Assignment 1: Maximum CPU rates

ITCS 4182: Introduction to High Performance Computing

Deadline: January 29th, 2017

1 Microbenchmark: Maximum Flops and Iops

The goal of this exercise is to try to reach the maximum rate of floating point operation a processor can perform. Let's do this on mamba if the accounts are created.

Expectations of performance. First, document which machine you are going to use for this experiment. What is it composed of? What kind of processor it has? Which architecture is the processor built on? (For instance, some core i7 are based on the ivy-bridge architecture, some on the haswell architecture, ...)
Question: What is the maximum number of floating point operations this machine can perform per second?
Question: What is the maximum number of integer operations this machine can perform per second?
Hint: Remember what matters: frequency, core count, but also instruction set and the number of execution units within the cores and what they can do.

Realization. Let's write two programs one to reach peak Flops and one to reach peak Iops in C or C++. (I recommend C++.) Note, that the purpose of this exercise is to get the highest amount of operations performed. There is no restriction on what operations are performed or even if it makes sense to perform them. The problem is really "do as many flops/iops as you can".
Question: Write a code that gets peak Flops
Question: Write a code that gets peak Iops
Hint: often when writing this kind of microbenchmark, the computational kernel is not doing anything useful and the compiler might see that and optimize it out. Putting the computation in its own function in a different file often prevents the compiler from realizing that.

Measurement. Measure the performance that you obtain and see if it matches your expectation.
Question: How many Flops did the code achieve?
Question: How many Iops did the code achieve?
Question: Does that match expectation? Where does the discrepancy come from?
Question: Can you do better?

Reporting. Let's write a report that summarize the work done.
Question: Write a short report that explain your techniques, explanation and findings.
Question: Submit an archive of the report and code. Or give a link to the git repository that stores that.

A Figuring out expected performance

Check out the reference slide of the lecture for external pointers.

B Compiler

Make sure you have a compiler for your processor (usually a 64-bit compiler). Some IDE come with 32-bit compiler on windows.

C Measuring time

Remember when measuring time to measure a long period to remove startup overheads and innacuracy of the measure. In C, `gettimeofday` is typically the time function you want to use. In C++, use the new chrono features `http://en.cppreference.com/w/cpp/chrono`.

Some forum post say: Tracking Time in Windows `https://stackoverflow.com/questions/17250932/how-to-get-the-time-elapsed-in-c-in-milliseconds-windows`

D Using multiple threads with OpenMP

You might need to use multiple threads. The simplest way of doing that is using OpenMP, which is a C/C++ extension supported by most compilers. You can find a simple openmp code that starts as many threads as execution contexts on canvas.

OpenMP Instructions Visual Studio. To enable OpemMP, go to Project Properties – Configuration Properties – C/C++ – Language – Open MP Support. Also, Visual Studio uses omp for OpenMP instead of openMP

E Manual vector instruction

You might also need to use compiler built-in/intrinsic functions to manually leverage SIMD operations. In most compilers they become available when you include `immintrin.h` but you might need to pass special flags to enable them (such as -march=native or -mavx).

Check out reference slide of lecture for links documenting these functions.

AVX2 Instructions for Visual Studio. To enable AVX2, go to Project Properties – Configuration Properties – C/C++ – Code Generation – Enable Enhanced Instruction Set. Then, include intrin.h and immintrin.h for at the top. For a list of functions, see here. `https://msdn.microsoft.com/en-us/library/hh977022.aspx`

F Reading Assembly

Reading the assembly code generated by the compiler might help understanding the performance that you get. In GCC, option -S gives you the assembly, use -g -fverbose-asm