Computer Architecture and Organization

Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 63
Some Case Studies

In this lecture, we shall be looking at case studies of some modern day processors that we see around us. We shall be basically looking at two such, one is that of a so-called graphics processing unit or GPU, and the other is the evolution of the Intel class of processors.

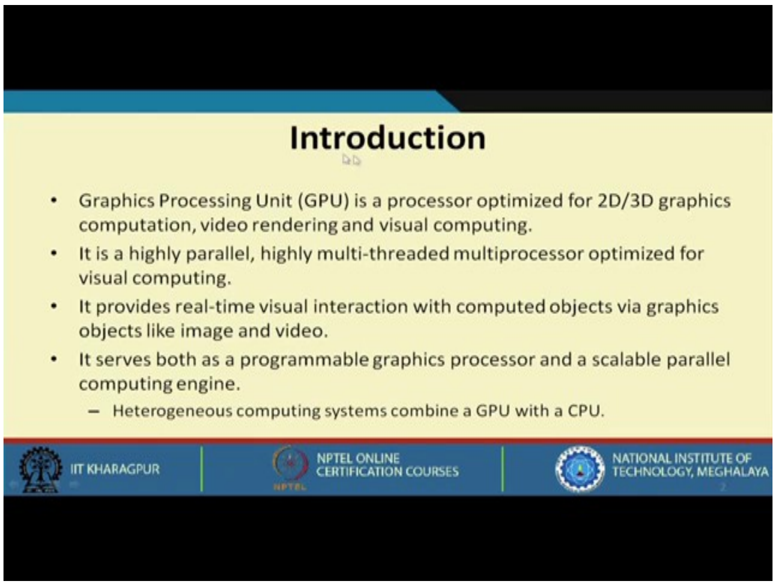
(Refer Slide Time: 00:48)

NPTEL ONLINE CERTIFICATION COURSES
NPTEL
IIT KHARAGPUR
NIT MEGHALAYA

Lecture 63: SOME CASE STUDIES

PROF. INDRANIL SENGUPTA
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, IIT KHARAGPUR

(Refer Slide Time: 00:50)



Introduction

- Graphics Processing Unit (GPU) is a processor optimized for 2D/3D graphics computation, video rendering and visual computing.
- It is a highly parallel, highly multi-threaded multiprocessor optimized for visual computing.
- It provides real-time visual interaction with computed objects via graphics objects like image and video.
- It serves both as a programmable graphics processor and a scalable parallel computing engine.
  - Heterogeneous computing systems combine a GPU with a CPU.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

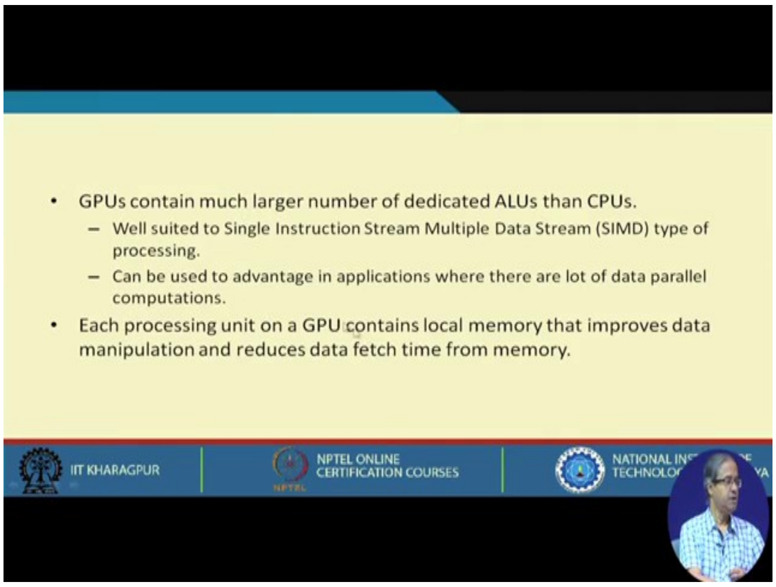
Let us try to explain what a graphics processing unit is. Traditionally GPU were used as a graphics accelerator for a processor, because with the years the requirements and the demand of graphic processing have increased. The processor with the help of its own instructions, it will become very difficult for it to handle all the graphics processing tasks. The trend is that a separate chip or a processor is used dedicated for all the graphics, multimedia, animation kind of applications that has come to be known as GPU.

So, GPU is a processor that is optimized for graphics computations, both in 2D and 3D. These processors have some very specific features that help in video processing, visual computing, etc. Essentially a GPU is a highly parallel highly multithreaded multiprocessor. We talked about parallel processing briefly earlier. A GPU is like a very highly parallel computer architecture which resembles the SIMD kind of processing where there are number of simple processors that are under control of a single control unit. The idea is something like that. With the help of very specific instructions, it can provide real time rendering of videos and similar applications.

The interesting thing is that because of its inherent architecture, it is a highly parallel SIMD machine. So, why use it only for graphics, we can also use it as a scalable parallel computing engine to execute some applications. In general, we can have a heterogeneous computing system where the parts of the competition that can be parallelized can run on

GPU, while the other part can run on a conventional CPU. You will see such computer systems available today where CPU and GPU are residing inside the same box.

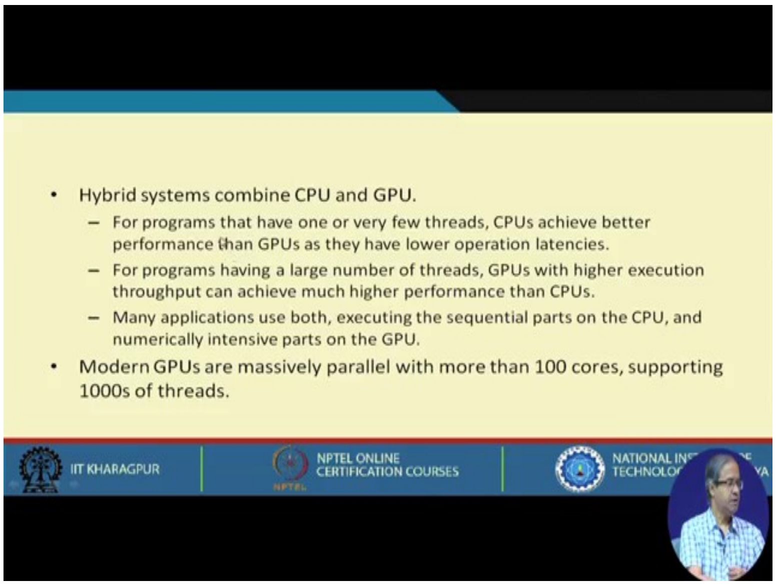
(Refer Slide Time: 03:53)



- GPUs contain much larger number of dedicated ALUs than CPUs.
  - Well suited to Single Instruction Stream Multiple Data Stream (SIMD) type of processing.
  - Can be used to advantage in applications where there are lot of data parallel computations.
- Each processing unit on a GPU contains local memory that improves data manipulation and reduces data fetch time from memory.

Some of the characteristics of GPU is that it is based on the SIMD mode of computation for processing. There are large numbers of ALUs, it goes in the order of 1000s. In a normal CPU, we talk about 2 core, 4 core or 6 core, but here, we are talking about 1000s of very simple processors that are basically ALUs. Such kind of architecture can be used in applications where you are processing on large arrays or vectors of data, and there is high parallelism available in the application. Not only the ALUs, each such processing unit also contains some small local memory so that some local data can be loaded and processing can be carried out parallely on all the processing units. And of course, there has to be a sophisticated load store unit. From the memory, you may have to load a vector or array of data, similarly you may have to store a vector or array of data.

(Refer Slide Time: 05:16)



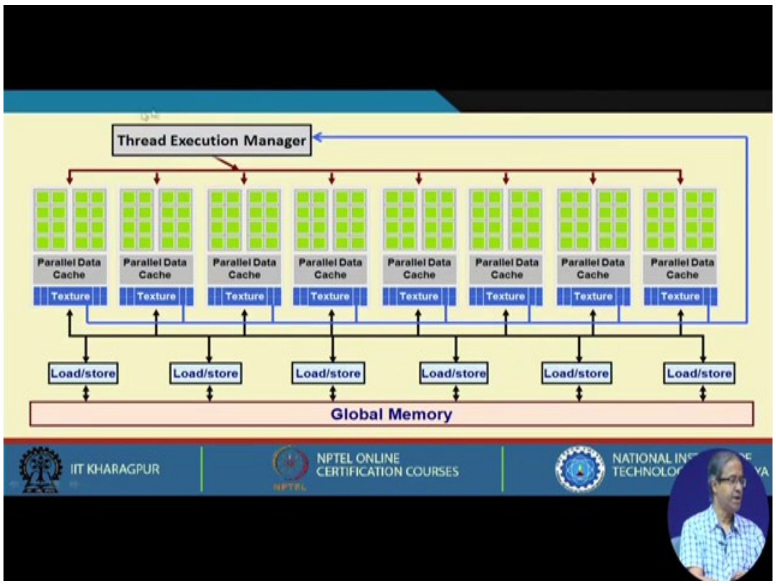
- Hybrid systems combine CPU and GPU.
  - For programs that have one or very few threads, CPUs achieve better performance than GPUs as they have lower operation latencies.
  - For programs having a large number of threads, GPUs with higher execution throughput can achieve much higher performance than CPUs.
  - Many applications use both, executing the sequential parts on the CPU, and numerically intensive parts on the GPU.
- Modern GPUs are massively parallel with more than 100 cores, supporting 1000s of threads.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY

Today you have many parallel computing systems in the market that combines CPU and GPU. For applications where the number of threads are very less or which does not use multithreading, CPUs work better, because CPUs with a few threads are much more efficient, their instruction sets are much more powerful their operation latency is also less.

But if you think of programs that have thousands of threads; then running them on a CPU may not be that efficient. Here GPUs come into the picture. GPUs can provide very high throughput for multithreading kind of application where the number of threads are very large. There are applications where we use both kinds of computing, the sequential parts can be running on the conventional CPU, and the numerically intensive parts that are heavily multithreaded can run on GPUs. There are GPUs that can consist of up to 1000 cores.

(Refer Slide Time: 06:51)

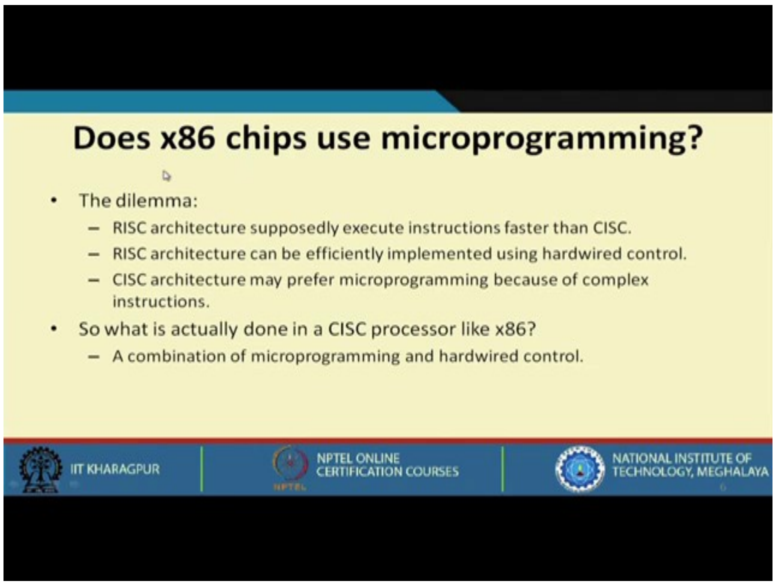


Now, I am showing you a typical architecture diagram of a GPU. You see the processors are divided into clusters, they are not places together they are placed in clusters. In this example, you can see each cluster is having 8 + 8 = 16 cores, and there are 8 such. So, 8 x 16 = 128 cores here. Each of these clusters is having data cache from where data can be loaded and stored, and there is a bus that connects these clusters to load store units. Now to increase the bandwidth between the memory and the cores, there can be multiple load store units that can run in parallel. Here I am showing 6 load store units.

There is a thread execution manager, which is more in the software part. The threads can be scheduled on these clusters and inside the cluster, there will be a simple operating system where depending on the available cores, a thread will be running on one of the cores. This is also a feedback path. As a result of computation, the thread manager can be informed and it can act accordingly. We have a highly parallel computing system where these so-called cores can be very simple as compared to a normal processor core. But suppose there are 128 cores, I can potentially carry out 128 multiplications at the same time, I can carry out 128 additions at the same time.

This is the advantage of vector or array processing.

(Refer Slide Time: 09:00).



Does x86 chips use microprogramming?

- The dilemma:
  - RISC architecture supposedly execute instructions faster than CISC.
  - RISC architecture can be efficiently implemented using hardwired control.
  - CISC architecture may prefer microprogramming because of complex instructions.
- So what is actually done in a CISC processor like x86?
  - A combination of microprogramming and hardwired control.

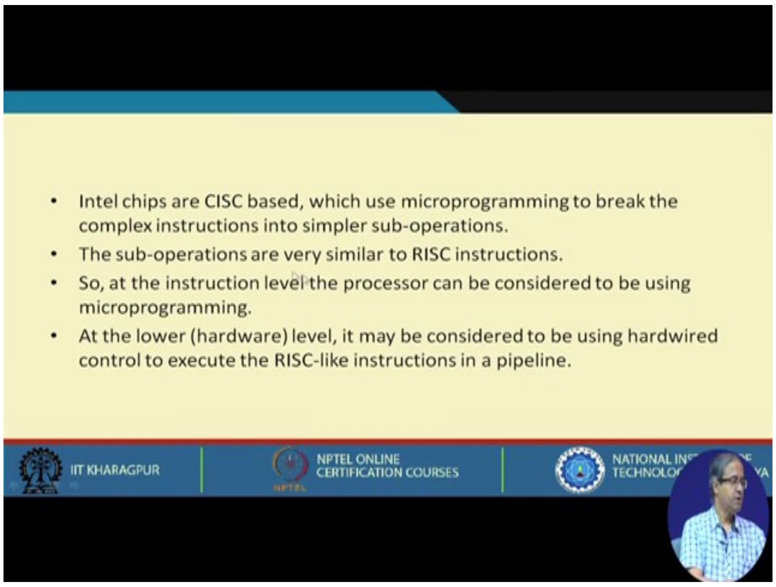
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us digress a little bit, let us look into one question. The x86 family of chips that you know is based on CISC architecture; do they use microprogramming because there is a dilemma that I talked about earlier? Towards the beginning of the class, you have seen that the RISC architectures provide you with an implementation that on the average runs faster than an equivalent CISC architecture platform. So, a program will run faster on a RISC architecture as compared to a CISC architecture, this has been found through experiments.

Now, the question is because of legacy reasons, because of the requirements of backward compatibility, the Intel series of processors that are most widely used in the world today are stuck with this CISC kind of architecture. So, are they compromising on poorer performance this is the question. The dilemma that I am talking about is that; RISC architectures execute instruction faster than CISC, and RISC architecture can be efficiently implemented using hardwired control. But in CISC architecture because of the fact that some of the instructions are very complex, if you want to build a pipeline directly from there, the pipeline may become quite complex. There can be a compromise you can have make.

For the complex instruction sets, you can use microprogramming to break them up into simpler instructions. And at the next level you can have a RISC kind of an architecture that will be executing the simpler instructions using a combination of micro programming and hardware control.

(Refer Slide Time: 11:22)

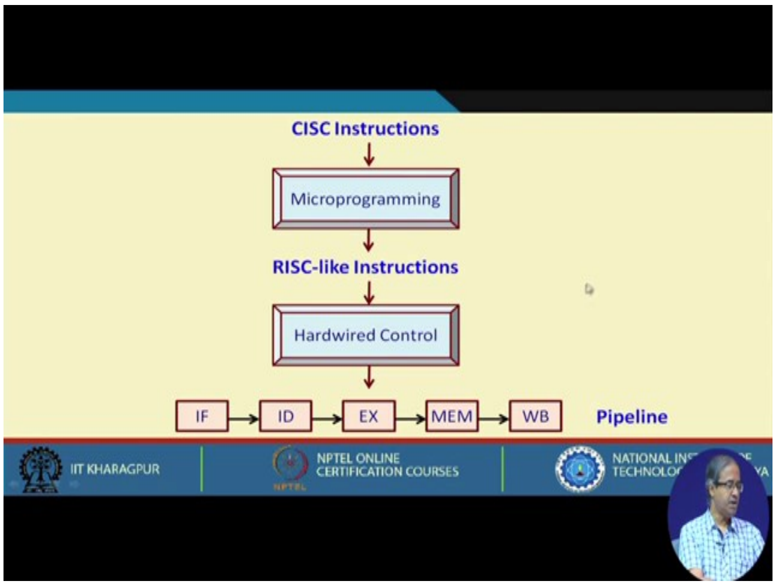


- Intel chips are CISC based, which use microprogramming to break the complex instructions into simpler sub-operations.
- The sub-operations are very similar to RISC instructions.
- So, at the instruction level the processor can be considered to be using microprogramming.
- At the lower (hardware) level, it may be considered to be using hardwired control to execute the RISC-like instructions in a pipeline.

So, what I have said is that the Intel processor chips are based on CISC instructions, they use in the first step microprogramming to break the complex instructions into simpler sub-operations, but here we note that all instructions need not be broken, there may be some instructions that are already like RISC, they need not have to go through this.

These sub-operations are very similar to RISC kind of instructions. So, at the level of the CISC instructions, you can regard that we are using microprogramming, but after the instructions are broken up into this RISC kind of instructions, you may consider that we are using hardware control to execute those instructions in a very efficient and compact pipeline.

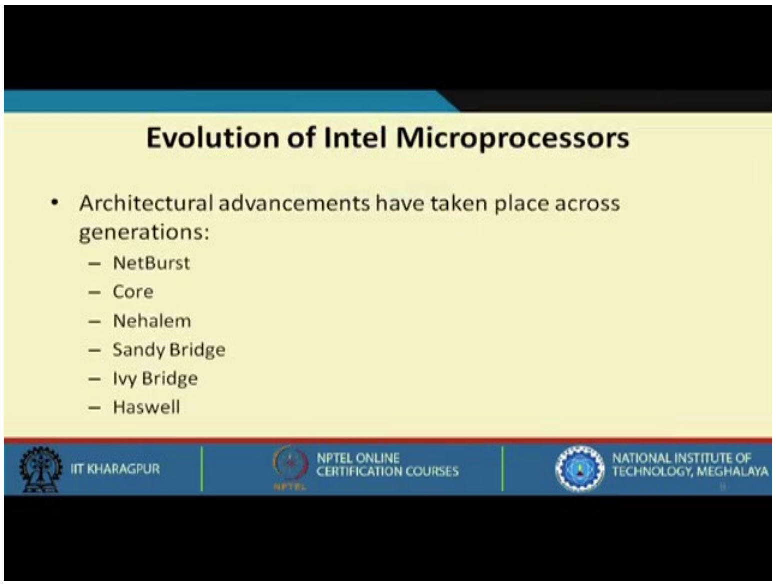
(Refer Slide Time: 12:28)



So pictorially it can look like this. You have CISC instructions at the top level, then you have microprogramming that will be translating them into simple operations, which are RISC like instructions. These in turn will be executed on a pipeline, here I am showing the pipeline of MIPS32; this means, something similar to this.

Just using hardware control, you can execute these RISC-like instructions on the pipeline. This is how instructions are typically executed on modern day Intel processors.

(Refer Slide Time: 13:10)



Evolution of Intel Microprocessors

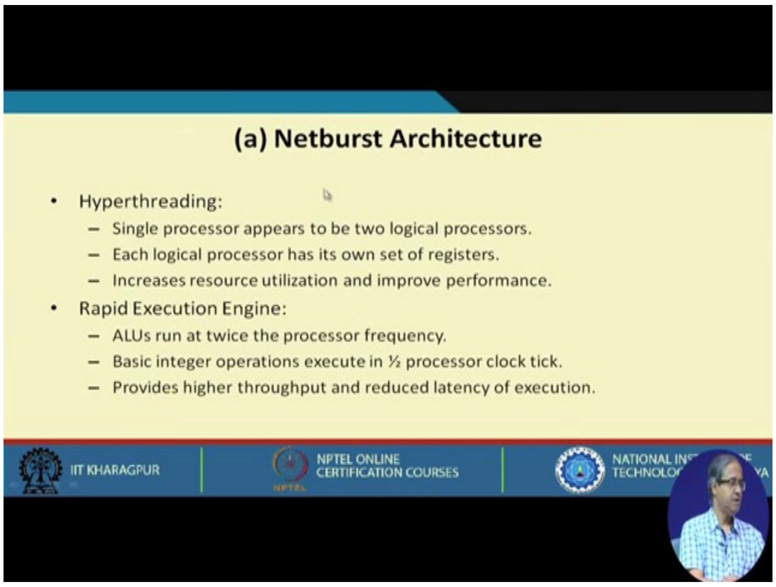
- Architectural advancements have taken place across generations:
  - NetBurst
  - Core
  - Nehalem
  - Sandy Bridge
  - Ivy Bridge
  - Haswell

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, I shall be looking at the evolution of the Intel processor. We shall not be going into the details because it may involve a lot of advanced topics that we have not covered in the class. I shall be trying to give an overview of how this evaluation has taken place, and what are these different families mean. You have been hearing about the Intel processor families, their coming through so many generations like Nehalem, Ivy bridge, Sandy bridge, etc.

So, what are these actually? Let us very briefly go through this. The point is that over the years, there have been architectural advancements that have taken place in the Intel processor families. Some of the earlier architectures were called Netburst that was not so popularly known. Then core, you have seen the core2-duo kind of processors, then Nehalem, Sandy bridge, Ivy bridge, Haswell; these are some names you may have heard. Let us look briefly what are the features.

(Refer Slide Time: 14:29)



(a) Netburst Architecture

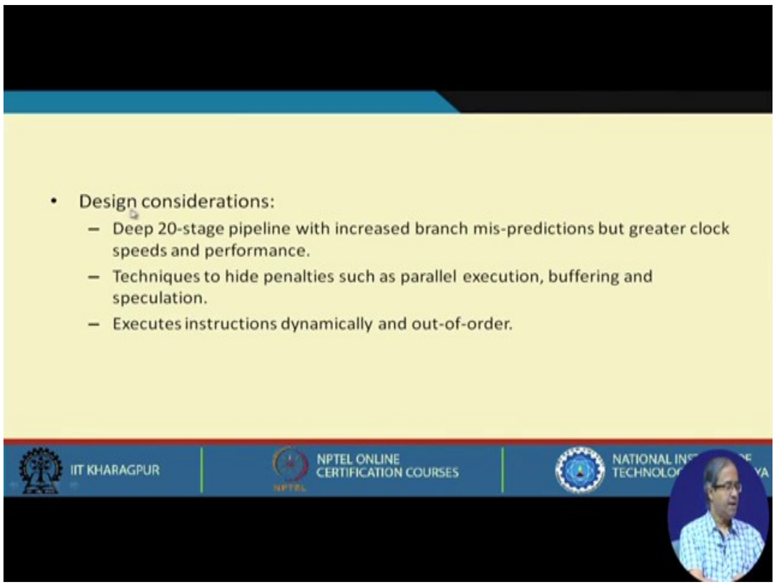
- Hyperthreading:
  - Single processor appears to be two logical processors.
  - Each logical processor has its own set of registers.
  - Increases resource utilization and improve performance.
- Rapid Execution Engine:
  - ALUs run at twice the processor frequency.
  - Basic integer operations execute in ½ processor clock tick.
  - Provides higher throughput and reduced latency of execution.

The first of these architectures is the Netburst architecture. The first thing was that it used hyper threading. Hyperthreading uses some kind of virtualization where a single processor appears like 2 logical processors; as if 2 threads are running on the same processor. To the threads the processor seems to be a separate dedicated virtual processor. Because you have 2 threads you can say that there are 2 virtual processors that run those 2 threads. Each of these logical processors had their own sets of registers, which means we had 2 different register sets, one for thread 1, one for thread 2.

By doing this resource utilization, performance can be improved to a great extent. Because you see, whenever you use multithreading with a single set of registers, when you switch from one thread to the other, you may have to save some registers and restore the new registers. But here because there is separate set of registers, thread switching is very fast. You simply switch to the other register set. There was another concept here that was called the rapid execution engine, where ALUs were running faster than the processor. The ALU clock was running at twice the frequency as compared to the processor clock.

The basic integer operations were executing in half the processor clock tick. If you just correlate this with the MIPS32 architecture pipeline, you are trying to say that for multi cycle operations in the EX stage, you are trying to reduce that time to half. The EX stage executes faster. This will result in higher throughput, and of course reduced latency of execution.

(Refer Slide Time: 16:53)



- Design considerations:
  - Deep 20-stage pipeline with increased branch mis-predictions but greater clock speeds and performance.
  - Techniques to hide penalties such as parallel execution, buffering and speculation.
  - Executes instructions dynamically and out-of-order.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY

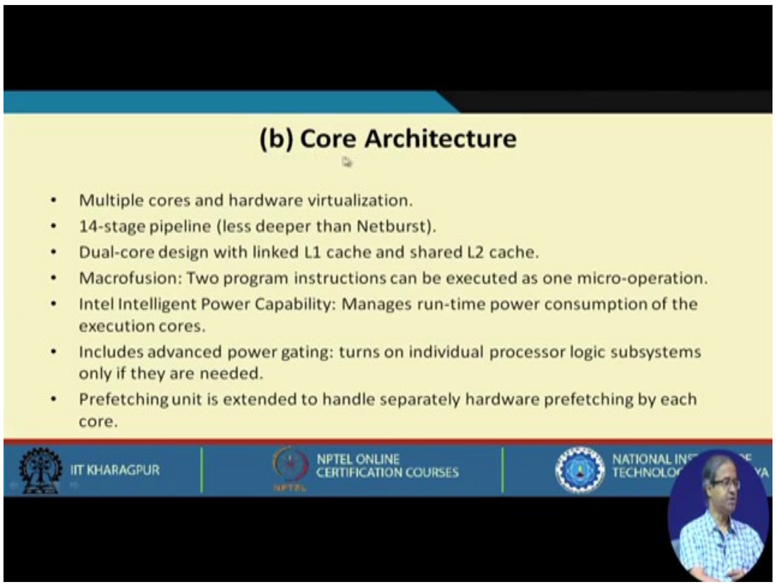
And here there was not much consideration on reducing the number stages in a pipeline to convert it into a RISC kind of architecture. It had a very complex pipeline comprising of 20 stages where the complex instructions were directly mapped and executed.

For branch instructions because of the deep pipeline branch mis-predictions were pretty high, but because of the simpler stages, the clock frequency could be made very high and performance could be made reasonably on the higher side. Various techniques to hide the penalties in the pipeline were also implemented, parallel execution using superscalar kind of processing, buffering particularly in the load store units and speculation. You have already seen in branch you can speculate, you can assume taken or not taken.

And by special hardware control instructions were executed dynamically and also out of order. What is the meaning of out of order? Like for example, say you are fetching some instructions, but you find that one of the instructions cannot start execution because there is a hazard. It requires an operand that depends on an instruction which is already executing.

So, what it will do? It will go to the next instruction see that whether we can execute the next instruction, if it finds there the next instruction is an independent instruction which can be executed, it will execute it first. This is called out of order execution.

(Refer Slide Time: 18:50)



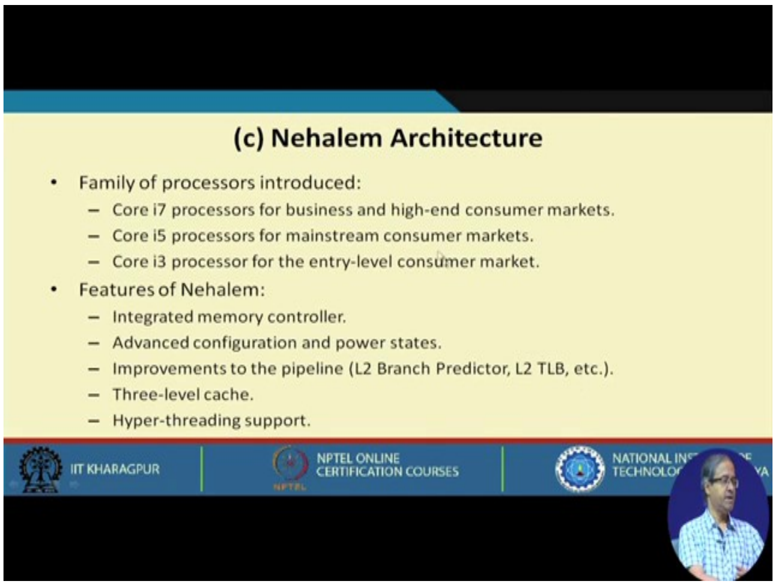
(b) Core Architecture

- Multiple cores and hardware virtualization.
- 14-stage pipeline (less deeper than Netburst).
- Dual-core design with linked L1 cache and shared L2 cache.
- Macrofusion: Two program instructions can be executed as one micro-operation.
- Intel Intelligent Power Capability: Manages run-time power consumption of the execution cores.
- Includes advanced power gating: turns on individual processor logic subsystems only if they are needed.
- Prefetching unit is extended to handle separately hardware prefetching by each core.

So, next came the so-called core architecture. The multi core architecture came from this family. Here you had multiple cores and again hardware visualization by using multithreading. Here the pipelines were made simpler; instead of 20 stages in Netburst it became 14 stages. The first version used 2 cores, they had dedicated L1 cache and shared L2 cache. L3 cache was not there in this family, and there was a concept of macro-fusion where two program instructions logically can be executed as one micro-operation, one in the first core, other in the second core. This is a high-level view, well it started to use some intelligent power capability where runtime power consumption of the execution core was measured, and accordingly, it was controlled. The temperature was also sensed; depending on the temperature the speed adjustment was done, lot of sophisticated features is incorporated here.

Then advanced power gating for low power, where some individual processor logic subsystems were turned on and off depending on whether they are required or not. And there was a separate pre-fetching unit that was extended to support instruction fetching by two cores in parallel.

(Refer Slide Time: 20:31)



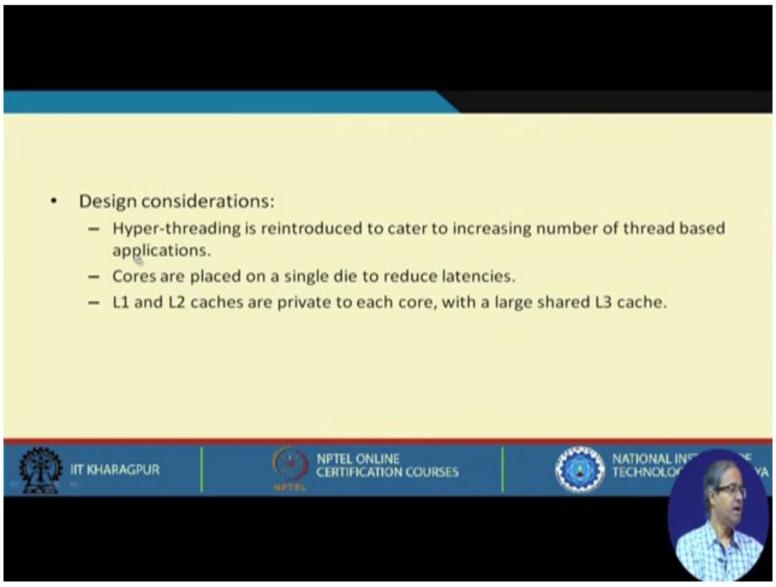
(c) Nehalem Architecture

- Family of processors introduced:
  - Core i7 processors for business and high-end consumer markets.
  - Core i5 processors for mainstream consumer markets.
  - Core i3 processor for the entry-level consumer market.
- Features of Nehalem:
  - Integrated memory controller.
  - Advanced configuration and power states.
  - Improvements to the pipeline (L2 Branch Predictor, L2 TLB, etc.).
  - Three-level cache.
  - Hyper-threading support.

Then came the Nehalem architecture that is the beginning of the kind of cores that we see today, i3, i5, i7. It is from the Nehalem architecture the family of processors was introduced. So, core i7 was primarily meant for business and high consumer markets. Today, we want these i7 processors to be used inside our laptops also for daily use.

So, the cost has gone down, power consumption has gone down. Similarly i5 was meant for mainstream consumer market, and i3 for entry-level market. Some of the features of Nehalem that started to appear was that the memory controller was integrated within the chip, and the power management was made more sophisticated. They are multiple power states like high power, medium power, low power. The operating system can initialize the power state of the processor, and accordingly run programs on that power state. Depending on that, you can conserve battery or you can run your application faster whatever you want. And there are several improvements made to the pipeline using branch predictors, sophisticated translation lookaside buffers, etc. Also here the third level of cache L3 was introduced, and of course, hyper threading support was there.

(Refer Slide Time: 22:11)



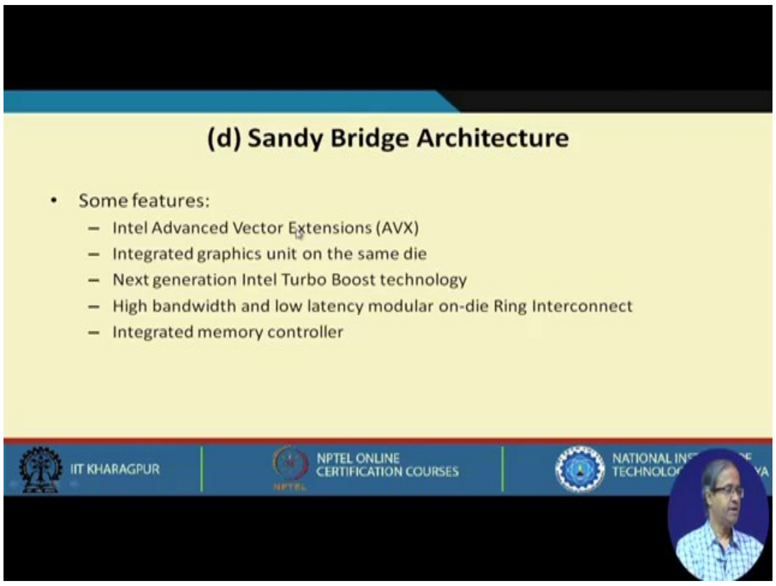
- Design considerations:
  - Hyper-threading is reintroduced to cater to increasing number of thread based applications.
  - Cores are placed on a single die to reduce latencies.
  - L1 and L2 caches are private to each core, with a large shared L3 cache.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Some of the design considerations here was that hyper threading was again introduced in this family because there were several application that demanded larger number of threads. All the cores were placed on the same integrated circuit die, it was a single chip and the first two levels of the caches were private to the core, while the L3 cache was shared among the cores.

Here the number of cores were either 2 or 4, both these 2 kind of families were there.

(Refer Slide Time: 22:55)



(d) Sandy Bridge Architecture

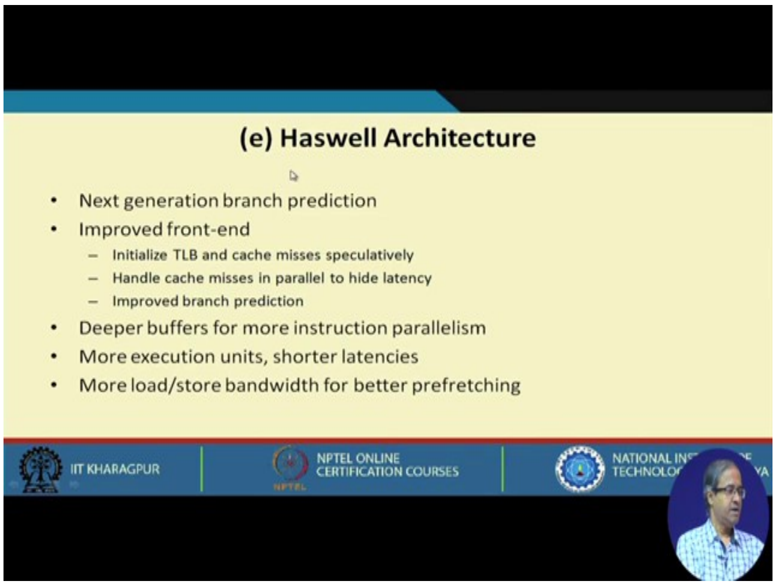
- Some features:
  - Intel Advanced Vector Extensions (AVX)
  - Integrated graphics unit on the same die
  - Next generation Intel Turbo Boost technology
  - High bandwidth and low latency modular on-die Ring Interconnect
  - Integrated memory controller

Then came the Sandy Bridge architecture. Here some vector extensions were made to the processor, and there was a separate engine that was called AVX advanced vector extension.

Secondly the GPU was also integrated on the same die. There was a feature called turbo boost technology that also appeared. Here the system can automatically check the temperature of the chip; if it finds that the temperature is not too high, then at least for a few seconds it can go in a turbo execution mode where the frequency will be increased. Of course, heat generated will also be increased but you can run your programs much faster.

So, for a few seconds you can move to a turbo boost mode and you can execute some applications very fast, but you cannot do it in a sustained manner because the temperature of the chip will be going up quite significantly. Inside the die there was some ring kind of interconnects that was proposed to connect the cores. And also the memory management unit and the memory controller was on chip.

(Refer Slide Time: 24:38)



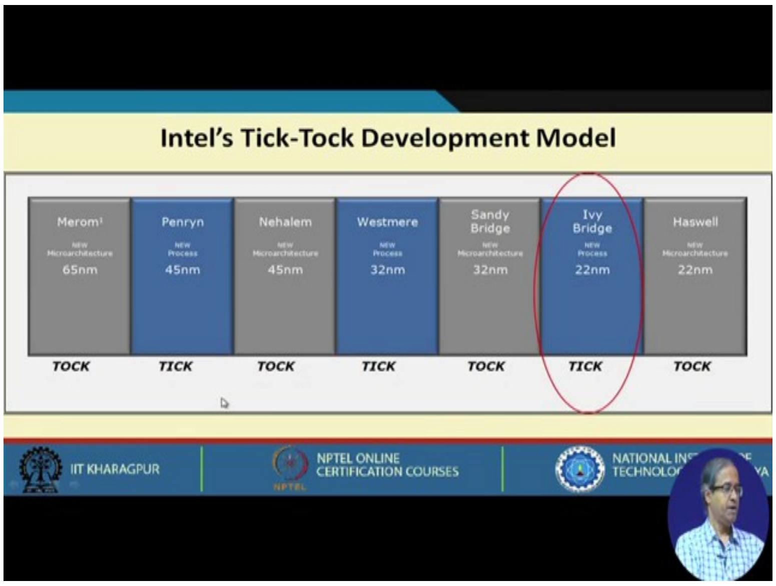
(e) Haswell Architecture

- Next generation branch prediction
- Improved front-end
  - Initialize TLB and cache misses speculatively
  - Handle cache misses in parallel to hide latency
  - Improved branch prediction
- Deeper buffers for more instruction parallelism
- More execution units, shorter latencies
- More load/store bandwidth for better prefretching

Then came the Haswell architecture that is one of the latest. Here the branch prediction was made much more sophisticated using lot of hardware support. And there was some improvement in the front-end like the TLB and the cache misses there were some speculative processing here; multiple cache misses were handle in parallel to hide the individual latencies. As I said branch prediction was improved to a great extent. For the load/store unit there were much deeper buffers; you could fetch larger number of instructions and keep them in the buffer; and from the buffer you can issue them to the pipeline as and when required.

Suppose you have 16 instructions in a buffer. You can decide out of those 16 which instruction to execute next. So, you can have out of order execution also. Here you have larger number of execution units, it is actually highly superscalar. The execution units are made more efficient, latencies are shorter, and again to support this increased speed the load/store bandwidth is also increased.

(Refer Slide Time: 25:59)



Let us look at this picture. This illustrates Intel's philosophy of developing newer generation of processors; this is called the tick-tock development model.

The idea of tick-tock development model is like this. Intel comes up with a new architecture based on the present fabrication technology, this is called tock architecture. Then it moves to tick, which means the same architecture is reimplemented using a better fabrication technology. Then in the next generation you again move from tick to tock; that means, a new architecture family with the same technology, then again tick a better technology comes you move the same architecture with a better technology. You see in this diagram Merom was one of the older architectures; this was fabricated using 65 nanometer technology; this was tock.

Now, the same micro architecture was moved to a new process this was given a different name this is called Penryn; the same architecture was re-fabricated using 45 nanometer technology, this is tick. Then in the next generation a new micro architecture was proposed in the same 45 nanometer technology called Nehalem, then we move to a next generation which was the same micro architecture, but with a better technology 32 nanometer this was called Westmere, then new micro architecture with the same technology 32 nanometer, this was called Sandy Bridge.

You move on to the new process 22 nanometer, this was called Ivy Bridge. Now a new micro architecture based on 22 nanometer, this is Haswell. Like this Intel likes to use the tick-tock kind of an architecture because it comes up with some architectural advancements, implements it with the present day technology, let it run for some time, by that time some newer technology of fabrications are available, port it to the newer technology. In this way it just advances.

With this we come to the end of this lecture, where we very briefly looked at a couple of case studies, one was that of a graphics processing unit and the other was the evolution of the Intel family of processors.

Thank you.