Parallel Processing

- Basic idea — speed up computation by using many processing elements (PEs)
- Applications (Hayes, 1988)
  - Long-range weather forecasting
  - Geophysical exploration via seismic data analysis
  - Fluid flow analysis
  - Medical diagnosis by computer-assisted tomography
  - Visual image processing
  - Nuclear reactor modeling
  - VLSI circuit design and simulation
- High-performance parallel computers are often referred to as *supercomputers*

Classification of Parallel Machines

- Michael Flynn (1966)
  - SISD — single instruction, single data
  - SIMD — single instruction, multiple data
  - MISD — multiple instruction, single data
  - MIMD — multiple instruction, multiple data
- More recent (Stallings, 1993)

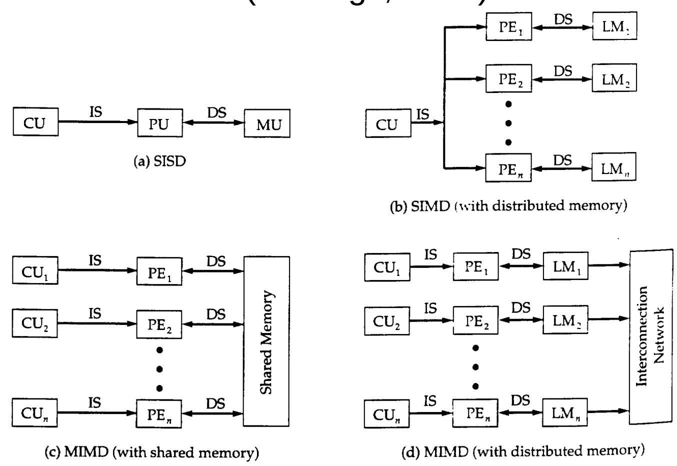


FIGURE 16.16. Alternative Computer Organizations

SIMD

- Single Instruction, Multiple Data
  - Many PEs, each doing the same thing at the same time, but on different pieces of data
    - Typically 1000s of PEs
  - Example: 10,000 PEs, each computing the wind chill for a location
  - Often combined with a SISD host, which:
    - broadcasts instructions to each SIMD PE,
    - performs sequential operations (branches, address calculation)
- Well suited for massive data parallelism — a large amount of data spread across a large number of PEs
  - Poorly suited for control parallelism — when **if** statements control the execution of each PE, so that some PEs execute, while others sit idle

SIMD Example

- Sum 100,000 numbers, using a SIMD computer with 10,000 PEs
- Host splits 100,000 numbers into 10,000 subsets, sends one subset to each PE
- Each PE computes the sum of its subset

```
/* A is full array on host, AL is local array */
sum = 0;
for (i = 0 ; i < 10 ; i++)          /* loop over each array */
    sum = sum + AL[i];              /* sum the local arrays */
```

- PEs add the partial sums using a divide and conquer approach

```
limit = 10000;
half = 10000;
repeat
    half = half / 2;                /* send vs. receive division */
    if (Pn >= half && Pn < limit)  send (Pn%half, sum);
    if (Pn < half)  sum = sum + receive( );
    limit = half;                   /* upper limit of senders */
until (half == 1);                  /* exit with final sum */
```

MIMD

- Multiple Instruction, Multiple Data
  - Small number of PEs, maybe doing same thing, maybe doing different things, but on different pieces of data
    - Typically 10s of PEs
- Well suited for control parallelism (PEs that aren't busy doing one thing can do something else), poorly suited for data parallelism (not enough PEs)
- Two key issues:
  - How do the PEs share data?
    - Shared memory — a single memory is shared between all PEs (not very common)
    - Distributed shared memory — each PE has its own memory, and they send messages containing data to others (more common)
  - How do the PEs coordinate?
    - Process for a bit, then synchronize…

Distributed-Memory MIMD Example (often called "Distributed Computing")

- Sum 100,000 numbers, using a distributed-memory MIMD computer with 10 PEs
- Host splits 100,000 numbers into 10 subsets, sends one subset to each PE
- Each PE computes the sum of its subset

```
sum = 0;
for (i = 0 ; i < 10000 ; i++)      /* loop over each array */
    sum = sum + AL[i];             /* sum the local arrays */
```

- PEs add the partial sums

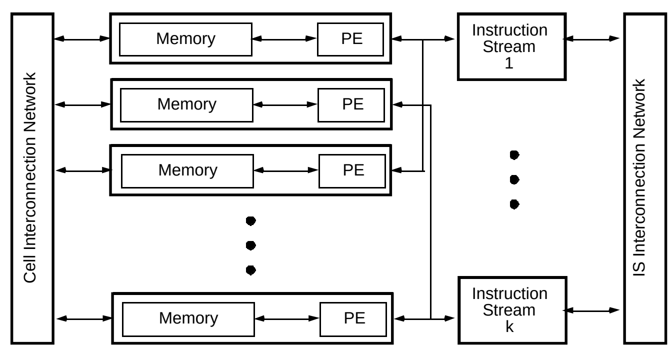
```
limit = half = 10;
repeat
    half = half / 2;              /* send vs. receive division */
    if (Pn >= half && Pn < limit)  send (Pn%half, sum);
    else if (Pn < half)  sum = sum + receive( );
    limit = half;                 /* upper limit of senders */
until (half == 1);                /* exit with final sum */
```

  - Receiving PE must *stall* until it receives a message from sending PE

Parallel Processing at Kent State

- Motivation:
  - SIMDs may be useful, but not many companies build SIMDs anymore
  - MIMDs aren't very good for massive data parallelism (not enough PEs)
- Solution: combine best features of SIMD (massive data parallelism) with best features of MIMD (control parallelism)
  - ASC model, developed at KSU
    - Base is SIMD — many simple PEs
    - PEs can be active or inactive
    - Active cells can perform search in constant time (e.g., find all red cars)
  - Multiple instruction stream ASC (MASC)
    - New innovation — multiple instruction streams — some PEs execute one instruction stream, while other PEs execute yet other instruction streams

Kent State MASC Model



- Many PEs provide massive data parallelism, just like normal SIMDs
- Multiple instruction stream (IS) processors supply different instructions to various PEs, providing efficient control parallelism, just like MIMDs
  - Some PEs do **then**, others do **else**
- Supports associative computing