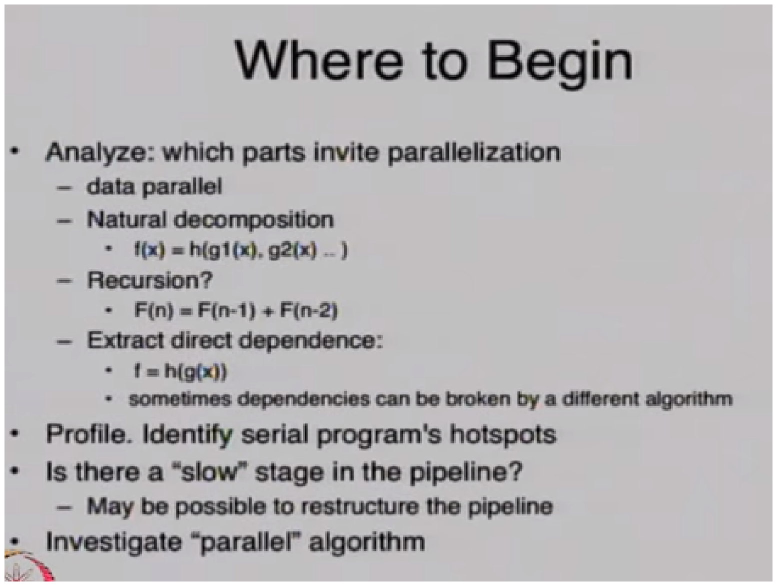
Parallel Computing

Prof. Subodh Kumar
Department of Computer Science & Engineering
Indian Institute of Technology – Delhi

Module No # 03
Lecture No # 13
Parallel Program Design

(Refer Slide Time: 00:32)



Where to Begin

- Analyze: which parts invite parallelization
  - data parallel
  - Natural decomposition
    - f(x) = h(g1(x), g2(x) ..)
  - Recursion?
    - F(n) = F(n-1) + F(n-2)
  - Extract direct dependence:
    - f = h(g(x))
    - sometimes dependencies can be broken by a different algorithm
- Profile. Identify serial program's hotspots
- Is there a "slow" stage in the pipeline?
  - May be possible to restructure the pipeline
- Investigate "parallel" algorithm

Let us talk about parallel program design and so now we are talking about computation side of things and of course when you start if somebody says write or sorting program to do on sorting on high computer and N computer where do you begin? Right you might have the underline sequential algorithm in mine based on that how do you take that and say now have a parallel version of it run it in parallel okay.

So sometimes you can inspect you can say here is the sequential version and these are the parts that can be done in parallel. If you are rate racing some seen in that every independent of each other and hence it can be computed in parallel may be other ways okay so there would be automatic parallelism that the program the underline algorithm will exhibit okay and you can take advantage of it.

You can simply say although those things that can be done in parallel global imparallel okay but it easier side it has done. So the places to look for would be are there similar things being done on various pieces of data for example matric multiplication you are treating the same thing to generate the each element of the final matrix product okay. So that suggest that there are lots of things that are happening in parallel again you can see are there lots of inputs data that are being processed one after the other.

Or are they have lots of output data that are being generated one after the other or I just some intermediate data you have taking input doing some processing and generating bunch of data and then processing each one so this input output it can happening at any stage of you algorithm okay they would also be functional composition.

For example you see that some function F that you need to compute is compute G 1 compute G3 compute sum number of functions on X same piece of data and then compute some H on all of those intermediate results and so H cannot be parallelized but G1, G2, G3 can probably need to be parallel similarly recursion can often give you some hints about parallelization for example (()) (03:18) relation says FN is the as FN – 1 FN – 2. So that means you can do FN by doing FN+1 FN -1 and FN – 2 and then adding.

So this is just starting point it says where to begin in default way you will say I will go FN – 1 and FN -2 and they will be added together right FN -1 in turn will do FN - 2 and 3 FN -2 will in turn do FN – 2 and 3 and then these are parallel piece of work except you are repeating lots of that. So then there is determination to be made about what can be independently computed where needed what or where should be taken from somewhere else which has already computed this result.

And so that read off very real read of that we are going to happen investigate and then basically a look for that kind of fill in this case you look deeper into the recursion see that there is a dependence either you separate that dependence either you say separate that dependence where replicating computation are you incur the cost of the dependence by doing it once and then having two people wait for the result for some K people wait for the result okay.

And the other proof of this dependence is when you say composition of function and your core kind of compute G on your input and then on G compute H that is the result. So G as to be done then H as to be done so this kind of dependent is has to be investigated and then figure out break the dependent or design your parallel program to respect that dependent right we have to do the first and we have to do the second thing.

And as a general principle you would want to take a serial program and if you really interested in saying I have got to run it ten times faster than it running at 100 times faster than it was running then you will see what are the lots of small things can be parallelized look of lot independent things happening together there is big chunk of code or may be physically as small set up instructions where most of the time is being spent.

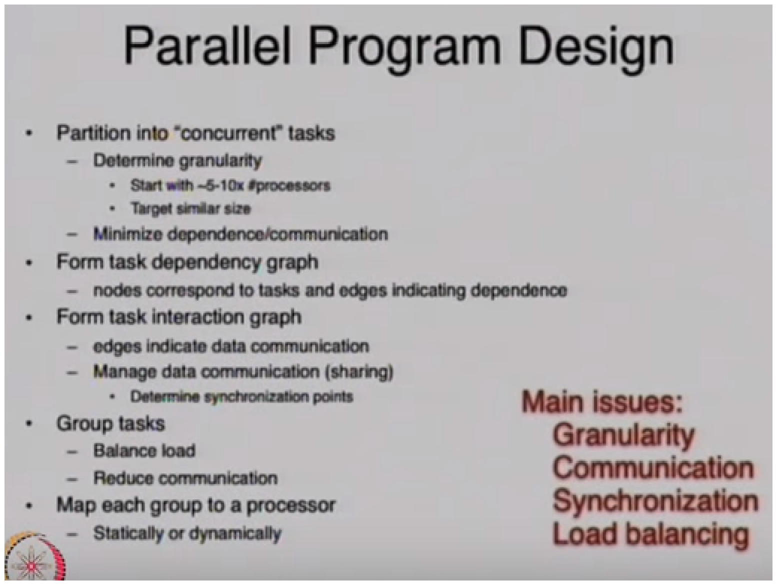
When you profile you say that 90% of the time is being sent here and 10% in the rest of the program and of course you want to parallelize the 90% 10 % may be very well parallelizable it would not get remember. So you will look at you will profile your code see the parts that are the most expensive and then focus your composition on that segment on those sections okay.

Similarly same issue apply when your sequential code is organized as do this then do that on sequence of things and then you will see that there is some stage of the pipeline which may be a bottle neck. Because that takes much longer to do that other stages the pause data is on the first stage of the pipeline will determine the speed then we will take the slow stage and do that part entire we break that into parts that can be done parallel and then do that okay.

But all of that is stating with the sequential algorithm and looking for places which you should parallelize and how we will talk about this basically saying see where parallelism may be a will and is useful to but at least half of time you will realize that the sequential algorithm is not he best algorithm to run parallel in a parallel environment. So then you will say can I restructure the this entire algorithm in a way that you break this dependence have lots of independence things to do which can be done in parallel.

So then the algorithmic side of design will have to investigate is this sequential algorithm what you want to parallelize or there are just means that this should be doing in typically there are in some cases the parallel algorithm is just completely different in the sequential one but in many cases they are just small tunings we have to do her in suddenly becomes ahhh parallel okay ahh.

(Refer Slide Time: 08:27)



Parallel Program Design

- Partition into "concurrent" tasks
  - Determine granularity
    - Start with ~5-10x #processors
    - Target similar size
  - Minimize dependence/communication
- Form task dependency graph
  - nodes correspond to tasks and edges indicating dependence
- Form task interaction graph
  - edges indicate data communication
  - Manage data communication (sharing)
    - Determine synchronization points
- Group tasks
  - Balance load
  - Reduce communication
- Map each group to a processor
  - Statically or dynamically

Main issues:
Granularity
Communication
Synchronization
Load balancing

So once we have determined that here is a section that we need to parallelize it is slow and needs to determine parallel what are the issues we need to look at. Dependency one we are talked about this part of the computation uses the result of that part of the computation which means that part has to be done first. You are going to ultimately doing two things we are going to say I have got this computation today tool of computation things.

And am going to decompose it into two kind of task the task is imagine a capsule of computation and I want to decompose a entire computation into task in a way that the dependence among the tasks is small right and dependence can be in terms of you using you needing a some data that are produce along the way right which is called dependence meaning that I produce a piece of data that am going to give it to and then you continue and I will continue.

And the other is task dependence which says that I cannot begin this task until you are finished you finish and then I start this other time. Data dependence is on a specific piece of data it say that I am producing lots of things right and am going to give you one of those things. So that is dependence on one piece of data right and the other is I am doing lots of computation and then I am done and only when I am done am I reduce something that you will be using right.

It is typically data you are waiting for me because you need something that I am doing right I am producing and but I am going to producing that until I have done versus I keep producing sequence of things then you keep using right. So one is listed as task dependence graph where you say that task is depends on this task and can be done only after the task is done and the other is task interaction which says that I interact with the other task.

Dependence is typically one way right I depend on you have done then I begin interaction is to go I give you some data I get some data I give you some more data I get some more data back there is no only way relationship that is establishing you begin after I did. Like a function called there is data dependence also there but that is that you would call task dependence right task dependence is related to data dependence but we typically think of two separate grass a because they have two different times of information.

One is the task dependence graph and the other is the task interaction graph the other is so you have got to design the task say I want to big task small task I want to have this task depend on each other or not why is big task or small task important. So there is some relationship between the task that will be scheduled on the computer if you have lot of task of things to schedule.

But even if you ignore the scheduling time but as soon as you done I am just going to be able to schedule the next task then there are some issues among that you would expect among the bigger task versus smaller task exactly how that is the one issues which is size of the task and that if lots of small task they are all small nobody is waiting for the something that is big we are small versus I take as the computation say rate thing.

Example again I have got one million pictures to produce one million race I take one task which is one array we are taking the bad example because there is no dependence among task but an example of granularity is still here so I can say one task is one rate and I have got million of them there you say one task is one square block on the screen and I have got hundred of them each task as more to do is perfectly balanced in both the cases every task is the same size but in case I got lots of very small task in case I have got small number of big task.

So what would you expect in general and that there will a problem but they are probably a less of problem than other issues that you would have to consider when breaking a computation your tasks it would be equivalent there is no dependence. As a result of which you have got you have to respect all these dependencies right which means that if the scheduling of lots of thing dependent in lots of other things and so you will just not be able to schedule them in the way you might want we have too many constraints and so why not use very large task but not I have got all new tasks.

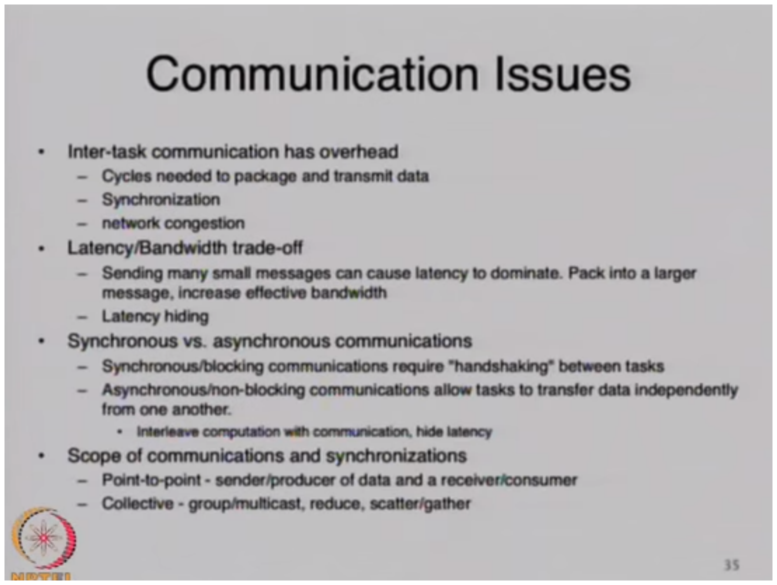
Typically the utilization was done I have got a huge task and some reason the task as to wait for something else because there is a task interaction graph versus you need to produce some data before I can use it right that is also big task and so I will have potential of lots of ideal time exactly. If I have lots of task then I have lots of things that can be scheduled in this down time when I am bust somebody else is ready to run but if there I have got only one task then either I am running or am not right.

And so you have got to figure out how to size the task and the dependence of that is on how well the communication task interaction is going to depend on how you size the task. Dependence will also depend because once you take lots of small tasks and group them into one big tasks then internally it just do the mean whatever order it sees it does not have to worry about somebody waiting for somebody else to give it something okay.

It is being done in sequential sense communication automatically comes with synchronization at some times communication always require some kind of synchronization at some point some time you need synchronization even when you are not communicating explicitly data I want to have done something before I want done something else right barrier you have seen for example or lock where only one of us should be taking the next item for example okay where we are not sharing data directly we are simply we are still sharing a resource.

But we are not directly communicating we are not producing here you do that here simply synchronizing because we want some global consistency to be written okay and then you are done. Once I have determined that these are the tasks that is a good distribution because it reduces communication reduces synchronization then how do I make sure that these task keep the processor with it okay that is the last important thing to consider

(Refer Slide Time: 18:23)



Communication Issues

- Inter-task communication has overhead
  - Cycles needed to package and transmit data
  - Synchronization
  - network congestion
- Latency/Bandwidth trade-off
  - Sending many small messages can cause latency to dominate. Pack into a larger message, increase effective bandwidth
  - Latency hiding
- Synchronous vs. asynchronous communications
  - Synchronous/blocking communications require "handshaking" between tasks
  - Asynchronous/non-blocking communications allow tasks to transfer data independently from one another.
    - Interleave computation with communication, hide latency
- Scope of communications and synchronizations
  - Point-to-point - sender/producer of data and a receiver/consumer
  - Collective - group/multicast, reduce, scatter/gather

35

So we will take all those for issues one after another in communication again the reason that you would be reducing the communication is not just that there is synchronization associated with the task associated with the communication also that communication itself as a way right. There is somebody writing data into some buffer and potentially and typically several levels of buffer.

I write into my software interface API that API writes into the network layer, network layer writes some data on to the physical and so there is several copies typically involved and so all of these copying, packaging, transmission has not over it even if there is no synchronization need I can send any time I wanted and they could receive in the background okay. The other is the issue of how much data can I send there is fixed amount of channel bandwidth available.

And so if I have taken a task that communicates a lot with other task in the bandwidth is going to common issue. Latency is the another issue where if I must communicate something and wait for the result in response to that communication then I must until the data has gone all the way there in accepted been looked at be answered and then the answer as come back all the way here okay. So you will close look at whether the band width is what kind of band width is available and what kind of communication you are going to require is something that depend on latency?

Right for example if you simply writing something to memory that is communication and that is typically latency driven communication right.

If you read something from memory you send the request to memory saying I want that piece of data that request in will be responded by piece of data and only then you can do the next step okay. And so you are going to have to find the ways to hide the latency typically in any program where you either run lots of threads just like you are talking you have lots of task when I am waiting for this thing to do it long trip somebody may be employing the CPU okay.

The other is pre fetching I send this request before I actually going to right so I anticipate that few steps from not I might need this piece of data and then I send this request off and then a few steps later and then I do need that piece of data hopefully that responses come back. So those are two general ways in which you hide latency and it is per fetching directly is not exactly what a program directly controls it is no like pipeline.

Say for example I want to multiply one row of matrix one column of matrix I know that I am going to go through the entire row and then the entire column ok. So I read a part of the row and I read chunks of the row and while I read I have read it meaning I put it in some local variable and that means all those request has been issued. For the second chunk I am processing the first chunk because they are stored in some local variable.

So instead of varying a variable which reads one from here one from there multiplies it you have two variables which reads one from here one from there again we read one from next one from here next one from there and you go the multiplied of first one or the other okay. So you read in and then two is just an elastration you typically do it in larger groups where you read 10 or 16 or 32 or something like that and then you read another 32 and then you process the first step.

Your code is itself structured in a pipeline fashion so that you are reading and then doing something else and then working on the variable you just read okay. No compiler will have no way of knowing this compiler can reorder this the compiler reordering is grey area right you can either way of reordering which is typically not a good idea or when you compiler will only reorder when you sees there is some need for reordering right.

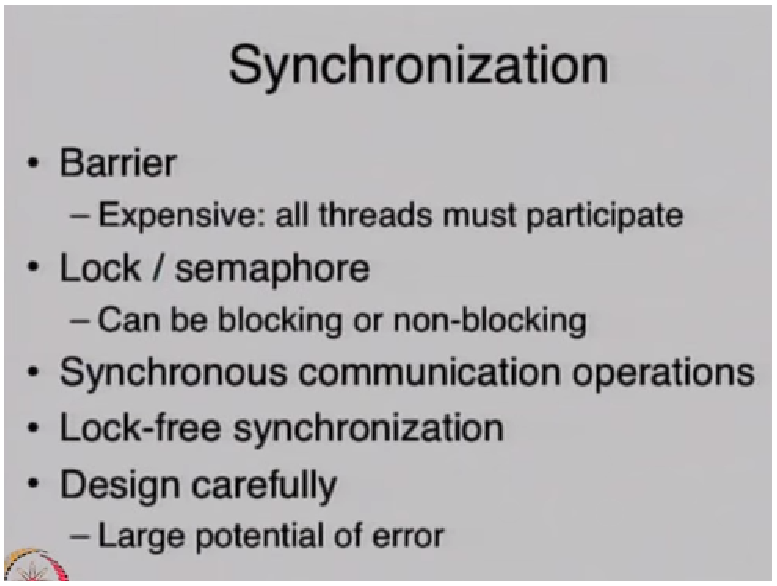
When you typically do this pipeline compiler has no motivation to reorder back to original for you because it that is nothing to gain. So in this case the compiler would not typically reorder. So you tried to keep it has joint as possible there are some highlight thing that you do which would be somewhat API architecture independent but you are going to get the most mileage when you take that into account then you have the package as well as the order.

There is a issue right the so state of the art programming or parallel architecture as well as the standard compiler and programming or interface and all that is not consistent enough across architecture that portability is let me say much more of an issue then its sequential programming. Sequential programming also there are issues not everything that we do on a limits we will apply on windows for pattern okay.

So in terms of the communication we have to look at whether you want t synchronize you need to do synchronize communication where sending data and that guy is receiving only when that guy as received it. Acknowledge the received whether or not the responses coming back can I proceed these are blocking in fact we will see very soon may be look at MPI we are blocking cause where I say send and when the send returns I am guaranteed that receive as in done or the other side versus I send and then immediately it returns and I do not know when the receive is going to happen.

That is the A synchronize kind of communication the other access on this design spaces whether you are going to be point to point or pair to pair communication as well as synchronization as well as group. In multi classes individual send or brought multi caste is not always available but broadcast may be similarly in the synchronization space also either you are synchronizing with a specific processor by sending a message of by locking shared variable that you and that specific processor shared of something like barrier versus of all of these group of processor together are being synchronized in a common way okay.

(Refer Slide Time: 27:20)



Synchronization

- Barrier
  - Expensive: all threads must participate
- Lock / semaphore
  - Can be blocking or non-blocking
- Synchronous communication operations
- Lock-free synchronization
- Design carefully
  - Large potential of error

Talking about synchronization typical things you would be using would be barrier. Barrier are relatively expensive there is also variant of barrier called memory barrier that some architecture support or where you basically say that all the operation to that section of memory has been completed it has more to do with the memory consistent issue we talked about before any other memory operation that happens after the barrier okay.

And they tend to be bit more light weight than compute barrier lock come of force you probably have seen in was courses as well communication is another way to synchronize when I return from send and I know that the receiver has received it some synchronization has happened. I can assume that something that the receive everything that was happening before the receive has been done.

I will also talk about synchronization that does not use lock it is called lock free or there is also a looser variance called weight free synchronization which can be efficient in many situation not always in blindly efficient but when we have your thinking of what synchronization method to use where to apply the synchronization it is generally a good idea to begin that being conservative okay.

Because synchronization is the single larger source of error parallel programs have so it is better to if you think synchronization may be need you synchronize and then once you have figure out the things are correct then you may do a second pass in worry about performance and if you get struck with the wrong synchronization a bug which is because of the synchronization there is tend to be a extremely hard to trace and you might want or you might have designed it to be great in terms of performance but you never be able to be run up.

If you do not find them so it is better typically to start by at least has a beginning parallel programming start having preservative get it right and then try to remove synchronization where you might think it. It does not you do not lock shared variable we will talk about it later on right now you assume on lock synchronization. So any time an in fact has a general rule basically strict with locks unless you have showed that you your program needs something else.

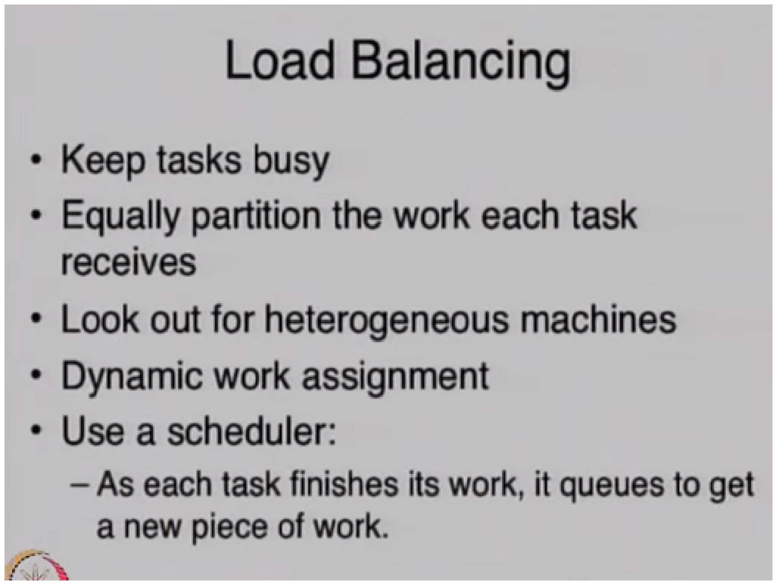
Whenever you need synchronization you first thing you should think is locks okay because it is recently efficient and generally would fit all kind of situations and so if you are not sure then use locks and then and a second pass you figure out what it is exactly that you need okay let me stop here so that we can start the section are there any questions on again we are talking about generic section right now.

So should not be too hard to follow no not to this is just no this is for all model whatever your underlying programming model these kinds of these rules apply equally well right. Whichever model may be you have to break it into tasks you have to take these task and then you run it somewhere right. So you have even if so that it is task a task is not necessarily breaking the entire program into task right.

You have I have got this so what is said was you start with the sequential program and this is section that there is hard because it is taking 90% of it time. So you will leave of that potentially sequential there is no much to gain like in the effort that you will put in that section right. So all of that is kind of untasks right it is just being done on sequential in one of the processes and there is this section which either because you are dependency can only begin after all of that has done or may be can be primitively scheduled early okay.

And that is the section that you are going to think about how you take that section and break it in two independent units which way interact with each other but these are still tasks okay. So when you are doing that for a loop that is what you are doing you are saying that is sequential this is sequential but let us figure out how to do this section parallel okay.

(Refer Slide Time: 33:12)



Load Balancing

- Keep tasks busy
- Equally partition the work each task receives
- Look out for heterogeneous machines
- Dynamic work assignment
- Use a scheduler:
  - As each task finishes its work, it queues to get a new piece of work.

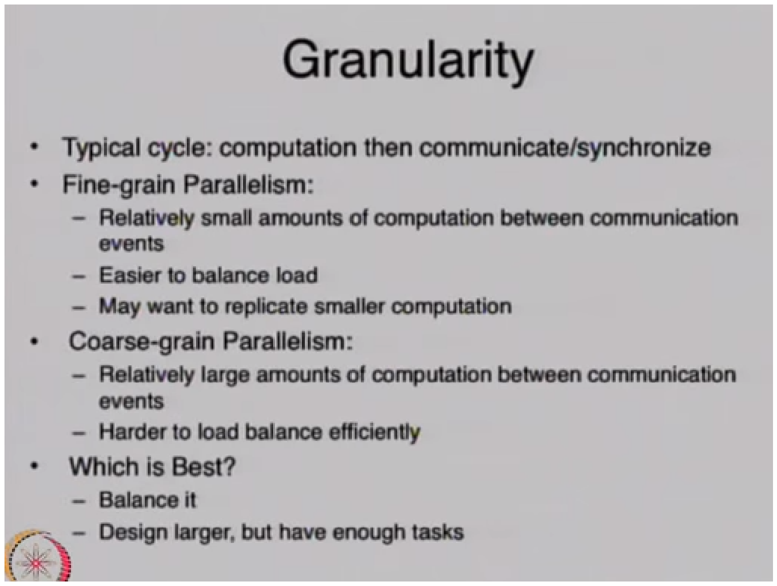
Alright for load balancing we will go one level deeper in for each of these slides very soon but at high level you basically half the load balancing is to keep the load balanced right. Keep every processor occupied all the time if the task are similar sized makes load balancing little bit easier but typically you will not be able to be task similar size and it is not even necessary for load balancing to require to be the task being same size okay.

You can load balance even when the task are dissimilar in size you can balance load even if the machines are similar in size need not have all the processors having exactly the same amount of power right. In general you can think of it has some variant of an MP complete problem that you have got lots of these tasks that you have figure out and then figure how to allocate them process that the processor are kept with okay.

So that basically is variant of an (()) (34:36) okay and more over not that (()) (34:42) problem is easy to solve but may not even know what the task size is and how long it take to run like you have (()) (34:50) got to know what size each thing is right. Task size may be vary dynamically so you have got to keep the load balance dynamically you have as the task each taking more time and you have to figure out what to do where the processor is become ideal because that task is finished.

And again that high level that trick is keep enough task in hand if some body become ideal I will give another task to him.

(Refer Slide Time: 35:23)



Granularity

- Typical cycle: computation then communicate/synchronize
- Fine-grain Parallelism:
  - Relatively small amounts of computation between communication events
  - Easier to balance load
  - May want to replicate smaller computation
- Coarse-grain Parallelism:
  - Relatively large amounts of computation between communication events
  - Harder to load balance efficiently
- Which is Best?
  - Balance it
  - Design larger, but have enough tasks

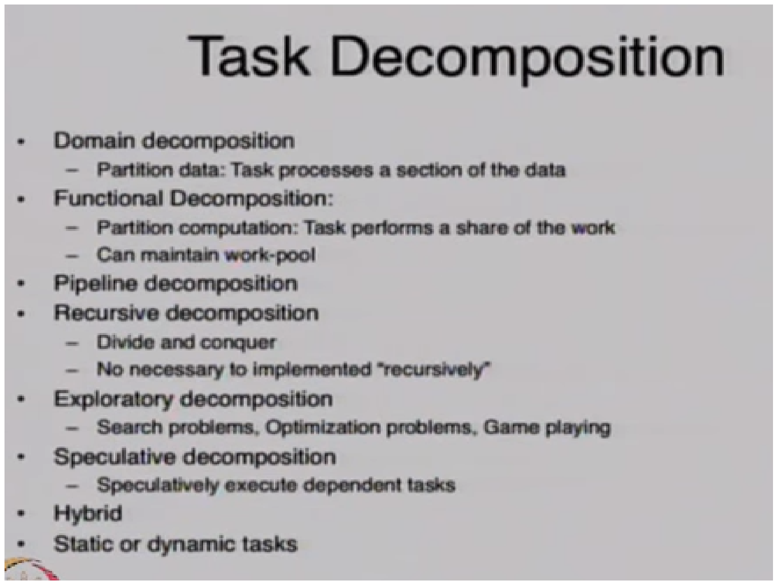
I think we are more or less talked about all that is there in terms of granularity the tradeoff is between fine grain as well as course being parallelism right it is fine grain it is typically to be able to keep dealing processor down as much as possible but dependence is an the amount of interaction can go on.

The other thing and so what listed here is also in fact is also is used as a metric which is the amount of computation divided be the amount of communication or the time spending communication per unit time of computation which is one of the metric that if you recall the BST model okay.

So when you have small task when you typically have that thing go down and you do small amount of computation some communication plus there is much more of communication happening and in large task you can even if all of that data is still is being shared by ensuring is shared locally as well as ensuring that it some more right send lots of data together rather than independently you can reduce the communication overhead overall for last task.

And again there is no one answer for whether you should be using the small or the large task right it typically is a you want to keep it as large as possible but the problem with large is that load balancing is going to be an issue. So you got to have enough of them on hand so you start with big and keep reducing until you got enough of them on hand and so you start with big and keep reducing until you got enough that is the one rule of the thumb.

(Refer Slide Time: 37:24)



Task Decomposition

- Domain decomposition
  - Partition data: Task processes a section of the data
- Functional Decomposition:
  - Partition computation: Task performs a share of the work
  - Can maintain work-pool
- Pipeline decomposition
- Recursive decomposition
  - Divide and conquer
  - No necessary to implemented "recursively"
- Exploratory decomposition
  - Search problems, Optimization problems, Game playing
- Speculative decomposition
  - Speculatively execute dependent tasks
- Hybrid
- Static or dynamic tasks

The decomposition of task one is domain decomposition what we talking about earlier take output or input out of some intermediate stage and you simply take that and divide it equally or non-equally that is not a matter you just divide. Typically you divide equally make large number of revision that you have more task that can use to fill up an ideal processor okay.

Functional decomposition is again we looked at examples if it is clear from the task that you are doing function one and function two and function three then and they can did in parallel then you do them all. So this is function decomposition and then you typically try to have as many functions as you can and enough functions so that once function finishes and there is NT processor side somewhere when you have some function waiting to be compute okay.

Pipeline to composition is another example to where you the underline algorithm automatically says you got to do this then that and then there is a sequence of data that goes to the same. You can kind of think in that way but not strictly okay not just because the decomposing domain does not necessarily mean that you are doing same thing on every piece of data is that look at the structure of the program and it says that I have got these sequences of data item to process.

Then how to process the each data item is not known right it and be different is more type of function. And in some sense there is this two kind data decomposition function decomposition and then these are you can see pipeline typically in function but you can see in data decomposition also because basically saying send this data to that and then that data to that.

So I have made you this slide should be structure slightly differently domain N decomposition function are two categories and then all those functions are techniques how you might do domain decomposition or functions decomposition. Recursive decomposition will typically again come from some kind of recursively provided algorithm divide and conquer you would typically not want your entire parallel program to recursive because recursive is supposed to have an overhead with sequential also with parallel will only goes on okay because we have to recompose it so many times.

But because original function is defined recursively does not mean that actually follow the same recursive algorithm in parallel also although you can do the same static recursion. This recursion would have said that gets broken in these two which gets broken into these four with gets broken into these 16 say okay. That means I have those 16 that will get divided okay specially if it is static recursion sometime recursion can be dynamic in this case you this recursion for few levels but at some point you stop following the recursive algorithm function.

And at the point you have got enough task say now the remaining you do not divide and call have you do divide and conquer but you do not decompose that divide. And there are more esoteric kinds of decomposition one is exploratory decomposition which is typically in a search type algorithm where or optimization type algorithm where I am looking for a solution and I do random works in so in a sequential case you will say do that then do that and then evaluate something okay.

And here you can say do this five things in parallel or do this ten things in parallel and then evaluate something and do this next from their you explore more okay and so again it is automatically coming from a style of the original sequential algorithm itself ah speculative decomposition is where it is typically in the dynamic setting where the task you do not know where the task their will be.

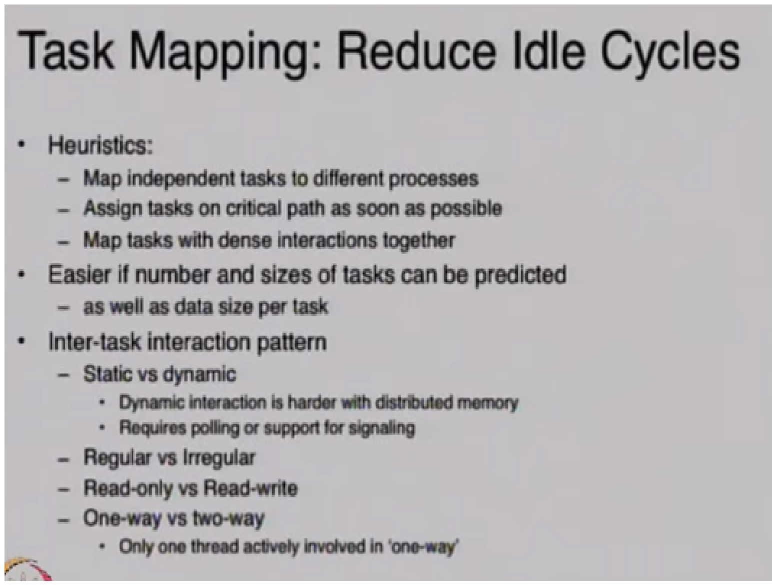
Task gets generated you do something new task get generated and you speculatively decide that in the beginning I have nothing I start and then I will do imagine recursion I will do something and then generate three nota task then I will take this task and expand it further will generate it five more tasks I do not but that is what happen then I will take the second task that will generate ten more task.

Third task will generate two more task so I might say instead of my running all the tasks and determining how many sub task to generate let me generate if I can all the subtask that might have been generate run them and then once that parents task is complete I do not know which of this sub task I actually needed to run take the results and then start the remaining once okay.

You typically do this in the beginning in a good app stage of a program but you do not have enough task and so either sit wait do nothing or I will speculatively do something else and if it is gets used okay does not get used I would anywhere been sitting ideal. I may be point some and there are hybrid decomposition technique where you do recursive somewhere and then once you have got to a big enough task and then you do some pipeline and some more domain decomposition and some more point and whatever.

And the thing that to be careful about whether the task are known statically or being generated dynamically okay.

(Refer Slide Time: 44:36)



Task Mapping: Reduce Idle Cycles

- Heuristics:
  - Map independent tasks to different processes
  - Assign tasks on critical path as soon as possible
  - Map tasks with dense interactions together
- Easier if number and sizes of tasks can be predicted
  - as well as data size per task
- Inter-task interaction pattern
  - Static vs dynamic
    - Dynamic interaction is harder with distributed memory
    - Requires polling or support for signaling
  - Regular vs Irregular
  - Read-only vs Read-write
  - One-way vs two-way
    - Only one thread actively involved in 'one-way'

And so now we have figured out what this task are if they are static they start with the set if they are dynamic then start with a small set with new task being generated as we get done with this old task okay. How do I decide which task to run over that mapping problem mapping of the task so the first two stages are decompose and then map decompose into some task or task if you were and map with where to run the task.

And typically you would not in fact have said that many you would not just have the as many task as number of processors you have live okay. So if you have K processors you will have 4, 5 K live task right which means you may in fact have many more task generate such that 4, 5 K live can be kept at a time okay and once somebody has nothing more to do we have got task waiting for them to do okay.

So you would generate if you have P processors well in 20, 30, 40 P task to be generate okay. Now we got some 30 P task P processors we are going to begin with some 4 or 5 P of them among from the 20 P so which 4 or 5 P done first okay because you want to keep that many live you want to keep several task per processor live and you have to keep some waiting so that once a task gets done you can replace it with them.

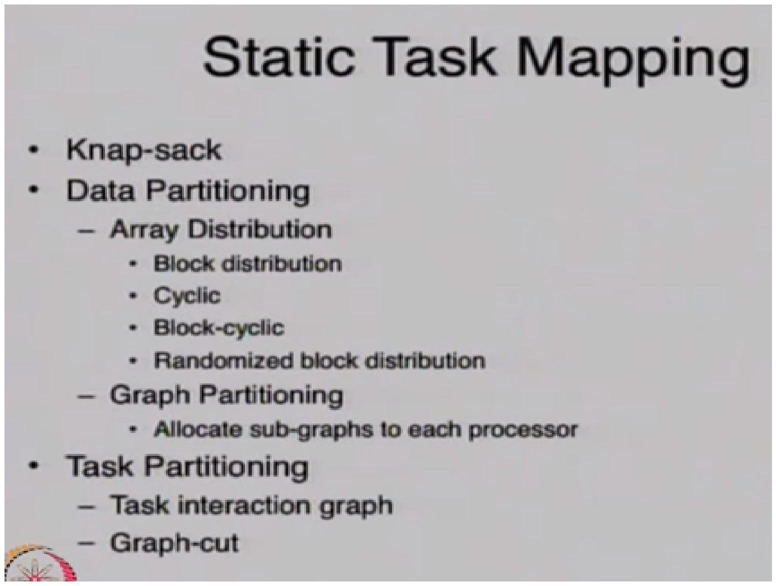
And so if you have got that many extra task and you have to choose a set which you to set running and that set also to be grouped into P groups and you got lots of it is an optimization problem to solve from the bit set of about let us say some number 20P I am going to choose 5P and from the set P I am going to break it to P groups okay. What are the parameters what would you be looking at you want a group such that it does lot of data communication if two task share a lot of data then that should be in same group.

And they should probably be launch together okay so communication at this point look at the task interaction graph and you will get which are the task that will interact with each other much more than other you club them together not to the same processor again the task map thing is easier if you know the size of the task then you can load balance if you know how many task there are if it turns it to a dynamic situation may task are being generated on them some task runs and then response from your task.

Then it becomes extremely hard to properly map them their have not result in specific types of mapping patterns rather communication patterns which are often found in application. Examples are grid communication only communicate with two or three neighbors of you or some kind of a restriction graph communication or communication where finally read your stuff but never write another thing or write something to you.

You both reading to common shared data so in this restriction environment people have worked about a mapping algorithm well that not going to get into detail of these techniques but if you are interested because again it will work only on restricted set environment. But if you are interested then you can look up these the paper objective.

(Refer Slide Time: 49:30)



Static Task Mapping

- Knap-sack
- Data Partitioning
  - Array Distribution
    - Block distribution
    - Cyclic
    - Block-cyclic
    - Randomized block distribution
  - Graph Partitioning
    - Allocate sub-graphs to each processor
- Task Partitioning
  - Task interaction graph
  - Graph-cut

The mapping itself is as pointed out knap sack probably which got to figure out somehow with some heuristics to arrange. And again for restricted types of this is related to the previous slide if they are restricted types of communication patterns as we well as dependency patterns then you can do mapping using those specific patterns especially these patterns are data dependences right.

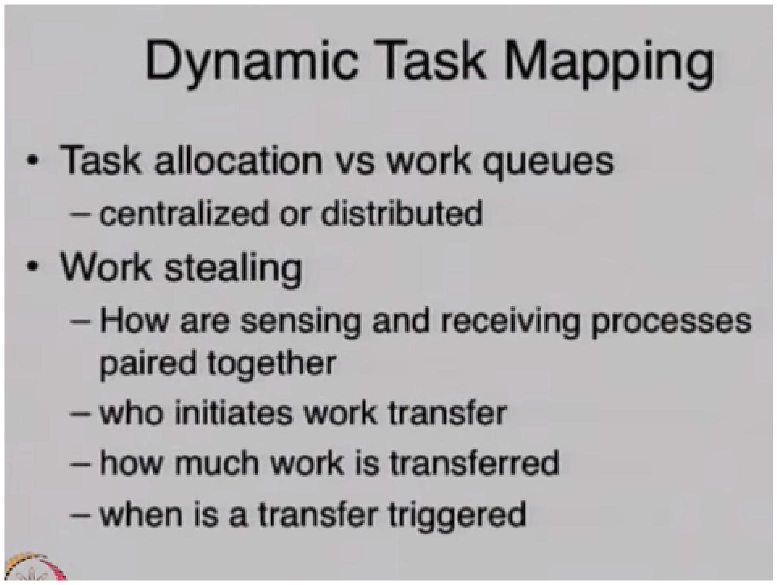
For example you say I have a task decomposition that is a domain decomposition based and I have got a 2D array of data that we are going to all be processing then I will simply do decomposition block I will take this block and that all the threads of that block become map together okay and then there are various refinements over decomposition where you say instead of some situations you may not want the entire block together because communication patterns across block and I and then you turn into cyclic block.

I will take one block you take another block every person takes that block then I will take the next block again. So instead of having the entire thing broken into P blocks where each block in divided into the task you break it into a fix size block but then cycle through the blocks first block goes to first processors second blocks goes to second processors and so on. You can also do it based on only if other static case based on the task dependence graph also.

You take the task dependence graph and you simply cut it into as many processor at K times processors. If I want to break it into 5 P task or 20P task it create 20 piece of graphs such that the ages that go from sub graph to another sub graph or minimize or the wait of this as also waited version where some dependencies are given higher weight because either communicate lot of data or synchronize more often about it.

And so it becomes a graph cut problem we have given this dependencies graph and you are going to cut some a ages such that A sub graph or generated okay and you want to cut the most number of it ages.

(Refer Slide Time: 52:29)



Dynamic Task Mapping

- Task allocation vs work queues
  - centralized or distributed
- Work stealing
  - How are sensing and receiving processes paired together
  - who initiates work transfer
  - how much work is transferred
  - when is a transfer triggered

For dynamic task mapping you have essentially one category or let us say two specialization of algorithm where I have got essentially a group of task maintain in a queue and the task being allocated to whoever has finished this approach works finished even if we were into have static task although the overhead of managing queue a is something that you can avoid if you do not do this work you based optimization.

And so there are two specialization is centralized is received where there is a central queue somewhere and everybody says I am done give me one task from the queue so there is some synchronization involved and I am going to get involved and do that and then somebody as to maintain this queue so when I generate more task they must get added to that center in the dynamic constant okay.

The other variant is distributed where you say everybody is intimately allocate some and they are dynamically generating more potentially and where I get done I simply go steal somebody task it is called work stealing algorithm and this is supposed to be one of the best techniques for load balancing okay. Again they are details comes off whose work should I steal and then how should I check which order should I check whether I should steal his work or not.

And when I decide that I check with someone and there is lot of work remaining there how much of work should I take on maybe there are others also waiting to take on some more work. So if I take all other work then others I have to steal from me so there is more overhead okay. And so essentially the overheads pair to pair so I decide and one common work is that randomly I simply say let see whether you have some task remaining and if you have I am going to take some function of the remaining task okay.

So let me stop with that at this point these are the high level techniques I have got few more slides which I am not going to go through because it is general program tips which is essentially a summary and somewhere the stuff we have talked about there are 2 or 3 slides these are just general parallel programming tips we have to keep in mind everything we are talking about today is not about open MPI but about parallel programming in general no that was also right.

And here those are in capsulated there are three slides here tips 1, 2 and 3 it will be posted in the website you just review that I will repeat one more time that it is at least in the beginning when you are not at an experience programmer focus and correctness before you focus on performance otherwise you are going to spend a lot of time on debugging and debugging support in parallel context is not that great so it is bug for most part.

And so it is very important that you take conservative decision first and then gradually improve performance as you might any questions I will just flash one after the another in case if you want to ask the question right now alternatively you can ask any question that you have next class do not use lot of this is again something that is generally a sequential program also but even more parallel do not use lot of new or malloc free types.

You do not want to be memory to become fragmented in the sense of the threads working set you want the threads working set to be contiguous as much as possible. It is not based on architecture use threads safe library thread safe library is just somebody is built for you that works even when multiple threads are calling that function. So malloc and new in the thread context takes more time because it is thread safe.

Correctness before performance they take more time because they are thread safe but the thread safe meaning that even in the context thread they are guaranteed to give you correct results. They may be but there is a cost to be paid for having thread says and that cost what you have seen any other questions okay. In that case we are going to stop here.