0.1 parallel computing

0.1.1 What is parallel computing

The problem when facing large calculations is that they require a lot of computing power and time. Performing these calculations can be done in different types of computation but the most common ones are serial and parallel computation. Serial computing means, you have one compute unit (e.g. a single core CPU) available to deal with all the calculations that have to be done on a certain set of data. The problem will be broken into multiple smaller subproblems that will be solved by a certain instruction. The single compute unit has to perform the instruction on every subproblem to solve the main problem as shown in figure 1.

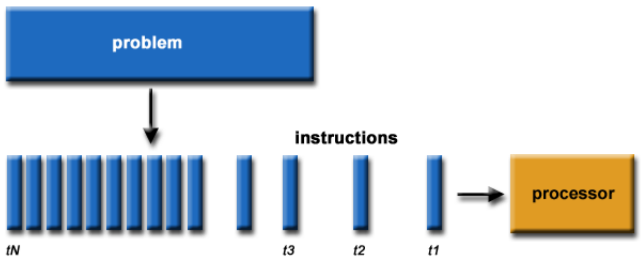


Figure 1: Workflow serial computing

Parallel computing, on the other hand, is the simultaneous use of multiple compute units (CPU+GPU), or a compute resource, to solve a computational problem. We break apart the main problem in smaller subproblems as we did with serial computing. Each part is further broken down to a series of instructions again.Now, since there are multiple compute units, we can distribute the subproblems among all these compute units. Every unit can now perform the instruction on their given subproblems simultaneously as shown in figure 2. One compute resource can constist of multiple compute units, for example it can include multiple processors or it's just a number of computers connected by a network.
With multiple compute units for one task, we will shorten the completion time aswell have potential cost savings. It also allows us to solve larger/complex problems since a single computer could suffer from limited memory. And last, we are able to access non-local resources in a network that wouldn't be accessable from a local computer.
It is easy to conclude that the concept of parallel computing was to have a more efficient way to handle with large sets of data such as images or simulations that involve multiple parameters. It's also easier to handle with complex data for example in algorithms [1].

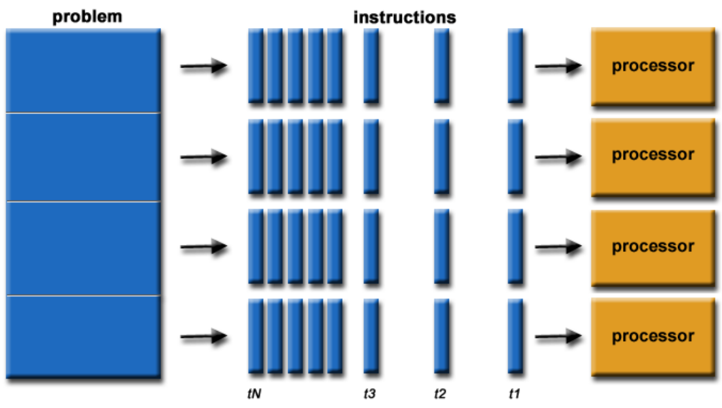


Figure 2: Workflow parallel computing

0.1.2 Classification of parallel computing

Parallel computing systems can be seperated into different classes. According to Flynn's taxonomy, we can roughly place any of these systems in one of the four classes he defined. This classification was first studied and proposed by Michael Flynn in 1972 [2]. The classifications are determined by two factors: instruction stream and data stream which both have two possible states being single or multiple. Figure 3 represents a matrix of the four possible classes from the Flynn's taxonomy [1].

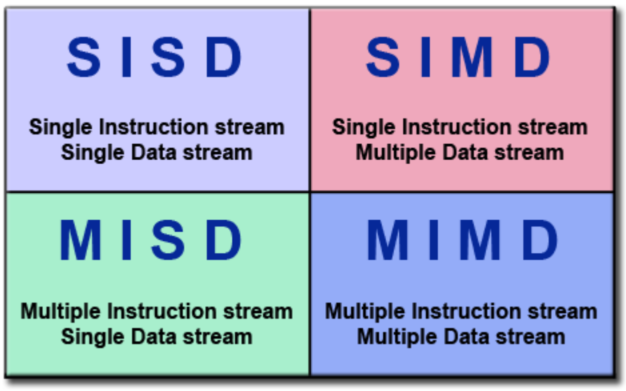

SISD
Single Instruction stream
Single Data stream
SIMD
Single Instruction stream
Multiple Data stream
MISD
Multiple Instruction stream
Single Data stream
MIMD
Multiple Instruction stream
Multiple Data stream

Figure 3: Four possible classifications according to Flynn's Taxonomy

- SISD

The SISD class will have a single processor executing a single data stream to operate on data stored in a single memory (figure 4a). This means that a parallel compute system cannot be classified as an SISD sytem, but Flynn's taxonomy wasn't made for just classifying parallel compute systems. It's usually old computers and other older compute units that can be classified as SISD.

- SIMD

Data is disctributed amongst multiple processors who all execute the same instruction (figure 4b). Since we have access to multiple compute units, parallel computing can be categorized in this class. Furthermore, this is the class where we can categorize our subject of the thesis in since we have a large set of data divided over multiple data streams (MD) and only one instruction stream (SI) since all processing units will perform the same instruction on the data. The SIMD class contains most modern computers, particularly those with a graphics processor unit (GPU).

- MISD

Each processing unit operates on the data independently via separate instruction streams while a single data stream is fed into multiple processing untis (figure 4c). This class knows very few applications. An example of an application is the use of multiple cryptography algorithms attempting to crack a single coded message.

- MIMD

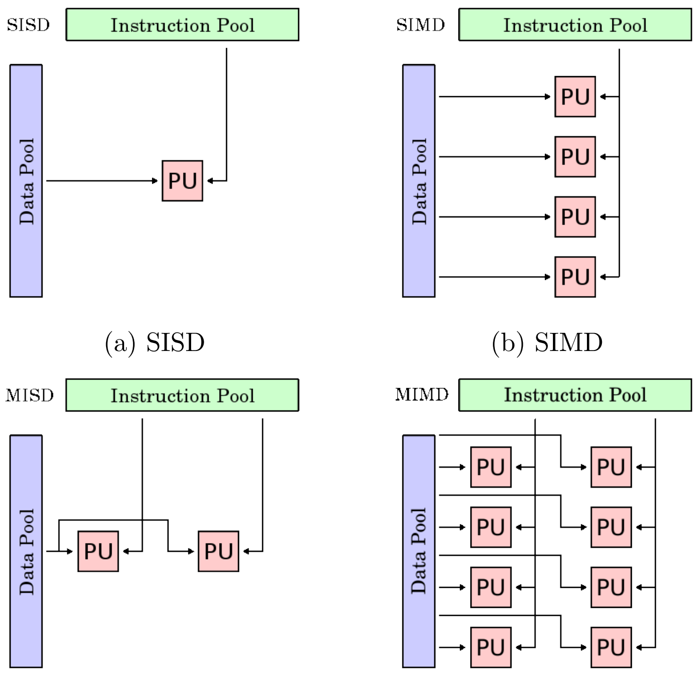
This time every processing unit is able to execute a different instruction stream on a different data stream (figure 4d). This means that any instruction can be applied on any data package for every compute unit. Most supercomputers, networked parallel computer clusters and "grids" can be classified as a MIMD compute system. Also, many MIMD architectures include SIMD execution sub-components.

0.1.3 Amdahl's law

Amdahl's Law can be used to theoretically calculate how much a computation can be sped up by running part of it in parallel, versus using only one serial processor [3]. A program can be split up in two parts, a part that cannot be parallelized (A) and a part that can be parallelized (B). The speedup factor S for both parts can be calculated by dividing the execution time for serial computation T(1) by the execution time for parallel computation T(j), with j the amount of processing units.

$$S = \frac{T(1)}{T(j)} \tag{1}$$

Since A is a serial computation, the amount of processing units j equals 1. And thus the speedup factor for A is also 1. Part A can only be executed



(a) SISD (b) SIMD

(c) MISD (d) MIMD

Figure 4: Architecture classes from Flynn's taxonomy

faster by optimizing the code. Let's call this optimization factor O. For part B, we define the number of compute units by N. The total execution time then equals:

$$T = \frac{A}{O} + \frac{B}{N} \tag{2}$$

Part B can also be interpreted as the total program minus part A. Note that non-parallelizable part A benefits from the optimization mentioned earlier:

$$B = \frac{1 - A}{O} \tag{3}$$

The total execution time T with variables O and N then equals:

$$T(0, N) = \frac{A}{O} + (\frac{1 - A}{O})/N \tag{4}$$

The total program speedup S can be calculated with Amdahl's law by comparing the execution time of the old program T with the execution time of the new program T(O, N):

$$S = \frac{T}{T(O, N)} \tag{5}$$

0.1.4 OpenCL

Bibliography

[1] B. Barney, "What is parallel computing," *Introduction to Parallel Computing*, 2012.

[2] *Elements of Parallel Computing and Architecture.* Gullybaba Publishing House, 2014.

[3] J. Jenkov. (2015) Amdahl's law. [Online]. Available: http://tutorials.jenkov.com/java-concurrency/amdahls-law.html