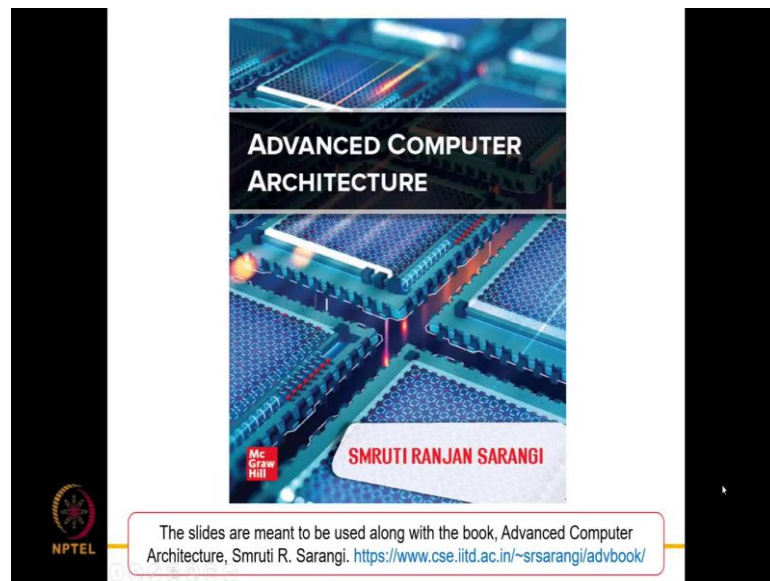


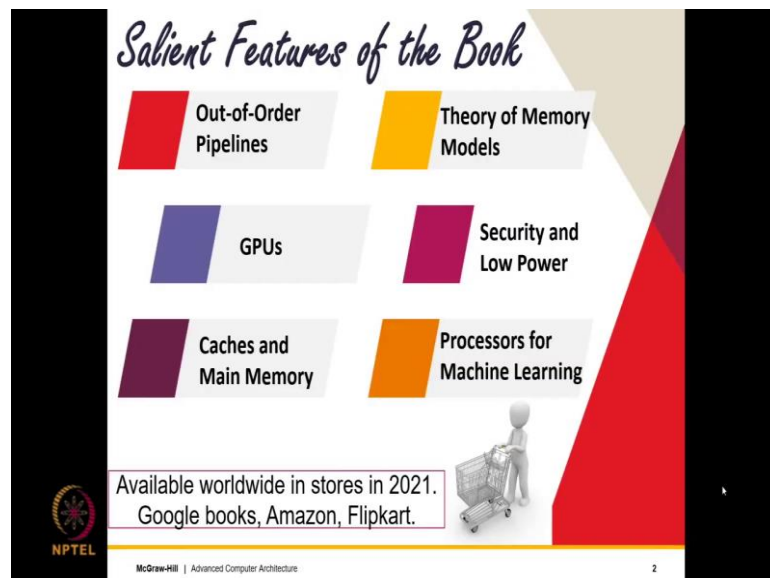
Advanced Computer Architecture
Prof. Smruti R. Sarangi
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi

Lecture - 22
Caches
Part - IV

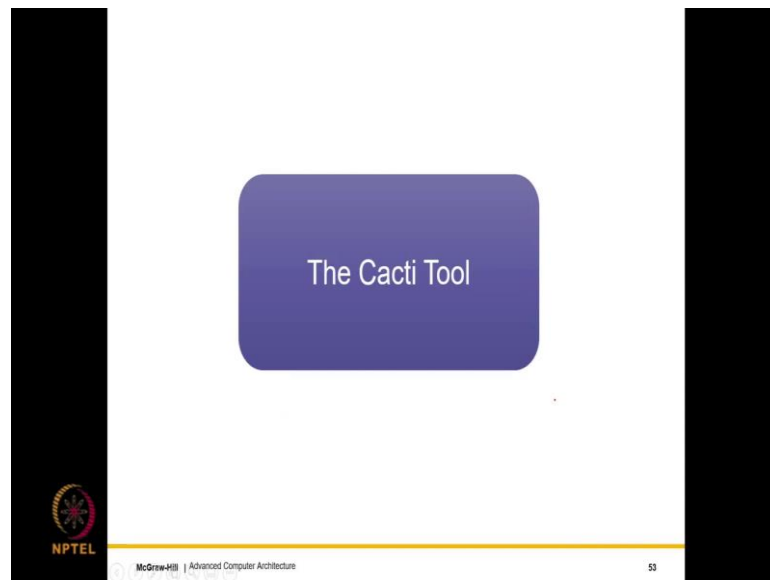
(Refer Slide Time: 00:17)



(Refer Slide Time: 00:23)



(Refer Slide Time: 00:35)



In this lecture we will discuss The Cacti Tool which is used to model caches and we will also discuss the Elmore Delay Model that is used by the Cacti tool to model timing and power.

(Refer Slide Time: 00:55)

A slide titled "Modeling and Simulating Caches" with handwritten annotations. The main text asks: "Should I use a 4 KB, 2-way assoc. cache or an 8 KB, 1-way assoc. cache? Should I allow concurrent accesses?". Handwritten red annotations include "2" above "2-way" and "1 cycle" above "1-way". A red question mark icon is next to the question. Below the question, a list of steps is provided: "First, we need to find the access times of both caches", "Convert access time into clock cycles", and "Simulate both the configurations: find the faster one". A purple circle icon is next to the text "To find the time it takes to access a cache". Below this, a list of points includes: "We need to use a simulation tool", "Most popular simulation tool -> Cacti (designed by HP labs)", "Along with that", "We need power and area data as well", and "Cacti 6.0 provides all of these". The slide is framed by black vertical bars on the left and right. At the bottom left, there is an NPTEL logo. At the bottom center, it says "McGraw-Hill | Advanced Computer Architecture". At the bottom right, the number "54" is visible.

So, let us start by asking a simple question. Any architect will always be confounded with such dilemmas. For example, should I use a 4 KB 2-way associative cache or an 8 KB 1-way associative cache? So, well then this is not a very clear cut question. Well, it does not have a clear cut answer. The reason is it could be that the access time of this is two cycles

and the access time of this is one cycle, because it has a lower associativity, it could be the reverse.

Furthermore, the miss rates could also be different, hence unless we perform a thorough simulation where we find out the access times of these two designs and then we simulate them for a large number of workloads. We will not be in a position to say which design is actually better, because they are sort of in the same range and without a lot of simulation studies it is hard to say which one is better. But what we definitely do need is given the design of a cache we should be in a position to find out how long it takes to access for a read and write and secondly, what is the average power dissipation?

How much energy is required to for making a single access? The other important question is how many concurrent accesses should we allow? Should we allow one access per cycle? should we allow two accesses per cycle? and so this of course, has a cost in the sense if we have a multi issue pipeline then having multiple accesses per cycle is a good idea, but this has a cost.

So, we will see what is the cost and how to do it, because the SRAM array that we have presented allows a single access right. It does not allow concurrent accesses. So, we need to do something more we will see what we need to do. But, what we need to do is that the basic three steps have to be understood first before we go into more advanced designs.

And, a three basic steps are that we need to find the access times of course, under model of how many accesses we will allow per cycle 1 or 2, the access time will then be converted to clock cycles. Then we perform detailed architectural simulations of both the configurations and then we find the faster one. Whichever one is faster the that design as chosen.

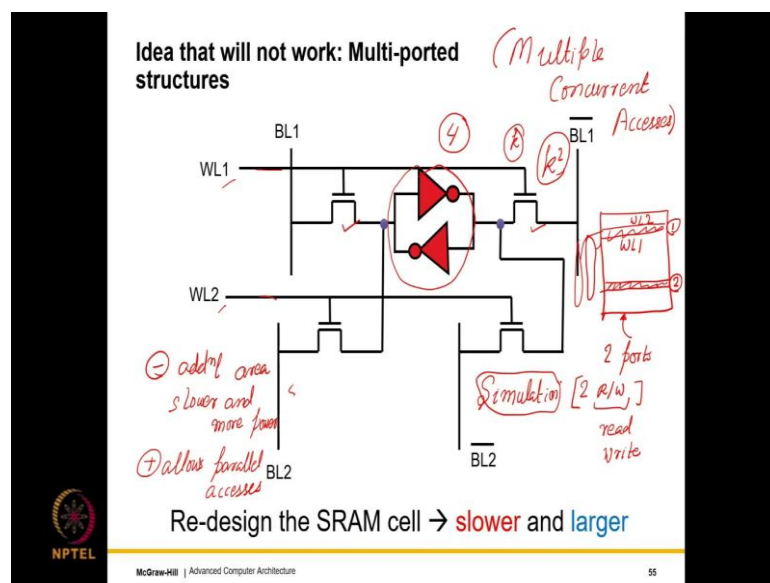
So, the Cacti tool which was designed by HP labs its very popular it used to have a web interface at least till 2 years ago, but now the tool is freely downloadable, it can be downloaded and modified. So, this will tell us the access time the timing power and area essentially gives three statistics. And, so the latest version will also give leakage powers.

So, in chapter 11, we will discuss what leakage power is, but for the time being let us see that it is the static power dissipation which is the power consumed even though there is no active access. So, to actually model a cache and see how long it takes we need to see how

it is constructed. So, we will actually find that it is at its heart, at its core, an optimization problem. So, we need to solve an optimization problem we can either use brute force that is a bad idea.

So, what I would suggest is that you should devise AI based techniques to actually solve the problems. So, I will discuss what are the norms, what is the optimization problem like and how it can be solved with AI based techniques. So, you never thought that you can use AI to design a cache. It turns out that you can.

(Refer Slide Time: 05:13)



So, first let us look at how to do multiple concurrent accesses. So, one simple solution, one simple idea which will well, which will work in a sense or not work in a sense. So, it will work from an electrical sense it will work, but from an computer architecture sense its a bad idea, which is that we modify the basic SRAM cell. So, the inverter pair is over here.

So, the basic SRAM cell that has fixed transistors would have consisted of this inverter pair, this word line transistor and this word line transistor. What we do is that we add two additional transistors. Which means; we add an additional word line, word line 1 and word line 2 and along with that we add two additional access transistors and two more bit line. So, one more bit line pair basically.

So, we have two bit line pairs, and we have two word lines. See, we do this for every single cell in the array. So, basically in the entire array what we do is that we have two decoders,

so we have a decoder 1 and we have a decoder 2. Decoder 1 drives the word line 1, decoder 2 drives the word line 2 for every cell.

So, it is possible to in parallel access two different rows of the SRAM array. So, we cannot still access the same row, but we can access two different rows, because row the first row we can activate word line 1 and bit line pair 1.

So, then we can read or write this row. And, the second row we can do the same we can activate word line 2 and bit line pair 2 and we can access that row. So, let us say it is one row is this one and the other row is this one. So, this one can, so the top one can use word line 1 bit line pair 1 in the bottom row can use word line 2 and bit line pair 2.

So, as far as we are concerned they are separate accesses, because they have that separate word lines and bit lines and we can have concurrent accesses in this way. So, in this case the SRAM actually has two ports, each port allows a read or a write. So, we say we have two read write ports. So, we say we have two read write ports. We can instead design the SRAM in such a way that we have one read write port just one read port. So, the other read port will not have a write driver, will not have the facility for writing, but it will have the facility for reading.

Similarly, we can say that look instead of a generic read write port we can have just a read port or just a write port. So, this will reduce the amount of hardware, but its architectural implication needs to be understood. So, what we need to understand is that we need to solve an optimization problem. So, there is a tradeoff. So, whenever, wherever there is a trade off there is an optimization problem at its heart.

So, let us understand what is good and bad about the design. So, let us consider this design that we have an option. Either we have an SRAM cell with 6 transistors that allows a single access per cycle or we have an SRAM cell with as you can see 8 transistors that allows two accesses per cycle. So, we will see whether these are read write or just read just write, that is a later decision.

But the important point is, that since we are doubling the number of word lines and we are doubling the number of bit lines, what actually happens when we layout this circuit is that the area of the circuit will increase by 4 times, because the y axis is having double the number of word lines and the x axis is also having double the number of bit lines

intersecting it, and because we need to maintain a minimum distance between two bit lines known as the pitch. So, it will turn out that the area will increase by roughly 4 times. So, on similar lines, if we have k ports, the port is an interface for a read or write.

If you have k ports, the area will increase K^2 times. So, of course, one of the negative factors is the additional area. So, we of course, need more area for the transistors, but the more important thing is we need more area for the wires and the latter dominates. The other thing is, of course this is slower and more power, because it's a bigger circuit bigger the circuit more power. So, these are all negative, but the positive is that it allows concurrent accesses. Let us say it allows parallel, that is the big positive over here that it does allow, it does allow parallel accesses.

So, this can be used to increase our IPC because let us say that one in three instructions is a memory instruction, and we are issuing six instructions per cycle we expect to have two memory instructions, in that case our IPC will undoubtedly increase. So, of course, here the question is that can we do 2 read write ports or just 1 read write and 1 read or 1 read write and 1 write.

So, of course, there we will pay an area penalty, but we can slightly reduce the amount of hardware there are a lot of choices. How do we resolve them? Well, so the gold standard is we do architectural simulation where we take a wide variety of programs and we just simulate the different configurations.

And of course, we analyze all the data so we decide you know complicated trade off, that is actually the job of a computer architect that different sites pull him from different directions. So, somebody might be saying that look let us have a bigger cache with cells like this that allow multiple ports. Others would say that look we do not have enough area for the cache, so keep the cache as small as possible because we need to fit other elements on the chip.

So, there is a lot of push and pulls. So, this creates a very fertile space for tradeoffs and optimizations, but the main problem, one reason why such designs did not really take off, is mainly because if we increase the number of ports k times the area increases K^2 times.

So, ultimately the area becomes the most dominating concern. So, we will look at how it is done how this problem is solved in a slightly different manner and for that we will have

to go through details of the Cacti tool and then I of course, I will discuss the optimizations that are involved.

(Refer Slide Time: 12:52)

Usage

`cacti C B A`

- C → cache size in bytes
- B → block size in bytes
- A → associativity
- Number of banks (a bank is a sub-cache) (more later ..., assume 1 for now)

Hidden inputs

- b_o → output width (32 or 64 bits)
- b_{width} → input address width (32 or 64 bits)

Diagram: A box labeled 'inputs' has an arrow pointing to a yellow box labeled 'Cacti'. An arrow points from 'Cacti' to a box labeled 'area,time,power'.

NPTEL
McGraw-Hill | Advanced Computer Architecture
56

So, the Cacti tool famously takes 3 inputs. Primarily, they are called A B C inputs. So, the orders in which they are given to the tool is reverse C B and A. C is the cache size and bytes, B is the block size and bytes and A is the associativity. So, then there is another parameter that can be provided it is the number of banks where a bank we can think of as a sub cache, but we will discuss more later.

And, there are two more inputs which are hidden, but you can always change the code of Cacti and change them. So, one is that what is the output width in the sense how many bits or bytes do we read out in one go. So, it could be 4 bytes, could be 8 bytes, can be configured. And the other is, what is the width of the memory address. So, that can also be 4 bytes or 8 bytes 32 or 64 bits again can be configured.

So, we give it the inputs and the output. So, the input such as A B C inputs with a few implicit ones and output is of course, the area timing and power, but of course there are many many ways of physically designing the cache as we should see. So, Cacti will choose the most optimal design based on what you tell it to optimize it for, minimum area, minimum time, minimum power any combination thereof and then it will give you the specs of the final design.

So, Cacti is both an estimator of area, time, and power as well as it also computes the most optimal design of the cache given your parameters.

(Refer Slide Time: 14:41)

Sample input and output

Normal Interface
 Detailed Interface
 Pure RAM Interface
 FAQ

Cache Size (bytes): 16384
 Line Size (bytes): 64
 Associativity: 4
 Nr. of Banks: 1
 Technology Node (nm): 32

Cache Parameters:

Number of banks: 1
 Total Cache Size (bytes): 16384
 Size in bytes of bank: 16384
 Number of sets per bank: 64
 Associativity: 4
 Block Size (bytes): 64
 Read/Write Ports per bank: 1
 Read Ports per bank: 0
 Write Ports per bank: 0
 Technology Size (nm): 32
 Vdd: 0.9

Access time (ns): 0.614536796032
 Random cycle time (ns): 0.22095719361
 Multisubbank interleave cycle time (of data array) (ns): 0.211907965098
 Total read dynamic energy per read port (nJ): 0.0779954960649
 Total read dynamic power per read port at max freq (W): 0.352989181841
 Total standby leakage power per bank (W): 0.00635633730771
 Refresh power (percentage of standby leakage power): 0.0
 Total area (mm²): 0.326190937722

Handwritten notes: $\sqrt{2}/12$, $22 \rightarrow 14$, $7 \leftarrow 10$, Scaling rules, 7nm, 0.68

So, a sample input output of how it works. So, we provide the cache size over here in bytes, it is a 16 KB cache line size the block size is 64 bytes, the associativity is 4 maybe you cannot see that 4 is written. The number of banks well we will take a look at the bank issue later, but let us assume its 1 for now.

And, the technology node that is very important the technology node here is said to 32 nanometers. So, as you know every 1 to 2 years use to be 1 is 2 now, the technology node reduces by a $\sqrt{2}$ factor of $\sqrt{2}$. So, given that each dimension reduces by $\sqrt{2}$ the area of a transistor reduces by a factor of half. So, this is how the technology nodes have been progressing we are currently at 7. So, what is the technology node?

Well, the technology node or the feature size is the minimum is the size of the smallest structure that can be fabricated on silicon reliably. So, traditionally it is the DRAM half page. So, well, when we discuss DRAM I can tell you more about what is DRAM half page and that is going to come in chapter 10, but essentially if let us see what happens is that in DRAM we have pretty much a single capacitor like this.

See, if we have let us say these parallel copper wires primarily with moss based capacitors. So, the pitch is essentially the distance between two parallel wires and half pitch is half of

that. So, the output is shown over here. So, the output shows the input first, number of banks, cache size, size and bytes it computes the number of sets and so on and of course, the read write ports read ports write ports.

So, we have a single read write port. So, Cacti also has an advanced interface where these things can be configured. So, the most important output is the access time, which is 0.61 nanoseconds and then a couple of more energy and power and area statistics. So, the area is 0.326 mm square.

So, most tools like Cacti will maybe end at 32 nanometers, but if you want to use it for 7 nanometers then we apply what are called scaling rules that take this data that Cacti has produced and multiply them with a set of well accepted factors derived empirically of course such that these values can be scaled to let us say a 7 nanometer process. A 7 nanometer feature size.

So, you will find a lot of I triple E papers that have different scaling rules. Some of the papers are well accepted. Any of those scaling rules can be used to convert numbers generated for a 32 nanometer technology to let us say a 10 or 7 nanometer technology.

(Refer Slide Time: 18:02)

How does Cacti work?

- Given the **inputs**, it tries to create the most optimal (often fastest) cache
- What can happen?
- Caches can get very asymmetric. rows \gg columns, or columns \gg rows (\gg means significantly more than)

The diagram illustrates two cache designs. On the left, a tall, thin yellow rectangle is labeled 'Slow design'. On the right, a square yellow shape is labeled 'Fast design'. A red arrow points from the text 'most optimal (often fastest) cache' to the 'Fast design' square. A red bracket above the 'Fast design' square is also present.

NPTEL
McGraw-Hill | Advanced Computer Architecture 58

So, how does Cacti work? Well as we have discussed given the inputs it tries to create the most optimal design. So, what can happen? Well, if you do not create an optimal design you will come up with bad designs like this, where you know it can be very long let us say

in 1 axis. One problem is physically constructing such a cache is hard, because you will have to unnecessarily displace other elements in the chip and it just might be very hard to place it.

So, just placement will become very hard. And also, if you have too many rows in two few columns your row decoder will become huge and very slow. Similarly, if I have my cache is like this well first it will be hard to place; second, we will have too many columns, so you know the column MUX DEMUX will become very slow. So, a fast design is typically squarish its not fully squarish, but typically looks somewhat, like a square it subject to optimization of course.

(Refer Slide Time: 19:13)

How to make it fast?

- Set the aspect ratio first:
- Number of sets mapped to a wordline $\rightarrow N_{spd}$
- A higher N_{spd} will increase the number of column multiplexers \rightarrow slowdown
- However, it will decrease the number of rows, and thus the size of the address decoder \rightarrow speedup

Then create subarrays of SRAM cells

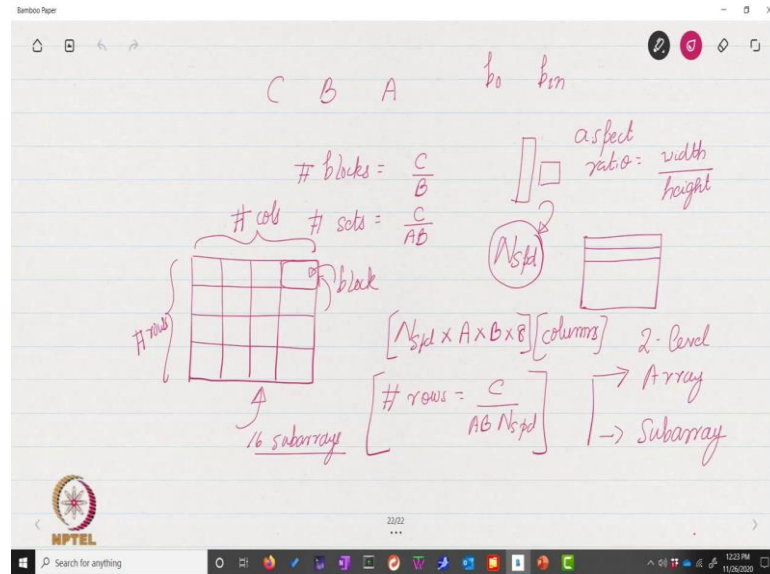
- $N_{dwl} \rightarrow$ Number of vertical cut lines
- $N_{dbl} \rightarrow$ Number of horizontal cut lines
- Total number of subarrays created $\rightarrow N_{dwl} * N_{dbl}$

The optimal values of N_{dwl} , N_{dbl} , and N_{spd} are found out by Cacti

NPTEL
McGraw-Hill | Advanced Computer Architecture
59

So, how do we do it? So, here is the idea that how do we actually design a cache. What exactly are the steps, the optimization processes? So, let me now talk about the broad space of optimization. So, what are our inputs?

(Refer Slide Time: 19:37)



The input is the cache size in bytes, block size in bytes and the associativity. So, from this we can calculate a couple of things. So, we can actually calculate everything and we also know the memory address and all of that. So, we know the number of bytes we want to read out and let us say let us call it B in or B width which is the size of the input address. So, on the basis of that we can say that the number of blocks in the cache = $\frac{C}{B}$.

The number of sets in the cache = $\frac{C}{AB}$. And, then what we can do is that the first question that needs to be answered is, we need to find out the aspect ratio of the cache, where we can say the aspect ratio is, let us say the height divided by the width. So, the aspect ratio.

So, when we buy a TV so let us define it the width divided by the height, because that is traditionally the way that it is used. So, the aspect ratio basically says whether the cache looks like this or the cache is squarish, because as you can see the aspect ratios are different. So, in a square the aspect ratio will be 1. So, we need to figure out what is the width by height. So, this in itself is a hard thing to figure out, but let us say this is one of the parameters that we will optimize.

So, let us say that let this be the parameters, let us say N spd for the data array, d for the data array sets per row of the data array let that be spd, so let this be a parameter. So, if this is the parameter, say every row of let us say the data array will have N spd sets.

So, if it has $N_{\text{spd}} * A * B * 8$ these many bits. So, it will essentially have these many columns. So, this is clear that N_{spd} is a number of sets per row multiplied with the number of blocks per set multiplied with the number of bytes per block number multiplied with the number of bits per byte. So, then we will have these many columns.

Now, the next question is to figure out the number of rows. So, the number of rows is not hard to find. So, the number of rows so this is number of columns. So, the number of rows is basically equal to the number of sets divided by the number of sets per row. So, we check that the product is correct, all that we have to do is we will have to multiply both and the final result is C into 8, which is C into 8 bits.

So, this is equal to the number of rows. So, this is one of the parameters that our optimizer needs to find out that what is N_{spd} . So, in earlier versions of Cacti this had to be an integer, but now even fractional values are allowed which means that part of a set can be in one row and the other part can be in the next row.

So, this will clearly give us the aspect ratio of the cache that whether it will more rectangular, whether it is squarish it will clearly tell us once. So, this is an unknown, this needs to be found out. So, once let us say that we know this; then what? Then what we will have is that we will have a large array, where we know the number of rows, where we know the number of columns, but the problem with such large arrays is that they are very slow to access. Why are they slow to access? Bit lines are very long word lines are very long.

So, the signal propagation time on the bit and word lines is very high, if there are many rows the decoder needs to be very big, many columns the column MUX DEMUX has to be very big. So, what we can do is, we can partition this big array into several sub arrays. So, what I have done is, that I have partitioned it on the vertical axes basically. So, I have essentially partitioned it.

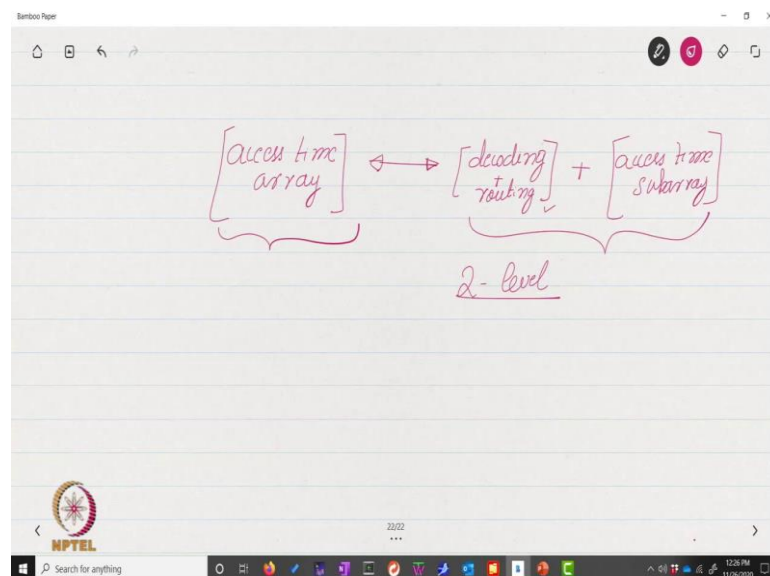
Then again I can partition it this way I can this is like a horizontal partition. So, what I have done is that I have partitioned it into 16 sub arrays. For sub each sub array contains a part of the data, so we can of course, partition it in different ways, but let us see that each sub array. In this case, so let us start with a simple design let us start with a simple two level hierarchy and later on I will break some of these assumptions. So, let us say a simple two level hierarchy has an array and a sub array.

So, let us say that a block is entirely contained in a sub array, then there is absolutely no problem. What we need to do is that we are essentially breaking a big cache into like this 16, this big SRAM array into these 16 mini arrays, so to speak and these 16 mini arrays where each one of them is called as sub array contains a block. So, what we do is, we go there and we read it. So, what is the time that is required? Well, the time that is required is that first we need to figure out in which of these sub arrays the block resides.

See, if let us say if there are 16 of those, so we will need some sort of a circuit to figure out where we need to go. So, this will essentially be a circuit that uses a decoder where will use some of the address bits and then we will find out where we exactly need to go, which specific sub array we need to access.

So, we go to that sub array and then we read in the block from there, and so since this is a smaller sub array smaller than the entire array its access time is expected to be faster. So, let us now draw a simple trade off graph.

(Refer Slide Time: 27:24)



So, on one side we have the access time of the entire array. We need to compare that with let us say the time required for finding out which sub array and then routing the address bits to that. So, this will require a de multiplexer. So, let us say that this is broadly decoding and routing. You need to find out which sub array to go to and send the address bits over there this is this overhead plus the access time of the sub array.

So, this is essentially the trade off that we are looking at. We cannot have too many sub arrays. Like, we cannot have a 100 sub arrays, maybe we cannot have a 1000. Number 1, because this overhead will become too much and number 2 is that now a block will be split across sub arrays, so we need to need, we need to read multiple sub arrays, because each one of them would be containing a part of a block.

So, even if let us see we are reading not a full block, we typically do not we typically do not read 64 bytes. But, let us see even if you are reading 4 bytes or 8 bytes even that needs to come from a single sub array, and if that is not the case then multiple accesses need to be made. So, all of this has to be kept into account. But, pretty much sub arrays do not come for free.

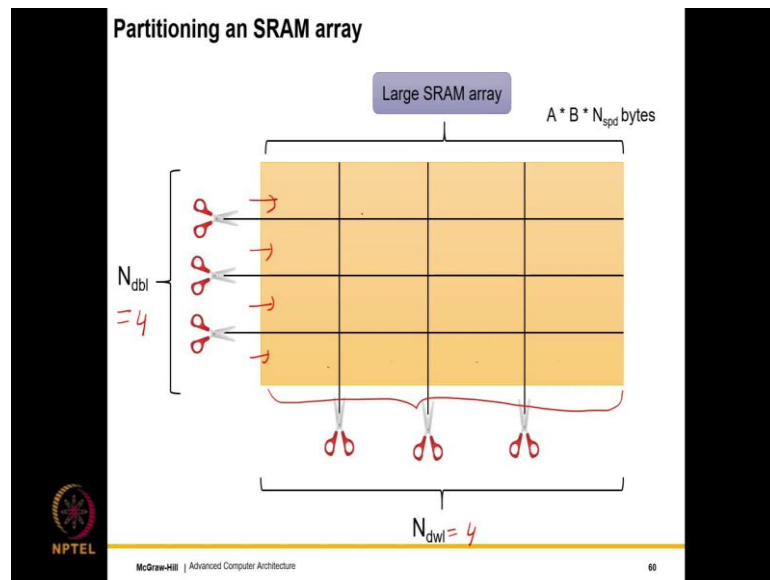
Sub arrays have a decoding and there is a decoding and routing overhead and also the additional wires are required and each sub arrays also have its own logic. So, they do not come for free, but keeping these costs in mind we need to basically see whether we have the original array or whether we split it.

So, a large array is typically split into smaller sub arrays and that would make it a two level hierarchy. So, find the two level hierarchy make sense and this is what Cacti used to do for a long time. Now of course, things have changed and things have moved towards the multilevel hierarchy. So, let me at least describe some slides of the two level hierarchy and then I will come to the multilevel.

So, as I have described, let me just go back to the 2 level part. So, the number of sets that are mapped to a word line in this case is N_{spd} as we said higher N_{spd} there are issues lower N_{spd} there are issues. So, we are basically changing the aspect ratio of the cache. When we create sub arrays, let us say we have N_{dwl} vertical cut lines, vertical partitions and N_{dbl} horizontal partitions. So, total number of sub arrays we create is this much N_{dwl} , dwl is wl is for word line, bl is for bit line, d is for data array.

So, similarly we will have t for tag array like N_{twl} and so on. So, pretty much our optimizer needs to find these three parameters, if you are doing a two level partition. So, it is a 3D three dimensional optimization.

(Refer Slide Time: 30:52)



So, just to tell you when the bit lines are being snapped, we just snap snap snap. So in this case, $N_{dbl} = 4$, because as you can see we make 4 partitions. And, each of these rows it contains $N_{spd} * A * B$ bytes N_{dwl} is where we do snap snap snap on the word lines. So, here again we create 4 partitions; 1 2 3 and 4. So, then we create 4 partitions and we make 16 sub arrays.

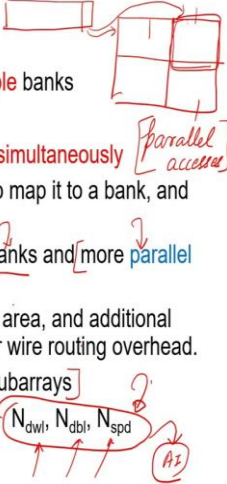
So, in early versions of Cacti each sub array used to have its own decoder its own sense amplifier it. It was a separate kind of a mini cache on its own, but this idea has changed. As we will see, current cache designs do not use this they do not use the two level hierarchy.

(Refer Slide Time: 31:46)

Banks and Subarrays

A large array can be split into **multiple** banks

- A bank is an **independent** array.
- Different banks can be accessed **simultaneously** *[parallel access]*
- Given an **address**, we first need to map it to a bank, and then access that specific bank
- **Advantage** of banking → faster banks and **more parallel access**
- **Disadvantage** of banking → More area, and additional delay in **deciding** the bank. Higher wire routing overhead.
- We split each bank into **multiple subarrays**
- Compute the three parameters: $N_{dwl}, N_{dbl}, N_{spd}$ *[A]*



NPTEL

McGraw-Hill | Advanced Computer Architecture 61

So, current cache designs are smarter. So, what they do is that they use a multilevel hierarchy. So, large cache is split into multiple banks. So, I should not use be using the word cache, to be more accurate, it is much better to say that a large array is split into multiple banks. A bank is pretty much an independent array. So, bank is an independent array. It has its own address and data lines, and of course, its own circuitry.

Here is the; here is the wonderful idea. The idea is that they do allow concurrent access. So basically, one is that because they are smaller they are faster, but that is not the key point. The key point is we can access different banks in parallel. So, what do we do? We take the address from the physical address, we find the; we find the block address from the block address let us say we take the 2 lsb bits, we use them to find the idea of the bank and the address is sent over there.

So, as long as two addresses do not map to the same bank, which is also called a bank conflict, as long as there is no bank conflict, we can support four parallel accesses very easily. So, this is of course, bank conflicts is an issue, but let us say that there is a three-fourth chance of not having a bank conflict.

In this case parallel accesses can be supported rather easily and this is an advantage of the multi bank design, where essentially a large array is just split into four smaller arrays. Of course, there is a disadvantage as we have talked about more area; routing overhead, decoding overhead, there are disadvantages.

Now, what do we do for a bank? So, what did we do? So, large array well we are not happy with that we made banks; banks gave us two advantages at the same time. They are faster, but the bigger thing is that they allow parallel access, so that solved our problem. So, then what we do? So, in this is easy.

So, deciding the number of banks is purely based on how many parallel accesses we would like to support. So, let us say that we can have 8 banks. Well, we just take out 3 bits from the block address and just have 8 banks. So, we have still not designed them. Now, each bank needs to be designed by Cacti and this is what needs to be broken into the sub arrays. So, bank in this case is an array and each of the banks need to be broken into sub arrays.

So, we need to compute these three parameters. And since, all the banks have the same size that is most often the case, if we optimize one bank, we optimize all. Which means, that for each bank we need to compare sorry compute these three parameters. Computing these parameters can be done by brute force in the sense you consider all possible values which is what Cacti does, not a good idea.

So, this is slow right, very slow. What you should do is that you should use a AI based heuristic right. To quickly find out the combination of these three values that gives you the best value of the objective function. The objective function could be minimum time, could be minimum area, minimum power or any combination thereof right. That should be done.

(Refer Slide Time: 35:34)

Arrange the Subarrays in a Hierarchy

Each **bank** can then be split into subbanks

- Subbanks do not allow parallel (concurrent) **access**
- A subbank **stores** a full block (Advantage: read the block in one go)

Each subbank is further divided into **mats** and **subarrays**

- Each mat **supplies** a portion of a block (enhanced parallelism)
- Subarrays in a mat share a decoder or predecoder

The diagram illustrates the hierarchy of subarrays. It starts with an 'Array' represented as a 2x2 grid of 'Bank' units. One 'Bank' is expanded to show it contains a 2x2 grid of 'Subbank' units. One 'Subbank' is further expanded to show it contains a 2x2 grid of 'Mat' units. Finally, one 'Mat' is expanded to show it contains a 2x2 grid of 'Subarray' units. This shows how a single array element is decomposed into smaller subunits for parallel access.

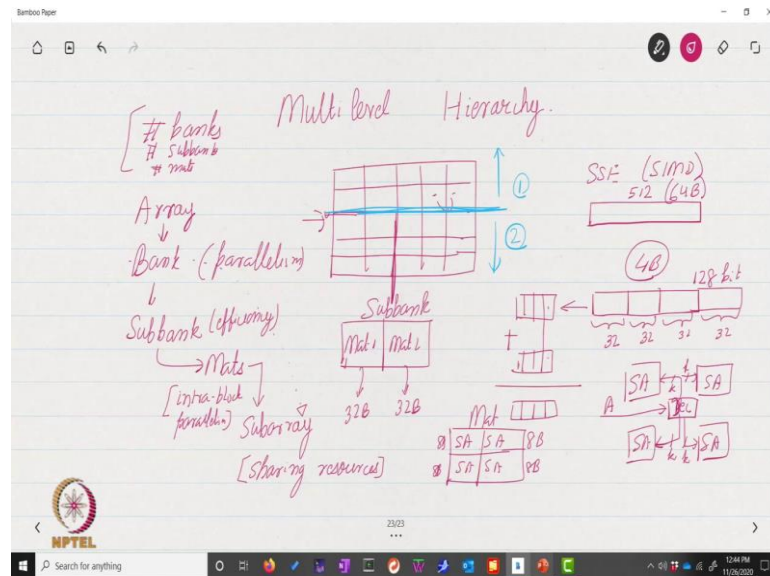
NPTEL

McGraw-Hill | Advanced Computer Architecture

62

Now, what do we do? What we do is, here is the fun part. Let me go back to the bamboo paper software that I use and then I will show you what exactly I am I intend to say in this slide. So, this slide is very loaded.

(Refer Slide Time: 36:01)



So now, we are talking of a multilevel hierarchy. So, we are talking of a single bank and let us say we have computed $N \times N$ single word, $N \times N$ double word. So, we have essentially broken it into a large number of small, small, small, small sub arrays. So, the question is that how do we organize them? One simple organization would be that we just break a bank into a two level hierarchy, a bank and a sub array.

But this is typically not what is done. The main reason being that the routing overhead is high and also modern processors have complex access patterns. So, I will tell you one access pattern which is important and which is why the cache has to be cache design needs to be relooked at.

So, most modern processors inclusive of intel and arm also has these features have what are called SSE instruction sets. Or, broadly speaking SIMD instruction sets, where we read let us say an entire block of 512 bits at the same time. How much is 512 bits? 512 bits just to your math, it is 64 bytes. $(2)^6 * (2)^3$ is $(2)^9 = 512$. So, many do 128 256 and 512 avx instruction sets do 512.

But it does not matter. 256, 512 all of them are large numbers, as far as we are concerned. See, if let us say they are read at the same time, we need to bear this in mind that this is done. So, we do not always read 4 bytes. So, 4 bytes is the case when you are reading an integer, but we let us say even we are talking of 128 bit access which SSE does allow. Even the earlier MMX instruction set also used to allow in that case we actually pack 4 integers.

So, this is a packed instruction format, where we are packing 4 instructions. So, they are kind of placed side by side and we read these 128 bits together in one go. The advantages that we put them into a 128 bit register and we add taken take another 128 bit register and we add them. So, what this does is that this is like a packed addition, in the sense that it adds corresponding integers and the final result is also packed.

So, this is very good for all AI kind of algorithms, matrix multiplication and so on, in the sense that it is kind of a intra instruction parallelism where 1 add instruction adds 4 pairs of numbers in one go. This is also called SIMD; single instruction multiple data paradigm, and SIMD is heavily used in numerical computing that is why most modern processors provide instruction sets for SIMD, hence it is necessary to read large chunks of data in one go from the memory system.

So, if we bear that in mind, what will happen is that we break it into very very small, small, small, small sub arrays. It is possible that let us see if you want to read 64 bytes, it might be split across multiple sub arrays that will cause some problems. So, this is not easy to manage further more from this point of view of power efficiency, we need not have a we need not have a large routing and decoding logic that you go here and then you finally locate which sub array or which set of sub arrays, read them and merge them.

That is a lot of work. We should break this into several more levels. So, the first level that we have seen of course, is the entire array. The entire array did not allow parallel access, so banks allowed. Next what we do is, we divide banks into sub banks. So, within a bank we cannot access different sub banks in parallel, so let us say we do a horizontal split. We change the color if I can, you say can. So, let us let me do a horizontal split over here.

So, let us say the top part is sub bank 1 and the bottom part is sub bank 2. So, then this kind of simplifies the circuit a little bit in the sense immediately from the address we know where to go. Now, within a sub bank we further divide it, so what Cacti does is that Cacti

further divides a sub bank into a into mats. So, what is the advantage? The advantage of dividing a sub bank into mats is like this. Then let us say we divide it into 2 mats.

So, consider a sub bank and let us divide it into 2 mats. What we can do is that let us say, we want to do an SSE access and we want to read 64 bytes in one go. One mat can provide 32 bytes and the second mat can provide 32 bytes and then we can merge them. So, here we are essentially having a parallel access and we are splitting a block between the 2 mats.

So, this is giving us a faster access time because the mats are smaller and also we have parallel access; in a sense, we have these smaller arrays we read 32 bytes from each and we merge them and this is much faster than having a larger array that has 64 bytes stored in one go. And, so basically from an electronics point of view when you read the Elmore delay model.

You will realize that the latency of a wire like a word line or a bit line is proportional to the square of the length of the wire. So, we typically do not want long word lines and bit lines, that is why we would like to have small arrays and where small arrays can provide a part of the data and we can merge it, sounds good.

So, we then unfortunately mats are further divided into our sub arrays. So, you recall that we had originally divided a bank into a sub array, but then we forgot about it, but now the entire picture starts to make sense. So, what did we do let us start again, we take a bank we divide it into many many small sub arrays. We arrange the sub arrays broadly into 2 sub banks or 4 sub banks. So, depends on how we would like to do.

So, Cacti of course, has some hard coded values, but in industry they run extensive simulations. Sub banks are divided into mats where different mats are activated in parallel and they provide a part of the data if there is a need, and then the mats are divided into the sub arrays which is the once that we had originally computed from N_{spd} , N_{dwl} , N_{dbl} .

So, this sub arrays within a mat are also activated in parallel, same idea. So, let us say the mat has to supply 32 bytes, then each of these sub arrays will supply a 4th of it like 8 bytes each ok. Each of them will supply 8 bytes each. So, this is also fast. So, there is one more things of, so then somebody could easily argue what was the need for having mats in the first place, why not have sub bank and a sub array why did you have a mat.

Well, the answer is; the answer is the different sub arrays can actually share resources. So, mats did not share resources, but different sub arrays within the mat, so let us say we have 4. They actually can share resources which will make the design more area efficient. So, what is the resource that they can share? They can share the decoder for example, the row decoder, sorry this is meant to look like a D. So, the row decoder what it can do is it can take the address bits and then the decoded outputs.

So, let us say there are k bit decoded outputs they can be sent to each of these sub arrays within the mat and the sub arrays need not have their own decoders, so we are saving a lot of area over here. So, this sharing of resources and even shearing of other circuits and buffers and so on can be done. So, the sharing of resources is something that is unique to sub arrays within the mat, but not resources are typically not shared between mats in the same sub bank.

So, this is a complicated hierarchy we are looking at, it is not two levels it is 1 2 3 4 5 levels, but different hierarchies play different roles, the reason we broke it into banks is for parallelism. The reason we broke banks into sub banks is for efficiency. The reason we broke sub banks into mats is basically because of intra block parallelism.

Intra block parallelism as well as faster access and the reason we have different sub arrays within a mat is basically for sharing resources notably the decoder. So, that is the reason we broke it. See, if you think about it this is a complex hierarchy and we have a few more optimization terms.

So, these things Cacti 6 hardwares, but need not be hardware. So, one optimization term is the number of banks. So, this would of course, require architectural simulations. The number of sub banks within a bank, number of mats within a sub bank right and the number of sub arrays within the mat, I mean that is not an optimization term because number of sub arrays is known in advance. It is essentially $N_{dwl} * N_{dbl}$.

So, if I were to summarize this entire complicated picture, we have a multilevel hierarchy where every hierarchy is made with different things in mind. And if we go back to our power point, what we see is the same thing sub banks do not allow concurrent access; however, a sub bank stores full block, but then the mats and sub arrays is supply portions of the full block and sub array share resources. You can see an array, a bank, a sub bank, a mat and a sub array.

So, this is this to us is a complicated design of a cache. And as you can see there are many many knobs that need to be turned for effectively and efficiently optimizing a cache, and architectural decisions the pattern of your program etcetera play a very strong role. All of these things have to be taken into account while actually designing a cache. It is not simple and its there are complicated computer science problems in simply doing the optimization.

It is a multilevel optimization space, brute force is not a good idea, it is slow. Many a time it is not possible because of the fact that is so slow. Nevertheless, AI based heuristics work well and they are they have traditionally not been used, but they are increasingly being used in architectural design for solving problems of this kind.

(Refer Slide Time: 48:47)

Summary

- First **set** the aspect ratio N_{spd}
- Then **compute** N_{dwl} and N_{dbl} \rightarrow Total $N_{dwl} * N_{dbl}$ subarrays
- **Divide** the cache into equal-sized banks
- Keep in mind the **probability** of bank conflicts
- A bank is a fully independent array with its separate address and data lines
- **Hierarchy**: [bank \rightarrow subbank \rightarrow mat \rightarrow subarray]
- subbanks **do not allow** independent or concurrent accesses
- Enhanced intra-block parallelism
 - A mat stores a part of a block. It contains multiple subarrays.
 - subarrays share their decoding logic in a mat.

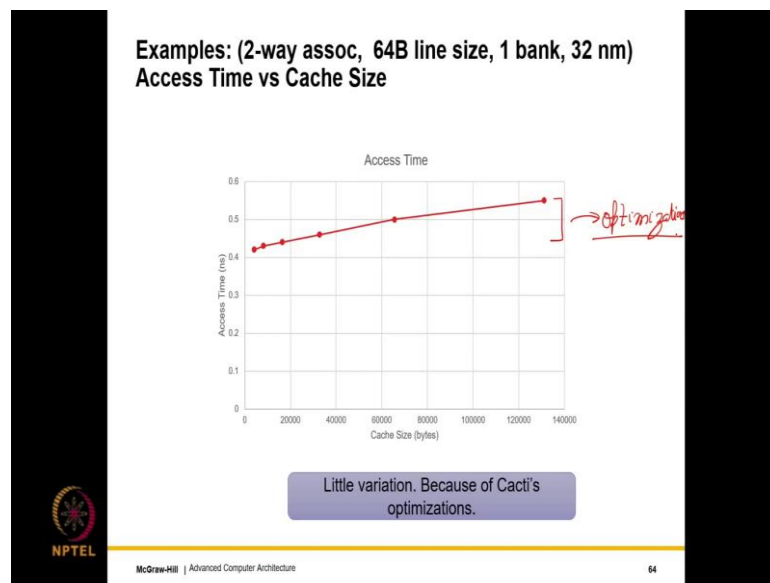
NPTEL
McGraw-Hill | Advanced Computer Architecture
63

The summary of this part, before you move to the Elmore delay part is that, look first decide the number of banks which is parallelism, but for each bank first compute the aspect ratio. Then a N_{dwl} and N_{dbl} and these three parameters can be used. So, the essentially these three parameters are what are optimizer has to give.

So, fine fair enough, and so then dividing a cache in the equal size banks well, we have to keep in mind the probability of bank conflicts, but a bank will at least array the bank will at least allow for parallelism and each bank is split into a sub bank mat and sub array. And, as we have discussed sub banks are separate, they do not allow independent accesses, but they still store an entire block and we have enhanced intra block parallelism because of mats and sub arrays.

And, sub arrays additionally share resources such as decoders. So, given this we need to now go one more level down and figure out that, let us say given a sub array how do we find its timing and power. So, given this how do we find its timing and power. So, finding the power is simpler given a proper model of the circuit, the timing is much harder. So, we will discuss the Elmore delay model. That will teach us how to compute the timing of such complicated circuits.

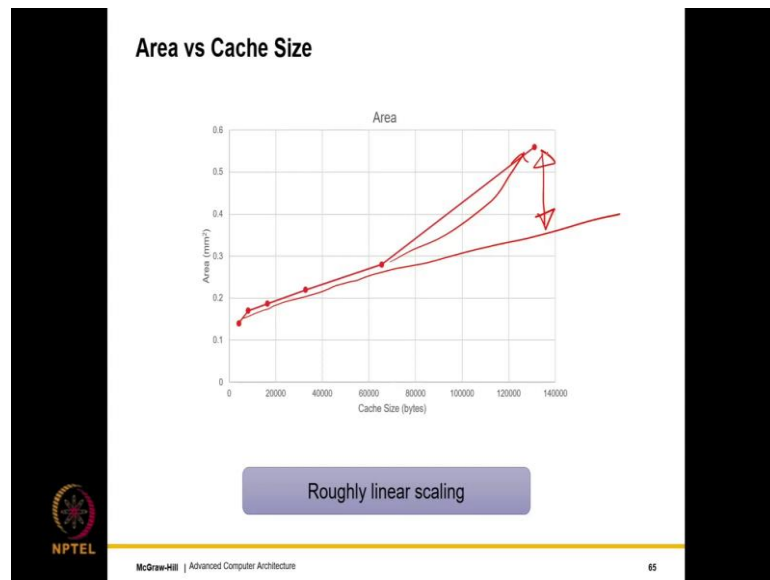
(Refer Slide Time: 50:27)



So, a few experimental results are due. So, let us consider a running example of a two way associative cache, single bank of course, 64 byte line size 32 nanometer technology. So, let us plot the access time versus the cache size where the cache size is varying from a few kilobytes still 128 KB. So, as you can see the access time is not really increasing that much, that is mainly because of our optimizations.

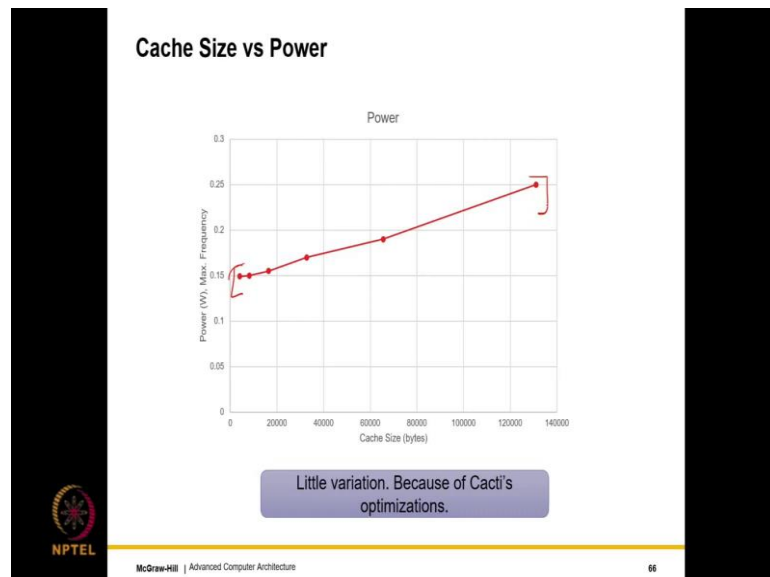
Because of the fact, that we ultimately end up accessing a small sub array, and the sub array is fast that is why even if we increase the size let us say by 64 times, nothing big seems to be happening. the access time is what increasing from 0.4243 nanoseconds to 0.55 nanoseconds and this increases mainly because of the additional wire routing and decoding overhead of finding the sub array and merging data and so on, but this is minimal as compared to what you would otherwise expect and this is only because of the optimizations that Cacti has.

(Refer Slide Time: 51:45)



In comparison, the area does not increase linearly it increases, super linearly as you can see. And super linear increases of course, because as we add more sub arrays we add more wires, more decoders, more routing logic. So, this increases with the size of the cache, so that is why we see the super linear increase in the area.

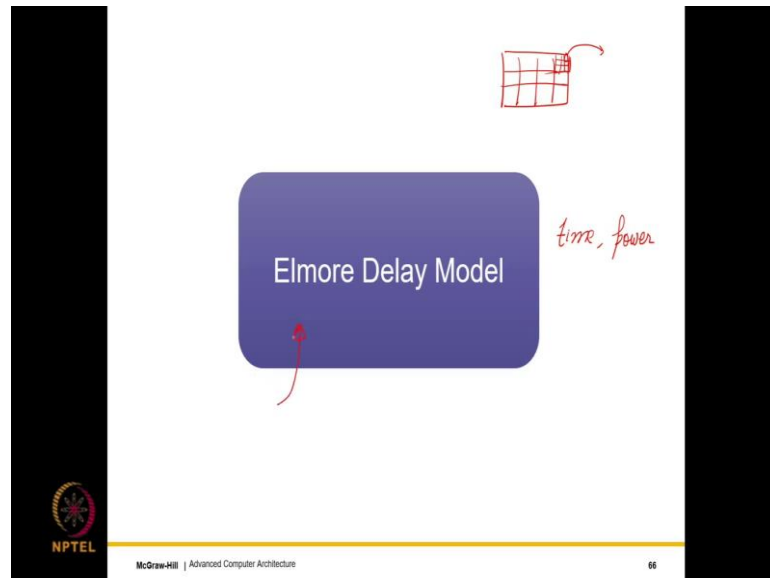
(Refer Slide Time: 52:16)



But we did not see that with the access time and you will also not see that with the power. So, even the power that is used assuming that we have an access every cycle, at the

maximum frequency is still going up from 0.15 to 0.25 watts which is again not much and this is also because of Cacti's optimizations.

(Refer Slide Time: 52:36)



Now, we will come to the final question of cache design, which is that given a sub array. How do we find the time area and power. So, the area is easy to find. The area we know the area of each transistor we can multiply that is easy to find. The power and the timing are the important questions. So, the time and power of a sub array howsoever you get it.

So essentially, we take a large array we then we split split split, and then we take a small one, we further split split split. So, regardless of that let us say, that we come at this particular sub array, what is the time that is required to access it, how do we compute it. So, this will be given to us by the Elmore delay model.

(Refer Slide Time: 53:27)

Time and Power Modelling

- How to compute the delay of a cache?
- Replace it by its equivalent RC model.

RC model of a wire

NPTEL
McGraw-Hill | Advanced Computer Architecture
67

So, what we can do to compute the delay of a cache or for that matter any electronic circuit is that we replace it with a simple model, something that we can actually work with. Because a wire or a transistor this is a very complicated thing to work with. So, we do not really know how to model them quickly. We can model them, with modelling them quickly is the issue.

So, we replace a wire with an equivalent RC model that only consists of resistor and capacitors. So, we have resistors then. So, resistor indicates the resistance of a short sequence of the wire. This is the capacitance, capacitance with respect to ground, the reason being that the wire is a metal conductor and every metal conductor can accumulate some charge, it is like a parallel plate and there are lot of parallel plates in the chip, in the sense there are lot of other wires.

So, whenever there is a potential difference across 2 parallel plates, we essentially have a capacitor and the way that this is modeled is that this is modeled as a finite capacitance to ground which you exactly see over here. Again, we consider a small chunk and we have a capacitance to ground, so on and so forth, so this is the RC model of a wire. Similarly, we can have RC models for a bunch of different elements such as, transistors, sense amplifiers, bit lines, word lines, SRAM cells, everything.

(Refer Slide Time: 55:08)

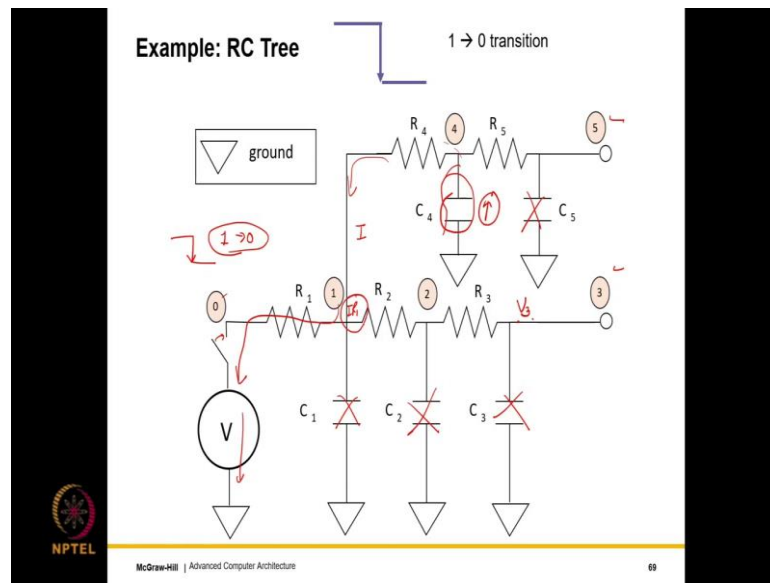
The slide is titled "Assumptions" and contains two numbered points. Point 1 is "Let us only consider RC trees" with a red bracket under "RC trees" and a red circle with a dot above it. Point 2 is "We only consider voltage sources that provide step inputs. 0 → 1 or 1 → 0" with a red bracket under "step inputs" and a red square wave diagram below it. A purple speech bubble with the text "LEARN MORE" points to a grey rounded rectangle containing the text "Elmore Delay Model". The slide also features the NPTEL logo in the bottom left corner, the text "McGraw-Hill | Advanced Computer Architecture" at the bottom center, and the number "68" in the bottom right corner.

So, what we will do is, we will proceed with two assumptions. That in our circuit we will only consider RC trees. So, we all know what is a tree data structure. So, an RC tree is essentially an RC circuit that does not have a cycle, and so that is a number 1. And the second is that we never have a cross path in the sense we have one path other path retching it like that.

So, we do not have this kind of a path. So, we will limit our discussion to RC trees, and a tree is the same as the data structure tree. Furthermore, we will consider only kinds of two only step inputs as voltage sources which means the voltage abruptly goes from 0 to 1 or falls from 1 to 0. So, we will in our basic model not consider a gradual transition.

So, this for example, is not ok for us. So, the Elmore delay model for such kind of a voltage source will help us compute the time that is required. For let us say, charging a bit line or changing the state of an SRAM cell or a CAM cell.

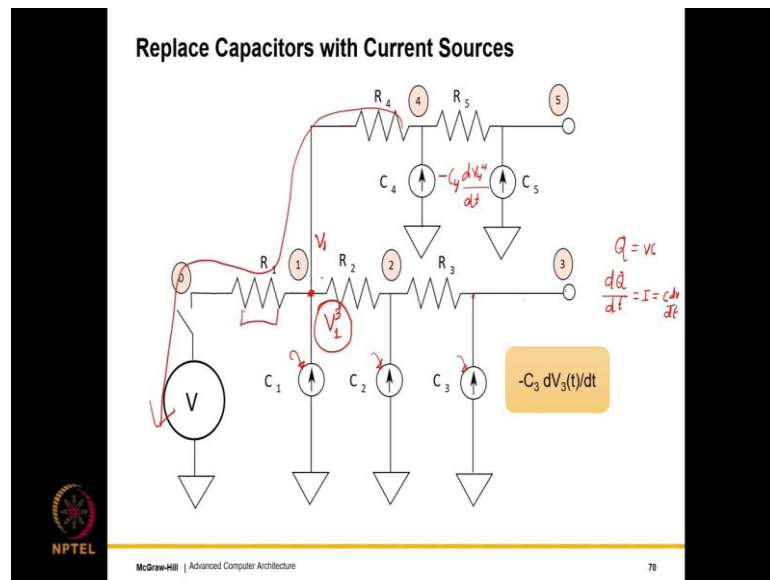
(Refer Slide Time: 56:24)



So, let us look at a typical RC circuit which is also an RC tree by the way. Note the positions of all the resistors and the capacitors and also the terminals 0 1 2 3 4 and 5. So, we have voltage source connected, so we will only consider the 1 to 0 transition in this lecture. The 0 to 1 transition is also easy to handle. So, that is left as an exercise for the viewers, but we will mainly talk about the 1 to 0 transition.

In the 1 to 0 transition, what we need to find is, that let us say, if the voltage source was connected. So, we connect the voltage source and then it suddenly drops from 1 volt to 0 volt, then what would be the voltage at each of these terminals. Particularly, the output terminals 3 and 5, that is what we would like to know. What it would be across time over time.

(Refer Slide Time: 57:29)



So, what we do is that we replace all the capacitors. So, just go back we replace all the capacitors with current sources right. So, these capacitors are all replaced with equivalent current sources. Each current source, if you recall your discussion on capacitors, the current that it would actually provide is $-C_3 \frac{dV_3(t)}{dt}$.

So, why is this? Well, because $Q = VC \frac{dQ}{dt}$ which is the current is essentially $C * \frac{dV}{dt}$. So, the current that C_3 for example, will provide. So, of course, given that it is flowing in this direction, the (-) sign comes down because the voltage is decreasing. So, it is $-C_3 \frac{dV_3(t)}{dt}$.

(Refer Slide Time: 58:29)

Principle of Superposition

- **AIM:** Compute the voltage at node V_x at time t .
- **Methodology:**
 - The aim is to find $V_j^i \rightarrow$ voltage at terminal i because of the current source placed at terminal j
 - While considering one current source, disconnect the rest of the current sources (replace by an open circuit)

$$V_3^4 = -R_1 C_4 \frac{dV_4^4}{dt}$$

NPTEL
McGraw-Hill | Advanced Computer Architecture 71

So, the aim is to compute the voltage of node V_x at time t . What is the methodology? So, what we would like to find is, we would like to find this quantity V_j^i , which is the voltage at terminal i because of the current source placed at terminal j . So, what we want to do, if you go back to this slide you would maybe want to find the voltage at terminal i , which is let us say V_1 because of the current source placed at terminal j which is in this case $j = 3$.

If we add up all the components, so since it is a linear system superposition holds. So, we add up all the components we will then get the voltage at terminal 1 which is V_1 . So, V_1 we will get it over time as a function of time. So, while considering 1 current source let us disconnect the rest of the current sources. So, replace them by an open circuit.

So, this is a quintessential method of analyzing such linear systems. So, let us say I am interested in V_3^4 . So, V_3^4 I will tell you in a second what it is essentially the voltage at terminal three because of the contribution of the current source placed over here, because of the current source placed over here, what is the voltage over here.

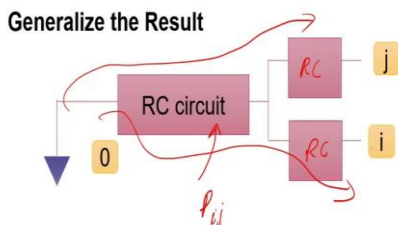
So, what will happen is that since we have disconnected the rest of the current sources, all of them will get disconnected. The current here is going to flow this way, this way, this way and go down to ground. So, if the current is here $= I$ then the voltage will actually $= I R_1$ and since there is no current flow along this route because everything is disconnected, $V_3 = I * R_1$.

So, this is exactly what we see over here, that the current that flows out of the capacitor, that the current source at terminal 4 is given by $-C \frac{dV_4}{dt}$. So, this is basically the current that flows out of this current source over here, which is this capacitance divided by the voltage. So, this voltage is primarily because of this capacitance by dt.

So, this is the current that is flowing. This is the current i and $i * R_1$ is essentially the voltage that will be there at terminal 3, because there is no current flow along this branch. Hence, $V_3 = -R_1 i$, why R_1 because R_1 is the only resistance from this path to the ground. So, that is why we will have $R_1 - R_1$ times the current which is given by this value.

(Refer Slide Time: 1:01:46)

Generalize the Result



- Let P_{ij} denote the RC circuit in the shared path from from $0 \rightarrow i$ and $0 \rightarrow j$

Definition $R_{ij} = \sum_{R \in P_{ij}} R$

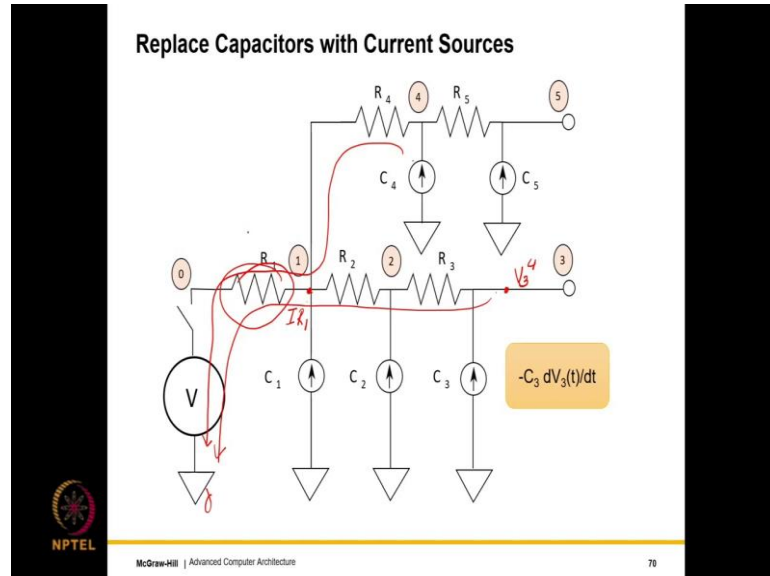
NPTEL
McGraw-Hill | Advanced Computer Architecture 72

So, what we can do is we can generalize the circuit. So, let us say, if you have 2 terminals i and j and then of course, this stands for an RC circuit. And, let us say that we want to find out what will be the effect of a current source placed at j on terminal i to do that, what we do is that we model the RC tree in this fashion where the of course, they will have their separate branches, but at some point they will merge. So, let P_{ij} denote the RC circuit in the shared path from 0 to i and 0 to j .

So, the shared path is actually this path. So, this path is shared. So, let P_{ij} be the shared path from 0 to i and 0 to j . So, then what we can do is we can define a term $R_{ij} = \sum_{R \in P_{ij}} R$. So, sum of all the resistances that are a part of the shared path. Where does this

basic notion of a shared path come from? Well, this will be amply clear if we take a look at this slide that.

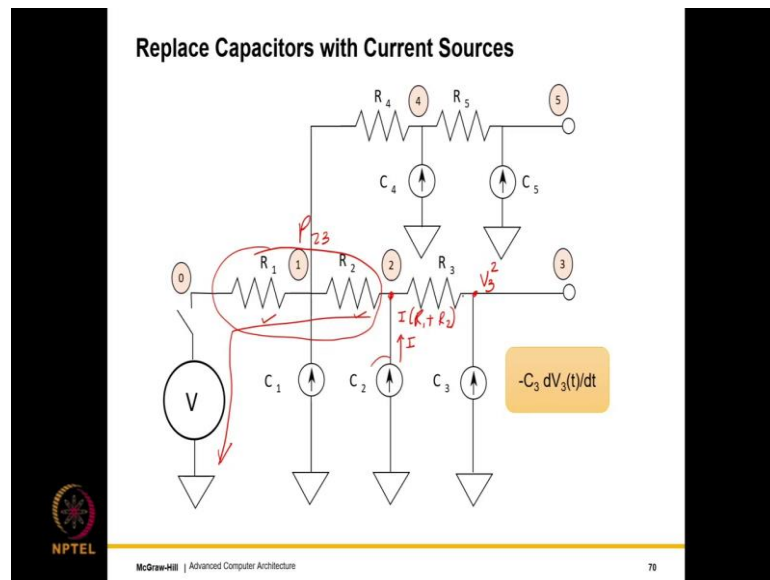
(Refer Slide Time: 1:03:03)



Let us say, we were interested in the voltage over here because of the current source placed over here. So, then we will have one current that is flowing in this direction, because the rest of the currents are disconnected. And, then the voltage over here will be $i * R_1$.

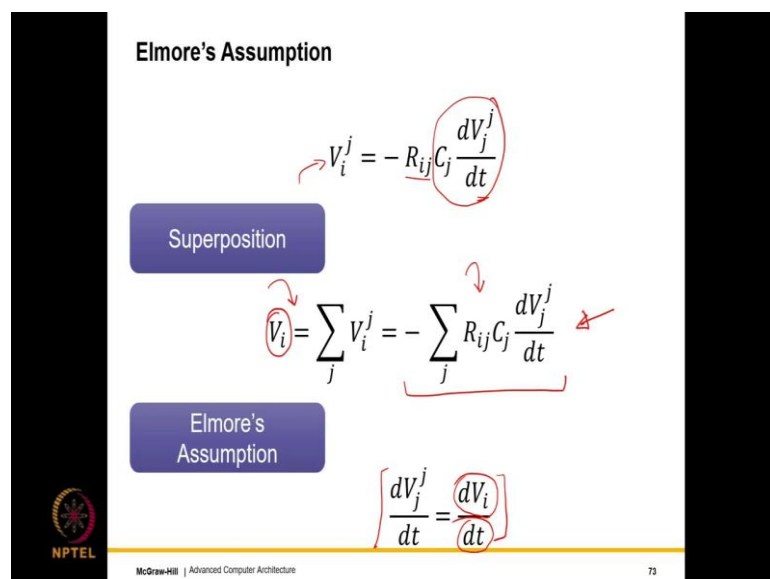
So, that will exactly be the same voltage over here, because there is no current flow along this branch and the important point to bear in mind is that the shared path is only this much. And this shared path in this case, contains only the resistor R_1 . Now, let me consider one more example. If let us say, I want to find the voltage over here.

(Refer Slide Time: 1:03:49)



Because of the current source placed at terminal 2, in this case the shared path will actually comprise these two resistors R_1 and R_2 . So, I will then say that P_{ij} which is P_{23} comprising of these two resistors R_1 and R_2 . So, if I were to multiply that current that is emanating out of your I and the voltage will actually be $I * R_1 + R_2$, which is exactly what that formula with a summation is capturing and the voltage over here will be the same voltage over here, because there is no current flow along this resistor.

(Refer Slide Time: 1:04:38)



So, if I were to incorporate this formula into the next one, what I can say is

$$V_j^i = - R_{ij} C_j \frac{V_j^j}{dt}$$

which is again the sum of resistances on the shared path multiplied with this quantity which we have seen before, this is essentially the amplitude of the current source. Since we have superposition in a linear circuit, we see $V_i = \sum_j V_i^j$ why?

Because it is essentially the sum of each of the current sources sum of the effects of each of the current sources. So, when I add that, I come up with this formula, which is essentially a generalization of this, $-\sum_j R_{ij} C_j \frac{dv_j^j}{dt}$. So, this formula is relatively hard to compute. So, if you take a look at this formula, it is not that easy to compute it and that is why prior to Elmore.

So, incidentally the Elmore delay model is very old, I believe it was proposed in the late 40s or early 50s, but prior to that it was hard to analyze such kind of circuits. So, Elmore made an assumption. So, you will find more justifications of the assumption in the book that why this was made, but Elmore's assumption was basically it's also called the single pole approximation right.

So, the single pole approximation cells that let us approximate $\frac{dv_j^j}{dt} = \frac{dv_i}{dt}$. So, this basically says that the rate of fall of the voltage or rise of the voltage is essentially the same across all the terminals and across all the terminals when we only consider one current source at a time. So, it's essentially the same. If I were to put in this assumption, this approximation, then you will see that this formula will certainly become very well behaved.

(Refer Slide Time: 1:06:44)

Using Elmore's Approximation

$$V_i = \sum_j -R_{ij}C_j \frac{dv_i}{dt} = -\tau_i \frac{dv_i}{dt}$$
$$\tau_i = \sum_j R_{ij}C_j$$

Shared path

$\sum_{R \in R_{ij}} R = R_{ij}$

Solution $\rightarrow V_i = V_0 e^{-\frac{t}{\tau_i}}$

RC tree

The slide features a white background with black text and equations. At the top, it says 'Using Elmore's Approximation'. Below this, a differential equation is shown: $V_i = \sum_j -R_{ij}C_j \frac{dv_i}{dt} = -\tau_i \frac{dv_i}{dt}$. A red arrow points from the summation term to a yellow box containing $\tau_i = \sum_j R_{ij}C_j$. To the right of this box is a handwritten diagram of an RC tree with a path highlighted in red and labeled 'Shared path'. Below the diagram, a handwritten note says $\sum_{R \in R_{ij}} R = R_{ij}$. A large blue arrow labeled 'Solution' points to a box containing the equation $V_i = V_0 e^{-\frac{t}{\tau_i}}$. Below this box, a red arrow points to the text 'RC tree'. The slide includes the NPTEL logo in the bottom left, the text 'McGraw-Hill | Advanced Computer Architecture' in the bottom center, and the number '74' in the bottom right.

This formula will look start looking like this $-R_{ij}C_j \frac{dv_i}{dt}$ which can which is a very simple differential equation. So, what I would do is that the $\sum_j -R_{ij}C_j$, this I will represent by tau i. So, tau i is essentially a time constant. So, what I will do is that R_{ij} is of course, the sum of resistances on the shared path.

So, for each C_j in the circuit, I can compute this the summation of this is like a time constant of the entire system. So, this differential equation is reasonably easy to solve. So, if I solve it and put in the boundary conditions, then the solution will come out to this. Where V_0 was the voltage before this is the before we set it to 0. So, we can put $t = 0$ over here. So, then $V_i = V_0$ at $t = 0$, and if you put $t = \infty$ over here then $V_i = 0$.

So, this is essentially the solution of the RC tree, and the solution of the RC tree does rely on the Elmore's approximation to a large extent. And, Elmore's approximation again uses this formula over here, $\tau_i = \sum_j -R_{ij}C_j$ which comes out from our basic analysis of the shared path in the tree. So, shared path in the tree just to recall, if let us say this is the tree and let us assume that every edge has a resistance and a capacitance to ground.

So, if let us say this was the tree and this is essentially the ground node and let us say this is node j and this is node i. So, R_{ij} will basically be the sum of resistances of the shared path which is this. So, we can say that we move towards the root and we will always have. So, wherever both of them meet after that this is the shared path.

And, if I sum up the resistances in the shared path, if I sum up the resistances over the sheared path then I come across then I come to R_{ij} , and if I sum this up for all the capacitors then I arrive at the time constant of the entire system. So, what did Elmore do? Well, Elmore had nothing to do with caches. So, Elmore at the time of Elmore caches had not been invented. Elmore essentially gave a general approximation to solve for the timing of an RC tree that is all that he did.

And, once you can solve for the timing of an RC tree well then you can use subsequent analysis to find the power that an RC tree would actually consume, and that can be done very easily. But Elmore's main contribution was just to find the time constant of an RC tree.

(Refer Slide Time: 1:10:02)

Results Derived by using the Elmore Delay Model

The latency of a wire is proportional to the square of this length.

The bit line and word line can be modelled very easily.

Full SRAM banks can be modeled as a sequence of simple elements.

NPTEL
McGraw-Hill | Advanced Computer Architecture
75

So, what are the results that can be derived by using the Elmore delay model? So, one of the key important results, so there is an example in the book where this is derived in a great amount of rigor, but essentially the key idea is that if let us say consider a wire, like a bit line or a word line. I was supposed to represent this with a triangle I replace this with its equivalent RC circuit. So, what I can do is that I need to take a look at the time constant which as we have seen over here is summation of $R_{ij}C_j$. So, i can be the end of the wire.

So, $R_{ij}C_j$ will basically be if I consider one capacitor as at a time. Let us say if it is this capacitor it is the sum of these resistances. If it is this capacitor is sum of these resistances.

So, if you work it out we will have a series of the form $1 + 2 + 3$ all the way till n which is an $O(n^2)$ term. So, that is why the most important result is that the latency of a wire is proportional to the square of its length. So, the latency of a wire is proportional to the square of its length and this can be directly derived from the Elmore delay model.

So, the Elmore delay model will essentially replace the wire with an RC tree. We know how to analyze RC trees, if we do that if we consider any capacitor we will just sum up the resistances, and if you number the capacitors from 1 to n then the summation of $R_{ij}C_j$ where C_j of course, is a constant, so it will come out.

The summation of R_{ij} will essentially be the baseline resistance $R * 1 + 2$ because these 2 3 4 5 6 up till n , if we are assuming that we divide a wire into n wire segments. So, this is clearly n times $n + \frac{1}{2}$ we are not interested in the constants it is $O(n^2)$. So, this is bad news for us in a sense we should never have long wires in our circuit, because if we double the length of our wire the latency will increase by 4 times. This result allows us the model long wires like the bit line and the word line very easily.

And, if we couple this with simple models for the decoder for the transistors for the MUX DEMUX and the sense amplifier which again can be modeled in these simple fashion we can model a full SRAM bank, and as the sequence of simple elements. So, we can easily find the time that it will take.

And, also this analysis will help us find the power as well in the sense that once we know the time we will know the individual transitions. So, we can model the power as well and it will be extremely easy to quickly model a large SRAM. So, what Cacti does is that whenever we want to let us say, split an array we need to compute these three quantities right. So, of course, then we will split it and then after a split we can do further splits.

So, we are not limited to one split, but we can we can split it once and then we can split it once more, once more, once more and so on. As we had seen we split a array into a bank a bank into a sub bank, a mat, sub array and so on. So, does not matter it can even be a 10 level hierarchy, but essentially at every level this is what we need to compute. And then.

So, this is just a way of splitting, so we keep on splitting. And, how do we compute these? Well, what we do is that one is that Cacti adopts a brute force approach in the sense it

looks at all combinations of these and then it essentially chooses the best one, but of course AI based techniques can be used like hill climbing and so on to find the best configurations. So, for every configuration we use the Elmore delay model we find the time. If you want the fastest design, we simply choose the best.

(Refer Slide Time: 74:52)

Contents	
1.	Overview of Memory Systems
2.	Modelling and Designing Caches
3.	Advanced Cache Design
4.	Trace Caches
5.	Instruction Prefetching
6.	Data Prefetching

So, we have spent a lot of time and effort in going over memory systems and in how to model and design caches. So, we leave the core electronics part now and we will move to the algorithmic part. So, in the next 4 subsections sections of this book actually, we will discuss advanced aspects of cache design, a new idea called a trace cache. See, it is not that new, but at least it's a very its nontrivial and novel idea which has been adapted in many different settings.

And finally, we will discuss prefetching. So, next we will get into advanced aspects of cache design and discuss the 6 or 7 most common advanced optimizations.