**Parallel Computing**

**Assignment-6**

**Akarsh Gupta**

**800969888**

**Solutions:**

1. **Reduction:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Case** | **Most Loaded Link** | **Most Loaded Node** | **Longest Chain of Communication** |
| Reduce Star on Chain | The link between  ‘0-1’ is the most loaded link.  On link i to i+1, where i is between [0,P-2], Data transit is O(P-(i+1)).  So, for link ‘0-1’, all the data from P-1 nodes will flow.  O(P-1). | Node 0 is the most loaded node. As, all the nodes send data to node 0.  Data on node 0 is O(P-1). | The communication going from node P-1 to Node 0 is longest with length O(P-1). |
| Reduce Start on Clique | On any link 0-i, where i is [1,P-1], Data transit is O(1)  Rest all the links are not used at all.  Most Loaded Link can be any of the link associated to Node 0. | Hence, Node 0 is the most loaded node with O(P-1) data. | All communications are of equal length of O(1). |
| Reduce Chain on Chain | All links have equal O(1) data transit.  All links are most loaded. | O(1) for all the nodes.  Hence all nodes communicates most data. | All nodes communicate with adjacent nodes only. So, all communications are of equal length with length O(1). |
| Reduce Chain on Clique | On any link i-i+1, where i is [0,P-2], Data transit is O(1)  Rest all the links are not used at all.  Most Loaded Link can be any the link having O(1) data transit. | O(1) for all the nodes as all the nodes handles same amount of data. | Most loaded nodes are first (logP-1) nodes.  The amount of data they handle is Log(P) |
| Reduce Tree on Chain | Link from nodes 2^(logP)-2 – 1 to P/2 have a load of O(P/2) | All Nodes from 2^(logP)-2 – 1 to P/2  The amount of data they handle is Log(P). | Length of longest chain of communication is P/2. This occurs between (P/2) number of pair of nodes. |
| Reduce Tree on Clique | On any link i-i+1, where i is [0,P-2], Data transit is O(1). | Most loaded nodes are first (logP-1) nodes.  The amount of data they handle is Log(P). | All communications are of equal length of O(1). |

**Of all the above algorithm,** Reduce Tree algorithm is the best algorithm for the above network topologies as there is less communication and loads even on the most loaded link and node.

1. **A.) Round Robin**

**Algorithm**:

Let p be the current Process and hk-1 be the previous iteration data. N is size of array and P is total number of processes.

**Compute\_heat\_Round\_Robin(N,p,P, hk-1)**

{

int curr = p;

while(curr < N)

{

if(curr == 0)

{

send hk-1[curr] to p+1;

recv hk-1[curr+1] from p+1;

hk[curr] = (2\* hk-1[curr] + hk-1[curr+1])/3;

}

else if(curr == P-1)

{

send hk-1[curr] to p-1;

recv hk-1[curr-1] from p-1;

hk[curr] = (2\* hk-1[curr] + hk-1[curr-1])/3;

}

else

{

if(p == 0)

{

send hk-1[curr] to P-1;

send hk-1[curr] to p+1;

recv hk-1[curr-1] from P-1;

recv hk-1[curr+1] from p+1;

}

else if(p == P-1)

{

send hk-1[curr] to p-1;

send hk-1[curr] to 0;

recv hk-1[curr-1] from p-1;

recv hk-1[curr+1] from 0;

}

else

{

send hk-1[curr] to c-1;

send hk-1[curr] to c+1;

recv hk-1[curr-1] from c-1;

recv hk-1[curr+1] from c+1;

}

hk[curr] = (hk-1[curr-1] + hk-1[curr] + hk-1[curr+1])/3;

}

curr += P;

}

return;

}

**Communication per iteration:**

For each element, there is 2 communications are occurring except for element 0 & N-1 which has only 1 communication.

Hence,

Total Communications: O(2N-2)

**B.) Block:**

**Algorithm**:

Let p be the current Process and hk-1 be the previous iteration data. N is size of array and P is total number of processes.

**Compute\_heat\_Block(N,p,P, hk-1)**

**{**

start = p\*(N/P);

end = (p+1)\*(N/P);

If(p == 0)

{

send hk-1[end-1] to p+1;

recv hk-1[end] from p+1;

}

else if(p == P-1)

{

send hk-1[start] to p-1;

recv hk-1[start-1] from p-1;

}

else

{

send hk-1[start] to p-1;

send hk-1[end-1] to p+1;

recv hk-1[start-1] from p-1;

recv hk-1[end] from p+1;

}

for(i = start; i< end; i++)

{

if(i == 0)

{

hk[i] = (2\*hk-1[i] + hk-1[i+1])/3;

}

if(i == N-1)

{

hk[i] = (2\*hk-1[i] + hk-1[i-1])/3;

}

else

{

hk[i] = (hk-1[i-1] + hk-1[i] + hk-1[i+1])/3;

}

}

return;

}

**Communication per iteration:**

For each node, there is 2 communications are occurring except for node 0 & P-1 which has 1 communication.

Hence, Total Communications: O(2P-2)

I will use Block Data partition as it has less communication between nodes.

1. **Horizontal:**

**Dense\_Horizonatal(N, c, P, A, x)**

{

start = c\*(N/P);

end = c+1\*(N/P);

count = 10;

while(count--)

{

// computing y = Ax

for(i = start; i<end;i++)

{

y[i]=0;

for(j = 0;j<N;j++)

{

y[i] += A[i][j]\*x[j];

}

x[i] = y[i]; // computing x = y

}

}

return;

}

**Memory Required**: O(N\*N/P + N + N/P) [A+x+y]

**No communication required here.**

1. **Vertical:**

**Dense\_Vertical(N,c,P,A,x)**

{

start = c\*(N/P);

end = c+1\*(N/P);

count = 10;

while(count--)

{

// computing y = Ax

for(i = 0; i<N;i++)

{

if(c == 0)

{

y[i] = 0;

}

else

{

recv y[i] from c-1;

}

for(j = start;j<end;j++)

{

y[i] += A[i][j]\*x[j];

}

if(c == P-1)

{

x[i] = y[i]; // computing x = y

}

else

{

send y[i] to c+1;

}

}

}

return;

}

**Memory Required**: O(N\*N/P + N/P + N) [A+x+y]

Communications happens in a chain like form here i.e. from link j-j+1 for every i , i=[0,N-1] and j=[0,N-2]  
**Total communication**: O(N\*N-1) or O(N2)  
**Communication per link**: O(N) for the links mentioned above, 0 otherwise

**Communication per Node**:O(N)

1. **Block:**

**Dense\_Block(N,c,P,A,x)**

{

startx = (c%sqrt(P))\*(N/sqrt(P));

endx = (c%sqrt(P)+1)\*(N/sqrt(P));

starty = (c/sqrt(P))\*(N/sqrt(P));

endy = (c/sqrt(P)+1)\*(N/sqrt(P));

count = 10;

while(count--)

{

// computing y = Ax

for(i = startx; i<endx;i++)

{

if(c%sqrt(P) == 0)

{

y[i] = 0;

}

else

{

recv y[i] from c-1;

}

for(j = starty;j<endy;j++)

{

y[i] += A[i][j]\*x[j];

}

if(c%sqrt(P) == sqrt(P)-1)

{

x[i] = y[i]; // computing x = y

}

else

{

send y[i] to c+1;

}

}

}

return;

}

**Memory Required**: O(N/sqrt(P)\*N/sqrt(P) + N/sqrt(P) + N/sqrt(P)) = O(N\*N/P + 2N/sqrt(P)) [A+x+y]

**Total communication**: O(N\*N-1) or O(N2)  
**Communication per link**: O(N/sqrt(P)) for links j-j+1, for all i, where i=[0,N-1] and j=[0,N-2], j+1%sqrt(P) != 0, zero otherwise

**Communication per Node**: O(N/sqrt(P)), for every Node i, where i%sqrt(P) != sqrt(P) -1