Runtime Techniques to Enable a Highly-Scalable Global Address Space Model for Petascale Computing

Vinod Tipparaju · Edoardo Apra · Weikuan Yu · Xinyu Que · Jeffrey S. Vetter

Received: 1 December 2010 / Accepted: 2 August 2012 © Springer Science+Business Media, LLC 2012

Abstract Over the past decade, the trajectory to the petascale has been built on increased complexity and scale of the underlying parallel architectures. Meanwhile, software developers have struggled to provide tools that maintain the productivity of computational science teams using these new systems. In this regard, Global Address Space (GAS) programming models provide a straightforward and easy to use addressing model, which can lead to improved productivity. However, the scalability of GAS depends directly on the design and implementation of the runtime system on the target petascale distributed-memory architecture. In this paper, we describe the design, implementation, and optimization of the Aggregate Remote Memory Copy Interface (ARMCI) runtime library on the Cray XT5 2.3 PetaFLOPs computer at Oak Ridge National Laboratory. We optimized our implementation with the flow intimation
technique that we have introduced in this paper. Our optimized ARMCI implementation improves scalability of both the Global Arrays programming model and a real-world chemistry application—NWChem—from small jobs up through 180,000 cores.

**Keywords**  Global address space · Global arrays · ARMCI · XT5 · NWChem · Flow control · PGAS · GAS · GA

1 Introduction

Systems with unprecedented computational power are continuously pushing the frontier of high performance computing (HPC) [35]. Several sites have deployed systems that can perform $10^{15}$ floating point operations per second (petaflop): Cray XT5 (a.k.a. Jaguar) at the Oak Ridge National Laboratory (ORNL), IBM Cell-based system at Los Alamos National Laboratory (LANL), BlueGene/P at Forschungszentrum Juelich (FZJ). These facilities are used to solve important computational science problems in areas such as climate modeling, life sciences, and energy production. Yet many challenges in scientific productivity and application efficiency continue to plague these systems as they grow to unprecedented number of processes.

In this regard, Global Address Space (GAS) programming models - both the Partitioned and Asynchronous Partitioned Global Address Space (PGAS)—are being considered as an alternative model for programming these complex machines to improve productivity and application efficiency.

Briefly, a GAS model provides an abstraction that allows threads to access the remote memory of other nodes as if they were accessing local node memory using hardware shared memory. By virtue of the abstraction they provide, PGAS languages like Unified Parallel C (UPC) [36], Co-Array Fortran (CAF) [9], and GAS libraries such as Global Arrays (GA) Toolkit [13] have the unique ability to expose features, such as low-overhead communication or GAS support in the underlying hardware. Systems that lack one or more of these features typically result in poor performance.

Conceptually, GAS models do not differentiate between local and remote accesses. By contrast, PGAS is a category of GAS models that requires applications to explicitly distinguish between local and remote memory accesses, while providing simple mechanisms for reading, writing, and synchronizing remote memory. One benefit of this explicit separation is that the user is forced to consider and optimize the performance of remote memory access while leaving the optimization of local memory accesses to the compiler.

Recently, a slightly different category of PGAS model, termed Asynchronous PGAS model, has emerged to add additional capabilities such as remote method invocations. IBM’s X10 language [31] and Asynchronous Remote Methods (ARM) [32] in UPC have pioneered this new model.

All the above mentioned GAS languages and libraries use the services of an underlying communication library (which we refer to as the GAS Runtime) for serving their communication needs. GAS languages normally use this runtime as a compilation target to do the data transfers on distributed memory architectures. They have a translation layer that translates a GAS access to a corresponding data transfer on the