CSE 491-Section 2
Parallel Processing

Homework 1

Due in class, October 5, 2015

For your questions about the homework, please use the D2L discussion forum for the class so that answers by the TA, myself (or one of your classmates) can be seen by others.
Start early to make sure that you can run your jobs on time!

1. **(25 points)** Assume you are the director of Simulation University's HPC center and you are given a budget of $1.1 million to purchase a new system. You did some market research and found out that the compute nodes you like cost $5,000/node and each interconnect link (together with the necessary network hardware) costs $1000/piece.
   a) Calculate the cost of building a 3D torus of dimensions 4x4x4.
   b) Calculate the cost of building a 7 dimensional hypercube.
   c) You did a survey among the HPC center users and (on average) they rated the utility of an additional compute node as 5 and the utility of increasing the bisection bandwidth of the network by one link as 3. Think of the network topologies we discussed in class - bus, ring, 2D/3D torus (or k-D torus), hypercube, tree. How many nodes would you buy and how would you connect them to maximize utility of the system for users?
   d) What if the users rate the compute vs. communicate utilities as 5 vs 1?

2. **(20 points)** Suppose a program must execute 10^{12} instructions in order to solve a particular problem. Suppose further that a single processor system can solve the problem in 10^6 seconds (about 11.6 days). So, on average, the single processor system executes 10^6 or a million instructions per second. Now suppose that the program has been parallelized for execution on a distributed-memory system. Suppose also that if the parallel program uses p processors, each processor will execute $10^{12}/p$ instructions and each processor must send $10^9(p-1)$ messages. Finally, suppose that there is no additional overhead in executing the parallel program. That is, the program will complete after each processor has executed all of its instructions and sent all of its messages, and there won't be any delays due to things such as waiting for messages or load imbalances.
   a) Suppose it takes 10^{-9} seconds to send a message. How long will it take the program to run with 1000 processors, if each processor is as fast as the single processor on which the serial program was run?
   b) Suppose it takes 10^{-3} seconds to send a message. How long will it take the program to run with 1000 processors?

3. **(20 points)** Consider a multi-core CPU with p cores. The peak single-precision performance of each core is 10 GFlops/s. The peak bandwidth for the memory bus is 60 GB/s. Assume that the full bus bandwidth can be saturated using as few as a single core. You are evaluating a polynomial of the form, $x = y^2+z^3+yz$, which can be implemented as follows:

```
float x[N], y[N], z[N];
for (i=0; i < N; ++i)
        x[i] = y[i]*y[i] + z[i]*z[i]*z[i] + y[i]*z[i];
```

a) What is the arithmetic intensity (flops/byte) of each iteration of the loop? Think of how many bytes need to be transferred over the memory bus (both reads and writes) and how many floating point operations are performed in total.

b) Plot the theoretical peak performance of the system (GFlops/s) as a function of the number of cores, where 1 <= p <= 8. For what values of p would this computation's performance be compute-bound and for what values would it be memory-bound?

4. **(35 points)** The outer product is a commonly used tensor operation. The input is two vectors of length N and M. The result is a matrix of dimensions N by M. In code it would look as follows:

```
double x[N], y[M], A[N][M];
for (i=0; i<N; ++i)
        for (j=0; j<M; ++j)
                A[i][j] = x[i]*y[j];
```

a) Assume that there is only a single level of cache of size 64 KBs, a cache line is 64 bytes and the Least Recently Used (LRU) policy is adopted for cache eviction. For N = M = 1000, how many cache misses would the implementation above incur? Note that matrix A does not need to be read into cache, it is a write-only operation.
b) Repeat A, but now for N = M = 10000.
c) Based on your insights from 4.A and 4.B, give the pseudo-code for a "high performance" outer product implementation with a reduced number of cache misses. *Hint:* Remember our discussion about blocked matrix-vector and matrix-matrix multiplications (to be discussed on Monday, September 28th class).
d) Implement both algorithms on HPCC using the **Intel07** cluster **as separate programs** (use the skeleton code provided for your convenience). Tune your "high performance" implementation for minimal L1 cache misses. Compare the execution times for both programs, as well as L1 cache misses incurred for N = M = 1000, 5000, 10000, 20000. Summarize your performance results in a table.
e) Interpret your findings in part d – how did you tune your new implementation, which algorithm performs better and why?

Timing and compiling your programs: Although it is possible to use `gettimeofday` system call in C to take timings, for convenience, we will use the `omp_get_wtime()` command. In that case, you need to compile your codes with the OpenMP flag, i.e.:
Using GCC: `gcc -o hw1 -fopenmp hw1.c`
Using Intel Compiler: `icc -o hw1 -openmp hw1.c`

Measuring cache misses on HPCC: You can use *perf* command for this purpose which is loaded by default on HPCC systems. After you compile your program (e.g `gcc -o hw1 -fopenmp hw1.c`), then on terminal, simply execute it as:

```
perf stat -e cache-misses ./hw1
```

This will return the number of cache misses during the execution of your program.

Measuring your execution time and cache misses properly: The `omp_get_wtime()` command will allow you to measure the timing for a particular part of your program (see the skeleton code). However, on the dev-nodes there will be several other programs running simultaneously, and your measurements may not be very accurate. After you make sure that your program is bug-free and executes correctly, a good way of getting performance data for different programs and various input sizes is to use the interactive queue. Suggested use for intel07:

```
qsub -I -l nodes=1:ppn=8,walltime=00:20:00,mem=6GB,feature=intel07 -N myjob
```

This will allow exclusive access to an intel07 node for 20 minutes. If you ask for a long job, your job may get delayed. Default memory is 750 MBs per job, so it is very important that you ask for more as above.

Turn in instructions: You will return a hardcopy homework including report and interpretation of your results for the programming assignment. You will turn in your source codes and output files by copying them to our class directory on HPCC, which is:

```
/mnt/research/cse491_fs15_s2/
```

This is a write-only directory for the group. For homework1, we have created a sub-directory called **hw1**. All your files must reside in this sub-directory in a separate folder named with your MSU NetID. For example, for this homework, I would need to copy all my files which resides on myhw1 folder on HPCC as:

```
cp -r myhw1 /mnt/research/cse491_fs15_s2/hw1/hma
```

Subsequent homeworks will have their own sub-directories. So make sure that you copy your files to the correct location!