Models In Parallel Computation

It is difficult to write programs without a good idea of how the target computer will execute the code. The most important information is knowing how expensive the operations are in terms of time, space, and communication costs

Last Week

- Matrix Multiplication was used to illustrate different parallel solutions
  - Maximum parallelism, O(log n) time, $O(n^3)$ processors, PRAM model
  - Basic (strips x panels), O(n) time, $O(n^2)$ processors
  - Pipelined (systolic), O(n) time, $O(n^2)$ processors, VLSI model
  - SUMMA algorithm, used many techniques, O(n) time, $O(n^2)$ processors, scalable, Distributed Memory Model

Last Week (continued)

- Different techniques illustrated --
  - Decompose into independent tasks
  - Pipelining
  - Overlapping computation and communication
- Optimizations
  - Enlarge task size, e.g. several rows/columns at once
  - Improve caching by blocking
  - Reorder computation to “use data once”
  - Exploit broadcast communication

The SUMMA algorithm used all of these ideas

Plan for Today

- Importance of von Neumann model & C programming language
- Recall PRAM model
- Valiant’s Maximum Algorithm
  - Analyze result to evaluate model
- Introduce CTA model
  - Analyze result to evaluate model
- Alternative Models
  - LogP is too specific
  - Functional is too vague

Successful Programming

When we write programs in C they are ...

- Efficient -- programs run fast, especially if we use performance as a goal
  - traverse arrays in row major order to improve caching
- Economical -- use resources well
  - represent data by packing memory
- Portable -- run well on any computer with C compiler
  - all computers are universal, but with C fast programs are fast everywhere
- Easy to write -- we know many 'good' techniques
  - reference data, don't copy

These qualities all derive from von Neumman model

Von Neumann (RAM) Model

- Call the 'standard' model of a computer (RAM) the von Neumann model
  - A processor interpreting 3-address instructions
  - PC pointing to the next instruction of program in memory
  - Flat, randomly accessed memory requires 1 time unit
  - Memory is composed of fixed-size addressable units
  - One instruction executes at a time, and is completed before the next instruction executes
- The model is not literally true, e.g., memory is hierarchical but made to "look flat"

C directly implements this model in a HLL

vN Model Contributes To Success

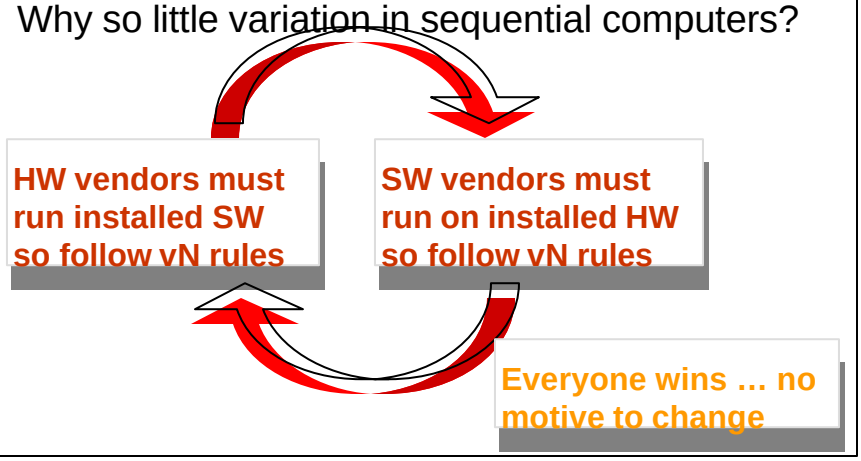
- The cost of C statements on the vN machine is "understood" by C programmers …
- How much time does **A[r][s] += B[r][s];** require?
  - Load row_size_A, r, s, A_base, B_base (5)
  - temp = (row_size_A * r + s) * data_size (3)
  - A_base + temp; B_base + temp; load both values (4)
  - Add values and return to memory (2)
- Same for many operations, any data size
- Result is measured in "instructions" not time
- Efficient and Economical and Easy to Write

Widely known and effectively used

Portability

- Most important property of the C-vN coupling:

  It is approximately right everywhere
- Why so little variation in sequential computers?



Von Neumann Summary

- The von Neumann model "explains" the costs of C because C expresses the facilities of the von Neumann machines in a set of useful programming facilities
- Knowing the relationship between C and the von Neumann machine is essential for writing efficient programs
- Following the rules produces good results everywhere because everyone benefits

What is the parallel version of vN?

PRAM Often Proposed As A Candidate

PRAM (Parallel RAM) ignores memory organization, collisions, latency, conflicts, etc.

Ignoring these are *claimed* to be benefits ...

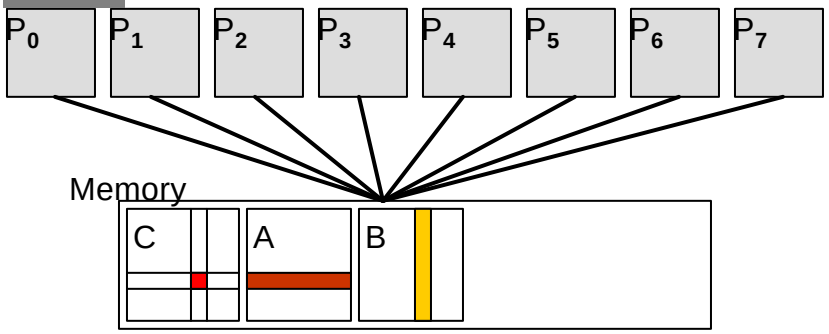
- Portable everywhere since it is very general
- It is a simple programming model ignoring only insignificant details -- off by "only log P"
- Ignoring memory difficulties is OK because hardware can "fake" a shared memory
- Start with PRAM then refine the program to a practical solution -- good for getting started

Recall Parallel Random-Access Machine

PRAM has any number of processors

- Every processor references any memory in “time 1”
- Memory read and write collisions must be resolved

PRAM



Variations on PRAM

Resolving the memory conflicts considers read and write conflicts separately

- Exclusive read/exclusive write (EREW)
  - The most limited model
- Concurrent read/exclusive write (CREW)
  - Multiple readers are OK
- Concurrent read/concurrent write (CRCW)
  - Various write-conflict resolutions used
- There are at least dozen other variations

All theoretical -- not used in practice

Find Maximum with PRAM (Valiant)

Task: Find largest of n integers w/ n processors

Model: CRCW PRAM (writes OK if same value)

L.G.Valiant, “Parallelism in comparison problems,” SIAM J. Computing 4(3):348-355, 1975

L.G. Valiant, “A Bridging Model for Parallel Computation,” CACM 33(8):103-111, 1990

R.J. Anderson & L. Snyder, “A Comparison of Shared and Nonshared Memory Models for Parallel Computation,” *Proc. IEEE* 79(4):480-487

Algorithm Sketch

Algorithm: T rounds of O(1) time each

In round, process groups of m vals, $v_1, v_2, \ldots, v_m$

- Fill m memory locations $x_1, x_2, \ldots, x_m$ with 1
- For each $1 \leq i,j \leq m$ a processor tests ...

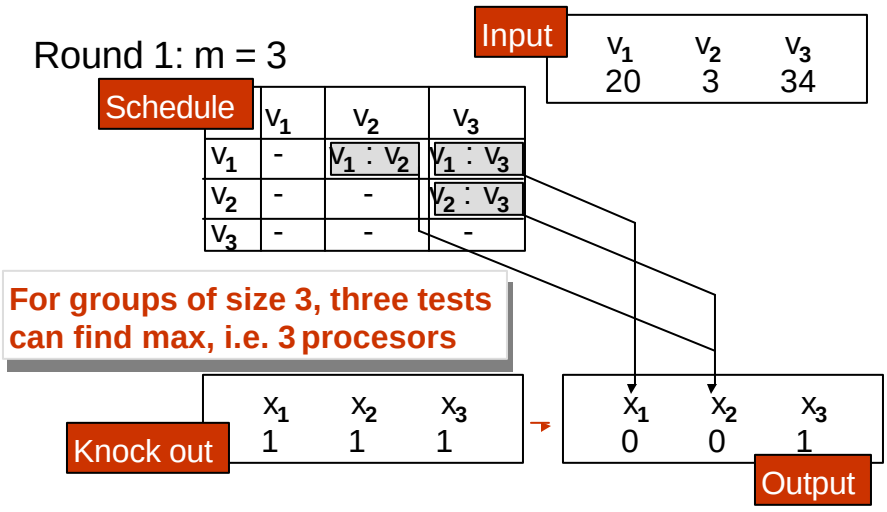
```
if v_i < v_j then x_i = 0 else x_j = 0
```

- If $x_k = 1$ it’s max of group; pass x_k to next round

The ‘trick’ is to pick m right to minimize T

Finding Max (continued)

Round 1: m = 3



Solving Whole Problem

- Round 1 uses P processors to find the max in groups of m=3 … producing P/3 group maxes
- Round 2 uses P processors to find the max in groups of m=7 … producing P/21 group maxes
- Generally to find the max of a group requires m(m-1)/2 comparisons
- Picking m when there are P processors, r maxes … largest m s.t. (r/m)(m(m-1)/2) £ P i.e. r(m-1) £ 2P

Finding Max (continued)

- Initially, r = P, so
  r(m-1) £ 2P
  implies m = 3, producing r = P/3
- For (P/3)(m-1) £ 2P implies next group = 7
- Etc.
- Group size increases quadratically implying the maximum is found in O(loglog n) steps on CRCW PRAM

It's very clever, but is it of any practical use?

Assessing Valiant's Max Algorithm

The PRAM model caused us to ...

- Exploit the "free use" of read and write collisions, which are not possible in practice
- Ignore the costs of data motion, so we adopt an algorithm that runs *faster* than the time required to bring all data values together, which is Ω(log n)
- So what?

Running Valiant's Algorithm

- PRAM's don't exist and can't be built
- To run the algorithm we need a simulator for the CRCWPRAM
- In order to simulate the concurrent reads and the concurrent writes, a parallel computer will need $\Omega(\log P)$ time per step, though there are bandwidth requirements and serious engineering problems to attain that goal [details in future lecture]

- Observed performance of Valiant's Max:

  $O(\log n \log\log n)$

Alternative Solution

- What is the best (practical) way of computing max?
  - The tournament algorithm, a variation on global sum
  - O(log P) time on P processors (as with sum, the problem size can be Plog P)
  - The tournament algorithm doesn't need to be simulated … it runs in the stated time directly on all existing parallel processors
- Since O(log n) < O(log n loglog n) the PRAM model mispredicted the best practical algorithm

The PRAM didn't help, it hurt our effort

Is The PRAM A Good Abstraction?

Different Opinions ...

- OK for finding theoretical limits to parallelism ↑
- It is a simple programming model ignoring only insignificant details -- off only by log P ↓
- Ignoring memory difficulties is OK because hardware can "fake" a shared memory ↓
- Start with PRAM then evolve to more realistic solution -- good for getting started ↓

Break

- Question to think about …

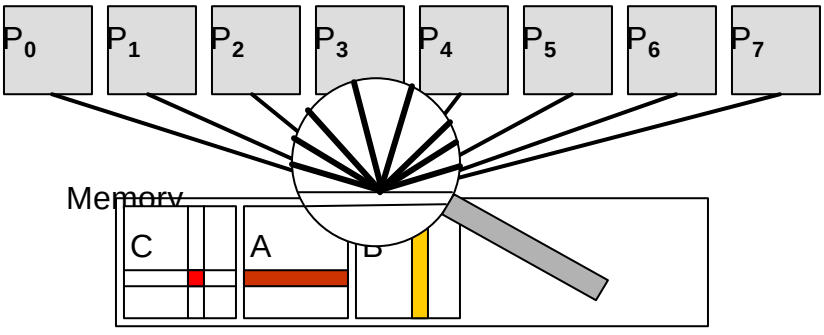
  "Should a model of computation be 'close to the machine' so that the programs are efficient, or 'far from the machine' so the programs are machine independent?"

Requirements Of A Practical Model

- Like von Neumann …
  - Capture key features of HW implementations
  - Reflect costs realistically
  - Portable across physical machines
  - Be as simple as possible relative to the above
- Problems for parallel programmers …
  - Parallel computers have widely different architectures -- steady-state cost not established
  - Existing SW for Cray vector machines, poor guide
  - Sequential computers keep getting faster

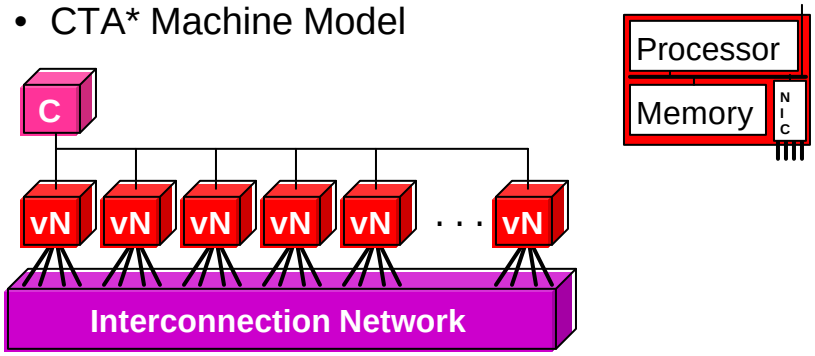
Model Must Be “Realizable”

- There should be no “magic”
- Explain how the processors connect to the memory



Proposed Practical Parallel Model

- CTA* Machine Model



Benefit from everything the vN model does right

CTA = Candidate Type Architecture

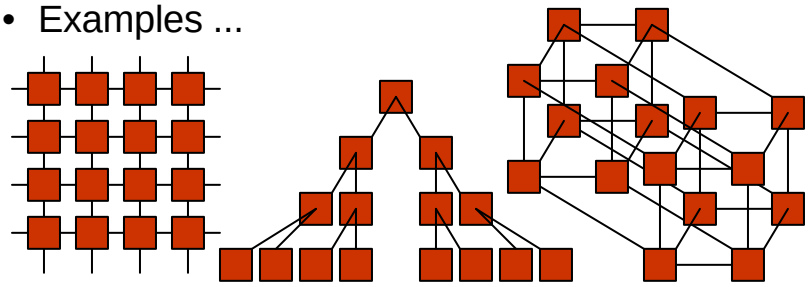
Properties of the CTA ...

- Processor Elements are von Neumann machines -- they could be SMPs or other parallel designs, but they execute vN ISA
- PEs are connected by a sparse (O(n) edges) point-to-point network, not specified
- PE's connection to network is "narrow"
- Latency l >> 1 for non-local memory reference
- Controller has "thin" broadcast connection to all processors

The distributed memory model of last week

Network Not Specified

- Historically, programmers have “programmed to the network” harming their computations
- Examples ...



Network is like the system bus -- forget it exists

Using CTA What Is Best Max Algorithm?

CTA charges for operations, memory reference, communication … PRAM solution of little use

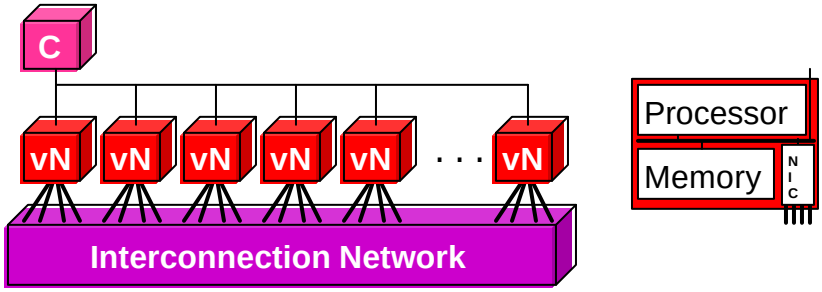
- Tournament algorithm finds maximum of n elements in O(log n) time like the global sum
  - Odd PEs send value to next lower processor
  - Even PEs recv, compare and send larger value to “parent”
  - IDivide PE index by 2, and repeat until one processor left
  - Time is O(log n) communication + parallel compare steps

This is a practical algorithm that can be tested

CTA Emphasizes Locality

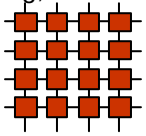
CTA models … where the data is & where it goes

- The l parameter captures the benefit of exploiting locally stored values
  - l says moving data to other PEs has significant cost



The λ value

- λ is a purposely vague quantity
  - The cost is not known and cannot be known because every machine is different
    - There is no agreement yet on *the* parallel architecture
    - λ depends on bandwidth, engineering, interfaces, mem
  - The cost cannot be a single number because of network topology, conflicts, etc.



  - Optimizing for communication -- if possible -- should be done by the compiler not a programmer

CTA Abstracts Existing Architectures

- The MIMD parallel machines -- all machines since mid 1980s -- are modeled by CTA
- A few minor adjustments may be needed
  - Some machines do not have a controller (front end machine), but one PE can be both a PE and Controller
  - Some machines such as bus-architectures fail to meet the point-to-point network feature, but a decent bus is an OK approximation (and there is no other model!)
- Since the CTA contains the von Neumann architecture C is good for programming PEs but more global abstractions are needed

Assessing the CTA

The considerations were

- Capture key features of HW implementations
- Reflect costs realistically
- Portable across physical machines

  The above have been established experimentally
- Be as simple as possible relative to the above

  Difficult to measure, but programmers can use it

We will use the CTA in this class

Improving CTA may be possible, but not in some ways

Alternatives ...

Many models have been proposed, but they have had weaknesses

- LogP [UC Berkeley] is CTA+parameters
- The model measures latency and other variables related to communication -- too many free variables
  - The model was effective at predicting behavior on the Thinking Machines CM-5, which motivated it
  - With many parameters, it was difficult to decide which were significant, especially when their values are free
  - It remains a model of theoretical interest

LogP's parameters try being exact about 1. Not possible

Alternatives (continued)

- Functional programming has motivated many models
  - *Concurrency is natural part of parameter evaluation*
  - Processors not specifically identified -- they simply work on next task
  - Write-once memory avoids races and indeterminacy
  - No notion of locality since it is not possible to associate memory with any logical processor
  - Functional models presume shared memory computers and have considerable overhead; functional has not become popular

Summary

- The von Neumann model allows us to be successful C programmers
- A parallel alternative is needed
  - The PRAM is not a good alternative because it leads us to impractical algorithms like the O(loglog n) max
  - The CTA does work because it charges for data motion and communication using the parameter l
  - Capturing locality is a central benefit of the CTA
  - CTA abstracts all contemporary MIMD machines

Finding a model is a balancing act in which the “right” things are modeled and everything else is ignored