PGAS Models using an MPI Runtime: Design Alternatives and Performance Evaluation

Jeff Daily Abhinav Vishnu Bruce Palmer Hubertus van Dam

Pacific Northwest National Laboratory,
902 Battelle Blvd, Richland, WA 99352

1. INTRODUCTION

Programming models play a critical role in designing scalable applications. In the past few decades, MPI [3] has become the de facto programming model for writing parallel applications. At the same time, alternative programming models such as Partitioned Global Address Space (PGAS) programming models are gaining traction due to the asynchrony, ability to read/write distributed data structures and natural mapping for many algorithms. These models are being designed to meet the needs of upcoming revolutionary architectures such as the need to mitigate power requirements for large scale systems. PGAS models such as Global Arrays [6] and Unified Parallel C [4] are being used by several applications such as NWChem [7], STOMP and CP2K. These models rely on scalable communication subsystems such as ComEx [10] and GASNet [4] for providing Remote Memory Access (RMA) operations such as Put, Get bulk atomic Accumulate, in addition to single-value Atomic Memory Operations such as read-modify-write and compare-and-swap. These communication subsystems are readily available on several high-end communication networks such as InfiniBand [9], Cray Gemini [8] and Aries and Blue Gene/Q [10].

However, it may not be possible to provide an optimized PGAS communication subsystem for every system. In several cases, the user-level access layers (such as Verbs for IB, DMAPP for Cray Gemini/Aries) is restricted, and an open specification is not available for use. While MPI is ubiquitous and readily supported by vendors, PGAS communication subsystems are not always supported on a newer system by vendors. This is a serious restriction for applications which rely on PGAS models such as Global Arrays [6] to solve important scientific problems. Under these circumstances, the only available choice for designing PGAS communication subsystems is to use an MPI communication layer.

Vendors typically optimize MPI by in-depth evaluation of the MPI primitives which are heavily used by scientific applications, primarily two-sided (send/receive) and collective operations. Hence, two-sided semantics may appear to be a useful choice for designing PGAS communication subsystems. At the same time, MPI provides MPI-RMA as a part of the specification. However, due to sub-optimal implementations [2], MPI-RMA may suffer from significant overhead in comparison to the native ports. Additionally, MPI provides features such as Multithreading and Dynamic Process Management which can be utilized for making communication progress on RMA requests should the MPI-RMA implementation be sub-optimal. Hence, there are several choices for designing PGAS communication subsystem using MPI.

In this poster, we address the problem of designing a PGAS communication subsystem using MPI and make the following contributions: an implementation based on MPI two-sided semantics, with multithreading, with dynamic process management, and an implementation using MPI-RMA. We prove their limitations and propose a progress-rank based approach which transparently splits user processes into computation and communication ranks and uses the progress ranks for serving RMA requests. The empirical evidence with communication benchmarks and applications proves the efficacy of the proposed PR approach.

2. SUMMARY OF APPROACHES

The summary of the proposed approaches are as follows:

MPI-RMA: MPI-RMA 2.0 supports the one-sided operations *get, put* and *atomic memory operations* in addition to supporting *active and passive* synchronization modes. MPI-RMA has completely different semantics than the popularly used send/receive and collective communication interface. An important implication is that an optimal design of MPI-RMA needs a completely different approach than two-sided semantics. Since MPI-RMA has achieved low acceptance in comparison to two-sided semantics [2], most vendors choose to only provide a compatibility port due to resource limitations, resulting in sub-optimal performance.

MPI Two-Sided: In the proposed design of RMA operations using two-sided semantics, every process must service requests for data while at the same time performing computation and initiating communication requests on behalf of the calling process. As a result, this design is never allowed to make synchronous requests; all operations must be non-blocking. Otherwise, deadlock is inevitable. Furthermore, synchronization barriers and collective operations must also be non-blocking to facilitate progress while servicing requests. The primary shortcoming of this design is the requirement of explicit progress by processes, making it un-

SC '13 Denver, CO USA

suitable for irregular PGAS applications. However, it is an attractive choice since MPI two-sided semantics are heavily optimized by vendors.

MPI Multithreading: Multithreading support is a feature which allows multiple threads to make MPI calls with different threading modes. As an improvement over the previous send/receive design, progress is made using an asynchronous thread. In our proposed design, the asynchronous thread calls MPI_Iprobe after it has finished serving the send requests. We use a separate communicator for communication between process-thread and thread-process. The primary shortcoming of this approach is the contention among threads which can be alleviated by a high quality multithreaded MPI implementation. However, the common use case of MPI multithreading which decomposes the work equally among threads is not applicable to our approach, which restricts the usefulness of this approach as demonstrated in the performance evaluation.

MPI Dynamic Process Management: DPM is an MPI feature which allows an MPI process to spawn new processes dynamically. Using DPM, a new inter-communicator can be created which can be used for communication. An advantage of such an approach is that it alleviates a need to use multithreading while still providing asynchronous progress by spawning new processes. The original and spawned processes would then attach to the same shared memory region in order for the spawned processes to make progress on behalf of the processes within its shared memory domain [5]. Unfortunately, dynamic process management is not available on most high-end systems such as the Cray Gemini system used in our evaluation. Due to a lack of available implementations of DPM, we do not evaluate this approach.

MPI Progress Rank: In this approach, the user level processes are automatically *split* among ones which execute the algorithm and ones which provide the asynchronous progress. The data-centric view of the PGAS models, which does not address process ranks directly, facilitates this *splitting* without requiring any change in the application. The PR approach leverages the highly tuned two-sided semantics, alleviates any thread contention issues, is compatible with the ubiquitous MPI-1 spec, and does not rely on a potentially subpar MPI-RMA design.

3. SUMMARY OF RESULTS

Our performance evaluation reveals that the proposed PR approach outperforms each of the other MPI approaches on a spectrum of evaluation criteria: communication benchmarks, community detection kernel in graphs, sparse matrix-vector multiply and a full application, NWChem. In a select few cases MPI-RMA did perform as good or slightly better, as was the case for get performance on the IB system and a few functions profiled within NWChem. The multithreading approach showed promise in the communication benchmarks, however its performance was stagnant for a real application even though other applications using multithreaded MPI's thread multiple mode have been shown to scale well[1].

4. CONCLUSIONS

As the popularity of PGAS models continue to rise, it becomes more important that highly tuned communication subsystems are available to enable these models across a wide range of systems. This work has demonstrated that highly-tuned two-sided semantics are sufficient for implementing one-sided semantics in the absence of a native implementation. This result should continue to affirm system procurement requirements of optimized two-sided communication while suggesting that one-sided communication can be readily improved in the future using the existing MPI interface. This work narrows the performance gap between native and MPI-based runtimes for PGAS models and succeeds in making MPI-based runtimes for PGAS models an acceptable alternative when native implementations are either not feasible to implement or not readily available.

5. ACKNOWLEDGMENTS

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The research conducted was supported by Extreme Scale Computing Initiative - an LDRD Initiative at Pacific Northwest National Laboratory.

6. REFERENCES

[1] J. Daily, S. Krishnamoorthy, and A. Kalyanaraman. Towards Scalable Optimal Sequence Homology Detection. In *ParGraph*, 2012.

[2] J. Dinan, P. Balaji, J. R. Hammond, S. Krishnamoorthy, and V. Tipparaju. Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication. In *IPDPS*, pages 739–750, 2012.

[3] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[4] P. Husbands, C. Iancu, and K. A. Yelick. A Performance Analysis of the Berkeley UPC Compiler. In *ICS*, pages 63–73, 2003.

[5] M. Krishnan and V. Tipparaju. Extending the MPI2 One-sided Model. In *HPC Asia*, 2009.

[6] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global Arrays: A Nonuniform Memory Access Programming Model for High-Performance Computers. *Journal of Supercomputing*, 10(2):169–189, 1996.

[7] M. Valiev, E. Bylaska, N. Govind, K. Kowalski, T. Straatsma, H. V. Dam, D. Wang, J. Nieplocha, E. Apra, T. Windus, and W. de Jong. NWChem: A Comprehensive and Scalable Open-Source Solution for Large Scale Molecular Simulations. *Computer Physics Communications*, 181(9):1477 – 1489, 2010.

[8] A. Vishnu, J. Daily, and B. Palmer. Scalable PGAS Communication Subsystem on Cray Gemini Interconnect. In *HiPC*, Pune, India, 2012.

[9] A. Vishnu, H. V. Dam, W. D. Jong, P. Balaji, and S. Song. Fault Tolerant Communication Runtime Support for Data Centric Programming Models. In *HiPC*, 2010.

[10] A. Vishnu, D. J. Kerbyson, K. Barker, and H. J. J. V. Dam. Designing Scalable PGAS Communication Subsystems on Blue Gene/Q. Boston, 2013. 3rd Workshop on Communication Architecture for Scalable Systems.