SE498 Parallel Computing
Lab 2: Analyzing the Performance of Matrix Multiplication using pthreads
Due: September 24, 2013 23:59

1. Objectives

- Explain the difference between interactive and batch processing systems
- Understand how programs execute in a supercomputing environment.
- Write and submit a simple job to a supercomputer for processing.
- Analyze the performance of pthreads when parallelized
- Understand the impact of cache hits and misses on program performance

2. Introduction

In operating systems, you constructed programs which required multiple threads to execute. These programs used the POSIX Pthreads mechanism to allow tasks to execute concurrently and independently. This allowed you to build a networked application, play a game of pong, or complete some other task.

In this lab, you are going to look at a program which computes a parallel matrix-vector product. The Matrix is distributed by block rows. Vectors are distributed by blocks. This version uses a random number generator to generate A and x. We will use the Valgrind cachegrind program to determine how the system performs as the size of the data changes.

3. Specific Activities

To begin with, you will want to download the tar file from the instructor's website, which includes the source code for this lab. After you have done this, make the code and try running it to see how it performs outside of the supercomputing environment.

The program should first run using a 5000 by 5000 matrix on a range of between 1 and 8 cores. You may want to prove it out on your local machine, but ultimately you will want to run it on the Glenn cluster. (Note: This will take longer to run than last week's examples.) The number of cores is the first parameter passed on the command line to the program. The last two define the array size.

After you have done this, you will want to change the size of the program to run a 5000000 by 5 and a 5 by 5000000 matrix over the range of 1 to 8 processors. Not that the total number of elements in the matrix is the same, just the shape of the matrix is different. In each case, you will compare the runtime of the program to the others, noting any differences in performance. A starting job file for this is provided on the next page. You will also want to look at the cache miss and hit rates, specifically for data, between the three different data arrangements. (Note: An explanation of the output from valgrind is given at http://valgrind.org/docs/manual/cg-manual.html).

```
#PBS -N matrixVectorMultiply
#PBS -l walltime=00:010:00
#PBS -l nodes=1:ppn=8
#PBS -j oe
# Change to the directory from which the job was submitted.
#cd $PBS_O_WORKDIR

# Make the code cleanly
make all

# Execute the program, varying the number of threads between 1 and 8.
echo starting with a 5000 by 5000 matrix and varying between 1 and 8 threads
valgrind --tool=cachegrind --cachegrind-out-file=valgrind.out
./pth_mat_vect_rand 1 5000 5000
cg_annotate valgrind.out

valgrind --tool=cachegrind --cachegrind-out-file=valgrind.out
./pth_mat_vect_rand 2 5000 5000
cg_annotate valgrind.out

valgrind --tool=cachegrind --cachegrind-out-file=valgrind.out
./pth_mat_vect_rand 3 5000 5000
cg_annotate valgrind.out
```

4. Deliverables

Each person is responsible for submitting a report with the following

1. What did you learn by doing this lab?
2. For each of the of the runs, determine what is the miss rate for the D1 cache? How does the rate change as the size of the matrix changes?
3. For each of the 3 array sizes used, determine the speedup and efficiency as more cores are added. Plot the results.
4. What things went right and wrong when doing this lab?
5. What conclusions can you draw about the effectiveness of using pThreads to parallelize code from this experience?