Multi-core Computer Architecture - Storage and Interconnects

Dr. John Jose
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 17
QoS of NoC and Caches in TCMP Systems

Welcome to the 17th lecture. Today our topic of discussion is on Quality of Service of Network on-chip and Cache memories in a Tiled Chip Multi-core system. We know that on a TCMP, ultimately it is some application that is going to run, the operating system is going to map various applications into one of this tiles, and the application is going to execute there. And during the execution of these applications which happens parallelly across all the cores, they are going to use the caches for instruction and data access; and whenever there is a miss that happens in the caches, especially in the L 1 caches through the interconnect you are trying to access the data from L 2 or from main memory.

So, in this context, the role that is played by the caches and to the network on-chip is very crucial as for as the application level performance is concerned. In the last few lectures, when we were discussing on low power network on-chip by deflection routers, we got familiarized with some of the matrix, quality matrix that we used something like average packet latency, deflection rate. Similarly, in caches we have seen average memory access time, hit rate, miss rate, miss penalty. So, all these are factors which will show how good that particular resource is with respect to a cache hit rate or a miss penalty or average memory access time will tell you, how good that particular cache memory is.

Similarly when you come to network on-chip, this average packet latency, the number of hobs that is travelled, buffering time, deflection, these are all parameters which will tell you, how good NoC router micro architecture is. Ultimately, it is not these individual performance on that particular resource that is important as for as a user is concerned.

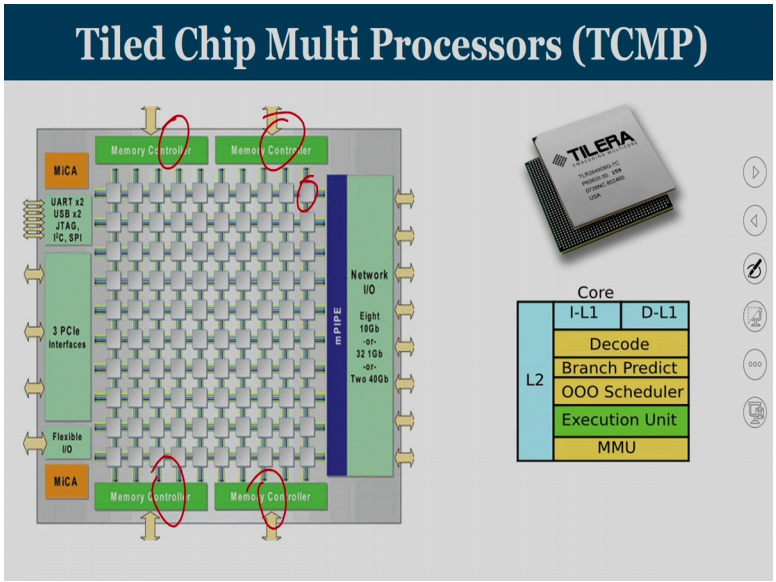
When I am going to run an application, for example, I am going to play a media file on my smart phone a multi-core smart phone, I am not really bothered about what is average memory access time, what is average packet latency, these are all underlined micro architectural characteristics. From a user perspective what I really look is, when I click

on an application, when I activate an application, how long I have to wait for the application to trigger.

Let us say media file is being played, I wanted to drag the progress bar to some point, I wanted to watch a movie. Let us say first 20 minutes I do not wanted to watch, because there is nothing interesting there. I know that there is a good song at the 20th minute, I am going to drag the progress bar of the media application into the 20th minute, I experienced some delay. And this basically happening is during the fetching of those particular wave file, it is not available in cache memory, it has to go through the underlined network or interconnect and there has to bring it back.

So, it is not the average packet latency how much time from the performance perspective of an application, what is a application stalled time. So, with respect to the quality of experience or quality of service me as a user is going to get is from a different metric we have to asses. So, today we are going to study about few techniques that are applied on tiled chip multi-core processors, consisting of caches and interconnect, which will improve the quality of service has together to the end user perspective.

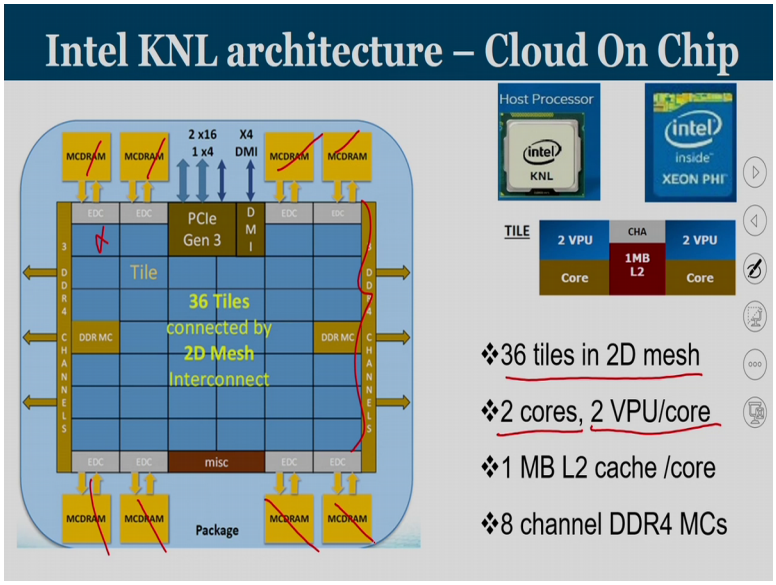
(Refer Slide Time: 04:23)



So, we have already seen the tiled chip multi-core processors are there. And we are having many such chips and each of this chip is going to have a core, which consist of the decoders, the branch prediction, the schedulers, the execution unit, the memory

management unit all these things are there and we have memory controllers also multiple memory controllers on the chip. So, our future is going to be with these kind of chips.

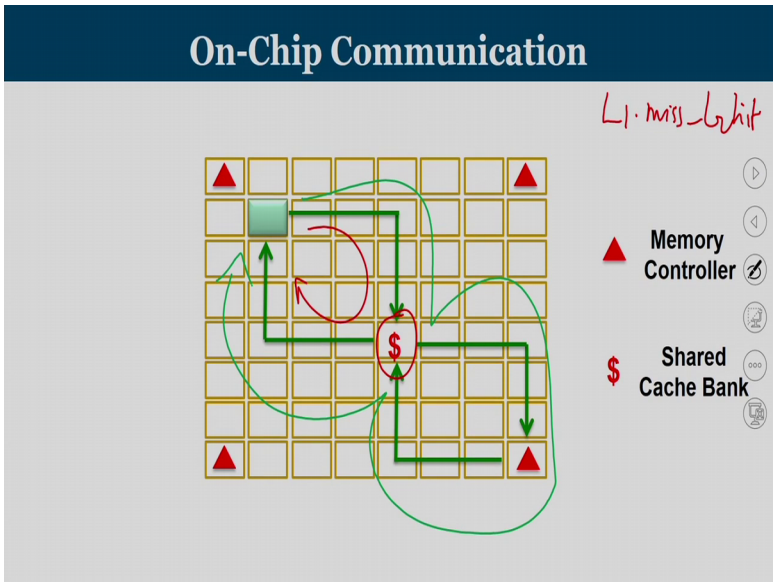
(Refer Slide Time: 04:53)



And one such very prominent chip is known as the Intel KNL its called Intel Knights Landing Processor, it is also known as Intel XEON PHI, which has total of 36 tiles organised two-dimensional mesh. So, you can see that it is 6 cross 6; 6 row wise and 6 column wise, and each of this tile you have 2 cores, and it is basically 2 virtual processing unit per core, so all together 4 virtual processing unit is there per core.

You have 1 MB of L 2 cache that is per core, and the 8 channels of DDR4 memory controllers. So, you can see that these all are the memory controllers that are there on the both side. This is a very common TCMP Tiled Chip Multi-core Processor that that is there in the market. Similarly, future multi-core processors are going to be of the order of this XEON PHI processor, what is Intel has developed or something on a larger magnitude.

(Refer Slide Time: 05:49)



Now, let us try to understand the one chip communication from the perspective of an application. So, consider the case that you have an application that is going to be there and this application is having some L 1 hits and the application is working very fine. Now, it got a L 1 miss where you have to generate a packet into the network and that is going to come like this. So, what you see in the corner are basically memory controllers, you assume that there are 4 memory channels in this particular TCMP. And it is so happened that your application that we are talking about is going to have its L 2 cache addressed that is mapped in this particular core. So, there need to be an NoC packet that travelled all the way from the source core into the L 2 cache core, using x, y routing.

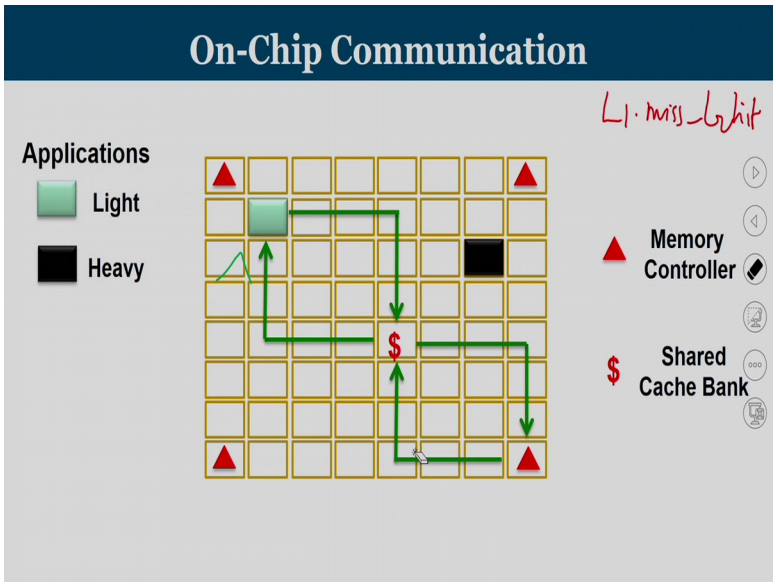
So, you can see that the packet move in x direction and then y direction and then it is going to reply with the data. So, this is the case how you are going to handle with an L 1 miss which is essentially an L 2 hit. There can be cases where you know that sometimes you may not be able to get a data, which is going to be a hit in L 2. So, in that case when it goes to L 2, it comes to know that it is a miss in L 2, then based on address mapping; you know that it is going to be connected to this memory controller or the data is physically present in that portion of your main memory.

So, one more request is generated from the L 2 tile into the main memory tile that is towards the corner, data is return back to L 2. L 2 is completely filled with the data that is block is transferred to L 2 and from L 2 it creates the reply packet. There is a order in

which the data is created is through the network. So, you have a processing core that is going to generate a request as say cache memory packet into the L 2 cache core; and if it is the hit, this will take care of an L 2 hit.

And if it is going to be a miss, then things will take little bit more time it goes to L 2, and then it goes to main memory, and then it comes back to L 2, and then only you are going to process; so that is the way how L 1 misses are going to be handled from L 2 and from main memory controller. Meaning depending on the address you may have to generate packets to various tiles inside the chip. So, for a different address, it may be two a different portion inside the chip. So, this is how an on-chip communication happens.

(Refer Slide Time: 08:33)

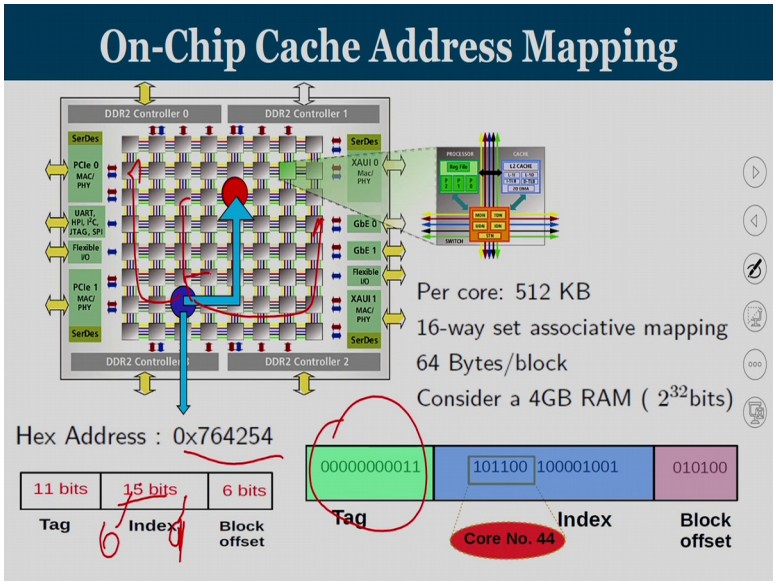


And your applications can be categorised as some applications are going to be light and some applications are going to be heavy. Let us try to understand, what is this concept of light and heavy applications. An application is let to be light from the perspective of a network, when the number of misses the application generate is rather less; when there is less number of cache misses that is the application is encountering, that will lead to less number of packets meaning less traffic in the network.

When application is going to generate more number of cache misses, it will pumping more packets into the NoC, we call such kind of applications are heavy applications. So, the classification of an application into light or heavy is based upon number of cache misses that the application is going to have in a different unit of time or indirectly it is

going to be the number of packets this application will trigger into the underlined network on-chip.

(Refer Slide Time: 09:27)



So, let us try to revisit the concept of cache address mapping once again. So, consider the case that you have a TCMP where per core you have 512 KB of cache and 16 way associating. So, since it is 64 core totally I have 32 megabytes of L 2 cache, and this L 2 cache is 16 way associative and 64 byte block, and total we have 4 GB of DRAM. So, total of 2 power 32 locations are there in the DRAM.

So, the address the physical address consist of 32 bits. This 32 bits are divided into 11, 15 and 6; that means, there are 2 power 15 sets and each set has 16 lines, because it is 16 way associative. And each of this line can accommodate 64 bytes of data that is why it is called block offset and remaining is called your tag. So, for a different cache configuration, different size or different associativity of a cache, this numbers that is given to the tag index and offset is going to changed.

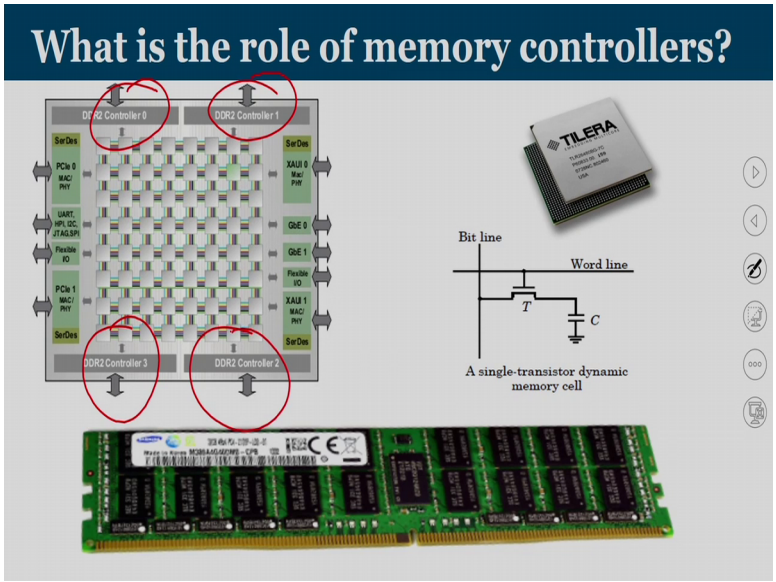
So, this is an example we have seen in our previous lecture. Now, consider the case that you are going to have an address. Let us say, you are going to have an address which is represented in the hexadecimal form. Now, this address when you do the split up, this is the way how this 32 bits are going to split; the green portion indicates the tag bits, and the blue portion indicates this set, there are 2 power 15 sets. So, 15 bits are reserved for the set index and the last 6 bit are for the offset.

Now, since the entire 2 power 15 set is scattered across 64 tiles, when we get an address in order to find out where the address is mapped; where the L 2 is been mapped, we have to divide the address portion that is the set index portion into two components, out of the 15 bits in the set; you are going to take the first 6 bit, which will tell you which is the destination tile, and the last 9 bit is going to tell you which is going to be the set within the tile.

So, this shows that for a packet that is generated from this blue core that is shown in the TCMP; this particular address is mapped to the 44th core, and 44th core is been given by the red colour spot. So, packet are generated from the blue core into the red one. They are depending on a different address, let us say it is a different address rather than 0x764254 if it is a different address, then packet may be going from blue to different, different tiles; and some maybe to nearby cores, there are some may be to the further core.

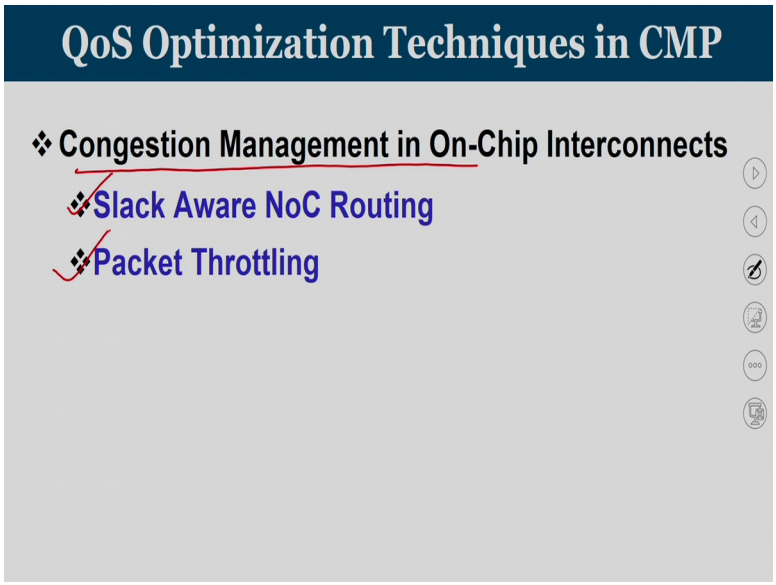
So, it is basically the cache address mapping that governs what is going to be the destination of a request packet. So, some packets will travel longer in the network, when the address is mapped to those particular cores; and some may will some may be travelling relatively to smaller distance in the network.

(Refer Slide Time: 12:33)



Let us try to see, what is the role of memory controllers, we have already seen that memory controllers are there on-chip and they generate appropriate timing and control signals for effective transfer of data from the memory controller into the DRAM device.

(Refer Slide Time: 12:51)

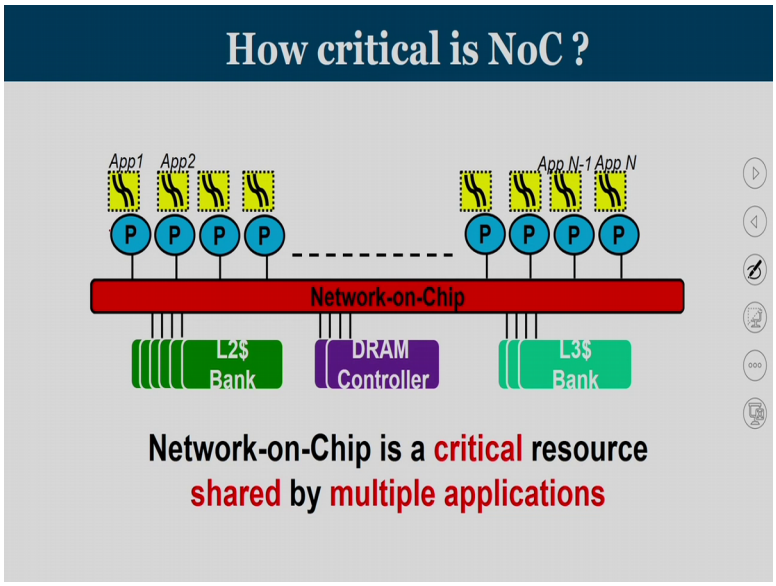


Now, having said this background let us try to understand, how can you improve quality of service in a chip multi-core processor. So, today in this lecture we are going to understand two such proposed works that are published in the recent computer architecture conferences from good research groups. So, I am going to discuss about those two published works, and many computer architecture so researches following these kind of works.

So, students who wanted to work further are into these topics, as we are moving to the end of this course only if couple of lectures left; those who wanted to explore further deeper or work on a thesis, may be at masters or doctoral level then it is good to follow these kind of published works in this domain. So, with the background that we have about a fair idea of storage on-chip, and a fair idea of interconnect on-chip. These papers will give you a better idea about how can you improve quality of service.

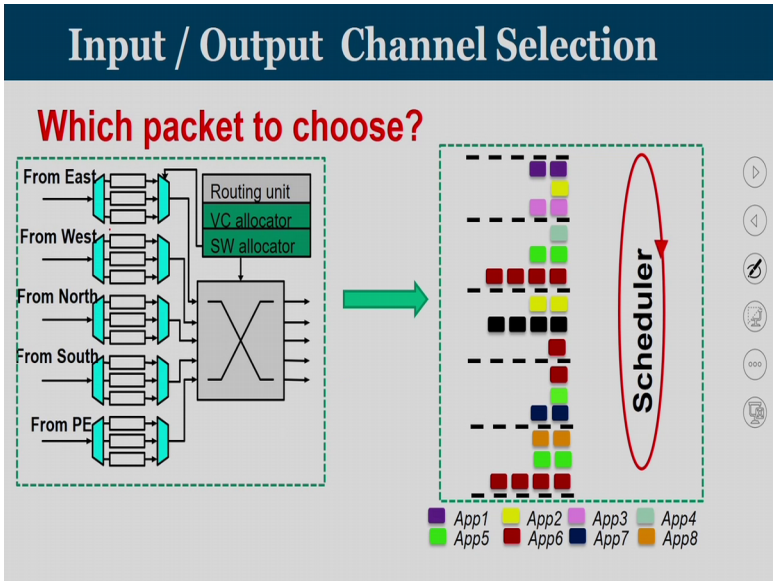
So, we will try to look into the aspect of congestion management in on-chip interconnects. So, the first one is going to be slack aware routing and second one is called packet throttling.

(Refer Slide Time: 14:07)



So, we have seen that network on-chip is a critical results, where multiple applications are going to run on this processors. And these applications will trigger packets which are essentially cache misses which will travel through the network. So, NoC is a critical resource shared by multiple applications.

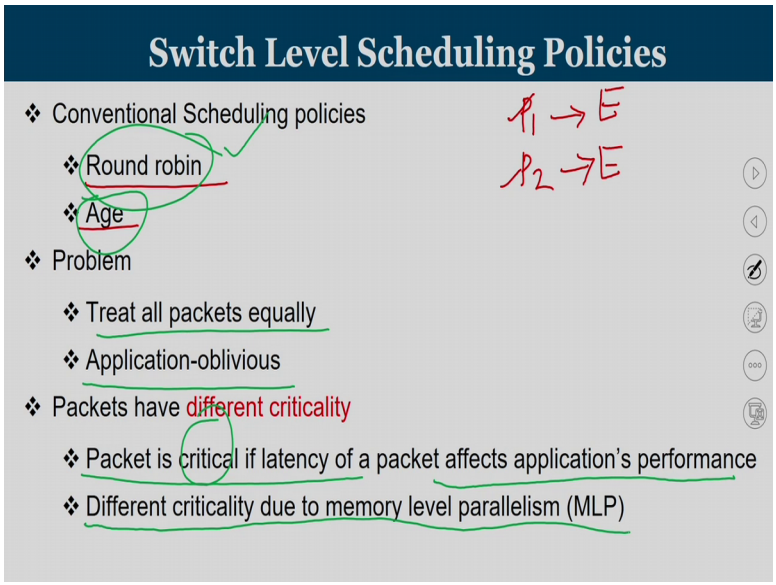
(Refer Slide Time: 14:27)



Now, we have also seen that input and output channel selection. When you have multiple packets looking for the same output port, it is the scheduler or basically the switch allocator unit inside a NoC router that takes a call, which of this packet has to be chosen.

Similarly, when a packet has multiple output ports that is available; it is the adaptive routing function and the selection function that pick one of the output port for this packet. So, in all these aspect the adaptivity is going to be brought into picture.

(Refer Slide Time: 15:01)



So, the conventional policies that we work on switch level scheduling is one is a round robin policy and the other one is age policy. So, I try to draw your attention into a scenario, consider the case that you have a packet P 1 and you have a packet P 2. Let us a both P 1 and P 2 reached the processor at the same time, and both P 1 and P 2 wanted to go to the east direction.

As per the structure of the NoC router we have discussed, it is not possible to forward two packets in the same output port. In this case, we have to pick one from P 1 or from P 2. Under this context some works have been done where in at this particular point you give preference to a split of P 1, in the next clock cycle you give preference to a split of P 2, that is called the round robin approach or the second approach is rather than giving a round robin priority, one of this packet say P 1 or P 2 is been picked based upon the age; how old these packets are there in the NoC and the oldest one is been preferred.

So, we have round robin scheduling policy and the age based scheduling policy, which are the conventional scheduling policies. Now, what are the problems associated with this scheduling policy; there are going to treat all packets equally and they are not carrying any of the property of the application that triggered these packets. So, they are

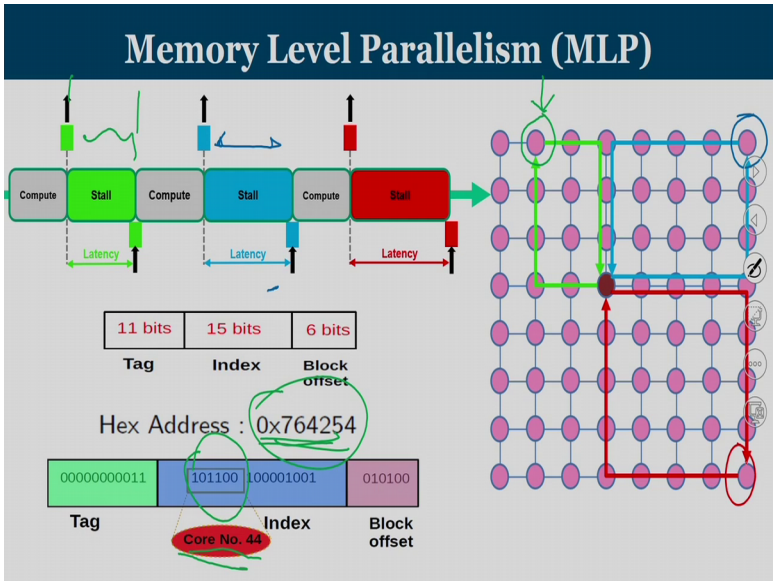
basically application oblivious, but we have to understand the fact that packets have different criticality. I would like to draw your attention to a new term known as criticality, packets have different criticality.

Now, what do you mean by critical packet? A packet is critical, if the latency of a packet affects applications performance. So, I will rephrase it a packet is set to be critical, if the latency of a packet is going to affect the application performance. So, we have a packet that is travelling through the network the packet is taking more time in the network; that means, it is having more latency. Then more latency is going to incur more performance degradation, then the packet is critical.

There may be some packet even if they take slightly more latency; it may not impact the application at all. So, in that case the packet is not critical. So, criticality is basically associated with how much sensitive the application is with respect to the arrival time of this packet. So, a packet if delayed is going to affect the application, then we called the packet as critical.

On the other case if a packets arrival at a node is not going to affect the overall stall time of the application, then we call it as a non-critical packets. And why we get criticality, why packets are having different criticality; different criticality of packets is due to the concept of memory level parallelism.

(Refer Slide Time: 18:13)



Now, we will try to understand the concept of memory level parallelism. Consider the previous split up where, we have used for the cache memory. Consider the case where we have a TCMP system that is been shown in the diagram, where we have close to 4 GB of physical memory, main memory.

And we have a cache memory with 2 power 15 sets. And this 2 power 15 sets are distributed across 64 core by the static (Refer Time: 18:39) principle. And this is the hexadecimal address we have shown. So, this address if it is generated by a core, it is going to generate a packet to core number of 44. Similarly, any change in the address will generate a packet to some other cores as well.

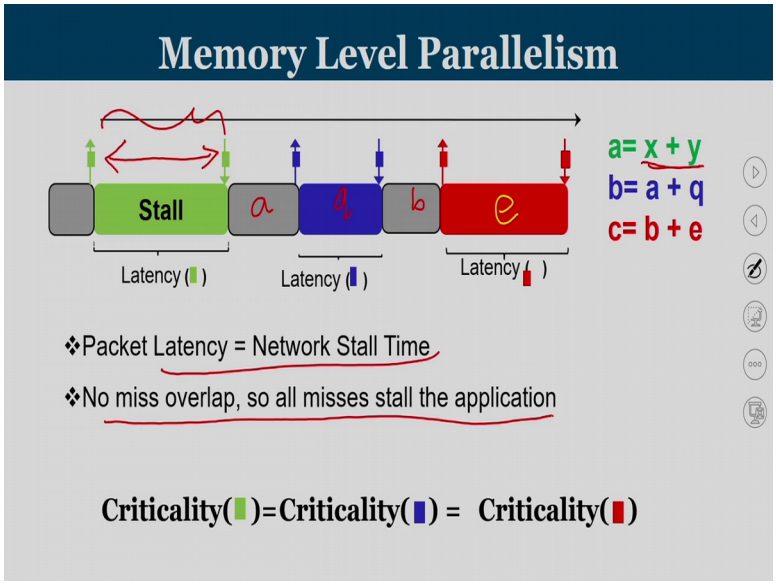
Now, we have to understand that, this core is going to generate three different cache misses. Let us say, the first miss is the green colour; you can see that the green miss is mapped on to this particular core. So, while we doing the split up it is so happen that the number in this 6 bit is corresponding to this core. So, there is a request the travel from this processing core into that particular destination, and the data is brought back from there to the source core. So, we have to understand that this much time, this is going to be the request and the reply time; a packet started with this point, is going to come back only at this point and that is called the latency of this green miss, we can called this miss as green, because that is the convention that is used here.

Now, the processor is going to work with the computation. It is going to incur a different cache miss, and that is for a blue colour. Now, we have to understand that this blue is marked on the top right corner. So, this time it is to a different destination. So, packet is going to move into this blue corner router that is been marked over there, and the packet is going to comeback. So, till it comeback there is a stalled at we are going to see from the application perspective. So, the blue packet has a latency and the application is going to stall for that much time.

Now, let us consider that there is one more miss with the red colour that is mapped, and that is going to the bottom right corner, and the packet goes all the way there and it is going to come back. So, this is a classical case where different addresses are going to create packets to different locations inside your TCMP; and some may be to nearby locations and some may be to further locations. In all these cases, the amount of stalling that the application had is more or less dependent on the latency the number of the more

number of hopes and misses travel; more is going to be the stalled of the corresponding application.

(Refer Slide Time: 21:11)



Now, consider the case that you have to look three task. And the task are been defined something like a is equal to x plus y. So, you have to consider in this case let us say, you have to process an image and the image is obtained by getting two images x and y. You are going to few some image or some task where you required a data of x as well as data of y. And then the second step b is equal to a plus q; and the third step c equal to b plus e. What is the peculiarity, if you carefully watch these three task what is the peculiarity of these three task.

The second task you can see that it is dependent on the first task and the third task is dependent on the second task. So, I cannot start processing the second task without completion of the first task. So, the first task is going to produce a result in a; only if I get a, then only I can process with the cube. Similarly, the third task can be done only after a completion of the second task. Let us try to understand these two in terms of an NoC context.

We are going to start your computation. You are not having the values of x and y with you; so you are going to generate cache misses, and this misses are going to travel through the network all the way to destination core. And then the data is been brought

back. So, by this time you got your x and y ready by which I can compute the value of a. So, a is essentially computed at this point.

You have to understand that while the cache misses in progress since the processor cannot do anything, the processor was in the waiting stage for this much amount of time. Now, after the computation phase now I have got a value of a, but this q is not available. So, there is one more miss that happens, all the way and you are going to bring the value of q and once you get it you can compute b. So, processor is going to wait as a stall until q is obtained.

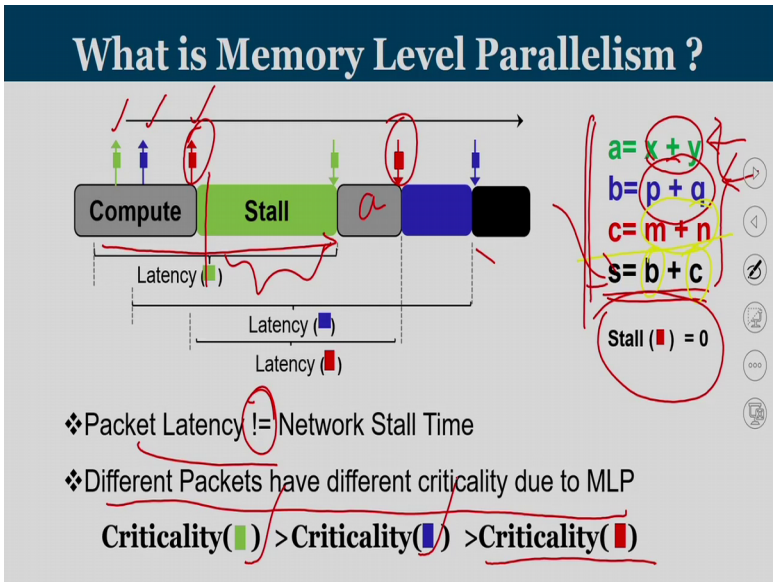
Now, again when you work on it, you do not have the e with you. So, e is going to have one more miss and based upon that you are going to create this e also here. So, you get the value of e. This is the context in which these application run. Now, which of this stall we can see that the processor is having three different stalls; one for the green miss, one for the blue miss, and one for the red miss, out of these three stalls which one is critical.

We have defined the concept of criticality before, a packet is set to be critical if it is going to affect the stall time of the application. So, if the green stall is reduced or if the packet that is going to bring the data correspond to x and y the green component; if you reduced that, stall time will reduced. Similarly if you can reduce the blue stall, again it is benefit. So, we can tell that these are the latencies of these three packets.

And packet latency is approximately equal to the network stall time. And the amount of time that the processor is stalling is roughly equal to the packet latency. So, whenever there is the packet in transit or it is there in the network the processor is stalling. Actually there is no overlap of misses, so I can tell that all these misses will stall the application or rather I can tell the criticality of green, and criticality of blue, as well as the criticality of red is more or less equal.

Every miss what we have seen now all the three misses are critical, because any delay in servicing these misses is going to impact the application, the application is going to get stalled.

(Refer Slide Time: 25:11)



Now, let us revisit the example with a different set of a task. I request you to kindly go through these 3 or 4 task rather 4 task that is given a equal to x plus y; b equal to p plus q; and c equal to m plus n; and s is equal to b plus c. What is a peculiarity, when compare to the previous task that is been already mentioned. We know that the first three task are mutually independent; the green line, the blue line, as well as the red line can be carried out parallely, but the black line the forth one can be done or can be processed only if the second and the third values are available.

So, in these case if it is out of order super scalar processor, whenever you do not have the value of x and y, you just look into the adjustment line x and y. So, proper steps are initiated to bring x and y, it is a non-blocking cache. I hope you are familiar with what is the terminology called non-blocking cache; whenever there is a miss in progress, the processor can still service hits or there can be more number of misses, and this is been handled by miss service holding registers.

So, when the processor has generated a request for bringing x and y, it can look to the adjacent instruction; and p and q also, you can go through and p and q is also a miss it is not there with you, so one more request is generated to bring p and q. And similarly you can go and find out m and n as well. And then you reach a point, where further we cannot do, because without getting b and c we cannot process. So, this is the case that has been depicted here during the computation phase you are incurring with the miss, because you

are using out of order super scalar processor, you could look into the adjacent instruction and it is a non-blocking cache. So, you have had a green miss and a blue miss and a red miss.

So, whenever you have a green miss, processor is not stopping. Processor is going to work on adjacent independent instruction that can be fetched and executed, since it is a non-blocking cache, a cache miss service is in progress. Again you incur one more miss; it is a non-blocking cache, so caches can still work on it. And then you reach the point, where after the third miss we cannot find out any other instruction that can be operated at this point and then you are going to stalled.

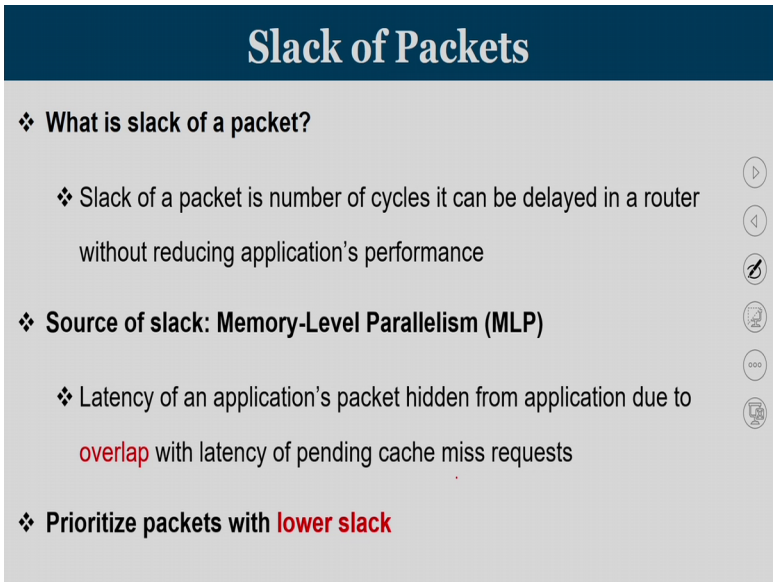
So, the first stalling over once you get back the green and then due to some kind of a computation that is there. So, the value of a is computed essentially there, and then you got the red packet back, and then still you are waiting for this blue miss and then you do the last system ,this is basically the last step. Luckily we can see that there is no red bars in this, the processor is not waiting for the misses that are triggered by the red packet. Why it has happened, there was a miss that occurred the red packet came back. So, now we can see that the latency of green, the latency of blue, as well as the latency of the red.

Here there is no stall with respect to the red packet. So, packet latency is not equal to network stall time, there are so we can see that even though the green has started at this point, this is going to be the latency of the green packet, but processor is waiting only in this much time for the green packet, the remaining time the processor is still in the computation phase. So, just because you have a cache miss processor need not be in the stalled condition, processor may be still working on some other data which is independent of the missed data that is what we can see that the packet latency component is not equal to network stall time when we have non-blocking cache and out of order super scalar processor.

This means that different packets have different criticality due to memory level parallelism, just because the cache misses are overlapped; we can see that the cache misses of green, blue and red are overlapped. And red is completely contained inside blue, red is coming back to the processor may be the red is mapped to a nearby tile that is why the miss to the tile is going to return faster. So, red is coming little bit early, red is not at all critical. So, in that case we can see that the red is less critical whereas, green

and blue are more critical than the red. So, we were trying to understand the concept of memory level parallelism from the perspective of overlapping cache misses and this can happen only if the processor is super scalar, out of order super scalar processor, at the same time using non-blocking caches.

(Refer Slide Time: 29:57)



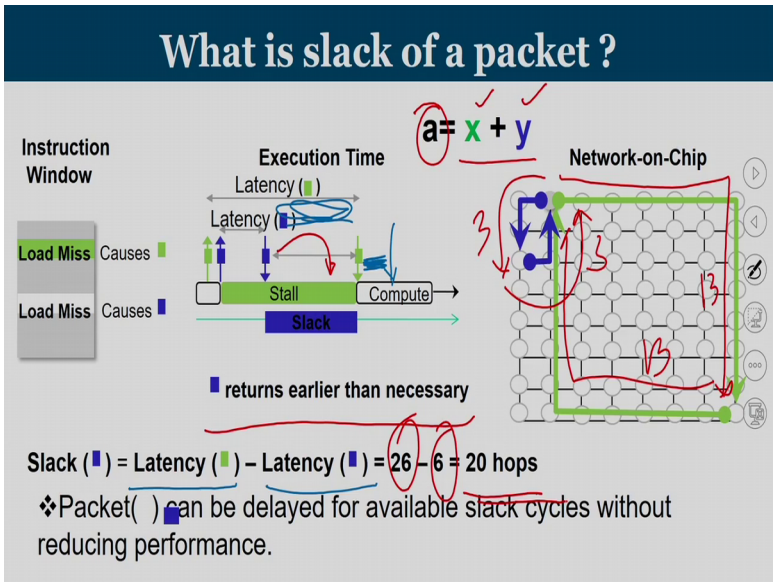
Now, we try to understand the concept of a slack of a packet. So, slack of a packet is defined as the number of cycles a packet can be delayed inside a router without reducing the application performance. So, what is a source of slack; it is basically memory level parallelism. So, latency of an applications packet is hidden from the application due to overlap with latency of pending cache miss request. So, what we have to do, essentially it is called prioritization of lower slack.

So, to reemphasis the concept we are trying to introduce a new concept called the slack of a packet. Slack of a packet is defined as the number of cycles a packet can be delayed in the network without causing applications performance. So, we have slack for those packets which are critical say if a packet is critical, I cannot delay it. So, critical packet has a slack 0, whereas non-critical packet may have some slack. And memory level parallelism is a source for having slack. So, some packets may return early than necessary.

So, what we are trying to do is rather than working for the conventional switch scheduling policy is, like age based switch scheduling or round robin based switch

scheduling, it is always better to go and exploit to the concept of slack. So, if the slack value is recorded inside a packet, whenever two packets come together we have to prioritized that packet with a lower slack.

(Refer Slide Time: 31:25)



So, trying to understand the concept of a equal to x plus y, again this is the instruction window that is been shown there. Let us say your computation phase and there is a load miss that happens and it is to this further core that is been shown. And then we are going to have the second miss. So, one for the green value x and second one is the blue value y. So, y is to the nearby core and the y is going to return early and then comes green.

So, we can see that you carry out the task a you require both the values of x and y; x comes a bit late, you incurred a miss on the value x as well as on y; x came a bit late, because the address in which x is mapped is very far with respect to the source core under consideration. And the address to which y is mapped is rather very close to the tile under consideration. So, you have x plus y and then we are going to compute. So, this is the latency of the blue packet as well as the latency of the green packet.

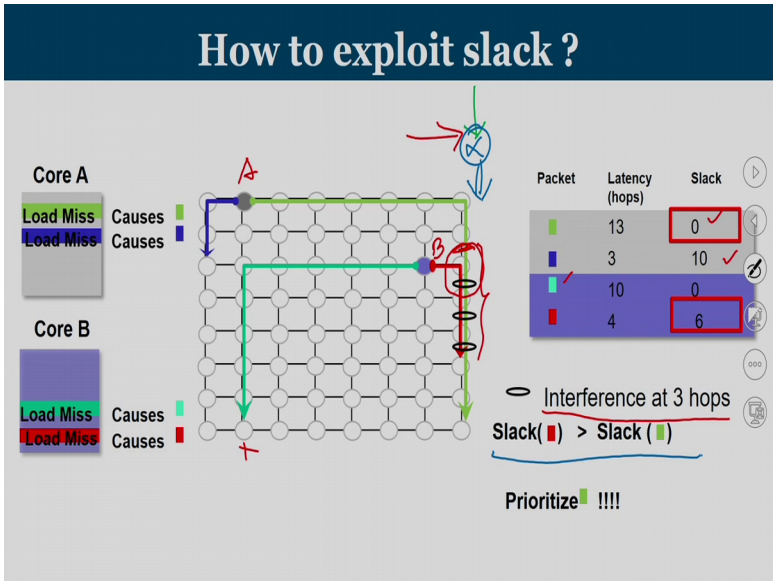
And we know that the blue packet returns earlier than necessary; there is no problem even if this blue packet is going to return at this point. There is no stall that this going to happen, because the application is going to stall for the green packet. If the blue packet is going to returned at this point, then there is going to be a blue stalling that happens. So, as long as the blue packet is going to returned somewhere in this region, it is not going to

affect the performance; that means, blue packets can be delayed that is what it known as the slack of blue.

So, the slack of blue packet is defined as the latency of the green packet minus the latency of blue packet. Now, in this context let me define the latency in terms of hops; you can see that the green packet has to travel all the way, 13 hops in one direction and 13 hops in another direction, thereby making total of 26 hops the green packet has to travel in terms of request as well as reply, but the blue has to travel only 3 hops onward and 3 hops returned, so that makes only 6 hops that it is going to travelled.

So, the green packet has to travel 26 hops whereas, the blue packet will travel only 6 hops leading to a slack of 20 hops that is available with the blue. Meaning the blue packet can be delayed to a time equal to that of 20 hops, this is because blue has a predecessor that is travelling to further distance in the TCMP. So, packet blue can be delayed for available slack cycles without reducing the performance that is the way how we look this things up.

(Refer Slide Time: 34:05)



Now, how are we going to exploit the slack? Consider now two applications, core A as well as core B; so this is the place where A is running and this is the place where B is running. Let us say A is going to generate two misses the blue and the green; and we can see that where is blue mapped and where is green mapped. Similarly, the second one is

going to have a red miss as well as the lighter green miss and this is also going to two similar places.

And then if you look at this slack of these packets, the green packets does not have a slack, because it is not having a predecessor. So, slack of green is 0 whereas, slack of blue is that it is going to be 10 because the predecessor for blue green is travelling longer. So, blue can be delayed by 10 hops when you come to this grey, it is not having a slack, because that is travelling to a longer distance whereas the red is travelling shorter distance, so red has slack.

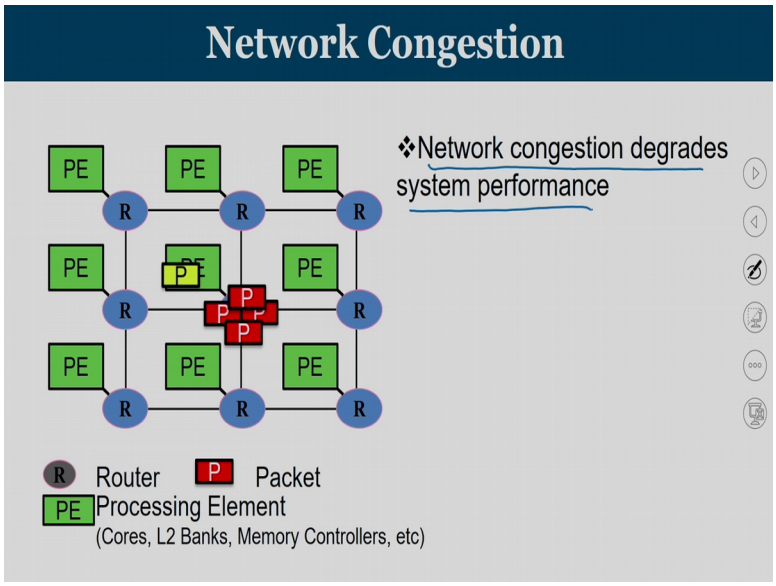
Now, we can understand that this are the points where the both the red flit as well as the green flit are going to interfere each other. So, these two packets are going to interfere at 3 hops. So, when you have a green packet coming from the north looking for the south output port, and you have the red packet that is coming from the west looking for the south output port, these things happen in this router.

So, you have a green packet that is going to come from the north and looking for south, and you have a red packet that is coming from the west and a both are looking for the south output port, both are looking for the south output port. So, the conventional policy say, it can be based on age or it can based on round robin.

And now what we are going to look is, we are going to look at the slack of the packets. If the slack of the packet is already recorded, then you can see that the slack of red is greater than that of slack of the green packet, so that means green can have priority, because delaying green is going to affect the application, whereas delaying red will not affect the application, because red is having a slack.

So, the whole idea that we learned just now is trying to represent this slack of a packet inside a packet, and when the packet is travelling through various intermediate routers, everywhere if the slack based prioritised routing can be done, this is known as slack aware routing for applications.

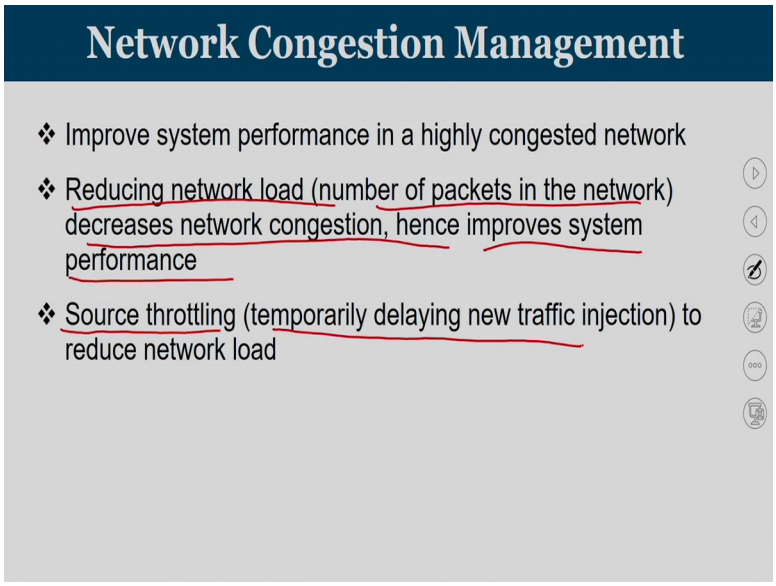
(Refer Slide Time: 36:41)



Now, let us try to understand another problem of network congestion, and how so research work proposing this domain, how address this issue. Try to see what is network congestion here, we have packets, that is starting from various processing elements and all of them are coming to one point. So, there are too many packets in the router buffer leading to a scenario were the locally created packets. You see the yellow packet cannot make any progress; yellow packet cannot make any progress, because injection is suppressed.

So, naturally scenario is like this where packets cannot be injected into the network due to the heavy traffic in the network, it is going to degrade problem. So, network congestion degrades a system performance that is what a take.

(Refer Slide Time: 37:29)

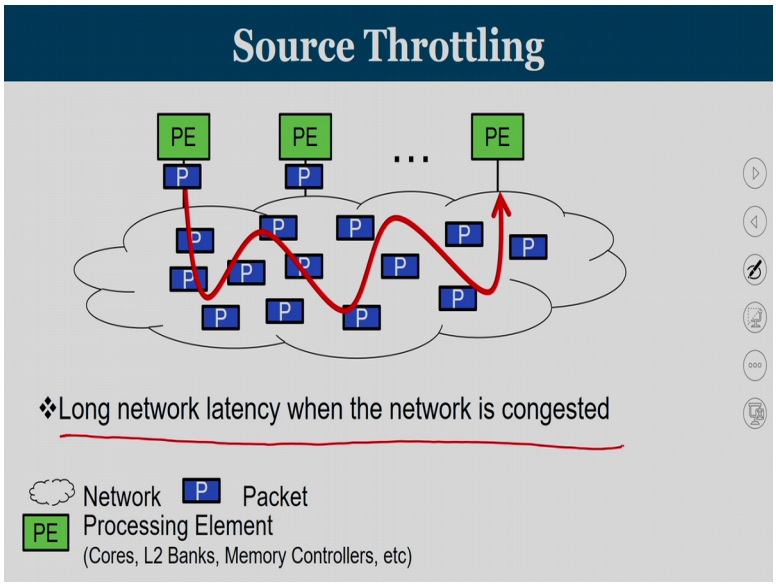


Now, network congestion management how can you manage. So, how will you improve system performance in a highly congested network? That is a goal. So, this can be done with a concept called reducing the network load, how can you reduce network load, what is load in the network; when a packet is coming to network that is called a load.

So, how will you reduce the load, we have to reduce the number of packets, but packets are essentially triggered by cache misses, we cannot reduce cache miss just by working on the network. So, what we are trying to do is the number of packets in the network decreases network congestion, and hence improve performance. So, based upon that the work is already being proposed by so research groups and one of that one is called source throttling, we are trying to understand what is called the concept of source throttling.

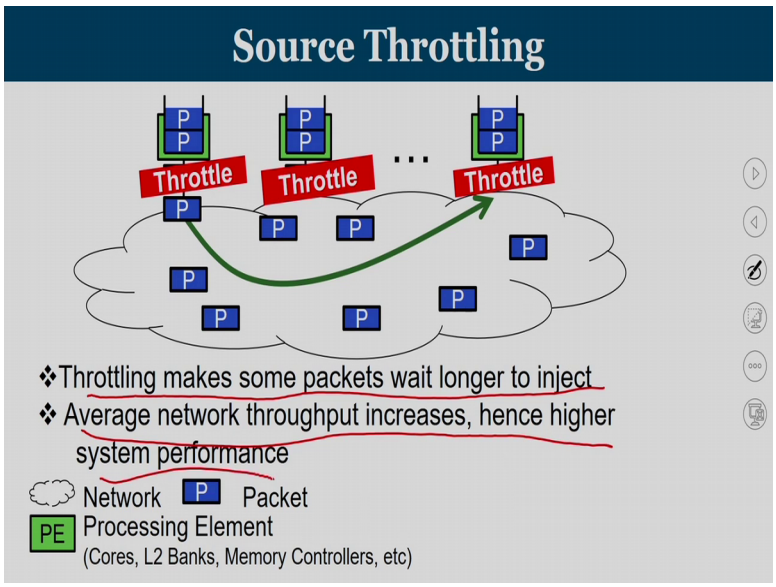
Source throttling is a technique where packets are temporarily delayed, it is only a temporarily delay to injection. So, let us say you are going to injected clock cycle number 10, I would not permit the packet to be injected at clock cycle number 10, it is slightly delay to clock cycle number say 12 or 14 or 20. So, the whole idea of managing congestion by packet throttling is some of the applications are been carefully chosen based upon certain characteristic of the application and delaying those applications in injecting packets into the network.

(Refer Slide Time: 38:51)



So, consider the scenario you have lot of processing elements, and this processing elements are going to generate packets when they encounter cache misses. And let us say, there are lot of packets in the network, this is a logical view of the underlined NoC. So, when there are too many packets in the network, it is going to have long network latency when the packet is congested.

(Refer Slide Time: 39:21)



So, what we can do is we are going to throttle, throttle means do not allow all the packets to enter into the network. So, once you apply the throttling technique packets are getting

cued up, we can see the packets are getting cued up in the processing elements. And the number of packets in the network substantially reduced that gives to a slightly easy path.

So, throttling makes some packets wait longer to inject. Average network throughput increases, hence it will give to higher performance. So, whenever there are lot of packets in the network, we cannot move through the network. So, when some of the packets are we delayed, then that will lead to less amount of traffic in the network leading to better performance.

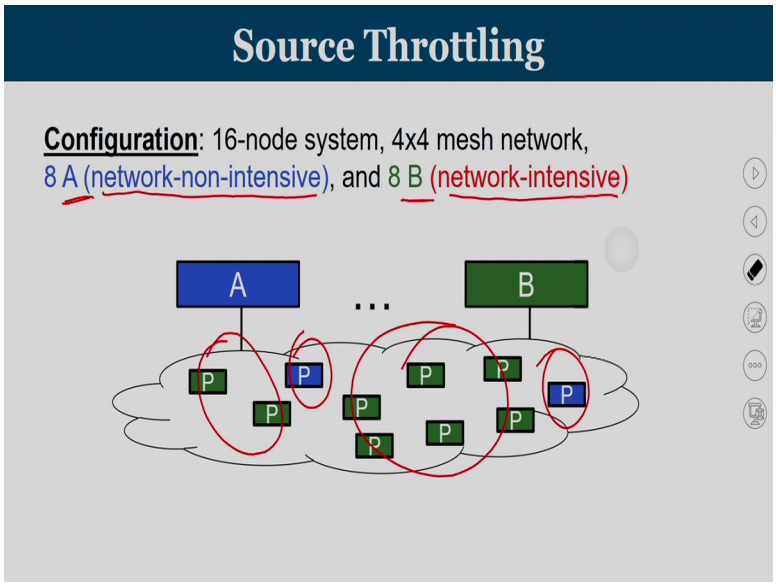
So, I would like to draw your attention to a common life example. Let us say you are living in a city, and you know that evening say 3 o clock the schools school time ends, and lot of school vehicles are going to come out from the school. So, in and around a school in a city there is going to be some heavy congestion. So, the number of vehicles in the road is relatively high, during the closing time of a school; lot of school vans and parents who how come to pick up the children are all going to be there in the road.

Now, you just imagine a scenario you are a third person, you are passing through nearby place of the school. So, when you travel around 3 o clock, it may be you may take 1 hour to reach the destination rather than travelling at 3 o clock, now let us say you travel at 3.30 that is the time when you pass through the school, nearby vicinity of the school; you may not experience that much congestion, so you could reach the destination in 30 minutes.

So, when you travel at 3 o clock, you make take 1 hour to reach the destination; but when you travel at 3.30, you may reach only 30 minutes to reach a destination. Why this context, the particular time there were too much packets in the network. So, if you also go there, you may also get delayed.

So, the policy that we rather than going into the traffic at some point, we are delaying them to certain clock cycles, that is essentially the idea of throttling.

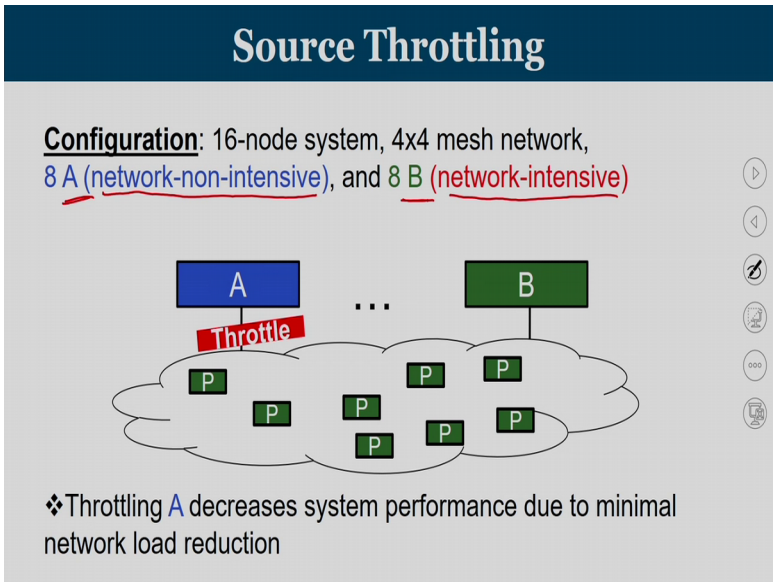
(Refer Slide Time: 41:41)



Now, let us try to work on some example. So, consider you have a 4 by 4 mesh network, which is going to have 16 application. So, when you have 4 by 4 mesh network, you have talking about a 16 tile TCMP. Let us say out of the 16 tiles in 8 of the tiles an application A is running, and in 8 of the tiles application B is running. The peculiarity of A is, A is network non-intensive, so A is a light application; and B is a network-intensive application or B is going to be a heavy application. So, this is a scenario you have when application A and application B. A is going to be a network non-incentive light application.

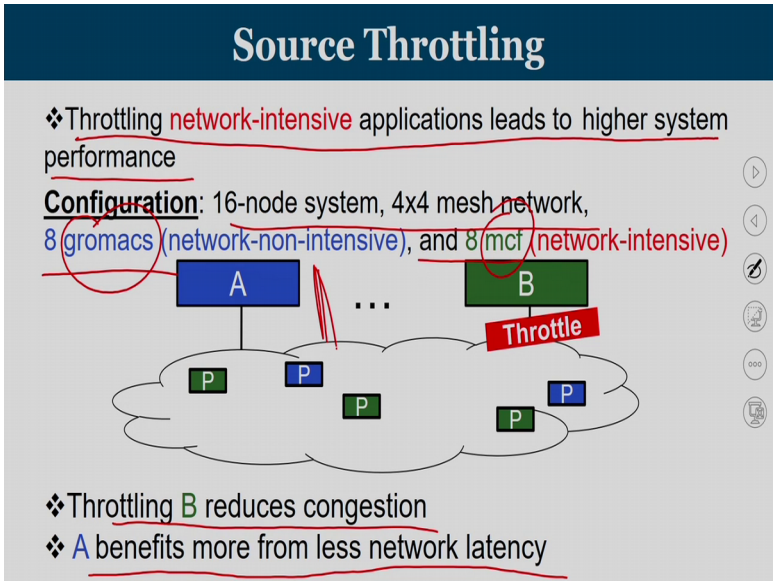
So, you can see that the number of packets by A is rather less, where as the number of packets by B is going to be relatively on the higher side.

(Refer Slide Time: 42:39)



Now, if you throttle A, then hardly very few packets are reduce in the network. It is not going to affect the performance of A, rather since the most important packets of A are been throttled, you may degrade the system performance. So, throttling A decreases system performance due to minimal network load reduction, rather than that now throttling of B is just considered I am going to throttle B, then there is sufficient number of packet reduction.

(Refer Slide Time: 43:07)

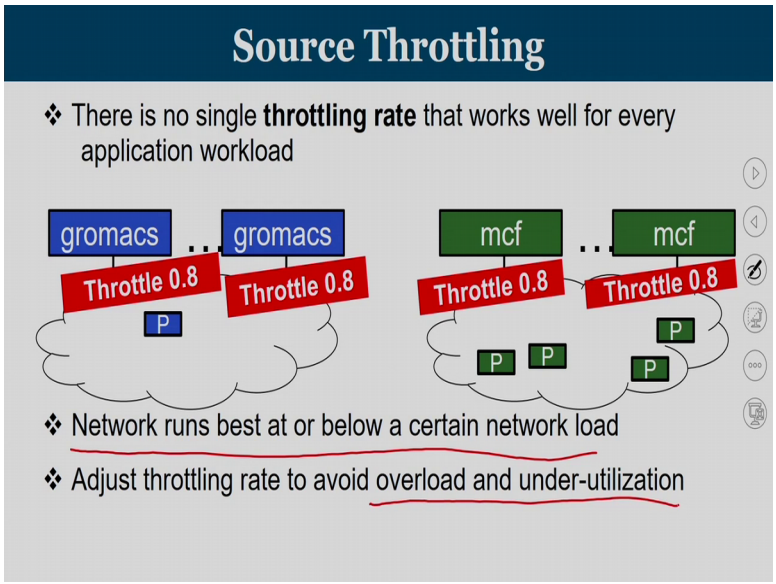


So, throttling B increases system performance due to the reduced congestion. So, the take away is throttling a network-intensive application lead to higher system performance. So, when you have multiple applications running on a TCMP, it is not like arbitrarily we pick one of the tile and then trying to throttle. Essentially you have to find out, which is a network-intensive application that application which pumping too many packets in to the network and throttle them.

So, which application to throttle is really an important question that we have to answer. So, consider this configuration we have a 16 node system, where 8 of them are network non-intensive and 8 of them are network-intensive. So, this is an example of a network non-intensive application called gromacs, it is taken from spec benchmark; this is another example of an application which is called the MCF that is a network-intensive application.

So, throttling B is going to reduce congestion, and A gets benefit more from getting the so the network non-intensive application. So, once in a while only, once in a while only they are going to generate packets and these packets will get a easier role, because B is throttling it is not A that gets throttled.

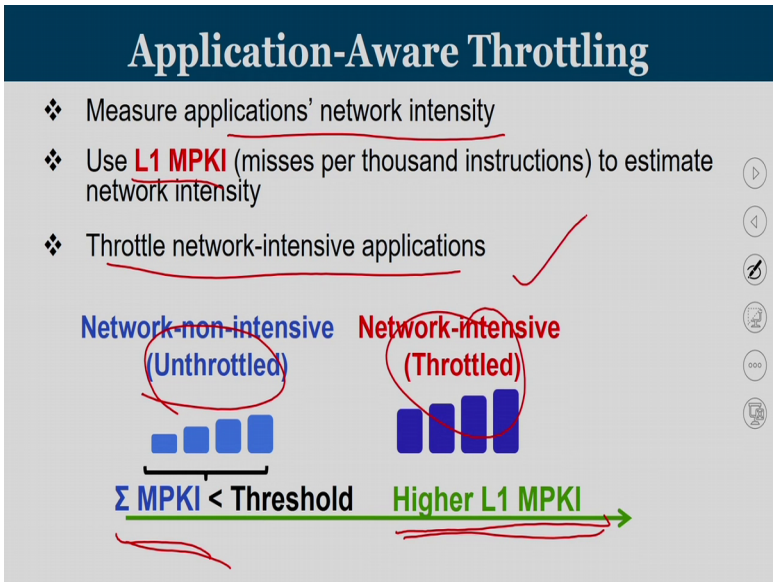
(Refer Slide Time: 44:35)



So, there is no single throttling rate that works well for every application workload. So, you have many applications like this, like what I have mentioned from the spec benchmark if you throttle one of them, you get less benefit whereas, if you throttle the

other, you are going to get more benefit. So, network runs best at or below a certain network load. So, we have to adjust the throttling rate or how much to throttle to avoid overload and under-utilization scenario.

(Refer Slide Time: 45:07)



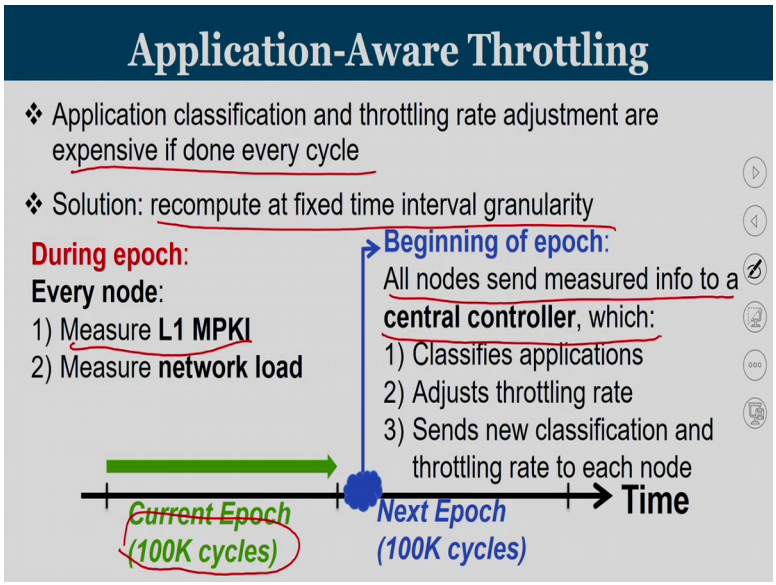
So, what do you mean by application aware throttling, because every application has their own network-intensiveness, which we cannot define early. During different phases of execution, certain applications may be network-intensive for the first one slack cycles; and may not be network-intensive for the next one slack cycles. So, we have to continuously monitor these applications, and try to come up with the parameter which will define whether the application should be throttled or not.

So, we have to measure the network intensity of the application and one such parameter is called MPKI, it is called Misses Per Kilo Instructions. So, use L 1 MPKI Misses Per Kilo Instruction to estimate the network intensity and throttle network-intensive application, so how are you going to compute. So, essentially we have to find out number of cache misses that each of the tile is going to generate, record it may be in a hardware counter and try to exchange this values to some central controller.

So, every tile may be associated with some counters, whenever there is a packet that is generated into the network this counters are updated. And once in a while a regular intervals this counter values are been collected, and try to find out which of these tile is generating more number of misses; take up some conclusion and reported back to them.

So, applications are non-intensive meaning if the MPKI of the application is less than a threshold, and applications are called as network-intensive, if the MPKI is larger than a threshold. So, the non-intensive applications are not throttled whereas network-intensive applications are throttled.

(Refer Slide Time: 46:47)



So, how will you perform application aware throttling? We have to classify applications so application classification and throttling rate adjustments are expensive, if it is done on every cycle. So, we cannot look for whether every core is injecting more or less it all the cycles. So, we need to define a time window during the time period the network is trying to learn what happens, we are trying to analyse what happens in the network by setup; by virtue of setting up some extra hardware units.

And for a reasonable time window, once we understand the behaviour of network then initiate appropriate corrections mechanism. So, solution is you recompute at fixed time interval granularity what is going to be. So, what happens for example, in this research work the authors have found that you watch for a period of 100 K clock cycles, so during the epoch.

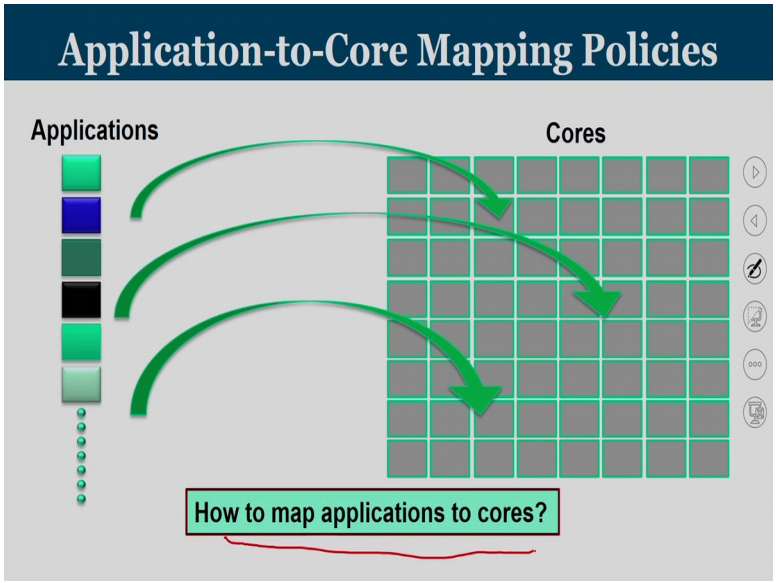
Every node is going to measure this L 1 MPKI, and it is also going to measure the network load at the end of this say 100 K clock cycles all nodes send the measured information to a central controller, which classify the application based upon L 1 MPKI; and find out what should be the rate of throttling, will say a tile 1 it is pumping so many

packets, so throttle at 0.8; tile 2 it is not generating much packets, so tile 2 is not throttled; tile 3 it is also pumping reasonable amount of packets may be throttling at 0.4.

So, if you have 64 tiles or if you have 128 tiles, this controller is going to get the amount of traffic all these tiles are going to generate into the network, and this traffic is studied come up with the threshold parameter; those who are generating packets above this threshold, these tiles are been informed back by some control messages to reduce the injection. So, these are the cores which are getting throttled; the other cores are not throttled.

So, it is application based throttling; some of the tiles are throttled, some of the tiles are not throttled, so that is the way how the whole concept of application aware throttling is going to work. Now, we have learned about two concept one is based on applications property you route, second one based on applications network-intensiveness, you prohibit them or denied them injection into the network. So, slack away routing is what we discussed once and second one is packet throttling.

(Refer Slide Time: 49:25)

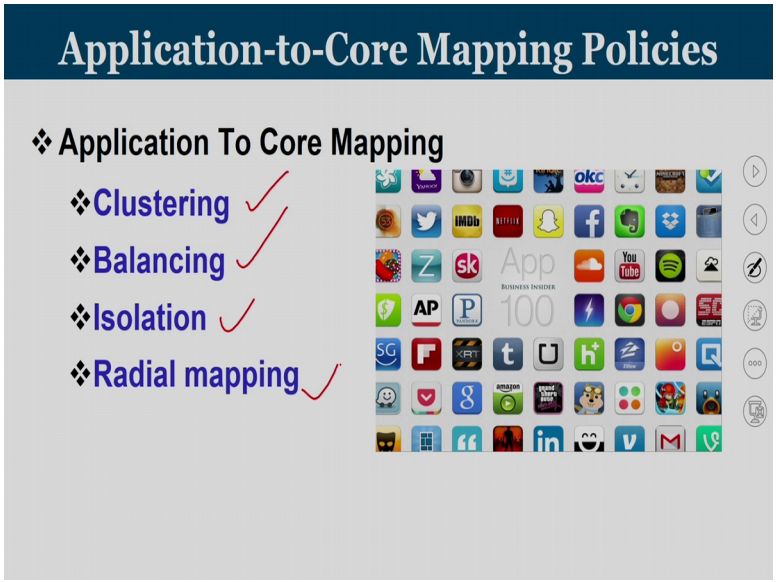


Now, let us move into another area which is called application-to-core mapping. We know that we have lot of applications that we used on a day to day basis, and we are going to work on smart phones or tablet us which are having many cores. Now, the question is which application is going to mapped to which core; daily we work on applications like WhatsApp, your Facebook, your news feed, your movie playing, your

calculators, your alarm, all these are going to be different apps that are going to run on a tiled chip multi-core system.

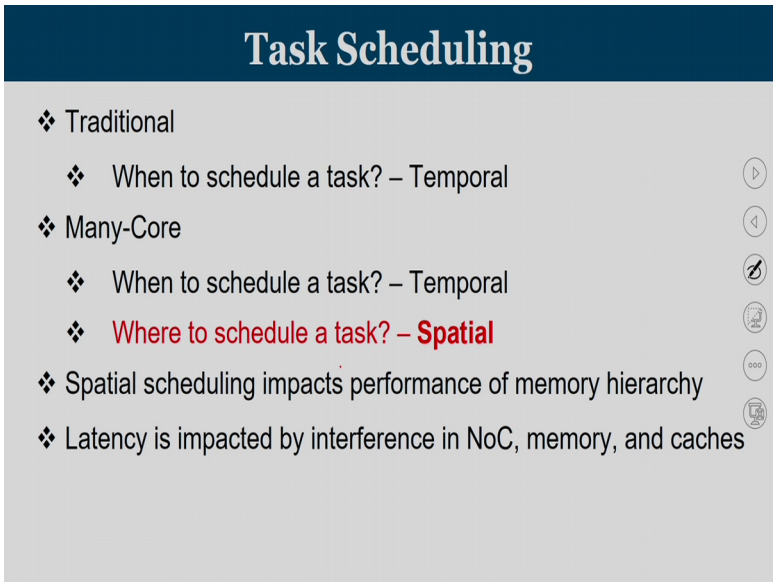
Now, where will my WhatsApp application run, will my WhatsApp application run on core number 1 or core number 2 or core number 100, where will my Google chrome application will run, where will my banking application will run. So, it is now we are trying to look from a slightly upper level perspective, the job that is done by an operating system in finding out appropriate core for an application to run; And that is been termed as application-to-core mapping policies. So, how to map applications to cores.

(Refer Slide Time: 50:29)



Application-to-core mapping to a we are going to discuss about four different application to core mapping techniques; one is called clustering, other one is balancing, third one is isolation, and the forth one is radial mapping, this is also taken from the literature from the recent published works.

(Refer Slide Time: 50:49)

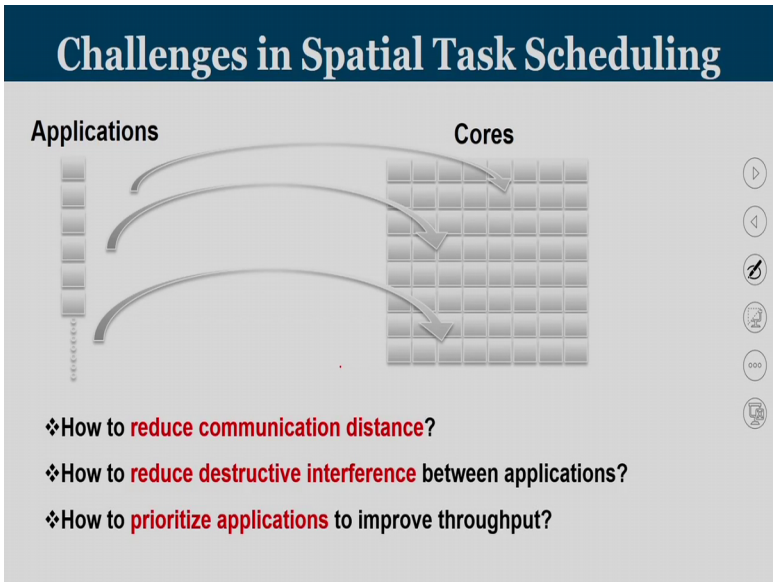


So, what about task scheduling? So, in operating system task scheduling is a very common term, which tells when you have multiple task and when you have one processor, it will basically tell when to schedule a task. I will repeat once again, when you have many task or many process P 1, P 2, P 3, P 4 and I have only one processor which process will run on the processor that is called temporal scheduling, when to schedule a task?

When you move into multi-core systems, like our chip multi-core system TCMP, we have now two design issues, apart from temporal scheduling when to schedule a task? You have one more problem to solve like, where to schedule a task? And that is called spatial scheduling. So, spatial scheduling impacts the performance of memory hierarchy.

And the latency of an application is depended on spatial scheduling. If an application runs on core number 2, let us say it is generating a miss all the way to core number 60; it has to travel a lot. So, the latency of the packet is very high, rather than scheduling the application on core number 2; had it been scheduled at core number 60; then 60 to 63 is a shorter distance. So, the latency of the same application is going to vary, because the application itself is scheduled on a different core.

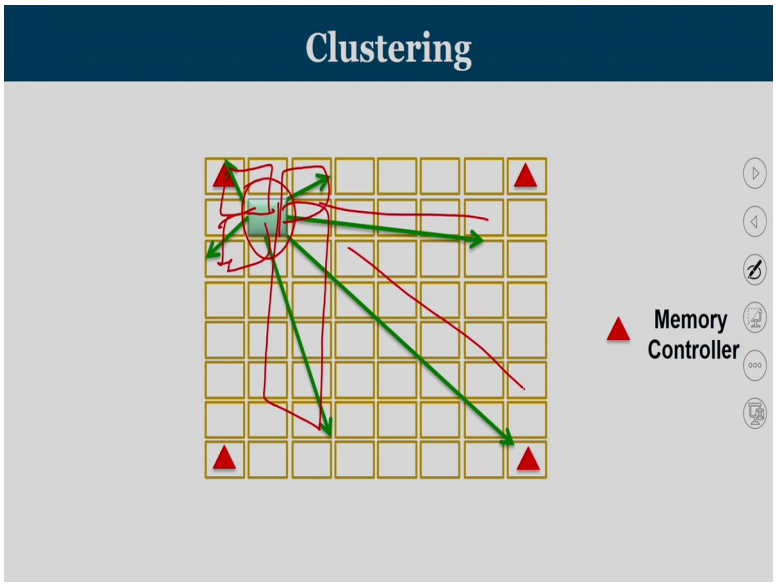
(Refer Slide Time: 52:16)



So, what are the challenges in spatial task scheduling? First we have to find out, how a scheduling is going to impact the communication distance, how it will reduce communication distance? A tile is communicating with some other tile in forms of packets, sometimes if you carefully scheduled, then the communication distance can be reduced.

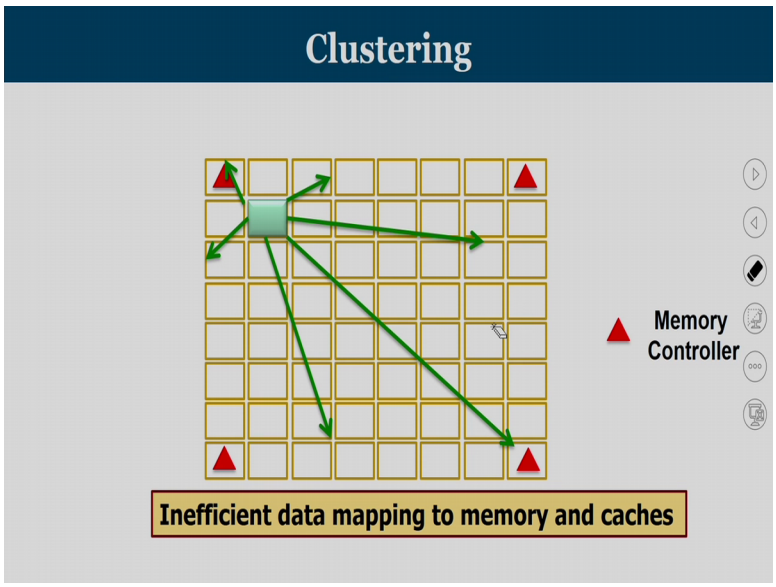
How to reduce destructive interference between applications? One application can impact with some other applications packet or packets from too different applications may reach at the same junction, and both may be wanting to do for the same output port. So, one is going to stop the other, how that can be reduced. And third one, how to prioritize applications to improve the throughput?

(Refer Slide Time: 52:59)



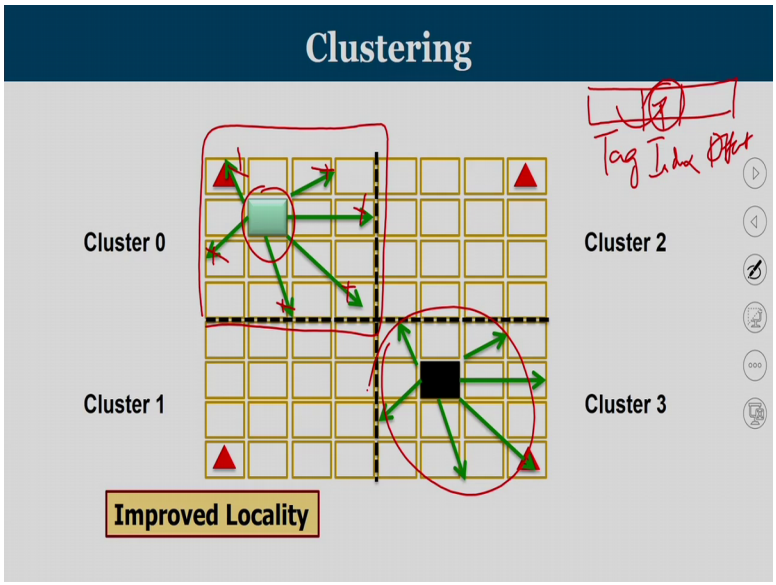
So, let us try to learn one by one if the first application to core mapping is called the clustering. Consider a TCMP where we can see that the memory controllers are kept in the corner tiles, let us say there is an application that runs here. Now, this application is going to have a memory foot print, which are going to incur L 2 misses to this these cores. So, this application is going to generate packet into these cores, and some of these packets are coming like this. So, you are getting a wide range of cores to which these memory mappings are done.

(Refer Slide Time: 53:37)



So, this is an inefficient data mapping to memories and caches, because this applications packet will travel all the way to network; and it is going to affect the performance of other applications as well.

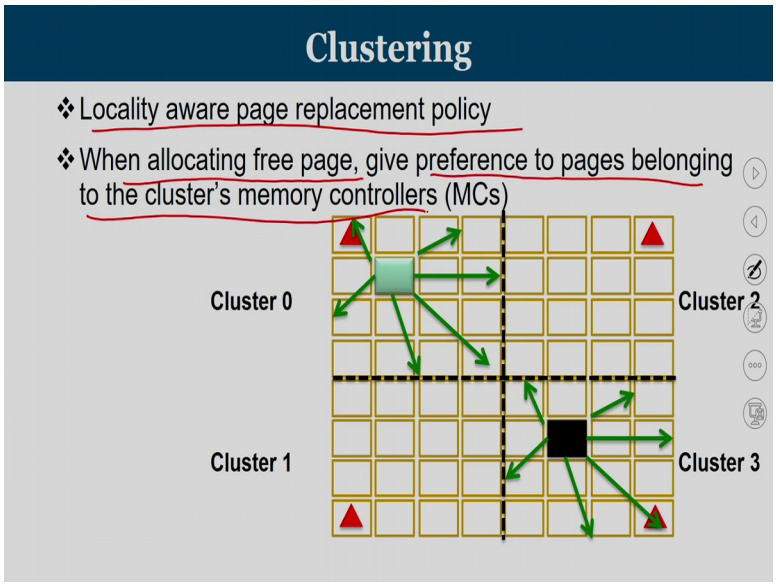
(Refer Slide Time: 53:51)



One way of mitigating this is, you are going to divide the network into multiple clusters. And make sure that the physical address assigned to this applications data is organized in such a way that they are mapping only to that cluster. So, we know what are the tile numbers for this cluster and from the address that we have a tag portion, we have an index portion, and we have an offset portion. So, from the address that we have if you carefully give indexes, then you can make sure that once they are coming into the tiles; they are mapped to nearby locations or into the same cluster.

This actually reduces the latency of the misses generated by the application, and reduces its interference with packets generated by other applications from other clusters also. This is improved your locality, so there will be some other application we just strictly mapped into that core. So, operating system place or your android, that is going to run on the cell phones. So, this operating system is going to run is have a very very crucial role are important role that governs the performance.

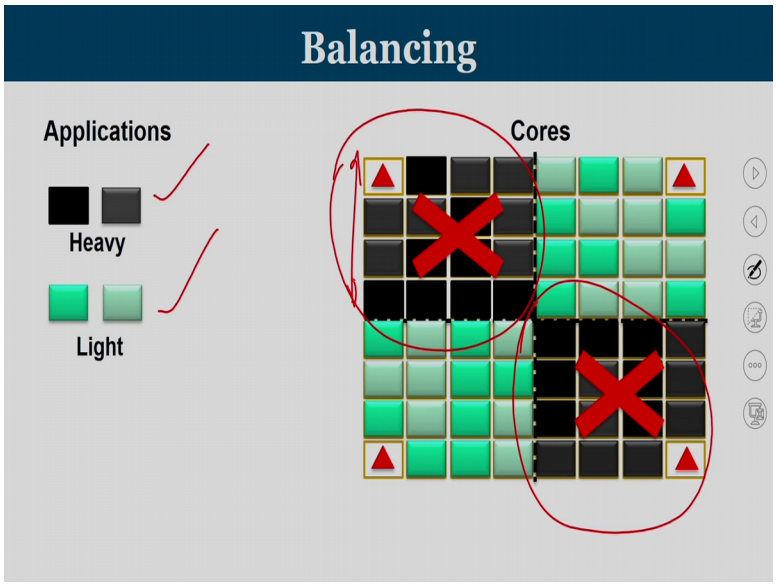
(Refer Slide Time: 55:11)



So, where you schedule a task is very important, so it will reduce the interference. How it is possible? It is locality aware page replacement policy, so when you bring a data from the secondary memory and put it in a main memory you are going to assign an address. The moment we give a space to an instruction or a data inside, the main memory or inside your DRAM its physical address is fixed. And it is the portion of the physical address is the index, you know that the physical address is split into tag, index and offset.

So, certain bits in the physical address is coming as your index, and in a TCMP certain bits of your set index is nothing but your destination core. So, destination core should be given those addresses carefully such that it result in locality. So, when we allocate free page, give preference to pages belonging to the clusters memory controllers only, these are the way how clustering can be done.

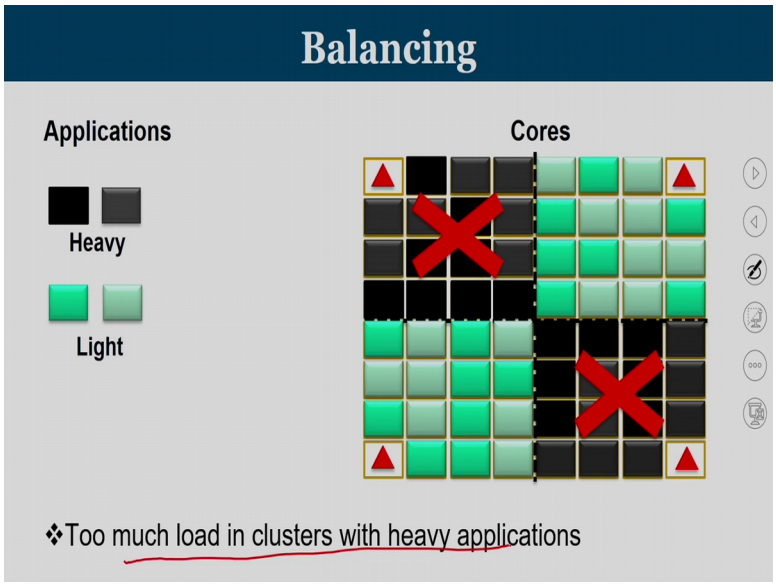
(Refer Slide Time: 56:03)



Now, other mapping technique is called balancing. We have different application some applications are heavy, this is very heavy, this is still heavy and some applications are light, this is based upon the number of packets, these applications are going to inject into the network. Now, consider the case that if you map the applications like this; all the heavy applications are in one core; and all the lighter applications are in other core, essentially we can see that this particular core is mapped with the applications, which are relatively very heavy.

Consider the case that you are going to work with your WhatsApp application and a video player. Let us say these are the very heavy used application in your cell phone. If these two applications are mapped in one location maybe in one cluster, and the other clusters are relatively of less use; then the activity factor in the cluster is going to be very high. So, when you have heavy applications mapped in one of the clusters, the amount of network activity, the amount of cache activity, the amount of work done by the processor is heavy, it lead to uneven where and there physical activity is very high there; so that is not a good approach.

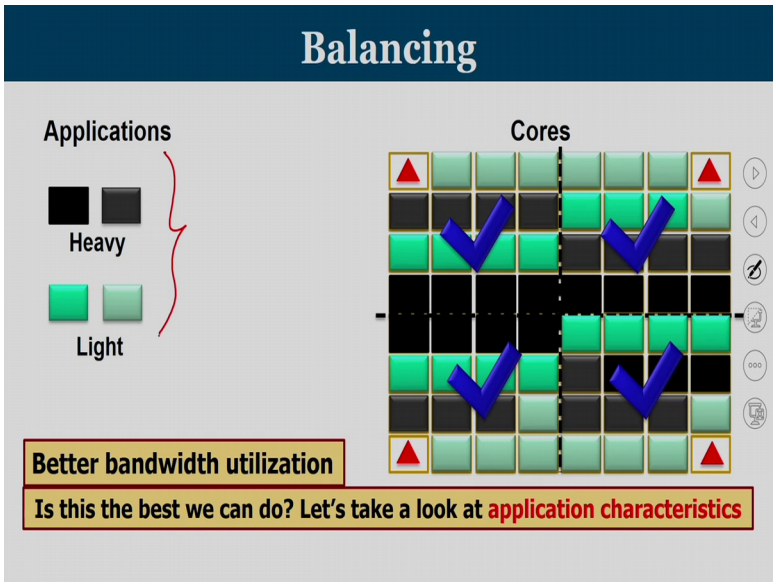
(Refer Slide Time: 57:27)



So, the best approach in this context can be we have to come up with solutions in such a way that too much load on a cluster with heavy applications. This will lead to early dying of the chips, because certain portions of the chips are very heavily used. It is as good as like our normal traffic roads, certain roads where there is heavy traffic which is prompt more where and there and that can lead to damage of the roads.

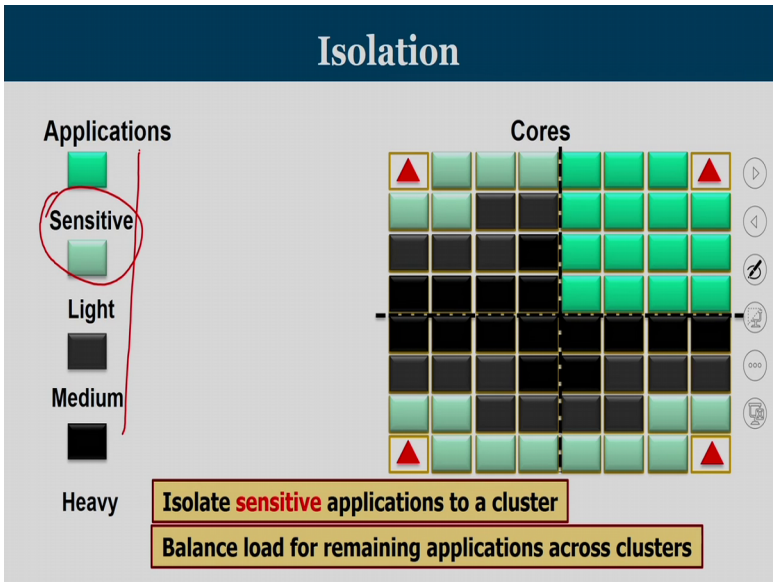
Whereas if certain roads are not heavily used, there may be having less damage. So, in order to increase the life's pair of chips, it is always better to map applications to this chip in such a way that every area of the chip is been fairly used.

(Refer Slide Time: 58:03)



So, how balancing can be done, you should have applications in all the cluster such that we have a right mix of heavy as well as light applications. So, this is called better bandwidth utilization. Now, can we improve upon this further?

(Refer Slide Time: 58:21)

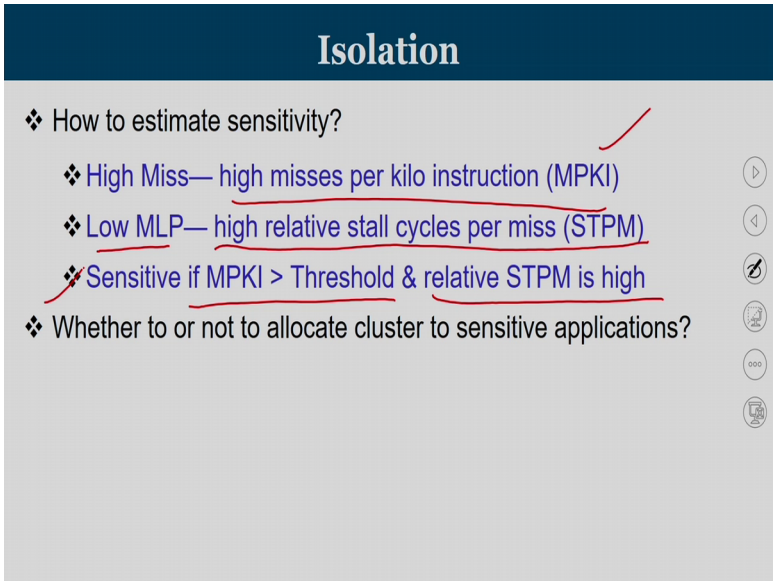


So, there is another operation that is called sensitive. Certain applications are going to be sensitive, sensitive means this are the applications, which will generate misses only once in a while, and this misses are very very important; this misses need to be service very fast.

So, they do not want to meet with the traffic of others like for example, when VVIP so ministers are travelling through our road, they do not want to wait for others, they do not want to get their work is delayed. So, the convey of the security make sure that these all are sensitive traffic, very sensitive traffic; they cannot afford a delay, so all others are been blocked.

So, in such kind of context it is always preferred to map the sensitive applications in to one cluster. And map the remaining clusters with a fair mix of heavy as well as light applications. So, thereby we make sure that sensitive applications are not interfering with other non-sensitive applications. So, certain area can be reserved for sensitive applications such that their performance is not impacted by the presence of others, and the rest of the area can be filled with the right mix of heavy as well as a light applications.

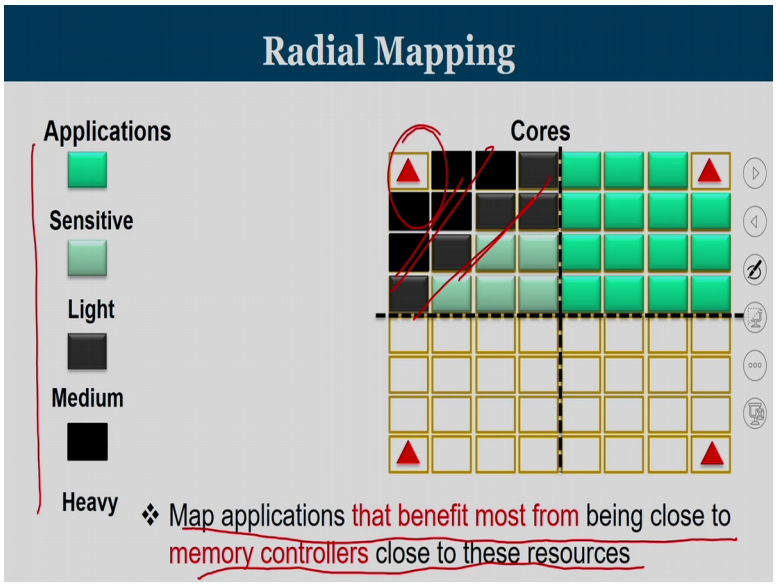
(Refer Slide Time: 59:41)



Now, how will you estimate sensitivity that is a crucial design question. When you have high number of misses, it is been defined with the help of MPKI. So, MPKI define the application have the number of misses per kilo instruction. And when you have low memory level parallelism, so you have misses and for each miss there is high relative stall time. And the application is to be sensitive if the MPKI is greater than threshold, and relative stall time per misses stall cycles per miss is very high.

So, it generate so many misses and this misses are not overlapping. So, the stall cycle is very less such applications are going to be sensitive. So, whether to or not to allocate cluster to sensitive application is a design issue.

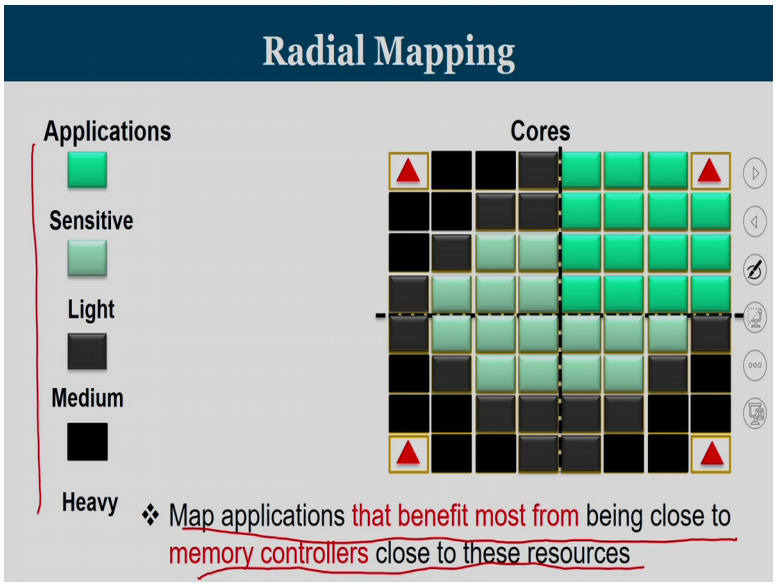
(Refer Slide Time: 60:31)



Now, the last work that we are going to see today is radial mapping. So, what we try to do is we have different applications here; some are heavy, some are light. Now, we know that the heavy applications incur more number of misses, more packets, so they are moving into L 2 cache tiles, they come back. And the probability that they have miss in the L 2 also is very high. So, they may some of these misses generated by heavy application can go to the memory controllers.

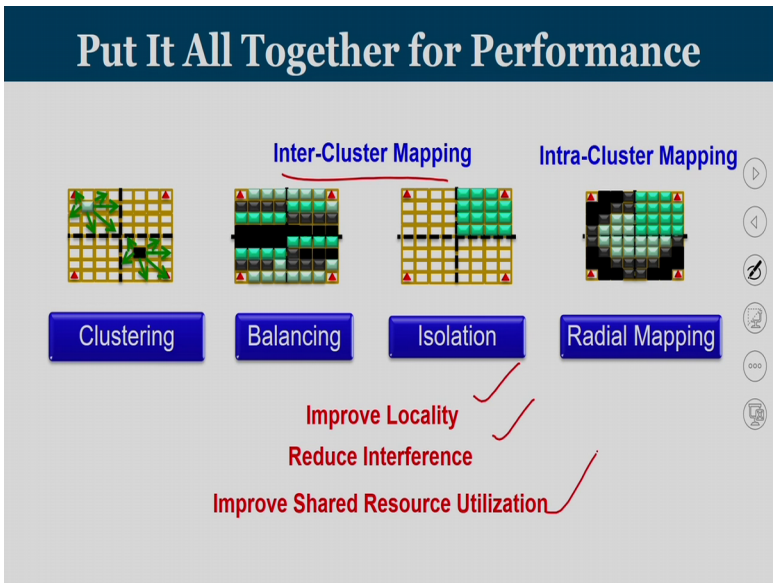
So, what we can do is map applications that benefit most from being close to memory controllers, keep them close to the memory that is the way by which first you keep the isolated or the sensitive applications. Then the heavy applications are put near to the memory controller, and then relatively light applications close to the centre and the light applications are put it in the centre.

(Refer Slide Time: 61:49)



So, the idea is very simple heavy applications are mapped to let us say corner or the edge routers the heavy one, then the medium, intensive ones and the light ones and then you apply it across all the cores. So, this make sure that the sender of the tile is having applications which are relatively of less network activity, and the edges and corners are having applications with high network activity.

(Refer Slide Time: 62:05)



Now, put it all together for performance what we have seen we have clustering, we have balancing and isolation that takes care of inter-cluster mapping, and then we learned

about the radial mapping that is called intra-cluster mapping. So, all together they are going to improve your locality, it will reduce the interference, and it will improve the shared resource utilization. So, these are the things that we discuss today.

Just to summarize today's lecture. We started our discussion with looking into application level parameters that is called trying to understand and define terminology called slack and memory level parallelism. Find out slack of a packet slack of a packet is depended on how far the predecessor packets have travelled, and embed this slack value in the packets and that is why we do slack aware routing. And then we find out the intensity of network usage, network intensive applications are throttled, and then we have seen what is the role played by operating system in spatial scheduling, where which will tell you where an application is kept in a TCMP. So, this gives a total picture.

So, there are certain things that we can work at the hardware level by modifying the micro-architecture that is why application aware routing, slack aware routing and all. So, all these are changes that we make at the micro-architecture level, at the cache memory level or the NoC level such that application performance is improved. And towards the end we have seen what is the role played by the operating system in properly mapping certain applications into certain cores.

So, all together, we have seen that if you make certain small changes both of the hardware level and at the operating system level, then the quality of service offered by NoCs and cache can be improved. So, with these we come to the end of this quality of service experience. And there are few tutorials that are been put up problem sheets. I request you to just go through this problem sheets, and get yourself in familiarised with this course. So, with this I conclude today's lecture.

Thank you.