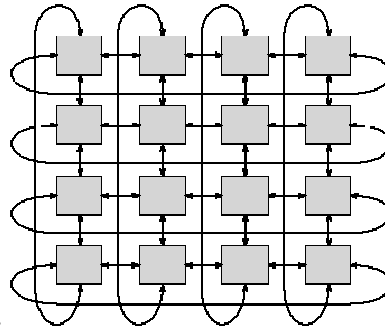


Homework #4
Due Friday 10/27/06 at 5pm

1. **Interconnect.** A k -ary d -cube is an interconnection network with k^d nodes, and is a generalization of the mesh and hypercube interconnects. There are k nodes along all d axes with end around connections. A 4-ary 2-cube is pictured below.



- a. How many links are there in a k -ary 2-cube? A k -ary 3-cube? A k -ary d -cube?

k -ary 2-cube: $2k^2$, k -ary 3-cube: $3k^3$, k -ary d -cube: dk^d

- b. What are the diameter and bisection bandwidth of a k -ary d -cube? Assume that the bandwidth of a link is the quantity 'B.'

Diameter: $d\lfloor k/2 \rfloor$, bisection bandwidth: $2Bk^{d-1}$

- c. What is the broadcast time for short messages, assuming a message start time α ?

$\alpha d\lfloor k/2 \rfloor$ same as the diameter.

- d. A ring is a special case of k -ary 1-cube. Give a strategy that maps a ring with k^d nodes onto a k -ary d -cube with k^d nodes. Let $\text{ring}(i)$ represent node (i) of a ring, and $\text{kdcube}(i_0, i_1, \dots, i_{d-1})$ represent node $(i_0, i_1, \dots, i_{d-1})$ of a k -ary d -cube. Start by working out the special case of $d=2$ or 3, then generalize to d -dimensions. There are many possible mappings, but choose just one. Describe in words, with an appropriate diagram as necessary.

There are two cases; when k is odd and even.

i) When k is even.

For example, 4-ary 2-cube has a mapping from a ring as follows:

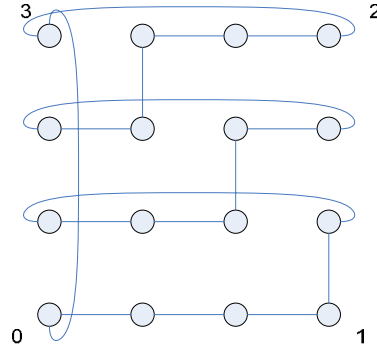


Fig1. Traversing in 2D

This strategy holds for every even number k . (Actually it holds for odd number too.) As we can see, if we start traversing from node 0, then the last node we visit is node 3 (-1). By symmetry, the last node can be node 1 (+1). The next step is stacking up this plane k times, which is k -ary 3-cube. It is always possible. At the lowest plane, we start from node 0 and end at node 3. Then, we go up one level higher. We start from node 3 at second level. By symmetry we can go to node 0 (+1) or node 2 (-1) at the end of your traversal. What we have to care is the start and end points at each level. If we start traversing at node p ($0 \leq p \leq 3$), we can reach node $(p+1)$ or $(p-1) \bmod 4$ at the end of traversing. The goal is that the traversing should end at node 0 at level k because node 0 at level 1 and node 0 at level k are connected. Let x denotes the number of planes that start at node p and end at node $(p+1) \bmod 4$. Let y denotes the number of planes that start at node p and end at node $(p-1) \bmod 4$. The goal can be represented by;

$$\begin{aligned} x + y &= k; \\ x - y &= 0 \bmod 4; \end{aligned}$$

Solving the equations yields $x = y = k/2 \bmod 4$. For example, 4-ary 3-cube can be mapped to a ring as follows. The blue nodes denote the starting points at each level and the red nodes denote the last nodes after traversing using the Fig 1 strategy.

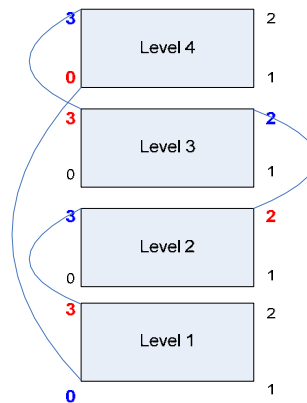


Fig 2. 4-ary 3-cube

ii) When k is odd.

The basic strategy is almost the same as when k is even. In 2D, we can traverse all the nodes as follows;

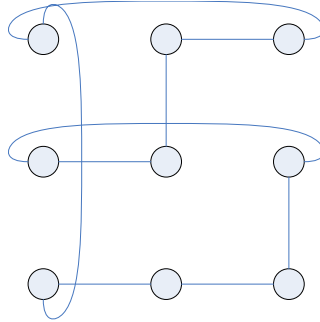


Fig 3. 3-ary 2-cube

But in 3D, we can not go back to node 0 at level k when k is odd. However, there is another traversing strategy;

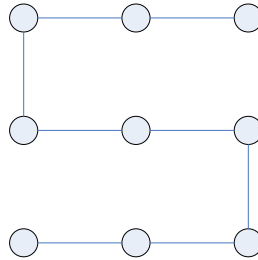


Fig 3. Alternative traversing strategy

This method starts at node p and arrives at node $(p+2) \bmod 4$. Therefore, if we combine all the three operations $(+1, -1, +2)$;

$$\begin{aligned} x + y + z &= k \\ x - y + 2z &= 0 \bmod 4 \end{aligned}$$

The set of equations always has solutions because x, y, z are all integer.

4D case can be similarly extended from 3D version.

2. *Parallel prefix.*

The prefix sum (also called a sum-scan) of a sequence of numbers x_k is a sequence of running sums S_k defined as follows

- $S_0 = 0$
- $S_k = S_{k-1} + x_k$

Thus, $\text{scan}(3, 1, 4, 0, 2) = (3, 4, 8, 8, 10)$

a. Design an algorithm for prefix sum based on the hypercube interconnect.

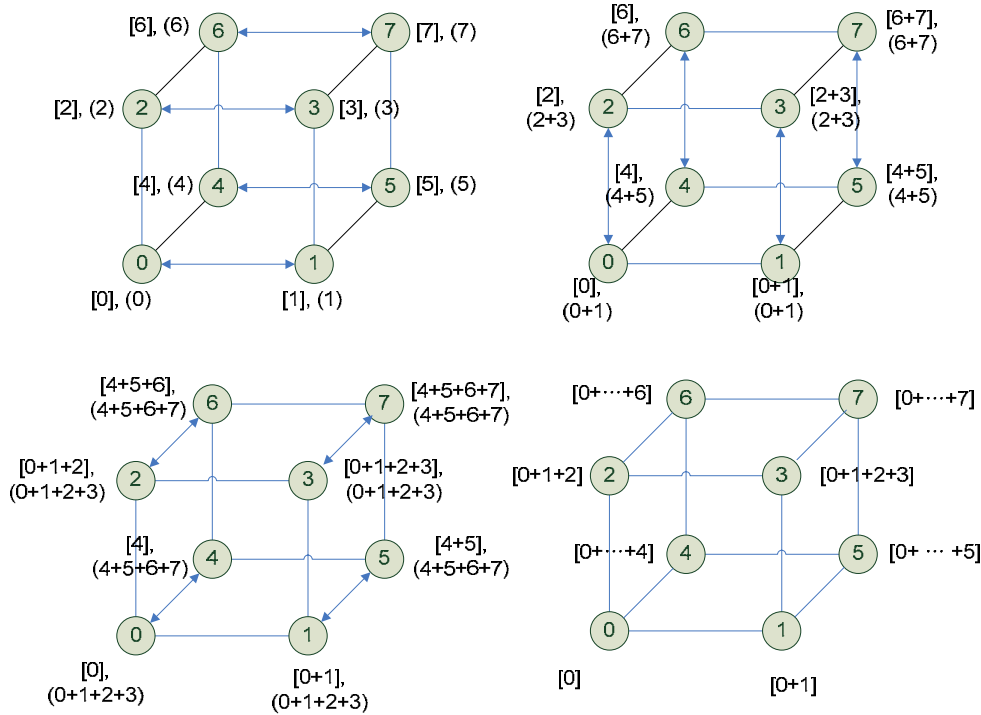
```

// my_id: rank
// my_number:  $x_k$ 
// d: dimension
// result:  $S_k$ 
Procedure PREFIX_SUMS_HCUBE(my_id, my_number, d, result)
Begin
  result := my_number;
  msg := result;
  for i:=0 to d-1 do
    partner := my_id XOR  $2^i$ ;
    send msg to partner;
    receive number from partner;
    msg := msg + number;
    if(partner < my_id) then result:= result + number;
  endfor;
end PREFIX_SUMS_HCUBE

```

Each node keeps two values: result and outgoing message. Result is a local prefix sum at the node and the outgoing message is sent to neighbor nodes. The difference between the two is the result value is updated only when message is received from nodes of which id is smaller than my_id.

[[: result, (): outgoing msg



b. Derive an accompanying performance model

The total number of iteration is $\log P$, where P denotes the number of nodes.

Therefore, each node sends $\log P$ messages.

At each communication, the size of message increases exponentially up to $p/2$.

Therefore, the total size of message being sent is $(p-1) \cdot (1 + 2 + \dots + 2^{k-1})$, if $p = 2^k$

$$\text{Communication Cost} = \alpha \log P + 4 \beta (P-1)$$

3. Communication optimization.

- a. In class we studied the 5-point 2D point Jacobi solver for Poisson's equation. In some applications, we need higher accuracy, and we solve the equation using 9-point stencil, updating each value of the solution as a function of its 8 nearest neighbors, according to the following algorithm. Assume that U_{new} and U are dimensioned as $N+2 \times N+2$ arrays.

```
forall (i=1:N-1, j=1:N-1)
    Unew[i,j] = (-20*U[i,j]+4*[U[i,j+1]+U[i,j-1] + U[i+1,j]+U[i-1,j]] +
                U[i+1,j+1]+U[i+1,j-1] + U[i-1,j+1]+U[i-1,j-1]) / (6*h)

forall (i=1:N, j=1:n)
    U[i,j] = Unew[i,j]
```

Derive a performance model for the parallel implementation of the 9-point stencil computation, expressing the parallel running time T_p as a function of P and N . Clearly designate the separate costs of computation and communication.

Assumption: 1) each assignment statement takes 1 unit time, 2) The data type for U and U_{new} is double (8bytes), 3) 2D decomposition, and 4) $P = k^2$ for some k .

$$T_p(N, 1) = 2N^2;$$

$$\text{Computation cost: } 4(\alpha + 8 \beta N / \sqrt{P}) + 4(\alpha + 8 \beta)$$

The first term represents the communication between Manhattan directions. The second term is required for communication between NW, NE, SW, and SE directions.

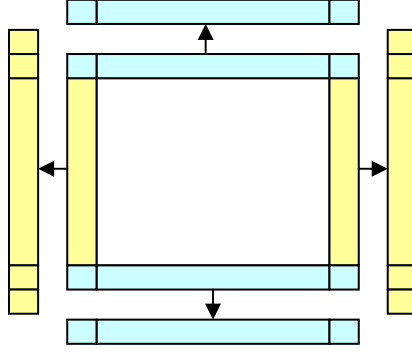
$$\text{Computation cost: } 2(N / \sqrt{P} * N / \sqrt{P})$$

The area that one node has to compute is $N / \sqrt{P}$ and there needs two update.

$$\text{Therefore, } T_p(N, P) = 2(N / \sqrt{P} * N / \sqrt{P}) + 4(\alpha + 8 \beta N / \sqrt{P}) + 4(\alpha + 8 \beta)$$

- b. A straightforward approach to updating ghost cells requires communication from the 8 nearest neighbor processors. However, there is a different communication algorithm that involves only 4 messages from nearest neighbors along the 4 Manhattan directions. Derive this algorithm and compare the communication cost with that of the 8-message method of treating the 8 neighbors. Determine which method incurs a higher communication overhead, expressed as a fraction of T_p . Express your answer in terms of the parameters α , β , N , and P .

Each node needs to send values of four corners to NW, NE, SW, and SE nodes. However, the values are also sent when communicating with N, S, E, and W directions. When sending to E and W, if we contain additional values received from N and S nodes.



As a result, we can save four messages communicating with diagonal nodes.

$$T_p(N, P) = 2(N/\sqrt{P} * N/\sqrt{P}) + 4(\alpha + 8\beta N/\sqrt{P}) + 4*8\beta$$

$$\begin{aligned} \text{Performance gain} &= 4(\alpha + 8\beta N/\sqrt{P}) + 4(\alpha + 8\beta) - \{4(\alpha + 8\beta N/\sqrt{P}) + 4*8\beta\} \\ &= 4\alpha \end{aligned}$$