Parallel Computing 39 (2013) 132–145

Contents lists available at SciVerse ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

A scalable infrastructure for the performance analysis of passive target synchronization

Marc-André Hermanns [a,b,*], Sriram Krishnamoorthy [d], Felix Wolf [a,b,c]

[a] German Research School for Simulation Sciences, 52062 Aachen, Germany
[b] Dept. of Computer Science, RWTH Aachen University, 52056 Aachen, Germany
[c] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
[d] Computer Science and Mathematics Division, Pacific Northwest National Laboratory, Richland, WA, USA

ARTICLE INFO

Article history:
Available online 8 October 2012

Keywords:
Performance analysis
Event tracing
One-sided communication
Remote memory access

ABSTRACT

Partitioned global address space (PGAS) languages combine the convenient abstraction of shared memory with the notion of affinity, extending multi-threaded programming to large-scale systems with physically distributed memory. However, in spite of their obvious advantages, PGAS languages still lack appropriate tool support for performance analysis, one of the reasons why their adoption is still in its infancy. Some of the performance problems for which tool support is needed occur at the level of the underlying one-sided communication substrate, such as the Aggregate Remote Memory Copy Interface (ARMCI). One such example is the waiting time in situations where asynchronous data transfers cannot be completed without software intervention at the target side. This is not uncommon on systems with reduced operating-system kernels such as IBM Blue Gene/P where the use of progress threads would double the number of cores necessary to run an application. In this paper, we present an extension of the Scalasca trace-analysis infrastructure aimed at the identification and quantification of progress-related waiting times at larger scales. We demonstrate its utility and scalability using a benchmark running with up to 32,768 processes.

1. Introduction

The evolution of high-performance computing (HPC) systems in the last decade has led to an exponential increase in parallelism. Computing systems in the top ten of the world's 500 fastest supercomputers today feature an average of more than 180,000 cores [1]. In 2011, the largest system in terms of the number of cores (RIKEN's K Computer) offers a total of 548,352 cores on 68,544 distributed-memory nodes. At larger scales, even small waiting times can propagate and accumulate throughout the application and significantly hinder acceptable application performance [2]. Performance-analysis tools for HPC platforms are designed to aid application and library developers, as well as compiler writers, in the often overwhelming task of investigating and understanding the application's behavior at such a large scale. However, they are often focused only on the predominant programming paradigm—message passing using the Message Passing Interface (MPI) [3].

With the advent of partitioned global address space (PGAS) languages, purely one-sided communication libraries gained more momentum, as these are employed in the communication runtime of those languages. In one-sided communication, all communication parameters, such as source and destination memory locations, are provided by one of the communication

* Corresponding author at: German Research School for Simulation Sciences, 52062 Aachen, Germany. Tel.: +49 241 80 99753; fax: +49 241 80 6 99753.
E-mail addresses: m.a.hermanns@grs-sim.de (M.-A. Hermanns), sriram@pnnl.gov (S. Krishnamoorthy), f.wolf@grs-sim.de (F. Wolf).

0167-8191/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.parco.2012.09.002

partners only—the origin. The second communication partner—the target—does not explicitly call a communication function to match the origin's communication call. Seen from the programmer's view, one-sided data transfers are completed without active participation of the target. One of these one-sided communication libraries is the Aggregate Remote Memory Copy Interface (ARMCI) [4], used as the communication back-end of Global Arrays [5], a PGAS-style library. The efficiency of the communication relies greatly on whether the data exchange can be completed without the active participation of the other process. This is often provided through the communication hardware's remote direct memory access (RDMA) support. When this support is unavailable either for the entire platform or only for a specific type of communication construct, a software component provides this progress. While this component can sometimes be executed by a helper thread, large-scale architectures with reduced kernels such as IBM's Blue Gene/P require an extra core to run it, effectively doubling the required hardware. Interrupt-driven progress, an alternative to a dedicated thread, on the other hand, introduces the cost of an interrupt for every communication call and may pollute the cache. Without a separately scheduled progress engine, however, progress can only occur when the application calls the communication library directly. Yet, one of the inherent characteristics of PGAS applications is that individual processes do not necessarily communicate at the same time. Significant waiting times can therefore occur at the origin of a one-sided operation, while it is waiting for progress at the target side. In addition, inter-process dependencies may induce further waiting times on remote processes via propagation, even if the original waiting times are small [2]. The impact of the lack of remote communication progress on application performance has not been studied before, although this knowledge is crucial to assess the costs of alternatives such as extra threads or interrupts.

To assist in performance tuning at a larger scale, performance-analysis tools must be scalable as well. Event tracing is a widely used method for performance analysis of parallel applications, and it has been successfully applied by several performance-analysis tools [6–10] available on typical HPC platforms. We have shown in previous work that trace-based performance analysis can be successfully employed at a large scale [11]. The main advantage of event tracing is the richness of the inter-process information that can be captured, allowing the analysis of extremely complex inter-process relationships.

Waiting time induced by insufficient message progress on the remote side is an example of such an inter-process relationship, where event data from multiple processes have to be taken into account. The waiting time on the remote process can only be quantified by knowing the start and end time of the communication call on the origin, as well as of the progress function on the target.

The number of performance analysis tools supporting one-sided communication libraries is currently rather small. The Parallel Performance Wizard (PPW) [10] supports the analysis of general one-sided communication constructs. It relies on the GASP interface [12], a callback interface specifically designed for the analysis of PGAS applications and one-sided communication. Although now supported by several Unified Parallel C (UPC) compilers, it is unfortunately not yet widely supported by current one-sided communication libraries. To the best of our knowledge, only GASNet [13] and Quadrics SHMEM [14] support this measurement interface so far.

The asynchronous parallel-programming framework Charm++ [15] supports the investigation of one-sided communication through its proprietary performance tool Projections. MPI Peruse [16] allows implementation-internal events related to MPI one-sided implementations to be captured, and could be used to measure the necessary internal information. However, it is limited to MPI and to the best of our knowledge is only supported by OpenMPI [17]. The Cray Pat and Apprentice performance tools [18] support measurement of Cray SHMEM [19] using a mixture of instrumentation and sampling. The TAU performance toolkit [9] has recently been extended to support measurement and analysis of Global Arrays and ARMCI calls [20], however, it records only time profiles and the communication matrix. In their study [21], Balaji and colleagues show that system-specific waiting times can be an important factor when analyzing application performance. They investigated overheads in the MPI implementation on Blue Gene/P due to computations done by the implementation itself, focusing on another architecture characteristic of these systems—the comparatively low clock rate of the compute elements.

In our earlier work in the context of the Scalasca performance analysis tool [22], we showed how large-scale parallel trace analysis can be facilitated using parallel message replay. So far, supported communication constructs include MPI point-to-point, collective, and one-sided operations with active target synchronization. The latter can be easily accomplished [23] because the active target synchronization following the one-sided exchange, which involves both parties, provides a welcome opportunity to exchange relevant information during the replay.

However, ARMCI one-sided communication provides only passive target synchronization, which does not actively involve the target process. During the replay, the origin process, where the progress-related waiting time occurs, would not know the location of relevant information on the target processes, and the target process would not know how to locate this information on behalf of the origin process. This missing opportunity for data exchange poses serious challenges for Scalasca's trace-based performance-analysis approach. In this work, we present two advanced techniques for data exchange during the replay of one-sided communication that overcome the absence of triggering events on the target side. We describe how we use these techniques to detect and quantify the waiting times caused by untimely remote progress in one-sided communication. We demonstrate this functionality using three different applications based on either Global Arrays or ARMCI directly across multiple scales on up to 32,768 processes.

The remainder of this paper is organized as follows. Section 2 gives an overview of the Aggregate Remote Memory Copy Interface (ARMCI), the one-sided library which is the subject of our investigation. We present the event model that we use to model ARMCI communication in Section 3. Based on this model, we define the *Wait for Progress* inefficiency pattern in Section 4. Section 5 gives a short introduction to Scalasca's message-replay-driven analysis and presents our extension to