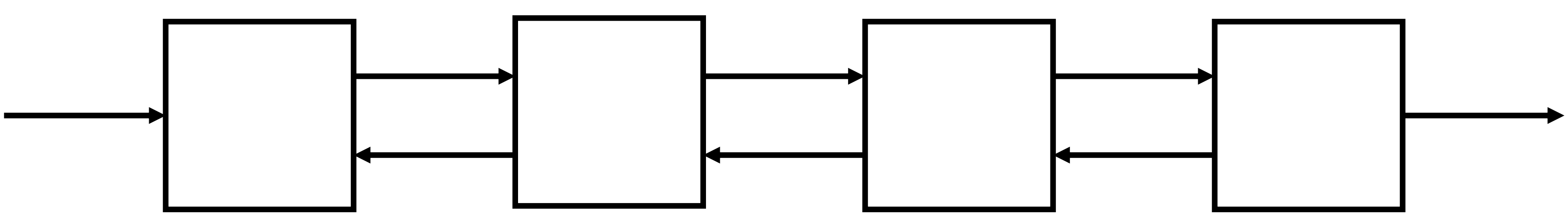
Parallel Computing

Processors with Connections

Systolic Arrays

The Simplest Systolic Array:

- n simple processors
- Arranged in a row
- Each processor can exchange information only to its neighbours on its left and right
- Processor 1 takes an input
- Processor n give an output



Using Parallel Computing to Solve the n-body Problem

Even the simplest systolic array can solve many problems much more quickly than a single processor. E.g.:

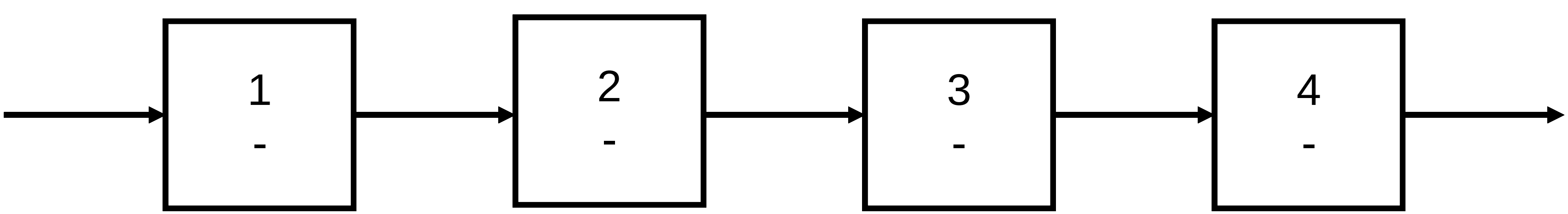
- n-body problem – Compute the path of n bodies moving through space under the influence of their mutual and combined gravitational attractions
- A sequential computer, can solve this in $O(n^2)$ steps since there are $(n^2 - n)/2$ attractions
- A systolic array can achieve this in $O(n)$ steps

Using Parallel Computing to Solve the n-body Problem

- Each cycle of an n-body computation would involve calculating, for each body, the summed attractions of the other $n-1$ bodies
- Assume each processor carries a program to calculate these attractions: $F_{ij} = km_i m_j / d_{ij}^2$, given values for k, m_i, m_j, and the co-ordinates of i and j
- For each body, starting with B_n, co-ordinates and mass is inputted to P_1 - they are then passed rightwards until each processor, P_i holds the co-ordinates and mass of a unique body, B_i - This takes n steps

Using Parallel Computing to Solve the n-body Problem

- There is then the same input, but this time, as the information for each new body B_j arrives from the left, P_i executes (using B_i from before):
    - Calculate d_{ij}
    - Calculate the Force F_{ij}
    - Add F_{ij} to the force previously calculated
- It takes $2n$ steps to complete this process:



Using Parallel Computing to Solve the n-body Problem

- Another program shifts the final F values across to the output
- Absorbing n more steps

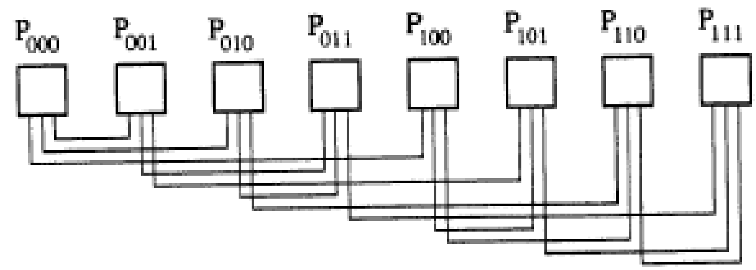
Using a Systolic Array takes $4n$ steps if each shift of information and execution of the algorithm takes 1 unit of time

Clearly this can be computed in linear, $O(n)$, time

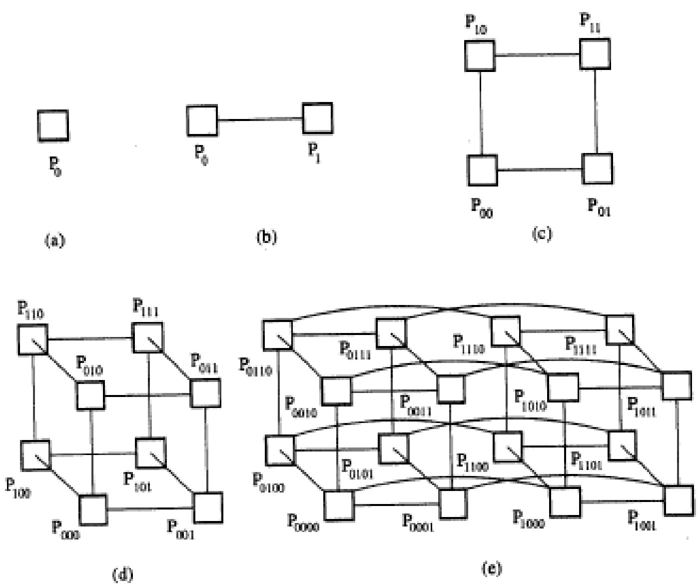
Systolic Arrays

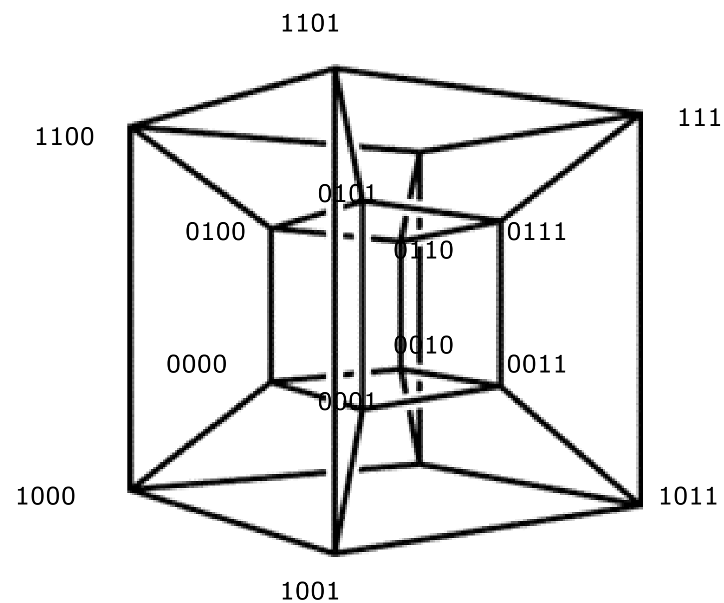
- In reality, systolic arrays are much more sophisticated than a single row of processors, and there are other types of parallel computers
- More general and powerful schemes can be represented by a cube-connected computer

Hypercubes



- A d-dimensional hypercube allows the connection between n processors in a cube-connected computer
- Each processor is a vertex of the cube, giving $n = 2^d$





Matrix Multiplication

- Something like a hypercube can also help us to solve some problems more quickly than with a sequential machine – E.g. Matrix Multiplication:
- Given two $n \times n$ arrays X and Y, it would take a sequential computer somewhere between $O(n)^2$ and $O(n)^3$ steps to produce the n^2 elements in the product, Z

Matrix Multiplication

- For each element of the product matrix Z:

$$Z_{ij} = \sum_{k=1}^{n} x_{ik} y_{kj}$$

- The optimal number of processors is n^3
- d separate links are required so that one processor might communicate with another
- Hence $d = logn^3 = 3\log n$
- So any communication between processors alone will require $O(logn)$ steps

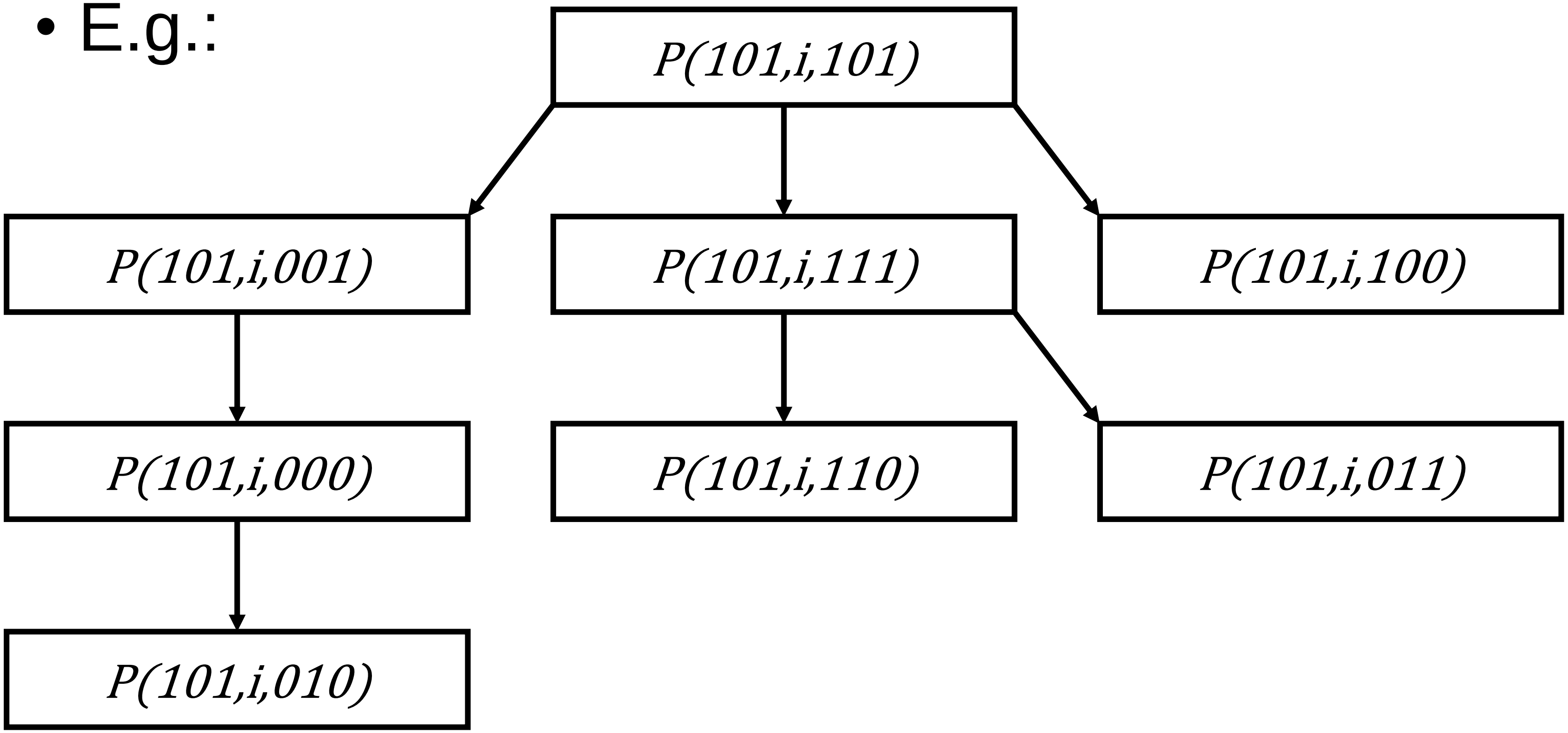
Matrix Multiplication

- The first step is to distribute the array elements from X and Y throughout the n^3 processors so that $P(k, i, j)$ contains x_{ik} and y_{kj}
- This is done by transmitting the x_{ik} data from $P(0, i, j)$ to $P(k, i, j)$ by a route of fewer than $logn$ processors
- At each stage of the journey, the next processor it goes to is the one that has an index that is 1 bit closer to k – E.g. where $k = 5 = 101$:

$$P(000, i, j) \rightarrow \rightarrow P(100, i, j) \rightarrow P(101, i, j)$$

Matrix Multiplication

- Now that $P(k,i,j)$ has received x_{ik}, it is broadcasted to $(k,i,0)$, $(k,i,1)$, …, (k,i,n)
- E.g.:



Matrix Multiplication

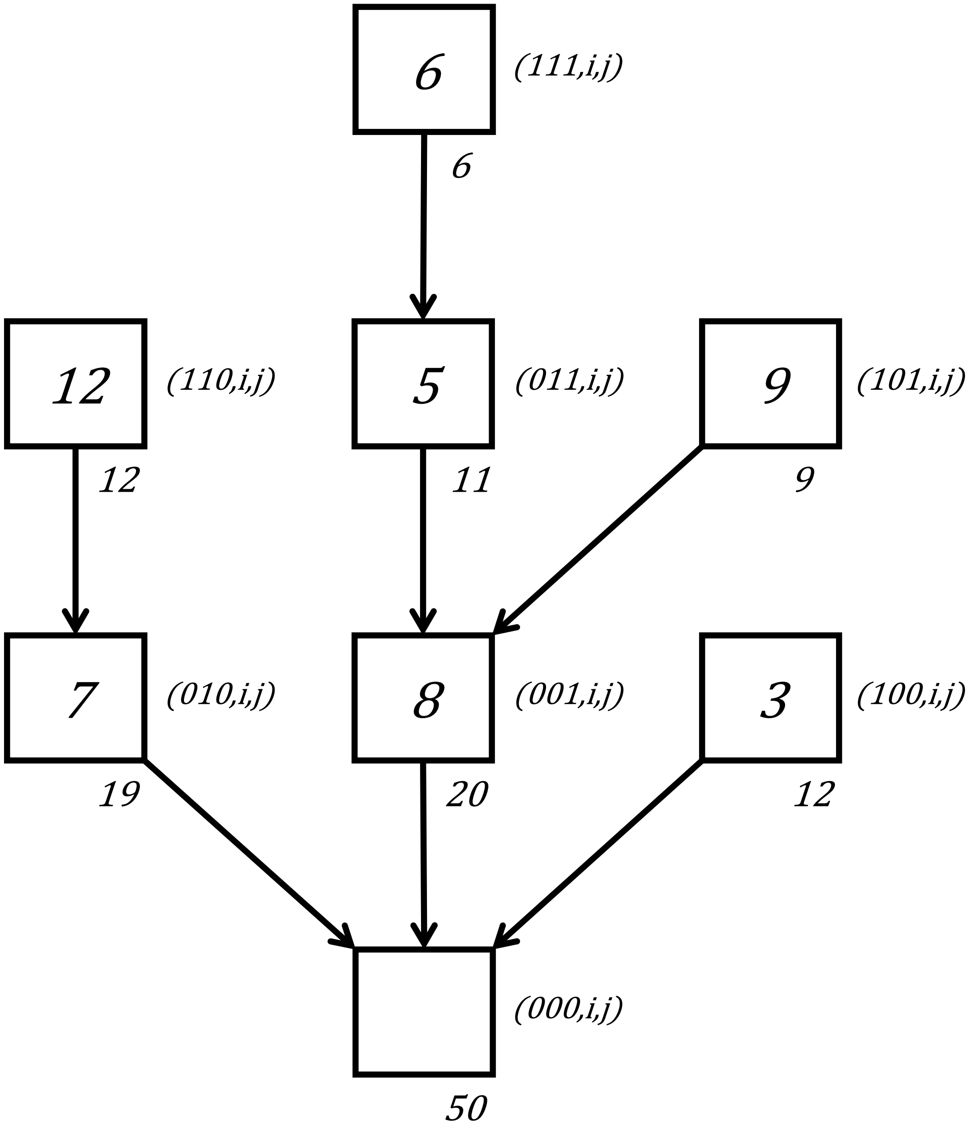
- Now the product $x_{ik} \times y_{kj}$ is calculated and stored in the relevant processor

Matrix Multiplication

- The products are drawn from all the processors $P(1, i, j)$, $P(2, i, j)$, …, $P(n, i, j)$ and stored in $P(0, i, j)$
- This is done by a continual ‘fanning in’ of accumulated sums, always changing one of the bits to 0
- All transmission and summing are carried out in parallel so the number of steps is bounded by the maximum distance between two processors, $log n$

Matrix Multiplication

E.g.



Matrix Multiplication

- Using a cube-connected computer, matrix multiplication can now be solved in $O(logn)$ steps rather than $O(n)^{2.61}$