ECE471: Embedded Systems – Homework 11
Power/Performance

Due: Thursday, 8 December 2016, 9:30am EST

Power and Energy

Table 1: OpenBLAS HPL N=10000 (Matrix Multiply)

Machine	Processor	Cores	Frequency	Idle Power	Load Power	Time	Total Energy
Raspberry Pi 2	Cortex-A7	4	900MHz	1.8W	3.4W	454s	1543J
Dragonboard	Cortex-A53	4	1.2GHz	2.4W	4.7W	241s	1133J
Raspberry Pi 3	Cortex-A53	4	1.2GHz	1.8W	4.3W	178s	765J
Jetson-TX1	Cortex-A57	4	1.9GHz	2.1W	13.4W	47s	629J
Macbook Air	Broadwell	2	1.6GHz	10.0W	29.1W	14s	407J

1. Table 1 shows the energy use of various machines when doing a large Matrix-Matrix multiply.

   (a) Which machine has the lowest under-load power draw?
   Pi 2

   (b) Which machine consumes the least amount of energy?
   Broadwell Macbook Air

   (c) Which machine computes the result fastest?
   Broadwell Macbook Air

2. Consider a use case with an embedded board taking a picture once every 60 seconds and then performing a matrix-multiply similar to the one in the benchmark (perhaps for image-recognition purposes). Could all of the boards listed meet this deadline?
   No, only the Jetson and Macbook Air can meet the deadline

3. Assume a workload where a device takes a picture once a minute then does a large matrix multiply (as seen in Table 1). The device is idle when not multiplying, but under full load when it is.

   (a) Over an hour, what is the total energy usage of the Jetson TX-1?
   Each Minute = (13s Idle * 2.1W) + (47s Load *13.4W) = 657J
   Each hour = 60*657 = 39,426J

   (b) Over an hour, what is the total energy usage of the Macbook Air?
   Each Minute = (46s * 10W) + (14*29.1) = 867J
   Each hour = 867*60 = 52,044J

4. Given your answer in the previous question, which device would you choose if you were running this project off of a battery?

   Jetson-TX1. In general the lowest energy will lead to best battery life, although this can be complicated depending on the battery's characteristics and the device's peak power draw

Performance

Raspberry Pi Model 2 results, no Optimization

```
$ perf stat -e instructions,cycles,L1-dcache-load-misses,branch-misses \
./dgemm_naive 250
Will need 2000000 bytes of memory, Iterating 10 times

 Performance counter stats for './dgemm_naive 250':

    5,042,022,526      instructions              #  0.48  insns per cycle
    10,414,207,828      cycles
        38,943,964      L1-dcache-load-misses
         1,234,120      branch-misses

      11.639344013 seconds time elapsed
```

Raspberry Pi Model 2 results, `-O2` Optimization

```
$ perf stat -e instructions,cycles,L1-dcache-load-misses,branch-misses \
./dgemm_naive.O2 250
Will need 2000000 bytes of memory, Iterating 10 times

 Performance counter stats for './dgemm_naive.O2 250':

     1,049,408,721      instructions              # 0.38  insns per cycle
     2,779,835,783      cycles
        43,771,532      L1-dcache-load-misses
           780,467      branch-misses

       3.123459828 seconds time elapsed
```

5. Performance questions

   You are running a matrix-multiply benchmark on pi2 with no optimizations and you obtain the perf results at the top. Your friend recommends compiling with the `-O2` compiler flag and you obtain the results on the bottom.

   (a) Which is faster, none or O2 optimization?
   `-O2` is faster at 3s

   (b) How many instructions were executed in none vs O2?
   none: 5 Billion, O2: 1 Billion

   (c) Some metrics, such as IPC and cache misses, are actually worse in the optimized code. How can these be worse yet the program still runs faster? Despite IPC being lower (worse) and cache misses being higher (worse) the program ran around 4x faster. Most of this is probably due to executing 5x fewer instructions. Non-optimized code often does a lot of redundant things (especially redundant memory accesses), so even just using -O2 optimization can make a big difference.