The Operator Formulation and Amorphous Data-Parallelism

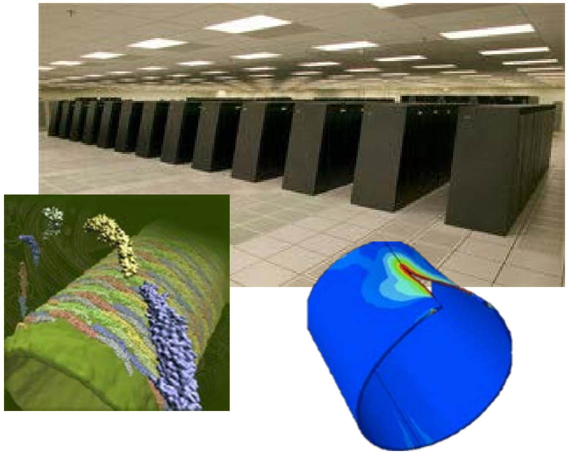
Keshav Pingali

The University of Texas at Austin

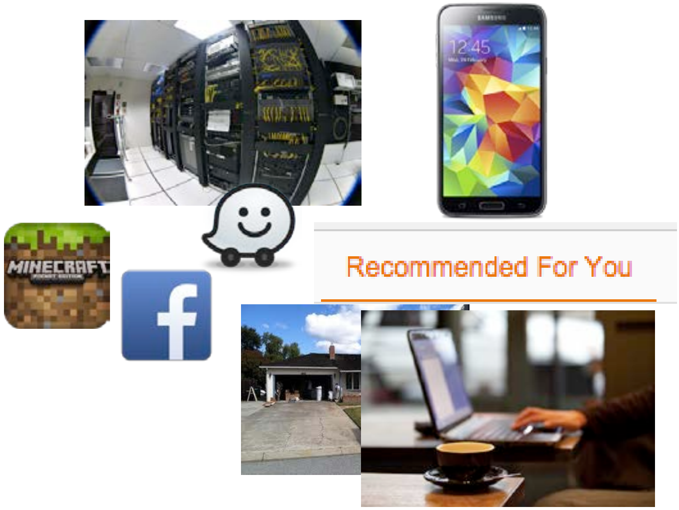
Intelligent Software Systems group (ISS)

- Faculty
  - Keshav Pingali, CS/ECE/ICES
- Research staff
  - Andrew Lenharth
  - Sree Pai
- PhD students
  - Amber Hassaan
  - Rashid Kaleem
  - Michael He
  - Yi-Shan Lu
  - Roshan Dathathri
  - Gurbinder Singh
  - Sepideh Maliki
- Visitors from China, France, India, Norway, Poland, Portugal
- Home page: http://iss.ices.utexas.edu
- Funding: DARPA, NSF, BAE, HP, Intel, NEC, NVIDIA…

Parallel computing is changing



Old World

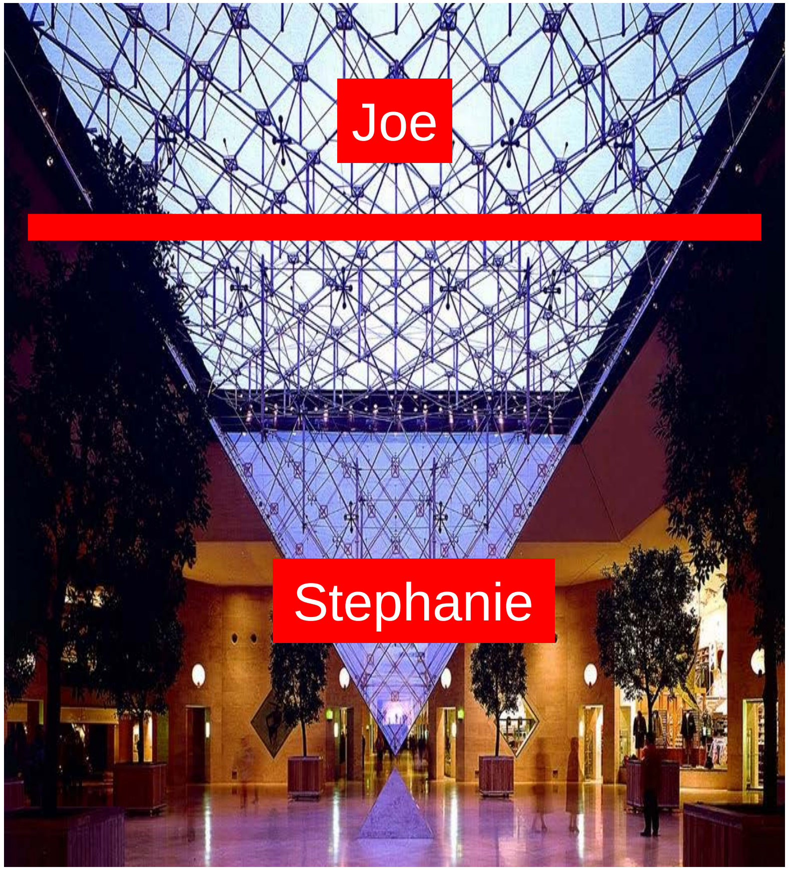


New World

- Platforms
  - Dedicated clusters versus cloud, mobile
- People
  - Small number of scientists and engineers versus large number of self-trained parallel programmers
- Data
  - Structured (vector, matrix) versus unstructured (graphs)

The Search for “Scalable” Parallel Programming Models

- Tension between productivity and performance
  - support large number of application programmers with small number of expert parallel programmers
  - performance comparable to hand-optimized codes
- Galois project
  - data-centric abstractions for parallelism and locality
    - operator formulation
  - scalable parallel system



What we have learned

- Abstractions for parallelism
  - Yesterday: computation-centric abstractions
    - Loops or procedure calls that can be executed in parallel
  - Today: data-centric abstractions
    - Operator formulation of algorithms
- Parallelization strategies
  - Yesterday: static parallelization is the norm
    - Inspector-executor, optimistic parallelization etc. needed only when you lack information about algorithm or data structure
  - Today: optimistic parallelization is the baseline
    - Inspector-executor, static parallelization etc. are possible only when algorithm has enough structure
- Applications
  - Yesterday: programs are monoliths, whole-program analysis is essential
  - Today: programs must be layered. Data abstraction is essential not just for software engineering but for parallelism.

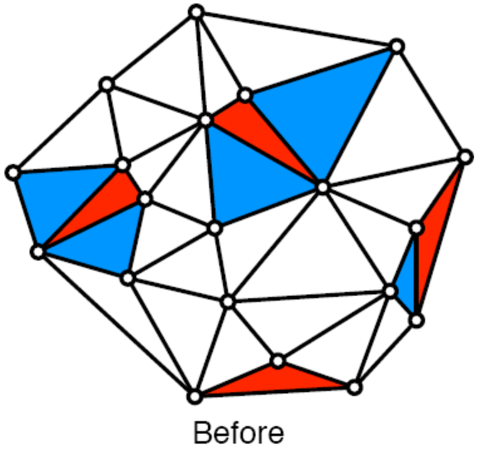
Parallelism: Yesterday

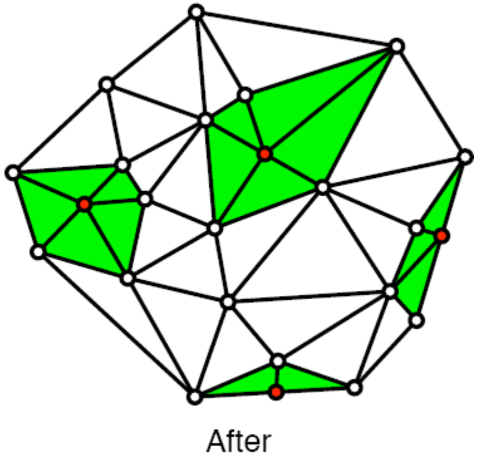
```
Mesh m = /* read in mesh */
WorkList wl;
wl.add(m.badTriangles());
while (true) {
   if (wl.empty()) break;
   Element e = wl.get();
   if (e no longer in mesh)
      continue;
   Cavity c = new            Cavity();
   c.expand();
   c.retriangulate();
   m.update(c);//update mesh
   wl.add(c.badTriangles());
}
```

- What does program do?
  - It does not matter.
- Where is parallelism in program?
  - Loop: do static analysis to find dependence graph
- Static analysis fails to find parallelism.
  - May be there is no parallelism in program?
- Thread-level speculation
  - Misspeculation and overheads limit performance
  - Misspeculation costs power and energy

Computation-centric view of parallelism

Parallelism: Today





Delaunay mesh refinement
Red Triangle: badly shaped triangle
Blue triangles: cavity of bad triangle

- Parallelism:
  - Bad triangles whose cavities do not overlap can be processed in parallel
  - Parallelism must be found at runtime
- Data-centric view of algorithm
  - Active elements: bad triangles
  - Local view: operator applied to bad triangle:

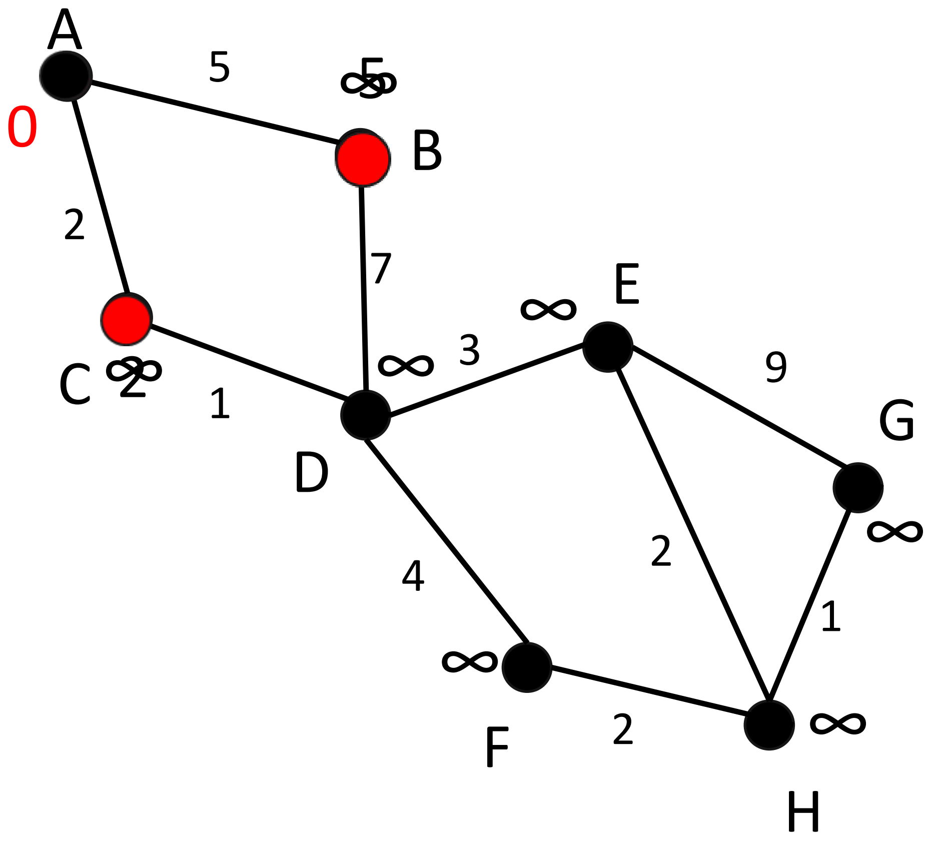
    {Find cavity of bad triangle (blue);
    Remove triangles in cavity;
    Retriangulate cavity and update mesh;}
  - Global view: schedule
  - Algorithm = Operator + Schedule
- Parallel data structures
  - Graph
  - Worklist of bad triangles

Example: Graph analytics

- Single-source shortest-path problem
- Many algorithms
  - Dijkstra (1959)
  - Bellman-Ford (1957)
  - Chaotic relaxation (1969)
  - Delta-stepping (1998)
- Common structure:
  - Each node has distance label d
  - Operator:

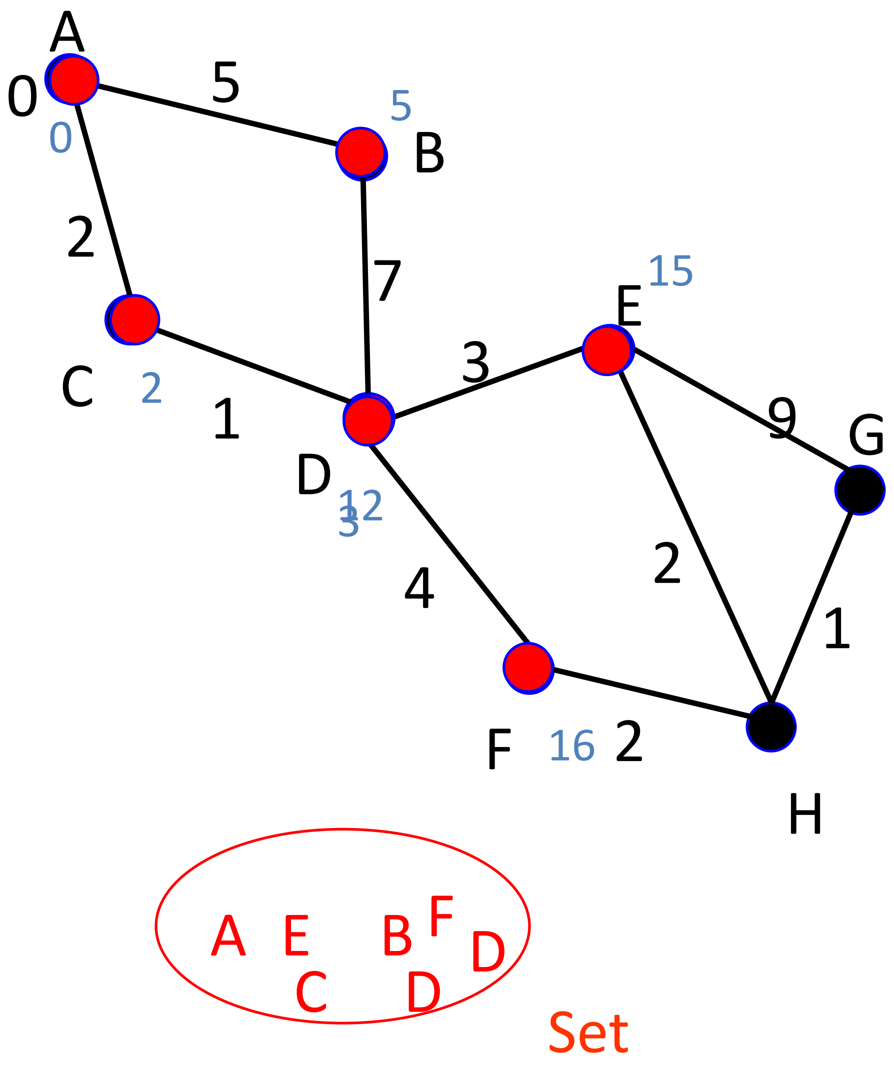
```
relax-edge(u,v):
        if d[v] > d[u]+length(u,v)
        then d[v] ← d[u]+length(u,v)
```

  - Active node: unprocessed node whose distance field has been lowered
  - Different algorithms use different schedules
  - Schedules differ in parallelism, locality, work efficiency



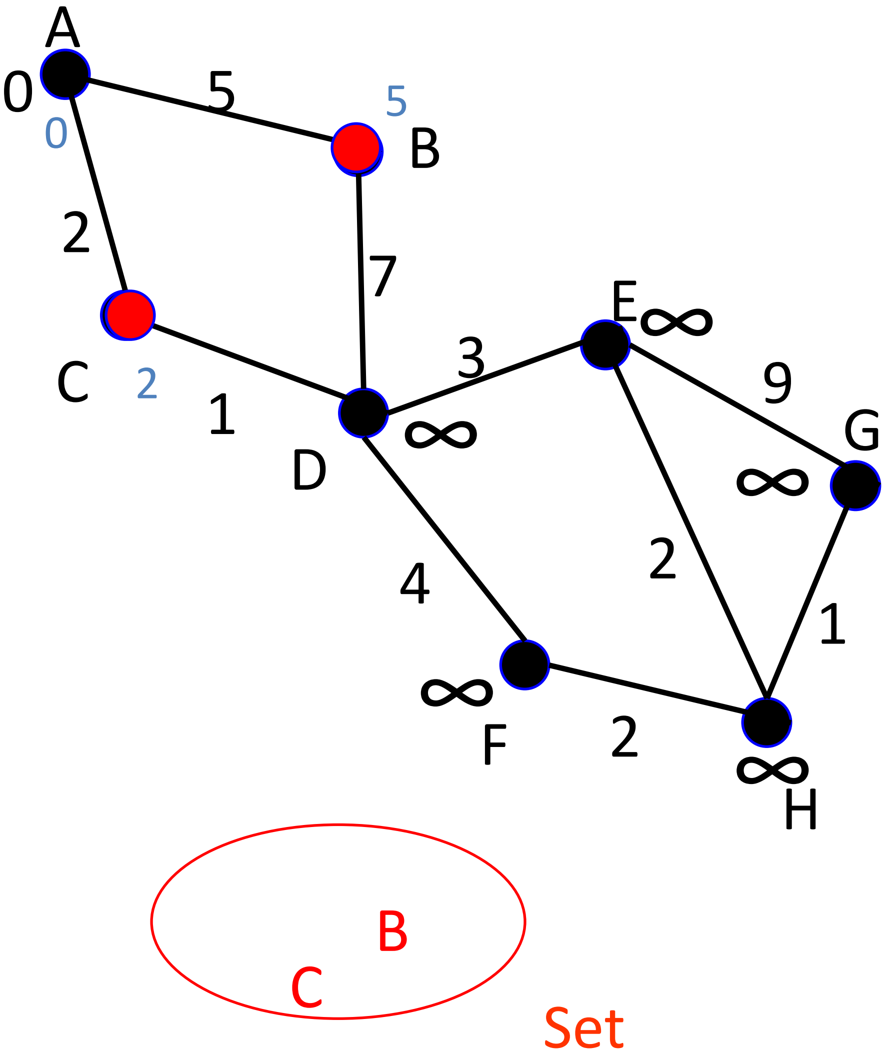
SSSP algorithms (I)

- Chaotic relaxation (1969):
  - use set to track active nodes
  - iterate over nodes in any order
  - nodes can be relaxed many times
    - may do more work than Dijkstra
- Key data structures:
  - Graph
  - Work set/multiset: unordered
- Parallelization:
  - process multiple work-set nodes
  - need concurrent data structures
    - concurrent set/multiset: elements are added/removed correctly
    - concurrent graph: simultaneous updates to node happen correctly
- Unordered algorithm



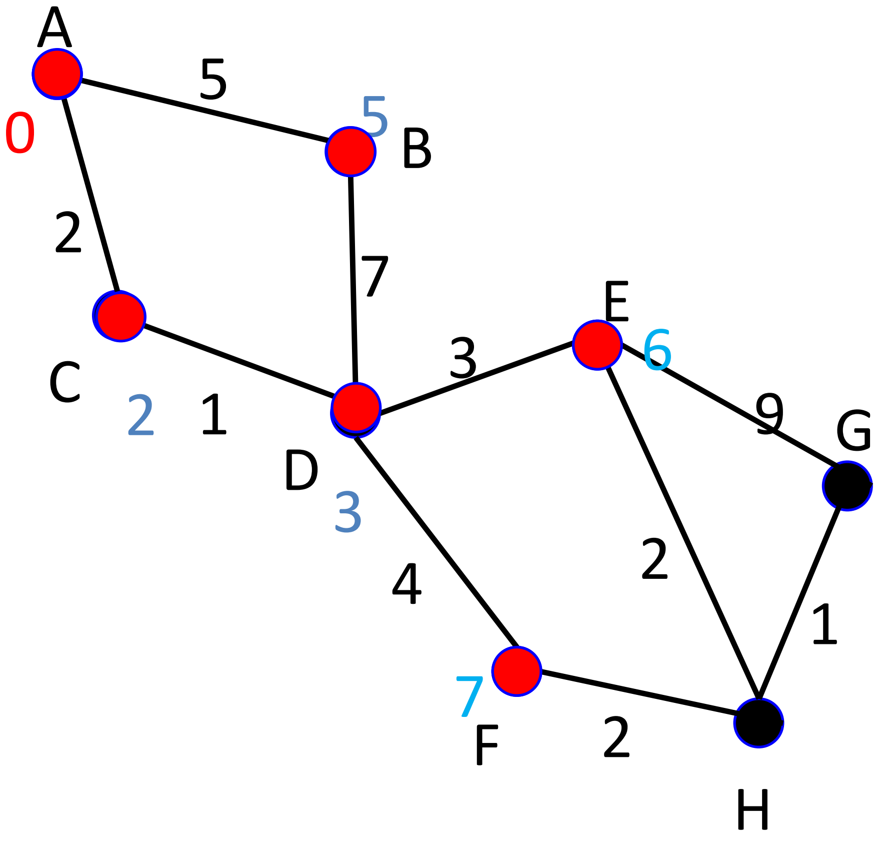
SSSP algorithms (I contd.)

- Need for synchronization at graph nodes
  - Suppose nodes B and C are relaxed simultaneously
  - Both relaxations may update value at D
    - Value at D is infinity
    - Relax-C operation reads this value and wants to update it to 3.
    - At the same time, Relax-D operation reads D’s value and wants to update it to 12
    - If the two updates are not sequenced properly, final value at D after both relaxations may be 12, which is incorrect
  - One solution: ensure that the “read-modify-write” in edge relaxation is “atomic” – no other thread can read or write that location while the read-modify-write is happening
- Also need synchronization at node being relaxed to ensure its value is not changed by some other core when the node relaxation is going on



SSSP algorithms (II)

- Dijkstra’s algorithm (1959):
  - priority queue of nodes, ordered by shortest distance known to node
  - iterate over nodes in priority order
  - node is relaxed just once
  - work-efficient: O(|E|*lg(|V|))
- Active nodes:
  - nodes in PQ: level has been lowered but node has not yet been relaxed
- Key data structures:
  - Graph
  - Work set/multiset: ordered
    - Priority queue
- Parallelism in algorithm
  - Edges connected to node can be relaxed in parallel
  - Difficult to relax multiple nodes from priority queue in parallel
  - Little parallelism for sparse graphs
- Ordered algorithm

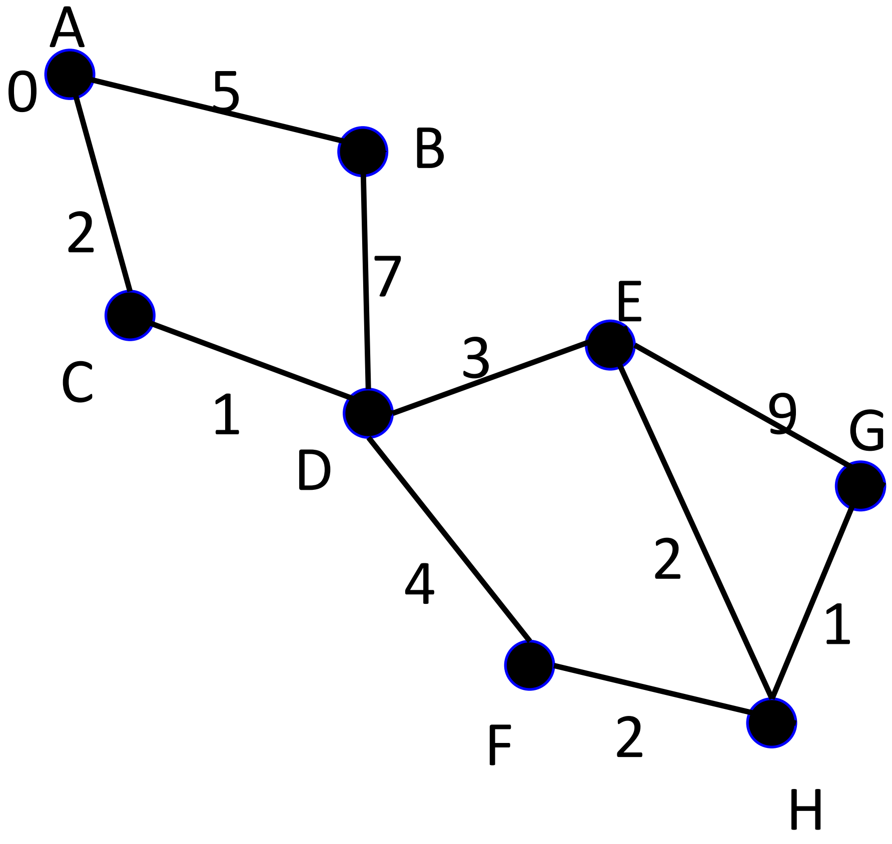


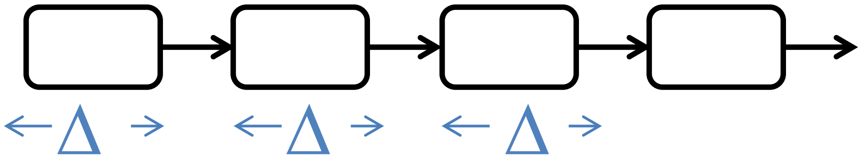
<A,0> <C,2> <D,3>	<B,5> <E,6>	<F,7>	

Priority queue

SSSP algorithms (III)

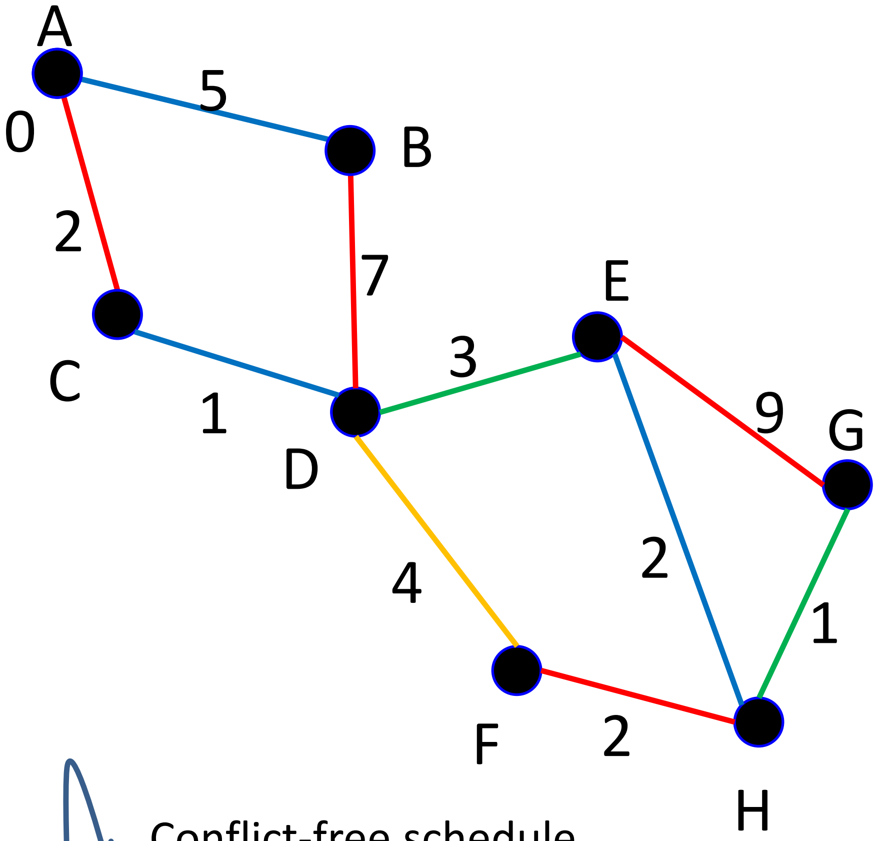
- Delta-stepping (1998)
  - variation of chaotic relaxation
  - active nodes currently closer to source are more likely to be chosen for processing from set
- Work-set/multiset:
  - Parameter: Δ
  - Sequence of sets
  - Nodes whose current distance is between $n\Delta$ and $(n+1)\Delta$ are put in the nth set
  - Nodes in each set are processed in parallel
  - Nodes in set n are completed before processing of nodes in set (n+1) are started
- $\Delta = 1$: Dijkstra
- $\Delta = \infty$: Chaotic relaxation
- Picking an optimal Δ :
  - depends on graph and machine
  - Do experimentally





SSSP algorithms (IV)

- Bellman-Ford (1957):
  - Iterate over all edges of graph in any order, relaxing each edge
  - Do this |V| times
  - O(|E|*|V|)
- Parallelization
  - Iterate over set of edges
  - Inspector-executor: use graph matching to generate a conflict-free schedule of edge relaxations after input graph is given
  - Edges in a matching do not have nodes in common so they can be relaxed without synchronization
  - Barrier synchronization between successive stages in schedule



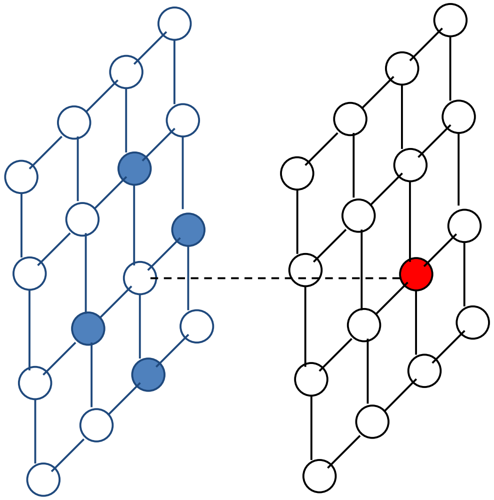
Conflict-free schedule

1. {(A,B),(C,D),(E,H)},
2. {(A,C),(B,D),(E,G),(F,H)},
3. {(D,E),(G,H)}
4. {(D,F)}

Example: Stencil computation

- Finite-difference computation
- Algorithm
  - Active nodes: nodes in A_{t+1}
  - Operator: five-point stencil
  - Different schedules have different locality
- Regular application
  - Grid structure and active nodes known statically
  - Application can be parallelized at compile-time

"Data-centric multilevel blocking"
Kodukula et al, PLDI 1997.

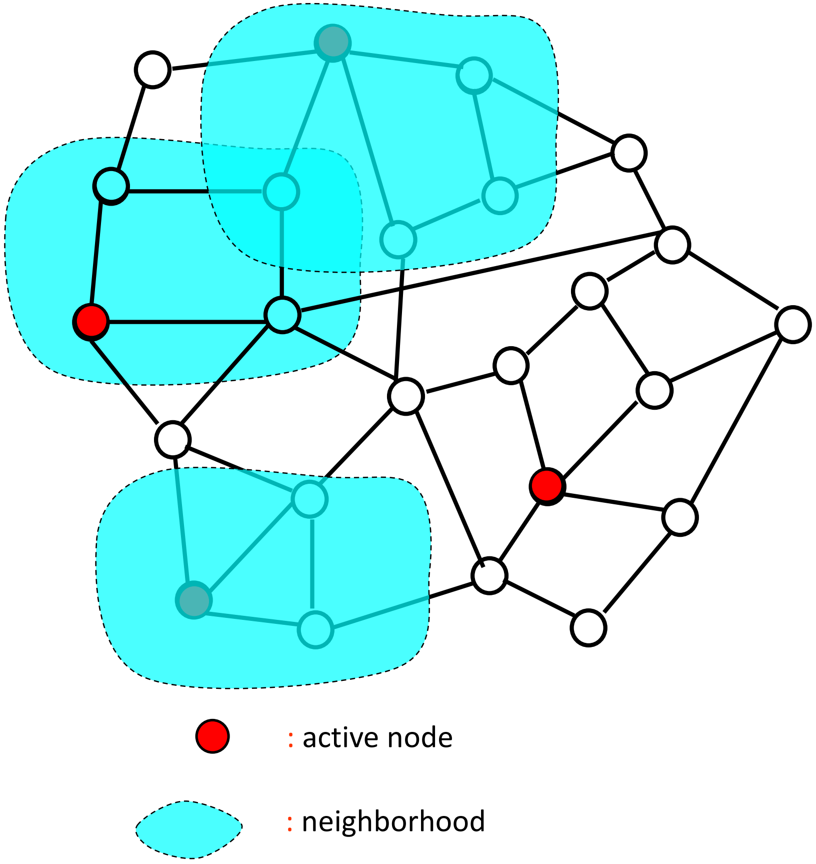


Jacobi iteration, 5-point stencil

```
//Jacobi iteration with 5-point stencil
//initialize array A
for time = 1, nsteps
   for <i,j> in [2,n-1]x[2,n-1]
       temp(i,j)=0.25*(A(i-1,j)+A(i+1,j)+A(i,j-1)+A(i,j+1))
   for <i,j> in [2,n-1]x[2,n-1]:
       A(i,j) = temp(i,j)
```

Operator formulation of algorithms

- Active element
  - Node /edge where computation is needed
- Local view: operator
  - Update at active element
  - Activity: application of operator to active element
  - Neighborhood: Set of nodes/edges read/written by activity
- Global view: schedule
  - Unordered algorithms: no semantic constraints but performance may depend on schedule
  - Ordered algorithms: problem-dependent order
- Amorphous data-parallelism
  - Multiple active nodes can be processed in parallel subject to neighborhood and ordering constraints



Parallel program = Operator + Schedule + Parallel data structure

Parallelization strategies: Binding Time

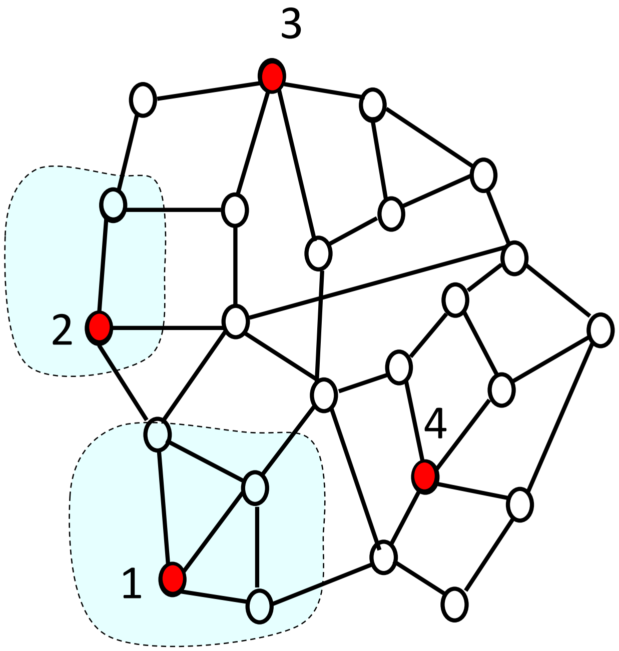
When do you know the active nodes and neighborhoods?

Compile-time — **Static parallelization (stencil codes, FFT, dense linear algebra)**

After input is given — **Inspector-executor (Bellman-Ford)**

During program execution — **Interference graph (DMR, chaotic SSSP)**

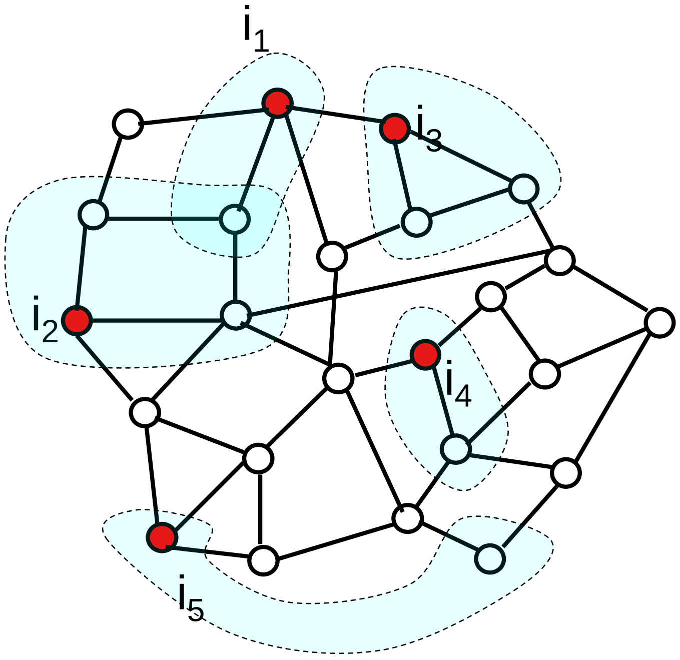
After program is finished — **Optimistic Parallelization (Time-warp)**



“The TAO of parallelism in algorithms” Pingali et al, PLDI 2011

Locality

- Temporal locality:
  - Activities with overlapping neighborhoods should be scheduled close in time
  - Example: activities i_1 and i_2
- Spatial locality:
  - Abstract view of graph can be misleading
  - Depends on the concrete representation of the data structure
- Inter-package locality:
  - Partition graph between packages and partition concrete data structure correspondingly
  - Active node is processed by package that owns that node



Abstract data structure

src	1	1	2	3
dst	2	1	3	2
val	3.4	3.6	0.9	2.1

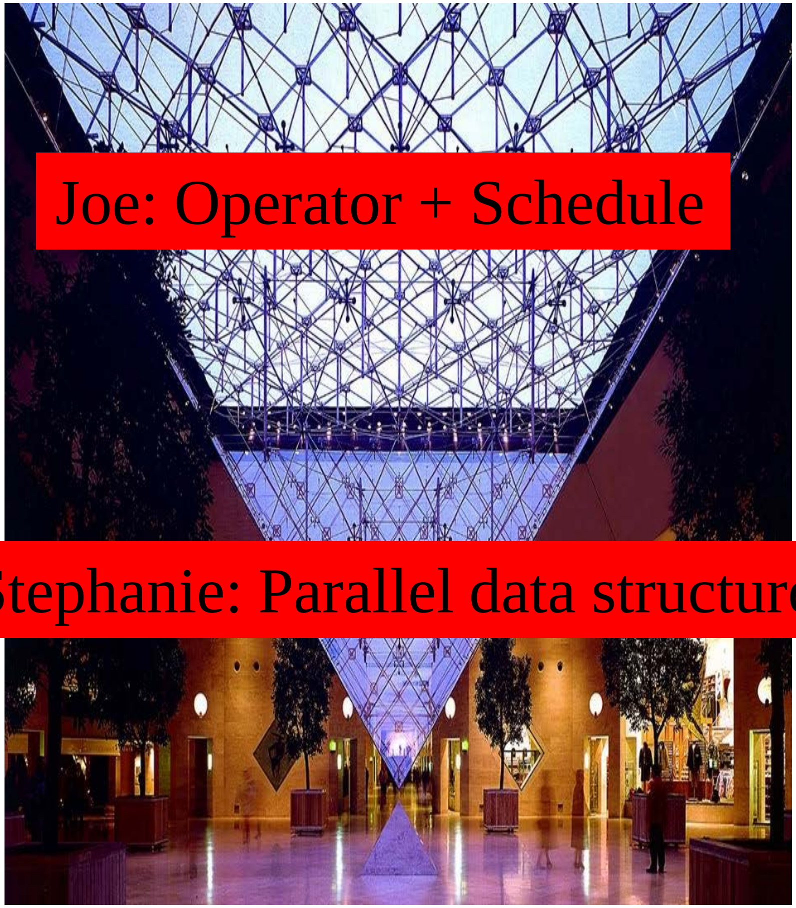
Concrete representation: coordinate storage

GALOIS SYSTEM

Galois system

Parallel program = Operator + Schedule + Parallel data structures

- Ubiquitous parallelism:
  - small number of expert programmers (Stephanies) must support large number of application programmers (Joes)
  - cf. SQL
- Galois system:
  - Stephanie: library of concurrent data structures and runtime system
  - Joe: application code in sequential C++
    - Galois set iterator for highlighting opportunities for exploiting ADP



Joe: Operator + Schedule

Stephanie: Parallel data structures

Hello graph Galois Program

```
#include “Galois/Galois.h”
#include “Galois/Graphs/LCGraph.h”

struct Data { int value; float f; };

typedef Galois::Graph::LC_CSR_Graph<Data,void> Graph;
typedef Galois::Graph::GraphNode Node;

Graph graph;

struct P {
  void operator()(Node n, Galois::UserContext<Node>& ctx) {
    graph.getData(n).value += 1;
  }
};

int main(int argc, char** argv) {
  graph.structureFromGraph(argv[1]);
  Galois::for_each(graph.begin(), graph.end(), P());
  return 0;
}
```

Data structure Declarations

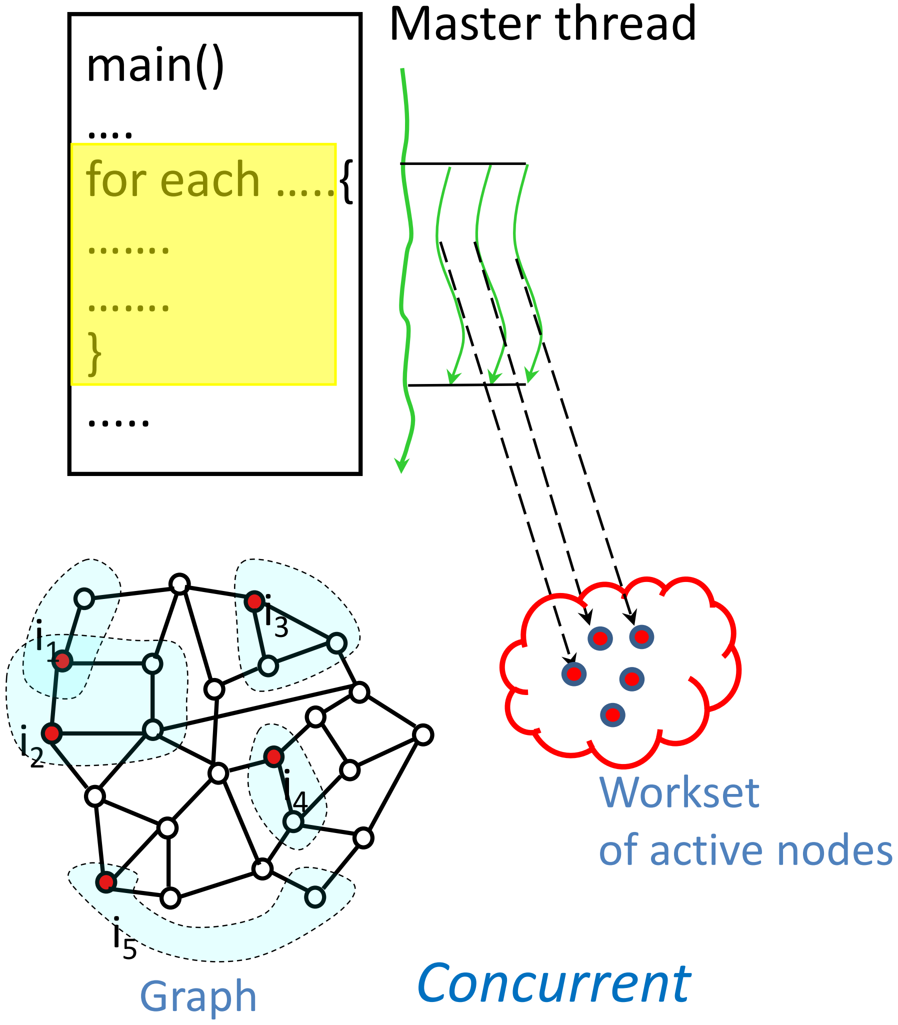
Operator

Galois Iterator

Parallel execution of Galois programs

- Application (Joe) program
  - Sequential C++
  - Galois set iterator: for each
    - New elements can be added to set during iteration
    - Optional scheduling specification (cf. OpenMP)
    - Highlights opportunities in program for exploiting amorphous data-parallelism
- Runtime system
  - Ensures serializability of iterations
  - Execution strategies
    - Speculation
    - Interference graphs

Application Program



```
main()
....
for each .....{
.......
.......
}
.....
```

Master thread

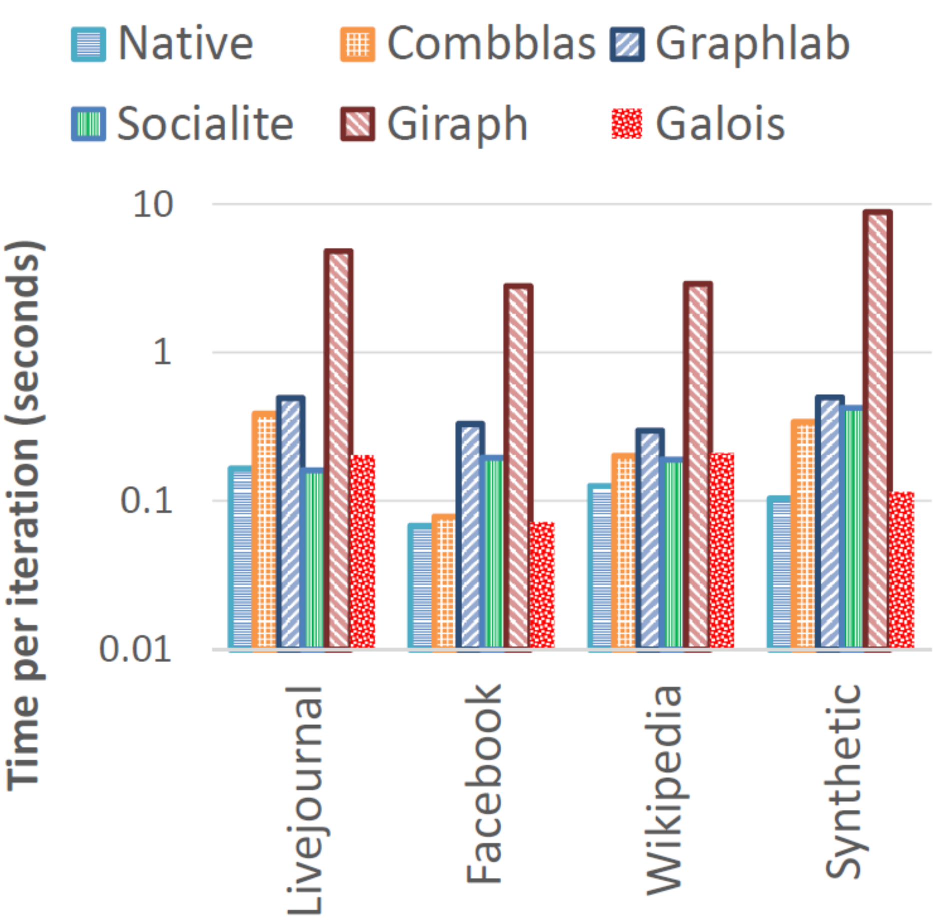
Workset of active nodes

Graph

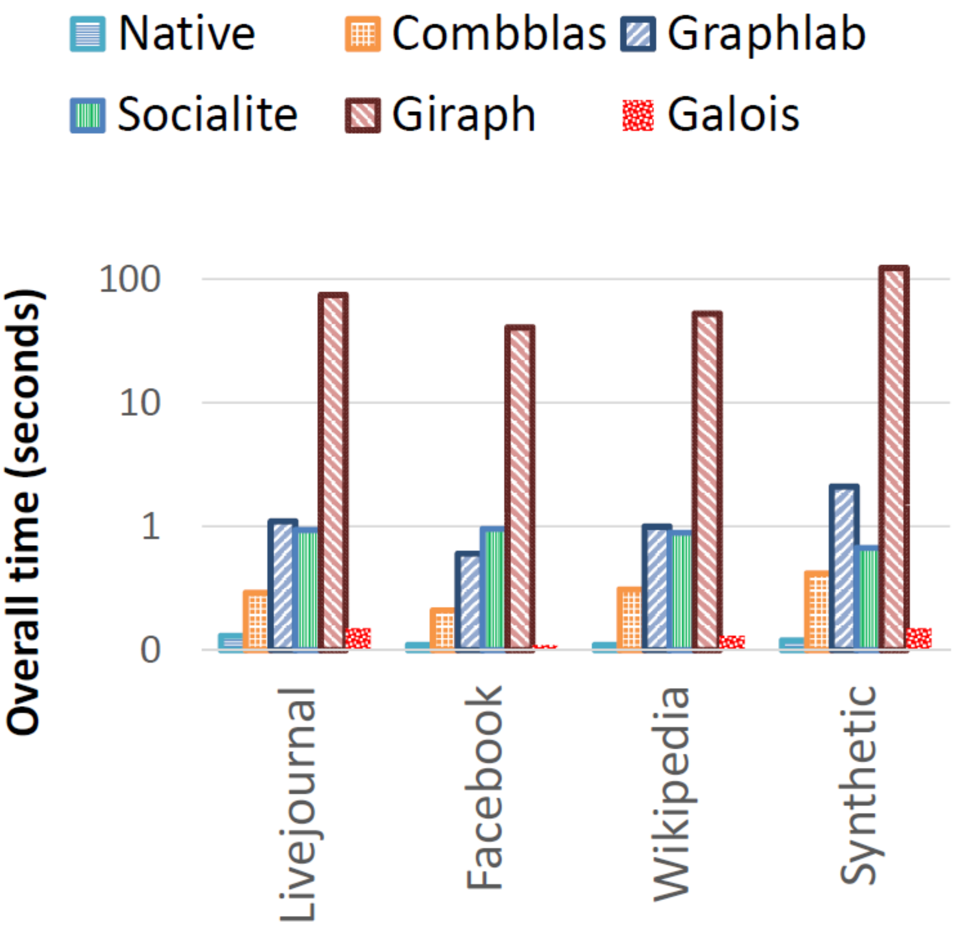
Concurrent data structures

PERFORMANCE STUDIES

Intel Study: Galois vs. Graph Frameworks



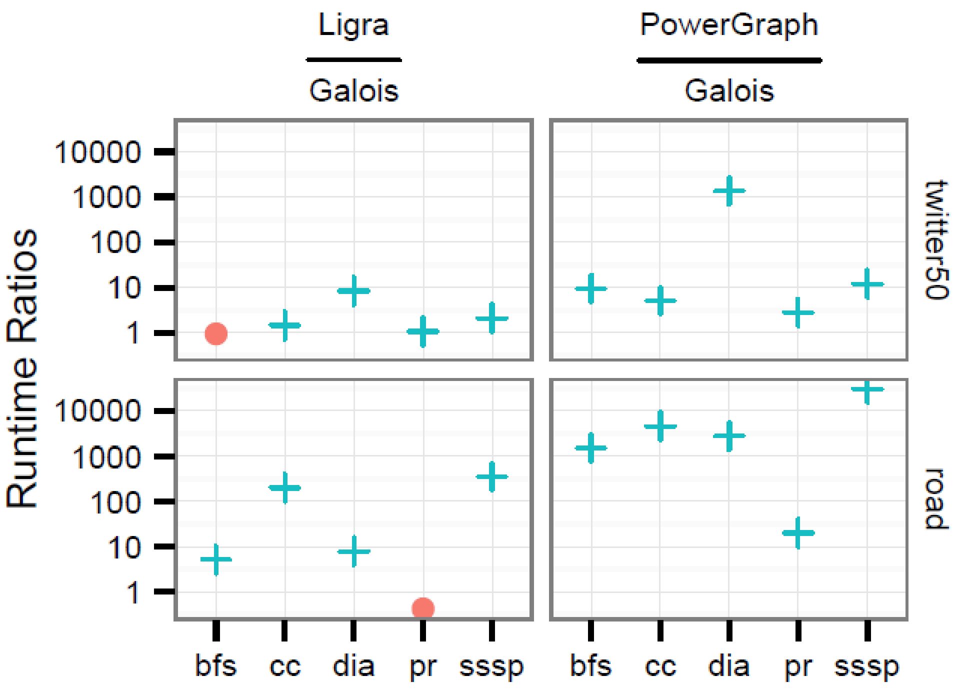
(a) PageRank

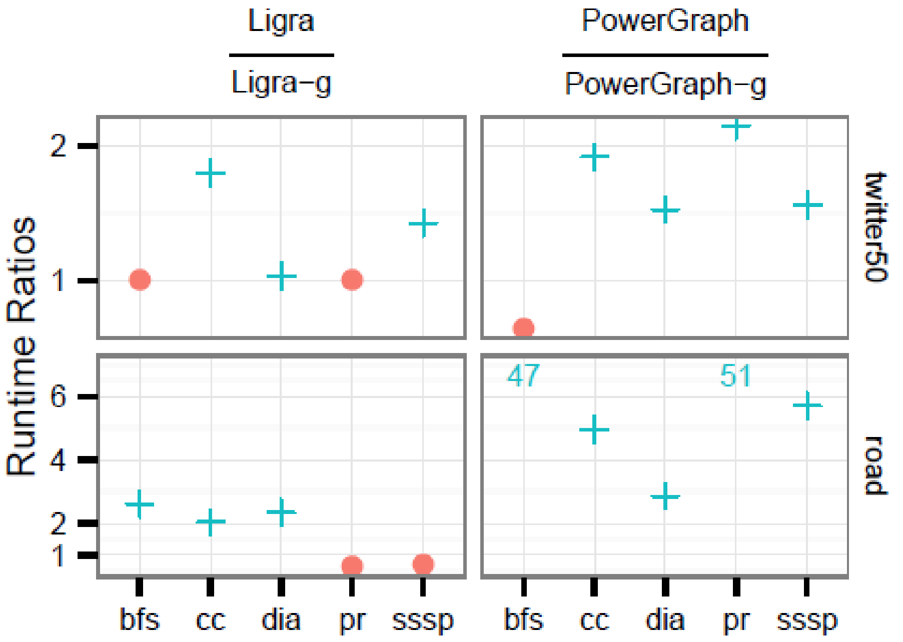


(b) Breadth-First Search

“Navigating the maze of graph analytics frameworks” Nadathur et al SIGMOD 2014

Galois: Graph analytics

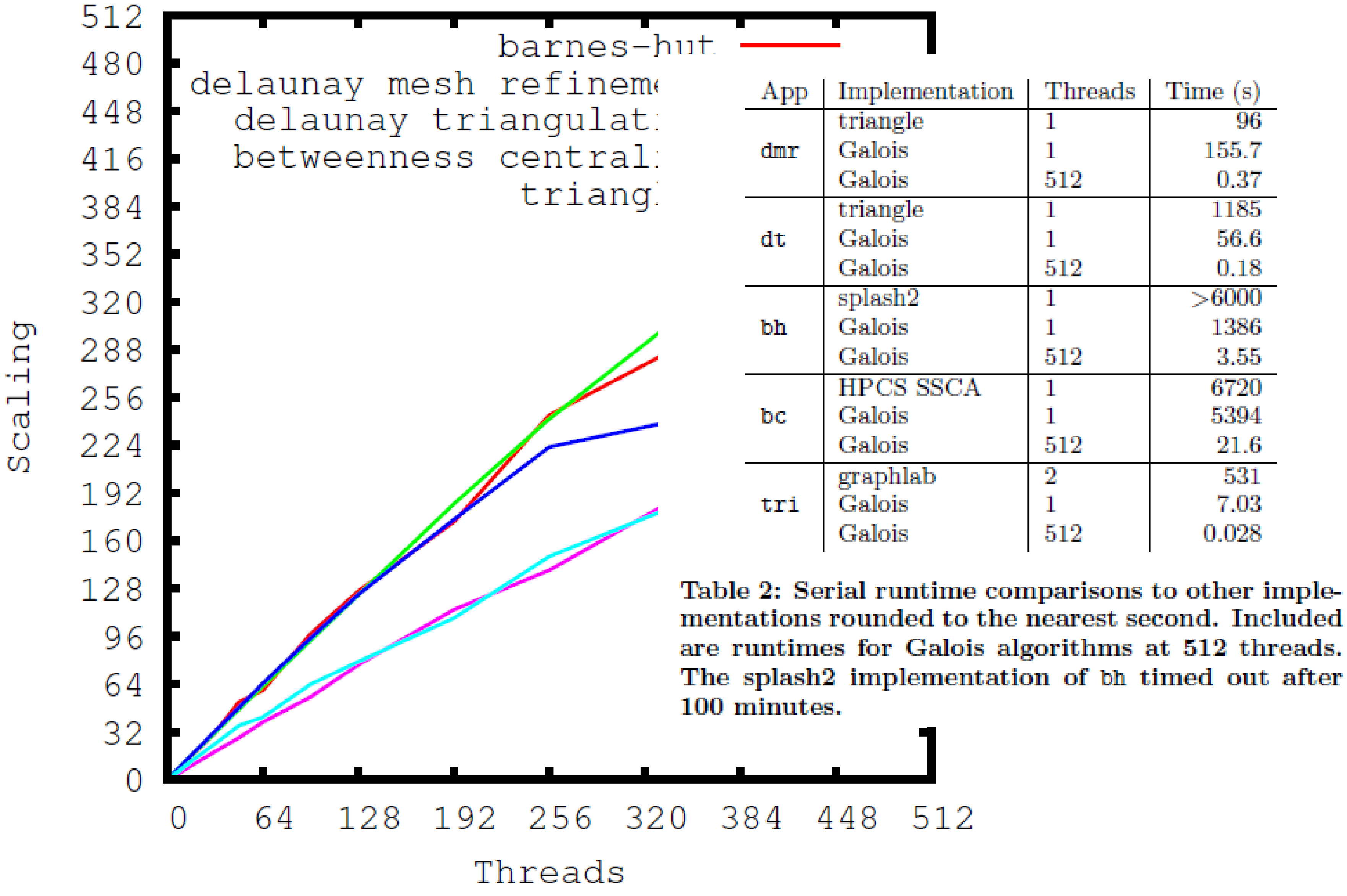




- Galois lets you code more effective algorithms for graph analytics than DSLs like PowerGraph (left figure)
- Easy to implement APIs for graph DSLs on top on Galois and exploit better infrastructure (few hundred lines of code for PowerGraph and Ligra) (right figure)

“A lightweight infrastructure for graph analytics” Nguyen, Lenharth, Pingali (SOSP 2013)

Galois: Performance on SGI Ultraviolet

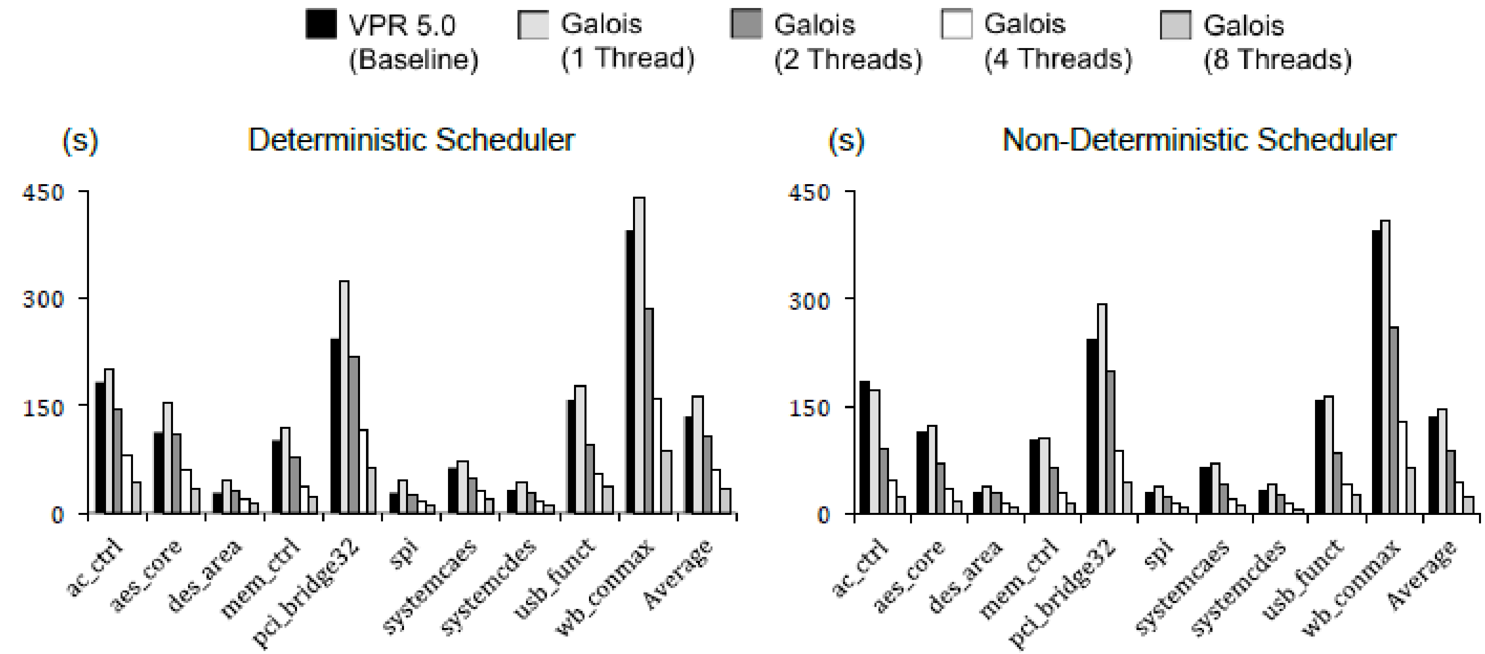


App	Implementation	Threads	Time (s)
dmr	triangle	1	96
	Galois	1	155.7
	Galois	512	0.37
dt	triangle	1	1185
	Galois	1	56.6
	Galois	512	0.18
bh	splash2	1	>6000
	Galois	1	1386
	Galois	512	3.55
bc	HPCS SSCA	1	6720
	Galois	1	5394
	Galois	512	21.6
tri	graphlab	2	531
	Galois	1	7.03
	Galois	512	0.028

Table 2: Serial runtime comparisons to other implementations rounded to the nearest second. Included are runtimes for Galois algorithms at 512 threads. The splash2 implementation of bh timed out after 100 minutes.

FPGA Tools

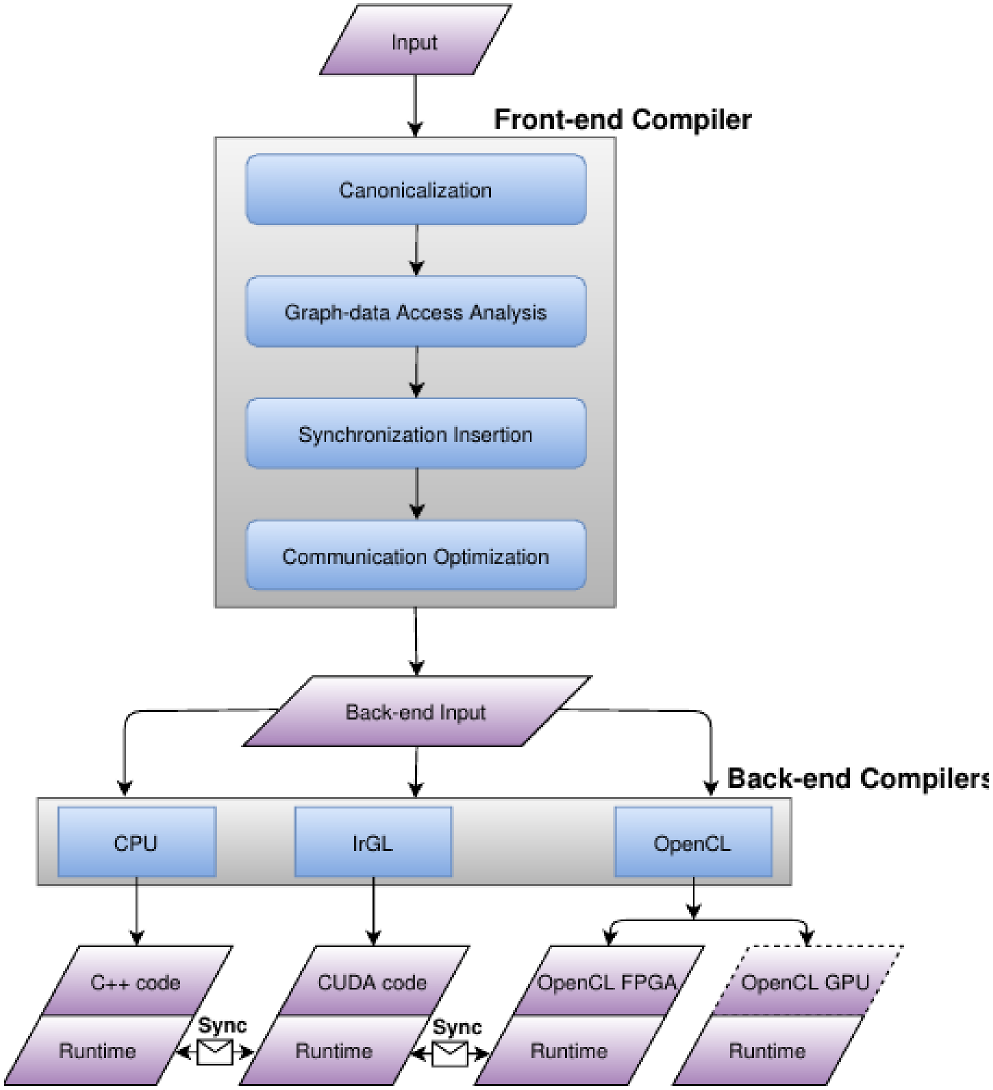
Maze Router Execution Time



Averages				
	VPR 5.0	134.6 seconds	VPR 5.0	134.6 seconds
	Galois (1 Thread)	162.4 seconds	Galois (1 Thread)	145.3 seconds
	Galois (2 Threads)	106.6 seconds	Galois (2 Threads)	88.8 seconds
	Galois (4 Threads)	59.2 seconds	Galois (4 Threads)	43.0 seconds
	Galois (8 Threads)	33.7 seconds	Galois (8 Threads)	22.6 seconds

**Moctar & Brisk, “Parallel FPGA Routing based on the Operator Formulation”
DAC 2014**

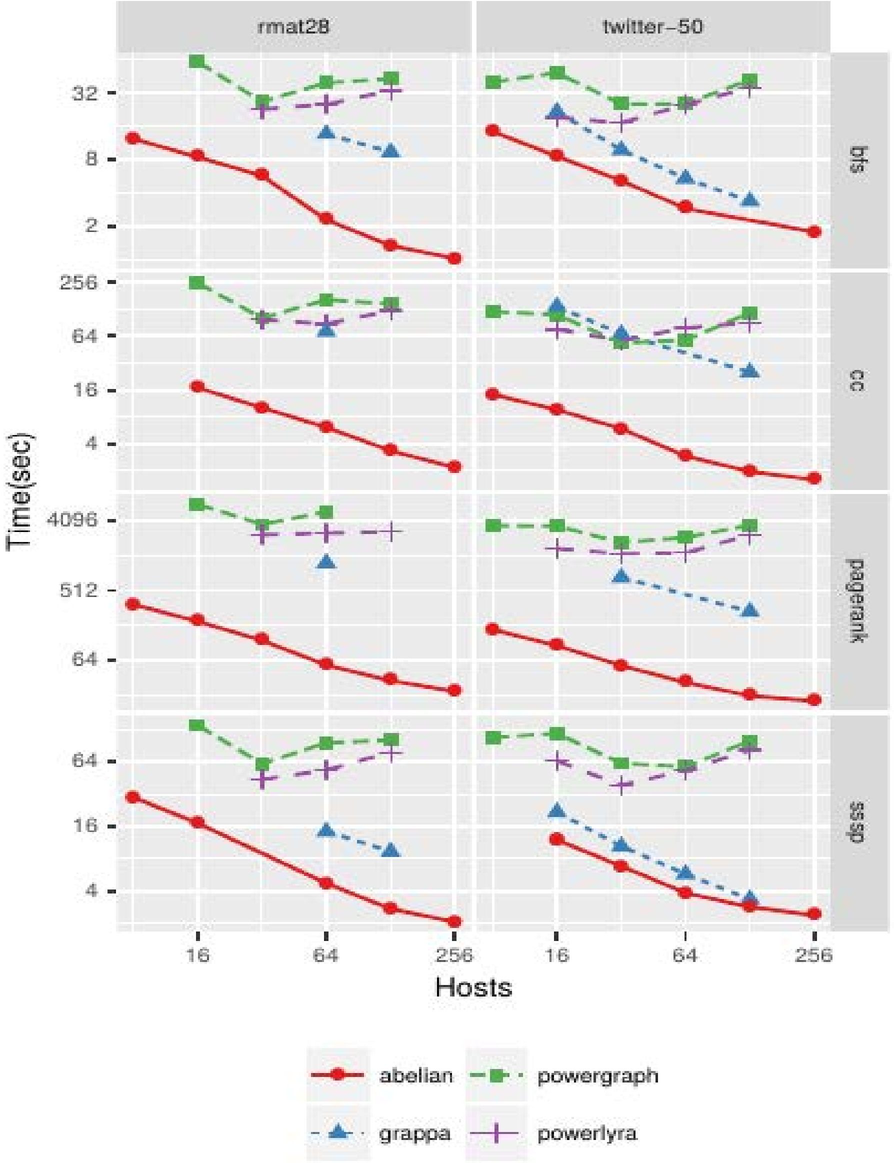
Abelian Compiler



Distributed-memory

	rmat28	Twitter-50
|V|	268M	51M
|E|	4,296M	1,963M

Stampede cluster at TACC

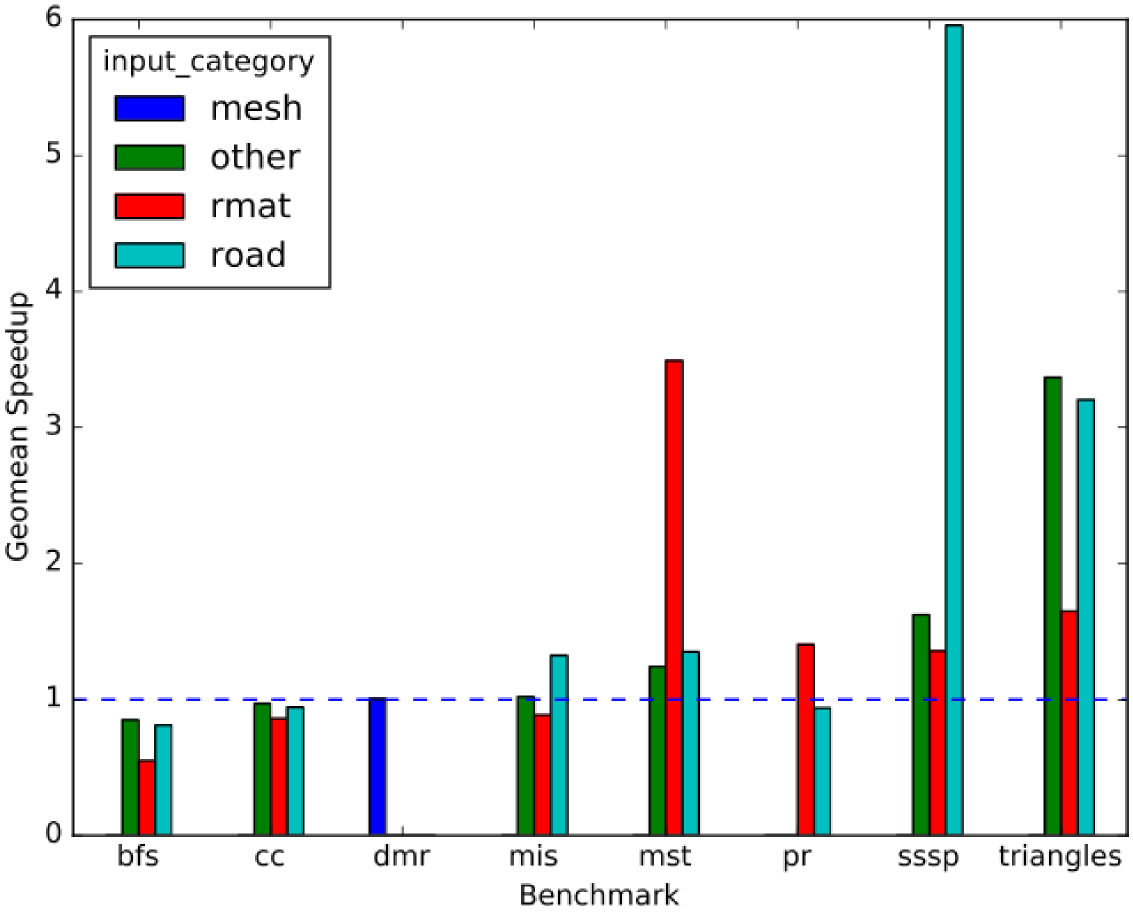


GPU Code Generation*

- Three core optimizations:
  - Nested Parallelism: improves load-balance
  - Cooperative Conversion: reduces # of atomics
  - Iteration Outlining: reduces GPU underutilization for short kernels
- Applied automatically by compiler

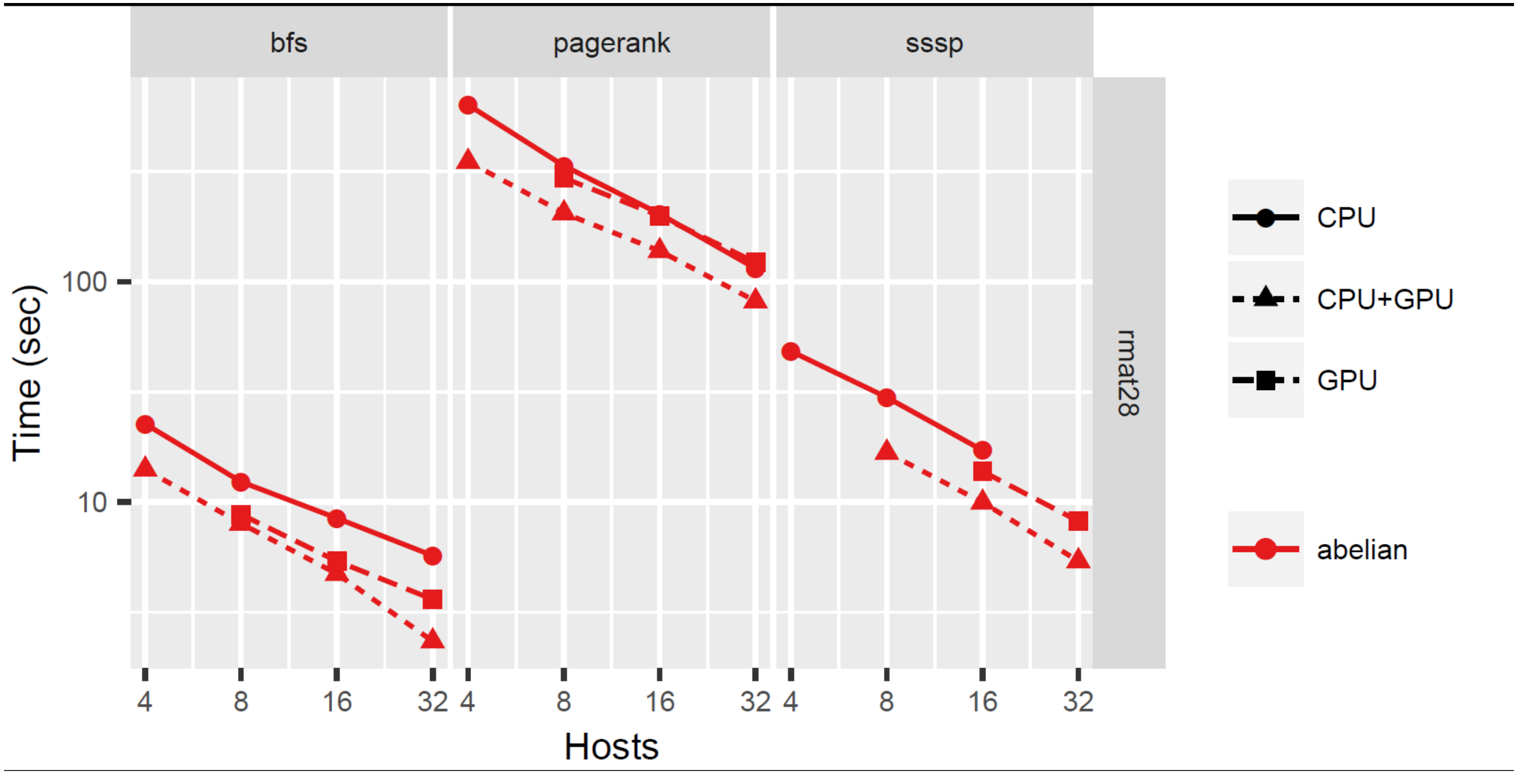
- Sreepathi Pai, Keshav Pingali, “A Compiler for Throughput Optimization of Graph Algorithms on GPUs”, OOPSLA 2016, To Appear.

GPU Performance



Baseline: best publicly available CUDA code for application

Heterogeneous Execution (CPU+GPU)

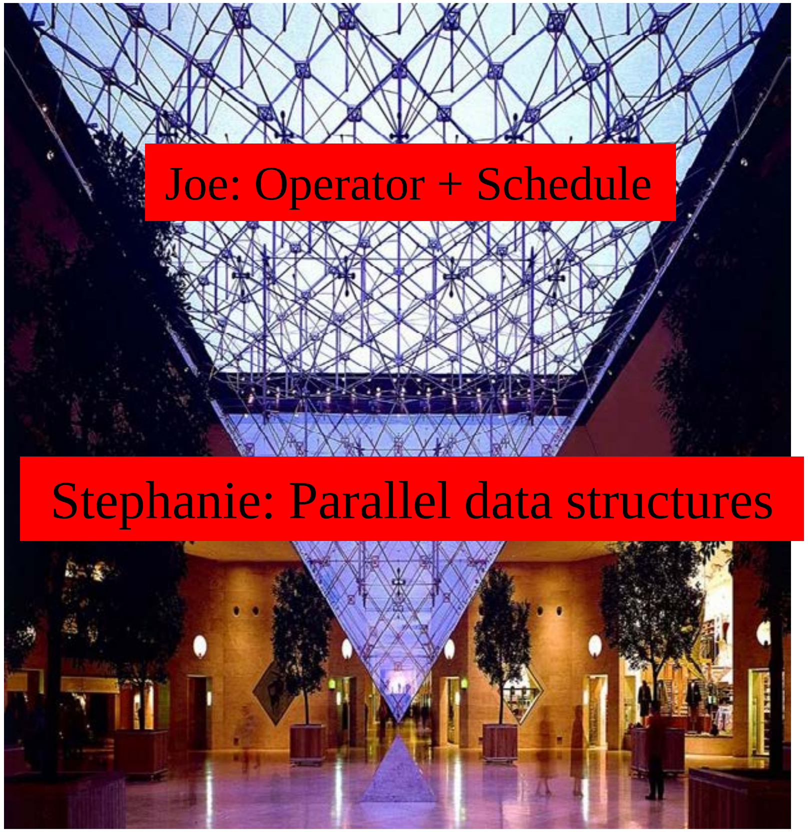


Collaborations

- BAE:
  - RIPE system for intrusion detection (DARPA project)
  - Distributed, heterogeneous (multicore+GPU+FPGA) implementation of Galois
- HP Enterprise:
  - Systems evaluation for graph analytics workloads
- Raytheon
  - Parallel machine learning algorithms for smart weapons systems
- Proteus (DARPA project with MIT, Rice, Chicago)
  - Approximate computing
- Maciej Paczynski, Krakow
  - Multi-frontal sparse direct solvers for fracture problems

Conclusions

- Yesterday:
  - Computation-centric view of parallelism
- Today:
  - Data-centric view of parallelism
  - Operator formulation of algorithms
  - Permits a unified view of parallelism and locality in algorithms
  - Joe/Stephanie programming model
  - Galois system is an implementation
- Tomorrow:
  - DSLs for different applications
  - Layer on top of Galois



Joe: Operator + Schedule

Stephanie: Parallel data structures

Parallel program = Operator + Schedule + Parallel data structure

More information

- Website
  - http://iss.ices.utexas.edu
- Download
  - Galois system for multicores
  - Lonestar benchmarks
  - All our papers