Lecture 1 Notes

Parallel Processing CS471

1 Parallel Processing Classifications [1]

One way to classify parallel systems can be classified by the number of instructions and data items that are manipulated simultaneously. This classification is known as 'Flynn's Taxonomy'.

Note 1. While classified as different architectures a single computer may exhibit several of these architectures. E.g. most computers have a multicore processor (MIMD) as well as a graphics processor (GPU) (SIMD).

1.1 Single Instruction Stream, Single Data Stream (SISD)

A sequential computer with no parallelism. A single control unit fetches a single instruction that is processed by a single processing unit on a single data stream.

Cheap, low power microcontrollers might be designed this way. SISD systems are very limited in terms of performance.

1.2 Single Instruction Stream, Multiple Data Streams (SIMD)

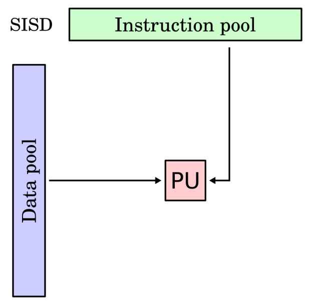
SIMD describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously. SIMD computers utilize data-level parallelism.

SMID computers are designed to perform the same operation on many datapoints. This makes them useful for multimedia processing (image, audio) as well as rendering graphics. GPUs are a common example of SIMD systems.

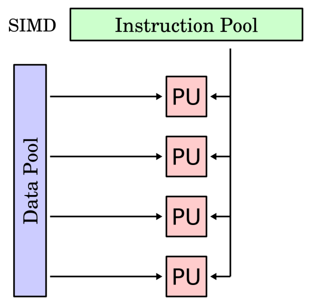
1.3 Multiple Instruction Streams, Multiple Data Streams (MIMD)

Machines using MIMD have a number of processor cores that function asynchronously and independently. At any time, different processors may be executing different instructions on different pieces of data. MIMD systems target task-level parallelism.

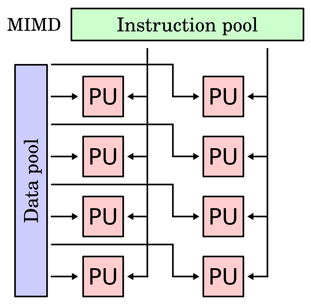
Most multiprocessor and multicomputer systems are considered MIMD. MIMD systems are useful for applications like modeling, simulation, and computer aided design (CAD).



(a) SISD.[2]



(b) SIMD. [3]



(c) MIMD. [4]

Figure 1: Visualization of classifications in Flynn's Taxonomy.

Other Parallelism Classes

While Flynn's classes are useful when describing computer architecture, sometimes we're interested in describing parallelism based on the operations and interconnections being made.

In terms of application we have two kinds of parallelism:

- **Data-Level Parallelism** - Many data items being operated on at once.
- **Task-Level Parallelism** - Many tasks being operated on at once.

Application parallelism is usually exploited in the following ways:

- **Instruction-Level Parallelism** - Exploits data-level parallelism at modest levels with compiler help. Pipelining is considered instruction-level.
- **Vector Architectures and Graphic Processor Units (GPUs)** - Exploit data-level parallelism by applying a single instruction to a collection of data in parallel.
- **Thread-Level Parallelism** - Exploits either data-level parallelism or task-level parallelism in a tightly coupled hardware model that allows for interaction among parallel threads.
- **Request-Level Parallelism** - Exploits parallelism among largely decoupled tasks specified by the programmer or the operating system.

2 Pipelining [1]

2.1 Pipelining

Definition 1. Pipelining - An implementation technique whereby multiple instructions are overlapped in execution.

Pipelining breaks down a sequantial process into suboperations, with each being executed in a 'segment' (or 'stage') concurrently with other segments. You can imagine an instruction entering one end of the pipeline, progressing through the different segments, and exiting at the other end.

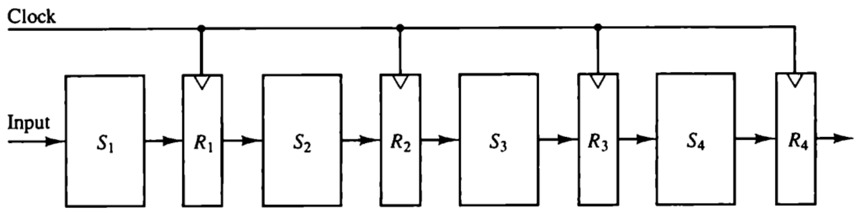
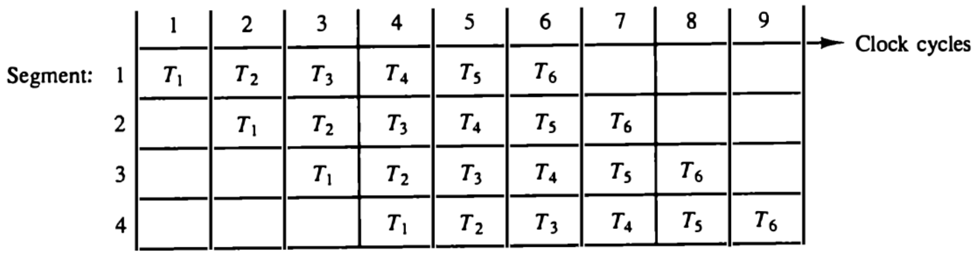


Figure 2: Illustration of a 4 segment pipeline. Each segment S$_n$ is separated by a register R$_n$. Registers store intermediate results between segments

Pipelines are better visualized with a space-time diagram.



Segment:	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

Clock cycles

Figure 3: Space-time diagram of a pipeline with 4 segments and 6 tasks. Notice how the in the beginning we wait for 4 clock cycles to get an output from one of our tasks. However, once the pipeline is full, we obtain output from a task every clock cycle.

2.2 Speedup

We denote the time taken by a non-pipelined execution of a task t_n. For a k segment pipeline with a clock cycle time of t_p used to execute n tasks, we consider the speedup of pipeline processing to be defined

$$S = \frac{nt_n}{(k+n-1)t_p} \tag{1}$$

As the number of tasks increases ($n >> k-1$), $k+n-1$ approaches the value of n. This simplifies our ratio to

$$S = \frac{t_n}{t_p}$$

Since time it takes to process a task is the same for pipelined and non-pipelined approaches

$$t_n = kt_p \tag{2}$$

$$S = \frac{kt_p}{t_p} = k \tag{3}$$

This means that the maximum speedup that a pipeline can provide is k (the number of segments in the pipeline).

Pipelining Example

For a pipeline with $k = 4$ segments executing $n = 100$ tasks, assuming it takes $t_p = 20$ ns to process a sub-operation in a segment, we first find t_n using Equation 2:

$$t_n = (4)(20) = 80,$$

then we can find the speedup ratio using Equation 1:

$$S = \frac{(100)(80)}{(4+99)(20)} = \frac{8000}{2060} = 3.88,$$

$$\lim_{n\to\infty} \frac{n(80)}{(4+99)(20)} = 4.$$

2.3 Considerations

There are various reasons why the pipeline cannot operate at its maximum theoretical rate.

- Different segments may take different times to complete their sub-operation. This causes all other segments to waste time while waiting for the next clock.
- In Equation 1 we make the assumption that a non-pipelined circuit has the same time delay as a pipelined circuit. This does not take into account the delay generated by the use of registers.

Despite not achieving maximum theoretical speed pipelining provides a large benefit over serial sequential processing.

References

[1] M Morris Mano. Pipeline and vector processing. In *Computer System Architecture*, chapter 9, pages 299–331. Pearson, Upper Saddle River, NJ, 3rd edition, oct 1992.

[2] Wikipedia contributors. Single instruction, single data — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Single_instruction,_single_data&oldid=1170073255`, 2023. [Online; accessed 22-December-2024].

[3] Wikipedia contributors. Single instruction, multiple data — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Single_instruction,_multiple_data&oldid=1232816966`, 2024. [Online; accessed 22-December-2024].

[4] Wikipedia contributors. Multiple instruction, multiple data — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Multiple_instruction,_multiple_data&oldid=1235681846`, 2024. [Online; accessed 22-December-2024].