J. Parallel Distrib. Comput. 72 (2012) 1481–1492

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

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

HiCOO: Hierarchical cooperation for scalable communication in Global Address Space programming models on Cray XT systems

Weikuan Yu [a,*], Xinyu Que [a], Vinod Tipparaju [b], Jeffrey S. Vetter [b]

[a] Department of Computer Science, Auburn University, AL 36849, USA
[b] Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

ARTICLE INFO

Article history:
Received 18 July 2011
Received in revised form
3 January 2012
Accepted 30 January 2012
Available online 6 February 2012

Keywords:
GAS
ARMCI
Multicore
Multinode
Virtual Topology
Contention

ABSTRACT

Global Address Space (GAS) programming models enable a convenient, shared-memory style addressing model. Typically this is realized through one-sided operations that can enable asynchronous communication and data movement. With the size of petascale systems reaching 10,000s of nodes and 100,000s of cores, the underlying runtime systems face critical challenges in (1) scalably managing resources (such as memory for communication buffers), and (2) gracefully handling unpredictable communication patterns and any associated contention. For any solution that addresses these resource scalability challenges, equally important is the need to maintain the performance of GAS programming models. In this paper, we describe a Hierarchical COOperation (HiCOO) architecture for scalable communication in GAS programming models. HiCOO formulates a cooperative communication architecture: with inter-node cooperation amongst multiple nodes (a.k.a multinode) and hierarchical cooperation among multinodes that are arranged in various virtual topologies. We have implemented HiCOO for a popular GAS runtime library, Aggregate Remote Memory Copy Interface (ARMCI). By extensively evaluating different virtual topologies in HiCOO in terms of their impact to memory scalability, network contention, and application performance, we identify MFCG as the most suitable virtual topology. The resulting HiCOO architecture is able to realize scalable resource management and achieve resilience to network contention, while at the same time maintaining or enhancing the performance of scientific applications. In one case, it reduces the total execution time of an NWChem application by 52%.

1. Introduction

Several supercomputing sites have deployed systems with extreme amounts of computational power [30]. For example, the Jaguar Cray XT5 system in the US, the Tianhe-1A system in China, and the *K* Computer in Japan can perform to the order of 10^{15} floating point operations per second (petaflop). While supercomputing systems grow to unprecedented number of processors (with LLNL Sequoia [21] system and NCSA Blue Waters [22] system in the near future), scientific applications continue to face many challenges such as programming productivity, application scalability, and efficiency. Global Address Space (GAS) or Partitioned Global Address Space (PGAS) models are emerging as scalable alternatives because they have the ability to alleviate programming burden by supporting data access to both local and remote memory through a simple shared-memory addressing model.

PGAS languages such as Unified Parallel C (UPC) [31], Co-Array Fortran (CAF) [10], and X10 [26], and GAS libraries such as Global Arrays (GA) Toolkit [14] are becoming increasingly popular. These languages and libraries use the services of an underlying communication library (which we refer to as the GAS runtime) for serving their communication needs. They normally convert data transfers through compilation techniques into one-sided communication messages on distributed memory architectures. They have a translation layer that translates memory access to various one-sided messages, with which programmers no longer have to orchestrate complicated message passing schemes among many pairs of parallel processes.

ARMCI (Aggregated Remote Memory Copy Interface) [23] is a popular runtime that has been used to implement both PGAS languages (such as Co-Array Fortran) and GAS libraries (such as GA). While some MPI applications have reached a sustained petaflop performance and beyond, NWChem [17] computation chemistry code is a GAS-based application and is one of the three applications to have crossed the petaflop barrier in terms of sustained performance [2] on Jaguar. This was made possible by the porting of Global Arrays toolkit, and more specifically, its GAS runtime, ARMCI [29].

* Corresponding author.
E-mail addresses: wkyu@auburn.edu, weikuan.yu@gmail.com (W. Yu), xque@auburn.edu (X. Que), tipparajuv@gmail.com (V. Tipparaju), vetter@ornl.gov (J.S. Vetter).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.01.022

Unfortunately, running a GAS model and its underlying GAS runtime in the context of a real scientific application at a scale similar to Jaguar (200,000 + cores) has brought forth a few staggering challenges. These challenges are a result of the characteristics and asynchronous one-sided features of the GAS runtime. The first is that of resource management, incurred by unpredictable communication patterns and communication resources (such as buffers) that need to be allocated to support it. The second challenge is that of network contention—allowing any process to access the address space of any other process and supporting load balancing at the same time create an environment that is prone to contention.

In this paper, we describe a Hierarchical COOperation (HiCOO) architecture for scalable GAS programming models. HiCOO formulates a cooperative communication architecture with inter-node cooperation amongst multiple nodes (a.k.a multinode) and hierarchical cooperation among multinodes that are arranged in various virtual topologies. We have implemented HiCOO for a popular GAS runtime library, Aggregate Remote Memory Copy Interface (ARMCI). It leverages the existing multicore cooperation in ARMCI and extends with *multinode cooperation* and *hierarchical cooperation*.

In multinode cooperation, compute nodes form a multinode group and work together to handle one-sided communication requests. Their cooperation is realized through (1) request forwarding in which one node can intercept a request and forward it to the target node, and (2) request aggregation in which one node can aggregate many requests to the same target node. With multinode cooperation, HiCOO no longer has to create one set of communication buffers on every node for all possible pairs of peer processes. Instead, it divides the requirement of communication buffers amongst themselves in a cooperative manner. When a request reaches one node in a multinode group, it is forwarded to the target node, and handled accordingly. Through request aggregation, multinode cooperation also exploits the presence of multiple requests to the same target node. It consolidates them together to reduce network contention, thereby alleviating the pressure to the underlying physical network.

With hierarchical cooperation arranged in various virtual topologies, HiCOO attenuates contention and efficiently manages communication resources in ARMCI (and any GAS/PGAS runtime in general) at petascale and beyond. HiCOO represents the allocation of communication resources as directed graphs. While the original model can be depicted as a fully connected graph (FCG), two new scalable virtual topologies, Meshed FCGs (MFCG) and Cubic FCGs (CFCG), have been exploited for scalable resource management and contention attenuation in HiCOO. We have systematically examined the communication characteristics of MFCG, CFCG, and Hypercube. We have successfully implemented these virtual topologies in ARMCI on Jaguar, and conducted experiments to evaluate these topologies using microbenchmarks and real large-scale applications. We then choose MFCG as the default topology for HiCOO.

While addressing the challenges of resource scalability and network contention, equally important is the need to maintain the performance of GAS programming models. Our experimental results on a large-scale Cray XT5 system indicate that HiCOO is able to greatly increase memory scalability by reducing communication buffers required on each node. In addition, it improves the resiliency of GAS runtime system to network contention. Furthermore, HiCOO is able to maintain or improve the performance of scientific applications. In one case, it reduces the total execution time of an NWChem application by 52%.

The rest of the paper is organized as follows. Section 2 discusses background and motivation. Section 3 describes the architecture of HiCOO and its two key components: multinode cooperation and virtual topology. Experimental results are provided in Sections 4 and 5, followed by related work in Section 6. We conclude the paper in Section 7.

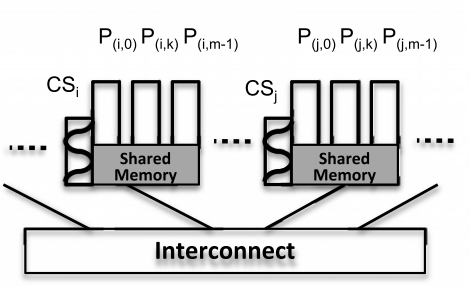

Fig. 1. ARMCI process management.

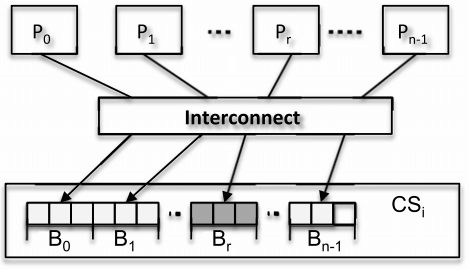

Fig. 2. ARMCI server's request buffer management.

2. Background and motivation

2.1. An overview of ARMCI

ARMCI has recently been enabled for Cray XT5 using the native portals communication library [29]. ARMCI guarantees that its one-sided operations are fully unilateral, i.e., may complete regardless of the actions taken by the remote processes. In particular, polling the application by remote processes (implicitly when making a library call, or explicitly by calling provided polling interface) is not required for communication progress. This is realized by introducing a communication helper thread (a.k.a communication server) at each compute node. This communication helper thread is created by the lowest ranked process (*master*) on a node. An area of shared memory is allocated for these processes. The communication server (CS) handles remote one-sided requests on behalf of all local processes, and exchanges data with them through the shared memory. Similar to what described earlier, the communication server pre-allocates buffers and related data structures for remote requests, in order to support direct one-sided communication for all operations (particularly for lock, unlock, accumulate, and noncontiguous data transfer operations) and allow one process to asynchronously initiate an operation without the involvement of the targeted process.

2.2. ARMCI process management for one-sided communication

Fig. 1 shows the process management of ARMCI. On two arbitrary nodes, i and j, each has a set of parallel processes. All processes have a global rank. Processes on node i are also denoted as $P_{(i,k)}, \forall k \in [0, m-1]$. An area of shared memory is allocated for these m processes. The lowest ranked process $P_{(i,0)}$ creates a separate thread as a communication server CS_i. The communication server CS_i communicates with all intra-node processes through the shared memory and handles all incoming inter-node one-sided communication requests on behalf of them.

Every communication server has to pre-allocate request buffers for all remote peer processes. Fig. 2 shows the request buffer management of CS_i. Each processes is denoted based on its global rank $P_r, \forall r \in [0, n-1]$. A set of request buffers are allocated for each remote process, e.g. B_r for P_r.