PVP SHMEM and PVM Performance Issues Using MPT

Enrique Lopez-Pineda, Benchmarking Services, Cray Research, Inc., Eagan, Minnesota USA

ABSTRACT: *An overview of usage and performance issues concerning the porting of CRAY T3D (shared-memory/PVM) and network (PVM) parallel aplication codes onto Parallel-Vector Processor (PVP) machines using MPT - a message passing toolkit which has been developed to port such codes to a CRAY PVP machine.*

Introduction

Parallel Applications

Parallel applications written for distributed-memory computers often need to be ported to a Cray Research Parallel-Vector-Processor (PVP) platform. These applications are typically developed using message passing with PVM (non-network or network PVM) or, when coming from the Cray T3D, using logically shared data passing with the Shared Memory library (SHMEM). It is desirable to have this porting done with a minimum amount of effort. Furthermore, it is important that this effort maximizes the machine's resources.

Message Passing Toolkit

The Message Passing Toolkit (MPT) was developed by Cray Research to consolidate support for message and data passing into a single software package. It allows the running of parallel applications written in PVM, SHMEM (as well as in Message Passing Interface - MPI) on a shared-memory platform while maintaining the original code's structure and focus. MPT provides optimized implementations of its components, and is designed to preserve the distributed memory concept and usage, thus allowing the user to run a parallel code on a PVP system with minimal effort.

User Issues

In order to utilize MPT, there are a couple of porting issues which must be addressed. However, most importantly, there are performance issues which should be understood in order to evaluate the overall performance of a particular code. This paper presents some of these issues in order to provide a better understanding of the SHMEM and PVM components of MPT.

MPT Usage

Cray Programming Environment

The MPT package, as part of the Cray Programming Environment, contains three main components which allow the capability of message and data passing models to work on PVP systems: PVM, libsma (SHMEM), and MPI. The Cray Programming Environment uses the Modules package to combine compiler and application tools into an integrated environment. The integration of MPT onto the Cray Programming Environment is accomplished through a single module load once the Programming Environment is loaded. By doing this module load, the user is ready to compile, load and execute a parallel program.

Getting Started

Once MPT is loaded into the Cray Programming Environment, there are a couple of code adaptations which must be done. First, a call to the routine *start_pes* near the beginning of the program must be added. This call starts a processing element (PE) task as an additional multitasked task. The call requires one argument to indicate how many PE's to start. The number of PE's can be dynamically set by using the environment variable NPES and setting the argument to 0.

Another coding issue is data locality; the programmer must insure that data is allocated as private to each task since data and message passing models precisely work with the assumption that all data is indeed private. This is in contrast to shared memory multitasking, where global and static data is shared among all tasks. When compiling with Fortran 90 (f90), this can be accomplished through the command line by using the '-ataskcommon' option (when using cc with C programs use '-htaskprivate'). When compiling with Fortran 77 (cf77), one must change all common blocks to taskcommon.

When using PVM, one can use the stand-alone version (SPMD model) which does not require the PVM daemon to execute. This implies that the master and slave programs used in network PVM should be converted to a single SPMD executable, replacing the call to *pvm_spawn* with a call to *start_pes*. However, MPT also recognizes *pvm_spawn* so network PVM can also be run.

Running with MPT

After a program has been modified, compilation is trivially done as usual, depending on the PVP compiler of choice (f90, cf77, cc). Hence, execution is done simply by typing the name of the executable. One must only remember to set the environment variable NPES to the number of tasks desired (MPT allows a maximum of 32 tasks).

It is important to remember that these PE tasks from MPT are still multitasked PVP tasks and not PE MPP tasks. This simply implies that an MPT run may not necessarily execute all PE tasks in parallel (especially true if the number of PE tasks is greater than the number of CPUs available on the system).

Performance Evaluation

Performance Issues

The implementation of a message or data passing model on a PVP system allows for the development of parallel applications on a platform which can enhance the performance of certain codes (as are, for instance, highly vectorized parallel codes). The total performance, however, will highly depend on how fast the data is passed/sent, and how efficiently synchronization of tasks is handled.

Measurements of the latency, bandwidth, barrier performance, and global communications are essential for a proper evaluation of the expected performance of MPT. Actual code computations are extraneous to MPT and thus should be dealt with according to other measures (as are Mflop/s, etc.).

In the following sections, tests to determine measurements were done on a CRAY T3D with 128 PE's and on a CRAY J90 with 32 CPU's.

Latency and Bandwidth

The startup time for a message, the latency, and the sustainable data transfer rate, the bandwidth, are the most crucial elements of any data or message passing model. A short latency is necessary to insure high efficiency of even small data packets. A high bandwidth rate allows for fast transfer of data.

On the CRAY T3D, hardware and network elements allow for low latency and high bandwidth. Since on a PVP system using MPT network transfers are simply memory to memory access, one would expect higher bandwidth but a higher latency.

In order to measure latency and bandwidth, a point-to-point communications program was used. By sending messages of various sizes back and forth between two PE's, the asymptotic rate of transfer is tracked and the latency determined.

Table 1 shows the latency and asymptotic bandwidth results for this measurement.

Table 1. Latency and Bandwidth Rates

	MPT Performance		
System	Library	Latency (μ secs)	Bandwidth (Mbytes/s)
CRAY T3D	PVM	230	26
	SHMEM	2	130
CRAY J90	PVM	450	340
	SHMEM	15	700

With MPT we observe that the bandwidth rate is indeed quite high. However, because of the higher latency, the higher bandwidth requires larger data packets. The PVM and SHMEM components of MPT should achieve similar peak bandwidths on a PVP system. However, the timings for the tests used here include the time for packing and unpacking of data by PVM sends and receives. Thus, the bandwidth rates appear lower in the measurements. Also, these tests include a couple of necessary barriers hence increasing the latency of MPT. Finally, the T3D achieved better transfer rates for smaller packets when comparing corresponding libraries.

Barriers

Task synchronization through barriers is a necessary mechanism to insure proper functionality of data and message passing applications. PE tasks wait at a barrier in the code until all PE's reach the barrier. The ability of a platform to handle barriers quickly is essential to guarantee efficient performance of a parallel application.

In order to measure the barrier synchronization rate, a test was used with multiple barriers. This test was repeated for various PE's. Increasing the number of PE's clearly increases barrier time.

Table 2 shows the barrier time and million barrier rate per second (Mbarr/s).

Table 2. Barrier Rates

	MPT Performance		
System	No. PE's	Barrier Time (μ-secs)	Barrier Rate (Mbarr/s)
CRAY T3D	2	0.65	1.538
	4	0.67	1.483
	8	0.72	1.398
	16	0.74	1.353
CRAY J90	2	7.6	0.130
	4	10.6	0.094
	8	27.0	0.037
	16	46.6	0.021

The barrier for MPT uses the default batch system barrier which relies on the routine *barsync* for queueing up waiting tasks. This works best for a heavily loaded system. However, a fetch-and-increment barrier using registers performs an order of magnitude better, and is expected to be implemented on a future release of MPT. On the T3D, barrier time increases logarithmically with the number of PE's while with MPT barrier time increased at approximately the same rate as the number of PE's. Given that the CRAY T3D has special hardware mechanisms for fast barriers, while MPT relies on software to perform the barrier synchronization, this is no surprise. However, this is an indication that barriers should be kept to a minimum.

Global Communications

A set of global reduction functions was used to measure overall global communication performance. These functions were designed to perform a specific one word reduction on all PE's by doing an all-put, all-get from the master PE (PE 0). This type of reductions are typical of library routines used on massively parallel systems. The tests used do all the communication using SHMEM routines.

The measurements shown on Table 3 correspond to functions which perform a global sum (GSUM), find the global maximum (GMAX), and perform global logical AND (GLAND).

Table 3. Global Communications Rates

	MPT Performance			
System	PE's	GSUM (millisecs)	GMAX (millisecs)	GLAND (millisecs)
CRAY T3D	2	0.027	0.012	0.008
	4	0.027	0.014	0.010
	8	0.030	0.016	0.011
	16	0.034	0.022	0.016
CRAY J90	2	0.039	0.039	0.039
	4	0.044	0.044	0.044
	8	0.079	0.058	0.058
	16	0.098	0.090	0.090

These results again reflect the special hardware latency hiding and synchronization mechanisms of the CRAY T3D. However, the CRAY J90 does not really lag behind too much and it should be noted that more general reductions allowing for a sequence of values per PE increase the length of the data packet hence allowing for better results on the PVP machine. As evidence, a global vector sum was performed, with each PE having a sequence of 100000 numbers to add. Note that this is a standard implementation of a log2 reduction, and not the T3D shared memory library routine, which is quite faster. Table 4 shows these results.

Table 4. Global Vector Sum

	MPT Performance	
PE's	CRAY T3D millisecs	CRAY J90 millisecs
2	43.53	17.89
4	87.71	30.67
8	133.30	64.71
16	178.07	181.88

Finally, as an example of an application porting, the NAS Parallel multigrid (MG) benchmark was taken and modified for MPT. On 16 PE's, this code runs in 13.127 seconds on the CRAY T3D, while taking 7.611 seconds on the CRAY J90 using MPT with 16 PE's.

Summary

The Message Passing Toolkit, released in early 1996, is a tool which allows nearly transparent porting of parallel applications to PVP systems. These applications can expect a good level of performance as long as the advantages of the PVP platform using MPT are exploited using long data packets, while maintaining barrier synchronization to a minimum.

Acknowledgments

Recognition and gratitude is extended to Charles Grassl for supplying barrier and latency tests; to Margaret Cahir for supplying global communications functions; to Paul Hastings for his special contributions; and to Heidi Poxon for her insight on SHMEM MPT libraries.