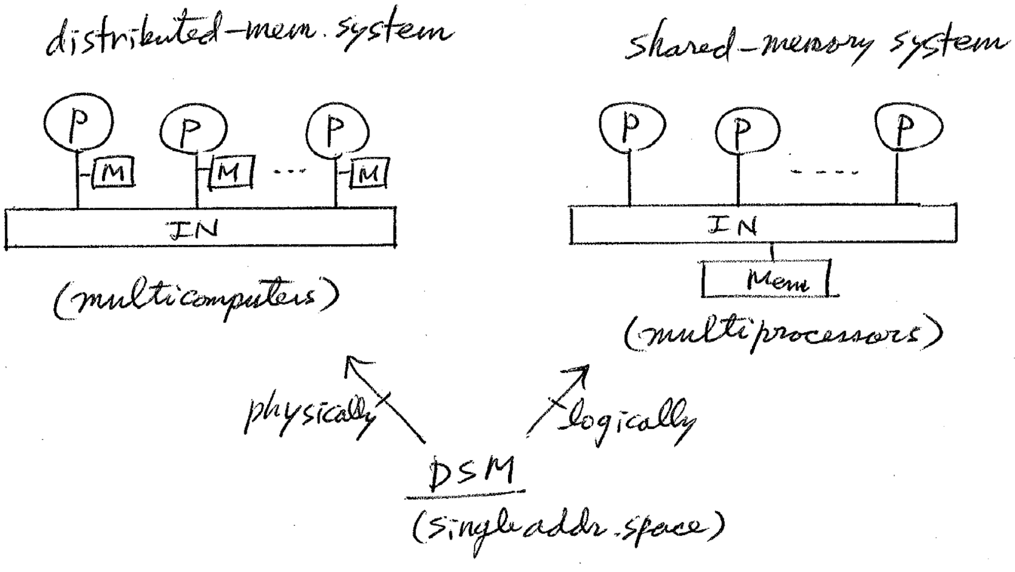
— Parallel Arch. and Taxonomy (Springer book, etc.)

- Flynn's taxonomy of parallel architectures
  - SISD — serial computer (uniprocessor)
  - MISD — X
  - SIMD — one control processor, multiple data processors
    (PEs execute same instr. with different data
  - MIMD
    - multiple PEs
    - each PE (control processor + data processor)
    - each PE has a separate instr. and data access
    - PEs execute different instructions with different data.

- Memory organization of parallel computers
  - shared-memory (multiprocessors) — single addr. space
  - distributed memory (multicomputers) — multiple addr. spaces
  - virtual shared-memory (DSM)



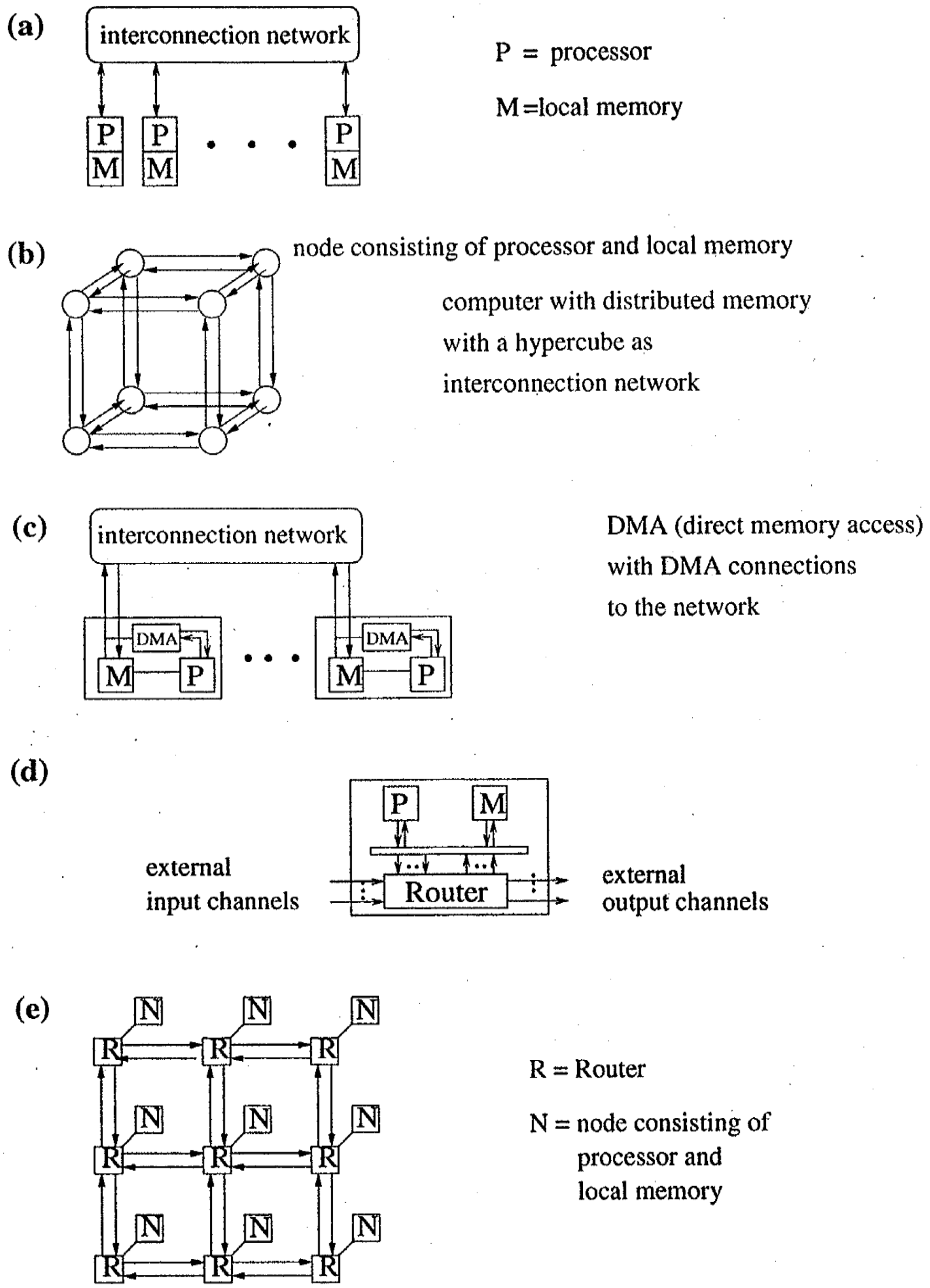


Fig. 2.3 Illustration of computers with distributed memory: (**a**) abstract structure, (**b**) computer with distributed memory and hypercube as interconnection structure, (**c**) DMA (direct memory access), (**d**) processor-memory node with router, and (**e**) interconnection network in form of a mesh to connect the routers of the different processor-memory nodes. *(from Springer book)*

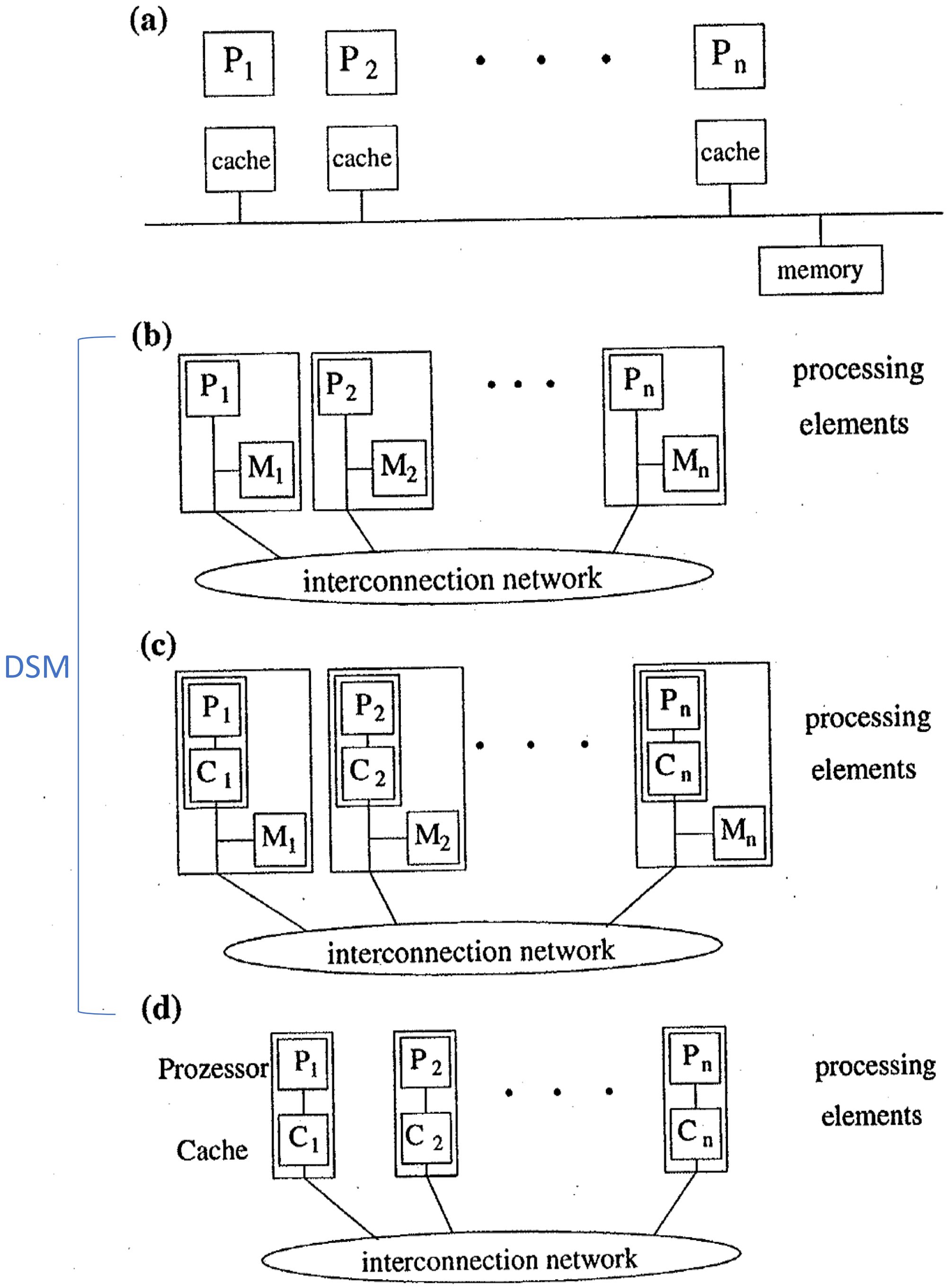
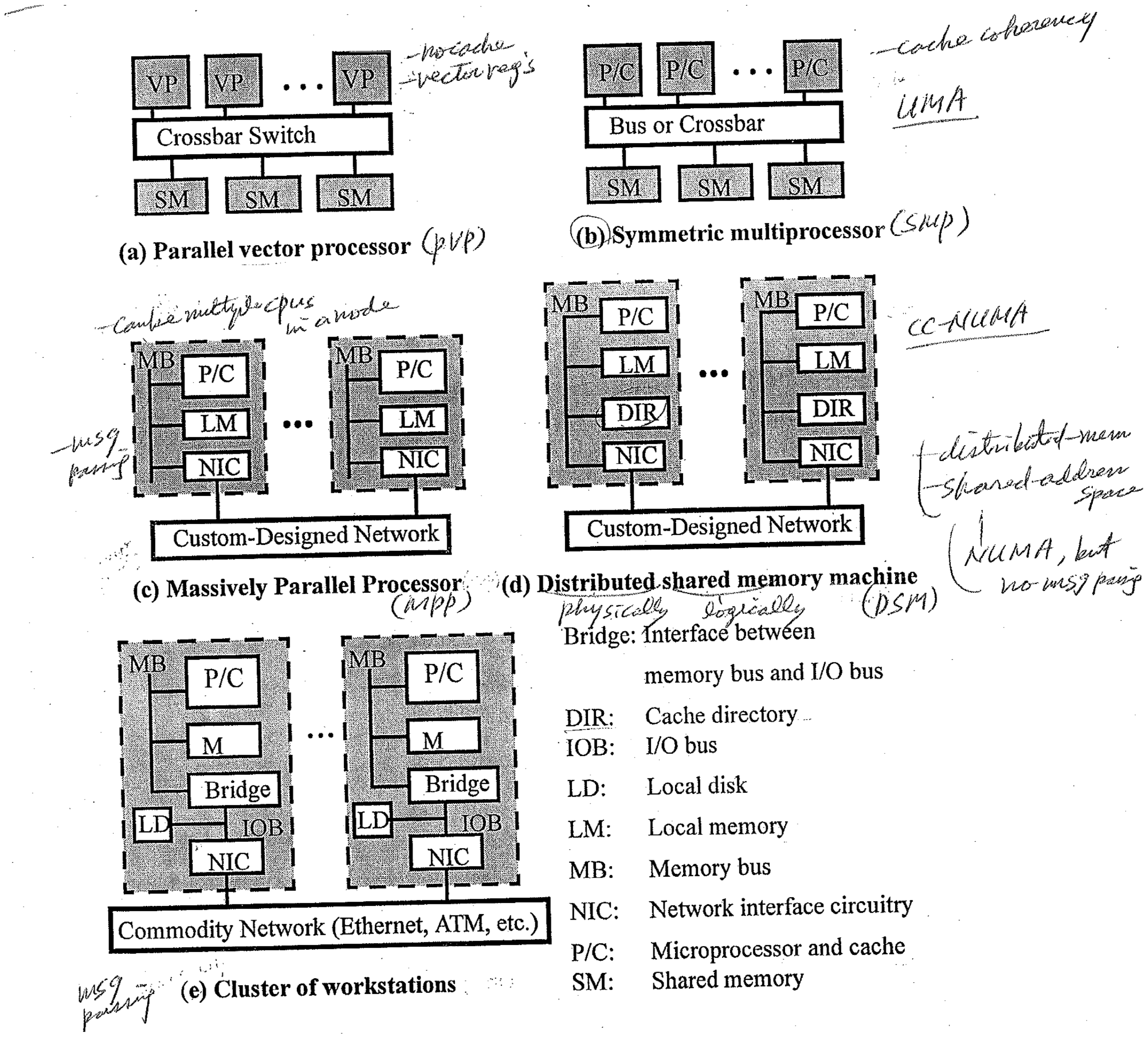


Fig. 2.5 Illustration of the architecture of computers with shared memory: **(a)** SMP – symmetric multiprocessors, **(b)** NUMA – nonuniform memory access, **(c)** CC-NUMA – cache coherent NUMA and **(d)** COMA – cache only memory access. *(from Springer book)*

Kai Hwang's Taxonomy – all MIMD



(a) Parallel vector processor (pVP)

(b) Symmetric multiprocessor (SMP)

(c) Massively Parallel Processor (MPP)

(d) Distributed shared memory machine (DSM)

physically logically

(e) Cluster of workstations

-no cache
-vector reg's

-cache coherency
UMA

-Can be multiple cpus in a node

-msg passing

cc-NUMA

-distributed-mem
-shared-address space

NUMA, but no msg passing

msg passing

Bridge: Interface between memory bus and I/O bus

DIR: Cache directory

IOB: I/O bus

LD: Local disk

LM: Local memory

MB: Memory bus

NIC: Network interface circuitry

P/C: Microprocessor and cache

SM: Shared memory

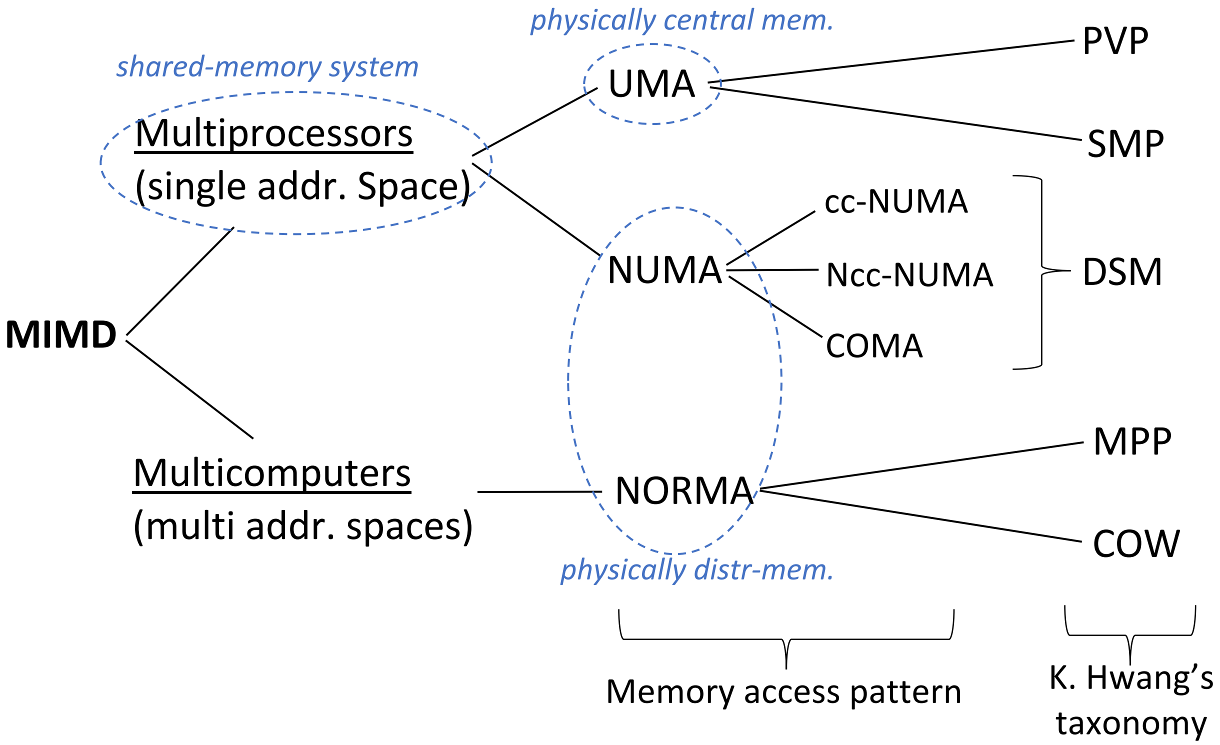
Figure Five physical parallel computer models: the PVP, SMP, MPP, DSM, and COW.

K-Hwang book

(a)(b)(d) —shared-mem machines

(c)(e) —distr. mem machines

- PVP – parallel vector processors
  vector registers, instr. buffer, no cache;
- SMP – symmetric multiprocessors
  each processor has equal access to resources (symmetric);
  centralized shared-memory;
- MPP – massively parallel processors
  very large scaled system;
  1 (or few) host OS and micro kernels;
- COW – cluster of workstations
  each node is a workstation without peripherals;
  complete OS in each node + layer for single system image;
- DSM – distributed shared-memory system
  physically distributed-mem, but logically shared-memory system;
  special HW/SW for memory consistency and coherency.



SIMD mode computation (data level parallelism) with

- vector processors – no cache, vector registers, vectorized/pipelined FUs, vector instructions; ex) Cray X-MP/4 (PVP structure);
- array processors – SIMD system, 1 CP and multiple DPs,
  ex) ILLIAC-IV (1966), CM-2 (Connection Machine, 1980's),
  SPE in Cell processor (2005~6, IBM/SONY);
- GPU – multithreaded SIMD processors

ex) vector processing

vector A vector B vector C

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ \vdots \\ C_n \end{bmatrix}$$

vector instr: $C_i = A_i + B_i$, $1 \le i \le n$

on SISD:

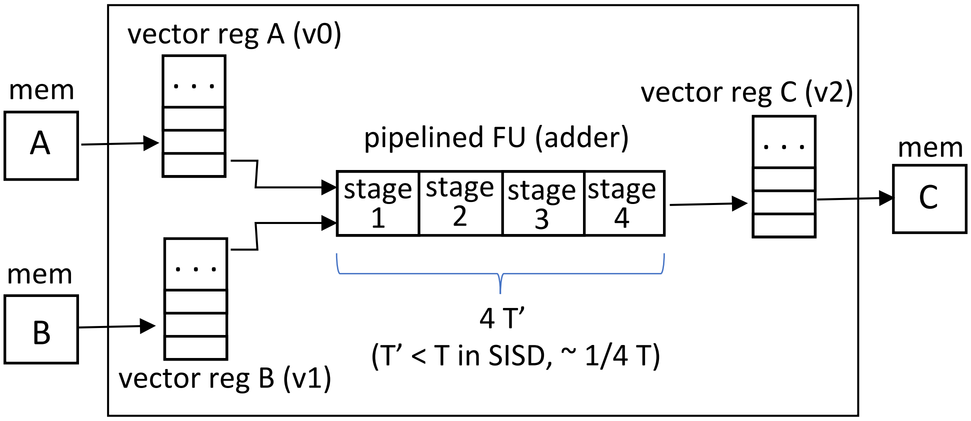
```
for i = 1 ~ n
    C[i] = A[i] + B[i];
```

→ for each i, 2 instr's for loop control,
4 instr's for { read Ai; read Bi; add; write Ci;}

→ exec. time = $\underline{6 * n * T}$
where T is ave. instr. exec. time

vs.

on vector processor (SIMD):



exec. time = $4T' + (n-1)T' = \underline{(n + 3)\, T'}$

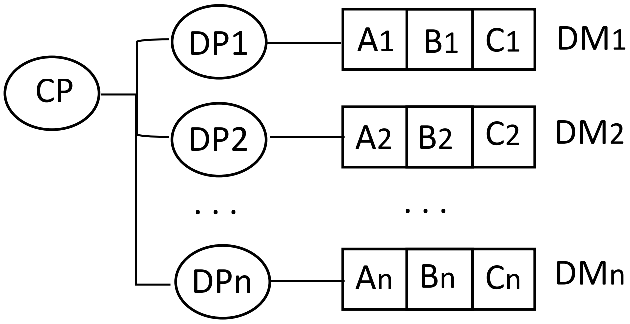
→ ~ 6 fold (if T' = T)

~ 4*6=24 fold (if T' = 1/4T)

vs. on array processor (SIMD):

for all

C[i] = A[i] + B[i]; *($1 \leq i \leq n$)* → *SIMD instr: C = A + B; //one instr.*



with n PEs (1 CP, n DPs),

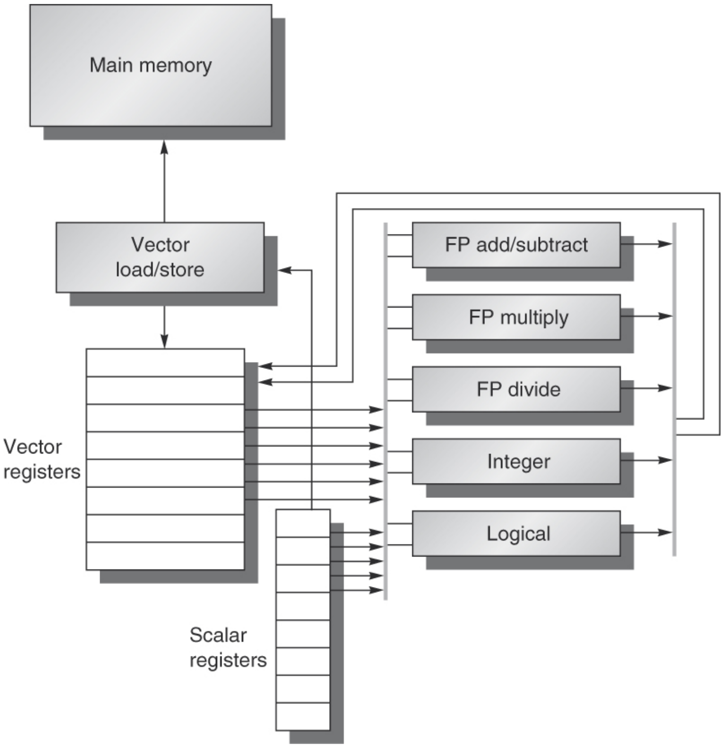
exec. time = 1 addition time

= 6T (read Ai; read Bi; add; write Ci;
+ synchronization time (assume 2T))

→ n (# of DPs) fold from SISD

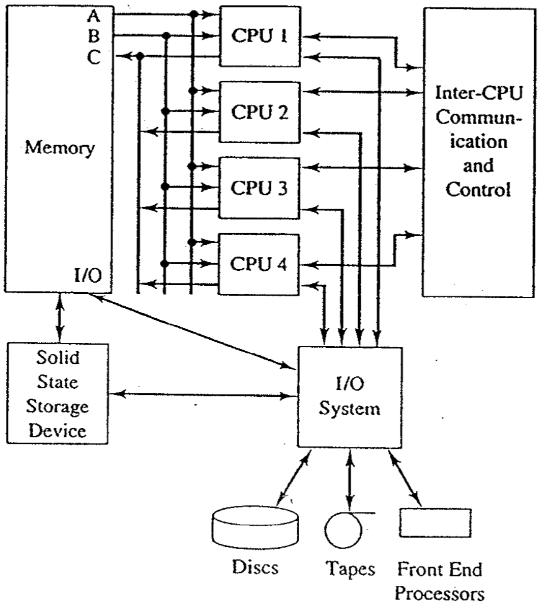
example vector processors – VMIPS, Cray X-MP/4

VMIPS



The basic structure of a vector architecture, VMIPS. This processor has a scalar architecture just like MIPS. There are also eight 64-element vector registers, and all the functional units are vector functional units. This chapter defines special vector instructions for both arithmetic and memory accesses. The figure shows vector units for logical and integer operations so that VMIPS looks like a standard vector processor that usually includes these units; however, we will not be discussing these units. The vector and scalar registers have a significant number of read and write ports to allow multiple simultaneous vector operations. A set of crossbar switches (thick gray lines) connects these ports to the inputs and outputs of the vector functional units.

Cray X-MP/4



Cray X-MP/4 Structure

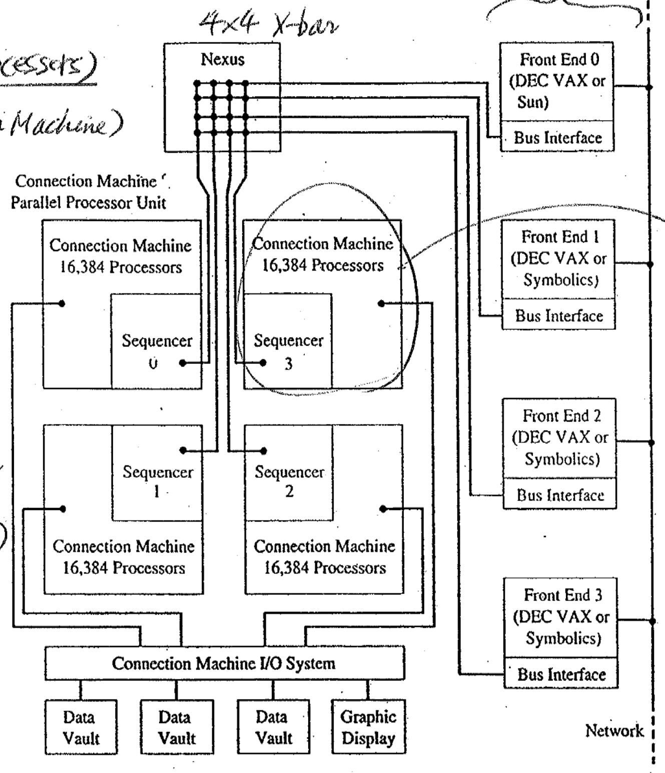
example array processors – CM-2, SPE in Cell processor

9f) SIMD (array processors)

=> CM-2 (Connection Machine)
MIT

total
∃ 64k 1-bit DPs

4* (1024 processor chips
16 PEs (DPs)/processor chip
(local mem in each proc. chip)

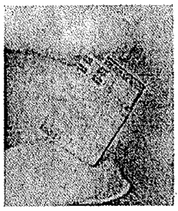


4 front-end processors
- handles (control flow, serial operations

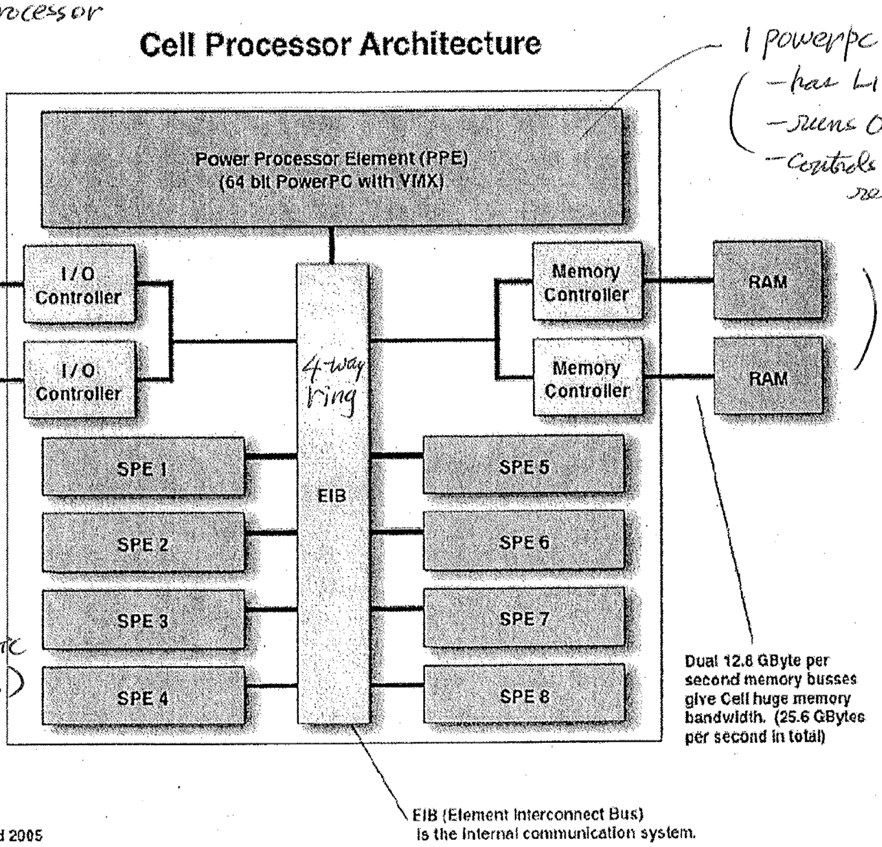
each Connection Machine:
(1 sequencer (CP), 16,384 DPs
- sequencer (CP) decodes Assembly code to nano-instr's
- all DPs receive same instr. from CP.

Architecture of CM-2 (Courtesy of Thinking Machines Corporation.)

4r) IBM/SONY Cell processor



Cell Processor Architecture



1 powerpc (PPE)
- has L1, L2 caches
- runs OS
- controls all resources

2 I/O channels

2 mem channels

Dual "configurable" High speed I/O channels.

(76.8 GBytes per second in total)

Dual 12.8 GByte per second memory busses give Cell huge memory bandwidth. (25.6 GBytes per second in total)

∃ 8 SPEs (synergistic processing ele.)

each SPE
- no opsys.
- no cache
- SIMD mode

© Nicholas Blachford 2005

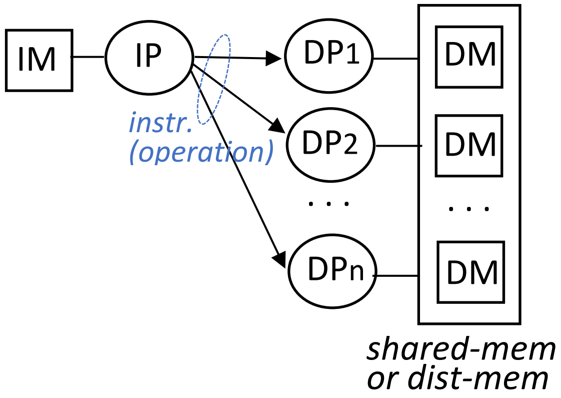
EIB (Element Interconnect Bus) is the internal communication system.

Figure 2.1: Overview of the architecture of a Cell chip

Concluding remarks

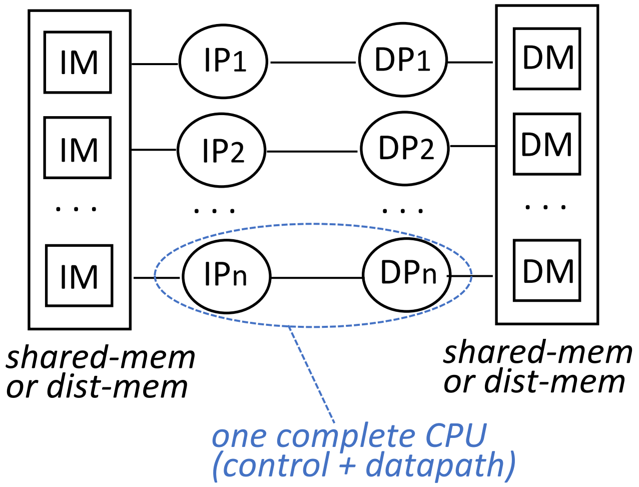
SIMD vs. MIMD

SIMD

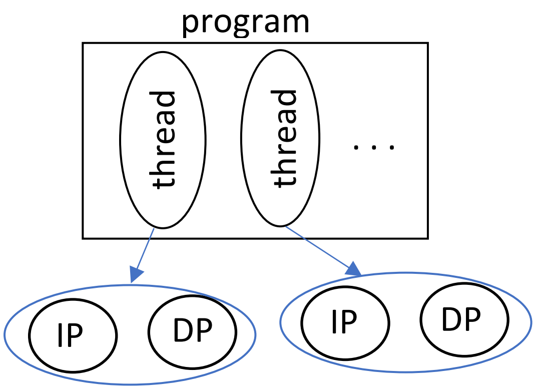


- efficient for data parallel applications, e.g., multimedia;
- each DP accesses DM for data and executes instr.;
- fine-grained data parallel operations on many DPs;

MIMD



- coarse-grained parallel applications;
- divide a task into pieces (either data-parallel or task-parallel) and run each thread/process on a processor/node;



MIMD – some thoughts

- shared-mem (e.g., SMP) vs. distributed-mem (e.g., COW) systems

2~100 processors ----- 2~1000 processors (nodes)

shared-mem comm. ----- msg-passing comm.
(supported by ISA, l/s) (supported by OP Sys)

low scalability ----- high scalability
(factory made – fixed) (cheaper way of building HP system)

Q1: Which system is more efficient in communication?

→ shared-mem system

ISA supports shared-mem based communication via load/store instr.

dist-mem system: packet forming, sending to next node's router, etc.

Q2: So, why not making all HP systems with shared-mem way?

→ # of nodes are limited, not scalable, expensive.

Q3: When is the dist-mem system advantageous?

→ small task – SMP is beneficial;

big task – cluster (COW) is beneficial since it hides comm. latency and there exist many nodes to assign divided tasks.

(relatively heavier computation than communication)