Lecture 2: Single-processor Computing

CMSE 822: Parallel Computing

Prof. Sean M. Couch

Exercise 1.1

Exercise 1.1. Let us compare the speed of a classical FPU, and a pipelined one. Show that the result rate is now dependent on n: give a formula for $r(n)$, and for $r_\infty = \lim_{n\to\infty} r(n)$. What is the asymptotic improvement in r over the non-pipelined case?
Next you can wonder how long it takes to get close to the asymptotic behaviour. Show that for $n = n_{1/2}$ you get $r(n) = r_\infty/2$. This is often used as the definition of $n_{1/2}$.

On FPU: n results takes: $t(n) = n\ell\tau$ { ℓ : number of stages; τ : clock cycle time

rate of results: $r_{serial} \equiv \left(\frac{t(n)}{n}\right)^{-1} = (\ell\tau)^{-1}$

Pipelined: $t(n) = [s + \ell + n - 1]\tau$, s: setup cost

$= [n + n_{1/2}]\tau \rightarrow n_{1/2} = s + \ell - 1$

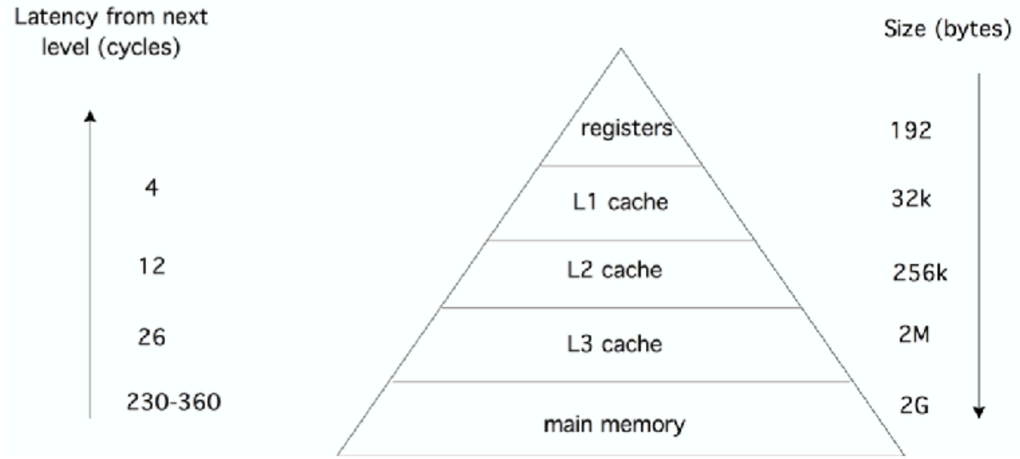
$r_{pipe} = \left(\frac{[n+n_{1/2}]\tau}{n}\right)^{-1} = \left(\tau + \frac{n_{1/2}\tau}{n}\right)^{-1} \longrightarrow r_\infty = \tau^{-1}$

new let $r(n) = r_\infty/2 \Rightarrow \frac{t(n)}{n} = 2\tau \Rightarrow n_{1/2} = \frac{t(n)}{2\tau}$

Exercise 1.5

Exercise 1.5. The L1 cache is smaller than the L2 cache, and if there is an L3, the L2 is smaller than the L3. Give a practical and a theoretical reason why this is so.

The L1 cache has lower latency and higher bandwidth and is therefore more expensive to manufacture. Also, suppose the L2 was smaller than the L1, then there would never be reuse of data out of it: it is not possible that data would be in L2 but not be in L1.

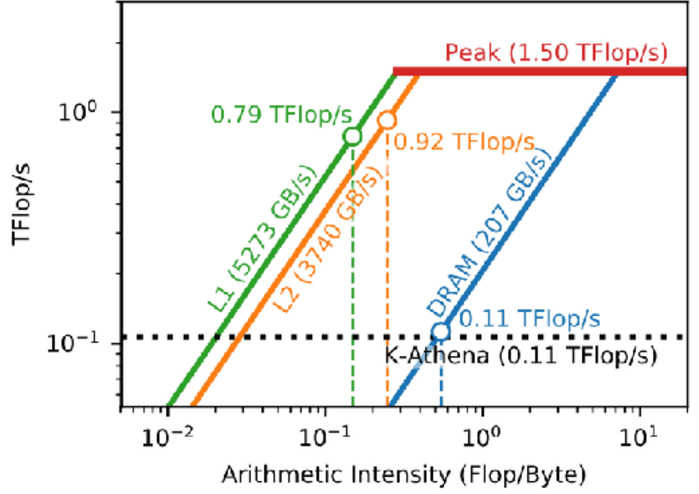


Why not have more levels of cache? Diminishing returns…

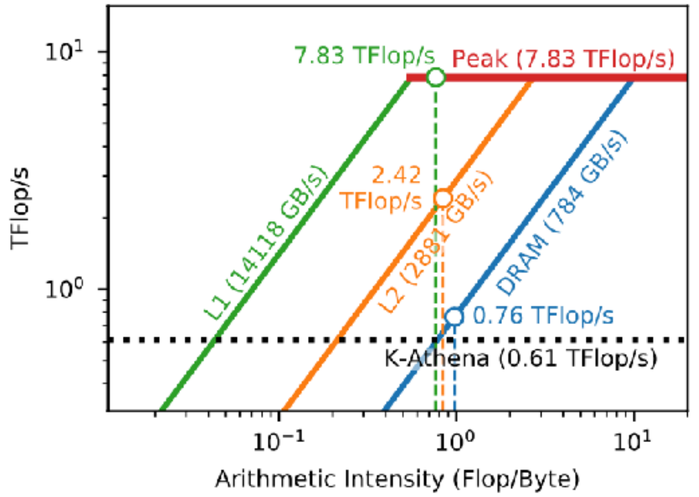
Exercise 1.15

Exercise 1.15. How would you determine whether a given program kernel is bandwidth or compute bound?

Profiling!



(a) CPU Roofline



(b) GPU Roofline

Fig. 2. Roofline models of a 2 socket Intel Xeon Gold 6148 "Skylake" CPU node on NASA's Electra (2a) and a single NVIDIA Tesla V100 "Volta" GPU on ORNL's Summit (2b). For both cases shown here and all other architectures we tested, DRAM bandwidth (or MCDRAM bandwidth for KNLs) is the limiting bandwidth for K-ATHENA's performance.

Grete et al. 2019, IEEE TPDS

Exercise 1.21

Exercise 1.21. Give an example of a doubly-nested loop where the loops can be exchanged; give an example where this can not be done. If at all possible, use practical examples from this book.

Independent: matrix-vector product

Dependent: successiver over-relaxation iteration (i.e., Gauss-Seidel)

PCA Questions

Division vs. multiplication

- Division algorithms are *iterative*. Typically at least 2x more expensive.
- Should you avoid division? Yes, if you can… (division by true variable). Compile will try to help. Try. BUT! Never let optimization slow you down ;-)
- See Wikipedia entry on division algorithms

PCA Questions

Section 1.7.2 talks about loop unrolling, in order to increase the number of operations per second. Is this sort of optimization something a compiler can do? Also, in Table 1.2, there's a large reduction in the cycle time; for more complicated loops, is there a similar speedup?

- Can compiler unroll? Yes! See this StackOverflow response.
- More complicated loops? Should see similar speed up, maybe better since there will be more work in the inner loop. Watch out for loop dependencies, though!

PCA Questions

Is there a way for a program to tell how large the L1 cache is and make adjustments accordingly? If there were a magic function int l1Size = getL1Size(), you could use that pretty effectively I imagine.

- Not during runtime, really. Need to profile.
- During compilation, or pre-compilation, can set or detect size of cache for specific architecture and make automated optimizations accordingly
- See “cache oblivious” algorithms.

PCA Questions

In Page 57, it says "we note that we use the vectors sequentially, so, with a cacheline of eight doubles, we should ideally see a cache miss rate of 1/8 times the number of vectors m". How does this 1/8 calculated?

- double is 8 bytes, so 8 fit into a 64K cache line
- after every eight entries in the vector, a cache miss will occur

In-class Group Exercise

Spatial and temporal locality

Consider the following pseudocode of an algorithm for summing n numbers $x[i]$ where n is a power of 2:

```
for s=2,4,8,...,n/2,n:
  for i=0 to n-1 with steps s:
    x[i] = x[i] + x[i+s/2]
sum = x[0]
```

Analyze the spatial and temporal locality of this algorithm, and contrast it with the standard algorithm

```
sum = 0
for i=0,1,2,...,n-1
  sum = sum + x[i]
```

In-class Group Exercise
Reuse

Exercise 1.17. Consider the following code, and assume that `nvectors` is small compared to the cache size, and `length` large.

```
for (k=0; k<nvectors; k++)
  for (i=0; i<length; i++)
    a[k,i] = b[i] * c[k]
```

How do the following concepts relate to the performance of this code:

- Reuse
- Cache size
- Associativity

Would the following code where the loops are exchanged perform better or worse, and why?

```
for (i=0; i<length; i++)
  for (k=0; k<nvectors; k++)
    a[k,i] = b[i] * c[k]
```

Group work on Homework 1!