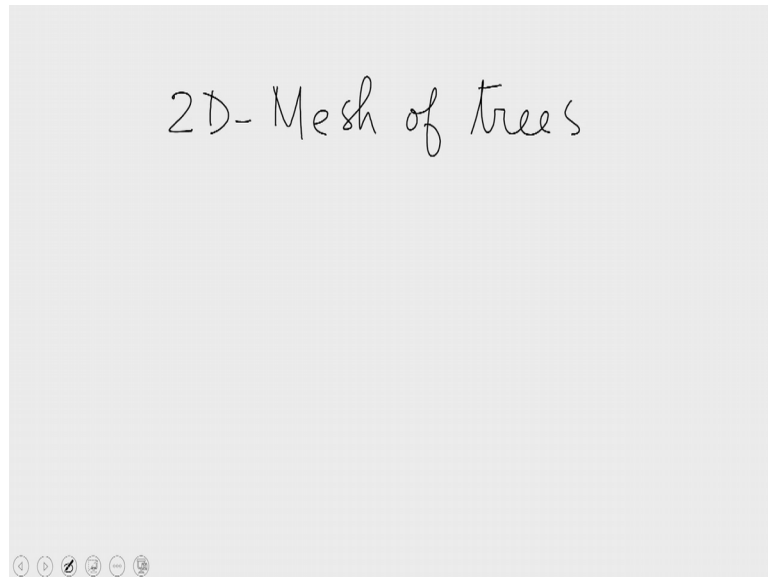


Parallel Algorithms
Prof. Sajith Gopalan
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture - 28
Mesh of Trees, Hypercube

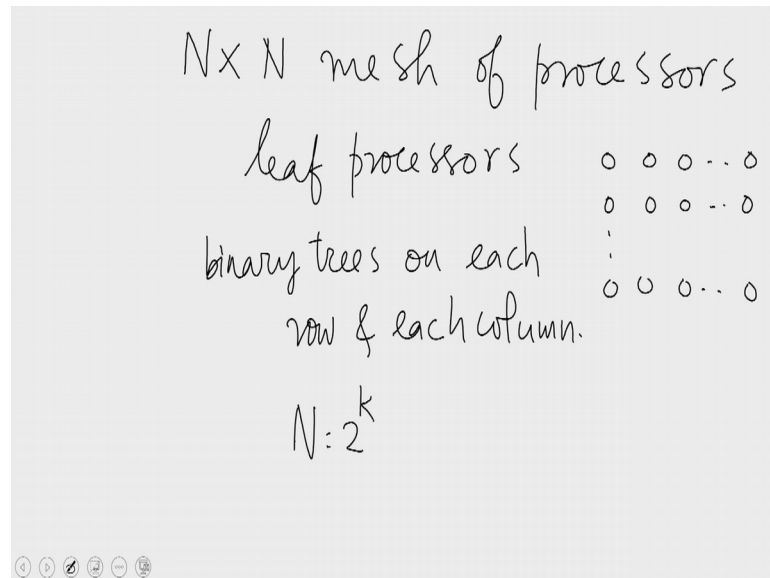
Welcome to the 28h lecture of the MOOC on parallel algorithms. In the last few lectures we have been studying algorithms designed on multi dimensional measures. Today we shall see some architectures that are richer than measures. The first one that we see is the mesh of trees.

(Refer Slide Time: 00:49)



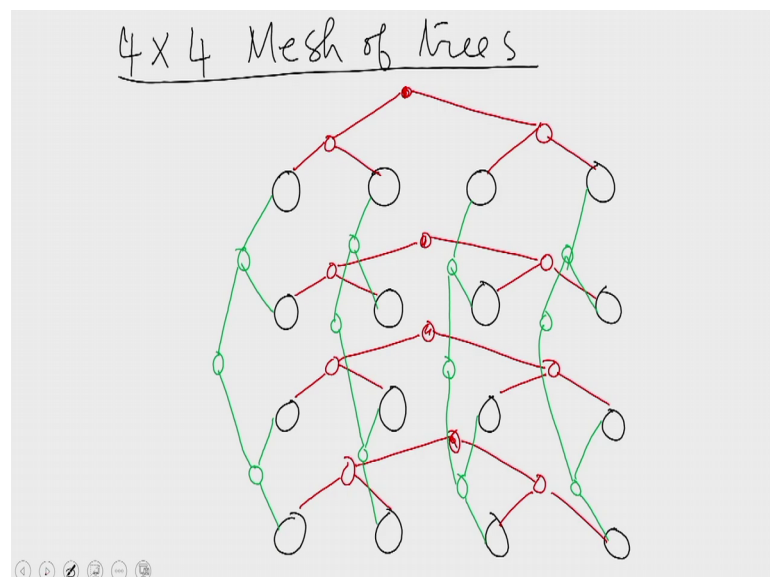
So, let us begin with the 2 dimensional mesh of trees. In a 2 dimensional mesh of trees we have an N by N mesh of processors. These processors are called the leaf processors.

(Refer Slide Time: 01:03)



They are arranged exactly as in a mesh in 2 dimensions. The difference is that the connection among them is different in this case. Instead of having linear connections along the rows and columns here we have binary trees on each row and each column. So, we can assume that N equal to 2 power K a power of 2 because we are going to construct a binary tree over N elements.

(Refer Slide Time: 02:06)



So, to take an example; let us consider a 4 by 4 mesh. In a 4 by 4 mesh we have one 16 processes arranged in 4 rows. And then on top of these we construct binary trees. So, for

every row we construct a binary tree. We will denote the row tree using red lines and red vertices. So, we take extra vertices to construct these binary trees. So, that is the first row tree. Similarly, we have the second row tree constructed like this. It is the third row tree.

So, every row has a tree constructed in this fashion and then every column also has tree constructed. We will draw the column trees using green links and green notes. Such a connection is called a 2 dimensional mesh of trees. In particular, this is a 4 by 4 mesh. In general, this can be extended to any N by N mesh where N is a power of 2.

(Refer Slide Time: 04:30)

Handwritten notes on a slide:

- Leaf processors } N^2
- N^2 : degree of 2 }
- Root processors } $2N(N-1)$
- $2N$: degree of 2 }
- other processors
- $3N^2 - 2N - (N^2 + 2N)$ degree 3

So, in this case we find that the leaf processors there are N squared of them. They all have a degree of 2. Every leaf processor belongs to a column as well as a row. Therefore, it belongs to a column tree as well as a row tree. It was a leaf in both of these trees. It has a pair and a unique pair and in both of these trees. So, all these leaf processors have exactly 2 parents, one in the row tree in which it belongs in one in the column tree in which it belongs. Therefore, all of them have a degree of exactly to the root processors. How many of them are there? There are 2 N root processors. For every row there is a tree which has a root and for every column there is a tree which has a root. Therefore, we have one root processor for every column and every row.

So, there are 2 N root processors. For example, this is a root processor this is a root processor. And this is a root processor and so on. So, there are 4 row roots similarly there are 4 column roots the green roots. So, on the whole there are 8 root processors in this

figure. So, in general for an N by N mesh we have $2N$ root processors. All of them have a degree of 2 each. Every other processor are of degree 3.

Now, what is the total number of processors in this arrangement. We have a N squared leaf processors. For every row we have a tree a binary tree. This binary tree is constructed on top of N leaves. Therefore, the binary tree constructed on a row will have N minus 1 nodes. For example, here each red tree has N minus 1 nodes N is 4 here. So, there are 3 nodes in each of the trees apart from the leaves.

So, there are N minus 1 nodes in every single row tree. Similarly, there are N minus 1 nodes in every single column tree. Therefore, the total number of other nodes is twice N into N minus 1.

(Refer Slide Time: 07:29)

$$\begin{aligned} \# \text{ nodes in an } N \times N \text{ MoT} &= N^2 + 2N(N-1) \\ &= 3N^2 - 2N \\ \hline \text{Sum of degrees} &= 2(N^2 + 2N) + 3(2N^2 - 4N) \\ &= 8N^2 - 8N \\ \# \text{ edges} &= 4N^2 - 4N \end{aligned}$$

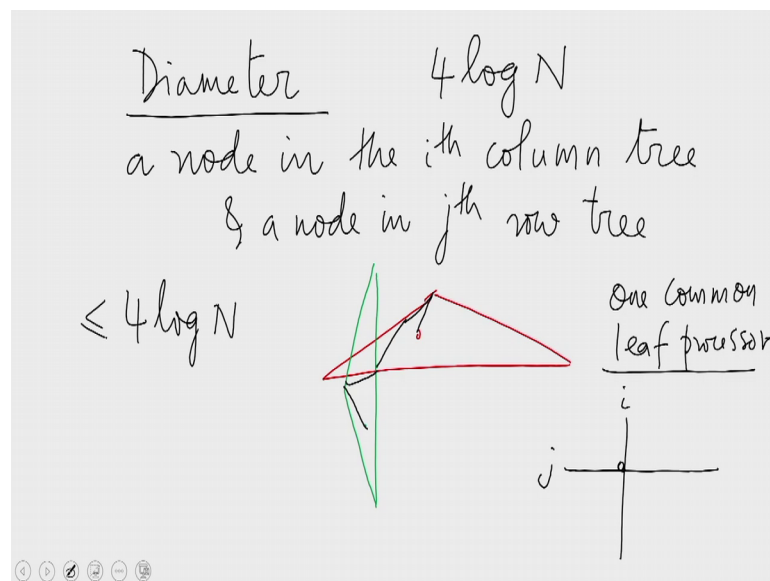
Which means the total number of nodes here is the number of nodes in an N by N mesh of trees is N squared leaf processors plus $2N$ into N minus 1, which is $3N$ squared minus $2N$.

The number of nodes in an N by N mesh of trees was $3N$ squared minus $2N$. And what is the number of edges? To calculate the number of edges in a graph we can add up the total degree of all the vertices. This will be twice the number of edges because when you take the sum of degrees in a graph, we are counting every edge twice. One at this end

and once at the other end. Therefore, the sum of degrees of the vertices of a graph will be exactly twice the number of edges in the graph.

So, if you calculate the total number of degrees, we will get the twice the number of edges. So, there are N^2 leaf processors and there are $2N$ root processors. So, that is $N^2 + 2N$. Then the others would be $3N^2 - 2N$ minus these $N^2 + 2N$. So, there are $2N^2 - 4N$ nodes of degree 3, $N^2 + 2N$ nodes of degree 2. Therefore, the sum of degrees is twice $N^2 + 2N$ plus thrice it would be $2N^2 - 4N$. Which is $8N^2 - 8N$ therefore, the number of edges in the graph is $4N^2 - 4N$ half the sum of degrees. So, in an $N \times N$ mesh of trees there are $3N^2 - 2N$ vertices and $4N^2 - 4N$ edges.

(Refer Slide Time: 09:55)



Now, let us see some nice properties of a 2 dimensional mesh of trees of $N \times N$ leaf nodes. Let us first estimate the diameter. The diameter of this is $4 \log N$. The diameter of an $N \times N$ dimensional mesh of trees is $4 \log N$ that is because the maximum distance between any 2 nodes can be shown to be at most $4 \log N$.

So, let us see how this can be shown. First consider a node in the i^{th} column tree and a node in the j^{th} row tree. Let us say we want to go from a node in the i^{th} column tree to a node in the j^{th} row tree. For example, we have a column tree we started some node in the column tree and we want to go to some node in row tree. We want to start from the i^{th}

column tree which is shown in green here and the j th row tree which is shown in red here. And let us say we want to start from some vertex in the green tree and go to the mark vertex in the red tree or visa versa.

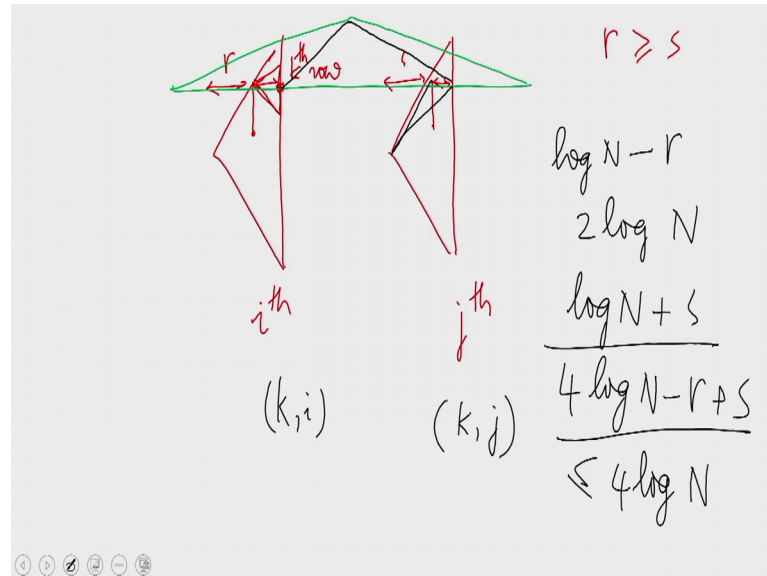
So, what could be the maximum distance here. So, we are considering the i th column tree and the j th row tree. These 2 trees are going to have one common leaf. They have one common leaf processor. In the mesh of the leaf processors if you take the j th row and the i th column, we get the leaf processor which is common to these 2 trees. So, that is one vertex which is common to these 2 trees. Therefore, if you want to start from a node in this tree and want to go to that node in that tree then you could use this common leaf node which happens to be here.

So, to go from the chosen source to the chosen destination you could first travel to the common node. But then how do you go to the common node? You could go all the way to the root, and then come down to the common node. Again go all the way to the root if necessary and then come down to the chosen node. You may not have to go all the way to the root in the second case, you may only have to go to the least common ancestor of the j th leaf processor and the chosen node. Similarly, for the first half 2, but even if you have to do this that is if the least common ancestor of the chosen node and the leaf processor happens to be the root. Even then you have to travel at most to login distance in the first tree and to login distance in the second tree.

So, we find that the maximum distance between 2 such nodes for $\log N$ at the most. If you take a node in the i th column tree and a node in the j th row tree we find the maximum distance between these 2 nodes at most for $\log N$. That is because the distance we are breaking down into 2. First we move to the common vertex of these 2 trees. To move to the common vertex, you have to rise from your source all the way to the root of the column tree and then come down to the common vertex in the worst case. Or strictly speaking you have to go from start from the source go to the least common ancestor of the source and the common vertex i, j, i and then come down to vertex j, i this will take at most to $\log N$ steps. Then similarly in the other tree you have to rise from the common least common ancestor of the j th leaf processor and the chosen destination and then come down to the chosen destination which will again take a most to \log in steps. In the worst case the root is the least common ancestor.

So, in at most $4 \log$ in steps you can go from any node in the i th column tree to any node in the j th column j th row tree.

(Refer Slide Time: 14:30)



Instead let us say we want to move from one column tree to another. We have some vertex in this column tree and we want to go to some vertex in this column tree let us say. What we do is this consider the levels of these 2 nodes. The height of the tree is $\log N$. Let us say the level of the source is r and the level of the destination is s . So, without loss of generality let me assume that r is greater than or equal to s . Otherwise we will swap the source and the destination. That is if we managed to find the path from there to here we can use the same path to go from here to there as well.

So, in discovering this path, we will assume that r is greater than or equal to s . Now what we want is this here we have the i th column tree and here we have the j th column tree. We want to start from a node that a certain level r in the right column tree and go to a node which is at level s in the j th column tree. We assume that the root is at level 0. Now what we do is this, we start from the source and go to some leaf in the subtree which is rooted at the source.

So, in this binary tree let me consider some subtree which is rooted at this node. I go to some leaf of that the distance I have to travel is \log in minus r . This node is at level r the maximum height of the tree is $\log N$. Therefore, I have to travel a distance of \log in minus r to come to the leaf. Let us say this leaf happens to be in the k th row. So, what I

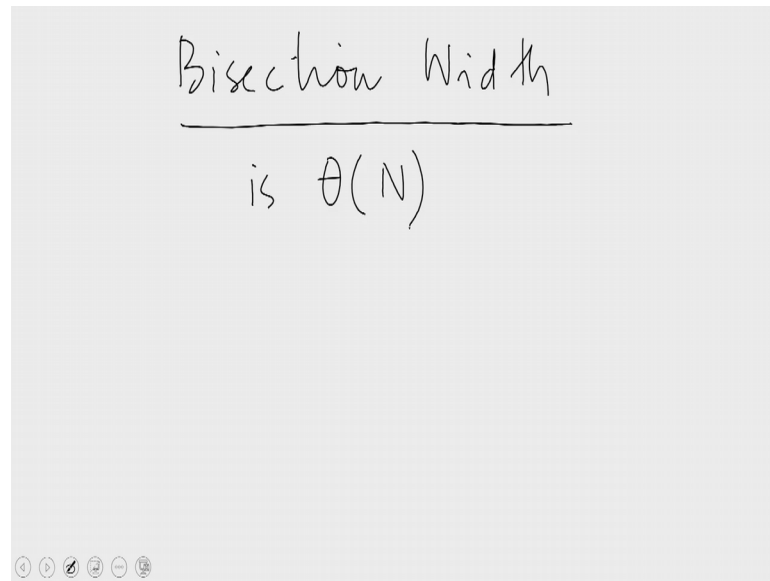
now do is this I start from the source and go to one descendant of the source in the column tree to which it belongs, suppose this descendant happens to be this descendant leaf happens to be in the k th row.

So, after reaching the descendant node, since I know that it is in the k th row I also know that it belongs to the k th row tree. It is a leaf of the k th row tree. Now I use the k th row tree in the k th tree I will travel from this leaf node to the leaf node of the k th row tree which belongs to the j th column. That is now I am in leaf node $i k$ or other $k i$, I am in leaf node $k i$ after descending from my source and then I will travel to leaf node $k j$. Both of which are leaf nodes of the same row tree. This I can do by going all the way up to the root and then descending to the required leaf node. That is, I will go for all the way from here to the root and then descend to the required leaf node. I could do this in $2 \log N$ steps at the most. Once I have done this now to reach the destination node all I have to do is to climb up the tree and then climb down again.

So, I may have to go all the way to the root and then come down to my destination, but then the distance from the root to the destination is at most s . Therefore, the distance I have traveled totally is from here from the source to the leaf node I have traveled a distance of $\log N$ minus r . Then from the leaf node to leaf node $K i$ to go to leaf node $K j$, I require to \log in steps and then to go from leaf node page a to destination I will require an additional \log in plus f steps. For a total of $4 \log N$ minus r plus s , but since R_s greater than or equal to s this is less than or equal to $4 \log N$.

Which establishes that the maximum distance between any 2 nodes in this mesh is $4 \log N$. Or in other words the diameter of a an N by N 2 dimensional mesh of trees is $4 \log N$ which is a rather low diameter.

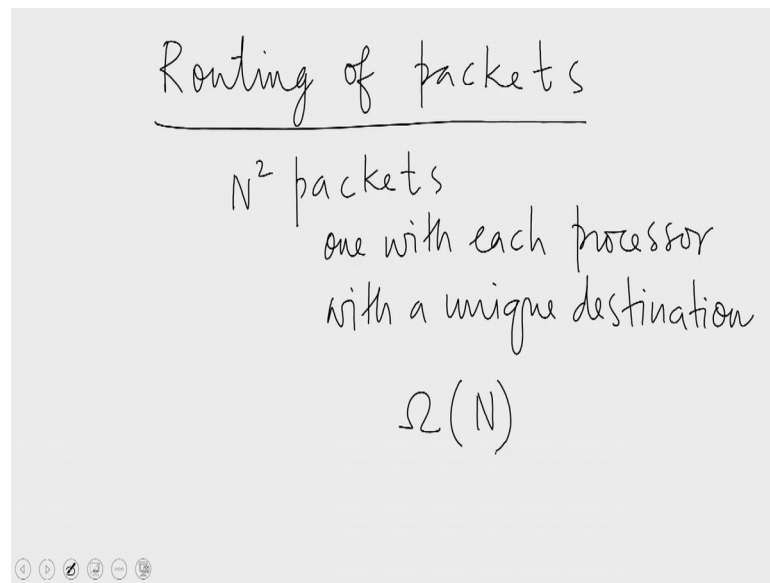
(Refer Slide Time: 19:09)



Now, coming to the bisection width; the bisection width of an N by N mesh of trees is $\theta(N)$. And stayed this without proof we do not have time for the proof here. Therefore, you take it from me without proof, you can find the proof in the references.

So, the bisection width of an N by N mesh of trees is $\theta(N)$. So, this network has a high bisection width and a low diameter. What it means is that for the problem of sorting once again the lower bound is $\omega(N)$ not much better than an N by N mesh, but since the diameter is low certain problems can be solved faster on this model. For example, let us consider the problem of routing; routing of packets..

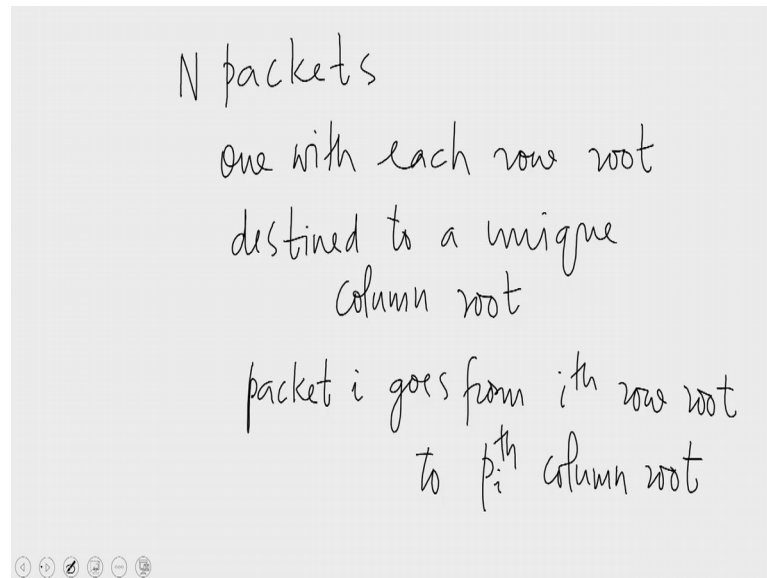
(Refer Slide Time: 20:05)



Of course, if we have a N^2 packets, one with each processor, each with a unique destination. If these N^2 packets have to be routed simultaneously, then we have a total traffic of N^2 . If we ensure that for any bisection the packets which are destined to the other side are kept on the side and vice versa then all these packets will have to cross the bisection.

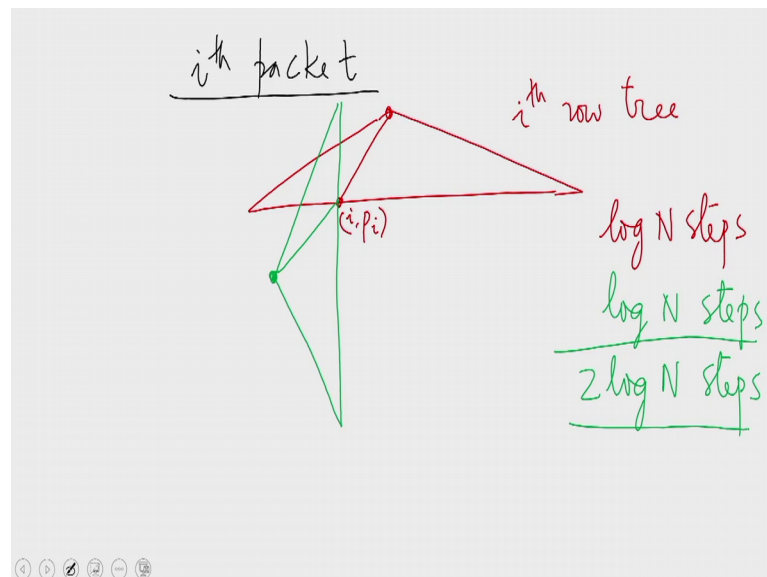
Now, we know that the bisection with this traffic of N^2 . Therefore, the packets will take at least $\Omega(N)$ time to cross the bisection. Therefore, this routing problem will have a lower bound of $\Omega(N)$. Which is not much better than the 2 dimensional N by N mesh, but then there is a routing problem which can be solved faster on the mesh of trees. For example, let us say we have N packets.

(Refer Slide Time: 21:31)



One with each row root and let us say the packet is destined to a unique column root. In other words, packet i goes from the i^{th} row root to the p_i^{th} column root. The i^{th} packet goes from the i^{th} root i^{th} row root to the p^{th} column root. And let us say this is the routing problem that we want to solve.

(Refer Slide Time: 22:37)



So, let us look at the i^{th} packet. Let us root the i^{th} packet from the root of the i^{th} row tree. So, packet i is held by the root of the i^{th} row tree. Let us say we send this down to the p^{th} leaf processor of this row tree. So, the coordinates of this processor would be $i p i$

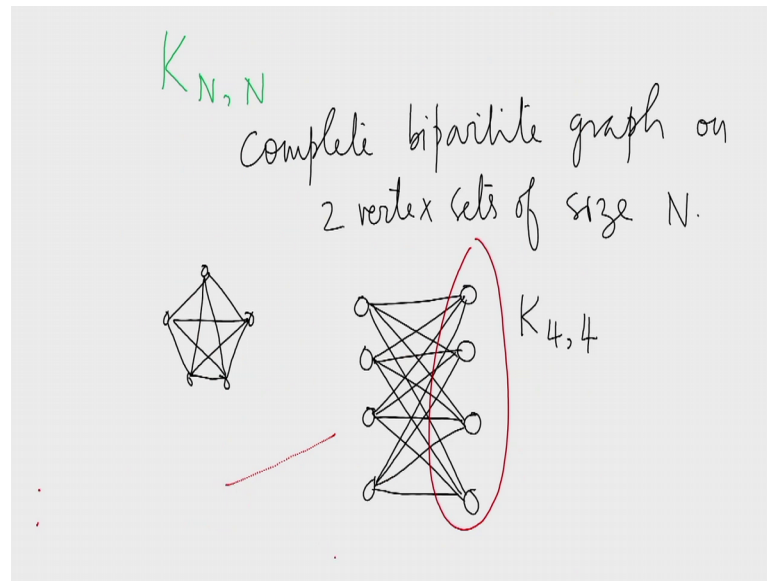
it is in the i th row and in the p th column. So, let us send the packet down to the node. This would take $\log N$ steps. This transportation is entirely within the i th row. There is only one packet within the i th tree therefore, this packet can be delivered without any conflicts to the $i p i$ labeled leaf processor. Now this leaf processor is also a leaf processor of the p th column tree the root of the p th column tree is our destination.

Now, the packet is already delivered to the correct column tree. All we require now is to root this packet from the leaf at which it is delivered to the root of the tree, which requires sending the packet of the tree. Now this transportation is entirely within one single column tree. What we have said is that all destinations are unique. So, there is only one packet reaching this column. Therefore, this packet can reach its destination in $\log N$ steps again. So, you will require an additional $\log N$ steps to deliver the packet the total time taken as to $\log N$ steps.

So, this routing problem can be solved in order $\log N$ time, but this routing problem cannot be solved in order $\log N$ time on an N by N mesh. If every row had a packet which had to be delivered to an appropriate column, then depending on the source and the destination, the maximum distance that a packet has to travel could be $\theta(N)$ in an N by N mesh. Therefore, this problem could take $\theta(N)$ time on a 2 dimensional mesh of size N by N . On a mesh of trees of size N by N this takes only order $\log N$ time.

So, there are certain problems for which mesh of trees are more powerful than 2 dimensional meshes. 2 dimensional mesh of trees can be used to simulate KNN the complete bipartite graph KNN.

(Refer Slide Time: 25:31)



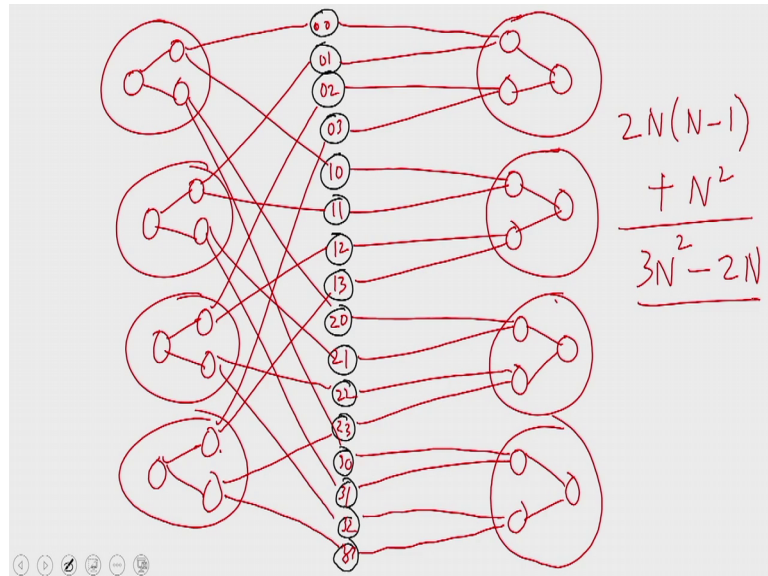
$K_{N,N}$ is the complete bipartite graph on 2 vertex sets of size N . $K_{N,N}$ can be used to denote an interconnection between all processes. When you have N processors, for example, when N equal to 5 we may want to interconnect all the 5 processors that is if every processor has to be connected to every processor then we may want a configuration of the sort. This is the richest possible interconnection network. Once we have a connection of the sort then every processor can communicate with every other processor in every single step.

So, this is the richest possible connection that you can imagine. Essentially the same connection can also be achieved using a bipartite graph. If you have 4 processors and 4 memory units, we could connect every processor to every memory unit. In this fashion so, this is as good as connecting every processor to every processor. If a processor wants to send a message to another of its processes all it has to do is to write in its own memory bank at a designated place, then every processor can read every memory bank. Therefore, the value returned could be read off by the other processors.

So, effectively we are creating the complete graph situation. That is if in a complete graph we have every processor connected to every processor. Here we are achieving the same result using a $K_{N,N}$ architecture, but then $K_{N,N}$ has a close relation with a 2-dimensional mesh of trees. In particular, let us consider this $K_{4,4}$. In $K_{4,4}$ we have 2 vertex sets of size for each every edge in the graph is between the 2 vertex sets. There is

no edge between 2 vertices belonging to the same vertex set. So, this is a complete bipartite graph on 4 vertices on either side.

(Refer Slide Time: 28:06)



Let us modify this graph in this fashion. We take 16 vertices of the sort and label them in this fashion. This vertex 00 this is 01 02 03 10 11 12 13 20 21 22 23 30 31 32 33. And then let us say we take 4 vertices that we draw these vertices as big circles. So, we have now numbered the nodes as if they are nodes of the leaf processors of an N by N mesh of trees, a 4 by 4 mesh of trees. In a 4 by 4 mesh of trees we have 4 rows and 4 columns and the processors are numbered from 00 to 33.

So, in a similar manner we have now numbered them. And then we interconnect nodes in those fashions. Here we take 2 nodes within this big node and interconnect nodes in this manner. So, on the right side we have connections made like this. So, here the big circles correspond to the nodes in the right vertex set of the K 4 4. So, we are transforming K 4 4 in this manner. Every vertex on the right side here is replaced with a super vertex. Within the super vertex we introduce a binary tree.

So, there are 4 edges coming into the super vertex. On each of these edges we place a vertex, one of the leaf processors. And these vertices are then interconnect are connected to the leaves of the binary tree that we have formed. So, each super vertex transforms into a 4 leaf binary tree. And the leaves of these are the new leaf processors that we have

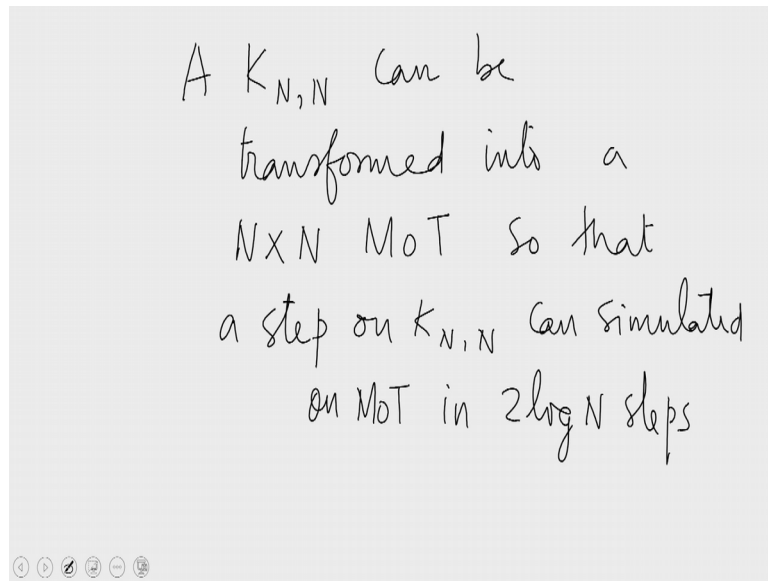
introduced. On the other side similarly we take 2 vertex per super vertex and then connect them in this fashion.

So, of the 2 nodes in the first super vertex the first node is connected to 0 0 and 1 0 and the second node is connected to 2 0 and 3 0. And here in the second super vertex the first node is connected to 0 1 & 1 1. And the second node is connected to 2 1 & 3 1 and so on and then we add an extra vertex in the super vertex and make it the root of the binary trees. Then what we find is that this is exactly the layout that a mesh of trees provide us. The 2 super vertices on either side correspond to the row trees and the column trees within each super vertex, we have constructed a binary tree. And these correspond to exactly the row trees and column trees of our N by N mesh of trees.

So, strictly speaking the transformation is this you take an N by N mesh of trees and then on you take a KNN to convert this into and N by N mesh of trees what we do is this on every edge of this N by N click we introduce a processor. These processors will form the leaf processors of our mesh of trees. And then each vertex of KNN is converted into a binary tree. A binary tree with N leaves these N leaves have to find side with the leaf processors. So, in each of these we have N minus 1 nodes.

So, you can see that the total number of nodes is $N \times (N - 1) \times 2$. When you account for all the super vertices and then there are the original N^2 processors, which is $3N^2 - 2N$. And then when you check the interconnections you find that they are exactly what they are in the mesh of trees. So, this graph that we have laid out is nothing but a different drawing of an N by N mesh of trees.

(Refer Slide Time: 34:16)

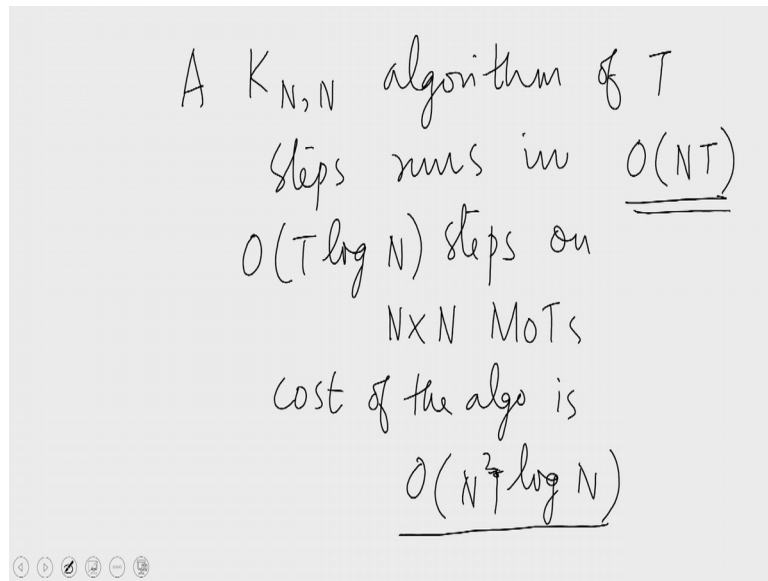


A $K_{N,N}$ can be transformed into a $N \times N$ MoT so that a step on $K_{N,N}$ can be simulated on MoT in $2 \log N$ steps

Therefore, what we have established is that a KNN can be transformed into an N by N mesh of trees. So, that a step on KNN can be simulated on the mesh of trees in $2 \log N$ steps. Why $2 \log N$? That is because in KNN when 2 neighbors exchanged a message in one single step, to pass the same message in this case where it is as if you had a message going from here to there. To pass the same message now we have to assume that the message originates at the root of this binary tree and it is designed to the root of this binary tree. To transport this message, you have to climb down the binary tree, come to the leaf node and then climb up the other binary tree to reach the corresponding root.

So, this amounts to a distance of $2 \log N$. Therefore, every single step on KNN can be simulated in $2 \log N$ steps on a mesh of trees.

(Refer Slide Time: 35:53)

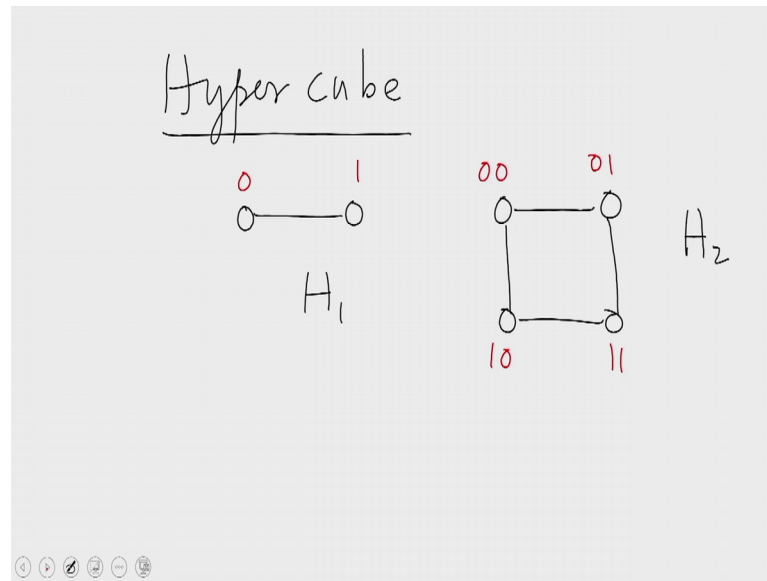


A $K_{N,N}$ algorithm of T
steps runs in $O(NT)$
 $O(T \log N)$ steps on
 $N \times N$ MoTs
Cost of the algo is
 $O(N^2 \log N)$

Or in other words an algorithm which runs in t steps runs in order of $T \log N$ steps on an N by N mesh of trees. But an N by N mesh of trees of course, has to use order of N squared processes to achieve this. In addition to that the cost of the algorithm therefore, is order of N squared $\log N$. In a KNN we have N plus N^2 equal to $2N$ processors executing the algorithm in t steps. Therefore, the cost of the original algorithm was order of N into t , but the simulation takes order of N squared $t \log N$ steps N squared $t \log N$ cost.

So, the simulation is rather expensive in terms of cost, but still what it establishes is that the mesh of trees is a versatile architecture. We shall see architectures which are more versatile than mesh of trees as we go by, but then this is certainly a more versatile architecture than a 2 dimensional mesh.

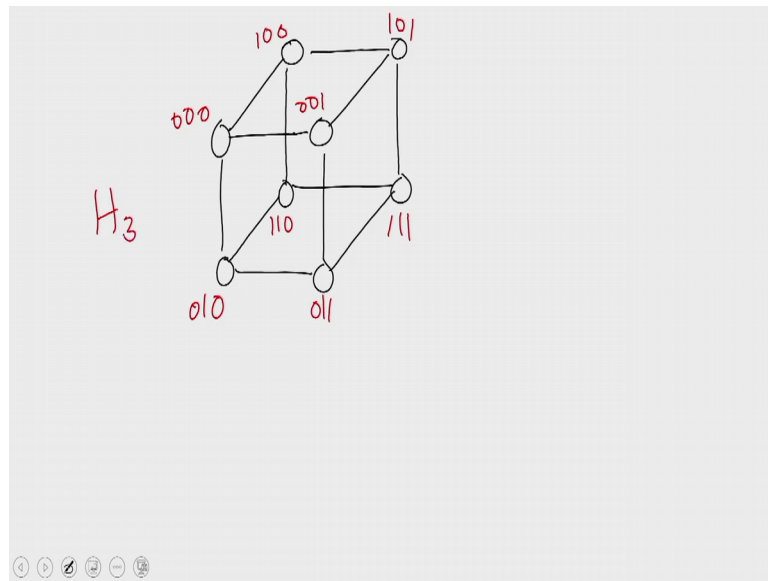
(Refer Slide Time: 37:25)



The next architecture we shall see is a hypercube. We talked about a hypercube in an earlier class. A hypercube of one dimension is made up of just two nodes. And here the nodes are labeled in this fashion. One node is labeled 0 and the other node is labeled 1. To form a hypercube of 2 dimensions we take 2 identical copies of one dimensional hyper cubes and number them exactly as we did before.

Now, we connect the corresponding nodes that the 0's in the two copies will be connected together and the 1's in the two copies will be connected together. This is a 2 dimensional hypercube. Further when we go on to the third dimensional hypercube which is a H_3 .

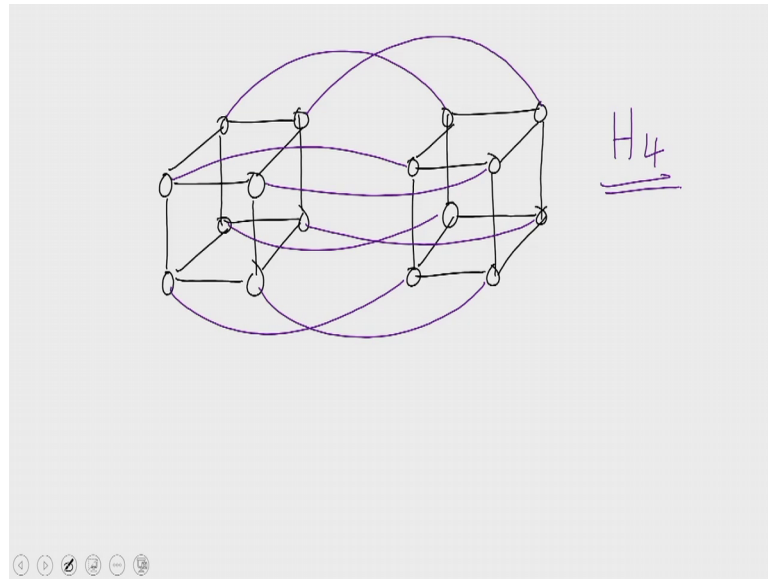
(Refer Slide Time: 38:37)



We take 2 copies a 2 dimensional hyper cube with the appropriate numbering. The numbering is enhanced in this manner. We add a 0 on the left side for the first copy and one on the left side for the second copy. So, that all 4 4 vertices will have unique numbering.

So, we have taken two such copies. And then we connect the corresponding nodes. And now the numbering has to be enhanced we add one more bit. In the first copy we add a 0 or to the left side in the second copy we add a one to the left side. That ensures that all the vertices are distinguished.

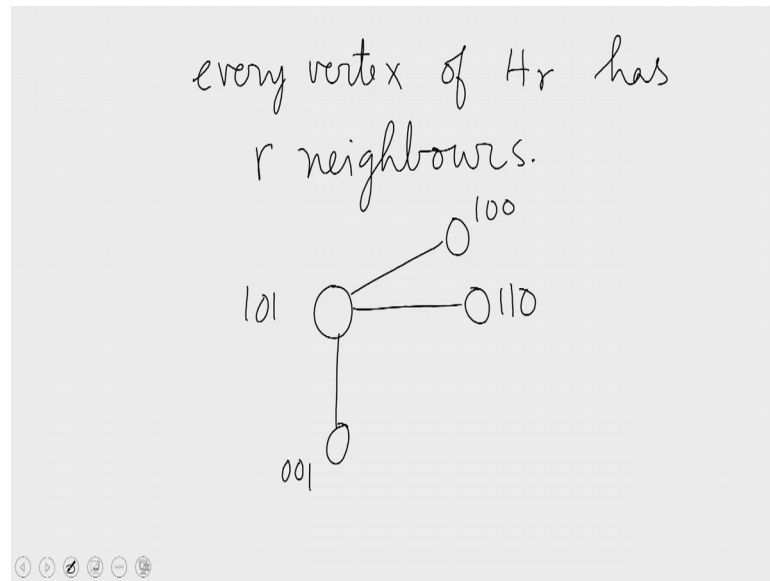
(Refer Slide Time: 40:00)



Now when we go on to the 4th dimensional hypercube we take 2 copies of 3 dimensional hyper cubes, and connect the corresponding vertices. For example, these 2 vertices correspond we connect them these 2 vertices current correspond we connect them. And these two correspond these two correspond we connect up the vertices in this 4 manner to form the 4 dimensional hypercube.

Like this we can go on to construct higher dimensional hyper cubes. In general, to construct H_r from H_{r-1} , we take 2 copies of H_{r-1} and connect the corresponding nodes. Now the nodes are numbered in this manner once the nodes of the 2 copies of H_{r-1} are recursively numbered, we add one more bit to the left. We add a most significant bit to the numbering all the vertices of one copy will get a most significant bit of 0. Whereas, all the vertices of the other copy will get a most significant bit of 1, this will ensure that the newly constructed H_r will have a unique number for every single vertex. So, this is how we number the vertices of a hypercube. Now hypercube has some interesting properties.

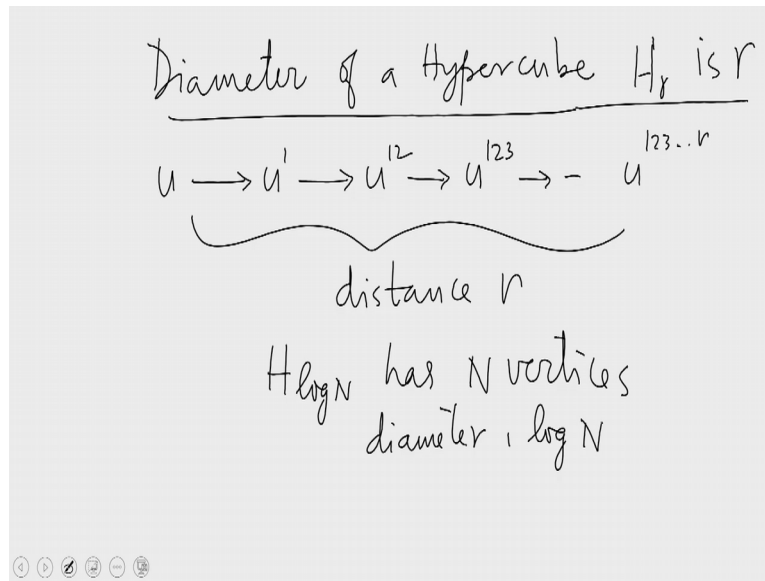
(Refer Slide Time: 41:38)



One is that every vertex of an r dimensional hypercube has r neighbors. For example, in a 3 dimensional hypercube a node 1 0 1 has got three neighbors along dimension one we have 1 0 0 where the least significant bit is flipped. Along dimension 2 we have 1 1 0 where the second bit is flipped and along the third dimension we have 0 0 1 where the third bit is flipped. So, given a vertex we can find the neighbors of the vertex from the binary representation of the label of the vertex. All you have to do is to flip the bits one by one. When you flip one bit you get one neighbor when you flip another bit you get another neighbor and so on.

So, there are r dimensions in the hypercube. Then there are r bits in the label of each node. There are r bits to flip when you flip each of them we get one of the r neighbors.

(Refer Slide Time: 42:58)

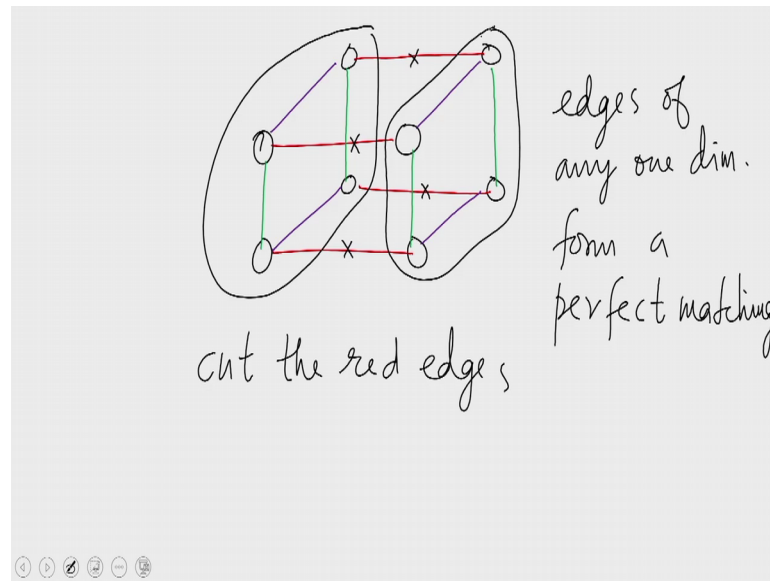


This enables us to determine the diameter of a hypercube. Let us say we start with some vertex, u if we keep on flipping the bits, we can come to the diametrically opposite vertex of the source. So, let us say we start with vertex u and travel along dimension 1 by flipping the first bit let u^1 denote the binary representation that you obtain by flipping only the last bit.

So, when the last bit of u is complemented, we get u^1 . When the second last bit of u is complemented we get u^{12} ; u^{12} is a binary string that is obtained from you by complementing the 2 least significant bits. When you complement the third bit you get u^{123} and so on. Finally, when all the bits are complemented you reach the diametrically opposite point. So, the maximum distance you have traveled when you go from u to u^1 to u^{12} to u^{123} up to $u^{123\dots r}$ the exact complement of it is r .

Let us you have to travel one edge along each dimension to complete this travel. What does establish is that the diameter of a hypercube H_r is r . In particular, when we consider a hypercube of N vertices, its diameter is $\log N$. $H_{\log N}$ has N vertices its diameter is $\log N$ what is the bisection width.

(Refer Slide Time: 45:08)

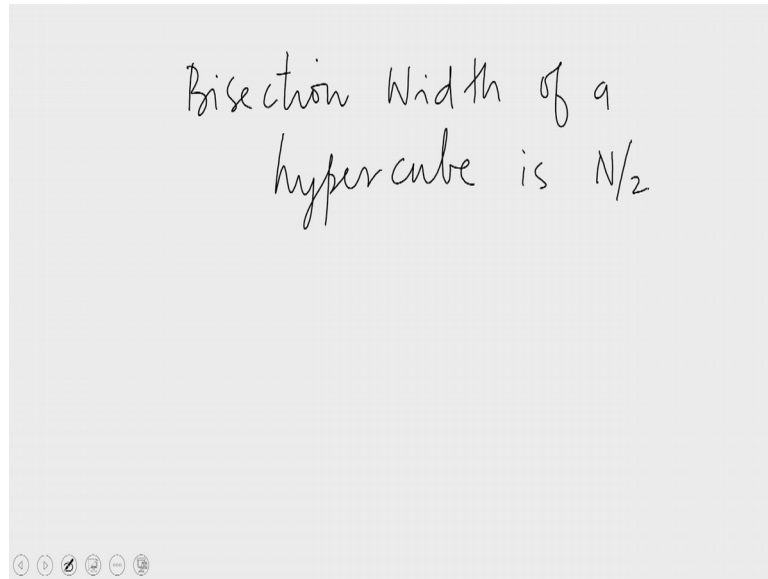


Let us consider a 3 dimensional hypercube. Let us say the red lines represent dimension 1. The green lines represent the vertical dimension. And let us say the blue lines represent into the both dimension.

So, I have drawn the edges of the hypercube using 3 different colors for each dimension I have used a different color. So, when you find that when you look at this coloring you find that edges of any one dimension form a perfect matching. In particular, if you consider all edges of color red you find that they form a vertex matching, the horizontal edges in this case they form a perfect matching. If you cut all these in particular, in this case let us say we cut the red edges. If these edges are disconnected, then the hypercube divides into 2 equal halves.

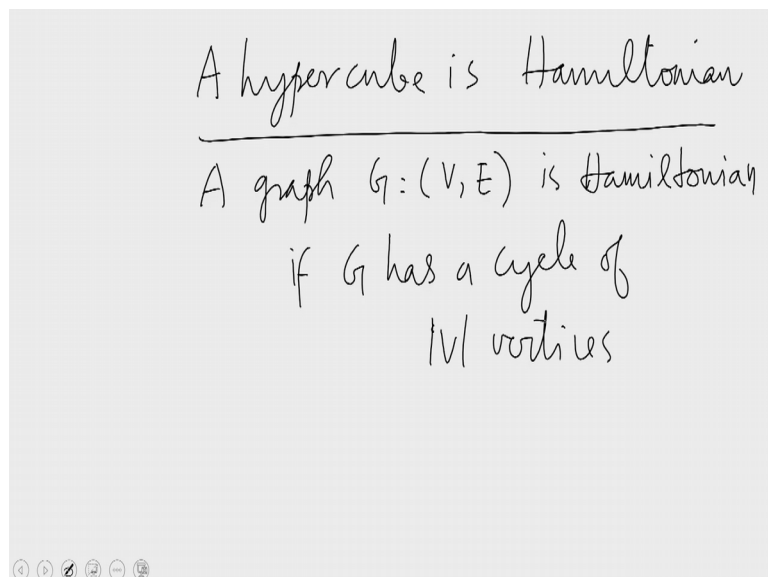
This is true for any single dimension. Instead of the red edges if you cut all the blue edges then we will again get 2 hyper cubes one in the front and one in the back we will get to 2 dimensional hyper cubes or if you cut the green edges. We will get one upper hyper cube and the lower hypercube and this argument can be extended to any dimension.

(Refer Slide Time: 47:03)



So, what this establishes is that the bisection width of a hypercube is $N/2$. So, hypercube indeed has high bisection width and a low diameter. A hypercube has several other nice graph theoretic properties.

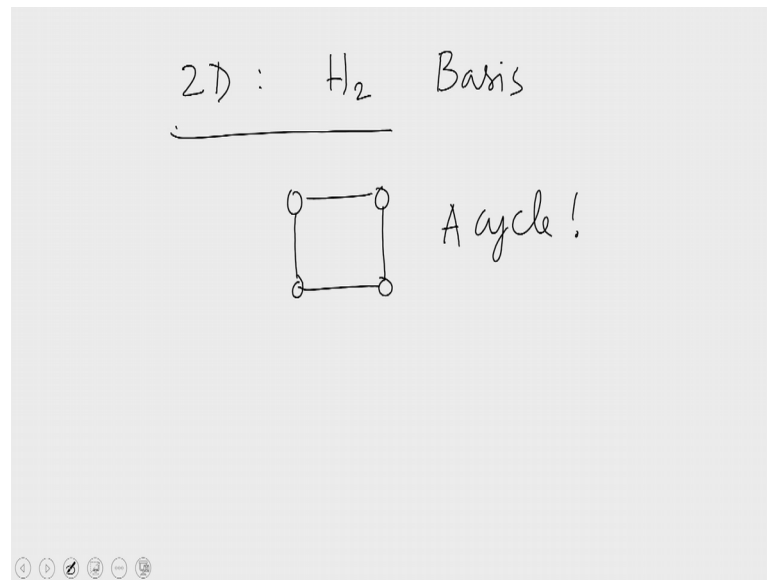
(Refer Slide Time: 45:32)



One is that it is Hamiltonian. What is the Hamiltonian graph? Graph G equal to $V E$ is Hamiltonian if G has a cycle of $|V|$ vertices. In other words, it is possible to permute the vertices so that this permutation forms a cycle within the graph. Or in other words it is possible to start at a vertex and travel through every vertex exactly once and get back

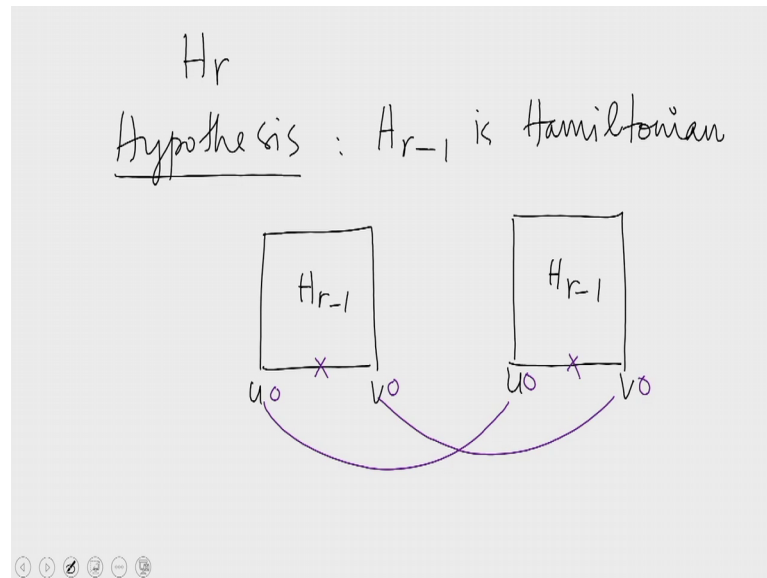
to the starting vertex. If this property is satisfied by the graph, then we say that the graph is Hamiltonian. So, to prove that a hypercube is Hamiltonian what we need is to start at the vertex travel through every single vertex and come back to the original vertex. Every vertex should be traversed exactly once.

(Refer Slide Time: 48:46)



So, let us start with a 2 dimensional hypercube. We consider H_2 this forms the basis we will prove this using induction. So, the basis of the induction will be provided by the 2 dimensional hypercube. A 2 dimensional hypercube is a cycle indeed. Therefore, the property is satisfied here trivially. You can start at the vertex travel to every vertex exactly once and come back to the original vertex; the vertices of H_2 to form a cycle.

(Refer Slide Time: 49:25)

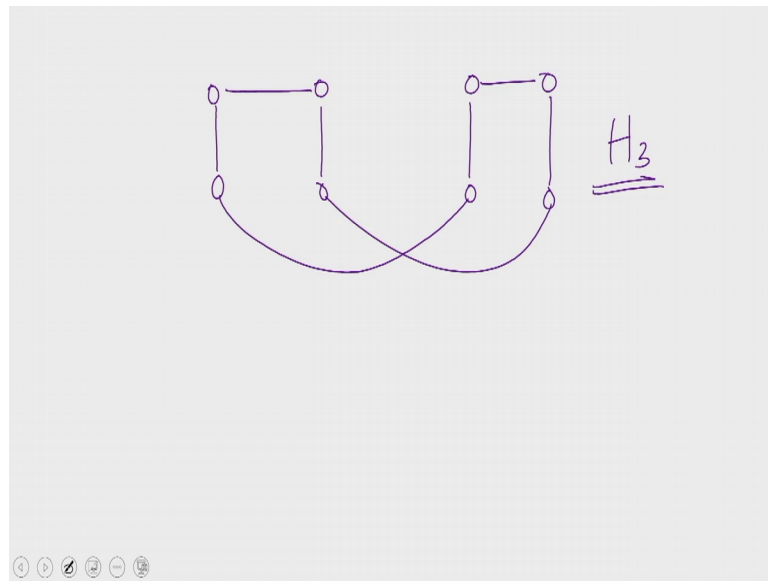


Now when you come to a higher dimensional hypercube, let us say we are considering H_r . So, let me hypothesize that H_{r-1} is Hamiltonian. Then we will take two identical copies of H_{r-1} . Find identical Hamiltonian circuits in each. And then consider 2 identical edges, let us say we have an edge from u to v in this H_{r-1} and there is an edge u to v in this H_{r-1} as well. We find identical Hamiltonian cycles in both the copies of H_{r-1} . When we construct H_r from H_{r-1} what we do is to take 2 copies and connect the corresponding nodes.

So, we have connections like this. The u here is connected to the u here and the v here is connected to the v here. These are the arc dimensional connections and because of these connections we also add an extra bit to distinguish these vertices. In other words, what we have done is we have taken in H_r and we have removed all dimension or edges. Then we get 2 identical copies of H_{r-1} and we find the Hamiltonian cycle in the 2 of them. And then we now consider the dimension r connections.

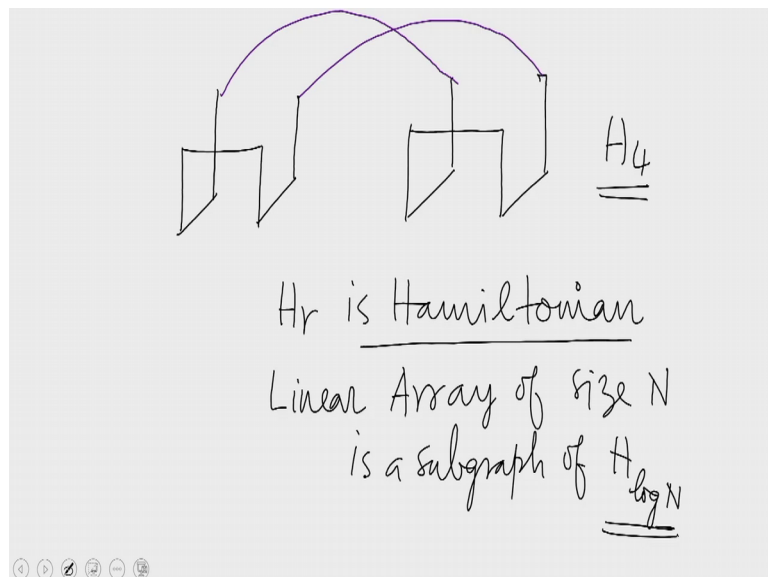
Using these dimension r connections I can stitch the 2 Hamiltonian cycles together into one single Hamiltonian cycle; for example, if I disconnect this edge in this inch from the 2 respective Hamiltonian cycles and add the blue edges instead then the 2 cycles have become stitched together into one single Hamiltonian cycle. So, let us see this working in the case of a 3 dimensional and 2 dimensional cases.

(Refer Slide Time: 51:42)



In the 2 dimensional case as we have seen just now, we take 2 copies of H_2 and then connect in this manner, and delete these edges. So, this forms a Hamiltonian cycle of H_3 . So, this forms a Hamiltonian cycle of H_3 . From a Hamiltonian cycle of H_2 now we have invented a Hamiltonian cycle for H_3 .

(Refer Slide Time: 52:18)



Now when we go on to H_4 we take 2 copies of this. So, this is a Hamiltonian cycle of H_3 we take an identical copy. And then choose any particular corresponding pair of nodes and interconnect them. For example, if we choose this edge we could interconnect them

in this manner. And then delete the original pair of edges. This forms a Hamiltonian cycle of the 4 dimensional hypercube. In this manner we can extend the Hamiltonian cycle of H_{r-1} into a Hamiltonian cycle of H_r . This establishes that by induction H_r is Hamiltonian. The immediate consequence of this is that, a linear array of size N is a sub graph of $H_{\log N}$.

A linear array is a chain of N nodes when you connect the last node to the first node we have a cycle of size N , this can be embedded on a hypercube of N nodes, which will have a dimensionality of $\log N$. Therefore, any algorithm that you have designed for a linear array of size N can run on a hypercube without any change. You consider the embedding you activate only the connections of the embedding and ignore all the other connections. Then a hypercube becomes a linear array with loop a feedback loop from the last node to the first node.

So, any linear array algorithm can be run on a hypercube without any change. We shall see more of the versatility of a hypercube in the next class. So, that is it from this lecture.

Thank you.