Processor Performance and Parallelism
Y. K. Malaiya
**Processor Execution time**

The time taken by a program to execute is the product of:

- Number of machine instructions executed
- Number of clock cycles per instruction (CPI)
- Single clock period duration

Clock Cycles = Instruction Count × Cycles per Instruction

CPU Time = Instruction Count × CPI × Clock period

**Example:** 10,000 instructions, CPI=2, clock period = 250 ps

\[
\text{CPU Time} = 10,000 \text{ instructions} \times 2 \times 250 \text{ ps}
\]

\[
= 10^4 \times 2 \times 250 \times 10^{-12} = 5 \times 10^{-6} \text{ sec.}
\]
Processor Execution time

Instruction Count for a program

- Determined by program, ISA and compiler

Average Cycles per instruction (CPI)

- Determined by CPU hardware
- If different instructions have different CPI
  - Average CPI affected by instruction mix

Clock cycle time (inverse of frequency)

- Logic levels
- Technology

CPU Time = Instruction Count $\times$ CPI $\times$ Clock Cycle Time
Reducing clock cycle time

- Has worked well for decades.
- Small transistor dimensions implied smaller delays and hence lower clock cycle time.
- Not any more.
CPI (cycles per instruction)

What is LC-3 cycles per instruction?

Instructions take 5-9 cycles (p. 568), assuming memory access time is one clock period.

- LC-3 CPI may be about 6*. (ideal)

No cache, memory access time = 100 cycles?

- LC-3 CPI would be very high.

Cache reduces access time to 2 cycles.

- LC-3 CPI higher than 6, but still reasonable.
Parallelism to save time

Do things in parallel to save time.

Approaches:

• Instruction level parallelism
  ➢ Pipelining: Divide flow into stages. Let instructions flow into the pipeline.
  ➢ Multiple issue: Fetch multiple instructions at the same time

• Concurrent processes or thread (Task-level parallelism)
  ➢ For true concurrency, need extra hardware
    – Multiple processors (cores) or
    – support for multiple thread

Demo: Threads in Mac
Pipelining Analogy

Pipelined laundry: overlapping execution

- Parallelism improves performance

- Four loads:
  - time
  = 4x2 = 8 hours

- Pipelined:
  - Time in example
  = 7x0.5 = 3.5 hours
  - Non-stop
  = 4x0.5 = 2 hours.
Pipeline Processor Performance

Single-cycle ($T_c = 800\text{ps}$)

Program execution order (in instructions)

- lw $1$, 100($0$)
- lw $2$, 200($0$)
- lw $3$, 300($0$)

Pipelined ($T_c = 200\text{ps}$)

Program execution order (in instructions)

- lw $1$, 100($0$)
- lw $2$, 200($0$)
- lw $3$, 300($0$)
Pipelining: Issues

- Cannot predict which branch will be taken.
  - Actually you may be able to make a good guess.
  - Some performance penalty for bad guesses.

- Instructions may depend on results of previous instructions.
  - There may be a way to get around that problem in some cases.
Instruction level parallelism (ILP):

Pipelining is one example.

Multiple issue: have multiple copies of resources

- Multiple instructions start at the same time
- Need careful scheduling
  - Compiler assisted scheduling
  - Hardware assisted ("superscaler"): "dynamic scheduling"
    - Ex: AMD Opteron x4
    - CPI can be less than 1!
Task Parallelism

Program is divided into tasks that can be run in parallel

- **Concurrent Processes**
  - Can run truly in parallel if there are multiple processors, e.g. multi-core processors

- **Concurrent Threads**
  - Multiple threads can run on multiple processors, or
  - Single processor with multi-threading support (Simultaneous Multithreading)

- **Process vs thread**
  - All information resources for a process are private to the process.
  - Multiple threads within a process have private registers & stack, but not address space.
Task Parallelism

Program is divided into tasks that can be run in parallel

Example: A program needs subtasks A, B, C, D. B and C can be run in parallel. They each take 200, 500, 500 and 300 nanoseconds.

- Without parallelism: total time needed = 200 + 500 + 500 + 300 = 1500 ns.
- With Task level parallelism: 200 + 500 (B and C in parallel) + 300 = 1000 ns.
Task Parallelism
Flynn’s taxonomy

Michael J. Flynn, 1966

<table>
<thead>
<tr>
<th>Instruction Streams</th>
<th>Single</th>
<th>Multiple</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single</td>
<td>SISD: Intel Pentium 4</td>
<td>SIMD: MMX/SSE instructions in x86</td>
</tr>
<tr>
<td>Multiple</td>
<td>MISD: No examples today</td>
<td>MIMD: eg. Multicore Intel Xeon e5345</td>
</tr>
</tbody>
</table>

- Instruction level parallelism is still SISD
- SSE (Streaming SIMD Extensions): vector operations
- Intel Xeon e5345: 4 cores
- *Does not model Instruction level/task level parallelism*
Multi what?

- **Multitasking**: tasks share a processor
- **Multithreading**: threads share a processor
- **Multiprocessors**: using multiple processors
  - For example multi-core processors (multiples processors on the same chip)
  - Scheduling of tasks/subtasks needed
Multi-core processors

- Power consumption has become a limiting factor
- Key advantage: lower power consumption for the same performance
  - Ex: 20% lower clock frequency: 87% performance, 51% power.
- A processor can switch to lower frequency to reduce power.
- N cores: can run n or more threads.
Multi-core processors

Cores may be identical or specialized
Higher level caches are shared.
Lower level *cache coherency* required.
Cores may use superscalar or simultaneous multi-threading architectures.
LC-3 states

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD, AND, NOT, JMP</td>
<td>5</td>
</tr>
<tr>
<td>TRAP</td>
<td>8</td>
</tr>
<tr>
<td>LD, LDR, ST, STR</td>
<td>7</td>
</tr>
<tr>
<td>LDI, STI</td>
<td>9</td>
</tr>
<tr>
<td>BR</td>
<td>5, 6</td>
</tr>
<tr>
<td>JSR</td>
<td>6</td>
</tr>
</tbody>
</table>