Parallel architectures

Prof Rajeev Barua

E-A 001 -- Slide set 5

Instruction level parallelism (ILP)

ILP is when instructions from the same serial thread of computation are run in parallel with each other on different functional units.

Two types of architectures exploit ILP:

- Superscalar processors
- VLIW processors

ILP examples

Example of ILP when no data dependence:

```
    add $2, $1, $8
    sub $4, $3, $11
```

Since the above instructions have no data dependence, we can execute them in parallel.

Example of ILP when control dependence can be resolved:

```
        beqz $2, L1
        xor $4, $3, $11
        …
L1:     and $4, $6, $7
```

In the above code, once we have predicted the branch, we can execute the branch at the same time as the target. This is also an example of ILP.

Superscalar processors

PLAKSHA TECH LEADERS FELLOWSHIP

A superscalar processor is one in which more than one instruction is fetched per cycle in a one pipeline.
EG: a 3-way superscalar can issue up to 3 instructions/cycle

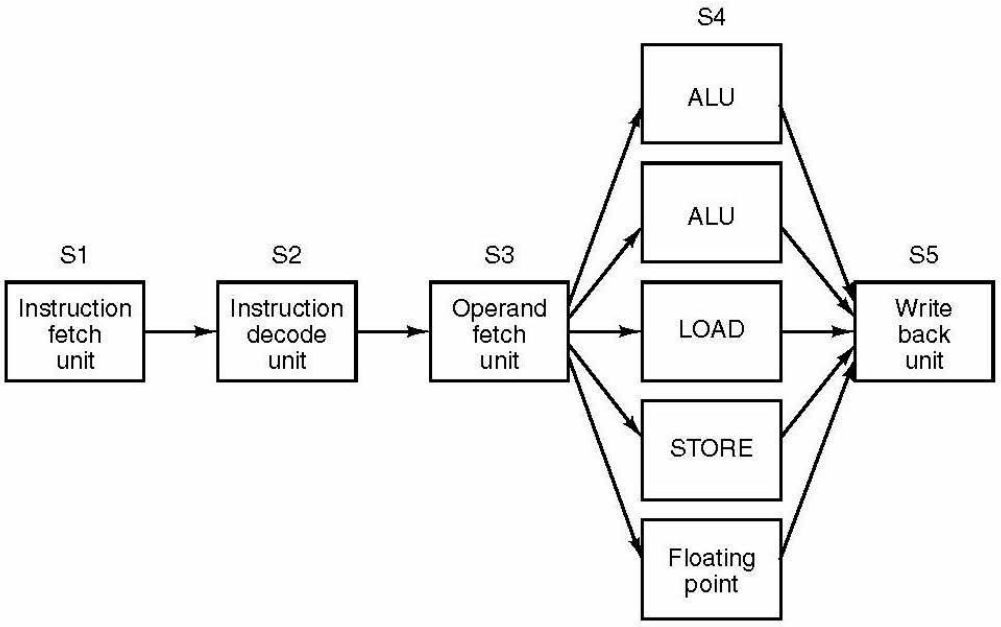

Figure 2-6. A superscalar processor with five functional units.

Complications:

Instructions in the same cycle:

- Cannot be dependent
- Cannot access same resource in same cycle (for example, memory)

⇒ Hardware does complex dependence checking, and does out-of-order execution.
⇒ Significant cost in silicon area and power
⇒ Done for desktops, laptops, & servers. Unaffordable for embedded systems.

VLIW processors

Very Long Instruction Word (VLIW) processors:

- Similar to superscalar, except that dependence-freedom ensured by the compiler.
  - Simpler hardware, but more complex compiler.
  - Used in embedded systems because of lower power consumption.
  - Also in failed Intel EPIC architectures.
  - Today, VLIWs are widely used for the digital signal processors (DSP) cores of mobile devices, but not in the application processor.

VLIW processors

The machine code for a 2-way VLIW is shown below:

	Slot 1	Slot 2
Cycle 1	`Add $5, $4, $3`	`Xor $4, $2, $7`
Cycle 2	`Sub $7, $2, $8`	`Nop`
Cycle 3	`Xor ...`	`Beqz $3, _target`
...		

- Instructions in one cycle execute in parallel (at the same time).
- Instructions read registers and memory state from before the cycle.
  - Eg: in cycle 1, `$4` is both read (old value) and written (new value).
  - This is why there is no data dependence on `$4` in Cycle 1.
- Sometimes Nops are needed if not enough non-data-dependent instructions.
- Only one instruction per cycle can be a control transfer instruction (CTI).
  - Control transfer takes effect after cycle.

Other types of parallelism

Other types of parallelism exploit larger amounts of parallelism than instructions within a single thread.

Types:

- Array processors
- Vector processors
- Multiprocessors

Array processors

Also known as Single Instruction, Multiple Data (SIMD) architectures.

- Same instruction is executed on each tile, but with different data values across tiles.
- Register file has many values per register instead of one.
- Memory is shared but high-bandwidth.
- Only useful for programs with high amounts of array-based parallelism, eg, scientific and multi-media programs.
  - Not useful for other programs.

These are no longer popular for general purpose machines, but used in special-purpose machines like GPUs and AI chips.

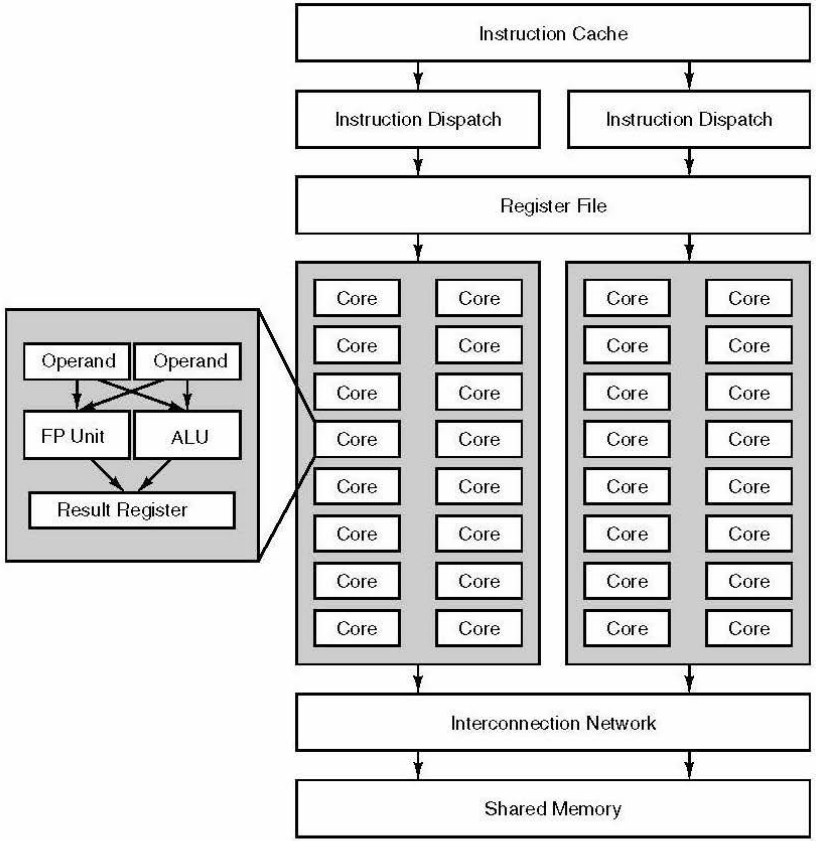

Figure 2-7. The SIMD core of the Fermi graphics processing unit.

Vector processors

Identical programming to array processors, but use different hardware.

- Vector processors use single, heavily-pipelined functional units.
  - Instead of lots of functional units.
  - These break up a single add or multiply ALU into multiple stages (pipeline parallelism within EX!)
  - Why are they fast?
    - Pipelining is done across array elements instead of across instructions
    - Lack of dependence checking.
    - Very fast pipeline stages.
    - Pipeline stages other than EX are run only once per vector.
- Better hardware utilization than array processors.
- Modern GPUs use either or both of array and vector processing ideas.

Multiprocessors and Multicores

- A multiprocessor has the ability to run multiple independent threads on different independent processors, each on one silicon chip.
  - ➢ A multicore does the same thing on multiple “cores” within a processor on a single silicon chip.
  - ➢ Architecturally, multiprocessors and multicores are essentially the same thing, except the former uses multiple silicon chips, whereas the latter uses one silicon chip.
  - ➢ Also called Multiple Instruction Multiple Data (MIMD) architectures.
- Most modern processors are multicores. Eg. Intel Core i9 (desktops and laptops) or Xeon (servers).
- Larger machines called multiprocessors group several multicores together using a very fast interconnection network.
  - Can have tens, hundreds, thousands, or hundreds of thousands of cores.

Multiprocessor memory organizations

- A multiprocessor may have one of two memory organizations:
  - Centralized memory (shown on the left in figure below)
    - When all the memory is in one place, and is shared by CPUs via a bus
  - Distributed memory. (shown on the right. Shared portion may or may not exist.)
    - When memory is distributed across CPUs.

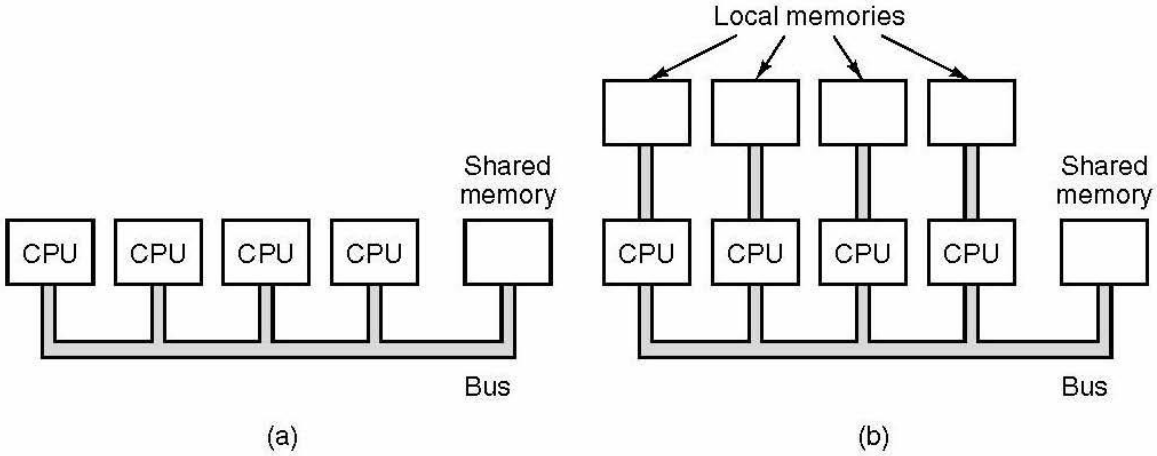

Figure 2-8. (a) A single-bus multiprocessor. (b) A multicomputer with local memories.

Multiprocessor programming models

Not the same thing as memory organizations!

Programming models refer to the view of memory from programs running on the multiprocessor. Two kinds of programming models:

a. Shared memory
   - ➢ Any processor can directly address any memory location.
   - ➢ could be on centralized or distributed memory organization!! (Latter case is called Distributed Shared Memory (DSMs).
   - ➢ Advantage: Easier programming than with message passing. (Harder than a single core programming). But harder to build DSMs.

b. Message passing
   - ➢ Processors can only directly address their own local memory's locations.
   - ➢ Implies distributed memory organization.
   - ➢ Accessing a remote memory requires the programmer to explicitly send a message on interconnection network.
   - ➢ Advantage: hardware is simpler and can scale to more cores. No need for hardware to provide unified view of memory, or memory consistency (consistent view of caches).