- **Program properties and communication:**

  To divide a program into segments so that it can be executed in parallel, we need to look at various dependence among instructions and the amount of communication among processors.

  – **Data dependences**

    Dependence graph: A directed graph

    Node: instructions (statements)

    Arc: ordered relations among the instructions

* **Flow dependence**

  The output of S_1 is the input of S_2, denoted as $S_1 \rightarrow S_2$.

* **Antidependence**

  The output of S_2 overwrites the input of S_1, denoted as $S_1 \rightarrow S_2$.

* **Output dependence**

  S_1 and S_2 write to the same variable, denoted as $S_1 \rightarrow S_2$.

* **I/O dependence**

  S_1 and S_2 access the same file, denoted as $S_1 \stackrel{I/O}{\rightarrow} S_2$.

* **Unknown dependence, e.g.**

  - $A[B[i]]$
  - $A[i^2]$
  - $A[i]$ **where i is a global variable.**
  - $A[i]$ **and** $A[2i]$ **both appear.**

– Examples:

Example 1:

S1: Load R1, A /R1 ← Memory(A)/

S2: Add R2, R1 /R2 ← (R1)+(R2)/

S3: Move R1, R3 /R1 ← (R3)/

S4: Store B, R1 /Memory(B) ← (R1) /

Example 2:

S1: Read (4), A(I) /Read array A
from tape unit 4/

S2: Rewind(4) /Rewind tape unit 4/

S3: Write (4), B(I) /Write array B
into tape unit 4/

S4: Rewind (4) /Rewind tape unit 4/

– Control dependence

Execution order can only be determined at run time.

- **Example 1: control-independent loops**

```
Do 20 I = 1, N
    A(I) = C(I)
    If (A(I) .LT. 0) A(I) = 1
20 Continue
```

- **Example 2: control-dependent loops**

```
Do 10 I = 1, N
    If (A(I-1) .EQ. 0) A(I) = 1
10 Continue
```

– Resource dependence

Such as using the same ALU or storage

– Bernstein's conditions (the conditions two processes can be executed in parallel)

$$I_1 \cap O_2 = \phi$$

$$I_2 \cap O_1 = \phi$$

$$O_1 \cap O_2 = \phi$$

where, $I_i \ (i = 1, 2)$ **is the input set of process** P_i **and** $O_i \ (i = 1, 2)$ **is the output set of process** P_i**.**

– Processes $P_1, P_2, \ldots, P_n$ **can be executed in parallel if** $P_i || P_j$ **for any** $i \neq j$**.**

$||$ **relation is commutative but not transitive.**

– Example: schedule the following program

$$P_1 : \ C \ = \ D \times E$$

$$P_2 : \ M \ = \ G + C$$

$$P_3 : \ A \ = \ B + C$$

$$P_4 : \ C \ = \ L + M$$

$$P_5 : \ F \ = \ G/E$$

Dependences:

$$P_1 \rightarrow P_2 \ data \ (C)$$

$$P_1 \rightarrow P_3 \ data \ (C)$$

$$P_1 \rightarrow P_4 \ output \ (C)$$

$$P_2 \rightarrow P_4 \ data \ (M)$$

$$P_2 \rightarrow P_4 \ anti \ (C)$$

$$P_3 \rightarrow P_4 \ anti \ (C)$$

$$P_1||P_5, P_2||P_3, P_2||P_5, P_5||P_3, P_4||P_5.$$

$$P_2||P_3||P_5$$

– Hardware parallelism

- **Determined by machine architecture**
- **Indicates peak performance**
- **Characterized by the number of instruction issues per machine cycle**
- **Instruction issue:**
  reserve a functional unit, send an op code to it and reserve the result register.
- k**-issue processor: issues** k **instructions per machine cycle.**
  $k \leq 1$**: one issue machine (conventional machine)**
  $k > 1$**: pipelined computer**

- **Software parallelism**
  - **Determined by algorithm, programming style and compiler**
  - **Maximum parallelism allowed by dependence**
  - **Example: Mismatch between software and hardware parallelism**
    - **The program**
    - **Executed by a two-issue superscalar processor**
    - **Executed by a dual-processor**

- **Job scheduling on parallel computers**
  - **Grain: a segment of the program executed by a processor**
  - **Grain size**
    - **Fine grain: at instruction level (about 20 instructions)**
    - **Medium grain: at loop level (about 500 instructions)**
    - **Coarse grain: at procedure level (about 2000 instructions)**
  - **Finer grain has more parallelism, but requires more communications among processors.**

– Communication latency:

Time required to communicate between PEs.

– Basic communication patterns

(determined by algorithms and architectures)

- **Permutation (one-to-one)**
- **Broadcast (one-to-all)**
- **Multicast (one-to-many)**
- **Conference (many-to-many)**

– Grain-size problem:

Determine the number and the size of the grains in a parallel program to yield the shortest possible execution time.

The smaller grain size, the more communication overhead.

– An example of grain packing:

- **Program graph:**

  Node: (n,s)

  n: - node name

  s: grain size (# machine cycles)

  Edge: (v,d)

  v: output variable of the source or input variable of the destination

  d: communication delay.

- **Basic idea: divide the program as fine as possible to achieve the highest parallelism, then pack some grains to reduce communication delay to achieve the shortest execution time.**
- **Scheduling for fine grain**
- **Scheduling for coarse grain**
- **Node duplication**

- **Steps of scheduling a job on parallel machine**
  - **Construct a fine-grain program graph (exploit the maximum parallelism)**
  - **Schedule the fine-grain computation**
  - **Grain packing (reduce delay)**
  - **Schedule the packed graph.**
  - **Repeat if necessary.**
- **Example: matrix multiplication.**

- **Three types of computers**
  - **Control-driven: von Neumann machines**
  - **Data-driven: data flow machines, driven by data availability.**
  - **Demand-driven: reduction machines, start the computation only when the results are needed.**
- **Eager evaluation and lazy evaluation**
- **Comparison of dataflow and control-driven computers**