Cray Message Passing Toolkit

Heidi Poxon, Cray Research, Inc., Eagan, Minnesota, USA

ABSTRACT: *The Cray Message Passing Toolkit consolidates Cray Research, Inc. (CRI) support for message-passing and data-passing into a single software package. The new toolkit provides optimized implementations of PVM, MPI, and SHmem data passing software for Cray parallel vector (PVP) machines, increasing performance and portability across Cray PVP and MPP systems. This paper provides a status of the toolkit, and offers performance results for its components.*

1 Introduction

The Cray Message Passing Toolkit (MPT), introduced in 1Q96, consolidates support for both message passing and data passing programming models into a single software package. The intent of this new separately licensed software package is to provide highly optimized, consistent message passing and data passing programming models across CRI systems within a single package. The Message Passing Toolkit consists of the following components:

- Parallel Virtual Machine (PVM)
- Message Passing Interface (MPI)
- Logically shared, distributed memory (SHMEM) data passing capability

2 MPT release schedule for 1996

Three releases of MPT are currently planned for 1996. MPT 1.0 was released in February. The first release of MPT includes optimizations that target fast intra-node and inter-node communication (where a node is defined to be a single CRI system) on CRI's parallel vector (PVP) machines. This initial release includes features such as the first release of MPI for Cray PVP systems, a shared memory version of PVM, internal optimizations to PVM to improve communication across sockets, and support of a subset of the SHMEM routines originally developed for the Cray T3D machine.

MPT release 1.1, with planned availability in 2Q96, will contain a shared memory version of MPI providing high-speed intra-node communications between processors on Cray PVP systems similar to that of the shared memory version of PVM available in MPT 1.0. MPT 1.1 will also contain homogeneous PVM support for the T3E (stand-alone mode only.)

MPT release 1.2, with planned availability in 3Q96, will contain homogeneous MPI support for the T3E, as well as heterogeneous support for PVM on the T3E (includes PVM daemon support on T3E.)

3 Features included in MPT 1.0 release

CRI's continuing support for the message passing style of programming is shown with the release of a message passing toolkit that provides implementations of both de-facto standards: PVM and MPI. Software included in the first release of MPT was designed to be used with the Cray Programming Environment, release 2.0. MPT also makes use of the Modules software package, providing easier access to the software. The following subsections describe enhancements to the individual components of the package.

3.1 PVM enhancements

The version of PVM released with MPT replaces Cray Network PVM-3. This new version of PVM is based on the public domain version of PVM, version 3.3.7, released jointly by developers at Oak Ridge National Laboratory (ORNL), Emory University and the University of Tennessee. MPT 1.0 PVM contains the following architecture-specific enhancements that target Cray PVP systems.

3.1.1 Communication optimizations

By default, communication in MPT PVM between processes on the same machine is based on data transfers using UNIX domain sockets, and on TCP sockets between processes across different machines. The change to use UNIX domain sockets instead of TCP sockets (as used until recently in Cray Network PVM-3) within a machine helps to reduce communication overhead. Communication overhead is still significant however. Because of this, in addition to the default communication mechanism, the MPT implementation of PVM uses memory loads and stores as an alternate mechanism for communication

between processes on the same machine. Using memory for communication is faster than sockets because it doesn't involve the operating system, and is achieved with the use of CRI's macrotasking software. Macrotasking was chosen to imitate a shared memory system because current CRI systems do not support globally accessible memory segments, or System V_{TM} shared memory.

If a program runs across multiple machines, the default communication mechanism can be used, or if the machines have HIPPI connections, communication can be based on HIPPI to reduce communication overhead. A program uses the HIPPI device by specifying it in the PVM host file.

3.1.2 Shared memory PVM

The shared memory version of PVM uses CRI's macrotasking software to allow communication via memory on PVP systems. The goal is to preserve the original PVM message passing environment, where all data is private to each PVM task, as much as possible while providing aggressive performance improvements, greatly increasing communication bandwidth and reducing latency. Two modes of execution are provided with shared memory PVM. The mode that offers the best performance is the stand-alone mode that requires no PVM daemon. This mode of execution is similar to that of PVM for the Cray T3D. The second mode of execution allows PVM tasks that are spawned to be run in a multitasking environment. This mode of execution is useful for PVM applications where the spawned tasks do a lot of communication amongst themselves because this communication can now be done using memory loads and stores. Communication to the parent task or to the daemon however is still done using sockets.

Running in a multitasking environment can change the behavior of a message passing program because the program is now running within a single user address space. For example, all members of the multitasking group can access global or static data (in other words global or static data becomes shared between the PVM tasks.) In a message passing environment however, all data is private to each PVM task. To preserve this message passing environment, special support has been added to MPT and the Programming Environment for both Fortran and C programs so that expected behavior is preserved, and so that source code changes to programs running in a multitasking environment are minimized. This special support includes new `f90 -ataskcommon` and `cc -htaskprivate` compiler command line options to convert all global or static data normally shared between tasks in a multitasking environment, to data that is private to each PVM task. It also includes a fully reentrant libc, and support via the assign command to make Fortran I/O unit numbers private to each task.

With automated help from CRI software, the bulk of the changes needed to use the shared memory version of PVM comprise of converting the program from multiple executables to a single SPMD-like executable, and removing calls that interface to the PVM daemon, such as `pvm_spawn()` function calls.

In addition to the increased bandwidth and reduced latency achieved by running a program with the shared memory version of PVM, a benefit of this new mode for PVPs, is the increase in portability across Cray PVP and MPP systems. With shared memory PVM, it is possible to run programs currently written for the T3D on PVP systems with only minor source code changes. This can be useful for debugging or preparation work as programs can be run on either platform, allowing developers to work with a single version of the source.

3.1.3 Miscellaneous internal optimizations

System calls are the largest source of communication overhead in PVM. To address this overhead, several internal optimizations have been added to PVM that result in increased bandwidth and reduced latency for programs using standard PVM (as opposed to shared memory PVM) either within a machine or across multiple hosts in a virtual machine. These optimizations include a reduction in the number of system calls associated with the `pvm_send()` and `pvm_recv()` functions, an optimization to reuse memory where possible to reduce the number of memory management calls, and the inlining of some commonly called internal PVM functions to reduce function call overhead.

In addition to system calls which contribute to PVM overhead, the size of data transfers across the network also impacts performance. Because of this, a new `PvmWinShift` `pvm_setopt()` option has been added so that users have more flexibility in determining the size of send and receive buffers in TCP. Using both the `PvmFragSize` and `PvmWinShift` options can increase the amount of data transmitted in TCP, resulting in a reduction in the number of system calls in PVM, and therefore lower communication overhead.

3.1.4 NQE load balancing support

Another optimization that has been added to the MPT PVM software to improve communication across multiple machines, is the integration of Network Queuing Environment (NQE) load balancing support for `pvm_spawn()` calls. If NQE is enabled on a system, the load balancing server rates the hosts specified in the PVM host file. PVM tasks are placed on the various machines based on these host ratings. Characteristics of a machine that influence its ratings include processor speed, current load on the system, number of CPUs, etc.

3.2 MPI

CRI introduces support for MPI, a standard specification for message passing libraries, on its PVP systems in the first release of the Message Passing Toolkit. CRI's implementation of MPI for its parallel vector machines is based on MPICH, a public domain implementation developed jointly by Argonne National Laboratory and Mississippi State University. MPI included in the first release of the Message Passing Toolkit, includes internal performance enhancements similar to those applied to MPT PVM. These initial optimizations are the first phase of enhancements to MPI on Cray PVP systems, adding improved performance over the public domain implementation.

Architecture-specific optimizations which will greatly increase bandwidth and reduce latency, will be provided in the MPT 1.1 release.

In addition to standard MPI library functionality, MPT also includes the Multiprocessing Environment (MPE). MPE, originally created by developers of MPICH, contains a set of extensions to MPI including profiling and tracing interfaces designed to aid in the development and debugging of MPI applications.

3.3 SHMEM data passing support

The MPT software package also includes the libsma library for Cray PVP machines. This library contains a subset of the SHMEM functions that are available today on the Cray T3D system. The library offers extremely fast communication between processors that exist on a single PVP system, and offers increased portability across CRI platforms for programs that use SHMEM data passing capability on the Cray T3D. The following libsma functions are supported on PVP systems:

- `shmem_put()`
- `shmem_get()`
- `shmem_my_pe()`
- `shmem_n_pes()`
- `shmem cache routines`
- `shmem single word put routines`

The SHMEM library functions can be used alone in a program on PVP systems, or as an optimization on "hot spots" in PVM or MPI programs to greatly reduce communication costs since the advantage of SHMEM data passing support on PVP systems is its extremely low latency.

3.4 Miscellaneous MPP compatibility functions

The following additional routines are supported in MPT to increase portability between Cray PVP and Cray T3D systems, helping to provide a consistent programming model between the PVP and T3D systems. They can be used with shared memory PVM or SHMEM data passing functions in MPT 1.0, and may be useful with shared memory MPI when it is available. These additional functions minimize the number of source code changes required when moving applications that use the message passing or data passing programming models on both CRI's MPP and PVP machines.

- `my_pe()`
- `num_pes()`
- `barrier()`
- `start_pes()`

These functions are available in both Fortran and C. The `start_pes()` function is used to start shared memory tasks since there is no `mppexec` or `-X` load-time option as on the Cray T3D.

4 MPT 1.0 performance

The goal of the first release of the Message Passing Toolkit was to address both functionality and performance on Cray PVP systems. This section describes performance results for the various MPT components.

4.1 PVM / SHMEM performance

Both inter-node and intra-node communication overhead are addressed with the enhancements to PVM and the introduction of SHMEM for PVP systems. Performance measurements have been gathered comparing Network PVM-3, version 2.1 (essentially the public domain implementation of PVM), the new MPT PVM in standard mode (run with the PVM daemon) and the new MPT PVM run in shared memory mode (T3D style with no PVM daemon.)

Various styles of pingpong tests have been run to measure latency and bandwidth improvements. Measuring bandwidth, it was found that using shared memory PVM can yield up to a *45 times* improvement in bandwidth over the public domain or Cray *network* version of PVM. Bandwidth achieved when running with shared memory PVM is now mostly limited by the memory bandwidth of the machine. Communication latency was reduced up to *4 times* with the shared memory PVM. Using the SHMEM library can reduce this latency another *order of magnitude*. (Bandwidth is the same for SHMEM or shared memory PVM.)

Inter-node communications were also greatly improved as can been seen when comparing network PVM to the new MPT PVM. Using MPT PVM between 2 J90 systems over a HIPPI connection can yield at least a *2 times* improvement in bandwidth over network PVM.

4.2 MPI performance

Similar pingpong tests were run to compare the performance between the public domain implementation of MPI (MPICH) and CRI's MPT implementation of MPI (MPICH plus optimizations) on a single J90 system, and between 2 J90 systems. Even though heterogeneous MPI is not addressed as a part of the MPI standard, implementations such as MPICH have provided added capability to run MPI applications across multiple machines. The implementation of MPI found in the Message Passing Toolkit supports the running of jobs across Cray PVP systems. Support for this inter-operability is based on MPICH's P4 layer. Performance results showed that MPT MPI can yield more than a two-fold increase in bandwidth over the public domain implementation both within a machine and across two machines. The higher bandwidth and lower latency measured on PVM programs is achieved with the shared memory version of PVM, and since a shared memory MPI does not exist in MPT 1.0, performance improvements for MPI applications are not yet as great. Much larger bandwidth and latency improvements are expected for MPI in the 1.1 release of the Message Passing Toolkit with the introduction of shared memory MPI.

4.3 MPT performance numbers

The following tables show MPT message passing and data passing performance results measured within a single J90 and between two J90 systems. Performance was measured using a round-trip pingpong test with a fragment size of 600,000 bytes and the direct routing option available in PVM. The bandwidth listed below is in Mbytes per second, and the round-trip latency is given in microseconds.

Table 1. PVM / SHMEM (single J90)

PVM	Bandwidth	Latency
Cray Network PVM-3	11.0	3880.0
MPT PVM	32.1	2717.0
Shared memory PVM	715.5	284.0
SHMEM	715.6	4.6

Table 2. PVM (between 2 J90s)

PVM	Bandwidth	Latency
Network PVM-3 (FDDI)	3.5	6531.0
Network PVM-3 (HIPPI)	8.5	4343.0
MPT PVM (TCP default)	5.4	6238.0
MPT PVM (FDDI)	5.4	5598.0
MPT PVM (HIPPI)	20.4	3485.0

Table 3. MPI (single J90)

MPI	Bandwidth	Latency
MPICH	7.6	3862.0
MPT MPI	14.5	3974.0

Table 4. MPI (between 2 J90s)

MPI	Bandwidth	Latency
MPICH (FDDI)	3.8	7506.0
MPICH (HIPPI)	5.8	4930.0
MPT MPI (FDDI)	3.9	7138.0
MPT MPI (HIPPI)	13.2	4656.0

5 Summary

The initial release of the Message Passing Toolkit targets Cray PVP systems and offers improved performance for message passing programs, and increased portability for PVM message passing and SHMEM data passing programs between Cray PVP and MPP systems. The toolkit includes extensive optimizations to PVM over CRI's existing Network PVM-3 software, and introduces new MPI and SHMEM data passing support for Cray PVP systems. The intent of the toolkit is to provide architecture specific, highly optimized implementations of both message passing and data passing software, while increasing programming model portability between Cray MPP and PVP platforms. The increase in portability allows users to work with a single version of their source code which runs on multiple Cray platforms.