# FIT3142 – Tutorial 4 Solutions

1. Explain the data dependency problem at instruction, routine and program level. What impact does it have on control flow in a program?

Topic 8: Slide 6,

Some computations exhibit a property which is termed “data dependency”. A data dependency between two computations arises when one of these computations depends upon the result of the other. Therefore the dependent computation cannot be executed until the computation upon which it depends is completed.

Data dependencies can arise at the instruction/operand level, function or subroutine level, and at the program level. The level at which data dependency occurs determines what techniques we can use to exploit parallelism and make the program run faster. Programs with frequent data dependencies at all levels do not lend themselves to speedup via parallel techniques.

Instruction level data dependencies:
Ex:
X = Y; #Statement 1
Y = X + 5; #Statement 2
Statement 2 which is a basic instruction depends on the result of Statement 1. So, these two statements cannot operate independently and hence cannot be executed concurrently.

Routine level (functional level) data dependencies:
Ex:
While i>=0
Compute(i) = Compute(i-1) + 1;
The above recurrent statement cannot be computed without the knowledge of Compute(i-1).

Program level data dependencies:
Ex:
When two different applications depend on eachother’s outputs to step forward to the next stage.

In order to achieve parallelism, statements have to be reordered and scheduled. While reordering, correctness has to be maintained regarding dependencies. In case of parallel processing, more dependencies imply more access time or more inter-process communication. This degrades the overall performance of the system.

In computer science, control flow (or flow of control) is the order in which individual statements, instructions or function calls of an imperative program are executed or evaluated. You can design control flows that support parallel processing and scheduling. The parallel container operator groups together independent activities that can run concurrently. The control flow in program depends heavily on the data dependencies and hence the amount of parallelism that can be achieved.

2. Explain Amdahl's Law. What is the dimension of S? Explain what happens when p= 0, and when s= 0.

Topic 8 – Slides 11, 12

Assume N is the number of processors, and s, p are the respective fractions of time spent on the serial and parallel components of a program, where

$$S = \frac{1}{s + \frac{p}{N}} \; where \; s + p = 1$$

This expression is known as Amdahl's Law, and it shows that the performance gain in a multiple processor system depends strongly upon the behaviour of the program. Adding CPUs only makes sense when the program has a large parallel component.

S is dimensionless

p = 0 ➔ s =1: means no parallelism is possible in the given task. Everything is done serially so adding processors is just a waste of resources and it does not speed up the operation.

s = 0 ➔ p =1: All parts of the task can be parallelised is all levels of operation. Therefore you can an arbitrary number of processors to increase the speed of operation.

3. In a distributed system, the terms can be represented as $s_{compute} + s_{network}$. Explain why.

Topic 8: Slides 38,

In a grid environment, the fraction of time consumed performing "serial" computation comprises the time consumed by the CPUs doing the work, and the time consumed communicating, when hosts working on data with dependencies must transmit the results of communications.

•Therefore: $s = s_{comp} + s_{network}$

•Where $s_{network}$ is the time for these results to be encapsulated in a message, sent across the network, and extracted for use at the destination.

4. In a distributed system, $s_{network}$ comprises several components. Explain each component, and why some are time variant and some are not.

Topic 8: Slides 40

The time it takes a message to travel across a network depends on a number of factors:

1) Processing delays on the middleware libraries and protocols stack at the transmitting end;

2) Cumulative queueing delays in routers along the transmission route through the network;

3) Propagation delay through cables in the network, proportional to total cable length;

4) Processing delays on the middleware libraries and protocols stack at the receiving end.

•Only 3) is constant in time, the remainder being time variant, and depend on network congestion.