Message Passing in the 1990's: Performance, Safety, Correctness*

Anthony Skjellum†

NSF Enginering Research Center
& Computer Science
Mississippi State University
Mississippi State, MS

Brian K. Grant

Department of Computer Science
& Engineering
University of Washington
Seattle, WA

Origins of Message Passing

First-generation message passing systems (*e.g.*, NX, Vertex, PICL, Express) were designed with vendor-specific goals, performance characteristics relative to their underlying hardware, and limited notions of safety (*i.e.*, tag management). Emerging from this era are common support for blocking and non-blocking point-to-point communication, global operations over the ensemble of processes (often processors), and basic process management. Typically, the point-to-point messaging primitives operated on user-supplied (*e.g.*, NX) or system-managed buffers (*e.g.*, Reactive Kernel). Users were to allocate and free message buffers, align data, convert data (as needed) to/from heterogeneneous host processes, and guarantee that message contents aligned on sender and receiver. Originally, the recipient was required to know a message's size before it could be accepted; later probe mechanisms were provided more uniformly. Receipt selectivity was either on tag only, or on both tag and sender. Processes were typically named with hardware-oriented `{node,pid}` naming; the trend to limiting users to one process per processor increased rather than decreased during the 1980's. The software overhead of these systems was 50-200 times the latency of the underlying messaging networks. Transmission protocols were typically three-way, offering reasonable flow-control and reliability.

In parallel with the development of multicomputer message passing came the development of network file systems in the workstation environment and widespread use of workstation clusters. These developments created the need for heterogeneous data-transmission and simplified interprocess-communication mechanisms. Sun's XDR service became the de facto standard for a machine-independent data representation, and nearly all communication systems were based on the Internet-standard TCP/IP Protocol Suite. Unfortunately, most XDR and TCP/IP implementations had high overheads and were not suitable for high-performance communication.

Evolution

PVM emerged as one of the most popular cluster message-passing systems in 1992, though PVM did not originally work on nodes of multicomputers (more recently, multicomputer vendors have offered both layered and native versions of PVM for multicomputer message passing). PVM's model and semantics closely resembled those of systems such as PICL and NX from the mid to late 1980's. PVM's main contribution was its broad interoperability across workstation platforms. Another attraction was the portability of its TCP- and XDR-based implementation. However, its programming paradigm offered no significant advantage over the paradigms of the earlier systems.

By way of contrast, the MPI standardization effort had its roots in other systems, particularly PARMACS and Zipcode, which included higher-level concepts in their programming models. Formalization of message-passing models, notions of correctness in loosely synchronous programming, and considerations of safety had become concerns of programmers building significant applications with message passing, which were ignored by the early systems and also by PVM. Furthermore, MPI was to incorporate concepts that would deliver higher performance to the user; these concepts were the product of experience with higher-level message systems (*e.g.*, invoices (aka buffer descriptors), powerful collective operations).

*Parallel Computing on a Network of Computers — A Minisymposium during Supercomputing '93, Invited Presentation.

†To whom correspondence should be addressed. e-mail: *tony@cs.msstate.edu*

Comparison of PVM and MPI

PVM provides the following types of services:

- Process management for clusters,
- Dynamic process creation,
- Dynamic group management,
- Point-to-point communication,
- Multicasts and barriers,
- Explicit data conversion capabilities for users.

Things to think about when choosing PVM:

- User specification of communication protocols,
- User management of buffers and data conversion,
- Close ties to XDR for data conversion,
- Excess data motion, even in absence of heterogeneity,
- Lack of full set of collective communication operations,
- Lack of group/context safety mechanisms found in Zipcode and MPI,
- Race conditions involving dynamic groups,
- Race conditions involving virtual machine management,
- Lack of thread safety,
- PVM does not support a model of parallel I/O,
- PVM's cluster software is extremely portable among Unix and Unix-like systems.

MPI provides the following types of services:

- Groups of processes,
- Communicators for group-scope message passing (group+context),
- Point-to-point operations with communicator scope,
- Collective operations with communicator scope,
- Virtual topology for logical process naming,
- Thread safety.

Importantly, the gather-scatter semantics of MPI provides the ability to:

- Optimize data motion when sending and receiving messages,
- Hide heterogeneous conversions from the user,
- Relieve the user of buffer management.

Group scope in MPI guarantees that special properties of hierarchies can be exploited:

- Homogeneous subgroups can avoid data conversions,
- Specialized communication modes can be exploited where available (*e.g.*, channels, local shared-memory clusters).

The collective operations allow the user to exploit high performance collective capabilities of certain architectures, while avoiding the need to develop collective calls from point-to-point calls on all systems (slow, error prone).

Things to think about when moving to MPI1:

- MPI1 process management is left to implementations,
- MPI1 does not enforce a model of parallel I/O,
- MPI1 does not guarantee interoperability between implementations,
- MPI1 will provide a stable core for future standardization efforts,
- A user-manual is under development,
- A coherent, useful subset is specified alongside the standard,
- Several (but not all) vendors have committed to implementing MPI1,
- A portable cluster implementation of MPI1 is under development.

Finally, MPI documents outline a logical programming paradigm for writing parallel libraries and applications in the safest possible way without explicit compiler support for parallelism.