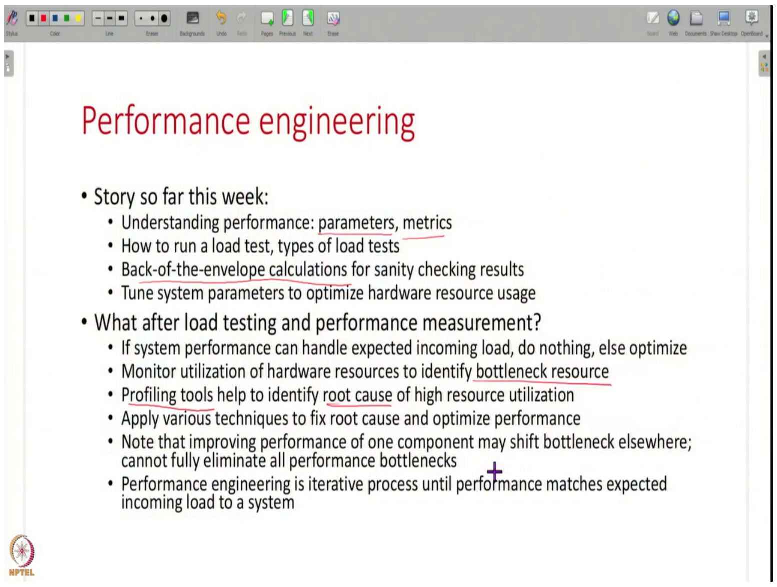
Design and Engineering of Computer Systems
Professor Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 47
Performance profiling and optimization

Hello everyone, welcome to the thirty third lecture in the course design and engineering of computer systems. So, in this lecture, we are going to continue our discussion on performance engineering. So, let us get started.

(Refer Slide Time: 0:28)



Performance engineering

- Story so far this week:
  - Understanding performance: parameters, metrics
  - How to run a load test, types of load tests
  - Back-of-the-envelope calculations for sanity checking results
  - Tune system parameters to optimize hardware resource usage
- What after load testing and performance measurement?
  - If system performance can handle expected incoming load, do nothing, else optimize
  - Monitor utilization of hardware resources to identify bottleneck resource
  - Profiling tools help to identify root cause of high resource utilization
  - Apply various techniques to fix root cause and optimize performance
  - Note that improving performance of one component may shift bottleneck elsewhere; cannot fully eliminate all performance bottlenecks
  - Performance engineering is iterative process until performance matches expected incoming load to a system

So, far this week we have understood what is performance? What are the parameters you vary? And what are the metrics you measure when you do a load test, how to run a load test, the types of load tests. And in the previous lecture, we have seen how to do various back of the envelope calculations in order to do a sanity check of your load test results. And we have also seen basics of how do you tune some system parameters in order to optimize your hardware resource usage.

Now, after you have done a load test, and you have measured performance, if you find that your performance numbers make sense and you can handle all the load coming into your system then you are done nothing else needs to be done here, you expect only a load of 50 requests per second your system already has a capacity of 100 requests per second you are done. But if you find that no no, I expect a lot more load into my system and my capacity is not enough then you have to optimize your system to improve its capacity.

So, how do you go about doing this? So, first, you will find out which is the bottleneck resource at your bottleneck component, you will monitor the utilization of all the hardware resources to find that which is my bottleneck resource. And then you will use what are called profiling tools. So, these tools will help you identify why is your resource utilization so high, for example, if your CPU utilization is 100 percent, in handling certain traffic, then your profiling tools will tell you where is the CPU time being spent, why is it so high?

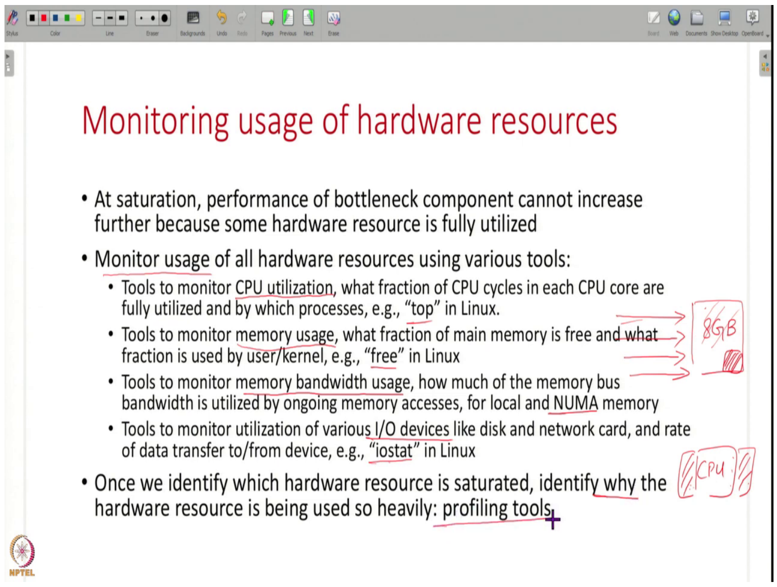
And once you find out the root cause, then you can fix the root cause you can apply various techniques to fix the root cause, and you can improve performance.

So, this is how this is what we want to study in today's lecture. How do you use these profiling tools? And what are some of the techniques to optimize performance. And note that if you improve performance of one component bottleneck will shift to some other component. So, it can never be the case that you fully eliminated all bottlenecks. Suppose your database is the bottleneck, and you are you improved its capacity from 100 requests to 1000 requests per second.

Then some other component that has a capacity of 500 requests per second now, that becomes the bottleneck, then the bottleneck will shift if you optimize that then the bottleneck will shift somewhere else. So, at some point, there will always be the slowest component in a system at any point of time. So, completely eliminating bottlenecks is something that can never happen, only thing you can do is you just keep on improving your bottleneck performance until you can handle all the load coming in.

And once you are satisfied that no matter whatever load I expect, I can handle once you reach that level of comfort, your performance engineering stops. So, this is an iterative process until your performance matches whatever is the expected load into your system. So, now, let us see how you monitor the utilizations and use profiling tools to observe what the root cause of your problem is and some techniques that you can use to mitigate this root cause.

(Refer Slide Time: 3:35)



Monitoring usage of hardware resources

- At saturation, performance of bottleneck component cannot increase further because some hardware resource is fully utilized
- Monitor usage of all hardware resources using various tools:
  - Tools to monitor CPU utilization, what fraction of CPU cycles in each CPU core are fully utilized and by which processes, e.g., "top" in Linux.
  - Tools to monitor memory usage, what fraction of main memory is free and what fraction is used by user/kernel, e.g., "free" in Linux
  - Tools to monitor memory bandwidth usage, how much of the memory bus bandwidth is utilized by ongoing memory accesses, for local and NUMA memory
  - Tools to monitor utilization of various I/O devices like disk and network card, and rate of data transfer to/from device, e.g., "iostat" in Linux
- Once we identify which hardware resource is saturated, identify why the hardware resource is being used so heavily: profiling tools

So, the first important thing is at your bottleneck component, some hardware resource has would have been at full utilization. So, the question is, which is that hardware resource, so, what you have to do is you have to monitor the utilization of all the hardware resources in your system to find out which one is getting exhausted that is limiting the performance of your system.

So, there are various tools available to monitor CPU utilization, for example, top is a very commonly used tool in Linux that will tell you for each CPU core, what fraction of the CPU is 100 percent utilized, 50 percent utilized for each CPU core it will give you that number by this is my CPU fully utilized, is that why I am not able to handle more requests then my capacity or is something else the problem. Then there are tools to monitor memory usage, if you have 8 GB or 4 GB of RAM in your system, what fraction of that RAM is used by users, by OS, what fraction is free, you running out of memory in your system is that why your system is slow.

So, for example, the free command in Linux will tell you that. Then there are other tools to monitor memory bandwidth utilization, note that this is different from memory usage. For example, I could have some 8 GB of memory this monitoring memory usage will tell me what fraction of this is occupied and what fraction is free. On the other hand memory bandwidth is so there is this memory bus between the CPU and DRAM that can only serve some megabytes per second or some gigabytes per second this bus has a certain capacity and how much of this bus bandwidth is utilized and how much is free.

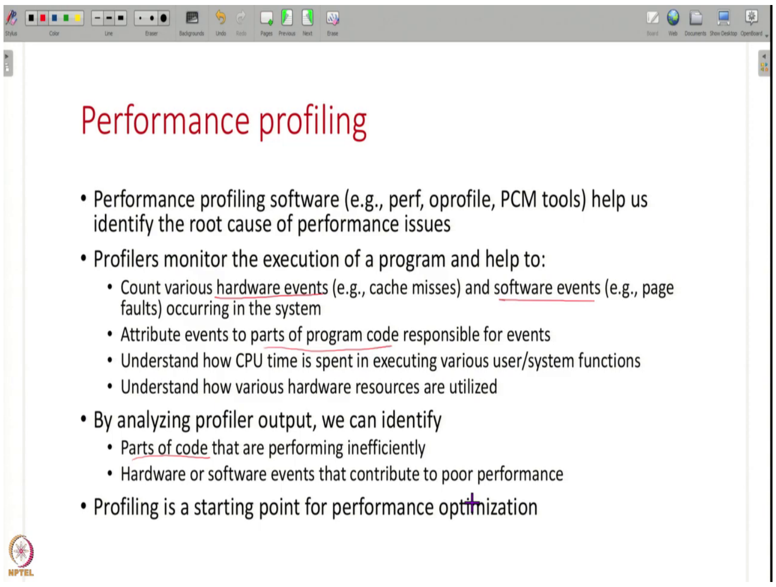
For example, you can only be using a very small fraction of your memory but you could be using this all the time accessing this continuously so, that this bus becomes busy or you could be using your entire memory, but this bus is lightly utilized you are only rarely you filled up your entire 8 GB but you are only very rarely accessing it. So, this bus is free. These are two different things the actual memory usage and the memory bandwidth usage both of these you should monitor to see which one is the bottleneck and especially when you have local memory versus NUMA memory.

So, this non uniform memory we have studied to some CPU cores some memory is closer to some other CPU core some other memory is closer therefore, you have to monitor what is the usage of this local memory, what is the usage of the memory that is farther away is any of this becoming the bottleneck all of this you have to monitor. Then you also have to monitor utilization of IO devices.

So, every IO device can only process data at a certain rate, the disk can only do like, so many reads per second, so many writes per second it has a capacity and if you are exceeding that capacity, then your system performance will hit a bottleneck. So, you can monitor all of this there are tools like iostat in Linux that tell you what is the rate at which your devices reading or writing data, all of these you monitor and from all of these things, you will be able to find out which hardware resources saturated that is limiting the performance of my bottleneck component is it CPU is it, memory is it memory bandwidth? Is it the disk? Is it the network card? Which of these is the performance bottleneck?

Now, once you identify which hardware resources are saturated, the next question comes up, why? Why is that hardware resource saturated? Why is my CPU at 100 percent utilization? Where is the CPU spending all its time? And what is causing this high utilization? That is the next question. And for that we use what are called profiling tools.

(Refer Slide Time: 7:20)



Performance profiling

- Performance profiling software (e.g., perf, oprofile, PCM tools) help us identify the root cause of performance issues
- Profilers monitor the execution of a program and help to:
  - Count various hardware events (e.g., cache misses) and software events (e.g., page faults) occurring in the system
  - Attribute events to parts of program code responsible for events
  - Understand how CPU time is spent in executing various user/system functions
  - Understand how various hardware resources are utilized
- By analyzing profiler output, we can identify
  - Parts of code that are performing inefficiently
  - Hardware or software events that contribute to poor performance
- Profiling is a starting point for performance optimization

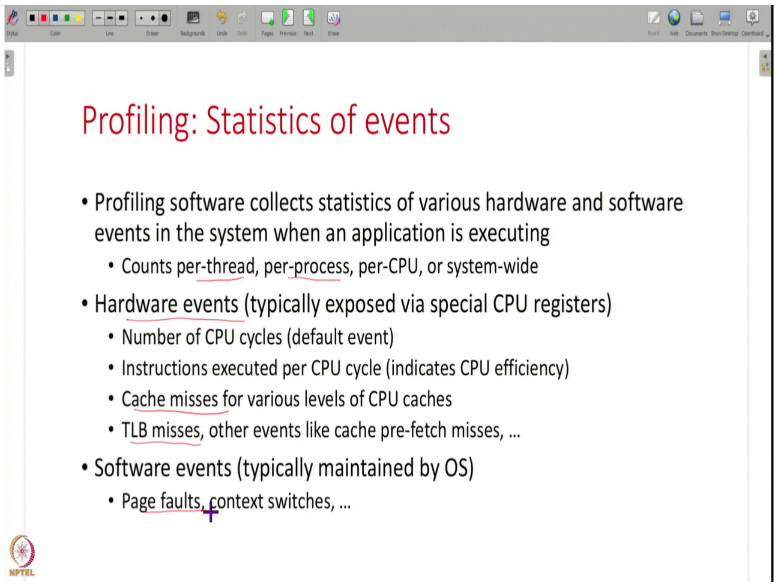
So, there are many profiling software available here are some names of the commonly used ones, but there are many others also out there. So, what these profiling tools do is these profiling software run alongside your application software on the computer, and they will monitor the execution of the program and they will give you various pieces of information about the execution of your program.

For example, this profilers will count various events, like hardware events, software events, like cache misses, page faults, context switches, all of these things that are happening in your system, they will continue. And they will also help you attribute these events to parts of the program.

For example, you can find out that this function in my code is causing a lot of cache misses, this part of my code is causing a lot of page faults you can do this level of analysis, you can count events, you can attribute the events to parts of your code, then this will help you understand how is my hardware resources being used? How is CPU time spent? Where is the CPU spending a lot of its time in which functions, which parts of the code and which part of the code is using which hardware resource all of this you can understand.

So, from this profiler output, then you will be able to identify which are the parts of the code that are performing inefficiently. And what hardware software events are happening that contribute to poor performance? Is it poor cache performance is a page fault, what is the issue, which part of the code is causing that issue? So, once you get do this analysis, then you can go about optimizing your system. So, if your system is performance is poor, you cannot just stare at the million lines of code and wonder what should I do about it? These profilers will help you pinpoint saying, here is the problem. And you can go about optimizing that one part.

(Refer Slide Time: 9:11)



Profiling: Statistics of events

- Profiling software collects statistics of various hardware and software events in the system when an application is executing
  - Counts per-thread, per-process, per-CPU, or system-wide
- Hardware events (typically exposed via special CPU registers)
  - Number of CPU cycles (default event)
  - Instructions executed per CPU cycle (indicates CPU efficiency)
  - Cache misses for various levels of CPU caches
  - TLB misses, other events like cache pre-fetch misses, ...
- Software events (typically maintained by OS)
  - Page faults, context switches, ...

So, what are the events that these profiling software collect? So, they collect statistics about various type of hardware and software events, and these statistics are collected, you can either do per thread per process for each CPU system wide at many granularities you can specify most of these tools gives you the flexibility to specify at what granularity I want to count these events and what all events do I want to come. And the set of events are you have various hardware events that usually the CPU will maintain itself will maintain some count in various registers.

And these profiling tools will read those CPU registers and print it out to you in a nice format that you can understand. So, the various events that are counted are of course, the number of CPU cycles, everything CPU has a certain clock, a certain frequency at which the clock runs, which counts the number of CPU cycles. And you can just count the number of CPU cycles happening. Of course, this is the default event does nothing much interesting, your CPUs specification, how many cycles are there?

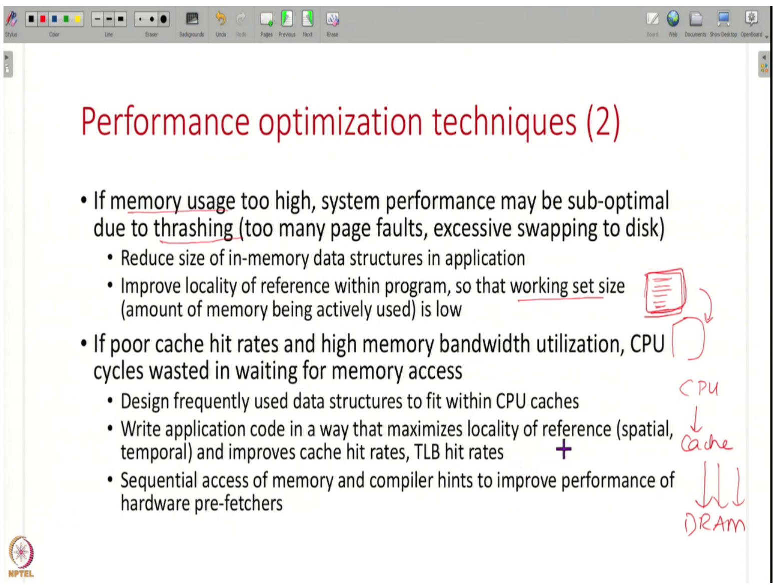
But the more interesting events are how many instructions are executed for every CPU cycle? Is your CPU efficiently executing a large number of instructions? Or is it executing only very low number of instructions for every CPU cycle? So, those instructions per cycle is a very important metric to understand the efficiency. So, why will these instructions be low? For example, if your CPU is waiting for a lot of time, for memory access, your cache performance is very poor, for a lot, it is spending a lot of time waiting for data to be fetched from DRAM.

So, all of these things, there could be many reasons why your CPU is not executing as many instructions for every cycle as it should. So, those instructions per cycle will tell you, is your CPU efficient or is it not being very efficient? Then you have various cache misses, your CPU has many levels of cache. And at each level of the cache, what is the hit rate? What is the miss rate, all of these you will find out? Then other things like TLB misses and your hardware does various optimizations, like pre-fetch things into cache, how many of those pre-fetches have worked, how many of them not work.

So, if you take an architecture course, you will understand a lot more about what are all the, smart things that your CPU is doing? And you can measure all of those counters to see are all of these optimizations like caches, TLB, everything working well in the CPU or is it not working well? So, the all of these hardware events that you can monitor. And some of the most important ones are, your cache misses and TLB misses and instructions per cycle. So, these will tell you, is my system being efficient or not?

Then there are also software events, these are maintained by the OS, like page faults, context switches, is my system seeing a lot of page faults? Is that why performance is poor? So, you can count all of these events using profiling tools.

(Refer Slide Time: 12:08)



Performance optimization techniques (2)

- If memory usage too high, system performance may be sub-optimal due to thrashing (too many page faults, excessive swapping to disk)
  - Reduce size of in-memory data structures in application
  - Improve locality of reference within program, so that working set size (amount of memory being actively used) is low
- If poor cache hit rates and high memory bandwidth utilization, CPU cycles wasted in waiting for memory access
  - Design frequently used data structures to fit within CPU caches
  - Write application code in a way that maximizes locality of reference (spatial, temporal) and improves cache hit rates, TLB hit rates
  - Sequential access of memory and compiler hints to improve performance of hardware pre-fetchers

And what is more, these profilers will also help you attribute the event to specific portions of the code. So, whenever an event occurs, this profiler can note down what is the program counter value at which this event has occurred. So, now it will help me identify all the cache misses caused due to this part of the code. Of course, these events occur at a very high frequency, so every time an event occurs, you cannot, store the program counter somewhere that would be too much overhead. So, what these profilers do is they sample they do what is called sampling.

So, for every 100 cache misses, I will see what the program counter value is something like that, for some subset of events, information about the code responsible for the event is also captured, like the program counter. And this profilers will not just, display some hexadecimal address of the program counter, but they will actually convert it to a function name or something also for easy readability of the user. So, that it is actually this function in my code that is responsible.

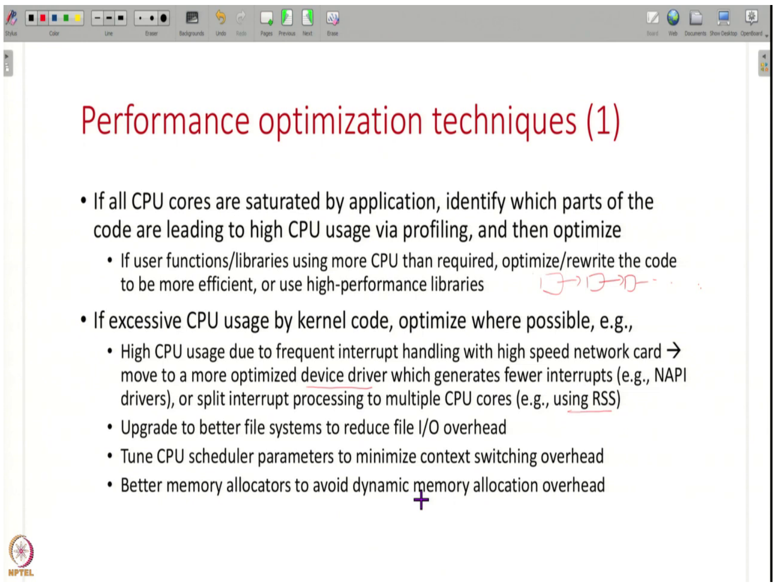
So, by sampling this program counter value periodically, we know which part of the code is consuming, what fraction of CPU cycles, every time for every few CPU cycles, if you profile your program counter, the program counter was here, once here, once here, once here, here, here. Most of the time the program counter was here, then that this function is the one that is actually very time consuming. And very few times in very few samples, the program counter was in other places.

So, that this parts of your code, these functions are not taking up a lot of time, whereas this function is taking up a lot of time. So, by sampling these program counter values, you will know where our CPU cycles being spent by sampling program counter values on cache misses, you will know which part of the code is responsible for cache misses, things like that. So, not just counting events, but you are also able to attribute events to specific parts of the code. So, that this is a starting point for you to optimize your code, you will know which part of your application needs optimization.

So, now, let us briefly see now that you have, monitored hardware resources, you found out which resource is the bottleneck, then you have profiled your core to try and understand which specific parts of your code are performing sub optimally. Once you have looked at all of these measurements, you can go ahead and do some performance optimizations, what optimization you do will depend on what your measurement results are like. But in the next few slides, what I am going to do is point you to some of the common patterns, common issues that occur in systems and some of the common optimizations that are undertaken.

So, of course, there are many more optimizations that can be done and in one lecture, I cannot cover all of them. This is a full course by itself, but I will strive to present to you the most common optimizations that people do once they profile their system.

(Refer Slide Time: 15:05)



Performance optimization techniques (1)

- If all CPU cores are saturated by application, identify which parts of the code are leading to high CPU usage via profiling, and then optimize
  - If user functions/libraries using more CPU than required, optimize/rewrite the code to be more efficient, or use high-performance libraries
- If excessive CPU usage by kernel code, optimize where possible, e.g.,
  - High CPU usage due to frequent interrupt handling with high speed network card → move to a more optimized device driver which generates fewer interrupts (e.g., NAPI drivers), or split interrupt processing to multiple CPU cores (e.g., using RSS)
  - Upgrade to better file systems to reduce file I/O overhead
  - Tune CPU scheduler parameters to minimize context switching overhead
  - Better memory allocators to avoid dynamic memory allocation overhead

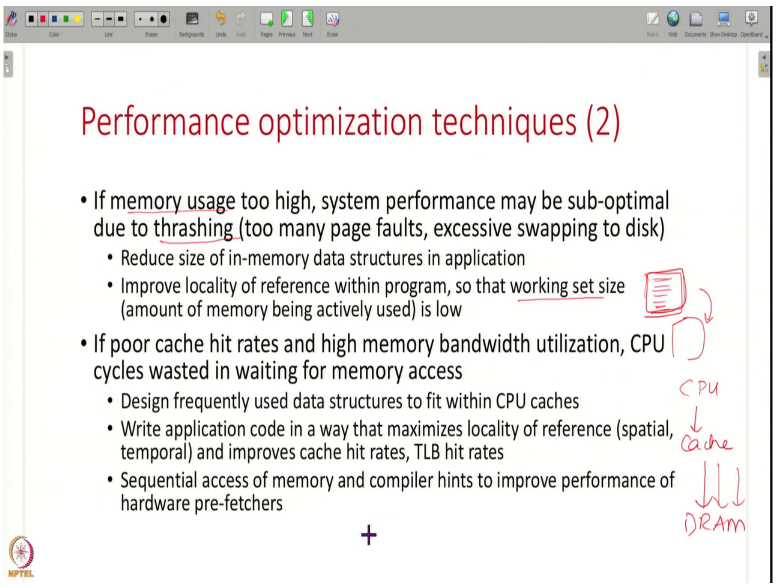
Of course, the most common thing is most systems will have a CPU bottleneck, the CPU cores are fully saturated by the application. And then you will profile the core to see which specific functions in your application are using up most of the CPU. And then you will try and optimize those functions. For example, if there is some function, some library, the link list library that is actually using up a lot of CPU, then you might try and move to a better implementation of the linked list or if it is a, a hash table or map library that is using up a lot of CPU might move to a better library.

Or in your code if you have written a store data in some linked lists that is being traversed every time and most of your CPU cycles are spent traversing the linked list suppose that is what you find out, then you might replace this linked list with a more efficient data structure. Once you pinpoint the source of the inefficiency, you can go and rewrite that code, optimize that code, use better high performance libraries and try and eliminate this performance issue.

But if sometimes, not just user code sometimes even OS code might also be consuming a lot of CPU cycles. In such cases, of course, you cannot replace your operating system that easily, but you can optimize wherever possible. For example, if you find that a lot of CPU time is being spent in handling interrupts, you have a network card, a high speed network card it is getting a lot of packets and most of your CPU time is spent in handling interrupts, then what you can do? You can optimize your device driver, we have seen this there are device drivers like the NAPI driver, that generates fewer interrupts or if all interrupts are coming to one CPU core using techniques like RSS, this also we have seen before you can split your interrupt processing to multiple CPU cores.

So, you can optimize your OS also, in some ways, or if your file system, file IO is the bottleneck of use a better file system. If context switching overhead is the bottleneck tune your CPU schedule, in parameters, tune your scheduler algorithm so that you do not have so many context switches. If you find that memory allocation malloc is the bottleneck, then you use a better memory allocator instead of doing this dynamic, general purpose malloc. So, depending on where your problem is inside the system software, also sometimes you will be able to optimize it. So, this is one set of techniques.

(Refer Slide Time: 17:28)



Performance optimization techniques (2)

- If memory usage too high, system performance may be sub-optimal due to thrashing (too many page faults, excessive swapping to disk)
  - Reduce size of in-memory data structures in application
  - Improve locality of reference within program, so that working set size (amount of memory being actively used) is low
- If poor cache hit rates and high memory bandwidth utilization, CPU cycles wasted in waiting for memory access
  - Design frequently used data structures to fit within CPU caches
  - Write application code in a way that maximizes locality of reference (spatial, temporal) and improves cache hit rates, TLB hit rates
  - Sequential access of memory and compiler hints to improve performance of hardware pre-fetchers

Now what happens if your memory usage is too high, you have all the RAM in your system is somehow being used. And because of that, we have seen this before, what happens if your memory is too fully occupied, you will have thrashing, what is thrashing, some of your pages will be sent to the swap disk.

And every time you want to access a page, there will be a page fault, or most of the times there will be a page fault, you have to send some other page to the swap space, get this old page back then this you will send to swap space get something else back, you are constantly doing this swapping to disk and servicing page faults and all the time is spent handling this and not actually doing application work.

So, if you are using too much memory in your system, this can happen. This is called thrashing. Then how do you handle this, if your profiler shows that a lot of time is spent in swapping, and it shows that page faults are high, then you can try and reduce the size of your in memory data structures, maybe not store everything in memory, but store them on disk wherever possible. And you can also improve the locality of reference in your program.

So, suppose I have, access some parts of my memory, let me try and finish all the processing on this memory before moving on to some other part. So, if my current set of pages that I am accessing the working set size is small, then these will be in memory most of the time, I do not have to swap them from disk. So, improve locality of reference that is try and repeatedly use whatever memory you have used so that you are actively being used, whatever memory is actively being used, the working set sizes small. That is one technique.

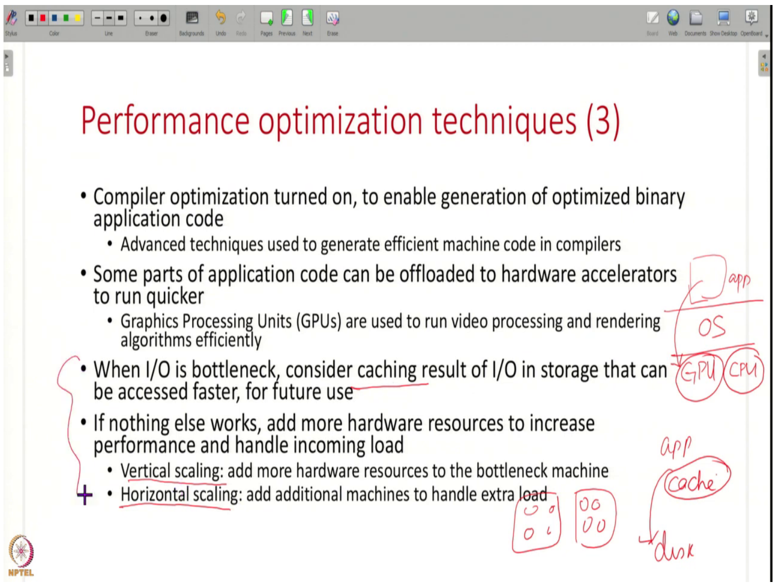
Then if you find that you actually have poor cache hit rates and your memory bandwidth usage is high. The total memory consumed is low, but the memory bandwidth to read and write this DRAM that has been consumed. Why is that being consumed because, the CPU will check caches. And if it is not there in cache, then it will go to DRAM. And if your cache hit rate is very low, your cache is not supplying most of the data, then most of the time you are going to DRAM and this bandwidth will become the bottleneck.

If that is what you find this utilization of this bandwidth is high and cache hit rate is poor, if that is what your profilers tell you. What you can do is we have studied this before how to optimize your cache usage of your system. You can design your data structure such that they fit into caches, you can write your code such that your locality of reference is improved.

Whatever you have gotten to cache, you are trying to access that first, instead of jumping around in your code, you can improve your TLB hit rates, you can try to sequentially access your memory, so that the CPU hardware today does prefetching tries to fetch ahead of time, whatever memory things you will use. So, all of these things you can do in order to improve your cache performance.

And we have studied this in a lot of detail when we were studying the memory management part of the course. So, depending on what the problem is, if this is the problem, then you do this optimization. Then there are many other optimization techniques also.

(Refer Slide Time: 20:37)



Performance optimization techniques (3)

- Compiler optimization turned on, to enable generation of optimized binary application code
    - Advanced techniques used to generate efficient machine code in compilers
- Some parts of application code can be offloaded to hardware accelerators to run quicker
    - Graphics Processing Units (GPUs) are used to run video processing and rendering algorithms efficiently
- When I/O is bottleneck, consider caching result of I/O in storage that can be accessed faster, for future use
- If nothing else works, add more hardware resources to increase performance and handle incoming load
    - Vertical scaling: add more hardware resources to the bottleneck machine
    - Horizontal scaling: add additional machines to handle extra load

For example, compilers themselves do a lot of optimizations, when the compiler generates the machine code, the binary executable itself will do some optimizations in order to use the underlying hardware better. So, you can when you compile your code, you can provide various options to the compiler to do these optimizations. And there are also other techniques being used today, where some part of your application code can be actually offloaded to some special pieces of hardware.

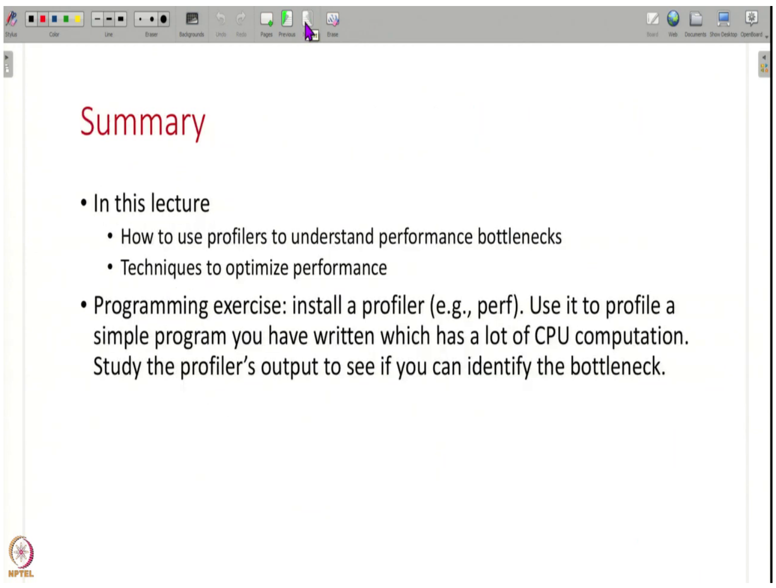
Like for example, you have the GPU called the graphics processing unit. Anytime you have to do any graphics processing, like video processing, or, displaying videos, rendering videos, all of that you can actually offload it to the GPU. The GPU will do it much faster than running some software on the CPU. So, there are special hardware accelerators available for specific applications, you can use those. If you are using up all the CPU to do video processing, then the site will be very inefficient instead use the GPU to do the video processing.

Similarly, when IO is the bottleneck, suppose the application is doing a lot of IO to from disk or something this is the bottleneck, then what you can do is you can maybe use a cache like the disk buffer cache. You can store the results of IO somewhere, so that they can be accessed faster instead of going to the disk. So, caching is one important technique that you can use.

Then, if nothing else works, you have tried everything all possible things, nothing else works, still your hardware resources like fully utilized, its performance is not improving, then the only option is you just add more hardware to your system, no matter what you do, your CPU is 100 percent utilized, your system is only doing 100 requests per second. But you want to do 200 requests per second then what do you do?

You just add more CPU cores to your system or you just add another machine and run your system into different machines. So, that is called Scaling. Vertical scaling is to one machine itself add more CPU cores. Horizontal scaling is add another machine altogether with extra CPU cores. So, this is the last and final option, I have done the best possible I can with my resources, I cannot do anything more than the only way to improve performances add more resources. So, this caching and scaling both of these, we are going to study in the next 2 lectures.

(Refer Slide Time: 22:59)



Summary

- In this lecture
  - How to use profilers to understand performance bottlenecks
  - Techniques to optimize performance
- Programming exercise: install a profiler (e.g., perf). Use it to profile a simple program you have written which has a lot of CPU computation. Study the profiler's output to see if you can identify the bottleneck.

So, that is all I have for today's lecture. In this lecture, what I have told you is how you can use software called profilers to understand the performance of your system and pinpoint where is the performance bottleneck? And I have described to you some general techniques that people follow in order to fix this performance bottleneck. In the next couple of lectures also, we are going to see these some of these techniques like caching and scaling in more detail.

So, as a programming exercise, you can try and install one of these profilers, for example perf, use it to profile a simple program that you have written and actually see the output of the profiling. Suppose you have written a program where some heavy computation has been done in a function, then you can actually see the profiler output that it will tell you that this function is where most of the CPU cycles are being spent. So, get some hands on experience in using profilers and understanding the output of these profilers. So, that is all I have for this lecture. Let us continue our discussion in the next lecture. Thank you.