HARVEY MUDD COLLEGE

Computer Science Clinic

Statement of Work for
Sandia National Laboratories

Parallelizing Intrepid with Kokkos

October 5, 2014

Team Members
Brett Collins (Project Manager)
Alex Gruver
Ellen Hui
Tyler Marklyn

Advisor
Jeff Amelang

Liaisons
Carter Edwards
Robert Hoekstra

1 Problem Statement

Sandia National Laboratories currently has many libraries that do not utilize new multicore architectures. Using their Kokkos library, which was designed to make it possible and easy to port code to new architectures (specifically multicore architectures), we will make a tensor contractions library and, as time permits, other libraries thread scalable.

2 Background

Sandia National Laboratories is a federally funded research and development center owned and managed by the Sandia Corporation. The laboratory's primary focus is the maintenance, management, and development of the United States' nuclear arsenal. Sandia also performs research in the fields of supercomputing and scientific computing. The Trilinos Project, developed and maintained by Sandia, is a collection of open source libraries intended for use in large-scale scientific applications.

One of the packages in Trilinos is Kokkos, which is designed to aid portability and performance in software written for manycore architectures. The vast majority of scientific software produced and used by the national labs rely on message passing parallelism to leverage both inter-node and intra-node parallelism. Until the present, this approach of using MPI across a single node sacrifices some performance for the ease of a monolithic programming model, and the performance penalty has not been high enough to motivate the usage of threads instead of processes. However, as the exascale push hits the power wall, interest has been growing in the area of using higher-performing and less power-hungry co-processors on each node, still with message passing across nodes. Unfortunately, this means that codes will have to be re-written, as message-passing cannot be used to leverage the parallelism of coprocessors such as Graphical Processing Units (GPUs).

GPUs are characterized by high thread counts, decreased memory per thread, and relatively small instruction sets when compared to CPUs. These differences lead to many advantages when it comes to high performance computing. Since GPUs have a smaller instruction set, they can devote more of their transistors to arithmetic computation. This means that GPUs are capable of executing significantly more floating point operations per second (flops). Additionally, GPUs use less power than CPUs, which makes them appealing for supercomputers, where power consumption is a major concern.

Well implemented GPU code can yield significant speedup in certain processing-heavy applications. However, the ways in which an algorithm must be optimized to run on a GPU are highly dependent on hardware architecture. In addition, highly parallel code may suffer from race conditions between threads, requiring the use of mutexes which can significantly slow down performance.

Kokkos attempts to mitigate these issues by allowing programmers to write their code once, and compile for optimization on a variety of many-core architectures. This is possible because the library manages the allocation of memory for the programmer and optimizes storage for speed on the target architecture. Kokkos also allows users to write thread scalable software by providing an API for using fast, non-locking data structures.

However, the Kokkos package is still new and relatively untested. No large-scale projects have yet been written with it. Some small kernels have been rewritten using Kokkos, but there are still many kernels in Trilinos that would benefit from increased thread scalable parallelization.

3 Goal

The goal of this clinic is to rewrite several kernels from libraries within Trilinos to be efficient and thread-scalable on manycore architectures, using Kokkos. These redesigned and reimplemented kernels will be integrated into Trilinos' production code, as both a performance improvement and as a use case for Kokkos.

We plan on reengineering a tensor contraction library within Trilinos. This library is part of the Intrepid sub-package of Trilinos. The Intrepid package is a library of kernels designed for use by developers who want to reuse large parts of their existing code frameworks while gaining access to state of the art tools for compatible discretizations of partial differential equations. Once we have incorporating Kokkos into Intrepid and demonstrated measurable performance and usability improvements, we plan on incorporating Kokkos into the salient kernels of another library within Trilinos. The second library will deal with either graph analytics or solving algebraic multigrid systems.

4 Objectives

The primary goal of this clinic is to update Intrepid's tensor manipulation for thread scalability using Kokkos. After this, we will begin work on one of the secondary options listed above. This goal can be broken down into a couple different objectives.

- First, our team will familiarize ourselves with Kokkos' structure and usage by implementing several warm up algorithms with different memory access patterns. We also plan on doing performance comparisons with serial CPU implementations to learn the most effective mechanisms for parallel speedup.
- Our second major objective will be use what we have learned about massively parallel code to speed up Intrepid's tensor contraction kernel.
- The third and final objective of this project is to speed up another kernel from Intrepid. This kernel will be focused on one of the following.
  - graph analytics
  - algebraic multigrid solving

4.1 Optional Objectives

The optional objectives presented in this section are to be done if time permits. First, if the team has finished, or has made significant progress towards making Sandia's tensor manipulation library thread scalable, then the team will present their methods and results to a group of Sandia employees during the winter site visit. After we've finished work on the Intrepid kernels, there are a couple different objectives we can pursue. The first possible objective is to continue making more of Sandia's libraries thread-scalable. Another possibility is to code versions of the Intrepid libraries in CUDA, or another framework such as OpenCL, in order to compare their performance with the Kokkos' implementations. The performance tests can show the strengths and weaknesses of Kokkos and give information that is useful to the developers of Kokkos.

5 Tasks

Our first task for this semester was to acquire a desktop with suitable computational capabilities. We have ordered a machine with an NVIDIA K20 Tesla Card, Intel Phi 5110p Co-processor, and a 8-Core Ivy Bridge Processor. This computer should allow us to test NVIDIA's GPU computing platform as well as Intel's, and allow us to compare the performance of our code on GPUs to the Ivy Bridge's traditional 8-Core hardware.

After this, we will try to familiarize ourselves with massively multithreaded programming by working through a series of example problems in a variety of environments. So far, we have identified the following problems as likely candidates.

- scalar integration
- histogram calculations
- calculating polynomials for a long list of x values and coefficients.
- naive matrix multiplication
- smart matrix multiplication
- scalar product of two second order tensors
- contraction of fourth order tensor with second order tensor

The last of these problems should prepare us to work with the Intrepid library, since these two problems are indicative of the types of kernels we will be working with.

For all of these problems, we intend on creating the following solutions

- fast serial
- implementation using Intel's Threaded Building Blocks
- implementation using OpenMP
- implementation using Cuda
- implementation using Kokkos

After familiarizing ourselves with TBB, CUDA and OpenMP, we will have a background to understand the ideal way to write code using Kokkos. At this point, we will begin to write solutions to our practice problems using Kokkos. We can then compare the speedup gains on Kokkos compared to the other libraries, and also the ease of writing thread scalable code using Kokkos compared to the other libraries. This will be good practice for applying the same process to Intrepid kernels, to thoroughly evaluate performance improvement and ease of coding.

At this point, we will shift our focus to our first kernel. After we implement the practice problems mentioned above we should have a good idea of where to go with the tensor contraction implementation. Tensor contraction is very similar to smart matrix multiplication in particular. We will need to explore the main algorithms used for tensor contraction and determine if there are existing procedures for parallelizing the process. The original Kokkos source code, along with online materials should help us with this.

Once we are familiar with Kokkos, and we feel we have enough of a background with the technologies, we will begin working on improving the tensor contraction library. At this point in the project it is difficult to create an accurate timeline for this process, but we hope that we will be able to present our progress at the end of semester site visit at Sandia.

After the tensor contraction library we will continue work on one of the Intrepid kernels previously mentioned. We hope that we will be able to accomplish significant speedup in this secondary library after we finish work on tensor contraction.