Parallel Proton Synchrotron Tomography using CUDA vs. High Performance Fortran and OpenMP

James Boelen, Raymond Hung, Stanley Tsang
School of Information & Communications Technology, Seneca College

Problem Overview

Tomography is a topic with a wealth of algorithms for the reconstruction of both qualitative and quantitative images. One of the simplest algorithms has been modified to take into account the non-linearity of large-amplitude synchrotron motion. This permits the accurate reconstruction of longitudinal phase space density from one-dimensional bunch profile data. The method is a hybrid one which incorporates particle tracking, and considerable effort has been invested to optimize the computer code so that it may also be compiled to exploit parallel architectures efficiently.

TOMO is an tomography application for the European Organization for Nuclear Research's (CERN) Proton Synchrotron, which implements this hybrid algorithm method. Originally written with Fortran by CERN in a serial implementation, TOMO has since been modified to work to utilize High Performance Fortran (HPF) for parallelism, and then also subsequently modified with the OpenMP platform to support multi-platform shared-memory parallel execution.

Recently, a mutual interest was expressed by the developers of TOMO at CERN and students at Seneca College taking a Parallel Programming Course to write a CUDA-enabled version of TOMO to compare the performance of a CUDA implementation versus the HPF and OpenMP implementations (with four environment threads).

Test Environment

CPU	Intel Core i5 2500k, 3.30 Ghz, Quad-Core
Motherboard	Gigabyte GA-P35-DS3L
Memory	8 GB DDR3 RAM
GPU	GTX 670 2GB GDDR5, Compute Capability 3.0
Operating System	Ubuntu 12.04 LTS 32-bit
Graphics Driver Version	310.19
CUDA Toolkit Version	5.0

Methodology & Analysis

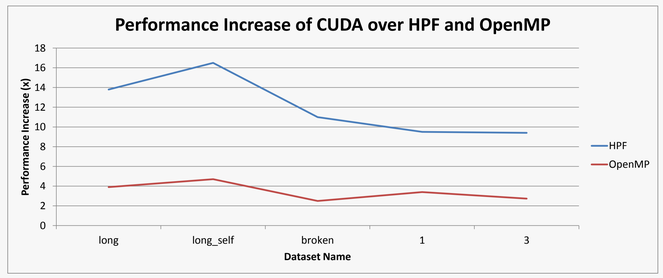
Initial profiling of the HPF implementation of TOMO using gprof revealed a significant bottleneck in one key Fortran subroutine called **longtrack,** with approximately 80% of program execution time consumed. As such, we decided to focus our efforts on writing a port of the original Fortran longtrack code to a C version called **gputrack**, and then writing a CUDA implementation of gputrack. The remainder of TOMO was left in the original HPF version, which calls the gputrack function instead of the original longtrack. We also upgraded a similar subroutine longtrack_self that showed the same bottleneck.

Further analysis of the code showed that there were four initialization loops on four separate arrays, all of size 352,100 float-type elements. After, a series of computations were performed on each element of two of these arrays, looped 120 times. While each series of initializations and computations had to be performed sequentially, operations on the individual array elements could be done entirely in parallel.

CUDA-Specific Design & Implementation Considerations

- Break down TOMO operations as finely as possible and write lightweight kernels for each (divide and conquer approach)
- One-dimensional block layout with dimension equal to maximum dimension given device's compute capability
- Each thread performs calculations on one array element
- Rewrite logic/program flow to erase possibility of thread divergence
- Maximize use of shared memory and registers by storing all arrays into device memory
- Minimize data transfer between host and device until copying final results back to host
- Implement streams to maximize concurrency

Performance Results



The five sample datasets are categorized into two charts above by execution times, namely longer times (> 2 min) and shorter times. (< 15 sec). General performance characteristics remained the same across the two categories of datasets.

The relative performance increase, in multiples of time, of CUDA vs. HPF, and CUDA vs. OpenMP (with four environment threads) across all datasets is shown on the left.

Conclusions

In general, the CUDA-powered implementation of TOMO exhibited a maximum of **16.5x** greater performance than HPF, and **4.7x** greater performance than OpenMP with four environment threads. While there is a very minor (generally < 0.001) difference in output results between CUDA and HPF, this difference is generally within the accepted margin of error granted.

As a result of this successful experiment, CERN has agreed to open up additional candidate projects that may be amenable for GPU-based parallelization using CUDA to Seneca College, for groups of students taking the parallel programming course in the future.

Seneca

CERN