Computer Architecture
Prof. Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 08
Lecture – 28
Multicore Processors

So, in the last module, we looked at multi threading and in this module we are going to discuss the need for multicore processors, or also called as chip multi processors.

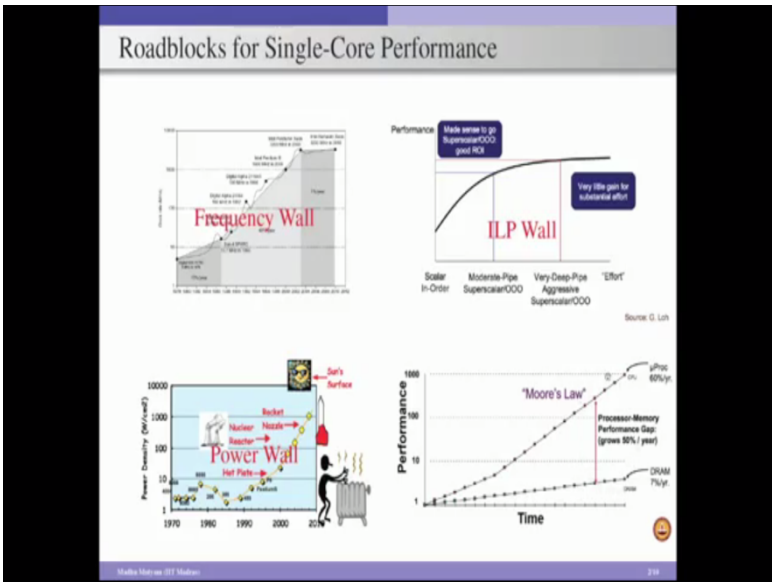
(Refer Slide Time: 00:24)

So, there is a significant demand from the user side for high computational power. Applications such as image processing, speech, video and several other applications are actually requiring high performance computation. So, in order to support these demands, we need to come up with the efficient processors so that the processors will have necessary hardware components to process the different application requirements.

So, in order to come up with high performance processors, we have a support from VLSI process technology, in terms of the number of transistors integrated on a chip. So, if we see here over a period of time, the number of transistors integrated on a chip is tremendously increased. Now, currently we are in stage where we can integrate billions of transistors on a single chip. So, once we have billions of transistors, in a single chip, so we can use these transistors to provide whatever the functionalities required by the users. So that we can take care of their demands and provide high performance computation.

So, we need to exploit these available transistors to provide various functionalities, to satisfy user requirements and also we can improve the overall performance of the processor. In addition to this, there are certain things which led us to go for multicore processors.

(Refer Slide Time: 02:19)



So, the superscalar processor whatever we discussed earlier are corresponding to a single core systems, even the multi threading systems, whatever we considered are also part of a single core systems. So, once we have single core system the amount of performance we get is not significantly increasing, because of various reasons.

So, one of the main contributors for the overall performance improvement for single core processors is the frequency scaling. But so this scaling up of frequency cannot go beyond a certain value mainly because once we have billions of transistors clocking with the high frequency, then all these transistors are going to consume energy. And that translates to the heat because once the amount of power increases, and the power per unit area is called as the power density. And the power density translates to heat, and once the power density increases the amount of heat generated on the chip will be significantly higher. And as a result we will have thermal issues.

So, in order to tackle these thermal issues we should not increase the clock frequencies significantly. So, this is called as frequency wall. So we cannot scale up the frequency beyond certain value, and we scale up the frequency beyond that value then we are going to have

power and the thermal issues. So, as a result we should not go for the frequency scaling up significantly. And the second the driving force for us to go for multicore system is ILP wall.

So, here we already discussed in the superscalar processors, so if we go for the wide issue superscalar processor with a deeper pipeline and with aggressive superscalar techniques, we can improve the performance. So, when I say aggressive superscalar techniques, the techniques such as speculative issue of instructions and having support of advanced branch predictors and so on.

Once we have deeper pipeline with the multi issue, the super scalar processor, we can have a significant number of in-flight instructions, in our processor at various stages in the pipeline. And once you have many in-flight instructions we can look at independent instructions among these in-flight instructions, and then we can schedule these independent instructions onto functional units. And as a result we can overlap the execution of instructions, which in turn improve the overall performance. But from this graph if we can see here the x axis, shows the effort we put in and the y axis, show the performance we get in return.

So, started with scalar in order processor moved to moderate pipeline superscalar processor with out of order execution, and further moved to very deep pipeline superscalar processor with out of order execution, as well as aggressive superscalar mechanisms, if we see here, as we move from scalar in order processor all the way up to very deep pipeline superscalar processor, the amount of effort we put in will be significantly increased. And also in order to support this aggressive superscalar mechanisms, out of order executions, and deeper pipelines so our hardware also is going to take more area under that things.

Once we have these extra components, and what is that we are going to get in return in terms of the performance? So, the performance improvement is not significant compared to the amount of effort we put in, so we can go up to this point, but after that we can clearly see the performance curve is almost flatten. So, this indicates that we are going to get very little gain in terms of performance with a significant amount of effort, we have to put in to extract the amount of instruction level parallelism available in applications. So that is called as ILP wall.

So, applications are not having significant instruction level parallelism, as a result whatever the effort we put in to extract the available ILP, the amount of returns we are going to get is very small. And as a result we have to look for some other alternative. The third driving force for us to go for single core to a multicore is the power density. So, as I mentioned earlier

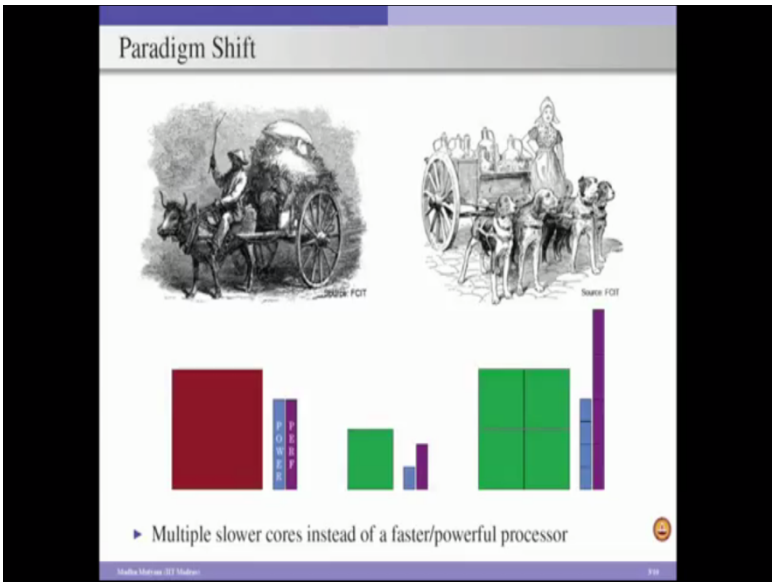
power per unit area is called as power density and once we have more and more transistors integrated on a single chip, and that too the area of chip is very small.

Once we are clocking all the available transistors in the chip with high clock frequency, these transistors are going to consume power and as a result the overall power density is going to increase significantly. And because of that our power density is very high and this power density translates to heat, and so if you are not going to take care of this then the chip may burn out or otherwise, like it is going to give undesirable results when we use that for some computation. So, this is called as the power wall.

Finally, the last one, but this is very important one this is the memory wall. This says that the performance gap between the processor and the memory is widening over a period of time. In other words the processor performance is improving year by year significantly, but the memory technology is not developing significantly as a result the memory performance improvement rate is not as high as that of the processor performance rate. And because of that we can clearly see here there is a significant gap between the processor performance and the memory performance.

So, once we are executing an application on a processor, which is working with high frequency. And when it wants a data, the memory is the one which is going to supply the data, so if the memory is not supplying the data as and when required by the processor then the overall computation will be stalled. And as a result the overall performance of the system will be degraded. So, in order to tackle this memory wall problem we already discussed in one of the previous modules, about the multi-level cache hierarchy.

(Refer Slide Time: 09:16)



So, we need to break all these walls, so in order to break these walls that is ILP wall, frequency wall, memory wall, and the power wall. So, we have to come up with different designs. So that means we need to think of a paradigm shift, in our processor designs. So, rather than getting the work done by using a bullock, what we can do is we can get the same thing done by a collection of the dogs. This shows that rather than considering a complex superscalar processor with aggressive superscalar techniques, and deeper pipelines to get our computation done quickly.

We can actually consider a collection of simple cores, but all these cores are working in parallel, so that our computation can be finished quickly. And also we can reduce our overall power consumption, and the design complexity can also be minimized. As a result the overall performance improved at the same time without increasing, the amount of power consumed for the computation.

So, in other words, if we have a superscalar processor with the deeper pipeline with an aggressive superscalar mechanisms, let us assume that this processor is going to take one unit of area, and it is going to consume one unit of power, but it is going to give you one unit of performance. Now, rather than considering this complex superscalar processor which may be working with x gigahertz clock frequency, now we scale down the clock frequency of a new processor and also we consider a simple processor, and assume that this is a simple

superscalar processor, or even a simple in order processor and which is actually working with much lower clock frequency as compared to this one.

Because this is working with lower clock frequency and also it is not having any aggressive superscalar mechanisms. So, as a result the performance may not be as that of the performance, whatever we get with this complex processor. And because we are running this with lower clock frequency, the power consumption will be significantly reduced as compared to the power consumption for this complex core. And also because of the simplicity in design, so this is also actually taking a less amount of area. That may be because like we do not have to consider our dynamic scheduling mechanisms, we do not have to consider aggressive the superscalar mechanisms such as advanced branch predictors, deeper pipelines and so on. So, as a result we may reduce the overall area occupied by this a processor.

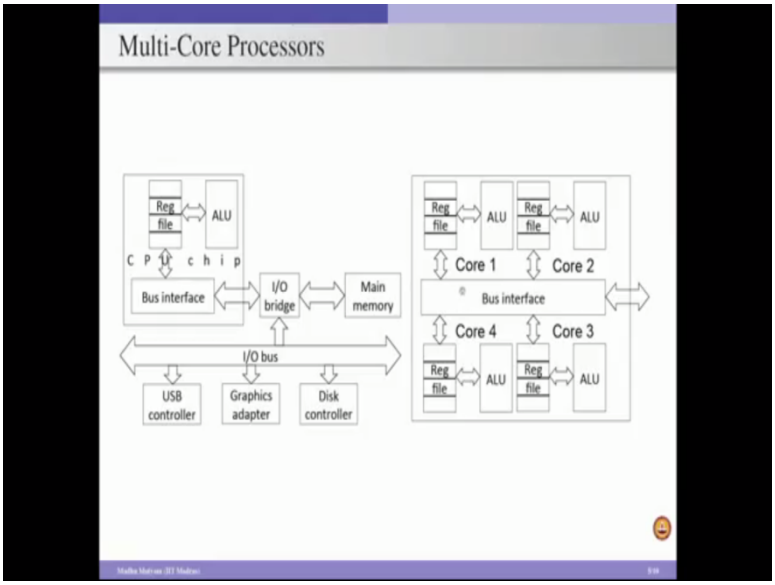
And once we reduce the clock frequency, we know that the power consumption is proportional to c v square f where f - is the clock frequency, v - is the supply voltage, and c - is the capacitance. So, once we clock down our frequency we have to reduce our voltage also. Once we reduce the voltage then because this power consumption is quadratically dependent on the voltage. So, power consumption will be significantly reduced because of scaling down of voltage and as a result we consume very little power consumption, as compared to the power consumed by this bigger one.

So, assume that this power consumption is one fourth of whatever the power we consumed here. And the performance is may be somewhere around 40% of this particular thing. The area occupied by this processor, you can assume as one fourth. Now we have this much area dedicated for our chip, but because of our revised design a simplified design, we are consuming only one fourth of the area. Now, what we can do is, we replicate this simple core four times, so as a result now we are actually taking the same amount of area as that of this complex core.

Since this is consuming one fourth of the total power consumption, whatever consumed by this complex core, and when we replicated this four times. So we can clearly see here our power consumption is almost same as this, but the performance is improved as compared to this. So, this actually gives us a motivation to go for multi core systems, rather than considering a single core system.

Here we have collection of simple cores and all these cores are working simultaneously to compute our required task. And if the task is highly parallelizable, then we can divide our task into four subtasks and each of this subtasks can be executed on each of these simple cores. So, effectively our overall throughput will be improved, in other words overall performance is going to be improved, when we consider a multi core system. And so as a result rather than considering a faster or a powerful processor, consider a collection of simple and the slower cores. And once we have this collection of simple and slower cores, we can execute our parallelized application. So that the overall performance can be improved, so this is the overall idea behind going from a single core processor to a multi core processor.

(Refer Slide Time: 15:30)



So, what is a multi core a processor? If we see a single core system, so we have set of registers called as the register file, we have functional units, we have the inter connection network connecting different components, like register file is connected to the external components by using this bus interface, and there is an interconnection here. And also there is an inter connection between this register file to this ALU or the functional units. And similarly, there is an I/O input output bridge and which is connected to the main memory, and also we have not shown here, but we can consider like multiple levels of cache hierarchy within this.

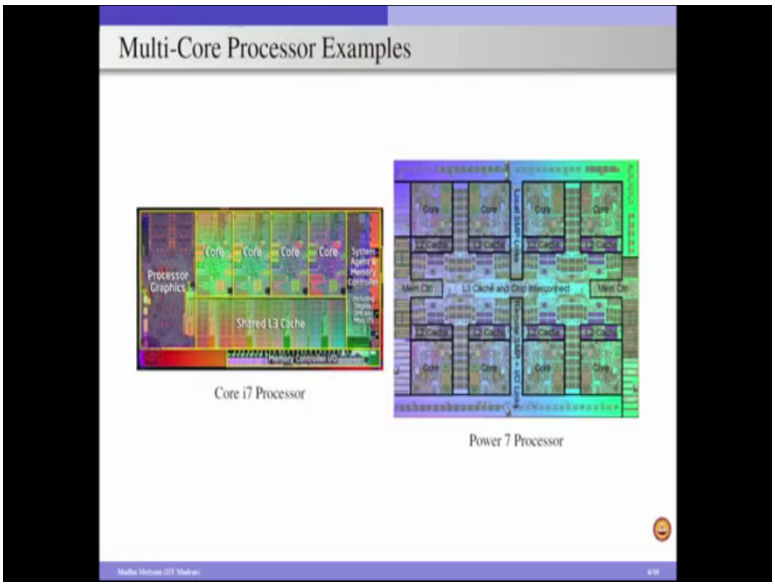
And also there will be several other components connected to this I/O bus such as disk controllers, graphics adapters, USB controllers and so on. So, this entire thing corresponds to

a single core system, so here this part is called as a core and rest of the things or the other components associated with this core to make this into a full fledge system. Now, in a multi core processor we just replicate this component multiple times, we can see here this is a bus interface. And currently this bus interface is connected to this single set of register file, and the functional units and also not shown, but also we have the cache memory and other things.

Now, we replicate each of these things into multiple number, so here we have four core system, where all this four cores are connected to this bus interface. And this bus interface is in turn connected to the other components of the system, and these other components are shared by all the cores of this particular system. So, here this is called as the single core system and whereas here this a four core system. So, the only difference between single core and the four core is here we will have four single cores replicated, and rest of the other things are same.

So, once we have this multi core system, now because each of this core can work independently and they can share all other resources similar to the they can share all other resources, with all other the cores. So as a result so once we have this multi core system where each of this core can work independently, and they can share and each of this. Now, so once we have this multi core system each of this cores can work independently, and all these cores share all these resources. And once such system is there, now we can actually execute independent applications, or multiple threads of a same application on all these cores simultaneously. So that we can improve the overall through put of the system which in turn improve the performance of the system.

(Refer Slide Time: 18:45)



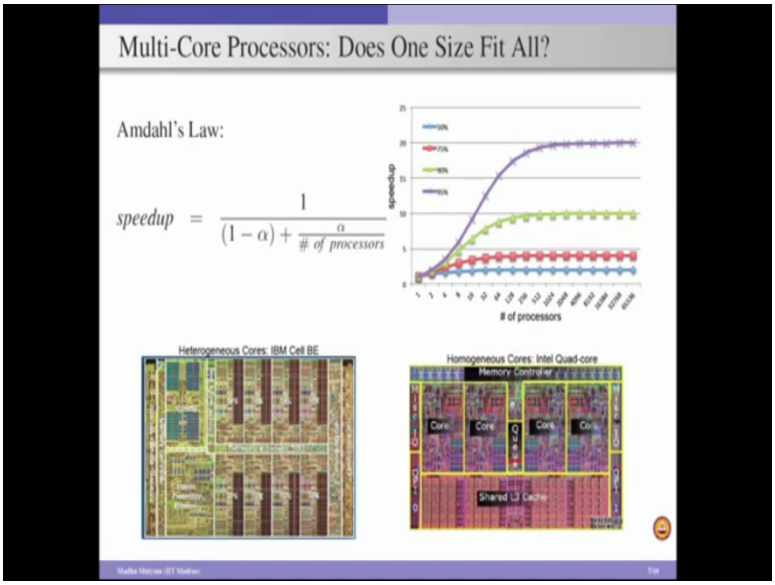
So, example multi core processors are one is like core i7 processor from Intel and power -7 processor from IBM here we can see core i7 processor, which has four cores. And each of this core has all the support for executing instructions in an out of order fashion, and also it has multi levels of caches L1, L2 caches, private to each of these cores. And there is a shared cache and this cache is shared by all these four cores. And also in order to handle the graphics related things, this processor also has a separate component the processor graphics, which takes care of processing all the graphic applications.

Also this extra components, which are actually having a support for memory controllers and also connecting to the other components of the system. So, this whole thing corresponds to, this whole thing corresponds to a core i7 processor from Intel. And similarly, when we consider IBM the power 7 processor we can clearly see here we have eight cores associated with this. And the L3 cache is there and each of the core has internally private L1 and L2, and also support for memory controllers and support for connecting to the other resources in it.

So, in other words these example show that multiple cores can be integrated on a single chip, and because the cores are now placed on a single chip. So, the communication between core to core is not going to take significant amount of time, as compared to multi processor system, where multiple processors are physically apart and these systems are connected through external communication links, and so on. So as a result this multi core processors are going to improve the overall performance of the system. So once we have this multi core

processors, now the question is whether we have to consider all the cores similar in type, or do we have to consider different collection of cores in our multi core system.

(Refer Slide Time: 21:00)



According to Amdahl's law if alpha is the fraction of the code that can be executed parallely, then speed up what we can achieve with n number of processors is,

$$Speedup = \frac{1}{(1-\alpha)+\frac{\alpha}{n}}$$

where n is the number of processors. So, if alpha is, let us say 95%, so that means when you have 95% of your code can be parallelizable. And even when you consider 64 k, 64 into 1024 that is 65536 processors to execute your computation, you can clearly see here we can get around 20x speed up in our system.

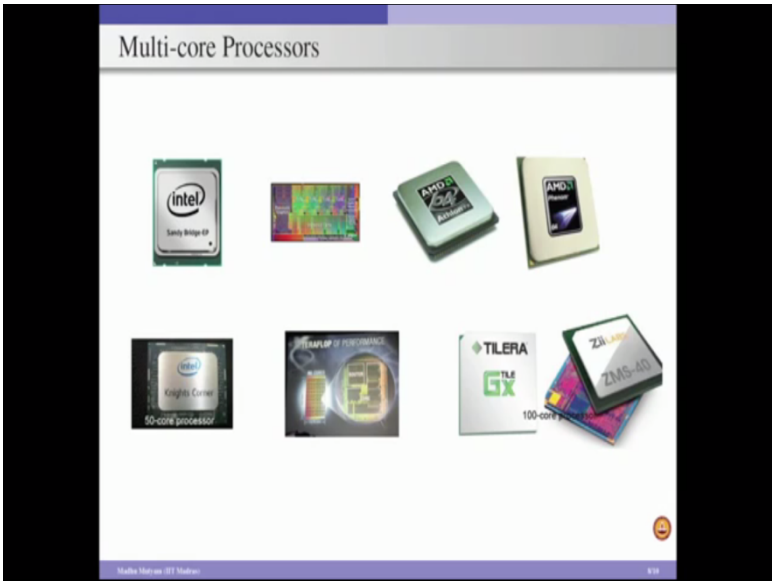
So that indicates that the amount of speed up we can achieve is limited by the fraction of the code that can be parallelizable. Even when we have 95% of the code that is parallelizable, because of this 5% sequential code in the program, we are not gaining significant speed up in our system, even when we consider 65536 cores. So that shows that it is not always good idea to consider uniform cores, in our multi core system. We can actually go for heterogeneous multi core system, when we consider all the cores in our system same, then that is called as homogeneous multi core system. But when we have some cores different from the other

cores, then we can say this is a heterogeneous multi core system. Example for heterogeneous multi core system is IBM cell.

So, here we can clearly see we have one complex core or a processor and a collection of simpler cores, and when there is an application which is highly sequential in nature, then we can use this a complex processor, or a powerful processor to execute the sequential portion. Whereas if we have highly parallel application we can execute the parallel portion on this collection of simple cores. So that we can improve the overall performance.

On the other hand if we see a homogeneous multi core system, this is from Intel, so here we have four cores and so here all the cores are similar in nature, in terms of the superscalar techniques, in terms of the amount of cache associated with each of the cores and in terms of the clock frequency with which each of the cores can work. So, all these are similar so as a result this is called as a homogeneous core, but again depending on the type of applications, we have to consider whether we have to go for a homogeneous multi core or heterogeneous multicore processors.

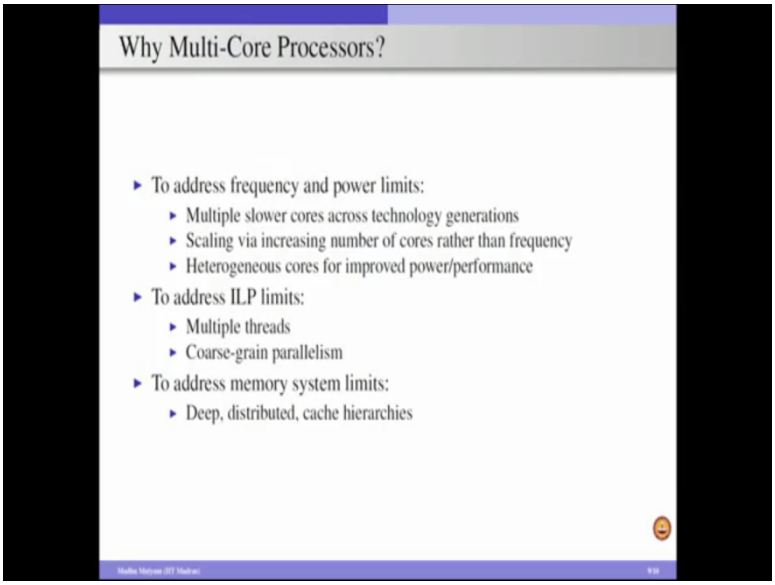
(Refer Slide Time: 24:29)



So, these are the examples of the current day multicore processors from different vendors processors from Intel, processors from AMD, processors from Tilera and so on. So, there are different type of the multi core processors available in the market, and each of these have different characteristics, and different computational powers, and the different number of cores inside it. And so in summary, we are in the multi core era where rather than looking at

scaling up the clock frequency, we are now looking at scaling up the number of cores in the processor.

(Refer Slide Time: 25:19)



We need multi core processors because this multicore processors are going to address our different wall problems. The ILP wall problem, power wall problem, frequency wall problem and so on. So, the first thing is to address the frequency and the power limits whatever we faced with the single core processors in order to improve the performance. So, we can consider multiple slower cores rather than a single complex core and we can scale the performance by increasing the number of cores rather than scaling up the frequency.

And depending on the type of applications, we can go for heterogeneous cores and to address the instruction level parallelism a limits, whatever we faced with the single core processors because once we have multiple cores in our multi core system, we can execute multiple threads on each of these individual cores. We can exploit coarse grain parallelism to improve the overall performance, and finally once we have multiple cores and still we have this the memory wall problem. In order to address this memory wall problem, we have to go for deep distributed cache hierarchies, these cache hierarchies are distributed across the multiple cores in our multicore system, but these caches can be shared.

So, effectively we can consider a shared distributed cache across all the cores, and also we can consider multi levels of the cache hierarchy. So, rather than one level or two levels, we can consider three levels, even the there are the current designs targeting at four level cache

hierarchy. So, once we have deeper cache hierarchies, so we can minimize the number of times going to the memory to get the requested data.

So, we can keep the requested data in our multi levels of cache hierarchy, where typically first one or two levels can be considered as private caches for each of the cores, and the remaining levels of the cache hierarchy can be considered as shared across all the cores. And with that I am concluding this module and in the next module I am going to discuss the issues that are associated with this multicore processors. And how to take care of those issues, so that we can use multi core processors without any problem and we can improve the overall performance of the system.

Thank you.