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

Juan Fernández[1], Fabrizio Petrini[2], Eitan Frachtenberg[2]

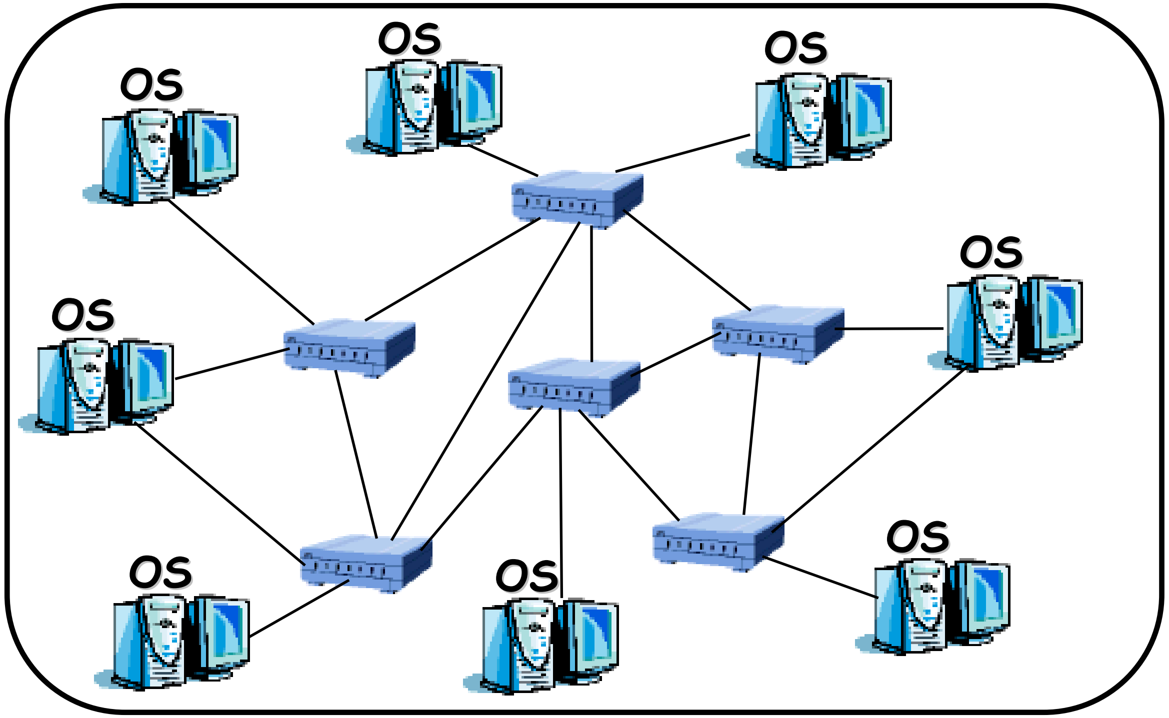
[1]Grupo de Arquitectura y Computación Paralela (GACOP)
Dpto. Ingeniería y Tecnología de Computadores
Universidad de Murcia, 30071 Murcia, SPAIN
URL: http://www.ditec.um.es

[2]Performance 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

Motivation

- **Clusters have been the most successful player in high-performance computing in the last decade**



Independent Nodes / OSs glued together by System Software:

Parallel Development and Debugging Tools
Resource Management
Communications
Parallel File System
Fault Tolerance

Paralell Apps:
Message passing (MPI)

HARDWARE = Independent Nodes + High-speed Network
SOFTWARE = Commodity OS + System Software + Parallel Apps

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

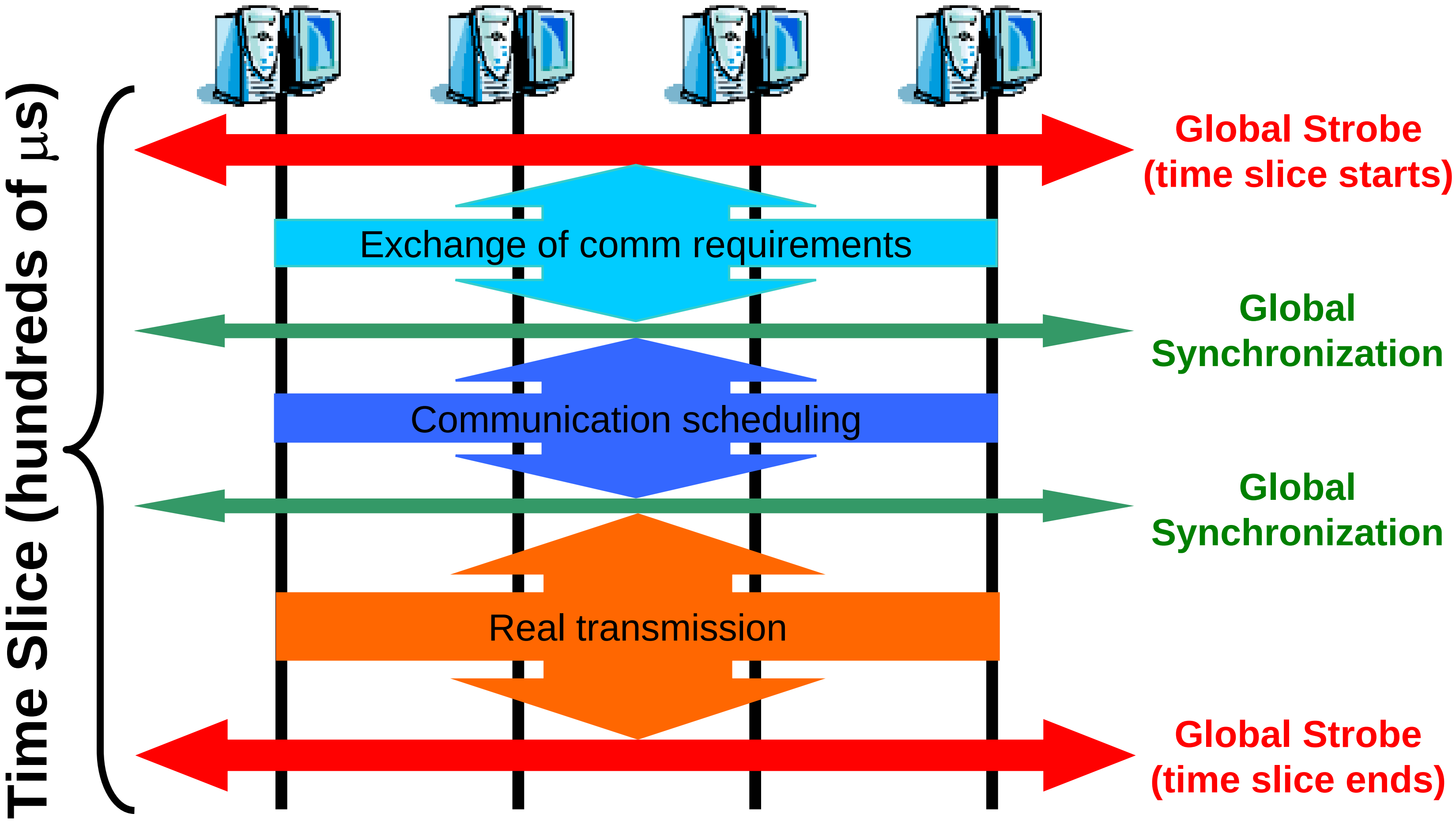
Monitoring and Debugging System (MDS) which integrates with the BCS-MPI runtime system

Outline

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

BCS-MPI

- Real-time communication scheduling

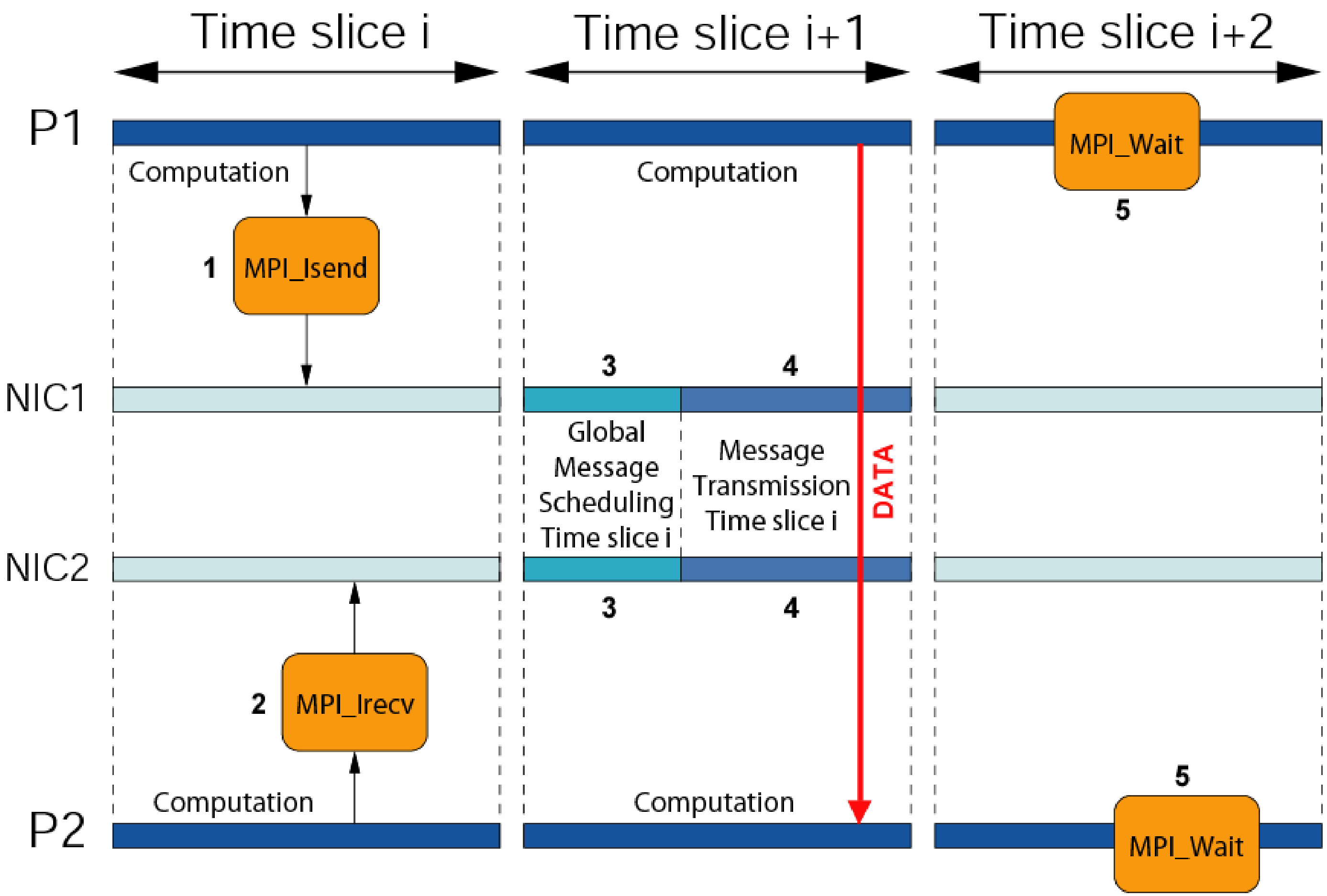


BCS-MPI

- **Implementation in *Network Interface Card***
  - Application processes interact with NIC threads
    - – MPI primitive ⇨ 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

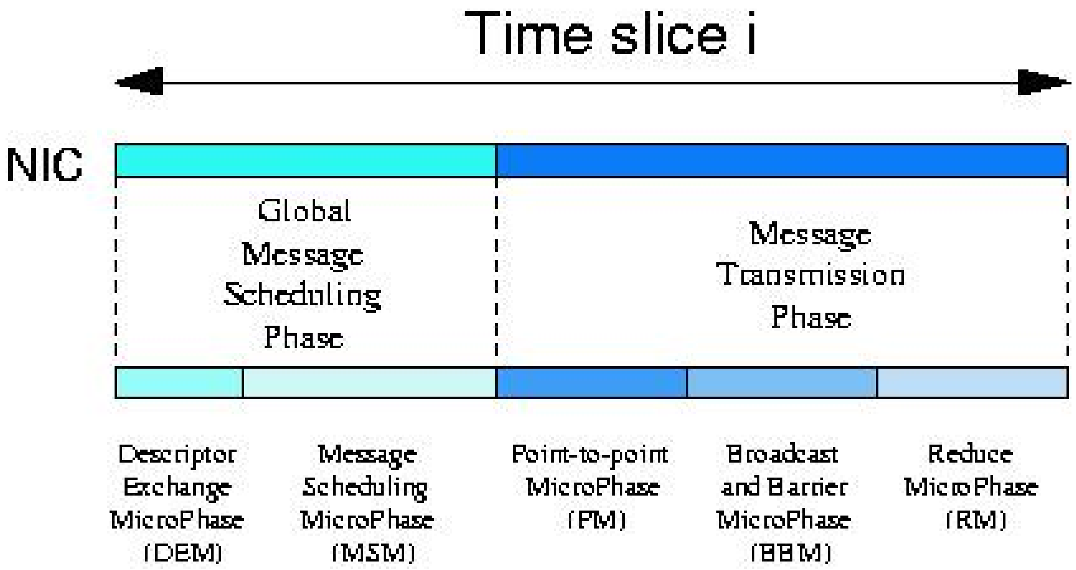
BCS-MPI

- Non-blocking primitives: MPI_Isend/Irecv



BCS-MPI

- **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

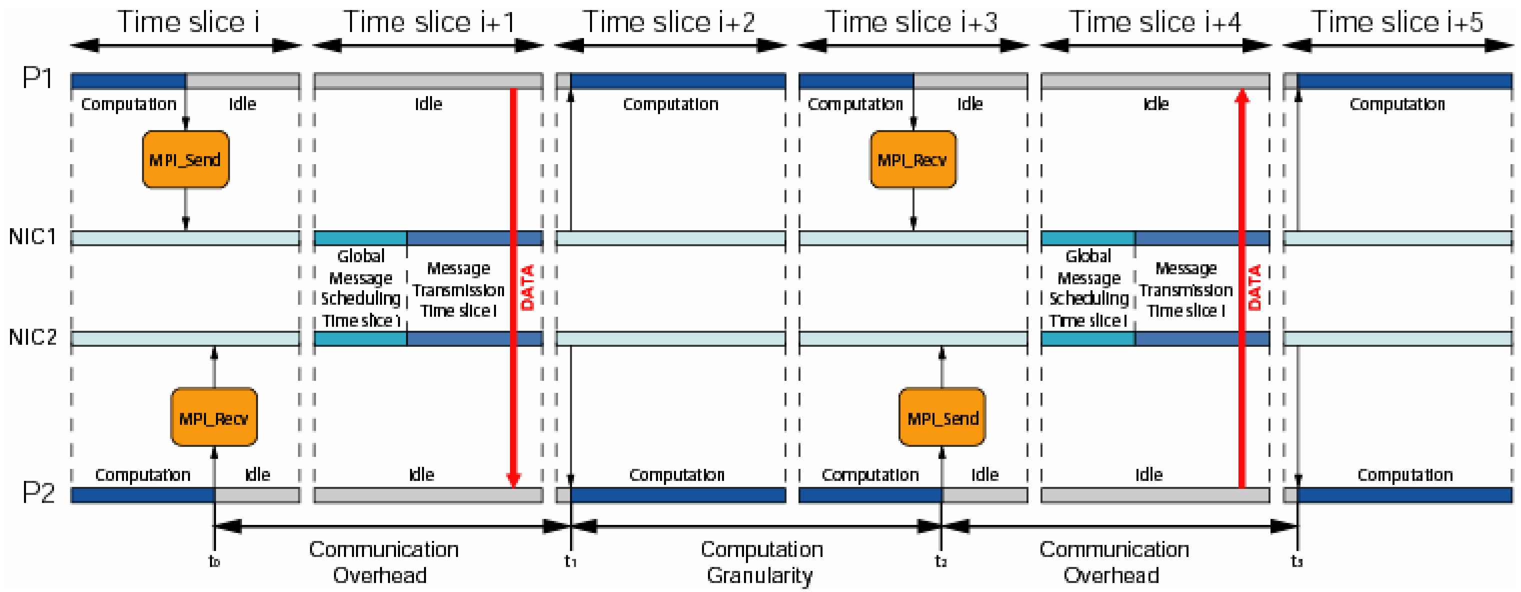
- Motivation
- Introduction
- Design and Implementation of BCS-MPI
- Monitoring and Debugging System (MDS)
- Concluding remarks

MDS

- **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

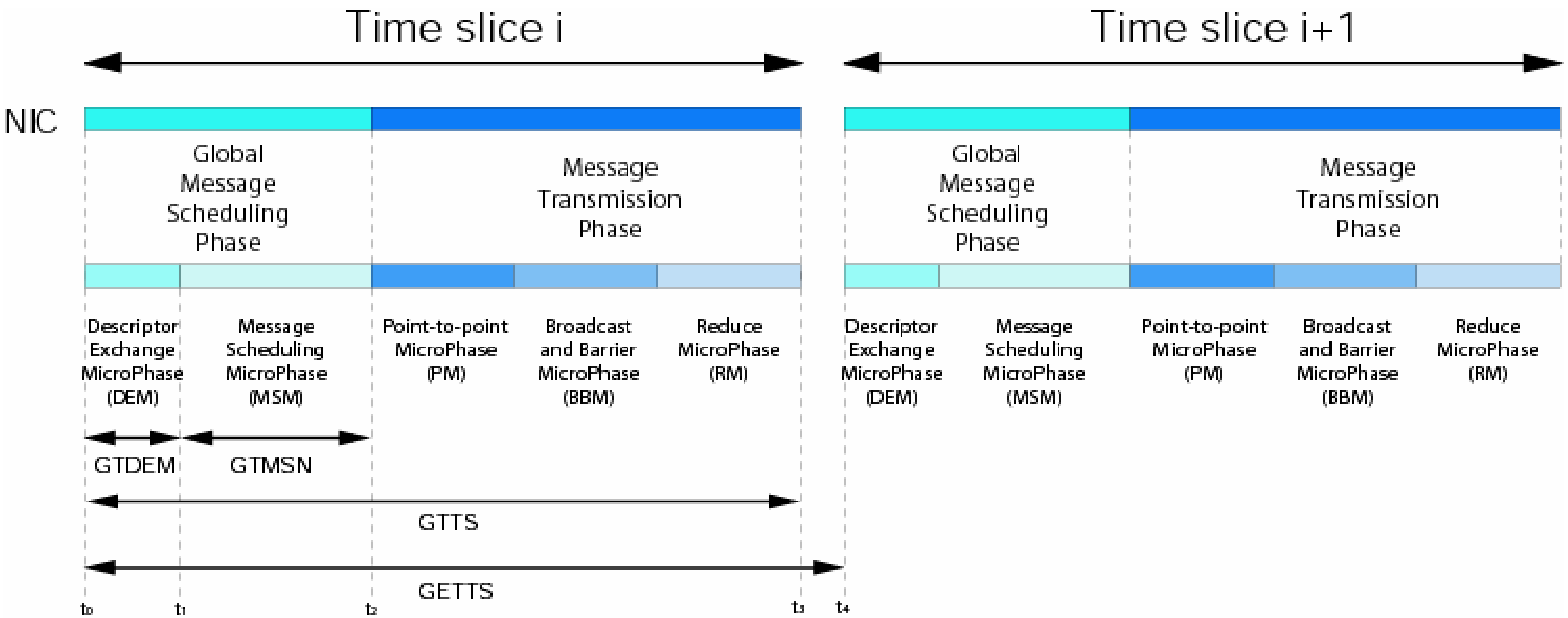
Main MDS Implementation

- **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



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



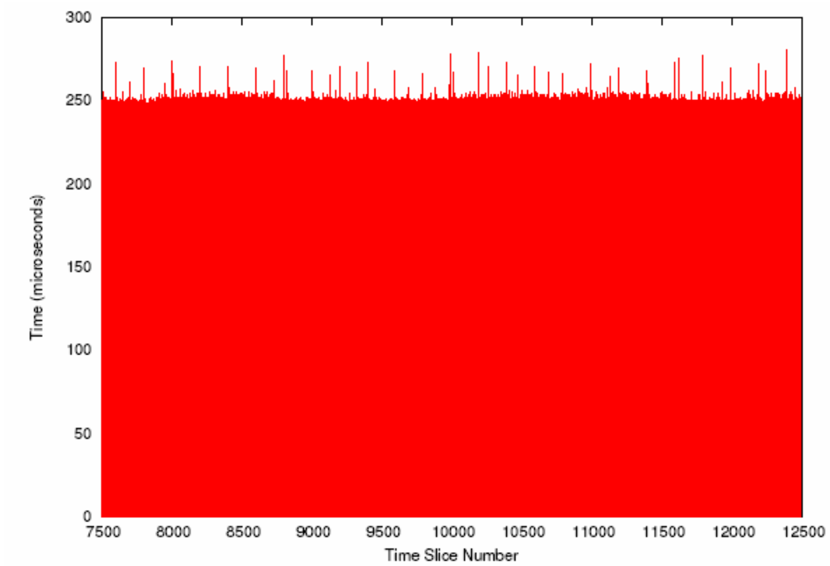
MDS

Experimental Setup

Characteristic	Wolverine Cluster
Nodes	64 AlphaServer ES40
CPUs/Node	4 x 833MHz EV68
Memory/Node	8 GB
Network Cards	QM-400 Elan3
OS	RH 7.1 + QsNet kernel
Software	Qsnetlibs v1.5.0-0

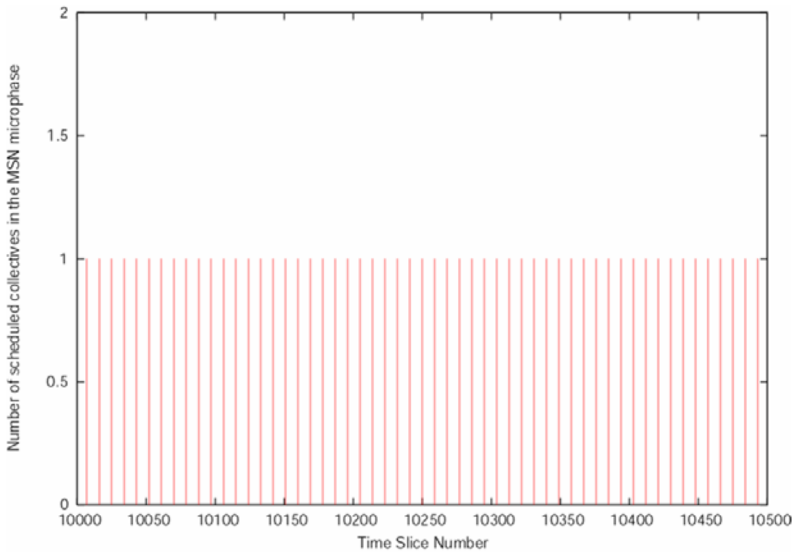
MDS

- **Analyzing the BCS-MPI runtime system**
  - Microbenchmark (MPI_Barrier in a loop)



Global Elapsed Time from previous Time Slice (GETTS)

Number of scheduled collectives in the MSN microphase (NCOLLMSN)



MDS

- **Analyzing a real MPI application**
  - SAGE spends most comm time in three MPI primitives
  - Top-down approach to debug application

Primitive	Min (ms)	Max (ms)	Total (ms)	Count	Average (ms)
MPI_Isend	0.588	16.576	21026	4396	4.783
MPI_Irecv	0.736	16.644	24280	5617	4.323
MPI_Allreduce	0.366	24.753	18906	7025	2.691
...	...	...	...	...	...

MDS

- **Operational overhead incurred by the MDS**
  - MMDS overhead < 0.5%
  - EMDS overhead slightly higher
    - – Small TLB and cache sizes in the Elan3 NIC

Input Deck	MDS Disabled	MMDS Runtime	MMDS Overhead	EMDS Runtime	EMDS Overhead
timing_h	114.604s	115.023s	0.36%	116.102s	1.31%
timing_c	193.202s	193.345s	0.07%	193.419s	0.11%

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 paralell 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 Fernández[1], Fabrizio Petrini[2], Eitan Frachtenberg[2]

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

[2]Performance 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

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