Towards Portable and Adaptable Asynchronous Communication for One-Sided Applications

Min Si,[*] Jeff Hammond,[†] Masamichi Takagi,[‡] Yutaka Ishikawa[‡]

[*]*Argonne National Laboratory, USA* msi@anl.gov
[†]*Intel, USA* jeff_hammond@acm.org
[‡]*RIKEN AICS, Japan* {masamichi.takagi, yutaka.ishikawa}@riken.jp

***Abstract*—Irregular data communication makes applications hard to optimize because of unpredictable run-time behavior and the high risk of imbalanced computational loads. The MPI one-sided communication model is a portable and efficient approach for building irregular applications on distributed-memory systems, because the semantics of one-sided operations provide a more natural mapping to irregular algorithms as well as one-sided networks. The asynchronous completion of operations is important for applications to hide the overhead of intensive data movements within user computation. However, most implementations of the MPI one-sided communication model are not truly asynchronous in practice. That is, the remote process still relies on explicit MPI calls to complete some incoming operations that cannot be offloaded to network hardware. Therefore, support of software asynchronous progress is essential for high-performance irregular communication.**

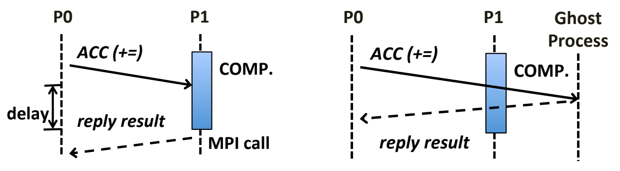
Casper is a portable and efficient process-based asynchronous progress model for MPI one-sided communication on multicore and many-core architectures. It provides flexible core deployment and dynamic adaptive strategies to optimize various situations of communication in user applications. In this paper, we briefly introduce the portable and adaptable design of Casper and demonstrate its efficient utilization by studying a real chemistry application suite.

I. Introduction

Irregular computation commonly exists in many scientific applications, especially in quantum chemistry and bioinformatics. The irregular and unpredictable runtime behavior makes this type of application often difficult to optimize. For instance, application developers are increasingly looking at ways to minimize the performance overhead coming from the irregularity in data communication across processes and the resulting imbalance in the computational load. In this work, we focus on the critical communication issues in irregular applications based on MPI one-sided communication.

MPI is the de facto standard communication interface for modern distributed-memory systems. The one-sided communication model, known as MPI RMA, is introduced in the MPI-2 and MPI-3 standards. The RMA semantics allow one process to access remote data located on the other process by using asynchronous one-sided operations such as *put* and *get*, without requiring the other process to be explicitly involved in the communication. Therefore, this model provides a more natural mapping to irregular algorithms as well as one-sided RDMA networks. In practice, however, most MPI implementations still heavily rely on the remote process to make software progress in MPI stack (i.e., by making MPI calls) and complete some incoming communication that cannot be offloaded to network hardware. Consequently, a process can be idle for a long time while it waits for the completion of communication if the remote process is busy computing outside the MPI stack. Therefore, additional asynchronous progress needs to be implemented in software. Unfortunately, traditional asynchronous progress approaches cannot help many applications because of restrictions from multithreaded MPI communication [1] or expensive overhead of frequent system interrupts [2], [3].

Casper, as an alternative approach to traditional asynchronous progress approaches, provides an efficient and practical solution for MPI one-sided applications [4]. Casper enables users to specify a small number of cores as background "ghost processes" on modern multicore or many-core systems. Casper then transparently intercepts all one-sided operations to the application processes through MPI's PMPI profiling interface and redirects them to the ghost processes, thus enabling asynchronous completion of one-sided operations without visible overhead. Figure 1 demonstrates the operation redirection design in Casper for an MPI *accumulate* operation. Moreover, Casper is carefully designed as a PMPI-based external library of MPI; thus it becomes a portable solution for applications running with different MPI implementations and platforms.



(a) Original Accumulate (b) Accumulate with Casper

Figure 1. Using Casper's asynchronous progress in MPI RMA.

Casper not only supports various underlying MPI platforms but also provides flexible strategies for diverse communication situations in user applications. According to the characteristics of communication, we categorize the asynchronous progress demands of applications in the following three types.

Sparse communication, where every process performs expensive computation and rarely communicates (or transfers only small amounts of data) with others. On the one hand, such a small amount of communication can be significantly delayed if the target process is busy in computation and might not be able to handle incoming data immediately. Therefore, support of asynchronous progress is the essential key for performance. On the other hand, we want to minimize the number of cores dedicated to asynchronous communication in order to ensure that sufficient computing power still remains for intensive user computation.

Dense communication, where processes frequently communicate with each other and perform only lightweight computation. Although asynchronous progress may still be beneficial, it has to occupy more processor cores in order to avoid overaggregating the large amount of communication handling work on a limited number of cores.

Multiphase communication, where processes coordinate through the execution of multiple internal phases of an application. This situation commonly exists in complex and large-scale computational problems implemented with the integration of multiple solvers and algorithm modules. Because each phase may follow a different communication pattern, we have to consider the needs of asynchronous progress in a more fine-grained way.

Casper provides several flexible strategies to optimize all these situations. For single-phase applications with either sparse communication or dense communication, the user can simply specify the appropriate core deployment of Casper according to the workloads of communication. For complex multiphase applications, where a fixed core deployment may not satisfy all internal phases, the user can utilize the dynamic adaptive mechanism of Casper to adjust the configuration of asynchronous progress for each internal phase at runtime.

We design two approaches for the dynamic adaptation mechanism. One is a *user-guided* approach, which allows the user to empirically enable or disable the redirection of communication for each particular internal phase during the application execution by passing MPI info hints at the beginning of that phase. This approach does not involve any overhead for adaptation, but it does require the user to understand the performance characteristics and code construction of the application. The other approach does not require any change in application code; instead, it utilizes the idea of *self-profiling and prediction*. In this approach, we insert profiling code in every MPI call through PMPI in Casper to dynamically track the change of communication performance during execution. Based on the profiling data, we can periodically predict the efficiency of asynchronous progress redirection for the next execution period and then reconfigure automatically.

In the next section, we showcase the efficient utilization of Casper by studying a real chemistry application.

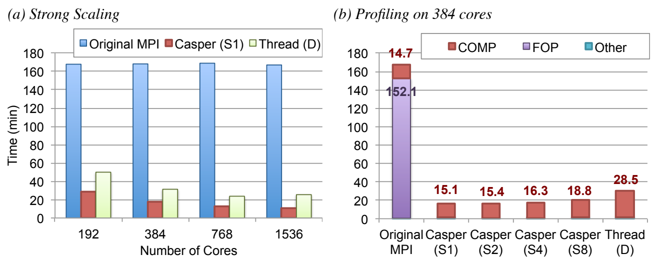


Figure 2. Single-phase DFT task for C240 with asynchronous progress.

II. Case Study: Chemistry Application

NWChem [5] is one of the most widely used computational chemistry application suites offering many simulation capabilities. NWChem is developed on top of the Global Arrays [6] toolkit, which provides an abstraction of global shared arrays over distributed-memory nodes. ARMCI-MPI is its portable communication port implemented over MPI RMA [7]. Most internal phases of NWChem utilize the *get-compute-update* mode, which every process essentially performs by varying the size of matrix-matrix multiplication for multidimensional tensor contraction by coordinating with others through RMA *get*, *put*, and *accumulate* operations. The generic task-scheduling component *nxtask* assigns the "owner" for subdomain computing tasks. It is implemented as a single *fetch_and_op* operation (denoted by FOP).

We study two widely used modules of NWChem, the single-phase density functional theory (DFT) and the multiphase CCSD(T), to evaluate the impact of different asynchronous progress strategies in Casper. Specifically, we compare Casper's performance with that of the original MPI without asynchronous progress support and with the conventional thread-based approach that always statically dedicates 50% of cores to asynchronous communication (denoted by Thread(D)). We execute all the experiments on the NERSC Edison Cray XC30 supercomputer [1] and compile with Intel compiler Composer XE 2015.1.133 and Cray MPI 7.2.1. We use NWChem version 6.3 with MKL 11.2.1 as the external math library.

A. Single-Phase DFT

Density functional theory provides a good mix of efficiency and accuracy for investigating the structural and electronic properties of atoms and molecules. It contains only a single internal phase in the implementation, which follows the *get-compute-update* mode and utilizes *nxtask* task scheduling. We measure the DFT calculation for Carbon 240 (denoted by C240) with the 6-31G* basis set.

Figure 2(a) shows the strong scaling over a varying number of system cores. We statically specify one ghost process on each node in Casper (denoted by Casper(S1)). The original MPI does not scale at all because of the significant delay in enormous blocking FOP operations in

[1]http://www.nersc.gov/users/computational-systems/edison/

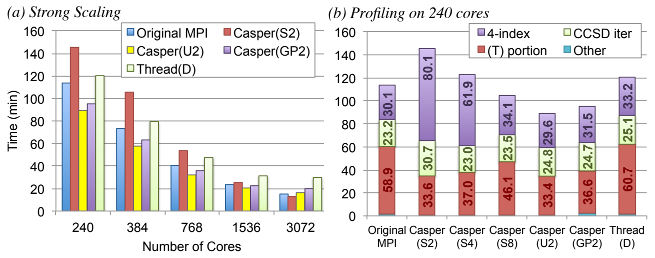


Figure 3. Multiphase CCSD(T) task for Tet with asynchronous progress.

nxtask. As shown in Figure 2(b), both Casper approaches and Thread(D) can eliminate such overhead; however, dedicating more ghost processes in Casper degrades the computational performance because of reduced computing powers. Similarly, the thread-based approach dedicates 50% of the cores (12 cores on Edison) to background threads, resulting in twice as much computational overhead.

B. Multiphase CCSD(T)

We next study the "gold standard" *coupled cluster with singles and doubles and perturbative triples* method, known as CCSD(T). This is one of the most accurate coupled cluster methods applicable to large molecules. It comprises four internal phases: self-consistent field (SCF), four-index transformation (4-index), CCSD iteration, and non-iterative (T) portion [8]. The (T) portion is an extremely computation-intensive phase, and the others follow a dense-communication pattern.

Although Casper with statically specified core deployment is sufficient for single-phase DFT, it does not satisfy multiphase CCSD(T). Figure 3(a) measures the strong scale of CCSD(T) calculation for tetracene (denoted by Tet) with the aug-cc-pVDZ basis set. We compare both the static approach of Casper (Casper(S2)), and the dynamic adaptive strategies including the user-guided approach and the self-profiling and prediction based approach, denoted by Casper(U2) and Casper(GP2), respectively. We specify two ghost processes in all Casper approaches. The static approach does not help performance because it overaggregates intensive RMA operations to only two ghost processes, thus degrading the performance of the 4-index and CCSD iteration phases, as profiled in Figure 3(b). Such degradation can be gradually released by utilizing more ghost processes; however, doing so also reduces the performance of (T) because fewer resources are used in heavy computation. The dynamic adaptation with user guidance delivers optimal performance for all phases, while the fully automatic self-profiling approach provides a relatively optimized performance for each phase without any change by the user.

REFERENCES

[1] W. Gropp and R. Thakur, "Thread-Safety in an MPI Implementation: Requirements and Analysis," *Parallel Comput.*, vol. 33, no. 9, pp. 595–604, 2007.

[2] Cray Inc., "Cray Message Passing Toolkit," http://docs.cray.com/books/S-3689-24, Cray Inc., Tech. Rep., 2004.

[3] M. Gilge, *IBM System Blue Gene Solution: Blue Gene/P Application Development*. IBM, Jun. 2013.

[4] M. Si, A. J. Peña, J. Hammond, P. Balaji, M. Takagi, and Y. Ishikawa, "Casper: An Asynchronous Progress Model for MPI RMA on Many-Core Architectures," in *Parallel and Distributed Processing, 2015. IPDPS 2015.*

[5] E. J. Bylaska et al., "NWChem, A Computational Chemistry Package for Parallel Computers, Version 6.3," 2013.

[6] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global Arrays: A Portable "Shared-Memory" Programming Model for Distributed Memory Computers," in *ACM/IEEE conference on Supercomputing*, 1994.

[7] J. S. Dinan, P. Balaji, J. R. Hammond, S. Krishnamoorthy, and V. Tipparaju, "Supporting the Global Arrays PGAS Model Using MPI One-Sided Communication," in *IPDPS*, May 2012.

[8] J. R. Hammond, S. Krishnamoorthy, S. Shende, N. A. Romero, and A. D. Malony, "Performance Characterization of Global Address Space Applications: A Case Study with NWChem," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 2, pp. 135–154, 2012.