Buffered Coscheduled MPI: A New Approach in the System Software Design for Large-Scale Parallel Computers

Juan Fernández¹,², Eitan Frachtenberg¹, Fabrizio Petrini¹

¹CCS-3 Modeling, Algorithms and Informatics
Computer and Computational Sciences (CCS) Division
Los Alamos National Laboratory, NM 87545, USA
URL: http://www.c3.lanl.gov

²Grupo de Arquitectura y Computación Paralelas (GACOP)
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia, SPAIN
URL: http://www.ditec.um.es

email:{juanf,eitanf,fabrizio}@lanl.gov
Motivation

System software is a key factor to maximize usability, performance and scalability on large-scale systems!!!
Motivation

- **System software complexity due to multiple factors:**
  - Extremely complex global state
    - Thousands of processes, threads, open files, pending messages, etc.
  - Non-deterministic behavior
    - Inherent to computing systems ⇒ OS process scheduling
    - Induced by parallel applications ⇒ MPI_ANY_SOURCE
  - Local OSs lack global awareness of parallel applications
    - Interferences with fine-grain synchronization operations ⇒ Non-scalable collective communication primitives.
Motivation

- System software complexity due to multiple factors:
  - Independent **design** of different components
    - Redundancy of functionality ⇒ Communication protocols
    - Missing functionality ⇒ QoS user-level traffic / system-level traffic
  - User-level applications rely on system software
    - System software performance/scalability impacts user-application performance/scalability
Goals

- **Target**
  - Simplifying design and implementation of the system software for large-scale computers
  - Simplicity, performance, scalability

- **Approach**
  - Built atop a basic set of three primitives
  - Global synchronization/scheduling

**Vision**
- SIMD system running MIMD applications (variable granularity in the order of hundreds of µs)
Outline

- Motivation and Goals
- Introduction
- Core Primitives
- Design and Implementation
- Performance Evaluation
- Concluding remarks
Introduction

In this paper we make our point by implementing the communication layer: BCS MPI

- Built atop the three core primitives we proposed as the basics for all system software components (STORM is a resource manager implemented atop the same primitives)
- SIMD-like behavior: communications are synchronized and scheduled every few hundreds of microseconds
- Implemented almost entirely in the Network Interface Card
- Sacrifices neither performance nor scalability
- Paves the way to provide improved system utilization, traffic segregation, deterministic replay of user applications and system-level fault tolerance
Outline

- Motivation and Goals
- Introduction
- Core Primitives
- Design and Implementation
- Performance Evaluation
- Concluding remarks
BCS Core Primitives

System software built atop three primitives

- Xfer-And-Signal
  - Transfer block of data to a set of nodes
  - Optionally signal local/remote event upon completion

- Compare-And-Write
  - Compare global variable on a set of nodes
  - Optionally write global variable on the same set of nodes

- Test-Event
  - Poll local event
BCS Core Primitives

System software built atop three primitives

- **Xfer-And-Signal (QsNet):**
  - Node S transfers block of data to nodes D₁, D₂, D₃, and D₄
  - Events triggered at source and destinations
System software built atop three primitives

- Compare-And-Write (QsNet):
  - Node S compares variable V on nodes D₁, D₂, D₃ and D₄

Is V {=, ≠, ≤, >} to Value?
System software built atop three primitives

- **Compare-And-Write (QsNet):**
  - Node $S$ compares variable $V$ on nodes $D_1$, $D_2$, $D_3$ and $D_4$
  - Partial results are combined in the switches
Design and Implementation

- 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)
Global synchronization
- Strobe sent at regular intervals (time slices)
  - Compare-And-Write + Xfer-And-Signal (Master)
  - Test-Event (Slaves)
- All system activities are tightly coupled
- Global information is required to schedule resources, global synchronization facilitates the task but it is not enough

Global Scheduling
- Exchange of communication requirements
  - Xfer-And-Signal + Test-Event
- Communication scheduling
- Real transmission
  - Xfer-And-Signal + Test-Event
Implementation in the *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
Design and Implementation

Non-blocking primitives: MPI_Isend/Irecv
Design and Implementation

Blocking primitives: MPI_Send/Recv

- Time slice i
  - P1: Computation
  - NIC1: Global Message Scheduling
  - NIC2: Message Transmission
  - P2: Computation

- Time slice i+1
  - NIC1: Data
  - NIC2: Time slice i

- Time slice i+2
  - P1: Computation
  - P2: Computation
Global Synchronization/Scheduling Protocol

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

Processes and Threads

- Synchronization: Strobe Sender + Strobe Receiver
- Scheduling: Buffer Sender, Buffer Receiver
- Transmission: DMA, Collective and Reduce Helpers
Simplicity
- Hierarchical design atop a single kernel module which implements the three core primitives

Determinism
- Synchronization facilitates resource scheduling
- Resource scheduling enforces reproducibility

Performance / Scalability
- Hardware-supported primitives

BCS MPI implementation
- QsNet hardware-supported multicast / events
Outline

- Motivation and Goals
- Introduction
- Core Primitives
- Design and Implementation
- Performance Evaluation
- Concluding remarks
Performance Evaluation

**BCS MPI vs. Quadrics MPI**

- **Experimental Setup**
  - *crescendo* cluster at LANL/CCS-3
    - 32 Dell 1550 compute nodes (two 1GHz P-III processors)
    - Quadrics QM-400 Elan3 NIC (1 RAIL)
    - RH 7.3 + Quadrics mods + qsnetlibs v1.5.0-0
    - Intel C/Fortran Compiler v5.0.1 (-O3)
  - *accelerando* cluster at LANL/CCS-3
    - 32 HP Server rx2600 (two 1GHz Itanium-II processors)
    - Two Quadrics QM-400 Elan3 NICs (2 RAILs)
    - RH 7.2 + Quadrics mods + qsnetlibs v1.5.0-0
    - Intel C/Fortran Compiler v7.1.17 (-O3)
Performance Evaluation

BCS MPI vs. Quadrics MPI

- Experimental Setup
  - User-level implementation of BCS MPI
    - Scheduling parameters
      - 500µs communication scheduling time slice (1 rail)
      - 250µs communication scheduling time slice (2 rails)
  - Benchmarks and Applications
    - NPB (IS,EP, MG, CG, LU) - Class C
    - SWEEP3D - 50x50x50
    - SAGE - timing.input
Benchmarks and Applications (IA32)

<table>
<thead>
<tr>
<th>Application</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS (32PEs)</td>
<td>10.40%</td>
</tr>
<tr>
<td>EP (49PEs)</td>
<td>5.35%</td>
</tr>
<tr>
<td>MG (32PEs)</td>
<td>4.37%</td>
</tr>
<tr>
<td>CG (32PEs)</td>
<td>10.83%</td>
</tr>
<tr>
<td>LU (32PEs)</td>
<td>15.04%</td>
</tr>
<tr>
<td>SWEEP3D (49PEs)</td>
<td>-2.23%</td>
</tr>
<tr>
<td>SAGE (62PEs)</td>
<td>-0.42%</td>
</tr>
</tbody>
</table>
Performance Evaluation

- SAGE - timing.input (IA32)

0.5% SPEEDUP
Performance Evaluation

- Blocking vs Non-blocking SWEEP3D (IA32)
  - $\text{MPI	extunderscore Send}/\text{Recv} \Rightarrow \text{MPI	extunderscore Isend}/\text{Irecv} + \text{MPI	extunderscore Waitall}$

<table>
<thead>
<tr>
<th>Number of Processes</th>
<th>Quadrics MPI (Blocking)</th>
<th>BCS MPI (Blocking)</th>
<th>Quadrics MPI (Non-blocking)</th>
<th>BCS MPI (Non-blocking)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>40</td>
<td>50</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>9</td>
<td>45</td>
<td>55</td>
<td>45</td>
<td>55</td>
</tr>
<tr>
<td>16</td>
<td>50</td>
<td>60</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td>25</td>
<td>55</td>
<td>70</td>
<td>55</td>
<td>70</td>
</tr>
<tr>
<td>36</td>
<td>60</td>
<td>80</td>
<td>60</td>
<td>80</td>
</tr>
<tr>
<td>49</td>
<td>65</td>
<td>90</td>
<td>65</td>
<td>90</td>
</tr>
</tbody>
</table>

30% SLOWDOWN

2% SPEEDUP
Performance Evaluation

- Blocking vs Non-blocking SWEEP3D (IA64)
  - `MPI_Send/Recv` ⇒ `MPI_Isend/Irecv + MPI_Waitall`

- 64% SLOWDOWN 1 RAIL
- 34% SLOWDOWN 2 RAILS
- 1% SPEEDUP 1 RAIL
- 2% SPEEDUP 2 RAILS
Performance Evaluation

BCS MPI: Similar Performance with a Simplified Design
Concluding Remarks

- New approach to the design of system software
  - Kernel module which implements the basics of most system software components (STORM) ⇒ Simplicity
  - Hardware-supported primitives ⇒ Performance / Scalability

- Prototype implementation of BCS MPI
  - Global synchronization of all system activities
  - Global scheduling of communications
  - Implemented almost entirely in the NIC
  - Global state optimization vs latency/bandwidth optimization

Promising preliminary results with real apps
  - BCS MPI can compete with a production-level MPI
Future Work

- Kernel-level implementation of BCS-MPI
  - User-level solution is already working with negligible performance degradation

- Improved system utilization
  - Scheduling multiple jobs

- QoS for different types of traffic
  - Scheduling messages may provide traffic segregation

- Deterministic replay of MPI programs
  - Ordered resource scheduling may enforce reproducibility

**Transparent fault tolerance**
- BCS MPI simplifies the state of the machine
Buffered Coscheduled MPI:
A New Approach in the System Software Design
for Large-Scale Parallel Computers

Juan Fernández¹,², Eitan Frachtenberg¹, Fabrizio Petrini¹

¹CCS-3 Modeling, Algorithms and Informatics
Computer and Computational Sciences (CCS) Division
Los Alamos National Laboratory, NM 87545, USA
URL: http://www.c3.lanl.gov

²Grupo de Arquitectura y Computación Paralelas (GACOP)
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia, SPAIN
URL: http://www.ditec.um.es

email:{juanf,eitanf,fabrizio}@lanl.gov