Scheduling Strategies for Optimistic Parallel Execution of Irregular Programs

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

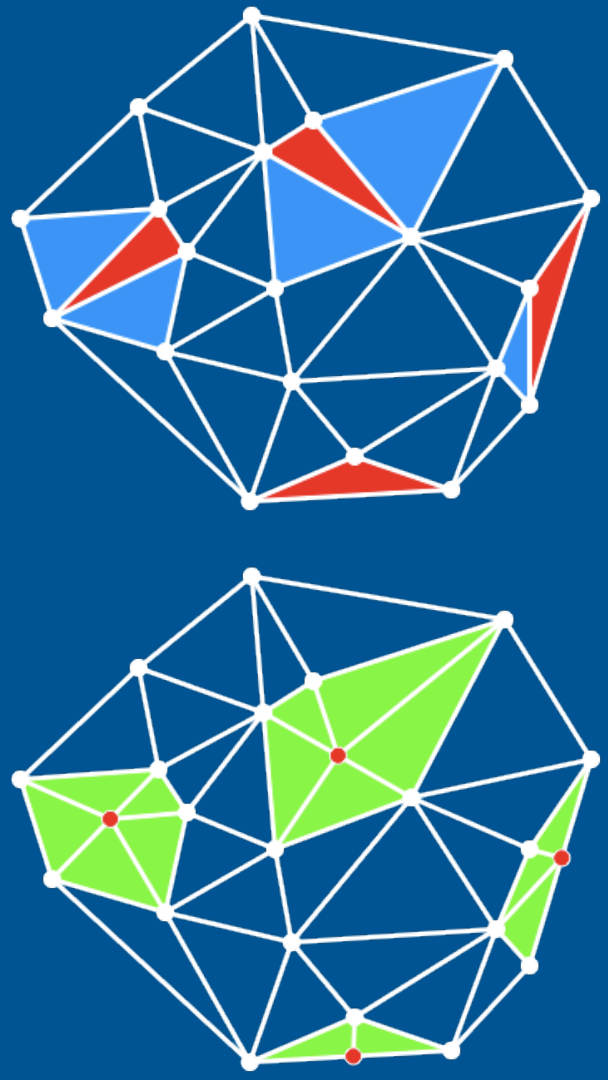
University of Texas at Austin
Cornell University

Amorphous Data Parallelism

- Many irregular programs implement iterative algorithms over worklists
  - Mesh refinement, agglomerative clustering, maxflow algorithms, compiler analyses, ...
- Complex dependences between iterations
- But many iterations can be executed in parallel
- New elements can be added to worklist

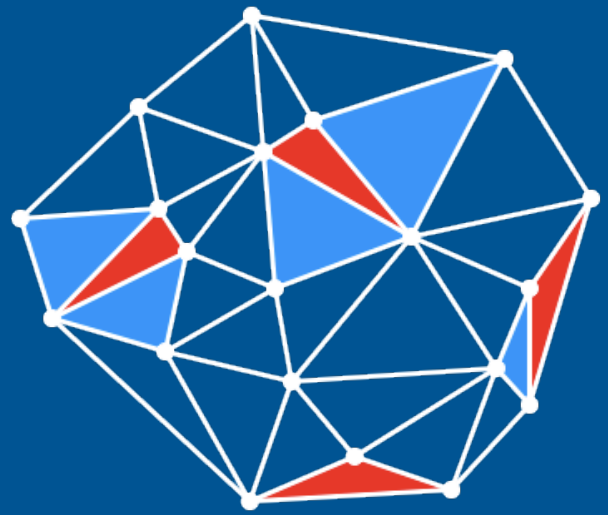
Delaunay Mesh Refinement (DMR)

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

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



Delaunay Mesh Refinement (DMR)

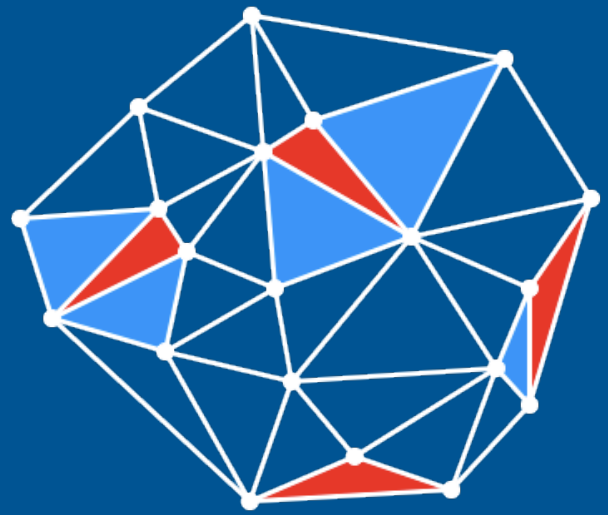
```
Worklist wl;
wl.add(mesh.badTriangles());

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



No ordering constraints on processing of worklist items

Parallelism in DMR

- Can process bad triangles concurrently
  - As long as cavities do not overlap
  - Cannot determine this until run time
- Example of amorphous data parallelism
- Our approach: Galois system for optimistic parallelization [PLDI'07, ASPLOS'08]

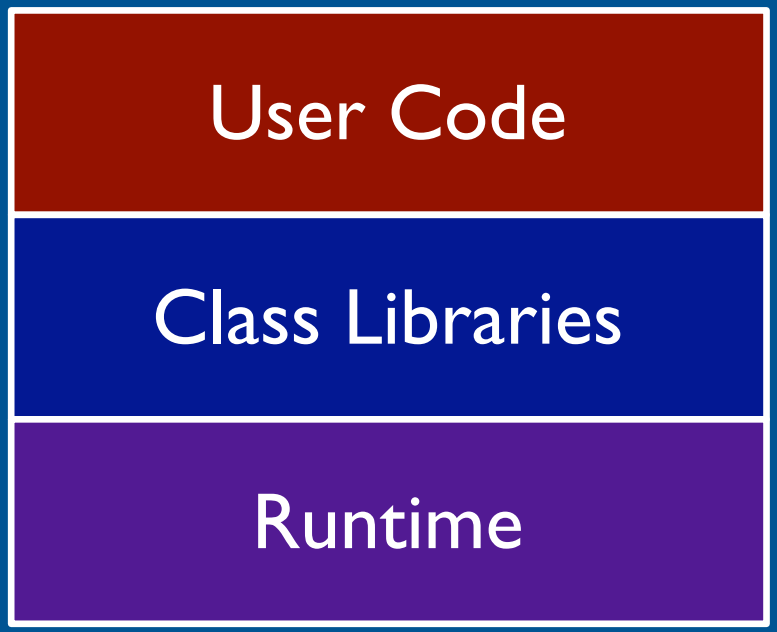


Galois System

- User code
  - Optimistic iterators

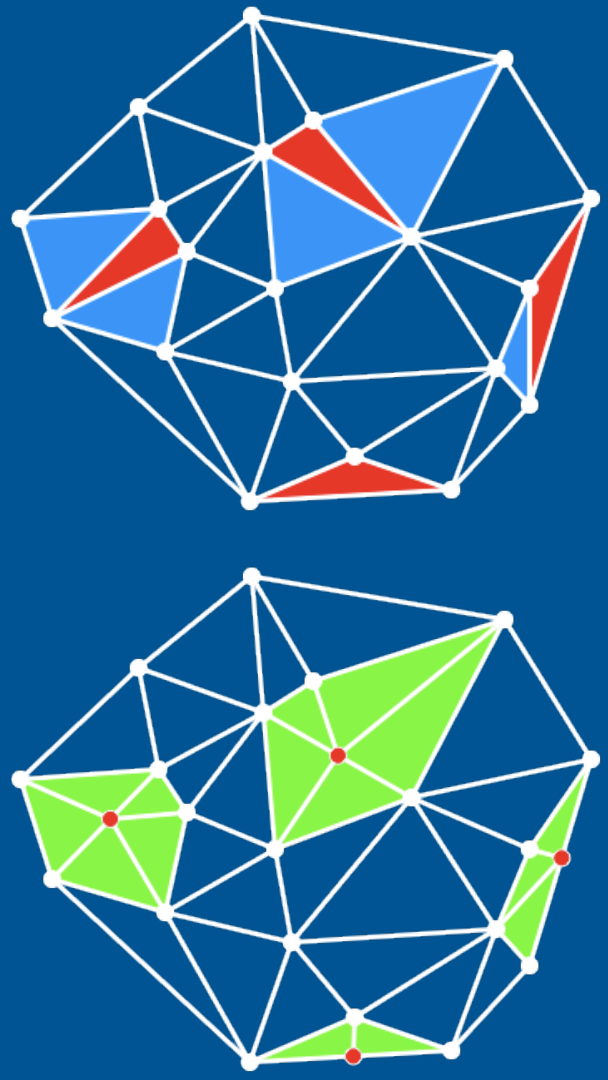
    ```
    foreach e in Set s do B(e)
    ```

  - Sequential Semantics
- Class libraries
  - Data structures
  - Conflict conditions
- Runtime system
  - Optimistic parallelization
  - Conflict detection & handling



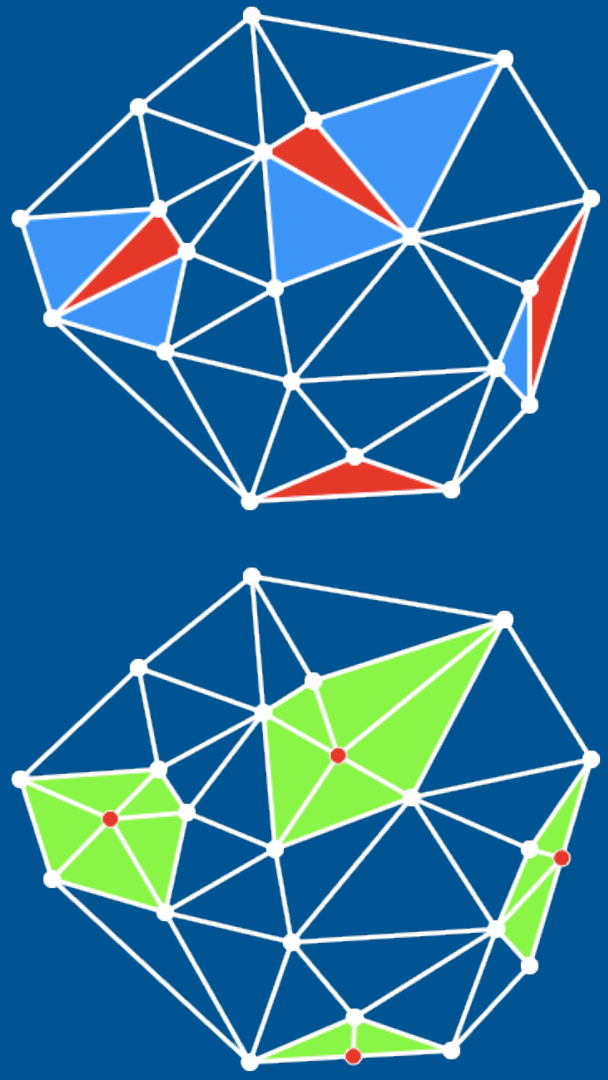
DMR User Code

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

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

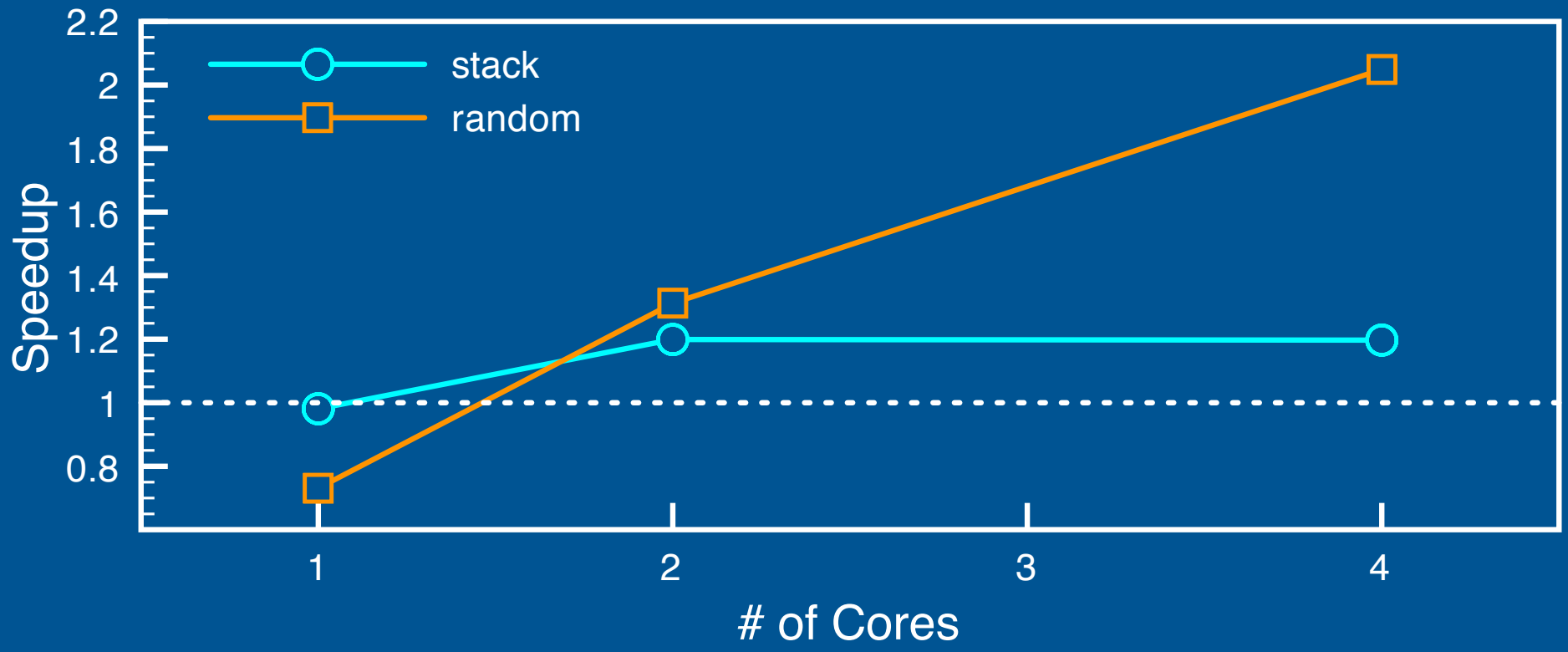


DMR User Code

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

foreach Triangle t in wl {

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



Scheduling Impact: DMR



Evaluation platform: 4-core Xeon system, running Java 1.6 HotSpot JVM

Input mesh: 100K triangles, ~40K bad triangles

Scheduling in OpenMP

- OpenMP provides parallel DO-ALL loops for regular programs
- Major scheduling concerns are load-balancing and overhead
- OpenMP scheduling policies address these issues
  - static, dynamic, guided

Amorphous Data Parallelism Issues

- Algorithmic – The efficiency of the algorithm or data structures
- Conflicts – The likelihood that two iterations executed in parallel will conflict
- Locality – The temporal or spatial locality exhibited in the data structures
- Dynamically created work
- Load-balancing and contention still an issue

Scheduling Basics

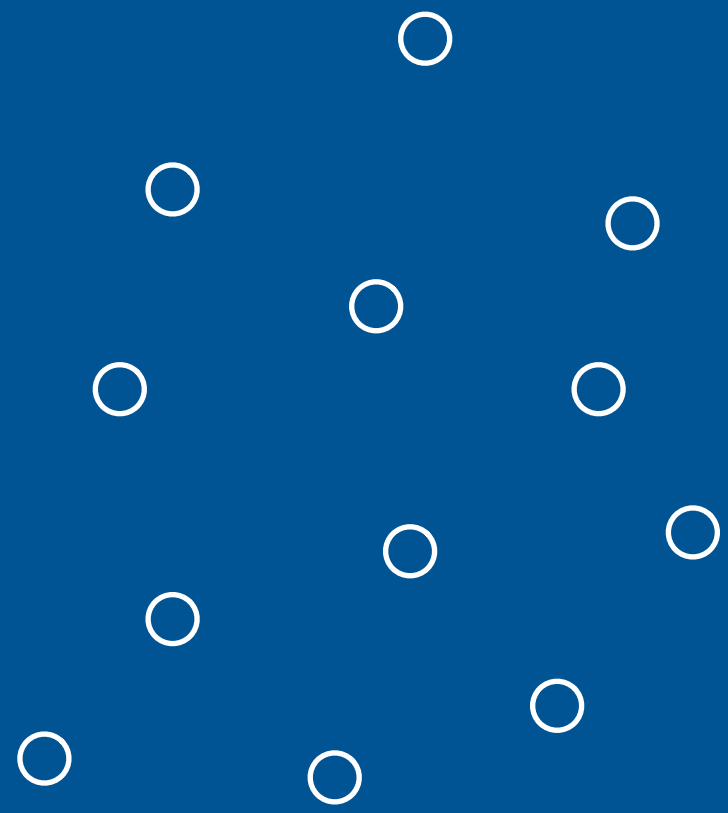
- Each iteration is executed by a single core
- Each core executes a set of iterations in a linear order
- Scheduling maps work from an "iteration space" to positions in an "execution schedule"
  - Each iteration is mapped to a core, and a position in that core's execution schedule

Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

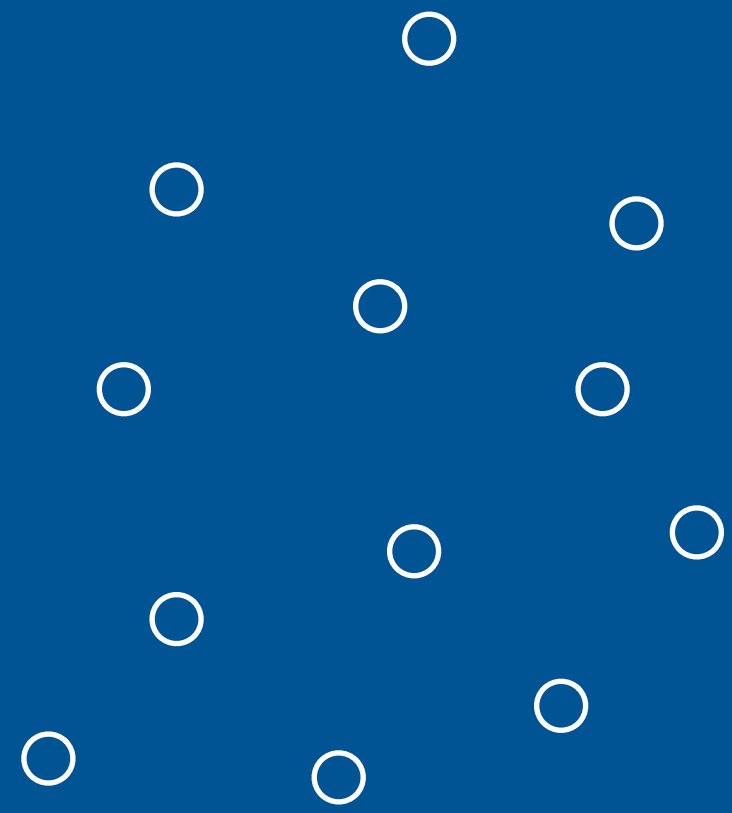
Labeling – Maps clusters to cores; Each core can have multiple clusters

Ordering – Specifies a serial execution order for each core



Scheduling Functions

- Clustering – Groups iterations into clusters; Each cluster executed on a single core
- Labeling – Maps clusters to cores; Each core can have multiple clusters
- Ordering – Specifies a serial execution order for each core

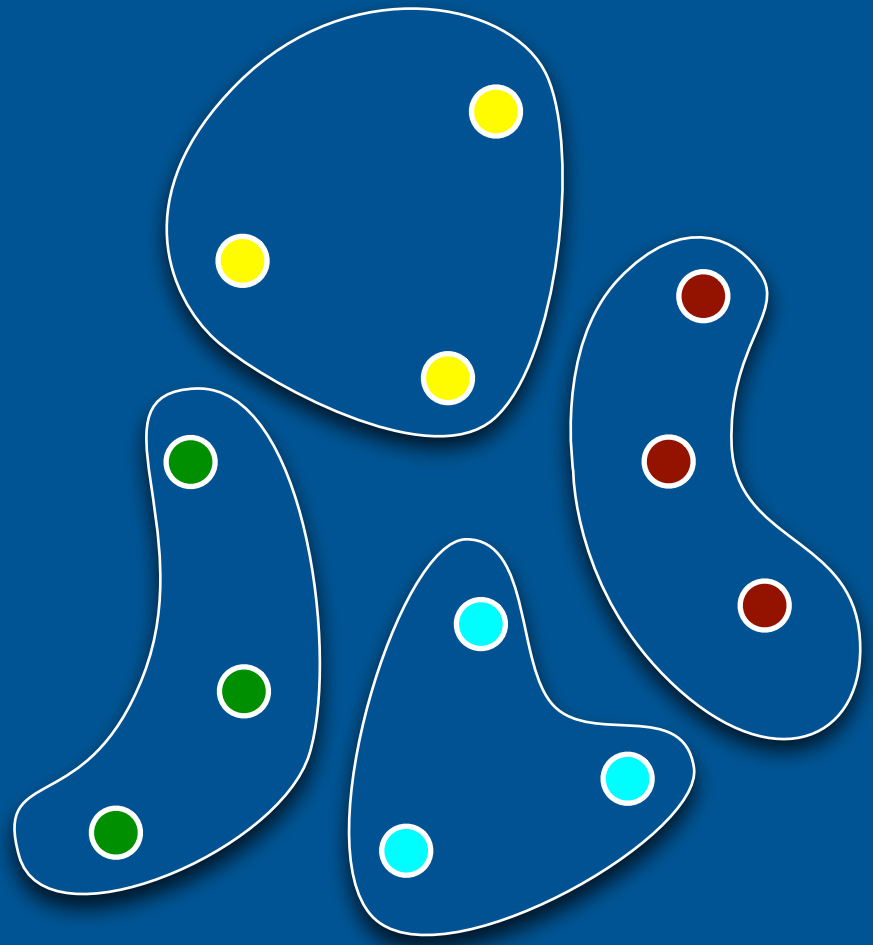


Scheduling Functions

- Clustering – Groups iterations into clusters; Each cluster executed on a single core

  Labeling – Maps clusters to cores; Each core can have multiple clusters

  Ordering – Specifies a serial execution order for each core

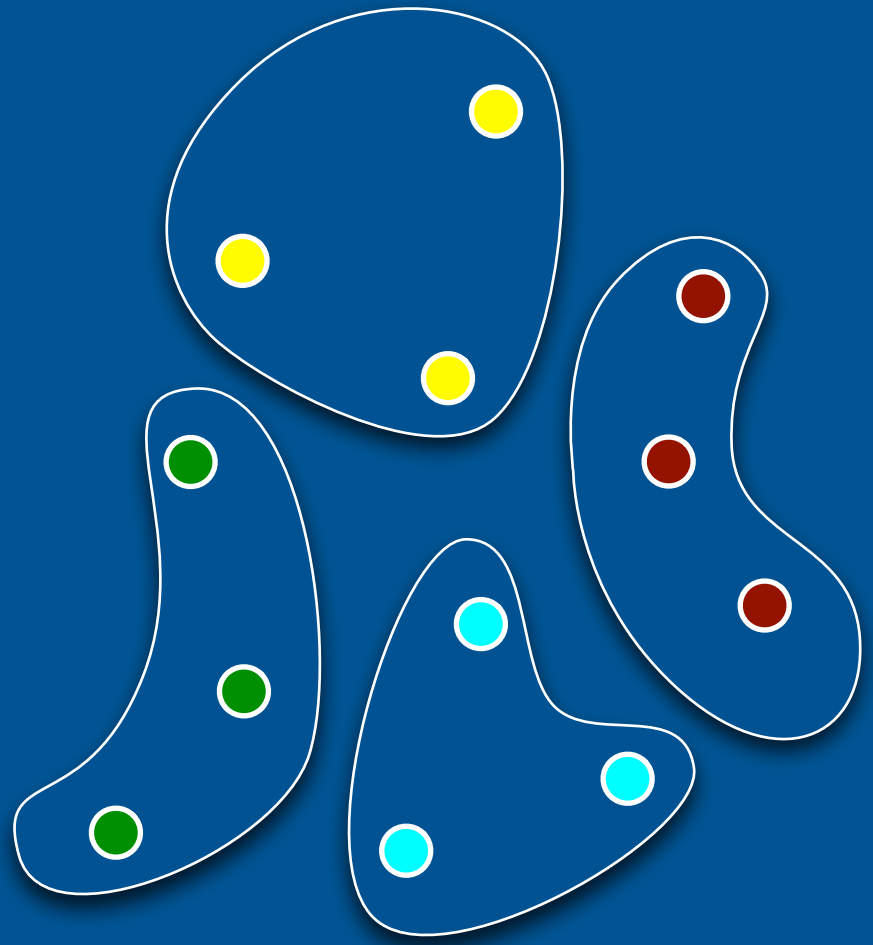


Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

- Labeling – Maps clusters to cores; Each core can have multiple clusters

Ordering – Specifies a serial execution order for each core

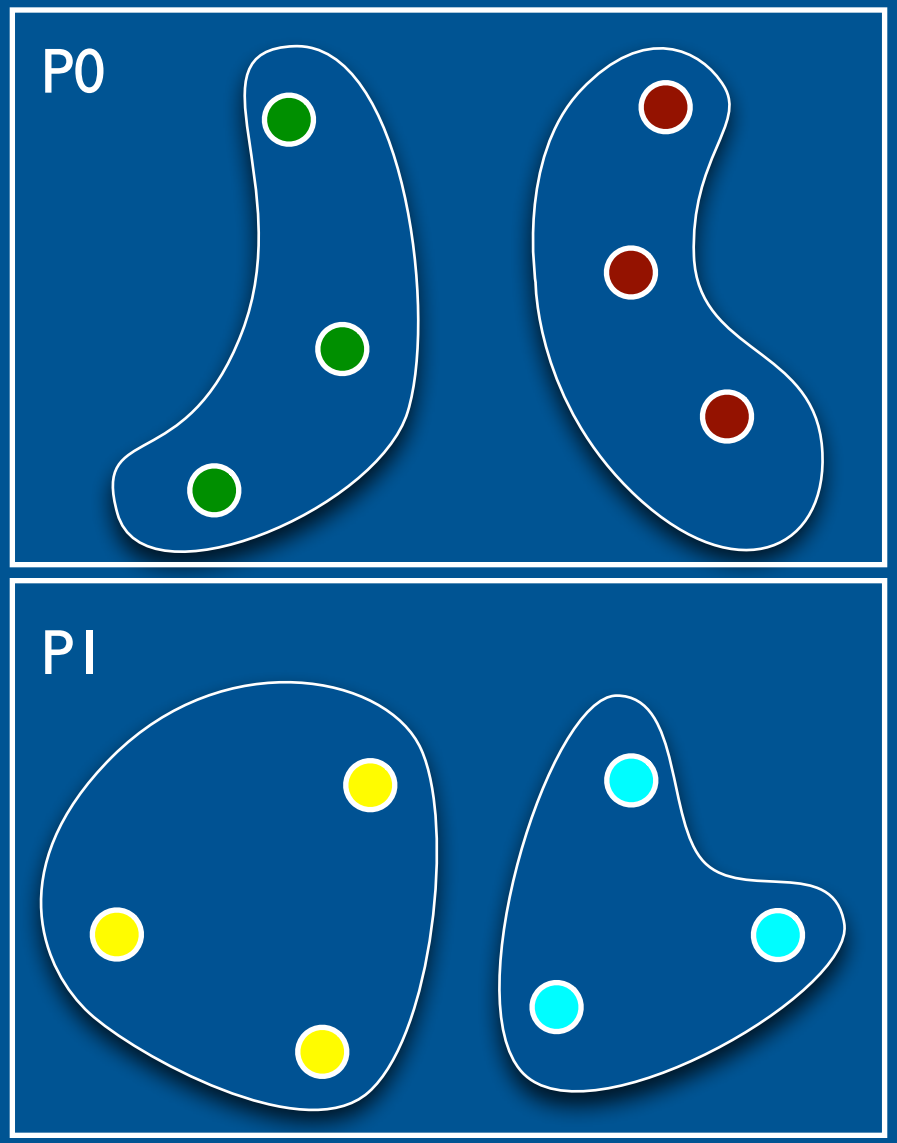


Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

- Labeling – Maps clusters to cores; Each core can have multiple clusters

Ordering – Specifies a serial execution order for each core

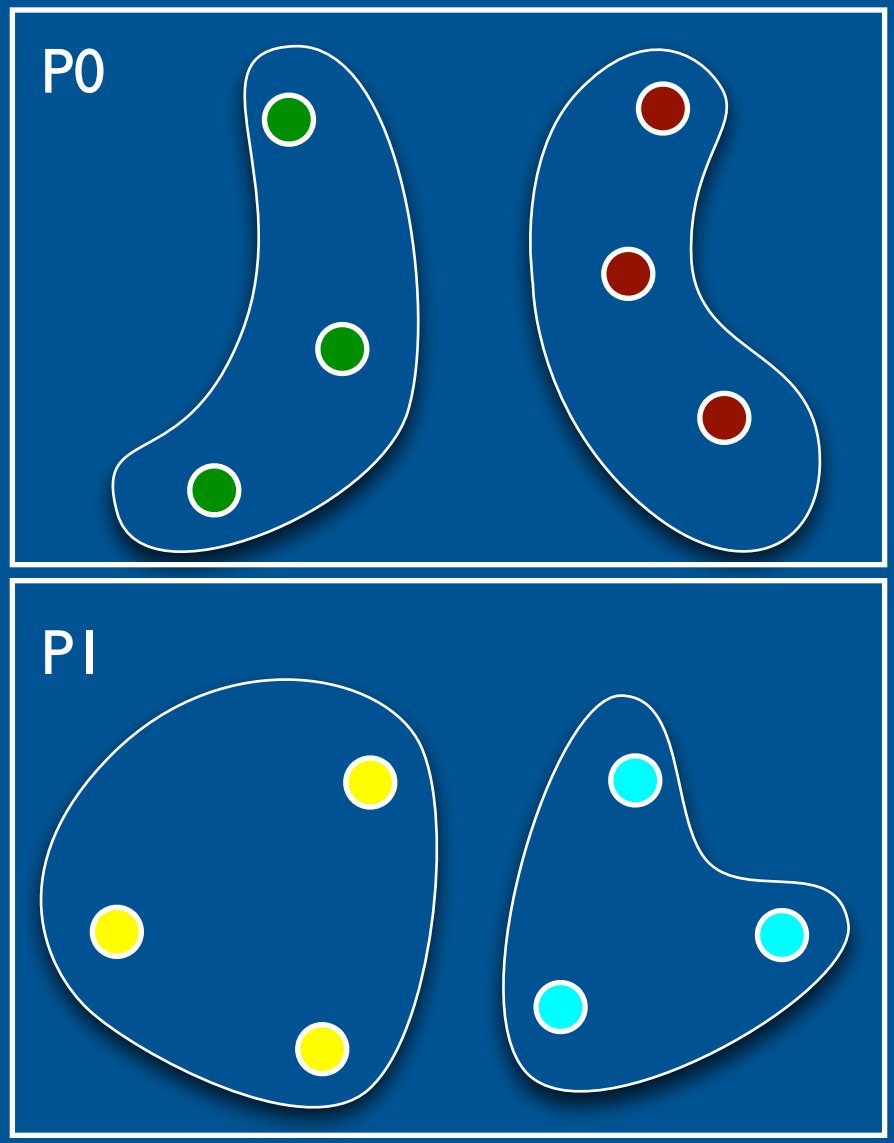


Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

Labeling – Maps clusters to cores; Each core can have multiple clusters

➡ Ordering – Specifies a serial execution order for each core

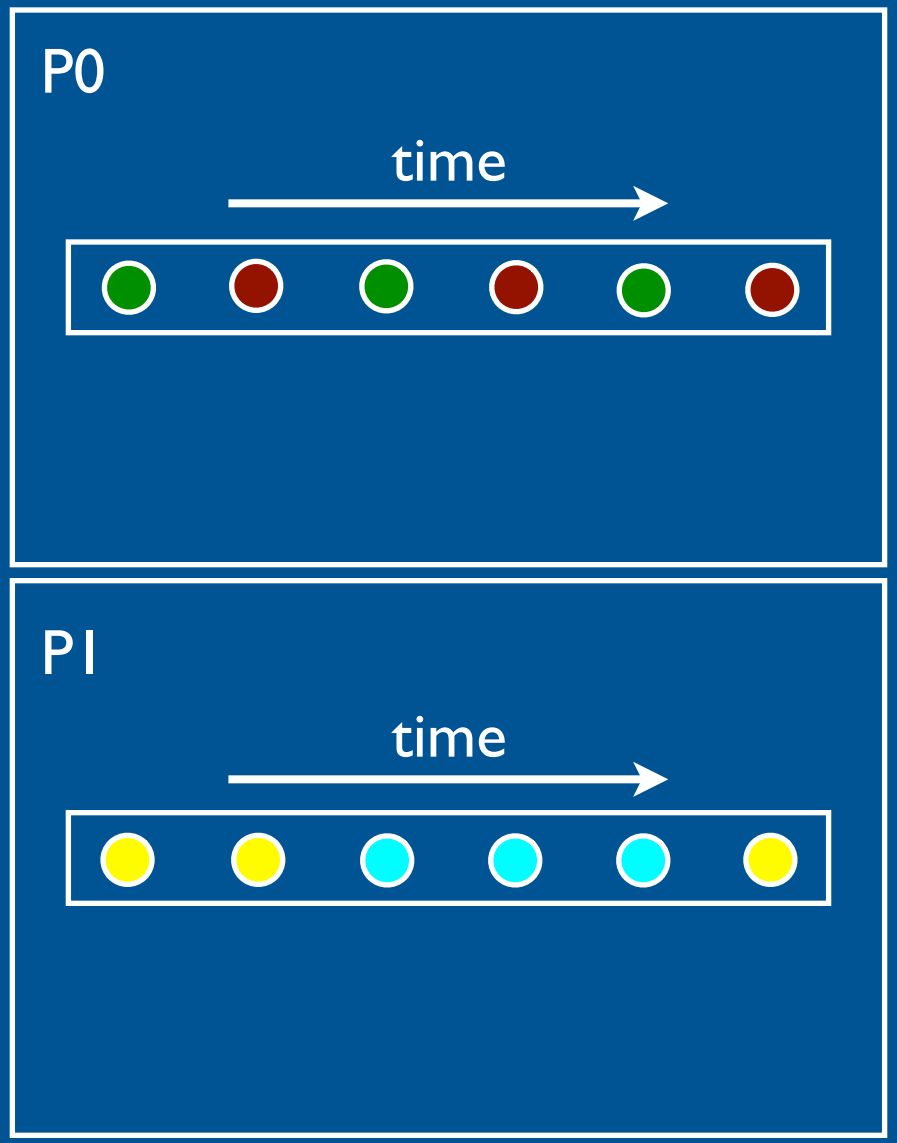


Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

Labeling – Maps clusters to cores; Each core can have multiple clusters

➡ Ordering – Specifies a serial execution order for each core



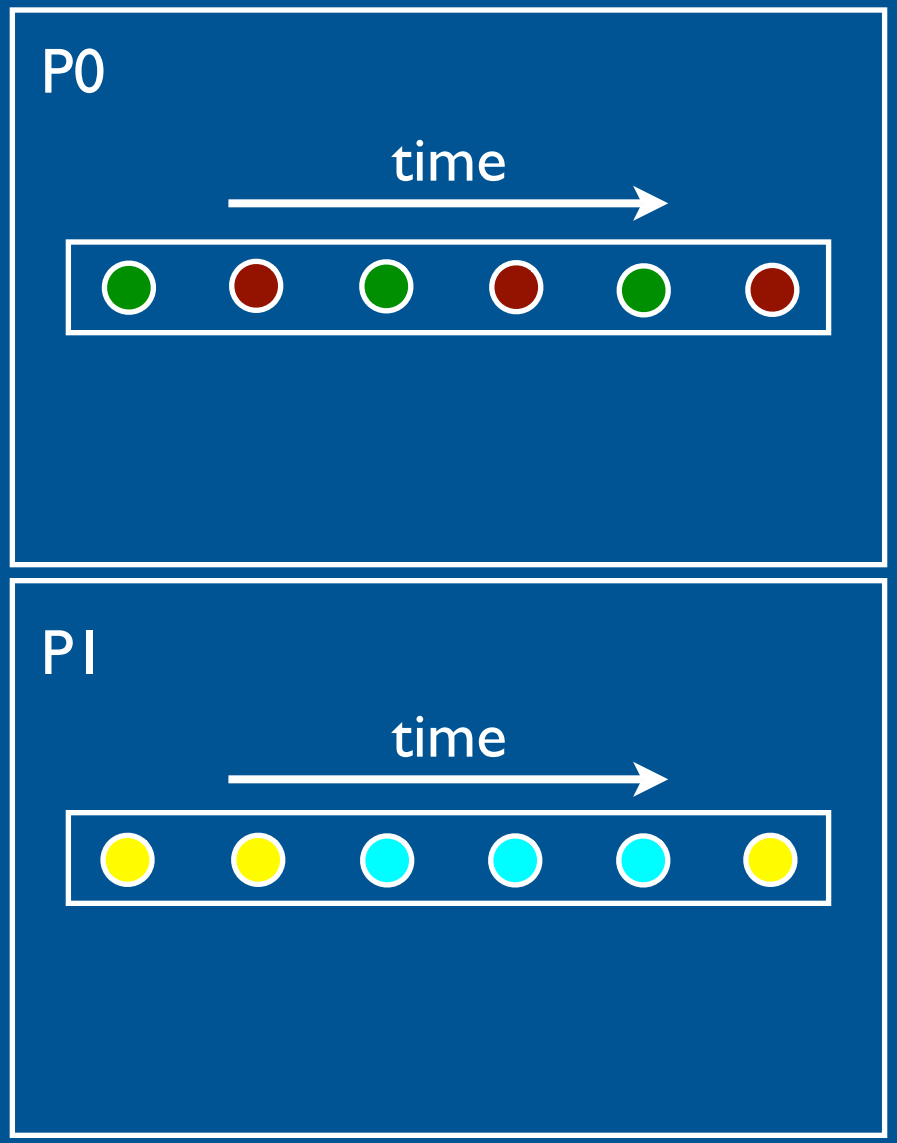
Scheduling Functions

Clustering – Groups iterations into clusters; Each cluster executed on a single core

Labeling – Maps clusters to cores; Each core can have multiple clusters

➡ Ordering – Specifies a serial execution order for each core

Functions can be defined "online"



Example Instantiations

- OpenMP's chunked self-scheduling
  - Clustering: chunked
  - Labeling: dynamic
  - Ordering: cluster-major
- DMR's "generator-computes"
  - Clustering: chunked + generator-computes
  - Labeling: dynamic
  - Ordering: LIFO

The Galois system provides a number of built-in scheduling policies

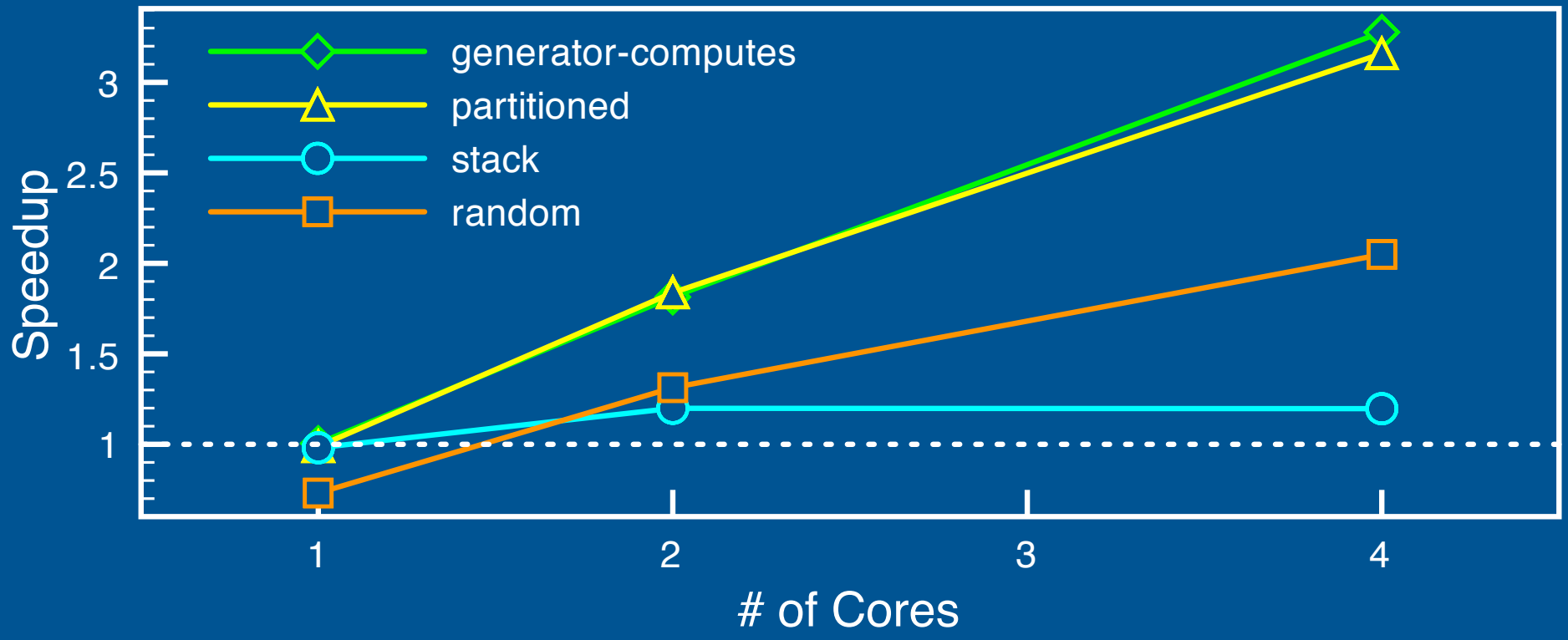
Evaluated Applications

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

Sample Schedules for DMR

- random – default Galois schedule
- stack – LIFO schedule
- partitioned – data-centric schedule, based on partitioning of mesh
- generator-computes – random schedule, new work immediately processed by core that created it

DMR Results


generator-computes
partitioned
stack
random
Speedup
3
2.5
2
1.5
1
1
2
3
4
# of Cores

Summary of Results

- Best combination of policies for each application

	Clustering	Labeling	Ordering
Delaunay Mesh Refinement	random/ inherited	dynamic/ random	—/ LIFO
Delaunay Triangulation	data-centric/ —	static/ data-centric	cluster-major/ random
Augmenting Paths Maxflow	data-centric/ inherited	static/ data-centric	cluster-major/ LIFO
Preflow Push Maxflow	data-centric/ inherited	static/ data-centric	cluster-major/ LIFO
Agglomerative Clustering	unit/ custom	dynamic/ custom	—/ —

Summary of Results

- Best combination of policies for each application

	Clustering	Labeling	Ordering
Delaunay Mesh Refinement	random/ inherited	dynamic/ random	—/ LIFO
Delaunay Triangulation	data-centric/ —	static/ data-centric	cluster-major/ random
Augmenting Paths Maxflow	data-centric/ inherited	static/ data-centric	cluster-major/ LIFO
Preflow Push Maxflow	data-centric/ inherited	static/ data-centric	cluster-major/ LIFO
Agglomerative Clustering	unit/ custom	dynamic/ custom	—/ —

Conclusions

- Developed a general framework for scheduling programs with amorphous data parallelism
  - Subsumes OpenMP scheduling policies
- Implemented framework in Galois system
  - Provides several default scheduling policies
  - Allows programmers to specify their own scheduling policies when needed