May 14, 2009

A Scalable Multicast Scheme for Distributed DAG Scheduling

Fengguang Song[1], Jack Dongarra[1,2], Shirley Moore[1]

University of Tennessee[1]

Oak Ridge National Laboratory[2]

Scheduling for Large-Scale Systems Workshop

Knoxville, May 13-15, 2009

INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY of TENNESSEE

Department of Electrical Engineering and Computer Science

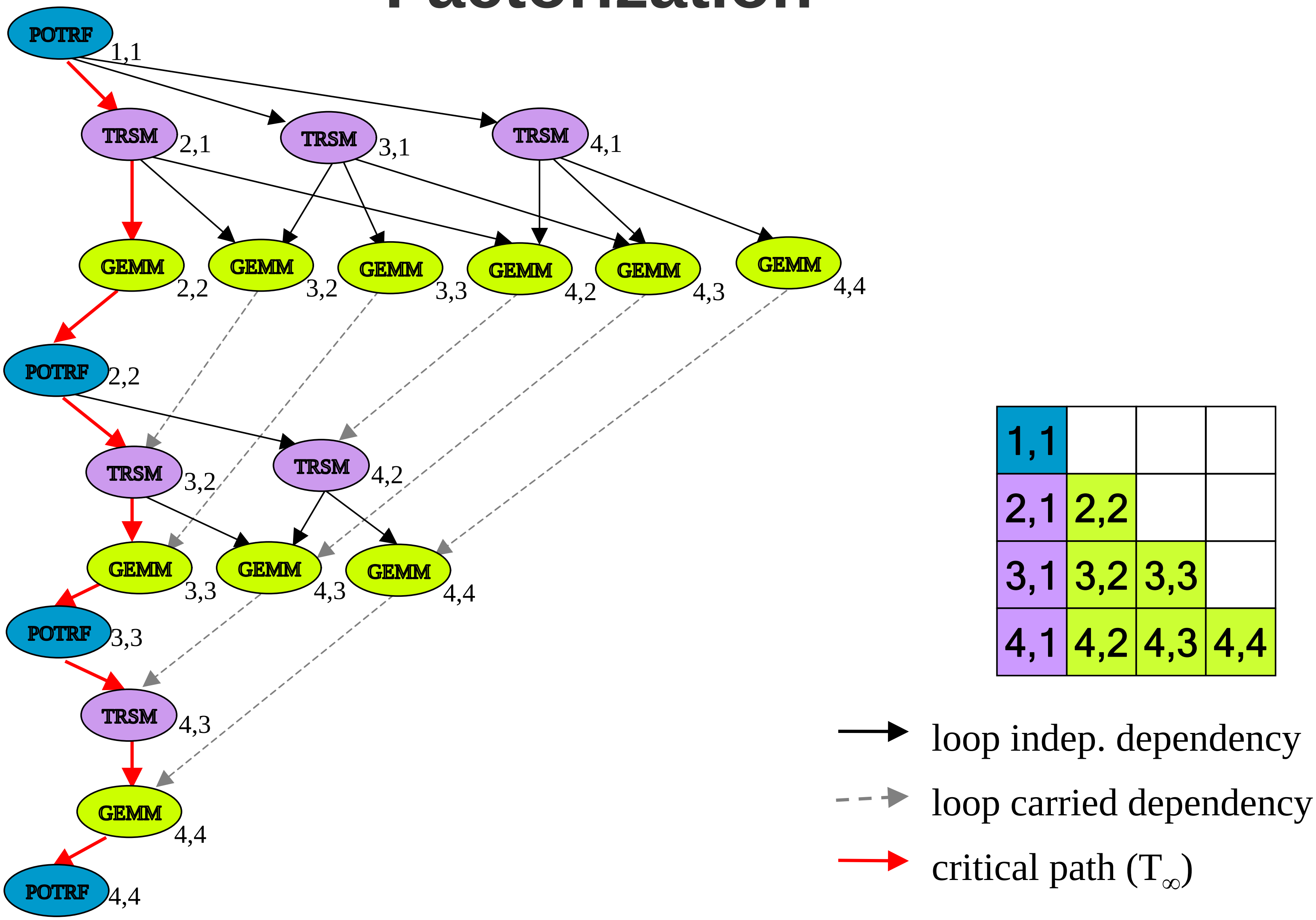
Contents

- Motivation
- Background: a programming model for DAG scheduling
- Overview of the multicast scheme
- Topology ID
- Compact routing tables
- Multicast examples
- Experimental results

Motivation

- High performance on multicore machines
- New software should have two characteristics:
  - Fine grain threads
  - Asynchronous execution
- We want to use dynamic DAG scheduling
- Extremely Scalable
  - We are thinking of millions of processing cores.
  - Distributed-memory

A DAG Example for Cholesky Factorization



Simple Programming Model

- Symbolic DAG interface:
  - int get_num_parents(const Task t);
  - int get_children(const Task t, Task children);
  - set_entry_task(const Task t);
  - set_exit_task(const Task t);

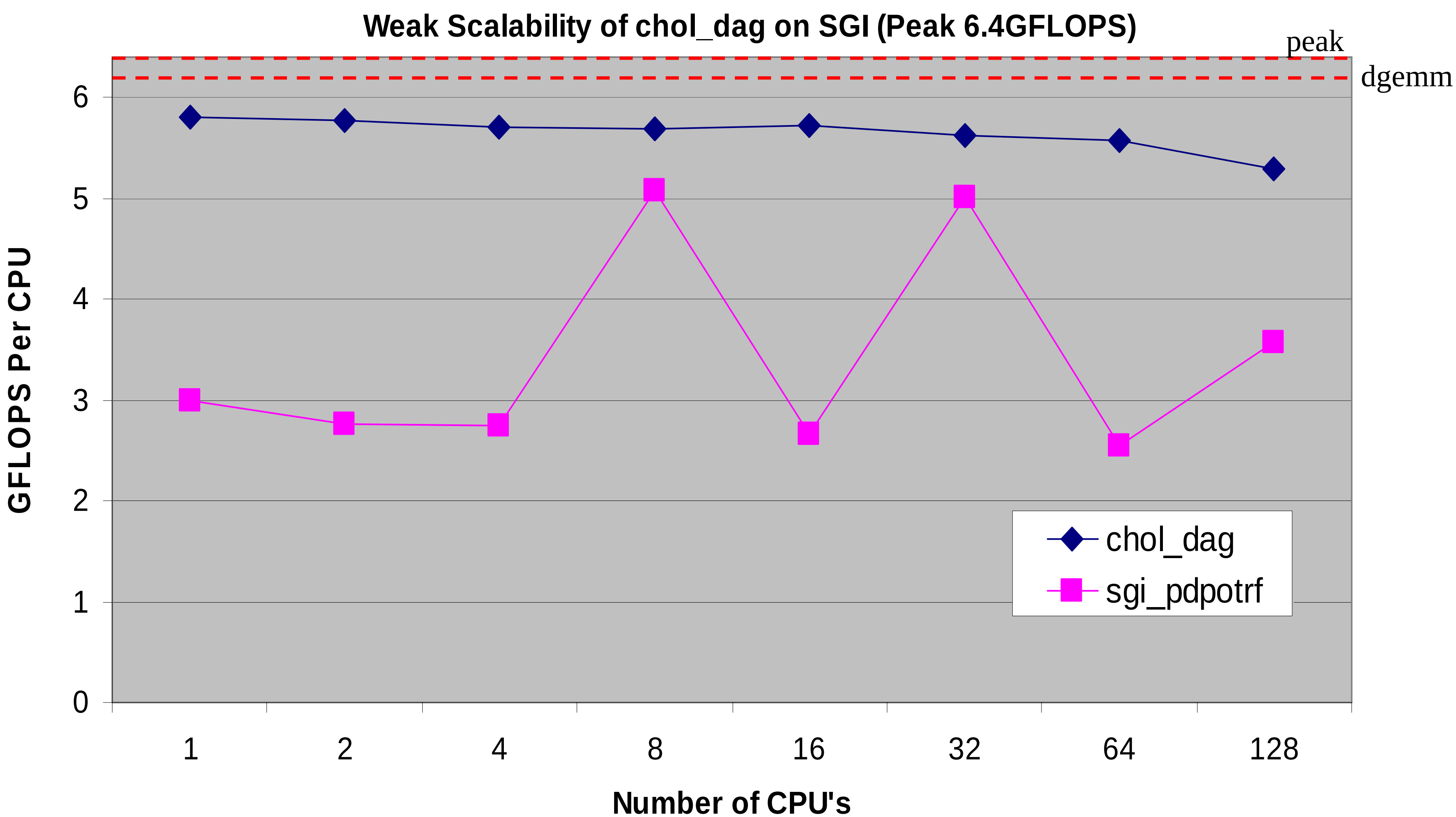
Interface Definition for Cholesky Factorization

```
struct Task {
    int type; // what task
    int k;      // iteration index
    int i, j;    // row, column index
    int priority;
}
```

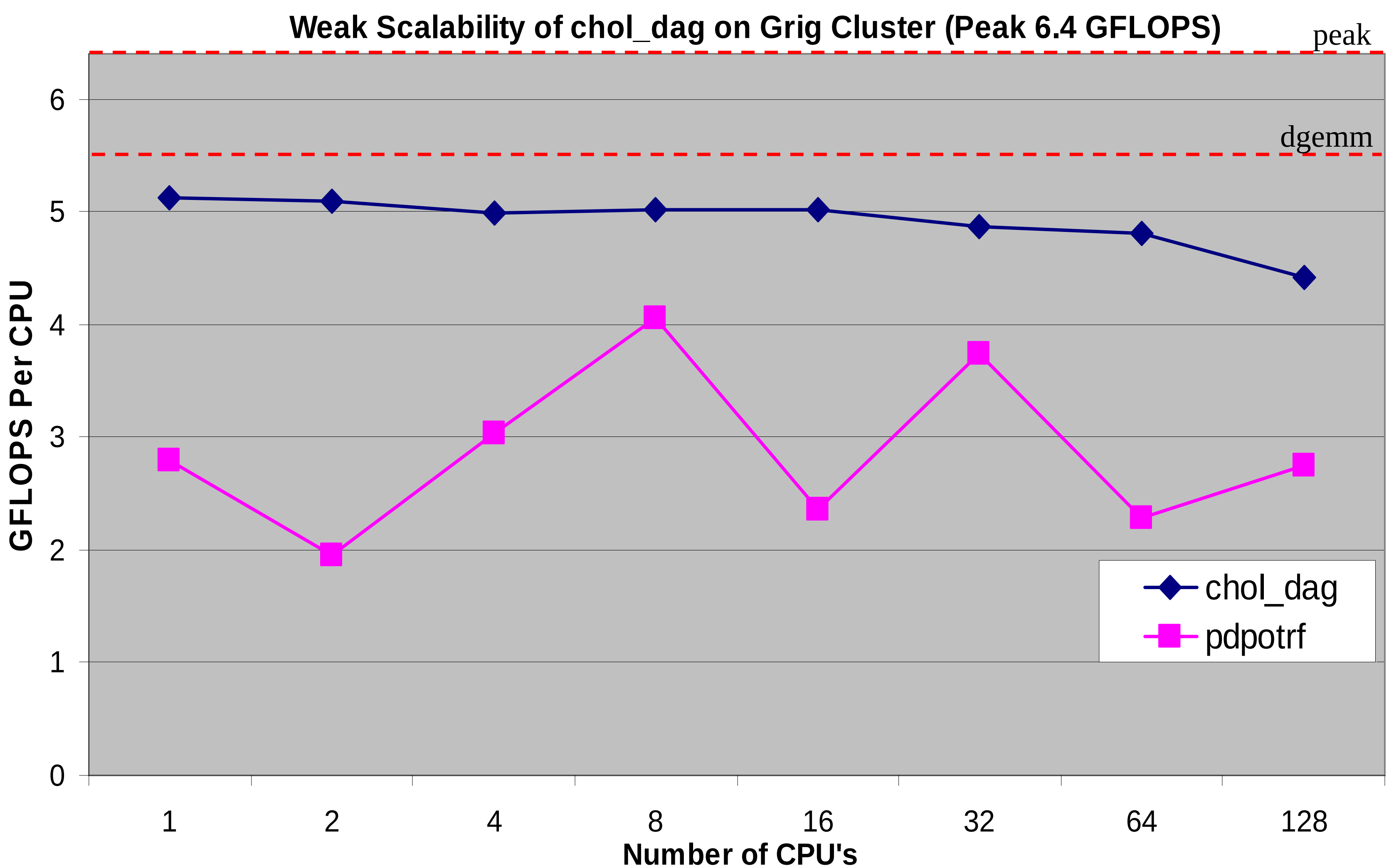
```
int get_num_parents(Task t) {
   if (t.type = POTRF) return 1
   if (t.type = TRSM)  return 2
   if (t.type = GEMM) {
      if (diagonal) return 2
      else return 3
   }
}
```

```
int get_children(Task p, Task* buf, int nblks) {
   if (p.type = POTRF) {
     /* along p's column but below p */
     buf := {TRSM task t | t.j = p.j & t.i ∈( p.i, nblks]}
   }
   if (p.type = TRSM) {
     /* a row and a column (both with index p.i) */
     buf := {GEMM task t | t.i = p.i & t.j ∈(p.j, p.i] or
                                   t.j = p.i & t.i ∈ [p.i, nblks] }
   }
   if (p.type = GEMM) {
     /* has a single child */
     if (diagonal) buf := a POTRF task
     else if (below diag) buf := a TRSM task
     else buf := a GEMM task
   }
   return |buf|
}
```

Performance on SGI Altix 3700 BX2

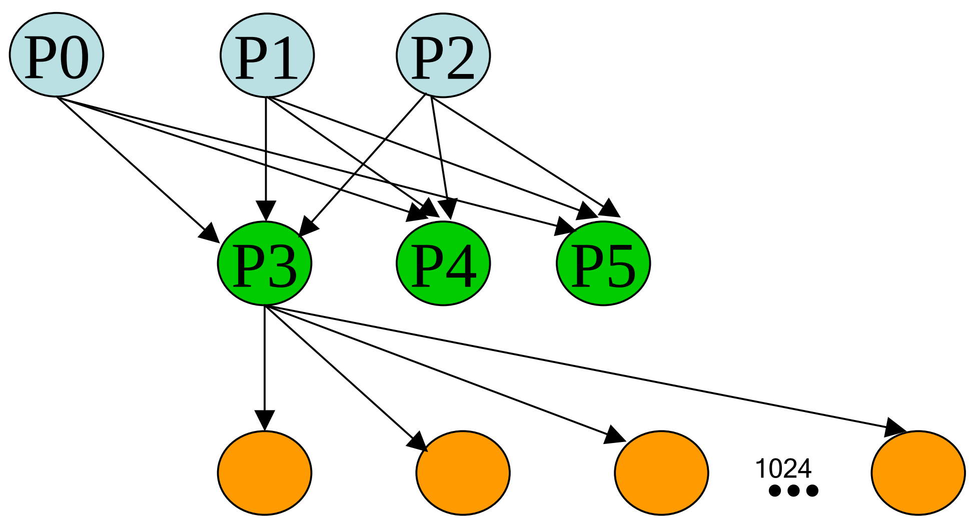


Performance on the Grig Cluster



The Multicast Problem

- Problem: a set of processes are executing a DAG where multiple sources notify different groups simultaneously.



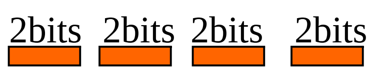
Multicast Scheme Overview

- Application-level routing layer
- Hierarchical abstraction of a system
- Each process has a topology ID.
  - Like zip code
  - The longer the common prefix of two topo_ids, the closer they are.
- Compact routing table
  - An extension to Plaxton's neighbor table [1]

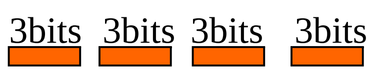
[1] Plaxton, C. G., Rajaraman, R., and Richa, A. W. 1997. Accessing nearby copies of replicated objects in a distributed environment. SPAA '97.

Topology ID

- Assign IDs to the whole system (i.e., T_{system})
  - $T_{program}$ of a user program $\subset T_{system}$
- A topology ID is a number of digits.
  - E.g., 256 nodes consist of 4 digits with base 4.

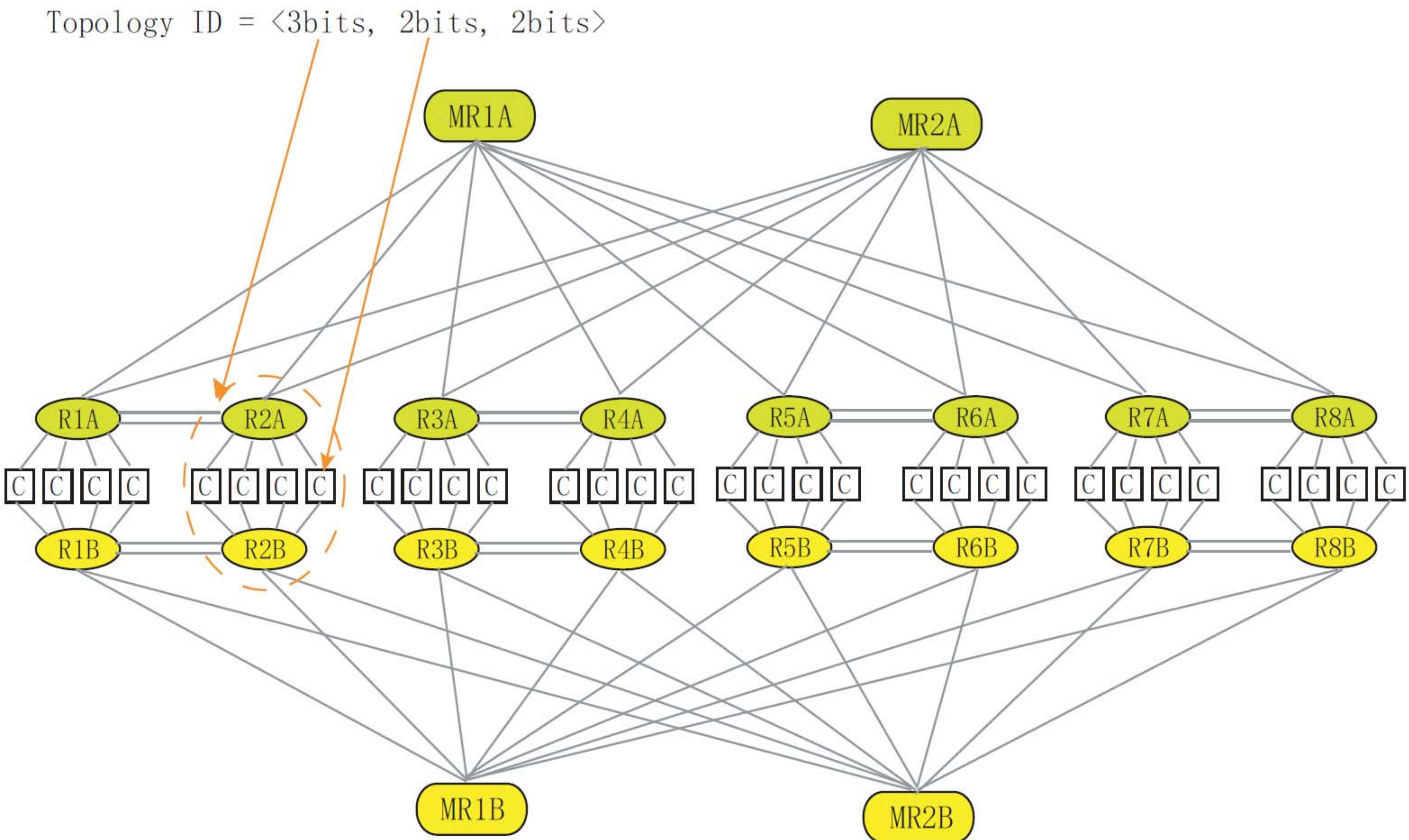


  - E.g., 2048 nodes consist of 4 digits with base 8.



- We assume that two nodes with a longer common prefix are closer on the physical network.

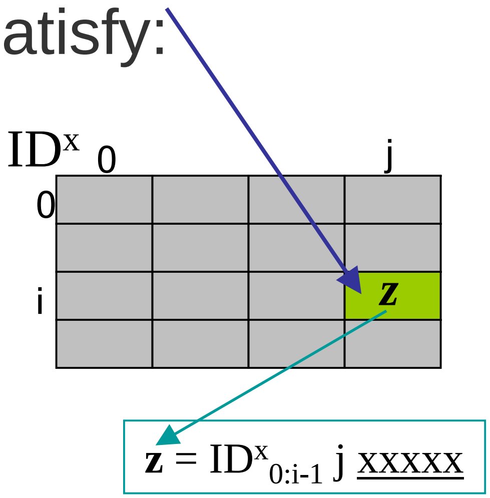
Topology ID Example



SGI Altix 3700 System

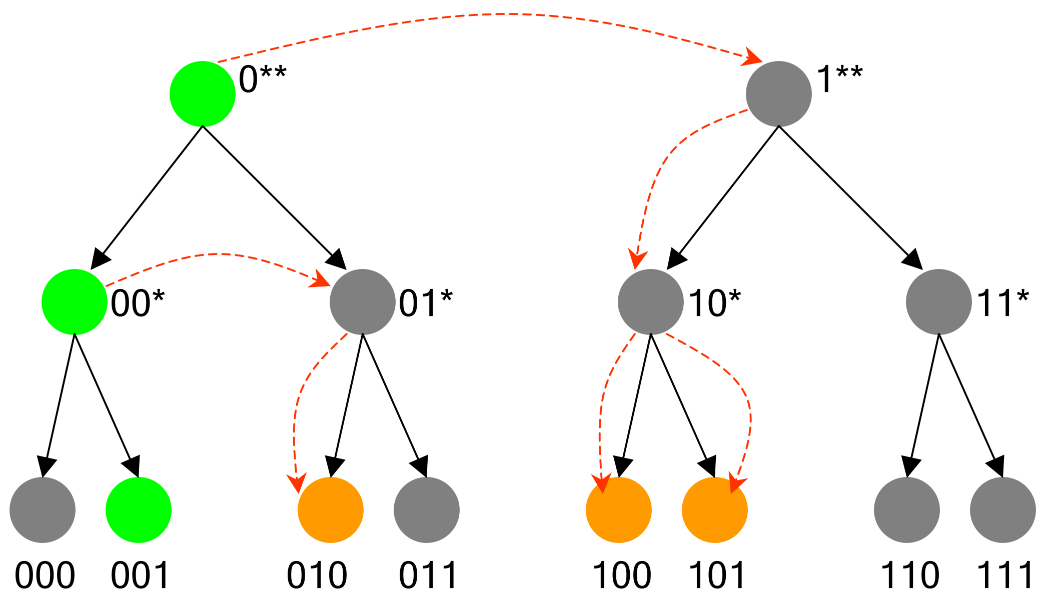
Compact Routing Table

- Suppose process *x* has a routing table and Table[i,j] stores process *z*, then ID^x and ID^z must satisfy:
  - $ID^x_0 ID^x_1 \ldots ID^x_{i-1} = ID^z_0 ID^z_1 \ldots ID^z_{i-1}$,
  - $ID^z_i = j$ (i.e., (i+1)th digit of $ID^z = j$).
- Routing table could have empty entries.
- Always search for the forwarding host on the LCP(*x*,*y*) row.
- At most $(\log_2 P)/(\text{base})$ steps
- O(lgP) space cost
  - 1 million cores → 80(5x16) entries
  - 1 billion cores → 192(6x32) entries



A Multicast Example

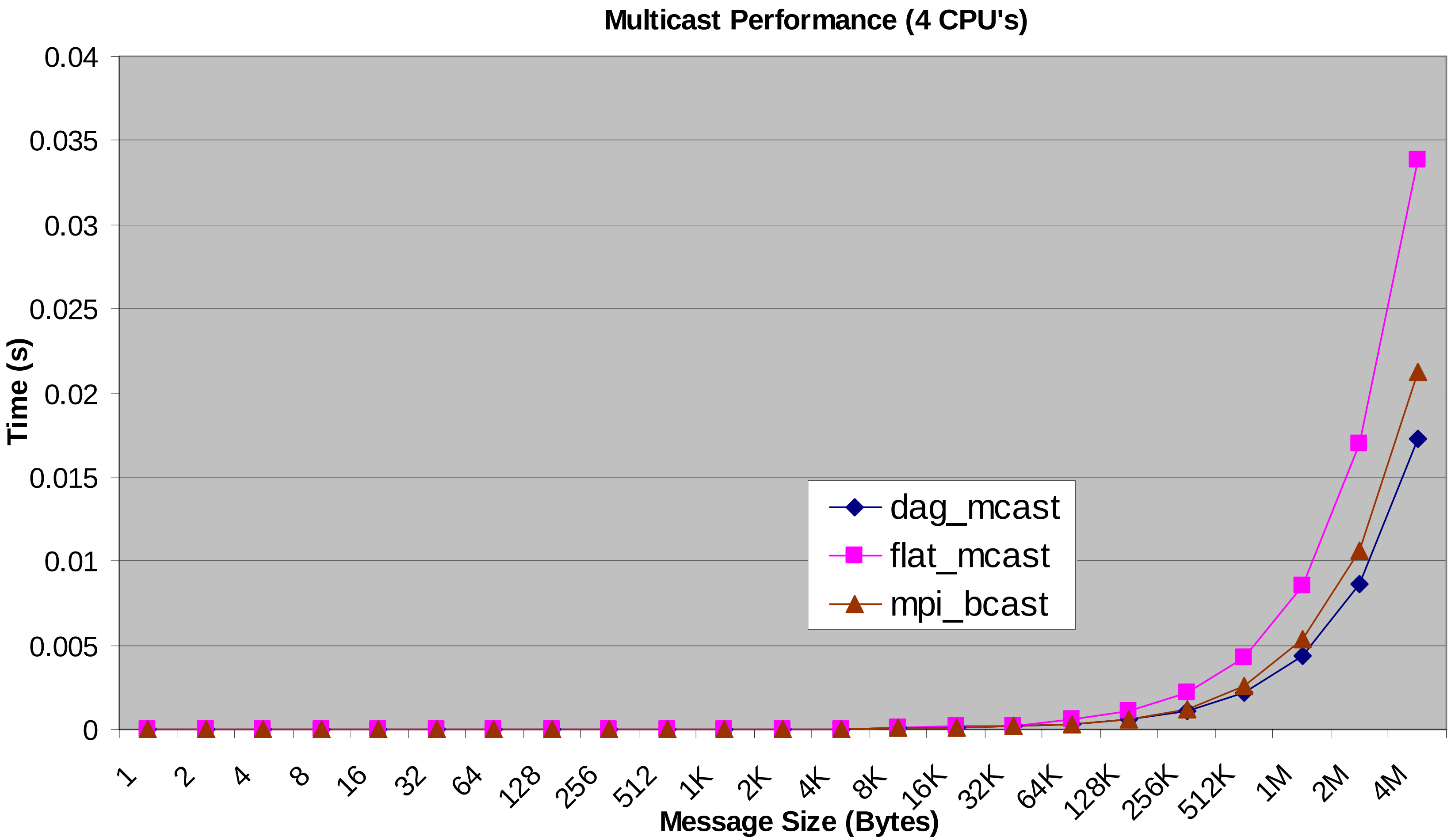
- Node 001 multicasts data to nodes {010, 100, 101}.



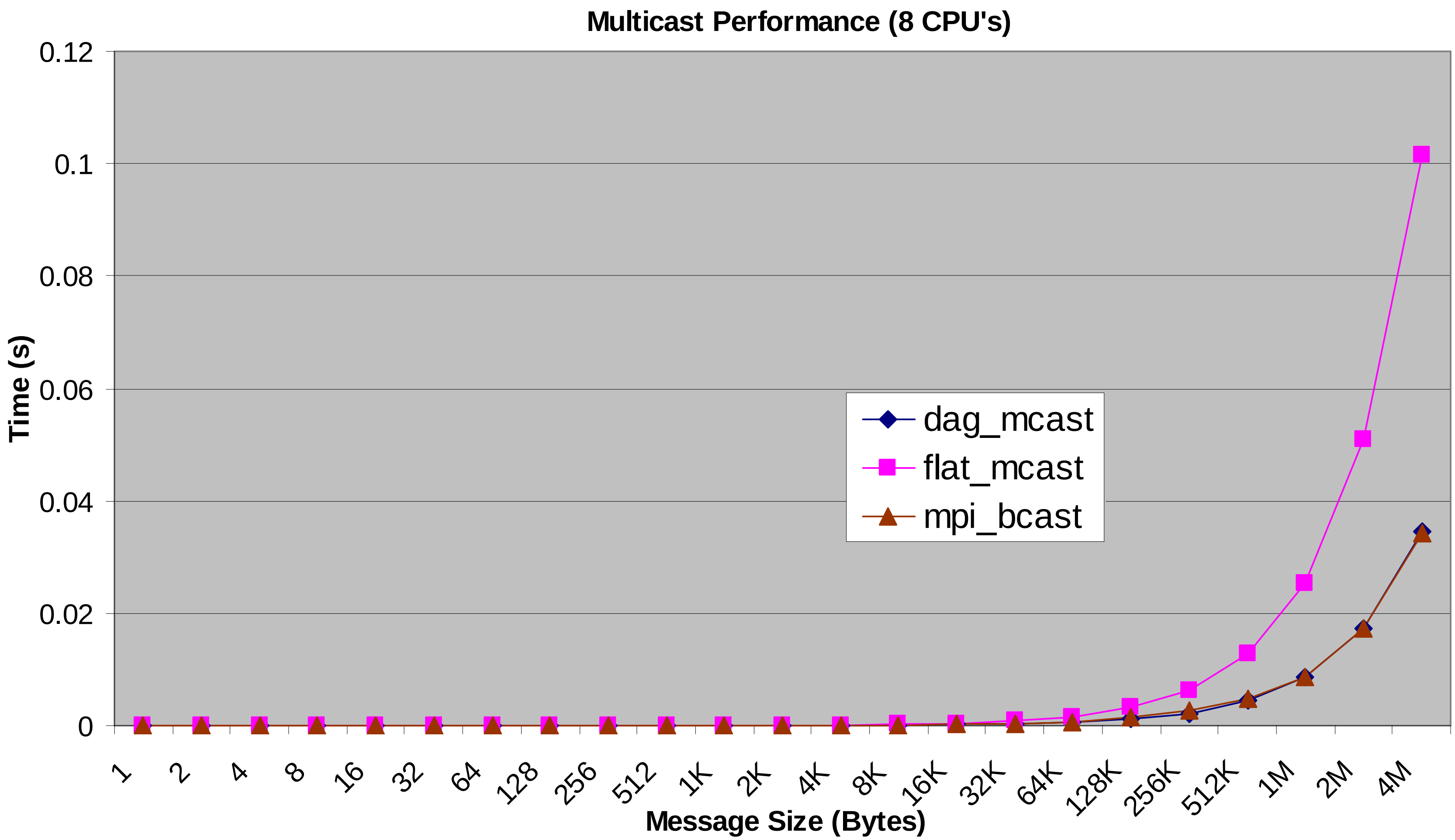
Grig Cluster

- grig.sinrg.cs.utk.edu
- 64 nodes, dual-CPU per node
  - Intel Xeon 3.20GHz
  - Peak performance 6.4 GFLOPS
  - **Myrinet interconnection** (MX 1.0.0)
- Goto BLAS 1.26
  - DGEMM performance 5.57 GFLOPS
  - 87% of peak performance (upper bound)
- **MPICH-MX 1.x**
- gcc 64 bits

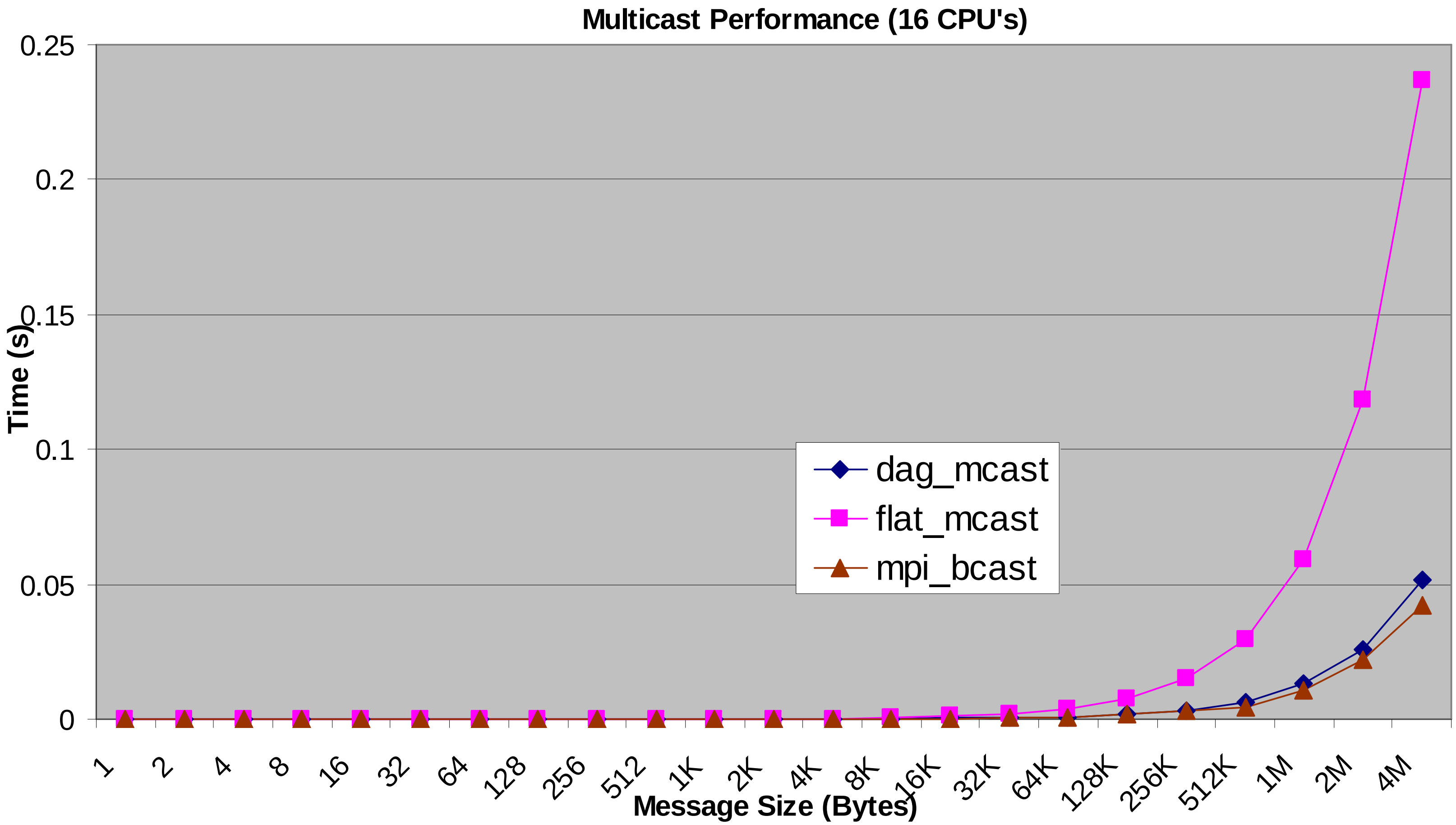
Multicast on a Cluster (4 CPUs)



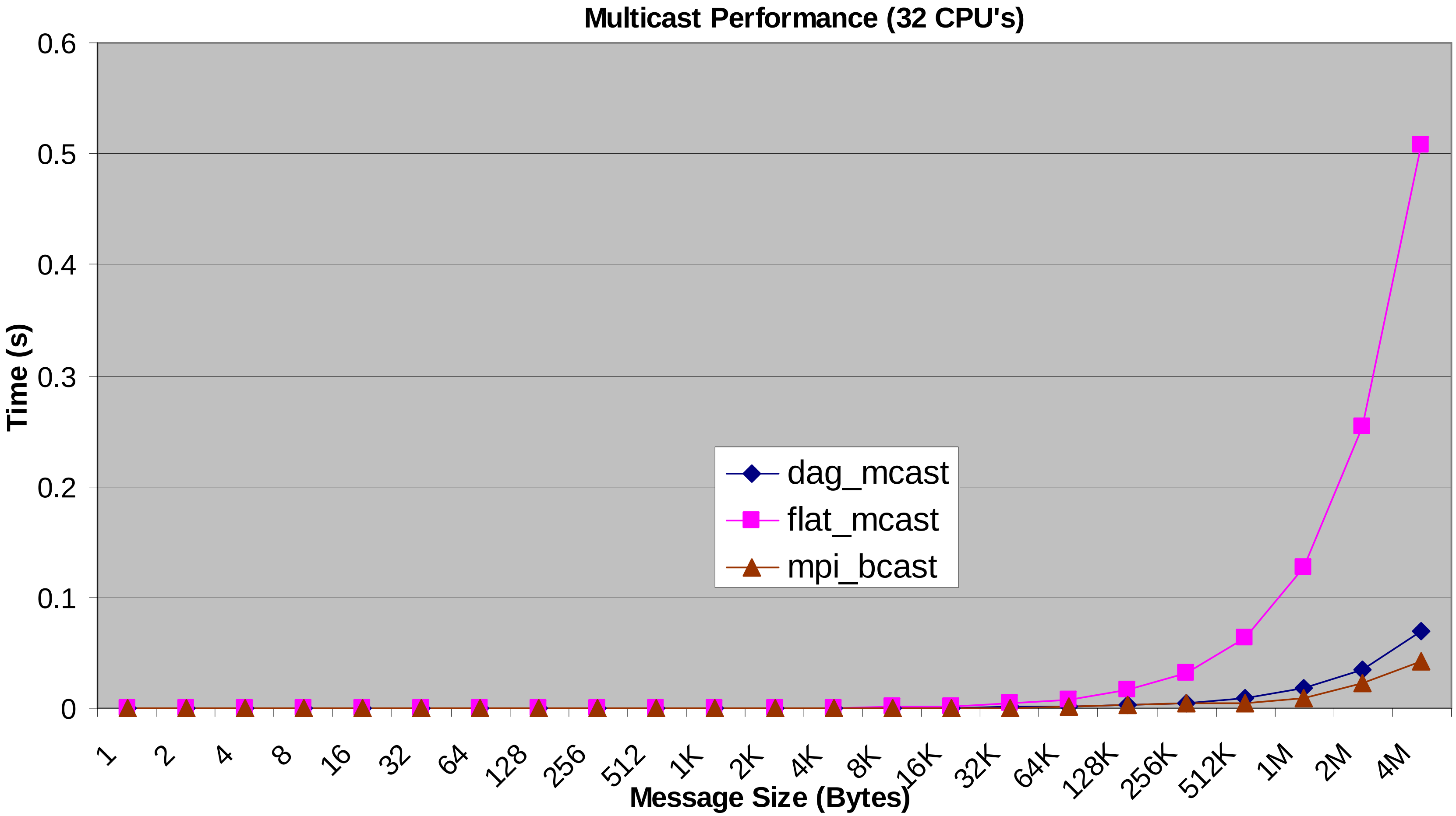
Multicast on a Cluster (8 CPUs)



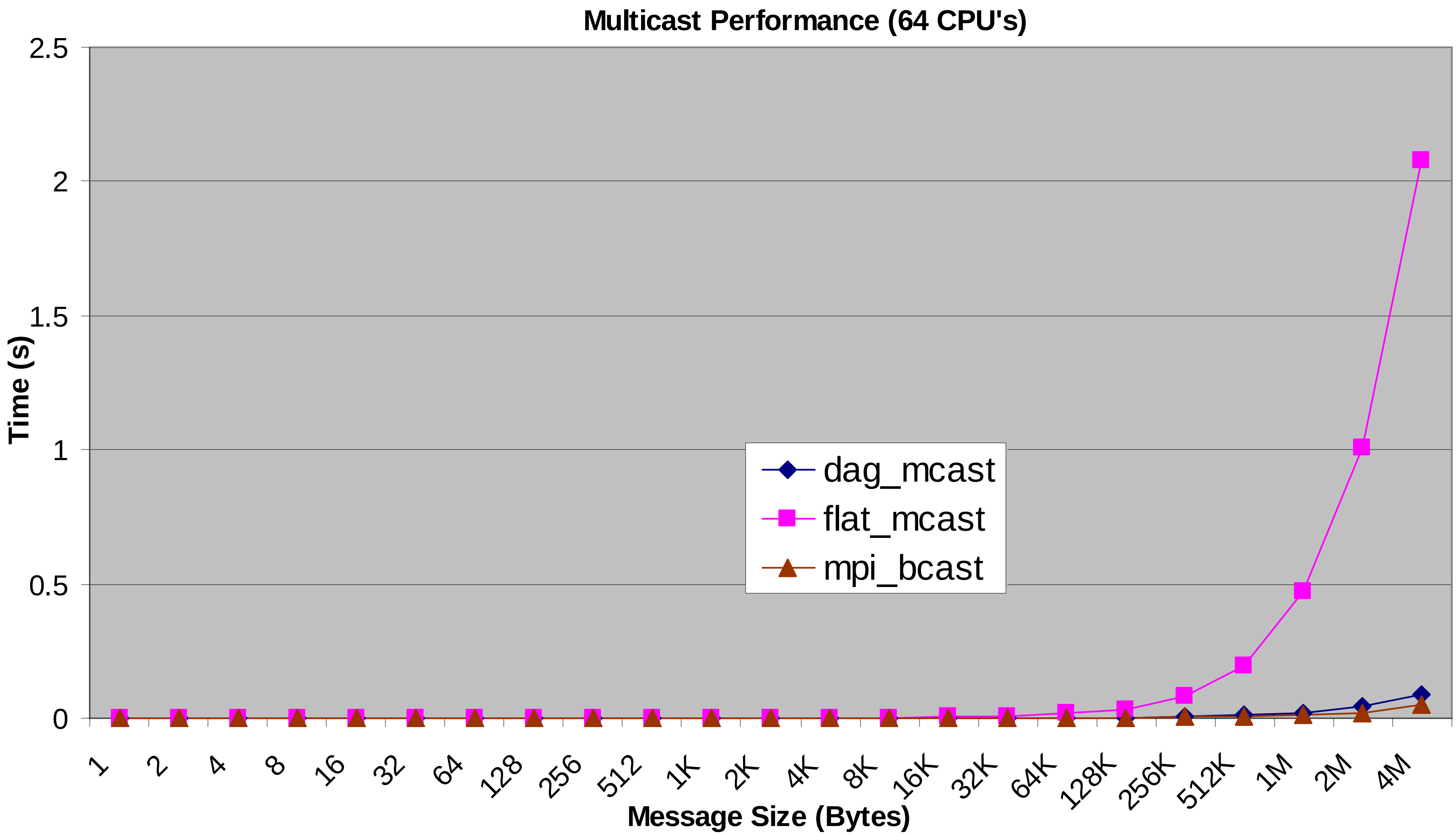
Multicast on a Cluster (16 CPUs)



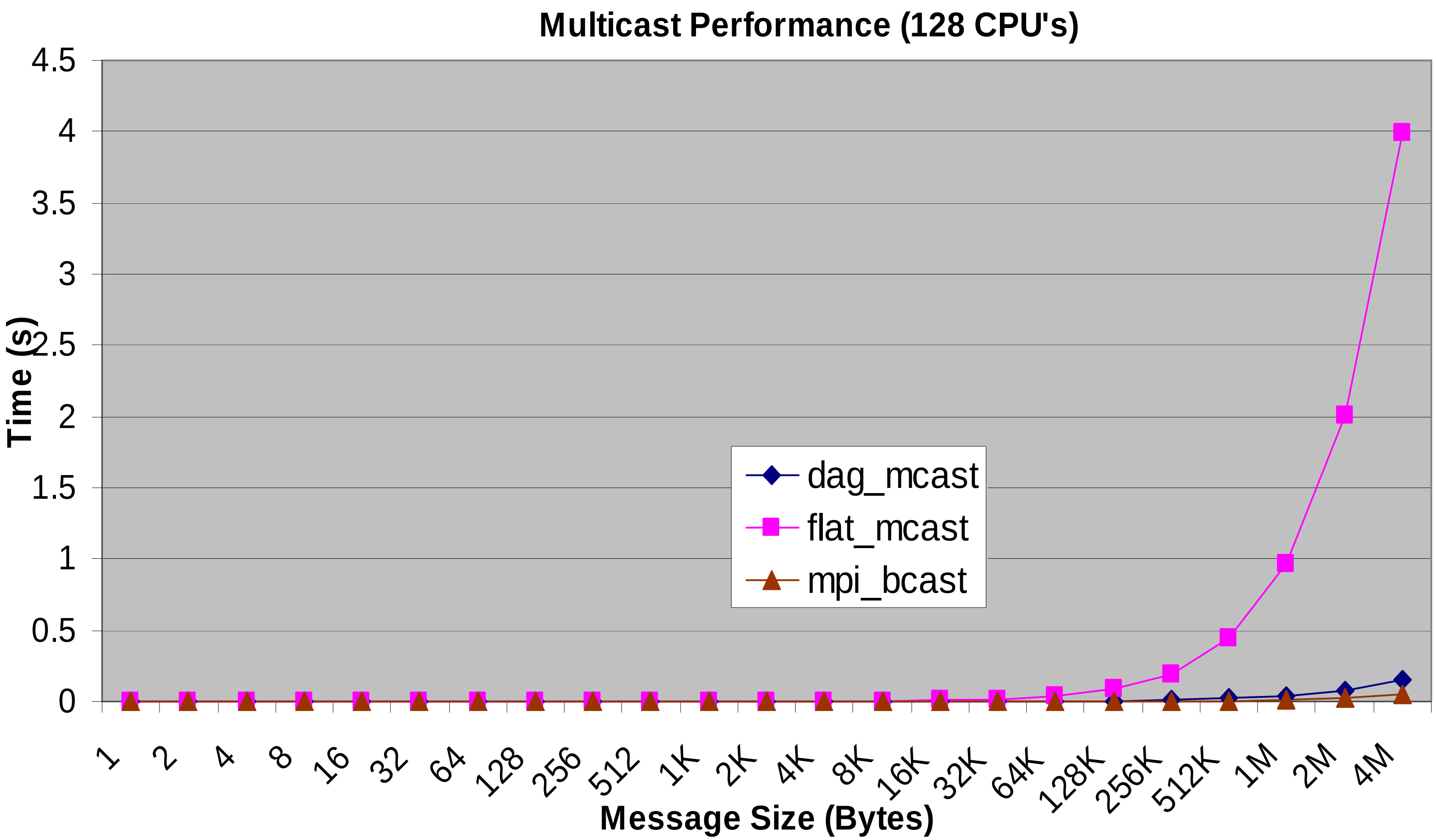
Multicast on a Cluster (32 CPUs)



Multicast on a Cluster (64 CPUs)



Multicast on a Cluster (128 CPUs)



Summary

- Support scalable multicast in distributed DAG scheduling
- Important features:
  - Non-blocking
  - Topology-aware
  - Scalable in terms of routing-table space and #steps
  - Dead-lock free
  - No requirement of communication group creation
  - Support multiple concurrent multicasts
- Performance is close to vendor's MPI_Bcast.