- **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/

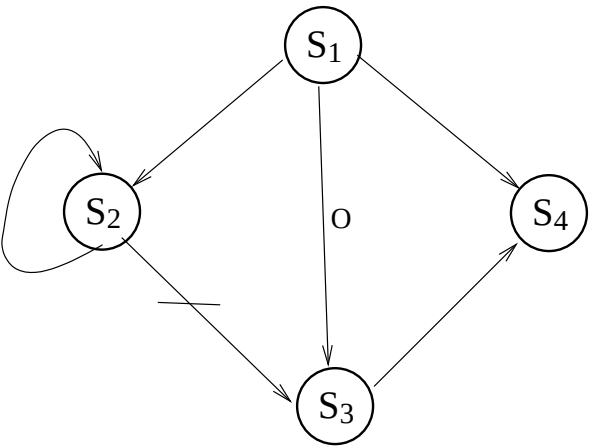
Dependence graph:

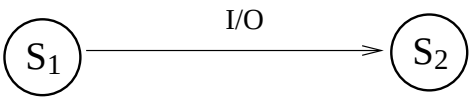
$S_1 \longrightarrow S_2$ flow dependence

$S_1 \not\longrightarrow S_2$ antidependence

$S_1 \xrightarrow{O} S_2$ output dependence

$S_1 \xrightarrow{I/O} S_2$ I/O dependence





– 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:

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

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

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

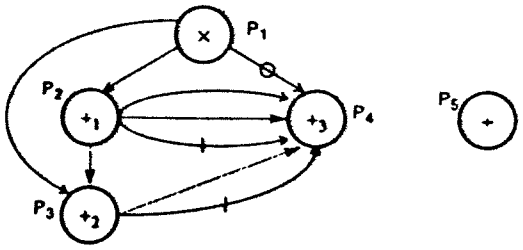
$$P_2 \rightarrow P_4 \ data \ (M)$$

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

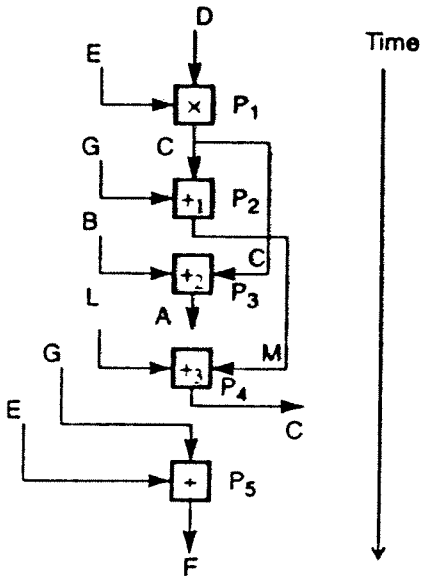
$$P_3 \rightarrow P_4 \ anti \ (C)$$

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

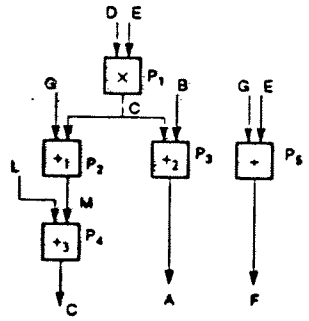
$$P_2||P_3||P_5$$



(a) A dependence graph showing both data dependence (solid arrows) and resource dependence (dashed arrows)



(b) Sequential execution in five steps, assuming one step per statement (no pipelining)



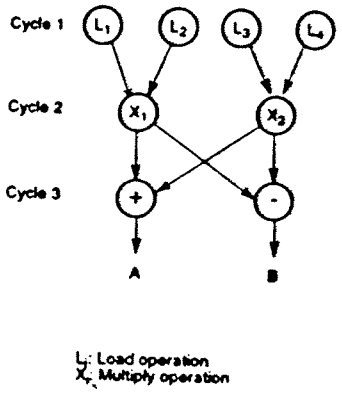
(c) Parallel execution in three steps, assuming two adders are available per step

Figure 2.2 Detection of parallelism in the program of Example 2.2.

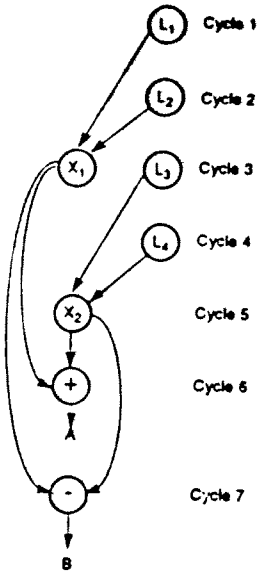
– Hardware parallelism

- **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**



(a) Software parallelism



(b) Hardware parallelism

Figure 2.3 Executing an example program by a two-issue superscalar processor.

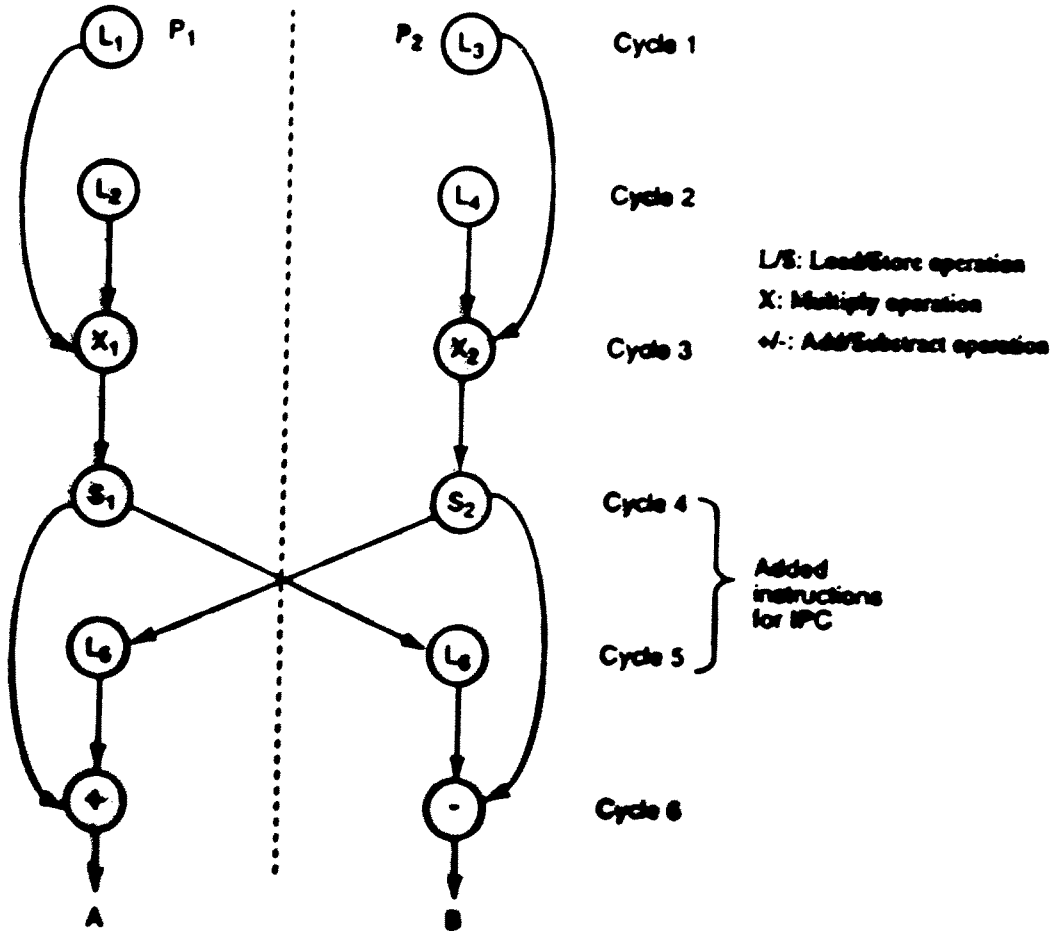


Figure 2.4 Dual-processor execution of the program in Fig. 2.3a.

- **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.**

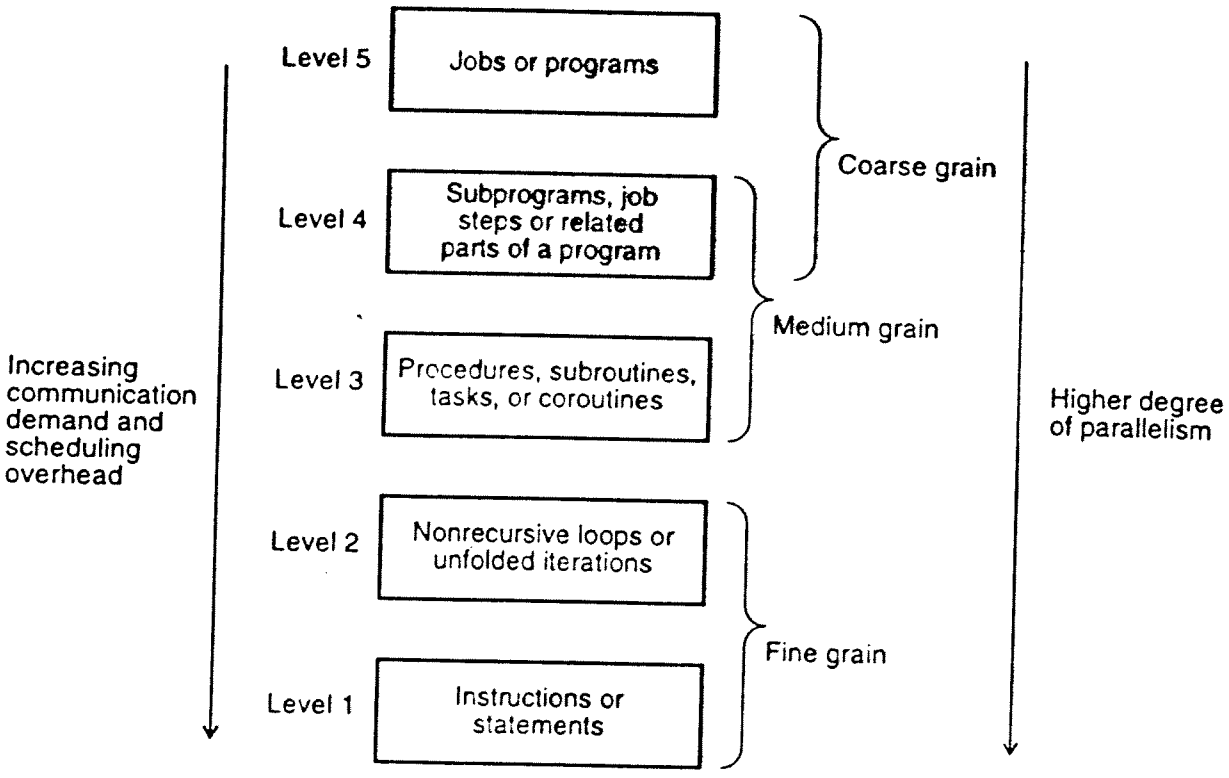


Figure 2.5 Levels of parallelism in program execution on modern computers. (Reprinted from Hwang, *Proc. IEEE*, October 1987)

– 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

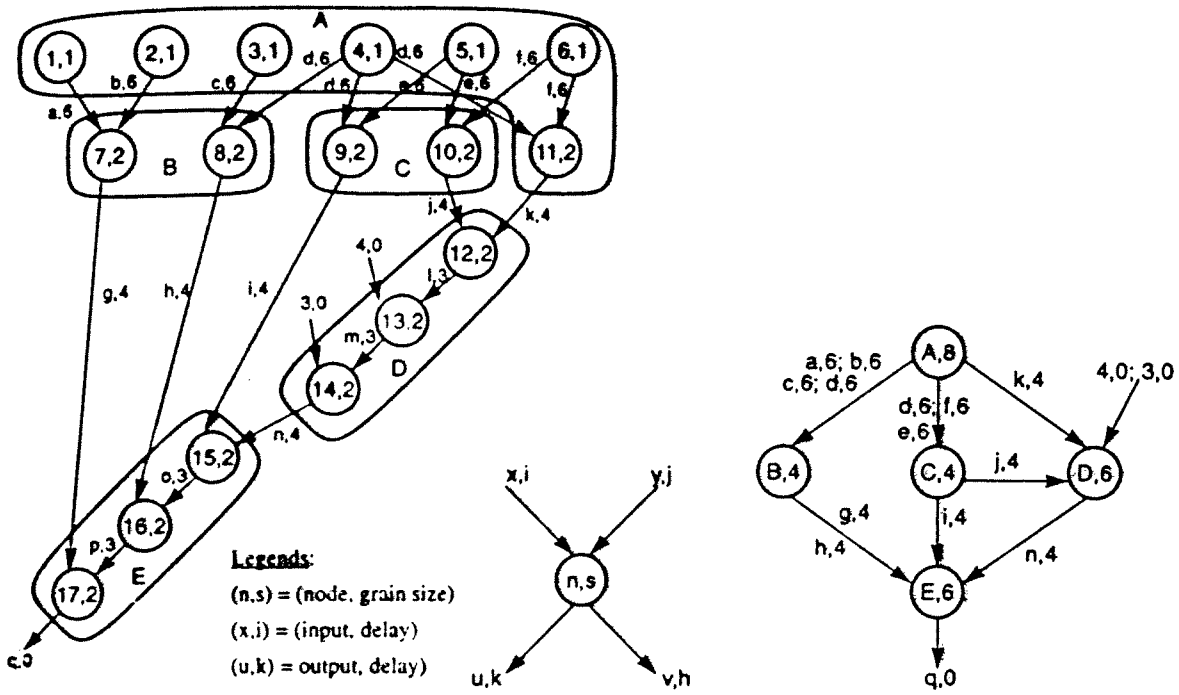
  s: grain size (# machine cycles)

  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**



(a) Fine-grain program graph before packing

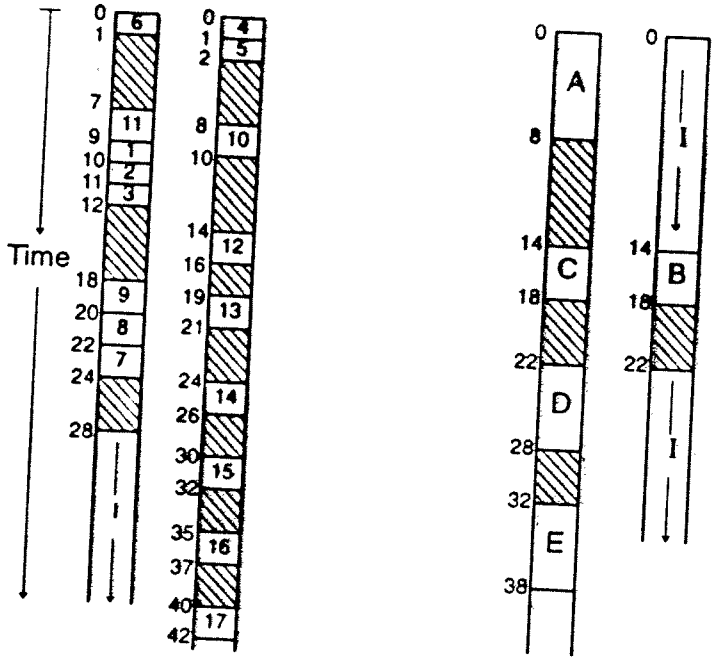
(b) Coarse-grain program graph after packing

Figure 2.6 A program graph before and after grain packing in Example 2.4. (Modified from Kruatrachue and Lewis, *IEEE Software*, Jan. 1988)

Begin

1.	$a := 1$	10.	$j := e \times f$
2.	$b := 2$	11.	$k := d \times f$
3.	$c := 3$	12.	$l := j \times k$
4.	$d := 4$	13.	$m := 4 \times l$
5.	$e := 5$	14.	$n := 3 \times m$
6.	$f := 6$	15.	$o := n \times i$
7.	$g := a \times b$	16.	$p := o \times h$
8.	$h := c \times d$	17.	$q := p \times q$
9.	$i := d \times e$		

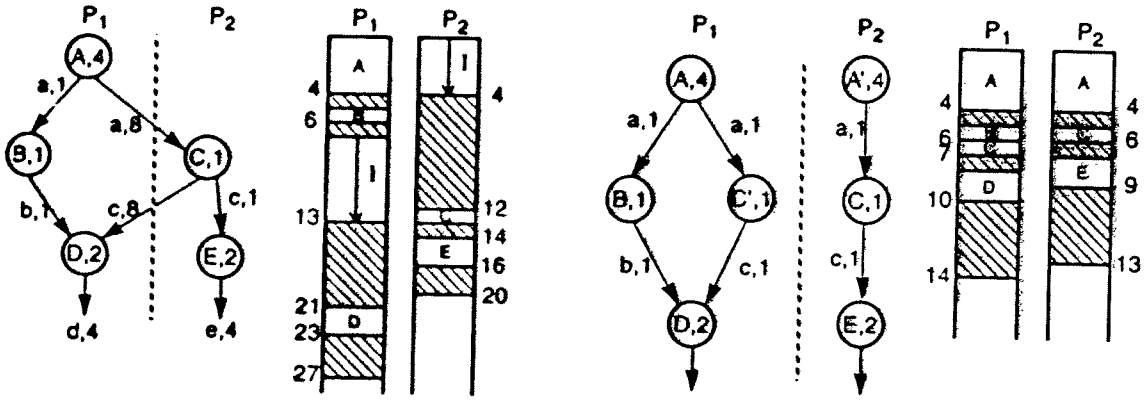
End



(a) Fine grain (Fig. 2.6a)

(b) Coarse grain (Fig. 2.6b)

Figure 2.7 Scheduling of the fine-grain and coarse-grain programs. (I: idle time; shaded areas: communication delays)



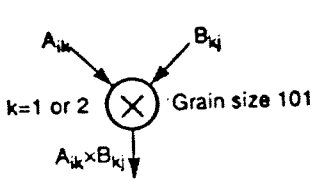
(a) Schedule without node duplication

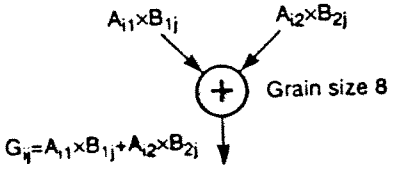
(b) Schedule with node duplication (A → A and A′; C → C and C′)

Figure 2.8 Node-duplication scheduling to eliminate communication delays between processors. (I: idle time; shaded areas: communication delays)

- **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.**

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

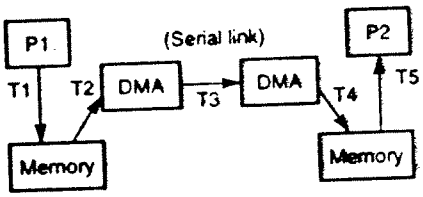




			CPU CYCLE
Move W	Axx,	D1	15
Move W	Bxx,	D2	15
MPTY D1,	D2		71
MOVE L	D2,	PAR	20

			CPU CYCLE
Move L	PAR1,	D1	20
Move L	PAR2,	D2	20
ADD L	D1,	D2	8
MOVE L	D2,	PSUM	20

(a) Grain size calculation in M68000 assembly code at 20-MHz cycle

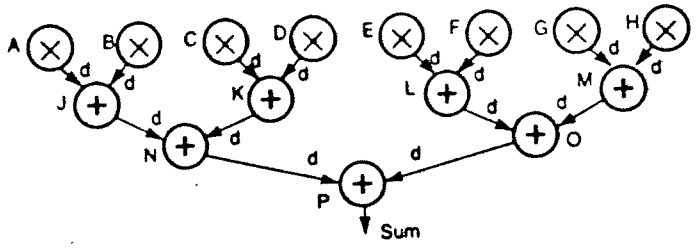


d = T1+T2+T3 +T4+T5+T6
= 20+20+32+20+20+100
= 212 cycles

T3 = 32-bit transmission time at 20 Mbps normalized to M68000 cycle at 20 MHz.

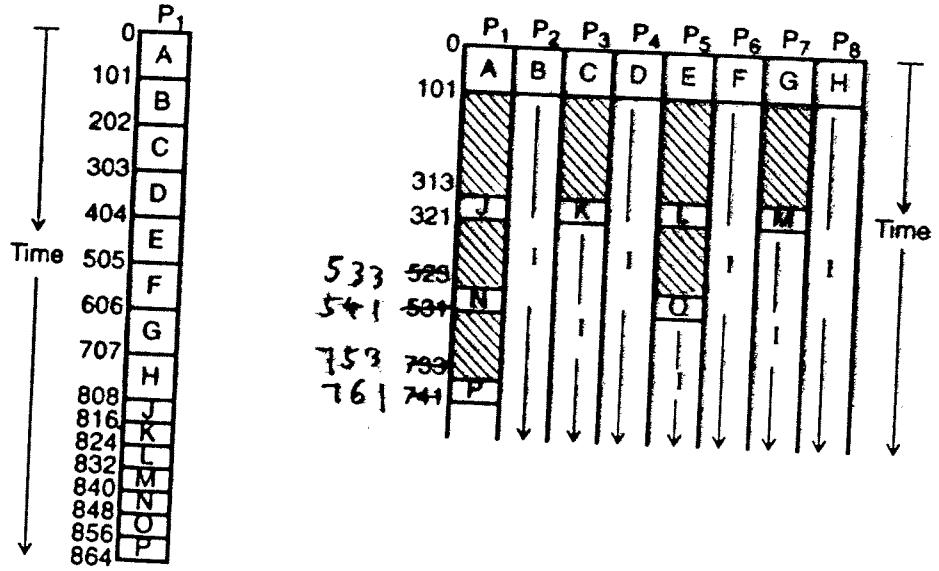
T6 = delay due to software protocols (assume 5 Move instructions, 100)

(b) Calculation of communication delay d



(c) Fine-grain program graph

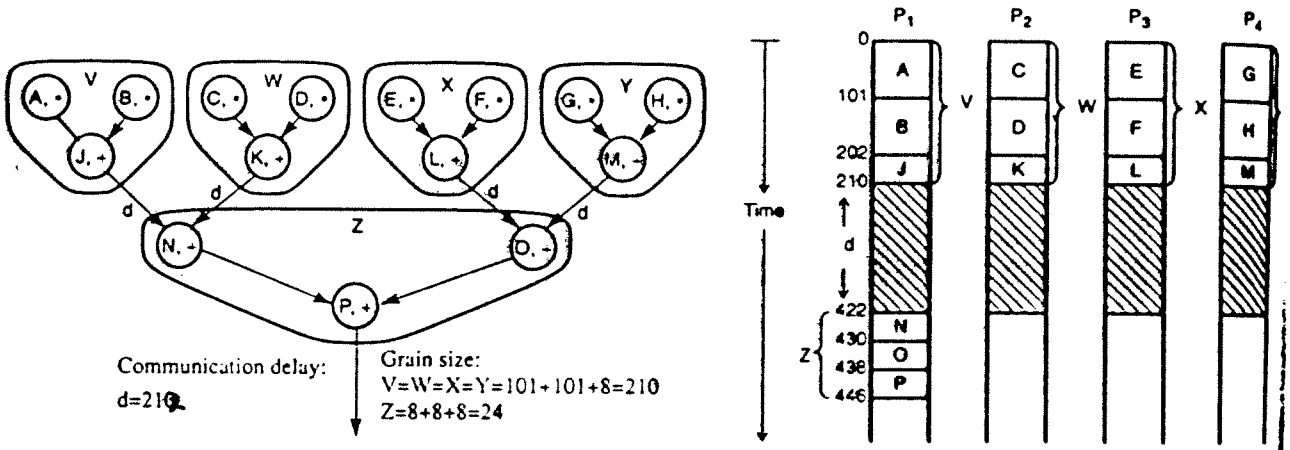
Figure 2.9 Calculation of grain size and communication delay for the program graph in Example 2.5. (Courtesy of Kruatrachue and Lewis; reprinted with permission from *IEEE Software*, 1988)



(a) A sequential schedule

(b) A parallel schedule

Figure 2.10 Sequential versus parallel scheduling in Example 2.5.

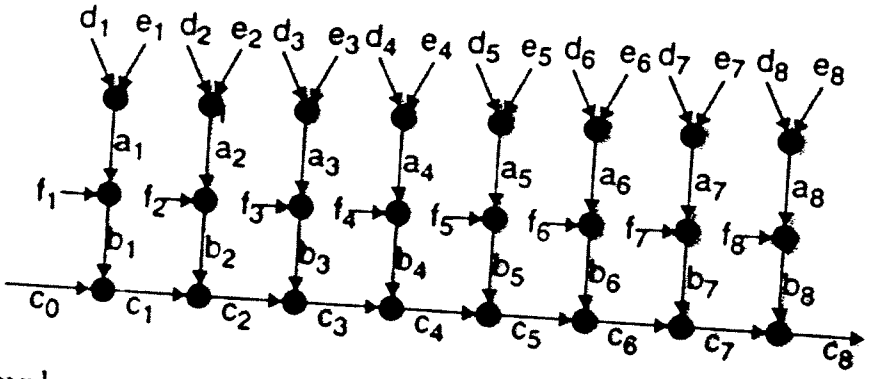


(a) Grain packing of 15 small nodes into 5 bigger nodes (b) Parallel schedule for the packed program

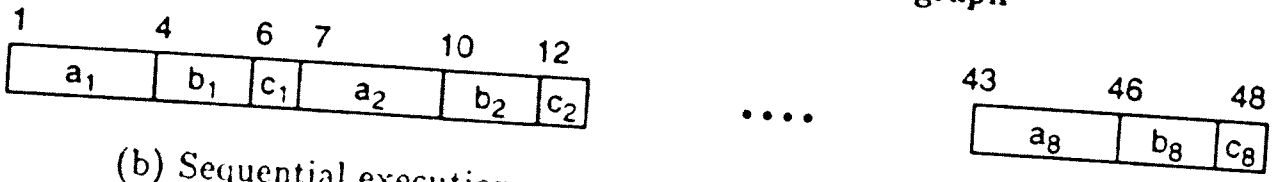
Figure 2.11 Parallel scheduling for Example 2.5 after grain packing to reduce communication delays.

- **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**

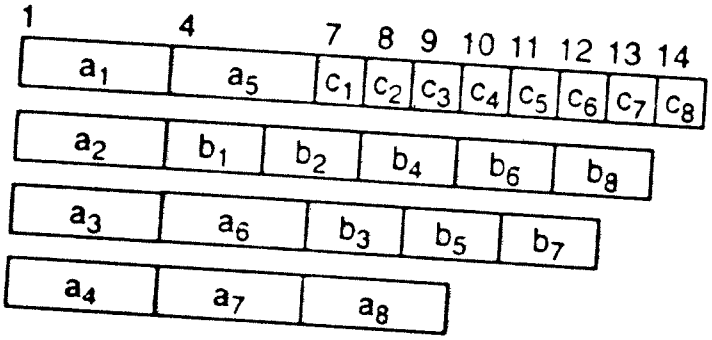
```
input d, e, f
  c0 = 0
for i from 1 to 8 do
  begin
    ai := di + ei
    bi := ai * fi
    ci := bi + ci-1
  end
output a, b, c
```



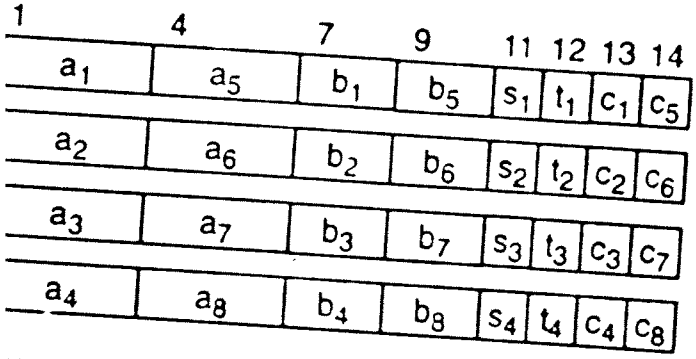
(a) A sample program and its dataflow graph



(b) Sequential execution on a uniprocessor in 48 cycles



(c) Data-driven execution on a 4-processor dataflow computer in 14 cycles



l) Parallel execution on a shared-memory 4-processor system in 14 cycles

Comparison between dataflow and control-flow computers. (Adapted om Gajski, Padua, Kuck, and Kuhn, 1982; reprinted with permission from *EEE Computer*, Feb. 1982)