Buffered Coscheduled (BCS) MPI

A New Approach in the System Software Design for Large-Scale Parallel Computers

Juan Fernández[1,2], Fabrizio Petrini[1] and Eitan Frachtenberg[1]
{juanf,fabrizio,eitanf}@lanl.gov
[1] Modeling, Algorithms and Informatics Group (CCS-3)
Los Alamos National Laboratory
[2] Computer Engineering Department
University of Murcia, SPAIN

Motivation

BCS-MPI introduces a new approach to design system sofware for large-scale parallel machines. The goal is to reduce the complexity, non-determinism and redundancy of the main components of the system software with a minimal performance penalty.

BCS-MPI globally organizes all the system activities at a very fine granularity. Both computation and communication are scheduled at regular intervals, in a real-time fashion, and the scheduling decisions are taken after a global exchange of control information.

BCS-MPI is a lightweight MPI implementation that represents a trade-off between simplicity and performance. It is designed on top of a minimal set of communication primitives that are almost entirely implemented in the network interface card.

BCS-MPI has been succesfully validated with several scientific codes representative of the ASCI workload.

Goals

Goals	Current Status	Future Work
-Target: large-scale parallel machines **-Simplify the design of the communication library and its implementation** **-Minimize/eliminate non-determinism during the execution of MPI programs** **-Automatic functional and performance debugging of MPI programs** **-Minimal performance penalty**	-NIC-based implementation on state-of-the-art hardware (low level of intrusion) -Integrated Monitoring and Debugging System which provides different levels of non-determinism -Most existing scientific codes run efficiently with BCS-MPI (based on MPICH)	-Improved Functional Debugging -Job Prioritization -μKernel Implementation -Checkpointing -Fault Tolerance

Design

Intuition: a SIMD communication library runs MIMD MPI programs.

Hierarchical design based on a basic set of communication/synchronization primitives.

Global scheduling of computation, communication and synchronization operations for MPI user code: Global Heartbeat (500µsec time slices).

System activitities are organized in microphases within every time slice.

NIC-based OS-bypass implementation.

Scalability is facilitated by tightly coupling the collective communication operations with the collective primitives provided by the hardware.

Integrated Monitoring and Debugging Mode which provides selectable level of non-determinism (in the strictest mode, the system is able to rerun an arbitrary large parallel program in a completely deterministic way).

Integration as a plugin in a resource management system for parallel jobs.

Figure 1
Blocking Communication

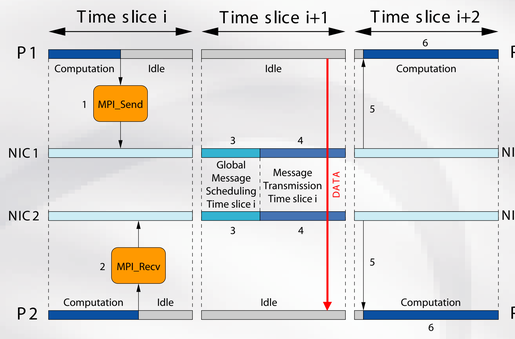


Figure 1. Blocking Send/Receive Scenario.
Process P1 sends a message to process P2 and P2 receives a message from P1:
1) P1 posts a send descriptor to the NIC and blocks.
2) P2 posts a receive descriptor to the NIC and blocks.
3) The transmission of data from P1 to P2 is scheduled since both processes are ready (all the pending communication operations posted before time slice i are scheduled if possible).
4) The communication is performed (all the scheduled communication operations are performed before the end of time slice i+1).
5) P1 and P2 are restarted at the beginning of time slice i+2.
6) P1 and P2 resume computation.

Note that the delay per blocking primitive is 1.5 time slices on average.

Figure 2
Non-Blocking Communication

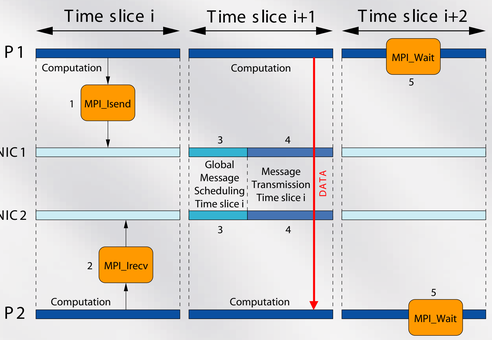


Figure 2. Non-Blocking Send/Receive Scenario.
Process P1 sends a message to process P2 and P2 receives a message from P1:
1) P1 posts a send descriptor to the NIC.
2) P2 posts a receive descriptor to the NIC.
3) The transmission of data from P1 to P2 is scheduled since both processes are ready (all the pending communication operations posted before time slice i are scheduled if possible).
4) The communication is performed (all the scheduled communication operations are performed before the end of time slice i+1).
5) P1 and P2 verify that the communication has been performed and continue their computation.

In this case, the communication is completely overlapped with the computation with no performance penalty.

Performance Evaluation

Cluster Configuration

- 32 HP rx2600 compute nodes
- 128-port Quadrics switch

Compute Node Configuration

- Dual Itanium-II processor
- 2 GB of ECC RAM
- 2 133MHz/64-bit PCI-X buses
- 2 Quadrics QM-400 Elan3 NIC
- 100 Mbit Ethernet NIC

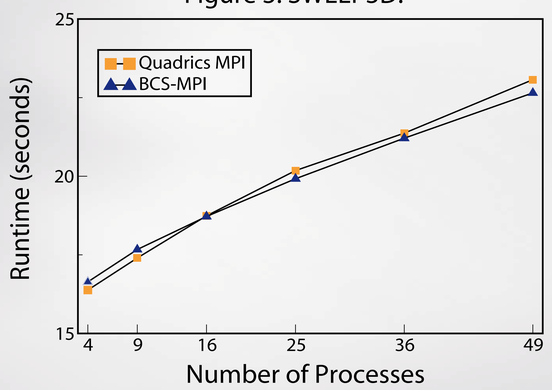
Software Configuration

- Red Hat Linux 7.2
- Intel C/Fortran 7.1.17
- SWEEP3D (50x50x50)

Results

- Comparison between BCS-MPI and Quadrics MPI for different numbers of processors.

Figure 3. SWEEP3D.



BCS-MPI provides similar performance of a production-level MPI, with a much simpler design and implementation!!!