CSE/CMSE 822: Parallel Computing
Fall 2017, Homework 3

Due 12 pm, Thursday, October 5th

This homework is 8% of your ***homework 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.*

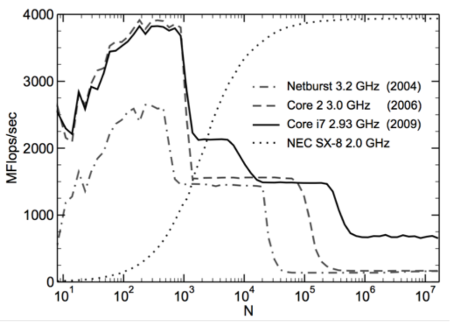
1) [50 pts] Compilers and auto-vectorization

As discussed over the last few lectures, writing high performance software in a high-level programming language partially depends on the compiler's ability to generate optimized machine code. In this problem, you are asked to analyze this aspect of high performance computing in the context of the vector triad kernel.

a. For the system you are running on (i.e., an intel16 node), determine the clock speed of the processor and the cache size/layout by looking up its specifications (HPCC Wiki and Intel's website are useful references, or alternatively you may use Unix commands such as lscpu). Assuming the processor is capable of one floating point operation per clock cycle, what would you expect the theoretical peak performance of this system to be?

b. Now consider the fact that modern CPUs support wide vector instructions (SSE, AVX, etc.) and the fused multiply add operation (FMA: https://en.wikipedia.org/wiki/FMA_instruction_set). How does this change your expectations regarding the peak CPU performance of part (a)?

c. Using the provided vector_triad.c file, plot the performance of the vector triad kernel at several vector lengths *N* (similar to the one shown on the right but in Gflop/s) using different compilers and options (see below). On these plots, place horizontal lines representing your theoretical peak performance estimations from parts (a) & (b) (i.e., 1. assuming one flop/cycle, 2. with SSE vectorization, 3. with AVX vectorization, and 4. using FMA).



You are expected to generate two plots, one using the GCC compiler, another using the Intel compiler both of which are available at HPCC. For GCC, experiment with compiler options such as `-O1, -O3, -O3 -march=native`. For Intel, experiment with options such as `-O1, -O3, -O3 -xSSE, -O3 -xAVX, -fast.`

d. How does the maximum measured performance using different compilers & options compare to the different peaks you estimated? Can you tell which compiler options have enabled what kind of optimizations?

e. Are there any sudden features in your plot? Explain them in the context of the hardware architecture (i.e., cache layout) of your system.

2) [50 pts] Stream benchmark and memory bandwidth

Besides the peak CPU performance, another important factor impacting performance is the memory bandwidth as we discussed within the context of the Roofline model. In this problem, you are asked to analyze the memory performance of your system.

a. Download, compile, and run the STREAM memory bandwidth benchmark (http://www.cs.virginia.edu/stream/). Report the bandwidth computed by STREAM for your system (i.e., Intel16 at HPCC). *Note: For properly using the software, it is highly recommended to read the instructions and comments included within the source code.*

b. Write a simple program (in C, C++, or Fortran) for performing a vector copy operation, e.g. in C:

```
double c[n], a[n];
for (k=0; k<ntrials; k++) {
  for (i=0; i<n; i++)
    c[i] = a[i];
  }
```

For starter code, you can use the vector_triad.c code from Problem 1. Measure the memory bandwidth of your code, in GB/s, for `n=2000000` and compare this to the STREAM result. What factors may impact the performance of your code?

c. Now run your code for `n=1, 10, 1000, 10000, 100000, 1000000, 10000000, 100000000`, again using different compilers and options as in Problem 1. Plot the results for memory bandwidth versus n. Also modify the STREAM benchmark to run at the same values of n and include this also on your plot. How does your simple vector copy code compare to STREAM results?

d. Discuss how the memory bandwidth performance varies with n and why. In particular, do you observe regions where your measured memory bandwidth is much higher than what it should be? Can you use the information in such regions to estimate the bandwidth from various levels of the cache?

Instructions:

- **Compiling your programs:** No makefiles are provided in this assignment. You are expected to do manual compilation or write up your own makefiles (you can use the one in the previous assignment for starter). A simple way to do manual compilation with gcc is as follows:

  ```
  gcc -O3 vector_triad.c -lm
  ```

  Note: *–lm* option is needed to link with the math library.

- **Measuring your execution time and performance properly:** The wall-clock time measurement mechanism (based on the gettimeofday() function) implemented in the provided source 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 also 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 only one file:
  - A pdf file named "HW3_*yourMSUNetID*.pdf", which contains your answers to the questions in 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.