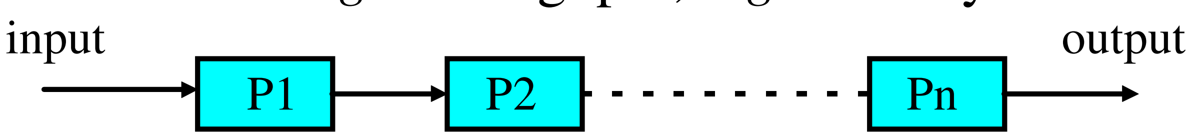
Multiprocessor systems

Even the fastest, modern RISC microprocessors and DSPs (which are very similar) cannot meet the highest processing requirements =>

- use special purpose VLSI devices
    - * task specific, eg convolution
- use several (many) processors

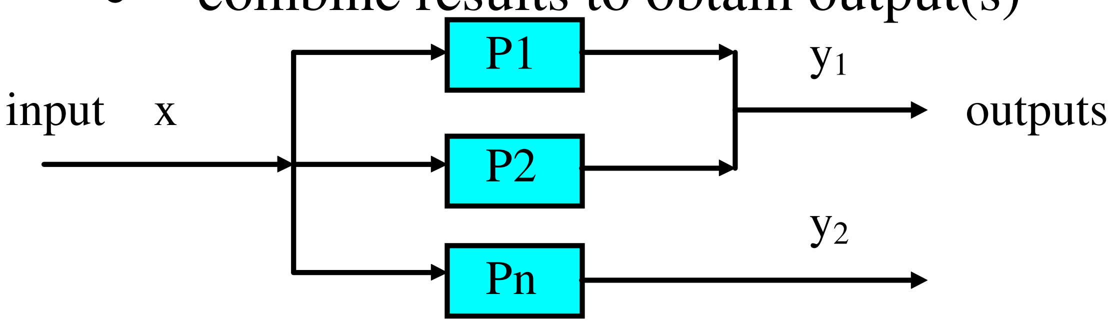
♦ in a pipeline

- calculation is broken down into stages
- output from previous stage used as input
    - * *high* throughput, *high* latency



♦ in parallel

- several processors working concurrently
- combine results to obtain output(s)



♦ both have to be balanced

- processors do similar amounts of work

A modern DSP: TMS320C40

- 50 Mhz, 40 nsec cycle time
- global and local busses
  - 2 x 100 Mbytes/sec
- DMA co-processor
  - data access)
  - address) => 75 MOPS
  - transfer counter)
- high performance CPU

2 data accesses	50
1 FP multiply	25
1 FP addition	25
2 address modifications	50
1 loop counter	25
1 branch	25
=>	275 MOPS

  - all instructions executed in two cycles
- parallel communication ports
  - 6 x 20 MBytes
- => called parallel DSP (pDSP)
- available in faster 4 processor multimedia video processor version (MVP) => claims 2GOPS

TMS320C40

- internal architecture

- example building block for a parallel system
- unit replicated
- => global and local (distributed) memory
- communication network

TMS320C80

Multiprocessor architectures

Three basic approaches:

- ♦ **pipelined**: as above
  - used in vector processors, especially in supercomputers and signal processing
- ♦ **shared memory**
  - several processors access same memory
    - * via multiport memory
    - * common bus
    - * simple, cost effective
    - * **but:** problems of maintaining data integrity, and resolving access contention
    - * does not scale as # of processors is increased
  - used in powerful workstations and supercomputers
- ♦ **distributed memory/message passing**
  - communications bandwidth increases as # of processors increases => scales better
  - **but:** problems of data staleness
    - * complexity of communication network unless a simple, fixed topology is used
  - used in embedded systems, especially for flexibility and fault tolerance, etc

Multiprocessor parallel architectures

- **SISD** - single instruction, single data

(conventional)

- **MISD** - multiple instruction, single data

- **SIMD** - single instruction, multiple data

  - simple, can use a very large number of (simple) processors
  - may utilise shared memory **or** a communication network and distributed memory, **or** a combination of both

Multiprocessor architectures (cont'd)

- typical SIMD communication networks
  - eg mesh   tree

- **MIMD** - multiple instruction, multiple data

- different processor may (usually) perform different tasks/functions
  - => known as functional decomposition
- complex to program
  - => usually only a small # of processors

Interconnection network

- important factors include:
  - (communication bandwidth)/(CPU performance) ratio
  - communication delays
  - routing of signals (topology)
    - should be free from bottle-necks and deadlocks (cf: Vaillant)
  - more links or ports =>
    - more complicated, eg TMS320C40

MIMD Implementation

- place **processes** on **processors**
  eg: for transputers (C40 similar)

- place processes by hand
  - does not scale
- use a processor farm
  - given communication network
  - place communication processes
  - place master processes, eg:
    * farmers - distribute work
    * reapers - gather results
- in principle, systems built out of standard modules for processing, communication (switches), interfaces, etc, eg
  - transputers: TRAMS
  - C40s: TIMS

Performance of parallel systems

We might expect that if N processors are used to implement an algorithm, that it will run N times as fast as when 1 processor is used, ie:

- speedup, $S = \dfrac{T_{seq}}{T_{par}} = N$
- **BUT:**
  - it may not be possible to implement all of the algorithm in parallel
    - * ie, to keep all N processors busy
  - if a fraction f of the work (number of operations) must be performed sequentially
  - $S \leq 1 \Big/ \left(f + \dfrac{1-f}{N} \right)$ Amdahl's Law
- **Note:**
  - if f is large (close to 1), there is little to be gained
  - ignores communication overheads etc
    - * hence an upper bound
  - assumes sequential and parallel algorithms are similar

A useful parallel architecture?

When using parallel processing to speed up implementation of a real-time system, we can:

- ♦ design a specific system, eg:
  - • architecture, communication network, memory organisation, etc adapted to the problem, the algorithms required, the data structures used etc, =>
  - • high performance, **but**
    - * high cost (to develop and produce)
    - * may not be flexible or fault tolerant
- ♦ use a fairly general architecture that gives most of the performance for much reduced design effort, eg:
  - • mesh or hypercube network
  - • pipeline of processor farms
    - * Downton
  - • parallel pipelines (of processor farms)
    - * Castelow, Gent

high throughput)		(high performance
low latency)	=>	(low cost
can be balanced)		(potential for flexibility

Examples

Pipeline of processor farms

- ♦ can be used to balance pipeline and reduce latency
  - cf simple temporal multiplexing (b, below)
  - see A C Downton et al, IEE Proc. Vis. Image and Signal Process. 161(6), Dec 1994