Parallel Computer Architecture
Hemangee K. Kapoor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati
Week - 01
Lecture - 06

Lec 6: Some Laws

Hello everyone. We are doing module 1, Introduction to Parallel Architectures. This is lecture number 6. Here we are going to see at two laws which are very important for assessing computer architecture. The two laws are Amdahl's law and Gustafson's law. Okay. In the previous lecture we have seen speed up.

So speed up was how does a machine perform with respect to a reference machine. So with respect to a reference machine what improvement did we get. Now if I want to talk about speed up of a multi-core. So speed up of a multi-core is my reference machine is a single core machine and when I replace a single core machine with a multi-core machine how much speed up do we get.

So the slide shows that the speed up on this machine is $T1/Tn$ where T1 is the execution time on a single core system divided by the time taken on a machine having n cores, Tn. So speed up is time taken on a single core divided by time taken on n core system. So what do you think which value will be bigger, would a single core systems time will be more than Tn. Yes correct. So T1 is always going to be greater than Tn, because on a multi-core I can run the same program exploit some parallelism and have smaller execution time.

So if I take this ratio this ratio will be always greater than 1. So the speed up when I am using n cores is always greater than or equal to 1. We also define another term called efficiency. So efficiency is how useful my n cores were. So speed up we got on by using n cores and we divide this by n because the max value we can get is n as we have n processors we can do as good as n.

So efficiency is speed up I obtain divided by n and what do you think this fraction would look like. What will be the value of this fraction? This fraction's max value will be equal to 1. Okay. So the efficiency of my system can be as max as equal to 1. Right. The speed up on a multi-core can be maximum n and we do the execution time on a single core divided by execution time on a multi-core. Alright. So what is Amdahl's law? So Jean Amdahl in 1967 defined this law.

What he observed is when we add n processors to the system we may or may not get n times speed up and why does this happen? Because I have a sequential or a serial program or a big program which I was running on a single core machine this program we want to now migrate to a multi-core system. But every program cannot be starting in parallel. We cannot just cut the program into small pieces which will immediately start executing. You have some portion which is sequential. For example you want to read the data, there is some calculation, some control logic, you have to write the results, communicate information between one segment of the program to the other.

So there is a significant portion in the program which is sequential which cannot be broken into pieces. Okay. So I can divide a program into a fraction that is sequential F_ Seq, which I am showing on the slide and a portion which is parallelized. So I can make it into a parallel part and another a sequential part. Okay, so if I take execution time we can divide a program into two parts, one is the sequential fraction and then one is a parallel fraction. The fraction that can be parallelized is F_par and the fraction which cannot be is called F_sequential.

Okay, so this is the portion which we can make into parallel that is we can employ multiple cores to make this portion fast. How fast can we make this? By adding more and more cores. So I will just quickly show you a pictorial representation. Here, we have the light color, the sequential and then we have a significant portion which can be parallelized. On a single core machine where n is equal to 1 it is going to take this long execution time.

If we replace that with another core, go from a single core to a dual core system, the parallel portion can be scheduled on two different cores and hence it halves the execution time. So the dark orange portion reduces to half. If I go from n equal to 2 to n equal to 4 it reduces further, to n equal to 8 it reduces further. But if you see with n equal to 8 cores, the orange portion is much smaller than the sequential. So essentially the sequential portion then starts to dominate the parallel one if you add more and more cores.

So there is a limit to how much speed up I can achieve. That is even if I keep on adding n equal to 8 or more, the sequential portion is not going to go away and it is going to limit the performance. Right. So if I derive this as a formula, we will go to the speed up formula and try to derive a similar conclusion. okay. So here, speed up of , speed up on n cores is execution time on a single core divided by execution time on a n core system. What is T1? T1 is the time taken for the sequential fraction F_Seq plus the time taken by the fraction which can be parallelized.

That is time taken on the F_par segment. Alright. For Tn, how would this value be? The F_sequential time remains same and what happens to the parallelizable fraction? Now this will get divided by n. Okay. So this portion will get divided by n. Okay. Then if we go back and substitute these values in the speed up, so the speed up on n cores, I am not using the time T as a variable, we will just talk in terms of fractions.

So if single core takes one unit, then the multi core is going to take F_seq amount of time for the sequential and time for the parallelizable portion is going to be F_{par}/N. So if I take one unit of time on a sequential on a unicore system, so numerator becomes one and in the denominator, we have F_sequential fraction of the sequential portion plus the parallelizable fraction gets divided by n. Okay. Solving this little further, we will replace, so what is this F_seq in terms of F_par? So this is 1 minus the parallelizing factor. I will just rewrite this. This becomes $(1 - F_{par}) + (F_{par}/N)$. okay. Alright.

So the speed up achieved by multi core is $1/((1 - F_{par}) + (F_{par}/N))$. If I ask you what is the max speed up we are achieving? What is the max speed up I can achieve in a n core system? How will you get that? By increasing the number of cores, right? You are going to increase the number of cores further and further and how do I derive this? So to derive this, the SP_n equation is with you, you simply have to take a limit of this equation as you increase the number of cores. So you have to take a limit as n tends to infinity. So I will take SP of n because I want to find out the max speed up. So the max speed up is that equation which is in the blue color and limit it as n tends to infinity, $lim_{n->\infty} S_{p_n}$.

So this is SP_max is $lim_{n->\infty} S_{p_n}$ and if I go back to this blue formula here there is only one n. If I limit this n tending towards infinity what value will you get? This term becomes 0, so it becomes $1/(1 - F_{par})$. And what is $1 - F_{par}$? It is $1/F_{seq}$ because we had replaced F_ sequential with 1 minus F_parallel. So this maximum speed up we can achieve is $1/F_{seq}$. Okay. So the same formula is written on this slide here. Right.

So the maximum possible speed up is limited by the portion which cannot be parallelized that is the F_sequential. The sequential fraction is going to limit the amount of speed up you are able to get which was also visible in this pictorial representation. The orange portion was getting smaller and smaller but the light orange portion remained the same and that is what which limited the maximum speed up. So that was Amdahl's law and what do we learn from this? What are the lessons from Amdahl's law? One is you propose innovations as an engineer you are going to propose enhancements to the

designs but any enhancement you propose how much effect will it have on the final product? Will it really give you the speed up you expect? Will all depend on how much portion of the execution time this enhancement is going to contribute. If it is going to contribute a major portion, yes you are going to get a required amount of speed up. But if it is not then you cannot expect significant changes in the original and the new enhanced system.

So the lesson is you have to always optimize on the common case. Do not come up with a very complex arithmetic or an adder unit or a floating point unit which is not going to be used every time. If you are going to use a small ALU every time try to optimize this ALU rather than work on a very complicated pipelined new design which you are going to use only some amount of time. So concentrate on the common case, make the major portion of the execution time as parallel as possible, make sure that your enhancement is contributing to a major portion. Alright. And in the example of multi-cores we saw that we were adding more and more cores so it was improving the performance but how much? So how do I know where to stop because everything has a limit. So Amdahl's law says that we have a law of diminishing returns because any enhancement you propose, it is going to need more resources right.

So you will design a new hardware it will require control logic, more data structures, extra storage registers. So additional resources are to be reserved for this enhancement. Adding more cores will give you some speed up initially. But if you keep on adding more and more cores the program is going to divide the task across the several cores which are available but it would happen at some point of time that every core has got very little amount of work to do, but they spend most of their time communicating with each other. So we are not doing effective work but this because this communication is not part of the program it is done because we divided the task across multiple cores. Hence we should know where to stop an enhancement that is how many cores to add to solve a particular problem. Okay.

Then the speed up can be defined in different manners. For example we talk of speed up of a given algorithm or unicore versus multicore. I can say that how much does my bubble sort algorithm take on a single core machine versus a multi-core machine. However if you want to do it in a better way it is nicer to do speed up with respect to an application and not an algorithm. That is I would say I want to speed up my sorting algorithm or sorting as an application and not want to speed up a particular sorting algorithm. Right. So sorting can be done either by a quick sort or bubble sort.

So depending on the machines you have or the resources you have you can choose a particular or better algorithm suiting to the hardware setup and not only keep trying to

improve a particular given algorithm. Okay. So application level speed up would mean that the amount of work done will be definitely different on a single machine as well as a parallel machine and hence it is always good to also think of having better algorithms when you move from a unicore to a multicore system. Next we are going to look at Gustafson's law. So here I will go back to the Amdahl's speed up. So what was the max speed up given by Amdahl if you remember? The max speed up on a N core system was $1/F_{seq}$. Right.

So the sequential fraction was there. And we could divide the execution time into two parts S, small s for sequential portion and small p for the parallel portion. Okay. So this was the setup when we discussed Amdahl's law. But Gustafson observed that this parallel portion although it is not linked to the max speed up, but the small p this parallelizing portion when I move from a single core to a multicore system I can exploit the multiple cores to do this p faster. In other words I could do multiple such p's. Right. So we could extend the problems domain for handling larger data sets because S is similar or would remain almost similar but I am able to parallelize the p so why not extend the parallelizable portion.

So that is the proposal which Gustafson proposed and he said that how do I use the capability of a multicore by giving it more and more work. If you recall the GPUs had so many cores we were supposed to give lots of work to utilize those arithmetic units. So similarly this parallelizable portion which is running on a multicore, I am supposed to give it more work solve larger problems, solve more complex problems, use it for bigger data sets. So effectively what is going to happen? the f power is going to increase. So F_sequential yes it is limiting the performance according to Amdahl's law.

So the max speed up is this but if I am increasing F_par very much I would get good performance. And therefore here the added value will not come from the speed up but it will come from the added functionality. Okay. So we will work it out to understand how. Alright. So we will do some calculations. So here we had divided the execution time on a unicore.

I will use some new terminologies to explain this concept. So instead of using T1, I am using T_uni to say that this is the execution time on a unicore system which is simply the sequential portion plus the parallel portion. It is going to take both these times. When we go from a unicore to N processors that is N core system what do we get? T_execution on multicore. How much would this be? It will be S which remains same.

This P parallelizable portion gets divided by N because I have N cores. Okay. So this is how I would compute the execution time on a multicore. But suppose I say that let the

T_uni be equal to T_multi. Okay. So if I say T_multicore if I want this to be equal to the unicore then what would happen? So this is going to be equal to $S + P$ but this is on a larger problem.

So this is for a larger workload. So when my workload increases the multicore is able to exploit that by dividing the work across N cores. But the unicore would not be able to do this and hence, if I am running a larger workload the T_unicore on a larger workload. I will call it a new. I am just using the suffix new to say that I am running this on a larger workload. So how much would that be? It will be S plus.

Now this P would become N*P because N multiplied by P because the multicore was able to do this P in time unit P by dividing it across N cores. But if I have larger workloads, for larger workloads the sequential system that is the unicore would take N times P amount of time to do the same work. Okay. So I hope it is clear. Multicore was assuming $S + P$..Multicore was able to do this P faster by dividing it across N cores and if we move the concept to handle larger workloads we say that my P is now working on huge amount of data which can be sent across N cores.

So multicore is able to do it at $S + P$ time but the same P because it is very large supported across n systems when it comes to a new unicore setup, my new unicore time will become $S + NP$. So now let us do the speed up calculation. This was the time, execution time. So what is the speed up? Speed up on the new unicore. Now we are moving the setup from a simple data set to a larger data set.

So time on the unicore but the new one which is working on large data divided by time on the multicore. Okay. So how much is this? On the new unicore to do the large P, I am going to take N time units whereas the multicore is going to take some P time units because this P is very large multicore distributes it and in the numerator the unicore is not able to distribute it hence it becomes $S + PN$.. So that is the speed up achieved on N core multicore system. Okay. Fine. So now we will look at the fractions. So the fractions which can be parallelized in this program. How much is that? P is the amount of fraction which can be done in parallel and the complete fraction or the complete thing is $S + P$.

So from this I will derive the equation for S which is sequential and then substitute it there. So solving this for S. So we have $S + P$ is P/F_{par} and therefore $S = P/F_{par} - P$. Okay. So that becomes $(P - P * F_{par})/F_{par}$ and finally take the P common $(1 - F_{par})/F_{par}$. Got it? So we have solved for the small s, the small s and we are going to replace it here in this equation.

Okay. So that I will do on the next slide. Okay. So some similar information is there here which I explained on the previous slide. Now we will continue the calculations and the objective of this calculation is to show that Gustafson's law helps us to achieve a linear speed up with respect to the number of cores. Okay. I will just flash the previous slide to show you the speed up formula.

We had stopped here. The red one is showing the speed up, $(S + PN)/(S + P)$ and the green one at the bottom is showing the value of S. So we are going to replace the value of S in the, the green S in the red one at the top. Okay. Speed up SP was small $(s+ N_p)/(s + P)$.

Okay. And then the small s, the sequential portion was $P(1 - F_{par})/F_{par}$. This we saw in the previous slide. We have derived this. Now we will derive this speed up using the value of small s. So because it is a bigger term, I will do both the parts separately.

I will do the numerator first. So I will do the SP's numerator. So numerator is small $s + NP$. So small s is this value. So we will take that $P(1 - F_{par})/F_{par}$that is the fraction which is parallelizable + NP. I can take P common and then solve it a little and you get $P((1 - F_{par} + NF_{par})/F_{par})$.

Okay. So you get this, this is the numerator. Now let us solve for the denominator. The speed up's denominator. So what is the denominator? It was $s + P$. Now let us substitute the value of s.

$P((1 - F_{par})/(F_{par})) + p$. Okay. So if we solve this, what you will get is if we take P common, we get $P(1 - F_{par} + F_{par})/F_{par}$. All right. So we have solved the numerator and the denominator. Now let us join the results.

So the final SP is the numerator divided by the denominator. So this is the numerator here, this value. So I will say this is V1 is the first expression and V2 is the second expression. So we want to do V1 over V2.

If you do this, the P cancels out, the F_par cancels out. Okay. So let me write it here. If you work it out for yourself on a notebook, you solve it and what you will get at the end is $1 - F_{par} + NF_{par}$. Okay. Because this V2, if you look at V2, it remains, actually I will just reduce that further. What is V2? V2 is actually P *, because the F_par cancels out, it is $P(1/F_{par})$ because the F_par cancels, you get this.

Okay. So V2 is P/F_{par} and then if you do V1 over V2, what you get is there in this green color. And observing this green equation, what do you conclude? What is the speedup on N processes? You can see it is now linear with respect to the number of cores. Okay. So if we add more and more cores, we are going to get better and better speedup.

Alright. So the same formula given again here in an neater manner. So Amdahl's law gave us better speedup only if the F_par reached around 90 and more. So beyond 90% parallelizable portion, Amdahl's law gave us good speedup. But Gustafson's law says that more and more cores will give you a linear speedup because we can do more work on these N cores. I can assign better workloads, parallelizable workloads and use these N processes. Thus the multi-core architectures which we plan to design and which do exist will be indeed useful to us.

Alright. So to summarize this, we have looked at Amdahl's law which said that the speedup you can achieve is limited by the fraction which you cannot parallelize. And we eventually start getting diminishing returns even if we add more cores or extend the functionality, the sequential portion will start dominating after a while. So the max speedup is limited by fraction which is sequential which cannot be parallelized. Okay. So of course we should try to use applications and algorithms which suit the given architecture.

So for unicores, one algorithm could be good but you can design a newer algorithm when you move to a multi-core system. And Gustafson's law said that we can have scalable designs with multi-core, have a parallel portion which can be divided across multiple things, have larger data sets, work on complex problems and hence if we exploit this concept, you will get a speedup which is linear with the number of cores which we can add. And this is a good news for us, because this says that multi-cores will indeed be useful to the community. Thank you.