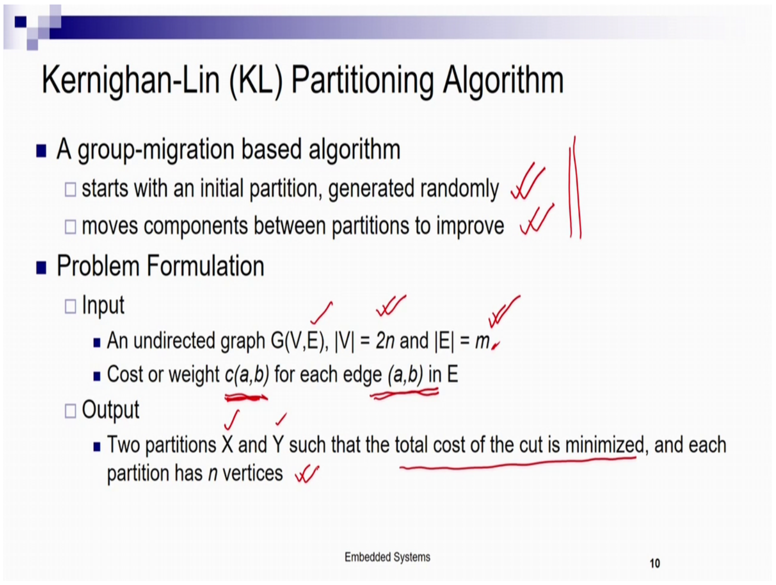
Embedded Systems – Design Verification and Test

Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 6
Hardware/Software Partitioning- 2

We now look at the first partitioning algorithm Kernighan and Lin.

(Refer Slide Time: 00:33)



Kernighan-Lin (KL) Partitioning Algorithm

- A group-migration based algorithm
  - starts with an initial partition, generated randomly
  - moves components between partitions to improve
- Problem Formulation
  - Input
    - An undirected graph G(V,E), |V| = 2*n* and |E| = *m*
    - Cost or weight *c(a,b)* for each edge *(a,b)* in E
  - Output
    - Two partitions X and Y such that the total cost of the cut is minimized, and each partition has *n* vertices

Embedded Systems 10

Now, let us say the objective for our partitioning process for this algorithm is very simple. We want to minimize the number of bits transmitted between hardware and software and not look at anything else. An example why you would we want this is, let us say, we have a extremely data dominated embedded system, where the cost minimizing the transmission between hardware and software is of prime importance because transmission of data between hardware and software is very costly. So, we want to minimize the amount of data transmitted between software and hardware that is essential big part of the cost. And the actual execution times of the nodes let us say is less important in for this for our example problem here.

So, the communication cost is much more. So, we assume that each of the nodes whether it is implemented in hardware or software, just consumes a unit cost, unit execution cost and the communication; however, is the edge have weights. So, I want to minimize the amount of data transfer or the total summation of the edge weights that goes from hardware to software or vice versa. Now this algorithm is a group migration based algorithm it starts with an initial partition generated randomly. So, we start with an initial partition. And we the here the one of the important principles of this algorithm is that the number of components in hardware and software will be same.

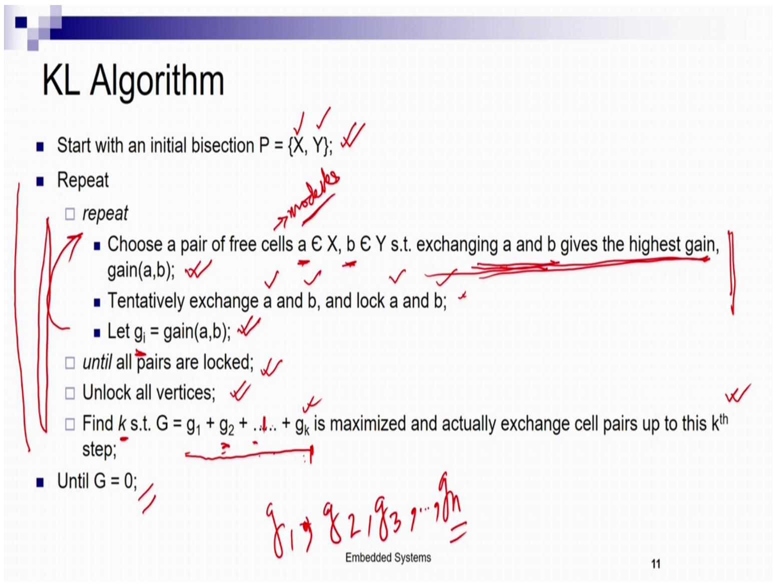
So, it moves components between partitions to improve. This is the characteristic of all group migration based algorithms. Now the input is provided in the form of a graph; where V is a set of nodes modules to any number and E is the set of edges, and number of edges equals to m. The cost or weight of an, you we have a cost or weight of an edge which indicates the total cost of data transfer.

For each edge between hardware and software and for each edge a b in b we have a cost of the data transfer c a b. And we want to minimize the amount of data transferred between in between hardware and software. So, the output what do we have? We have 2 partitions X and Y such that the total cost of the cut is minimized. By cut we mean the total data transfer between hardware and software, and such that each partition has n vertices. So, we say that this algorithm will always return in return equal sized partitions. So, it will only try to find out which nodes to put in hardware and which nodes to put in software multiple options it will test for it, but it will never have unequal size partitions.

So, the number of nodes in Maha software will always be equal to number of nodes in hardware, keeping this constraint I want to minimize the data transfer cost. Why do we want to have equal sizes in hardware and software? Could be that that gives me a balance of the cost because otherwise if you put everything in hardware, then the cost may shoot up because hardware implementation is much more costly.

So, we will always keep a balance, and hence we will keep equal number of vertices in software and hardware.

(Refer Slide Time: 04:32)



KL Algorithm

```
■ Start with an initial bisection P = {X, Y};
■ Repeat
    □ repeat
        ■ Choose a pair of free cells a ∈ X, b ∈ Y s.t. exchanging a and b gives the highest gain, gain(a,b);
        ■ Tentatively exchange a and b, and lock a and b;
        ■ Let $g_i$ = gain(a,b);
    □ until all pairs are locked;
    □ Unlock all vertices;
    □ Find k s.t. G = $g_1 + g_2 + \ldots + g_k$ is maximized and actually exchange cell pairs up to this kth step;
■ Until G = 0;
```

modules
$g_1, g_2, g_3, \ldots, g_n$

Embedded Systems 11

Now, as we told it starts with an initial bisection X comma Y. Bisection meanings it will randomly choose let us say by some heuristic or randomly, it will choose n of the nodes it has 2 n nodes. The total number of nodes is 2 n, it will take one set of n nodes put in X and the other set of n nodes put in Y, then it goes into a loop. It chooses free cells, cells or modules it is same.

So, you can say these are program modules, it chooses a pair of free cells or modules one from X a from X and b from Y; such that, exchanging a and b gives us the highest gain. Gain in terms of the partition cost gain in terms of reduction in partition cost. For us here gain in terms of reduction in data transfer. So, we choose a pair of free cells a from X and b from Y such that exchanging a b gives me the highest gain. And then we act actually tentatively exchange a and b tentatively. Exchange a and b it is not a final move, but we for the time being we exchange and then lock a b.

So firstly, we had n nodes in X and n nodes in Y, and then we took a from X and b from Y we exchange them and locked them from further consideration. Let the gain a b be given by g i. Now we go into this loop again now a and b is removed from consideration again we choose another node from X another node from Y such that, exchanging a b. Now gives me the highest gain, and then we tentatively exchange a and b and lock a and b. Likewise when we go on doing this all my nodes in a and b will finally, be it is exhausted after n moves.

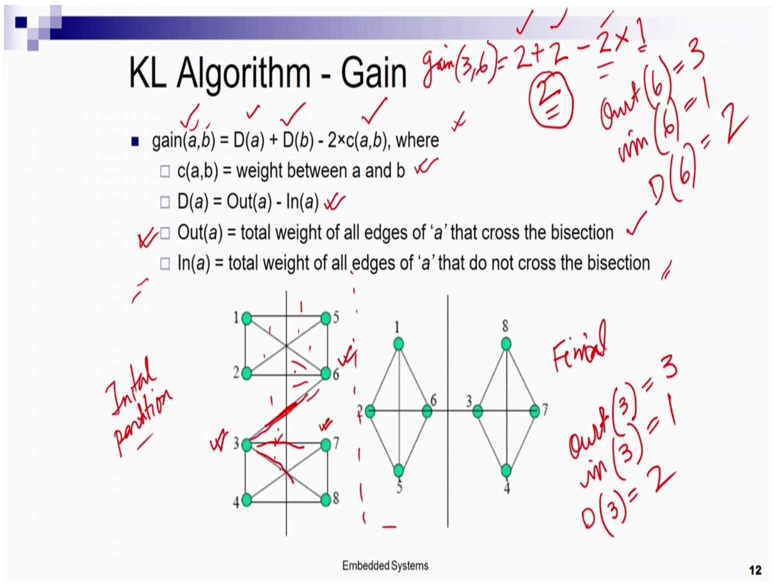
So, we will have n pairs and after n such loop iterations, all my nodes will be locked all my nodes in 2 n nodes will be locked. And then after all my nodes are locked we unlock first all vertices. And when we have unlocked we actually have a series of course, g 1 g 2 dot up to g k dot up to g n because we have made n exchanges, ok. So, we have g 1 plus g 1 comma g 2 comma g 3, likewise we have these n exchanges because for each exchange we have kept the estimate of the cost g i. So, we have this g 1 g 2 up to g n values.

From that we find a k such that g 1 plus g 2 plus g k is maximized, and actually exchange cell pairs up to this k th step. So, we said that these n pairs of exchanges were actually tentative. We do not actually exchange all of them, but how many pairs do be actually exchanged, we find some k and we go on exchanging pairs from g 1 up to g k because if we take after g k if we include gk plus 1 gk plus 2 it only reduces the gain. The gain is highest when I have g 1 plus g 2 up to g k. Now what may happen is that this gain could actually some of these gains could actually be a loss.

Meaning that at a certain point in time let us say you have made a few exchanges and at a certain point in time all my pairs are given a loss. So, the highest gain is essentially the least loss. So, some of these gains could be a negative value. So, why do we at all include? May have include may have negative values in between some of the gain values even if it is negative, may be included in this in between.

Because some of the negative values due to this exchange may afterwards result in a very high gain for the subsequent. If I did not do this exchange, then later the subsequent exchanges would not give me that very high gain. Because I have now allowed this negative gain subsequently I get a very high gain. And then I finally, take up to go up to that value of k which gives me the final highest gain.

(Refer Slide Time: 09:16)



KL Algorithm - Gain

$\text{gain}(3,6) = 2+2-2\times1$

$= \boxed{2}$

- gain(*a*,*b*) = D(*a*) + D(*b*) - 2×c(*a*,*b*), where
  - c(a,b) = weight between a and b
  - D(*a*) = Out(*a*) - In(*a*)
  - Out(*a*) = total weight of all edges of '*a*' that cross the bisection
  - In(*a*) = total weight of all edges of '*a*' that do not cross the bisection

$\text{Out}(6) = 3$

$\text{in}(6) = 1$

$D(6) = 2$

Initial partition

Final

$\text{Out}(3) = 3$

$\text{in}(3) = 1$

$D(3) = 2$

1
5
2
6
3
7
4
8
1
2
6
5
8
3
7
4

Embedded Systems 12

We will take an example. Before that we define what is gain before going into the example. So, the gain is defined as D a plus D b minus 2 c a b where cab is the weight between a and b. So, c a b tells me what is the total cost of the data transfer between a and b. What is a and b? A and b are the nodes which we are thinking of exchanging. The highest among all such pairs we will take an actually tentatively exchange, but we have to test all pairs. Before finding out so, see in this algorithm such that exchanging a and b gives me the highest gain.

Therefore, to find out which one is the highest gain from all pairs which are currently unlocked. We have to test all pairs and find their gains and from them we have to find the highest gain. So, therefore, we have to find the gain of each pair. Now to we take any arbitrary pair a and b and then find out it is gain. And that is given by D a plus D b minus 2 c a b. Where c a b is the cost of the data transfer between a and b. D a is out a minus a; that means, the total cost of edges which cross the partition and in a is the total cost of edges which do not cross the partition.

So, D a is out a minus in a, were out a is the total weight of all edges that cross the bisection and in a is the total weight of all edges that do not cross the partition, ok. Now how what is the essential meaning of this? Let us understand this through an example. For simplicity let us assume that all these edges have cost 1. All these edges have cost 1. Now and I have an initial partition initial this one my phase one, and this let us say is my final partition. This is final and this is my initial random, initial partition I start with, this is my initial partition I start with.

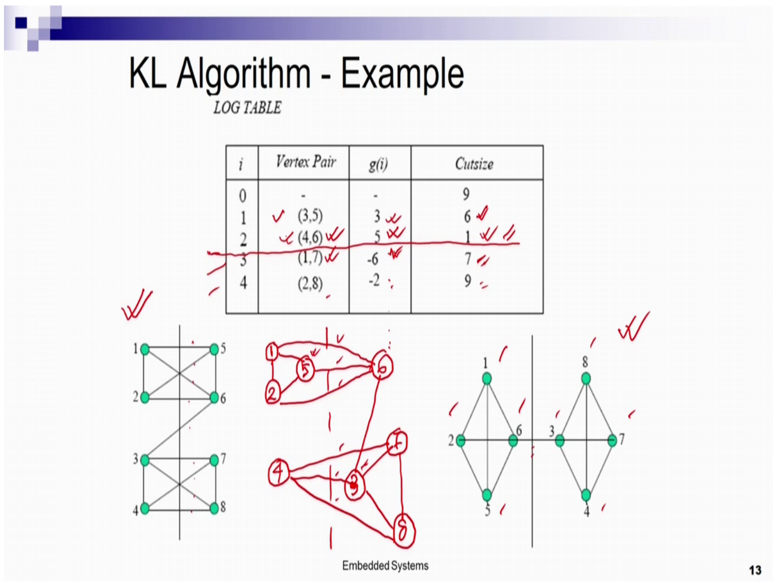
Now, let us say we exchange node 3 and node 6. What will be the gain? All edges have one weight. So, the number of edges that in out of 3; that means, edges which cross partition is 1 2 and 3; out of 3 equals to 3, what is in of 3 only one. So, in of 3 equals to 1. And therefore, what is D of 3 D of 3 is 3 minus 1 which is 2 which is what is out of 6 similarly 1, 2, 3. So, out of 6 is also 3 out of 6 equals to 3. And what is in of 3? In our 3 is also one it is got enough 6 in of 6 is equal to 1.

So, what is D of 6? D of 6 is also 2. Now what is therefore, gain of gain 3 comma 6, therefore, gain 3 comma 6 equals to 2 plus 2 minus 2 into 1. What is 1? The this edge 3, 6. So, what is the cost of the edge 3 6 is 1. So, it is 2 plus 2 minus 2 which is 2. Why do we do this minus 2? Because what the essential the actual gain is only 2 because although 3 goes to this side of the partition and 6 goes to this side of the partition. The 3 to 6 edge still remains in out. It does not go from out it does. So, all the other edges this 3 to 7 edge and 3 to 8 edge becomes part of in of 3 when it goes to the other side of the partition, but this 3 to 6 edge does not change from out of 3 to in of 3 because 6 also goes to the other side.

So, it still remains in out of 3. Similarly, the edge 6 to a is in out of 6 and when it goes to the other side of the partition. This 6 to 2 and 6 to 1, this edges move to in of 6, but 3 to 6 does not move to in of 6. It still remains in out of 6. And this is this edge 3 to 6 has been taken twice; one for this one in the calculation of D of 6 and other in the calculation of D of 3. So, therefore, we have to subtract it twice.

So, the essential gain is 2 which is 2 plus 2 minus 2. This is how we calculate gain.

(Refer Slide Time: 14:59)



KL Algorithm - Example

LOG TABLE

i	Vertex Pair	g(i)	Cutsize
0	-	-	9
1	(3,5)	3	6
2	(4,6)	5	1
3	(1,7)	-6	7
4	(2,8)	-2	9

Embedded Systems

13

Now, we will take the, this example in a bit more detail. So, we said that how does this algorithm proceed. This algorithm proceeds by initially starting with an initial partition this is my initial partition. And then in that initial partition for all pairs of vertices I find gain. And let us say we do not go into this procedure, but you can see it and check it that the highest gain is provided by the exchange of pair 3 comma 5. We already saw that if we exchange 3 and 6 we get a gain of 2.

And here it tells that if we exchange and here if we exchange 3 and 5 we get a gain of 3. Before that for the initial partition the size of the cut is 9. Each edge has a cost of one, how many edges cross the partition? 1, 2, 3, 4, 5, 6, 7, 8, 9 so, the cost of the partition is 9. We said that if we transfer3 and exchange 3 and 5, the cost of the partition will reduce to 6. Let us see how because the gain of 3 comma 5 is 6 will just take an example and see this.

So, after this step, after this step we add after this exchange of 3 comma 5, the what we will have is 1 2. And 5 will go to this side of the partition. Let us say this is the partition that we have. And 3 will go to this side of the partition this is let us say this is 3, 3 we have 7 here and we have 8 here, ok. And we also have a 4 here, and we also have a 6 here. So now, what are the edges that we have? We had 5, 2, 2 edge we had a 5 to 1 edge; we had a 1 to 2 edge. We had a 1 to 6 edge. We also had we also had a 2 to 6 edge. We have ha so, we have 6 to 3 edge, we have a 7 to 8 edge. 7 to 3 8 to 3 8 to 4 and also 7 to also we have 7 to 4, ok. And I have 5 to 6, I have 4 to 3 I have 4 to 3.

So now this these are all my connections. And the what is the cost of the cut 1, 2, 3, 4, 5, 6. So, when I transfer 3 and 5, I get a gain of 3 and the cost of my cut produces to 6. Now again the next highest gain that is we have found out. So, after this exchange I will lock 5 and 3. I will not consider them from further exchanges. So, what are the remaining nodes? 1, 2, 6, 4, 7, 8 out of these out of these nodes the pairs that we can consider exchanging is say 1 6, 2 6 and then 4 6, and then 1 7, 2 7 say 4 7.

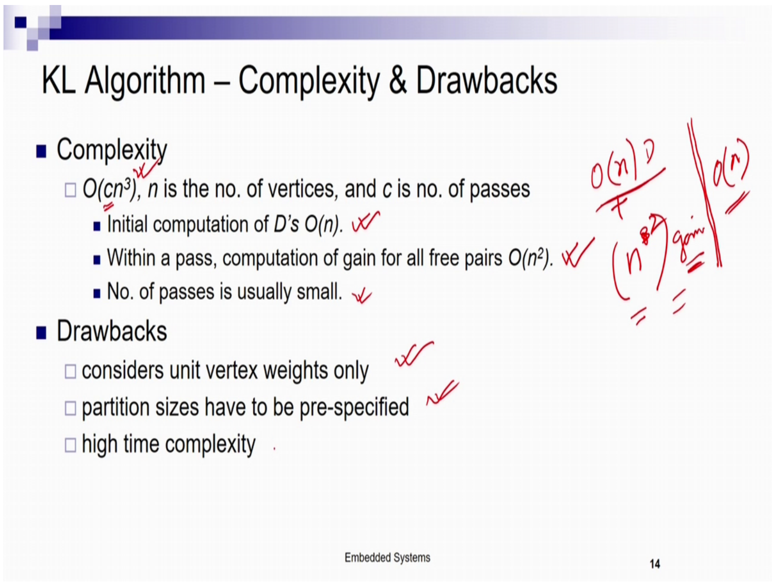
So, these are and likewise we will have these many pairs out of these many and many pairs like this. Let us say rather we will have n square pairs. And the highest gain that is given by let us say is 4 comma 6 and the gain is 5. Now if the gain is 5 what we get is finally, a cut size of 1 and here is this partition where we have already exchanged the 4 and 3 also. After we get 4 this side and 6 this side.

So, on this side we get 1 2 5 and 1 2 5 6. So, 1 2 5 6 on one side and 8 3 7 4 on the other side and the size of the cut is just one. Now if we exchange further we have to go on further until all vertices are locked, we will see in for this case we will see that the we do not get any benefit in gain further we only have negative gain. So, minus 6 gain. So, the cut size increases to 7 if we do an exchange of 1 7 and also 2 8. But this does not mean that this will happen in all cases sometimes for very large cases for very large task graphs it could be that some of my exchanges.

Let us say it increased cut size, but subsequently it may due to this exchange subsequently it allowed many more exchanges which reduced the cut size to a very small value. Therefore, allowing negative gains is also essential. So, this is how essentially the Kernighan and Lin algorithm works. So, for this case we will take because steps 3 and 4 do not give me good gains. So, we will only take up to the second step.

So, this is my final partition. So, we will consider g 1 plus g 2 because g 1 plus g 2 plus g 3 plus g 4 does not give me the maximum gain. G 1 plus g 2 plus g 3 also does not give me the maximum gain, but g 1 plus g 2 gives me the maximum gain. So, we will only go up to the second step and that will be my final partition. What about the complexity of the Kernighan and Lin algorithm?

(Refer Slide Time: 21:20)



KL Algorithm – Complexity & Drawbacks

- Complexity
  - $O(cn^3)$, n is the no. of vertices, and c is no. of passes
    - Initial computation of D's $O(n)$.
    - Within a pass, computation of gain for all free pairs $O(n^2)$.
    - No. of passes is usually small.
- Drawbacks
  - considers unit vertex weights only
  - partition sizes have to be pre-specified
  - high time complexity

Embedded Systems 14

O(n) D
(n²) gain
O(n)

The complexity is big O of c and cube. C essentially so, if we go into the algorithm back once more, we see that this is an inner loop. And the outer loop says until g equals to 0. So, I will take up to g k ha and then I will unlock all pairs and then repeat the process with the current partition.

So, this was the initial partition at the end of this I have taken up to g k, I have modified the partition and with this modified partition I will go into a second loop, until we have a phase we have a entire pass in which we have no improvement in gain, ok. And this path is typically not a large number of times, it is only a small constant number of times at the outer loop will be required to execute it quickly converges down to a gain of 0. I mean a few number of iterations of the outer loop. And hence c the c component in the complexity is essentially the complexity or with the number of times the outer loop goes. The inner loop has a complexity of n cube; why do we have that?

So, at each at each loop iteration, I will have big O of n nodes, which are unlocked. And for them for those big O of n nodes I have to compute the gain of for each of them because we have to compute the d values for each of those nodes. So, the cost of out edges and the cost of in edges, cost of edges that go outside the partition and minus cost of edges that have that go inside the partition.

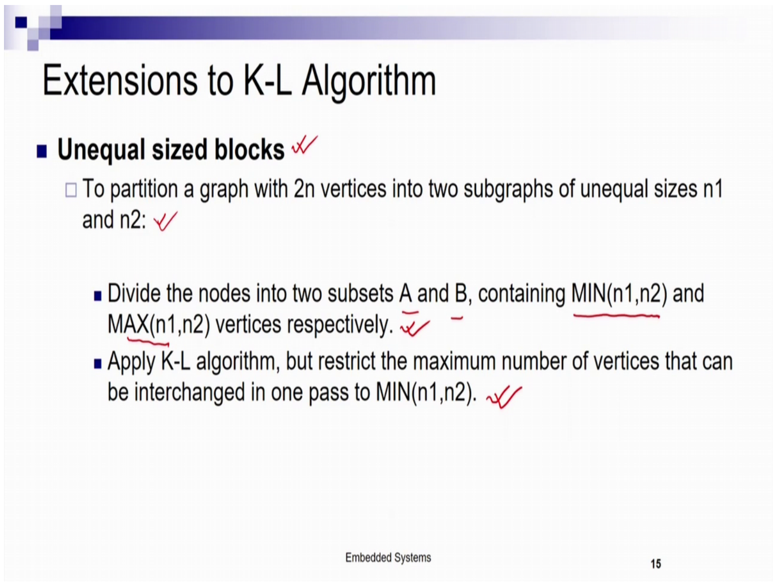
So, to calculate the d values of the n nodes for each node I will take a constant amount of time to calculate it. And therefore, for my n nodes I will take big O of n time to calculate the cost of D's to find the d values of the nodes. Then after finding the D values I will have to find the gain of n square pair of nodes. So, if I let us say have n nodes, then I have n square pairs of those nodes.

And for each of those pairs I have to find the gain values. So, to computing the gain values takes constant time. However, I have n such at each phase I have to recalculate the D values. So, I will have n plus n square. I will first calculate n then I will have n square, and then this will go on n times, because I have to go on locking unless all pairs are locked, unless all n pairs are locked I have to go on doing. So, that is another big O of n times. So, I will have to incur a big O of n cube complexity in this.

So, n square complexity in the calculation of gains firstly, you have to calculate d big O of n plus after calculating this you have this big O of n square complexity in calculating the in calculating for each pair of nodes, the gain values this is for gain values this is for D 1. First you calculate d in a separate phase with these ds you go on calculating the gain values in one phase and then you go on doing unless un until all these nodes are locked.

Unless and so, will the number of times this loop will go is again big O of n times because unless and until all pairs are locked, you will go on doing this loop ok. So, the overall complexity of this algorithm becomes bigger c n cube. So, within a past computation of gain for all 3 pairs this big O of n times number of passes is small and hence you have this c. Drawbacks of this algorithm it considers unix vertex weights only partition sizes have to be pre specified. In fact, equal size partition and the time complexity is high relatively high.

(Refer Slide Time: 25:48)


Extensions to K-L Algorithm

- **Unequal sized blocks**
  - To partition a graph with 2n vertices into two subgraphs of unequal sizes n1 and n2:
    - Divide the nodes into two subsets A and B, containing MIN(n1,n2) and MAX(n1,n2) vertices respectively.
    - Apply K-L algorithm, but restrict the maximum number of vertices that can be interchanged in one pass to MIN(n1,n2).

Embedded Systems 15

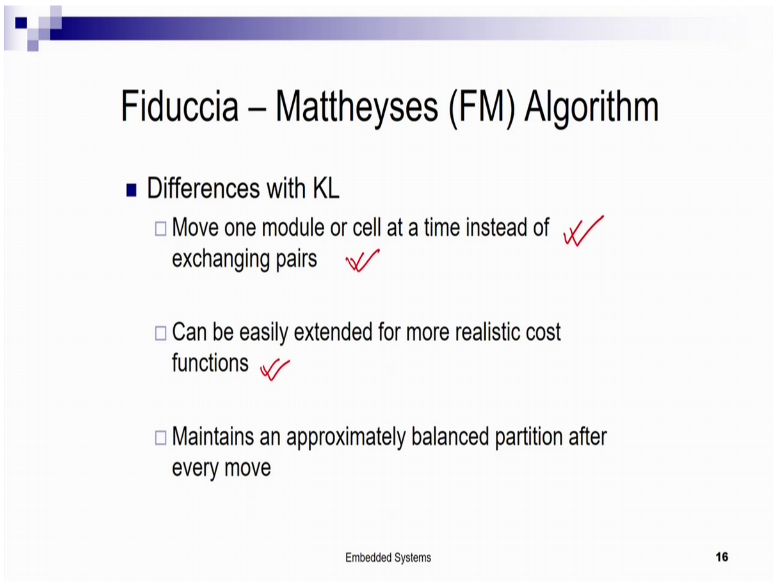
Now, some of the restrictions of the Kernighan and Lin algorithm one important restriction for example is to have unequal sized blocks. That can be extended Kernighan Lin algorithm can be extended to circumvent some of these problems

So, to partition a graph of 2 end vertices into 2 sub graphs of unequal sizes n 1 and n 2. So, now I do not want the equal number of nodes in n and n on each side of the partition on, let us say or for software I want n 1 node for hardware I want n 2 nodes and n 1 and n 2 are not same. So, how do we go about how can we extend Kernighan Lin algorithm to take care of the situation. We divide the nodes into 2 subsets a and b containing min of m n 1 n 2 and max of n 1 n 2 vertices respectively. Then what do we do? Then we apply kl algorithm, but restrict the maximum number of vertices that can be interchange in a path to min n 1 n 2. So, previously we were doing unless all pairs were locked.

So now the number of exchanges will be min of n 1 and n 2. So, one of n 1 in among n 1 and n 2, 1 is small, one is large. So, at least you have mean n 1 n 2 pairs. So, up to n 1 I know in one coming in 2 pairs you do and leave the rest. Because that this so, what will happen to the rest of the nodes which are not taken care it will finally, be taken care over many number of passes. So, we will go on doing this until g equals to 0. Not all the same node set of nodes will not be pawed be starved and not be exchanged iteration after iteration if it so, deserves right.

So, therefore, at each iteration it works fine if we allow only n min of n 1 comma in 2 pairs to be exchanged and then we go on through some passes.

(Refer Slide Time: 27:58)



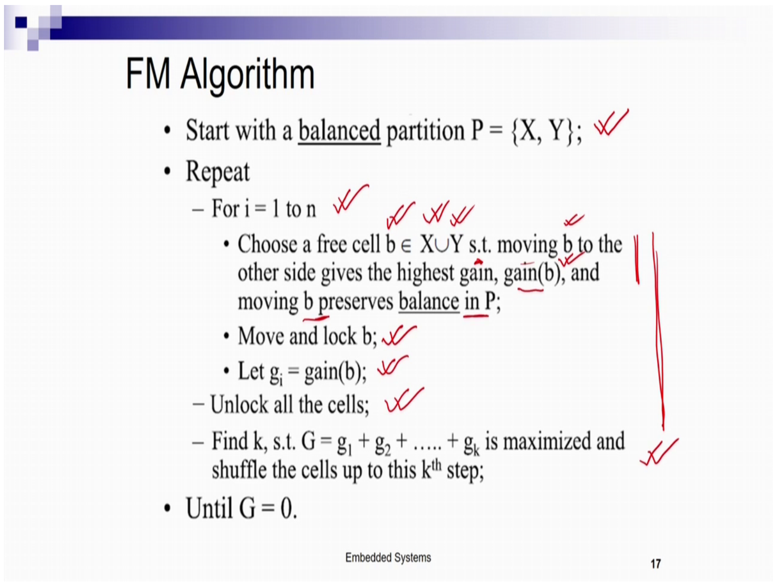
Fiduccia – Mattheyses (FM) Algorithm

- Differences with KL
  - Move one module or cell at a time instead of exchanging pairs ✓
  - Can be easily extended for more realistic cost functions ✓
  - Maintains an approximately balanced partition after every move

Embedded Systems 16

Last algorithm before finishing this lecture is the Fiduccia method s algorithm. It is an extension of the Kernighan and Lin algorithm. We move instead of node pairs in this algorithm we move one module or cell at a time instead of exchanging pairs. And can easily be extended for more realistic cost function, for example, let us say our objective of minimizing execution time. So, we had that and that cost estimation of execution time of modules. And this can be considered in this algorithm easily. And it maintains an approximately balanced partition after every move. So, let us go into and look at to this algorithm.

(Refer Slide Time: 28:43)



FM Algorithm

- Start with a balanced partition $P = \{X, Y\}$;
- Repeat
  - For $i = 1$ to n
    - Choose a free cell $b \in X \cup Y$ s.t. moving b to the other side gives the highest gain, gain(b), and moving b preserves balance in P;
    - Move and lock b;
    - Let g_i = gain(b);
  - Unlock all the cells;
  - Find k, s.t. $G = g_1 + g_2 + \ldots + g_k$ is maximized and shuffle the cells up to this kth step;
- Until $G = 0$.

Embedded Systems 17

So, we start with a balanced partition P X comma Y. So, what do we mean by balance? Let us say, that we want to implement we want to minimize execution time. And if let us say execution in hardware always minimizes execution time, wanting to minimize the execution time you will get that finally, everything should be in hardware. However, the balance tells me that you must have at least some of the nodes in software and the nodes in hardware a minimum number.

So, at least 70 suppose 75 percent at least three fourth of the nodes in software and only one fourth in hardware, you cannot have more than that. And that will be bounded within a certain value let us say between say 70 percent to 80 percent will be the case. Let us say, you cannot have less than 70 percent nodes in software. You cannot have more than 80 percent nodes in software and this balance will be maintained.

And maintaining this balance we will try to minimize what is the minimum execution time that we can obtain for a given embedded system. Let us say this is my objective. So, we start with a balanced partition p, and like Kernighan and Lin we repeat for I equals to 1 to m. Let us say the number of nodes that I have is n. And we choose a free cell b or free module b, either in X or in Y does not matter it is not exchanging.

So, I just choose a module from X or Y such that moving b to the other side gives me the highest gain. And now what is the gain? The gain is in terms of minimizing the execution time, of what? Let us say I have this entire application n 1, which calls say n 2 n 3 and n

4 we saw. And n 1 n 2 n 3 n 4 can be either implemented in hardware and software and then it is a movement of either the. So, so this moving of the cell represents therefore, the moving of n 1 n 2 n 3 or n for any one of them from hardware to software or from software to hardware, wherever it is such that this movement gives me the highest gain. And what is that gain for me now.

The minimum execution time that I have for n 1 because n 1 is the ultimate execution time of the overall application. This is my objective; the objective is to minimize the execution time of n 1. For that only I want to move modules from software to hardware or hardware to software.

So, I choose a free module b belongs to X union Y, such that moving b to the other side gives me the highest gain. And also moving b preserves the balance. So, after moving b we have to see that if I move b does it will preserve the balance that I required for example, at least 70 percent of the nodes will be in software and at most 70 percent of the nodes will be in a ha software. So, the number of nodes in software should always be the number of loads are the total and ha the total number of nodes in software will always be between 70 to 80 percent. Let us say is this balance maintained, after I have moved b.

So, if this balance is not maintained I cannot move b, I cannot move b. And I have to choose another particular module for movement another module for movement not this module. So, we have to see that does moving b preserves balance if. So, move b and lock b. Let us say gain is gained after moving b.

You unlock all cells and just like the next step is just like the Kernighan and Lin algorithm. You go on moving b and lock b unless all the b’s are considered for movement while preserving the balance. And after that is done one this inner for loop is done, and then you repeat this for a few number of steps. So, with this we come to the end of this lecture.