Project Progress Report

On No-Cost-Extension Period for Reservoir Labs.

DynAX: Innovations in Programming Models, Compilers and Runtime Systems for Dynamic Adaptive Event Driven Execution Models

Award Number: DESC0008716
Dates of Performance: 9/1/2015 to 10/31/2015
Report Date: 10/31/2015

Principal Investigator
Guang Gao
ET International Inc
100 White Clay Center Dr, Newark DE 19711

CoPIs:
Benoit Meister, Reservoir Labs, Inc
David Padua, University of Illinois Urbana Champaign
John Feo, Pacific Northwest National Laboratories

Introduction

This report outlines the work performed by the Reservoir Labs team during the two-month no-cost extension period granted to them.

Summary

During this period, we have improved the cluster target for R-Stream in two ways. First, we addressed portability limitations in the cluster runtime layer. Second, we modified the R-Stream mapper and backend to expose tile-specific communications, which removes any overhead associated with data tiling from the runtime. These items are presented in the next sections.

Portability of the R-Stream cluster backend

Our initial implementation of the R-Stream cluster backend for dense arrays was based on PNNL's Global Arrays (GAs), which offers an abstraction for a (possibly tiled) distributed array through DMA calls.

While GAs were a perfect match for dense arrays, we realized that GAs were not a good layer to implement a block sparse array abstraction. This is because they maintain a rigid mapping between the address of a data element in a global array and the rank to which the element belongs. In the context of a block sparse array, in which blocks are created and destroyed on demand as a function of the element values, the use of GAs would have entailed crippling collective synchronization in order to enable concurrent data block creation and destruction. Implementing each data block as a distinct array was also not viable, since with GAs, array creation is a global operation, requiring a barrier.

Hence we designed a runtime layer in which each tile is statically "owned" by a particular rank[1]. Tile owners are responsible for reading and writing to the tiles (wherever the data tiles are actually distributed). This removed the need for global synchronizations during the computation, but required the combination of remote procedure call (RPC) capabilities in conjunction with asynchronous communications.

To be precise, in order to increase asynchrony of our communications, we wanted to be able to spawn codelets (asynchronous tasks) on different nodes remotely. The SWARM network layer was able to perform this. Additionally, it relies on the Open Fabrics (OFED) libraries, an industry standard for high-performance low-level communications. Along with this, we needed something to perform the asynchronous transfers of data, optimized for RDMA.
ARMCI, also developed at PNNL, seemed like a good candidate for communications. It is a lower-level communication layer used by GAs and implementations of higher-level distributed execution models (including co-array Fortran, shmem). It has decent portability, supports RDMA and has several built-in optimizations (such as small message coalescing). An extension of

[1] As described in detail in our Q12 Quarterly report.

ARMCI under development, ComEx, supports a greater set of hardware platforms, and in particular supports OFED. Unfortunately, according to the ARMCI and ComEx documentation, the RPC capability coming with it is still under development. ComEx appears as a good candidate for future versions.
Hence, we tried to combine ARMCI for communications with the SWARM network layer. Unfortunately, this required the cooperation of the MPI and SWARM distributed runner scripts, which we were unable to achieve. Note that this combination is different from the MPI+SWARM experiments presented in the Q12 report by ETI, in which SWARM's network API was only used by the “pure SWARM” implementation.

We also briefly considered implementing RPC using MPI, so a single distributed runner script could be used. Paradoxically, we found that MPI was not necessarily the best choice for portability. While it has the advantage that most if not all high-performance cluster has some version of MPI installed, each implementation supports a different set of features. For instance, some implementations are thread-safe, mono- or multi-threaded, etc. Hence, writing an optimized runtime layer to MPI either is a complex exercise or requires the use of the common denominator of available features across implementations (instances of this are the use of a special communication worker or thread[2][3], which sequentializes all the data transfers). As a result, we do not consider MPI optimal as a support for our block-sparse cluster backend.

We finally opted for GPI-2, a package distributed by the OFED consortium. GPI-2 implements an asynchronous, one-sided RDMA API, as well as an RPC mechanism based on RDMAs, and the rank and barrier concepts as in MPI. GPI-2 supports OFED, and can be used to program clusters of multi-core x86, Xeon Phi, and GPUs. An interesting special mode of GPI-2 also enables a one-to-one mapping between ranks and NUMA domains in clusters. Timeouts are also a built-in feature, which can be used to avoid blocking a thread or a worker on communication and balance computation vs. communication needs.
We also reimplemented the dense array part of our distributed runtime in the terms of the API used by the dense array runtime. This way, both dense and block sparse arrays can be used conjointly (R-Stream can produce such a program if directed so by the user).

Data tiling optimization

The GA backend pushes data tiling to the runtime. This is convenient, as transfers that span several tiles can be written as a single GA API call. However, this convenience comes with a cost, which we would like to avoid since RDMA calls are expected to be executed quite often. Or solution, was to add an optimization pass to the R-Stream compiler to form tile-specific DMA transfers. This way, we expose the tiled layout to the compiler.
The pass consists in tiling the DMA transfers along the dimensions of the distributed array, with the tile sizes and alignment matching the distributed array data tiles. Extra parameters to the DMA commands formed by R-Stream represent the array ID and tile coordinates, which are

[2] Open Community Runtime (OCR), https://01.org/projects/open-community-runtime

[3] Roshan Dathari, “Compiling for a Dataflow Runtime on Distributed-Memory Parallel Architectures.” Master's thesis, Indian Institute of Science, Bangalore, April 2014.

dynamically resolved to addresses in the case of a block sparse array, and statically resolved in the case of a dense array. Addresses of elements of the distributed array are relative to their tile's base address, leaving the minimal amount of address computations to the runtime.

Status

We have completed a basic implementation of the backend and runtime based on GPI-2. This development branch will be brought to production quality as part of its integration into the next R-Stream release, scheduled end of November. We plan to publish performance results (and share them with the PM) at that point, at which we will be able to produce enough stable results to make useful conclusions.