CSE/CMSE 822: Parallel Computing
Fall 2017, Homework 2
Due 12 pm, Thursday, Sep 21st

This homework is 12% of your ***homework grade*** *(not your total grade).*

Important note: *Please use a word processing software (e.g., MS Word, Mac Pages, Latex, etc.) to type your homework. Follow the submission instructions at the end to turn in an electronic copy of your work.*

1) [25 pts] Performance Modeling: Finite difference time-domain (FDTD) method for 3D Maxwell's equations

The following kernel performs a portion of the finite difference time-domain (FDTD) method for computing Maxwell's equations in a three-dimensional space, part of one of the SPEC06fp benchmarks:

```
for (int x=0; x<NX-1; x++) {
   for (int y=0; y<NY-1; y++) {
      for (int z=0; z<NZ-1; z++) {
         int index = x*NY*NZ + y*NZ + z;
         if (y>0 && x >0) {
            material = IDx[index];
            dH1 = (Hz[index] – Hz[index-incrementY])/dy[y];
            dH2 = (Hy[index] – Hy[index-incrementZ])/dz[z];
            Ex[index] = Ca[material]*Ex[index]+Cb[material]*(dH2-dH1);
         }
      }
   }
}
```

Assume that dH1, dH2 are single-precision floats, and Hy, Hz, dy, dz, Ca, Cb, and Ex are all single-precision floating-point arrays. Assume IDx is an array of unsigned int (4 bytes).

a. [10 pts] What is the arithmetic intensity of this kernel?
b. [10 pts] Assume this kernel is to be executed on a processor that has 30 GB/sec of memory bandwidth. Under what conditions will this kernel be memory bound, and under what conditions will it be compute bound?
c. [5 pts] Develop a roofline model for this processor, assuming it has a peak computational throughput of 85 GFLOP/sec. Mark the arithmetic intensity of the kernel on your plot and determine its expected performance.

2) [75 pts] Memory hierarchy: Cache blocked matrix transpose

The transpose of a matrix interchanges its rows & columns:

$$\begin{bmatrix} A_{00} & A_{01} & A_{02} & A_{03} \\ A_{10} & A_{11} & A_{12} & A_{13} \\ A_{20} & A_{21} & A_{22} & A_{23} \\ A_{30} & A_{31} & A_{32} & A_{33} \end{bmatrix} \xrightarrow{} \mathit{Transpose} \begin{bmatrix} A_{00} & A_{10} & A_{20} & A_{30} \\ A_{01} & A_{11} & A_{21} & A_{31} \\ A_{02} & A_{12} & A_{22} & A_{32} \\ A_{03} & A_{13} & A_{23} & A_{33} \end{bmatrix}$$

A simple nested loop is sufficient to transpose an NxM matrix:

```
for (i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
      output[j][i] = input[i][j];
   }
}
```

However, as we discussed in class, this simple implementation can not make efficient use of the memory hierarchy. In particular, assuming row-major storage, while the accesses to the input matrix can achieve ideal cache miss rates, the accesses to the output matrix may result in high number of cache misses.

a. [25 pts] *Implementation*: Following the cache blocking idea outlined in class, implement a cache optimized version of the matrix transpose operation. For this purpose, you will use the source code provided which already includes the data structures you will use, an implementation of the simple transpose algorithm, as well as time measurement mechanisms. Your task is to fill in the `optTranspose` function in the file src/optTranspose.c.

b. [20 pts] *Performance Analysis*: Test the performance of your implementation for a set of different N (number of rows), M (number of columns) and B (blocking factor) values. Identify the different regimes as indicated by the relative speed up of the optimized implementation over the naïve one, and outline the conditions (in terms of matrix sizes, shapes, etc.) under which you observe these regimes. **Hint:** Experiment with different matrix sizes (small to large) and input matrix shapes (short&wide rectangle, square, tall&skinny rectangle) while using a reasonable blocking factor.

c. [20 pts] *Cache Performance Measurement*: Using a cache performance measurement tool (see instructions below on TAU) and your knowledge about how caches work, explain the root causes for the different regimes you observe in part b. For example, you can point to the ratio of cache misses you observe at various levels (L1, L2 or L3) using the naïve vs. optimized implementations to explain the relative performance differences. Can you make an estimation on the relative L1 and L2 latencies based on the performance and cache misses data you obtained?

d. [15 pts] *Inference about the Memory Hierarchy:* Experiment with different blocking factors B to make inferences about the memory hierarchy. In particular, can you estimate the size of the L1 and L2 caches based on the performance and cache misses data you obtained? Compare your estimations with the hardware specifications.

Instructions:

- **Compiling your programs:** While you can use any compiler (different versions of GCC or Intel compiler), throughout this assignment please use GCC/4.8.3. GCC/4.8.3 together with OpenMPI 1.8 are necessary for compiling and running with TAU, a performance analysis tool that we will be using. So whenever you login to HPCC or in your (batch or interactive) queue runs, make sure to execute the following commands:

  ```
  module unload GNU OpenMPI
  module load GNU/4.8.3 OpenMPI/1.8
  ```

  After loading the proper modules, you can compile the provided source file in one of the two ways below.

  ```
  make transpose
  make transpose-tau
  ```

  The first option produces uninstrumented native code (useful for timing only data), the second option produces TAU instrumented code (necessary for collecting cache utilization data). The TAU instrumented code can be significantly slower, therefore use the native code to collect timings.

- **Running the executable**: After compilation, the provided source can be executed either in the "test" or in the "perf" modes, conveniently on any of the dev-nodes:

  ```
  transpose.x test inputfile
  transpose.x perf N M B
  ```

  where
  inputfile contains a set of test runs. It starts with a line indicating the number of tests, followed by the tests themselves, each on a separate line. Each test is specified with N M B values.
  N: number of matrix rows
  M: number of matrix columns
  B: blocking factor.

  We provide you a sample test file, which is the exact file that we will be using to test the accuracy (not performance) of your submission.

- **Measuring cache misses on HPCC:** For this purpose, we will be using the TAU performance analysis tool as mentioned above. Using hardware counters available through PAPI, TAU allows the measurement of an extensive list of CPU events. To compile your programs with TAU, follow the instructions below:

  i. Append the following lines to your Bash profile by editing the .bashrc file, which is a hidden file in your home directory and can be edited using vim or emacs, e.g. vim ~/.bashrc. See the bash_changes.txt file for easily copying & pasting these lines into the .bashrc file.

  ```
  # PAPI
  export PATH="/mnt/home/ohearnku/.local/papi/5.5.1/bin:$PATH"
  export INCLUDE_PATH="/mnt/home/ohearnku/.local/papi/5.5.1/include:$INCLUDE_PATH"
  export LD_LIBRARY_PATH="/mnt/home/ohearnku/.local/papi/5.5.1/lib:$LD_LIBRARY_PATH"
  export MANPATH="/mnt/home/ohearnku/.local/papi/5.5.1/man:$MANPATH"

  # TAU
  ```

```
export PATH="/mnt/home/ohearnku/.local/tau/2.26/x86_64/bin:$PATH"
export INCLUDE_PATH="/mnt/home/ohearnku/.local/tau/2.26/include:$INCLUDE_PATH"
export
LD_LIBRARY_PATH="/mnt/home/ohearnku/.local/tau/2.26/x86_64/lib:$LD_LIBRARY_PATH"
export TAU_MAKEFILE="/mnt/home/ohearnku/.local/tau/2.26/x86_64/lib/Makefile.tau-papi-
mpi"
export TAU_OPTIONS=-optCompInst
export TAU_METRICS="P_WALL_CLOCK_TIME:PAPI_L1_DCM:PAPI_L2_TCM:PAPI_L3_TCM"
```

ii. To see the complete list of available hardware counters, run on a dev-node

```
papi_avail
```

where supported counters have "Yes" in the third and fourth columns.

iii. Now, compile the source file using the Tau MPI wrapper by issuing the following command at the terminal:

```
make tranpose-tau
```

iv. You can now run the executable *instrumented* with TAU on a dev-node or using the queue system:

```
mpiexec -np 1 tau_exec ./transpose-tau.x perf 1000 100000 200
```

v. The performance data collected by TAU is saved automatically into default directories/files. Below are the directories/files you will see (as a result of the TAU_METRICS options specified):

MULTI__P_WALL_CLOCK_TIME/profile.0.0.0
MULTI__PAPI_L1_DCM/ profile.0.0.0
MULTI__PAPI_L2_TCM/profile.0.0.0
MULTI__PAPI_L3_TCM/profile.0.0.0

vi. Normally, these files are intended to be viewed through a visualization software, for example using the paraprof command in the directory where the data was created:

```
paraprof &>/dev/null &
```

To view L1 data cache misses by function, click on the PAPI_L1_DCM item, and within the window it opens, click on either of the colored sections of the bar chart.

vii. Paraprof requires X11, which can sometimes be tricky to run on a Windows or Mac, or if you do not have a very fast internet connection. We recommend simply viewing the profile.0.0.0 files for each event using the cat command or a text editor (vim or emacs). The contents are simple enough to be understood in this way.

- **Measuring your execution time and cache misses properly:** The wall-clock time measurement mechanism (based on the gettimeofday() function) implemented in the provided transpose.c file will allow you to measure the timings for a particular part of your program (see the skeleton code) precisely. However, while they are convenient to use, the dev-nodes will have several other programs running simultaneously, and your measurements may not be very accurate due to the “noise”.

After making sure that your program is bug-free and executes correctly on the dev-nodes, a good way of getting reliable performance data for various input sizes is to use the interactive queue. **Please use the intel16 cluster for all your performance measurement runs!**

You can submit an interactive job request that will give you a dedicated node as follows:

```
qsub -I -l nodes=1:ppn=28,walltime=00:20:00,mem=120gb,feature=lac
```

This will give you exclusive access to an intel16 node for 20 minutes. If you ask for a long job, your job may get delayed. **The default memory limit is set to be 750 MBs per job on HPCC systems, so it is very important that you ask for more as above**.

Once you are granted an interactive job, make sure to run your jobs one after the other (i.e., do not run them as background jobs, and do not worry about it if you do not know what background jobs mean). This is important because having multiple background jobs running simultaneously will create "noise" in the data you obtain.

- **Batch jobs**: Interactive jobs may sometimes be delayed significantly, and therefore insisting on this option may be very counter-productive. For your convenience, we have provided a sample batch script named **job_script.qsub**. Once you make sure that your program is running correctly on the dev-nodes, you can submit jobs to the intel16 cluster using this sample job script. It is possible to set email alerts for when your job starts and ends (see the corresponding comments in the sample job script and replace with your e-mail address). Last but not the least, the sample job script runs the compiled code in the test mode only. You can (and should) modify the end part of the job script based on what kind of performance data you would like to collect (i.e., using the naive code for various matrix sizes, collecting cache utilization data using the TAU-instrumented version, etc.)

- **Submission:** Your submission will include exactly two files:
  - Your final optTranspose.c file. Note that you should only modify the body of the optTranspose function within this file. *Do not make modifications to any other source file, or the function name, arguments, return value, etc. as this will prevent your assignment from being properly graded.* Your code is required to compile and run within the framework provided to you *without modification*.
  - A pdf file named "HW2_*yourMSUNetID*.pdf", which contains your answers to the non-implementation questions of the assignment.

Further instructions regarding the use of the git system and homework submission is given in the "Homework Instructions" under the "Reference Material" section on D2L. Make sure to strictly follow these instructions; otherwise you may not receive proper credit.