Optimistic Parallelism Benefits from Data Partitioning

Milind Kulkarni, Keshav Pingali, Ganesh Ramanarayanan, Bruce Walter, Kavita Bala and L. Paul Chew

Optimistic Parallelism Benefits from Data Partitioning

Milind Kulkarni, Keshav Pingali, Ganesh Ramanarayanan, Bruce Walter, Kavita Bala and L. Paul Chew

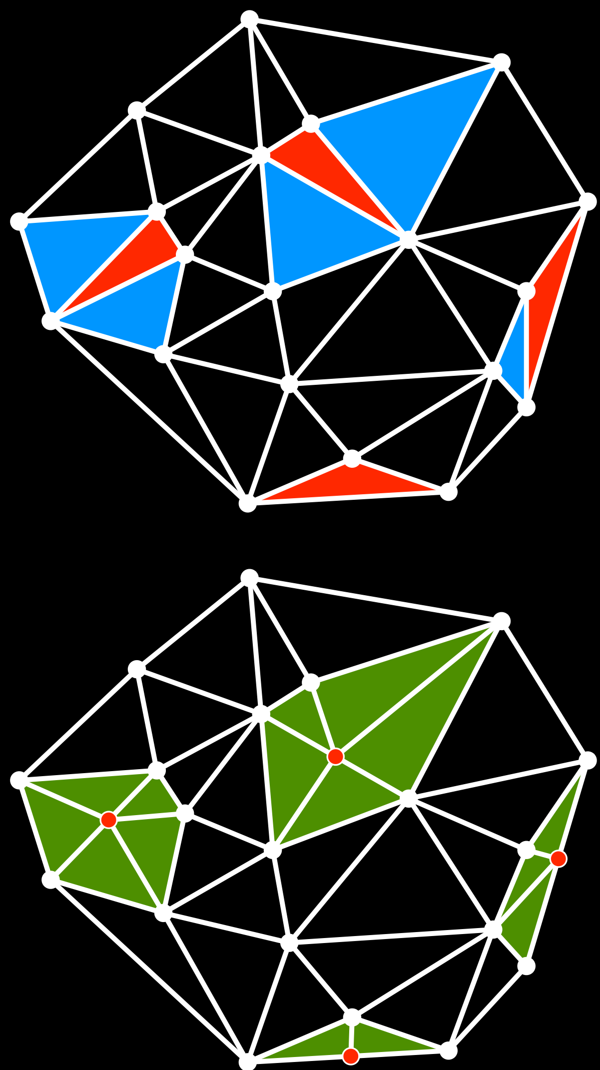
Parallelism in Irregular Programs

- Many irregular programs use iterative algorithms over worklists of various kinds
  - Delaunay mesh refinement
  - Image segmentation using graphcuts
  - Agglomerative clustering
  - Delaunay triangulation
  - SAT solvers
  - Iterative data-flow analysis
  - ...

Running Example
Mesh Refinement

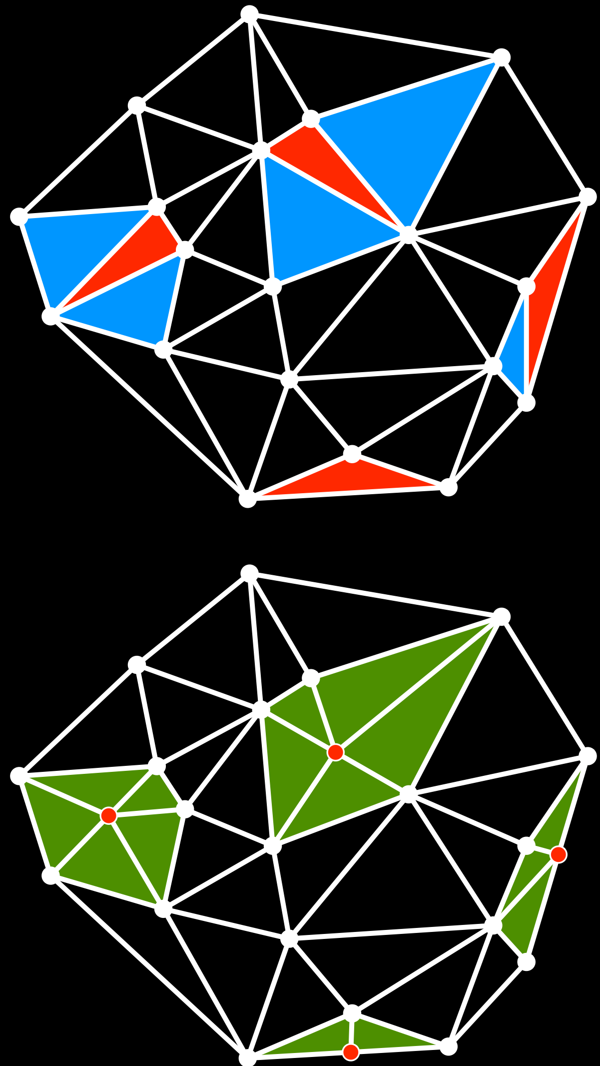
```
wl.add(mesh.badTriangles());

while (wl.size() != 0) {
   Element e = wl.get();
   if (e no longer in mesh)
      continue;
   Cavity c = new Cavity(e);
   c.expand();
   c.retriangulate();
   mesh.update(c);
   wl.add(c.badTriangles());
}
```



Generalized Data Parallelism

- Process elements from worklist in parallel
  - Deciding if cavities overlap must be done at runtime
- Can use optimistic parallelism
  - Speculatively process two triangles from worklist
  - If cavities overlap, roll back one iteration
- Implementation: Galois System (PLDI ’07)



Scalability Issues

- General Parallelization Issues
  - Maintaining Locality
  - Reducing contention for shared data structures
- Optimistic Parallelization Issues
  - Reducing mis-speculation
  - Lowering cost of run-time conflict detection

Locality vs. Parallelism in Mesh Refinement

- In sequential version, worklist implemented as stack, for locality
  - If run in parallel, high likelihood of cavity overlap
- Another option: assign work to cores randomly, to reduce likelihood of conflict
  - Reduces locality

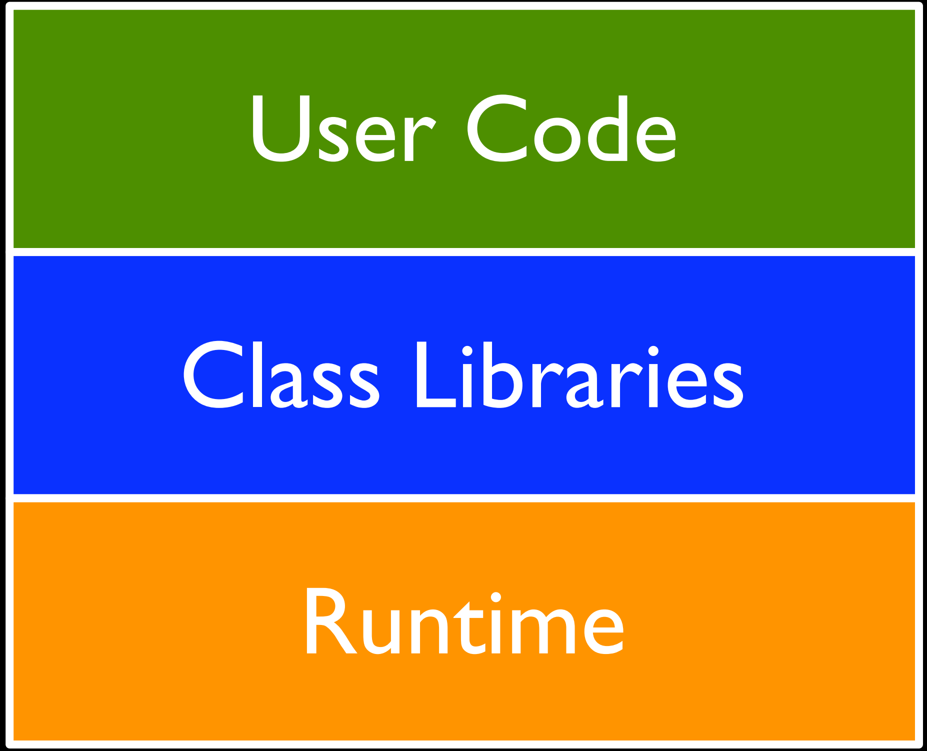
Outline

- Overview of Galois System
- Addressing Scalability
  - Data Partitioning
  - Computation Partitioning
  - Lock Coarsening
- Evaluation and Conclusion

The Galois System

- Programming Model and Implementation to support optimistic parallelization of irregular programs

- User code: What to parllelize
- Class Libraries + Runtime: How to parallelize correctly



User Code

- Sequential semantics
- Use optimistic set iterator to expose opportunities for exploiting data parallelism

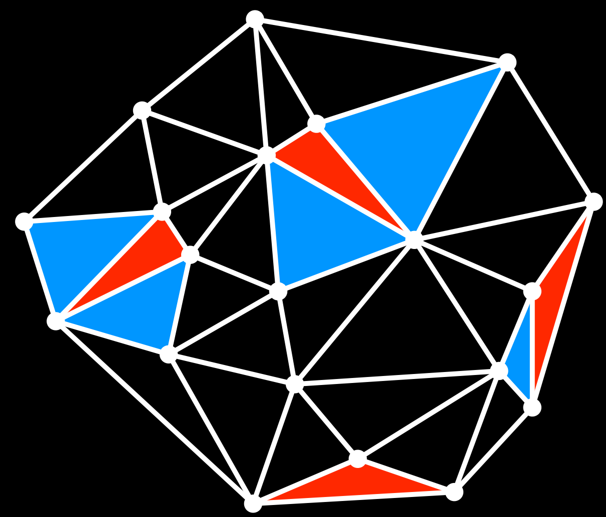
```
foreach e in Set s do B(e)
```

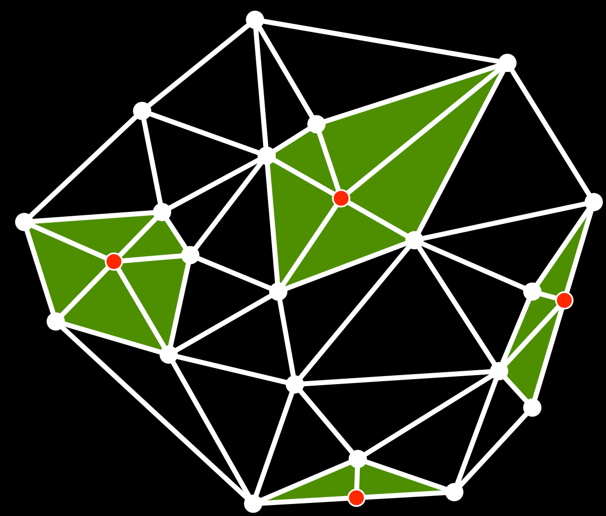
- Can add new elements to set during iteration

What to Parallelize

```
wl.add(mesh.badTriangles());

while (wl.size() != 0) {
    Element e = wl.get();
    if (e no longer in mesh)
        continue;
    Cavity c = new Cavity(e);
    c.expand();
    c.retriangulate();
    mesh.update(c);
    wl.add(c.badTriangles());
}
```



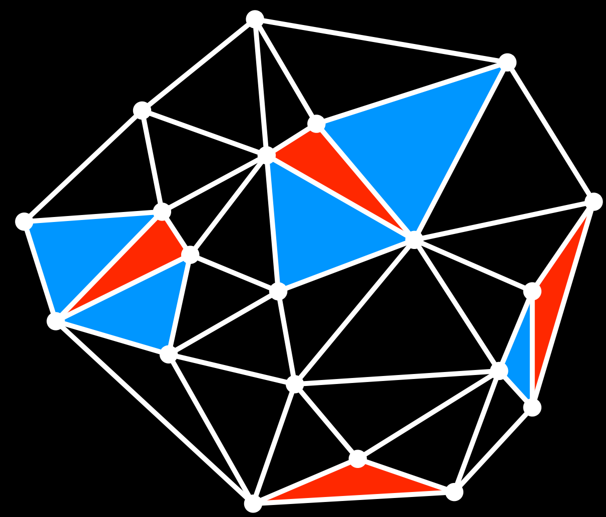


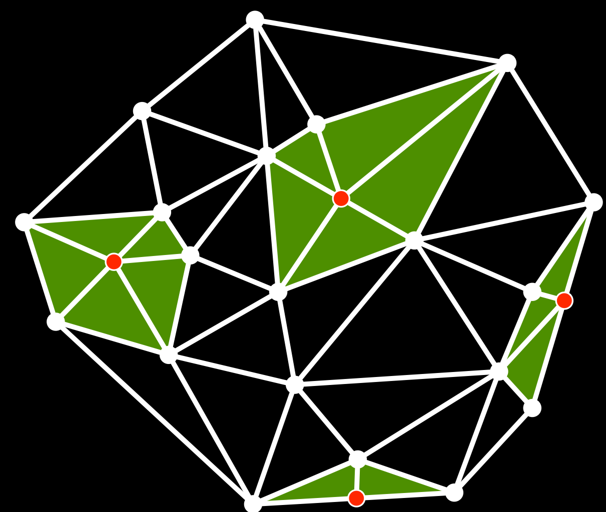
What to Parallelize

```
wl.add(mesh.badTriangles());

foreach Element e in wl {

    if (e no longer in mesh)
       continue;
    Cavity c = new Cavity(e);
    c.expand();
    c.retriangulate();
    mesh.update(c);
    wl.add(c.badTriangles());
}
```





Execution Model

- Shared memory encapsulated in objects
- Program runs sequentially until set iterator encountered
- Multiple threads execute iterations from worklist
    - Scheduler assigns work to threads

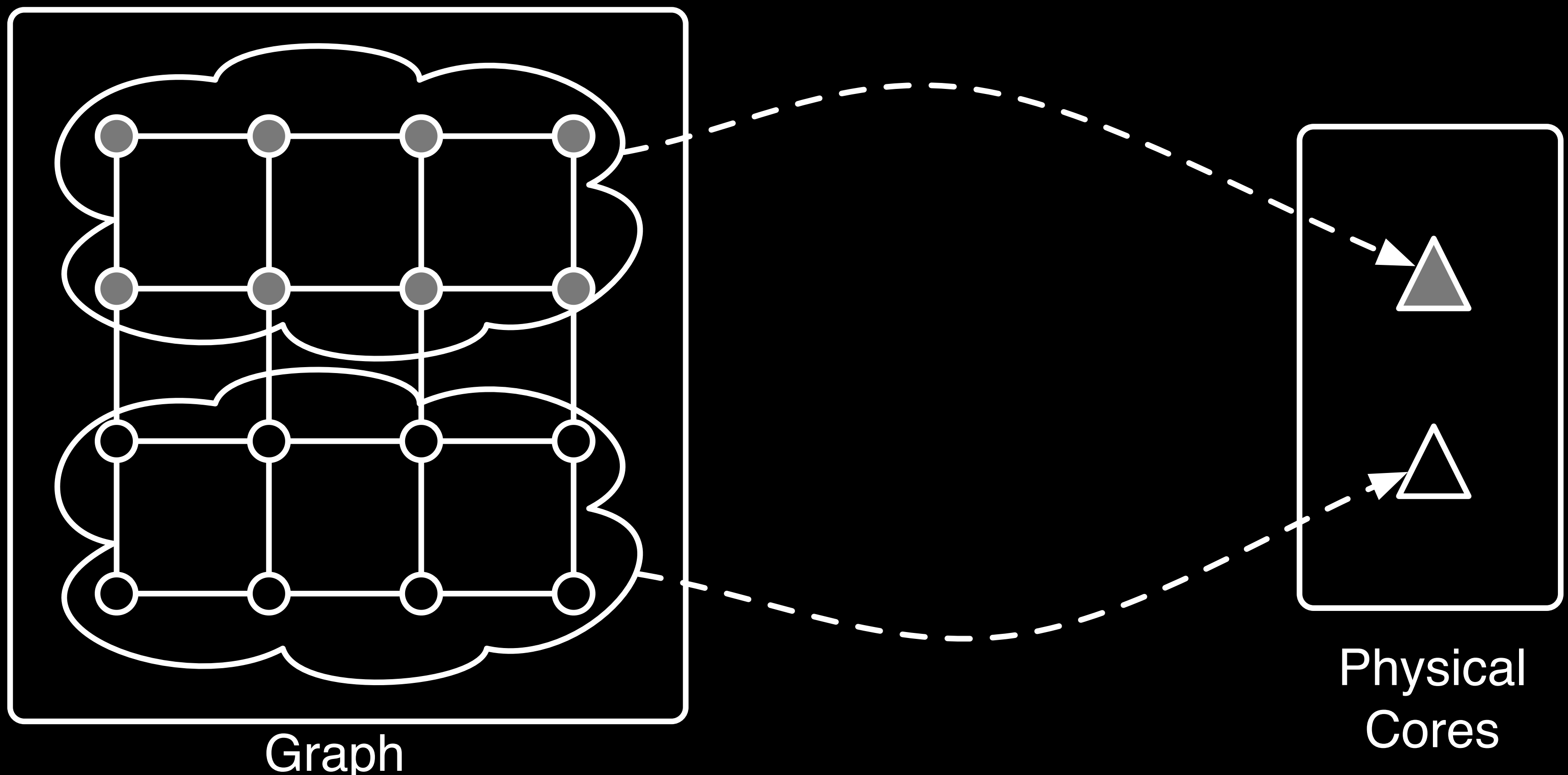
Class Libraries + Runtime

- Ensure that iterations run in parallel only if independent
- Detect dependences between iterations using semantic commutativity
    - Uses semantic properties of objects to determine dependence
- If conflict, roll back using undo methods

Outline

- Overview of Galois System
- Addressing Scalability
  - Data Partitioning
  - Computation Partitioning
  - Lock Coarsening
- Evaluation and Conclusion

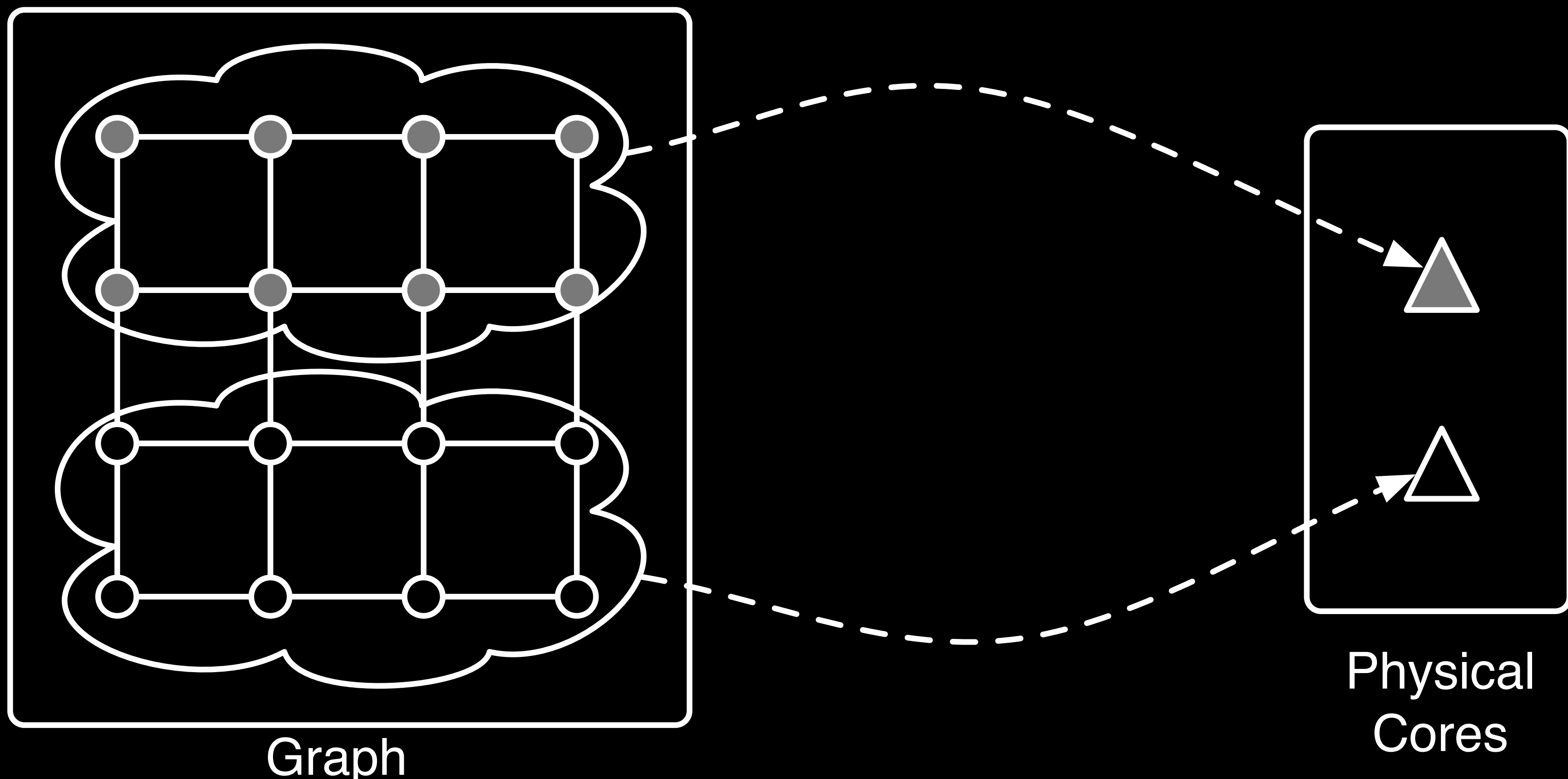
Data Partitioning



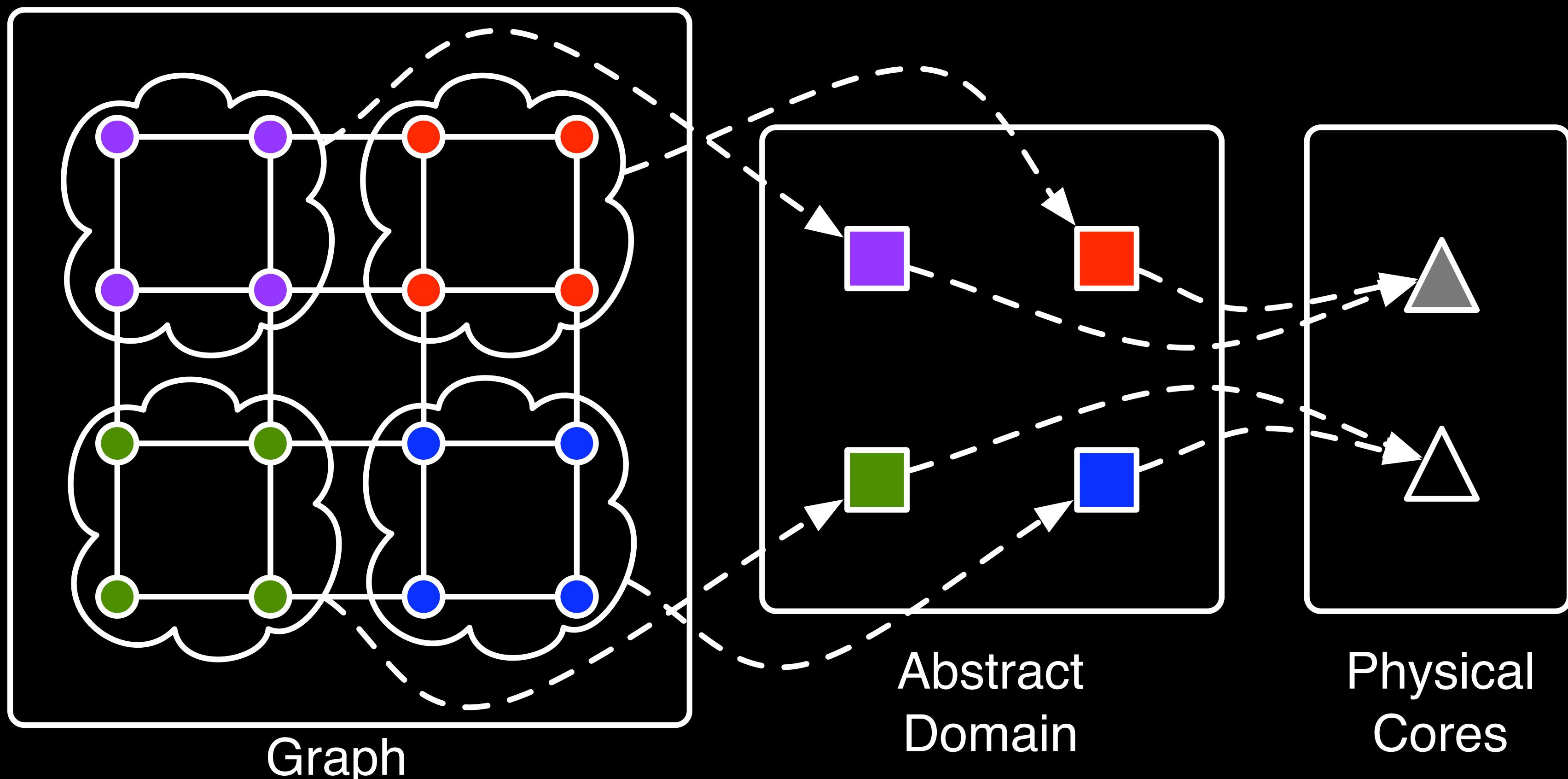
Abstract Domain

- Set of abstract processors mapped to physical cores
  - Data structure elements mapped to abstract processors
- Allows for overdecomposition
  - More abstract processors than cores
  - Useful in many contexts (e.g. load balancing)

Abstract Domain

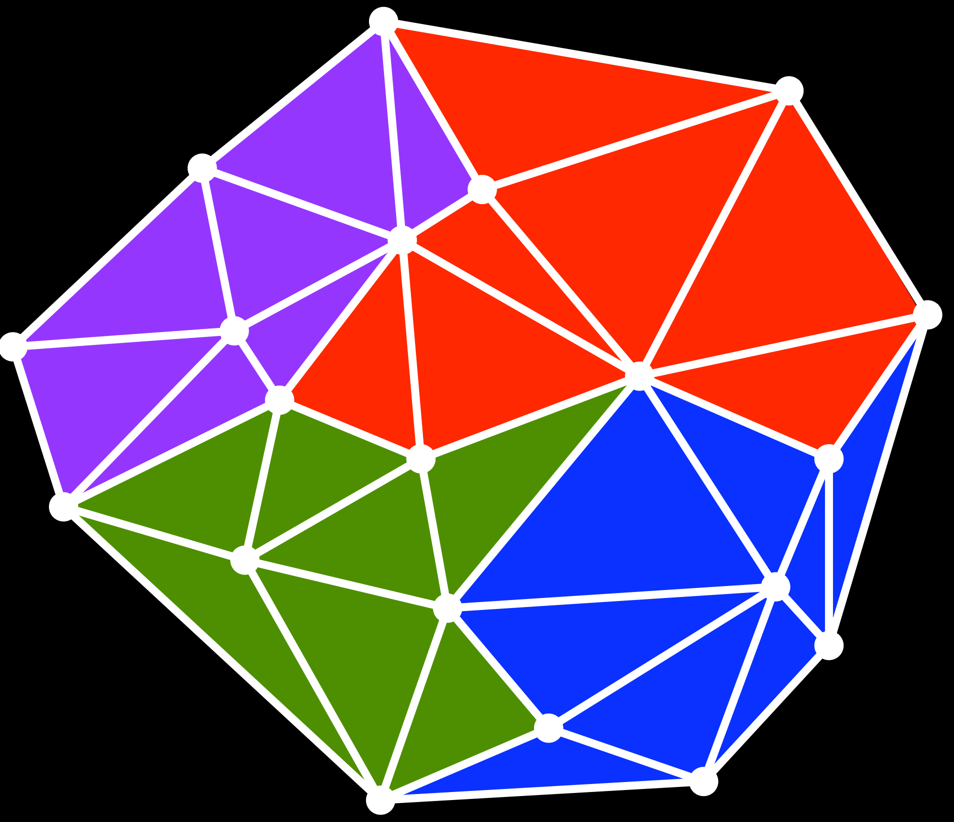


Abstract Domain



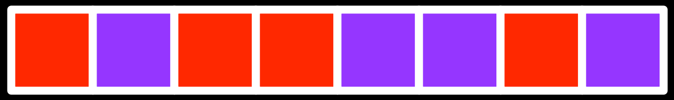
Logical Partitioning

- Elements of data structure (e.g. triangles in the mesh) are mapped to abstract processors
- Add "color" to data structure elements
- Promotes locality
  - Cavities small and contiguous → likely to be in a single partition



Physical Partitioning

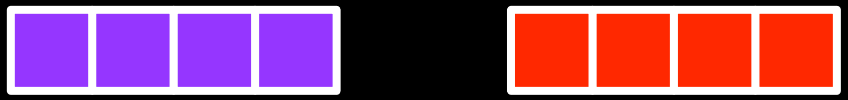
- Reimplementation of data structure to leverage logical partitioning
  - e.g. Worklist:



- Allows different partitions of data structure to be accessed concurrently
  - Reduces contention

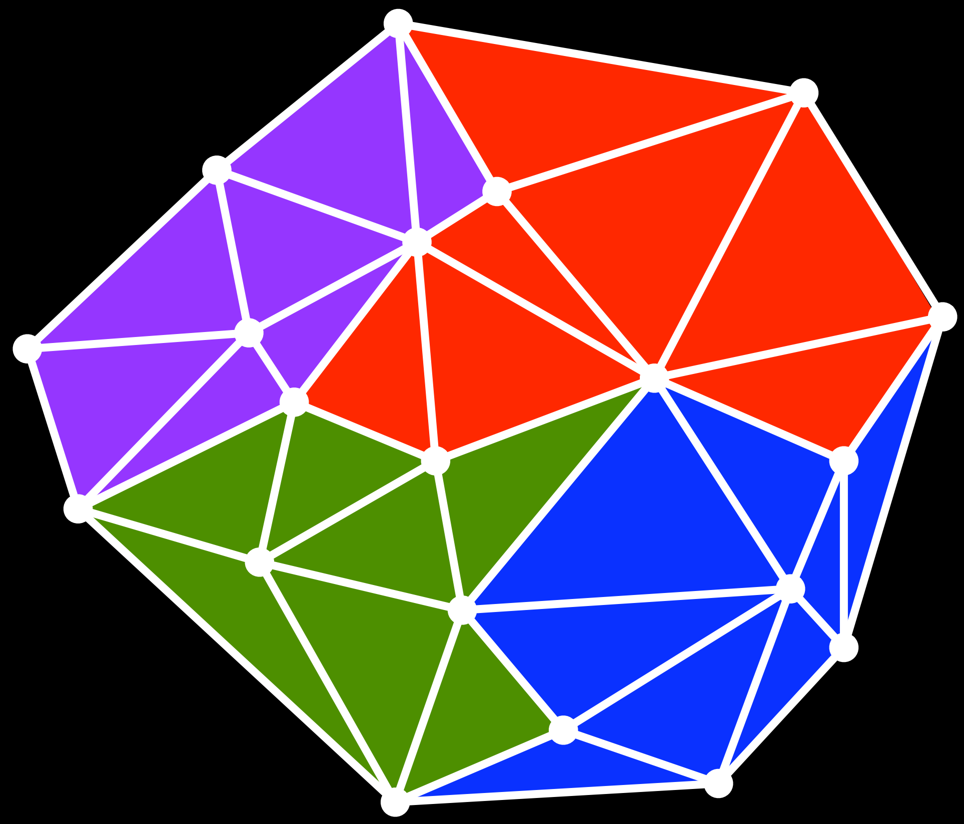
Physical Partitioning

- Reimplementation of data structure to leverage logical partitioning
    - e.g. Worklist:

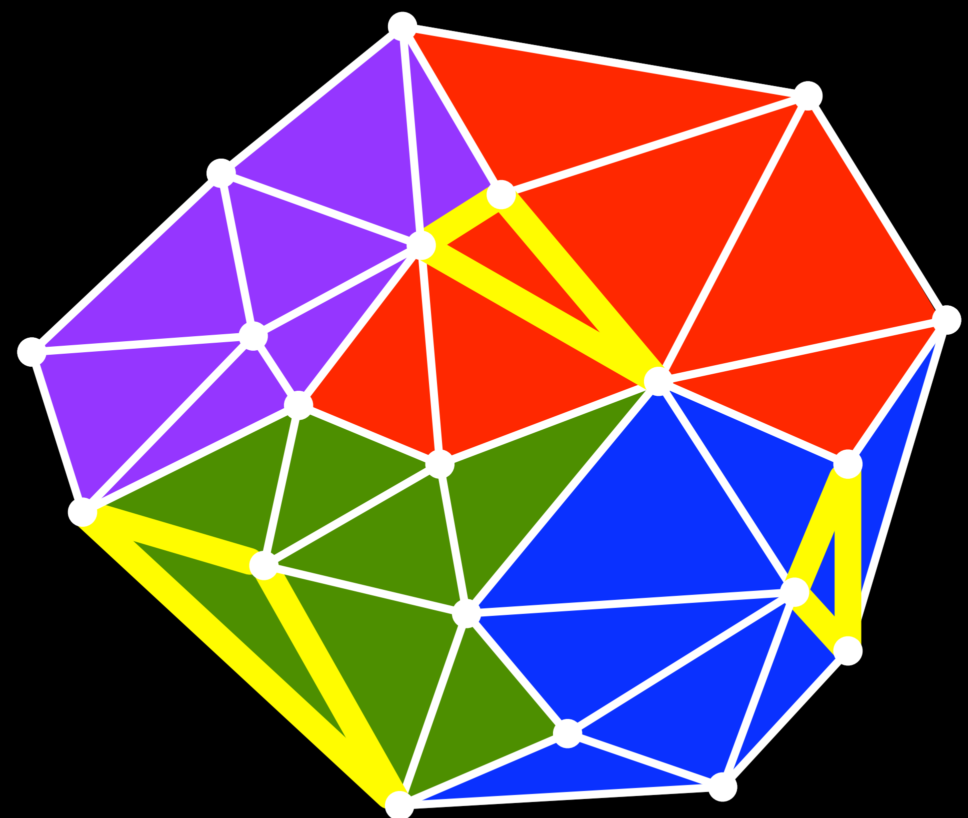


- Allows different partitions of data structure to be accessed concurrently
    - Reduces contention

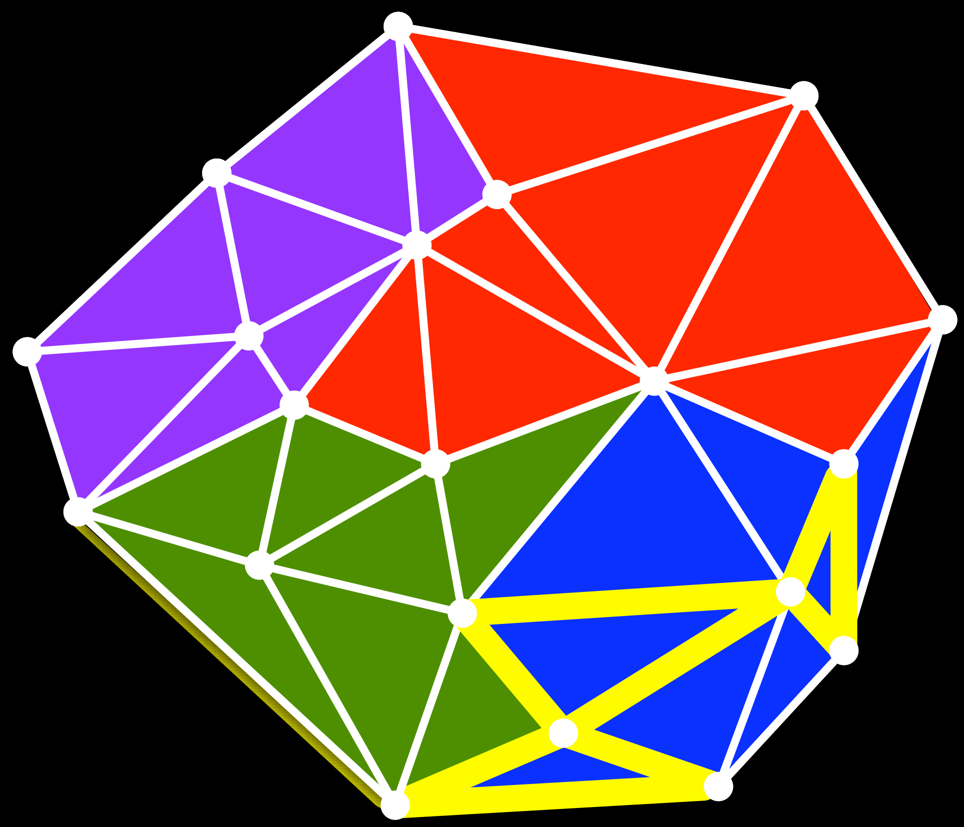
Computation Partitioning



Computation Partitioning



Computation Partitioning



Data + Computation Partitioning

- Data Partitioning → most cavities contained within a single partition
- Computation Partitioning → each partition touched mostly by one core

➡ Partitions are effectively "bound" to cores

- Maintains good locality
- Reduces misspeculation

Outline

- Overview of Galois System
- Addressing Scalability
  - Data Partitioning
  - Computation Partitioning
  - Lock Coarsening
- Evaluation and Conclusion

Overheads from Conflict Checking

- Significant source of overhead in Galois: conflict checks
- Checks themselves computationally expensive
- Checks for each object must serialize to ensure correctness → bottleneck
- Can we take advantage of partitioning?

Optimization: Lock Coarsening

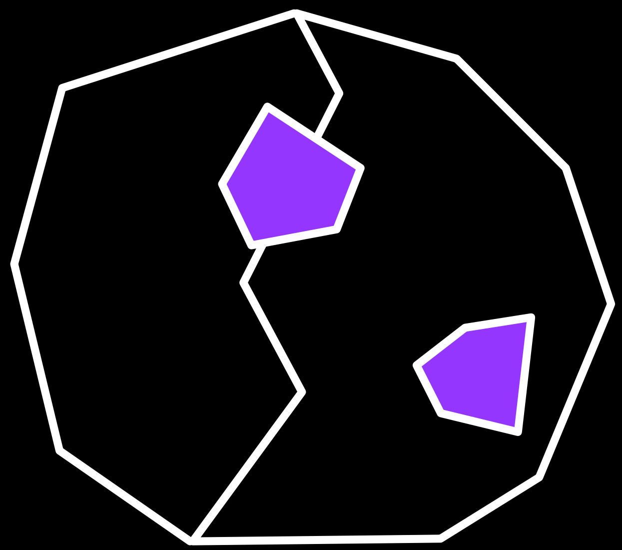
- Can often replace conflict checks with lightweight, distributed checks
  - Iteration locks partitions as needed
    - Lock owned by someone else → conflict
  - Release locks when iteration completes

Upshot

- Synchronization dramatically reduced
  - While iteration stays within a single partition, only one lock is acquired
- Conflict checks are distributed, eliminating bottleneck

Overdecomposition

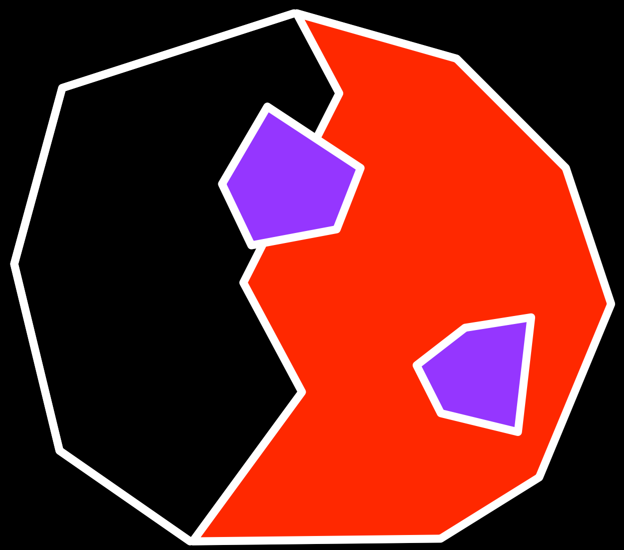
- Lock coarsening is an imprecise way to check for conflicts



- Overdecompose to reduce likelihood of conflict

Overdecomposition

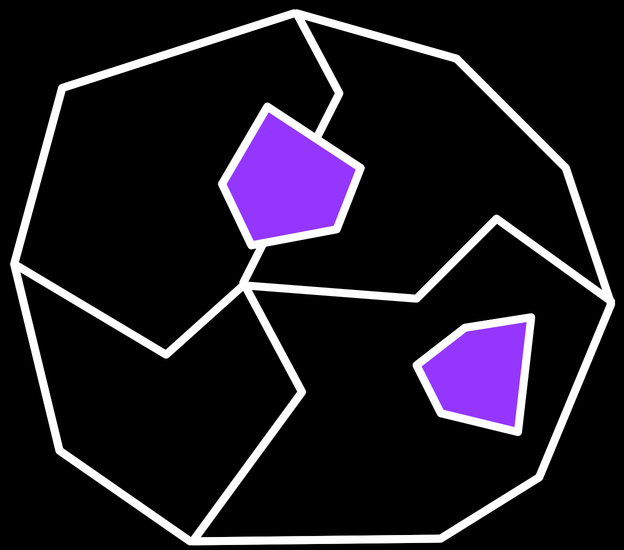
- Lock coarsening is an imprecise way to check for conflicts



- Overdecompose to reduce likelihood of conflict

Overdecomposition

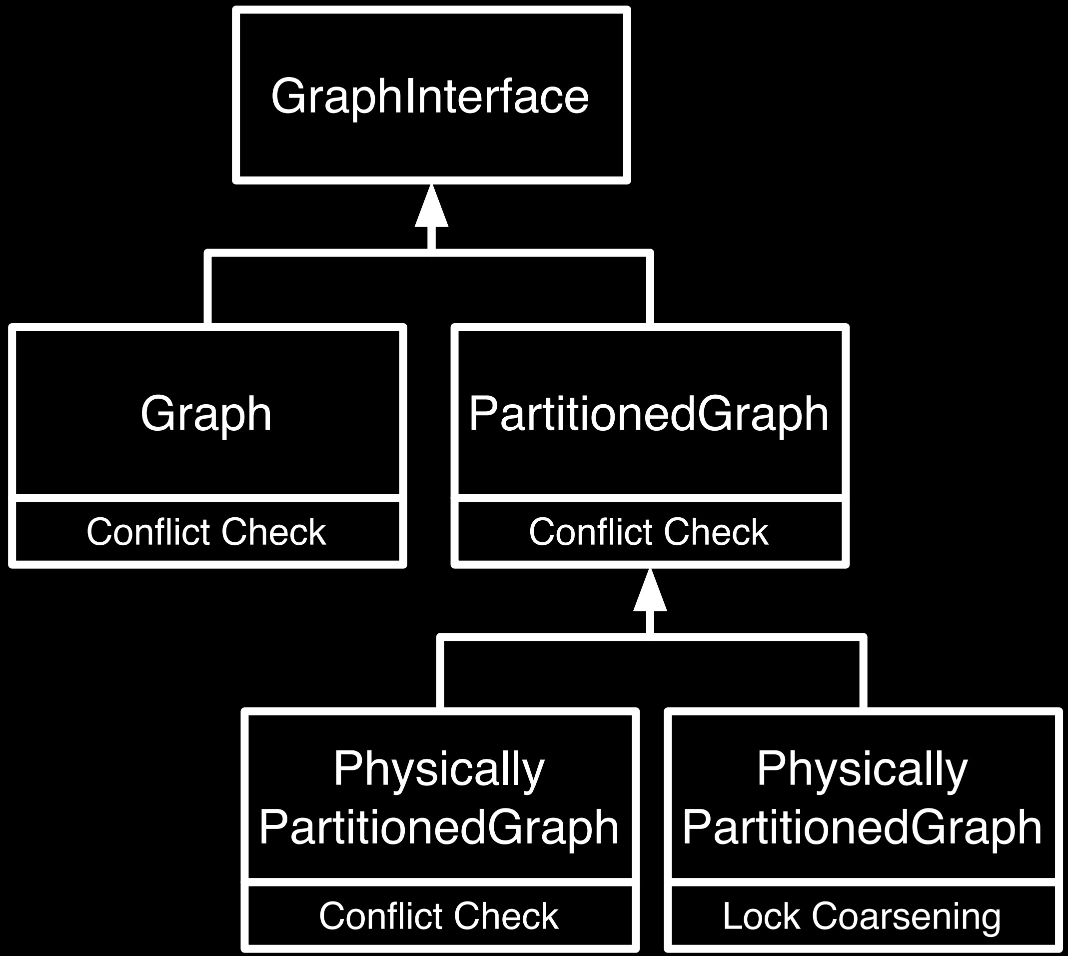
- Lock coarsening is an imprecise way to check for conflicts



- Overdecompose to reduce likelihood of conflict

Implementation

- Modify run-time to support computation partitioning
- Extend classes in Class Library to support data partitioning and/or lock coarsening
- User code only needs to change object instantiation



Outline

- Overview of Galois System
- Addressing Scalability
  - Data Partitioning
  - Computation Partitioning
  - Lock Coarsening
- Evaluation and Conclusion

Evaluation

- Four-core system
  - Intel Xeon processors @ 2GHz
- Implementation in Java 1.6

Benchmarks

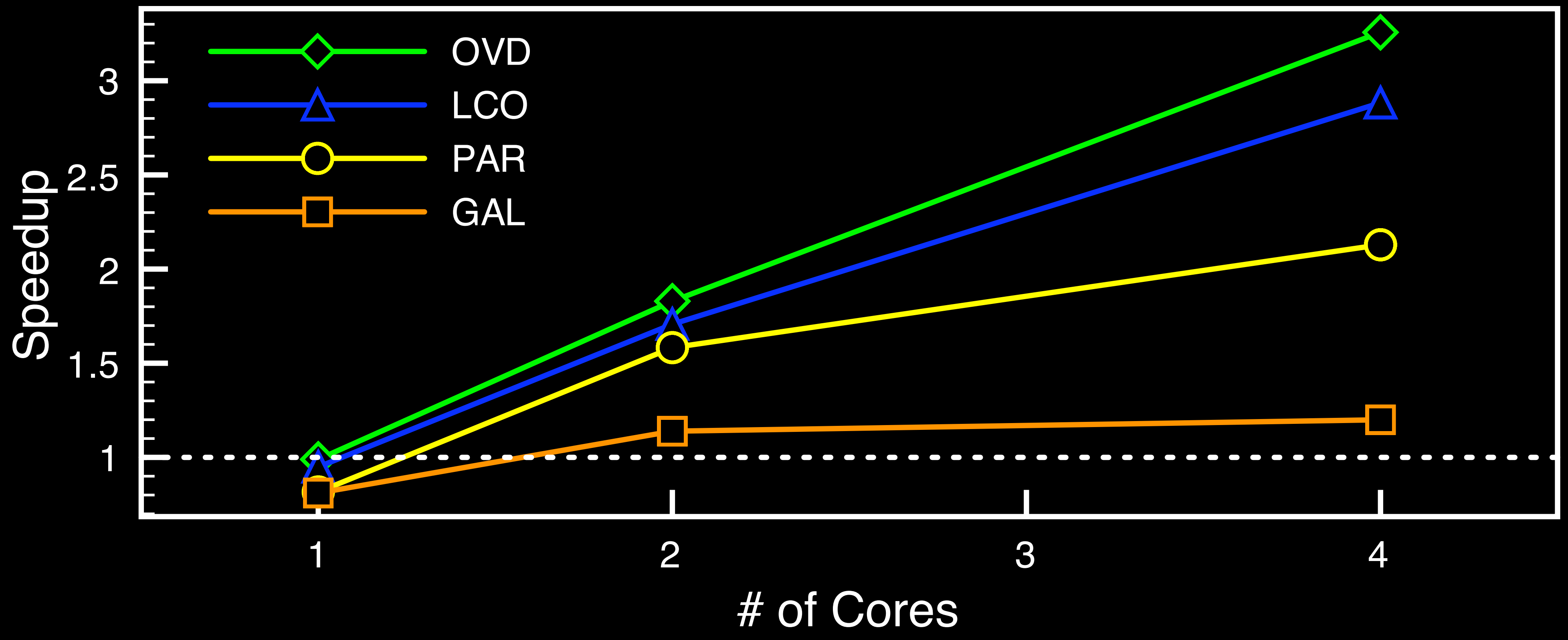
- Delaunay mesh refinement
- Augmenting-paths maxflow
- Preflow-push maxflow
- Agglomerative clustering

Different parallelization strategies

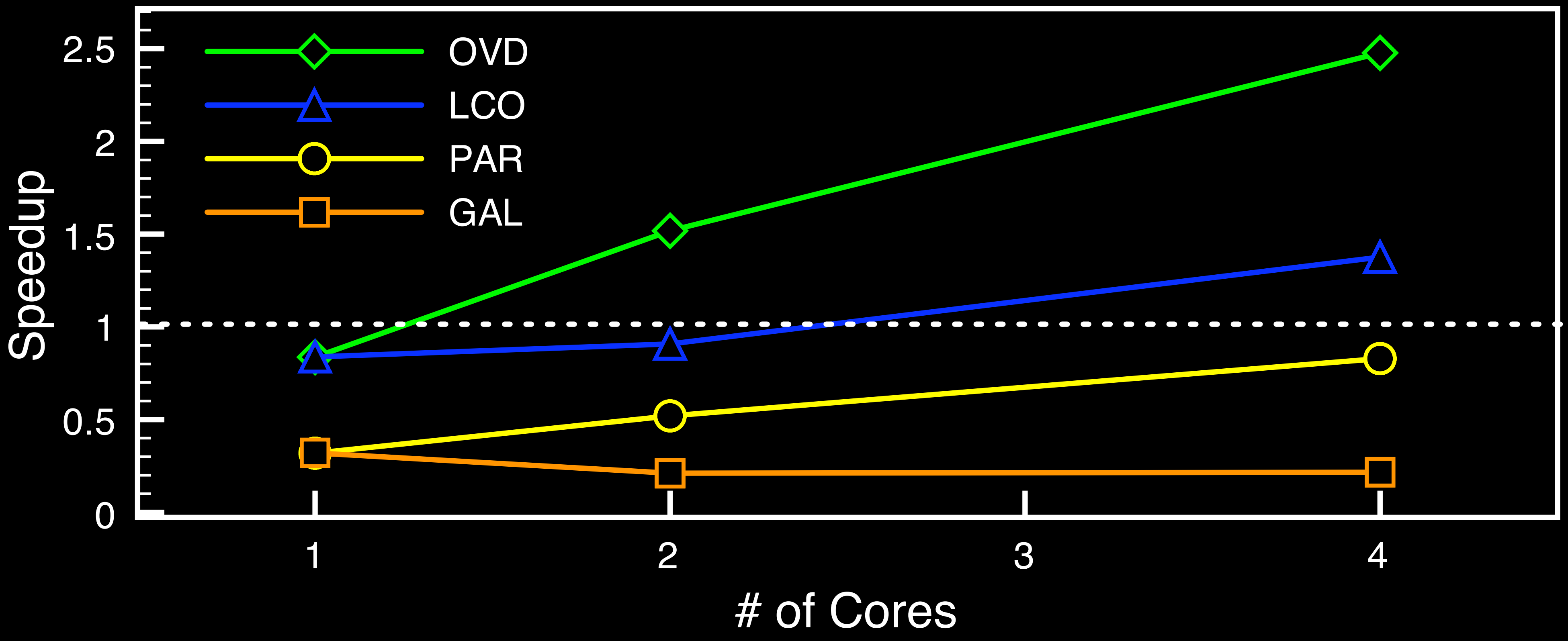
- Baseline Galois (*gal*)
- Partitioned Galois (*par*)
- Lock coarsening (*lco*)
- Lock coarsening + overdecomposition (*ovd*)

- Measure speedup versus sequential execution time

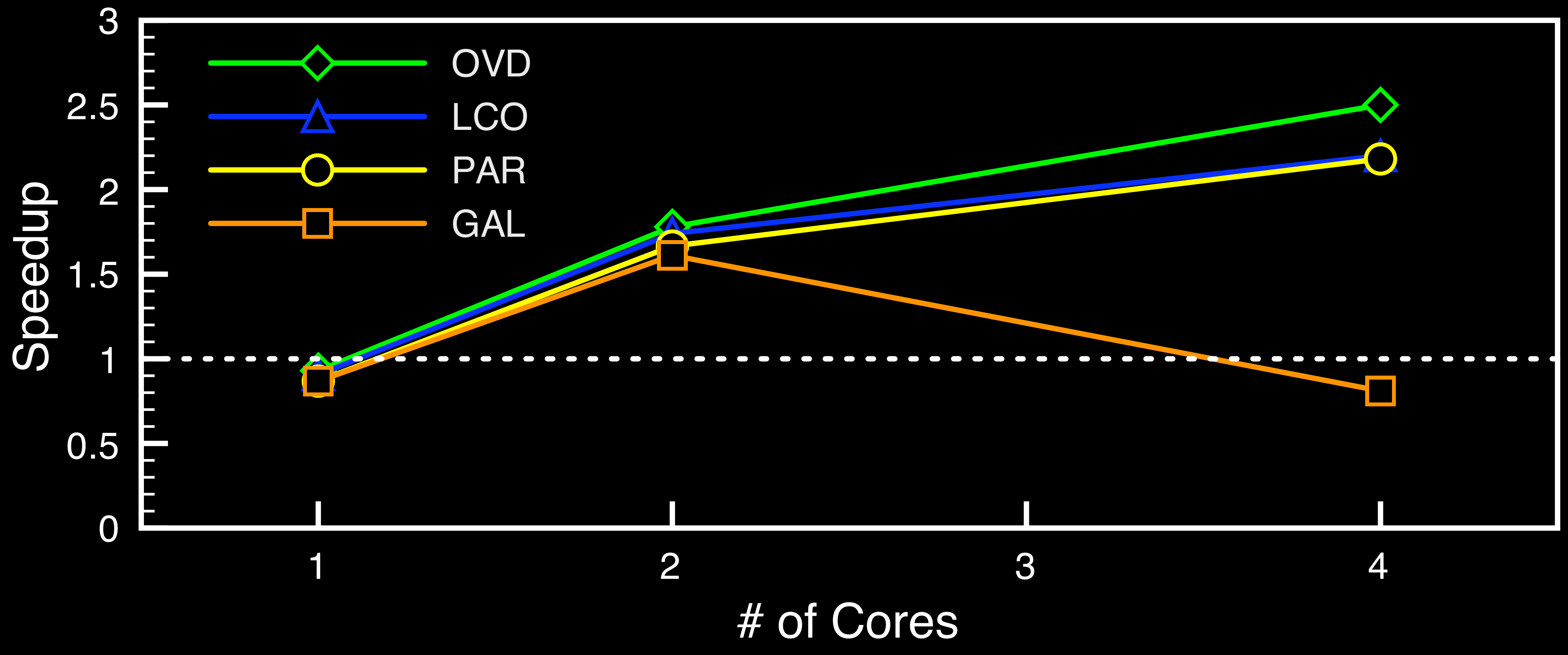
Delaunay Mesh Refinement



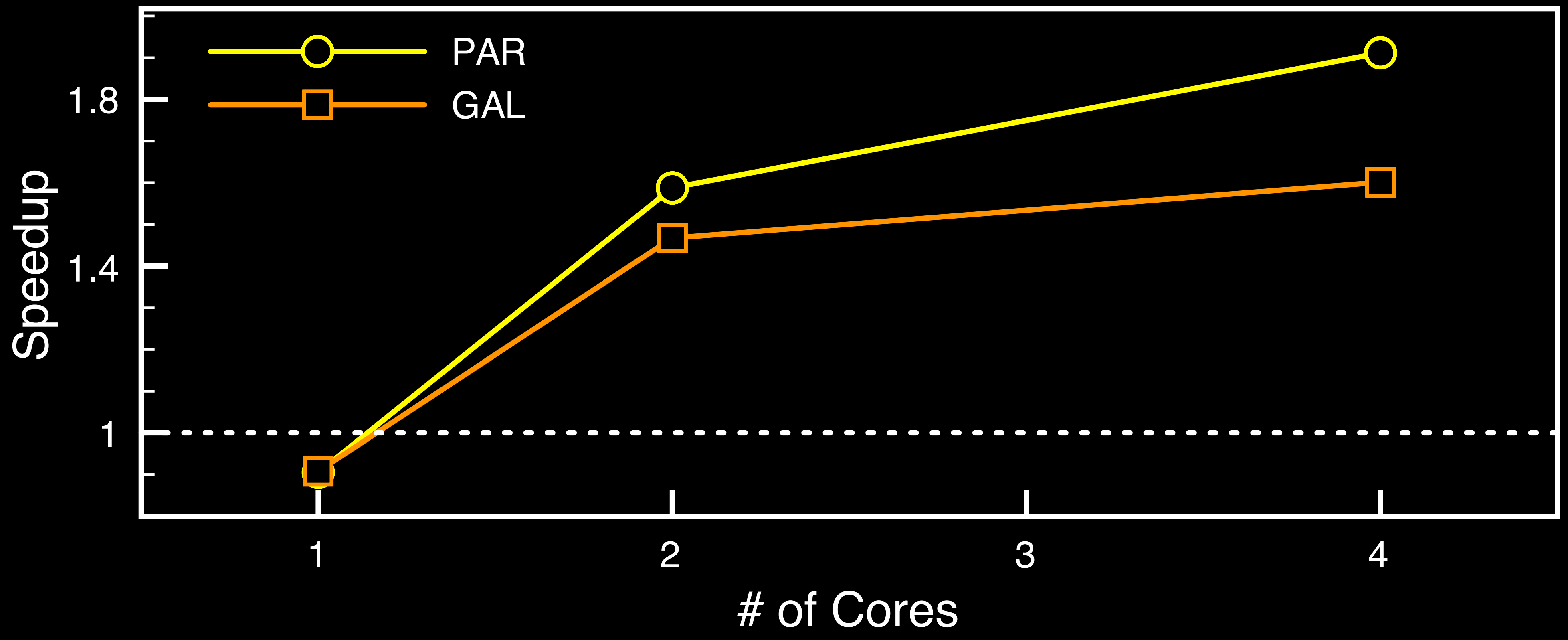
Augmenting Paths



Preflow Push



Agglomerative Clustering



Summary

- Addressed issues that arise in any optimistic parallelization system:
  - Tradeoff between locality and parallelism
  - Contention for shared data structures
  - Overhead of conflict checks

- Low programmer overhead

Summary

- Addressed issues that arise in any optimistic parallelization syste…
  - Trade… …y and parallelism
  - Contention for shared data structures
  - Overhead of conflict checks

Logical Partitioning +
Computation Partitioning

- Low programmer overhead

Summary

- Addressed issues that arise in any optimistic parallelization syste…
  - Trade … parallelism
  - Cont … uctures
  - Overhead of conflict checks

Logical Partitioning +
Computation Partitioning

Physical Partitioning

- Low programmer overhead

Summary

- Addressed issues that arise in any optimistic parallelization syst
  - Trade ... parallelism
  - Cont ... uctures
  - Over

Logical Partitioning + Computation Partitioning

Physical Partitioning

Lock Coarsening + Overdecomposition

- Low programmer overhead

Questions/Comments?

milind@cs.cornell.edu
http://www.cs.cornell.edu/w8/~milind