Brian Quinn

Multicore Programming

Professor Mitchell

15 February 2017

Homework 1

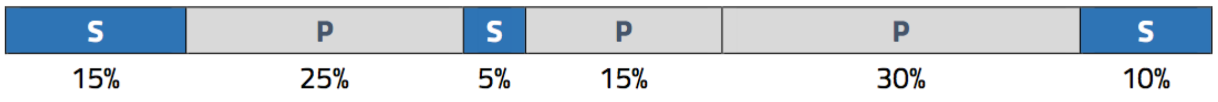
1. What makes MISD processor architectures unusual? What are they generally used for, and why aren't they used more widely?
    - MISD Processors are Multiple Instruction Single Data architecture. They are unusual because trying to run multiple different operations with the same data creates a lot of problems. Therefore, it is only really practical to use for the same operations so they are used for accuracy tests in planes and power plants, etc.

2. Discuss the five types of parallelism. What are they, and for each one, what is necessary from the hardware and/or programmer?
    - Temporal Parallelism is CPU instruction level parallelism. It is also known as pipelining. In this case, the CPU runs multiple instructions for each assembly instruction. First, the instruction is fetched, decoded, any mathematical computations are executed, memory is altered, and then finally written back permanently. This type of parallelism requires hardware buses forwarding each stage back to another so that non-conflicting CPU instruction states can be performed at the same time.
    - A chip multi-processor is a type of parallel computing that contains multiple cores on a single chip, which requires the programmer to make use of these cores in

parallel. Multiple processes will be run on different cores in the same chip. This encapsulates the idea of Multicore Programming and is a subset of process level parallelism.

- Multiprocessor systems are a type parallel computing where multiple chips are used to achieve parallelism and also requires the programmer to account for such. Multiple processes will be run on different chips at the same time, whether on multiple cores (defined above) or not, although the two are often used together. This the another subset of process level parallelism and known as parallel computing,
- Thread level parallelism occurs when multiple threads are executed at once. This allows other threads to complete operations while another thread is waiting for more costly operations like i/o or memory accesses.
- Task level parallelism is designating different parts of code to run on a designating cpu. Languages like VHDL and Verilog make use of this.

3. Apply Amdahl's Law to compute the speedup for the following program if you have (a) 1, (b) 2, (c) 4, (d) 8, (e) 16, and (f) ∞ CPUs. (1/(F+(1-F)/P))

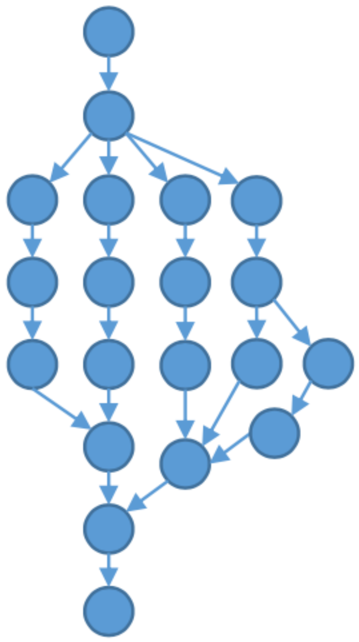


a) 1/(3/10+(1-3/10)/1) = 1.0
b) 1/(3/10+(1-3/10)/2) = 1.538
c) 1/(3/10+(1-3/10)/4) = 2.105
d) 1/(3/10+(1-3/10)/8) = 2.581

e) 1/(3/10+(1-3/10)/16) = 2.909

f) 1/(3/10+(1-3/10)/∞) = 3.333

4. Assuming that each block of the diagrammed program takes 1 unit of time, what is the work and span of the following program? What is the parallelized execution time TP on (a) P=1 processors? (b) P=5 processors? (c) P=6 processors? (d) How long is the critical path? (Total Work = 20, Span = 9)



a) 20

b) 2.222

c) 2.222

d) Critical Path: 9

5. Explain the difference between concurrency and parallelism with an example: if an operating system is executing two long-running programs, how would its scheduler

execute the programs concurrently on one core, concurrently on two cores, or in parallel on two cores? Comment on running the programs in parallel on one core.

- Concurrency is at least two tasks making progress in the same time frame. An example of this would be a simple lottery scheduler where processes on a computer are run based on a certain quanta and tickets. Each process will be run intermittently, but due to a small quantum it appears as if each process is running at the same time to the user. Concurrency on two cores would have a single process sending information to the cores to be executed.
- Parallelism is when two processes run exactly at the same time and would result in both processes sending information to be executed to the cores. Running parallel on a single core is much trickier as the instructions would be intermixed and the fetch stage would somehow need to switch between instructions on each process. It also may not be considered completely parallel because a single stage could not be running at the exact same time for both processes (if p1 is in execute, p2 must be in fetch, decode, etc.)

6. Broadly, what's the point of cache? Does its purpose differ between single- and multi-core processors (if so, how)? Does its implementation differ between single- and multi-core processors (if so, how and why)?
   - Cache memory is essentially storing frequently or recently used variables in SRAM rather than storage memory in order to make quick accesses to the information possible. Implementation for multicore processors have to incorporate a coherence protocol to account for cache changes by other cores.