Monitoring and Debugging Parallel Software with BCS-MPI on Large-Scale Clusters

Juan Fernández¹, Fabrizio Petrini², Eitan Frachtenberg²

¹Grupo de Arquitectura y Computación Paralela (GACOP)
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia, SPAIN
URL: http://www.ditec.um.es

²Performance and Architecture Lab (PAL)
CCS-3 Modeling, Algorithms and Informatics
Los Alamos National Laboratory, NM 87545, USA
URL: http://www.c3.lanl.gov

email: juanf@um.es
Clusters have been the most successful player in high-performance computing in the last decade.

**HARDWARE** = Independent Nodes + High-speed Network

**SOFTWARE** = Commodity OS + *System Software* + Parallel Apps

---

**Independent Nodes / OSs**

- Glued together by **System Software**:
  - Parallel Development and Debugging Tools
  - Resource Management
  - Communications
  - Parallel File System
  - Fault Tolerance

**Parallel Apps**:
- Message passing (MPI)
Motivation

- Ever-increasing demand for computing capability is driving the construction of ever-larger clusters

1. BlueGene/L DD2 32768 Processors
2. Columbia 10160 Processors
3. Earth Simulator 5120 Processors

System Software and Parallel Applications grow in complexity as cluster sizes increase
Motivation

- Developing and maintaining parallel software is far more complicated than sequential software
  - Commodity hardware/OSs not designed for clusters
    - Hardware conceived for loosely-coupled environments
    - Local OSs lack global awareness of parallel applications
  - Complex global state of MPI apps
  - MPI apps rely on services provided by system software
  - Non-deterministic behavior inherent to clusters (local OS scheduling) and parallel applications (MPI_ANY_SOURCE)

Development of Parallel Software is a very time-and resource-consuming task
Introduction

Monitoring and debugging parallel software:

- Compile-time and run-time techniques
  - Additional software that somehow interacts with MPI applications to either gather data or perform checks of different nature
  - System Software is often ignored and assumed to be reliable
- Buffered CoScheduled MPI (BCS-MPI)
  - Based on a methodology to reduce system software complexity:
    - Small set of efficient and scalable hardware-supported primitives
    - Global control and coordination of all system activities
  - BCS-MPI imposes an execution model where processes and communications are scheduled at a fine granularity

Monitoring and Debugging System (MDS) which integrates with the BCS-MPI runtime system
Outline

- Motivation
- Introduction
- Design and Implementation of BCS-MPI
- Monitoring and Debugging System (MDS)
- Concluding remarks
Real-time communication scheduling

- Exchange of comm requirements
- Communication scheduling
- Real transmission

Time Slice (hundreds of µs)

Global Strobe (time slice starts)
Global Synchronization
Global Synchronization
Global Strobe (time slice ends)
Implementation in *Network Interface Card*

- Application processes interact with NIC threads
  - MPI primitive $\Rightarrow$ Descriptor posted to the NIC
  - Communications are buffered
- Cooperative threads running in the NIC
  - Synchronize
  - Partial exchange of control information
  - Schedule communications
  - Perform real transmissions and reduce computations
- Computation/communication completely overlapped
  - Incoming messages do not generate interrupts
  - User processes do not need to poll for communication completion
Non-blocking primitives: MPI_Isend/Irecv
Global Synchronization/Scheduling Protocol

- Global Message Scheduling Phase
  - Microphases: Descriptor Exchange + Message Scheduling
- Message Transmission Phase:
  - Microphases: Point-to-point, Barrier and Broadcast, Reduce
Outline

- Motivation
- Introduction
- Design and Implementation of BCS-MPI
- Monitoring and Debugging System (MDS)
- Concluding remarks
Monitoring and Debugging System (MDS)

- *A posteriori* data analysis not only for MPI apps but also for the BCS-MPI runtime system itself
- MDS is logically divided into two main components
  - Main MDS (MMDS)
    - Process scheduling and communication primitives
  - Elan MDS (EMDS)
    - Communication pattern of MPI applications
    - Behavior of the BCS-MPI runtime system itself
- Both modules can be enabled/disabled by just setting and env variable without compiling or linking the code
Main MDS Implementation

MMDS profiles the BCS-MPI API
- Comp granularity/comm overhead distribution
- BCS-MPI primitives usage (minimum / maximum / average latency, latency and size distributions)
- Counters and distribution arrays in main memory

Workshop on System Management Tools for Large-Scale Parallel Systems (IPDPS´05) - Denver, CO
Elan MDS Implementation

- EMDS profiles the NIC threads
  - Global metrics on the sync/scheduling protocol
  - Local metrics regarding process and comm scheduling
  - Debugging metrics (time to complete specific routines)
  - Counters and distribution arrays in NIC memory

Workshop on System Management Tools for Large-Scale Parallel Systems (IPDPS’05) - Denver, CO
## Experimental Setup

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Wolverine Cluster</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>64 AlphaServer ES40</td>
</tr>
<tr>
<td>CPUs/Node</td>
<td>4 x 833MHz EV68</td>
</tr>
<tr>
<td>Memory/Node</td>
<td>8 GB</td>
</tr>
<tr>
<td>Network Cards</td>
<td>QM-400 Elan3</td>
</tr>
<tr>
<td>OS</td>
<td>RH 7.1 + QsNet kernel</td>
</tr>
<tr>
<td>Software</td>
<td>Qsnetlibs v1.5.0-0</td>
</tr>
</tbody>
</table>
Analyzing the BCS-MPI runtime system

- Microbenchmark (MPI_Barrier in a loop)

Number of scheduled collectives in the MSN microphase (NCOLLMSN)

Global Elapsed Time from previous Time Slice (GETTS)
Analyzing a real MPI application

- SAGE spends most communication time in three MPI primitives
- Top-down approach to debug application

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Min (ms)</th>
<th>Max (ms)</th>
<th>Total (ms)</th>
<th>Count</th>
<th>Average (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Isend</td>
<td>0.588</td>
<td>16.576</td>
<td>21026</td>
<td>4396</td>
<td>4.783</td>
</tr>
<tr>
<td>MPI_Irecv</td>
<td>0.736</td>
<td>16.644</td>
<td>24280</td>
<td>5617</td>
<td>4.323</td>
</tr>
<tr>
<td>MPI_Allreduce</td>
<td>0.366</td>
<td>24.753</td>
<td>18906</td>
<td>7025</td>
<td>2.691</td>
</tr>
</tbody>
</table>
Operational overhead incurred by the MDS

- MMDS overhead < 0.5%
- EMDS overhead slightly higher
  - Small TLB and cache sizes in the Elan3 NIC

<table>
<thead>
<tr>
<th>Input Deck</th>
<th>MDS Disabled</th>
<th>MMDS Runtime</th>
<th>MMDS Overhead</th>
<th>EMDS Runtime</th>
<th>EMDS Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>timing_h</td>
<td>114.604s</td>
<td>115.023s</td>
<td>0.36%</td>
<td>116.102s</td>
<td>1.31%</td>
</tr>
<tr>
<td>timing_c</td>
<td>193.202s</td>
<td>193.345s</td>
<td>0.07%</td>
<td>193.419s</td>
<td>0.11%</td>
</tr>
</tbody>
</table>

Negligible performance degradation!
Outline

- Motivation
- Introduction
- Design and Implementation of BCS-MPI
- Monitoring and Debugging System (MDS)
- Concluding remarks
Concluding Remarks

- BCS-MPI globally schedules all system activities in deterministically reproducible, global steps.
- Leveraging the BCS-MPI parallel execution model, we have developed an innovative Monitoring and Debugging System (MDS).
- MDS can be used to monitor and debug not only parallel MPI applications but the BCS-MPI runtime system itself with negligible performance degradation.
Monitoring and Debugging Parallel Software with BCS-MPI on Large-Scale Clusters

Juan Fernández¹, Fabrizio Petrini², Eitan Frachtenberg²

¹Grupo de Arquitectura y Computación Paralela (GACOP)
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia, SPAIN
URL: http://www.ditec.um.es

²Performance and Architecture Lab (PAL)
CCS-3 Modeling, Algorithms and Informatics
Los Alamos National Laboratory, NM 87545, USA
URL: http://www.c3.lanl.gov

email: juanf@um.es