Advanced Cache Optimizations

Joannah Nanjekye

July 23, 2024

Advanced Cache Optimizations

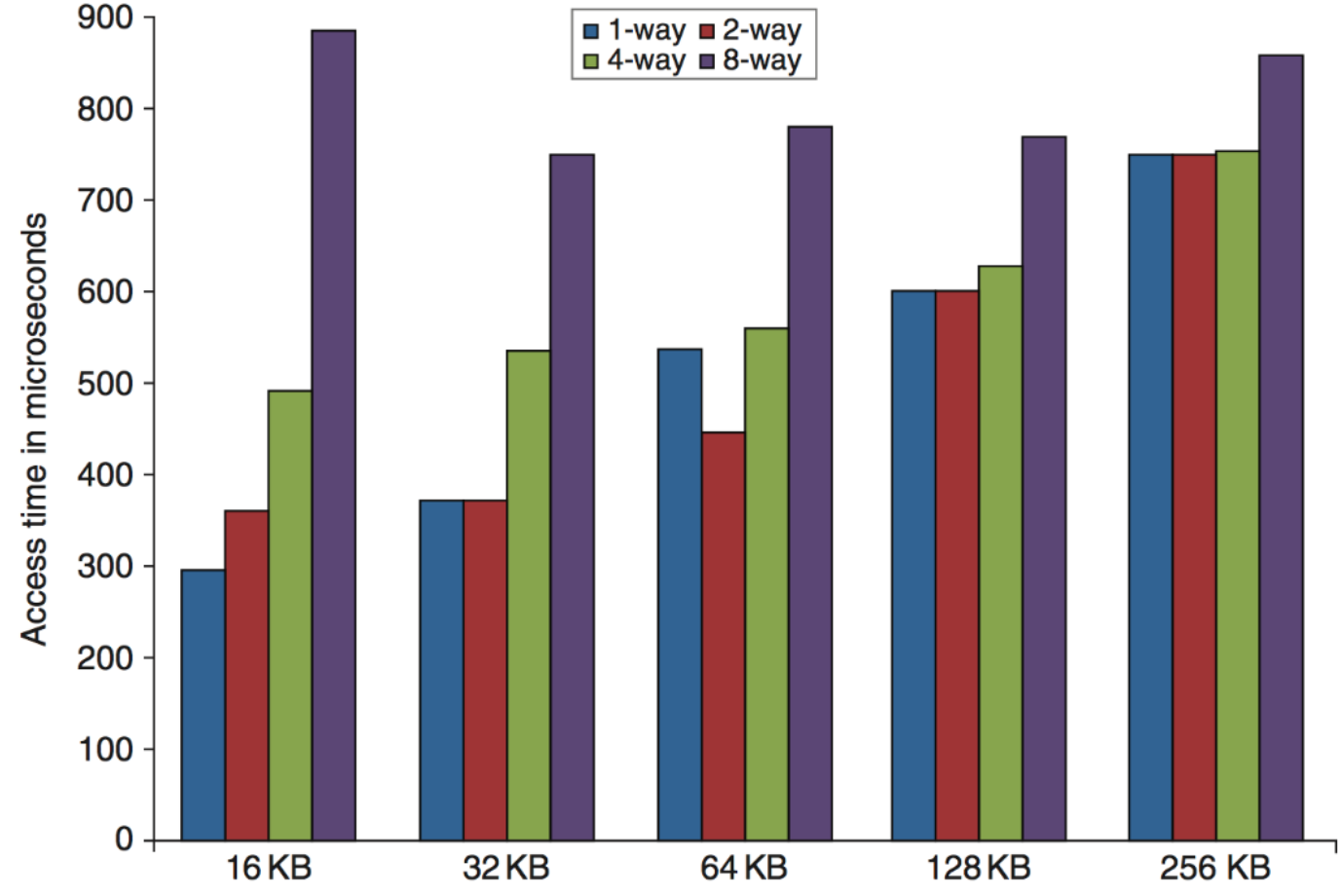
- **Reduce the hit time:** Small and simple first-level caches and way- prediction.
- **Increase cache bandwidth:** Pipelined caches, multibanked caches, and nonblocking caches.
- **Reduce the miss penalty:** Critical word first and merging write buffers
- **Reduce the miss rate:** Compiler optimizations
- **Reduce the miss penalty or miss rate via parallelism:** Hardware prefetching and compiler prefetching

1. Small and Simple First-level Caches

- Smaller hardware is faster, small data cache and thus fast clock rate
  - L1 cache is smaller or increased slightly
  - L2 cache is also small enough to fit on the chip with the processor (reduced off chip penalty)
- Simpler hardware is faster
  - Direct-mapped caches reduce ht time due to concurrent tag checks and data transmission
  - Lower level of associativity involve fewer cache lines hence low power consumption
- Small and simple cache for 1st-level cache
  - Use L2 cache to avoid going to memory
  - Keep tags on chip and data off chip for L2

Cache Size and AMAT

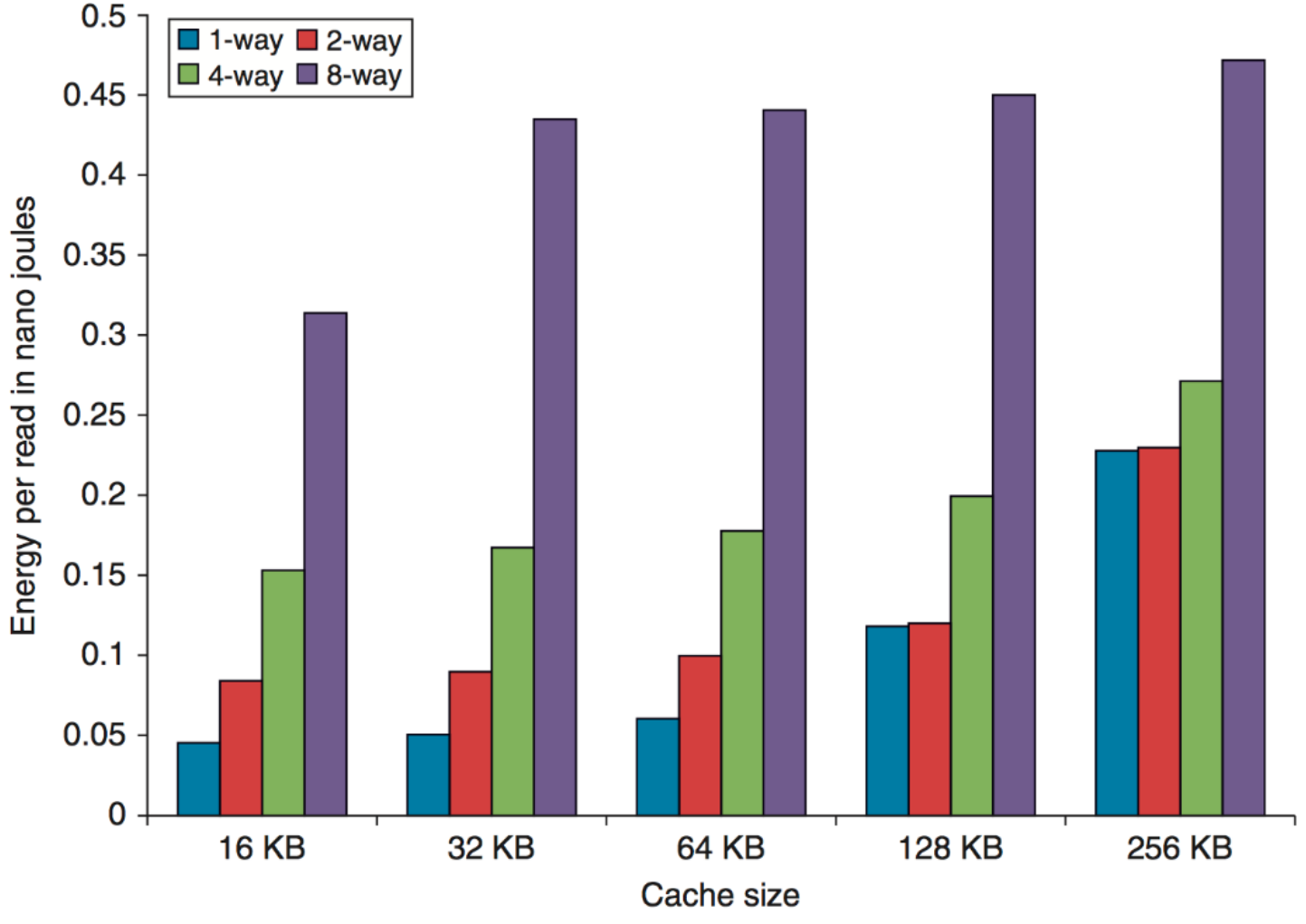
AMAT increases as cache size and associativity are increased



http://www.hpl.hp.com/research/cacti/

Cache Size and Power Consumption

Energy consumption per read increases as cache size and associativity are increased



2. Fast Hit Times Via Way Prediction

- Direct mapped caches has faster hit time but 2-way associative cache has lower conflict misses
- We can get both benefits by predicting a block within a set for the next access
- We keep extra bits in the cache for this optimization
- It has an 85% accuracy but CPU pipeline is harder if hit time is variable length

3. Increasing Cache Bandwidth by Pipelining

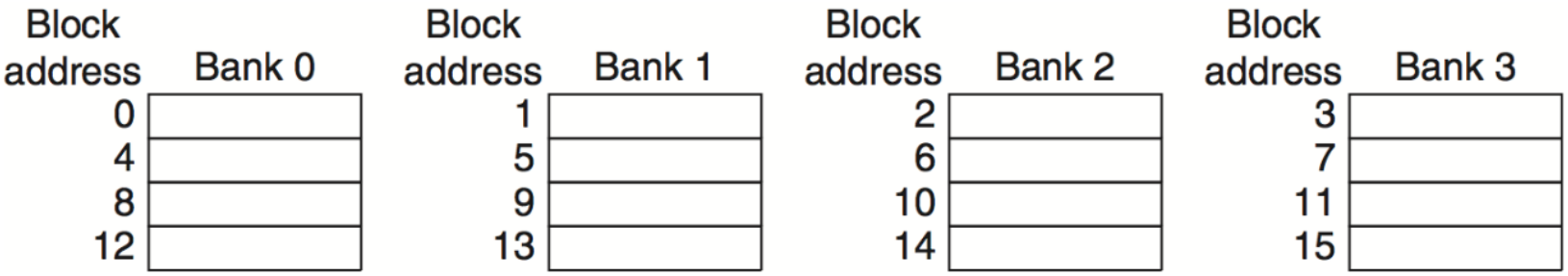
- Pipelining is applied to cache access
- It has fast cycle time and slow hit
- An increased number of pipe line stages leads to more penalty for mispredicted branches
- And more clock cycles between the issue of the load and the use of the data

4. Increasing Cache Bandwidth with Non-Blocking Caches

- Non-blocking or lockup-free cache allows continued cache hits during miss
  - Requires out-of-order execution CPU
- Hit under miss reduces effective miss penalty by working during miss vs. ignoring CPU requests
- Hit under multiple miss or miss under miss further lowers effective miss penalty by overlapping multiple misses
  - Significantly increases complexity of cache controller since can be many outstanding memory accesses
  - Requires multiple memory banks

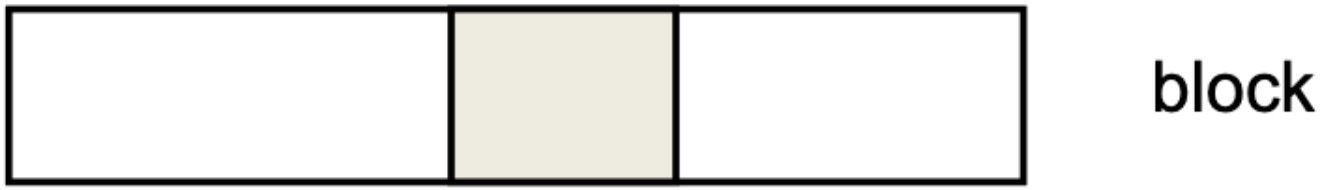
5. Increasing Cache Bandwidth Via Multiple Banks

- Rather than treating cache as single monolithic block, divide into independent banks to support simultaneous accesses
- Works best if access is spread across banks
- Sequential interleaving mapping is best
- Where block addresses are sequentially placed across banks, *i* modulo *n* for each bank



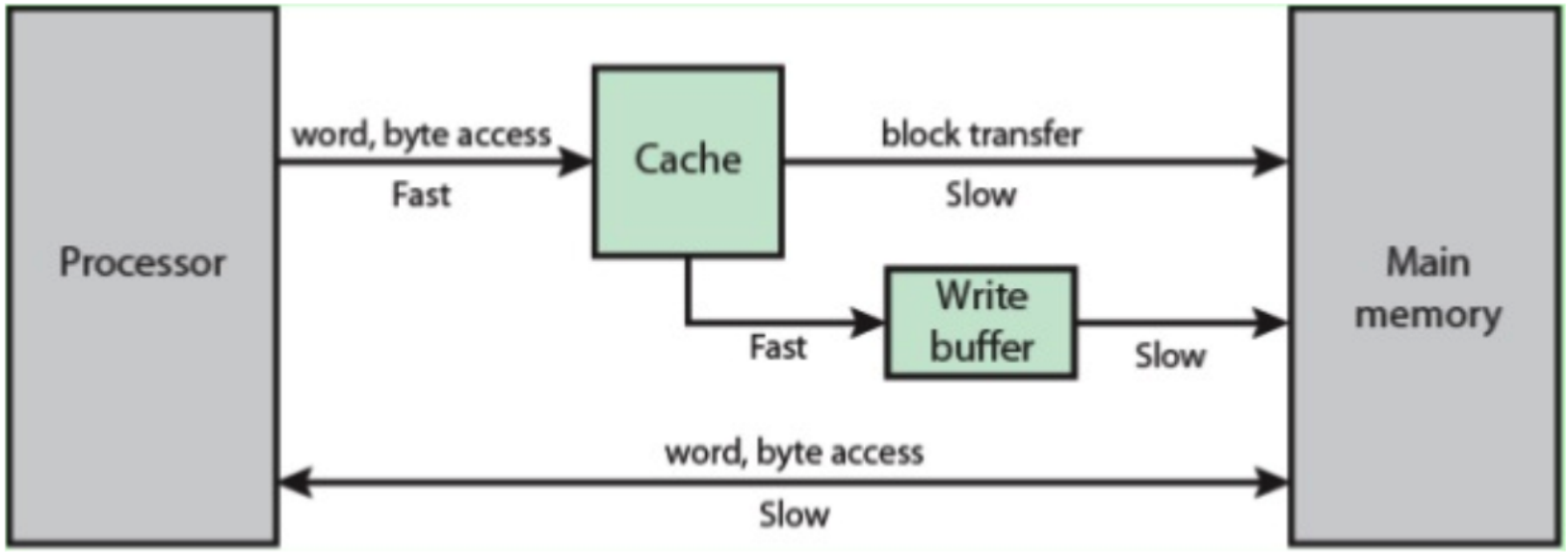
6. Reduce Miss Penalty: Early Restart and Critical Word First

- Don't wait for full block before restarting CPU
- **Critical Word First:** Request missed word from memory first, send it to CPU right away; let CPU continue while filling rest of block
- **Early Restart:** As soon as requested word of block arrives, send to CPU and continue execution



7. Merging Write Buffer to Reduce Miss Penalty

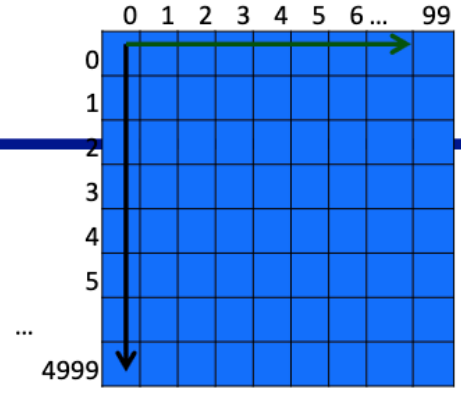
- Write buffer lets processor continue while waiting for write to complete
- Merge write buffer:
  - If buffer contains modified blocks, addresses can be checked to see if new data matches that of some write buffer entry
  - If so, new data combined with that entry



8. Reducing Misses by Compiler Optimizations Software-only Approach

- Instructions:
  - Reorder procedures in memory to reduce conflict misses
  - Profiling to look at conflicts
- Data:
  - Loop interchange: Change nesting of loops to access data in memory
  - Blocking: Improve temporal locality by accessing blocks of data repeatedly vs. going down whole columns or rows
  - Merging arrays: Improve spatial locality by single array of compound elements vs. 2 arrays
  - Loop fusion: Combine 2 independent loops that have same looping and some variable overlap order

Loop Interchange Example




```
/* Before */
for (k = 0; k < 100; k = k+1)
   for (j = 0; j < 100; j = j+1)
    for (i = 0; i < 5000; i = i+1)
        x[i][j] = 2 * x[i][j];
```

Sequence of access:
X[0][0], X[1][0], X[2][0], …

```
/* After */
for (k = 0; k < 100; k = k+1)
   for (i = 0; i < 5000; i = i+1)
    for (j = 0; j < 100; j = j+1)
        x[i][j] = 2 * x[i][j];
```

Sequence of access:
X[0][0], X[0][1], X[1][2], …

Sequential accesses instead of striding through memory every 100 words; improved spatial locality

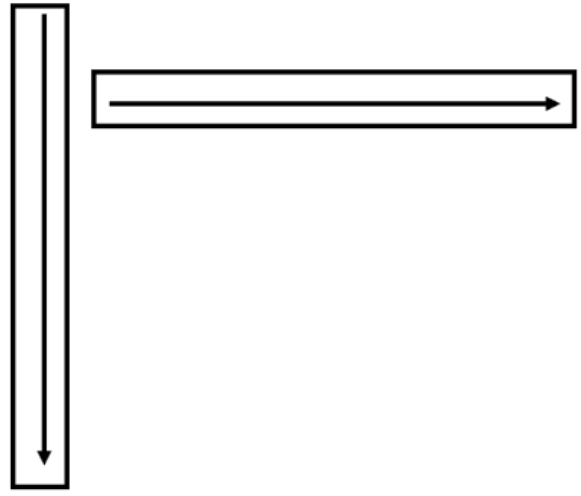
Loop Fusion

```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];

for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];

/* After */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
   {a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];}
```

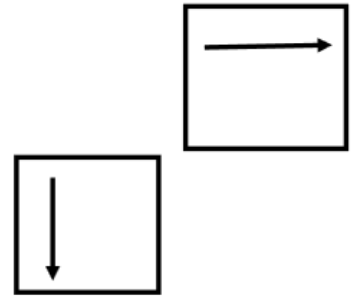
2 misses per access to a & c vs. one miss per access; improve spatial locality

Blocking

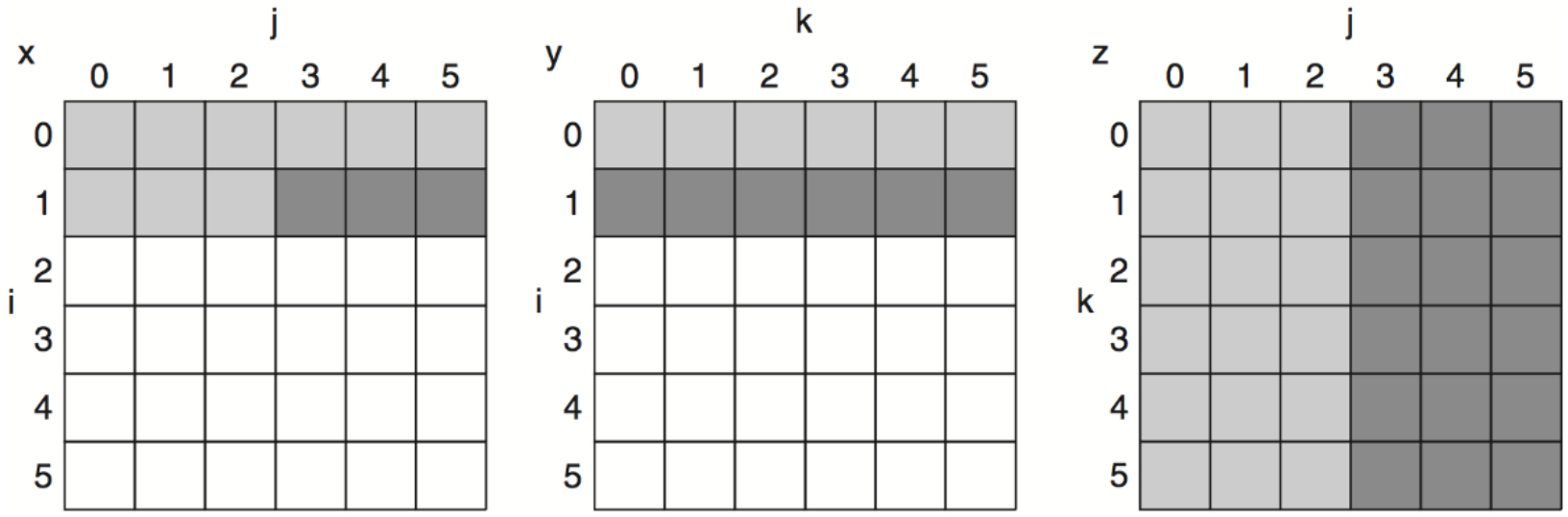
```
/* Before */
for (i = 0; i < N; i = i+1)
   for (j = 0; j < N; j = j+1)
      {r = 0;
       for (k = 0; k < N; k = k+1){
          r = r + y[i][k]*z[k][j];};
       x[i][j] = r;
      };
```



- Two inner loops:
  - Read all NxN elements of z[]
  - Read N elements of 1 row of y[] repeatedly
  - Write N elements of 1 row of x[]
- Capacity misses a function of N & Cache Size:
  - $2N^3 + N^2$ => (assuming no conflict; otherwise …)
- Idea: compute on BxB submatrix that fits



Merging arrays



9. Reducing Misses by Hardware Prefetching of Instructions & Data

- Hardware prefetch items before the processor requests them
  - Both instructions and data can be prefetched
  - Either directly into the caches or into an external buffer that can be more quickly accessed than main memory
  - Can have a negative impact for unused data

Compiler-Controlled Prefetching to Reduce Miss Penalty or Miss Rate

Compiler to insert prefetch instructions to request data before the processor needs it

Resources

- https://www.info425.ece.mcgill.ca/tutorials/T08-Caches.pdf
- https://passlab.github.io/CSCE513/notes/lecture11_CacheAndPerformance.pdf
- https://passlab.github.io/CSCE513/notes/lecture12_CacheOptimizations.pdf