A Generalized Portable SHMEM Library for High Performance Computing

Krzysztof Parzysz\textsuperscript{1}, Jarek Nieplocha\textsuperscript{1,}\textsuperscript{2}, Ricky A. Kendall\textsuperscript{1}

\textsuperscript{1}Scalable Computing Laboratory, Ames Laboratory
Department of Computer Science, Iowa State University, Ames, IA 50011 USA
kparz@scl.ameslab.gov, rickyk@ameslab.gov

\textsuperscript{2}Advanced Research Software Group
Environmental Molecular Sciences Laboratory, Pacific Northwest National Laboratory, Richland, WA 99352 USA
jarek.nieplocha@pnl.gov

Keyword: SHMEM, Tools and environments for software development, Parallel programming paradigms, Parallel/distributed algorithms

Abstract

This paper describes a portable one-sided communication library GPKGSHMEM that follows the interfaces of the successful SHMEM library introduced by Cray Research Inc for their distributed memory systems: the Cray T3D and T3E. The portability is achieved by relying on ARMC, a low-level communication library developed to support one-sided communication in distributed array libraries and compiler run-time systems, and the MPI message passing interface. The paper discusses implementation, requirements, and initial experience with GPKGSHMEM.

1 Introduction

Most scientific applications have a much longer life cycle than the computational resource and even the specific programming model used for their development. The one-sided communication mechanisms made available to scientists over a wide variety of computational hardware have found wide spread but sometimes sporadic utilization in applications as they have been developed. The most popular programming model is message passing but there are sometimes serious limitations to this model that are imposed by algorithmic choices and the development of message passing based applications is often quite error prone.

Cray Research Inc. introduced the SHMEM library in 1993 to support the global address space programming model on the distributed-memory architecture of the Cray T3D [1]. The global address space model combines some characteristics of the shared-memory programming model (direct access to non-local data) with the distributed memory model (data must be explicitly distributed and managed by the programmer). For example, on the T3D any process was able to read or write data from any other process at any time. For some applications, the global address model is more suitable than the message passing. Many of them are characterized by irregular data structures, and dynamic or
unpredictable data access patterns. On the Cray T3D and its follow-up model Cray T3E, SHMEM library has been the most efficient communication library with other programming interfaces (including PVM, MPI) implemented directly upon it. A common porting/optimization strategy for these platforms included replacing message-passing calls with SHMEM calls. The excellent performance of SHMEM on these systems attracted many applications and prompted other vendors to provide this interface on their systems. SGI, upon purchase of the CRI, incorporated SHMEM into their Message Passing Toolkit (MPT) and offered it on both the IRIX/MIPS and UNICOS or UNICOS/mk systems (T3E, J90, T90, and SV1). SHMEM has also been provided on the Quadrics-based Linux clusters on the Compaq Alphaserer clusters [2]. In addition, HPVM package included a SHMEM-like library as one of its several interfaces [3] initially on the Linux and later on the Windows platform only [4]. Advanced Computing Technology Center of IBM developed a subset of the SHMEM library on top of the IBM LAPI library primarily for internal use as a porting tool of the Cray-based applications to the IBM SP [5].

This paper describes a communication library called GPShmem that follows the interfaces of the original Cray SHMEM library but attempts to achieve full portability. We have been developing this system to: 1) provide a porting tool of the existing codes that use SHMEM, and 2) offer an alternative interface to the MPI-2 1-sided model that for some applications might be considered to be too restrictive or lead to unacceptable performance degradations. The paper is organized as follows: Section 2 describes the Cray SHMEM library and other related interfaces, Section 3 focuses GSHMEM assumptions and implementation, and Section 4 reports performance and experience in using the library. Conclusions are given in Section 5.

2 SHMEM and related communication systems

2.1 The SHMEM model

The original SHMEM library on the Cray T3D offered four types of interfaces:

1. Noncollective remote memory operations including copy e.g., shmem_put and atomic operations e.g., shmem_swap
2. Collective operations e.g., barrier, shmem_collect
3. Environment initialization and information e.g., start_pes, _my_pe, _num_pes
4. Operations supporting memory consistency model e.g., shmem_udcflush, shmem_set_cache_inv

* Presenting Author.
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.
By default, the memory of the Cray T3D was not consistent w.r.t. SHMEM store operations. A programmer was required to deal with the consistency explicitly (essentially inserting flush operations for cache or cache line) or implicitly by setting (via `shmem_set-cache_inv`) hardware to flush cache on every remote operation targeting the given processor memory. The SHMEM operations were ordered w.r.t. destination. Later the library interfaces were extended with introduction of a new hardware i.e., Cray T3E. The adaptive routing introduced in that system meant that remote operations were no longer ordered. Therefore, new operations `shmem_fence` and `shmem_quiet` were added for applications that required ordering. At the same time, since the memory of the Cray T3E was consistent w.r.t. SHMEM operations, the cache invalidation operations became obsolete.

Implementations of SHMEM done by SGI for IRIX introduced new interfaces that attempted to address portability issues of the SHMEM library interfaces w.r.t. the datatypes (e.g., assumptions that size of C int datatype is equal to the size of C long datatype were not valid on the MIPS systems). Other projects and vendors implemented a subset of the SHMEM library of the Cray T3D and/or T3E, and introduced some extensions to that model. For example, the HPVM version added support for nonblocking get operation (`shmem_cget`, `shmem_cwait`) and explicit initialization and termination calls (`shmem_initialize`, `shmem_finalize`). The Quadrics version of SHMEM developed on top of the Elan for the Quadrics switch equipped Linux, Tru64, and Solaris systems, added an initialization (`shmem_init`), and included many of the SHMEM library extensions developed by SGI.

## 2.2 SHMEM and other models

The SHMEM library on the Cray MPP systems provided the lowest-level programming interface capable of delivering the highest performance to the applications on these systems. In addition, the one-sided communication operations offered applications a straightforward mechanism to move data without a need for cooperation between the sender and receiver mandated by the message-passing model. At the time when Cray T3D was released, the SHMEM library offered superior performance of its collective operations [6] than were available by the native to this platform message-passing library PVM. Several other vendors followed Cray by offering support for one-sided communications, MPlib on Fujitsu VPP/VX [7], LAPI on the IBM SP [8], Paralib/CJ on NEC Cenju [9]. All these interfaces implement similar progress rules as the SHMEM library but also differ in terms of specific interfaces, e.g., put/get operations use byte rather than datatype-based API of SHMEM or offer extra capabilities e.g., active messages in LAPI.

In 1997, the MPI Forum added some support for one-sided communication to the MPI-2 standard; however, its implementations are still very limited. MPI-2 offers two modes of one-sided communications: active and passive. The
"active" model requires extra synchronization (remote side has to post a synchronizing call MPI_Win_fence or MPI_Win_Post) thus it is not compatible with SHMEM where one-sided operations do not require any actions of the remote process to complete an operation. The "passive" model does not require synchronization but imposes another significant restriction: serialization of access to a remote "window". In SHMEM and other vendor-specific one-sided libraries mentioned above, there is no restriction on how many processes can simultaneously access address space of remote process. MPI-2/"passive" requires that only one process can access a partition ("window") of address space at a time, and offers locking operations (MPI_Lock/Unlock) to facilitate that constraint. Locking and remote memory serialization might not be necessary from an algorithmic standpoint for some applications, and they introduce performance penalty.

2.3 ARMCI

ARMCI was developed to support remote memory operations in the context of distributed array libraries and compiler run-time systems [10]. It is portable and compatible with message-passing libraries including MPI or PVM. Comparing to the Cray SHMEM, it puts more focus on the noncontiguous data transfers that correspond to the data structures used in scientific applications (e.g., sections of multi-dimensional dense or sparse arrays). Such transfers are optimized thanks to the non-contiguous data interfaces available for the ARMCI data transfer operations: multi-strided and generalized UNIX I/O vector interfaces. ARMCI supports three classes of operations:

- data transfer operations including put, get and accumulate,
- synchronization operations—local and global fence and atomic read-modify-write, mutex operations, and
- utility operations for allocation and deallocation of memory and error handling.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMCI_Put, ARMCI_PutV, ARMCI_PutS</td>
<td>contiguous, vector and strided versions of put</td>
</tr>
<tr>
<td>ARMCI_Get, ARMCI_GetV, ARMCI_GetS</td>
<td>contiguous, vector and strided versions of get</td>
</tr>
<tr>
<td>ARMCI_Acc, ARMCI_AccV, ARMCI_AccS</td>
<td>contiguous, vector and strided versions of atomic accumulate</td>
</tr>
<tr>
<td>ARMCI_Fence</td>
<td>blocks until outstanding operations targeting specified process complete</td>
</tr>
<tr>
<td>ARMCI_AllFence</td>
<td>blocks until all outstanding operations issued by calling process complete</td>
</tr>
<tr>
<td>ARMCI_Rmw</td>
<td>atomic read-modify-write</td>
</tr>
<tr>
<td>ARMCI_Malloc</td>
<td>memory allocator, returns array of addresses for memory allocated by all processes</td>
</tr>
<tr>
<td>ARMCI_Free</td>
<td>frees memory allocated by ARMCI_Malloc</td>
</tr>
<tr>
<td>ARMCI_Lock, ARMCI_Unlock</td>
<td>mutex operations</td>
</tr>
</tbody>
</table>
ARMCI is available on clusters on common Unix and Windows workstations/servers and MPP systems. On clusters, ARMCI can use Myrinet and Quadrics in addition to the standard networking hardware and protocols. For better portability and performance ARMCI model does no assume or require any particular implementation mechanism (e.g., threads). The library implementation tries to exploit the most efficient mechanisms available on a given platform, which might involve active messages (IBM SP), native put/get operations (LAPI on the IBM SP, SHMEM on Cray T3E and clusters with Quadrics switch, MPlib on Fujitsu), shared memory, proprietary low-level messaging layers (GM on Myrinet), sockets, and/or threads. In implementations where ARMCI forks extra threads or processes to service one-sided communication, extra care is needed to reduce CPU utilization (by using blocking than active polling of the communication interfaces).

3 Generalized Portable SHMEM (GPSHMEM)

Our generalized portable SHMEM (GPSHMEM) is implemented on top of ARMCI and a message passing library, currently MPI. ARMCI is used for one sided, asynchronous data transfers (e.g., shmem_get, shmem_put) and the message-passing library is the transport layer for collective communications. The functionality of the current implementation of GPSHMEM covers most of the SHMEM implementation available on Cray T3D (What is not implemented?). This includes shmem_barrier, shmem_bcast, shmem_coll, shmem_fcoll, shmem_get, shmem_put, shmem_iget, shmem_iput, shmem_ixget, shmem_ixput, shmem_wait and reduction routines. The reduction routines include logical operations “and,” “or” and “xor” available for short and int data types, as well as arithmetic operations sum, product, minimum and maximum available for int, short, float and double. All routines, except shmem_barrier, shmem_swap, shmem_wait and reduction routines are not type-aware. They come in the “standard” 64-bit version and 32-bit specific version. The user is obligated to take make sure that the size of the used data type in the application is coordinated with the assumed size of the library routine utilized.
By default, remote addresses used for the GPSHMEM operations must be symmetric data objects. By definition a symmetric data object is one for which the local and remote addresses have a known relationship. You can determine the address of a remote symmetric object relative to the address of the local symmetric object. Examples in the T3E environment are arrays in FORTRAN common blocks and arrays allocated with shpalloc. This requirement is present on Cray parallel vector processing systems (J90, T90) but not on Cray T3D and T3E systems where any memory address may be passed with the obvious cache dependencies outlined above for the Cray T3D system. GPSHMEM enforces this limitation and provides routines for allocating symmetric data objects where the set of pointers specific to each process is managed internally. GPSHMEM handles the translation of one “visible” data object from the calling process to the corresponding “visible” data object of any other process involved. For portability and performance reasons, all remote objects passed to the GPSHMEM routines need to be previously allocated by the provided routines: g_shmalloc for C and shpalloc for FORTRAN. They correspond directly to the CRUSGI counterparts. Deallocation routines have also been provided to free up memory no longer required by the application. No other objects, including

```
program ftndymm
implicit real*8 (a-h,o-z)
real*8 x
integer i, pe, me, nproc
call start_pes(0)
me = my_pe()
np = num_pes()
x = real(me + 1)
if (me.eq.0) then
  sum = x
  if (np.gt.1) then
    do pe = 1,(np-1)
      call shmem_get(x_rem,x,1,pe)
      sum = x_rem + sum
    enddo
  endif
  write(6,*), 'computed sum is ',sum
endif
end
```

```
program ftndymm
implicit real*8 (a-h,o-z)
real*8 x
integer (ip,x)
integer pe, me, np, len, err, abrt
me = my_pe()
np = num_pes()
err = 0
abrt = 0
len = 2
call shpalloc(ip,len,err,abrt)
x = dble(me + 1)
call barrier()
if (me.eq.0) then
  sum = x
  do pe = 1,(np-1)
    call shmem_get(x_rem,x,1,pe)
    sum = x_rem + sum
  enddo
  write(6,*), 'computed sum is ',sum
endif
end
```

**Figure 2.** SHMEM code from the Cray T3E (left) and the equivalent GPSHMEM code (right)

automatic variables or pointers obtained by a malloc call or FORTRAN90 allocate mechanism can be passed to
SHMEM routines as remote objects. Note that it also includes pointers obtained by association with arrays in FORTRAN common blocks.

Because pointers allocated by the underlying ARMCI library must be directly associated with the array address being utilized in FORTRAN application, the GPSHMEM FORTRAN interface utilizes pointers available in Fortran 90 or as an extension in Fortran 77 provided by some compilers (such as Sun Workshop, xlf compiler from IBM, or Cray Fortran). Just as the CRI/SGI SHMEM, other SHMEM-like libraries, and most message passing libraries, GPSHMEM must be properly initialized. GPSHMEM relies on the message passing environment and ARMCI to define the process manifold and the logical processor topology. GPSHMEM must also collectively and gracefully shutdown the parallel application. GPSHMEM provides two its own routines for initialization and termination, \texttt{shmem_init} and \texttt{shmem_cleanup}, respectively.

A simple example illustrating GPSHMEM and SHMEM differences is given in Figure 2. The shared data is symmetrically allocated via the call to \texttt{shpalloc} and the parallel environment is set up via the initialization calls to MPI and \texttt{shmem_init}. The GPSHMEM code can be compiled and run on the Cray T3E by simply commenting out the \texttt{shmem_init} and \texttt{shmem_cleanup} calls and making sure the communications libraries for MPI are also included in the linked executable. The code has changed little but will now run on a variety of parallel systems.

4 Experience and Results

To test the overhead of the latency a pseudo-random put and get code was constructed where the "control" program used the native SHMEM implementation on the CRAY T3E. The "test" program utilized the GPSHMEM \texttt{shmem_put}, \texttt{shmalloc} and \texttt{shmem_get} functionality layered on top of the ARMCI library. Note that the ARMCI library on the CRAY T3E calls the native Cray SHMEM. The range of the overhead from 8000 data transfer operations to 16000 operations was 3.2 to 3.8 microseconds for GPSHMEM's \texttt{shmem_put} and 2.9 to 3.1 for GPSHMEM's \texttt{shmem_get}. This demonstrates that the GPSHMEM programming model will be as efficient as the underlying ARMCI memory access implementation on any given computational resource. Where ARMCI performs well GPSHMEM will perform well.

In addition to test and example programs, we developed a matrix multiply code that uses distributed data storage of the operand matrices and the resultant product matrix. The algorithm is rather naive but does exercise the remote memory access mechanism available in the GPSHMEM library. By an arbitrary choice of the elements of the values of the array elements in the operand matrices an internal check matrix to compare the resultant product matrix can be computed a priori as well. The algorithm computes \( AB = C \); where \( A \) is stored as a group of full rank row vectors, \( B \) is stored as a
group of full rank column vectors and both the check matrix and product matrix C are stored identically to A. The local patches of panels of \( AB \) were computed and placed in the appropriate component of the product matrix C. The appropriate remote elements of B were obtained via \texttt{shmemb} and a different patch of the product matrix was updated with the product result.

We tested scaling of this code on a Pentium II cluster that has a switched fast Ethernet network at Ames. The matrices tested were of rank 512, 768, 1024, and 1536 and we used up to 32 processors. The lowest efficiency computed from a reference run of 2 processors was 97% even including the sequential startup costs.

In addition we have been using GPSHMEM to port two real applications to the Solaris and IBM SP environments. One is an MPI based Molecular Dynamics code from Ames Laboratory and the other is a conjugate gradient code from the National Renewable Energy Laboratory that solves for a selected energy range of eigen-states for a fixed potential. In our experience, the most effort was required to switch between different numerical libraries than convert the codes to use GPSHMEM.

5 Conclusion and Future Work

The GPSHMEM library was developed relying on the portability of the ARMCII and MPI. It can utilize all the underlying technology in these packages to afford applications the ability to migrate from SHMEM based platforms to a wide variety of computational systems. The library requires the user to understand which data structures must be visible and allocate them in that way. The second order effect of this analysis of one's application is that the data used in the application is logically localized to what must be communicated to the rest of the application and data that is truly local.

Although we have shown a working library that applications can use to move their SHMEM based code to general computational resources, no library can remove the physical reality of the system resources. An application highly tuned for Cray T3E systems will have to be re-tuned and possibly redesigned to work efficiently on a Beowulf-like cluster. Our first priority for future efforts will be to complete the adaptation of two applications and compare their parallel efficacy compared to their native mode of operation. In addition, the GPSHMEM library will be expanded to include general allocation support for FORTRAN77 applications. A subset of functionality from the expanded SHMEM suite as available on the T3E and IRIX systems will be integrated.
Acknowledgements

This work was performed under the auspices of the U.S. Department of Energy (USDOE) under contract W-7405-ENG-82 at Ames Laboratory, operated by Iowa State University, and under contract DE-AC06-76RLO at Pacific Northwest National Laboratory, operated by Battelle Memorial Institute. This effort was in part, funded by the Computational Science Program of the Mathematical, Information and Computational Science Division of the Office in Advanced Scientific Computing Research at USDOE. This work was also supported through the U.S. DOE by the Mathematical, Information, and Computational Science Division the Office of Computational and Technology Research DOE 2000 ACTS project, and used the Molecular Science Computing Facility in the Environmental Molecular Sciences Laboratory at the Pacific Northwest National Laboratory. This effort also used the resources of the National Energy Research Scientific Computing Center at Lawrence Berkeley National Laboratory.

References