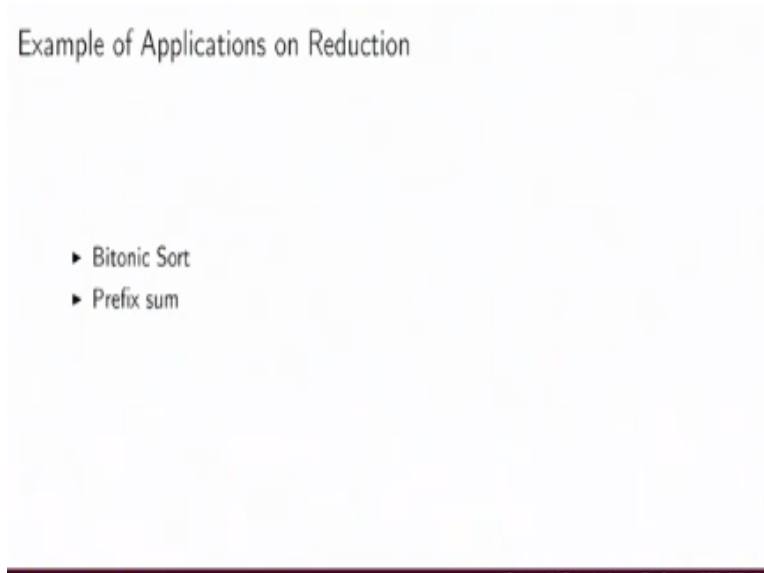


GPU Architectures and Programming
Prof. Soumyajit Dey
Department of Computer Science and Engineering
Indian Institute of Technology - Kharagpur

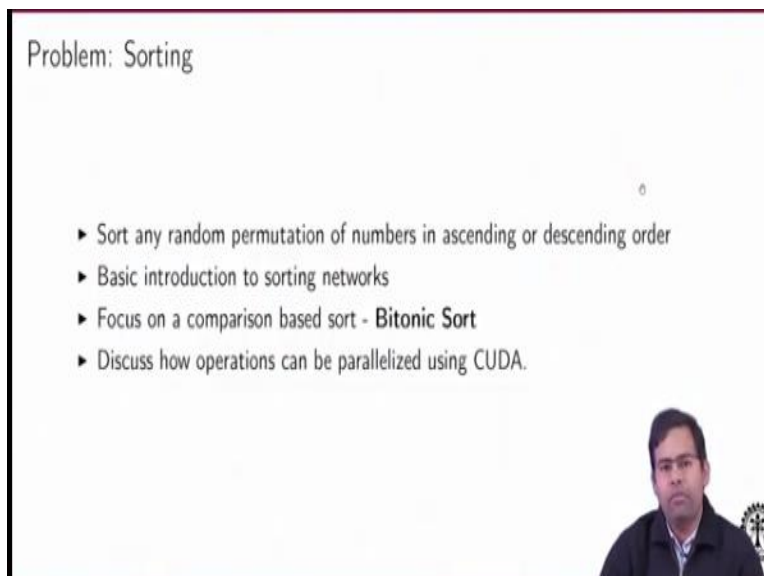
Lecture – 31
Optimising Reduction Kernels (Contd.)

(Refer Slide Time: 00:28)



Hi, welcome back to the lectures on GPU architectures and programming so, in the last set of lectures we were discussing the way the standard reductions can be optimized in with respect to GPU based parallelization. Now, we will go into some more algorithmic examples to be more specific, we will be talking about 2 examples; one is bitonic sort and the other is computation of a prefix sum.

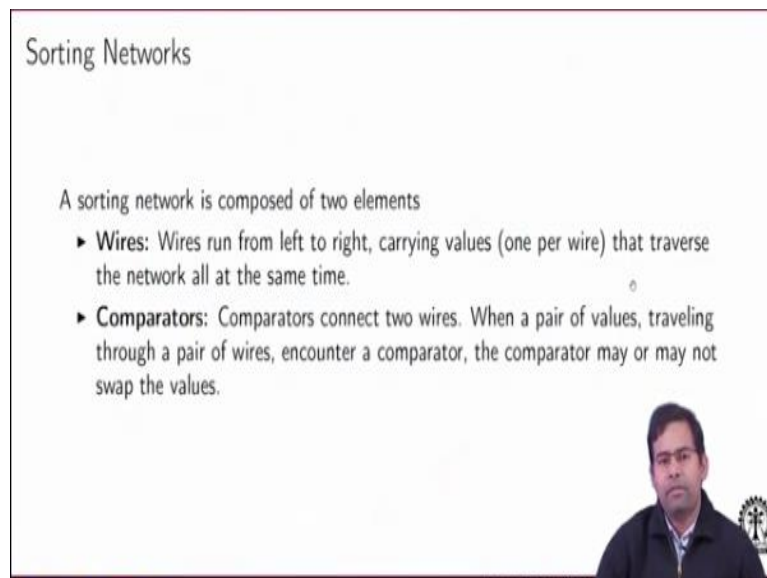
(Refer Slide Time: 00:55)



So, let us get started with this so, I mean when we are talking about bitonic sort; the basic problem of sorting is, you are given as input any random permutation of numbers and you want to output a sorted set which is either ascending or descending in there with respect to their respective keys so, as we know that there are a lot of well-known comparison based sorting algorithms.

So, in this case we will be choosing some algorithm, which is very well known for the amount of parallelization that it offers and so before going into this I mean, the idea of what basically is bitonic sort, we let us start with an introduction of sorting networks and because that is a key idea which will be used in the context of bitonic sorts.

(Refer Slide Time: 01:45)



Sorting Networks

A sorting network is composed of two elements

- ▶ **Wires:** Wires run from left to right, carrying values (one per wire) that traverse the network all at the same time.
- ▶ **Comparators:** Comparators connect two wires. When a pair of values, traveling through a pair of wires, encounter a comparator, the comparator may or may not swap the values.

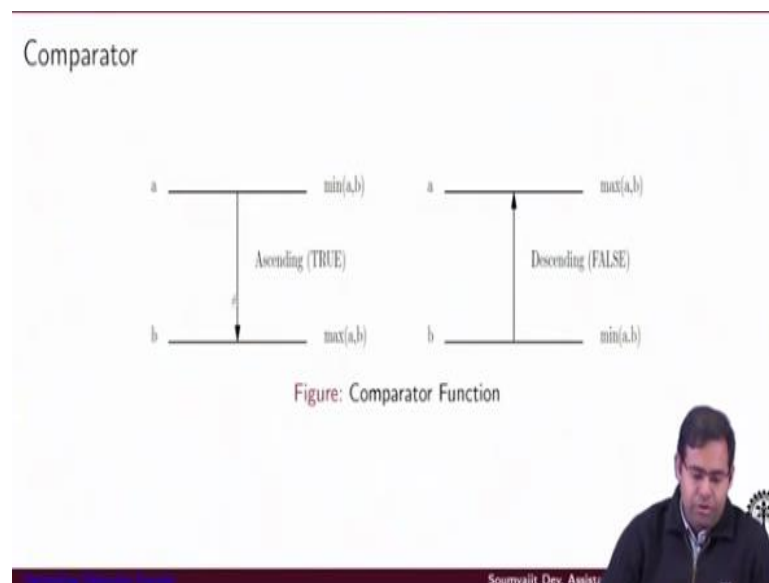
Copyright © 2015, All rights reserved.

So, first of all what is a sorting network? So, essentially it is a computational model I would say, network which is doing a sort which is essentially being used for performing the function of sorting, we are assuming that say, for example consider that you have some wires, the wires are running from left to right and they are carrying values. So, each wire is carrying some physical signal value, let us say some numerical value in terms of bits like that.

Consider an obstruction here now, the values are traversing through the network from the left to the right and in between, we have some comparator circuits connected to the wires, right. So, whenever I have a value coming in one input of the comparator circuit through one wire and another value coming through another input of the comparator circuit to another wire, the circuit will make a comparison.

And depending on whether I want the max or the min, it will mean, if I want that the maximum value should flow up or the maximum value should flow down accordingly, the comparator will work and swap the values across the wires, so that is the basic functionality here.

(Refer Slide Time: 02:58)



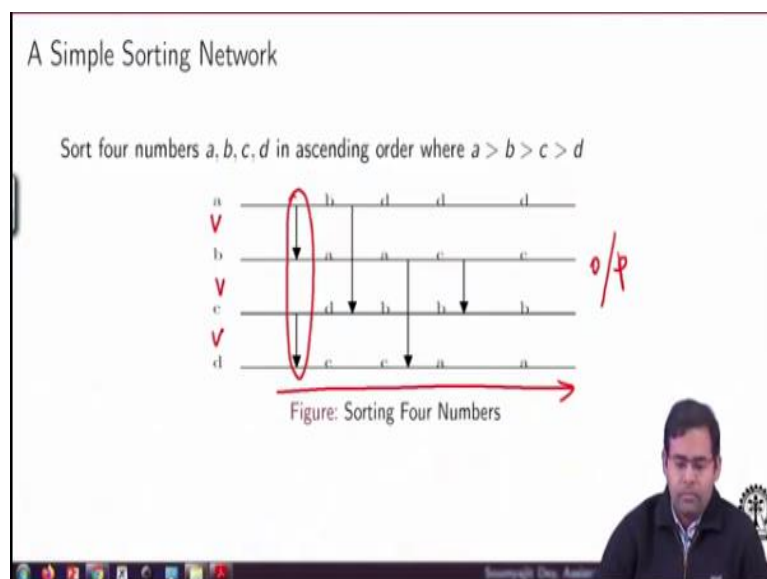
Let us take an example, so what we mean by this comparator operation? So, these are my wires running from left to right, they are being provided with this inputs a and b and this notation; this symbol down; this arrow pointing downwards, it is essentially telling me what I want is whatever is the input here, the comparator circuit is going to compare and find out what is the bigger value.

And it will push the bigger value to the downward and it will pop the smaller value to the upward wire, okay. Now, of course if a is anyway less than b , then nothing is going to change, if a is greater than b , then there will be a swap operation here. So, essentially the comparator ensures that whatever is the input at this point, I have the upper value to be less than the lower value, right that is the invariant that is going to be ensured in the output lines here, in this pair.

Now, similarly if we just reverse this arrow that means, we are looking for a comparator circuit which again compares a and b , the inputs and in the output, it is floating the maximum value in the upward wire and it is bringing the minimum value downward, so essentially we are saying that okay, we are trying to do a descending output here and here, we want an ascending output here, okay.

So, that is the primary goal of the comparator circuit, it is in this case so, you can just remember like this, the maximum value follows the arrow that is a simple way to remember. The max value is always going to follow the arrow in this case, it is going to percolate down in this case, it is going to percolate up so, here again in the output, the invariant that would be true is whatever is the value that I have in the wire here is always greater than the value that I have in a wire here, fine.

(Refer Slide Time: 04:57)



Now, what is the usefulness of these circuits and the sorting network we want to build? Now, consider a set of wires here, we have a more involved example, we are trying to build what we essentially mean by a sorting network here. So, essentially it is a collection of this kind of wires and comparative circuits are placed like this, okay. So, the important thing we need to understand here is the flow of time here.

So, essentially we would mean that leaves the input and this is the time following which, I mean this is the flow of computation following the time axis and here we are looking for the output. So, now if you observe what is going on suppose, the input is already in ascending order sorry, the input is in this order essentially, it is a descending order but we want the output to be in the ascending order, right.

So, since a is greater than b and b is greater than c , so on and so forth, in the output d , being the smallest element, if we want the output in ascending order, d should come followed by c , followed by b and followed by a and we can just to a small chip that this sorting network can

realize this functionality, so let us see what is happening. So, we have given a and b as input now, since a is greater than b, so a is going to flow down here, right.

The max will follow the arrow, a comes there, b goes there similarly, c comes here, d goes here again, as we can see that a is definitely also greater than b by transitivity and so b is also greater than d here, right so, d will go up and naturally, b will come down and similar behaviour we can take here. So, what is going to happen is after this again, we are comparing a and c here, so again c will go up and a will go down so on so forth, right.

Now, what is important to point out here is, you can look at the set of comparisons that are going on with respect to time and try and understand what are the activities that can be done in parallel, so essentially as long as I do not have a dependency between the inputs and outputs, those comparisons can easily be performed in parallel for example, here this set of comparisons, I can easily do them in parallel, right.

And can I do this 2 in parallel okay why not, because apparently there is no dependency between the inputs and outputs, right because here you are comparing a and d and here you are comparing a sorry, here you are comparing b and d and here you are comparing a and c, right, so even this could have been done in parallel. Now, what is happening is; so, as the switchings of the values between the wires go on with respect to time.

If we see the flow of values, the circuit is going to ensure that at the output, I get the values in ascending order; I get d followed by c, followed by b, followed by a, right. So, one thing we can now understand, it makes sense to think like this that I can use this kind of a sorting network to represent the underlying algorithm of a sorting function, right any sorting algorithm.

For example, bubble sort should have this kind of a sorting network representation or insertion sort should have a separate different, since it is a different algorithm, it should have its own sorting network representation, right.

(Refer Slide Time: 08:43)

Bubble Sort

Any comparison based sort can be done using a sorting network.

Soumyajit Dey, Asstt

So, for example this as we can see that this is a sorting network, which is going to do a bubble sort well, we will not discuss anymore here, we will provide the slides you can just look into this and understand why we are saying this represents the bubble sort algorithm here. Of course, the generalization would be that any comparison based sort algorithm can be represented using this kind of a sorting network.

So, the way we have drawn the picture is to make you understand what are the operations we can actually do in parallel without any dependencies and in that way, bubbles are or any other sorting algorithm as you can see, if we take a sorting network representation, it actually reveals to us what are the operations that are actually possible in parallel, right that is the most important thing.

(Refer Slide Time: 09:40)

Bitonic Sort

Bitonic sort takes place using two fundamental steps:

- ▶ Step I: Convert an arbitrary sequence to a bitonic sequence.
- ▶ Step II: Convert a bitonic sequence to a sorted sequence.

A Bitonic Sequence is a sequence of numbers which is first strictly increasing then after a point strictly decreasing. $a_1 < a_2 < \dots < a_m > b_1 > b_2 > \dots > b_n$

Soumyajit Dey, Asstt

So, coming back to our idea here we want to talk about bitonic sort, so this example; this algorithm has got 2 fundamental steps; the first step is that you convert any arbitrary sequence of numbers to a bitonic sequence, right. Now, what is the bitonic sequence? It is a sequence of numbers which is first increasing strictly and then it is decreasing strictly for example, this so, it is increasing up to a_m .

And then it is from a_m to b_m , it is strictly decreasing right, so this is a bitonic sequence and so, the first step of the bitonic sort method is to consider as input any permutation of values or key value pairs and convert it into this kind of a sequence and then the second step is well, you take this kind of a bitonic sequence as input and convert it into a sorted sequence. Just coming back once to this example, I hope from this it is clear to you that why we are trying to say that this is essentially a representation of bubble sort.

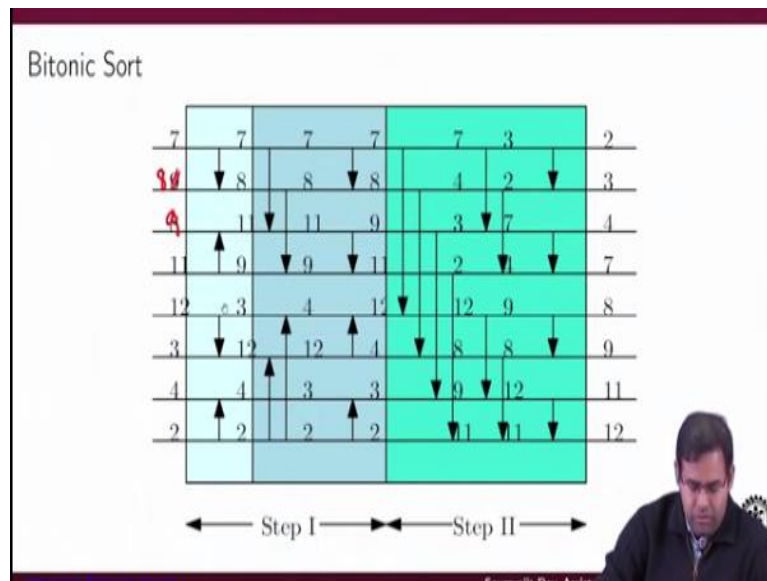
As you can see that if I just follow the track of these downward arrows, the max value is kind of flowing down right, the next max value will also similarly like flow down. The way we are drawing this picture is we are trying to show which are the operations that can be done in parallel, right so, maybe you can consider this kind of picture and try and figure out why this represents a specific sorting algorithm.

And you can also create sorting network representations of other sorting algorithms and figure out what is the parallelisation that is possible for that algorithm, right. So, coming back to this bitonic sort idea, so as we can understand the step 1 is to create the bitonic sequence, the step 2 is to take as input the bitonic sequence and then create a sorted sequence. The sort algorithm is actually going to implement these 2 steps in form of a and we shall be trying to view those methods in the form of a sorting network.

And then use the sorting network to identify, what are the operations that can be done in parallel similar to the previous examples of sorting algorithms, I will just repeat; so these are the 2 fundamental steps we are going to realize through a sorting network so, in step 1 we are going to convert, we are going to implement a sorting network which can take as input an arbitrary sequence convert the sequence to this kind of a sequence where it first increases and then it decreases, right, the sequence of key value pairs decreases.

And then, we have a stage 2 sorting network, which will take this kind of a sequence as input and it will provide a completely sorted sequence.

(Refer Slide Time: 12:50)



So, let us look into this for once, so let me just do a few corrections in this picture, here I should have 8 and this should be 9 in the input, I hope the rest is correct here yeah, okay. So, we are trying to give here a sorting network using which we are performing the bitonic sort, right. So, we have these inputs; 7, 8, 9, 11, 12, 3, 4, 2 as you can see, it is an arbitrary permutation and we are trying to mark out the different steps here.

So, I have a step 1 that means, as per our previous definition here, step 1 is going to create a sequence here where the values will first increase and then the values are going to decrease, right. Now, let us look at the network and see whether that is really happening or not so, in the first stage again, I will just repeat, this sorting network is trying to represent the operations with respect to time right.

That means, at this time slot I can do these operations in parallel, in the next time slot I can do the following operations in parallel, in the next timestamp I can do the following operations in parallel. Now, of course in between operations happening at different time stamps, if there are no dependencies, then you can parallelise the operations across time stamps, we have to understand this, right, we will give examples on that.

So, first point I am just repeatedly trying to make is, what is the sorting network trying to show you, it is trying to show you that this is the flow of time and at each time instant what

are the operations that I can do, go to the next time instant what are the operations I can do and again I will just repeat, if there are operations sitting at different time stamps but they really do not have any input output dependency, they can also be parallelised, right.

So, just; let us just first look at here the activities that are happening and try and figure out whether the functionality of the sorting algorithm is actually going to be realized through this sorting network. Well, first of all as we can see, there will be no downward flow here because 7 is always; 7 is of course less than 8, so the max is already here nothing flows down, here the max is below, so it will flow up, 11 goes up and 9 comes down.

Similar thing here and nothing changes here right, here what is happening; again, I have to compare between 7 and 11 nothing changes, 8 and 9 nothing changes but now 3 and so, now here, I have to compare 2 and 12 again, nothing changes and then I compared 2 with so sorry, so here we are comparing 2 and this value 12, so that remains same and then in this line, we are comparing the value of 2.

So, here we are comparing 2 with the value of 4 that is done and here, we have 2 and 12 and once I compare them again, there is no change that is going to happen and then again, when I compare here yes, so I will expect the maximum value to go up and it is already up here as I can see and just one minute, I think here so, the 4 is being here and yet again yeah, so the issue is yeah, this is from where it will start.

And so what I am going to get is; so I am comparing 2 and 12 nothing changes here and yeah now it is fine so, I am going to compare between 4 and 3 so, the 4 is the max it should go up following this so, 4 comes up and 3 goes down, right so, pardon the corrections here I have to do, extremely sorry for that and now, when I continue so again, I go to the next stage, nothing changes here but here I have 11 which should come down, right and 9 is going to come up.

And similarly, here 12 flows up, 4 flows down and then again, here nothing will change because 3 is the maximum and it is already here. So, now let us observe the outputs there because we are considering that step 2 finishes here, right. So, what are the outputs? As we can see in the output, I have 7, 8, 9, 11 followed by 12, 4, 3, 2 right so, it is ascending and then is descending, right.

So, it is ascending here and then is descending here, so that means this is a bitonic sequence because if you go back to our previous definition, a bitonic sequence is a sequence of numbers which will first strictly increase and then it will strictly decrease after that point right, so that is what is happening here. Then, we are going to the step 2 and in step 2, the definition is that okay, I am expecting the input in the form for bitonic sequence.

And I will transform it into either fully ascending order output or a fully descending order output right. So, in this case we are trying to transform it to a fully ascending output and let us see whether it is happening yes, it seems so, for example I am going to compare now 7 and 12, they are already sorted right, so then I compare 8 with 4, I exchange then I compare 9 with 3, nothing changes, then I compare 11 with 2 and so it changes within 2 and 11.

So that is done and then in the next stage, I compare 7 and 3, so again I have a change, I compare 4 and 2, again I have a change and here I compare 12 and 9, I have a change, I compare 8 and 11, nothing changes and then I am in the last change, where I compare 3 and 2 with a switch compare 7 and 4, again a switch 8 and 9, again a switch and 12 and 11, again a switch, right.

But what I get in the output as you can see is fully sorted in ascending order right, so apparently these bitonic sort network that we have shown here does this job. Question is does it really do its job in general? Well, you can study a bit about this algorithm and look into its correctness proof which is there in almost all of the algorithms books that are available because given the scope that we have here, let us assume that it works.


But one fundamental thing we will do; we look into the algorithm and try and understand why it works without going into proofs and other techniques for showing the correctness of the method.

(Refer Slide Time: 20:57)

Recursive Structure

- ▶ If you look closely, Step I uses Step II recursively on smaller sequences.
- ▶ Step II can be used to sort in any order (ascending or descending). The order can be controlled using the comparator.
- ▶ Step I uses Step II in a way to construct subsequences that are bitonic in nature.

<http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/bitonic/bitonicen.htm>



Soumyajit Dev. Asstt

So, if we look into the intuition behind the algorithm what is really happening, observe something that step 1 is using step 2 recursively on smaller sequences. Now, why is that so? Let us go back to the definition of a bitonic sequence so, it is first going to increase and then is going to decrease right, so then I would say that small as bitonic sequence would be to the just a collection of 2 numbers right, a collection of 2 numbers.

And then, if I apply my method of step 2, it is expecting as an input a bitonic sequence and it is going to provide as output in a sorted order ascending or at descending, right. So, essentially I can say that whatever is the method of step 2, it will take as input bitonic sequence and produce a fully sorted sequence, right. So, even for building the basic steps of step 1, I can apply step 2 recursively on the smaller sequences by starting with the smallest bitonic sequences which are of size 2, right.

I mean in that case for step 2, the only thing I will have is a single phase here right, is that that okay, you are going to; you are given the function bitonic sort which is expecting bitonic sort step 2 which is expecting the input to be as a bitonic sequence so, I can just say that the way I am going to draw up the algorithm is; so consider step 2, give it as input just as smallest bitonic sequence to different values.

And give the algorithm the order you want, right that whether you want it to be an ascending order or you want it to be a descending order. What you get is output is 1 comparator right, a smallest sorting network containing only 1 comparator, right. So, in that way I can keep on

applying the step 2 recursively on smaller sequences from here and keep on constructing the different phases of step 1, right.

So, in that way we need to remember that for constructing step 1, I can look into it as constructing sub sequences by using smaller versions of step 2, if we look into this what is happening here; here step 2 is expecting an input which is the 8 size bitonic sequence, right. If I look into the method here, just this sub part right, if you look into this sub part, then I can say that well what is the input and this input I mean, if this is an input; 4 size input, here you have a bitonic sequences of output, right.

Because it is ascending and then it is descending, right and then if you look into this part you are essentially applying step 2 on an input of size 4, this is your step 2 on an input of size 8, here you are applying step 2 on an input of size 4, right. In that way if you generalize, you can look into every part of step 1 and start saying that this is nothing but a smaller instance of step 2, so this is the larger step 2, this is the smaller instance of step 2.

And this is the even smaller instance of step 2, which is nothing but a single instance of a comparator right. So, we will use this recursive formulation to generate the algorithm.

(Refer Slide Time: 26:07)

```
Recursive C Program

//Comparator
void compare(int i, int j, boolean dir){
    if (dir==(a[i]>a[j]))
        exchange(i, j);
}

//Step II
void bitonicMerge(int lo, int n, boolean dir){
    if (n>1){
        int m=n/2;
        for (int i=lo; i<lo+m; i++)
            compare(i, i+m, dir);
        bitonicMerge(lo, m, dir);
        bitonicMerge(lo+m, m, dir);
    }
}
```

First, let us look at a recursive C program which can perform the bitonic sort by using the idea we just discussed so, this is our basic comparator, it is considering 2 inputs i and j and it is given a direction, right. Now, look at the way we are writing the program here so, if the

direction value is 1, a_i is greater than a_j , so you are giving a_i and a_j as input and you are doing the comparison.

If the direction is 1, if a_i is greater than a_j , then you are going to exchange right, so you are exchanging i and j , essentially in the output lines what you have; if a_i is greater than a_j , then you do the exchange and you get effectively the value here this and the value here a_i , right. So, what we get here is the bigger value is flowing down so, this is the equivalent sorting network representation.

Because since a_i is greater, you are doing the exchange right, so the bigger value is flowing down and this is the network that you have, right. So, this way this small comparator program represents this sorting network with a downward direction. Now, look at the merge step and let us understand why it works, why do I say that the merge step is important?

(Refer Slide Time: 28:29)

```
Recursive C Program

//Step I
void bitonicSort(int lo, int n, boolean dir){
  if (n>1)
  {
    int m=n/2;
    bitonicSort(lo, m, ASCENDING);
    bitonicSort(lo+m, m, DESCENDING);
    bitonicMerge(lo, n, dir);
  }
}
```

The diagram shows a 2x2 grid. In the top-left cell, there are two vertical arrows pointing downwards, labeled 'step I'. In the top-right cell, there are two vertical arrows pointing upwards, labeled 'step II'. A horizontal arrow points from the top-left cell to the top-right cell. A vertical arrow points from the top-right cell to the bottom-right cell. A horizontal arrow points from the bottom-right cell to the bottom-left cell. A vertical arrow points from the bottom-left cell to the top-left cell. This represents a bitonic merge operation.

Because we understand that fundamentally, we are going to use this recursive formulation because we have understood the underlying recursive structure of the problem, if we are given a bigger; if we have to implement a bitonic sort algorithm, what we will do is; we take the whole input, divide it into 2 parts right, these are entire input divide the input into 2 parts, for the first part you apply the bitonic sort algorithm with the direction ascending.

So, the values will be increasing right so, the values increase right, for the next part you again apply the bitonic sort algorithm with the direction reversed right, so the values would actually be decreasing downwards, so here you have an ascending sequence followed by a descending

sequence that gives you the bitonic sequence, so this is essentially your step 1 and then you have step 2 which is essentially the bitonic merge.

Because already we have discussed, it expects ascending followed by descending and it is going to provide a fully ascending or descending output depending on what is the requirement, right. Now, we have also discussed that the step 1 is nothing but a generalization of step 2, I can keep on applying step 2 recursively on sub structures to realize step 1 right, which means it is fundamental to understand the bitonic merge problem.

And all we need to do is; we will be recursively calling the bitonic sort on smaller instances and finally, the bitonic merge function is the function which is going to do the job for us because we have already identified I am just repeating here that step 2 or the merge is going to be the fundamental thing effectivity that is doing the sorting for you on the smaller instances and just flowing it out on the larger values.

So, if we look into bitonic merge in the general case so, you have given a low and a high essentially, the starting point and the ending point of values and you have given the direction, right and you are asking me to do the merge right, so what is really going to happen? So, now to better understand this, let us first look into the merge sorting network, as you can see this is the merge step.

The first thing we do is a sequence of comparisons how many; half of the size of the inputs, there is a number of comparisons we are doing, what is the stride of the comparison; again half of the size of the input, so at that size stride, I am doing 4 comparisons here and that is essentially followed by what; again a smaller instance of bitonic merger and the smaller instance of bitonic merge here, now this is important.

Let us understand again, the recursion present inside the bitonic merge step, so this is very; this is really something we need to understand. First point is we understood that step 2 is the most fundamental thing, step 1 is nothing but a recursive application of step 2 because what is happening in step 2 is; you are considering as input a bitonic sequence and then you are providing as output and the other sorted sequence, right.

And when we apply step 2 on smaller instances and combine them that is essentially giving me step 1. So, with this understanding we will end this lecture and discuss the details of step 2 in the next lecture, thank you.