Memory Hierarchy (IV): Programming Techniques to Cache Performance & Basic Pipelined Processor Design

Hung-Wei Tseng
Performance gap between Processor/Memory
Which of the following schemes can help Athlon 64?

• How many of the following schemes mentioned in “improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” would help AMD Phenom II for the code in the previous slide?

A. 0
B. 1
C. 2
D. 3
E. 4

- Missing cache  — help improving conflict misses
- Victim cache  — help improving conflict misses
- Prefetch  — can potentially hurt
- Stream buffer — only help improving compulsory misses
Multibanks & non-blocking caches
Early Restart and Critical Word First

- Assume the bus between L1/L2 only allows a quarter of the cache block go through it.

If the requesting data (offset within a block is already received), restart.
Write buffer!

- If the requesting data (offset within a block) is already received, restart.

Assume the bus between L1/L2 only allows a quarter of the cache block to go through it.
Summary of Optimizations

• Hardware
  • Prefetch — compulsory miss
  • Write buffer — miss penalty
  • Bank/pipeline — miss penalty
  • Critical word first and early restart — miss penalty
Consider the following data structure:

```c
struct student {
    int id;
    double *homework;
    int participation;
    double midterm;
    double average;
};
```

What’s the output of `printf("%lu\n", sizeof(struct student));`?

A. 20  
B. 28  
C. 32  
D. 36  
E. 40  

The result of `sizeof(struct student)`
Almost every popular ISA architecture uses “byte-addressing” to access memory locations.

Instructions generally work faster when the given memory address is aligned.

- Aligned — if an instruction accesses an object of size \( n \) at address \( X \), the access is aligned if \( X \mod n = 0 \).

- Some architecture/processor does not support aligned access at all.
- Therefore, compilers only allocate objects on “aligned” address.

Compiler optimization cannot help!
Team scores
Outline

• Programmer’s optimizations for cache performance
• Basic Pipelined Processor Design
Programming and memory performance
## Array of structures or structure of arrays

### Array of objects

```
struct grades {
    int id;
    double *homework;
    double average;
};
```

### Object of arrays

```
struct grades {
    int *id;
    double **homework;
    double *average;
};
```

<table>
<thead>
<tr>
<th>ID</th>
<th>*homework</th>
<th>average</th>
<th>ID</th>
<th>*homework</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### average of each homework

```c
for(i=0;i<homework_items; i++) {
    gradesheet[total_number_students].homework[i] = 0.0;
    for(j=0;j<total_number_students;j++)
    gradesheet[total_number_students].homework[i] += gradesheet[j].homework[i];
    gradesheet[total_number_students].homework[i] /= (double)total_number_students;
}
```
What data structure is performing better

<table>
<thead>
<tr>
<th>Array of objects</th>
<th>object of arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct grades</td>
<td>struct grades</td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>int id;</td>
<td>int *id;</td>
</tr>
<tr>
<td>double *homework;</td>
<td>double **homework;</td>
</tr>
<tr>
<td>double average;</td>
<td>double *average;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

for(i=0;i<homework_items; i++)
{
    gradesheet[total_number_students].homework[i] = 0.0;
    for(j=0;j<total_number_students;j++)
        gradesheet[total_number_students].homework[i] += gradesheet[j].homework[i];
    gradesheet[total_number_students].homework[i] /= (double)total_number_students;
}

for(i = 0; i < homework_items; i++)
{
    gradesheet.homework[i][total_number_students] = 0.0;
    for(j = 0; j < total_number_students; j++)
    {
        gradesheet.homework[i][total_number_students] += gradesheet.homework[i][j];
    }
    gradesheet.homework[i][total_number_students] /= total_number_students;
}

- Considering your workload would like to calculate the average score of one of the homework for all students, which data structure would deliver better performance?
  A. Array of objects
  B. Object of arrays
### What data structure is performing better?

<table>
<thead>
<tr>
<th>Array of objects</th>
<th>object of arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>struct grades</code> { int id; double *homework; double average; };</td>
<td><code>struct grades</code> { int *id; double **homework; double *average; };</td>
</tr>
</tbody>
</table>

#### Average of each homework

```c
for(i=0;i<homework_items; i++)
{
  gradesheet[total_number_students].homework[i] = 0.0;
  for(j=0;j<total_number_students;j++)
  {
    gradesheet[total_number_students].homework[i] += gradesheet[j].homework[i];
    gradesheet[total_number_students].homework[i] /= (double)total_number_students;
  }
}
```

```c
for(i = 0;i < homework_items; i++)
{
  gradesheet.homework[i][total_number_students] = 0.0;
  for(j = 0; j <total_number_students;j++)
  {
    gradesheet.homework[i][total_number_students] += gradesheet.homework[i][j];
  }
  gradesheet.homework[i][total_number_students] /= total_number_students;
}
```

- Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance?
  A. Array of objects  
  B. Object of arrays
What data structure is performing better

<table>
<thead>
<tr>
<th>Array of objects</th>
<th>object of arrays</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>struct grades</code></td>
<td><code>struct grades</code></td>
</tr>
<tr>
<td>{</td>
<td>{</td>
</tr>
<tr>
<td>int id;</td>
<td>int *id;</td>
</tr>
<tr>
<td>double *homework;</td>
<td>double **homework;</td>
</tr>
<tr>
<td>double average;</td>
<td>double *average;</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

average of each homework

for(i=0;i<homework_items; i++)
{
    gradesheet[total_number_students].homework[i] = 0.0;
    for(j=0;j<total_number_students;j++)
    {
        gradesheet[total_number_students].homework[i] += gradesheet[j].homework[i];
    }
    gradesheet[total_number_students].homework[i] /= (double)total_number_students;
}

for(i = 0;i < homework_items; i++)
{
    gradesheet.homework[i][total_number_students] = 0.0;
    for(j = 0; j <total_number_students;j++)
    {
        gradesheet.homework[i][total_number_students] += gradesheet.homework[i][j];
    }
    gradesheet.homework[i][total_number_students] /= total_number_students;
}

• Considering your workload would like to calculate the average score of **one of the homework** for **all students**, which data structure would deliver better performance? *What if we want to calculate average scores for each student?*

  A. Array of objects
  
  B. Object of arrays
If you’re designing an in-memory database system, will you be using

- **column-store** — stores data tables column by column
  
<table>
<thead>
<tr>
<th>RowId</th>
<th>EmpId</th>
<th>Lastname</th>
<th>Firstname</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>Smith</td>
<td>Joe</td>
<td>40000</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>Jones</td>
<td>Mary</td>
<td>50000</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>Johnson</td>
<td>Cathy</td>
<td>44000</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>Jones</td>
<td>Bob</td>
<td>55000</td>
</tr>
</tbody>
</table>

- **row-store** — stores data tables row by row
  
  001:10, Smith, Joe, 40000;  
  002:12, Jones, Mary, 50000;  
  003:11, Johnson, Cathy, 44000;  
  004:22, Jones, Bob, 55000;
Loop interchange/fission/fusion
Loop interchange

A

```c
for(i = 0; i < ARRAY_SIZE; i++)
{
    for(j = 0; j < ARRAY_SIZE; j++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

B

```c
for(j = 0; j < ARRAY_SIZE; j++)
{
    for(i = 0; i < ARRAY_SIZE; i++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```

\[O(n^2)\]

**Complexity**

- A: Same
- B: Same

**Instruction Count?**

- A: Better
- B: Same

**Clock Rate**

- A: Same
- B: Same

**CPI**

- A: Better
- B: Worse
AMD Phenom II

• D-L1 Cache configuration of AMD Phenom II
  • Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

What’s the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%
What if the code look like this?

• D-L1 Cache configuration of AMD Phenom II
  • Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What’s the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%
What if the code look like this?

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What’s the data cache miss rate for this code?
A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%
What if the code look like this?

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```c
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i]; //load a and then store to c
for(i = 0; i < 512; i++)
    c[i] += b[i]; //load b, load c, add, and then store to c
```

What's the data cache miss rate for this code?

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%
What if the code look like this? — intel

```c
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];
```

A. Version A
B. Version B
C. They’re about the same

- D-L1 Cache configuration of Intel Core i7
- Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?
A. Version A
B. Version B
C. They’re about the same
• D-L1 Cache configuration of Intel Core i7
  • Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?
A. Version A
B. Version B
C. They’re about the same
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
What if the code look like this? — intel

```
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];
```

```
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];  
}
```

- D-L1 Cache configuration of Intel Core i7
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

Which version of code will perform better?
A. Version A
B. Version B
C. They’re about the same

Loop fusion
Blocking
Case study: Matrix Multiplication

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

Algorithm class tells you it’s \(O(n^3)\)
If \(n=1024\), it takes about 1 sec
How long is it take when \(n=2048\)?
Matrix Multiplication

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

- If each dimension of your matrix is 2048
  - Each row takes 2048*8 bytes = 16KB
  - The L1 $ of intel Core i7 is 32KB, 8-way, 64-byte blocked
  - You can only hold at most 2 rows/columns of each matrix!
  - You need the same row when $j$ increase!
Block algorithm for matrix multiplication

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

for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
            for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
                        c[ii][jj] += a[ii][kk]*b[kk][jj];
        }
    }
}

You only need to hold these sub-matrices in your cache
How do you know it’s better?

• Discover the cache miss rate
  • valgrind --tool=cachegrind cmd
    • cachegrind is a tool profiling the cache performance
  • Performance counter
    • Intel® Performance Counter Monitor http://www.intel.com/software/pcm/
Matrix Transpose

// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+= (ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+= (ARRAY_SIZE/n)) {
        b_t[i][j] += b[j][i];
    }
}

for(i = 0; i < ARRAY_SIZE; i+= (ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+= (ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+= (ARRAY_SIZE/n)) {
            for(ii = i; ii < i+ (ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+ (ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+ (ARRAY_SIZE/n); kk++)
                        c[ii][jj] += a[ii][kk]*b[kk][jj];
        }
    }
}

// Compute on b_t
for(i = 0; i < ARRAY_SIZE; i+= (ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+= (ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+= (ARRAY_SIZE/n)) {
            for(ii = i; ii < i+ (ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+ (ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+ (ARRAY_SIZE/n); kk++)
                        c[ii][jj] += a[ii][kk]*b_t[jj][kk];
        }
    }
}
What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

A. Compulsory miss  
B. Capacity miss  
C. Conflict miss  
D. Capacity & conflict miss  
E. Compulsory & conflict miss

```c
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
            for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
                        // Compute on b_t
                        c[ii][jj] += a[ii][kk]*b_t[jj][kk];
        }
    }
}
```
What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss
What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

```c
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+= (ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+= (ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+= (ARRAY_SIZE/n)) {
            for(ii = i; ii < i+ (ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+ (ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+ (ARRAY_SIZE/n); kk++)
                        b_t[ii][jj] += b[j][i];
    }
}
```

```c
for(i = 0; i < ARRAY_SIZE; i+= (ARRAY_SIZE/n)) {
    for(j = 0; j < ARRAY_SIZE; j+= (ARRAY_SIZE/n)) {
        for(k = 0; k < ARRAY_SIZE; k+= (ARRAY_SIZE/n)) {
            for(ii = i; ii < i+ (ARRAY_SIZE/n); ii++)
                for(jj = j; jj < j+ (ARRAY_SIZE/n); jj++)
                    for(kk = k; kk < k+ (ARRAY_SIZE/n); kk++)
                        c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
}
```
Summary of Optimizations

• Software
  • Data layout — capacity miss, conflict miss, compulsory miss
  • Blocking — capacity miss
  • Transpose — conflict miss
  • Loop fission — conflict miss — when $ has limited way associativity
  • Loop fusion — capacity miss — when $ has enough way associativity
  • Loop interchange — conflict/capacity miss

• Hardware
  • Prefetch — compulsory miss
  • Write buffer — miss penalty
  • Bank/pipeline — miss penalty
  • Critical word first and early restart — miss penalty
Both version A and B produces the same output. Without compiler optimization, which version of code would have better performance?

A. Version A
B. Version B
C. They are about the same (less than 5% difference)
Which version is faster?

- Both version A and B produces the same output. Without compiler optimization, which version of code would have better performance?
  A. Version A
  B. Version B
  C. They are about the same (less than 5% difference)
Basic Pipelined Processor

Hung-Wei Tseng
Tasks in RISC-V ISA

- Instruction Fetch (IF) — fetch the instruction from memory
- Instruction Decode (ID)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (EX)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (MEM) — Read/write memory
- Write Back (WB) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4
Simple implementation w/o branch

```
add x1, x2, x3
ld  x4, 0(x5)
sub x6, x7, x8
sub x9, x10, x11
sd  x1, 0(x12)
```

IF | ID | EX | WB
---|----|----|----
IF | ID | EX | MEM | WB
IF | ID | EX | WB
IF
Pipelining
Pipelining

- Different parts of the processor works on different instructions simultaneously
- A clock signal controls and synchronize the beginning and the end of each part of the work
- A pipeline register between different parts of the processor to keep intermediate results necessary for the upcoming work
Pipelining
add x1, x2, x3
ld x4, 0(x5)
sub x6, x7, x8
sub x9, x10, x11
sd x1, 0(x12)
xor x13, x14, x15
and x16, x17, x18
add x19, x20, x21
sub x22, x23, x24
ld x25, 4(x26)
sd x27, 0(x28)

After this point, we are completing an instruction each cycle!
Both version A and B production the same output. Without compiler optimization, which version of code would have significantly better performance?

A. Version A
B. Version B
C. They are about the same (less than 10% difference)

— Because we have pipelined instructions, the CPI of one instruction doesn’t matter as long as we can keep the pipeline busy
• Project is up — check the website
• Assignment #3 due next Monday
• Midterm
  • Release next Tuesday (11/10) 12:00am, turn in before next Friday (11/13) 11:59pm
  • You can only open it once and you have to finish a total of 30 questions within 80 minutes.
  • You may open book, but you have to bare the risks of not being able to finish them
• Attendance
  • The attendance throughout the quarter count as one assignment
  • You only need to answer 50% of the Zoom polls to receive full credits
    • Please don’t email me for absence — we count only 50% to give you flexibility
    • If you just login but never answer questions, you won’t receive any.
• Reading Quizzes — 2 attempts, average
• Office Hours on Zoom (the office hour link, not the lecture one)
  • Hung-Wei/Prof. Usagi: M 8p-10p (make up for the last week), W 2p-3p
  • Quan Fan: F 1p-3p