Granularity and Schedules
Data Dependencies

- **Forward or flow dependency**: $S_1 \rightarrow S_2$ Reads $S_2$ is dependent on $S_1$. (RAW hazard)
- **Anti-dependency**: $S_1 \leftrightarrow S_2$ If $S_2$ occurs after $S_1$, and writes an input of $S_1$. (WAR hazard)
- **Output-dependency**: $S_1 \leftrightarrow S_2$ Both target the same storage. (WAW hazard)
- An I/O dependency may exist between two operations which target the same I/O device
- An unknown-dependency: must be assumed in the case of data which cannot be disambiguated at compile time.
Consider the following fragment:

S1  Load R1, A
S2  Add R2, R1
S3  Move R1, R3
S4  Store B, R1

- Forward dependency S1 to S2, S3 to S4, S2 to S2
- Anti-dependency from S2 to S3
- Output dependency S1 to S3
Other Dependencies

- Control dependencies (control hazards) are created when program flow cannot be resolved at compile time.
- If the branch predicate can be computed at compile time, the dependency does not exist. Otherwise, it does.
- Control dependencies are an issue when they affect data dependencies
- Resource dependencies (structural hazards) occur, when multiple operations contend for the same physical resource, such as a functional unit.
Bernstein’s Conditions

- Bernstein’s Conditions ensure that no dependencies between processes (or process fragments) exist.

- Define the input and output set of a process $P_i$ as $I_i$, $O_i$ respectively.

- Any two processes $P_i$ and $P_j$ can execute concurrently and produce unambiguous results if:

\[
I_1 \cap O_2 = \emptyset \\
I_2 \cap O_1 = \emptyset \\
O_1 \cap O_2 = \emptyset
\]  

(1)
HW and SW Parallelism

• The concurrency present in a particular piece of code, may or may not be exploitable on a particular architecture

• Hardware parallelism may be defined as functional parallelism times the processor parallelism. This leads directly to the notion of Peak Attainable Performance.

• Software parallelism is defined by the dependencies within the code as controlled by
  • algorithm
  • programming style, model, language
  • compiler optimization
Matching HW and SW Parallelism

- On a two issue machine with a load/store unit and one arithmetic unit, the schedule is:

  L1  
  L2  
  L3, X1  
  L4  
  X2  
  +  
  -

resulting in 1.14 Instructions per cycle

2.67 Instructions/cycle
Control vs. Data Parallelism

- SW parallelism is manifested as a mixture of control and data parallelism.
- It is possible to convert data parallelism into control parallelism e.g. a loop may be split into several loops, and mapped on to separate processors executing asynchronously.
- Control parallelism is limited by the serial fraction.
- Data parallelism scales as the data grows.
Compilers

• Building a parallel architecture is easy, but building a smart compiler for that architecture is hard.
Partitioning and Scheduling

- Granularity is the size of the program chunks we try to execute in parallel
  - fine grain: statements and loops
  - medium grain: processes, threads, procedures
  - coarse grain: programs or large procedures
- As grain size decreases, potential parallelism increases, and overhead also increases.
- Overhead is the cost of parallelizing a task. The principle overhead is communication latency.
- As grain size is reduced, there are fewer operations between communication, and hence the impact of latency increases.
- Surface to volume: inter to intra-node comm.
Communication Latency

• Communication latency is a function of hardware
  • network topology
  • network bandwidth
  • network switching and routing protocols

• and a function of software
  • message load
  • message patterns i.e. unicast, multi-cast, broadcast, synchronize, permutations, etc.

• Related issues include
  • deadlock avoidance
  • livelock (or sludgelock) avoidance
Static Schedules

- Consider $\sum C_{ij}$, where

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

- we have

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{21}$$
$$c_{22} = a_{21}b_{12} + a_{22}b_{22}$$
Eight-way partition and schedule:
Four way schedule