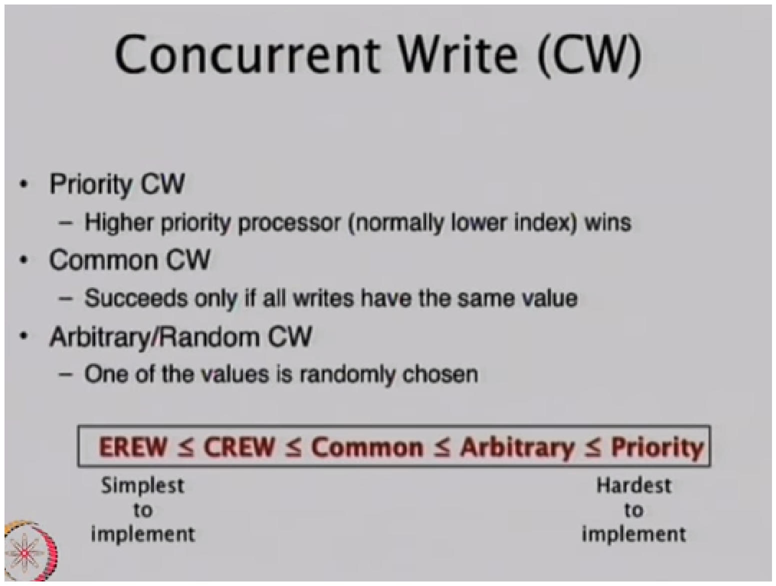
Parallel Computing

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

Module No # 02
Lecture No # 09
PRAM

(Refer Slide Time: 00:26)



Concurrent Write (CW)

- Priority CW
  - Higher priority processor (normally lower index) wins
- Common CW
  - Succeeds only if all writes have the same value
- Arbitrary/Random CW
  - One of the values is randomly chosen

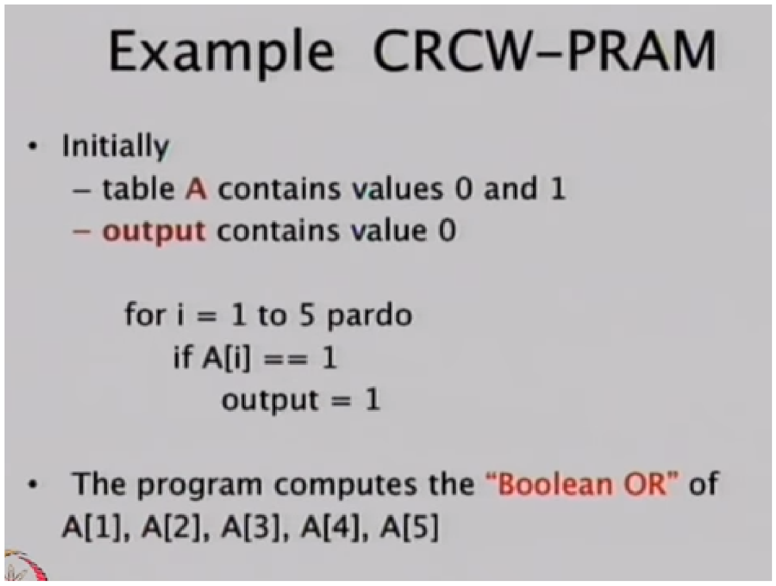
EREW ≤ CREW ≤ Common ≤ Arbitrary ≤ Priority

Simplest to implement — Hardest to implement

The concurrent right architecture and did you figure out what the common program would look like okay explain so again I will repeat first you read what was there okay. Just in case if you need to restore that everybody do well then the right instruction happens okay and somebody will win but the remaining the lower priority once can read it back and tell that something was got written was not the same thing has that they wrote okay.

So they all know that it needs to be unrolled undone how do they undo it they all have old value they are all going to write the old value once of them will still win okay. Whoever among them is higher priority okay and old value will get written now everybody is trying to write the same thing okay. Actually now that sometimes simulations can happen will take more steps priority can also be simulated with common and not in constant term.

(Refer Slide Time: 02:52)



Example CRCW–PRAM

- Initially
  - table A contains values 0 and 1
  - output contains value 0

```
for i = 1 to 5 pardo
    if A[i] == 1
        output = 1
```

- The program computes the "Boolean OR" of A[1], A[2], A[3], A[4], A[5]

So here is simple PRAM program we are not doing initialization over here how the input got in how processor got active and all that. This is CRCW PRAM model and you are basically checking right the code in the middle says and this is pseudo code see the same thing that would have seen in a RAM model pseudo code only does have become those in parallel of pardo okay.

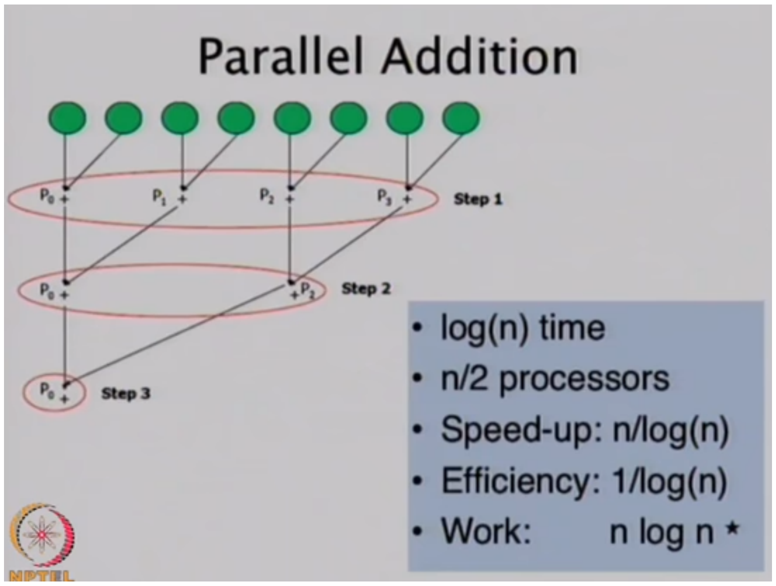
So it says I = 1 to 5 pardo if AI = 1 then output 1, AI is a shared memory array I is the processor now right. For every I is very similar to open MP style for Loop also work share right. So every single is on every single processor and the read exclusively they are reading different areas in A this is CRCW but they are allowed to be exclusive if that value turns out to be one then they are going to into one location out which has be agreed to Y all the processor the value 1.

So in CRCW model and the goal was as it says compute the Boolean or of all the things and that Boolean or will appear which CRCW model. The value should be 1 if anyone is 1 okay I will repeat the question the program in the middle it is says that all the processors are assigned early ah based on their ID one of the locations of A to check that location of A and that location of A happens to be 1 they write one into shared memory called output shared memory cell called output okay.

So let us look at each of them I have simplest is random right somebody results is going get written nobody trying to write a 0 right. So if your value is one you write it otherwise you keep quite so if people who have one trying to write somebodies results got written and somebodies results must get written and then says I do not whose results is get written but somebody's is going to get written one will appear okay.

In case of priority again one specific fellow will get written whoever priority is highest still we want okay and the third everybody who is written will be writing one okay. So whatever the model may be is going to complete the Boolean R okay that is not always nice is that but in this case it is.

(Refer Slide Time: 06:37)



Parallel Addition

- log(n) time
- n/2 processors
- Speed-up: n/log(n)
- Efficiency: 1/log(n)
- Work: n log n *

Okay we are going to add a few things together okay I got N objects values in N locations in shared memory and cells in the shared memory and am going to assume order N processors. How can I add in EREWT RAM exclusive read exclusive write okay so everybody has and there are N things everybody has read one of those N things then what happens two processors will read the same value that is correct.

But then proceed you want to start with N by 2 processors of A two values which two values? So you read I and I + 1 / 2 okay or I into Y 2I + 1 sorry 2I into Y + 1 that might remind you some okay. We are building a tree on this array okay so you read N things and everybody is going read two things they are going to submit together and where will they write they can overwrite assuming that the input is not needed more the input write in one of them I by 2 you want write in I by 2.

If it is needed means you cannot overwrite then you will write in some temporary condition but this time keep doing this on temporary location or for that matter you might say you are going to just copy over the original array into temporary location and do that summation in the temporary location okay. So that was one step in one step everybody read two things some did wrote one how many such steps will be needed log on right.

Again the place to read from right to confirm the keep structure at each level you know what your children values is R index is R. At the end of it you are going to have one processor is going read two things and write one thing okay. At each level some processors will be turning them so on okay how the simply check the level right and each time you can keep reducing the number right okay.

And you can do it in many different ways so there are log in steps involved yes right so that is the technicality right you can say that the processors were not reading one of those are reading from one location called infinite okay and they are writing into location called infinite. So in that detail is not necessary to manage as a part of the program itself in fact even if the processor does something it has to write some it might read do some computation read more does not have to write something okay.

So that I think probably is the good thing to mention that all these steps are optional which means all they are reading or doing nothing what you cannot do is while someone reading and somebody is writing and somebody else is doing some local computation. Alright so the time be will log in the number of processors we have used is N by 2 first of all N by 2 but it then kept going down but the maximum number of processors at any step was N by 2 so we have used N by 2 okay.

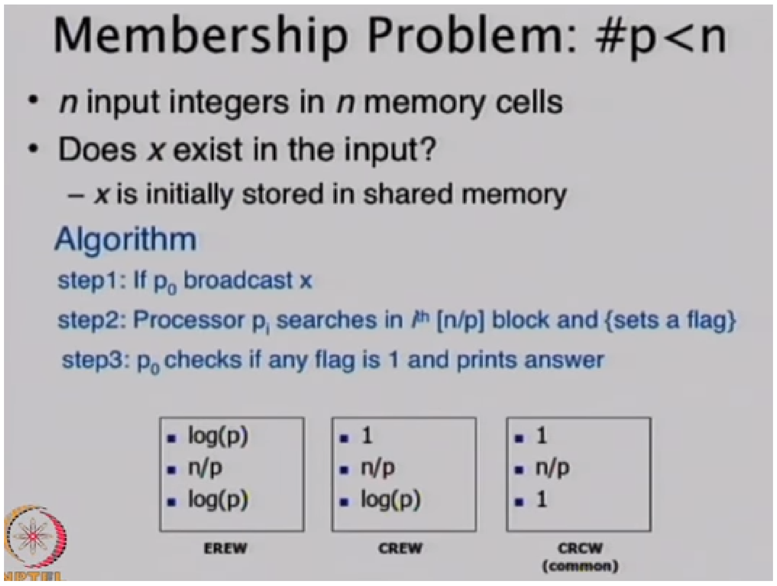
The speed up what is the sequential thing to add N thing N step right N -1 additions if you compare it to that then you have got N over log N speed up so these are the few remember from earlier slide and the same statistics. Efficiency is speed up over number of processors okay and we have used N again this is order we are not worrying about N and N by 2 the work done is the number of processors times the number of struts.

So work done is N log N and that is where we are going to now figure out what happens if i have P processor not N okay we did N operations but could not have done all the N operations at the same time using N processor. The work is the total number of steps * the total number of processors yes so this is not the summation of work done over time this work done says how much parallel work has been done.

But they could not have any other work if there was some other work to do then they should have been doing it right we are talking about context and all. Talking about that machine and in this program and at some stage because your algorithm is going to blow up computation you can do lots of processors then keep it ideal for the long time which object to algorithm okay. So if you can manage keep the processors one at every step keep all of them busy probably going to get a lower work done.

And the work done is going to be the most predictive statistic when you run in it on actual machine only P processors okay we will talk about that in more detail but let us look at the other algorithms.

(Refer Slide Time: 14:51)



Membership Problem: #p<n

- n input integers in n memory cells
- Does x exist in the input?
  - x is initially stored in shared memory

Algorithm

step1: If p_0 broadcast x

step2: Processor p_i searches in i^{th} [n/p] block and {sets a flag}

step3: p_0 checks if any flag is 1 and prints answer

EREW	CREW	CRCW (common)
log(p)	1	1
n/p	n/p	n/p
log(p)	log(p)	1

This is the member problem I have got N things written into N places membership research problem okay. And I have go a query I have got X exist in that input array okay and we can write it at high level so in the first step everybody gets value X okay. In the second step everybody checks if the X exists in some part of the input and in the last step they are together going to decide somebody found it okay.

Let us look at each of these 3 steps and the different models they are looking at in this case EREW. CREW and CRCW common okay not all of it yeah. But yes the last step is so the first step is broadcast so that everybody knows what we are looking for. Because in EREW X cannot be read by everybody in the same yes for concurrent read everybody will read X in the same the first step okay in CREW and CRCW that cost is written as one out of one.

P is the number of processor with number of processors in the case been given value rather than N. Exactly we are going to now move towards how do you take this models out of program in this model and talk about a real implementation of this model okay alright. No we are going to have come up with the algorithm to do it model has no notion at broadcast. We are not going to assume anything about the interconnect all the information is shared by writing to some place and somebody else reading it which takes each constant number of time.

Interconnect is exactly connecting it to a shared memory but it is hidden never directly say talk t the interconnect we never say talk to the processor PRAM there is no send receive type instruction that is what we do in distributed memory model in the shared memory model it is a strict pure memory model you write to a common place you read from a common place okay. So how do you get X to every processor knowing that they cannot all read the location of X in the first lock.

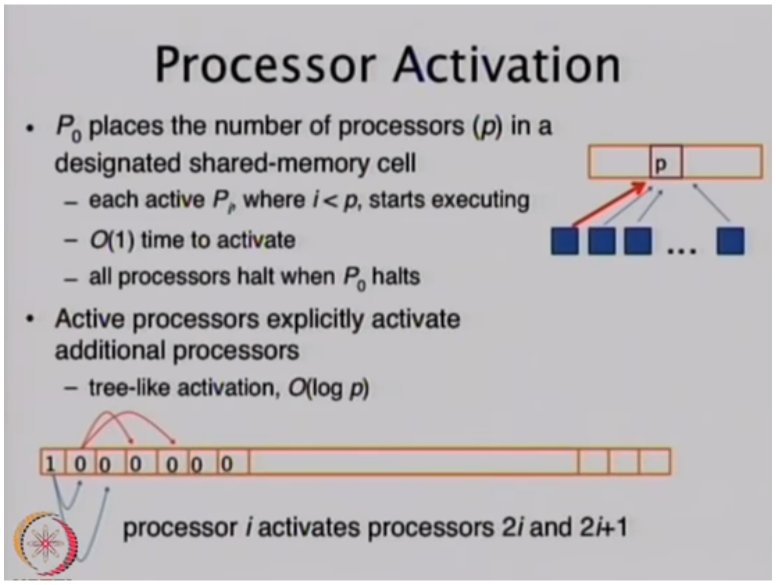
So the question Y is that log of P * 2 so the first processor can read X and read into you are not carrying about how many how much space you use right. So we are plenty of space shared memory space just allocate some temporary area and the first processor read X and write in one more locations. Now we have two places where X is now to more processors we will read those two things any each will write one more.

So now we have X in for loop okay next step will have X in 8 locations so how many steps to get into P locations log of P right the local computations we have determined that every processor will take N over P section of the input and search for X in that right so in over P steps will read in any model exclusive concurrent all models will read from their exclusive part of input check if that value = X.

If it is not leave a flag of otherwise turn on the local plan and then keep doing this on all the locations okay. So at the end of it somebody may have found it possible that no run of (()) (20:22) and they have local variable called F is local memory as much as I. And now everybody needs to agree need to come to an conclusion about somebody got a plan. I am not taking any mechanism I am giving 3 examples.

You initialize it is 0 and somebody has found it and they are going to write it as 1 all of it that R work for every single variable of all concurrent write okay. And so here we do not need to worry about with type of concurrent write it is so although I have said common it would be out of one in all but how about many you have exclusive right which is both first and second column reverse of the log view right instead of broadcasting you are collecting.

(Refer Slide Time: 21:57)



Processor Activation

- P_0 places the number of processors (p) in a designated shared-memory cell
  - each active P_i, where $i < p$, starts executing
  - $O(1)$ time to activate
  - all processors halt when P_0 halts
- Active processors explicitly activate additional processors
  - tree-like activation, $O(\log p)$

processor i activates processors $2i$ and $2i+1$

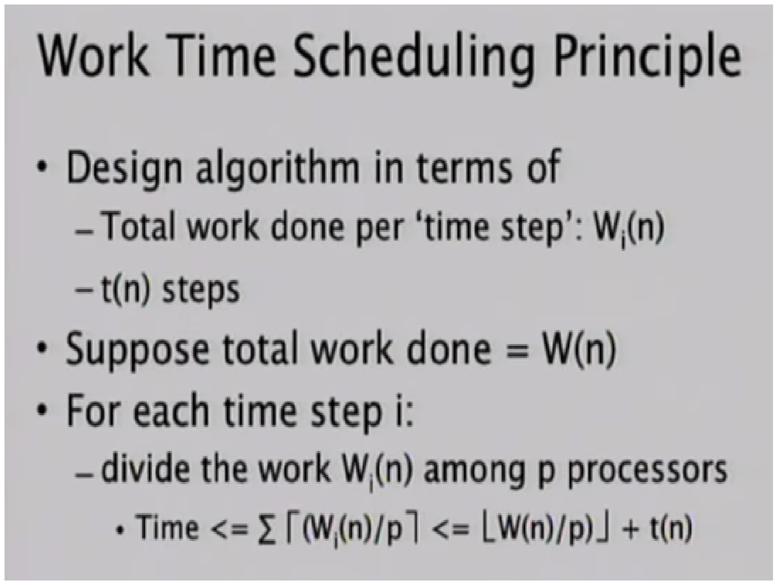
It is very similar to add you are not adding but saying compute all of these things okay similarly you can think of the activations we said there is a add protocol that start protocol can be done in same way we have the starting process which that says the ID is 0 in steps just like we did log reduction or log expansion. In log P steps can tell everybody to become active.

It tells to the active every processor is reading right so they do not where to read they do not know where the input is we do not know whether program is running or not so we are just reading from arbitrary location and processor 0 is going to now know that actually running the program that data that input etc. So not it is going to tell that whatever it has learned to everybody or else like everybody broadcasting you all start working on this part okay.

Alright so we look that arbitrary location is where you write so there is some hard wire things in the model that everybody is reading location number 0 if it is an exclusive if it is concurrent read allowed in this case you write 0 to or you write something whatever you want everybody to know location 0 and every reads to next part. If exclusive write is allowed everybody is reading from their own respective locations cells you got to write that new value all of okay.

That to read so they read together so if you are calling alternative cycles then am not sure they are reading from some location which will be filled with some input at some point which says now the programs over there or now the data is over there or over there okay and so that sixe is going to fit fill in log N steps log P steps if you have P processors and if you are exclusive read model then it takes log P2 initially they do or some okay.

(Refer Slide Time: 25:11)



Work Time Scheduling Principle

- Design algorithm in terms of
  - Total work done per 'time step': $W_i(n)$
  - $t(n)$ steps
- Suppose total work done = $W(n)$
- For each time step i:
  - divide the work $W_i(n)$ among p processors
    - Time $<= \sum \lceil (W_i(n)/p \rceil <= \lfloor W(n)/p) \rfloor + t(n)$

Any questions on the basic of the model? We are running out of time on this session work is what exactly number of processors * number of steps is and so now suppose I have got an algorithm and in each time step let us go into the next level detail. Each time you are doing some amount of work that work is how many processor where active okay.

In out summation for example the first step N / 2 work was done second step only N / 4 amount of work was done then the third N / 8 amount of work was done right if you are going to count that amount of work done with N processors and now if I have P processors I know that any amount of work that was done in one step was done in parallel for N processors meaning that P processors can also do parallel what they cannot do is do some of that or some of this that may depend on each other okay.

So the P processors can only take those N things are some functions of N things that was done in one step and allocate it among themselves and do it among WI things was done at step I. P processors to each stage WI over P of it and do it okay when I actually have a P processors not whatever a input is assumed how of the algorithm is assumed okay now let us added together over all the steps that is the amount of work the P processor PRAM will actually do.

That is also says that is the number of steps they are going to take because what was done in one step by WI processors is not taking WI over P steps. There is only P processors to share the code okay so you add it together and the summation over WI is P and each step this takes a healing right because if there is one processors if there is ten things to do and if there is 3 things to do then there will be somebody will be taking four steps okay so that things got done in four steps.

And the some of that is equal to W over P W being the summation of all over plus the time 9 is the number of steps that the original PRAM algorithm took with however many processor is assumed why at each step it is saying I am doing WI over P ceiling so when I say when I take WI over P an add it together inside but Y get P is the same in the denominator all the WI is get added with total work right total work divided by P but how much extra I might have added in WI over P ceiling.

At each step I could have at most added one right one minus (()) (29:02) by taking it to the ceiling and we did this TN time because the original algorithm took TN steps so we have added N more things. So tell me what is WI it is the sum of WI it is related will talk about that also or maybe you will have a question on that okay.

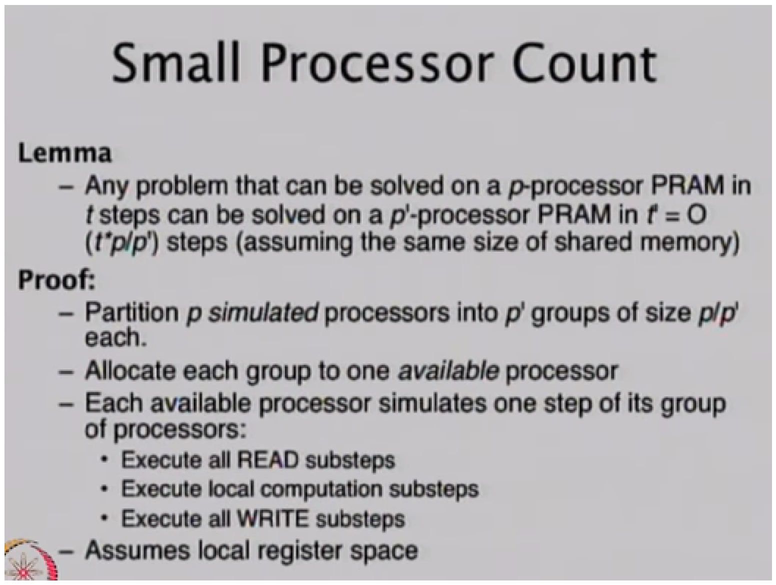
What we are going to do next week is auto simulate a general processor PRAM with the PRAM which is this was just an analysis of how much time it will take we will talk about an algorithm we will blindly take and N processor program or any size processor program and run it on a P processor PRAM. Similarly we will take any processor any program that runs on N cells and run it on N prime cells okay.

If I have only limited memory and only limited amount of processor I can still simulate whatever the PRAM algorithm simulate did at the cost of number of step I have put but it can be done good question find out. Alright so we were discussing the last time this notion of work time scheduling we will come back to it I have not finished talking about it but we will relate it to some of the other things will talk about shortly so am going to bring it back when makes sense.

The first couple of things to look at it is that theoretical level now where we are looking that PRAM make sense the said the infinite amount of memory is infinite amount of infinite but as many as you need some arbitrary number of processors unlimited number of processors and unlimited memory is that handicap? Meaning with bad assumption am I breaking the effectiveness the applicability of this model okay so one of the theorem we are going to prove or MI we will is that suppose if I give you PRAM algorithm we uses some F of N processors okay.

In general P processors so according to that level abut I want to run it on another PRAM that only has prime processors okay you cannot less than P could I do it same amount of shared memory same amount of local memory in fact we do not say anything about local memory. Local memory we are in fact going to use as much local memory need in order to do this simulation so we do not restrict to local memory and make sense because we can make local memory quite big of the RAM model assumes infinite local memory and there is quite useful.

(Refer Slide Time: 33: 08)



Small Processor Count

Lemma

- Any problem that can be solved on a *p*-processor PRAM in *t* steps can be solved on a *p'*-processor PRAM in *t'* = O (*t*p/p'*) steps (assuming the same size of shared memory)

Proof:

- Partition *p simulated* processors into *p'* groups of size *p/p'* each.
- Allocate each group to one *available* processor
- Each available processor simulates one step of its group of processors:
  - Execute all READ substeps
  - Execute local computation substeps
  - Execute all WRITE substeps
- Assumes local register space

So if I have only P prime processor I will give you P processor PRAM logarithm how would that P prime processor PRAM run this algorithm kind of like context each processors are in fact lemma I should be telling you any problem that can be solved on the P processor PRAM in T steps can we solve on the P prime processors PRAM in order T prime times P / P prime steps okay.

Essentially what you are going to do is just have each we are assuming that P prime is smaller if is bigger than P then it is not much to be done we just have the extra processor to be in (()) (33:37). But if you have fewer processor then you need when you say P processor then you need P prime that I have and going to have each processor act for be a proxy for P prime over the P of the original processors okay.

So we are going to call the original processors the once we are simulating the simulated processors P of them. And P prime real processors available processors okay each processor will simulate or will work for P over P prime virtual processors what is it have to do so how was the algorithm work for every step it was the algorithm said you read from here to do that local computation you write over here okay.

So now we are going to do that in a loop but nobody saying it is inactive there are P prime processors working each processor is simulating multiple prime over P of the virtual processor of the imagine the assumed number of processors and you so I am one of this processors and I have to simulate P prime over P okay. So P prime of P of them each one had a read local computation write step so I have a array of reads and array of local computation and array of write to do.

What do i have to do all the reads, read them in local all the array I have then I perform the local operation in local memory each one or one after the other and then I have to do all the writes. So I do all the rates I have done one after the other so they would have done in parallel I am doing it in C does not make a difference they are all reads they would have done their local computation in parallel am going to do this does not make a difference they are all independent.

I am taking one area of local memory as thinking of this as a local memory of processor 0 another processor 1 and so on. So they are independent to each other and similarly they were all going to write in parallel am going to write in sequence but after these many steps I have similar related all how many steps? Once approximate one right so if you say 3 steps are one steps together then I have done not one step like this but 3 times the number of processors sub steps which means as many steps as number of processor some simulative.

How do you simulate common write? How all can be done? So if we have a common CRCW PRAM then we will simulate a common CRCW PRAM how would you do that? All these writes if they are writing for the same address would write should write only if they are writing the same thing so what would you have to do you can locally check that first it is in the local memory okay. How much time it takes to check them? As many processors are simulate you can group force and do anything smart.

So that does not change number of steps needed to perform the write if it is random then do not do anything we are just keep writing. If it is priority then you write can you highest priority at the end but there are P prime and independently doing and I have 3 writes to that memory location and some of the processor have only one write to that memory location but is the highest priority one so when does that write?

That guy writes immediately then this after 3 steps this guy overwrite so you have to do somewhat the opposite of what you are saying. Which is perfectly fine so this processor have written its stuff first and this processor is going to write it is so it I going to write one and this is done two to the same location okay. If this is the highest priority and this guy first was the highest you sorted by in the order of highest priority.

Highest priority thing gets written first and this highest priority have written first higher of those will get written this guy now second write from do because it is already done on higher priority. Exactly then you are perfectly fine you write to an address only if you have not written to it before that is ok that you have data to a shared memory and some will not which is not common address not an conflicting address.

A and B want to write to one and B and C want to write to 2 right if they are going to figure out. First of all A and B will decide that we are going to take down highest priority thing we are writing right and we are going to write that first if there is any body is common then they automatically will take care of it the because pre prime processor are will take care of it if not so then I am simply going to write D and C synchronize.

We are assuming that everybody knows how many processors has everybody as in fact in this case we are for simplicity of proof we can start with assumption that everybody has the same number that does not do anything to this example you basically this is the basic rule is that if you are writing to a higher priority so it is not a higher priority memory location which is what I am going to say.

The higher priority processor right so this processor has N thing to write this processor has N thing to write and we have to figure out which order is going to write to them. And you do not know what where this guy is going to write there are N memory location is that you are going to write. N being the number of there are you are simulating T prime by T processors whereas T by P prime processors that many memory location is what you are going to write.

Alright this is one way to do it you basically write the values in whatever order of these memory locations you care for that the issue was you do not know what order was write the memory location you just write them in any order but you have to read it back to find out whether it go written or not whether anybody else wants to write the memory locations are not okay. You are going to do it T prime by T times if somebody else wants to write memory location you need to know what the priority is.

If it is higher priority than you do not want to write it again so now in every memory location you write what you wanted to write as well as the ID of the processor that who is going to write it so each memory location becomes a tool. You write your processor ID whose going to who want to write whoever one out his value is going to be written here right. If you do not win out then you are going to know it and you are not going to write it again.

These steps you will have to be figure out who have to be should be writing it who is higher over here. I understand but it is not only that there is a (()) (44:38) because you are not simulating a priority write on a non-priority write which is you do what you are saying right. The underlining hardware is still priority hardware but only it recognizes priority among only P processors sorry P prime processors.

Whereas you need you to figure out the priority among the P processors the original P some one of them one of the available processors. Not but the other processors is going to write it in one of the other steps that is the problem they were writing it on the same steps there is no problem but when it going to write it one of the other steps it must not overwrite something that someone else is written in that same step at high priority right.

So it must before writing it, must read it and write it again only if among the simulated steps nobody has written to it nobody at higher priority has written to it before so still P prime over P simulation steps are needed are over the P prime but you write to it by reading it first each write operation is done into a read do some compute and write operation another read write. Priority on original P internally you can distinguish right simulating five processors four processors.

Then you know that the simulating processor 8, 9, 10, 11 and 8 has a higher priority than 9, 10 and 11. So that is the local competent but the issue is that I am writing K things P over P over P prime things and some other processor is also writing P prime P over P prime things up to P over P prime things right and in some arbitrary order so what I write here may have a conflict over that fellow writes over there right.

If I have a high priority that fellow over write you can because it is PRAM hardware with P prime processors in only write P prime things concurrently. So you cannot say I am going to write these many things whereas that is another way of simulating because technically in PRAM there is no limit on size of the primary. Each memory cell nobody says is right memory cell.

Nobody cell can be as big as you like enough to hold all the processors ID so now you can simulate by you are writing your part of the data into your slot. So each memory cell as become array and all of them together somehow tell you what the value is in which case everybody gets to write Nth slot but the when you read only the highest priority is going get you are going to only use that others you are going to discard you can read the entire thing in one shot.

Because that is one memory cell okay but there is some technical way of doing this more algorithmic way doing this is simply add the information of who is writing okay alright.

Everything is clear on small number of processors before you write you have read it every time. So K writes are not only K writes and done to K struts of read local computer write okay but it is not just common among them that you are interested in one of your writes may be common with one of the writes on somebody else.

And because you have said am going to write that in this order and this is just some 3 random memory areas memory cells you are writing them in this order. There is some other 3 random memory cells you writing them in some arbitrary order begin even say am going to write the smallest than the biggest when than the bigger than the biggest does not make a different because these are 3 random independent values but one of them night match you do not know whether that guy is third and you are first matches or some other alright.

What if I had not enough memory and you now the same idea will apply instead of M memory cells I have got a M prime so shared memory is limited that local memory is not. If you want more local memory I can give it to you basically you are simulating many processors inside one now you will simulating many memory location else inside one right. How do you do that it basically make a local representation a copy or replica of the shared memory what you wanted right.

You wanted N cells of others you only have M prime so I will take local memory and distribute M locals into their local memory is so everybody is M over P local memory cells dedicated as the shared memory because shared memory has distributed but you still have M prime also available right. There are M Prime M shared memory cells that you wanted to simulate you wanted your algorithm to be able to have but you only got M prime.

Real hardware have only M prime shared memory so I want to write M different things possibly potentially over the period that program runs right. So am going to run those different things is that going to axis as a shared memory and allocate it among the P processors as you keep the replica of P, M over P cells you keep the replica of M over P cells and a last M over P cells is ended to the last part okay.

But now how do you share so you still have a shared memory right so as long as one memory cell is available for every processor one shared memory cell you simply figure out which of the shared memory things you want to write you write that thing in that memory cell and keep the other local to you right. So how do you make sure that you have written you write one of those values one of your allocated values other can see it.

And over the period of this step the original step you have to make sure that everybody gets a chance you do not know who wants to read what. So you are going to take all the cells you are responsible for and put it in a shared memory one at a time okay. So you are basically a loop you have to say now you take my memory location number 0 now you take my memory location 1 take my memory location 2 you have written one at a time.

But it is not a tight loop right you are not just writing because somebody else has to read some other write some other time. You are going to turn it into an entire step everybody writes their 0th location into the shared memory and then everybody gets a chance to read if you wanted to read it then want read it keep quite. Again you are allocating the shared memory among the different processors and because you are responsible for closing your part of the shared memory to everybody else.

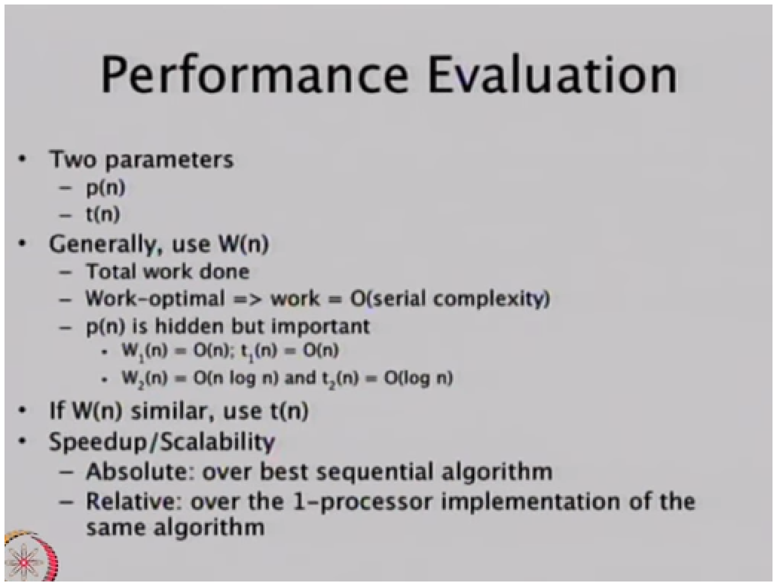
You are going to do that by exposing every single cell you have yes there are so many writes for once for original PRAM steps right. Now you have taken each of them and turn into a full step so you said now you are going to write my location 0 into shared memory location 0 processor one is going to write its location 0 it everybody has an array of memory cells now right it is going to write its 0th cell into the shared memory address one processor two will write itself its 0th cell into shared memory address 2 and in one step everybody has written one value okay.

P processors have written P things but we are saying that we have at least M prime processors thats why the number of processors available in max of PRAM M prime. So as long as there at least M prime processor all the M prime cells have been written by somebody now the next read step anybody you wanted to read that value is going to read it and now all the processors will write the second element of their array into the shared memory.

For writing part this is the just writing has turned into a big loop after that everybody is going to read they are saying do the local computation that is going to happen in anywhere. So you are simulating taking the M cells you have partitioning it into M prime segment blocks whatever you want to call them assign each block to a processor and have the processor essentially use the shared memory it is section of the shared memory and the staging area.

And after as many iteration as a M over M prime because that is how many memory cells each one is responsible for you have exposed all of the memory with us. So anybody who wanted to read anything in the original PRAM as go a chance to read the proper value in one of this RAM's alright.

(Refer Slide Time: 57:56)



Performance Evaluation

- Two parameters
  - p(n)
  - t(n)
- Generally, use W(n)
  - Total work done
  - Work-optimal => work = O(serial complexity)
  - p(n) is hidden but important
    - $W_1(n) = O(n)$; $t_1(n) = O(n)$
    - $W_2(n) = O(n \log n)$ and $t_2(n) = O(\log n)$
- If W(n) similar, use t(n)
- Speedup/Scalability
  - Absolute: over best sequential algorithm
  - Relative: over the 1-processor implementation of the same algorithm

In terms of evaluating performance we looked at a notion of how many processors we are going to use because now the algorithm is just saying because if you give me P processors am going to do this in Time T. So T becomes potentially a function of M the number of things in your input processing and the time taken becomes a function of the P you have and the N which is input size okay.

So instead of having one metric that we use to have how much is the time taken is it log N algorithm or N square algorithm we are now said it is M log N if you have N squared N processor. Now how do you compare two algorithms and that is where we use the notion of work and will just go through few things now to a see why work is what we should be considering but what is saying that is that if I have well you give me two algorithm one takes P1 processors and takes T1 prime.

Either takes P2 processors takes T time I will say hold on tell me how much work is each to it one takes W1 the other takes does work W2 then am going to take the one that does lower or smaller amount of work. And so example if I have got something that takes time N order M but does also work that is order M. I am going to prefer this over another one that is faster takes log in time but the work is un log in generally speaking we will be little deeper into that very shortly.

And if tow algorithms at the same work then we will look at the time you do the same amount of work but who does it in fewer steps okay that fewer steps means what that work is more evenly balance among steps right we also be looking at how scalable it is right now we are not saying it is if I have a value N then goes up by this much but you will say how well does it scale your N goes up.

So there are two notion one is of absolute P dash which says if I had the best sequential algorithm it took time T0. Your parallel algorithm takes time T1 to the speed up is T0 over T1 near that it is more relative. That is why the term is relative to speed up it says that if I have one processor you run your algorithm not the sequential algorithm your PRAM algorithm if it had only one processor available how much time would it have taken.

If that is P0 and on N size input it is PN then the speed up is relative speedup is P0 over here okay so usually relative speed up is going to be important not that absolute speedup not important but if you got a good relative speedup it is saying that you got an algorithm that scale well as hardware size will change it is going to do well okay we do not reduce the input size we just say that it is same algorithm we just the number of because algorithm now has to values how many processors it takes and how many what the input size.

If I keep the input size take the processor all the way to one and then compare that speed if I take the processor to some other higher value okay let us stop here.