EVALUATING PERFORMANCE OF TASK AND DATA COARSENING IN CONCURRENT COLLECTIONS

Chenyang Liu, Milind Kulkarni

PURDUE
UNIVERSITY

Overview

- Scientific Applications
  - Domain expert required for correctness, algorithm semantics, etc.
  - Performance expert for tuning performance, scalability, portability, etc.

- Concurrent Collections
  - “Separation of Concerns” Philosophy - decouple performance and algorithmic/domain concerns

- Contributions
  - Evaluate Performance benefits
  - Discuss High-Level transformations
  - Explore Automation and Tuning

PURDUE UNIVERSITY

Scientific Applications

- Programmability
  - Algorithmic correctness - requires understanding of the scientific domain or method
  - Algorithmic design – modular data/functions (ease of future programming)
  - Exploit semantic optimization (high-level tuning)

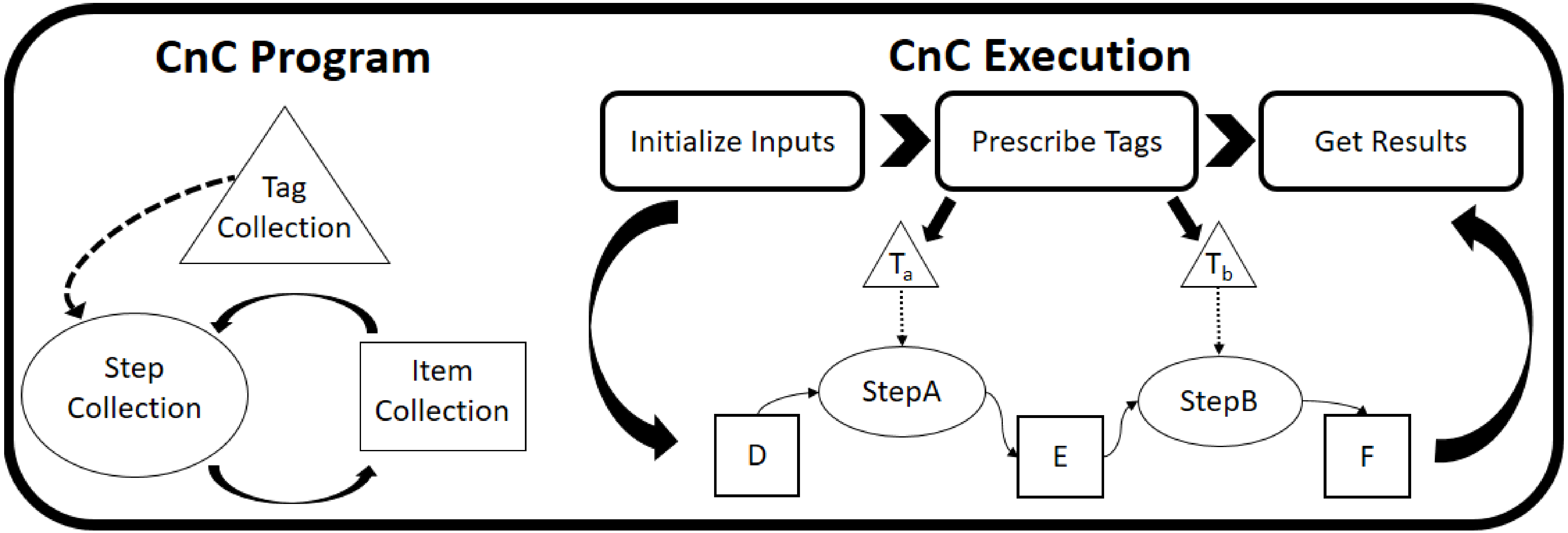
- Performance
  - Parallelization challenges
    - Expressing Parallelism
    - CPUs? GPUs? NUMA?
  - Memory Hierarchy/Locality
  - Communication Costs

- ***Idea: Explore high-level optimizations that will improve both programmability and performance for complex parallel applications***

PURDUE UNIVERSITY

Concurrent Collections

- A Parallel Programming Philosophy
  - “Separation of the Concerns” for domain expert and performance tuning expert
  - Flexible task-parallel model/runtime supports multiple platforms
  - Parallelism automatically exploited using task-based data-driven model
- High-level specifications, Low-level tuners
  - Declarative specification for computation/data/control dependencies
  - Hardware tuners allow machine-specific optimization
- Research
  - Task-level fusion/tiling
  - Data-tiling
  - Programming automation



- Data-Driven Model
  - Tags prescribe steps, creating dynamic step instances
  - Steps execute when inputs are ready (step-like property)
- Data is immutable (Dynamic single assignment)
  - Key/Value lookup using "get/put" operations
- Parallel Runtime
  - Exploits parallelism given dependency constraints

Problem

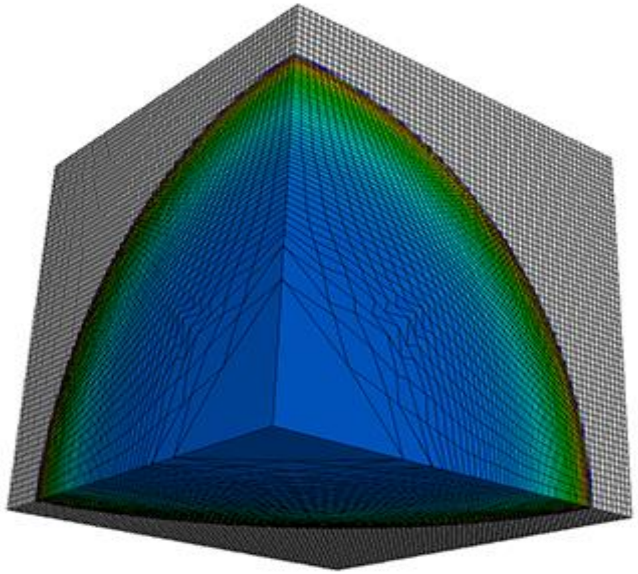
- How to optimize parallel performance for the LULESH code?
  - Start with decomposed algorithm, explore high-level transformation
  - Later modify data layout and create tiled computation steps

- Challenges
  - Task Granularity: Fine-grain parallelism
  - Data Granularity: Synchronization for large datasets
  - Explore Task and Data coarsening (Tiling)

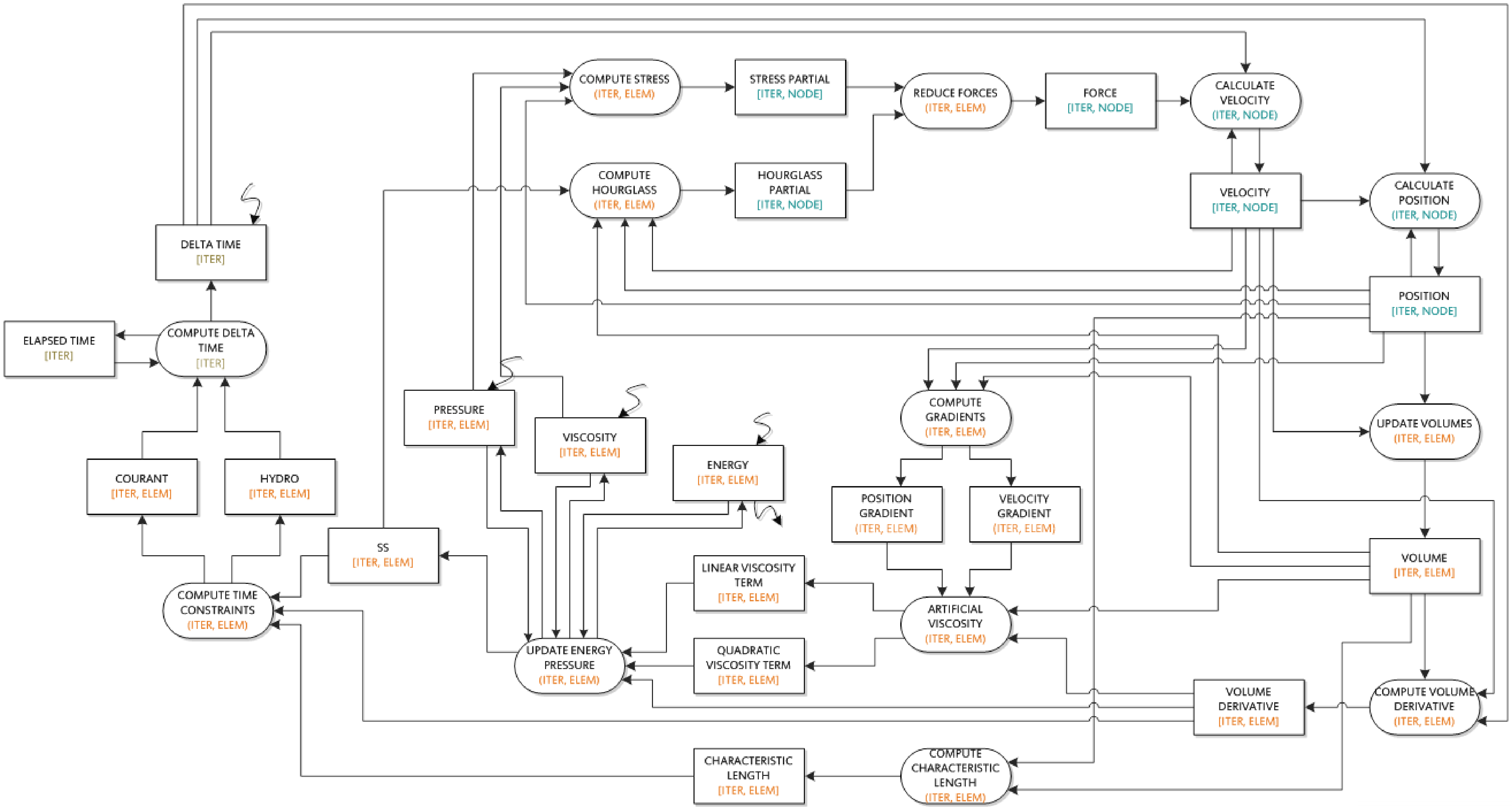
- Other Concerns
  - CnC Translator for semi-automatic code generation
  - Application or machine-specific tuning

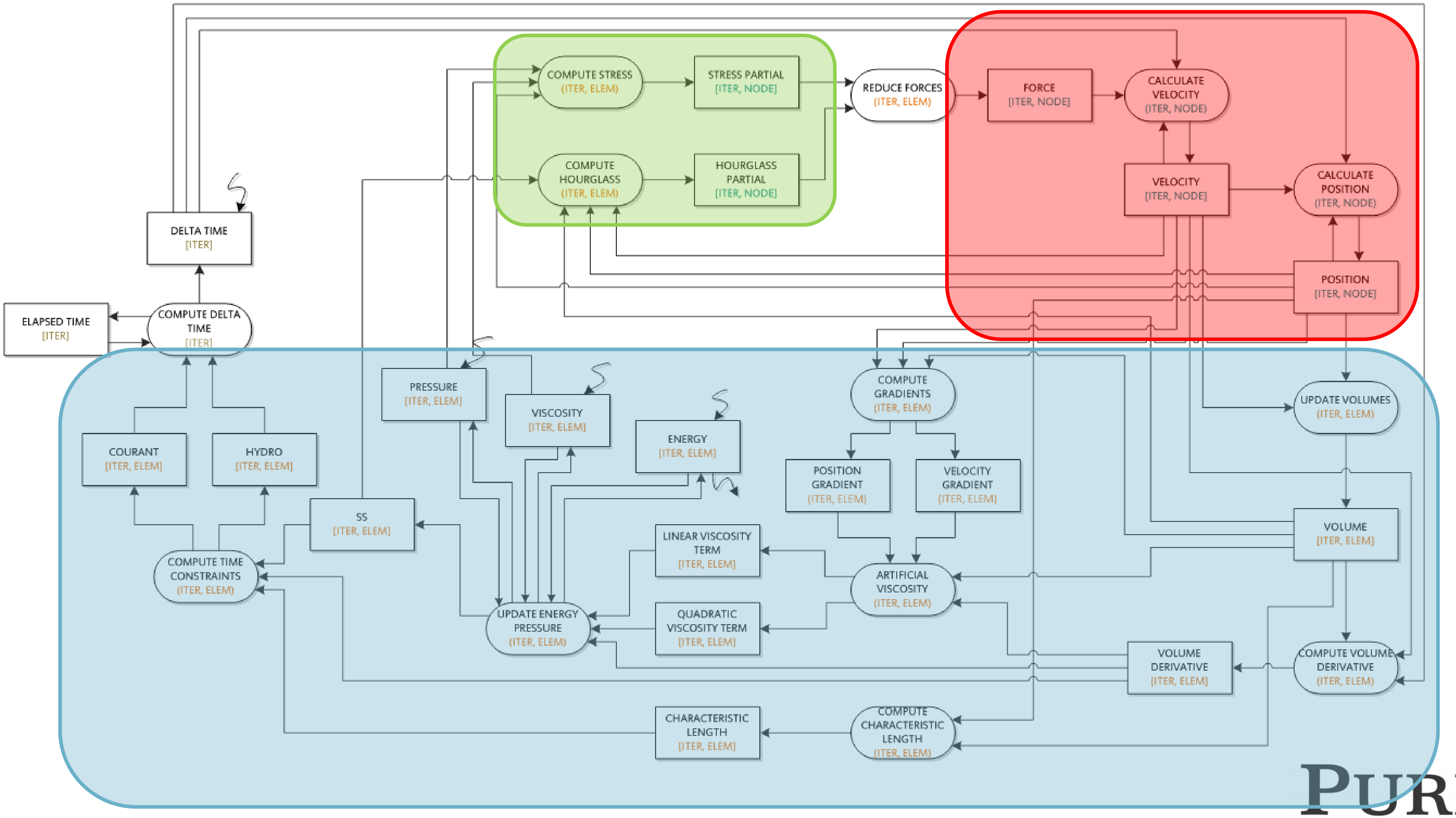
CnC LULESH

- **LULESH: Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics**
  - Challenge problem from the DARPA UHPC program
- 3D blast wave propagation simulation
  - Operates on a hexahedral mesh with 2 centerings:
  - Node/Element interactions/computations
  - Lots of control-flow
  - Multiple stencil updates
  - Ample Parallelism
- Build upon a fully decomposed algorithm



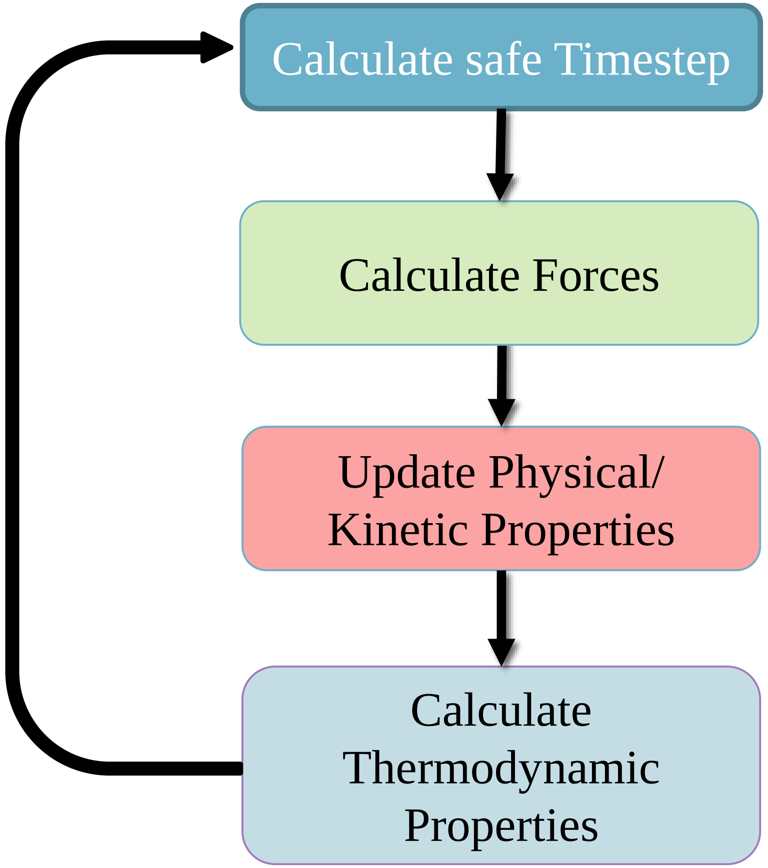
PURDUE UNIVERSITY


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)



PURDUE
UNIVERSITY

LULESH Algorithm



Requires minimum value across all elements

Element force computation for each node neighbor

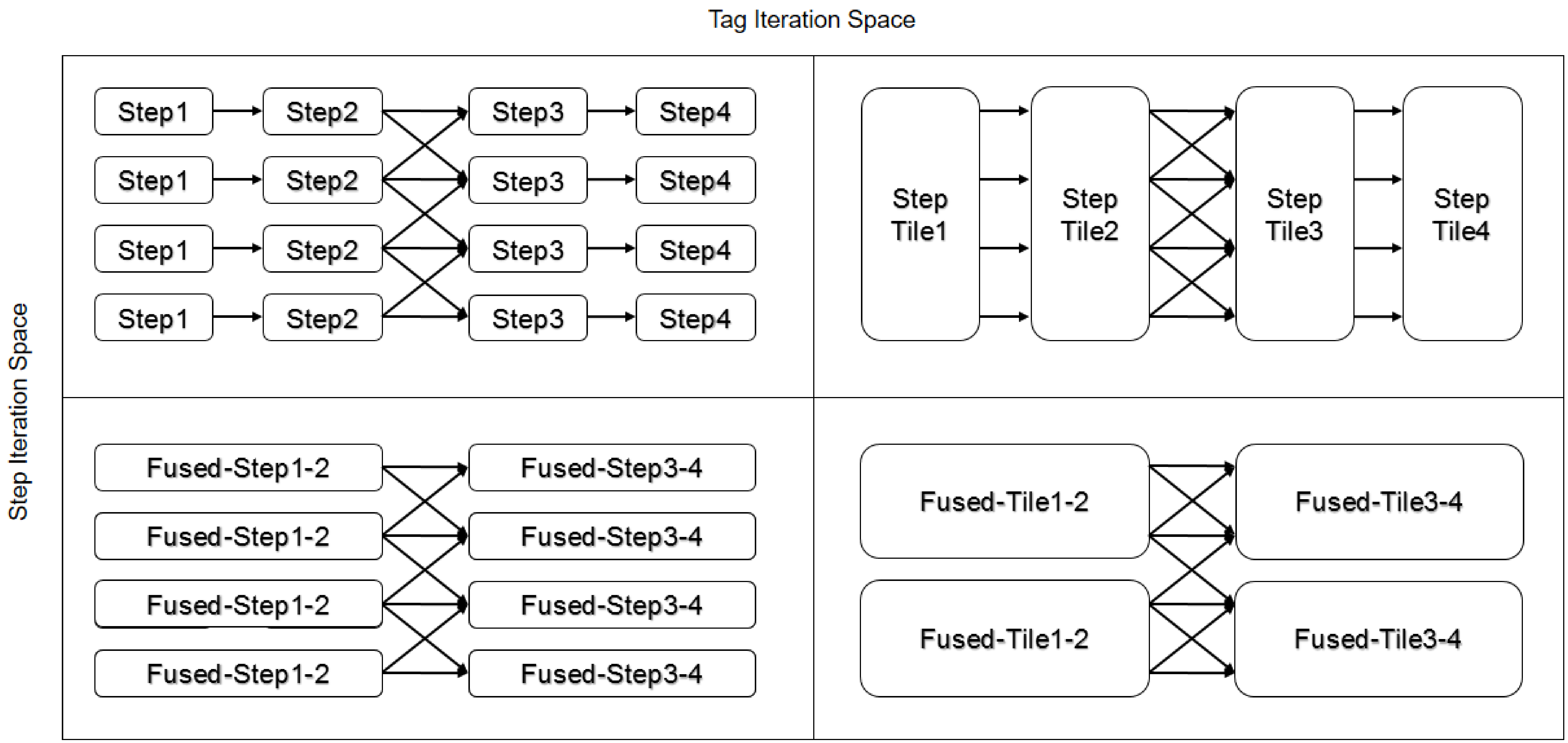
Requires force reduction from each element neighbor

Element-to-element data exchange for each face neighbor

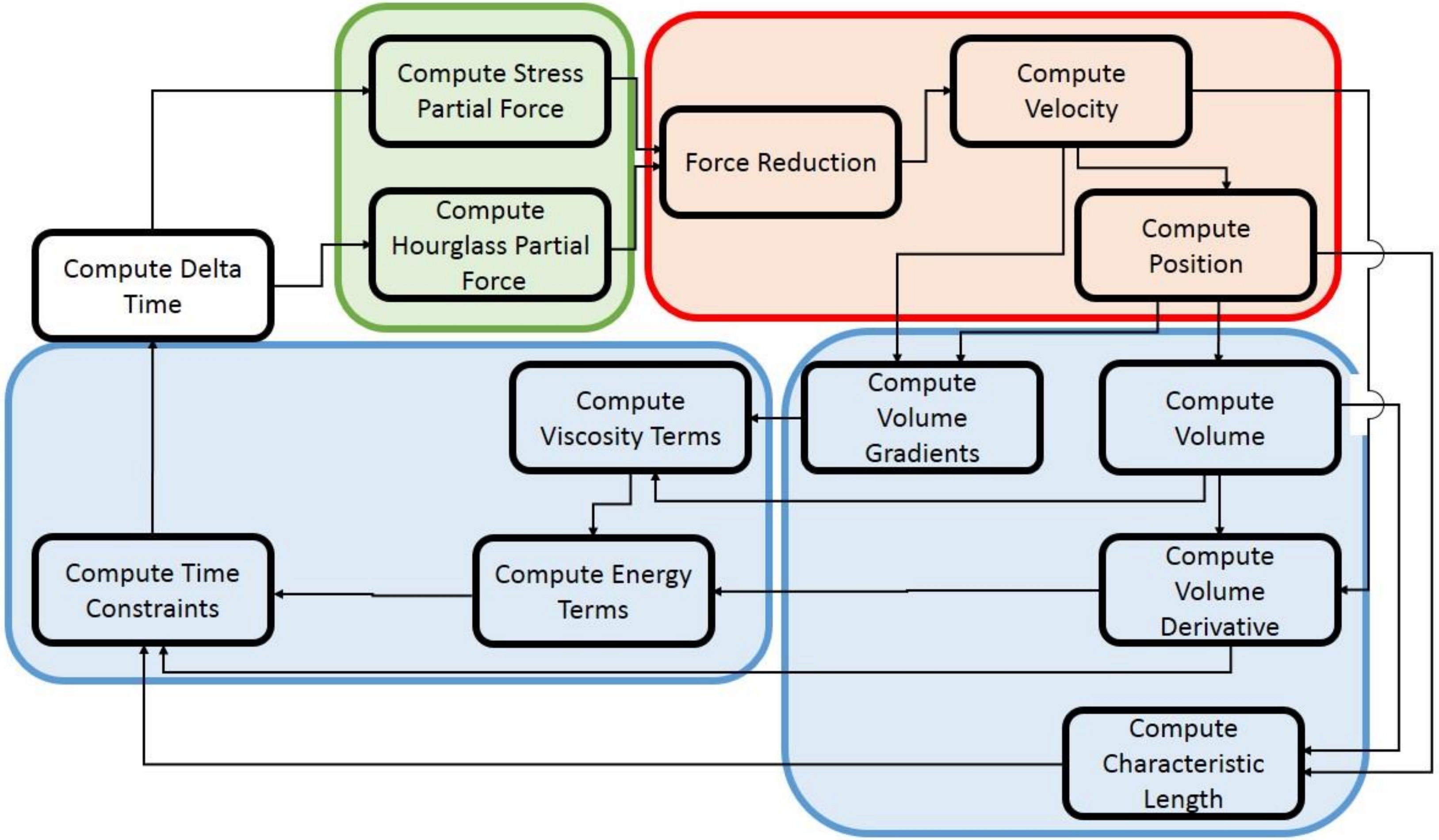
Task Coarsening

- Baseline program specifies the computation per iteration, per node/element
  - Too much overhead from fine-grain parallelism
- Solution: Coarsen through modifications through Collections
- Step Fusion
  - Serialize different steps operating under the same tag
  - Legal as long as no dependency cycles (co-routine)
- Tag Tiling
  - Serialize the same steps operating under different tags
  - Resulting tiled steps must be “step-like”

Step Fusion vs Tag Tiling



LULESH: Fused Algorithm



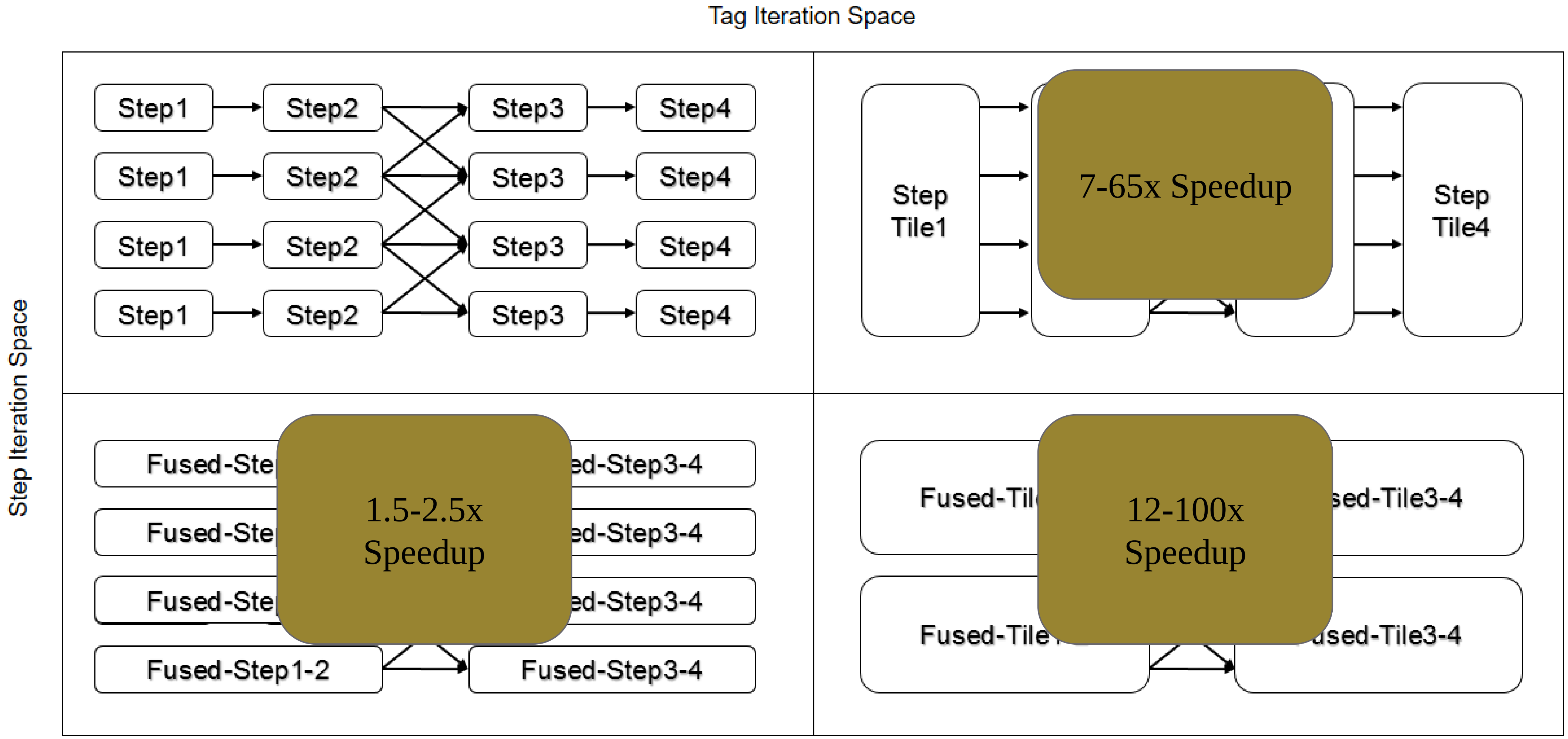
Step Fusion & Tag Tiling

- **Challenges**:
- Structural changes to collections and tag/step organization
  - Semi-automatic CnC Translator

- Fused steps require aggregated dependencies
  - Still need to maintain step-like behavior (‘get’s first, puts after)

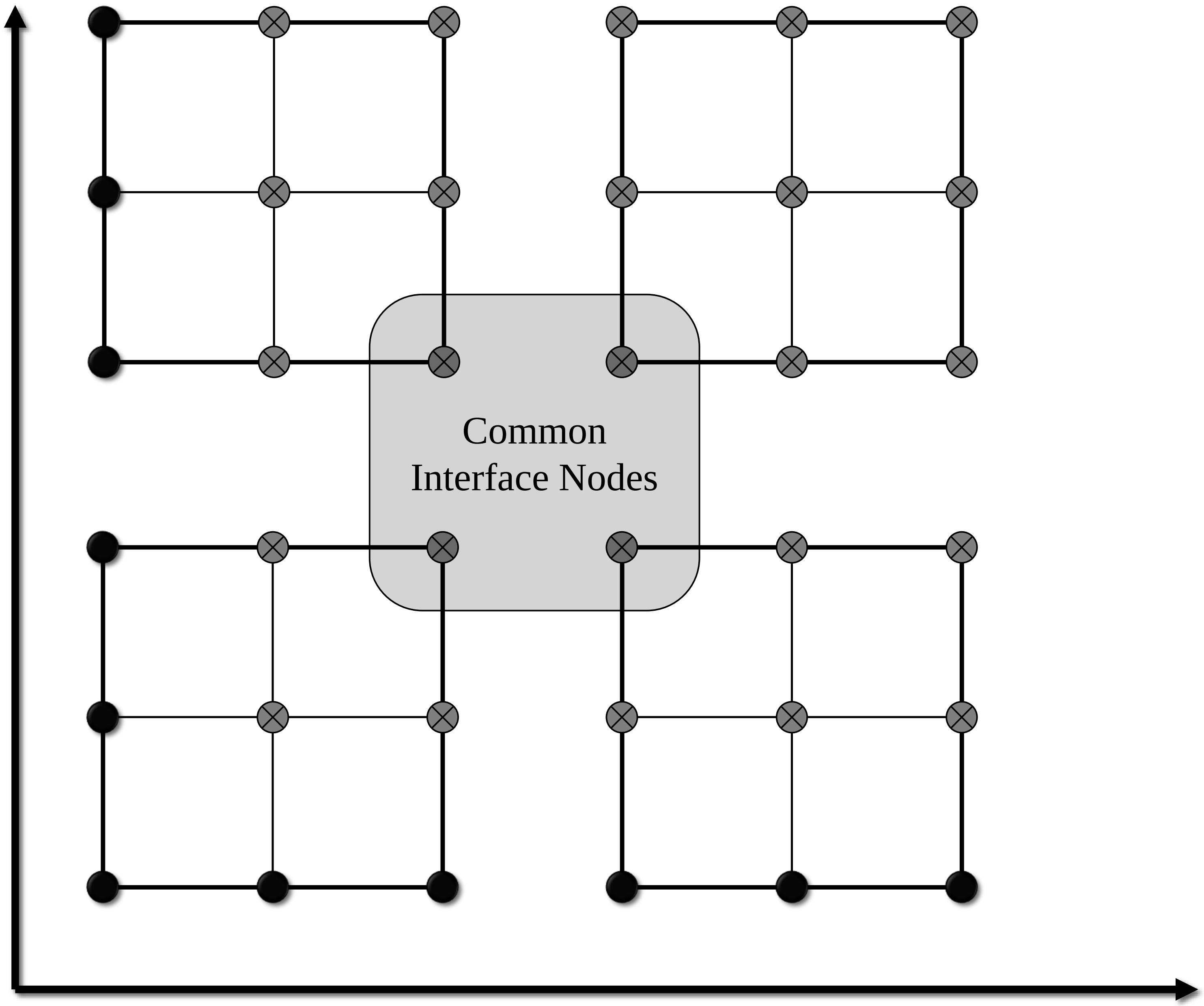
- Tiled steps need intermediate storage for temp. data
  - Possible re-use/sharing of neighboring data (code modifications)

Step Fusion vs Tag Tiling

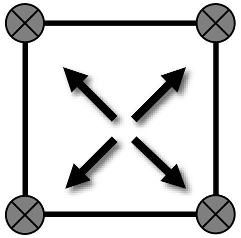


Data Tiling

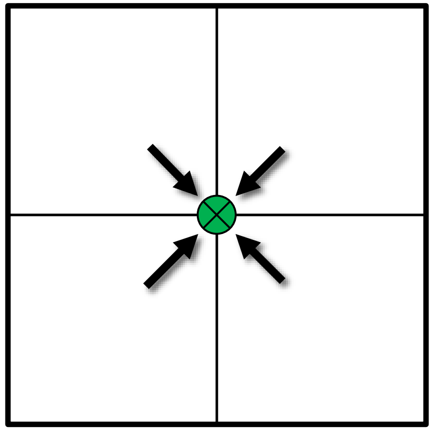
- Coarsened tasks, but data layout unchanged
  - Locality within tasks, but runtime synchronization required for data in item collection
- Challenges: How to tile inconsistent (node/element) centerings
  - Number of nodes = elements + 1
  - Recall stencil operation that requires update via neighbors
  - Data tiles must conform to dynamic-single-assignment
- Underlying code modifications required for data tiling
  - Kernel computations operates on tiled datasets
  - Further locality/re-use optimizations (variable elimination)
  - Modified communication patterns – intra-tile communication (fast)
- *Note: Data coarsening opportunity a result of task coarsening*



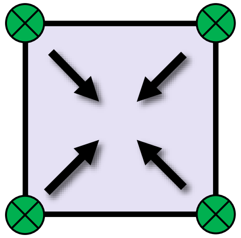
1-Element Force Computation

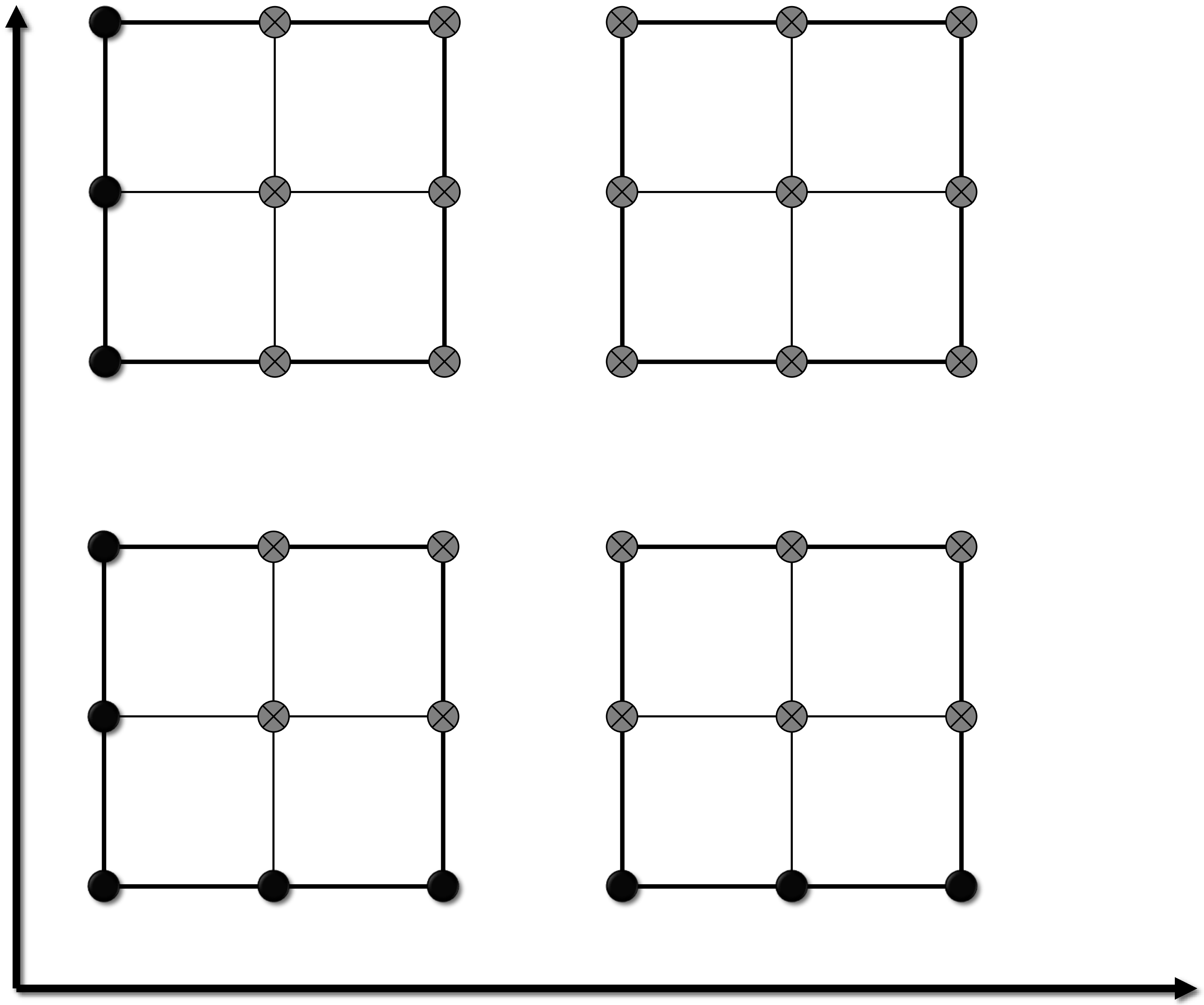


2-Nodal Force Reduction/Update

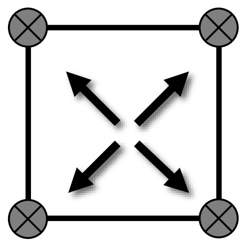


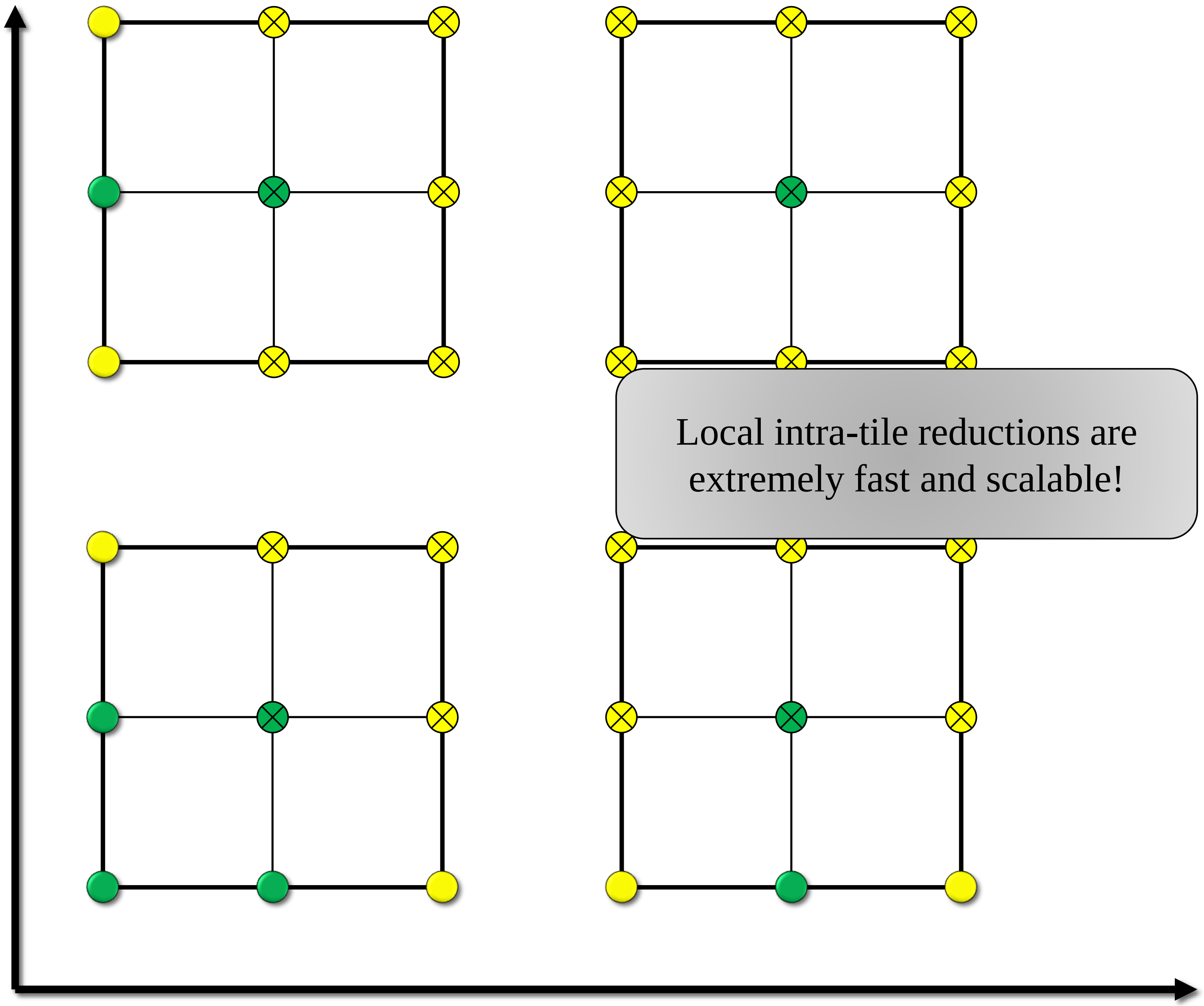
3-Element Update/ Computation



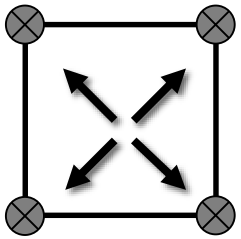


1-Element Force Computation

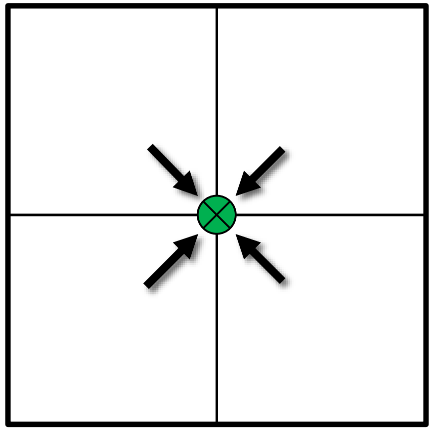


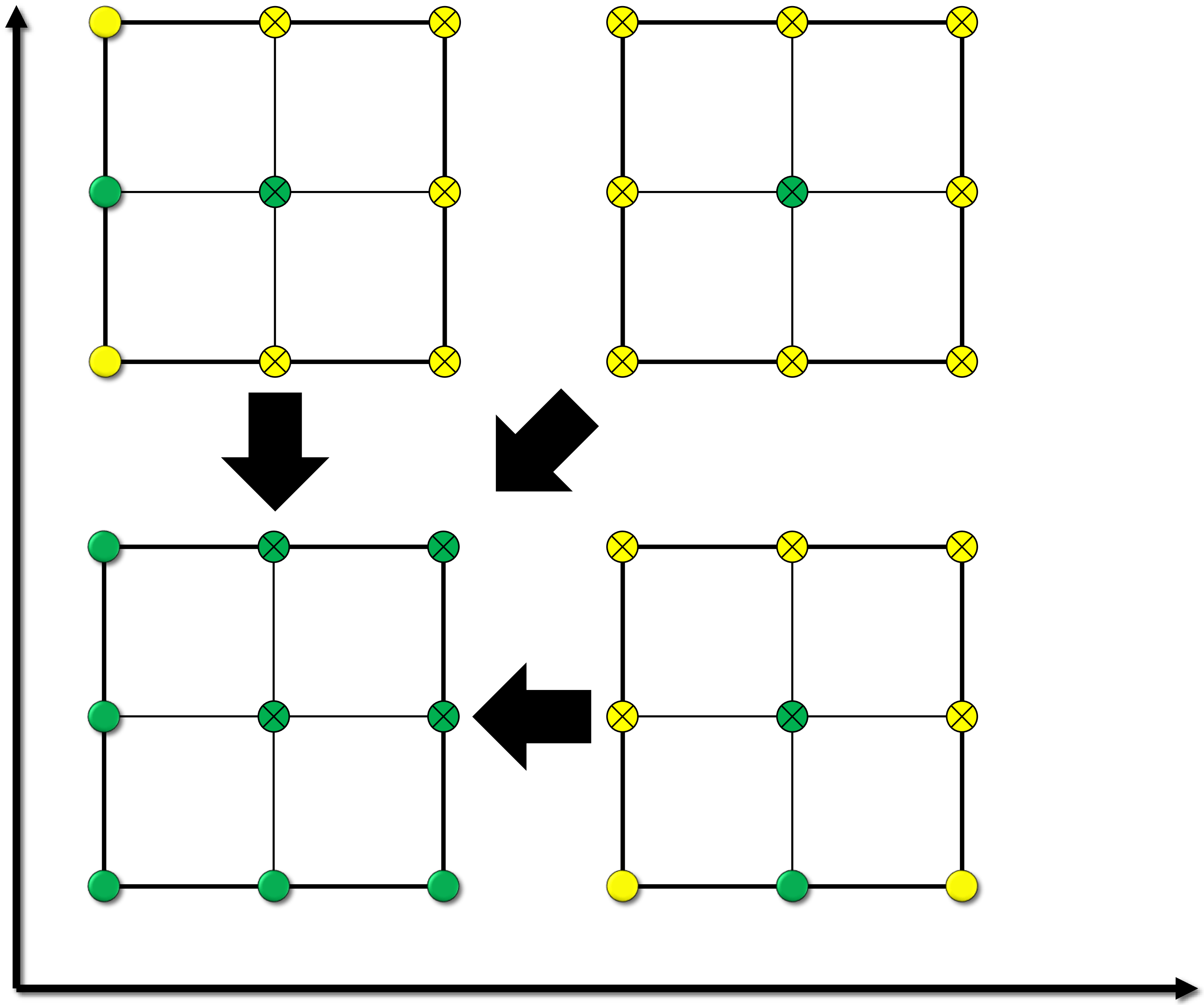


1-Element Force Computation

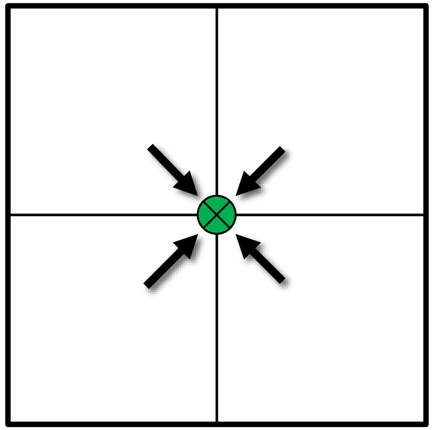


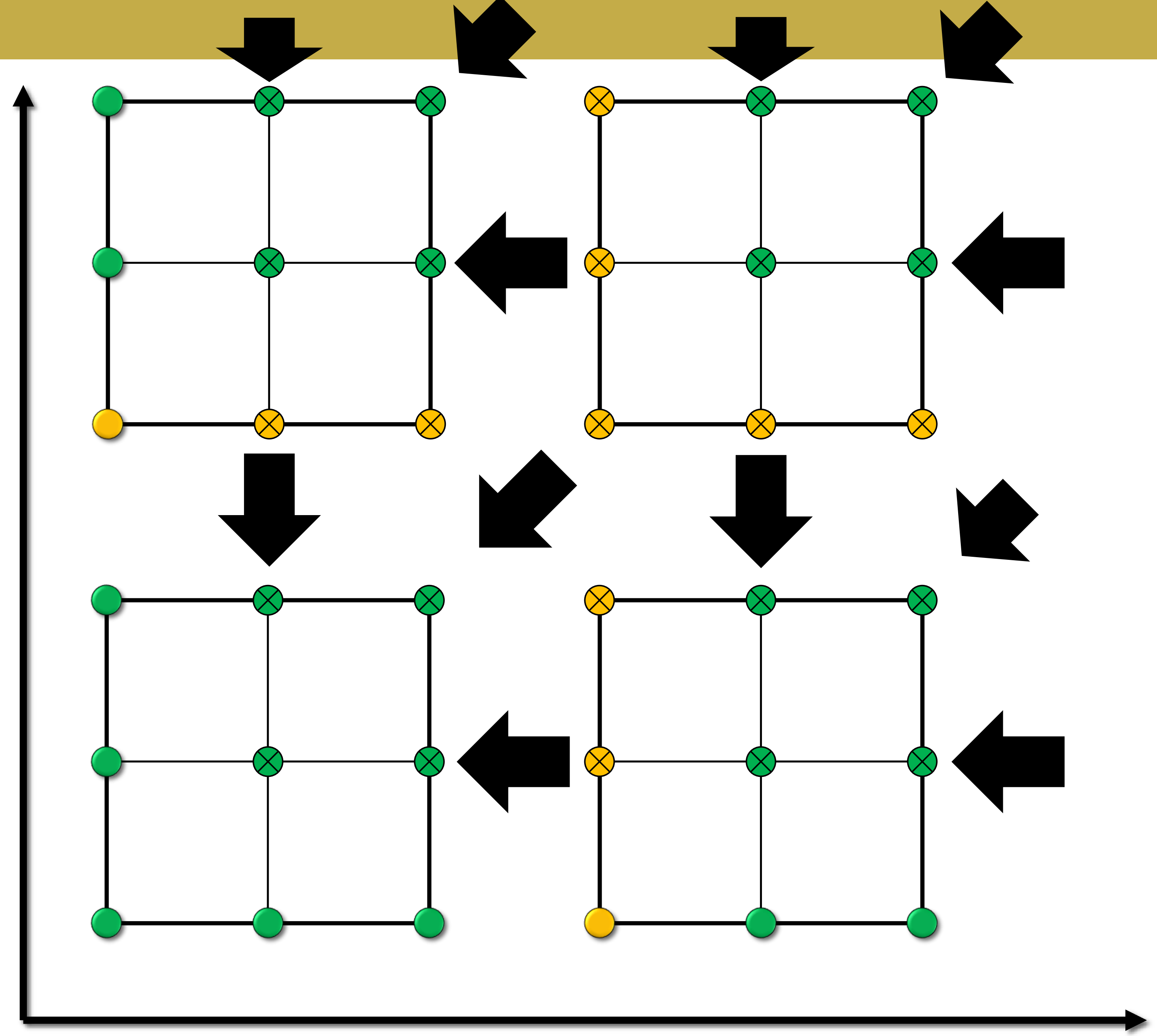
2-Nodal Force Reduction/Update



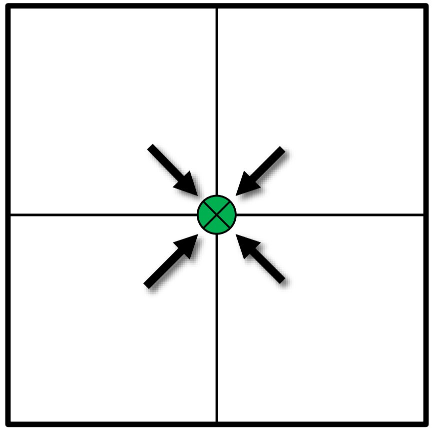


2-Nodal Force
Reduction/Update



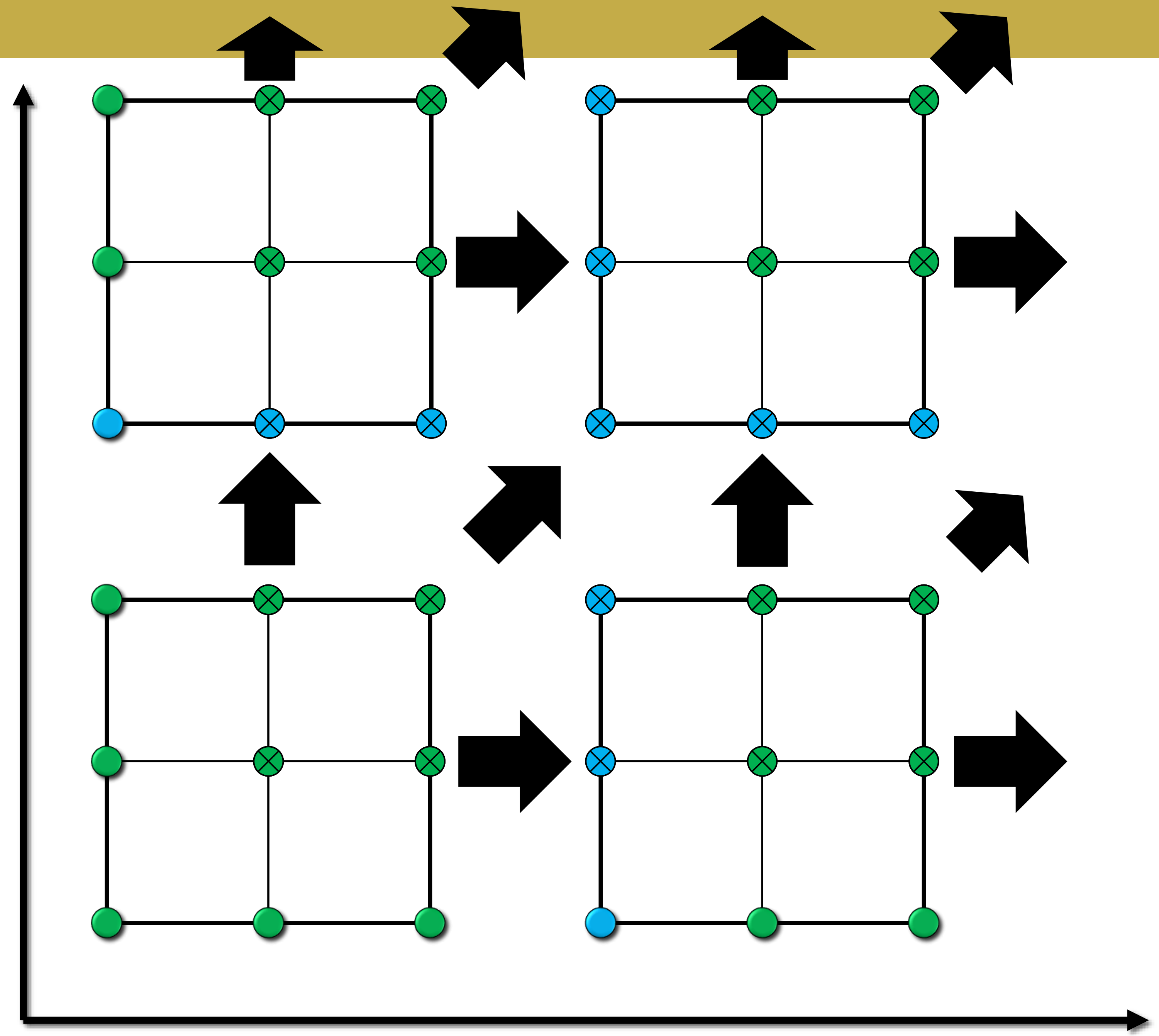


2-Nodal Force Reduction/Update

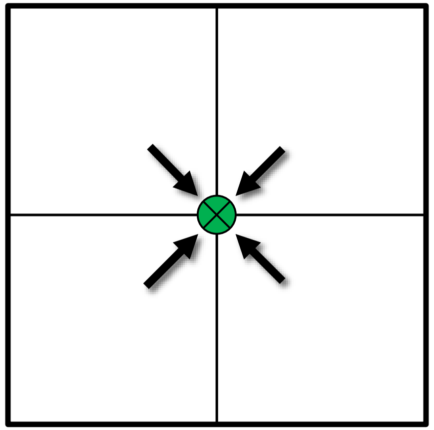


Redundant Interface Node

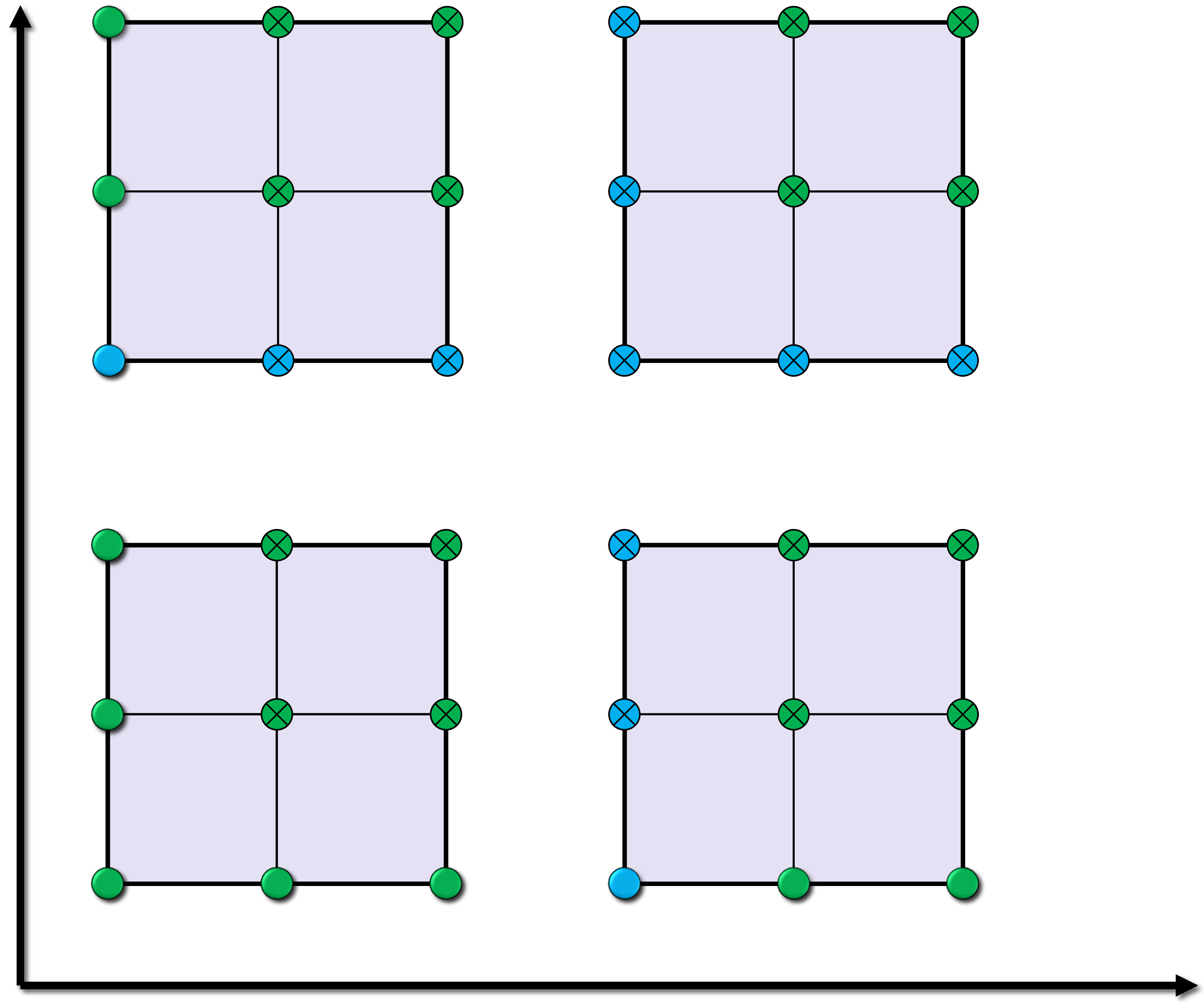
PURDUE
UNIVERSITY



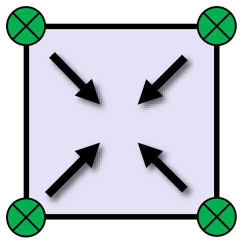
2-Nodal Force Reduction/Update

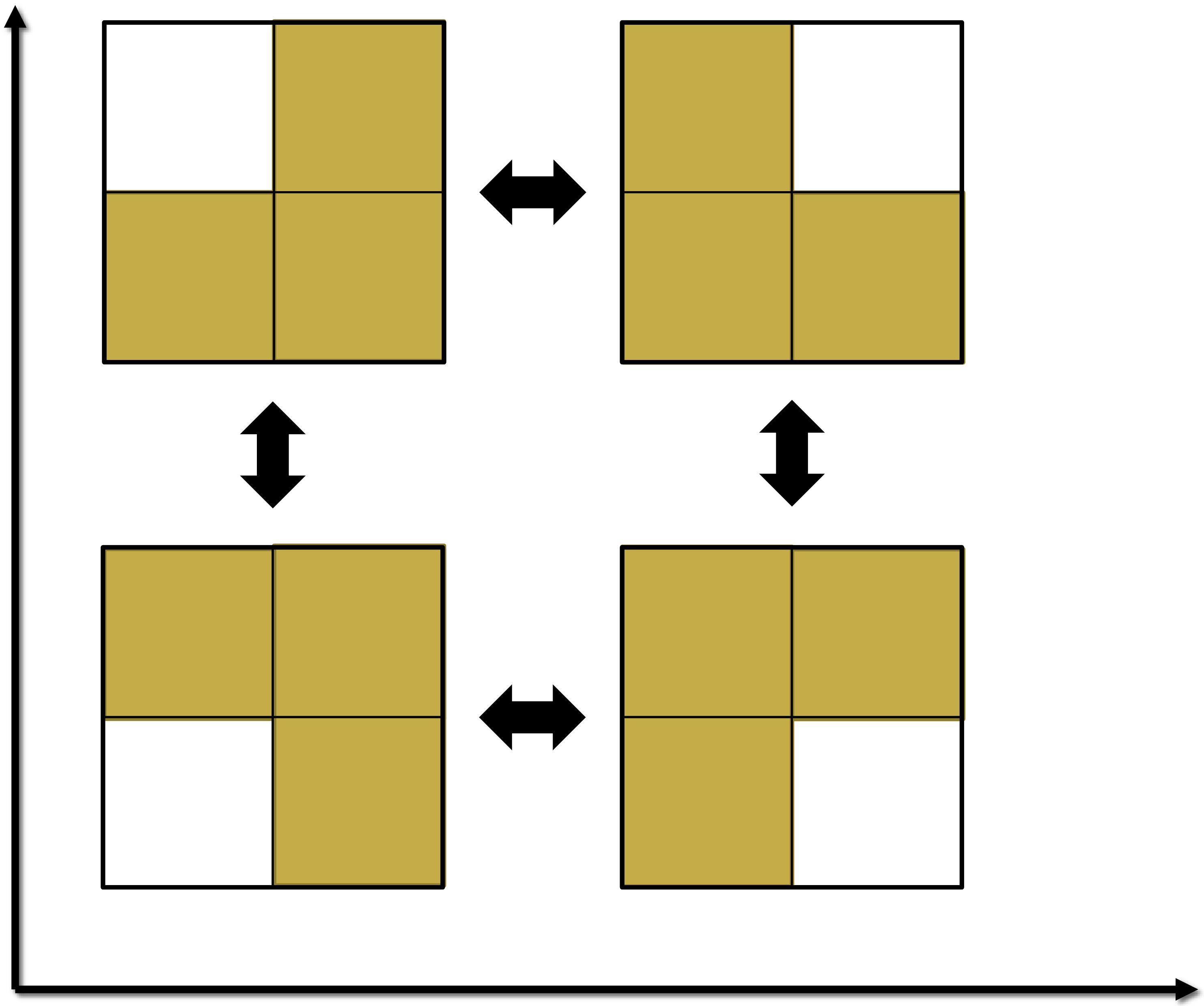


Updated Interface Node

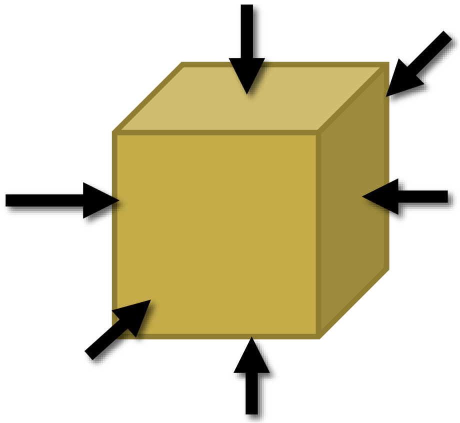


3-Element Update/
Computation





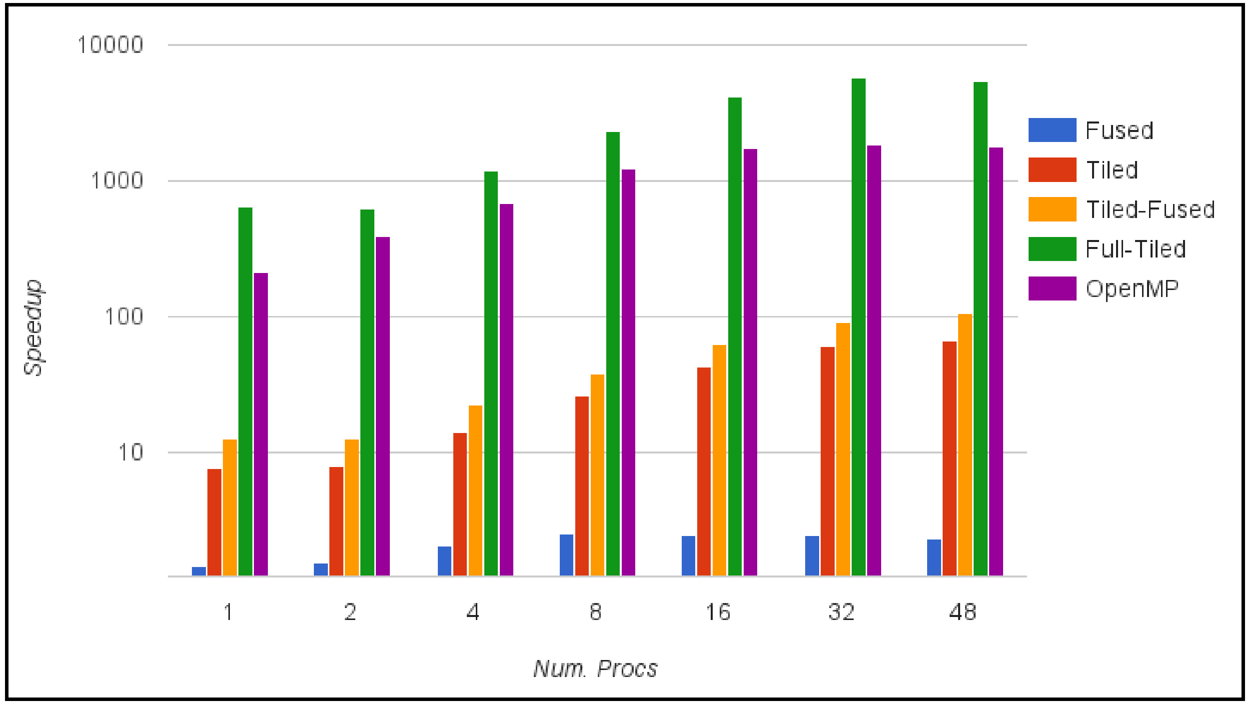
After element computation,
there is an element gradient update:
requires face neighbors

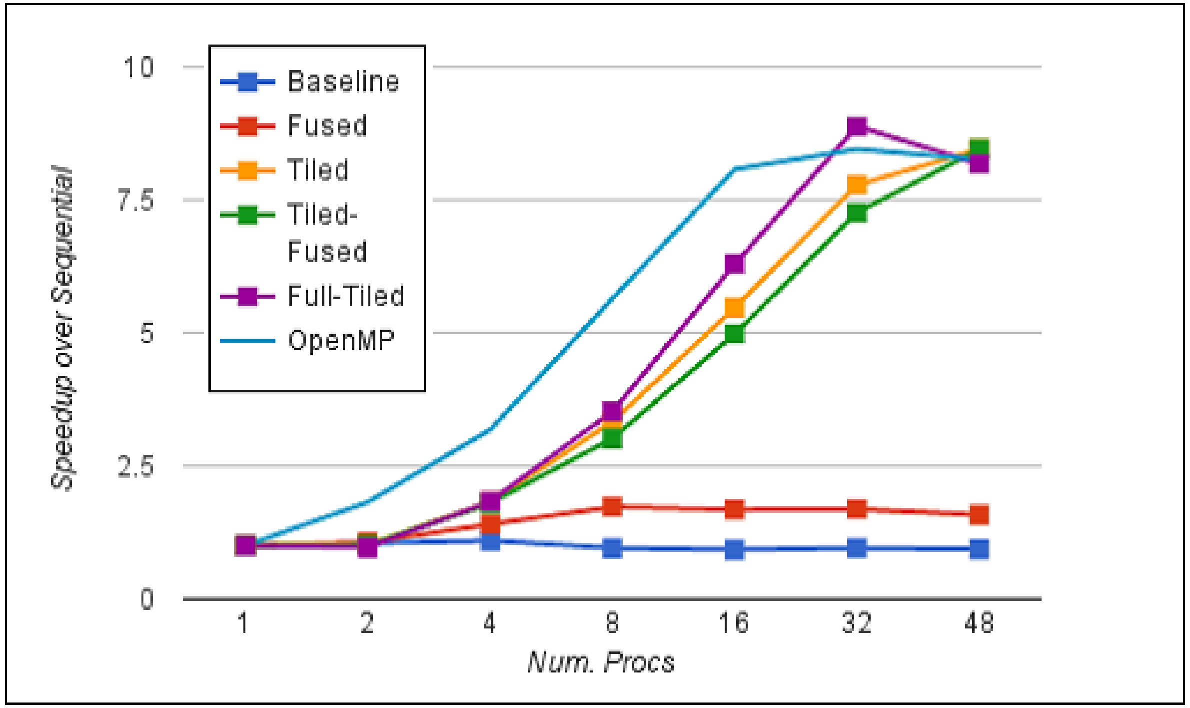


Experimental Results - LULESH

- 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: Mesh sized 60^3
  - Baseline
  - Fused-only
  - Tiled-only
  - Fused+Tiled
  - (Full) Data Tiled
  - OpenMP (LULESH 2.0.3 LLNL)

Speedup and Scalability





- OpenMP outperforms non-data-tiled implementations by ~10x
- [Full] Data Tiled is 3x faster than OpenMP
- Tag Tiling required for scalability, better than OpenMP (dedicated scheduling process)

PURDUE
UNIVERSITY

Contributions/Discussion

- Task coarsening helps obtain scalability
  - Legal transformations for step fusion & tag tiling at a high-level
- Task+Data coarsening gives strong parallel performance
  - Requires more underlying code modularity/changes
- Ease of Application/Programmability
  - CnC Translator: generates code skeleton from CnC graph (Nick)
  - Programmer: Still required to write sequential code - never worry about parallelism
  - Difficulty: Optimizing data layout and blocked routines for kernel computation
    - Required domain knowledge to capture semantic properties of method

Related

- Related Work
  - Varioius CnC Papers
  - Task Parallel Model – Charm++, OpenMP, Chapel, Legion
  - Fusion and Tiling – “*Fusion of Parallel Array Operations*”, MRB Kristensen

- Mentions
  - Nick Vrvilo – *“Declarative Tuning for Locality in Parallel Programs”*
  - Ellen Porter – HPC applications using CnC
  - Kath Knobe, Zoran Budimlic - Discussions

Conclusion

- CnC offers an easy way to obtain performance with ease of programmability
  - Provides the right abstractions for domain scientists and performance experts to decouple concerns
  - Gives a good platform to explore high-level optimizations such as computation reordering/tiling
  - Further performance improvements likely requires machine-specific tuning (HW mapping, etc)
- Future Improvements
  - Enhance CnC translator for automatic code generation
  - Pair data templates for 2D/3D spatial containers for common applications
  - Automatically locate redundant data synchronization for performance improvement

Thanks!

Concurrent Collections cont.

- Collections: General Program Structure
- Step Collections
  - Stateless computation tasks
- Item Collections
  - Data input/outputs used by steps
  - Explicit dependencies
  - Dynamically-single assignment using hashmap
- Tag Collections
  - Identifiers for the step collections
  - Dictates program control flow

```
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);
...
```

Declaration of CnC Specification