OPTIMIZING THE LULESH STENCIL CODE USING CONCURRENT COLLECTIONS

WOLFHPC'15

Chenyang Liu

Milind Kulkarni

11-16-2015

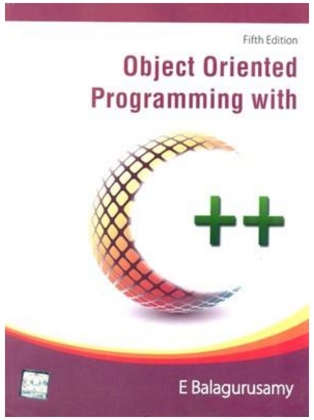
PURDUE
UNIVERSITY

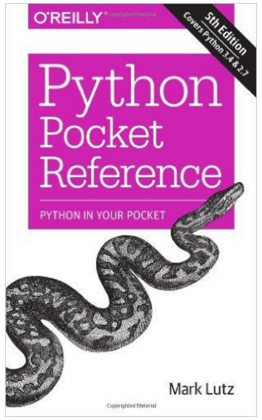
Overview

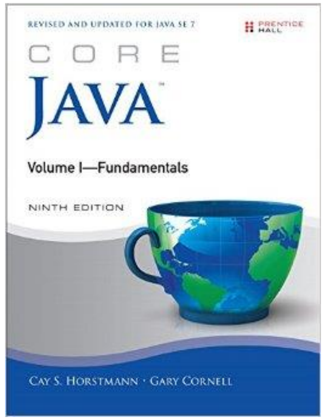
- Background
- LULESH
- Concurrent Collections
- CnC LULESH
- Optimizations
- Experiments
- Conclusion

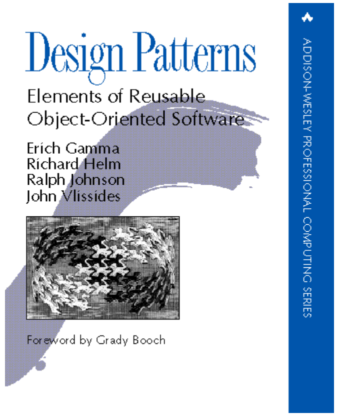
Scientific Programming…

Programmer

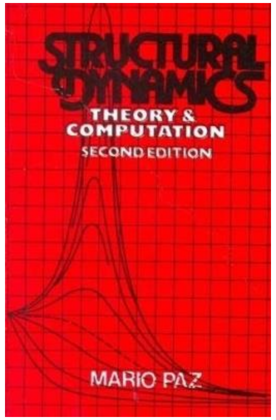


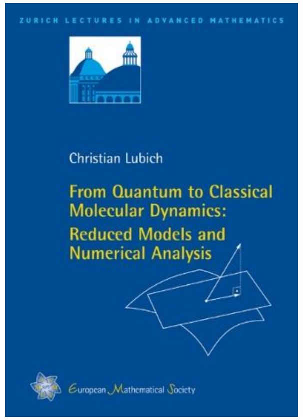


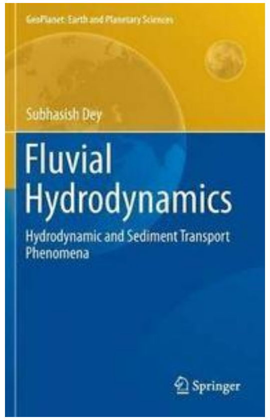


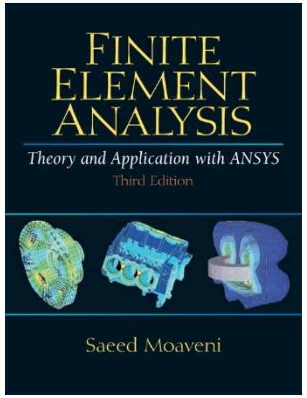


Scientist



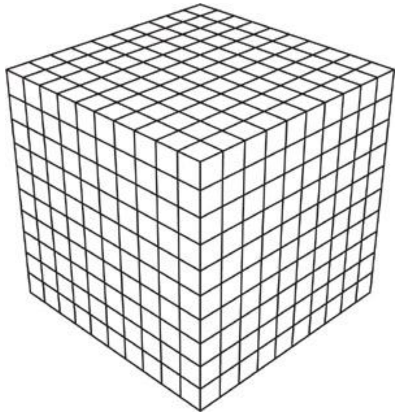




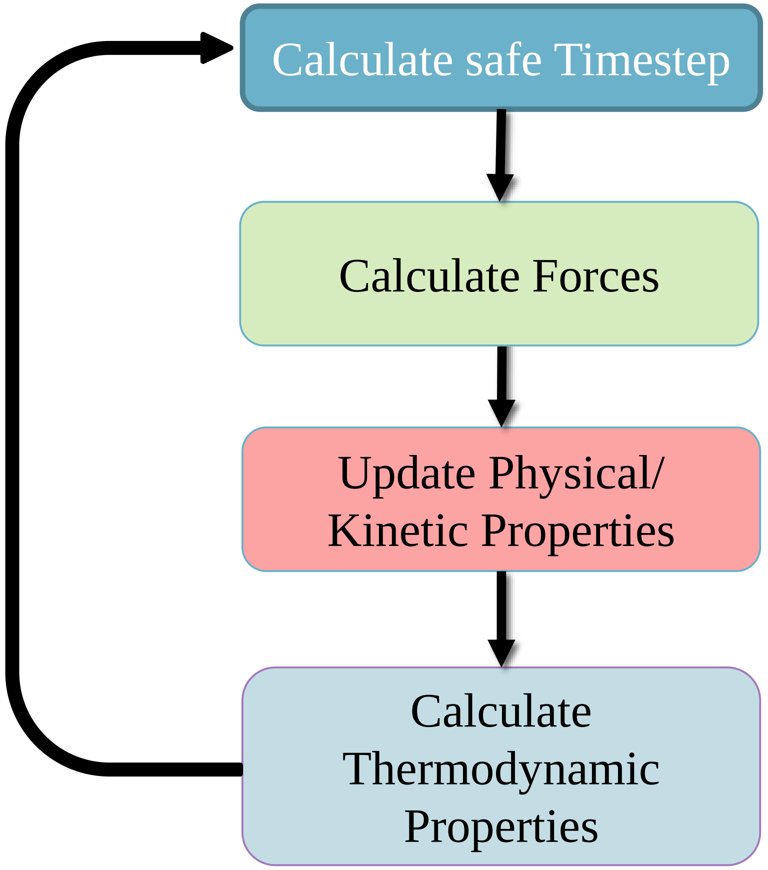


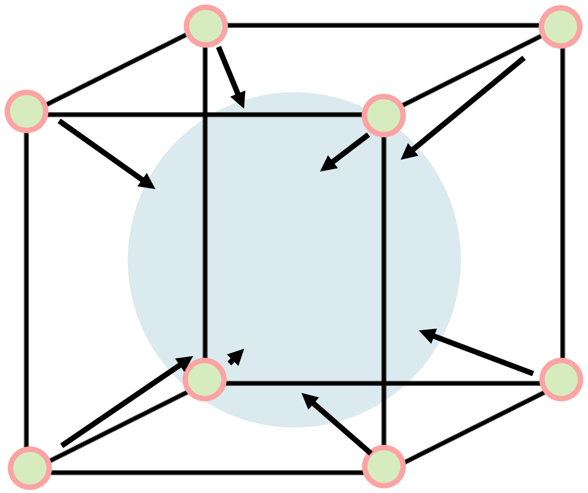
LULESH Stencil

- **LULESH: Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics**
  - 2010 DARPA UHPC challenge problem
- 3D stencil
  - Hexahedral mesh with 2 centerings
  - Nodal Computations
  - Element Calculations
  - Highly data dependent algorithm

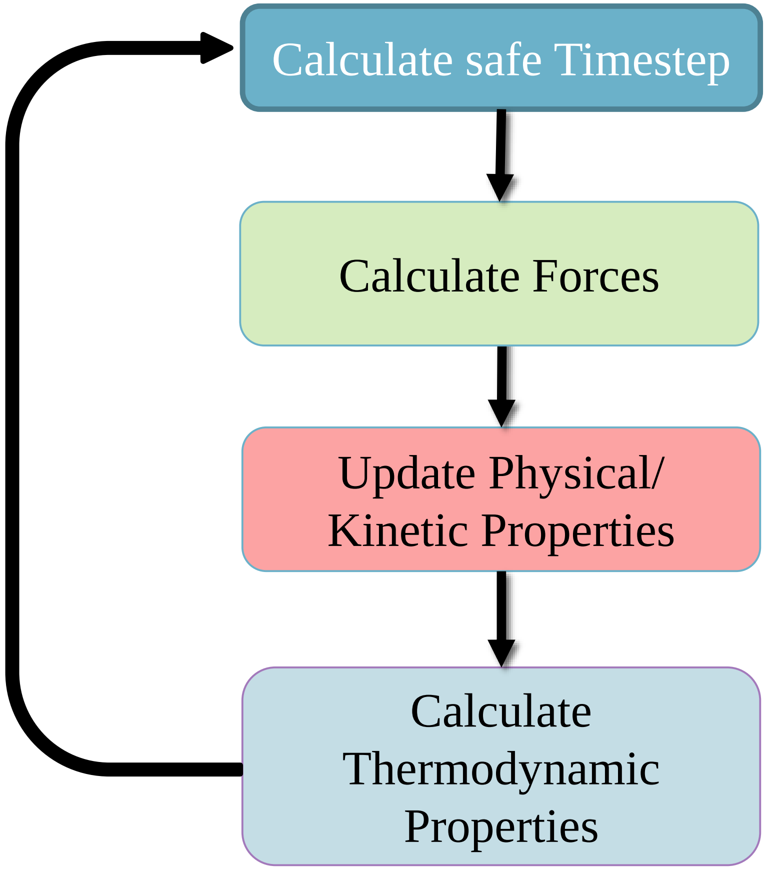


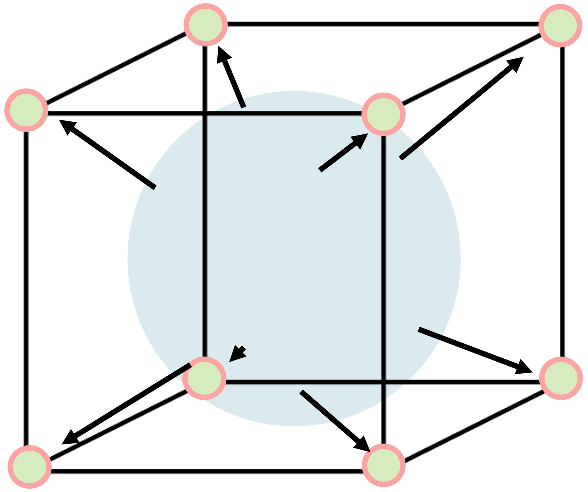
LULESH Algorithm



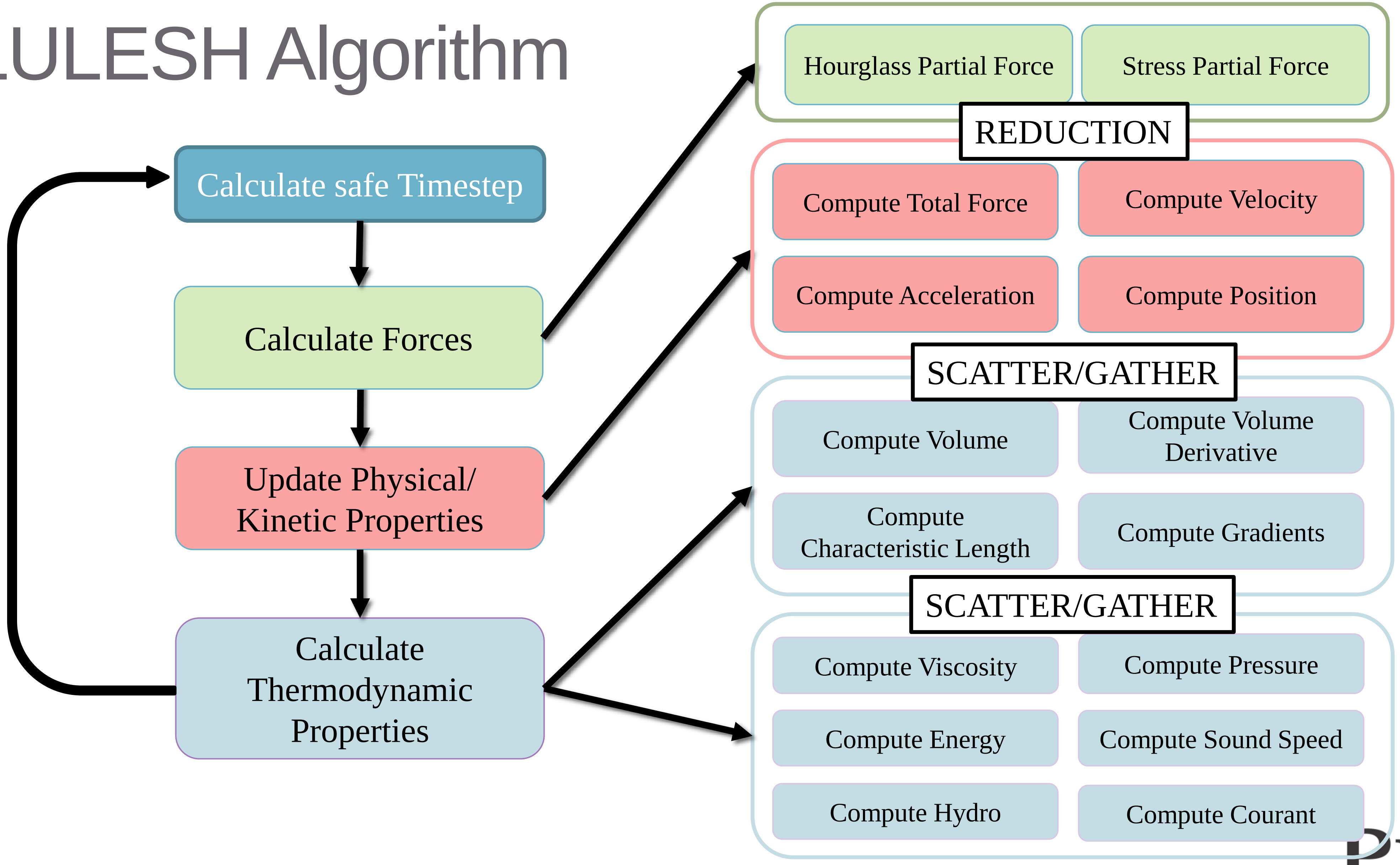


LULESH Algorithm



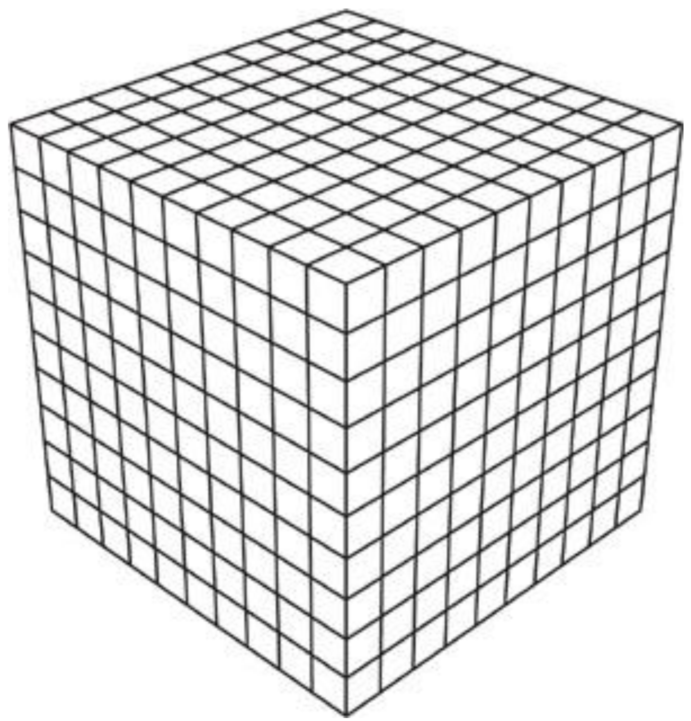


LULESH Algorithm



LULESH Challenges

- Programming the Stencil
  - Not a trivial problem
  - Complex dependencies
- Exploiting Parallelism
  - Managing communication patterns
- Data Layout and Tiling
  - Block sizing and cache
  - Optimizing memory performance
  - Element/Node size differences

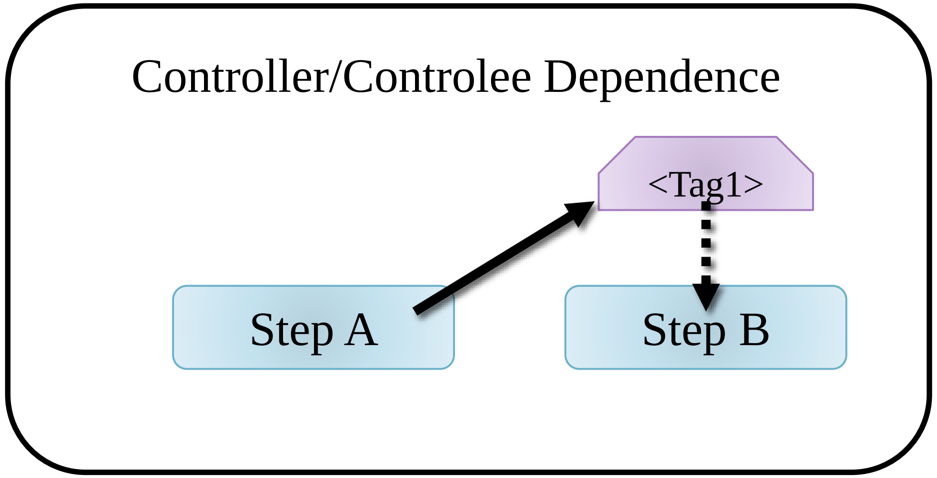


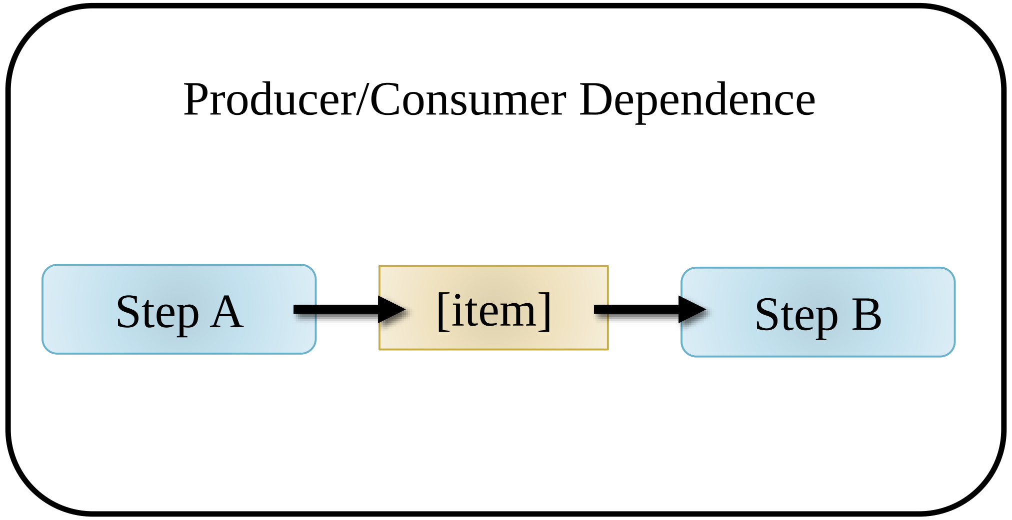
Concurrent Collections (CnC)

- Separation of Concerns Philosophy
  - Scientists focus on Algorithm
  - Programmers focus on Performance
- Programmability with Performance
  - Programmer expresses computation, runtime exploits parallelism
  - Low level tuners for performance
- Programming Model constraints:
  - Must use CnC Collections
  - Extension of common programming languages
  - Explicit dependencies

The Collections

- CnC Model:
- Step Collections
  - Stateless computation blocks
  - Dynamically instantiated using tags
- Item Collections
  - Key/Value lookup
  - Dynamic Single Assignment
  - Explicit dependencies (Get/Put)
- Tag Collections
  - Associated with every step collection
  - Dictates program control flow

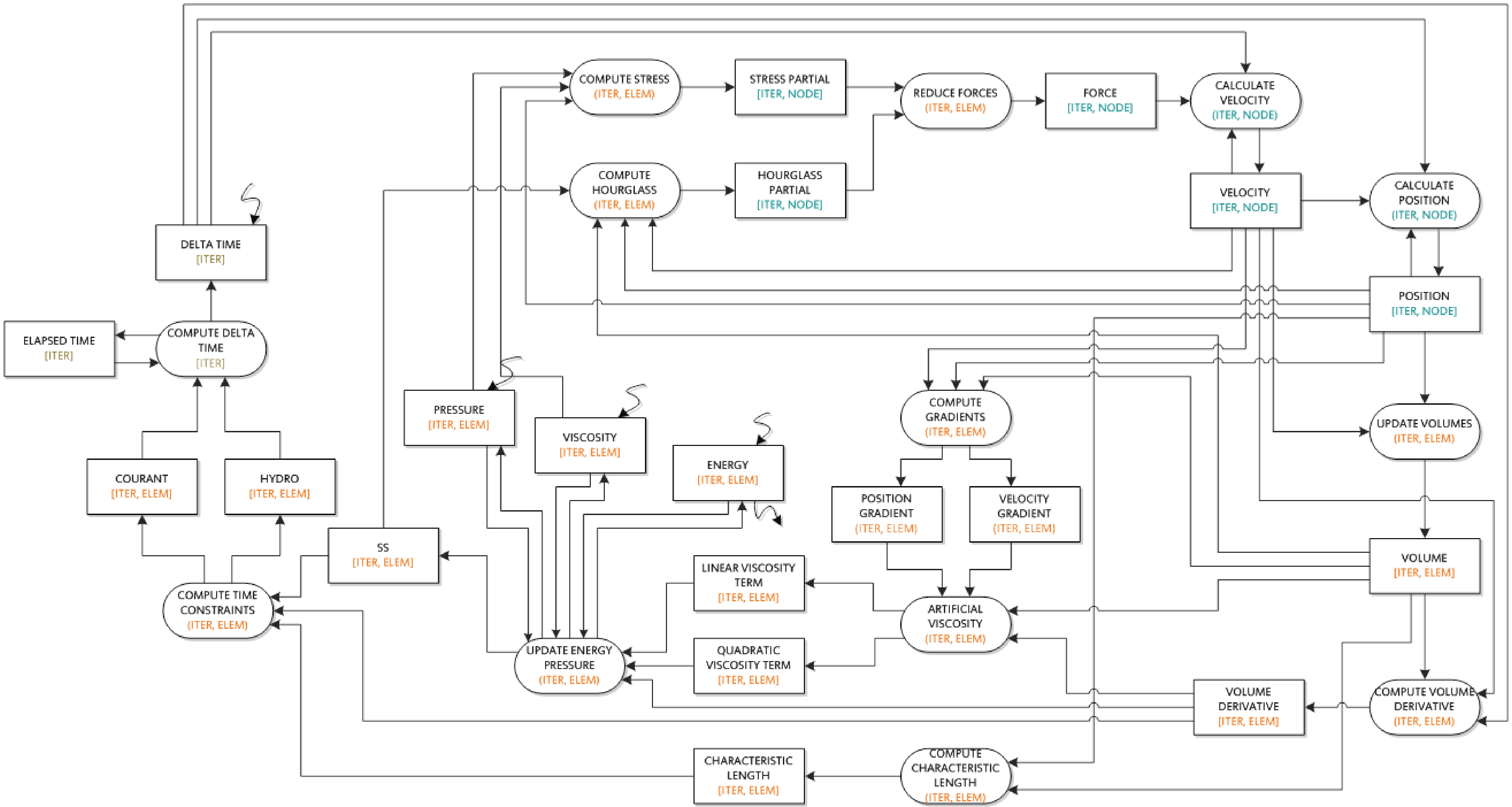




CnC LULESH

- CnC simplifies:
- Programming the Stencil
  - Domain expert can write the algorithmic program
  - High-level CnC specification
- Exploiting Parallelism
  - CnC handles the parallelism, maps to most platforms
- CnC Tuners (low-level)
- CnC does not solve:
- Data Layout and Tiling
  - Optimizing block sizing
  - Achieve good granularity

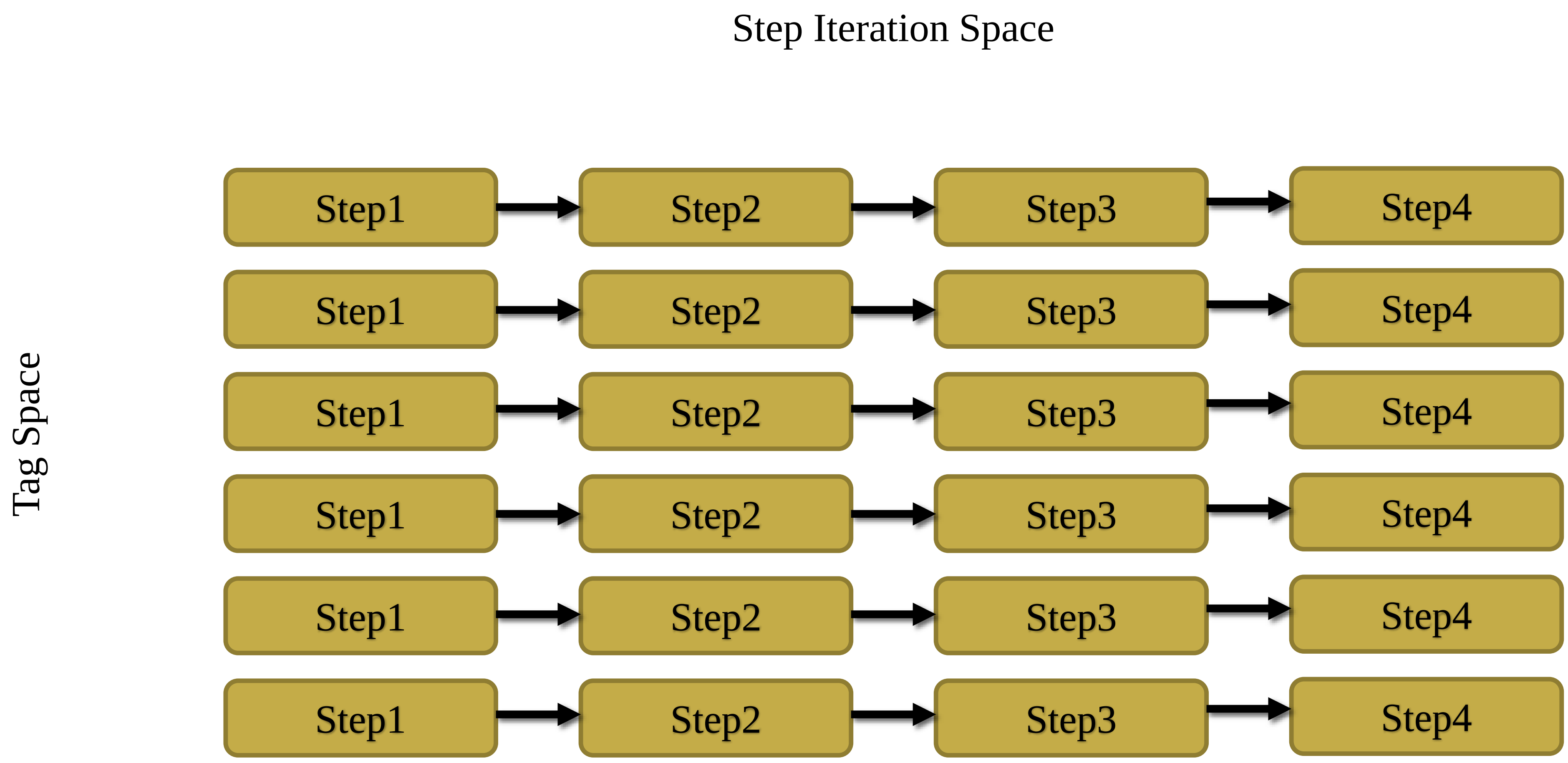
```
struct lulesh_context:public
    context<lulesh_context>{

// Step Collections
step_collection<compute_dt>
    step_compute_dt;
step_collection<reduce_force>
    step_reduce_force;
...

// Item Collections
// per node items
item_collection<pair,vector>force;
item_collection<pair,vertex>position;
item_collection<pair,vector>velocity;
// per element items
...

// Tag Collections
tag_collection<pair>iteration_node;
tag_collection<pair>iteration_element;
tag_collection<int>iteration;
...

// Producer Dependencies
step_compute_dt.consumes(dt);
...

// Consumer Dependencies
step_compute_dt.produces(dt);
...
```


COMPUTE STRESS (ITER, ELEM)
STRESS PARTIAL [ITER, NODE]
REDUCE FORCES (ITER, ELEM)
FORCE [ITER, NODE]
CALCULATE VELOCITY (ITER, NODE)
COMPUTE HOURGLASS (ITER, ELEM)
HOURGLASS PARTIAL [ITER, NODE]
VELOCITY [ITER, NODE]
CALCULATE POSITION (ITER, NODE)
DELTA TIME [ITER]
POSITION [ITER, NODE]
ELAPSED TIME [ITER]
COMPUTE DELTA TIME [ITER]
PRESSURE [ITER, ELEM]
VISCOSITY [ITER, ELEM]
COMPUTE GRADIENTS (ITER, ELEM)
UPDATE VOLUMES (ITER, ELEM)
ENERGY [ITER, ELEM]
COURANT [ITER, ELEM]
HYDRO [ITER, ELEM]
POSITION GRADIENT (ITER, ELEM)
VELOCITY GRADIENT (ITER, ELEM)
SS [ITER, ELEM]
VOLUME [ITER, ELEM]
LINEAR VISCOSITY TERM [ITER, ELEM]
COMPUTE TIME CONSTRAINTS (ITER, ELEM)
ARTIFICIAL VISCOSITY (ITER, ELEM)
UPDATE ENERGY PRESSURE (ITER, ELEM)
QUADRATIC VISCOSITY TERM [ITER, ELEM]
VOLUME DERIVATIVE [ITER, ELEM]
COMPUTE VOLUME DERIVATIVE (ITER, ELEM)
CHARACTERISTIC LENGTH [ITER, ELEM]
COMPUTE CHARACTERISTIC LENGTH (ITER, ELEM)

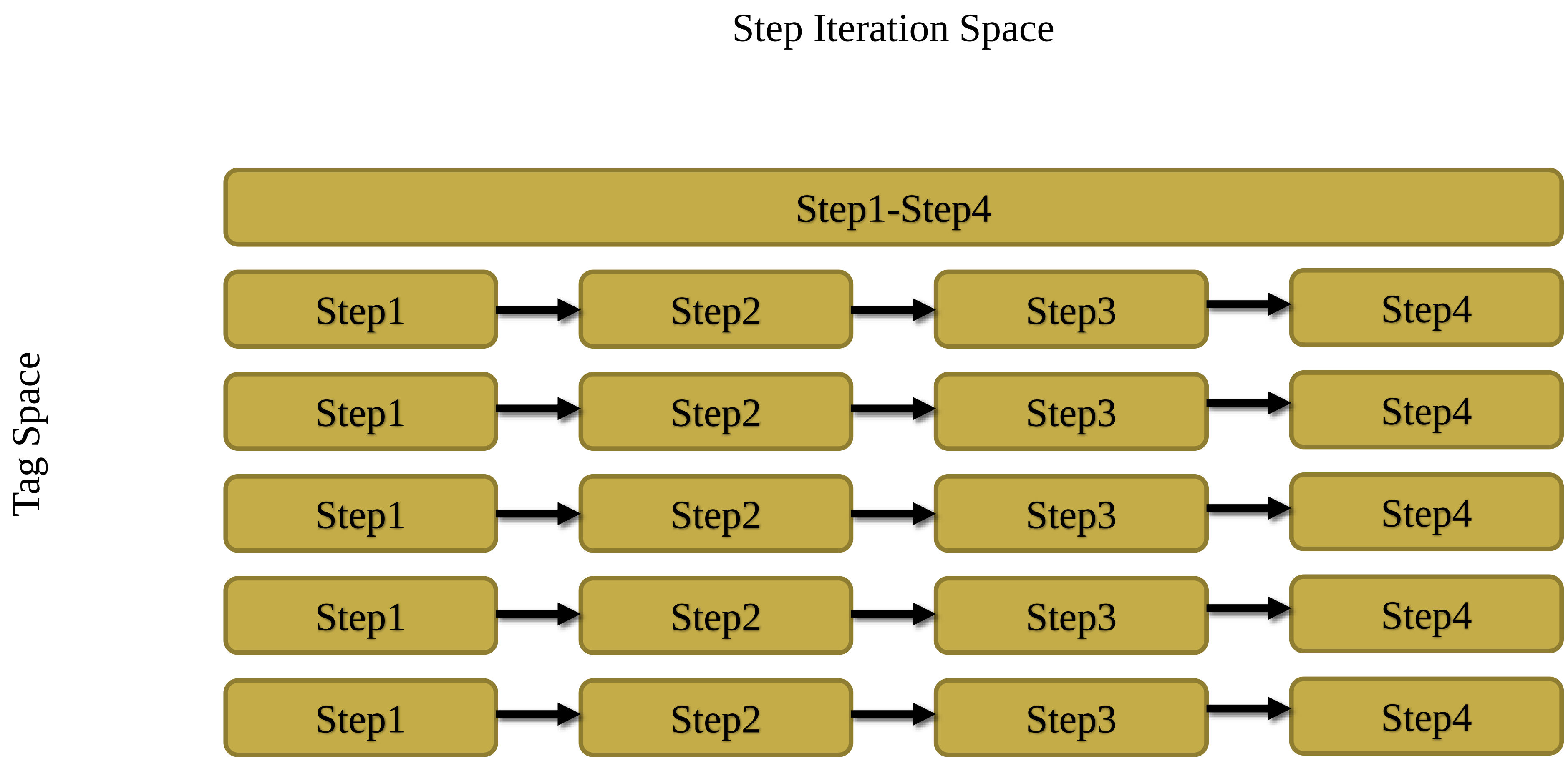
Motivation

- Fully decomposed algorithm results in fine-grained parallelism
  - Performance suffers if steps lack sufficient computation
- Need to coarsen the computation
  - Modify CnC step/tag collections
  - Fusion: Combine multiple steps (graph level)
  - Tiling: Combine multiple step instances from multiple tags
- Challenge: Take the provided singleton, sequential, algorithmic stencil and produce a scalable parallel application
  - Perform high level transformations
    - Legality and detection
  - Program modifications

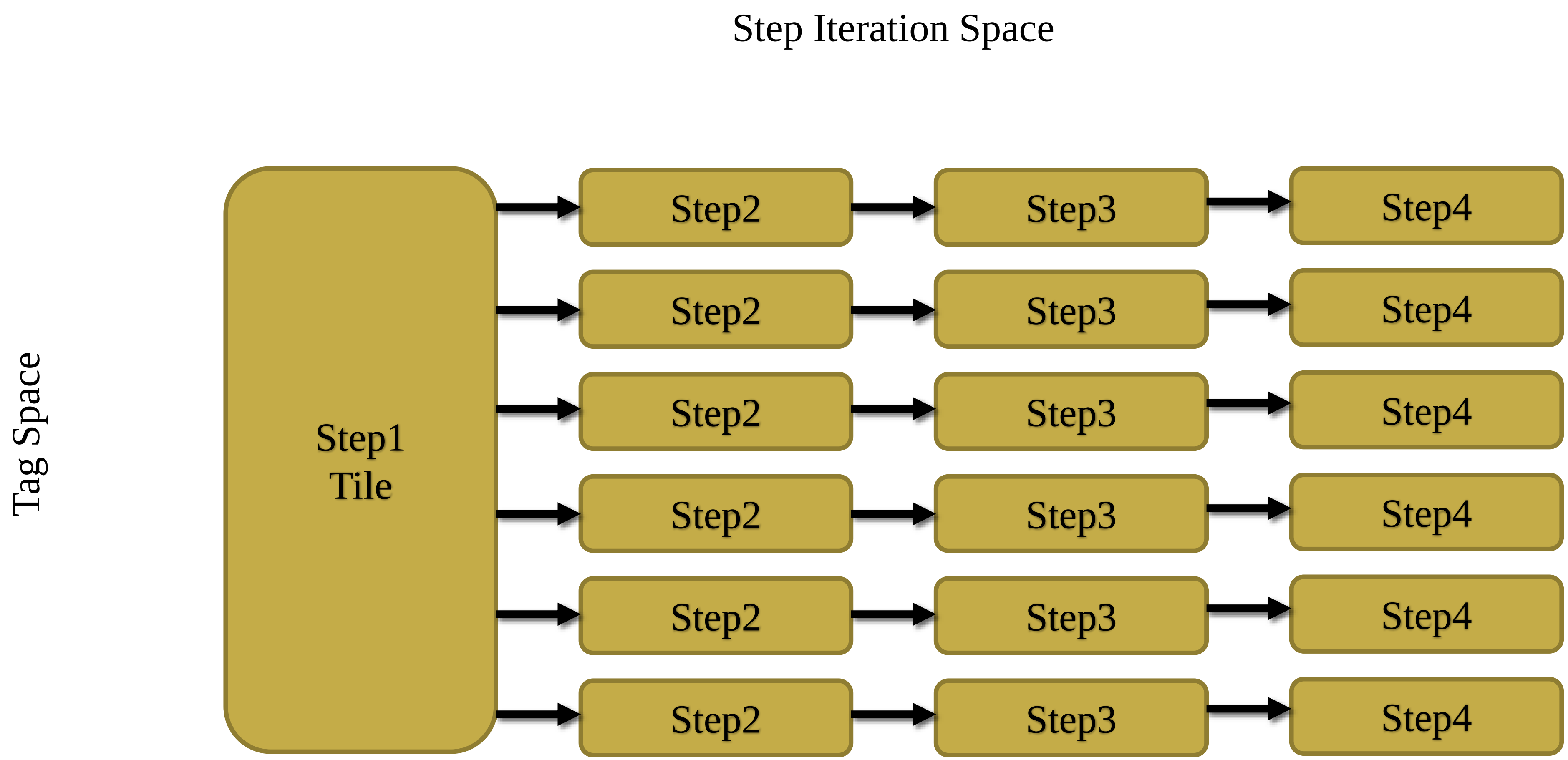
CnC computation space



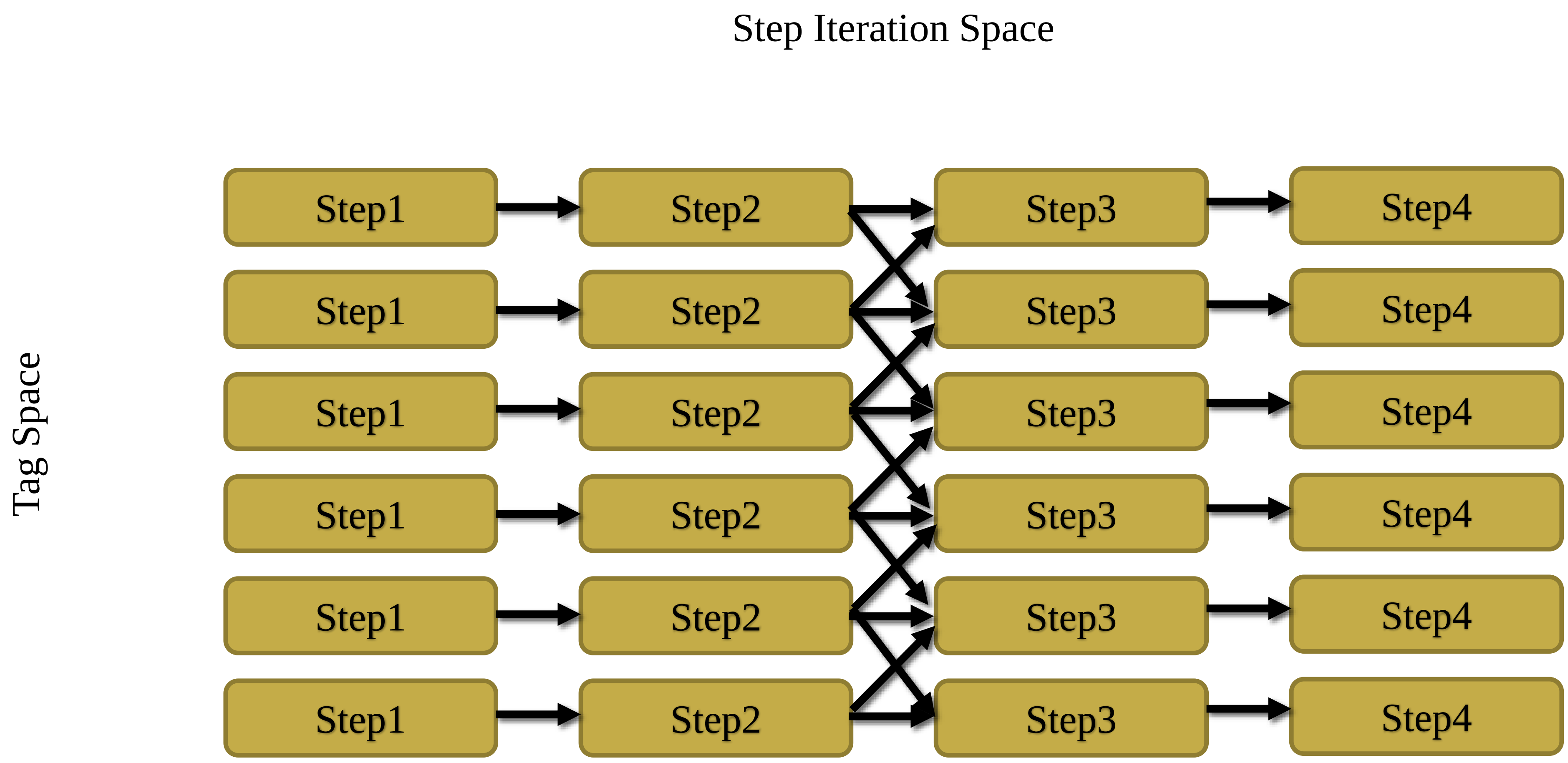
Step Fusion


Step Iteration Space
Tag Space
Step1-Step4
Step1
Step2
Step3
Step4
Step1
Step2
Step3
Step4
Step1
Step2
Step3
Step4
Step1
Step2
Step3
Step4
Step1
Step2
Step3
Step4

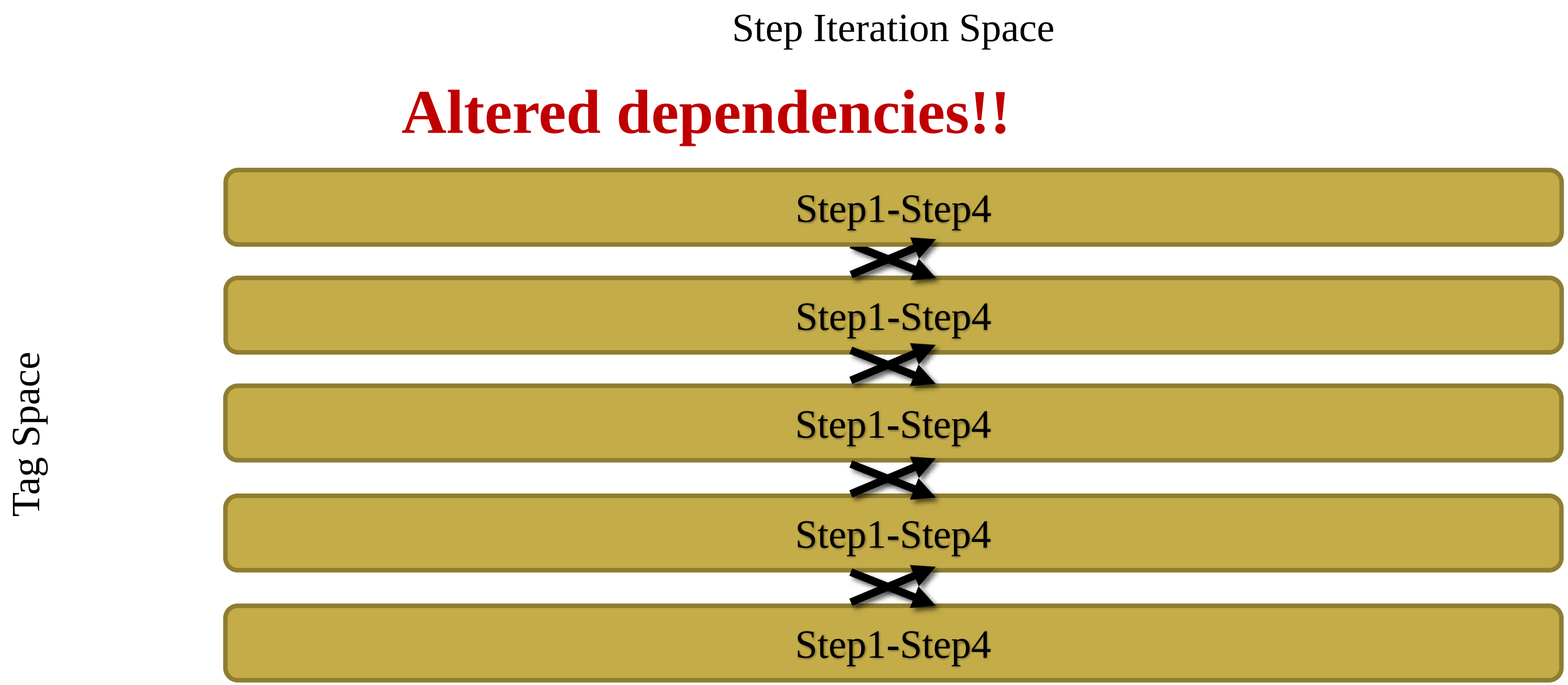
Tag Tiling


Step Iteration Space
Tag Space
Step1
Tile
Step2
Step3
Step4
Step2
Step3
Step4
Step2
Step3
Step4
Step2
Step3
Step4
Step2
Step3
Step4
Step2
Step3
Step4

Fusion Scenario



Fusion Scenario

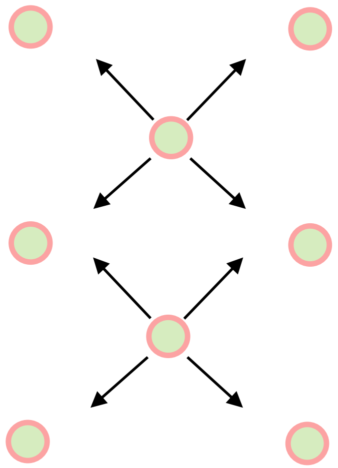


Fusion and Tiling Legality

- Step Fusion
  - Step collections are prescribed by the same tag, all dependencies are within that tag space
  - Serialize steps, remove intermediate dependencies
- Tag Tiling
  - Step collection operates on multiple tags, performing the same work, independently
  - Coalesce computation from multiple tags
  - Opportunity to exploit reuse
- Illegal Fusion
  - Cannot fuse if a "get" depends on value "put" from a different tag

Step1: cnc.dataout.put(produced_data, my_index)

Step2: For(All neighbors) {cnc.dataout.get(consumed_data, neighbor_index)}

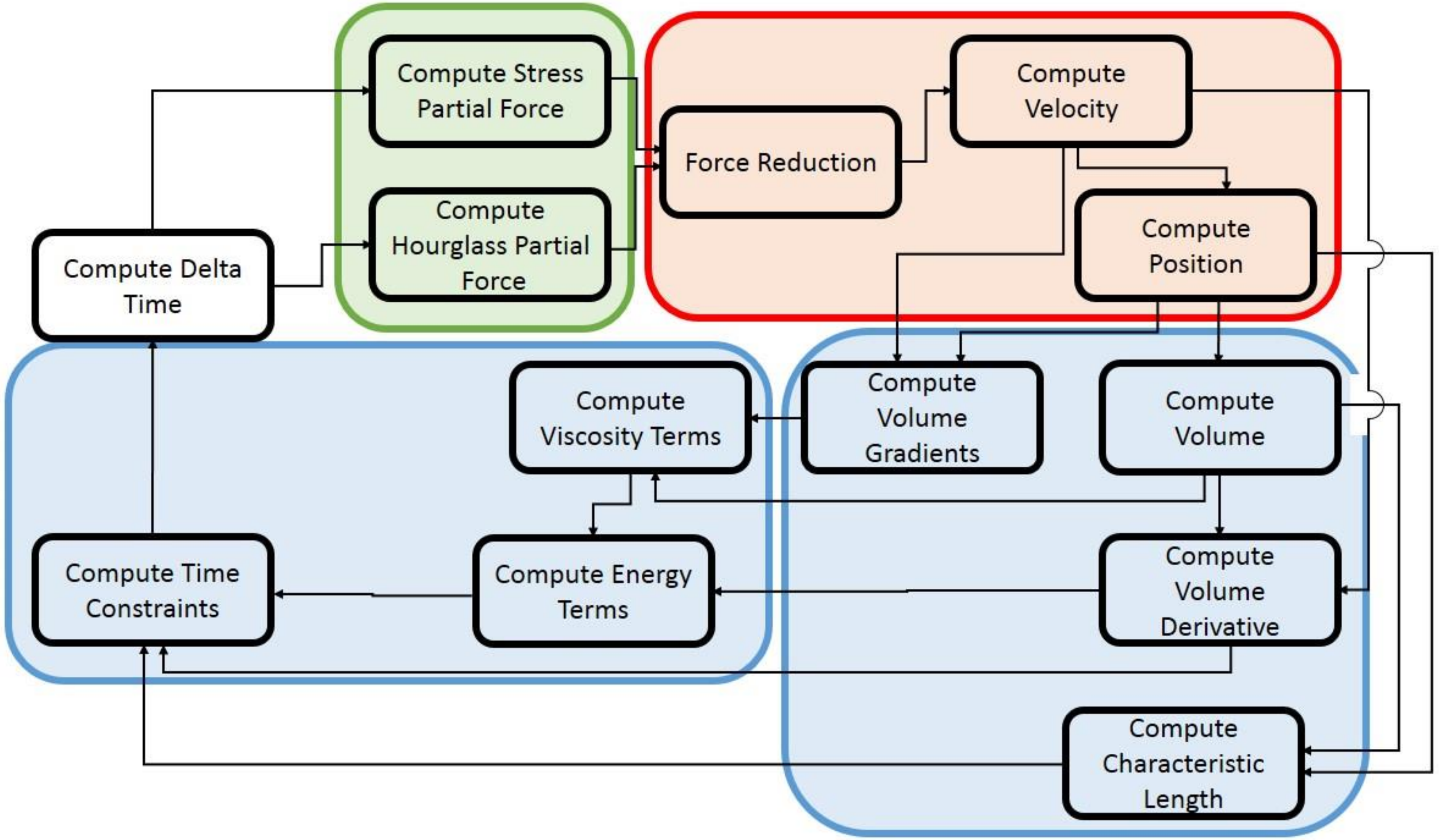


PURDUE UNIVERSITY

Code changes

- Step Collections
  - Aggregate dependencies from fused routines
  - Coalesce work from multiple steps and tags
  - Maintain proper working set size and intermediate storage
  - Must maintain step-like behavior
- Tag Prescription
  - New smaller tag set
- Other Optimizations
  - Block size
  - Dependency re-use
  - Item Collection - coalescing

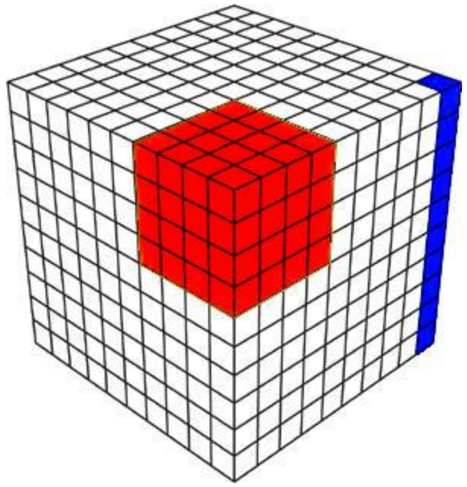
LULESH: Fused Algorithm



PURDUE UNIVERSITY

Experimental Results

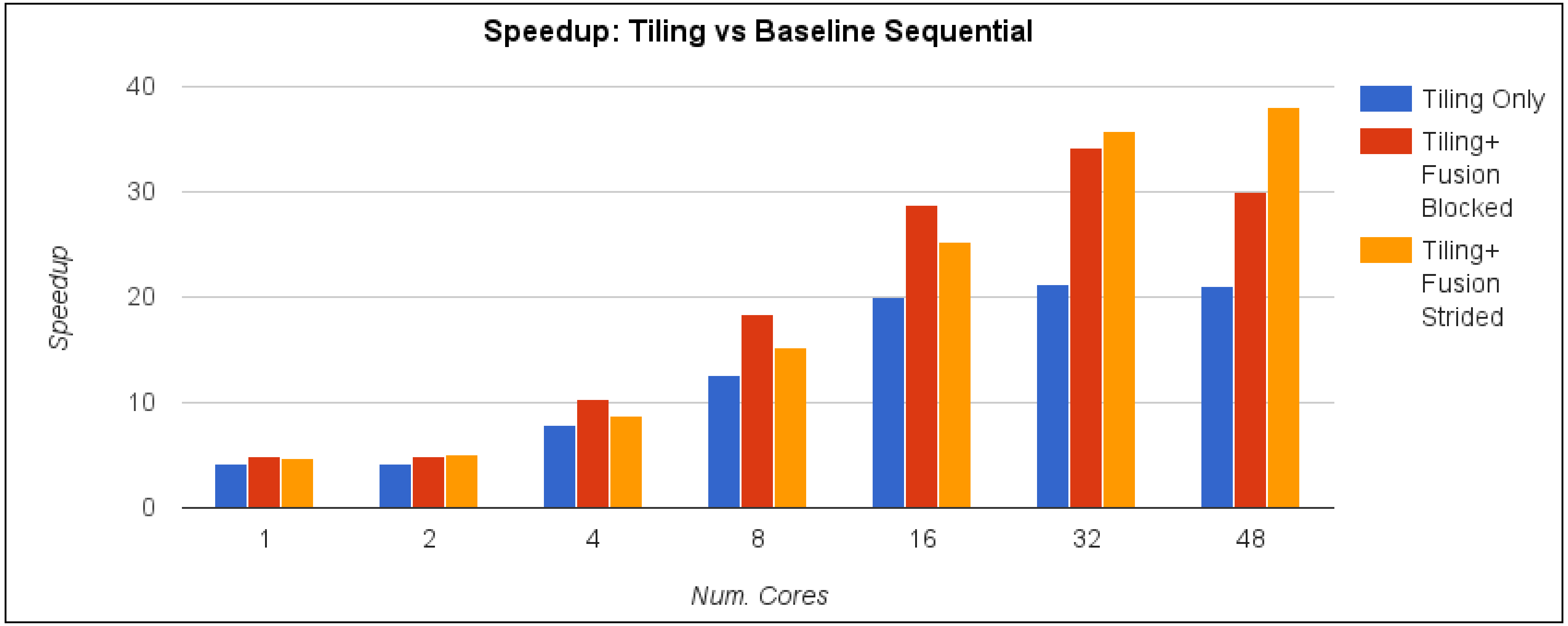
- AMD Opteron 6176 SE system with four 12-core processors (48 cores total) running at 2.3 GHz.
- gcc 4.7, -o3, Intel CnC implementation
- Experiments
  - Baseline
  - Fused-only
  - Tiled-only
  - Fused+Tiled Blocked (red)
  - Fused+Tiled Strided (blue)



Experimental Results cont.

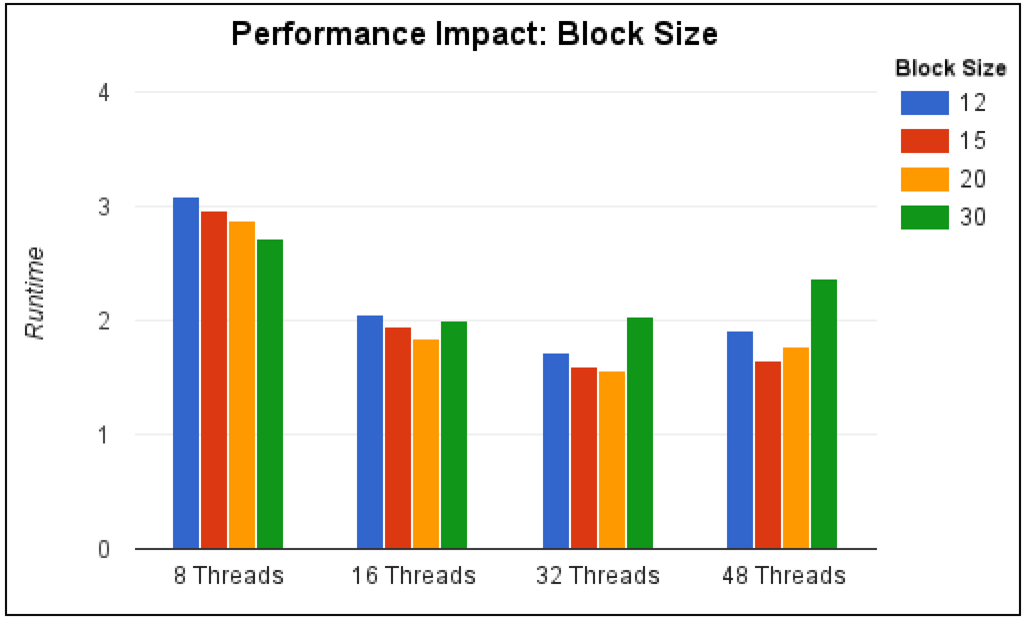
Table 1: Timing Results: 60^3 Sized Mesh

	Number of Cores						
	1	**2**	**4**	**8**	**12**	**32**	**48**
Baseline	53.180	52.900	71.852	108.531	110.515	110.572	119.466
Fusion Only	26.288	26.405	33.3389	52.224	51.391	54.340	57.430
Tiling Only	12.67596	12.5693	6.677482	4.234	2.652	2.504	2.526
Blocked Fuse-Tile	10.749	10.699	5.110	2.883	1.845	1.557	1.768
Strided Fuse-Tile	11.171	10.610	6.025	3.498	2.104	1.483	1.397



Tiling: Block Size

- Parameter: block size
  - Smaller size creates excess fine-grain parallelism
  - Larger size limits available parallelism
  - Tunable parameter



Summary

- Contributions
  - Performance & scalability obtained with step fusion, tag tiling
  - Minimal underlying code and data layout changes
  - Automatic optimization in works
- Improving CnC
  - Tiling/Grain size is a huge problem for stencils
  - Tuners available, but do not operate at a high level
- Future Goals
  - High level collection tuning
  - Hierarchical CnC

Thank you!

Questions?