Abstract

Introduction:

The initial goal was to provide an analysis of FLOPS/Watt of 3 different systems, a laptop with a first generation i7 processor, a desktop with a second generation i7 processor, and a Raspberry Pi 3b. Unfortunately, the power meter recommended by the reference paper has apparently been lost in the mail, and we no longer expect it to arrive before the due date. Instead, we will be comparing the laptop and desktop using temperature and processor load monitors on periods when they are either compute or memory bound.

Methodology:

To find flops per watt, we need to solve the equation

Because we don’t have a direct measure of power, we must use

Where T is the temperature (which we can measure) and C is some conversion factor which we’ll have to infer. I do have a rough upper bound of 20 Watts on the laptop and 60 Watts on the desktop, and the CoreTemp tool I’m using also provides a real-time but not a logged display of estimated wattage which will allow us to guess at a value of C which will hopefully provide a meaningful comparison. We will have T polled every 5 seconds on those two platforms.

For the memory-bound benchmark, we run threads which use 4 400x400 matrices. These matrices are populated by random reals ϵ [0, 1.0). They are then naïvely matrix multiplied, such that each matrix multiply requires 400 multiplications per row and then a summation of 400 products for each of 400. Therefore, each matrix multiply requires (2000\*(2000+2000)) = 8 million floating point operations, and each thread requires 32 million floating point operations to complete. Each loop iteration launches 4 threads and thus requires 128 million floating point operations.

For the high arithmetic intensity, we have threads which do two multiplies and one addition on 8 combinations of different random numbers. These get passed on to the next iteration for 10 million “steps”. This, thus, limits memory access because the only load necessary (in theory) is the initial randomization of the numbers, and they may then be carried on through successive steps. In addition, there are 8 threads per iteration of the loop. Each thread requires 10,000,000 \* 8 \* 3 = 240 million floating point operations. Because this doesn’t come even close to 100% processor utilization, but instead is about 35-40%, we also employ 3 iterations of the python program itself. This pushes the processor load to 90-100% for the full 10 minutes, at least on the laptop.