A Scalable Multicast Scheme for Distributed DAG Scheduling

Fengguang Song, Jack Dongarra, Shirley Moore
University of Tennessee
Oak Ridge National Laboratory

Scheduling for Large-Scale Systems Workshop
Knoxville, May 13-15, 2009
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

- Loop indep. dependency
- Loop carried dependency
- Critical path ($T_\infty$)
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**

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

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 \in (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 \in (p.j, p.i] or
           t.j = p.i & t.i \in [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|
}

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
    }
    return 3
}
```
Performance on SGI Altix 3700 BX2

Weak Scalability of chol_dag on SGI (Peak 6.4GFLOPS)

- GFLOPS Per CPU
- Number of CPU's

Chol_dag
Sgi_pdpotrf
Dgemm

May 14, 2009
Performance on the Grig Cluster

Weak Scalability of chol_dag on Grig Cluster (Peak 6.4 GFLOPS)

Number of CPU's

GFLOPS Per CPU

chol_dag
pdpotrf
peak
dgemm

May 14, 2009
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]

**Topology ID**

- Assign IDs to the whole system (i.e., $T_{\text{system}}$)
  - $T_{\text{program}}$ of a user program $\subseteq T_{\text{system}}$
- A topology ID is a number of digits.
  - E.g., 256 nodes consist of 4 digits with base 4.
    - 2bits 2bits 2bits 2bits
  - E.g., 2048 nodes consist of 4 digits with base 8.
    - 3bits 3bits 3bits 3bits
- We assume that two nodes with a longer common prefix are closer on the physical network.
Topology ID Example

Topology ID = \langle 3\text{bits}, 2\text{bits}, 2\text{bits} \rangle

SGI Altix 3700 System
Compact Routing Table

- Suppose process $x$ has a routing table and $\text{Table}[i,j]$ stores process $z$, then $\text{ID}^x$ and $\text{ID}^z$ must satisfy:
  - $\text{ID}^x_0\text{ID}^x_1...\text{ID}^x_{i-1} = \text{ID}^z_0\text{ID}^z_1...\text{ID}^z_{i-1},$
  - $\text{ID}^z_i = j$ (i.e., $(i+1)$th digit of $\text{ID}^z = j$).
- Routing table could have empty entries.
- Always search for the forwarding host on the $\text{LCP}(x,y)$ row.
- At most $(\log_2 P)/(\text{base})$ steps
- $O(\log P)$ space cost
  - 1 million cores $\rightarrow$ 80($5\times16$) entries
  - 1 billion cores $\rightarrow$ 192($6\times32$) 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 Performance (4 CPU's)

Message Size (Bytes)

Time (s)

- dag_mcast
- flat_mcast
- mpi_bcast
Multicast on a Cluster (8 CPUs)

Multicast Performance (8 CPU's)

- dag_mcast
- flat_mcast
- mpi_bcast

Time (s) vs. Message Size (Bytes)

0 0.02 0.04 0.06 0.08 0.1 0.12

1 2 4 8 16 32 64 128 256 512 1K 2K 4K 8K 16K 32K 64K 128K 256K 512K 1M 2M 4M
Multicast on a Cluster (32 CPUs)

Multicast Performance (32 CPU's)

- dag_mcast
- flat_mcast
- mpi_bcast

Message Size (Bytes)

Time (s)
Multicast on a Cluster (64 CPUs)

Multicast Performance (64 CPU's)

- dag_mcast
- flat_mcast
- mpi_bcast

Message Size (Bytes)

Time (s)

1 2 4 8 16 32 64 128 512

1K 2K 4K 8K 16K 32K 64K 128K 256K 512K 1M 2M 4M
Multicast on a Cluster (128 CPUs)

Multicast Performance (128 CPU's)

Time (s)

Message Size (Bytes)

dag_mcast
flat_mcast
mpi_bcast
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.