Service-Oriented Programming in MPI

Sarwar Alam, Humaira Kamal and Alan Wagner
University of British Columbia
Overview

Problem: How to provide data structures to MPI?

- Fine-Grain MPI
- Service-Oriented Programming
- Performance Tuning
Fine-Grain MPI
MPI

• Advantages
  • Efficient over many fabrics
  • Rich communication library

• Disadvantages
  • Bound to OS processes
  • SPMD programming model
  • Course-grain
Fine-Grain MPI

Program: OS processes with co-routines (fibers)

- Full-fledged MPI “processes”
- Combination of OS-scheduled and user-level light-weight processes inside each process
Fine-Grain MPI

- One model, inside and between nodes
- Interleaved Concurrency
- Parallel: same node between nodes
Integrated into MPICH2
Executing FG-MPI Programs

- Example of SPMD MPI program
  - with 16 MPI processes,
  - assuming two nodes with quad-core.

8 pairs of processes executing in parallel, where each pair interleaves execution
Decoupled from Hardware

- Fit the number of processes to the problem rather than the number of cores

```bash
mpiexec -nfg 350 -n 4 myprog
```
Flexibility

- Move the boundary between light-weight user scheduled concurrency, and processes running in parallel.

```
mpiexec -nfg 1000 -n 4 myprog

mpiexec -nfg 500 -n 8 myprog

mpiexec -nfg 750 -n 4 myprog: -nfg 250 -n 4 myprog
```
Scalability

- Can have hundreds and thousands of MPI processes.

```bash
mpiexec -nfg 30000 -n 8 myprog
```

- 100 Million processes on 6500 cores

```bash
mpiexec -nfg 16000 -n 6500 myprog
```
Service-Oriented Programming

- Linked List Structure
- Keys in sorted order
- Similar
  - Distributed hash table
  - Linda Tuple Spaces
Ordered Linked-List

An MPI process in ordered list

- Minimum key value of items stored in next MPI process
- Rank of MPI process with next larger key values
- Stores one or more key values
- Data associated with key

Previous MPI process in ordered list

Next MPI process in ordered list
Ordered Linked-List

INSERT
- Data sent after
- Sequence number

DELETE
- Success/Failure
- Sequence number

FIND
- Sequence number
- Return data
Ordered Linked-List
Step 1: ‘A’ requests for a new node, ‘C’ replies

Step 2: ‘A’ sends ‘C’ information about ‘B’
   ‘C’ then connects to ‘B’

Step 3: ‘A’ sends terminate to ‘B’
   before connecting to ‘C’

Step 4: ‘C’ is linked in and informs application
DELETE

Step 1: ‘A’ passes delete to ‘B’ and waits

Step 2: ‘B’ sends terminate to ‘C’ before acknowledging to ‘A’

Step 3: ‘A’ accepts ‘C’ from ‘B’ then ‘B’ informs success to application

Step 4: ‘B’ converts to a free node
Step 1: ‘A’ passes find to ‘B’

Step 2: If match found or match not possible
‘B’ informs application
(else request passed to ‘C’)

**LEGEND**

- **Node Types:**
  - ☺ - List Node
  - ○ - Free Node
  - ● - Application Node
  - ■ - Manager Node

- **Connection Arrows:**
  - → - Existing Link
  - ← - New Connection

- **Messaging Arrows:**
  - → - Message Sent
  - ← - Reply Sent
  - - - - - → - Free Node Service Route
Ordered Linked-List

L0 → L28 → L12 → L56 → L75 → L43 → L21 → L18

F65 → F30 → A12

L56 → L75
Local non-communication operations are ATOMIC
Re-incarnation

Local Process Ecosystem

Local non-communication operations are ATOMIC
Granularity

• Added the ability for each process to manage a collection of consecutive items.
• Changes to INSERT, changes into a SPLIT operation
• Changes to DELETE, on delete of last item
• List Traversal consists of:
  • Jumping between processes
  • Jumping co-located processes
  • Search inside a process
Properties

• **Total Ordered** – operations are ordered by the order they arrive at the root

• **Sequentially Consistent** – each application process keeps a hold-back queue to return results in order

• **No consistency** – operations can occur in any order
Performance Tuning

• **G** (granularity) the number of keys stored in each process.

• **K** (asynchrony) the number of messages in the channel between list processes.

• **W** (workload) the number of outstanding operations
Steady-State Throughput

Fixed list size, evenly distributed over O x M core

Throughput (Operations per Second)

Number of Cores

16,000 operations/sec

5793 operations/sec
Granularity (G)

Fixed-size machine (176 cores), Fixed list size ($2^{20}$)

Moving work from INSIDE a process to BETWEEN processes

Sequentially Consistent

No-consistency

10X larger
W and K

W : Number of outstanding requests (workload)
K : Degree of Asynchrony
Conclusions

• Reduced coupling and increased cohesion
• Scalability within clusters of multicore
• Performance tuning controls
  • Adapt to hierarchical network fabric
• Distributed systems properties pertaining to consistency
Thank-You