Cache-Friendly Code

Zhaoguo Wang
Step 1. Address Translation

CPU

a[0]++
incq (%rax)

TLB

CPU Cache

VA

PA

Word (8 bytes)

Cache line (64 bytes)

TLB Miss

Cache Miss

Level 0

Level 1

Level 2

Level 3

Page table

......

Data

Memory (RAM)
Step 1. Address Translation

```plaintext
a[0]++
incq (%rax)
```

Diagram:
- CPU
- TLB
- CPU Cache
- Memory (RAM)
- Page table
- Cache line (64 bytes)
- Cache Miss
- TLB Miss
- Word (8 bytes)
Step 1.1 Check TLB

```
CPU

a[0]++
incq (%rax)

TLB

CPU Cache

Cache Miss

Cache line (64 bytes)

Memory (RAM)

Page table

......

Data

```

**Diagram:**
- **CPU:**
  - `a[0]++`
  - `incq (%rax)`
  - VA
  - PA
- **TLB:**
- **CPU Cache:**
  - Word (8 bytes)
- **Memory (RAM):**
  - Level 0
  - Level 1
  - Level 2
  - Level 3
  - Page table
Step 1.1 Check TLB

Step 1.1.1 calculate the set index in TLB

e.g., TLB has 1024 sets
4 way associative

Index the set in TLB
Step 1.1 Check TLB

Step 1.1.2 find the buffered mapping in the set by comparing tag

e.g., TLB has 1024 sets 4 way associative

CPU

a[0]++
incq (%rax)

TLB

set

compare the tags

CPU Cache

Word (8 bytes)

CPU Chip

CPU Chip

Tag

Set Index

Page offset

0

12

11

31

63
Step 1.1 Check TLB

**Step 1.1.3** calculate the physical address on TLB hit

e.g., TLB has 1024 sets 4 way associative

compare the tags

if exist an entry e in the TLB set with index va[12:31] and its tag is equal to va[32:63], then PA is equal to e.PPN + va[0:11]
Step 1.2 Walk Through Page Table on TLB Miss

Step 1.2.1 find the PPN by walking through the page table with VA e.g., 4 level page table

TLB

CPU Cache

CPU Chip

a[0]++ incq (%rax)

set

CPU

VA

PA

Word (8 bytes)

Level 0

Level 1

Level 2

Level 3

CR3

VA

Reserved L0 off L1 off L2 off L3 off Page Offset

63 48 47 39 38 30 29 21 20 12 11 0
Step 1.2 Walk Through Page Table on TLB Miss

**Step 1.2.2 Buffer the mapping in TLB**

*Example: 4 level page table*

<table>
<thead>
<tr>
<th>Level 0</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>L0 off</td>
<td>L1 off</td>
<td>L2 off</td>
</tr>
</tbody>
</table>

`a[0]++ incq (%rax)`
Step 1.2 Walk Through Page Table on TLB Miss

Step 1.2.3 Calculate the physical address, e.g., 4 level page table

- CR3
- VA
- Word (8 bytes)
- CPU Chip
- CPU
- CPU Cache
- TLB
- Set
- VA
- PA
- Level 0
- Level 1
- Level 2
- Level 3
- CR3 Root Addr
- Reserved
- L0 off
- L1 off
- L2 off
- L3 off
- Page Offset
- 63 48 47 39 38 30 29 21 20 12 11 0

a[0]++
incq (%rax)
Exercise

32 bit machine, 2 level page table, 4K page size
Step 2. Fetch Data

```
CPU
a[0]++
incq (%rax)

CPU Chip

CPU Cache

TLB

TLB Miss

Faulting page table

Level 0

Level 1

Level 2

Level 3

Memory (RAM)

……

Data

Cache line (64 bytes)

Cache Miss

VA

PA

Word (8 bytes)
```
Step 2.1 Fetch Data from CPU Cache

Step 2.1.1 Calculate the set index

E.g., CPU Cache, 4 ways, 64 sets, 64 bytes cache line, virtual index physical tag,
Step 2.1 Fetch Data from CPU Cache

Step 2.1.2 find the buffered cache line by comparing the tag in PA

e.g., CPU Cache, 4 ways, 64 sets, 64 bytes cache line, virtual index physical tag,
Step 2.1 Fetch Data from CPU Cache

Step 2.1.3 on cache hit, find the data in the cache line using last 6 bits

- Example: CPU Cache, 4 ways, 64 sets, 64 bytes cache line, virtual index physical tag,

```
<table>
<thead>
<tr>
<th>Cache line tag</th>
<th>Cache line offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>11 6 5 0</td>
</tr>
</tbody>
</table>
```
Step 2.2 Fetch Data from Memory on Cache Miss

Step 2.2.1 on cache miss, load 64 bytes data from (PA&(~0x3f))
  e.g., CPU Cache, 4 ways, 64 sets, 64 bytes cache line,
  virtual index physical tag,
Writing Cache-Friendly Code

Principles

– Programs with better locality will tend to have lower miss rates
– Programs with lower miss rates will tend to run faster than programs with higher miss rates
  • Miss rate – fraction of memory references not found in cache (misses/references)
  • Typical numbers: 3-10% for L1, can be quite small (<1%) for L2, depending on size
How to write cache friendly code?

Memory access pattern

Memory layout

......
Simple example: sum of 2D array

```c
int64 sumarrayrows(int64** a, int r, int c) {
    int i, j = 0;
    int64 sum = 0

    for (int i = 0 ; i < r; i++)
        for (int j = 0 ; j < c; j++)
            sum += a[i][j];
    return sum ;
}

int64 sumarraycols(int64** a, int r, int c) {
    int i, j = 0;
    int64 sum = 0;

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

Which implementation is more cache friendly?
int64 sumarrayrows(int64** a, int r, int c) {
    int i, j = 0;
    int64 sum = 0
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            sum += a[i][j];
    return sum;
}

int64 sumarraycols(int64** a, int r, int c) {
    int i, j = 0;
    int64 sum = 0;
    for (int j = 0; j < c; j++)
        for (int i = 0; i < r; i++)
            sum += a[i][j];
    return sum;
}

How many cache misses?

Example:
CPU Cache – 2 way associative, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment
Assume all variables are in the registers
Simple Example

Example:
CPU Cache – 2-way, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

Memory

```
   a[5][7]
   a[5][1]
a[5][0]
a[4][7]
   a[4][1]
a[4][0]
a[3][7]
   a[3][1]
a[3][0]
a[2][7]
   a[2][1]
a[2][0]
a[1][7]
   a[1][1]
a[1][0]
a[0][7]
   a[0][1]
a[0][0]
```

CPU Cache

```
Set 0
Set 1
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int j = 0; j < c; j++)
for (int i = 0; i < r; i++)
    sum += a[i][j];
```

Miss
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0; j < c; j++)
  for (int i = 0; i < r; i++)
    sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
  for (int i = 0 ; i < r; i++)
    sum += a[i][j];
```

Set 0: a[0][0], a[0][1] ... a[0][7]  a[2][0], a[2][1] ... a[2][7]
Set 1: a[1][0], a[1][1] ... a[1][7]

CPU Cache
Simple Example

Example:

CPU Cache – 2 ways, 2 sets, 64 bytes cache line

Array – int64 a[6][8]

The address of a[0][0] is cache line alignment

```cpp
for (int j = 0; j < c; j++)
    for (int i = 0; i < r; i++)
        sum += a[i][j];
```
Simple Example

Example:

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]

The address of a[0][0] is cache line alignment

```
for (int j = 0; j < c; j++)
    for (int i = 0; i < r; i++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

for (int j = 0; j < c; j++)
    for (int i = 0; i < r; i++)
        sum += a[i][j];
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```

```
[CPU Cache]
Set 0  a[0][0], a[0][1] ... a[0][7]  a[4][0], a[4][1] ... a[4][7]
       a[0][0], a[0][1] ... a[0][7]  a[4][0], a[4][1] ... a[4][7]

Set 1  a[1][0], a[1][1] ... a[1][7]  a[5][0], a[5][1] ... a[5][7]
       a[1][0], a[1][1] ... a[1][7]  a[5][0], a[5][1] ... a[5][7]
```

Miss
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
    for (int i = 0 ; i < r; i++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
  for (int i = 0 ; i < r; i++)
    sum += a[i][j];
```

```
Set 0
  a[2][0], a[2][1] ... a[2][7]  a[0][0], a[0][1] ... a[0][7]

Set 1
  a[5][0], a[5][1] ... a[5][7]  a[1][0], a[1][1] ... a[1][7]
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int j = 0; j < c; j++)
    for (int i = 0; i < r; i++)
        sum += a[i][j];
```

```
  Miss
```

Set 0
| a[2][0], a[2][1] ... a[2][7] | a[0][0], a[0][1] ... a[0][7] |

Set 1
| a[3][0], a[3][1] ... a[3][7] | a[1][0], a[1][1] ... a[1][7] |
Simple Example

Example:

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0; j < c; j++)
    for (int i = 0; i < r; i++)
        sum += a[i][j];
```

CPU Cache

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[2][0], a[2][1]</td>
<td>a[4][0], a[4][1]</td>
</tr>
<tr>
<td>a[2][7]</td>
<td>a[4][7]</td>
</tr>
<tr>
<td>a[3][0], a[3][1]</td>
<td>a[1][0], a[1][1]</td>
</tr>
<tr>
<td>a[3][7]</td>
<td>a[1][7]</td>
</tr>
<tr>
<td>a[4][0], a[4][1]</td>
<td>a[1][0], a[1][1]</td>
</tr>
</tbody>
</table>

Miss
Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int j = 0 ; j < c; j++)
  for (int i = 0 ; i < r; i++)
    sum += a[i][j];
```

```
for (int i : 5, j : 1)
```

```
for (int j = 0 ; j < c; j++)
  for (int i = 0 ; i < r; i++)
    sum += a[i][j];
```

Set 0: a[2][0], a[2][1] ... a[2][7] a[4][0], a[4][1] ... a[4][7]
Set 1: a[3][0], a[3][1] ... a[3][7] a[5][0], a[5][1] ... a[5][7]

CPU Cache

Miss
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

Memory

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a[5][7]</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[5][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[5][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[4][7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[4][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[4][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[3][7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[3][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[3][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[2][7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[2][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[2][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[1][7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[1][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[1][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[0][7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[0][1]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a[0][0]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

CPU Cache

```
Set 0
a[0][0], a[0][1] ... a[0][7]

Set 1
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
  for (int j = 0 ; j < c; j++)
    sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
  for (int j = 0 ; j < c; j++)
    sum += a[i][j];
```
**Simple Example**

Example:

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

```
      a[5][7] ...
      a[5][1] a[5][0] a[4][7] ...
      a[4][1] a[4][0] a[3][7] ...
      a[3][1] a[3][0] a[2][7] ...
      a[2][1] a[2][0] a[1][7] ...
        a[1][1] a[1][0] a[0][7] ...
```

```
Memory
  a[5][7] ...
  a[5][1] a[5][0] a[4][7] ...
  a[4][1] a[4][0] a[3][7] ...
  a[3][1] a[3][0] a[2][7] ...
  a[2][1] a[2][0] a[1][7] ...
    a[1][1] a[1][0] a[0][7] ...
```

```
CPU Cache
  Set 0
    a[0][0], a[0][1] ... a[0][7]
      a[0][0], a[0][1] ... a[0][7]
  Set 1
    a[1][0], a[1][1] ... a[1][7]
      a[1][0], a[1][1] ... a[1][7]
```

Hit
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)    i:1, j:7
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

```
Set 0
---
a[0][0], a[0][1] ... a[0][7]  a[2][0], a[2][1] ... a[2][7]
---
Set 1
---
a[1][0], a[1][1] ... a[1][7]
---
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0; i < r; i++)
    for (int j = 0; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
  for (int j = 0 ; j < c; j++)
    sum += a[i][j];
```

Memory

CPU Cache

Set 0
- a[0][0], a[0][1] ... a[0][7]
- a[2][0], a[2][1] ... a[2][7]

Set 1
- a[1][0], a[1][1] ... a[1][7]
- a[3][0], a[3][1] ... a[3][7]
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```c
for (int i = 0; i < r; i++)
    for (int j = 0; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0; i < r; i++)
    for (int j = 0; j < c; j++)
        sum += a[i][j];
```

Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

for (int i = 0 ; i < r; i++)  
  for (int j = 0 ; j < c; j++)  
    sum += a[i][j];
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```

```c
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
  for (int j = 0 ; j < c; j++)
    sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0; i < r; i++)
    for (int j = 0; j < c; j++)
        sum += a[i][j];
```
Simple Example

Example:
CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – int64 a[6][8]
The address of a[0][0] is cache line alignment

```
for (int i = 0 ; i < r; i++)
    for (int j = 0 ; j < c; j++)
        sum += a[i][j];
```
Matrix Multiplication (ijk)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[
C = A \ast B \rightarrow C[i,j] = A[i:] \cdot B[:j]
\]

for (int i=0; i < N; i++) {
    for (int j=0; j < N; j++) {
        for (int k=0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
    }
}
Matrix Multiplication

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

```
for (int i=0; i < N; i++) {
    for (int k=0; k < N; k++) {
        for (int j=0; j < N; j++)
            C[i][j] += A[i][k] * B[k][j];
    }
}
```

```
for (int i=0; i < N; i++) {
    for (int j=0; j < N; j++) {
        for (int k=0; k < N; k++)
            C[i][j] += A[i][k] * B[k][j];
    }
}

```

MM—ijk

```
for (int k=0; k < N; k++) {
    for (int j=0; j < N; j++) {
        for (int i=0; i < N; i++)
            C[i][j] += A[i][k] * B[k][j];
    }
}
```

MM—kji

Which one is cache friendly? Which one is worst?
Matrix Multiplication ($ijk$)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

$$C = A \ast B \Rightarrow C[i,j] = A[i:] \cdot B[:j]$$

for (int $i=0$; $i < N$; $i++$) {
    for (int $j=0$; $j < N$; $j++$) {
        for (int $k=0$; $k < N$; $k++$)
            $C[i][j] += A[i][k] \ast B[k][j]$;
    }
}

\[
\begin{array}{cc}
\begin{array}{cc}
\text{C} & \text{A} \\
\text{j} & \text{i} \\
\end{array}
\end{array}
\ast
\begin{array}{cc}
\begin{array}{cc}
\text{B} & \\
\text{j} & \\
\end{array}
\end{array}
\]
Matrix Multiplication (ijk)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[
C = A \ast B \Rightarrow C[i,j] = A[i:] \cdot B[:j]
\]

for (int i=0; i < N; i++) {
    for (int j=0; j < N; j++) {
        for (int k=0; k < N; k++)
            C[i][j] += A[i][k] \ast B[k][j];
    }
}

\[
\begin{array}{ccccccc}
\text{\textbf{C}} & \text{\textbf{A}} & \text{\textbf{B}} \\
\hline
\end{array}
\]

---

Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned
Matrix Multiplication \((ijk)\)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[
C = A \times B \Rightarrow C[i,j] = A[i:] \cdot B[:j]
\]
for \((i = 0; i < N; i++)\) {
    for \((j = 0; j < N; j++)\) {
        for \((k = 0; k < N; k++)\) {
            \(C[i][j] += A[i][k] \times B[k][j];\)
        }
    }
}
Matrix Multiplication (ikj)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[
C = A \times B \Rightarrow C[i:] += A[i:k] \times B[k:]
\]
\[
(k \text{ in } [0, N))
\]

for (int i=0; i < N; i++) {
    for (int k=0; k < N; k++) {
        for (int j=0; j < N; j++)
            C[i][j] += A[i][k] \times B[k][j];
    }
}

C

A

B
Matrix Multiplication (ikj)

CPU Cache – 2 ways, 2 sets, 64 bytes cache line

Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[
C = A \ast B \rightarrow C[i:] += A[i:k] \ast B[k:] \\
(k \text{ in } [0, \, N))
\]

for (int i=0; i < N; i++) {
    for (int k=0; k < N; k++) {
        for (int j=0; j < N; j++)
            C[i][j] += A[i][k] \ast B[k][j];
    }
}
**Matrix Multiplication (kji)**

CPU Cache – 2 ways, 2 sets, 64 bytes cache line

Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

$$C = A * B \rightarrow C[:j] += \ A[:k] * B[k:j]$$

```java
for (int k=0; k < N; k++) {
    for (int j=0; j < N; j++) {
        for (int i=0; i < N; i++)
            C[i][j] += A[i][k] * B[k][j];
    }
}
```

Matrix A, B, C – double[8][8]

The address of the first element is cache line aligned
Matrix Multiplication (kji)

CPU Cache – 2 ways, 2 sets, 
64 bytes cache line
Matrix A, B, C – double[8][8]
The address of the first element is cache line aligned

\[ C = A \times B \rightarrow C[:j] += A[:k] \times B[k:j] \]

\[ \text{k in } [k, N) \]

for (int k=0; k < N; k++) {
    for (int j=0; j < N; j++) {
        for (int i=0; i < N; i++)
            C[i][j] += A[i][k] \times B[k][j];
    }
}
typedef struct {
    char g;
    int64_t id;
    char g1;
    int64_t id1;
    char g2;
    int64_t id2;
} info;

typedef struct {
    int64_t id;
    int64_t id1;
    int64_t id2;
    char g;
    char g1;
    char g2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g2 = 'y';
    a[i].g1 = 'e';
    a[i].g = 's';
}

Thanks Chien-Chin!! 😊
typedef struct {
    char g;
    int64_t id;
    char g1;
    int64_t id1;
    char g2;
    int64_t id2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g2 = 'y';
    a[i].g1 = 'e';
    a[i].g = 's';
}
```c
typedef struct {
    char g;
    int64_t id;
    char g1;
    int64_t id1;
    char g2;
    int64_t id2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g2 = 'y';
    a[i].g1 = 'e';
    a[i].g = 's';
}
```

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment

access a[0]
typedef struct {
    char g;
    int64_t id;
    char g1;
    int64_t id1;
    char g2;
    int64_t id2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g2 = 'y';
    a[i].g1 = 'e';
    a[i].g = 's';
}

The address of a[0] is cache line alignment

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]

access a[1]
typedef struct {
    int64_t id;
    int64_t id1;
    int64_t id2;
    char g;
    char g1;
    char g2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g2 = 'y';
    a[i].g1 = 'e';
    a[i].g = 's';
}

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment

access a[0]
typedef struct {
    int64_t id;
    int64_t id1;
    int64_t id2;
    char g;
    char g1;
    char g2;
} info;

for(int i = 0 ; i < N; i++) {
    a[i].id = i;
    a[i].id1 = i+1;
    a[i].id2 = i+2;
    a[i].g = 'y';
    a[i].g1 = 'e';
    a[i].g2 = 's';
}

CPU Cache – 2 ways, 2 sets, 64 bytes cache line
Array – info a[2]
The address of a[0] is cache line alignment

access a[1]