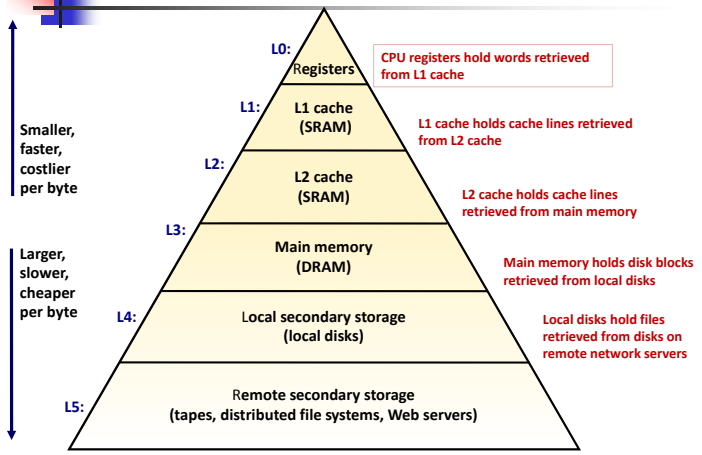
Caching and Performance

Kai Shen

1

An Example Memory Hierarchy



2

Caches (broader interpretation)

- *Cache:* A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work?
  - Because of locality, most accesses are satisfied by cache.
  - So you get the storage space of lower-level storage but enjoy the speed of higher level cache.

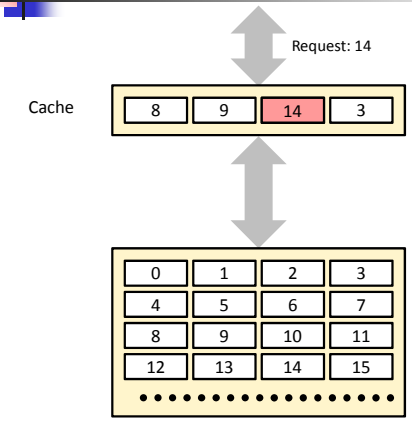
3

What Data to Keep in Caches?

- To achieve the goal that most accesses are satisfied by cache, what data to keep in cache?
- For temporal locality
  - Keep data that is recently accessed
- For spatial locality
  - Keep data that is adjacent (next) to currently accessed data

4

General Cache Concepts: Hit

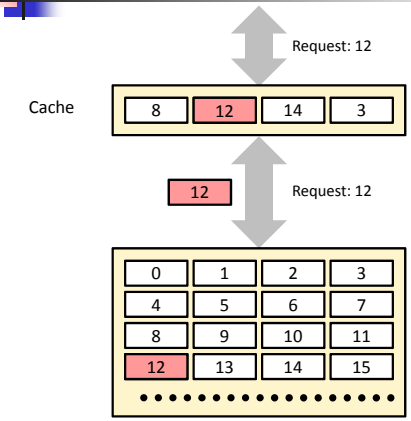


Data in block b is needed

Block b is in cache:
Hit!

5

General Cache Concepts: Miss



Data in block b is needed

Block b is not in cache:
Miss!

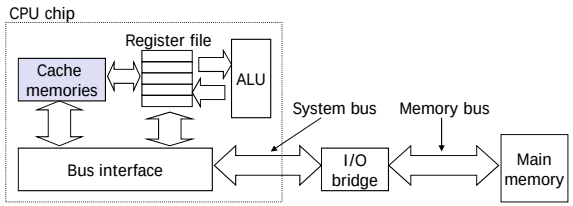
Block b is fetched from
lower-level

Block b is stored in cache
- Replacement policy: determines which block gets evicted (victim)

6

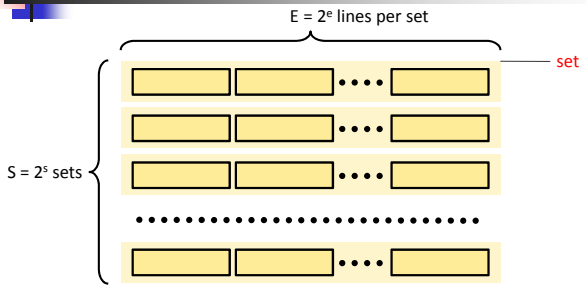
Caches (narrower interpretation)

- Cache memories are small, fast SRAM-based memories managed automatically in hardware.
  - Hold frequently accessed blocks of main memory
- CPU looks first for data in caches (e.g., L1, L2, and L3), then in main memory.
- Typical system structure:



7

Cache Set Association

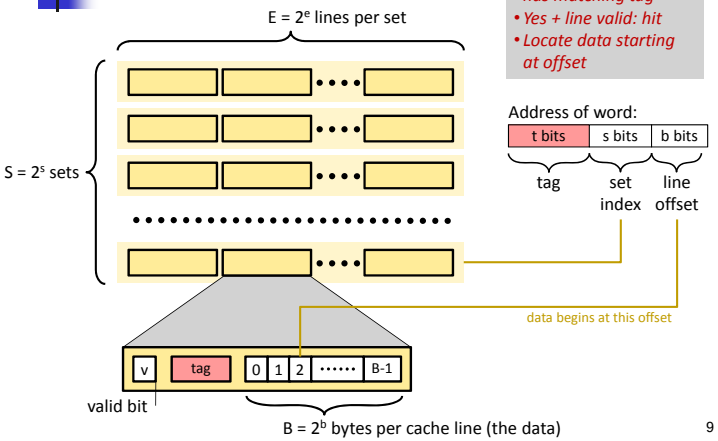


- A cache line is a basic unit of access to memory (e.g., 64 bytes)
- Each set is mapped to a certain section of the memory space (according to particular s bits in a segment of the memory address)
- E-way associative (flexibility of replacement)

8

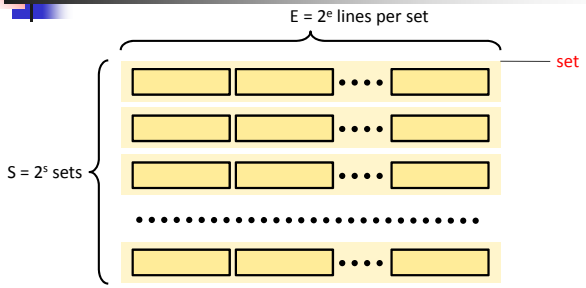
Cache Read

- *Locate set*
- *Check if any line in set has matching tag*
- *Yes + line valid: hit*
- *Locate data starting at offset*



9

Cache Set Association



- Each set is mapped to a certain section of the memory space
- E-way associative
- Special cases
  - Direct-mapped: One line per set.
  - Fully-associative: One set for the whole cache.

10

Cache Helps Programs with Locality

- Cache improves the performance of programs with temporal locality
  - Replacement policy prefers to retain more recently accessed data (limited in direct-mapped cache)
- Cache improves the performance of programs with spatial locality
  - Cacheline-grained memory access (implicit prefetching)
  - Explicit prefetching

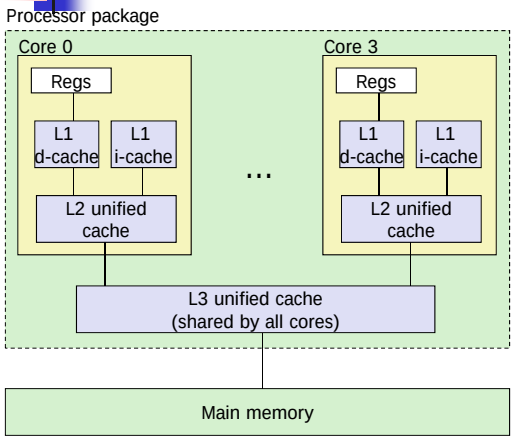
11

What about writes?

- Write-through
  - write immediately to memory
- Write-back
  - write to cache only (most of time)
  - defer write to memory until replacement of line

12

Intel Core i7 Cache Hierarchy



L1 i-cache and d-cache:
32 KB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MB, 16-way,
Access: 30-40 cycles

Cache line size: 64 bytes for all caches.

13

Cache Performance

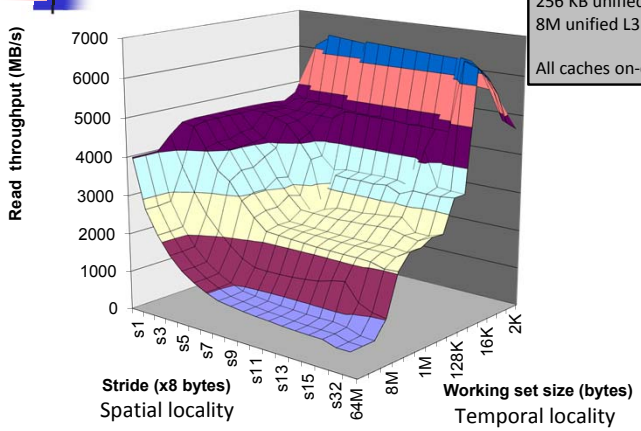
- Huge difference between a hit and a miss
  - Could be 100x, if just L1 and main memory
- Would you believe 99% hits (fraction of memory accesses in cache) is twice as good as 97%?
  - Consider:
    cache hit time of 1 cycle
    miss penalty of 100 cycles
  - Average access time:
    97% hits: 1 cycle + 0.03 * 100 cycles = **4 cycles**
    99% hits: 1 cycle + 0.01 * 100 cycles = **2 cycles**
- "Miss rate" is more illustrative than "hit rate"

14

The Memory Mountain

Intel Core i7
32 KB L1 i-cache
32 KB L1 d-cache
256 KB unified L2 cache
8M unified L3 cache

All caches on-chip

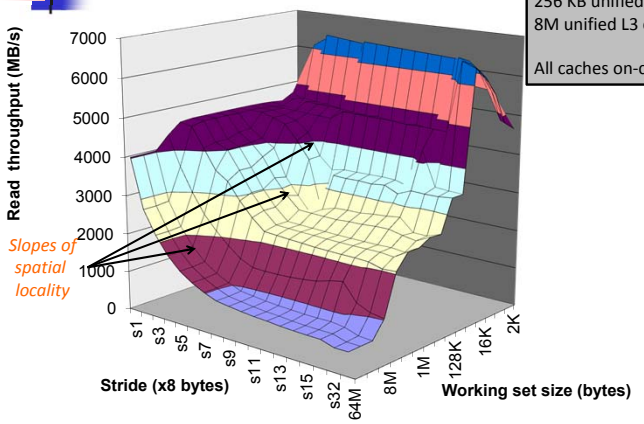


15

The Memory Mountain

Intel Core i7
32 KB L1 i-cache
32 KB L1 d-cache
256 KB unified L2 cache
8M unified L3 cache

All caches on-chip

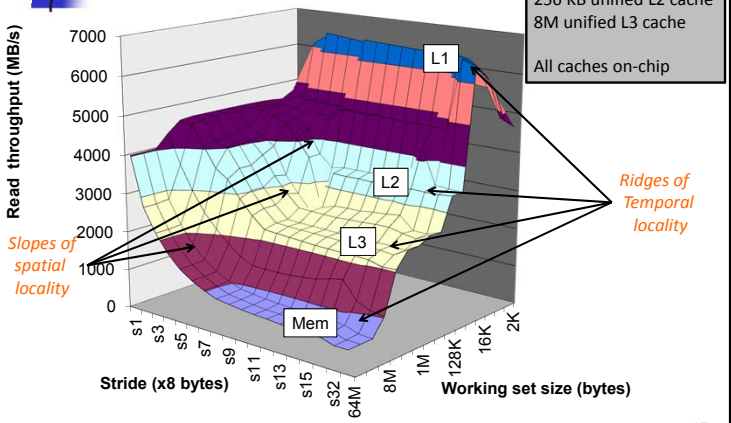


16

The Memory Mountain

Intel Core i7
32 KB L1 i-cache
32 KB L1 d-cache
256 KB unified L2 cache
8M unified L3 cache

All caches on-chip



17

Writing Cache Friendly Code

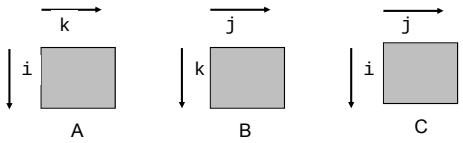
- Make the common case go fast
  - Focus on the (inner) loops of the core functions
- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Sequential reference patterns are good (spatial locality)
- Case studies
  - Rearrange loops to improve spatial locality
  - Use blocking to improve temporal locality

18

Matrix Multiplication Performance

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

Variable sum held in register



- Multiply n x n matrices
  - $O(n^3)$ total operations
  - $3n^3$ data accesses
- Cache misses dominate the performance
  - Some data accesses go to registers
  - Some memory accesses hit in cache

19

Layout of C Arrays in Memory (review)

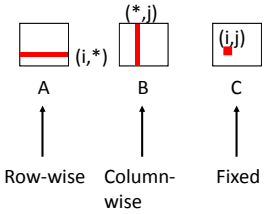
- For C arrays, each inner vector in contiguous memory locations
- Stepping through elements in one inner vector:
  - `for (i = 0; i < n; i++)`
    `sum += a[0][i];`
  - accesses successive elements
  - recall that we load a cacheline at a time
  - if cache line size (64) > 8 bytes, exploit spatial locality
    - miss rate = 8 / 64
- Stepping through the outer index:
  - `for (i = 0; i < n; i++)`
    `sum += a[i][0];`
  - accesses distant elements
  - no spatial locality!
    - miss rate = 1.0 (i.e. 100%)

20

Matrix Multiplication (ijk)

```
/* ijk */
for (i=0; i<n; i++)  {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

Inner loop:



Assume matrix dimension (n) is so large that a single row can't fit into cache.

Misses per inner loop iteration:

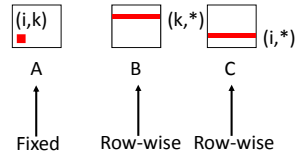
A	B	C
0.125	1.0	0.0

Other implementations?

Matrix Multiplication (kij)

```
/* kij */
for (k=0; k<n; k++) {
  for (i=0; i<n; i++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

Inner loop:



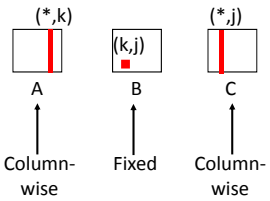
Misses per inner loop iteration:

A	B	C
0.0	0.125	0.125

Matrix Multiplication (jki)

```
/* jki */
for (j=0; j<n; j++) {
  for (k=0; k<n; k++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```

Inner loop:



Misses per inner loop iteration:

A	B	C
1.0	0.0	1.0

Summary of Matrix Multiplication

```
for (i=0; i<n; i++) {
  for (j=0; j<n; j++) {
    sum = 0.0;
    for (k=0; k<n; k++)
      sum += a[i][k] * b[k][j];
    c[i][j] = sum;
  }
}
```

ijk (& jik):
- misses/iter = 1.125

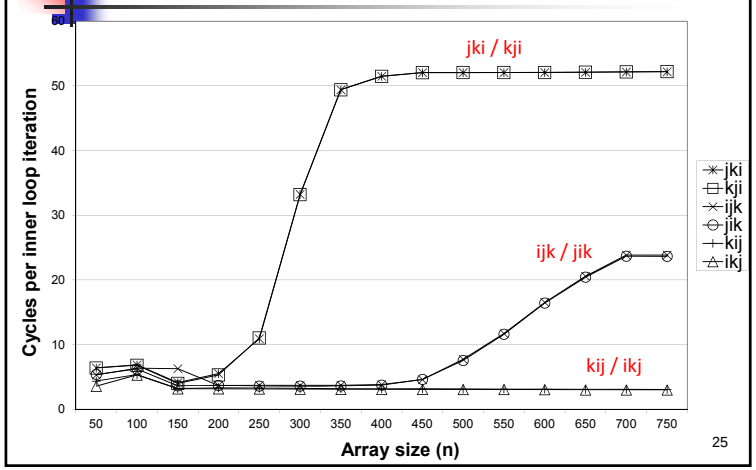
```
for (k=0; k<n; k++) {
  for (i=0; i<n; i++) {
    r = a[i][k];
    for (j=0; j<n; j++)
      c[i][j] += r * b[k][j];
  }
}
```

kij (& ikj):
- misses/iter = 0.25

```
for (j=0; j<n; j++) {
  for (k=0; k<n; k++) {
    r = b[k][j];
    for (i=0; i<n; i++)
      c[i][j] += a[i][k] * r;
  }
}
```

jki (& kji):
- misses/iter = 2.0

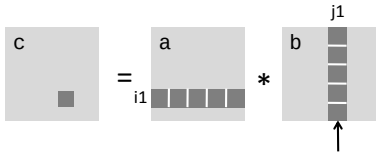
Core i7 Matrix Multiply Performance



25

Blocked Matrix Multiplication

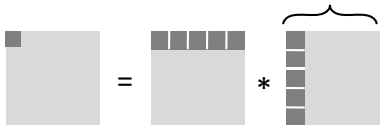
```
c = (double *) calloc(sizeof(double), n*n);

/* Multiply n x n matrices a and b  */
void mmm(double *a, double *b, double *c, int n) {
    int i, j, k;
    for (i = 0; i < n; i+=B)
        for (j = 0; j < n; j+=B)
            for (k = 0; k < n; k+=B)
                /* B x B mini matrix multiplications */
                for (i1 = i; i1 < i+B; i++)
                    for (j1 = j; j1 < j+B; j++)
                        for (k1 = k; k1 < k+B; k++)
                            c[i1*n+j1] += a[i1*n + k1]*b[k1*n + j1];
}
```



26

Cache Miss Analysis

- Assume:
  - Cache line = 8 doubles (elements)
  - Three blocks ■ fit into cache
  - B is the block size in number of elements
- First (block) iteration:
  - $B^2/8$ misses for each block
  - $2n/B * B^2/8 = nB/4$
    (omitting matrix C)
- Total misses
  - $nB/4 * (n/B)^2 = n^3/(4B)$

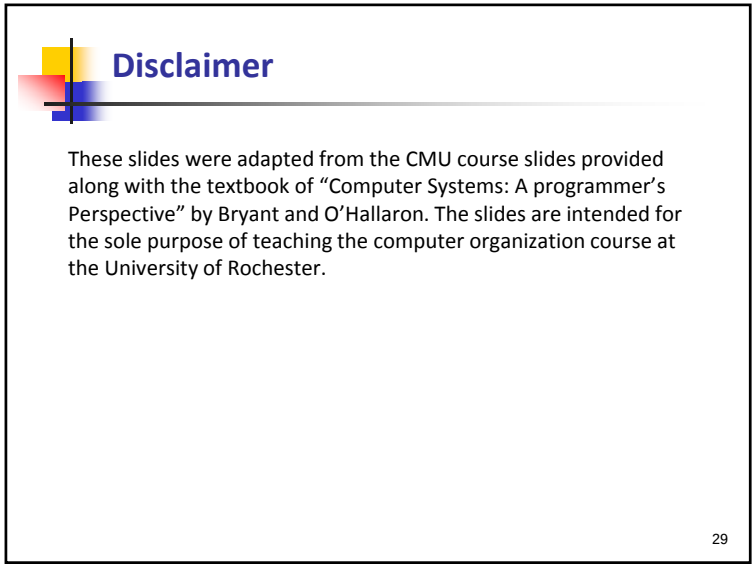


27

Summary

- Blocking: $n^3/(4B)$
- No blocking: $n^3 * 0.25$
- Suggest largest possible block size B, but limit it so three blocks fit into cache
  - I once used block size B=25 elements (doubles), what was the cache size?
- Foundation for performance enhancement:
  - Matrix multiplication has inherent temporal locality:
    - Input data: $3n^2$, computation $2n^3$
    - Every array elements used O(n) times!
  - But program has to be written properly

28


Disclaimer
These slides were adapted from the CMU course slides provided along with the textbook of “Computer Systems: A programmer’s Perspective” by Bryant and O’Hallaron. The slides are intended for the sole purpose of teaching the computer organization course at the University of Rochester.
29