Memories

Hung-Wei Tseng
Recap: von Neumann Architecture

By loading different programs into memory, your computer can perform different functions.
Recap: Performance gap between Processor/Memory
Performance of modern DRAM

<table>
<thead>
<tr>
<th>Production year</th>
<th>Chip size</th>
<th>DRAM type</th>
<th>RAS time (ns)</th>
<th>CAS time (ns)</th>
<th>Total (ns)</th>
<th>Total (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>256M bit</td>
<td>DDR1</td>
<td>21</td>
<td>21</td>
<td>42</td>
<td>63</td>
</tr>
<tr>
<td>2002</td>
<td>512M bit</td>
<td>DDR1</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td>2004</td>
<td>1G bit</td>
<td>DDR2</td>
<td>15</td>
<td>15</td>
<td>30</td>
<td>45</td>
</tr>
<tr>
<td>2006</td>
<td>2G bit</td>
<td>DDR2</td>
<td>10</td>
<td>10</td>
<td>20</td>
<td>30</td>
</tr>
<tr>
<td>2010</td>
<td>4G bit</td>
<td>DDR3</td>
<td>13</td>
<td>13</td>
<td>26</td>
<td>39</td>
</tr>
<tr>
<td>2016</td>
<td>8G bit</td>
<td>DDR4</td>
<td>13</td>
<td>13</td>
<td>26</td>
<td>39</td>
</tr>
</tbody>
</table>

**Figure 2.4** Capacity and access times for DDR SDRAMs by year of production. Access time is for a random memory word and assumes a new row must be opened. If the row is in a different bank, we assume the bank is precharged; if the row is not open, then a precharge is required, and the access time is longer. As the number of banks has increased, the ability to hide the precharge time has also increased. DDR4 SDRAMs were initially expected in 2014, but did not begin production until early 2016.
Both version A and B produces 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)
The impact of “slow” memory

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, consider we have DDR4 and the program is well-behaved that precharge is never necessary — the access latency is simply 26 ns. What’s the average CPI (pick the most close one)?
  A. 9
  B. 17
  C. 27
  D. 35
  E. 69
The impact of “slow” memory

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, consider we have DDR4 and the program is well-behaved that precharge is never necessary — the access latency is simply 26 ns. What’s the average CPI (pick the most close one)?
  A. 9
  B. 17
  C. 27
  D. 35
  E. 69
Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, consider we have DDR4 and the program is well-behaved that precharge is never necessary — the access latency is simply 26 ns. What’s the average CPI (pick the most close one)?

A. 9
B. 17
C. 27
D. 35
E. 69

\[
1 + 100\% \times (52) + 30\% \times 52 = 68.6 \text{ cycles}
\]
Recap: Four implementations

- Which of the following implementations will perform the best on modern pipeline processors?

A

```c
inline int popcount(uint64_t x) {
    int c = 0;
    while(x) {
        c += x & 1;
        x = x >> 1;
    }
    return c;
}
```

B

```c
inline int popcount(uint64_t x) {
    int c = 0;
    while(x) {
        c += x & 1;
        x = x >> 1;
    }
    return c;
}
```

C

```c
inline int popcount(uint64_t x) {
    int c = 0;
    int table[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
    while(x) {
        c += table[(x & 0xF)];
        x = x >> 4;
    }
    return c;
}
```

D

```c
inline int popcount(uint64_t x) {
    int c = 0;
    int table[16] = {0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4};
    for (uint64_t i = 0; i < 16; i++) {
        c += table[(x & 0xF)];
        x = x >> 4;
    }
    return c;
}
```

Not going to work out if memory is that slow
Alternatives?

<table>
<thead>
<tr>
<th>Memory technology</th>
<th>Typical access time</th>
<th>$ per GiB in 2012</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRAM semiconductor memory</td>
<td>0.5–2.5 ns</td>
<td>$500–$1000</td>
</tr>
<tr>
<td>DRAM semiconductor memory</td>
<td>50–70 ns</td>
<td>$10–$20</td>
</tr>
<tr>
<td>Flash semiconductor memory</td>
<td>5,000–50,000 ns</td>
<td>$0.75–$1.00</td>
</tr>
<tr>
<td>Magnetic disk</td>
<td>5,000,000–20,000,000 ns</td>
<td>$0.05–$0.10</td>
</tr>
</tbody>
</table>

Fast, but expensive $$$
Memory Hierarchy

- **Processor Core**
  - Registers
  - SRAM (32 or 64 words)
- **DRAM**
- **Storage**

Speeds:
- Fastest: < 1 ns
- A few ns
- Tens of ns
- Tens of us

Sizes:
- KBs ~ MBs
- GBs
- TBs

Larger memories are used for larger data sizes.
 Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got an SRAM cache with latency of just at 0.5 ns and can capture 90% of the desired data/instructions. What’s the average CPI (pick the most close one)?

A. 2
B. 4
C. 8
D. 16
E. 32
How can memory hierarchy help in performance?

• Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got an SRAM cache with latency of just at 0.5 ns and can capture 90% of the desired data/instructions. What’s the average CPI (pick the most close one)?

A. 2  
B. 4  
C. 8  
D. 16  
E. 32
How can memory hierarchy help in performance?

• Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got an SRAM cache with latency of just at 0.5ns and can capture 90% of the desired data/instructions. what’s the average CPI (pick the most close one)?

A. 2
B. 4
C. 8
D. 16
E. 32

\[
1 + (1 - 90\%) \times [(100\% \times (26) + 30\% \times 52)] = 7.76 \text{ cycles}
\]
L1? L2? L3?

CPU-Z - ID: wswpb

CPU
- AMD Ryzen 7 2700X
- Code Name: Pinnacle Ridge
- Max TDP: 105 W
- Technology: 12 nm
- Core Voltage: 1.36 V
- Specification: AMD Ryzen 7 2700X Eight-Core Processor
- Family: F
- Model: 8
- Stepping: 2
- Ext. Family: 17
- Ext. Model: 8
- Revision: PIR-B2
- Instructions: MMX(+), SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, SSE4A
- x86-64, AMD-V, AES, AVX, AVX2, FMA3, SHA
- Clocks (Core #0):
  - Core Speed: 4290.73 MHz
  - Core Multiplier: x 43.0
  - Bus Speed: 99.78 MHz
- Cache:
  - L1 Data: 8 x 32 kBytes, 8-way
  - L1 Inst.:
    - 8 x 64 kBytes, 4-way
  - Level 2:
    - 8 x 512 kBytes, 8-way
  - Level 3:
    - 2 x 8192 kBytes, 16-way
- Selection:
  - Processor #1
  - Clocks:
    - 8 Cores
    - 16 Threads

CPU-Z - ID: wswpb

CPU
- Intel Core i7 9700K
- Code Name: Coffee Lake
- Max TDP: 95.0 W
- Technology: 14 nm
- Core Voltage: 0.737 V
- Specification: Intel® Core™ i7-9700K CPU @ 3.60GHz (ES)
- Family: 6
- Model:
  - E
  - Stepping:
    - C
- Ext. Family: 6
- Ext. Model:
  - 9E
  - Revision:
    - P0
- Instructions:
  - MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x
  - AES, AVX, AVX2, FMA3, TSX
- Clocks (Core #0):
  - Core Speed: 4798.85 MHz
  - Multiplier:
    - x 48.0 (8 - 49)
  - Bus Speed: 99.98 MHz
  - Clocks:
    - Level 2:
      - 8 x 256 kBytes, 4-way
    - Level 3:
      - 12 MBytes, 12-way
- Selection:
  - Socket #1
  - Clocks:
    - 8 Cores
    - 8 Threads
Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has "perfect" memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got a 2-level SRAM caches with

- it’s 1st-level one at latency of 0.5ns and can capture 90% of the desired data/instructions.
- the 2nd-level at latency of 5ns and can capture 60% of the desired data/instructions.

What’s the average CPI (pick the most close one)?

A. 2
B. 4
C. 8
D. 16
E. 32

How can deeper memory hierarchy help in performance?
Assume that we have a processor running at 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got a 2-level SRAM caches with:
• its 1st-level one at latency of 0.5ns and can capture 90% of the desired data/instructions.
• the 2nd-level at latency of 5ns and can capture 60% of the desired data/instructions.
What’s the average CPI (pick the most close one)?
A. 2
B. 4
C. 8
D. 16
E. 32

How can deeper memory hierarchy help in performance?
How can deeper memory hierarchy help in performance?

- Assume that we have a processor running @ 2 GHz and a program with 30% of load/store instructions. If the computer has “perfect” memory, the CPI is just 1. Now, in addition to DDR4, whose latency 26 ns, we also got a 2-level SRAM caches with
  - it’s 1st-level one at latency of 0.5ns and can capture 90% of the desired data/instructions.
  - the 2nd-level at latency of 5ns and can capture 60% of the desired data/instructions

What’s the average CPI (pick the most close one)?

A. 2
B. 4
C. 8
D. 16
E. 32

\[
1 + (1 - 90\%) \times [10 + (1 - 60\%) \times 52 + 30\% \times (10 + (1 - 60\%) \times 52)] = 5 \text{ cycle.}
\]
Memory Hierarchy

- Processor
  - Core
    - Registers
    - SRAM$
  - DRAM
  - Storage

- Processor
  - Register
  - L1$
  - L2$
  - L3$

- Fastest to Slowest:
  - Processor Core: < 1ns
  - SRAM$: a few ns
  - DRAM: tens of ns
  - Storage: tens of ns
  - L1$: fastest
  - L2$: larger
  - L3$: larger
  - Storage: TBs

- Relationship:
  - Fastest to Slowest: Processor Core > SRAM$ > DRAM > Storage
  - Larger to Smaller: L1$ > L2$ > L3$
Why adding small SRAMs would work?
Locality

• Which description about locality of arrays sum and A in the following code is the most accurate?

```c
for(i = 0; i< 100000; i++)
{
    sum[i%10] += A[i];
}
```

A. Access of A has temporal locality, sum has spatial locality
B. Both A and sum have temporal locality, and sum also has spatial locality
C. Access of A has spatial locality, sum has temporal locality
D. Both A and sum have spatial locality
E. Both A and sum have spatial locality, and sum also has temporal locality
Locality

- Which description about locality of arrays sum and A in the following code is the most accurate?
```c
for(i = 0; i< 100000; i++)
{
    sum[i%10] += A[i];
}
```

A. Access of A has temporal locality, sum has spatial locality
B. Both A and sum have temporal locality, and sum also has spatial locality
C. Access of A has spatial locality, sum has temporal locality
D. Both A and sum have spatial locality
E. Both A and sum have spatial locality, and sum also has temporal locality
Locality

• Which description about locality of arrays sum and A in the following code is the most accurate?
for(i = 0; i< 100000; i++)
{
    sum[i%10] += A[i];
}

A. Access of A has temporal locality, sum has spatial locality
B. Both A and sum have temporal locality, and sum also has spatial locality
C. Access of A has spatial locality, sum has temporal locality
D. Both A and sum have spatial locality
E. Both A and sum have spatial locality, and sum also has temporal locality
Locality

• Spatial locality — application tends to visit nearby stuffs in the memory
  • Code — the current instruction, and then PC + 4

Most of the time, your program is just visiting a very small amount of data/instructions within a given window

• Temporal locality — application revisit the same thing again and again
  • Code — loops, frequently invoked functions
  • Data — the same data can be read/write many times
Architecting the Cache
Load/store only access a “word” each time

Load 0x000A
To capture “spatial” locality, $ fetch a “block”
How to tell who is there?

Processor Core

 Registers

<table>
<thead>
<tr>
<th>tag</th>
<th>0x000</th>
</tr>
</thead>
<tbody>
<tr>
<td>AABBCDDEEGGFFHH</td>
<td></td>
</tr>
<tr>
<td>IIJKKLLMMNOOPP</td>
<td></td>
</tr>
<tr>
<td>QQRSSSTTUUVVWWXX</td>
<td></td>
</tr>
<tr>
<td>YYZZAABBCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>AABBCDDEEGGFFHH</td>
<td></td>
</tr>
<tr>
<td>IIJKKLLMMNOOPP</td>
<td></td>
</tr>
<tr>
<td>QQRSSSTTUUVVWWXX</td>
<td></td>
</tr>
<tr>
<td>YYZZAABBCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>IIJKKLLMMNOOPP</td>
<td></td>
</tr>
<tr>
<td>QQRSSSTTUUVVWWXX</td>
<td></td>
</tr>
<tr>
<td>YYZZAABBCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>AABBCDDEEGGFFHH</td>
<td></td>
</tr>
<tr>
<td>IIJKKLLMMNOOPP</td>
<td></td>
</tr>
<tr>
<td>QQRSSSTTUUVVWWXX</td>
<td></td>
</tr>
<tr>
<td>QQRSSSTTUUVVWWXX</td>
<td></td>
</tr>
<tr>
<td>YYZZAABBCDDEEFF</td>
<td></td>
</tr>
</tbody>
</table>
The complexity of search the matching tag—$O(n)$—will be slow if our cache size grows!

Can we search things faster? —hash table! $O(1)$

0x404 not found, go to lower-level memory
The biggest issue with hash is — **Collision!**

### Hash-like structure — direct-mapped cache

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 x00</td>
<td>AABBCCDDEEEFGFFHH</td>
<td></td>
</tr>
<tr>
<td>1 x10</td>
<td>IIJKKLMMNNOPPP</td>
<td></td>
</tr>
<tr>
<td>1 xA1</td>
<td>QQRSSTUUVWWWXX</td>
<td></td>
</tr>
<tr>
<td>0 1 x10</td>
<td>YYZAABBCCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>1 1 x31</td>
<td>AABBCCDDEEEFGFFHH</td>
<td></td>
</tr>
<tr>
<td>1 1 x45</td>
<td>IIJKKLMMNNOPPP</td>
<td></td>
</tr>
<tr>
<td>0 1 x41</td>
<td>QQRSSTUUVWWWXX</td>
<td></td>
</tr>
<tr>
<td>0 1 x68</td>
<td>YYZAABBCCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>1 1 x29</td>
<td>IIJKKLMMNNOPPP</td>
<td></td>
</tr>
<tr>
<td>1 1 xDE</td>
<td>QQRSSTUUVWWWXX</td>
<td></td>
</tr>
<tr>
<td>0 1 xCB</td>
<td>YYZAABBCCDDEEFF</td>
<td></td>
</tr>
<tr>
<td>0 1 x8A</td>
<td>AABBCCDDEEEFGFFH</td>
<td></td>
</tr>
<tr>
<td>1 1 x60</td>
<td>IIJKKLMMNNOPPP</td>
<td></td>
</tr>
<tr>
<td>1 1 x70</td>
<td>QQRSSTUUVWWWXX</td>
<td></td>
</tr>
<tr>
<td>1 0 x10</td>
<td>QQRSSTUUVWWWXX</td>
<td></td>
</tr>
<tr>
<td>0 1 x11</td>
<td>YYZAABBCCDDEEFF</td>
<td></td>
</tr>
</tbody>
</table>

Processor Core

**Registers**

**Load 0x000A**

**Load 0x404A**

0x40 not found, go to lower-level memory
Way-associative cache

memory address:

\[ \text{0x0} \quad 8 \quad 2 \quad 4 \]

set block
index offset
tag

Set

V D tag data
1 1 0x29 IIJJKKLLMMNNOOPP
1 1 0xDE QQRRSSSTUUVVWWXX
1 0 0x10 YYYYAAABBCCDDEEFF
0 1 0x8A AABBCCDDEEEFF
1 1 0x60 IIJJKKLLMMNNOOPP
1 1 0x70 QQRRSSSTUUVVWWXX
0 1 0x10 QQRRSSSTUUVVWWXX
0 1 0x11 YYYYAAABBCCDDEEFF

V D tag data
1 1 0x00 AABBCCDDEEGGFFHH
1 1 0x10 IIJJKKLLMMNNOOPP
1 0 0xA1 QQRRSSSTUUVVWWXX
0 1 0x10 YYYYAAABBCCDDEEFF
1 1 0x31 AABBCCDDEEEFF
1 1 0x45 IIJJKKLLMMNNOOPP
0 1 0x41 QQRRSSSTUUVVWWXX
0 1 0x68 YYYYAAABBCCDDEEFF

memory address:

\[ \text{0b0000100000100100} \]

data tag

data

hit?

hit?