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:

A note on FLOPS: FLOPS is Floating Point OPerations per Second. Flops is Floating point OPerations.

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

On the laptop and desktop we have an internal measure of power, though it isn’t clear how trustworthy it is. On the Raspberry Pi, as we don’t have a direct measure of power, we must infer it using

Where T is the temperature (which we can measure with the sensors command) and C is some conversion factor which we’ll have to assume. If power is I \* V, then we have an upper bound on the Pi of 2.4A \* 5.25V, or 12.6 Watts. We poll power on the laptop and desktop every 5 seconds. We do have voltage, but not amperage, so that gives us one piece of the equation- we may assume amperage remains constant and calculate power by multiplying by the voltage we get by 2.4, or we may try to infer C from equation 2. Given the data from the laptop and the desktop, it seems that solving equation 2 for C gives us some value between .2 and .3. Specifically, because temperature is a somewhat trailing indicator, after the temperature stabilizes, we can take the average ratio of power to temperature and use that as C. Power jumps up immediately, and the ratio is skewed high because temperature is artificially low. The Laptop

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 results. Therefore, each matrix multiply requires 400^3 = 64 million Flops, and each loop iteration launches 4 threads and thus requires 256 million Flops. In addition, we run 4 of these programs simultaneously to achieve a 100% load.

Initially, we had attempted to use an optimized library (lib-atlas via numpy) to calculate this benchmark- however we had FLOPS that could only be described as ludicrously excessive. The optimized libraries were “cheating”, and not calculating the matrices the naïve way. Therefore, more than 6000 iterations were being completed in a single 10 minute run with a matrix 5 times larger. In contrast, only 4 were completed in the naïve sense, which means the optimizations have improved execution speed by more than 3 orders of magnitude- and that’s impressive.

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 Flops. Because this doesn’t come even close to 100% processor utilization, but instead is about 35-40%, we also employ 4 instances of the python program itself. This pushes the processor load to 90-100% for the duration of the experiment.

In both cases, the programs run for a minimum of 10 minutes. We don’t interrupt them at 10 minutes exactly, however, because we wouldn’t be able to accurately calculate how many Flops had been completed- instead they finish all threads that are running when ten minutes pass. Our programs report how many iterations have been completed and how long they took. Then, FLOPS can be found:

Where F is the number of Flops in an iteration, I is the number of iterations and Program\_Num is the number of programs. The numerator gives us the total Flops, the denominator is the length in seconds of the longest running program. See the data sheets for more detail on calculations. We calculate power by taking the average of all reported values when the load of the processors is greater than 50%, which has historically been a good indicator that the programs were running (typical load is around 15% for the laptop, 0% for the desktop). Then we use equation 1.

Results

We ran both programs several times, and include a report of a typical run on each platform and program combination.