How Much Parallelism is There in Irregular Applications?

Milind Kulkarni, Martin Burtscher, Rajasekhar Inkulu and Keshav Pingali

THE UNIVERSITY OF TEXAS AT AUSTIN

Călin Caşcaval

IBM

How Much Parallelism is There in Irregular Applications?

Milind Kulkarni, Martin Burtscher, Rajasekhar Inkulu and Keshav Pingali

THE UNIVERSITY OF TEXAS AT AUSTIN

Călin Caşcaval

IBM

Introduction

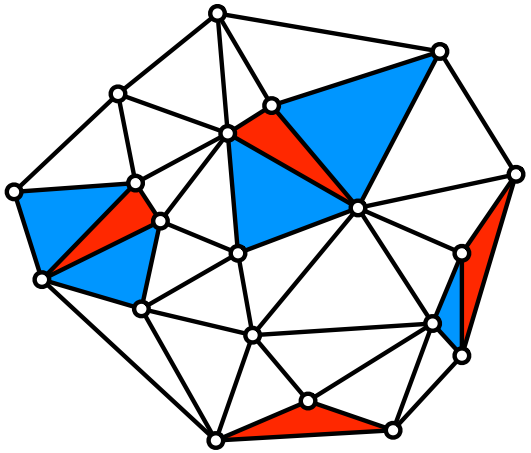
- We understand parallelism in regular algorithms
  - e.g., in N×N matrix-matrix multiply, can do N^3 multiplications concurrently
- What about irregular algorithms?
  - Operate on complex, pointer-based data structures such as graphs, trees, etc.
  - Is there much parallelism?

Example Algorithms

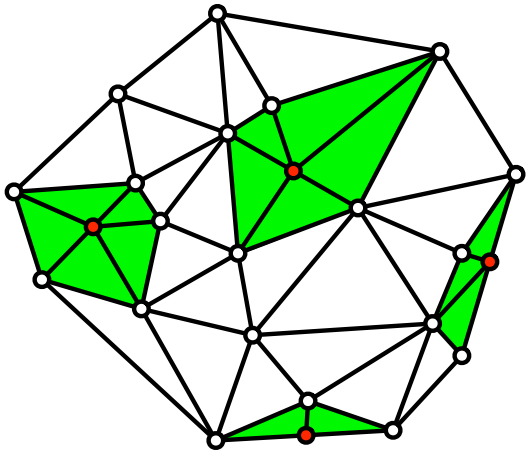
Application Domain	Algorithms
Data-mining	Agglomerative clustering, k-means
Bayesian inference	Belief propagation, survey propagation
Compilers	Iterative dataflow, Elimination-based dataflow
Functional interpreters	Graph reduction, static/dynamic dataflow
Maxflow	Preflow-push, augmenting paths
Minimum spanning trees	Prim's, Kruskal's Boruvka's
N-body methods	Barnes-Hut, fast multipole
Graphics	Ray-tracing
Linear solvers	Sparse MVM, sparse Cholesky factorization
Event-driven simulation	Time warp, Chandy-Misra-Bryant
Meshing	Delaunay mesh refinement, triangulation

Example: Delaunay mesh refinement

- Worklist of bad triangles
- Process bad triangles by removing "cavity" and re-triangulating
  - May create new bad triangles
- Triangles can be processed in any order
- Algorithm terminates when worklist is empty



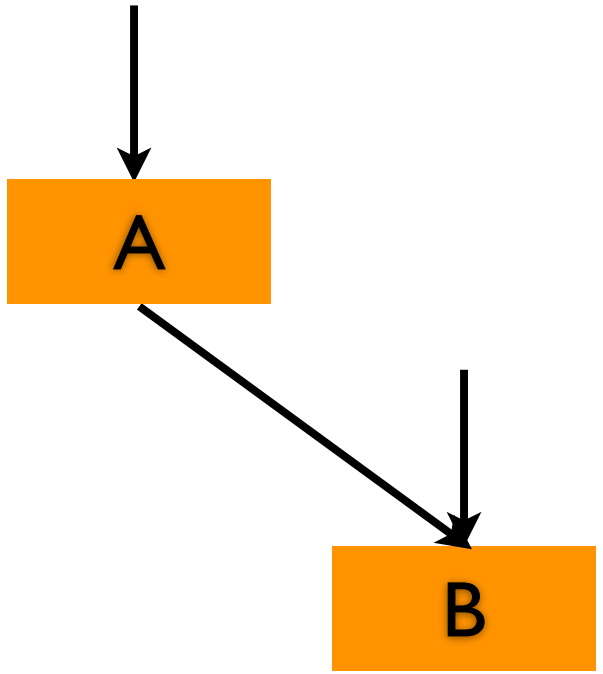
Before



After

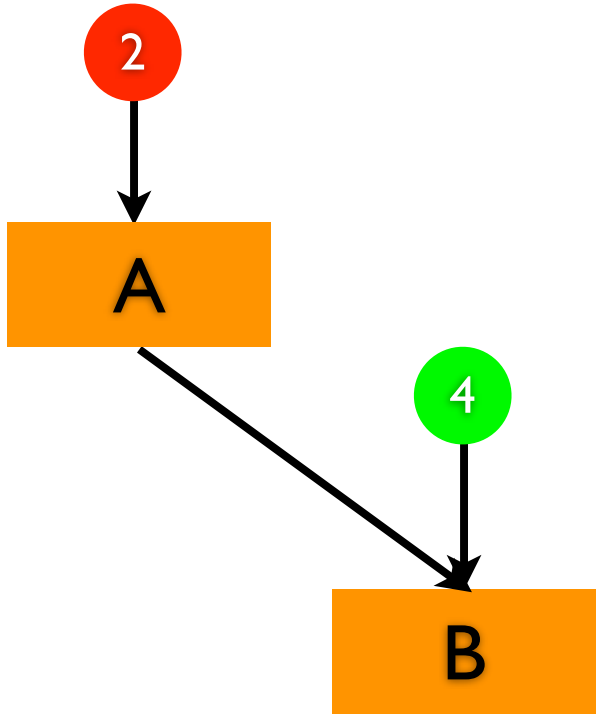
Example: Event-driven Simulation

- Network of nodes
- Worklist of events, ordered by timestamp
- Nodes process events, can generate new events to send to other nodes
- Events must be processed in global time order



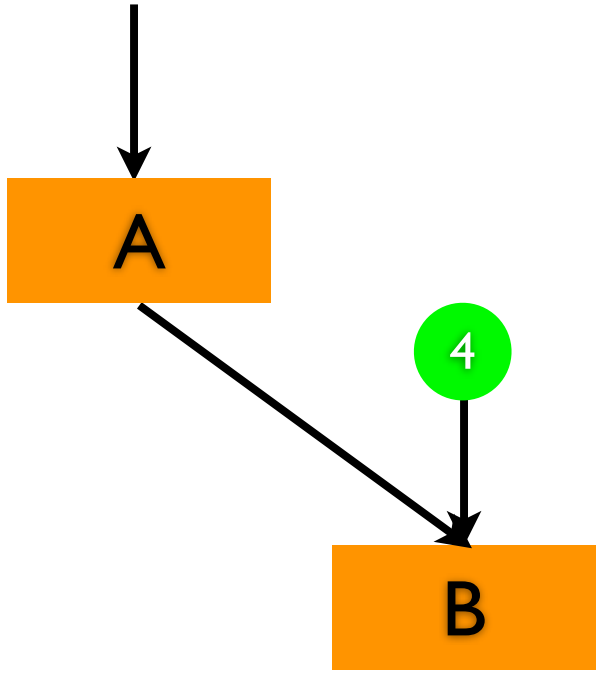
Example: Event-driven Simulation

- Network of nodes
- Worklist of events, ordered by timestamp
- Nodes process events, can generate new events to send to other nodes
- Events must be processed in global time order



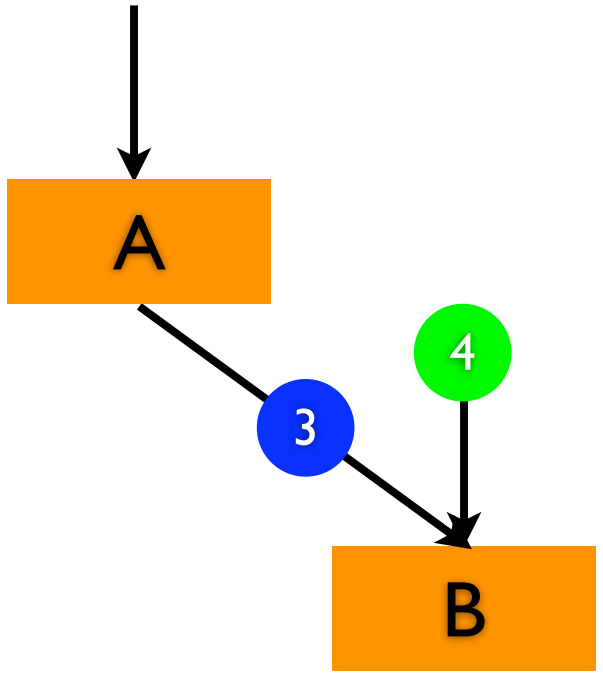
Example: Event-driven Simulation

- Network of nodes
- Worklist of events, ordered by timestamp
- Nodes process events, can generate new events to send to other nodes
- Events must be processed in global time order



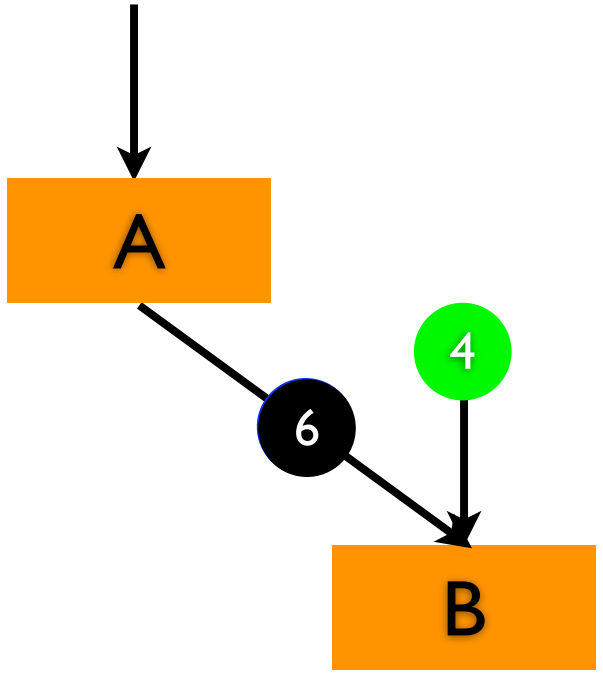
Example: Event-driven Simulation

- Network of nodes
- Worklist of events, ordered by timestamp
- Nodes process events, can generate new events to send to other nodes
- Events must be processed in global time order

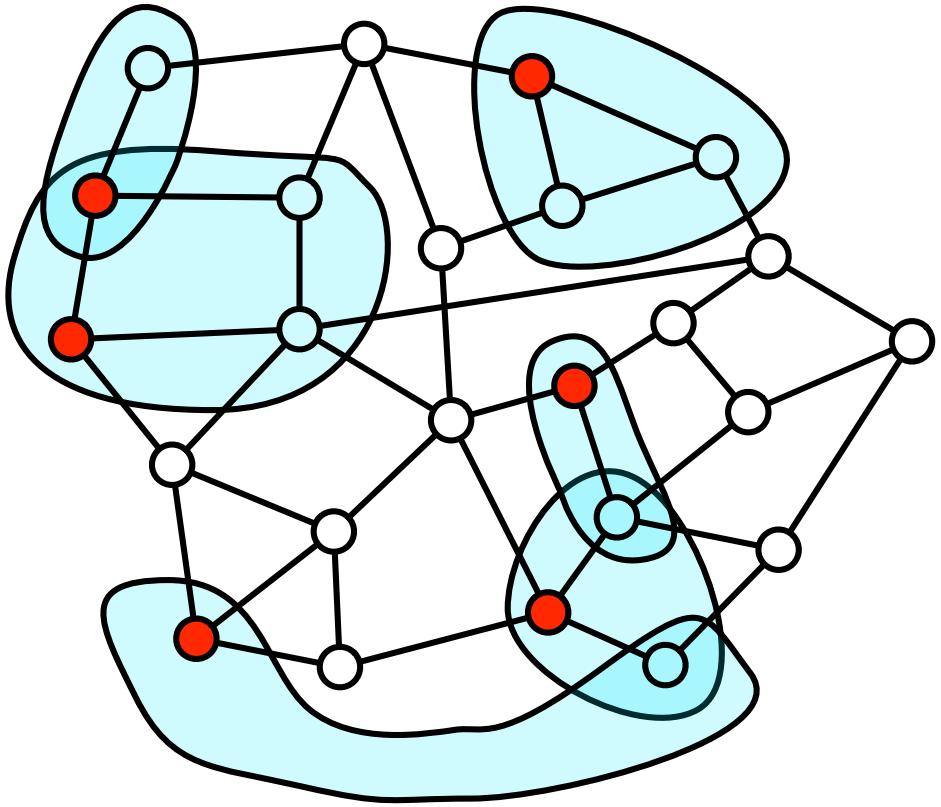


Example: Event-driven Simulation

- Network of nodes
- Worklist of events, ordered by timestamp
- Nodes process events, can generate new events to send to other nodes
- Events must be processed in global time order



Amorphous Data Parallelism



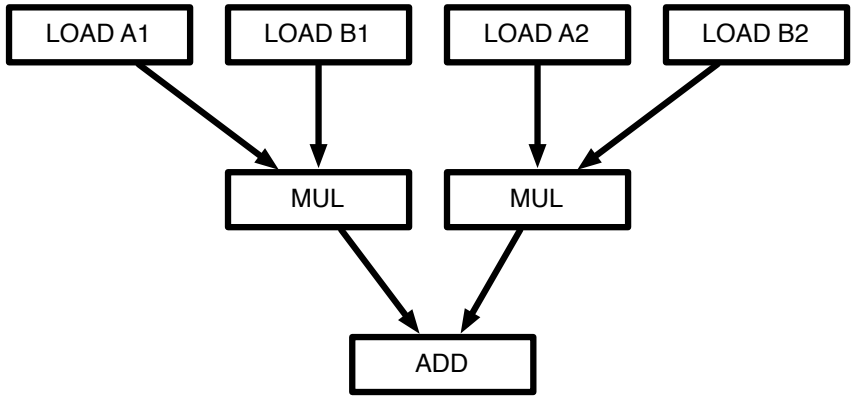
- Data structure: graph
- Operate over ordered or unordered worklists of *active nodes*
- Process an active node by accessing *neighborhood*
  - May generate new active nodes
- Can process nodes with non-overlapping neighborhoods in parallel
- Ordered worklists: must respect ordering constraints

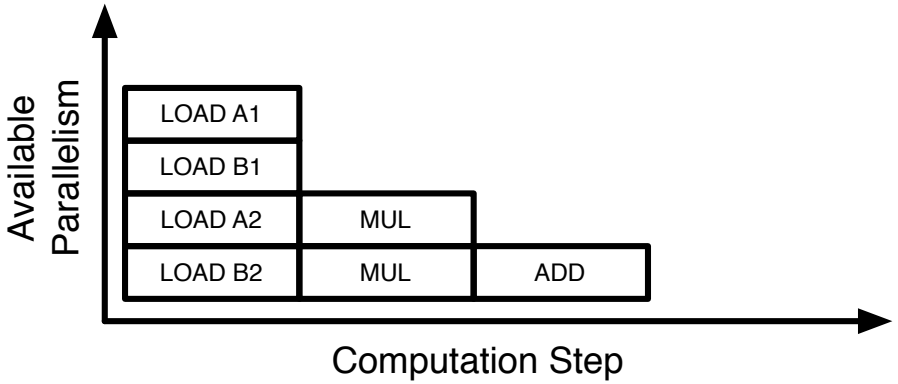
“Available Parallelism”

- A measure of the maximum amount of parallelism that can be extracted from a program
- Profile the algorithm, not the system
  - Disregard communication/synchronization costs, run-time overheads and locality concerns

Measuring Parallelism

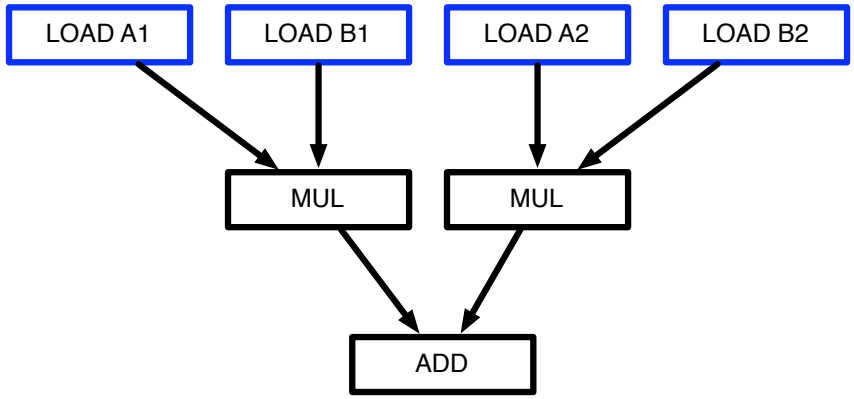
- Represent program as a DAG
  - Nodes: operations
  - Edges: dependences
- Execution strategy
  - Assume operations take unit time
  - Execute "greedily" – process all ready operations in each step
- Parallelism profile: # of operations executed in each step

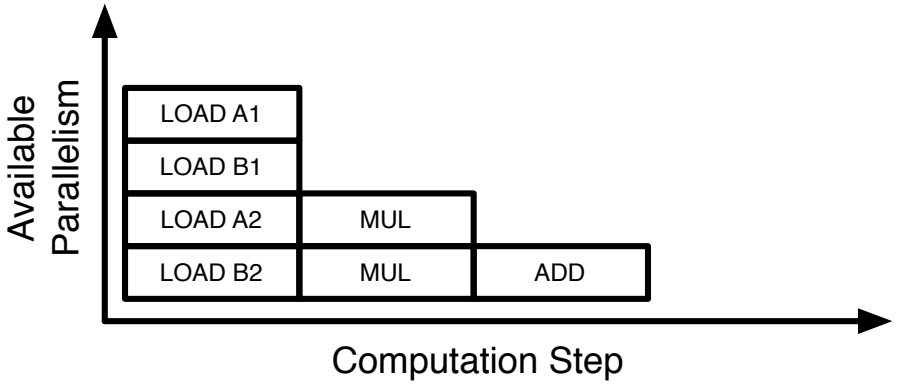




Measuring Parallelism

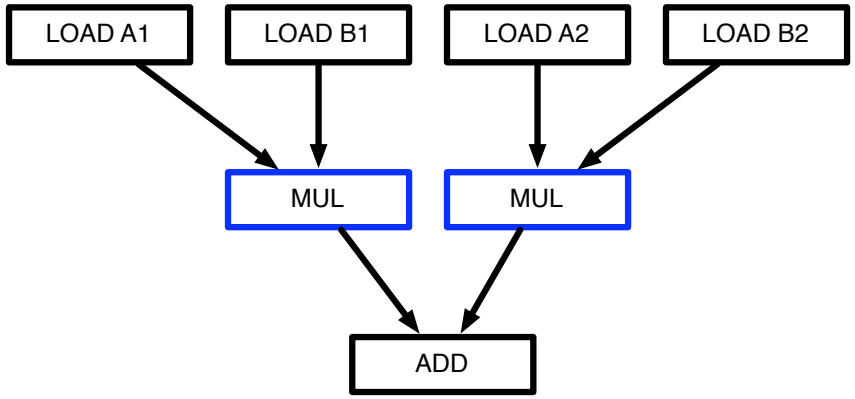
- Represent program as a DAG
  - Nodes: operations
  - Edges: dependences
- Execution strategy
  - Assume operations take unit time
  - Execute "greedily" – process all ready operations in each step
- Parallelism profile: # of operations executed in each step

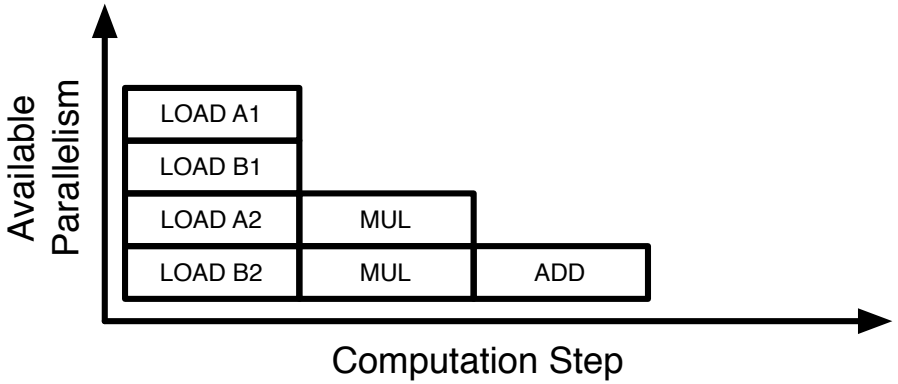




Measuring Parallelism

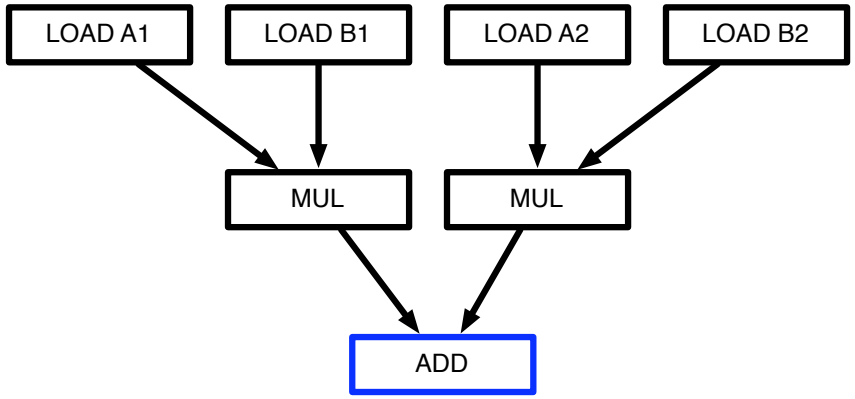
- Represent program as a DAG
  - Nodes: operations
  - Edges: dependences
- Execution strategy
  - Assume operations take unit time
  - Execute "greedily" – process all ready operations in each step
- Parallelism profile: # of operations executed in each step

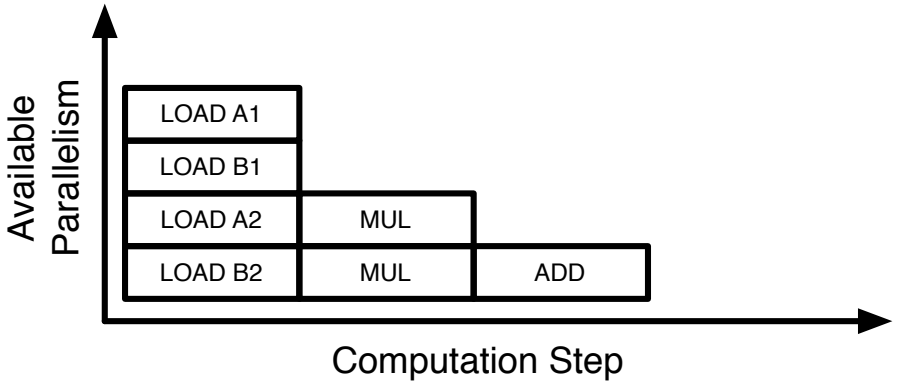




Measuring Parallelism

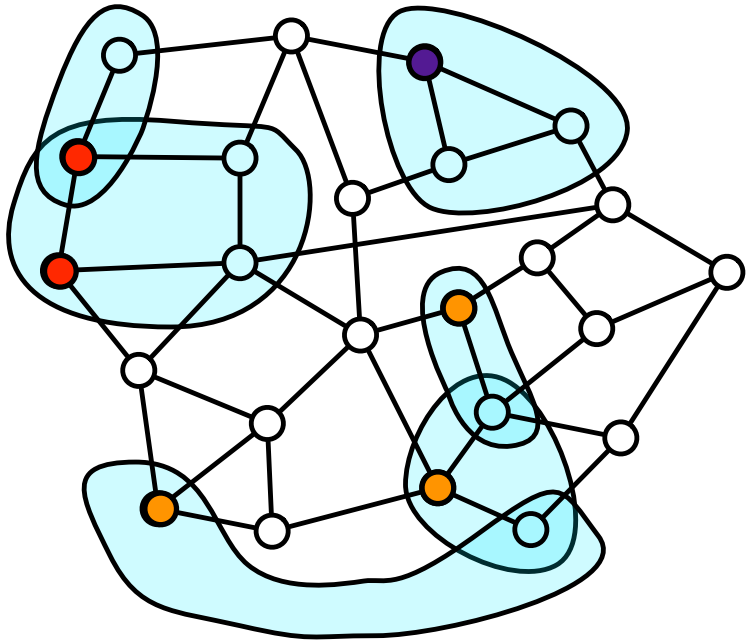
- Represent program as a DAG
  - Nodes: operations
  - Edges: dependences
- Execution strategy
  - Assume operations take unit time
  - Execute "greedily" – process all ready operations in each step
- Parallelism profile: # of operations executed in each step





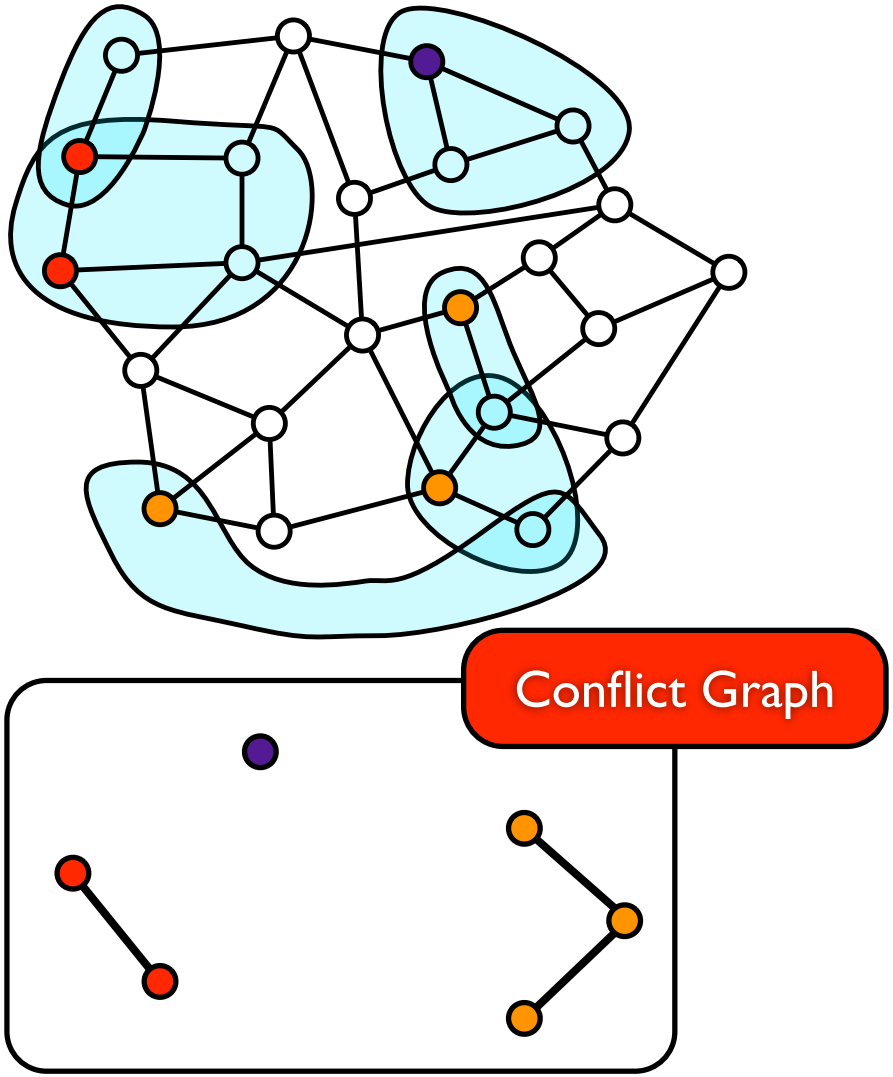
Amorphous Data Parallel Algorithms

- No notion of ordering
  - Represent program as a graph, not a DAG
- Execution: choose set of independent elements to process
- Different scheduling choices lead to different amounts of parallelism
  - Even with unlimited resources!



Amorphous Data Parallel Algorithms

- No notion of ordering
  - Represent program as a graph, not a DAG
- Execution: choose set of independent elements to process
- Different scheduling choices lead to different amounts of parallelism
  - Even with unlimited resources!



Greedy scheduling

- Finding schedule to maximize parallelism is *NP*-hard

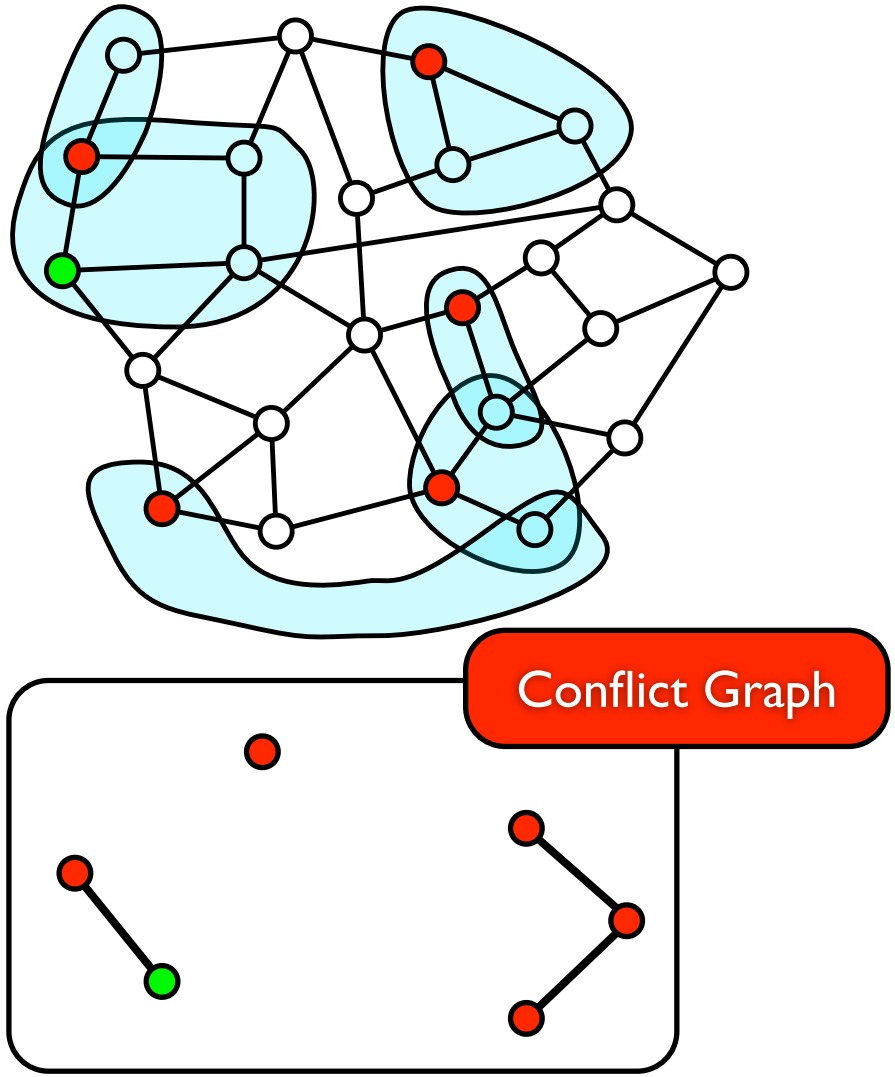
➡ Solution: Schedule greedily

  - Attempt to maximize work done in current step
  - Choose a maximal independent set in conflict graph

Incremental Execution

- Conflict graph can change during execution
  - New work generated
  - New conflicts
- Cannot perform scheduling *a priori*

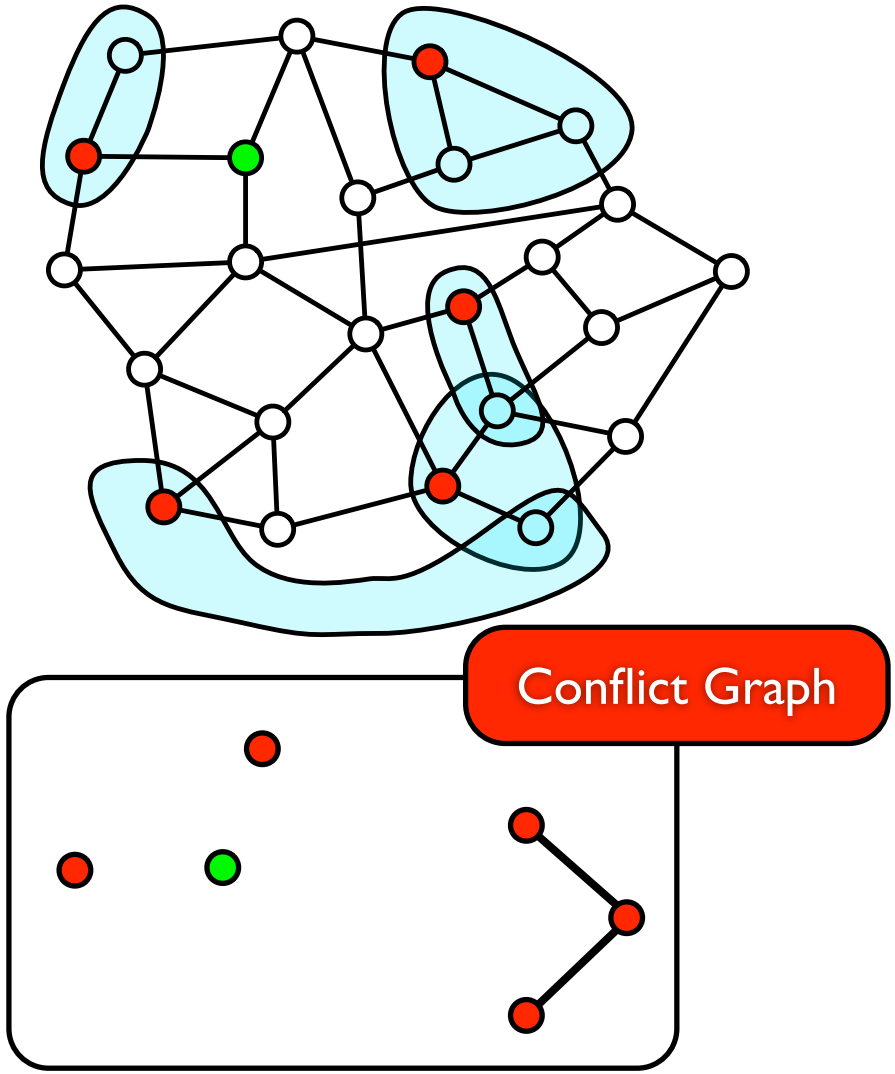
➡ Solution: execute in stages, recalculate conflict graph after each stage



Incremental Execution

- Conflict graph can change during execution
  - New work generated
  - New conflicts
- Cannot perform scheduling *a priori*

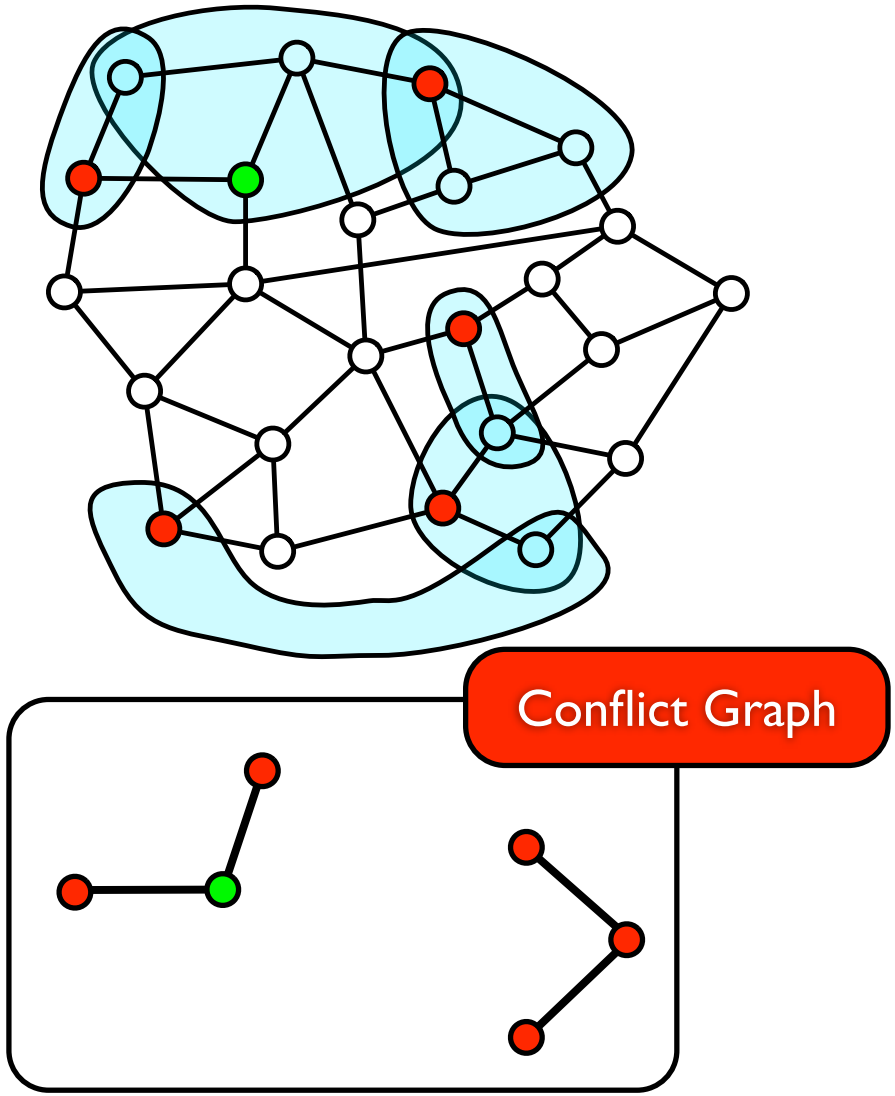
➡ Solution: execute in stages, recalculate conflict graph after each stage



Incremental Execution

- Conflict graph can change during execution
  - New work generated
  - New conflicts
- Cannot perform scheduling *a priori*

➡ Solution: execute in stages, recalculate conflict graph after each stage



ParaMeter

- Tool to generate parallelism profiles for amorphous data-parallel applications
- Uses *greedy scheduling* and *incremental execution* to handle dynamic nature of computation

ParaMeter Execution Strategy

- While work left
  - Generate conflict graph for current worklist
  - Execute maximal independent set of nodes in graph
  - Add newly generated work to worklist

ParaMeter Execution Strategy

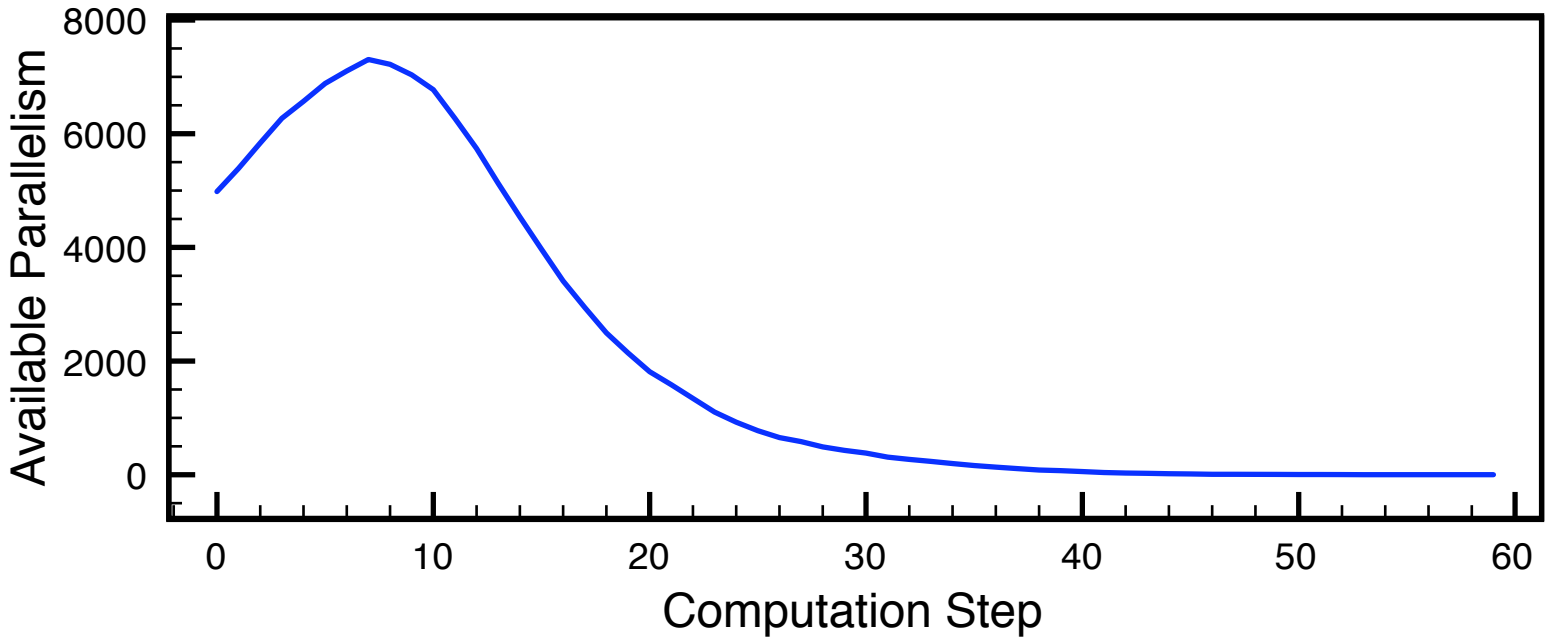
- While work left
  - Gener
    workl
  - Execute maximal independent set of nodes in graph
  - Add newly generated work to worklist

Generate parallelism profile by tracking # of nodes executed in each step

Experiments

- Profiled 7 applications:
  - Delaunay mesh refinement
  - Delaunay triangulation
  - Augmenting paths maxflow
  - Preflow push maxflow
  - Survey propagation
  - Agglomerative clustering (unordered)
  - Agglomerative clustering (ordered)

Delaunay Mesh Refinement

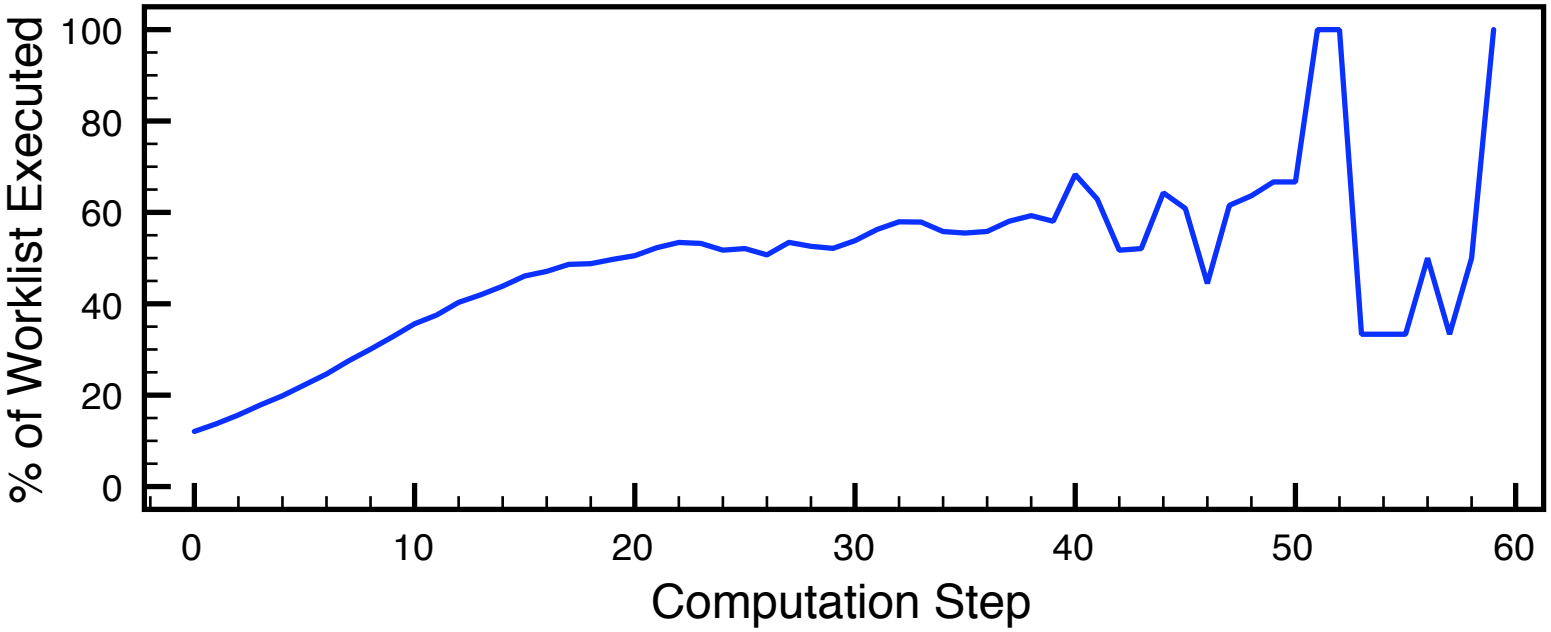


Input: 100,000 triangle mesh, 47,000 bad triangles

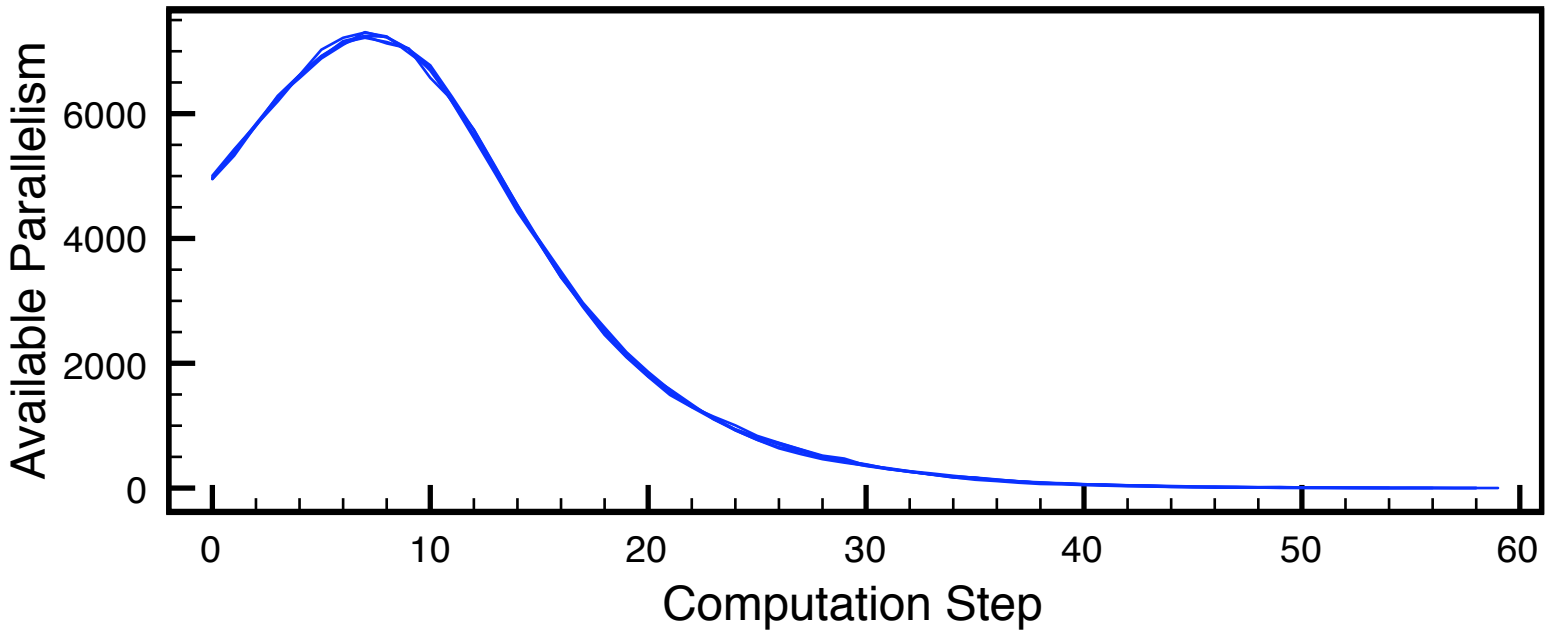
Parallelism Intensity

- Available parallelism shows absolute amount of parallelism in program
  - Is parallelism low because there is little work? Or many conflicts?
- Parallelism intensity: measure what percentage of worklist is executed in parallel

Mesh Refinement: Parallelism Intensity

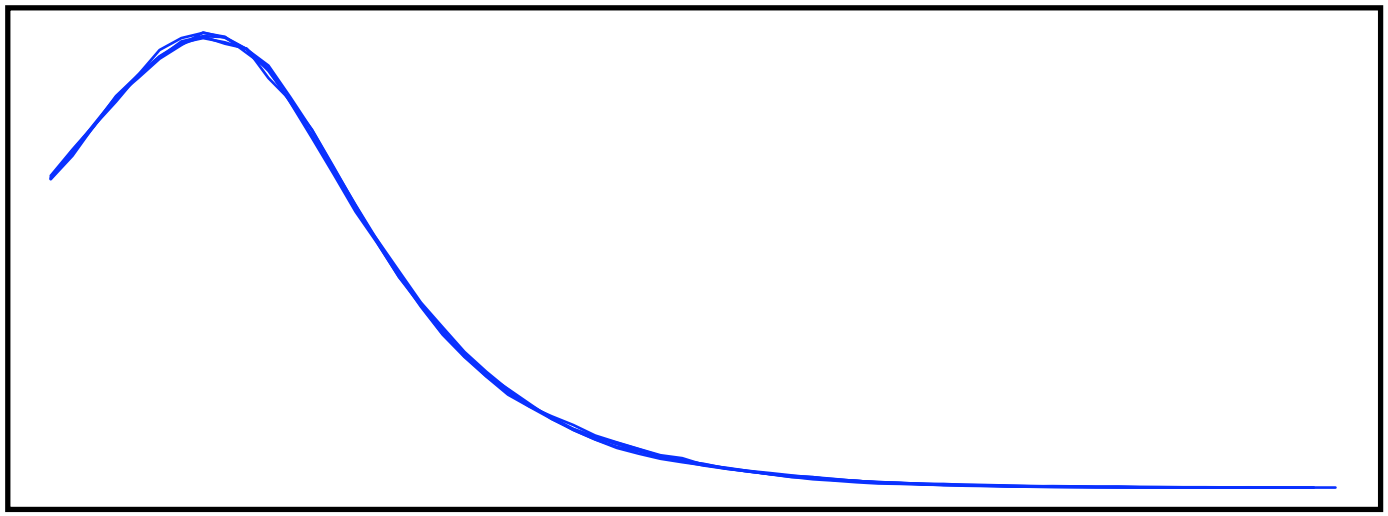


Effects of Scheduling on Parallelism



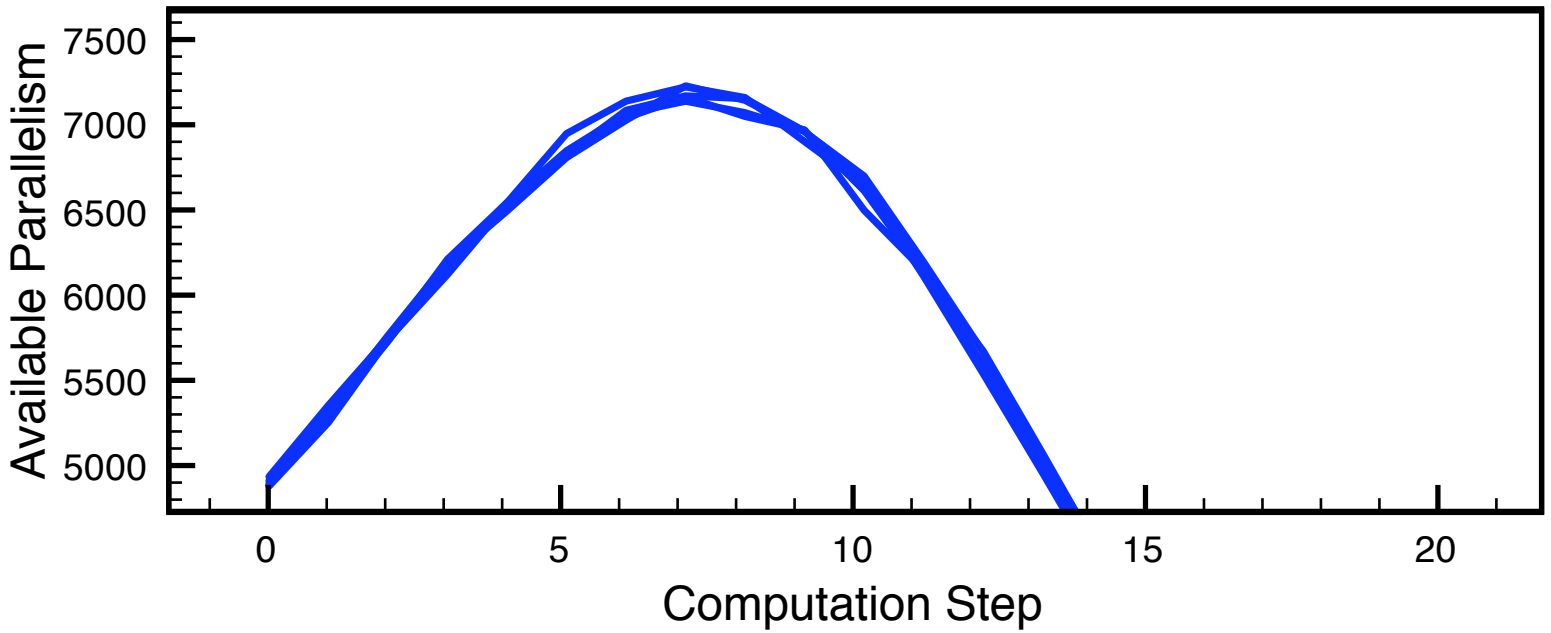
Input: 100,000 triangle mesh, 47,000 bad triangles

Effects of Scheduling on Parallelism



Input: 100,000 triangle mesh, 47,000 bad triangles

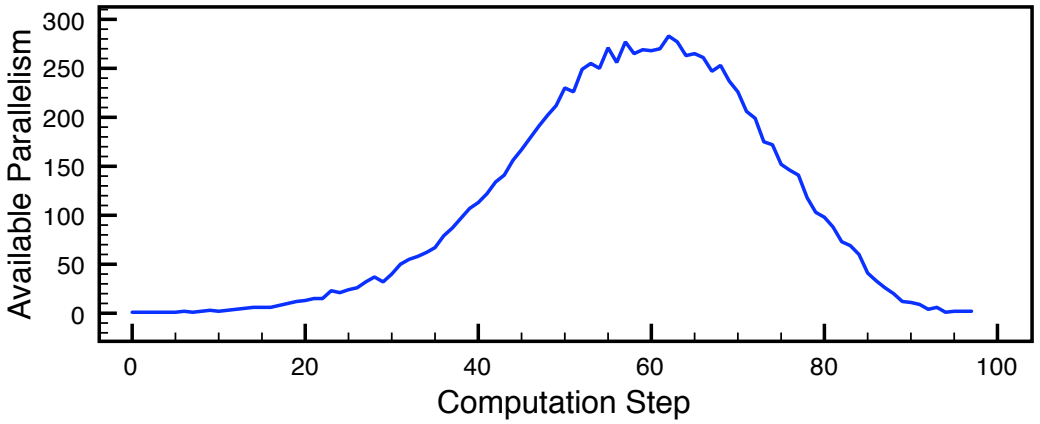
Effects of Scheduling on Parallelism

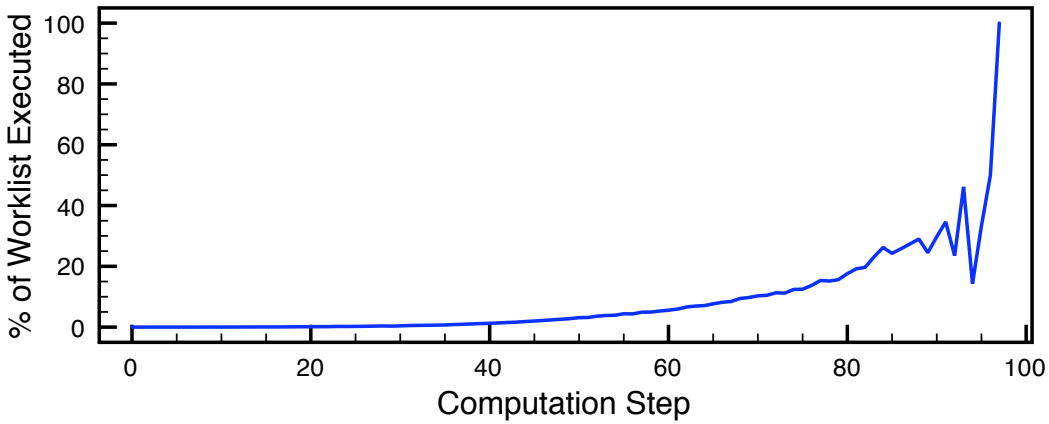


Input: 100,000 triangle mesh, 47,000 bad triangles

Delaunay Triangulation

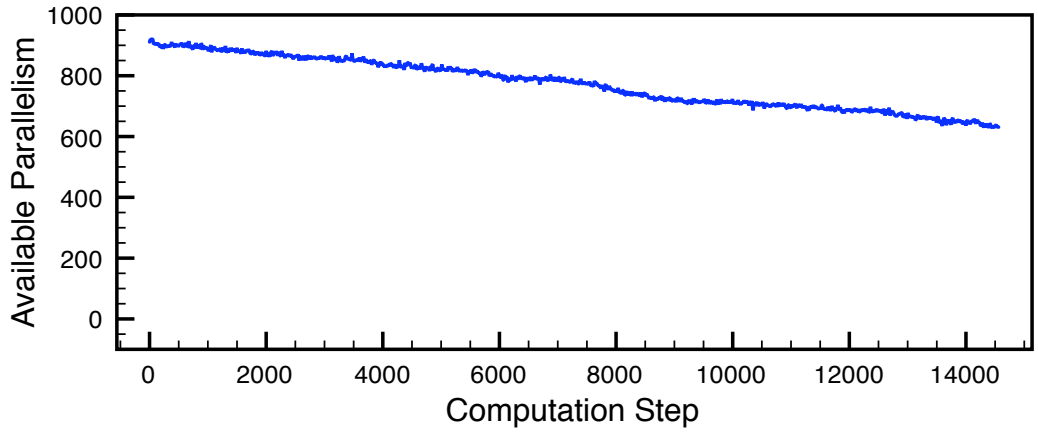
- Build a Delaunay mesh given a set of points
- Points in an unordered worklist
- Insert points by splitting triangles, flipping edges
- Input: 10,000 points

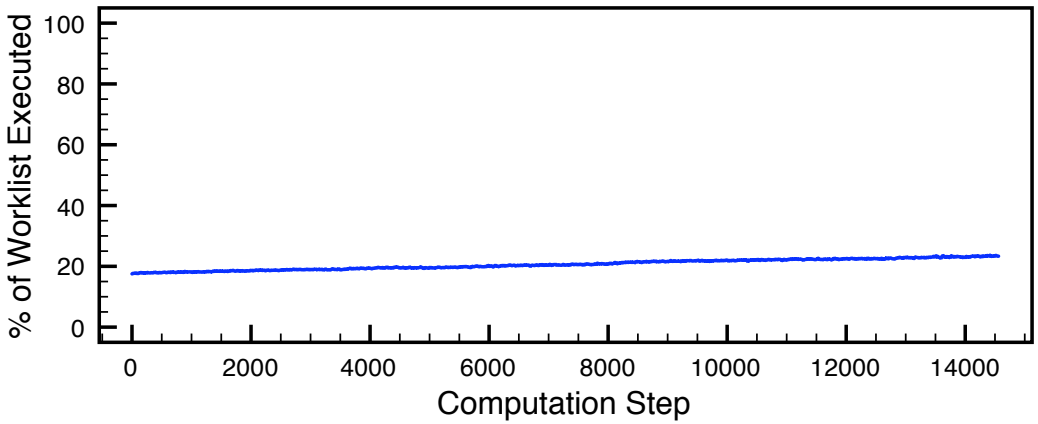




Survey Propagation

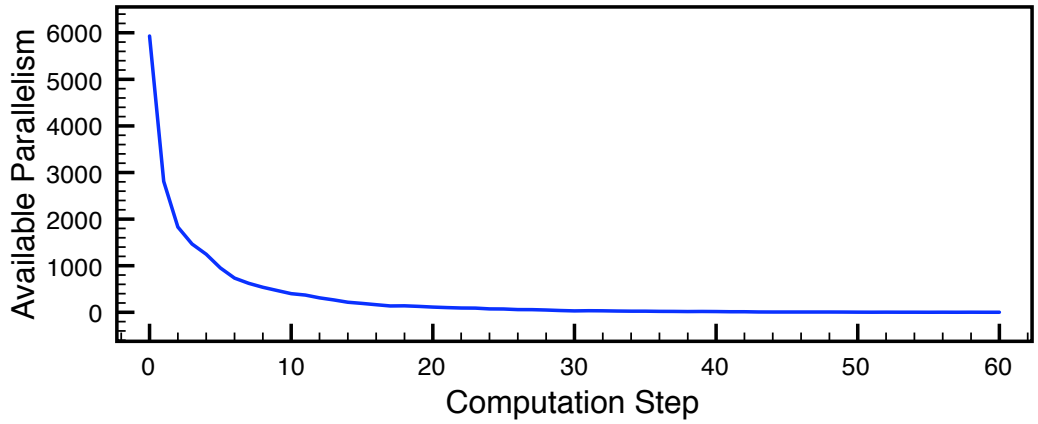
- Heuristic approach to solving SAT problems
- Bipartite graph of clauses and variables
- Iteratively update variables with possible truth values
- Input: formula with 1000 variables, 4200 clauses

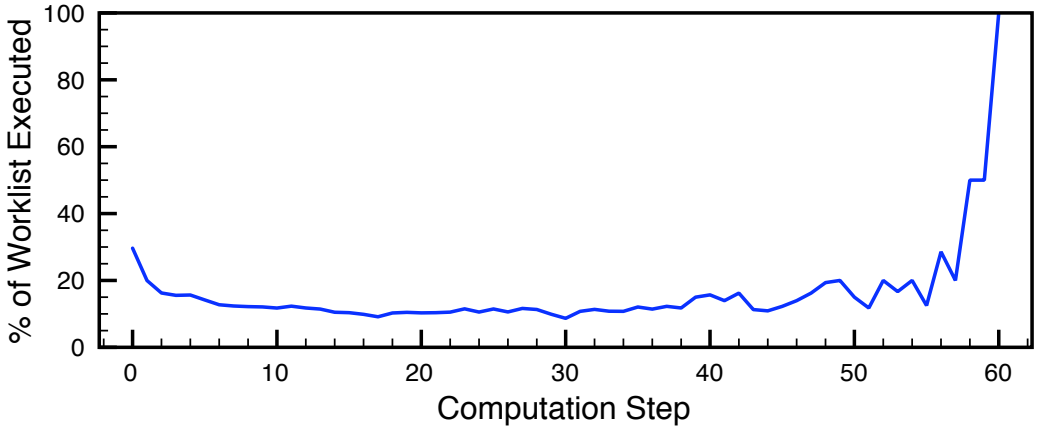




Agglomerative Clustering

- Cluster a set of points based on similarity
- Unordered worklist of point clusters
- Builds a tree bottom-up
- Input: 20,000 points





Conclusions

- ParaMeter: first tool to measure parallelism in irregular algorithms
  - Provides insight into an *algorithm*, rather than a particular parallel *implementation*
  - Also: ordered worklists, constrained parallelism (details in paper)
- ParaMeter shows that important irregular algorithms have significant parallelism
  - Mesh algorithms, SAT solvers, data mining algorithms, maxflow algorithms ...
- Parallelism varies during execution → adaptive control of number of threads may be useful

Thank you!

http://www.ices.utexas.edu/~milind
milind@ices.utexas.edu