EE382N (20): Computer Architecture - Parallelism and Locality

Lecture 5 – Parallelism in Hardware

Mattan Erez

The University of Texas at Austin
Outline

• Principles of parallel execution
• Pipelining
• Parallel HW (multiple ALUs)
Parallel Execution

• Concurrency
  – what are the multiple resources?

• Communication
  – and storage

• Synchronization
  – Implicit?
  – Explicit?

• what is being shared?

• What is being partitioned?
Parallelism – Circuits

- Circuits operate concurrently (always)
- Communicate through wires and registers
- Synchronize by:
  - clock
  - signals (including async)
  - design (wave-pipelined)
  - more?
Parallelism – Components

• Collections of circuits
  – memories
  – branch predictors
  – cache
  – scheduler ...
  – includes ALUs, memory channels, cores but will discuss later.

• Operate concurrently
• Communicate by wires, registers, and memory
• Synchronize with clock and signals
• Often pipelined
Reminders

• This is not a microarchitecture class
  – We will be discussing microarch. of various stream processors though
    • Details deferred to later in the semester

• This class is not a replacement for Parallel Computer Architecture class
  – We only superficially cover many details of parallel architectures
  – Focus on parallelism and locality at the same time
Outline

• Cache oblivious algorithms “sub-talk”
• Principles of parallel execution
• Pipelining
• Parallel HW (multiple ALUs)
  – Analyze by shared resources
  – Analyze by synch/comm mechanisms
  – ILP, DLP, and TLP organizations
Simplified view of a processor

- **fetch**
- **decode**
- **sequencer**

**dispatch** (issue)

**reg. access**
- **execute**
- **write-back**
- **commit**

**memory hierarchy**
Simplified view of a pipelined processor
Simplified view of a pipelined processor

1: add r4, r1, r2
2: add r5, r1, r3
3: add r6, r2, r3
Simplified view of a pipelined processor

1: add r4, r1, r2
2: add r5, r1, r3
3: add r6, r2, r3
Simplified view of a pipelined processor

1: add r4, r1, r2
2: add r5, r1, r3
3: add r6, r2, r3
Simplified view of a pipelined processor

fetch
decode

sequencer

dispatch (issue)

reg. access
execute
write-back
commit

1: add r4, r1, r2
2: add r5, r1, r3
3: add r6, r2, r3

What are the parallel resources?
Simplified view of a pipelined processor

1: add r4, r1, r2
2: add r5, r1, r4
3: add r6, r5, r3
Simplified view of a pipelined processor

1: add r4, r1, r2
2: add r5, r1, r4
3: add r6, r5, r3
Simplified view of a pipelined processor

- **fetch**
- **decode**
- **sequencer**
- **dispatch (issue)**
- reg. access
- execute
- write-back
- commit

1: add r4, r1, r2
2: add r5, r1, r3
3: add r6, r2, r3

Communication and synchronization mechanisms?
Simplified view of a pipelined processor

1: add r4, r1, r2
2: ld r5, r4
3: add r6, r5, r3
4: add r7, r1, r3

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>D</th>
<th>I</th>
<th>R</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>W</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></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>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>R</td>
<td>E</td>
</tr>
</tbody>
</table>
Simplified view of a OOO pipelined processor

Communication and synchronization mechanisms?
Pipelining Summary

• Pipelining is using parallelism to hide latency
  – Do useful work while waiting for other work to finish

• Multiple parallel components, not multiple instances of same component

• Examples:
  – Execution pipeline
  – Memory pipelines
    • Issue multiple requests to memory without waiting for previous requests to complete
  – Software pipelines
    • Overlap different software blocks to hide latency: computation/communication
Resources in a parallel processor/system

• Execution
  – ALUs
  – Cores/processors

• Control
  – Sequencers
  – Instructions
  – OOO schedulers

• State
  – Registers
  – Memories

• Networks
Communication and synchronization

• Synchronization
  – Clock – explicit compiler order
  – Explicit signals (e.g., dependences)
  – Implicit signals (e.g., flush/stall)
    • More for pipelining than multiple ALUs

• Communication
  – Bypass networks
  – Registers
  – Memory
  – Explicit (over some network)
Organizations for ILP (for multiple ALUs)
Superscalar (ILP for multiple ALUs)

- Synchronization
  - Explicit signals (dependences)
- Communication
  - Bypass, registers, mem
- Shared
  - Sequencer, OOO, registers, memories, net, ALUs
- Partitioned
  - Instructions

How many ALUs?
SMT/TLS (ILP for multiple ALUs)
SMT/TLS (ILP for multiple ALUs)

- Synchronization
  - Explicit signals (dependences)
- Communication
  - Bypass, registers, mem
- Shared
  - OOO, registers, memories, net, ALUs
- Partitioned
  - Sequencer, Instructions, arch. registers

Why is this ILP? How many threads?
VLIW (ILP for multiple ALUs)
VLIW (ILP for multiple ALUs)

- Synchronization
  - Clock + compiler
- Communication
  - Registers, mem, bypass
- Shared
  - Sequencer, OOO, registers, memories, net
- Partitioned
  - Instructions, ALUs

How many ALUs?
Explicit Dataflow (ILP for multiple ALUs)
Explicit Dataflow (ILP for multiple ALUs)

- **Synchronization**
  - Explicit signals
- **Communication**
  - Registers+explicit
- **Shared**
  - Sequencer, memories, net
- **Partitioned**
  - Instructions, OOO, ALUs

memory hierarchy
DLP for multiple ALUs

From HW – this is SIMD
SIMD (DLP for multiple ALUs)

- Synchronization
  - Clock+compiler

- Communication
  - Explicit

- Shared
  - Sequencer, instructions

- Partitioned
  - Registers, memories, ALUs
  - Sometimes: memories, net
Vectors (DLP for multiple ALUs)

Vectors: memory addresses are part of single-instruction and not part of multiple-data
TLP for multiple ALUs
MIMD – shared memory (TLP for multiple ALUs)

- **Synchronization**
  - Explicit, memory
- **Communication**
  - Memory
- **Shared**
  - Memories, net
- **Partitioned**
  - Sequencer, instructions, OOO, ALUs, registers, some nets

**Diagram:**

```
sequencer -> scheduler

sequencer -> scheduler

sequencer -> scheduler

sequencer -> scheduler

memory hierarchy
```
MIMD – distributed memory

- Synchronization
  - Explicit
- Communication
  - Explicit
- Shared
  - Net
- Partitioned
  - Sequencer, instructions, OOO, ALUs, registers, some nets, memories

sequencer

scheduler

memory hierarchy

scheduler

memory hierarchy

scheduler

memory hierarchy

scheduler

memory hierarchy
## Summary of communication and synchronization

<table>
<thead>
<tr>
<th>Style</th>
<th>Synchronization</th>
<th>Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLIW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataflow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIMD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Summary of sharing in ILP HW

<table>
<thead>
<tr>
<th>Style</th>
<th>Seq</th>
<th>Inst</th>
<th>OOO</th>
<th>Regs</th>
<th>Mem</th>
<th>ALUs</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMT/TLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VLIW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dataflow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EE38N(20) Spring 2015 -- Lecture 5  (c) Mattan Erez**
## Summary of sharing in DLP and TLP

<table>
<thead>
<tr>
<th>Style</th>
<th>Seq</th>
<th>Inst</th>
<th>OOO</th>
<th>Regs</th>
<th>Mem</th>
<th>ALUs</th>
<th>Net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>