CSC548 Parallel System Project

Abhishek Dhanotia (adhanot@ncsu.edu)
Fang Liu (fliu3@ncsu.edu)
Fei Meng (fmeng@ncsu.edu)

Project Description

CellBE and Nvidia GPUs are parallel architectures that could be exploited to accelerate simulations of parallel applications. In this project, we choose Nvidia GPU as the parallel computing platform. The parallelism could be two folders: one is thread level parallelism, in which the host offloads the kernel computation to available hardware thread blocks on a single GPU device; the other is process level parallelism, in which the entire computation are partitioned among multiple processes/hosts and each GPU device would take care of part of the assigned computation.

We choose three NAS PB 3.3 MPI benchmarks: CG, DT and IS. Using gprof profiling as well as source code inspection, we obtain their major computation hotspots that could be used for parallelism. Below are the description and characterization of the three benchmarks:

CG: stands for conjugate gradient: it approximate the largest eigenvalue of a sparse, symmetric, positive definite matrix, using inverse iteration. The dominant computation is the solution of a linear system which involves matrix-vector multiplications. The multiplication operation is repeated for each row in the sparse matrix for each non-zero element. This code is implemented in the “conj_grad” function in the cg.f file. 92.3% of program time is spent on this function when the program is run on 4 processors with Class A input size. Similarly, 83.7% time is spent on this function when running on 16 processors. For class B input on 16 processors, “conj_grad” function takes up 95.5% of program time.

DT: communication data flow graph (DGF) of processing nodes is built at the initialization phase. Each source node generates an array of random number and sends the array to nodes attaching to it. Each comparator node receives arrays from up-streaming nodes and combines them into one array and sends the new array to nodes attaching to it. Each Sink node does reduction on the received array and sends it to the root process 0. The dominant computation is random number generator “RandomFeatures”, which takes 100% and 50% of execution time with configuration of Class A, 21 Processes, graph type of BH, WH respectively. For Class B, 43 processes, it takes 27.27% and 42.86% of program time for BH and WH respectively.

IS: means integers sorting. Keys are generated by the sequential key generation algorithm, then sorted in parallel. The dominant computation is the rank calculation and random number generator on each node. Each node needs to sort its assigned partition ITERATION times. Each node will also call a huge number of random number generator to get its own sets. 50.47% of program time is spent on "randlc", which is called 33554457 times. The second time-consuming function is "rank", which is called 11 times each node. The configuration of the IS benchmark is NPROC=4 & CLASS=B.

Task Assignment:

Student Name	Benchmark	Language	Computation hotspot function
Abhishek Dhanotia	CG	Fortran	conj_grad
Fang Liu	DT	C	RandomFeatures
Fei Meng	IS	C	randlc

Based on the task assignment, the detailed approximating solution schedules are:

1. Read the source code of the profiled hotspot function thoroughly and identify the major loop bodies that could be optimized on CUDA threads. (Deadline: Nov 1, 2009)
2. Implement the identified optimization code in cuda kernel. Compile and debug the ported code to make sure correctness. (Deadline: Nov 5, 2009)
3. Collect CUDA results and compare them with results obtained on hery2. Based on the analysis, steps 1 and 2 might be repeated. (Deadline: Nov 8, 2009)
4. Write results and performance analysis for the assigned task. (Deadline: Nov 10, 2009)
5. Combine the individual report and finish the project. (Deadline: Nov 11, 2009)

The first 4 steps are mostly independent work. Group members will meet by each deadline. Each will report the progress and raise the potential problems and difficulties. We will discuss together and find out solutions.

References:

[1] M. Frumkin. Data Flow Pattern Analysis of Scientific Applications
[2] Stephen Whalen.
Optimizing the NPB CG benchmark for multi-core AMD Opteron microprocessors
[3]D. Bailey, E. et al. The NAS Parallel Benchmarks

URL: http://www4.ncsu.edu/~fmeng/csc548/