Parallel Algorithm Construction

Kudang B. Seminar

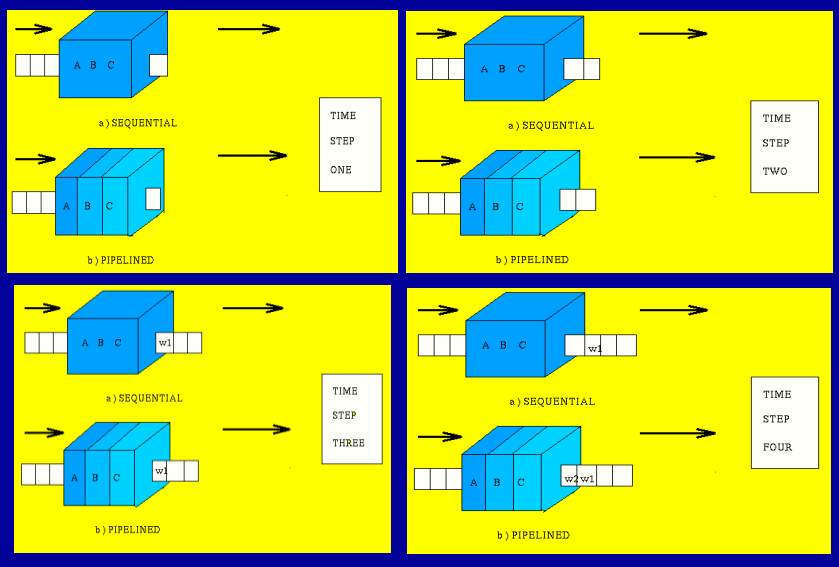
MIMD machines can be divided into 3 categories

- Pipelined Algorithms / Algorithmic Parallelism
- Partitioned Algorithms / Geometric Parallelism
- Asynchronous / Relaxed Algorithms

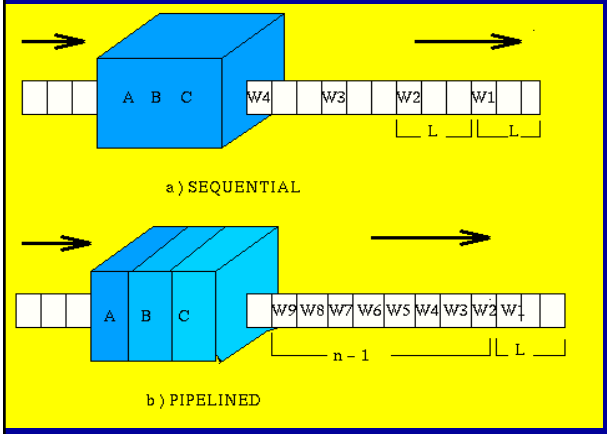
1. Pipelined Algorithms

- A pipelined algorithm is an ordered set of (possibly different) processes in which the output of each process is the input to its successor.
- Typically each processor forms part of a pipeline and performs only a **small part of the algorithm.** Data then flows through the system (pipeline) being operated on by each processor in succession.

Example: Sequential widget assembly machine



Example: Sequential widget assembly machine



Time Sequential = LTn
Time Parallel = [L + n-1]T

n is the number of widgets

T is the time for each step

L is the number of steps to be performed

This produces the first widget in 3 time units (as the sequential machine), but after this initial startup time one widget appears every time step.

Partitioned Algorithms

These algorithms arise when there is a natural way to decompose the data set into smaller "chunks" of data, which are then allocated to individual processors.

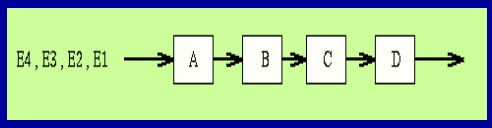
Each processor contains more or less the same code but operates on a subset of the total data.

The solution to these sub-problems are then combined to form the complete solution. This combining of solutions usually implies communication synchronization among the processors.

Partitioned algorithms are sometimes called synchronous algorithms

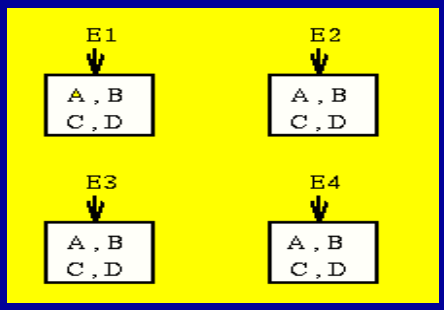
Example:

Say an algorithm consists of 4 parts A, B, C and D and this algorithm is to operate on a data set E consisting of 4 subsets E1, E2 , E3 and E4



pipelined algorithm

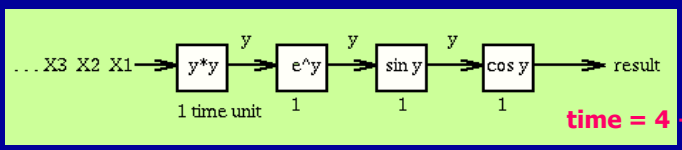
algorithm is distributed among the processors



partitioned algorithm

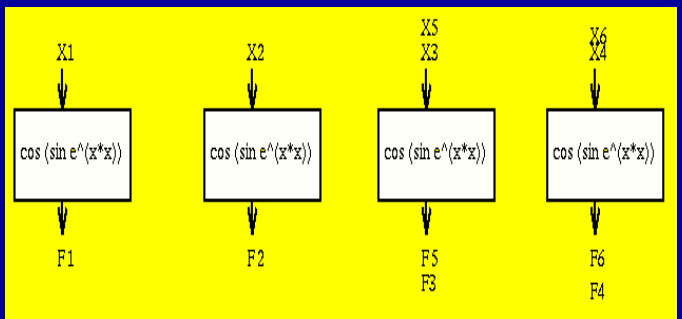
data is distributed among the processors

Example: $F_i = \cos(\sin e^{sqr(xi)})$ for x1, x2 ,....x6 using 4 processors.



pipelined algorithm

time = 4 + (6-1) = 9 units



partitioned algorithm

time = 8 units

Asynchronous / Relaxed Parallelism

There is no explicit dependency between processes.

Algorithms never wait for input. If they are ready they use the most recently available data.

The idea in using asynchronous algorithms is that all processors are kept busy and never remain idle (unlike synchronous algorithms) so speedup is maximized.

Example: Newton Raphson iteration for solving F (x) =0

$$X_{n+1} = X_n - F(X_n)/F'(X_n) \quad \ldots\ldots (1)$$

generates a sequence of approximations to the root, starting from a value X0.

Say we have 3 processors
P1 : given x, P1 calculates F (x) in time t1 units and sends it to P3
P2 :given y, P2 calculates F'(y) in time t2 units and sends it to P3
P3 : given a, b, c, P3 calculates d = a - b/c in time t3 units; if | d-a | > Epsilon then d is sent to P1 and P2 otherwise d is output

Example: Newton Raphson iteration for solving F (x) =0

Serial Mode
P1 computes F(Xn) then P2 computes F'(Xn) then P3 computes Xn+1 using (1)
So time per iteration is t1 + t2 + t3
If k iterations are necessary for convergence then total time is k (t1 + t2 + t3)

Synchronous Parallel Mode.
P1 and P2 compute F(Xn) and F'(Xn) simultaneously and when BOTH have finished the values F(Xn) and F'(Xn) are used by P3 to compute Xn+1
Time per iteration is max(t1, t2) + t3.
Again k iterations will be necessary so total time is k [max(t1, t2) + t3] X1 = X0 - F(X0)/F'(X0) ...etc

Example: Newton Raphson iteration for solving F (x) =0

Asynchronous Parallel Mode
P1 and P2 begin computing as soon as a new input value is made available by P3 and they are ready to receive it,
P3 computes a new value using (1) as soon as EITHER P1 OR P2 provide a new input i.e. (1) is now of the form
Xn+1 = Xn - F(SXi)/F'(Xj)

Time per iteration is at most min(t1, t2) + t3 [but may be as low as t3 - e.g. if P1 and P2 complete at consecutive time steps]
we cannot predict how many iterations will be necessary
Say t1 = 2, t2 = 3 and t3 = 1.

NOTE

At time 11 P2 has the choice of using X3 to calculate F'(X3) or X4 to calculate F'(X4), i.e. omit X3. Which choice is made should be determined experimentally to see which gives the best results.

We could relax the parameter i.e. use X4 or we could synchronise with the parameter i.e. using X3.