CPU design effects that can degrade performance of your programs

Jakub Beránek
jakub.beranek@vsb.cz
• PhD student @ VSB-TUO, Ostrava, Czech Republic
• Research assistant @ IT4Innovations (HPC center)
• HPC, distributed systems, program optimization
How do we get maximum performance?

• Select the right algorithm
How do we get maximum performance?

• Select the right algorithm
• Use a low-overhead language
How do we get maximum performance?

- Select the right algorithm
- Use a low-overhead language
- Compile properly
How do we get maximum performance?

- Select the right algorithm
- Use a low-overhead language
- Compile properly
- **Tune to the underlying hardware**
Why should we care?

- We write code for the C++ abstract machine
Why should we care?

• We write code for the C++ abstract machine
• Intel CPUs fulfill the contract of this abstract machine
Why should we care?

- We write code for the C++ abstract machine
- Intel CPUs fulfill the contract of this abstract machine
- But inside they can do whatever they want
Why should we care?

• We write code for the C++ abstract machine
• Intel CPUs fulfill the contract of this abstract machine
  • But inside they can do whatever they want
• Understanding CPU trade-offs can get us more performance
How fast are the individual array increments?

```c++
void foo(int* arr, int count)
{
    for (int i = 0; i < count; i++)
    {
        arr[i]++;
    }
}
```
• Performance effects caused by a specific CPU/memory implementation
Hardware effects

- Performance effects caused by a specific CPU/memory implementation
- Demonstrate some CPU/memory trade-off or assumption
• Performance effects caused by a specific CPU/memory implementation
• Demonstrate some CPU/memory trade-off or assumption
• Impossible to predict from (C++) code alone
Hardware is getting more and more complex

42 Years of Microprocessor Trend Data

- Transistors (thousands)
- Single-Thread Performance (SpecINT x 10^3)
- Frequency (MHz)
- Typical Power (Watts)
- Number of Logical Cores

Source: karlrupp.net
Microarchitecture (Haswell)

Microarchitecture (Haswell)

Frontend

Microarchitecture (Haswell)

How bad is it?

• C++ 17 final draft:
How bad is it?

• C++ 17 final draft: 1622 pages

How bad is it?

- C++ 17 final draft: 1622 pages
- Intel x86 manual:

How bad is it?

• C++ 17 final draft: 1622 pages
• Intel x86 manual: 5764 pages!

Plan of attack

- Show example C++ programs
Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with -03
Plan of attack

• Show example C++ programs
  • short, (hopefully) comprehensible
  • compiled with -03
• Demonstrate weird performance behaviour
Plan of attack

• Show example C++ programs
  • short, (hopefully) comprehensible
  • compiled with -O3
• Demonstrate weird performance behaviour
• Let you guess what might cause it
Plan of attack

• Show example C++ programs
  • short, (hopefully) comprehensible
  • compiled with -O3
• Demonstrate weird performance behaviour
• Let you guess what might cause it
• Explain (a possible) cause
Plan of attack

- Show example C++ programs
  - short, (hopefully) comprehensible
  - compiled with -O3
- Demonstrate weird performance behaviour
- Let you guess what might cause it
- Explain (a possible) cause
- Show how to measure and fix it
Plan of attack

• Show example C++ programs
  • short, (hopefully) comprehensible
  • compiled with -O3
• Demonstrate weird performance behaviour
• Let you guess what might cause it
• Explain (a possible) cause
• Show how to measure and fix it

• Disclaimer #1: Everything will be Intel x86 specific
Plan of attack

• Show example C++ programs
  • short, (hopefully) comprehensible
  • compiled with -O3
• Demonstrate weird performance behaviour
• Let you guess what might cause it
• Explain (a possible) cause
• Show how to measure and fix it

• Disclaimer #1: Everything will be Intel x86 specific
• Disclaimer #2: I'm not an expert on this and I may be wrong :-(
Let's see some examples...
```cpp
std::vector<float> data = /* 32K random floats in [1, 10] */;
float sum = 0;
// std::sort(data.begin(), data.end());
for (auto x : data)
{
    if (x < 6.0f)
    {
        sum += x;
    }
}
```
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>CPU</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter_nosort/32768</td>
<td>133460 ns</td>
<td>132992 ns</td>
<td>5284</td>
</tr>
<tr>
<td>filter_sorted/32768</td>
<td>63069 ns</td>
<td>62991 ns</td>
<td>12547</td>
</tr>
</tbody>
</table>
Why is it faster to process a sorted array than an unsorted array?

Here is a piece of C++ code that seems very peculiar. For some strange reason, sorting the data miraculously makes the code almost six times faster.

```cpp
#include <algorithm>
#include <ctime>
#include <iostream>

int main()
{
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // !!! With this, the next loop runs faster
    std::sort(data, data + arraySize);

    // Test
    clock_t start = clock();
    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }
    return 0;
}
```
What is going on? (Intel Amplifier - VTune)

Elapsed Time: 2.222s

- Clockticks: 6,736,800,000
- Instructions Retired: 2,942,800,000
- CPI Rate: 2.289
- MUX Reliability: 0.802
- Retiring: 9.5% of Pipeline Slots
- Front-End Bound: 44.6% of Pipeline Slots
- Front-End Latency:
  - ICache Misses: 31.1% of Pipeline Slots
  - ITLB Overhead: 13.4% of Clockticks
- Branch Resteers:
  - Mispredicts Resteers: 12.6% of Clockticks
  - Clears Resteers: 0.0% of Clockticks
  - Unknown Branches: 0.8% of Clockticks
- DSB Switches:
  - Length Changing Prefixes: 0.0% of Clockticks
  - MS Switches: 4.5% of Clockticks
- Front-End Bandwidth:
  - Bad Speculation: 13.5% of Pipeline Slots
  - Branch Mispredict: 48.6% of Pipeline Slots
  - Machine Clears: 48.6% of Pipeline Slots
- Back-End Bound:
  - Total Thread Count: 1
  - Paused Time: 0s

Issue: A significant portion of Pipeline Slots is remaining empty due to issues in the Front-End.

Tips: Make sure the

A significant proportion of pipeline slots containing useful work are being cancelled. This can be caused by mispredicting branches or by machine clears. Note that this metric value may be

This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the “pipe efficiency” ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.
What is going on? (perf)

```bash
$ perf stat ./example0a --benchmark_filter=nosort

853.67 2012 task-clock (msec) # 0.997 CPUs utilized
30 context-switches # 0.035 K/sec
0 cpu-migrations # 0.000 K/sec
199 page-faults # 0.233 K/sec
3 159 530 915 cycles # 3.701 GHz
1 475 799 619 instructions # 0.47 insn per cycle
419 608 357 branches # 491,533 M/sec
102 425 035 branch-misses # 24.41% of all branches
```
Branch predictor

- 32K L1 Instruction Cache
- Pre-Decide
- Instruction Queue
- MSROM
- Decoder
- Uop Cache (DSB)
- BPU
- Allocate/Rename/Retire/MoveElimination/Zeroidiom
- IDQ
- Load Buffers, Store Buffers, Reorder Buffers
- 32K L1 Data Cache

Scheduler

- Port 0
  - ALU, SHIFT, VEC LOG, VEC SHFT, FP mul, FMA, DIV, STTNI, Branch2

- Port 1
  - ALU, Fast LEA, VEC ALU, VEC LOG, FP mul, FMA, FP add, Slow Int

- Port 5
  - ALU, Shift

- Port 6
  - ALU, Shift

- Port 4
  - STD

- Port 2
  - LD/STA

- Port 3
  - LD/STA

- Port 7
  - STA

Memory Control

- Primary Branch
### CPU pipeline 101

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **7** xor rax, rdx
- **8** add rax, rcx
- **9** cmp rax, rbx
- **10** je 15
- **11** inc rcx
- **15** ret

...
<table>
<thead>
<tr>
<th></th>
<th>Fetch</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU pipeline 101

```
7   xor rax, rdx
8   add rax, rcx
9   cmp rax, rbx
10  je  15
11  inc  rcx
12  ...
15  ret
```
## CPU pipeline 101

<table>
<thead>
<tr>
<th>Fetch</th>
<th>Decode</th>
<th>Execute</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### CPU Code Snippet

```assembly
7  xor  rax, rdx
8  add  rax, rcx
9  cmp  rax, rbx
10  je  15
11  inc  rcx
15  ret
```
# CPU pipeline 101

## Code Snippet

```
7  xor rax, rdx
8  add rax, rcx
9  cmp rax, rbx
10 je 15
11 inc rcx
15 ret
```
# CPU pipeline 101

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7 xor rax, rdx
8 add rax, rcx
9 cmp rax, rbx
10 je 15
11 inc rcx
12 ...
15 ret
<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td><img src="1" alt="Red" /></td>
<td><img src="2" alt="Blue" /></td>
<td><img src="3" alt="Yellow" /></td>
<td><img src="4" alt="Green" /></td>
<td><img src="5" alt="Question" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decode</td>
<td></td>
<td><img src="6" alt="Red" /></td>
<td><img src="7" alt="Blue" /></td>
<td><img src="8" alt="Yellow" /></td>
<td><img src="9" alt="Green" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td><img src="10" alt="Red" /></td>
<td><img src="11" alt="Blue" /></td>
<td><img src="12" alt="Yellow" /></td>
<td><img src="13" alt="Green" /></td>
<td><img src="14" alt="Red" /></td>
<td><img src="15" alt="Blue" /></td>
<td><img src="16" alt="Yellow" /></td>
</tr>
<tr>
<td>Write</td>
<td></td>
<td></td>
<td><img src="17" alt="Red" /></td>
<td><img src="18" alt="Blue" /></td>
<td><img src="19" alt="Yellow" /></td>
<td><img src="20" alt="Green" /></td>
<td></td>
</tr>
</tbody>
</table>

7 xor rax,rdx
8 add rax,rcx
9 cmp rax,rbx
10 je 15
11 inc rcx
...
• CPU tries to predict results of branches
- CPU tries to predict results of branches
- Misprediction can cost ~15-20 cycles!
Simple branch predictor - unsorted array

if (data[i] < 6) {
  ...
}

Prediction: Not taken
Simple branch predictor - unsorted array

if (data[i] < 6) {
    ...
}

6 < 6?

Prediction: Not taken
Simple branch predictor - unsorted array

```c
if (data[i] < 6) {
    ...
}
```

6 < 6?

Prediction: Not taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

2 < 6?

Prediction: Not taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

2 < 6?

Prediction: Taken
### Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>1</th>
<th>7</th>
<th>4</th>
<th>8</th>
<th>3</th>
<th>9</th>
</tr>
</thead>
</table>

1 < 6?

Prediction: Taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>1</th>
<th>7</th>
<th>4</th>
<th>8</th>
<th>3</th>
<th>9</th>
</tr>
</thead>
</table>

1 < 6?

Prediction: Taken
Simple branch predictor - unsorted array

if (data[i] < 6) {
    ...
}

7 < 6?

Prediction: Taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

```
6 2 1 7 4 8 3 9
```

7 < 6?

Prediction: Not taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ... 
}
```

Prediction: Not taken
if (data[i] < 6) {
    ...
}

Prediction: Taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

8 < 6?

Prediction: Not taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Not taken
### Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
<th>1</th>
<th>7</th>
<th>4</th>
<th>8</th>
<th>3</th>
<th>9</th>
</tr>
</thead>
</table>

Prediction: Taken

3 < 6?
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Taken

![Diagram showing a sequence of numbers with 9 < 6 indicated and prediction set to Taken](image)
Simple branch predictor - unsorted array

if (data[i] < 6) {
    ...
}

Prediction: Not taken
Simple branch predictor - unsorted array

```java
if (data[i] < 6) {
  ...
}
```

6 2 1 7 4 8 3 9

Prediction: Not taken

2 hits, 6 misses (25% hit rate)
Simple branch predictor - sorted array

if (data[i] < 6) {
    ...
}

Prediction: Not taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
</table>

1 < 6?

Prediction: Not taken
if (data[i] < 6) {
    ...
}

Prediction: Taken
Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

2 < 6?

Prediction: Taken
Simple branch predictor - sorted array

if (data[i] < 6) {
    ...
}

Prediction: Taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

3 < 6?

Prediction: Taken
Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

3 < 6?

Prediction: Taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

4 < 6?

Prediction: Taken
Simple branch predictor - sorted array

```
if (data[i] < 6) {
    ...
}
```

Prediction: Taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Not taken

6 < 6?

Prediction: Not taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Not taken
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Not taken
Simple branch predictor - sorted array

Prediction: Not taken

```java
if (data[i] < 6) {
    ...
}
```
Simple branch predictor - sorted array

```java
if (data[i] < 6) {
    ...
}
```

Prediction: Not taken
if (data[i] < 6) {
    ...
}

Prediction: Not taken
Simple branch predictor - sorted array

if (data[i] < 6) {
    ...
}

Prediction: Not taken

6 hits, 2 misses (75% hit rate)
How can the compiler help?

```c
float sum = 0;
for (auto _ : state) {
    for (auto x : data) {
        if (x < 6) {
            sum += x;
        }
    }
}
```

With `float`, there are two branches per iteration.

```assembly
47  mov  rax, rcx
48  cmp  rcx, rdx
49  je   .L9
50  .L12:
51  movss xmm0, DWORD PTR [rax]
52  comiss xmm2, xmm0
53  jbe  .L10
54  addss xmm1, xmm0
55  .L10:
56  add  rax, 4
57  cmp  rdx, rax
58  jne  .L12
59  .L9:
60  sub  rbx, 1
61  jne  .L13
62  jmp  .L6
```
How can the compiler help?

```c
int sum = 0;
for (auto _ : state) {
    for (auto x : data) {
        if (x < 6) {
            sum += x;
        }
    }
}
```

With `int`, one branch is removed (using `cmov`)

```asm
.L12:
  mov    rax, r8
  cmp    r8, rdi
  je     .L9

.L11:
  mov    edx, DWORD PTR [rax]
  cmp    edx, 6
  lea    ecx, [rbx+rdx]
  cmovl  ebx, ecx
  add    rax, 4
  cmp    rdi, rax
  jne    .L11

.L9:
  sub    rbp, 1
  jne    .L12
  mov    rdi, r12
```
branch-misses

How many times was a branch mispredicted?
How to measure?

branch-misses

How many times was a branch mispredicted?

$ perf stat -e branch-misses ./example0a
with sort -> 383 902
without sort -> 101 652 009
How to help the branch predictor?

- More predictable data
How to help the branch predictor?

• More predictable data
• Profile-guided optimization
How to help the branch predictor?

- More predictable data
- Profile-guided optimization
- Remove (unpredictable) branches
How to help the branch predictor?

• More predictable data
• Profile-guided optimization
• Remove (unpredictable) branches
• Compiler hints (use with caution)

```c
if (__builtin_expect(will_it_blend(), 0)) {
    // this branch is not likely to be taken
}
```
• Target of a jump is not known at compile time:
• Target of a jump is not known at compile time:
  • Function pointer
Branch target prediction

- Target of a jump is not known at compile time:
  - Function pointer
  - Function return address
Branch target prediction

- Target of a jump is not known at compile time:
  - Function pointer
  - Function return address
  - Virtual method
```cpp
struct A { virtual void handle(size_t* data) const = 0; }
struct B: public A { void handle(size_t* data) const final { *data += 1; } };
struct C: public A { void handle(size_t* data) const final { *data += 2; } };

std::vector<std::unique_ptr<A>> data = /* 4K random B/C instances */;
// std::sort(data.begin(), data.end(), /* sort by instance type */);
size_t sum = 0;
for (auto& x : data)
{
    x->handle(&sum);
}
```
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>CPU</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle_nosort/4096</td>
<td>23350 ns</td>
<td>23349 ns</td>
<td>30734</td>
</tr>
<tr>
<td>handle_sorted/4096</td>
<td>7448  ns</td>
<td>7448  ns</td>
<td>86814</td>
</tr>
</tbody>
</table>
$ perf stat -e branch-misses ./example0b
with sort  ->   337  274
without sort ->   84  183  161
/\ Addresses of N integers, each `offset` bytes apart
std::vector<int*> data = ...;
for (auto ptr: data)
{
    *ptr += 1;
}
// Offsets: 4, 64, 4000, 4096, 4128
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Time</th>
<th>CPU</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset4/7</td>
<td>2.83 ns</td>
<td>2.83 ns</td>
<td>255750233</td>
</tr>
<tr>
<td>offset4/8</td>
<td>3.03 ns</td>
<td>3.02 ns</td>
<td>177109965</td>
</tr>
<tr>
<td>offset4/9</td>
<td>3.76 ns</td>
<td>3.75 ns</td>
<td>157739295</td>
</tr>
<tr>
<td>offset4/10</td>
<td>4.62 ns</td>
<td>4.61 ns</td>
<td>177899906</td>
</tr>
<tr>
<td>offset4/11</td>
<td>4.93 ns</td>
<td>4.92 ns</td>
<td>162959140</td>
</tr>
<tr>
<td>offset64/7</td>
<td>3.19 ns</td>
<td>3.18 ns</td>
<td>179723151</td>
</tr>
<tr>
<td>offset64/8</td>
<td>3.83 ns</td>
<td>3.65 ns</td>
<td>216288609</td>
</tr>
<tr>
<td>offset64/9</td>
<td>3.24 ns</td>
<td>3.74 ns</td>
<td>201008685</td>
</tr>
<tr>
<td>offset64/10</td>
<td>4.41 ns</td>
<td>4.40 ns</td>
<td>159949703</td>
</tr>
<tr>
<td>offset64/11</td>
<td>4.41 ns</td>
<td>4.41 ns</td>
<td>128933855</td>
</tr>
<tr>
<td>offset4000/7</td>
<td>3.69 ns</td>
<td>3.69 ns</td>
<td>187745245</td>
</tr>
<tr>
<td>offset4000/8</td>
<td>3.27 ns</td>
<td>3.26 ns</td>
<td>226401022</td>
</tr>
<tr>
<td>offset4000/9</td>
<td>3.19 ns</td>
<td>3.18 ns</td>
<td>157866983</td>
</tr>
<tr>
<td>offset4000/10</td>
<td>4.49 ns</td>
<td>4.48 ns</td>
<td>173084452</td>
</tr>
<tr>
<td>offset4000/11</td>
<td>4.53 ns</td>
<td>4.52 ns</td>
<td>128906229</td>
</tr>
<tr>
<td>offset4096/7</td>
<td>9.05 ns</td>
<td>9.05 ns</td>
<td>78087527</td>
</tr>
<tr>
<td>offset4096/8</td>
<td>10.4 ns</td>
<td>10.4 ns</td>
<td>67550724</td>
</tr>
<tr>
<td>offset4096/9</td>
<td>18.7 ns</td>
<td>18.7 ns</td>
<td>38875870</td>
</tr>
<tr>
<td>offset4096/10</td>
<td>25.5 ns</td>
<td>25.5 ns</td>
<td>26893946</td>
</tr>
<tr>
<td>offset4096/11</td>
<td>32.7 ns</td>
<td>32.7 ns</td>
<td>21369400</td>
</tr>
<tr>
<td>offset4128/7</td>
<td>3.23 ns</td>
<td>3.22 ns</td>
<td>250263727</td>
</tr>
<tr>
<td>offset4128/8</td>
<td>3.13 ns</td>
<td>3.13 ns</td>
<td>218371877</td>
</tr>
<tr>
<td>offset4128/9</td>
<td>3.75 ns</td>
<td>3.71 ns</td>
<td>157448182</td>
</tr>
<tr>
<td>offset4128/10</td>
<td>4.28 ns</td>
<td>4.25 ns</td>
<td>144839049</td>
</tr>
<tr>
<td>offset4128/11</td>
<td>5.47 ns</td>
<td>5.44 ns</td>
<td>128547528</td>
</tr>
</tbody>
</table>
How are (L1) caches implemented

• N-way set associative table
• Hardware hash table
How are (L1) caches implemented

- N-way set associative table
- Hardware hash table
- Key = address (8B)
How are (L1) caches implemented

- N-way set associative table
- Hardware hash table
- Key = address (8B)
- Entry = cache line (64B)
N-way set associative cache

Size = 8 cache lines
N-way set associative cache

Size = 8 cache lines

Associativity (N) - # of cache lines per bucket
N-way set associative cache

Size = 8 cache lines

Associativity (N) - # of cache lines per bucket
# of buckets = Size / N
N-way set associative cache

Size = 8 cache lines

Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

N = 1 (direct mapped)
N-way set associative cache

Size = 8 cache lines

Associativity (N) - # of cache lines per bucket
# of buckets = Size / N

N = 1 (direct mapped)

N = 8 (fully associative)
N-way set associative cache

Size = 8 cache lines

Associativity (N) - # of cache lines per bucket

# of buckets = Size / N

N = 1 (direct mapped)

N = 8 (fully associative)

N = 2
How are addresses hashed?

64-bit address:

- Tag
- Index
- Offset

63 0
How are addresses hashed?

64-bit address:
- Tag
- Index
- Offset

- **Offset**
  - Selects byte within a cache line
  - \( \log_2(\text{cache line size}) \) bits
How are addresses hashed?

64-bit address:

- **Offset**
  - Selects byte within a cache line
  - \( \log_2(\text{cache line size}) \) bits
- **Index**
  - Selects bucket within the cache
  - \( \log_2(\text{bucket count}) \) bits
How are addresses hashed?

64-bit address:

- **Offset**
  - Selects byte within a cache line
  - \( \log_2(\text{cache line size}) \) bits
- **Index**
  - Selects bucket within the cache
  - \( \log_2(\text{bucket count}) \) bits
- **Tag**
  - Used for matching
N-way set associative cache

Cache lines: A B C
Index bits: 0 1 0
N-way set associative cache

Cache lines:
A B C

Index bits:
0 1 0

N = 1

0 1 2 3 4 5 6 7
N-way set associative cache

Cache lines: A B C
Index bits: 0 1 0

N = 1

0 1 2 3 4 5 6 7

A
N-way set associative cache

Cache lines:
A  B  C

Index bits:
0  1  0

N = 1

A  B  0  1  2  3  4  5  6  7
N-way set associative cache

Cache lines:
A  B  C

Index bits:
0  1  0

N = 1
C  B  0  1  2  3  4  5  6  7

N-way set associative cache
N-way set associative cache

Cache lines: A B C
Index bits: 0 1 0

N = 1
N = 8
N-way set associative cache

Cache lines:

Index bits:

\[
\begin{array}{c}
\text{N = 1} \\
\begin{array}{cccccccc}
\text{C} & \text{B} & \text{A} & \text{D} & \text{E} & \text{F} & \text{G} & \text{H}
\end{array}
\end{array}
\]

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7
\end{array}
\]

\[
\begin{array}{c}
\text{N = 8} \\
\begin{array}{c}
\text{A}
\end{array}
\end{array}
\]

\[
\begin{array}{c}
0
\end{array}
\]
N-way set associative cache

Cache lines:
A
B
C

Index bits:
0 1 0

N = 1

A B

0 1 2 3 4 5 6 7

N = 8

A B

0
N-way set associative cache

Cache lines:

$A \quad B \quad C$

Index bits:

0 1 0

$N = 1$

A

B

C

0 1 2 3 4 5 6 7

$N = 8$

A

B

C

0
N-way set associative cache

Cache lines:

Index bits:

N = 1

N = 8

N = 2
N-way set associative cache

Cache lines:
A B C

Index bits:
0 1 0

N = 1
\[ \begin{array}{c}
\text{C} & \text{B} & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

N = 8
\[ \begin{array}{c}
\text{A} & \text{B} & \text{C} & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]

N = 2
\[ \begin{array}{c}
\text{A} & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } & \text{ } \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{array} \]
N-way set associative cache

Cache lines:

Index bits:

A
B
C

N = 1

N = 8

N = 2
N-way set associative cache

Cache lines:  A  B  C
Index bits:  0  1  0

N = 1

N = 8

N = 2
Intel L1 cache

$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE          32768
LEVEL1_DCACHE_ASSOC         8
LEVEL1_DCACHE_LINESIZE      64
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE      32768
LEVEL1_DCACHE_ASSOC     8
LEVEL1_DCACHE_LINESIZE  64

• **Cache line size** - 64 B (6 offset bits)
### Intel L1 cache

```bash
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE      32768
LEVEL1_DCACHE_ASSOC     8
LEVEL1_DCACHE_LINESIZE  64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** \((N)\) - 8
Intel L1 cache

```
getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE      32768
LEVEL1_DCACHE_ASSOC     8
LEVEL1_DCACHE_LINESIZE  64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** \((N)\) - 8
- **Size** - 32768 B
Intel L1 cache

```bash
$ getconf -a | grep LEVEL1_DCACHE
LEVEL1_DCACHE_SIZE      32768
LEVEL1_DCACHE_ASSOC     8
LEVEL1_DCACHE_LINESIZE  64
```

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** \( (N) \) - 8
- **Size** - 32768 B
- \( 32768 / 64 \rightarrow 512 \) cache lines
$ getconf -a | grep LEVEL1_DCACHE

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL1_DCACHE_SIZE</td>
<td>32768</td>
</tr>
<tr>
<td>LEVEL1_DCACHE_ASSOC</td>
<td>8</td>
</tr>
<tr>
<td>LEVEL1_DCACHE_LINESIZE</td>
<td>64</td>
</tr>
</tbody>
</table>

- **Cache line size** - 64 B (6 offset bits)
- **Associativity** *(N)* - 8
- **Size** - 32768 B
- **32768 / 64** => 512 cache lines
- **512 / 8** => 64 buckets (6 index bits)
Offset = 4B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>
Offset = 4B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>..100000</td>
<td>000000</td>
<td>000100</td>
</tr>
<tr>
<td>Number</td>
<td>Tag</td>
<td>Index</td>
<td>Offset</td>
</tr>
<tr>
<td>--------</td>
<td>--------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000000</td>
<td>000100</td>
</tr>
<tr>
<td>C</td>
<td>.100000</td>
<td>000000</td>
<td>001000</td>
</tr>
</tbody>
</table>

![Diagram with offset 4B]
<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000000</td>
<td>000100</td>
</tr>
<tr>
<td>C</td>
<td>.100000</td>
<td>000000</td>
<td>001000</td>
</tr>
<tr>
<td>D</td>
<td>.100000</td>
<td>000000</td>
<td>001100</td>
</tr>
</tbody>
</table>
Offset = 4B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000000</td>
<td>000100</td>
</tr>
<tr>
<td>C</td>
<td>.100000</td>
<td>000000</td>
<td>001000</td>
</tr>
<tr>
<td>D</td>
<td>.100000</td>
<td>000000</td>
<td>001100</td>
</tr>
</tbody>
</table>

- Same bucket, same cache line for each number
### Offset = 4B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000000</td>
<td>000100</td>
</tr>
<tr>
<td>C</td>
<td>.100000</td>
<td>000000</td>
<td>001000</td>
</tr>
<tr>
<td>D</td>
<td>.100000</td>
<td>000000</td>
<td>001100</td>
</tr>
</tbody>
</table>

- Same bucket, same cache line for each number
- Most efficient, no space is wasted
<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>Number</td>
<td>Tag</td>
<td>Index</td>
<td>Offset</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>Number</td>
<td>Tag</td>
<td>Index</td>
<td>Offset</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>..100000</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>..100000</td>
<td>000010</td>
<td>000000</td>
</tr>
</tbody>
</table>
Offset = 64B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100000</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>.100000</td>
<td>000010</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>.100000</td>
<td>000011</td>
<td>000000</td>
</tr>
</tbody>
</table>
Offset = 64B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>0000000</td>
<td>0000000</td>
</tr>
<tr>
<td>B</td>
<td>..100000</td>
<td>0000001</td>
<td>0000000</td>
</tr>
<tr>
<td>C</td>
<td>..100000</td>
<td>0000010</td>
<td>0000000</td>
</tr>
<tr>
<td>D</td>
<td>..100000</td>
<td>0000011</td>
<td>0000000</td>
</tr>
</tbody>
</table>

- Different bucket for each number
Offset = 64B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>..100000</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>..100000</td>
<td>000010</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>..100000</td>
<td>000011</td>
<td>000000</td>
</tr>
</tbody>
</table>

- Different bucket for each number
- Wastes 60B in each cache line
<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>100000</td>
<td>000001</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>100000</td>
<td>000010</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>100000</td>
<td>000011</td>
<td>000000</td>
</tr>
</tbody>
</table>

- Different bucket for each number
- Wastes 60B in each cache line
- Equally distributed among buckets
Offset = 4096B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>
**Offset = 4096B**

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100001</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

![Diagram showing memory allocation with tags and offsets](image-url)
Offset = 4096B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>100001</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>100010</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

![Diagram showing the offset and tag values]
Offset = 4096B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>..100001</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>..100010</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>..100011</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

![Diagram showing a 4-bit tag and index with offset values]
Offset = 4096B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>.100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>.100001</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>.100010</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>.100011</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

• Same bucket, but different cache lines for each number!
### Offset = 4096B

<table>
<thead>
<tr>
<th>Number</th>
<th>Tag</th>
<th>Index</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>..100000</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>..100001</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>C</td>
<td>..100010</td>
<td>000000</td>
<td>000000</td>
</tr>
<tr>
<td>D</td>
<td>..100011</td>
<td>000000</td>
<td>000000</td>
</tr>
</tbody>
</table>

- Same bucket, but different cache lines for each number!
- Bucket full => evictions necessary
How to measure?

`l1d.replacement`

How many times was a cache line loaded into L1?
How to measure?

```bash
$ perf stat -e l1d.replacement ./example1
4B   offset ->   149 558
4096B offset -> 426 218 383
```

How many times was a cache line loaded into L1?
float F = static_cast<float>(std::stof(argv[1]));
std::vector<float> data(4 * 1024 * 1024, 1);

for (int r = 0; r < 100; r++)
{
    for (auto& item: data)
    {
        item *= F;
    }
}
> time -p ./example2 0
real 0.12
user 0.12
sys 0.00
> time -p ./example2 0.1
real 0.47
user 0.46
sys 0.00
> time -p ./example2 0.3
real 0.70
user 0.69
sys 0.00
Denormal floating point numbers

\[ (-1)^0 \times 2^{00000-01110} \times 0.0000000001 \]
Denormal floating point numbers

Zero exponent: 00000000
Non-zero significand: 00000000000001

\((-1)^0 \times 2^{00000-01110} \times 0.0000000001\)
Denormal floating point numbers

Zero exponent: 0
Non-zero significand: 0.0000000001

$$(-1)^0 \times 2^{00000-01110} \times 0.0000000001$$

- Numbers close to zero
- Hidden bit = 0, smaller bias
Denormal floating point numbers

\[ (-1)^0 \times 2^{00000 - 01110} \times 0.0000000001 \]

- Numbers close to zero
- Hidden bit = 0, smaller bias

Operations on denormal numbers are slow!
Floating point handling
fp_assist.any

How many times the CPU switched to the microcode FP handler?
How to measure?

`fp_assist.any`

How many times the CPU switched to the microcode FP handler?

```bash
$ perf stat -e fp_assist.any ./example2
  0   ->          0
  0.3 -> 15 728 640
```
How to fix it?

- The nuclear option: `-ffast-math`
  - Sacrifice correctness to gain more FP performance
How to fix it?

• The nuclear option: -ffast-math
  • Sacrifice correctness to gain more FP performance
• Set CPU flags:
  • Flush-to-zero - treat denormal outputs as 0
  • Denormals-to-zero - treat denormal inputs as 0
How to fix it?

• The nuclear option: `-ffast-math`
  • Sacrifice correctness to gain more FP performance
• Set CPU flags:
  • Flush-to-zero - treat denormal outputs as 0
  • Denormals-to-zero - treat denormal inputs as 0

```c
_mm_setcsr(_mm_getcsr() | 0x8040);
// or
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
```
There are many other effects

- NUMA
- 4k aliasing
- Misaligned accesses, cache line boundaries
- Instruction data dependencies
- Software prefetching
- Non-temporal stores & cache pollution
- Bandwidth saturation
- DRAM refresh intervals
- AVX/SSE transition penalty
- ...

Thank you!

For more examples visit:

github.com/kobzol/hardware-effects

Jakub Beránek

Slides built with github.com/spirali/elsie
// tid - [0, NO_OF_THREADS)
void thread_fn(int tid, double* data)
{
    size_t repetitions = 1024 * 1024 * 1024UL;
    for (size_t i = 0; i < repetitions; i++)
    {
        data[tid] *= i;
    }
}
> time -p ./example2 1
real 2.60
user 2.60
sys 0.00
> time -p ./example2 2
real 2.92
user 5.83
sys 0.00
> time -p ./example2 3
real 3.15
user 9.04
sys 0.01
> time -p ./example2 4
real 3.39
user 13.44
sys 0.00
Cache system

- 32K L1 Instruction Cache
- Pre-Decide
- Instruction Queue
- MSROM
- Decoder
- IDQ
- Uop Cache (DSB)
- Allocate/Rename/Retire/MoveElimination/Zeroldiom
- Load Buffers, Store Buffers, Reorder Buffers
- BPU

Scheduler

- Port 0: ALU, SHFT, VEC LOG, VEC SHFT, FP mul, FMA, DIV, STTNI, Branch2
- Port 1: ALU, Fast LEA, VEC ALU, VEC LOG, FP mul, FMA, FP add, Slow int
- Port 5: ALU, SHFT
- Port 6: ALU, Shift
- Port 4: STD
- Port 2: LD/STA
- Port 3: LD/STA
- Port 7: STA

Memory Control

- Line Fill Buffers
- 32K L1 Data Cache
- 256K L2 Cache (Unified)
Cache coherency

Memory

Cache line: A B

Cache
Core 1

Cache
Core 2
Cache coherency

Memory

Cache line

A | B

Cache

Core 1

Read A

Cache

Core 2
Cache coherency

Memory

Cache line

Core 1

Cache

Core 2

Cache
Cache coherency

Memory

Cache line

A B

Cache

Core 1

A B

Core 2

A B

Write B
Cache coherency

Memory

Cache line

Core 1

Core 2
Cache coherency

Memory

Cache line

A B

Write A

Core 1

Core 2
Cache coherency

Memory

Cache line

A B

Core 1

Cache

A B

Core 2

Cache

A B

A B

A B

A B
False sharing

double arr[16];
double arr[16];
double arr[16];
double arr[16];
False sharing

```c
double arr[16];
```
False sharing

double arr[16];

Thread 0
Thread 1
```
double arr[16];
```
How to measure?

l2_rqsts.all_rfo

How many times some core invalidated data in other cores?
How to measure?

$l2_rqsts.all_rfo$

How many times some core invalidated data in other cores?

```
$ perf stat -e l2_rqsts.all_rfo ./example3
1 thread  ->        59 711
2 threads  -> 1 112 258 710
```