Models of Parallel Computation

Mark Greenstreet

CpSc 418 – Oct. 10, 2013

- The RAM Model of Sequential Computation
- Models of Parallel Computation
  - PRAM
  - CTA
  - LogP
The Big Picture

Parallelandia

start

LYSE

paradigms

algorithms

performance

software design

architecture

We are here

finish
Objectives

- Learn about models of computation
  - Sequential: Random Access Machine (RAM)
  - Parallel
    - Parallel Random Access Machine (PRAM)
    - Candidate Type Architecture (CTA)
    - Latency-Overhead-Bandwidth-Processors (LogP)

- See how they apply to some examples
  - find the maximum
  - reduce
  - FFT
The RAM Model

RAM = Random Access Machine

- Axioms of the model
  - Machines work on words of a “reasonable” size.
  - A machine can perform a “reasonable” operation on a word as a single step.
    - such operations include addition, subtraction, multiplication, division, comparisons, bitwise logical operations, bitwise shifts and rotates.
  - The machine has an unbounded amount of memory.
    - A memory address is a “word” as described above.
    - Reading or writing a word of memory can be done in a single step.
The Relevance of the RAM Model

- If a single step of a RAM corresponds (to within a factor close to 1) to a single step of a real machine.
- Then algorithms that are efficient on a RAM will also be efficient on a real machine.
- Historically, this assumption has held up pretty well.
  - For example, mergesort and quicksort are better than bubblesort on a RAM and on real machines, and the RAM model predicts the advantage quite accurately.
  - Likewise, for many other algorithms
    - ★ graph algorithms, matrix computations, dynamic programming, . . . .
    - ★ hard on a RAM generally means hard on a real machine as well: NP complete problems, undecidable problems, . . . .
The Irrelevance of the RAM Model

The RAM model is based on assumptions that don’t correspond to physical reality:

- Memory access time is highly non-uniform.
  - Architects make heroic efforts to preserve the illusion of uniform access time fast memory –
    - caches, out-of-order execution, prefetching, . . .
  - but the illusion is getting harder and harder to maintain.
    - Algorithms that randomly access large data sets run much slower than more localized algorithms.
    - Growing memory size and processor speeds means that more and more algorithms have performance that is sensitive to the memory hierarchy.

- The RAM model does not account for energy:
  - Energy is the critical factor in determining the performance of a computation.
  - The energy to perform an operation drops rapidly with the amount of time allowed to perform the operation.
The PRAM Model

PRAM = Parallel Random Access Machine

- Axioms of the model
  - A computer is composed of multiple processors and a shared memory.
  - The processors are like those from the RAM model.
    - The processors operate in lockstep.
    - I.e. for each $k > 0$, all processors perform their $k^{th}$ step at the same time.
  - The memory allows each processor to perform a read or write in a single step.
    - Multiple reads and writes can be performed in the same cycle.
    - If each processor accesses a different word, the model is simple.
    - If two or more processors try to access the same word on the same step, then we get a bunch of possible models:
      - **EREW**: Exclusive-Read, Exclusive-Write
      - **CREW**: Concurrent-Read, Exclusive-Write
      - **CRCW**: Concurrent-Read, Concurrent-Write
EREW, CREW, and CRCW

**EREW**: Exclusive-Read, Exclusive-Write
- If two processors access the same location on the same step,
  - then the machine fails.

**CREW**: Concurrent-Read, Exclusive-Write
- Multiple machines can read the same location at the same time, and they all get the same value.
- At most one machine can try to write a particular location on any given step.
- If one processor writes to a memory location and another tries to read or write that location on the same step,
  - then the machine fails.

**CRCW**: Concurrent-Read, Concurrent-Write
If two or more machines try to write the same memory word at the same time, then if they are all writing the same value, that value will be written. Otherwise (depending on the model),
- the machine fails, or
- one of the writes “wins”, or
- an arbitrary value is written to that address.
Finding the maximum element of an array of $N$ elements.

- The obvious approach
  - Do a reduce.
  - Use $N/2$ processors to compute the result in $\Theta(\log_2 N)$ time.
A Valiant Solution

L. Valiant, 1975

- Use \( P \) processors.

Step 1:
- Divide the \( N \) elements into \( N/3 \) sets of size 3.
- Assign 3 processors to each set, and perform all three pairwise comparisons in parallel.
- Mark all the “losers” (requires a CRCW PRAM) and move the max of each set of three to a fixed location.

Step 2:
- We now have \( N/3 \) elements left and still have \( N \) processors.
- We can make groups of 7 elements, and have 21 processors per group, which is enough to perform all \( \binom{7}{2} = 21 \) pairwise comparisons in a single step.
- Thus, in \( O(1) \) time we move the max of each set to a fixed location. We now have \( N/21 \) elements left to consider.
Visualizing Valiant

$max(x(0)\ldots x(20))$

max from group of 7
(21 parallel comparisons)

group of 7 values

max from each group
(3 parallel comparisons/group)

groups of 3 values

N values, N processors
A Valiant Solution

Subsequent steps:

- On step $k$, we have $N/m_k$ elements left.
- We can make groups of $2m_k + 1$ elements, and have $m_k(2m_k + 1) = \binom{2m_k + 1}{2}$ processors per group, which is enough to perform all pairwise comparisons in a single step.
- We now have $N/(m_k(2m_k + 1))$ elements to consider.

Run-time:

- The sparsity is squared at each step.
- It follows that the algorithm requires $O(\log \log N)$.
- Valiant showed a matching lower bound and extended the results to show merging is $\theta(\log \log N)$ and sorting is $\theta(\log N)$ on a CRCW PRAM.
Valiant Details

<table>
<thead>
<tr>
<th>round</th>
<th>values remaining</th>
<th>group size</th>
<th>processors per group</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>N</td>
<td>2 * 1 + 1 = 3</td>
<td>3 = 3 choose 2</td>
</tr>
<tr>
<td>2</td>
<td>( \frac{N}{3} )</td>
<td>2 * 3 + 1 = 7</td>
<td>3 * 7 = 21 = 7 choose 2</td>
</tr>
<tr>
<td>3</td>
<td>( \frac{1 \times N}{3} = \frac{N}{21} )</td>
<td>2 * 21 + 1 = 43</td>
<td>21 * 43 = 903 = 43 choose 2</td>
</tr>
<tr>
<td>4</td>
<td>( \frac{1 \times N}{43} = \frac{N}{21} )</td>
<td>2 * 903 + 1 = 1,807</td>
<td>903 * 1,807 = 1,631,721 = 1807 choose 2</td>
</tr>
<tr>
<td>( k )</td>
<td>( \frac{N}{m_k} )</td>
<td>2m_k + 1</td>
<td>( m_k(2m_k + 1) = (2m_k + 1) ) choose 2</td>
</tr>
<tr>
<td>( k + 1 )</td>
<td>( \frac{1}{2m_{k+1} + 1} )</td>
<td>2m_{k+1} + 1</td>
<td>( m_{k+1}(2m_{k+1} + 1) = (2m_{k+1} + 1) ) choose 2</td>
</tr>
</tbody>
</table>

- \( m_k \) is the “sparsity” at round \( k \):
  - \( m_1 = 1 \)
  - \( m_{k+1} = m_k(2m_k + 1) \)

- Now note that \( m_{k+1} = m_k(2m_k + 1) > 2m_k^2 > m_k^2 \).
- Thus, \( \log(m_{k+1}) > 2 \log(m_k) \).
- For \( k \geq 2 \), \( m_k > 2^{2^{k-1}} \).
- Therefore, if \( N \geq 2 \), \( k > \log \log(N) + 1 \Rightarrow m_k > N \).
The Irrelevance of the PRAM Model

The PRAM model is based on assumptions that don’t correspond to physical reality:

- Connecting $N$ processors with memory requires a switching network.
  - Logic gates have bounded fan-in and fan-out.
  - ⇒ and switch fabric with $N$ inputs (and/or $N$ outputs) must have depth of at least $\log N$.
  - This gives a lower bound on memory access time of $\Omega(\log N)$.

- Processors exist in physical space
  - $N$ processors take up $\Omega(N)$ volume.
  - The processor has a diameter of $\Omega(N^{1/3})$.
  - Signals travel at a speed of at most $c$ (the speed of light).
  - This gives a lower bound on memory access time of $\Omega(N^{1/3})$.

- Valiant acknowledged that he was neglecting these issues in his original paper.
  - but that didn’t deter lots of results being published for the PRAM model.
The CTA Model

CTA = Candidate Type Architecture

- Axioms of the model
  - A computer is composed of multiple processors.
  - Each processor has
    - Local memory that can be accessed in a single processor step (like the RAM model).
    - A small number of connections to a communications network.
  - A communication mechanism:
    - Conveying a value between processors takes \( \lambda \) time steps.
    - \( \lambda \) can range from \( 10^2 \) to \( 10^5 \) or more depending on the architecture.
    - The exact communication mechanism is not specified.
Communication Mechanisms

- **Shared Memory:** $\lambda \approx 100 - 1000$.
- **One-sided communication:**
  - Used on some supercomputers (e.g. Cray).
  - `$\text{put}(\text{addr}, \text{data})$`: copies data into the memory of a remote node.
  - `$\text{read}(\text{addr})$`: reads data from the memory of a remote node.
  - Called “one-sided” because the remote-node doesn’t do anything to receive or transmit the data involved.
- **Message passing:** $\lambda \approx 5000 - 10000^+$. 
Latency vs. Throughput

Definitions:
- Latency is the amount of time it takes to perform an operation from start to finish.
- Throughput is the number of operations that can be performed per unit time.

Relations:
- If we did everything sequentially, we would have
  \[ \text{Throughput} = \frac{1}{\text{Latency}} \]
- But, with pipelined and/or parallel execution, we can have
  \[ \text{Throughput} \gg \frac{1}{\text{Latency}} \]
Latency vs. Throughput

Why does it matter:
- Throughput (a.k.a. peak performance) is usually a lousy measurement of real performance: real programs have some latency critical operations.
- Latency does not completely capture the performance of a parallel architecture
  - If it takes $\lambda$ time units to send one word between two processors,
  - We can probably send two words in $< 2\lambda$ time units.
  - On the other hand, can we send a million words in $\approx \lambda$ time units?
  - Bandwidth matters.
The LogP Model

- **Motivation (1993): convergence of parallel architectures**
  - Individual nodes have microprocessors and memory of a workstation or PC.
  - A large parallel machine had at most 2000 such nodes.
  - Point-to-point interconnect –
    - Network bandwidth much lower than memory bandwidth.
    - Network latency much higher than memory latency.
    - Relatively small network diameter: 5 to 20 “hops” for a 1000 node machine.

- **The model parameters:**
  - $L$: the latency of the communication network fabric
  - $o$: the overhead of a communication action
  - $g$: the bandwidth of the communication network
  - $P$: the number of processors
LogP breaks communication into more detailed phases than CTA.

If $g$ is enough smaller than $L$, then LogP shows that the simple binary tree isn’t exactly optimal for broadcast.

Example: $L = 7$, $o = 6$, $g = 8$, $P = 8$ (thus $q = 2$):

- Simple binary tree completes broadcast in $\text{Time} = 3L + 6o = 57$.
  The extension of the logP solution in light-blue represents such a path.
- Optimized tree completes in 28 time units
  - $p0$ sends to $p5$, $p3$, $p1$, and $p1$
  - $\text{Time} = 5o + 3q + L = 43$.
- Is it worth it?
Broadcast: notes

- The optimized schedule can be derived by starting with the root.
- Determine when each processor is eligible to send a message:
  - If the processor just sent a message, it must wait \( \max(o, g) = o + q \) time units before it can send another message.
  - If the processor just received a message, it has the send-overhead time of \( o \) before it can send a message.

- Notes on the figure:
  - An “overhead” edge marked with a red \( o \) denotes the overhead for sending a message.
  - An “overhead” edge marked with a green \( o \) denotes the overhead for receiving.
  - The faint edges and vertices are not part of the optimized broadcast – they indicate the time that a broadcast would take if with the balanced tree schedule on the left.

- Big picture: the logP approach recognizes that the root can finish sending three messages before a processor that is two “latency” edges away is ready to send.
  - Thus, the time until the last message is received is reduced if the root sends one more message, and another node sends one less message.
The Fourier transform converts between time and frequency representations.

- Brute-force implementation: $O(N^2)$ operations.
- FFT: $O(N \log N)$ operations.
- The Fast-Fourier transform is used in many signal processing applications:
  - audio signals
  - wi-fi modulation and demodulation
  - image filtering
  - voice recognition
  - ...
LogP Example: FFT (2/8)

\[ y_1 = \cos\left(\frac{2\pi t}{64}\right) \]

\[ y_2 = \cos\left(\frac{2\pi t}{43}\right) \]

\[ y_1 + y_2 \]

Full-disclosure: spectra computed using a Hamming window.
The data flow of the FFT has the “butterfly” structure shown on the left.
LogP Example: FFT (4/8)

- processor 0
- processor 1
- processor 2
- processor 3

- First attempt to parallelize:
  - assign blocks of rows to processors.
  - lots of communication at the left
  - everything local at the right.
LogP Example: FFT (5/8)

Second attempt to parallelize:
- Interleave rows among processors
- Everything local on the left
- Lots of communication on the right.
LogP Example: FFT (6/8)

- processor 0
- processor 1
- processor 2
- processor 3

Combined approach
- Interleave rows on the left
- One big round of communication in the middle
- Block of rows on the right
Another view of the combined approach

- the FFT and transpose phases drawn separately.
LogP shows that the combined approach is better.
  ▶ So does CTA – one round of messages is clearly better than log $P$ rounds.
  ▶ The technique is well-known – the same approach is important to get good cache utilization.

LogP shows that staggering messages is better than naively flooding one destination at a time.
  ▶ So does CTA with its assumption of bounded fan-in and fan-out of the network.

Note: The “transpose in the middle” pattern of the FFT occurs in many other algorithms as well.
  ▶ It’s important to be able to handle this pattern efficiently.
Comparing the models

- CTA is simpler than LogP
- LogP accounts for more machine details
  - but these details don’t seem essential for the examples that they give in the paper.
  - It’s not clear that the extra details would account for more than a factor of 2 in time estimates,
  - and there are lots of other system details that LogP ignores that can cause errors of that magnitude or larger.
  - but the marketing is better: “LogP” just sounds better than CTA. 😊
- Both are based on a 10-20 year old machine model
  - That’s ok, the papers are 18-25 years old.
  - Doesn’t account for the heterogeniety of today’s parallel computers:
    - multi-core on chip, faster communication between processors on the same board than across boards, etc.
- CTA seems like a simple, and reasonable place to start.
  - But recognize the limitations of any of these models.
- Getting a model of parallel computation that’s as all-purpose as the RAM is still a work-in-progress.
For further reading


October 10: Models of Parallel Computation
Reading: Lin & Snyder, chapter 2, pp. 43–59.
Homework: Homework 2 due.

October 15: Peril-L
Reading: Lin & Snyder, chapter 4, pp. 87–100.

October 17: Scan
Reading: Lin & Snyder, chapter 5, pp. 112–125.

October 22: Midterm

October 24: PReach: a parallel model checker, and an example of a large-scale Erlang application

October 29: Work allocation
Reading: Lin & Snyder, chapter 5, pp. 125–142.

October 31: POSIX threads
Reading: Lin & Snyder, chapter 6, pp. 143–187.

Nov. 5: Sorting
• Compare and Contrast the main features of the PRAM, CTA, and LogP models?
• How does each model represent computation?
• How does each model represent communication?
• How does one determine parameter values for the CTA and LogP models? Describe at a high-level the kinds of experiments you could run to estimate the parameters.