

Advanced Systems Lab

Spring 2022

Lecture: Memory hierarchy, locality, caches

Instructor: Markus Püschel, Ce Zhang

TA: Joao Rivera, several more

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Organization

Temporal and spatial locality

Memory hierarchy

Caches

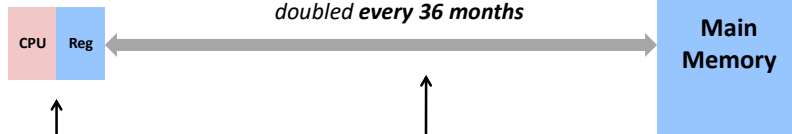
*Chapter 5 in Computer Systems: A Programmer's Perspective, 2nd edition,
Randal E. Bryant and David R. O'Hallaron, Addison Wesley 2010*

Part of these slides are adapted from the course associated with this book

Problem: Processor-Memory Bottleneck

Processor performance
doubled about
every 18 months

Bus bandwidth
doubled every 36 months



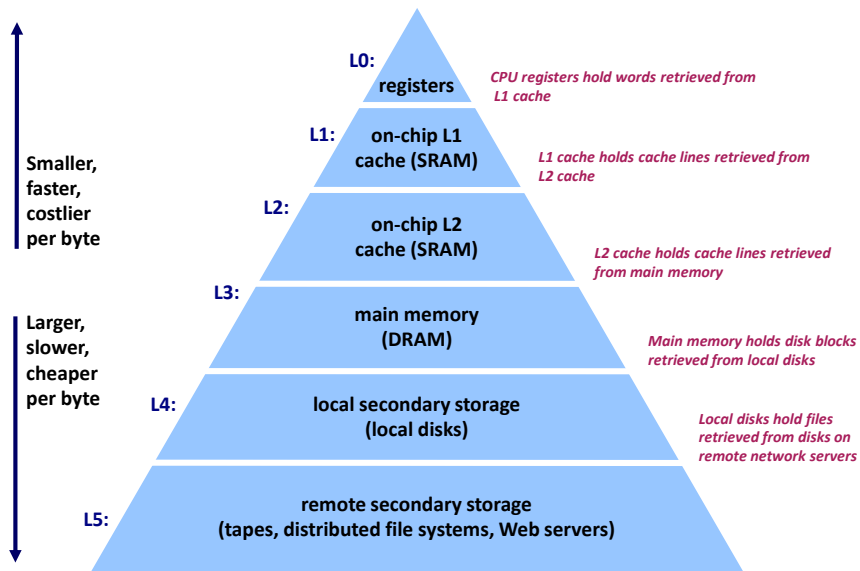
Core i7 Skylake:
Peak performance:
2 AVX three operand (FMA) ops/cycles
consumes up to 192 Bytes/cycle

Core i7 Skylake:
Bandwidth
16 Bytes/cycle

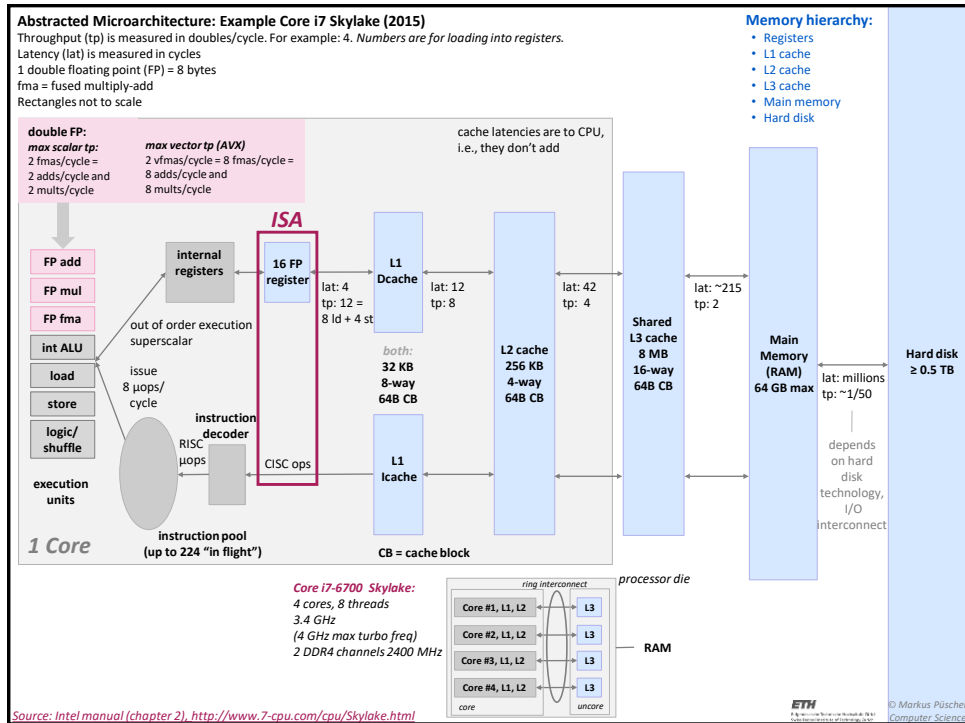
Solution: Caches/Memory hierarchy

3

Typical Memory Hierarchy



4



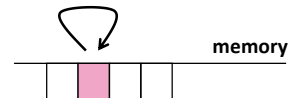
Why Caches Work: Locality

Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

History of locality

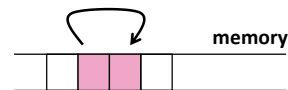
Temporal locality:

Recently referenced items are likely to be referenced again in the near future



Spatial locality:

Items with nearby addresses tend to be referenced close together in time



Example: Locality?

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

Data:

- *Temporal: **sum** referenced in each iteration*
- *Spatial: array **a[]** accessed consecutively*

Instructions:

- *Temporal: loops cycle through the same instructions*
- *Spatial: instructions referenced in sequence*

Being able to assess the locality of code is a crucial skill for a performance programmer

7

Locality Example #1

```
int sum_array_rows(double a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

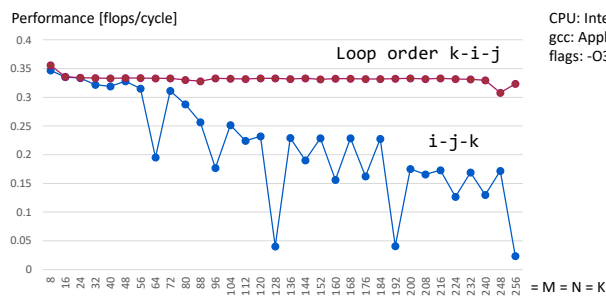
8

Locality Example #2

```
int sum_array_3d(double a[K][M][N])
{
    int i, j, k, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            for (k = 0; k < K; k++)
                sum += a[k][i][j];
    return sum;
}
```

How to improve locality?



CPU: Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz
gcc: Apple LLVM version 8.0.0 (clang-800.0.42.1)
flags: -O3 -fno-vectorize

9

Operational Intensity Again

Definition: Given a program P, assume cold (empty) cache

$$\text{Operational intensity: } I(n) = \frac{W(n)}{Q(n)}$$

#flops (input size n) → W(n)

#bytes transferred cache ↔ memory (for input size n) → Q(n)

Examples: Determine asymptotic bounds on I(n)

- Vector sum: $y = x + y$ $O(1)$
- Matrix-vector product: $y = Ax$ $O(1)$
- Fast Fourier transform $O(\log(n))$
- Matrix-matrix product: $C = AB + C$ $O(n)$

10

Compute/Memory Bound

A function/piece of code is:

- **Compute bound** if it has high operational intensity
- **Memory bound** if it has low operational intensity

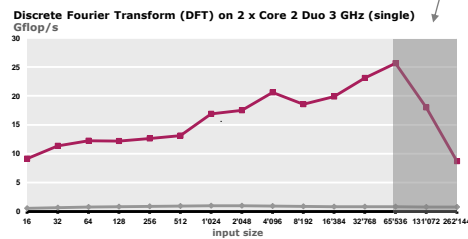
Relationship between operational intensity and locality?

- They are closely related
- Operational intensity only describes the boundary last level cache/memory

11

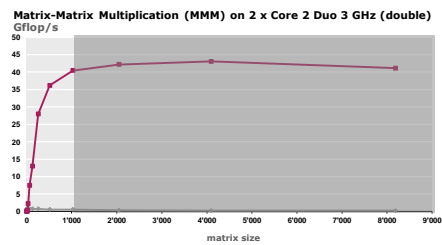
Effects

FFT: $I(n) = O(\log(n))$



Up to 40-50% peak
Performance drop outside last level cache (LLC)
Most time spent transferring data

MMM: $I(n) = O(n)$



Up to 80-90% peak
Performance can be maintained outside LLC
Cache miss time compensated/hidden by computation

12

Cache

Definition: Computer memory with short access time used for the storage of frequently or recently used instructions or data



Naturally supports *temporal locality*

Spatial locality is supported by transferring data in blocks

- Core family: one block = 64 B = 8 doubles

13

Types of Cache Misses (The 3 C's)

Compulsory (cold) miss

Occurs on first access to a block

Capacity miss

Occurs when working set is larger than the cache

Conflict miss

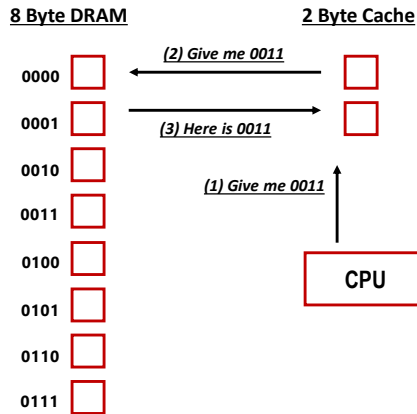
Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot

Not a clean classification but still useful

14

Direct Mapped Cache

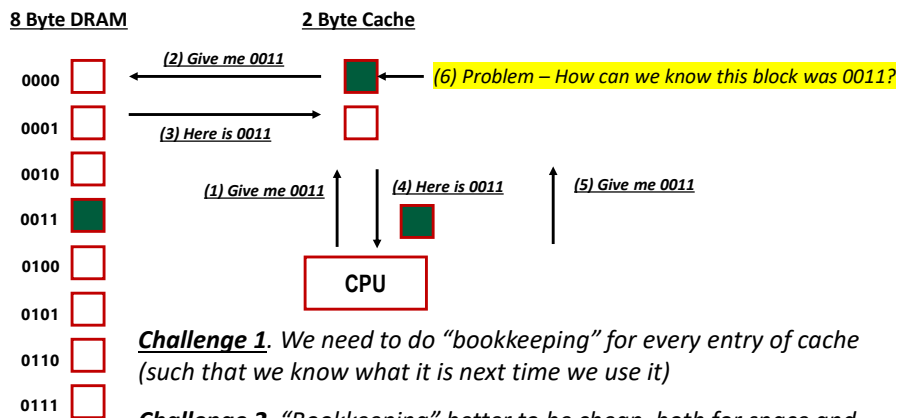
How would you build a Cache?



15

Direct Mapped Cache

How would you build a Cache?



Challenge 1. We need to do “bookkeeping” for every entry of cache (such that we know what it is next time we use it)

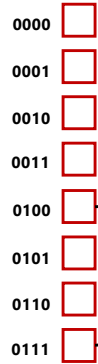
Challenge 2. “Bookkeeping” better to be cheap, both for space and computational efficiency (e.g., in the above example, having a 4-bit address along with every 1 Byte data is probably a very bad idea)

16

Direct Mapped Cache

How would you build a Cache?

8 Byte DRAM

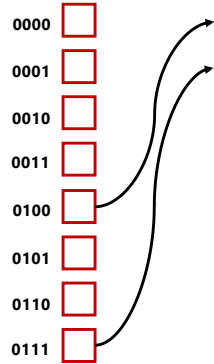


Design 1: Cache



Problem 1: not very efficient – 4 bits / 1 byte

Problem 2: no spatial locality

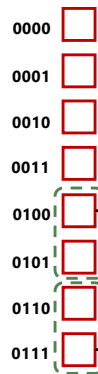


17

Direct Mapped Cache

How would you build a Cache?

8 Byte DRAM



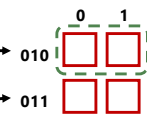
Design 1: Cache



Problem 1: not very efficient – 4 bits / 1 byte

Problem 2: no spatial locality

Design 2: Cache



More efficient – 3 bits / 2 byte

Better spatial locality

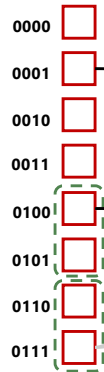
Problem 3: How to find out whether e.g., 0010 is in cache? – Scan all entries!

18

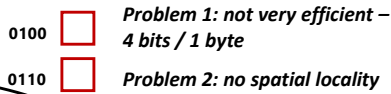
Direct Mapped Cache

How would you build a Cache?

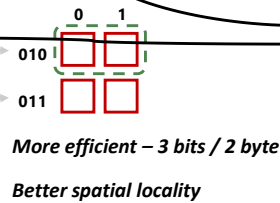
8 Byte DRAM



Design 1: Cache

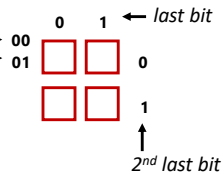


Design 2: Cache



Problem 3: How to find out whether e.g., 0010 is in cache?
- Scan all entries!

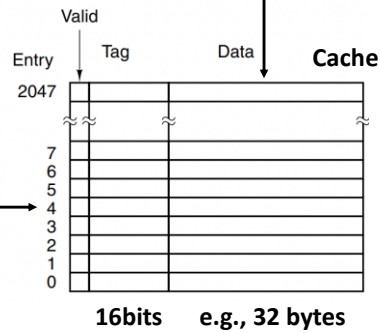
Design 3: Cache



19

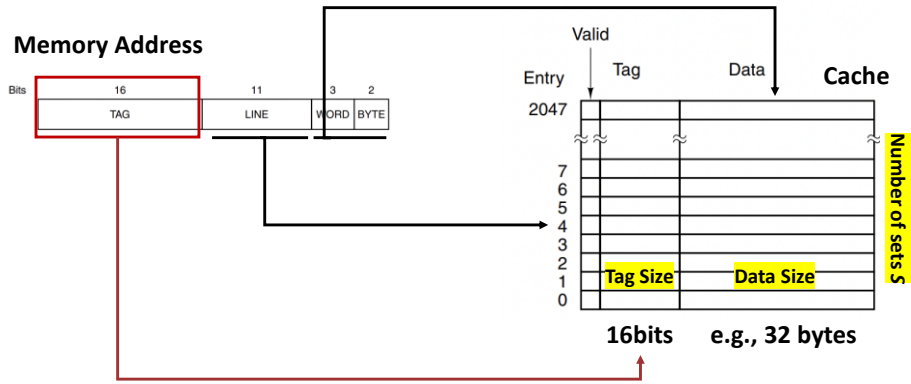
Direct Mapped Cache

Memory Address (e.g., 32 bits)



20

Direct Mapped Cache

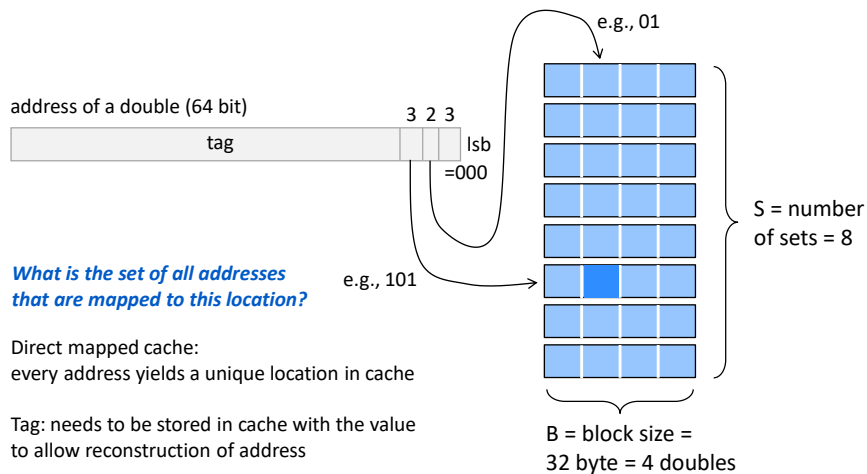


32 bit address \rightarrow Tag Size (# bits) stored in TAG
 + $\log_2 S$ bits encodes as a "row" in the cache
 + $\log_2 Data$ bits encodes position in the data block

21

Cache Structure

Example 1: direct mapped cache ($E = 1$, $B = 4$ doubles, $S = 8$)



What is the set of all addresses that are mapped to this location?

Direct mapped cache:
 every address yields a unique location in cache

Tag: needs to be stored in cache with the value to allow reconstruction of address

Always entire blocks (here 32 bytes) are loaded into cache

22

Example (S=8, E=1)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

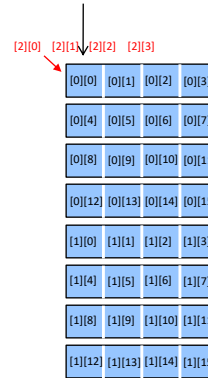
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables *sum, i, j*

assume: cold (empty) cache,
a[0][0] goes here



B = 32 byte = 4 doubles

How is the cache filled?

23

Example (S=8, E=1)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

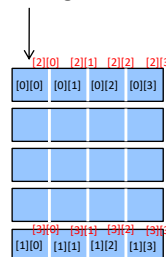
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables *sum, i, j*

assume: cold (empty) cache,
a[0][0] goes here

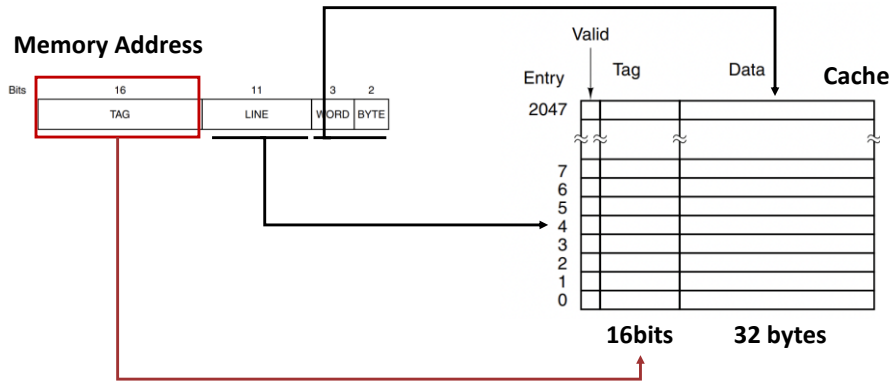


B = 32 byte = 4 doubles

How is the cache filled?

24

Direct Mapped Cache



How big is this cache? – 64K

How to make it bigger?

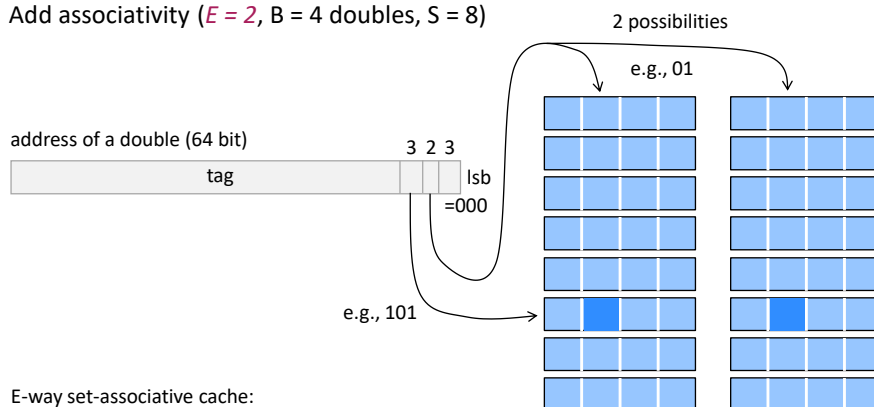
(1) Bigger data block – Yes, but this cannot continue forever

(1) Add more of these “building blocks”

25

Cache Structure: E-way set-associative cache

Add associativity ($E = 2$, $B = 4$ doubles, $S = 8$)



E-way set-associative cache:
every value has E possible locations

Usually, least recently used (LRU) is replaced

Always entire blocks (here 32 bytes) are loaded into cache

26

Example (S=4, E=2)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

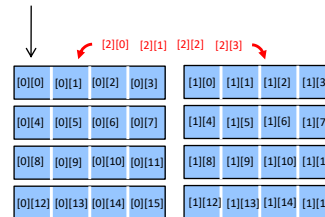
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables *sum, i, j*

assume: cold (empty) cache,
a[0][0] goes here



How is the cache filled?

27

Example (S=4, E=2)

```
int sum_array_rows(double a[16][16])
{
    int i, j;
    double sum = 0;

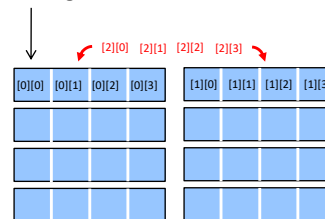
    for (i = 0; i < 16; i++)
        for (j = 0; j < 16; j++)
            sum += a[i][j];
    return sum;
}
```

```
int sum_array_cols(double a[16][16])
{
    int i, j;
    double sum = 0;

    for (j = 0; j < 16; j++)
        for (i = 0; i < 16; i++)
            sum += a[i][j];
    return sum;
}
```

Ignore the variables *sum, i, j*

assume: cold (empty) cache,
a[0][0] goes here

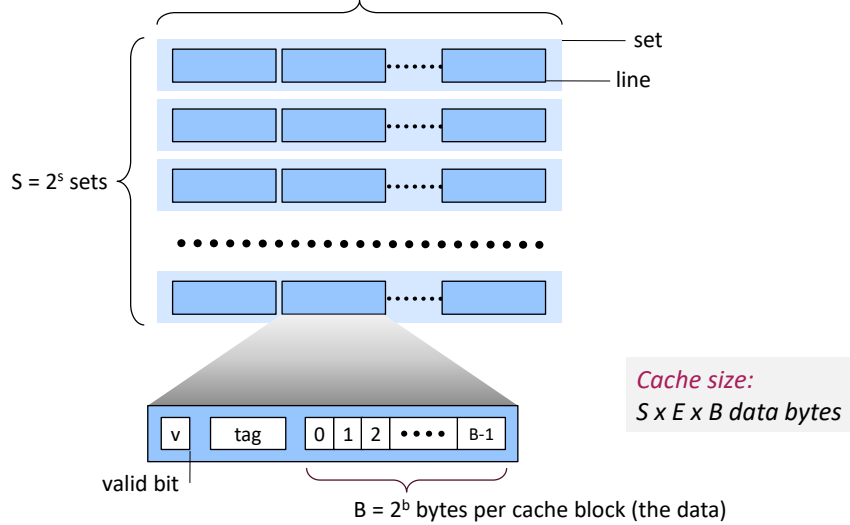


How is the cache filled?

28

General Cache Organization (S, E, B)

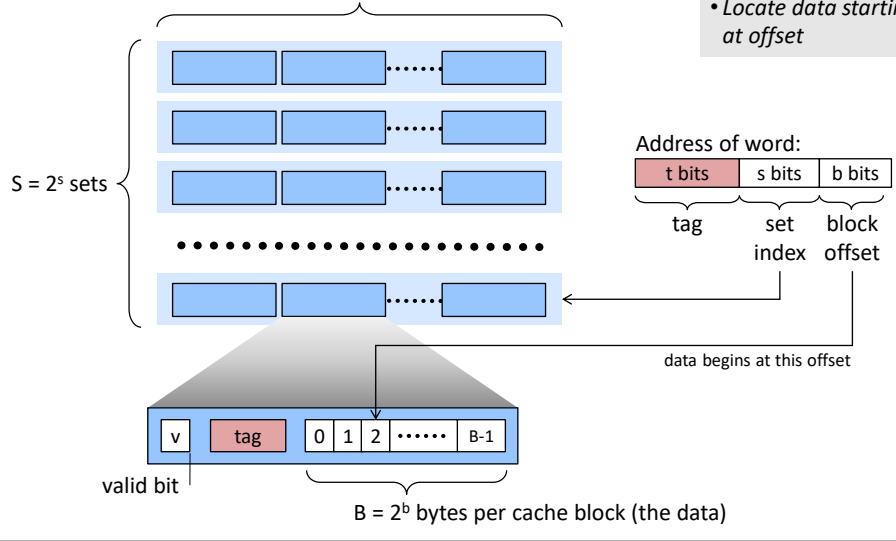
$E = 2^e$ lines per set
 $E =$ associativity, $E=1$: direct mapped



Cache Read

$E = 2^e$ lines per set
 $E =$ associativity, $E=1$: direct mapped

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



Terminology

Direct mapped cache:

- Cache with $E = 1$
- Means every block from memory has a unique location in cache

Fully associative cache

- Cache with $S = 1$ (i.e., maximal E)
- Means every block from memory can be mapped to any location in cache
- In practice too expensive to build
- One can view the register file as a fully associative cache

LRU (least recently used) replacement

- when selecting which block should be replaced (happens only for $E > 1$), the least recently used one is chosen

31

Small Example, Part 1

$x[0]$



Cache:

$E = 1$ (direct mapped)
 $S = 2$
 $B = 16$ (2 doubles)

Array (accessed twice in example)

$x = x[0], \dots, x[7]$

```
% Matlab style code
for j = 0:1
  for i = 0:7
    access(x[i])
```

Access pattern: 0123456701234567
Hit/Miss: MHMHHMHHMHHMHHM

Result: 8 misses, 8 hits

Spatial locality: yes

Temporal locality: no

32

Small Example, Part 2

x[0]



Cache:

E = 1 (direct mapped)

S = 2

B = 16 (2 doubles)

Array (accessed twice in example)

x = x[0], ..., x[7]

% Matlab style code

```
for j = 0:1
  for i = 0:2:7
    access(x[i])
  for i = 1:2:7
    access(x[i])
```

Access pattern:

0246135702461357

Hit/Miss:

MMMMMMMMMMMMMMMM

Result: 16 misses

Spatial locality: no

Temporal locality: no

33

Small Example, Part 3

x[0]



Cache:

E = 1 (direct mapped)

S = 2

B = 16 (2 doubles)

Array (accessed twice in example)

x = x[0], ..., x[7]

% Matlab style code

```
for j = 0:1
  for k = 0:1
    for i = 0:3
      access(x[i+4j])
```

Access pattern:

0123012345674567

Hit/Miss:

MHMMMMMMMMMMMM

Result: 4 misses, 12 hits (is optimal, why?)

Spatial locality: yes

Temporal locality: yes

34

Cache Performance Metrics

Miss rate

- Fraction of memory references not found in cache: $\text{misses} / \text{accesses}$
 $= 1 - \text{hit rate}$

Hit time

- Time to deliver a block in the cache to the processor
- Haswell:
 4 clock cycles for L1
 11 clock cycles for L2

Miss penalty

- Additional time required because of a miss
- Haswell: about 100 cycles for L3 miss

What about writes?

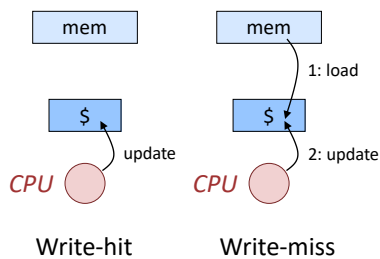
What to do on a write-hit?

- **Write-through:** write immediately to memory
- **Write-back:** defer write to memory until replacement of line

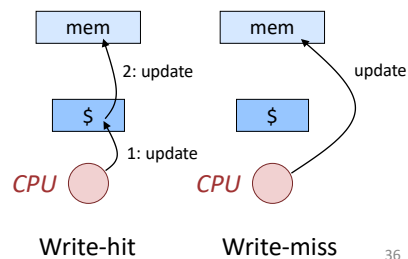
What to do on a write-miss?

- **Write-allocate:** load into cache, update line in cache
- **No-write-allocate:** writes immediately to memory

Write-back/write-allocate (Core)



Write-through/no-write-allocate



Example:

$z = x + y$, x, y, z vector of doubles of length n

assume they fit jointly in cache + cold cache

memory traffic $Q(n)$: $4n$ doubles = $32n$ bytes

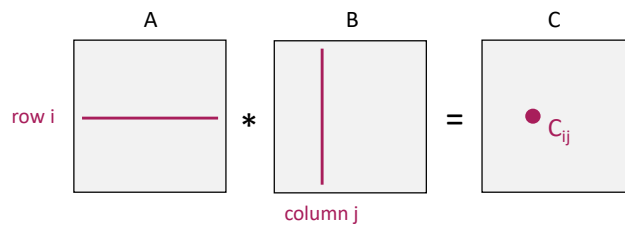
operational intensity $I(n)$? $W(n) = n$ flops, so
 $I(n) = W(n)/Q(n) = 1/32$

37

Locality Optimization: Blocking

Example: MMM

```
void mmm(double *A, double *B, double *C, int n) {  
    for( int i = 0; i < n; i++ )  
        for( int j = 0; j < n; j++ )  
            for( int k = 0; k < n; k++ )  
                C[n*i + j] = C[n*i + j] + A[n*i + k] * B[n*k + j]; }  
}
```



38

Cache Miss Analysis MMM

$$C = A * B, \text{ all } n \times n$$

Assumptions: cache size $\gamma \ll n$, cache block: 8 doubles, only 1 cache

Triple loop:

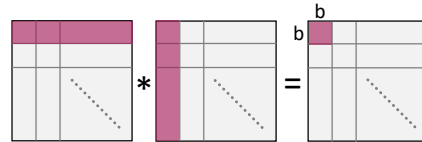


1. entry: $n/8 + n = 9n/8$ cache misses

2. entry: same

Total: $n^2 * 9n/8 = 9n^3/8$

Blocked (six-fold loop): block size b , 8 divides b



1. block: $nb/8 + nb/8 = nb/4$ cache misses

2. block: same

Total: $(n/b)^2 * nb/4 = n^3/(4b)$

How to choose b ?

The above analysis assumes that the multiplication of $b \times b$ blocks can be done with only compulsory misses. This requires $3b^2 \leq \gamma$.

$b = \sqrt{\gamma/3}$ which yields about $\sqrt{3}/(4 * \sqrt{\gamma}) * n^3$ cache misses, a gain of $\approx 2.6 * \sqrt{\gamma}$
 $I(n) = O(\sqrt{\gamma})$

39

Experiment

Cascade Lake (Intel® Xeon® Silver 4210)

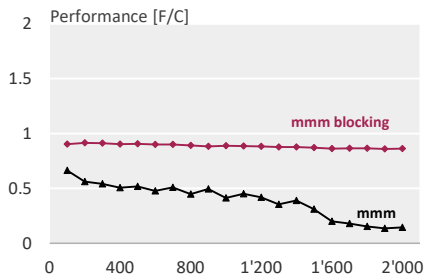
GCC 9.3.0

Flags: -O3 -ffast-math [-fno-tree-vectorize] -march=native

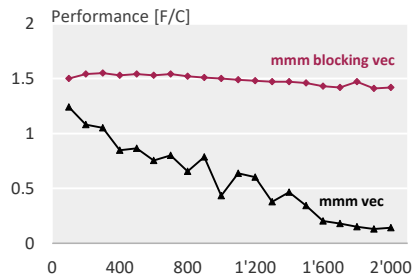
L1 cache: 4096 doubles

Block size $b = 32$

Vectorization disabled



Vectorization enabled



40

On Previous Slide

Refine the analysis by including the misses incurred by C

Compute the operational intensity in both cases

Try an analogous analysis for matrix-vector multiplication

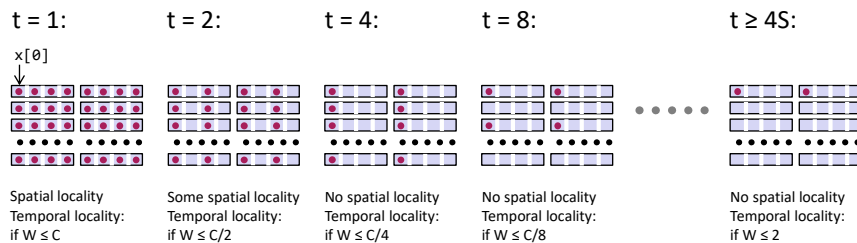
41

The Killer: Two-Power Strided Working Sets

```

% t = 1,2,4,8,... a 2-power
% size W of working set: W = n/t
for (i = 0; i < n; i += t)
  access(x[i])
for (i = 0; i < n; i += t)
  access(x[i])
  
```

Cache: E = 2, B = 4 doubles



Working with a two-power-strided working set is like having a smaller cache

42

The Killer: Where Can It Occur?

Accessing two-power size 2D arrays (e.g., images) columnwise

- *2d Transforms*
- *Stencil computations*
- *Correlations*

Various transform algorithms

- *Fast Fourier transform*
- *Wavelet transforms*
- *Filter banks*

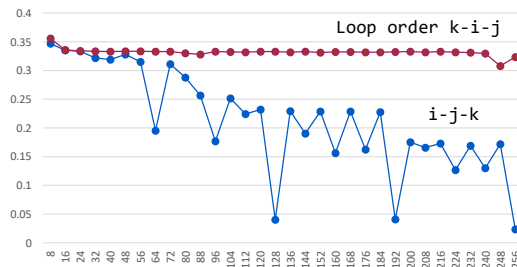
43

Example from Before

```
int sum_array_3d(double a[K][M][N])
{
  int i, j, k, sum = 0;

  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < K; k++)
        sum += a[k][i][j];
  return sum;
}
```

Performance [flops/cycle]



CPU: Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz
 gcc: Apple LLVM version 8.0.0 (clang-800.0.42.1)
 flags: -O3 -fno-vectorize

← 2-power strides

M = N = K

44

Summary

It is important to assess temporal and spatial locality in the code

Cache structure is determined by three parameters

- *block size*
- *number of sets*
- *associativity*

You should be able to roughly simulate a computation on paper

Blocking to improve locality

Two-power strides can be problematic (conflict misses)