Chapter 6

Parallel Processing

Evolution of parallel hardware

- I/O channels and DMA
- Pipelined functional units
- Vector processors (ILLIAV IV was built in 1974)

- Multiprocessors (cm* and c.mmp were built in the 70's)
- Instruction pipelining and superscalers

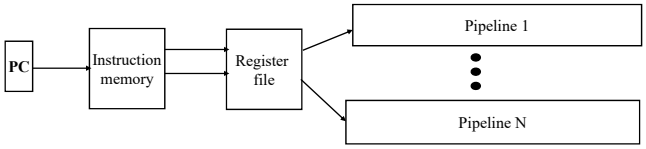
- Supercomputers - Massively Parallel Processors (Connection machine, T3E, Blue Gene, …)
- Symmetric Multiprocessors (SMPs)

- Distributed computing (Clusters, server farms, grids, clouds)

- Multi-core processors and Chip Multiprocessors
- Graphics Processor Units (GPU) as accelerators

Pipelining and Instruction Level Parallelism

- Pipelining overlaps various stages of instruction execution
- May use multiple pipelines → VLIW and Superscalers

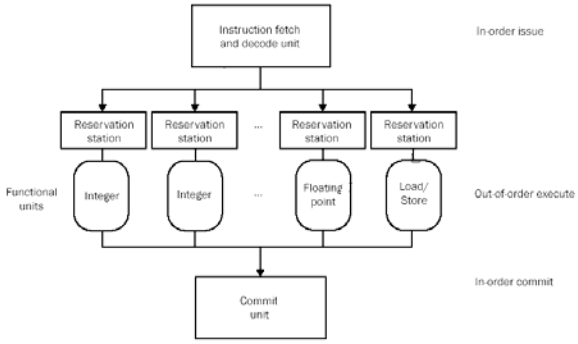


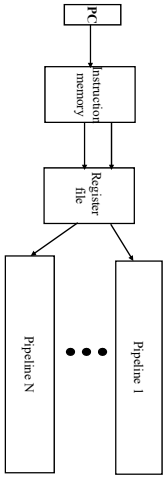
- Pipelining, however, has several limitations.
  - The speed of a pipeline is limited by the slowest stage.
  - Data and structural dependencies
  - Control dependencies

- **In-order issue/execution**: If an instruction cannot be issued because of potential hazard, the following instruction(s) cannot be issued.

Superscalar Execution

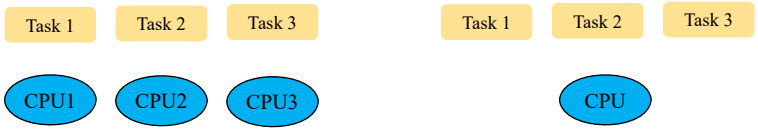
- **Out-of-order execution**: a more aggressive model where instructions can be issued to the pipeline(s) out of order. In this case, if an instruction cannot be issued because a potential hazard, the following instruction(s) can be issued (sometimes called dynamic issued).
- Usually, cannot keep all pipelines busy all the time



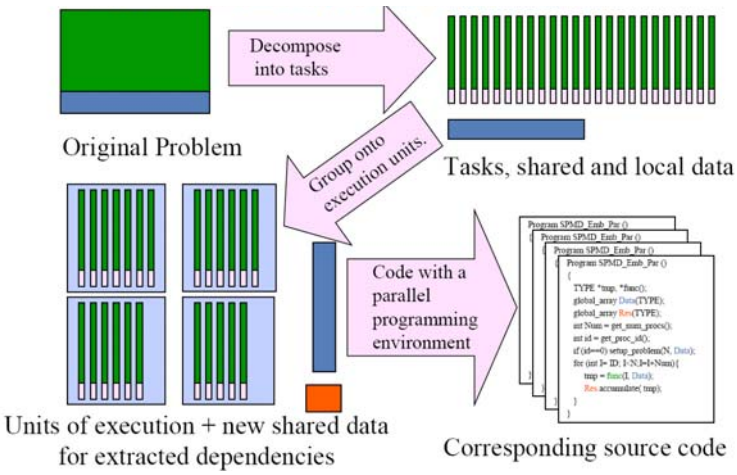


Exploring System Level Parallelism

- **Why ?**
  - ILP (Instruction Level Parallelism) is limited
  - Power consumption limits the increase in clock frequency
- **Multi-tasking:**
  - Divide your task into multiple sub-tasks to run on multiple CPUs.
  - Multi-threading is a form of multi-tasking (threads are light weight tasks).
- The number of tasks (threads) does not have to be equal to the number of CPU's – can multiplex tasks (threads) on a CPU.



How to create parallel applications

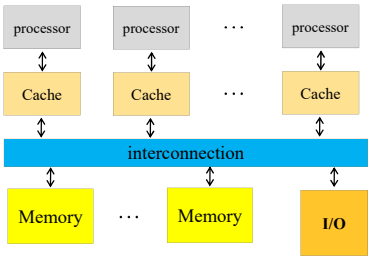


- Creation of multiple tasks (threads):
  - Automatically (for example, by the compiler)
  - Specified by the user (user needs to think *parallel*)

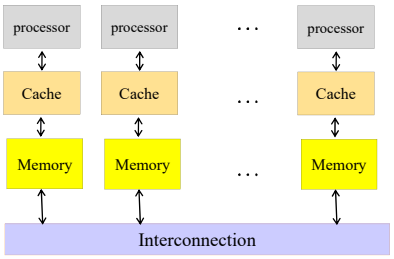
Multiprocessors

- **Idea:** create powerful computers by connecting many smaller ones
  good news: it works
  bad news: it is hard to write correct and efficient concurrent programs.

- Every CS/CoE professional has to deal with parallelism because Chip Multiprocessors are now the norm



SMP - Symmetric multiprocessors



Network connected MP

Speedup and efficiency (Section 6.2)

- For a given problem A, of size n, let $T_p(n)$ be the execution time on p processors, and $T_s(n)$ be the execution time (of the best algorithm for A) on one processor. Then,

  Speedup $S_p(n) = T_s(n) / T_p(n)$

  Efficiency $E_p(n) = S_p(n) / p$

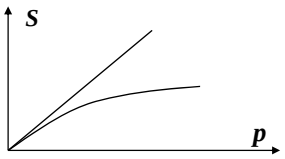
  Speedup is between *0* and p, and efficiency is between *0* and *1*.

Linear speedup:

Speedup is linear in p

Minsky's conjecture:

Speedup is logarithmic in p



Speedup and efficiency

Amdahl's law:

If f is the fraction of the task that can be executed in parallel

$T_p = (1\text{-}f) * T_s + f * T_s / p$

Speedup $S_p = \dfrac{1}{(1-f)+\dfrac{f}{p}}$ → (p is very large) $= \dfrac{1}{(1-f)}$

Maximum speedup, assuming infinite parallelism

- **Scalability**
  - If can maintain the efficiency for larger p independently of the size of the problem, n, then we have **strong scalability**.
  - If we can maintain the efficiency for larger p only by increasing the size of the problem, then we have **weak scalability.**

Scaling Example 1

- Problem *Dot(n)* → computing the dot product of two vectors $\sum_{i=0}^{n-1} x(i) * y(i)$
- *Dot(1000)* on a single processor: T_s = 1000 * (time to *add* + time to *multiply)*
- *Dot(1000)* on 10 processors (assuming t_{op} = time to *add* = time to *multiply*)
  - The 1000 multiplications can be done in parallel on the 10 processors
  - The 1000 additions cannot be done in parallel (accumulating 1000 values)
  - $T_{p=10} = 1000/10 \times t_{op} + 1000 \times t_{op} = 1100 \times t_{op}$
  - Speedup, $S_{10} = 2000/1100 = 1.82$ → (efficiency = 18.2%)
- *Dot(1000)* on 100 processors
  - Time = $1000/100 \times t_{op} + 1000 \times t_{op} = 1010 \times t_{op}$
  - Speedup, $S_{100} = 2000/1010 = 1.98$ → (efficiency = 2%)
- Amdahl law gives the maximum possible speedup

  f for the above problem is 0.5 → max speedup = 2.

Dot(1000) does not scale strongly when the number of processors changes from 10 to 100

Scaling Example 2

- Problem *Mat(n)* → add two $n \times n$ matrices then sum the diagonals of the result
- *Mat(10)* on a single processor: $T_s = (100 + 10) \times t_{op}$
- *Mat(10)* on 10 processors
  - The addition of two matrices can be done in parallel
  - The summation of 10 diagonal elements cannot be done in parallel
  - $T_{p=10} = 100/10 \times t_{op} + 10 \times t_{op} = 20 \times t_{op}$
  - Speedup, $S_{10} = 110/20 = 5.5$ (efficiency = 55%)
- *Mat(10)* on 100 processors
  - $T_{p=100} = 100/100 \times t_{op} + 10 \times t_{op} = 11 \times t_{op}$
  - Speedup, $S_{100} = 110/11 = 10$ (efficiency = 10%)
- Note: can use Amdahl law to find the maximum possible speedup
  - f for *Mat(10)* is 100/110 → max speedup = 11.

Mat(10) does not scale strongly when the number of processors changes from 10 to 100

Scaling Example 2 (cont.)

- *Mat(100)* → same problem but when matrix size is 100 × 100.
  - Single processor: $T_s = (10000 + 100) \times t_{op}$
  - $p = 10$ processors
    - Speedup, $S_{10} = 10100/1100 = 9.18$ (91.8% efficiency)
  - $p = 100$ processors
    - Speedup, $S_{100} = 10100/200 = 50.5$ (50.5% efficiency)

Mat(10) does not scale strongly when the number of processors changes from 10 to 100

- **However:**
  - *Mat(100)* on 10 processors → $S_{10} = 9.18$ → (91.8% efficiency)
  - *Mat (1000)* on 100 processors → $S_{100} = 91$ → (91% efficiency)
  - The efficiency of *Mat (n)* at $n=100$ and $p=10$ can be (almost) maintained at $p=100$ if we increase n to *1000*. Hence, *Mat(n)* is **weakly scalable**.

Mat() scales strongly when the number of processors changes from 10 to 100 and the matrix size increases from 100 to 1000