Architecture

François Trahay

CSC4508 – Operating Systems
2021–2022
Outlines

1 Introduction .................................................................................................................. 3
2 Sequential processor ...................................................................................................... 5
3 Pipeline .......................................................................................................................... 6
4 Parallel Processing ......................................................................................................... 14
5 Memory hierarchy .......................................................................................................... 19
1 Introduction

Why this lecture?

- To understand what is happening in the “hardware” part of the execution stack
- To write programs that are efficient on modern machines
1.1 Moore’s Law

1965 - 2005

- Moore’s Law (1965): the number of transistors in microprocessors doubles every two years
- The fineness of the processor engraving decreases
- The clock frequency increases

\[ \Rightarrow \text{Increased processor performance} \]

Since 2005

- The fineness of engraving continues to decrease (but less quickly)
- The clock frequency no longer increases
  - Heat dissipation depends on frequency, number of transistors, size of transistors
  - If we reduce the size of the transistors, we have to reduce the frequency
2 Sequential processor

- An instruction requires N steps
  - *Fetch*: load instruction from memory
  - *Decode*: identify the instruction
  - *Execute*: execution of the instruction
  - *Writeback*: storage of the result

- Each step is processed by a processor circuit
- Most circuits are not used at every stage

→ One instruction is executed every N cycles
3 Pipeline

- Instruction pipeline
- At each stage, several circuits are used
- One instruction is executed at each cycle

![Figure 1: Execution of instructions on a processor with pipeline](image-url)

Télécom SudParis — François Trahay — 2021–2022 — CSC4508 – Operating Systems
3.1 Micro architecture of a pipeline

- Each stage of the pipeline is implemented by a set of logic gates
- Execute step: one subcircuit per type of operation (functional unit)

![Diagram of pipeline stages](image)

Figure 2: Micro-architecture of a pipeline
3.2 Superscalar processors

- Use of different functional units simultaneously
  \[\Rightarrow\] several instructions executed simultaneously!
- Require to load and decode several instructions simultaneously

Figure 3: Micro-architecture of a superscalar processor
3.2.1 Superscalar processors

![Superscalar processors diagram]
3.2.2 Dependence between instructions

Limitations of the superscalar:

- There should be no dependency between statements executed simultaneously.
  - Example of non-parallelizable instructions
    
    ```
    a = b * c;
    d = a + 1;
    ```

- Degree of parallelism of the instructions: *Instruction Level Parallelism* (ILP)

- Instructions executed in parallel must use different functional units
3.3 Branching

- How to fill the pipeline when the instructions contain conditional jumps?

  ```
  cmp a, 7  ; a > 7 ?
  ble L1
  mov c, b  ; b = c
  br L2
  L1: mov d, b  ; b = d
  L2: ...
  ```

- In case of a bad choice: the pipeline must be “emptied”

  ⇒ waste of time
3.4 Branch prediction

The processor implements a prediction algorithm.

General idea:

- For each conditional jump, store the previous results.

0x12 loop:

```
...  
0x50  inc eax
0x54  cmpl eax, 10000
0x5A  jl loop
0x5C  end_loop:
```

<table>
<thead>
<tr>
<th>addr</th>
<th>branch history</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x23</td>
<td>0011</td>
</tr>
<tr>
<td>0x42</td>
<td>1000</td>
</tr>
<tr>
<td>0x5A</td>
<td>1111</td>
</tr>
<tr>
<td>0x7E</td>
<td>0000</td>
</tr>
</tbody>
</table>
3.5 Vector instructions

- Many applications run in *Data Parallelism* mode

- *Single Instruction, Multiple Data* (SIMD): the same operation applied to a set of data

  ```
  for(i=0; i<size; i++) {
      C[i] = A[i] * B[i];
  }
  ```

- Example: image processing, *scientific computing*

- Using *vector instructions* (MMX, SSE, AVX, ...)
  - Instructions specific to a processor type
  - Process the same operation on multiple data at once

  ```
  for(i=0; i<size; i+= 8) {
      *pC = _mm_mul_ps(*pA, *pB);
      pA++; pB++; pC++;
  }
  ```
4 Parallel Processing

4.1 Hyperthreading / SMT ................................................................. 15
4.2 Multi-core processors ................................................................. 16
4.3 SMP architectures ........................................................................ 17
4.4 NUMA architectures ..................................................................... 18
4.1 Hyperthreading / SMT

- Problem with superscalar / vector processors:
  - The application must have enough parallelism to exploit
  - Other applications may be waiting for the CPU

- Simultaneous Multi-Threading (SMT, or Hyperthreading):
  - Modify a superscalar processor to run multiple threads
  - Duplicate some circuits
  - Share certain circuits (eg FPU) between processing units

```
Thread A
  fetch
  decode & dispatch
  ALU (int)
  Mem/Branch

Shared
  FPU (float)
  writeback

Thread B
  ALU (int)
  Mem/Branch
```
4.2 Multi-core processors

- Limited scalability of SMT
  ♦ *dispatcher* is shared
  ♦ FPU is shared

→ Duplicate all the circuits
4.3 SMP architectures

- **Symmetric Multi-Processing**
- Multiple processors sockets on a motherboard
- The processors share the system bus
- Processors share memory
- Scalability problem: contention when accessing the bus
4.4 NUMA architectures

- NUMA nodes connected by a fast network
- Memory consistency between processors
- Privileged access to the local *memory bank*
- Access possible (with an additional cost) to *memory banks* located on other nodes

→ *Non-Uniform Memory Architecture*
5 Memory hierarchy
5.1 Motivation

- Until 2005: increase in CPU performance: 55 % / year
- Since 2005: increase in the number of cores per processor
- Increased memory performance: 10 % / year
- The memory accesses which are now expensive: Memory Wall
- Mechanisms are needed to improve memory performance
5.2 Cache memory

- Memory access (RAM) are very expensive (approx. 60 ns - approx. 180 cycles)
- To speed up memory access, let’s use a fast cache memory:
  - L1 cache: very small capacity (typically: 64 KiB), very fast (approx. 4 cycles)
  - L2 cache: small capacity (typical: 256 KiB), fast (approx. 10 cycles)
  - L3 cache: large capacity (typically: between 4 MiB and 30 MiB), slow (approx. 40 cycles)
- Very expensive hard disk access (SWAP): approx. 40 ms (150 $\mu$s on an SSD disk)
5.3 Memory Management Unit (MMU)

- Translates virtual memory addresses into physical addresses
- Look in the TLB (*Translation Lookaside Buffer*), then in the page table
- Once the physical address is found, request the data from the cache / memory
5.3.1 Fully-associative caches

- Cache = array with N entries
- For each reference, search for Tag in the array
  - If found \((cache \ hit)\) and Valid = 1: access to the cache line Data
  - Otherwise \((cache \ miss)\): RAM access
- Problem: need to browse the whole table
  - Mainly used for small caches (ex: TLB)
5.3.2 Direct-mapped caches

- Using the least significant bits of the address to find the index of the entry in the cache
- Comparison of the Tag (most significant bits) of the address and the entry.

→ Direct access to the cache line

- Warning: risk of collision
  - example: \(0x12345678\) and \(0xbff72678\)
### 5.3.3 Set-associative caches

- **Index** to access a set of $K$ cache lines
- **Search for the Tag** among the addresses of the set

→ **K-way associative cache** (in French: *Cache associatif K-voies*)
5.3.4 Cache consistency

What if 2 threads access the same cache line?

♦ Concurrent read: replication in local caches

♦ Concurrent write: need to invalidate data in other caches

▷ Cache snooping: the cache sends a message that invalidates the others caches
Bibliography