

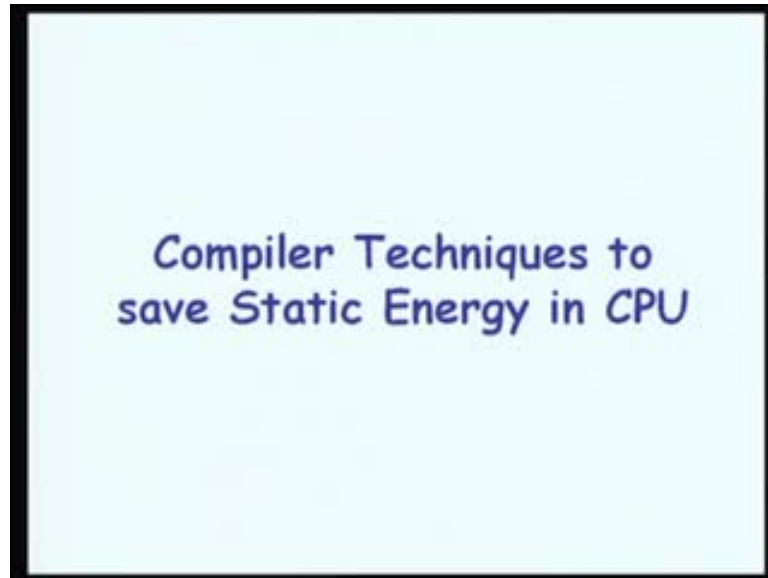
**Compiler Design**  
**Prof. Y. N. Srikant**  
**Department of Computer Science and Automation**  
**Indian Institute of Science, Bangalore**

**Module No. # 17**

**Lecture No. # 35**

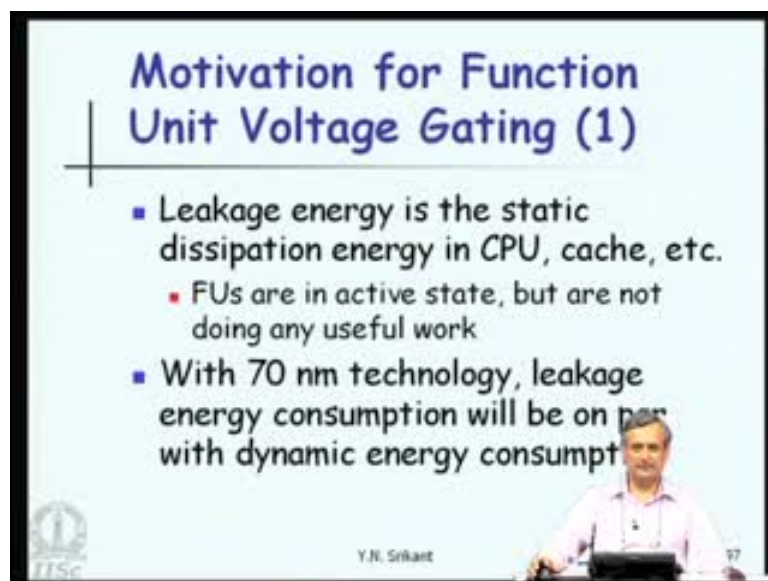
**Energy-Aware Software Systems-Part 4**

(Refer Slide Time: 00:21)



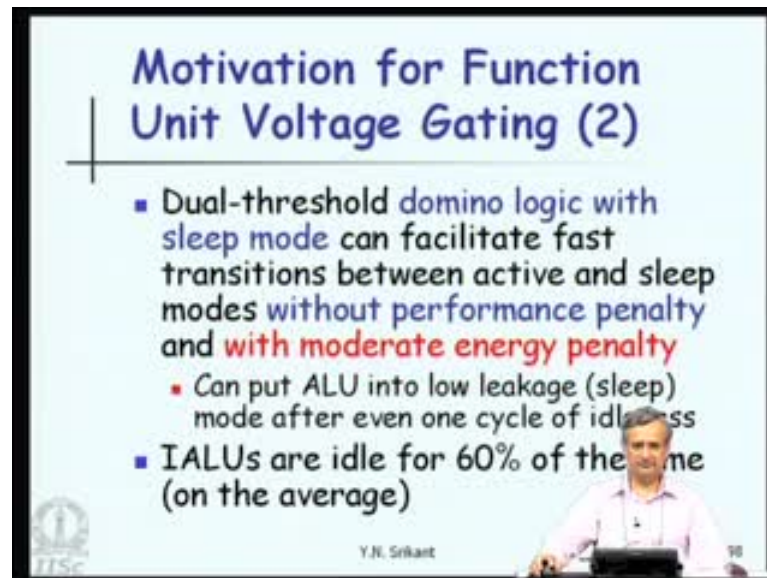
Welcome to part 4 of the lecture on Energy Aware Software Systems. We will now discuss Compiler Techniques to save Static Energy in CPUs.

(Refer Slide Time: 00:32)



So, as I mentioned in the last lecture what we want to do in this compiler technique is to save leakage energy; the function units are going to be switched on and switched off at various times during the running of the program. This is important, because with 70 nanometer technology this leakage energy consumption will be on par with dynamic energy consumption.

(Refer Slide Time: 01:01)



The slide is titled "Motivation for Function Unit Voltage Gating (2)". It contains two main bullet points. The first bullet point states: "Dual-threshold domino logic with sleep mode can facilitate fast transitions between active and sleep modes without performance penalty and with moderate energy penalty". A sub-bullet point under this states: "Can put ALU into low leakage (sleep) mode after even one cycle of idleness". The second main bullet point states: "IALUs are idle for 60% of the time (on the average)". In the bottom right corner of the slide, there is a small inset image of a man in a light blue shirt sitting at a desk with a laptop. The name "Y.N. Srikant" is visible at the bottom center of the slide.

How do we do all these? It is possible to incorporate these dual threshold domino logic with sleep mode circuits, which facilitate transitions between active and sleep modes very quickly without much performance penalty; infact, just 1 second 1 cycle and with very moderate energy penalty. If there is a very sharp transition, then usually there is an energy penalty as well. So, that is the reason for concern. So, integer ALUs are known to be idle for 60 percent of the time on the average; that is the scope for actually saving energy.

(Refer Slide Time: 01:43)

**Motivation for Function Unit Voltage Gating (3)**

- **Pure hardware scheme** (Dropsho et al)
  - has 26% energy overhead over ideal scheme (no overhead)
  - frequent transitions between active and sleep states
- **A software-based scheme** aids the hardware and together they save more energy with little performance loss

Y.N. Srikant 79

Pure hardware schemes are available; that is these dual threshold domino logic circuits, but they have 26 percent energy overhead over ideal schemes; whereas, software based schemes can do much better.

(Refer Slide Time: 02:01)

**Compiler - CPU function unit voltage/clock gating**

- Try to bunch instructions which use the same FUs so that "active" and "idle" periods of FUs are increased
- CPU uses supply voltage/clock gating during idle periods
- Leads to better benefits and saves transition energy

Y.N. Srikant 70

So, we essentially try to bunch instructions which use the same function unit, so that active and idle periods of function units are increased simultaneously. CPU uses supply voltage or clock gating during idle periods. This lead to better benefits and saves transition energy. Now, the question is - how is this performed?

(Refer Slide Time: 02:24)

## Instruction Scheduling

- Reordering instructions
  - To reduce pipeline stalls
  - To exploit instruction level parallelism
- NP-complete (with resource constraints also handled)
- Uses a DAG and is limited to basic blocks
- List scheduling with a ready queue is the most common approach

Y.N. Srikant 71

So, we go back to our good old instruction scheduling which is nothing but reordering instructions to reduce pipeline stalls. This is a technique that we use in order to save energy as well. It was used to exploit instruction level parallelism in our lectures earlier, but now we try to use the same technique in order to save energy. This is possible, because the priority ordering that we use in instruction scheduling could be with energy in mind or performance in mind; so, because of this flexibility, it is possible to use this same technique again to save energy as well. As we know this is an NP complete problem and it uses a directed acyclic graph and is limited to basic blocks. So, list scheduling with a ready queue is the most common approach that is adopted and that is what we use here as well.

(Refer Slide Time: 03:19)

## Clustered VLIW Architectures

Diagram illustrating Clustered VLIW Architectures:

- Left side: Three clusters (Cluster 0, Cluster 1, Cluster 2) connected to an INTER-CLUSTER COMMUNICATION NETWORK.
- Right side: An Individual Cluster showing LOCAL REGISTER FILEs connected to multiple Function Units (FU 0, FU 1, ..., FU n, CFU) which are also connected to the INTER-CLUSTER COMMUNICATION NETWORK.

Legend:  
FU: Function Unit  
CFU: Communication Function Unit

Y.N. Srikant 72

So, let us look at a new type of architecture, which is slightly more energy efficient, so that our techniques can actually work even better on such architectures. These are clustered VLIW architectures. We have clusters of function units connected to an interconnection bus and then within each cluster we have many function units. So, this is the architecture that we are looking at. Now, the question is during instruction scheduling, how we allocate clusters to instructions and function units within clusters to the instruction again. So, these are the important questions that we need to answer. The problem is when we actually look at just the function units they can possibly have a local register file and communicate within that, but when we actually look at clusters, inter cluster communication is not cheap. So, this is a reason for concern. So, we need to allocate cluster and function units very carefully. Within the cluster we need to make sure that the energy is saved when we assign cluster units; I mean function units.

(Refer Slide Time: 04:39)

**Energy-aware instruction scheduling**

An integrated energy-aware instruction scheduling algorithm for clustered VLIW architectures:

- Reduces #transitions between active and sleep states and increases the active/idle periods
- Reduces the total energy consumption of FUs
- Generates a more balanced schedule which helps to reduce the peak power and step power

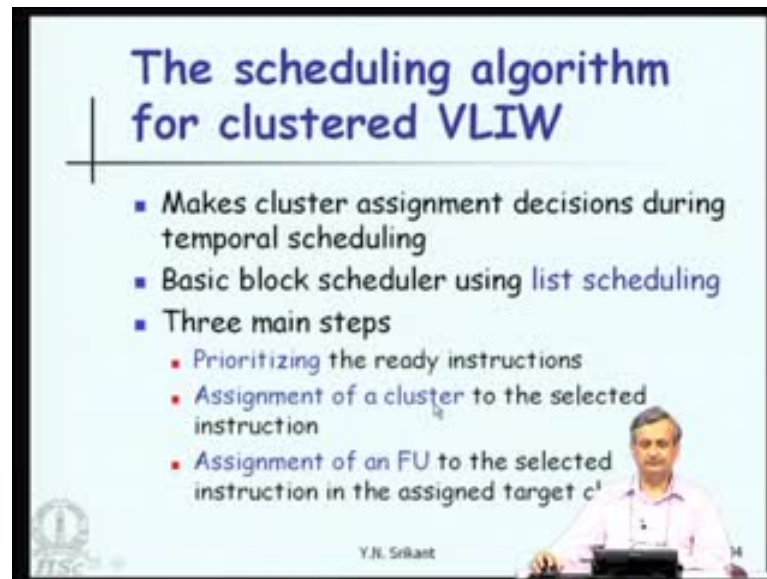
Y.N. Srikant

An integrated energy aware instruction scheduling algorithm for clustered VLIW architectures is what is proposed. So, it reduces the number of transitions between active and sleep states and increases the active idle periods as I mentioned before. It reduces the total energy consumption of the function units; and generates a more balanced schedule which helps to reduce peak power and step power as well. How is this last point taken care of? The point is if the function units which are busy are kept busier for a longer duration, then there is no need to actually make another unit which was sleeping; bring another unit which was sleeping into an active state. So, if we bring something from active to sleep or sleep to active state when there is a power requirement. So, since we



are going to continue using the same function unit, it reduces the peak and step power requirements as well.

(Refer Slide Time: 05:45)



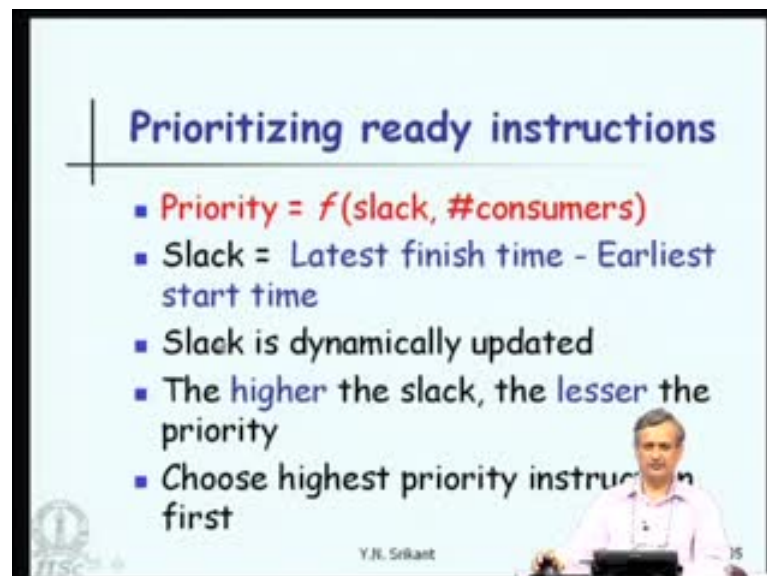
**The scheduling algorithm for clustered VLIW**

- Makes cluster assignment decisions during temporal scheduling
- Basic block scheduler using list scheduling
- Three main steps
  - Prioritizing the ready instructions
  - Assignment of a cluster to the selected instruction
  - Assignment of an FU to the selected instruction in the assigned target cluster

Y.N. Srikant

What is the scheduling algorithm? It makes the cluster assignment decisions during the temporal scheduling that is the cycle by cycle scheduling. Basic block scheduler is used and it uses the list scheduling algorithm. There are three main steps in the algorithm: the first one is prioritizing the ready instructions. This was done even for parallelism extraction the exploiting parallelism. Assignment of a cluster to the selected instruction and assignment of a function unit to the selected instruction in the assigned target cluster. So, this is important that we do it in this particular order as we will see very soon.

(Refer Slide Time: 06:35)



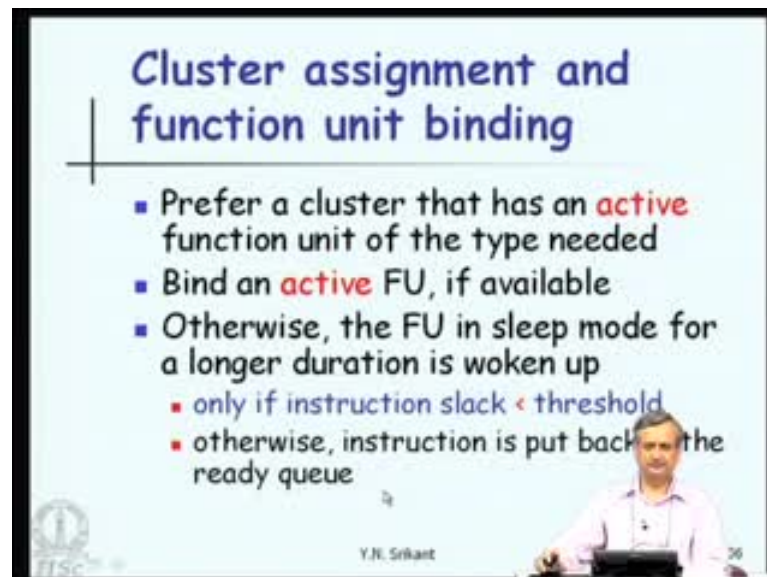
**Prioritizing ready instructions**

- $Priority = f(\text{slack}, \#consumers)$
- Slack = Latest finish time - Earliest start time
- Slack is dynamically updated
- The higher the slack, the lesser the priority
- Choose highest priority instruction first

Y.N. Srikant

What is priority? Priority is a function of the slack and the number of consumers of that particular instruction. So, if an instruction produces let us say, some data say an add instruction produces the sum. That sum will be used by many instructions so these are the consumers. The slack between the time of production and the time of consumption is exploited in this prioritization scheme. The slack is latest finish time and earliest start time, which is the difference between these two and slack is dynamically updated as we go on. Why is this necessary? It is possible that instructions are not scheduled at the same time slot. They could be in brought either closer or pushed to an earlier slot. So, because of this, the slack keeps changing during the instruction scheduling, when instruction scheduling is happening. The higher the slack, the lesser the priority; that is because it is possible to place an instruction with a very high slack in many slots; so, that is why the priority is low. If the slack is very low then, the number of slots available to place the instruction is small and therefore, it has higher priority. So, choose the highest instruction highest priority instruction first; then try to schedule it using the list scheduler.

(Refer Slide Time: 08:16)



The slide is titled "Cluster assignment and function unit binding" in blue text. It contains a bulleted list of four items:

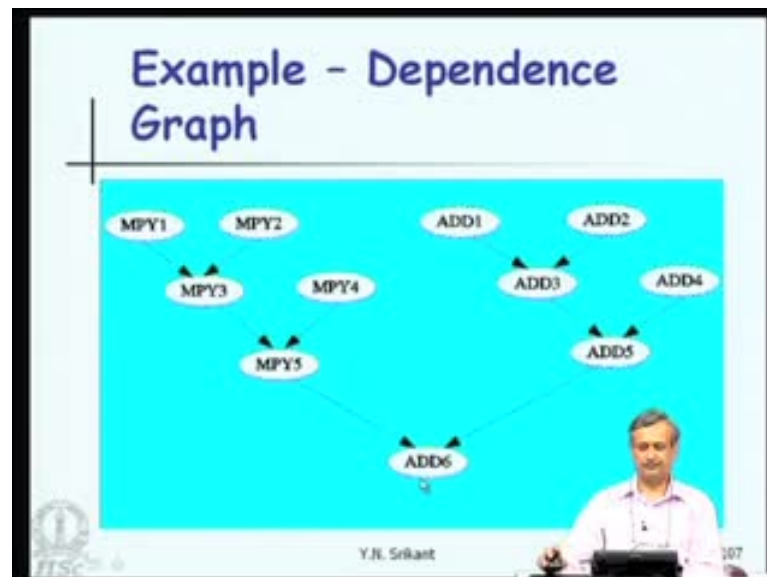
- Prefer a cluster that has an **active** function unit of the type needed
- Bind an **active** FU, if available
- Otherwise, the FU in sleep mode for a longer duration is woken up
  - only if instruction slack < threshold
  - otherwise, instruction is put back in the ready queue

In the bottom right corner of the slide, there is a small inset image of a man in a white shirt sitting at a desk. The name "Y.N. Srikant" and the number "26" are visible at the bottom of the slide.

What is the cluster assignment and function unit binding stage does? What is it that it does? So, we prefer a cluster that has an active function unit of the type that we require for the instruction. Why? The reason is; let us go back to what we wanted to do. We wanted to save energy. We wanted to keep a function unit which was active in the same active state as long as possible. That is why; we prefer a cluster that has an active function unit of the type that we require. So, bind an active function unit if available.

Once we assign a cluster; if none of the clusters have an active function unit of the type we need and then anyone of them would do, but otherwise this heuristic is possible. So, in that cluster bind an active function unit if available. So, again then, we want to keep the function unit active as long as possible. If it is not available then, the function unit in sleep mode for a longer duration is woken up. So, this is to make sure that the peak power consumption etcetera is reduced. So, something sleeping for a long time can be used instead of something which has been sleeping for a shorter duration so, only if the instruction slack is less than the threshold. So, we do not really otherwise, the instruction is put back into the ready queue itself. So, this threshold can be used; can be chosen experimentally. If the instruction slack is greater than a particular threshold then, there is no need to hurry and schedule that instruction right away; it can be put back in the ready queue and some other instruction can be chosen for scheduling at this time. This is because, the instructions which are very tightly; which have a tight slack need to be scheduled quickly. So, if there is a lot more time, we might as well schedule it a little later.

(Refer Slide Time: 10:28)



Here is a simple example of a directed acyclic graph actually a tree. We are going to use it is just it suffices to observe that there are multiply instructions and add instructions in this particular tree. So, we are going to schedule this tree and see the effect on different scheme the effect of different schemes on this particular example.



(Refer Slide Time: 10:53)

### Example - Schedules 1 & 2

Schedule 1				Schedule 2			
MPY1/M1	MPY2/M2	ADD1/A1	ADD2/A2	MPY1/M1	MPY2/M2		
MPY4/M1	ADD4/A1	ADD3/A2		MPY4/M1	ADD1/A1		
MPY3/M1	ADD5/A1			MPY3/M1	ADD2/A1		
				ADD3/A1			
MPY5/M1				MPY5/M1	ADD4/A1		
				ADD5/A1			
ADD6/A1				ADD6/A1			

<p><b>Traditional scheduler</b> (performance-oriented)</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th></th> <th>M1</th> <th>M2</th> <th>A1</th> <th>A2</th> <th></th> </tr> </thead> <tbody> <tr> <td>Schedule 1</td> <td>2</td> <td>2</td> <td>4</td> <td>2</td> <td># transitions from low to high and vice-versa</td> </tr> <tr> <td>Schedule 2</td> <td>2</td> <td>2</td> <td>2</td> <td>0</td> <td></td> </tr> </tbody> </table>		M1	M2	A1	A2		Schedule 1	2	2	4	2	# transitions from low to high and vice-versa	Schedule 2	2	2	2	0		<p><b>Energy-efficient scheduler</b></p> <p>Resource usage vectors</p> <p>Schedule 1 (4,3,2,0,1,0,1,0)</p> <p>Schedule 2 (2,2,2,0,1,1,0)</p>
	M1	M2	A1	A2															
Schedule 1	2	2	4	2	# transitions from low to high and vice-versa														
Schedule 2	2	2	2	0															

- #Transitions has reduced (energy savings)
- Cycle to cycle variation in resource usage has reduced
- this reduces step and peak power dissipation

VLW, one cluster, 2 MULs (2 cy latency), 2 ADDs (1 cy)

So, let us see what exactly we are doing here. So, in this example, we have presented 2 schedules: one of them is using a traditional performance oriented scheduler; the other one is using an energy efficient scheduler. In this schedule 1, we have started as many instructions as possible straightaway multiply; multiply 2 add 1 and add 2 and these are running on M 1, M 2, A 1 and A 2 respectively. Then we start multiply 4, add 4, add 3 then, multiply 3, add 5 and in cycle 5 multiply 5 and then finally, add 6 in cycle 7. So, this is the performance oriented scheduler and it requires 8 cycles in order to finish the whole thing. This schedule number 2 is an energy efficient scheduler and all these are on 1 cluster assuming 2 multiply units and 2 add units. So, if you look at this schedule 1 and see how many transitions are being made from low to high or high too low. For schedule 1, this multiplier 1 is being used here, here and here. So, assuming that it was sleeping before the schedule began; in cycle 1, we wake it up so, 1 transition. Then it is used in cycle 1, 2, 3 and then in 4 it is idle, but 1 cycle of idleness does not force it to go to sleep mode, so in this particular cycle 5, it is used again and then it sleeps. So, it goes down to sleep mode. So, 2 transitions for M 1; similarly, 2 transitions for M 2; M 2 is used here and then, it sleeps. A 1 actually has 4 transitions. So, A 1 was asleep to begin with; now, it has woken up. Then it is used in cycle 2; then cycle 3; then cycle 4, 5, 6 it sleeps. So, actually it has its idle so, it goes back to sleep mode; then again wakes up for cycle 7 and then goes to sleep again. So, there are 4 transitions as far as adder 1 is concerned; 2 for adder 2; adder 2 is used only in cycle 2. So, that means, the number of transitions is very high in this particular schedule. So, it is not very energy efficient; you should also observe that there are too many instructions function units being used in cycle 1. So, that

means, the peak power consumption is going to be high in cycle 1. Let us look at schedule 2. It requires the same number of cycles, but we have scheduled only 2 instructions per cycle. So, multiply our and let us look at the number of transitions for schedule 2. So, for multiplier 1, it is only 2 transitions; M 1 is used here and then in 5 and then it sleeps; M 2 has also only 2 transitions. Most importantly, adder 1 is used here, used here, used here and used here, used here and used here again. So, we have not used adder 2 at all. So, there are no transitions as far as adder 2 is concerned.

(Refer Slide Time: 15:14)

### Example - Schedules 3 & 4

Schedule 3		Schedule 4	
Cluster 1	Cluster 2	Cluster 1	Cluster 2
MPY1.ADD1	MPY2.ADD2	MPY1	
MPY4.ADD4		MPY2	
ADD3		MPY4.ADD1	
MPY3.ADD5		MPY3.ADD2	
		ADD3	
MPY5		MPY5.ADD4	
		ADD5	
ADD6		ADD6	

**Traditional scheduler  
(performance-oriented)**

	M1	M2	A1	A2
Schedule 3	2	2	4	2
Schedule 4	2	0	2	0

# transitions from low to high and vice-versa

**Energy-efficient scheduler**

Resource usage vectors

Schedule 3 (4,2,1,2,0,1,0,1)

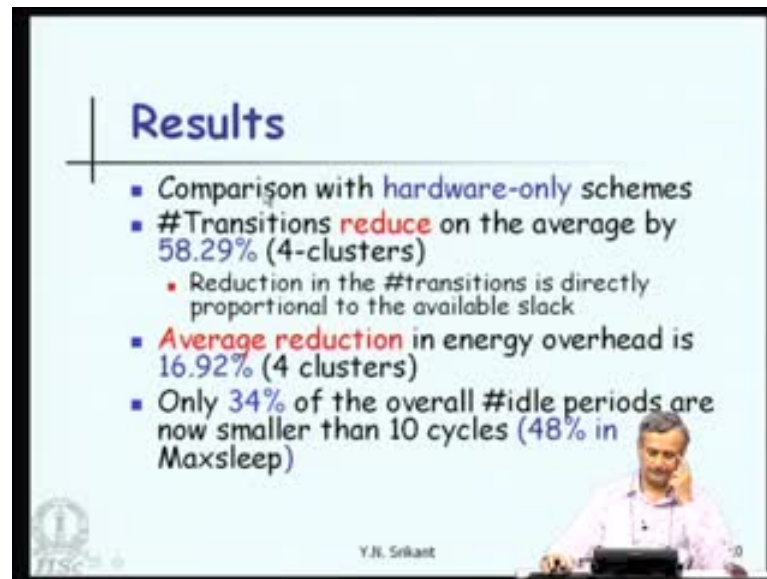
Schedule 4 (1,1,2,2,1,2,1,0)

In schedule 3, ADD3 (MPY3) is scheduled in cycle 3 (4) because adder 1 is needed to transfer the result of ADD2 (MPY2) from cluster 2 to cluster 1

VLIW, two clusters, 1 MUL (2 cy latency), 1 ADD (1 cy lat)

So, now we have tried to keep adder 1 busy as long as possible; and we have tried to keep adder 2 sleep as long as possible in fact, for the whole duration in this case. So, this reduces the number of transitions and therefore, we have saved energy. This is a very simple example, but it goes to show what we intend to do. Similarly, resource usages for schedule 1 and 2; there are 4 resources in cycle 1; then 3, 2 etcetera just count the number of add and multiply units that are active at various points in time. So whereas, schedule 2 is more uniform; it does not have very big changes within the schedule it is very uniform. So, the peak and step over requirements are being minimized as far as possible in schedule 2, which is a side effect of our scheduling strategy. So, the same example using 2 clusters so, I will not elaborate too much on this again. It is enough to show, that the schedule number 4 has lesser number of transitions and has a more uniform schedule compared to schedule number 3.

(Refer Slide Time: 15:36)



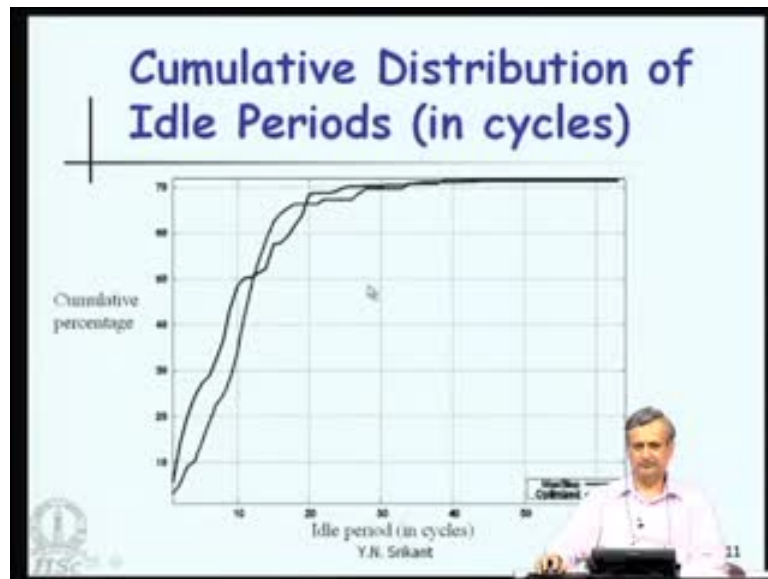
## Results

- Comparison with hardware-only schemes
- #Transitions **reduce** on the average by 58.29% (4-clusters)
  - Reduction in the #transitions is directly proportional to the available slack
- **Average reduction** in energy overhead is 16.92% (4 clusters)
- Only 34% of the overall #idle periods are now smaller than 10 cycles (48% in Maxsleep)

Y.N. Srikant

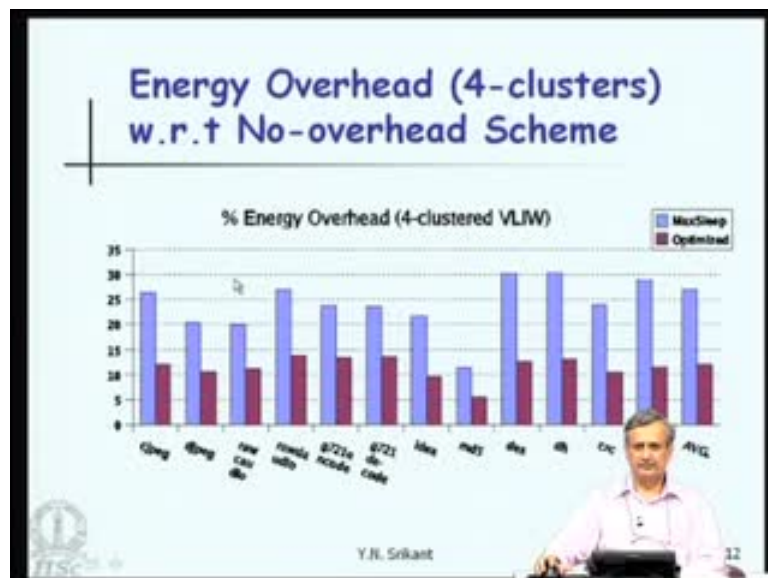
So, let us see what results we got out of this scheduling algorithm. The comparison was with hardware only scheme. So, that means, the architecture had hardware inside to switch on and switch off function units; if they were actually idle for more than 1 unit of time. So, if it is idle for more than 1 unit it switches it off; when it is required it is again switched on; the power supply is switched on. Number of transitions reduce on the average by about 58 percent for 4 cluster systems and the reduction in number of transitions is directly proportional to the available slack. Now, the average reduction in energy overhead; so observe that this is the reduction in energy overhead is about 17 percent for 4 clusters; only 34 percent of the overall number of idle periods are now smaller than 10 cycles; whereas, they were 48 percent in the max sleep or hardware only scheme. Now, what we want to emphasize here is that - since the cycles actually, the idle periods are now much larger; the number of transitions actually is also lesser; so there is a reduction in the transition energy and that is showing here. So, in the reduction by overhead implies that we are comparing against the max sleep scheme that is-the hardware only scheme. So, that is we gain 17 percent over the hardware based scheme.

(Refer Slide Time: 17:27)

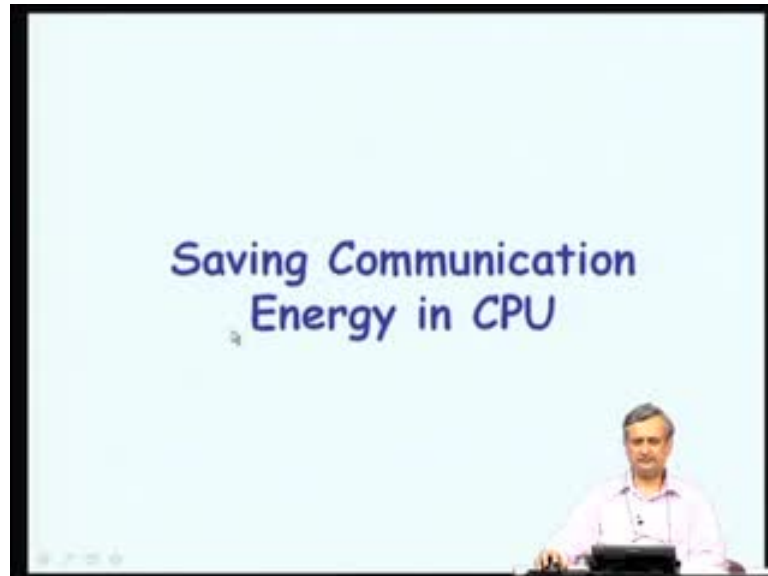


So, here the same idle period reduction size reduction is shown here. So, if you observe at our scheme actually is according to this graph and the older one is according to the hardware scheme is according to this. So, you can see that - for 10 the number of idle periods of about 10 cycles in size is much lesser in our scheme than the other one.

(Refer Slide Time: 17:59)



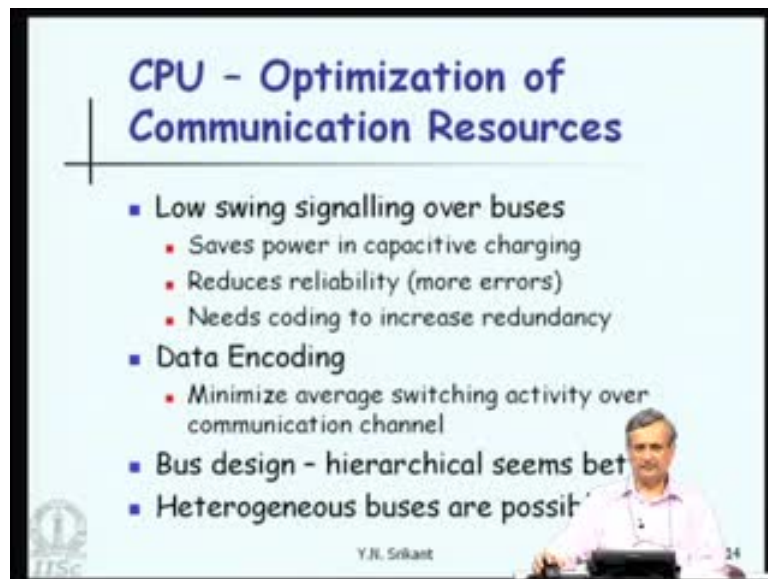
(Refer Slide Time: 18:24)



So, here the results are shown for a large number of benchmarks. You can uniformly see that this violet color which corresponds to the hardware max sleep scheme; the other darker one corresponds to our scheme. So, our scheme has much lesser energy overhead compared to the hardware only scheme. So, that is the summary of the results.

How do we save energy in the communication energy in the CPU? So, that is the next topic.

(Refer Slide Time: 18:36)



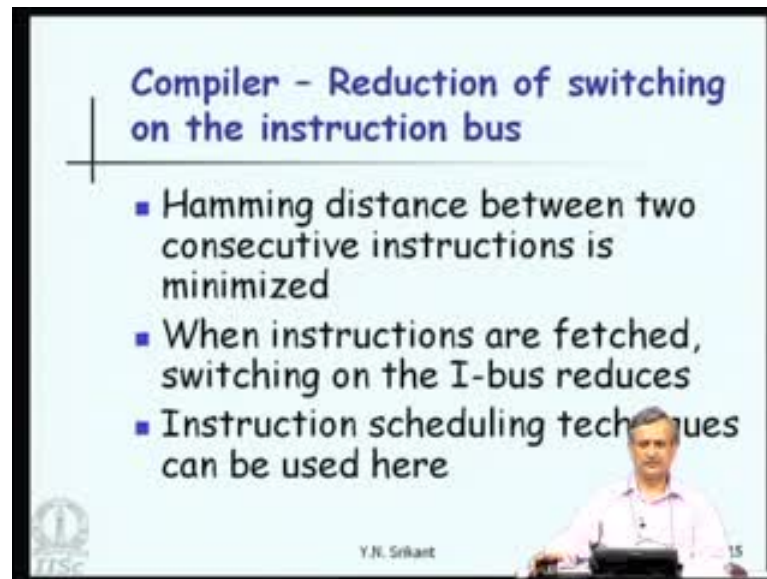
What are the communication possibilities inside a CPU? There are many buses. We want to transfer data; then addresses and all this in CPUs. So, we require buses for all this and



suppose, we want to reduce the voltage over the bus. So, low swing signaling over buses. So that means, if the bus is switching between 1 and 0 anyway, the all transmissions are digital. It may rise to say something like 2 volts or 3 volts or 1 volt depending on the power supply. So, if that voltage and the low can be maybe 0 or negative. So, if this swing is reduced; let us say, instead of 2 volts the high becomes 1 volt; instead of minus 1 actually, it can be reduced to say minus 0.5 or something like that. So, that means, we are reducing the total swing of the voltage over the bus. This saves power definitely, because capacitances now have to charge to a lesser extent. But then there are going to be more errors because reliability is reduced; noise increases when, we reduce the swing therefore, reliability gets reduced and there will be more errors during the transmission. Therefore, we require some coding in order to increase the redundancy in such a scheme. So, this may add to the overhead and may or may not help too much.

It is possible to encode data. So, minimize the average switching activity over the communication channel. Some types of data encoding can actually reduce communication. Say for example, we try to and it is also possible to rearrange instructions in order to reduce switching and so on. So, the bus design hierarchical buses actually, seem to reduce the energy consumption; it is also possible to use heterogeneous buses. What are these? For example, we may have a single bus which is used for the entire CPU. But suppose we have 2 buses: one of the buses is very fast; the other bus is bit slow. The advantage is the slower bus requires less energy consumption; the faster bus actually uses more energy. So, if we have communication which is very urgent, we can send it over the faster bus even though, it requires more energy and if there is communication, which can actually be sent in a relaxed manner; it can be sent over the slower bus which requires less energy.

(Refer Slide Time: 21:49)



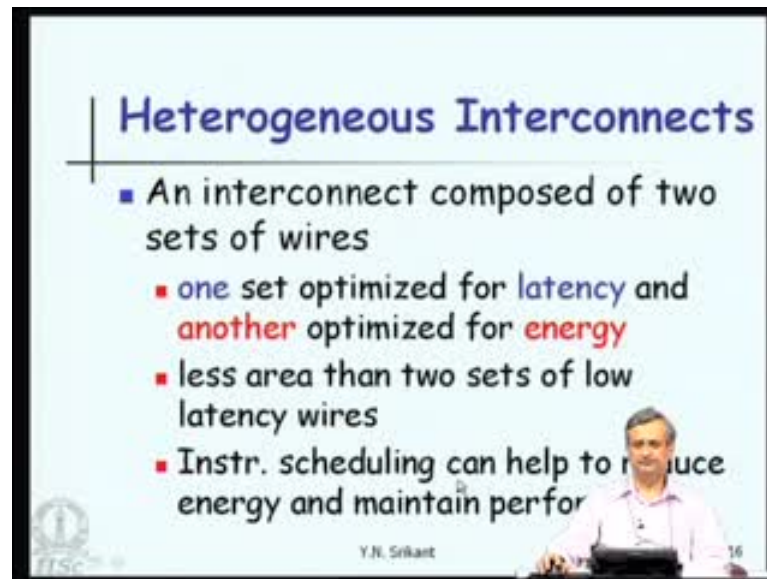
**Compiler - Reduction of switching on the instruction bus**

- Hamming distance between two consecutive instructions is minimized
- When instructions are fetched, switching on the I-bus reduces
- Instruction scheduling techniques can be used here

Y.N. Srikant 15

So, that is the principle behind the heterogeneous operation. We are going to see that a little later. So, reduction of switching on the instruction bus for example, can be achieved by instruction scheduling. So, the hamming distance between consecutive instructions is minimized. So, that means, the number of changes from 1 to 0; 0 to 1 between 2 instructions is minimized. So, if the bit pattern is 1 1 1 0 1; the other one obviously if it is 0 0 0 1 0; then the number of switches from 1 instruction to the other is maximum. Whenever there is a 1 in the first instruction; there is a 0 in the second one. Whereas, if the instructions are 1 1 0 1 and 1 0 0 1; then the number of switches is only between 1 and 0 is only 1; so, the bus actually switches. Number of switches on the bus will also be reduced. So, when the instructions are fetched switching on the instruction bus reduces if we rearrange instructions so, that the hamming distance between 2 consecutive instructions is minimized. So, this can also be actually used as a heuristic to perform instruction scheduling.

(Refer Slide Time: 23:10)



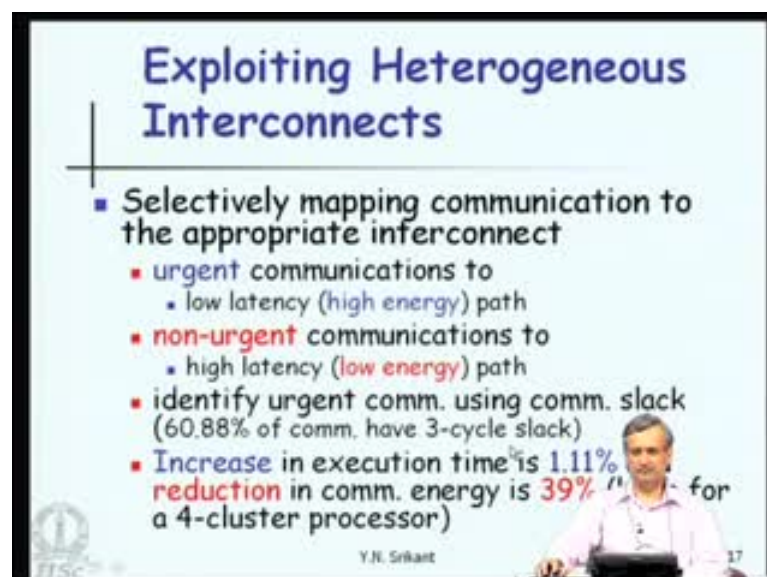
## Heterogeneous Interconnects

- An interconnect composed of two sets of wires
  - one set optimized for latency and another optimized for energy
  - less area than two sets of low latency wires
  - Instr. scheduling can help to reduce energy and maintain performance

Y.N. Srikant 16

So, I mentioned heterogeneous interconnects. Let us look at this a little more in detail. An interconnect composed of 2 sets of wires is heterogeneous; is a heterogeneous interconnect. So, 1 set is optimized for latency; the other set is optimized for energy. So, 1 is very fast; the other one is slow. Now, the both these together require less area than 2 sets of low latency or high speed wires. So, the other important point is even though, one of them is very fast; the other one is slightly slower; the by intelligent instruction scheduling; the performance bit of the program is not going to suffer. So, less area than 2 sets of low latency wires and instruction scheduling can help reduce energy; therefore, thereby maintain performance.

(Refer Slide Time: 24:13)



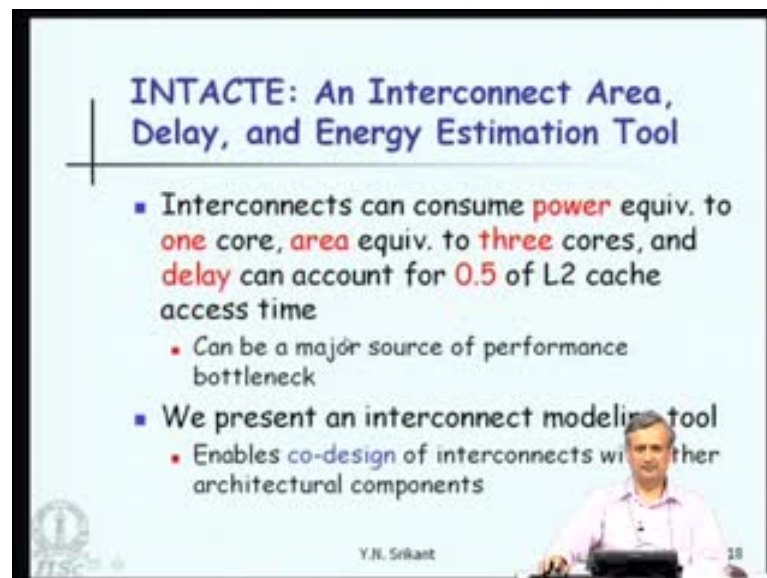
## Exploiting Heterogeneous Interconnects

- Selectively mapping communication to the appropriate interconnect
  - urgent communications to
    - low latency (high energy) path
  - non-urgent communications to
    - high latency (low energy) path
  - identify urgent comm. using comm. slack (60.88% of comm. have 3-cycle slack)
  - Increase in execution time is 1.11% reduction in comm. energy is 39% for a 4-cluster processor

Y.N. Srikant 17

What do we do? How are we going to do this? We selectively map communication to the appropriate interconnect during the instruction scheduling phase. So, urgent communications to the low latency path or high energy path. So, what do you mean by urgent communication, that is, which has less slack. So, that is what we want to measure again. We want to measure the slack and then see how much slack we have. So, those instructions which have less slack will be mapped to the urgent the low latency path; non-urgent communications will be mapped to the high latency path. So, we need to identify urgent communication using communication slack. So, it has been shown that - about 60 to 61 percent of communications have a 3 cycle slack. This can be exploited by our algorithm so, increase in execution time is just about 1.11 percent whereas, reduction in communication energy is about 39 percent for a 4 cluster processor. So, this is a variation of the instruction scheduling that we do. So, slack is used to map communication to either 1 bus or the other bus.

(Refer Slide Time: 25:34)



**INTACTE: An Interconnect Area, Delay, and Energy Estimation Tool**

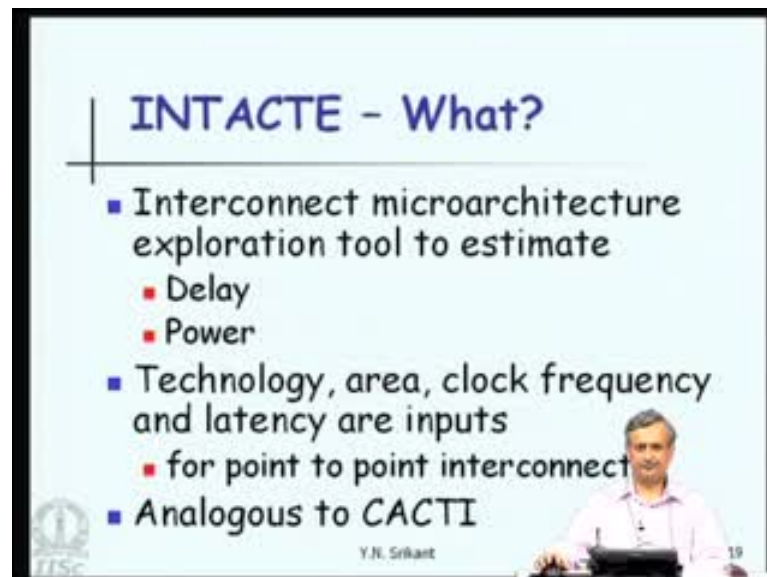
- Interconnects can consume **power** equiv. to **one** core, **area** equiv. to **three** cores, and **delay** can account for **0.5** of L2 cache access time
  - Can be a major source of performance bottleneck
- We present an interconnect modeling tool
  - Enables co-design of interconnects with other architectural components

Y.N. Srikant

Now, we move on to a tool called INTACTE that - we have developed. So, this is an interconnect area delay and energy estimation tool. So, it is well known that interconnects consume power equivalent to 1 core; an area equivalent to 3 cores; delay can account for 0.5 of half of the L 2 cache access time. So, in other words, interconnects can be a major source or performance bottleneck, if they are not designed properly. The problem is when architects design chips; let us say multi core chips and they need an interconnection bus in between inside the multi core chip. As of today, there is no way to assess the amount of energy that is, consumed by the interconnect - it is possible to

assess the amount of energy consumed by a core or a CPU by using simulators, but nothing of this kind is available for interconnect and our tool INTACTE will fill will supposed to fill this gap. So, we present an interconnect modeling tool, which enables co design of interconnects along with architectural components.

(Refer Slide Time: 26:57)



What does it do? What does INTACTE do? INTACTE - it is an interconnect microarchitecture exploration tool to estimate the delay and power of interconnects. The technology that is, 90 nanometer or 65 nanometer etcetera. The area of the chip or the area of the interconnects and the clock frequency and latency are the inputs to our tool. We have designed it for point to point interconnects only. Of course, it is possible to use several interconnects from 1 point to another; in order to have a bigger interconnect. And it is analogous to the cacti tool, which is used for cache you know energy estimation and so on.



(Refer Slide Time: 27:52)

**INTACTE - How?**

- Solves an optimization problem of minimizing power by finding the optimal values for
  - Wire width
  - Wire spacing
  - Repeater size
  - Repeater spacing

Y.N. Srikant 120

How does it work? It solves an optimization problem of minimizing power by finding the optimal values for wire width; wire spacing; repeater size and repeater spacing. So, what are these repeaters? The wires, we understand because interconnects are nothing, but they are supposed to be actually communicating from 1 point to another. So, between the various electronic components there is going to be some wire. So, the repeaters are required to boost the voltage levels in between; because voltage levels may drop on the way due to the resistance in the path.

(Refer Slide Time: 28:34)

**INTACTE - More**

- Additional design variables - can be either constraints or determined by the tool
  - Area
  - Pipelining
- Voltage Scaling Support
  - Tool optimizes power and delay for nominal (Maximum) supply
  - Power and Delay numbers reported
    - for 32 different voltage levels separately from the nominal values

Y.N. Srikant 121

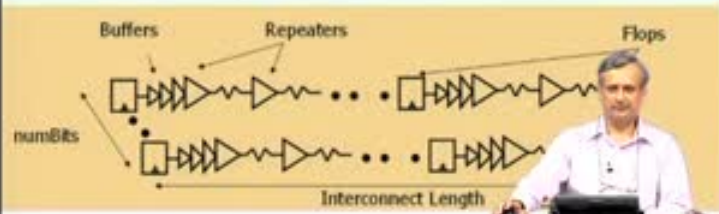
There could be additional design variables so, can be either constraints or they could even be determined by the tool itself; for example, area and the number of stages in the

pipeline. So, these can be either determined by the tool or they can be supplied by the user then, voltage scaling support. So, the tool optimizes power and delay for nominal maximum supply; power and delay numbers are reported for 32 different voltage levels; separated by a small voltage of 15 millivolts from the nominal value. So, in this way, the tool gives us power and delay numbers for various points in the space; various voltages in that we want. And then the architect can decide which one of these is the most suitable. He or she can say even though, I am going to actually use more power. This gives me a faster interconnect; they may also say, I do not mind the interconnect being a little slow, but since the power is most important to me I will choose this particular design. So, various design possibilities are shown and the architect can choose any one of them.

(Refer Slide Time: 30:01)

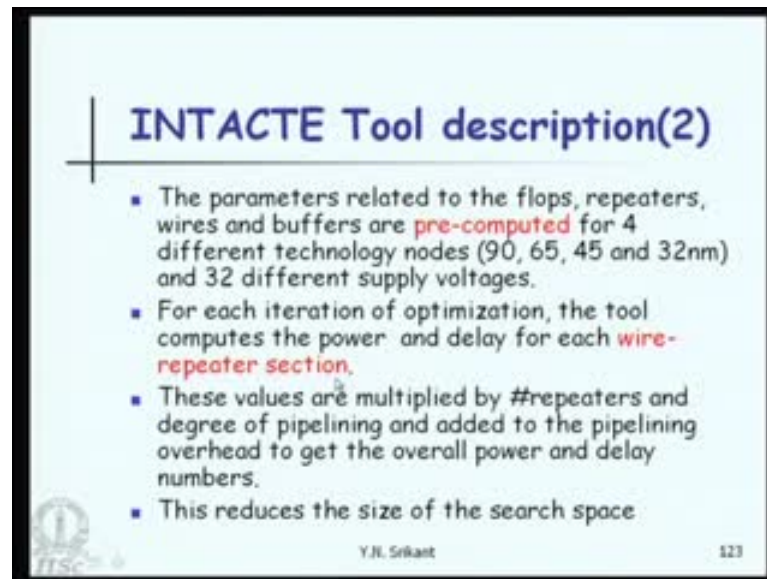
### INTACTE Tool description (1)

- The tool models the interconnect as consisting of a set of identical, equal length pipeline stages
- Each stage starts with a Flop driving a repeater through a set of buffers followed by equally spaced wire-repeater sections.
- All parameters for the model are taken from detailed HSPICE simulations and ITRS



The diagram illustrates the INTACTE tool model of an interconnect. It shows a series of pipeline stages connected in a line. Each stage begins with a square symbol labeled 'Flops'. This is followed by a series of triangles labeled 'Buffers', then a series of triangles labeled 'Repeaters', and finally a series of triangles labeled 'Flops'. The entire sequence is labeled 'Interconnect Length'. A label 'numBits' is positioned to the left of the first stage. A person is visible in the bottom right corner of the slide, sitting at a desk with a laptop.

(Refer Slide Time: 31:10)



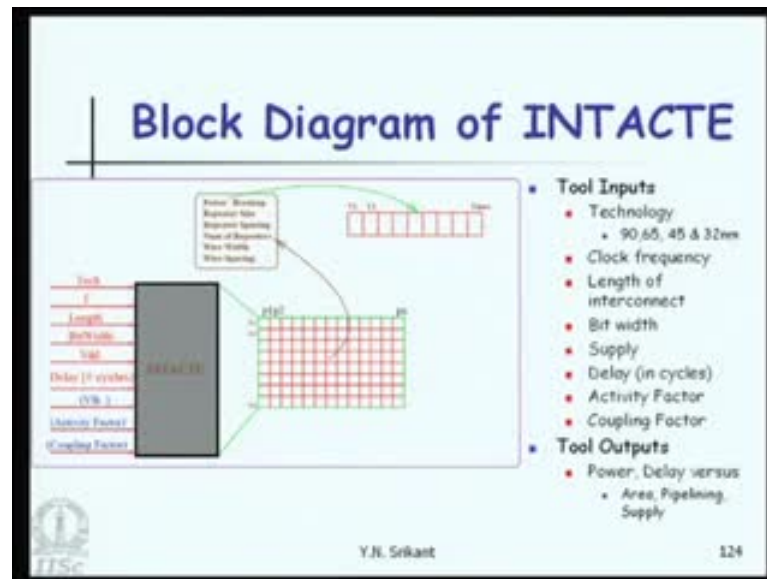
### INTACTE Tool description(2)

- The parameters related to the flops, repeaters, wires and buffers are **pre-computed** for 4 different technology nodes (90, 65, 45 and 32nm) and 32 different supply voltages.
- For each iteration of optimization, the tool computes the power and delay for each **wire-repeater section**.
- These values are multiplied by #repeaters and degree of pipelining and added to the pipelining overhead to get the overall power and delay numbers.
- This reduces the size of the search space

Y.N. Srikant 123

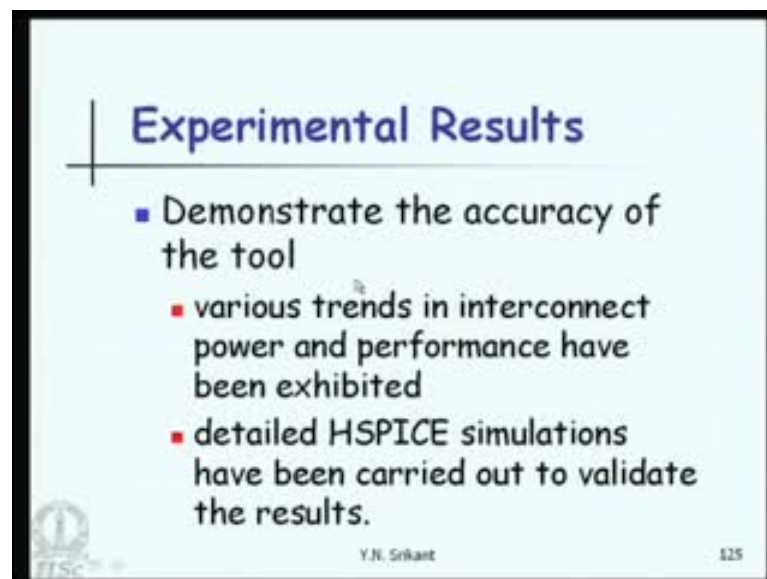
So, this explains the various terms. So, the tool models interconnect as consisting of a set of identical equal length pipeline stages. So, these are the various stages. Here is 1 stage then, there is another third etcetera. Each one of these stages starts with a flip flop. So, here is the flip flop and then driving a repeater so, these are all repeaters. Here this and this are repeaters; through a set of buffers; so, these are the buffers followed by equally spaced wire repeater sections. So, again this entire thing is 1 section; then we would have another section and so on. So, all parameters for the model are taken from detailed H SPICE simulations. So, we simulate each of these in a very detailed device level manner; then use very realistic parameters for these buffers; repeaters; flip flops etcetera. So, the parameters related to the flops, repeaters, wires and buffers are pre computed by simulations for 4 different technology nodes 90, 65, 45 and 32 nanometers and also 32 different supply voltages. So, by doing this computation early before we start optimization we are saving a lot of optimization time. For each iteration of the optimization, the tool computes the power and delay for each wire repeater section. So, these values are multiplied by the number of repeaters and the degree of pipelining and added to the pipelining overhead to get the overall power and delay numbers. So, we are assuming that they are all identical sections. So, it is possible to just take them; add the powers and so on. So, this reduces the size of the search space. So, the limitation is that - you cannot have pipe these sections of different varieties combined with each other. They are all supposed to be identical.

(Refer Slide Time: 32:22)



Here is the block diagram. There are many inputs. And then in this design space, we are going to enumerate various points and the user can pick any one of them.

(Refer Slide Time: 32:34)



What are the results that we can show out of this tool? So, we wanted to demonstrate the accuracy of the tool. So, various trends in interconnect power and performance have been exhibited in this tool. So, detailed HSPICE simulations have been carried out to validate the results. What are those?

(Refer Slide Time: 32:53)

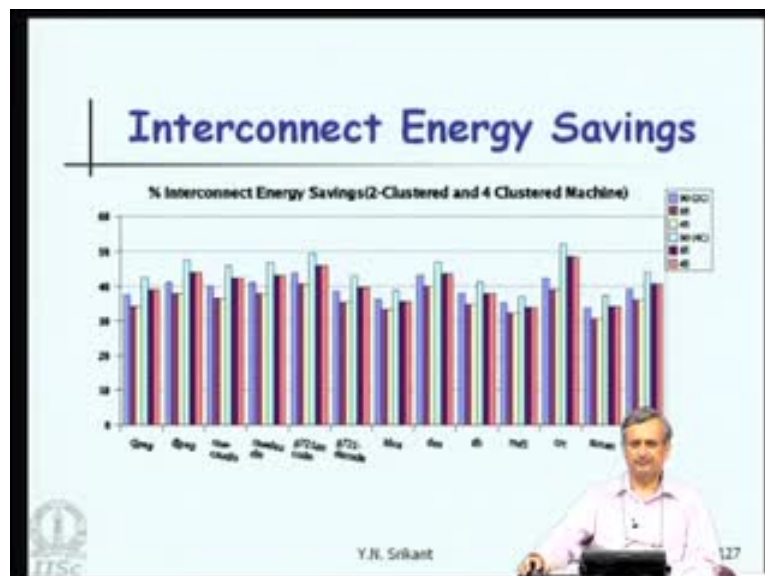
## Architectural Tradeoff Evaluation

- Architectural tradeoffs in having two heterogeneous wires can be evaluated using our tool
- Architect provides length, no. of bits, target technology, operating voltage, and delay estimates
- Tool provides a set of possible interconnect design options to choose from

Y.N. Srikant126

So for example, architectural tradeoffs in having 2 heterogeneous wires can be evaluated using our tool. So, I mentioned instruction scheduling with heterogeneous paths. How do we get realistic values for such paths the interconnects? Our tool provides these realistic values. The architect provides length, number of bits, target technology, operating voltage, and design delay estimates. The tool provides with a set of possible interconnect design options that we can choose from. It also tells us, how much energy and power consumption happens when a particular interconnect is used. So, these figures are used by our instruction scheduler in order to estimate whether, the high speed interconnect or the low speed interconnect should be used at a particular point in time.

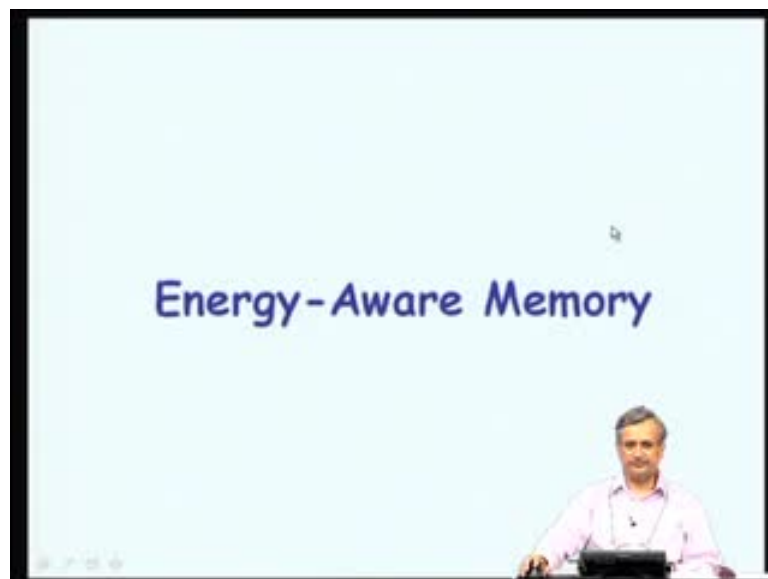
(Refer Slide Time: 32:50)





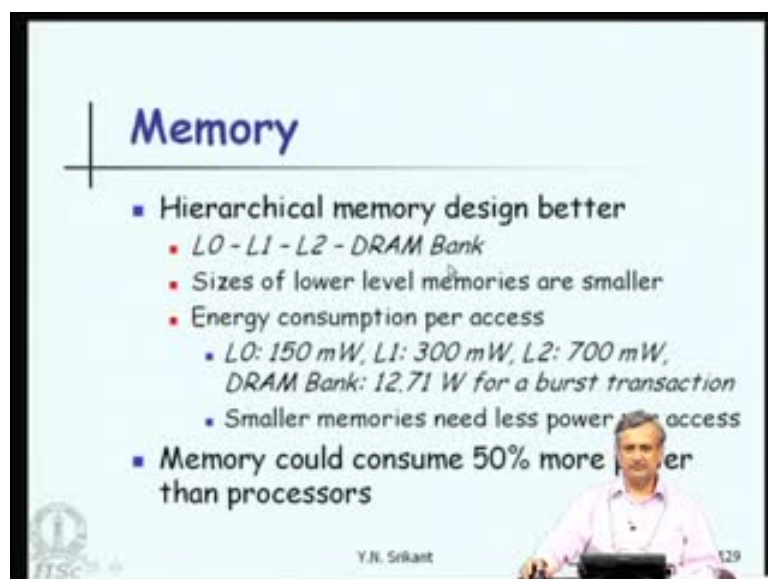
So, here are the results that we really have. So, what we just want to show is that it is effective. For example, interconnect energy savings for 2 clustered and 4 clustered machines are shown here. So, first 3 bars correspond to 2 clustered units; next 3 bars correspond to 4 clustered units. So, for various technologies, such as, 90 nanometer, 65 nanometer and 45 nanometer. This is the saving. So, on the average, we save about 35 percent to 40 percent of energy in the interconnect by using a dual interconnect strategy; rather than a single interconnect strategy.

(Refer Slide Time: 34:37)



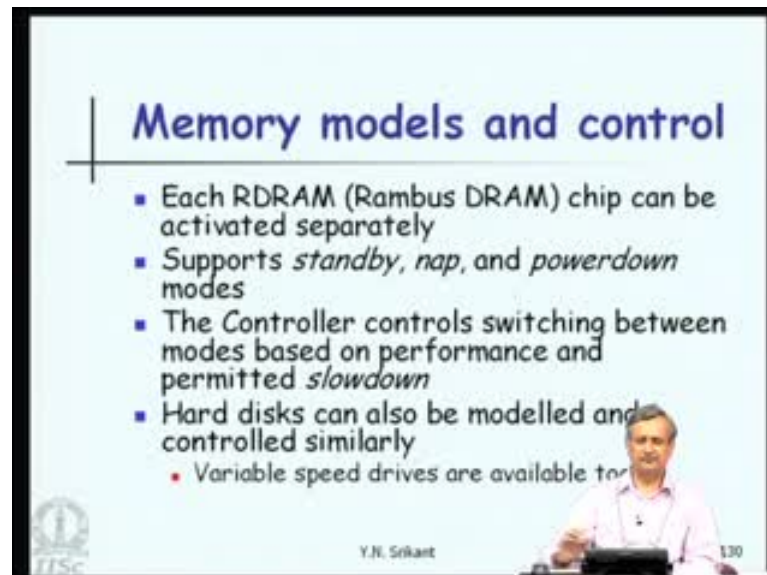
So, let us look at memory now and what exactly is Energy-Aware memory.

(Refer Slide Time: 34:44)



So, we have hierarchical memory designs. So, these are known to be much better than a single memory level. So, there are going to be L 0 caches, L 1 caches, L 2 caches and then the main memory DRAM. So, this is the hierarchy. Instead of having just DRAM; it is better to have such hierarchy. So, L 0 is the fastest and then DRAM is the slowest. So, sizes of lower level memories are smaller. So, L 0 is the smallest cache; L 1 is slightly bigger; L 2 is bigger than that and DRAM is obviously the largest size memory. Energy consumption per access; let us look at some samples to motivate us in saving energy. So, L 0 access is 150 milliwatt, L 1 access is 300 milliwatt, and L 2 is 700 milliwatt for each access. And DRAM bank, it is not possible to access 1 element; it is a burst transaction of say 1 block of memory elements. So, it requires about 12.71 watts for a burst transaction, but we are really transferring a huge block of memory say 256 bytes or words per burst transaction. So, this is the amount of power that is required. Smaller memories need less power and per access. So, that is - the inference that we have drawn; that is very clear. So, memory could consume 50 percent more power than the processor itself. If we have huge memories; nowadays, we have gigabytes; so, memory actually could be a power guzzler. So, there is every reason to believe that we must save energy in the memory banks.

(Refer Slide Time: 36:50)



**Memory models and control**

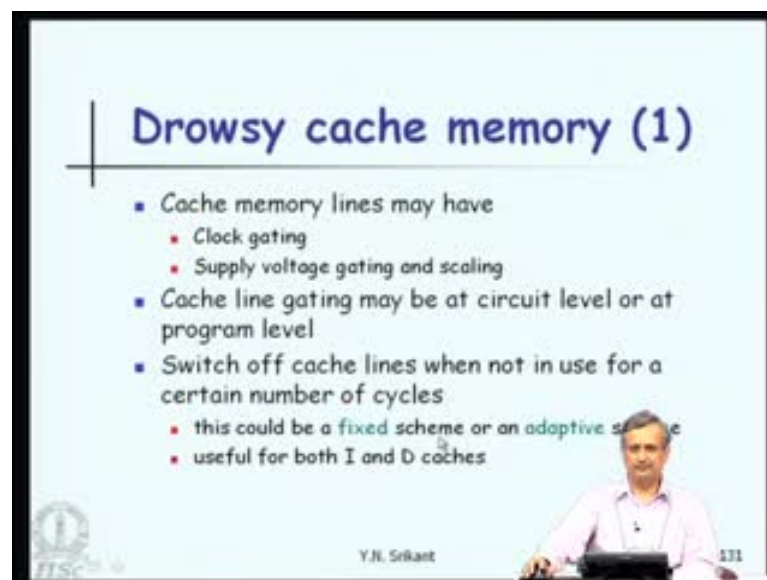
- Each RDRAM (Rambus DRAM) chip can be activated separately
- Supports *standby*, *nap*, and *powerdown* modes
- The Controller controls switching between modes based on performance and permitted *slowdown*
- Hard disks can also be modelled and controlled similarly
  - Variable speed drives are available to

Y.N. Srikant 130

What are the various memory models that are available? For example, each RDRAM Rambus Dynamic RAM chip can be activated separately. So, one can be in sleep state; the other one can be in powerup state; etcetera. Standby, nap and powerdown modes are possible so of course, active state is always there. Then it goes to standby state or nap

state or powerdown state. Powerdown state no power at all; nap state is slightly above powerdown, but deep into napping. So, when we bring it up the data may be lost. Whereas, standby state data is still available and bringing it to active state; requires less energy compared to the nap state or powerdown state. The power control, the controller memory, controller controls switching between the modes based on performance and permitted slowdown. So, the algorithms which go into controller are the ones that we actually need to worry about. The hard disk can also be modeled and controlled similarly. So, variable speed drives are now available. Hard disk can be made to spin at various speeds in order to save energy higher the speed the more energy. So, consumption lesser speed drives require less energy, but they also take rather more time to access the sectors and so on.

(Refer Slide Time: 38:27)



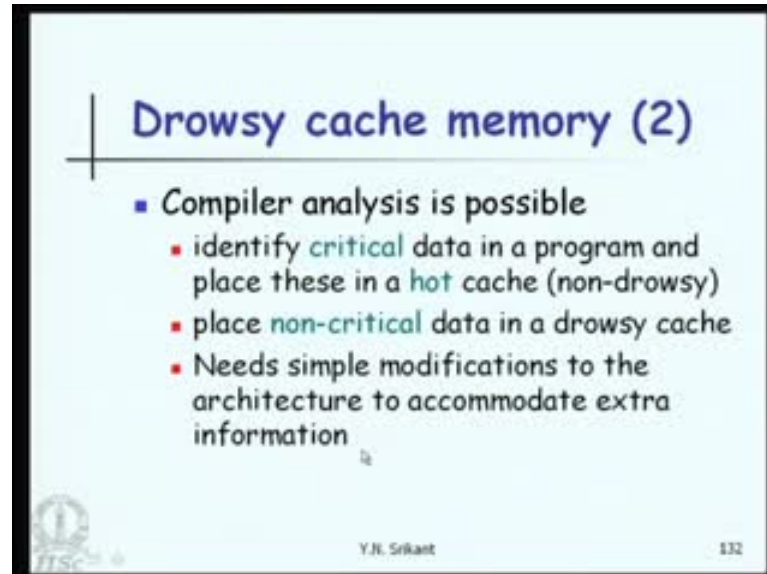
The slide is titled "Drowsy cache memory (1)" and contains a bulleted list of techniques for cache memory management. The list includes: Cache memory lines may have (Clock gating, Supply voltage gating and scaling); Cache line gating may be at circuit level or at program level; and Switch off cache lines when not in use for a certain number of cycles (this could be a fixed scheme or an adaptive scheme, useful for both I and D caches). The slide also features a small image of a man in a white shirt and a logo in the bottom left corner.

- Cache memory lines may have
  - Clock gating
  - Supply voltage gating and scaling
- Cache line gating may be at circuit level or at program level
- Switch off cache lines when not in use for a certain number of cycles
  - this could be a fixed scheme or an adaptive scheme
  - useful for both I and D caches

What about the cache memory? Whatever we studied about or studied so far was main memory. So, cache memory lines may have clock gating, supply voltage gating or even supply voltage scaling. So, in other words, cache lines may be switched off; if they are not in use that is the idea. So, cache line gating may be at circuit level or at the program level. So, it is probably a mixture of both and switch off the cache line when not in use for a certain number of cycles. So, this could be actually a fixed scheme or an adaptive scheme. So, in other words, we just check whether, the cache line has been not in use for a certain number of fixed number of cycles and then switch it off. Otherwise, it is also possible to take an adoptive stand thereby, saying let me not fix the time duration once for all, but it will vary based on the program. It could even vary. So, that is an adaptive

scheme. It could become smaller or larger and it can be used both for instruction and data caches.

(Refer Slide Time: 39:47)



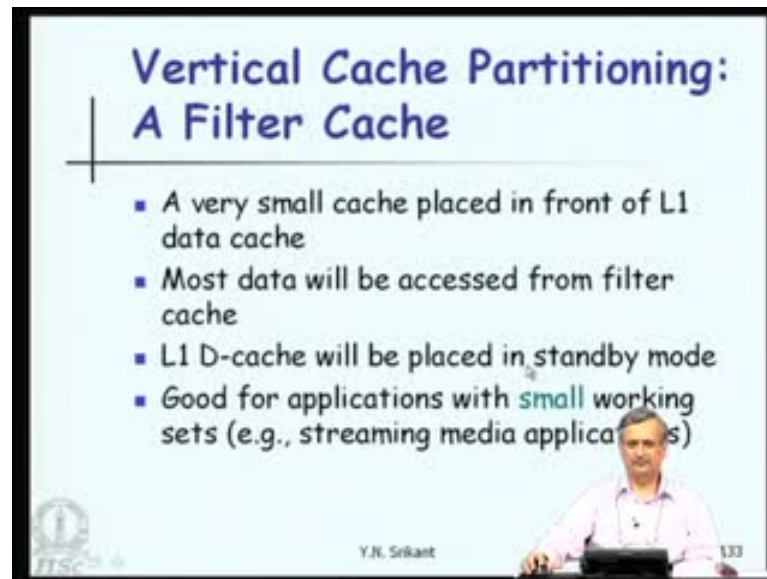
The slide is titled "Drowsy cache memory (2)" and contains the following content:

- **Compiler analysis is possible**
  - identify **critical** data in a program and place these in a **hot** cache (non-drowsy)
  - place **non-critical** data in a drowsy cache
  - Needs simple modifications to the architecture to accommodate extra information

At the bottom of the slide, there is a small logo on the left, the name "Y.N. Srikant" in the center, and the number "132" on the right.

So, the drowsy cache enables compiler analysis. So basically, we try to identify data critical data in a program. Critical data is something that is, required very often and place these critical data in a hot cache; so, that is non-drowsy cache. Whereas, the non-critical data can be accessed slowly; no harm done in that; there will be a lot of slack before that data is actually used. So, we can initiate access of that particular non critical data a little early; so, that it is available in time. So, non-critical data can be placed in drowsy cache and thereby, we are going to actually spend energy; more energy on the non-drowsy cache and less energy on the drowsy cache; this needs simple modification to the architecture; to accommodate extra information that is - necessary to tell the cache controller that my data is either is in the hot cache or in the cold cache. So, here we have found by experimentation that by partitioning the data appropriately into hot and cold critical and non-critical; there is a fair amount of energy say, 20 to 30 percent energy that - can be saved by placing them either in the non-drowsy or in the drowsy cache.

(Refer Slide Time: 41:10)



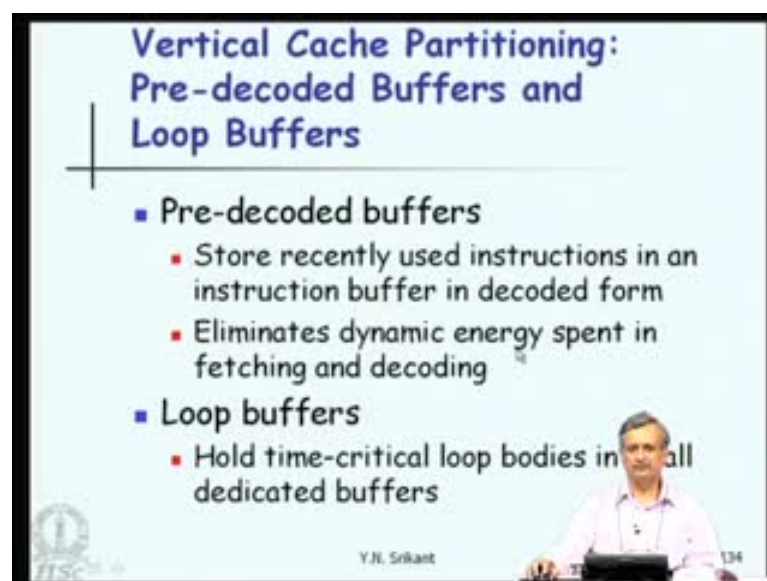
**Vertical Cache Partitioning:  
A Filter Cache**

- A very small cache placed in front of L1 data cache
- Most data will be accessed from filter cache
- L1 D-cache will be placed in standby mode
- Good for applications with small working sets (e.g., streaming media applications)

Y.N. Srikant 133

Then, there is a vertical cache partitioning scheme possible and this is called as a filter cache. A very small cache placed in front of the L 1 data cache is called as filter cache; so, this is even before L 1. So, most data will be accessed from the filter cache that is, the basic idea and L 1 data cache will be placed in standby mode. So, when is being cached accessed L 1 is going to sleep. So, this is good for applications with very small working sets; if the working set is large and all the data does not fit into; let us say, the filter cache then, it does not serve any purpose. So for example, streaming a media applications have such a property. So, filter caches are used in such embedded systems.

(Refer Slide Time: 42:04)



**Vertical Cache Partitioning:  
Pre-decoded Buffers and  
Loop Buffers**

- Pre-decoded buffers
  - Store recently used instructions in an instruction buffer in decoded form
  - Eliminates dynamic energy spent in fetching and decoding
- Loop buffers
  - Hold time-critical loop bodies in all dedicated buffers

Y.N. Srikant 134

So, vertical cache partitioning with pre-decoded buffers and loop buffers. So, pre decoded buffers are what are they. They store recently used instructions in an instruction buffer in decoded form. So, there is no need to spend time in and energy in decoding an instruction they are already decoded. So, this eliminates the dynamic energy in fetching and decoding. So, if the same instructions are used again and again; these pre decoded buffers are very useful they save energy. Loop buffers hold time critical loop bodies in dedicated buffers. So thereby, this eliminates the need to fetch them from memory and so on.

(Refer Slide Time: 42:51)

**Horizontal Cache Partitioning:  
Region-based Cache**

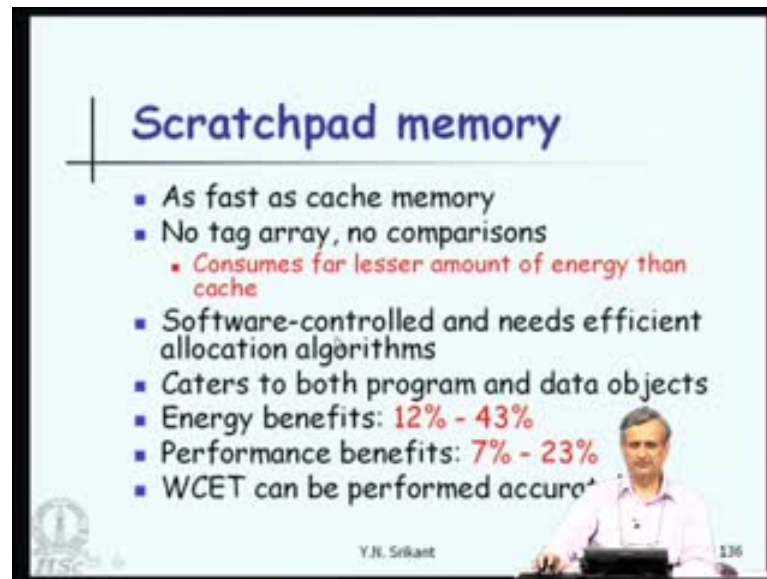
- Two small additional 2 KB L1 D-caches
  - one for stack and one for global data
  - dedicated decoding circuitry detects data access to the appropriate cache
- Substantial gain in dynamic energy consumption for streaming media application with negligible impact on performance

Y.N. Srikant 135

Now, what is horizontal cache partitioning? So, they are called region based caches. Two small additional say, 2 KB L 1 data caches are region based. One of them is meant for the stack variables; the other one for global data variables. So, dedicated decoding circuitry detects data access to the appropriate cache. So, whether if the data is global; it goes to the appropriate region based cache. And there is a substantial gain in dynamic energy consumption for streaming media application with negligible impact on the performance. So, these are all very special caches and they are useful only for specialized applications.



(Refer Slide Time: 43:34)



**Scratchpad memory**

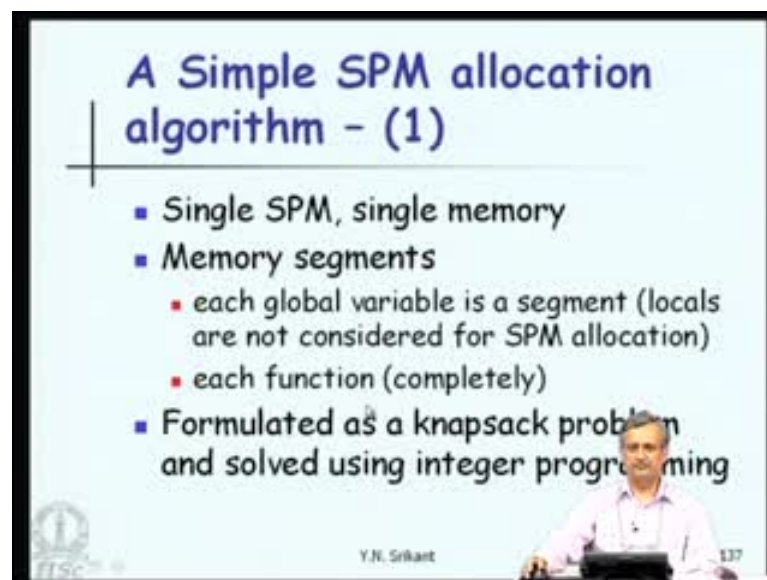
- As fast as cache memory
- No tag array, no comparisons
  - Consumes far lesser amount of energy than cache
- Software-controlled and needs efficient allocation algorithms
- Caters to both program and data objects
- Energy benefits: 12% - 43%
- Performance benefits: 7% - 23%
- WCET can be performed accurately

Y.N. Srikant 136

Now, what exactly is scratchpad memory? So, we know about cache. When we use a cache memory; so, there is actually a reasonable amount of hardware that goes with the cache memory. So, there is a tag array; it is necessary to compare whether, the tag array and find out whether the data is inside the cache or is it inside the main memory itself. So, because of this; when we access a cache line there is a need to spend energy on this comparison with the tag array and so on. So, the extra hardware that comes with the cache actually, makes us lose more at the runtime. So, scratchpad memory is more like a register set. It is very fast; it is as fast as cache memory, but it does not have any tag array. So, there are no comparisons; it consumes lesser amount of energy than cache; that is very obvious, because there is no extra hardware. Then, how are they really different from any registers or anything like that. Well they are not very different; it is just that the register allocation policy, which is used for smaller number of registers is not really useful in the same way here, but the principle is similar. It is actually, a very large set of registers, but we do not have a separate name for each register as we have in the register set. For example, we have names for registers r 1, r 2, r 3, r 4, r 10 etcetera. That is possible, because we may have just 16 or 32 or even 128 number of registers. But if we have a very large number of registers; the scratchpad memory elements say thousands; some kilobytes say 100, 10 kilobytes, 20 kilobytes etcetera. It is not possible to name them and the programmer cannot even think of using these names in the program. So, that is the difference between a cache memory, register set and a scratchpad memory. So, because they are very large software controlled; the scratchpads have to be software controlled and they need efficient allocation algorithms. It is not usually possible for the

programmer to say, I will use this part of scratchpad memory for this purpose and so on. Even though, theoretically it is possible; it is better if the energy usages are all controlled by the compiler. So, it caters to both program and data objects that is - instruction and data both. There are a huge energy benefits: 12 to 43 percent and then there are performance benefits as well 7 to 23 percent. Why? If we had used cache memory then, there would have been cache misses and there would have been a performance penalty. Whereas, with scratchpad memory there is always a hit. There is the compiler already knows what data is in scratchpad and what data is in main memory. So thereby, it can actually generate the right kind of addresses. So, performance improves as a side effect; that too not in a very small amount 7 to 23 percent. There is another advantage. Worst case execution time can be estimation can be performed very accurately. With a cache at a particular memory access is not known to be either a hit or a mess; at the time of compilation. So, the access time is not known. Whereas, with scratchpad if we do not have any cache; we know, where the data is; either in the scratchpad or in main memory. So, the estimation of time; how much time does a program take can be performed very accurately.

(Refer Slide Time: 47:58)



**A Simple SPM allocation algorithm - (1)**

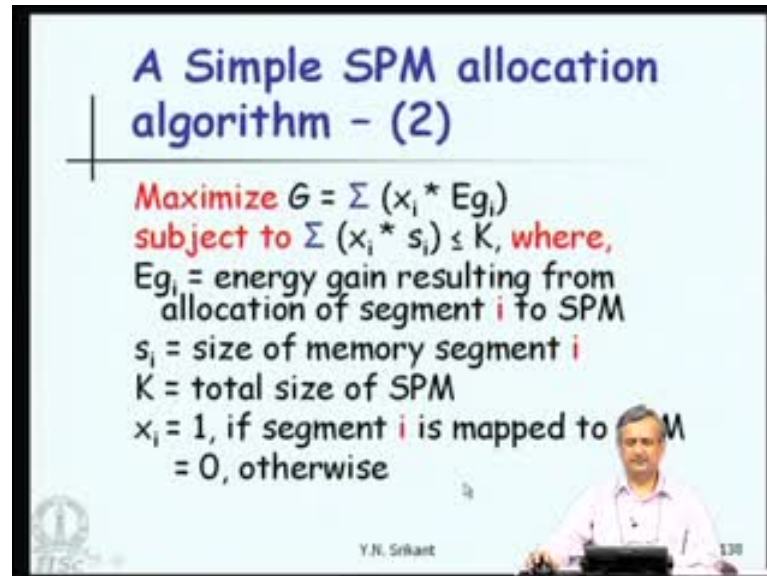
- Single SPM, single memory
- Memory segments
  - each global variable is a segment (locals are not considered for SPM allocation)
  - each function (completely)
- Formulated as a knapsack problem and solved using integer programming

Y.N. Srikant 137

So, let us get a feel for scratchpad memory allocation by looking at a very simple algorithm. Let us assume that - there is a single scratchpad memory and just a single memory only 2. So, memory segments are each global variable is a segment. So, locals are not considered for SPM allocation in our simple scheme. Each function completely the instructions of a function is a memory segment. So, each global variable is a

segment; each segment on its own. So, this whole thing is formulated as a knapsack problem and can be solved using integer programming. So, let us see how the formulation takes place.

(Refer Slide Time: 48:47)



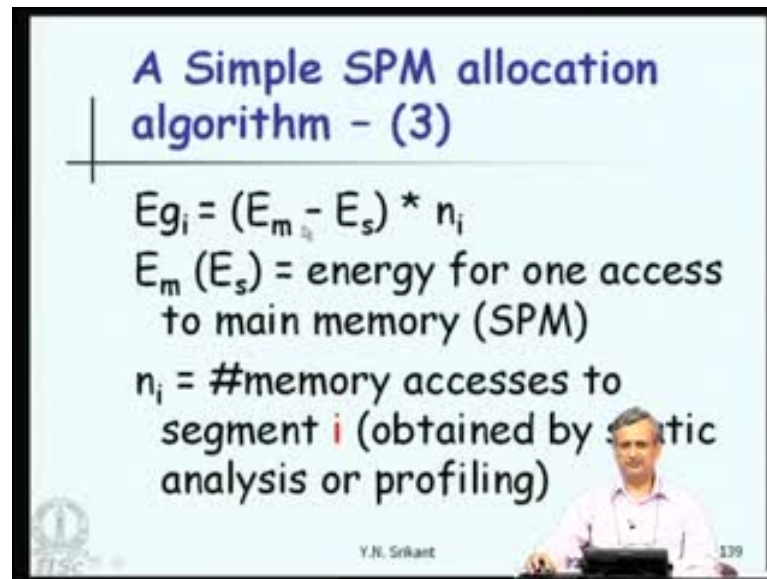
**A Simple SPM allocation algorithm - (2)**

**Maximize**  $G = \sum (x_i * E_{g_i})$   
**subject to**  $\sum (x_i * s_i) \leq K$ , **where,**  
 $E_{g_i}$  = energy gain resulting from allocation of segment  $i$  to SPM  
 $s_i$  = size of memory segment  $i$   
 $K$  = total size of SPM  
 $x_i = 1$ , if segment  $i$  is mapped to SPM  
 $= 0$ , otherwise

Y.N. Srikant 138

We need to maximize the gain; gain in energy. What is that? It is sigma of  $x_i$  star  $E_{g_i}$ . What are the constraints? Constraint is subject to sigma  $x_i$  star  $s_i$  less than or equal to  $k$ . So, let us look at the various symbols now. So,  $E_{g_i}$  is the energy gained resulting from the segment  $i$  to the scratchpad memory;  $s_i$  is the size of the memory segment  $i$ ;  $x_i$  is the optimization variable. So, 1, if segment  $i$  is mapped to the SPM 0, otherwise. So, for those segments, which are mapped to the scratchpad memory; we sum up them; sum up their sizes; then that should be less than or equal to  $k$ ; the total size of the SPM. And for those, which are mapped to the SPM again, we look at the energy gain and compute the sum that is - the sigma that we are computing. So, this  $g$  must be maximized.

(Refer Slide Time: 49:56)



**A Simple SPM allocation algorithm - (3)**

$$E_{g_i} = (E_m - E_s) * n_i$$

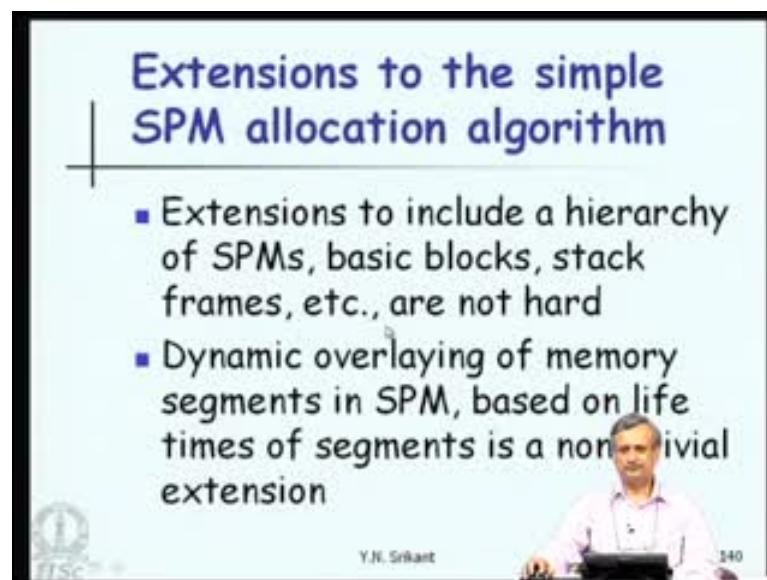
$E_m$  ( $E_s$ ) = energy for one access to main memory (SPM)

$n_i$  = #memory accesses to segment  $i$  (obtained by static analysis or profiling)

Y.N. Srikant 139

How do we compute the energy gain? We just look at  $E_m$ , which is nothing, but the energy for one access to main memory. And we look at  $E_s$ , which is the access to the scratchpad memory. So, the difference between these 2 is the energy gain. Look at the number of memory accesses to the segment  $i$  obtained by static analysis or by profiling the program. So, this count multiplied by this gain gives you the energy gain for this particular  $i$ .

(Refer Slide Time: 50:32)



**Extensions to the simple SPM allocation algorithm**

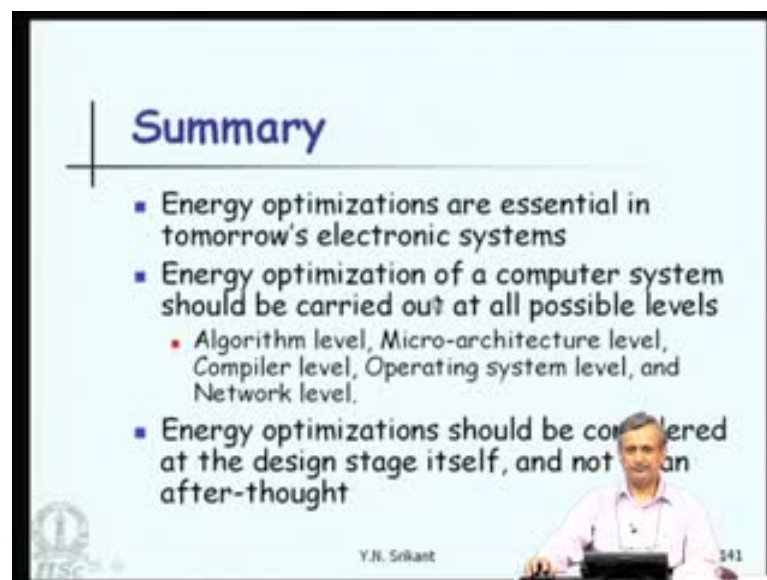
- Extensions to include a hierarchy of SPMs, basic blocks, stack frames, etc., are not hard
- Dynamic overlaying of memory segments in SPM, based on life times of segments is a non-trivial extension

Y.N. Srikant 140

So, once we know that - we can maximize the gain using any integer linear programming tools such as, whatever is available on the internet. There are many tools. There are many extensions possible to this SPM allocation algorithm. For example, if there is a

hierarchy of SPMs of various speeds and energy requirements. Then there are basic blocks, stack frames, etcetera. So, hierarchy of various types of scratchpad memories; then, we have basic blocks, stack frames, etcetera. So, we can include all these in our model. Instead of just function now, the unit becomes a basic block or a stack frame. So, it is not difficult to include all this into our model. However, if we want to do dynamic overlaying of memory segments in a scratchpad memory; based on the lifetimes of segment this is not trivial. In other words, one small part of data or instruction comes into the scratchpad; stays there for the lifetime of that particular segment and then, once it is of no use anymore; some other data memory segment is brought into the SPM and it is used. So, such dynamic overlaying of memory segments based on lifetime of segments is definitely a non-trivial extension and that a nice research topic for future.

(Refer Slide Time: 52:13)



**Summary**

- Energy optimizations are essential in tomorrow's electronic systems
- Energy optimization of a computer system should be carried out at all possible levels
  - Algorithm level, Micro-architecture level, Compiler level, Operating system level, and Network level.
- Energy optimizations should be considered at the design stage itself, and not as an after-thought

Y.N. Srikant 541

So, now we come to the end of this energy based software system lecture. Let us summarize so; energy optimizations are essential in tomorrow's electronics systems. So, energy optimizations of a computer system should be carried out at all possible levels. So, in fact, at the algorithm level, micro architecture level, compiler level, operating system level, and also the network level. It is not a good idea to forget some of these and say, let just the compiler or operating system handle my requirements; I would not do anything at the algorithm or micro architecture level. We need to try hard at every level in order to optimize energy. Energy optimizations should be considered at the design stage itself, and not as an afterthought. So, that is - something we should not be doing.

We can design the software and the hardware and then try to optimize, but we should be looking at these options at the design stage itself.

So, that is the end of this lecture. Thank you.