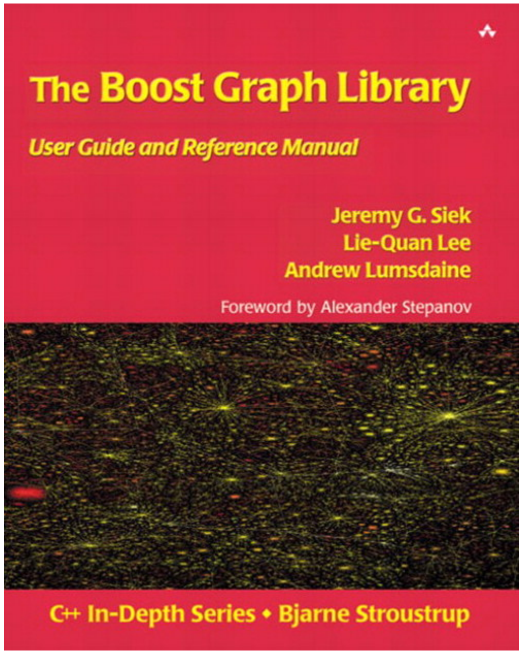
The Parallel Boost Graph Library `spawn`(Active Pebbles)

Nicholas Edmonds and Andrew Lumsdaine
Center for Research in Extreme Scale Technologies
Indiana University

Origins

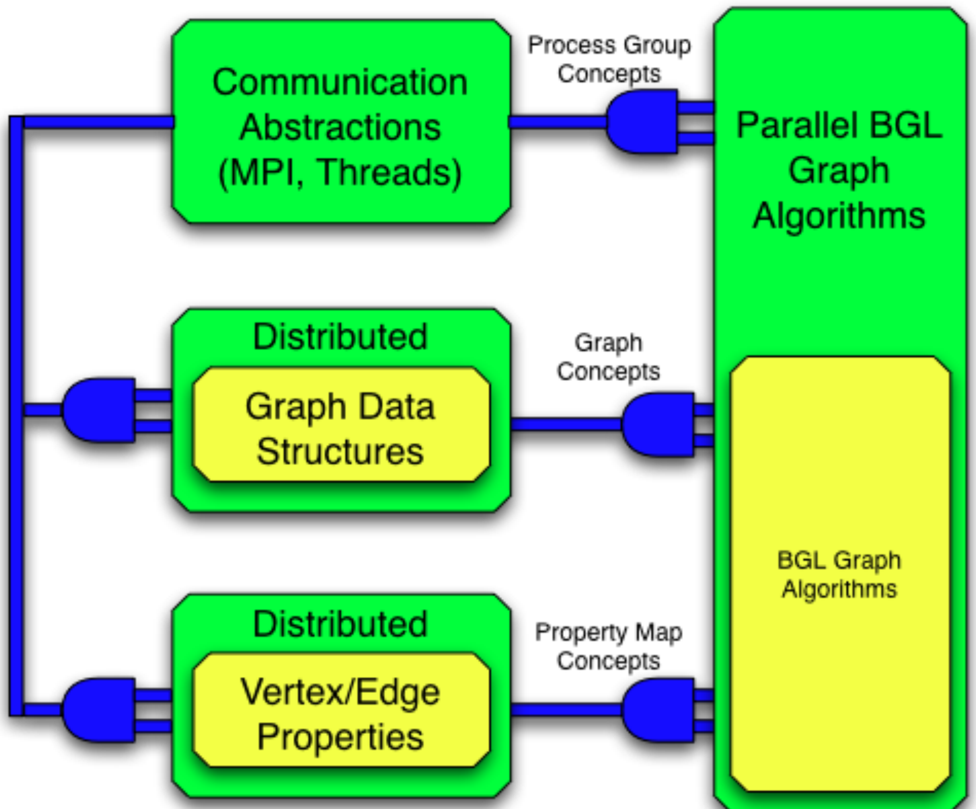
- Boost Graph Library (1999)
  - Generic programming with templates
  - Good sequential performance with high-level abstractions
  - Algorithm composition, visitors, property maps, etc.



Lie-Quan Lee, Jeremy Siek, and Andrew Lumsdaine. **Generic Graph Algorithms for Sparse Matrix Ordering**. ISCOPE. Lecture Notes in Computer Science 1732. 1999.

Origins

- Parallel BGL (2005)
  - Lifting abstraction
  - Make what we already have parallel
  - Same interfaces as sequential BGL
  - Coarse-grained, BSP

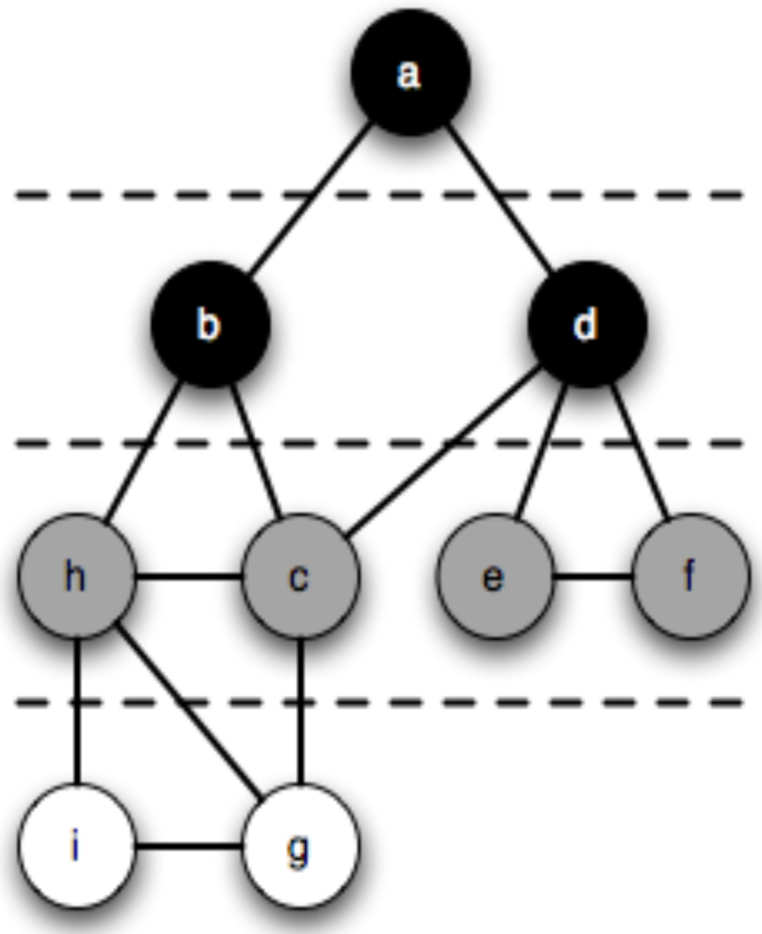


Douglas Gregor and Andrew Lumsdaine. **The Parallel BGL: A Generic Library for Distributed Graph Computation**. Workshop on Parallel Object-Oriented Scientific Computing. July, 2005.

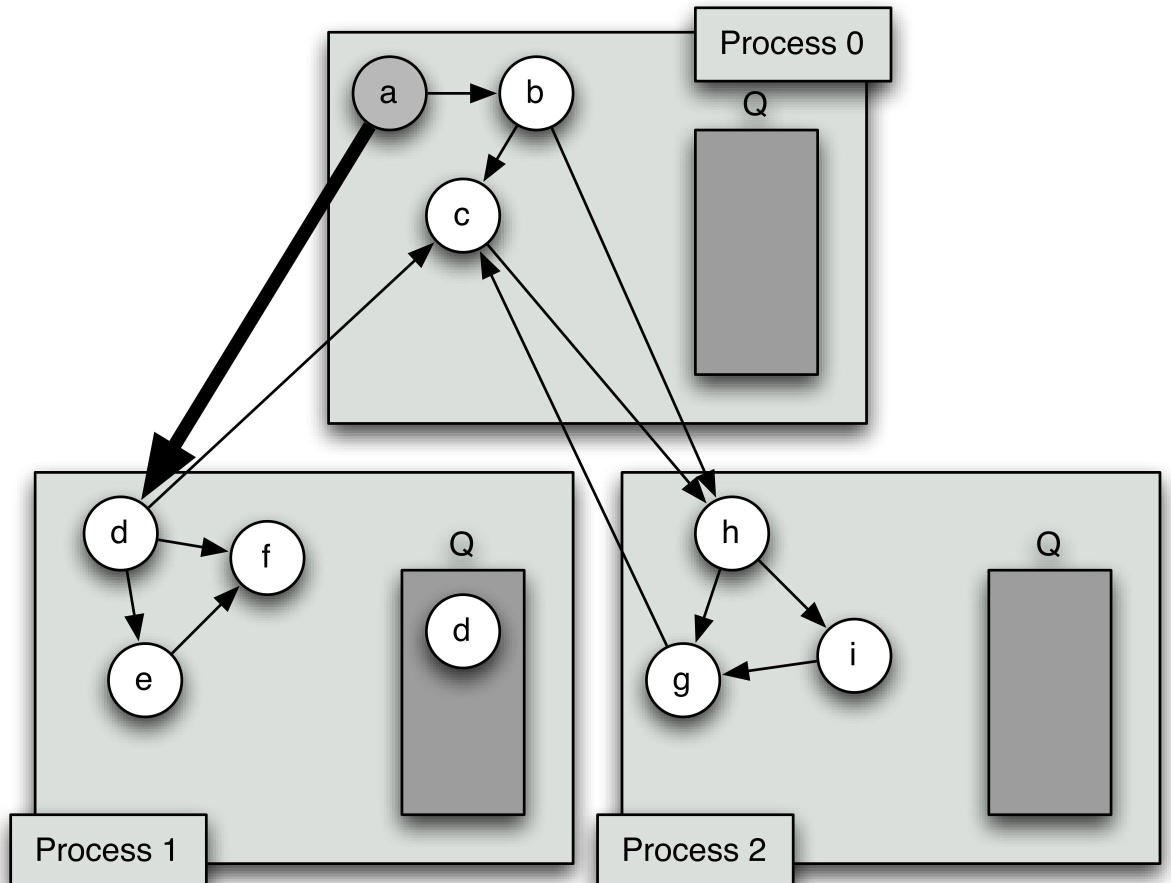
Douglas Gregor and Andrew Lumsdaine. **Lifting Sequential Graph Algorithms for Distributed-Memory Parallel Computation**. Object-Oriented Programming, Systems, Languages, and Applications. October, 2005

Case Study: Breadth-First Search

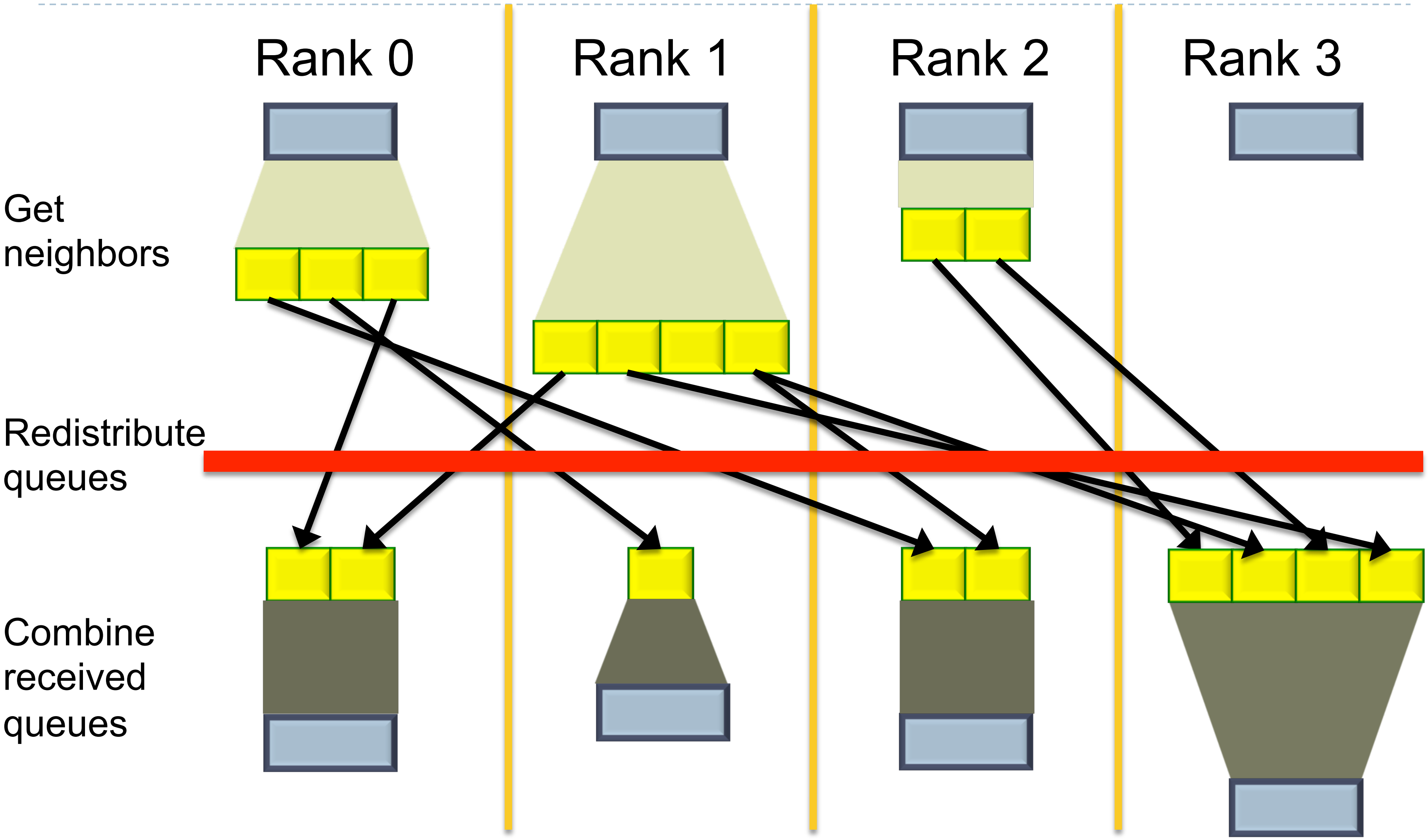
ENQUEUE(Q, s)
while ($Q \neq \emptyset$)
  $u \leftarrow$ DEQUEUE(Q)
  for (each $v \in Adj[u]$)
    if ($color[v] =$ WHITE)
      $color[v] \leftarrow$ GRAY
      ENQUEUE(Q, v)
    else $color[u] \leftarrow$ BLACK



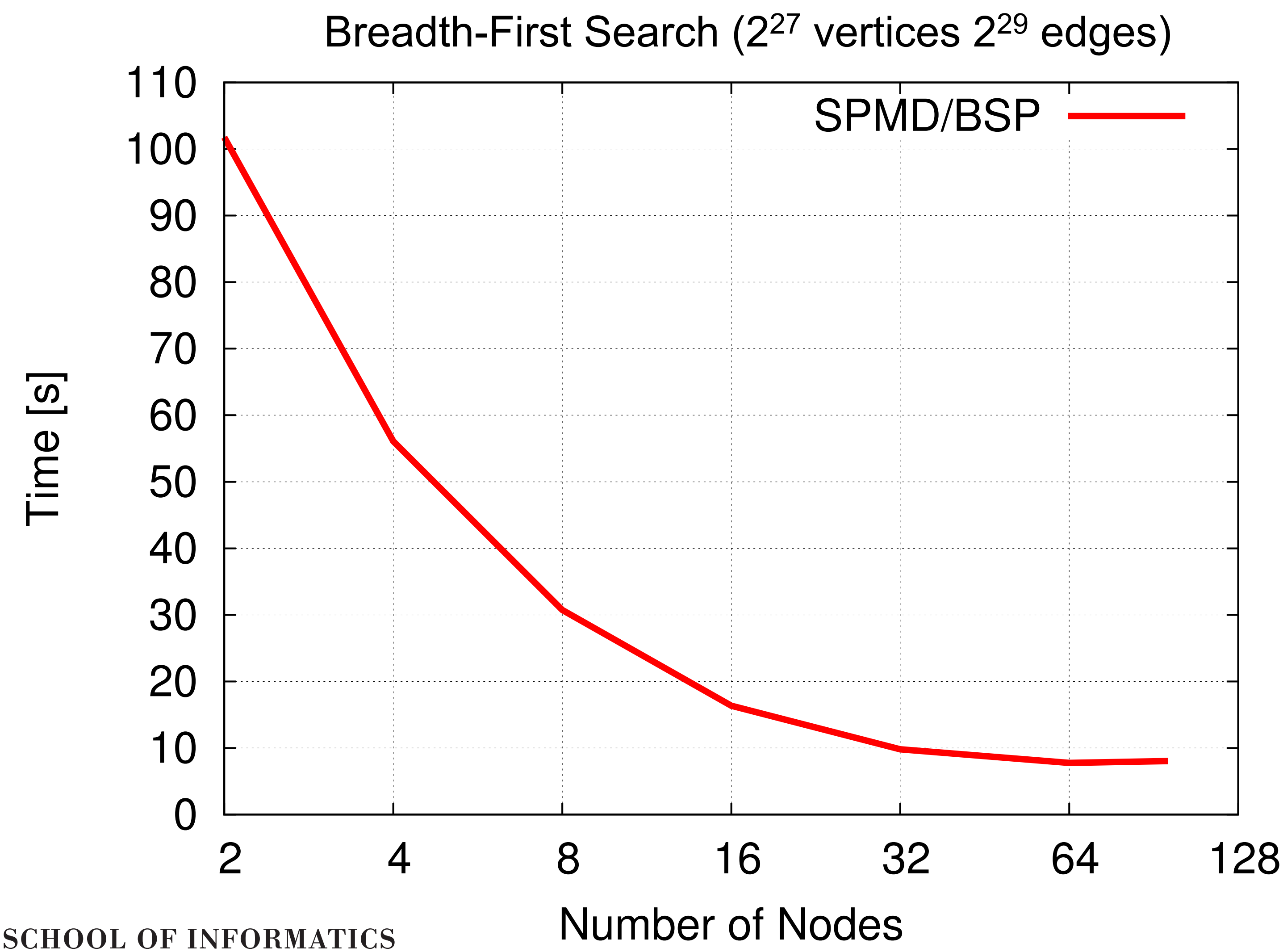
SPMD Breadth-First Search



SPMD Breadth-First Search



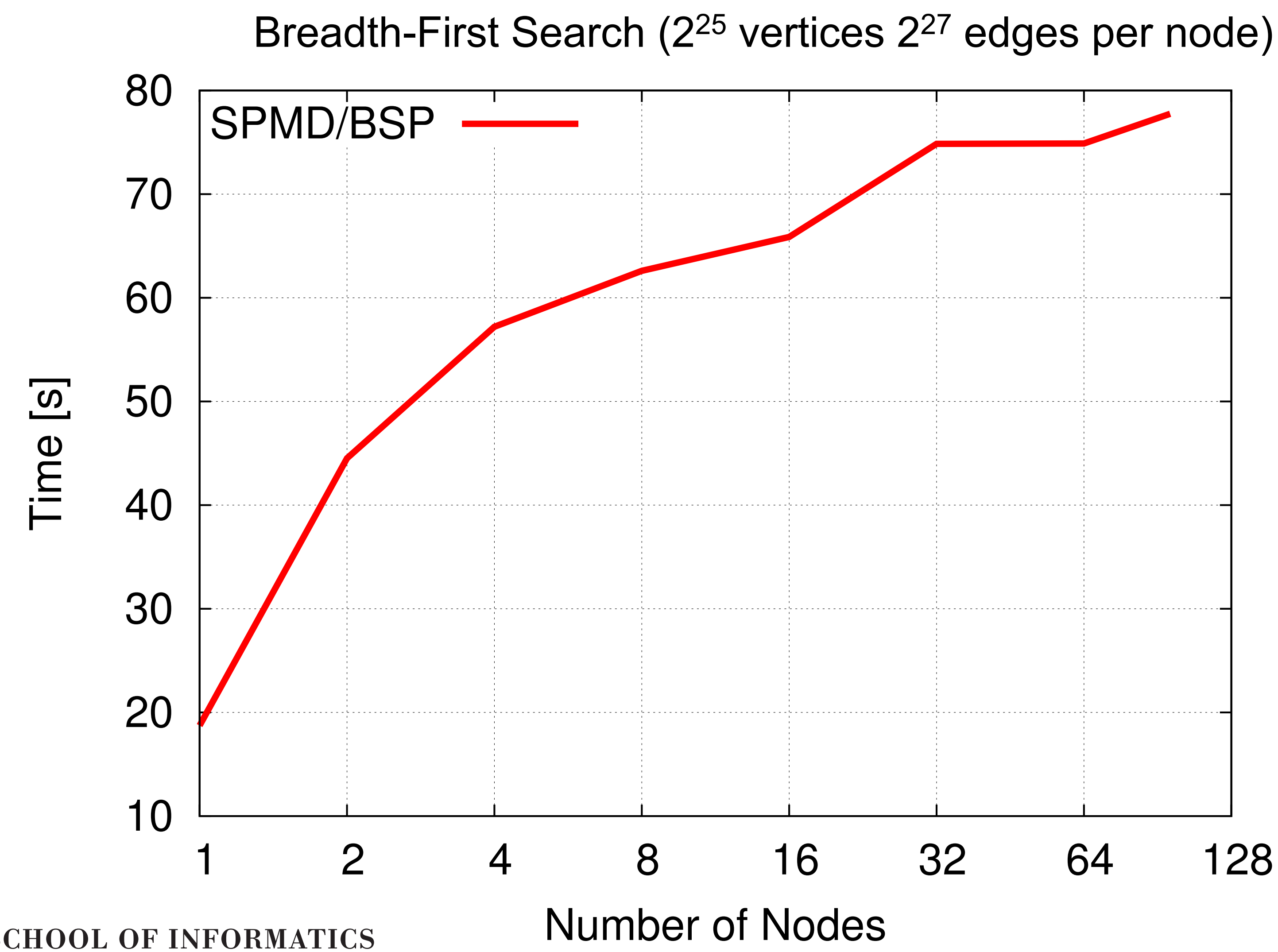
SPMD Breadth-First Search (Strong Scaling)



Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with four cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

SPMD Breadth-First Search (Weak Scaling)

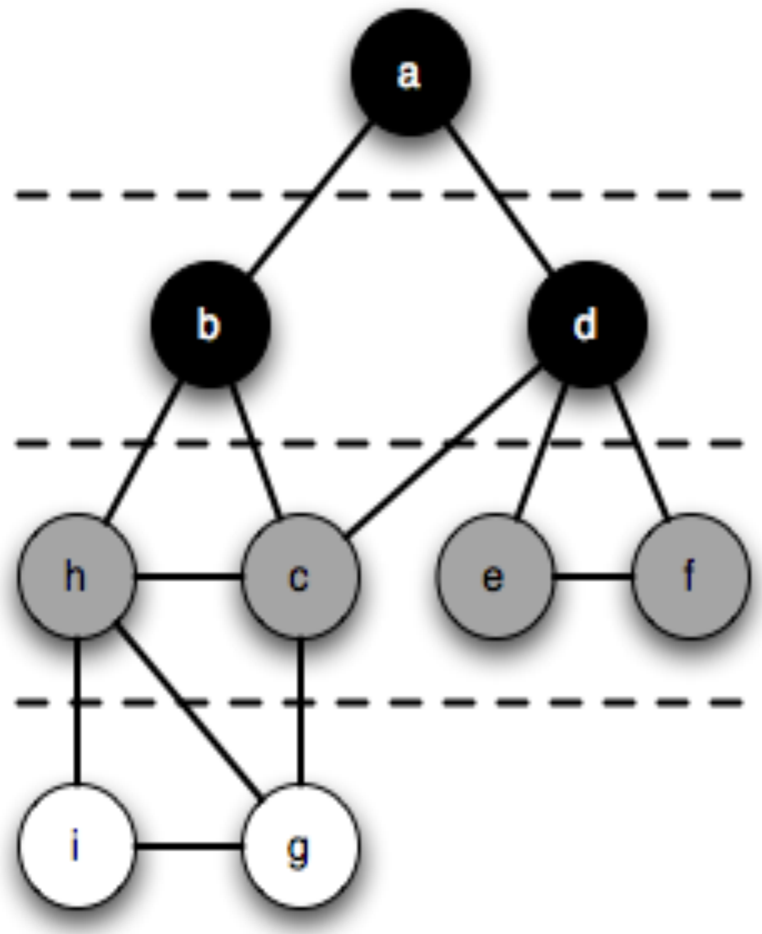


SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with four cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

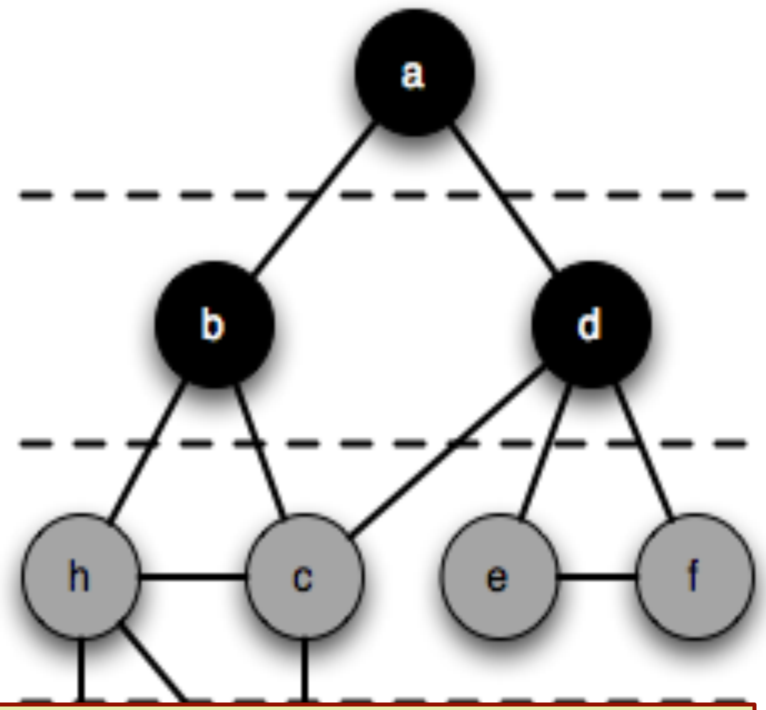
Find the Sequential Trap

ENQUEUE(Q, s)
while $(Q \neq \emptyset)$
  $\boxed{u \leftarrow \text{DEQUEUE(Q)}}$
  for (each $v \in Adj[u]$)
    if $(color[v] = \text{WHITE})$
      $color[v] \leftarrow \text{GRAY}$
      $\boxed{\text{ENQUEUE}(Q,\ v)}$
    else $color[u] \leftarrow \text{BLACK}$



Find the Synchronization Trap

ENQUEUE(Q, s)
while $(Q \neq \emptyset)$
  $u \leftarrow$ DEQUEUE(Q)
  for (each $v \in Adj[u]$)
    if $(color[v] =$ WHITE)
      $color[v] \leftarrow$ GRAY
      ENQUEUE$(Q,\ v)$
    else $color[u] \leftarrow$ BLACK



for *i* **in** *ranks*: **start receiving** *in_queue*[*i*] **from rank** *i*
for *j* **in** *ranks*: **start sending** *out_queue*[*j*] **to rank** *j*
synchronize and finish communications

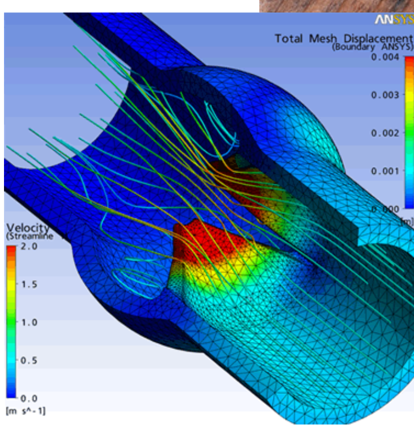
What's Wrong Here?

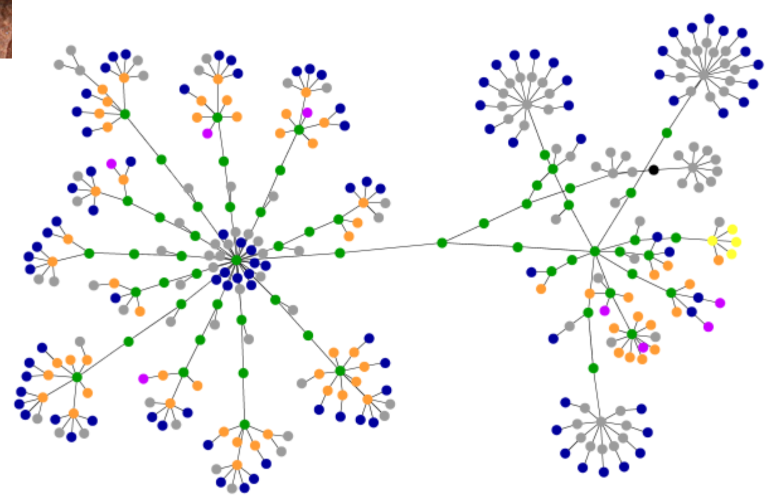
SPMD
Traditional Scientific Apps

- Coarse-grained
- Static
- Balanced
- Bandwidth-bound with large messages
- Communicates data

Opportunity
Data-driven Apps

- Fine-grained dependencies
- Dynamic, data-dependent
- Irregular
- Latency-bound with small messages
- Communicates control flow





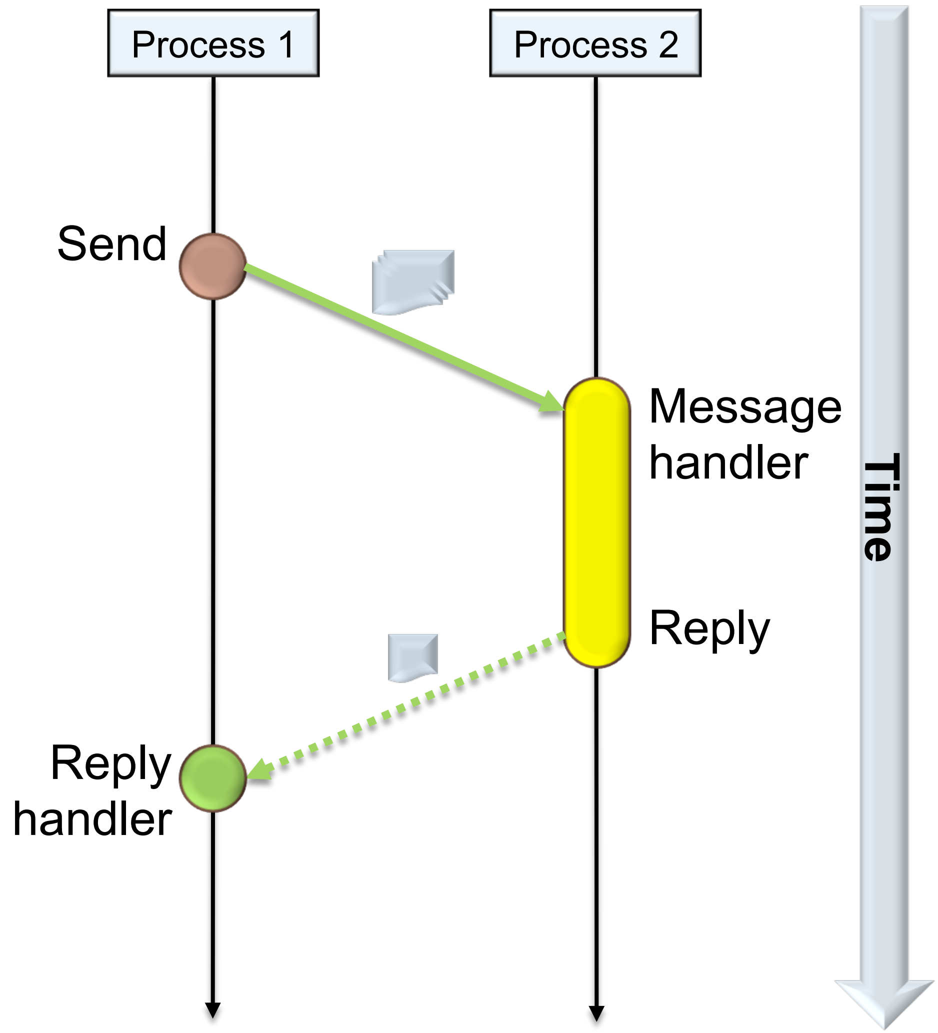
SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Messaging Models

- Two-sided
  - MPI
  - Explicit sends and receives
- One-sided
  - MPI-2 one-sided, ARMCI, PGAS languages
  - Remote put and get operations
  - Limited set of atomic updates into remote memory
- Active messages
  - GASNet, DCMF, LAPI, Charm++, X10, etc.
  - Explicit sends, implicit receives
  - User-defined handler called on receiver for each message

Active Messages

- Created by von Eicken et al, for Split-C (1992)
- Messages sent explicitly
- Receivers register handlers but not involved with individual messages
- Messages often asynchronous for higher throughput

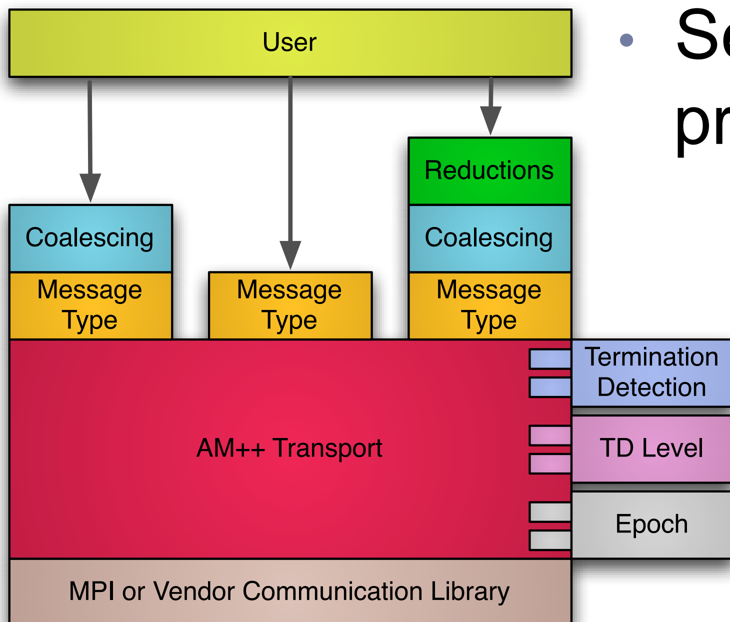


Active Messages

- Move control flow to data
- Fine-grained
- Asynchronous
- Uniformity of access

AM++

- Generic
- User-level
- Flexible/modular
- Send to *targets*, not processors



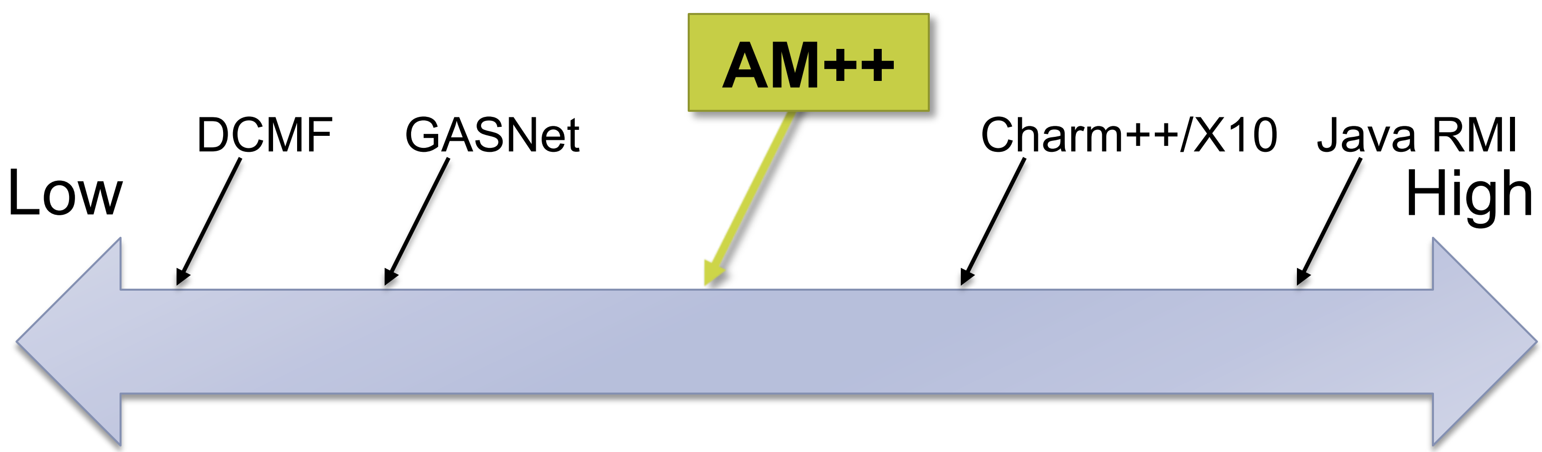
Low-Level vs. High-Level AM Systems

- Active messaging systems (loosely) on a spectrum of features vs. performance
  - Low-level systems typically have restrictions on message handler behavior, explicit buffer management, etc.
  - High-level systems often provide dynamic load balancing, service discovery, authentication/security, etc.

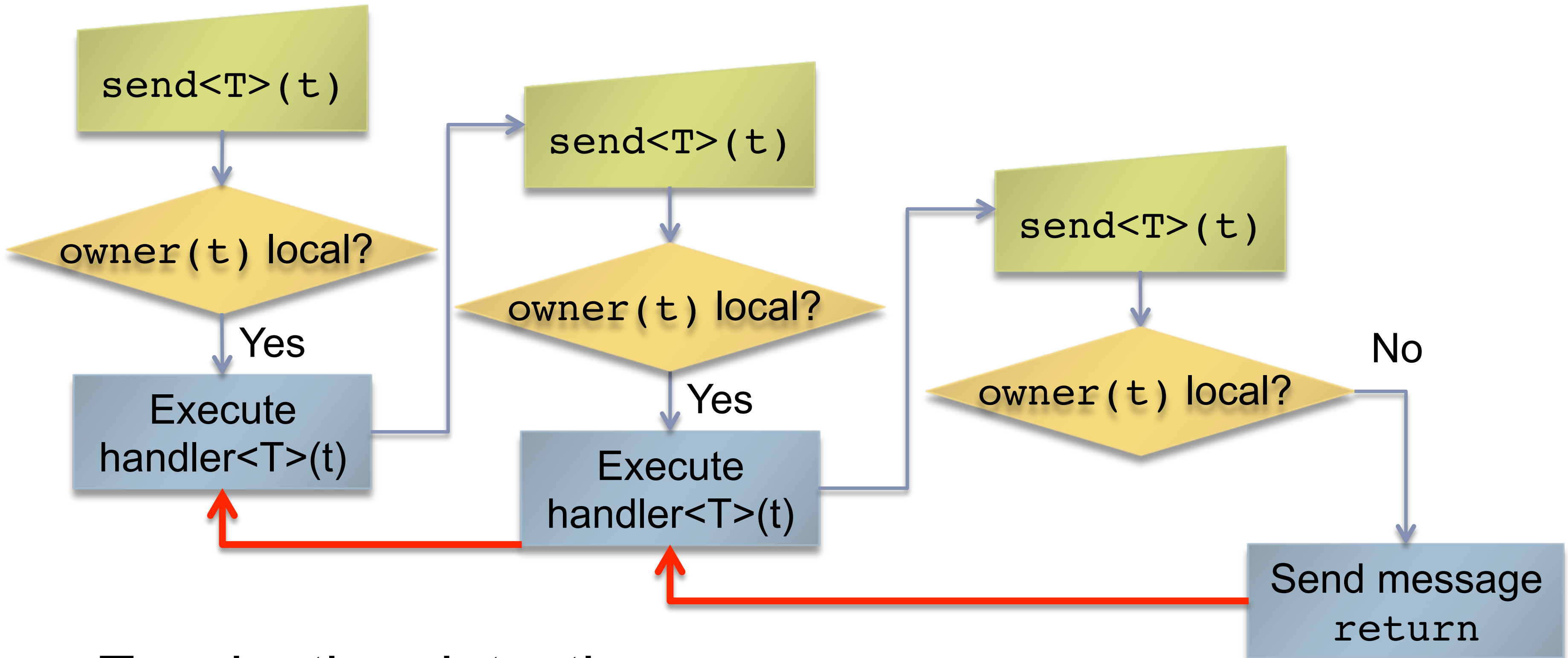
The AM++ Framework

- AM++ provides a “middle ground” between low- and high-level systems
  - Gets performance from low-level systems
  - Gets programmability from high-level systems
- High-level features can be built on top of AM++

Active Pebbles



Messages Can Send Messages



- Termination detection
  - Detect network quiescence
  - Pluggable

Active Pebbles

- Need to separate what the programmer expresses from what is actually executed
- A programming model **and** an execution model

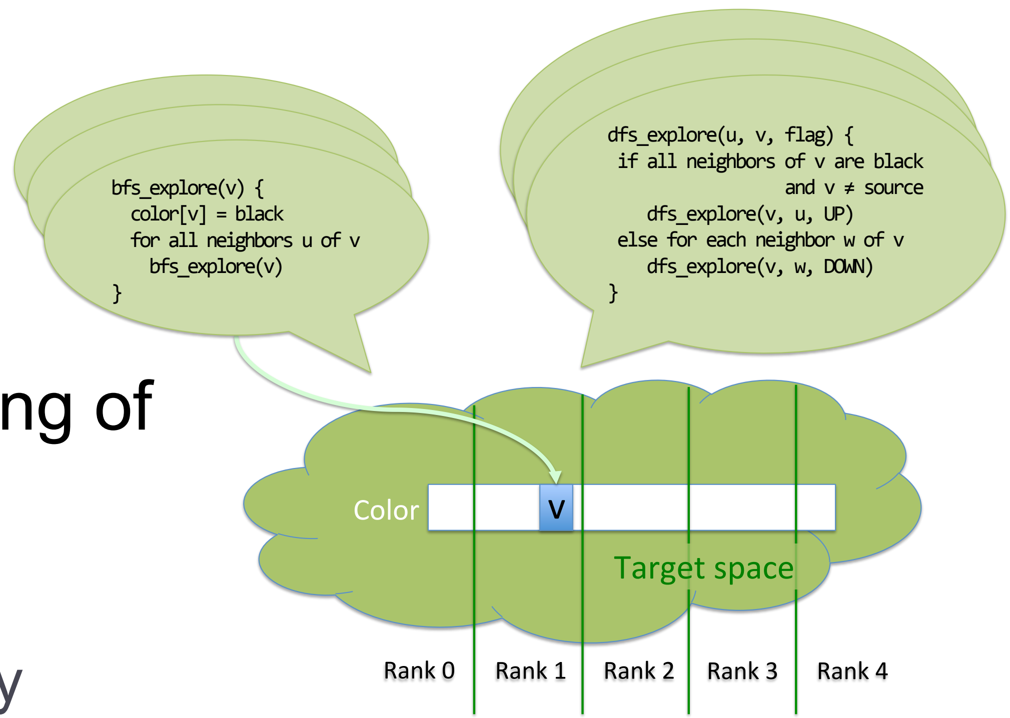
Active Pebbles Features

- Programming model
  - Active messages (*pebbles*)
  - Fine-grained addressing (*targets*)
- Execution model
  - Flexible message coalescing
  - Message reductions
  - Active routing
  - Termination detection

- Features are synergistic
- AM++ is our implementation of Active Pebbles model

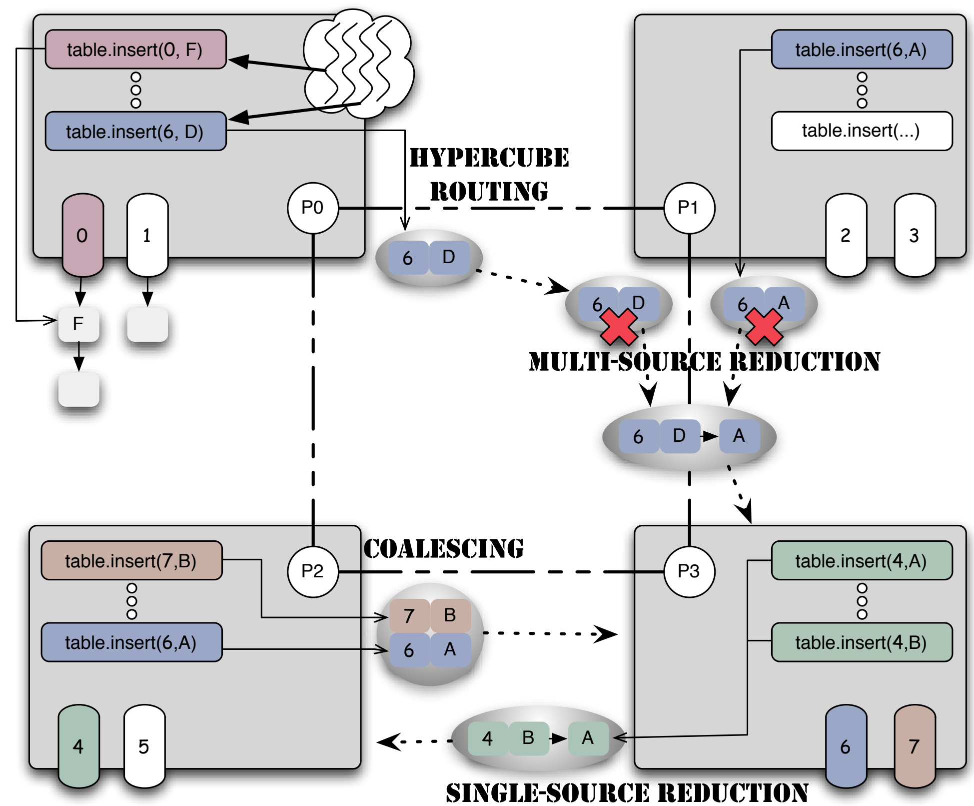
Programming Model

- Program with natural granularity
  - No need to artificially coarsen object granularity
- Transparent addressing
  - static and dynamic
  - local and remote
- Bulk, anonymous handling of messages and targets
- Epoch model
  - Enforce message delivery
  - Controlled relaxed consistency



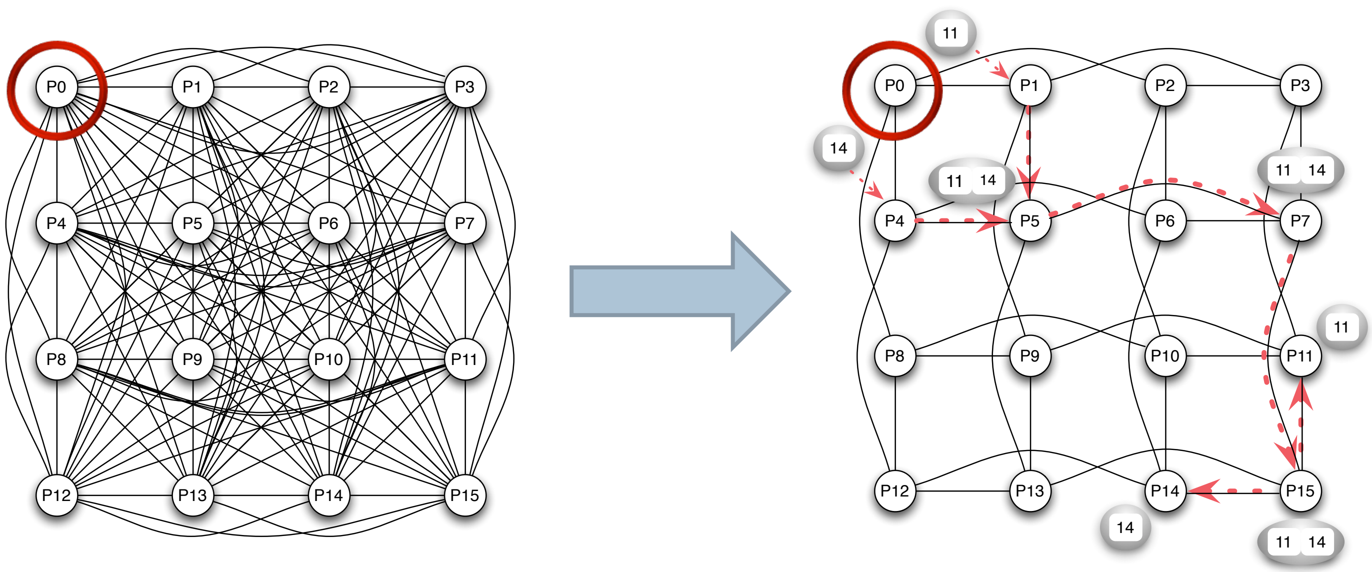
Execution Model

- Message coalescing
  - Amortize latency
- Message reduction
  - Eliminate redundant computation
  - Distributed computation into network
- Active routing
  - Exploit physical topology
- Termination detection
  - Handlers send messages
  - Detect quiescence



Routing + Message Coalescing

- Coalescing buffers limit scalability
  - Communications typically all-to-all
- Impose a limited topology with fewer neighbors
- Better scalability, higher latency



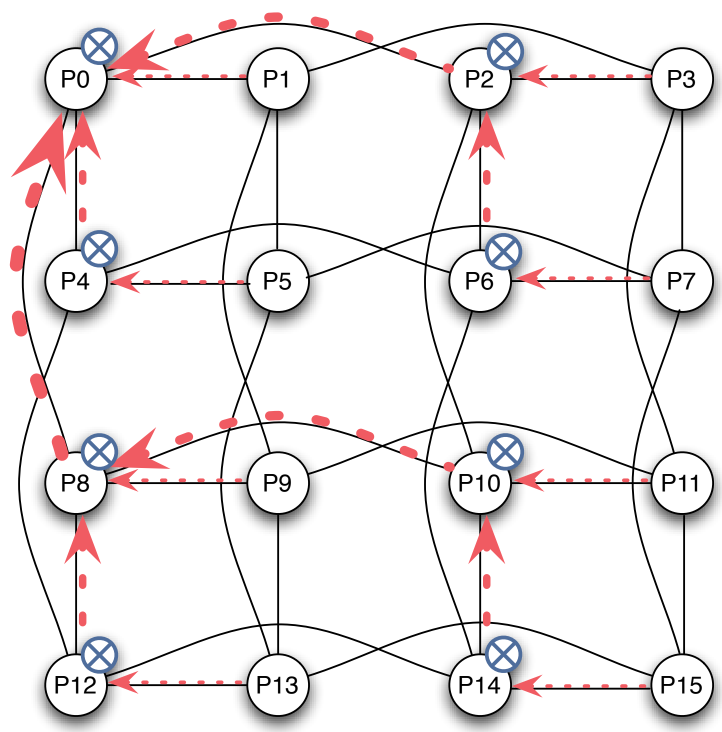
Multi-source coalescing

Message Reductions + Routing

- Messages to same target can often be combined
- Reductions application-specific, user-defined
- Routing allows cache hits at intermediate hops

Automatically synthesize optimized collectives

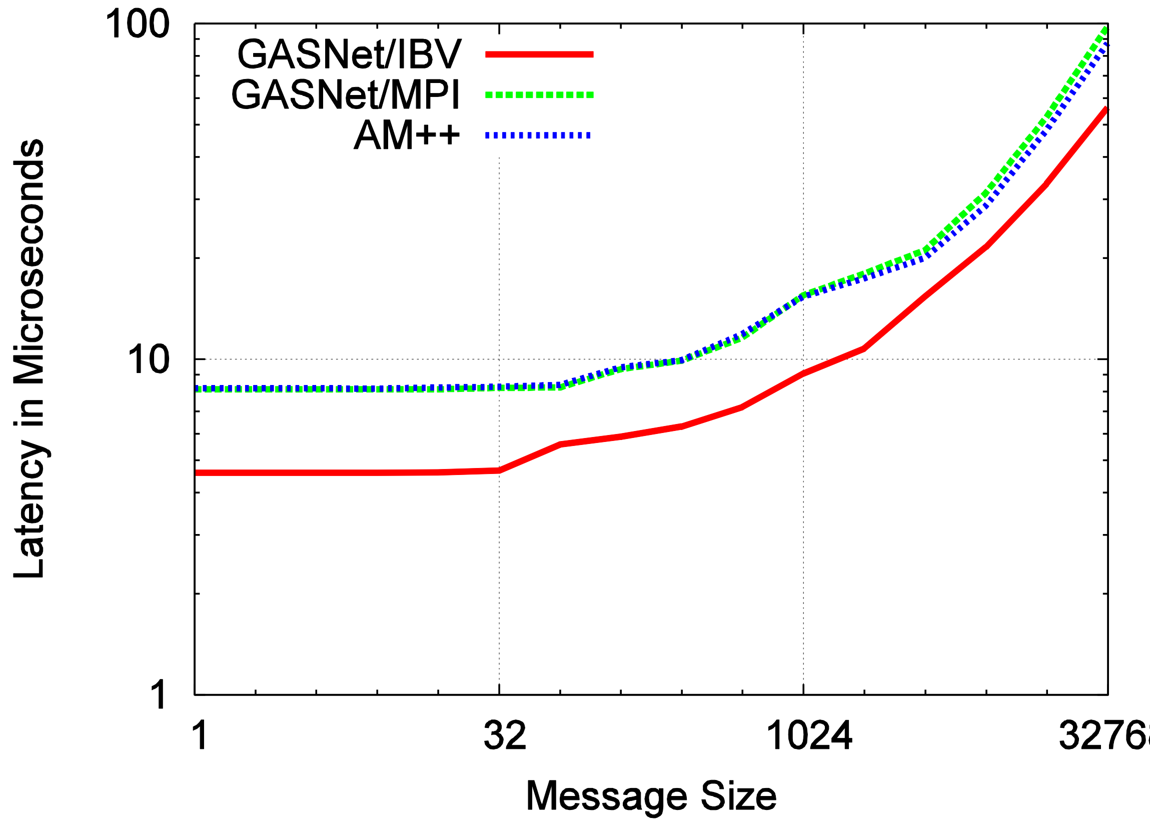
Distribute computation into the network



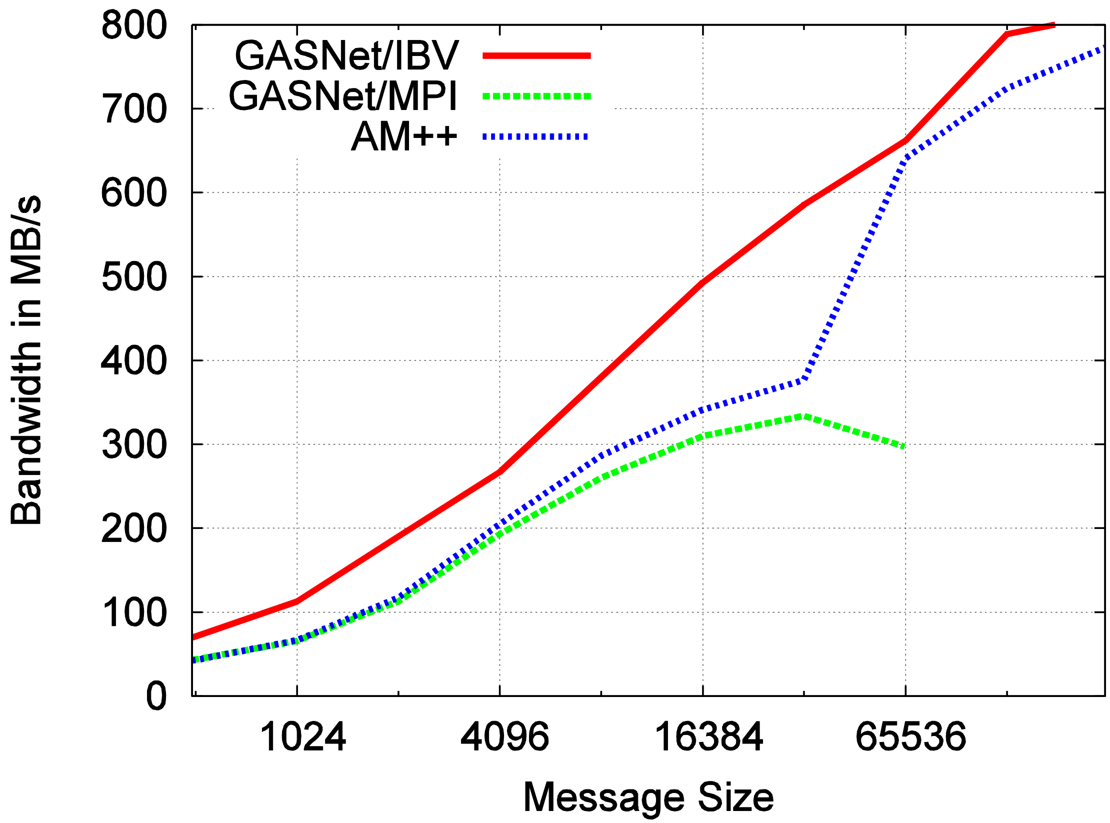
Active Message Abstraction

- *Pebbles* are agnostic as to where they execute, operate on *targets*
- Independent of how messages are processed
  - Network communication (MPI, GASNet, DCMF, IB Verbs…)
  - Work stealing (Cilk++, Task parallelism)
  - OpenMP (over coalescing buffers)
  - Immediate execution in caller (of send())
- Thread-safe metadata allows weakening message order
  - Updates to *targets* must be atomic
  - Algorithms may have to tolerate weak consistency

Evaluation: Message Latency



Evaluation: Message Bandwidth



Active Pebbles

- Meant to support all kinds of parallelism



- Started with optimizing distributed memory communication
- Same features allow integration of fine-grained parallelism

Active Messages for Work Decomposition

- Key idea is to find natural granularity
- Each *pebble* represents an independent computation that can be executed in parallel

$\text{ENQUEUE}(Q, s)$
while $(Q \neq \emptyset)$
  $\boxed{u \leftarrow \text{DEQUEUE(Q)}}$
  for $(\text{each } v \in Adj[u])$
    if $(color[v] = \text{WHITE})$
      $color[v] \leftarrow \text{GRAY}$
      $\boxed{\text{ENQUEUE}(Q,\ v)}$
    else $color[u] \leftarrow \text{BLACK}$

```
#pragma omp parallel for
```
for$(\text{each } v \in Q)$

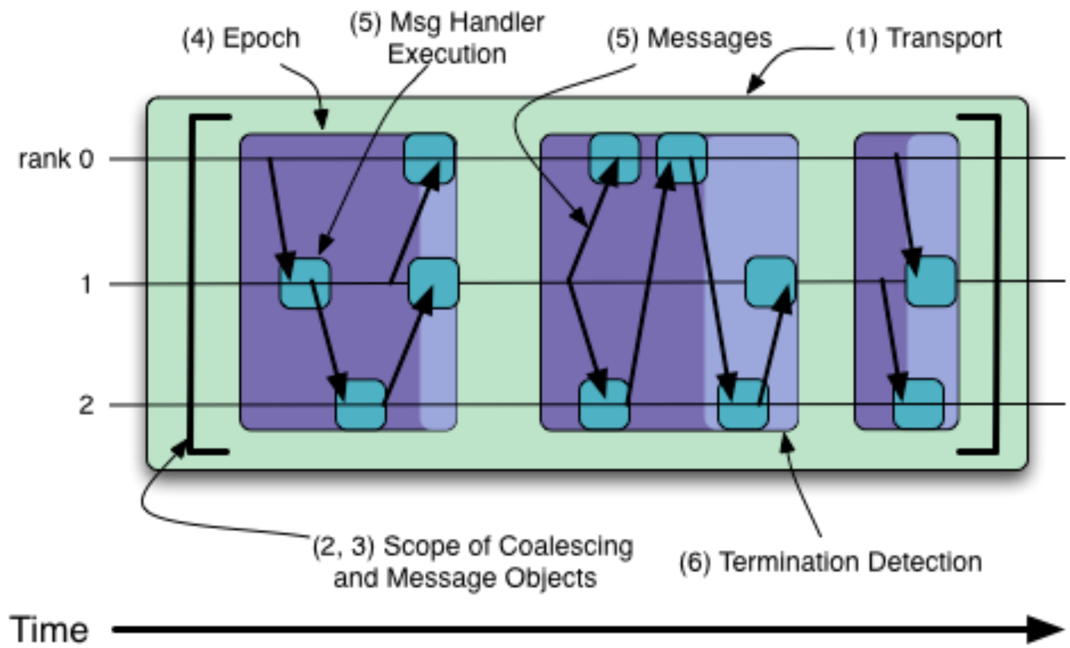
$\text{queue\_push\_msg} \rightarrow \text{send}()$

"Inner-Loop" Parallelism

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

What’s Thread-safe in AP?

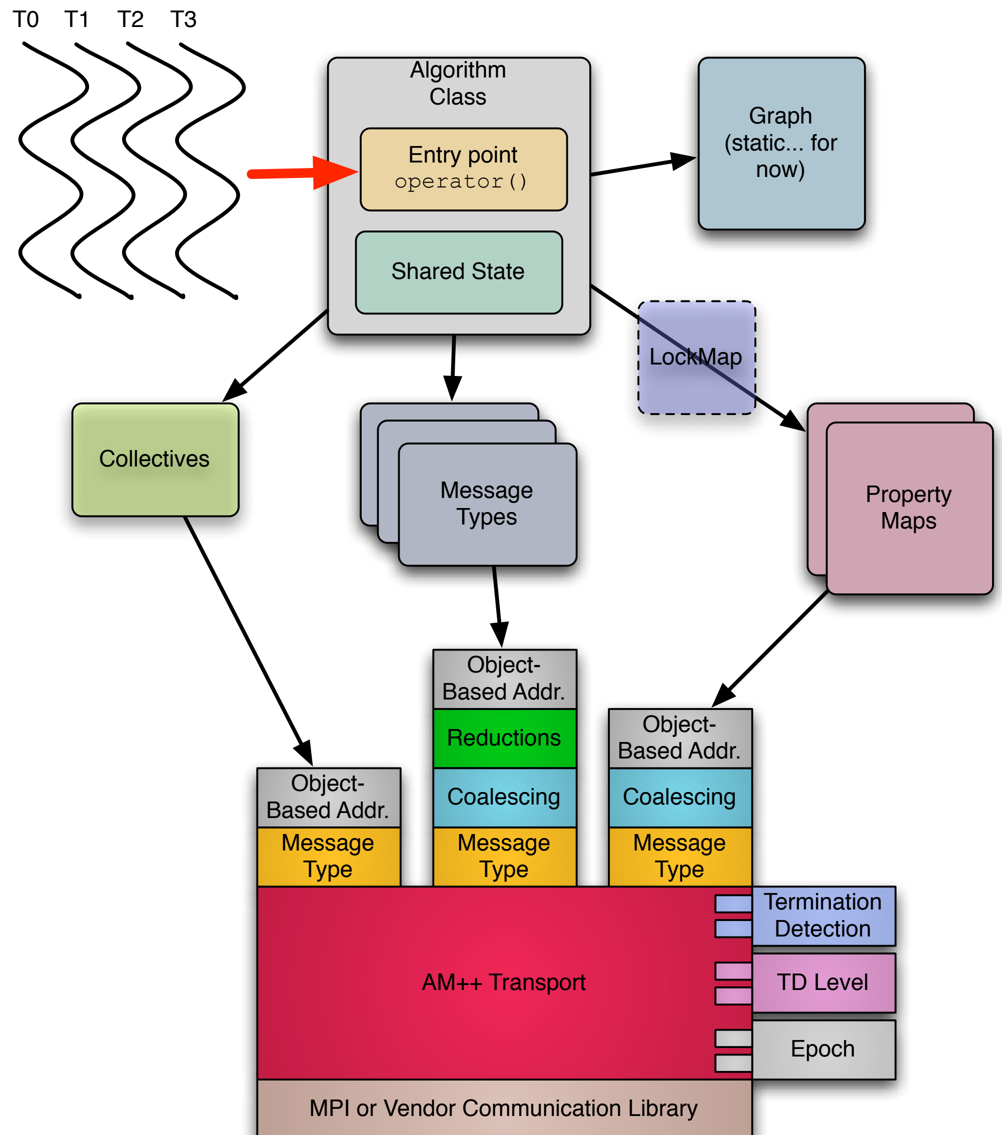
- Messaging
- Epoch begin/end
- Termination detection



- **NOT**:
  - Message creation
  - Modifying message features: routing, coalescing, reductions
  - Modifying termination detection
  - Modifying the number of threads

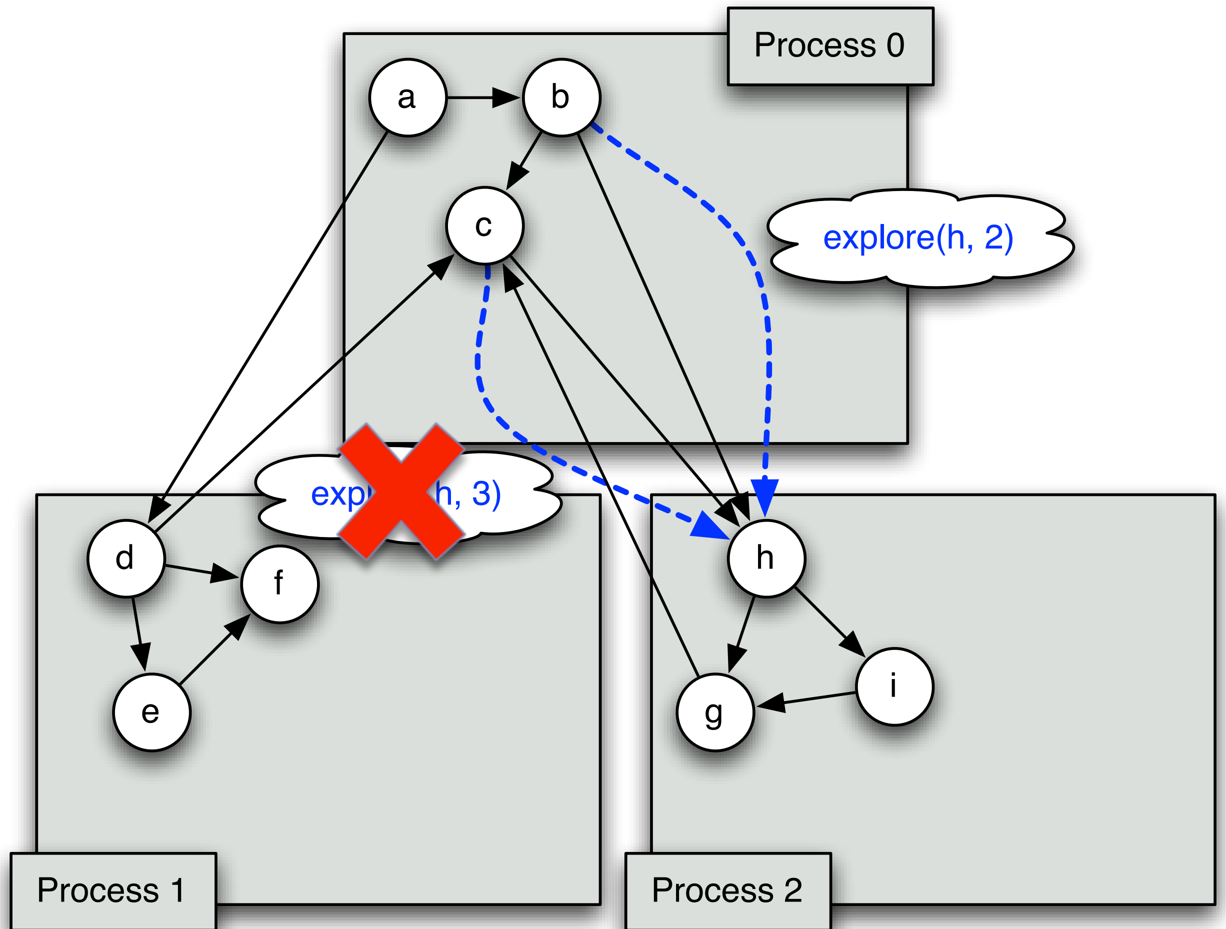
Parallel BGL Architecture

- AP makes messaging thread-safe.
- Property maps make metadata manipulation thread-safe and allow messages to be processed in arbitrary order
- *Just as transactional memory generalizes processor atomics to arbitrary transactions, Active Pebbles generalizes one-sided operations to user-defined handlers*

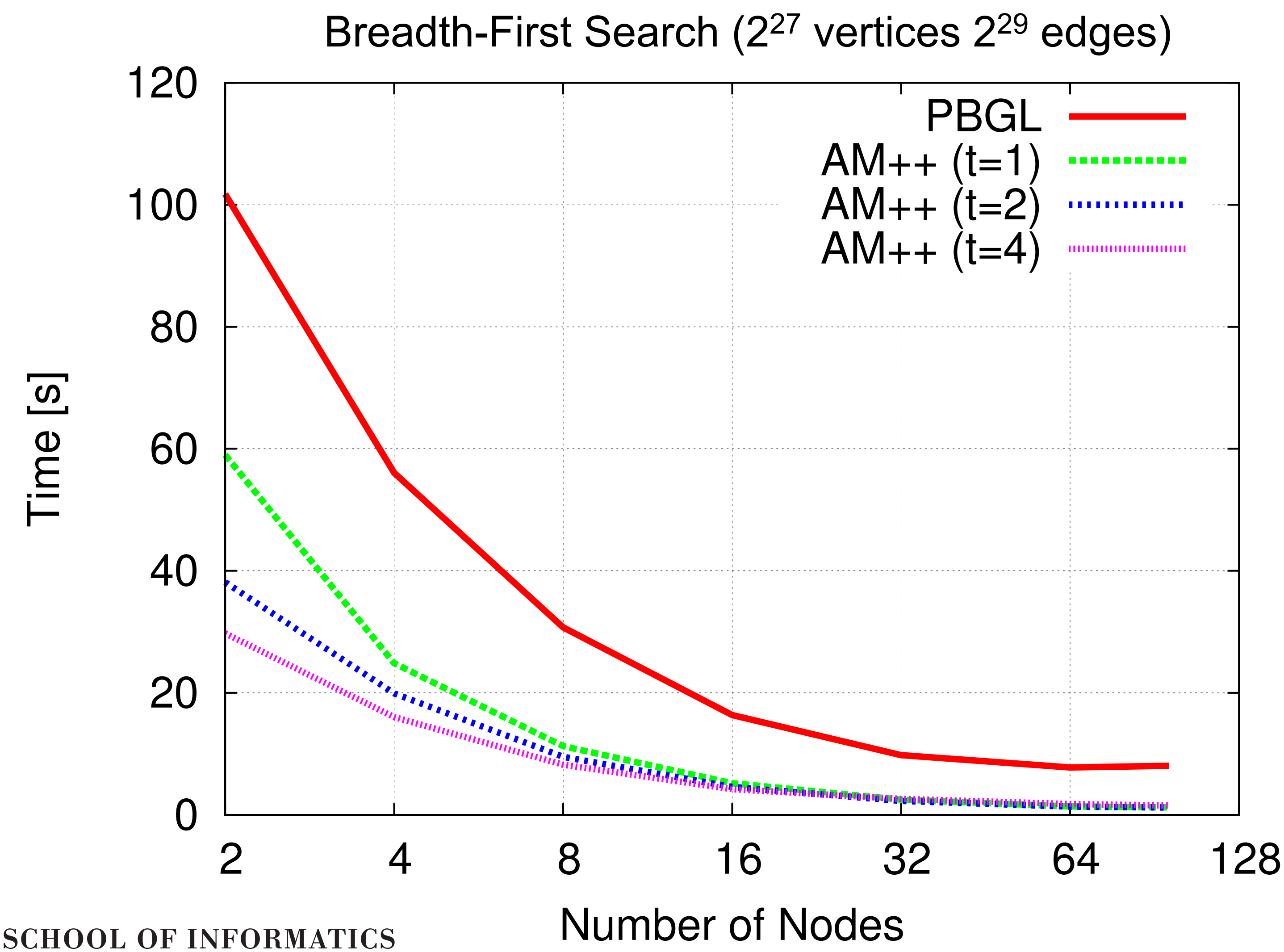


SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Active Pebbles Breadth-First Search



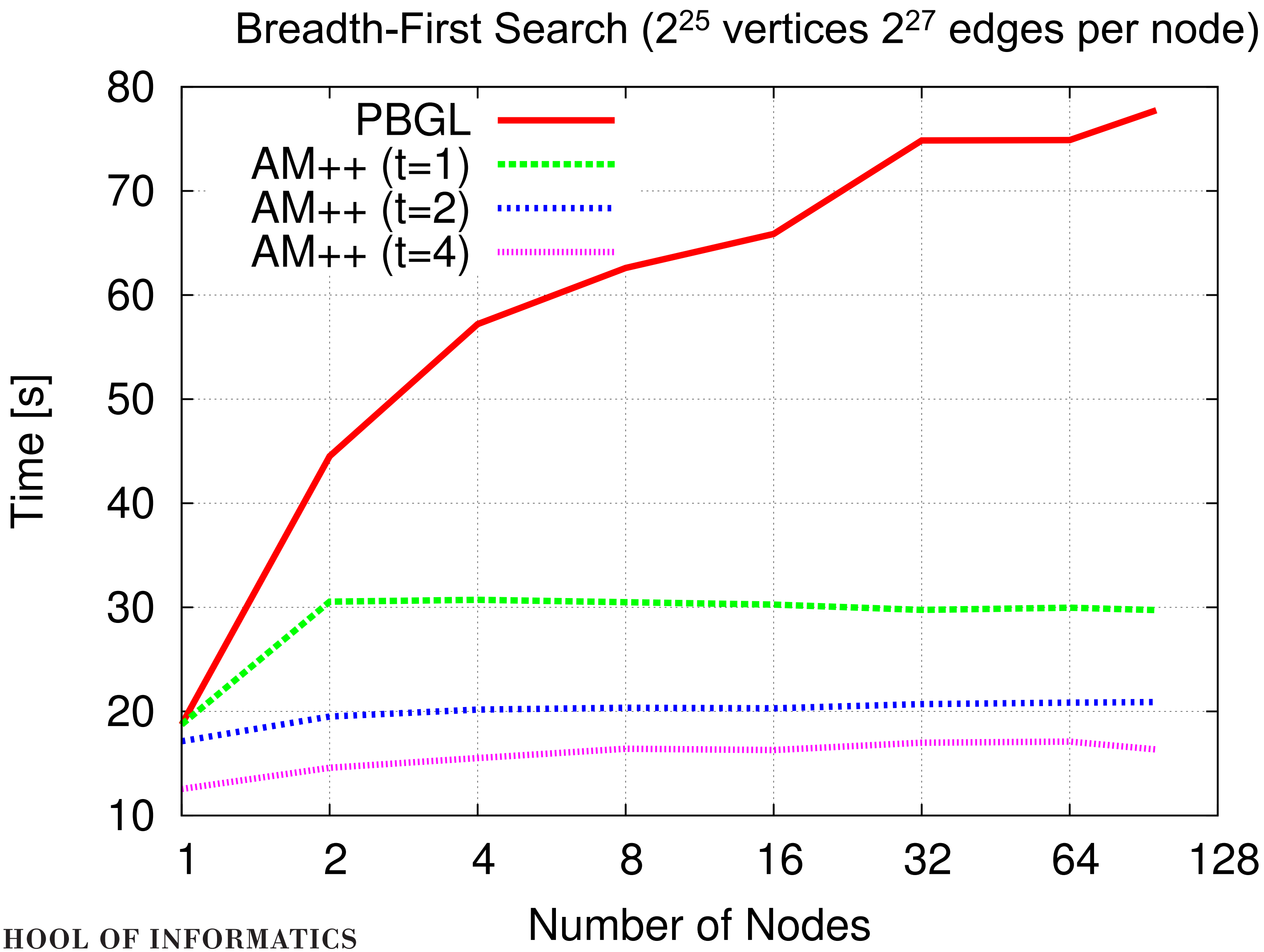
BFS: Strong Scaling



Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

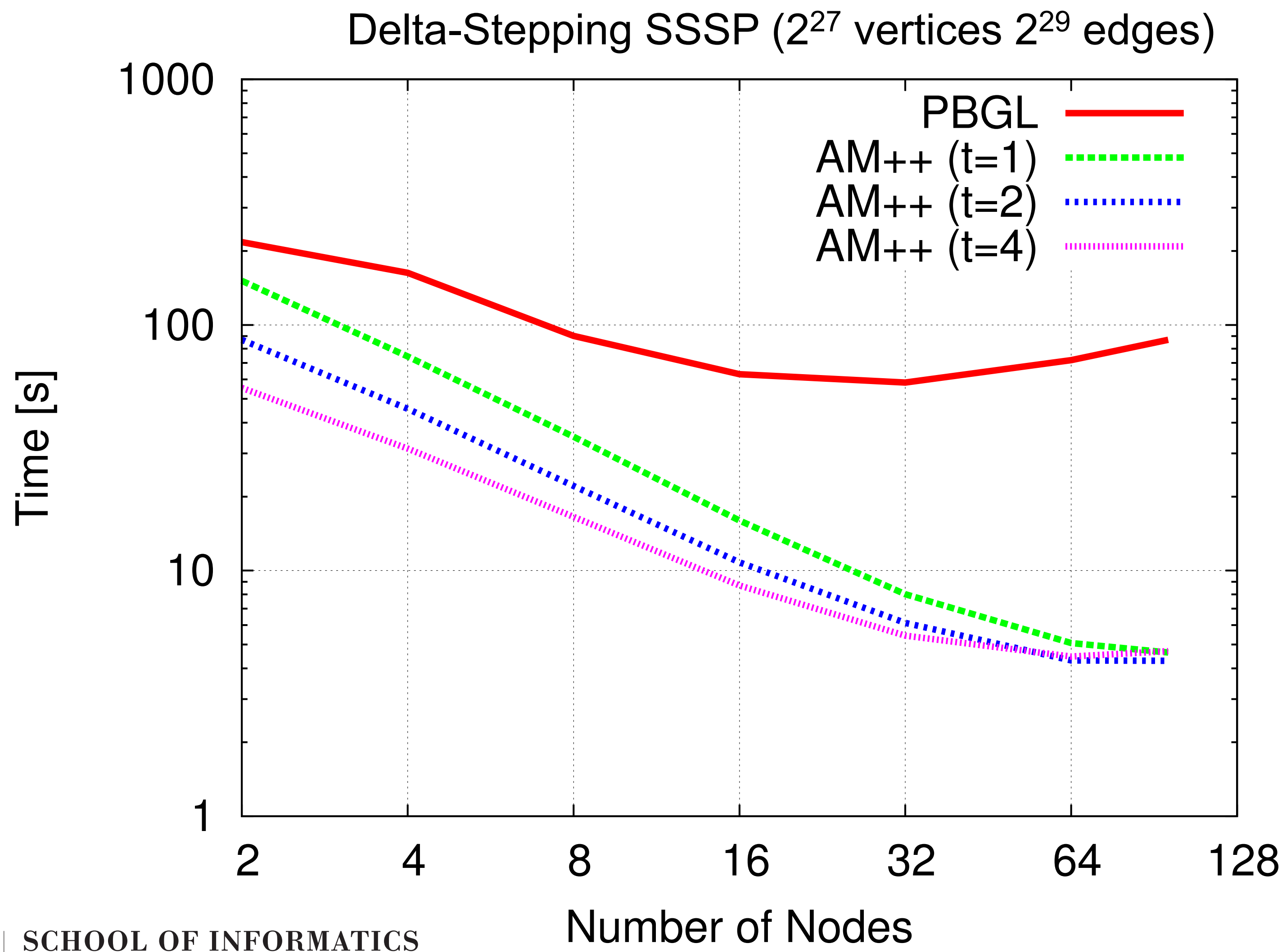
SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

BFS: Weak Scaling



Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

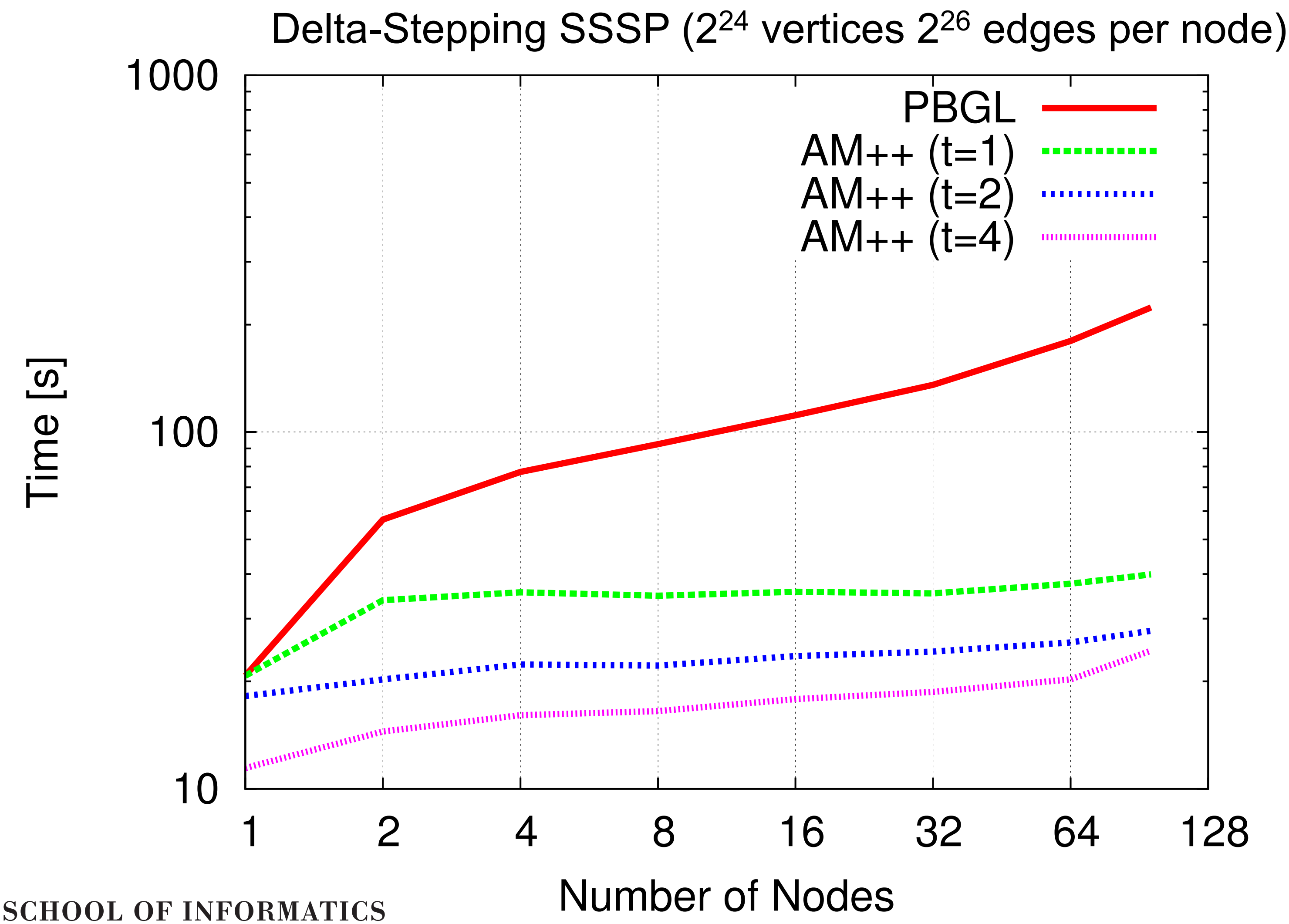
Delta-Stepping: Strong Scaling



SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

Delta-Stepping: Weak Scaling

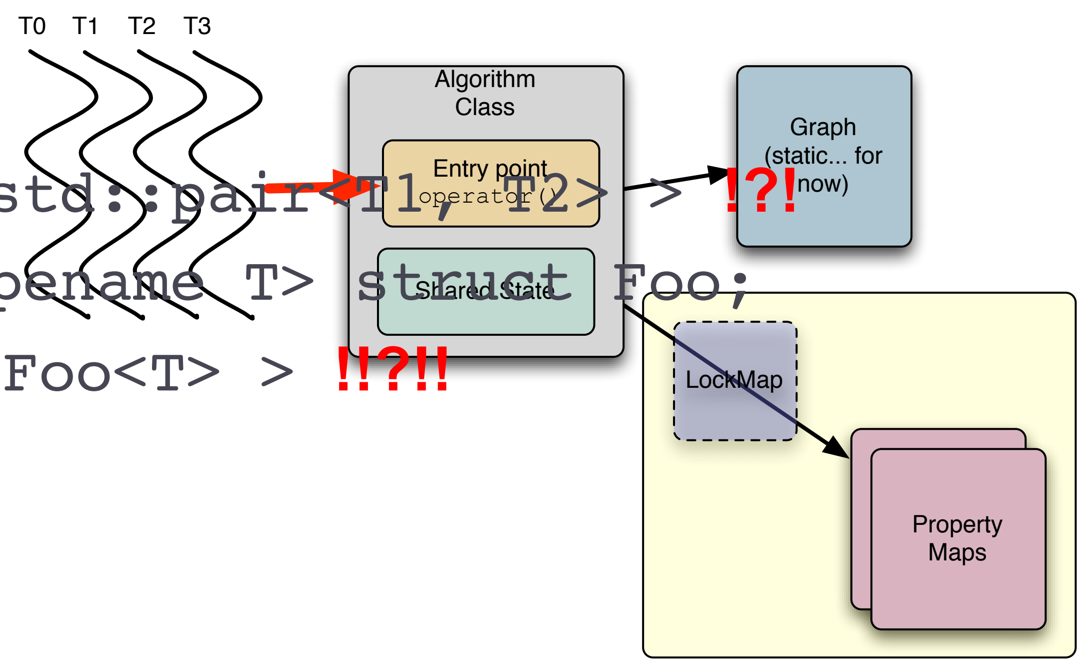


SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington

Results were run on Erdős-Renyí graphs using a cluster of 128 2.0Ghz Opteron 270 processors with two cores and 8GB of PC2700 DDR-DRAM per node connected via SDR Infiniband.

Transactional Metadata updates

- Sometimes we want to update non-contiguous, dependent data atomically
- Predecessor **and** BFS level
  - Or arbitrary visitor code supplied by users
- Limited-scope transactions, are these any easier?




```
• std::atomic<std::pair<T1, T2> >  !?!
• template <typename T> struct Foo;
  std::atomic<Foo<T> > !!?!!
```

Summary

- Active Messages
  - Express fine-grained, asynchronous operations elegantly
  - Well-matched to data-driven problems
  - Enable fine-grained parallelism (threads, GPUs, FPGAs, …)
  - Asynchrony allows latency hiding
- Concise expression **and** efficient execution
  - Separate programming and execution models
- Active Pebbles
  - Simple programming model
  - Execution model maps programs to hardware efficiently
    - AM++ is our implementation
    - Could be targeted as a runtime by languages (X10, Chapel, …)

Future Work

- Constrained parallelism in shared memory
  - Not entirely work queue-based
  - Not entirely recursion-based
- Acceleration
  - Coalesced message buffers offer great opportunities here… if getting to the accelerator is cheap
- Optimizing local memory transactions
  - Intel's TSX in *Haswell* CPUs
  - Declarative language for transaction code generation

Questions?

- More info on Active Pebbles
  - Jeremiah Willcock, Torsten Hoefler, Nicholas Edmonds, and Andrew Lumsdaine. **Active Pebbles: Parallel Programming for Data-Driven Applications**. ICS '11.
- More info on AM++
  - Jeremiah Willcock, Torsten Hoefler, Nicholas Edmonds, and Andrew Lumsdaine. **AM++: A Generalized Active Message Framework**. PACT '10.
- More info on the Parallel Boost Graph Library and graph applications:
  - http://www.osl.iu.edu/research/pbgl
  - http://www.boost.org
  - Watch for a new release of PBGL based on Active Pebbles, running on AM++ soon!

(Ask if you'd like access to a pre-release, *very* alpha copy)

ngedmond@cs.indiana.edu

SCHOOL OF INFORMATICS AND COMPUTING
INDIANA UNIVERSITY
Bloomington