Poster – High-Level, One-Sided Models on MPI: A Case Study with Global Arrays and NWChem

James Dinan, Pavan Balaji, Jeff R. Hammond
Argonne National Laboratory
{dinan, balaji, hammond}@anl.gov

Sriram Krishnamoorthy
Pacific Northwest National Laboratory
sriram@pnnl.gov

Vinod Tipparaju[*]
Advanced Micro Devices
tipparajuv@ieee.org

ABSTRACT

Global Arrays (GA) is popular high-level parallel programming model that provides data and computation management facilities to the NWChem computational chemistry suite. GA's global-view data model is supported by the ARMCI partitioned global address space runtime system, which traditionally is implemented natively on each supported platform in order to provide the best performance. The industry standard Message Passing Interface (MPI) also provides one-sided functionality and is available on virtually every supercomputing system. We present the first high-performance, portable implementation of ARMCI using MPI one-sided communication. We interface the existing GA infrastructure with ARMCI-MPI and demonstrate that this approach performance comparable to a native implementation and enhances portability for applications like NWChem.

Categories and Subject Descriptors: D.1.3 [Programming Techniques]: Concurrent Programming– Parallel programming; D.3.3 [Programming Languages]: Language Constructs and Features– Concurrent programming structures

General Terms: Design, Performance

Keywords: One-sided Communication, MPI, GA, ARMCI, PGAS

1. OVERVIEW

The Global Arrays (GA) [5] programming model has gained popularity in the scientific computing community, especially in the computational chemistry domain where GA has been used to support many applications, including the popular NWChem computational chemistry suite [6]. GA allows the programmer to create large, multidimensional shared arrays that span the memory of multiple nodes in a distributed memory system. The programmer interacts with global arrays through asynchronous, one-sided Get, Put, and Accumulate operations as well as through high-level mathematical routines provided by GA.

The programmer access a global array through high-level array indices; GA translates these high-level operations into low-level communication operations that may target multiple nodes depending on the underlying data distribution. GA's low-level communication is managed by the Aggregate Remote Memory Copy Interface (ARMCI) [4] one-sided communication runtime system. Porting and tuning GA for a new platform, is accomplished at the ARMCI level and ARMCI traditionally is implemented natively using each platform's low-level primitives. Because of its sophistication, creating an efficient and scalable implementation of ARMCI for a new system is challenging; and production-ready, native implementations are not yet available for several of the current top performing machines.

The Message Passing Interface (MPI) [3] is the industry standard communication runtime for high performance computing. An efficient and scalable MPI implementation is provided by the system vendor for virtually every parallel computing platform. One-sided Remote Memory Access (RMA) communication support was introduced in version 2 of the MPI standard. MPI's one-sided communication is unique in targeting extreme portability, even to non-cache-coherent architectures. This design goal, however, has lead to complexity in the model and semantic challenges that have been an impediment to adoption [1].

In this work, we present the first implementation of GA's low-level ARMCI runtime system that utilizes MPI-RMA. By harnessing the portability of MPI, we have created a highly portable, high performance runtime layer for GA that extends the usability of GA and the NWChem computational chemistry suite to a wide variety of systems, including systems where a native ARMCI implementation is not available or has not been fully tuned. In addition, although GA and MPI are commonly used together, they have separate runtime systems which leads to consumption of extra resources (e.g., duplicated pinned buffer management and asynchronous progress engines). By enabling GA to share MPI's runtime system, we achieve a greater degree of interoperability that increases the resources available to the application.

2. DESIGN OF ARMCI-MPI

We have created an intermediate global data management layer, called MPI global memory regions (GMR), to align MPI's RMA window semantics with ARMCI's PGAS model. GMR handles the allocation of shared segments and provides translation between ARMCI shared addresses and MPI windows. In addition, GMR must manage the use of global memory in ARMCI operations and direct local access in order to ensure MPI semantics are not violated. Direct load/store access to shared regions in MPI must be protected with MPI_Win_lock/unlock operations. When a shared buffer is provided as the local buffer for an ARMCI communication operation, GMR must perform appropriate locking and copy operations in order to preserve multiple MPI semantics, including the semantics that a window cannot be locked for more than one target at any given time.

Multiple strategies are possible for supporting ARMCI's strided and I/O vector noncontiguous communication operations. We describe and evaluate a conservative approach which maintains all

[*]Work conducted while at Oak Ridge National Laboratory.

SC'11 Companion, November 12–18, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1030-7/11/11.

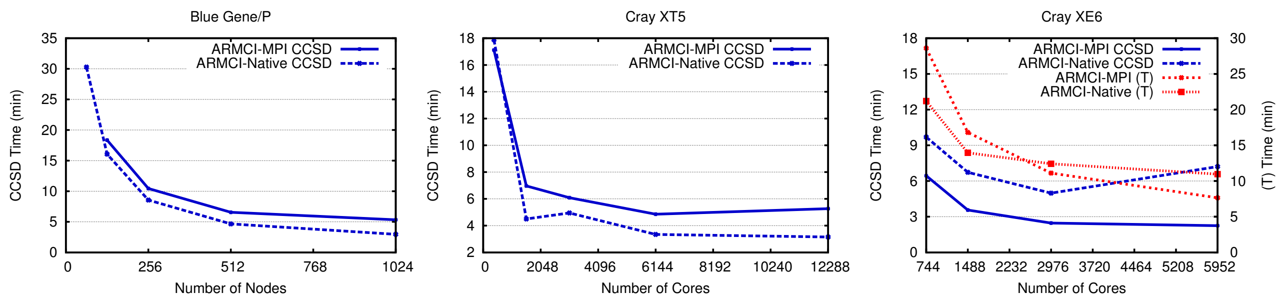

Figure 1: NWChem CCSD and (T) execution time for native ARMCI and ARMCI-MPI.

MPI semantics as well as two more optimized approaches that leverage information about the data accessed to achieve better performance. These methods perform batching of multiple communication operation in each passive mode epoch and generate MPI datatypes which can be used in one sided operations to hand off the operation to the MPI runtime which performs data packing. A low-level evaluation reveals that the datatype generation method provides the best performance on many systems, however batched dispatch of operations can yield higher throughput on systems where the data packing overhead is high (e.g. the Blue Gene/p).

ARMCI's mutexes interface was implemented on top of MPI-RMA using the algorithm of Latham et al. [2]. The implementation of ARMCI's atomic operations has exposed a significant gap in the MPI-RMA interface and were implemented with an atomic operations mutex per allocation. We discuss this and other limitations of the MPI 2.2 specification and how this work has driven the development of new features in the MPI 3.0 MPI-RMA draft specification.

3. EXPERIMENTAL EVALUATION

We have evaluated our system on several platforms that capture a broad range of the high performance computing landscape. We present here three of these systems: IBM Blue Gene/P, Cray XT5, and Cray XE6. Each of these systems currently has a native ARMCI implementation available and each represents an important execution target for NWChem.

Performance evaluation was conducted with NWChem, on a water cluster modeling problem using the CCSD(T) method and the aug-cc-pVTZ basis set. Water clusters are the subject of intense scientific interest and have been previously considered in performance-oriented studies involving NWChem. The water pentamer has 50 electrons, which are represented using 460 atom-centered Gaussian basis functions. Following convention, the 10 core electrons were frozen, that is, they were not included in the CCSD(T) calculation. Thus, $n_o = 20$ and $n_v = 435$ in the $O(n_o^3 n_v^4)$ computational cost of CCSD(T) using the spin-free formalism.

In Figure 1 we present a performance comparison between NWChem running on GA using ARMCI-MPI and GA using native ARMCI. We present CCSD computation timings for all platforms and (T) timings for the XE6. For each system, we selected the best strided method based on a microbenchmarking performance study: batched on the Blue Gene and direct on Cray XT and XE.

On the Blue Gene/P, we see that ARMCI-MPI's performance is comparable to the performance of the native implementation for the CCSD calculation and maintains good scaling. From the data we have collected, we can see that performance is roughly 15%-20% less for ARMCI-MPI. On the Cray XT, we demonstrate scaling to 12,288 processes with a similar performance trend for the CCSD calculation. Finally, on Cray XE6 we see that ARMCI-MPI performs 30% better than the current native implementation on the CCSD calculation. ARMCI-MPI also scales much better and continues to improve execution time of the expensive (T) calculation on 5,952 processors while the native implementation's performance flattens for (T) and worsens for CCSD.

In Figure 2 we present a microbenchmark evaluation of ARMCI contiguous communication bandwidth on the Cray XE6 that gives insight into the poor performance of native ARMCI on this system. From this data, we see that the communication performance of the native implementation is significantly lower than that of the MPI-RMA implementation. The performance of the MPI-RMA implementation is greater because the native implementation of ARMCI requires additional performance tuning. This demonstrates ARMCI-MPI's capability to enable early science on new platforms.

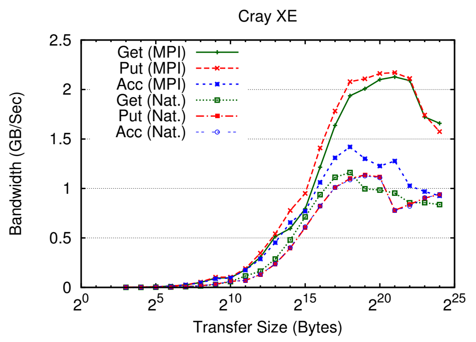

Figure 2: Bandwidth achieved for contiguous ARMCI operations using ARMCI-MPI and native ARMCI on Cray XE6.

4. REFERENCES

[1] D. Bonachea and J. Duell. Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations. *Int. J. High Perform. Comput. Netw.*, 1:91–99, August 2004.

[2] R. Latham, R. Ross, and R. Thakur. Implementing MPI-IO Atomic Mode and Shared File Pointers Using MPI One-Sided Communication. *Intl. J. High Perf. Comp. Appl.*, 21(2):132–143, 2007.

[3] MPI Forum. MPI-2: Extensions to the message-passing interface. Technical report, U. Tenn., Knoxville, 1996.

[4] J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. *Lecture Notes in Computer Science*, 1586, 1999.

[5] J. Nieplocha, B. Palmer, V. Tipparaju, M. Krishnan, H. Trease, and E. Aprà. Advances, applications and performance of the global arrays shared memory programming toolkit. *Int. J. High Perform. Comput. Appl.*, 20(2):203–231, 2006.

[6] M. Valiev, E.J. Bylaska, N. Govind, K. Kowalski, T.P. Straatsma, H.J.J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T.L. Windus, and W.A. de Jong. Nwchem: A comprehensive and scalable open-source solution for large scale molecular simulations. *Computer Physics Communications*, 181(9):1477 – 1489, 2010.