
 - Collection of **Cells**
 - Contains geometry and topology information
 - Energy and angular discretization
 - Fixed sources and boundary conditions
- Compute:
 - Particle flux at time $T+\Delta T$

Analysis

- Framework: unifying discretization methods
 - Element vs. Cell
 - ElementMap vs. Grid
- Solvers: generic operations
 - Scattering
 - Sweep
 - Convergence

Basic Design

- Set of basic entities:
 - Grid and cells
 - Element map and elements
 - Material repository
 - Energy and angular discretization
 - Generic solver
- Generality
 - Templated structures and routines
 - Polymorphism and virtual functions

Basic Design

Grid

	Cell

ElementMap

	Element	Element
	Element	Element

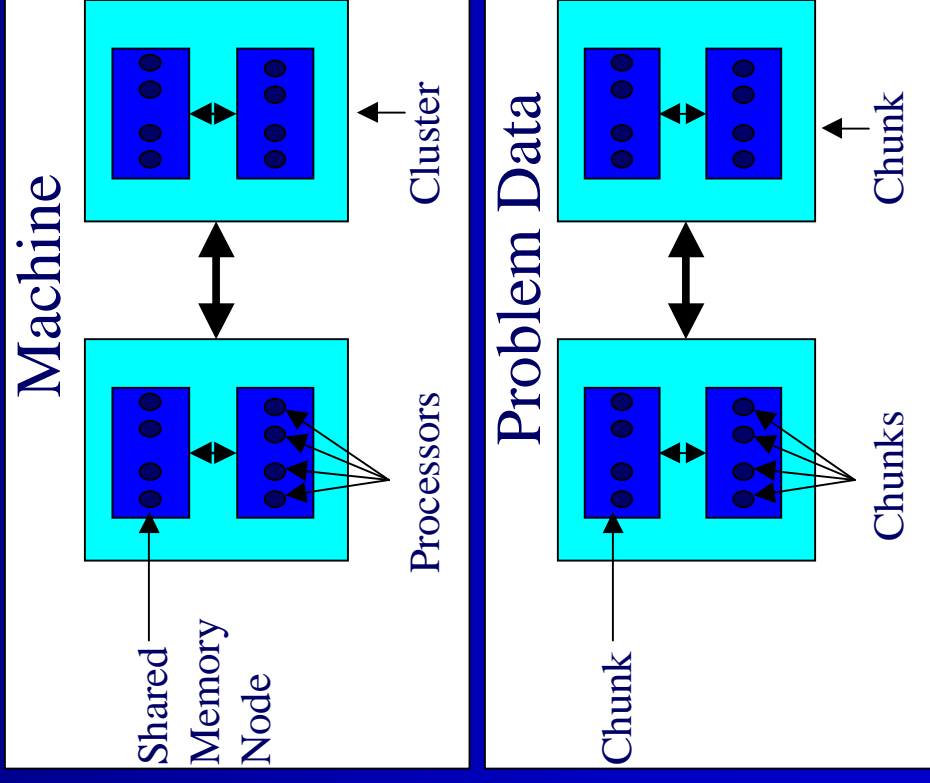
- **ElementMap** may borrow topology info from **Grid**
- **Element** may borrow geometry info from **Cell**

Hierarchical Design

- Computational resources
 - Memory and interconnection network hierarchy
- Data structures mapped on machine topology
- Main data structure:
 - **Chunk = (Cell Set, Energy Group Set, Angle Set)**

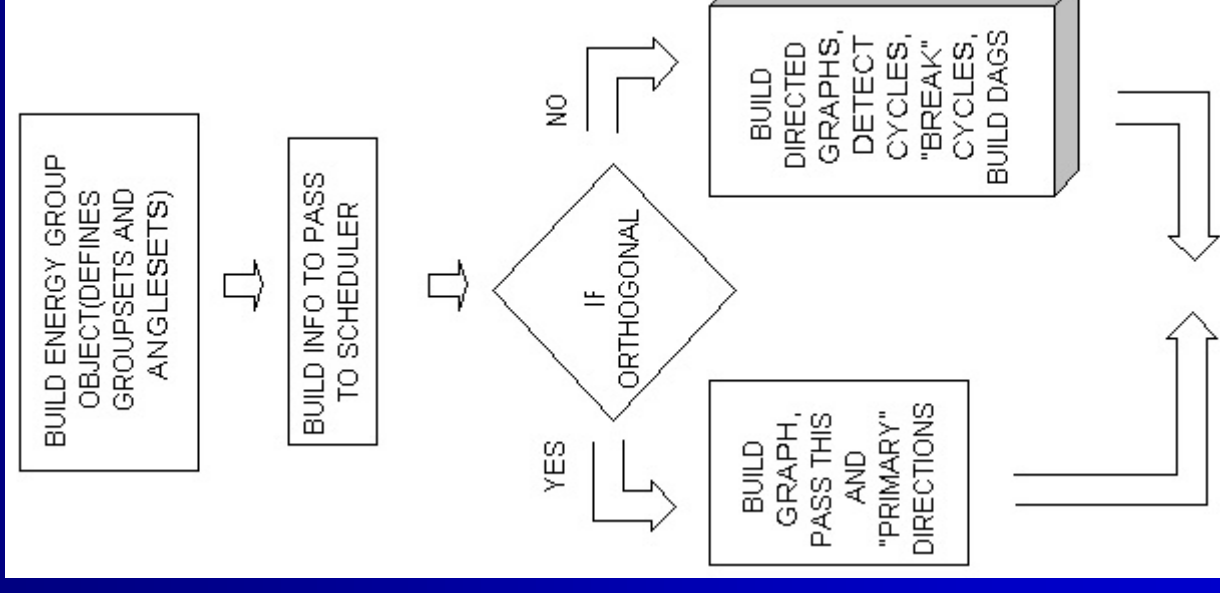
Hierarchical Design

- **Chunk**
 - Recursive data structure
 - As large as the whole problem space
 - As small as (**Cell, Energy Group, Angle**)
 - Basis for:
 - Memory allocation
 - Computation
 - Communication
 - Synchronization



Setup Data Structures

- A hierarchical partition is computed first
- Distribution across resources
- Computed by an external module
 - *Scheduling routines*

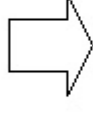


Setup Data Structures

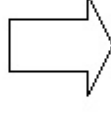
- Build grids distributed across the hierarchy
- Set up communication buffers if needed
- Set pointers to standardize Sweep interface
- Allocate message buffers if needed

FURTHER PREPARATION

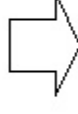
BUILD COMPLETE GRID OBJECTS
(EVENTUALLY, SEPARATE FOR EACH SHARED-MEMORY "NODE")



BUILD CELLSETS AND CHUNKS



SET POINTERS (PSIFACE_EXIT IN EACH ELEMENT)



BUILD MESSAGE BUFFERS

Algorithm Design

Loop across Group Sets

Iterate until convergence :

FOR EGS in Energy Group Sets

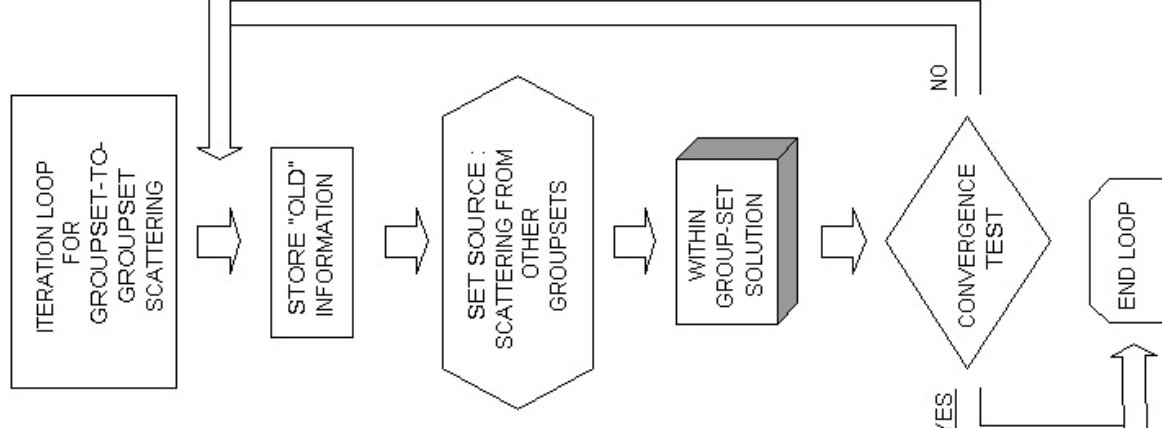
Store old information

Compute in-scattering from other Group Sets

Within EGS Solution

Convergence test across Group Sets

PROBLEM SOLUTION



Algorithm Design

Within a Group Set *EGS*

Iterate until convergence:

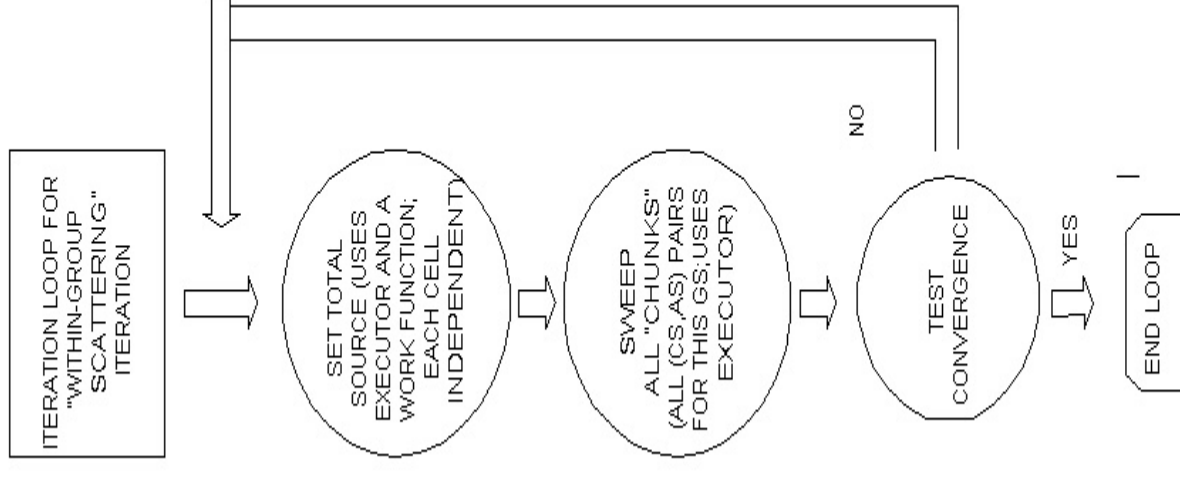
Compute scattering within *EGS*

FOR (*AS*, *CS*) in (*Angle Sets x Cell Sets*)

Sweep_Chunk(*EGS*, *AS*, *CS*)

Convergence test within *EGS*

GROUP LOOP



Algorithm Design

- Common interface between data structures and algorithms
 - Simple work functions:
 - In-scattering from other group sets
 - Scattering within group set
 - Sweep
 - Check convergence
 - Generic work functions
 - Specialize them for particular problem types only if needed
 - E.g. *Store old information vs. Sweep*
 - Use virtual function mechanism for selection

Performance – Single Processor

- Memory locality - tiling
 - **Chunk** - unit of
 - Memory allocation
 - Computation
- } Locality

Parallel Processing

- Goals:
 - Scalable performance
 - Exploit particular machine characteristics without losing generality: *Adaptive techniques*
- Abstraction mechanisms:
 - Internal: **Chunk** - unit of
 - Communication } *Communication*
 - Synchronization } *aggregation*
 - *Ideal size: trade-off between communication count and stalls*
 - External: use *STAPL* to express parallelism
 - *STAPL = Standard Template Adaptive Parallel Library*

Abstracting Parallelism

- Goal: select most efficient parallel constructs
 - Apply goal at every machine level recursively
 - Use both MPI and OpenMP within the same instance
 - Use architecture and application independent (thus portable) user level constructs
 - Do not sacrifice performance for generality: **Adaptive**
- Abstraction mechanism
 - Hide low level details
 - Data distribution
 - Parallel execution
- Solution: **STAPL**

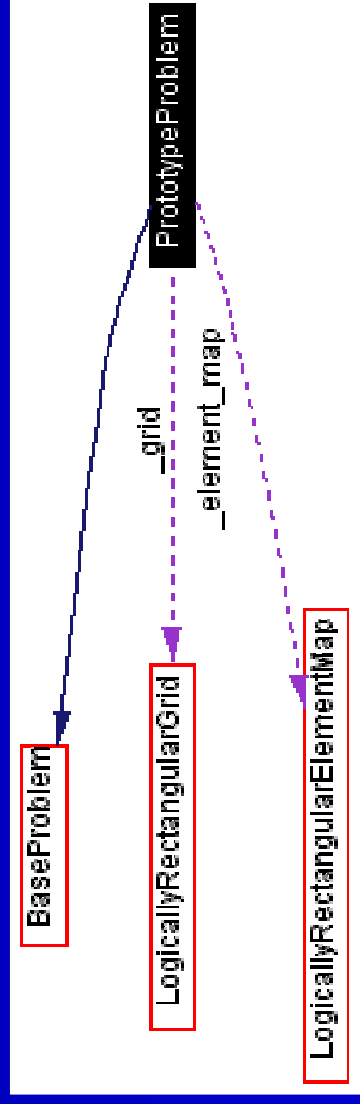
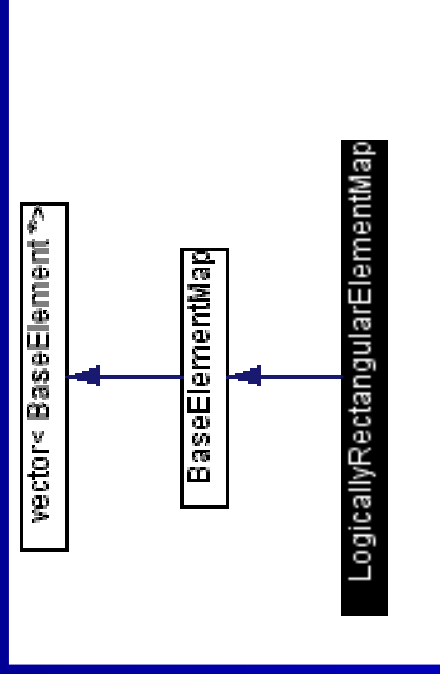
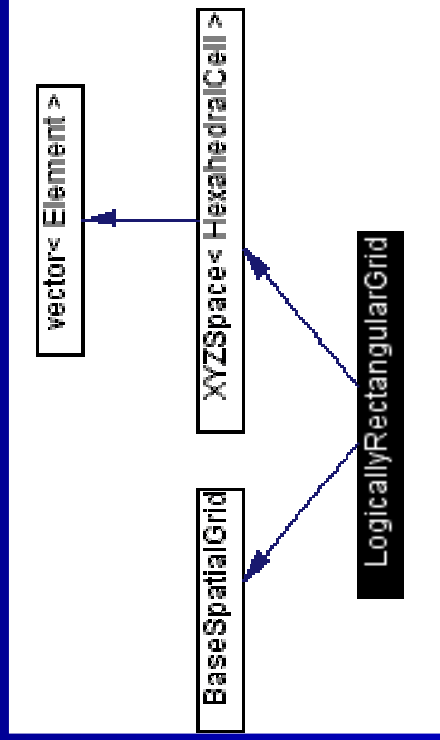
Parallel Processing

- Common interfaces to standalone modules:
 - **Scheduling module**
 - Generates data distribution and aggregation
 - Creates execution schedule
 - **Executor**
 - Manages parallel execution
 - Part of *STAPL*

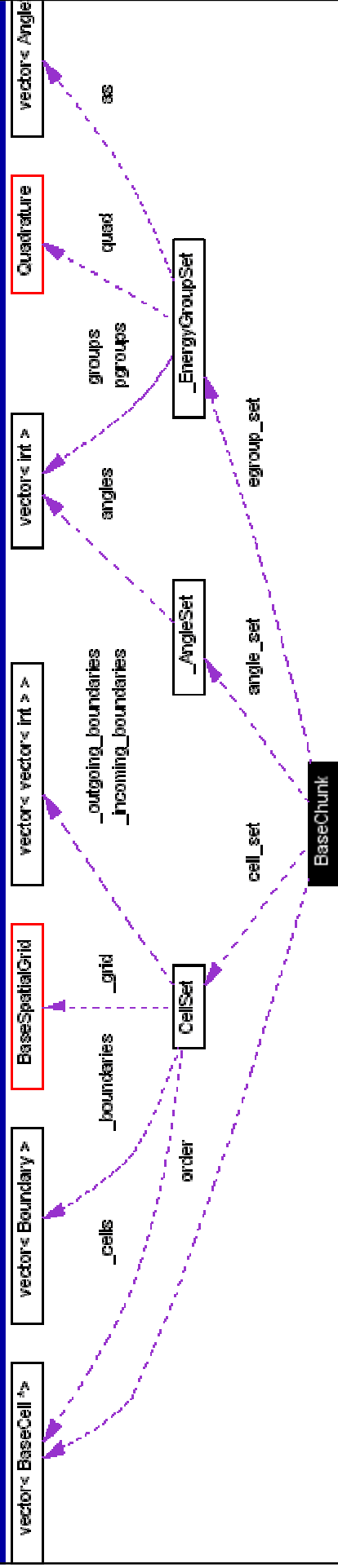
Current Implementation

- Horizontal structure
 - Implemented all basic data structures
 - Grid, Cell, ElementMap, Element, Material, Energy and Angular Discretization
 - Chunk, Boundary
 - Defined relations between data and algorithms
 - Partitioner, Scheduler, Executor standard interfaces
- Vertical prototype
 - Logically rectangular grid
 - *Diamond Difference* discretization method
 - Scheduling: *KBA, Volumetric, Hybrid*
 - Execution: *STAPL* using *MPI*

Prototype Structures



Prototype Implementation of *Chunk*



Implementation Issues

- Programming language: C++
- Libraries: *STL*, *STAPL*
- Input/output formatting: *XML*
- Source file versioning system: *CVS*
- Semi-automatic documentation generator:
Doxygen

Current Work

- The prototype showed us the need for:
 - Generic data structures designed for parallel processing (via *STAPL*)
 - Hierarchical
 - Distributed
 - Abstraction mechanism for specifying parallelism
 - Machine and system independent
 - Adaptive techniques to avoid trading performance for generality
- Need to implement other problem types