Models of Parallel Computation

Mark Greenstreet

CpSc 418 – October 10, 2018

- The RAM Model of Sequential Computation
- Models of Parallel Computation
- An entertaining proof

Objectives

- Learn about models of computation
  - Sequential: Random Access Machine (RAM)
  - Parallel
    - ★ Parallel Random Access Machine (PRAM)
    - ★ Work Span
    - ★ Candidate Type Architecture (CTA)
    - ★ Latency-Overhead-Bandwidth-Processors (LogP)
- An entertaining algorithm and its analysis
  - If a model has invalid assumptions,
  - then we can show that algorithm 1 is faster than algorithm 2,
  - but in real life algorithm 2 is faster.
  - Valiant's algorithm also provides some mathematical entertainment.

The RAM Model

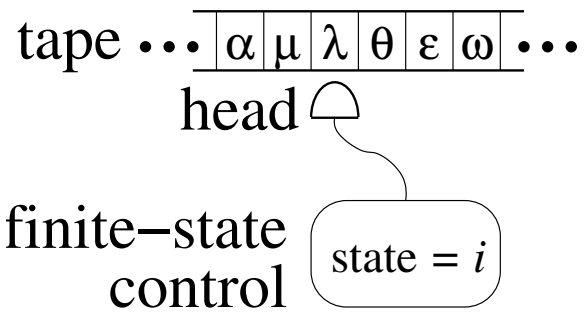
RAM = Random Access Machine

- 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, ...

RAM vs. Turing Machine



At each step:
  read symbol at current head position
  based on symbol and current state:
    write new symbol on tape
    update state
    move head one square to left or right.

- Turing machines are **universal**: anything that can be computed with a RAM can be computed with a Turing machine.
- Polynomial Time is the same class for Turing Machine and RAM.
  - Likewise for Exponential Time, and many other classes.
- But, classes $O(N)$, $O(N \log N)$ are different for Turing Machines and RAMs.
  - That's why we use RAMs for practical programming – the RAM is much closer to a real computer.

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

      ★ See slide 31 for more details.

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.
  - $\Rightarrow$ any 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 $\Omega(N^{1/3})$.

The Work-Span Model

- Represent computation as a DAG (directed acyclic graph)
  - The graph represents the dependencies in the computation.
  - Vertices correspond to operations.
  - If the result of operation `op1` is needed by operation `op2`, there is a directed edge from `op1` to `op2`.
- Work = the number of vertices in the graph.
  - Work corresponds rougly to sequential run-time.
- Span = height of the graph (i.e. the longest directed path).
  - Span corresponds to the ideal parallel time.
  - I.e. the time with an infinite number of processors.
  - Communication cost is ignored.

Bounding Speed-up Using Work-Span

- Let $T_1 = \text{Work}$. T_1 is the sequential time.
- Let $T_\infty = \text{Span}$. T_∞ is the ideal parallel time with an unbounded number of processors.
- Let T_P denote the time with P processors.

$$T_\infty \quad \leq \quad T_P \quad \leq \quad \frac{1}{P}(T\_1 - T\_infty) + T\_infty$$

- See Oct. 5 lecture, slides 25–27, for more details.

Limitations of Work-Span: Absurd Parallelism

```
% Compute C = A*B. A, B, and C are N × N matrices.
for(int i = 0; i < N; i++) { % for each row of A
  for(int j = 0; j < N; j++) { % for each column of B
    sum = 0.0;
    for(int k = 0; k < N; k++) { % dot-product
      sum += A[i][k] * B[k][j];
    C[i][j] = sum; % store result in C
  }
}
```

- What is the work for this algorithm?
- What is the span?

Limitations of Work-Span – communication costs

- Work-Span ignores communication cost. ☹
- We could "color" vertices to do a processor assignment, and charge for edges between vertices of different colors, but . . .
  - This makes the graph specific to the number of processors.
  - Doesn't reflect real communication costs
    - ★ A processor can engage in a bound number of communication actions at a time.
    - ★ Thus, a communication edge may need to wait for other communication actions even though there is no computational dependency.
    - ★ We can aggregate messages. If processor P_1 has many messages to send to P_2, it may be better to combine them and only pay the λ overhead once.
    - ★ Need to recognize the big messages take longer than small messages.

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.
  - There is a communication network connecting the processors.
    - The general model:
      - The communication network is a graph where all vertices (processors and switches) have bounded degree.
      - Each edge has an associated bandwidth and latency.
    - The simplified model:
      - Global actions have a cost of λ times the cost of local actions.
      - λ is assumed to be "large".
    - The exact communication mechanism is not specified.

The (Ir)Relevance of the CTA Model

- Recognizing that communication is expensive is the one, most important point to grasp to understand parallel performance.
  - CTA highlights the central role of communication.
  - PRAM ignores it.
- The general model is parameterized by the communication network
  - Can we apply results from analysing a machine with a 3-D toroidal mesh to a machine with fat trees?
  - PRAM ignores it.
- The simple model neglects bandwidth issues
  - Messages are assumed to be “small”.
  - But, bigger messages often lead to better performance.
  - If we talk about bandwidth, do we mean the bandwidth of each link?
  - Or, do we mean the bisection bandwidth?
- Lousy acronym.

A more realistic, but simple version of CTA

- Sending W words cost $\lambda + t_w W$.
  - t_w adds a cost for "large" messages.
- For real-world machines, t_w is "comparable" to the cost for an ALU operation:
  - On-chip cache-bandwidth: 4-16 words per clock cycles, t_w is less than 1, but not way less.
  - Network bandwidth: a few to a few dozen clock cycles per word sent – not that much worse than a branch mispredict.
  - We'll usually just pretend that $t_w = 1$ to keep our formulas simple.
- Who pays the communication cost? – Keep it simple.
  - The sender and receive each incur a cost of $\lambda + t_w$.
  - The time from starting to send a message until completing the receive is $\lambda + t_w$.
  - This could be ignoring a small factor, but we won't stress about it.
- Big picture: have a simple but realistic model for big-$\mathcal{O}$ analysis:
  - Use simple analysis to choose promising implementations.
  - Get the constant factors and additive constants by measuring executing code on real machines.

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

Why does **g** stand for “bandwidth”?

Marketing!

- What if we used **b** for “bandwidth”?
- Need a catchy acronym with ‘ℓ’, ‘o’, ‘b’, and ’p’ . . .
  - got it: **BLOP**
  - but the marketing department vetoed it.

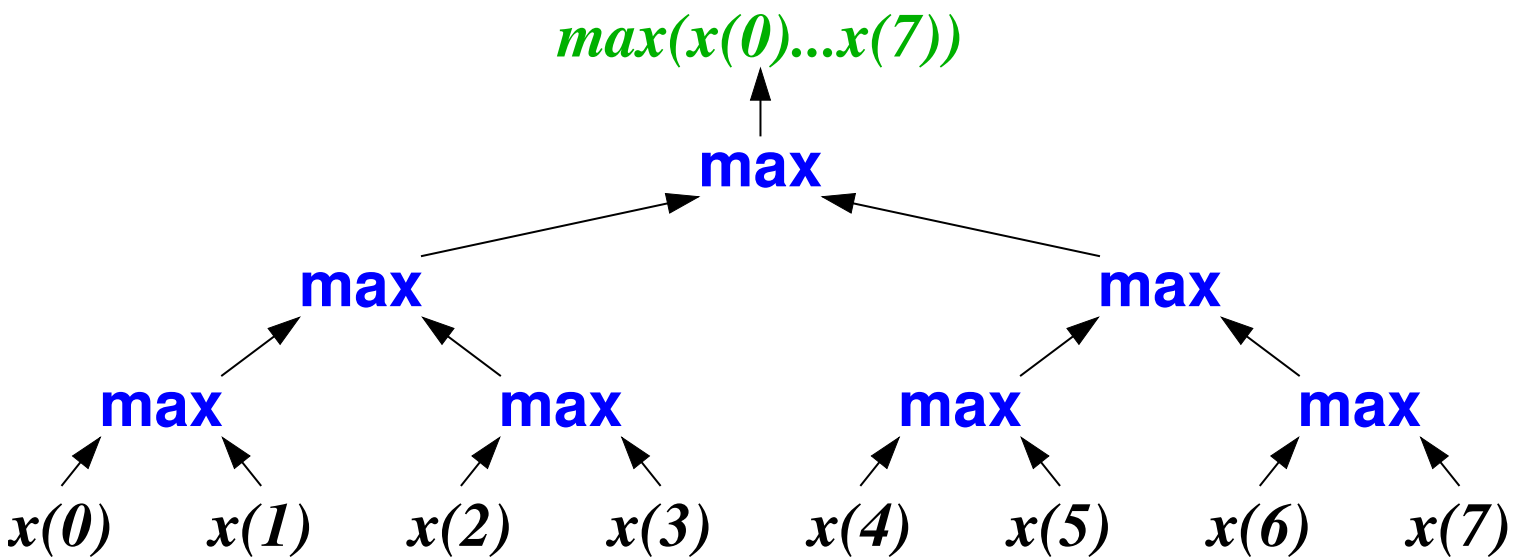
logP in practice

- The authors got some surprisingly good performance prediction for a few machines and a few algorithms by finding the "right" values for ℓ, o, g, and P for each architecture.
- It's rare to get a model that comes to within 10-20% on several examples. So, this looked very promising.
- Since then, logP seems to be a model with more parameters than simplified CTA, but not particularly better accuracy.
- Good to know about, because if you meet an algorithms expert, they'll probably know that PRAM is unrealistic.
  - Then, you'll often hear "What about logP"? – the paper has lots of citations.
  - In practice, it's a slightly fancier was of saying "communication costs matter".

Fun with the PRAM Model

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 N processors.
- The big picture:
  - Initially, we can use clumps of three processors to find the largest of three elements in $O(1)$ time – just do all three comparisons.
  - Now, we have $N/3$ elements but we still have N processors. We can perform all of the comparisons for larger clusters of elements in $O(1)$ time in a single step because we have more processors per element.
  - Valiant showed that the size of a cluster for which we can do all of the pair-wise comparisons in a single step grows as 2^{k^2} where k is the number of steps.
  - This leads to a $\log \log N$ time bound for finding the max.
- I'll sketch the proof.
- Then we'll look at why this shows that you can't actually build a PRAM.

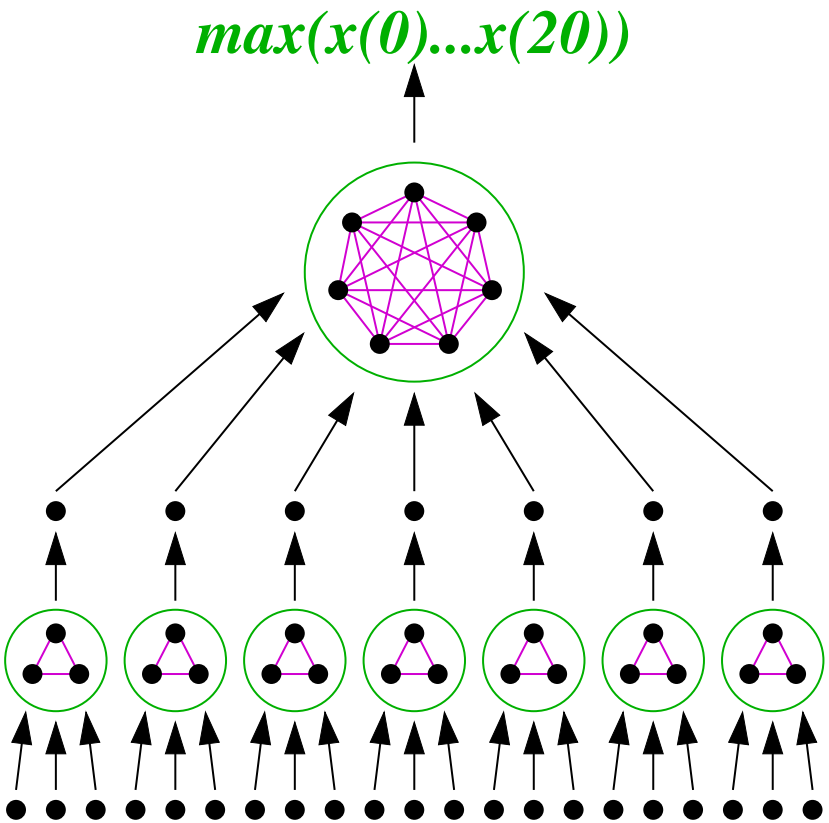
Valiant's algorithm, step 1

- 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.
- The PRAM operations in a bit more detail.
  - Initially, every element has a flag set to `1` that says "might be the max".
  - When $\binom{k}{2}$ processors perform all of the pairwise comparisons of k values,
    - ★ Each processor sets the flag for the smaller value to `0`.
    - ★ Note that several processors may write `0` to the same location, but the CRCW allows this because they are all writing the same value.
  - One processor for each value checks if its flag is still set to `1`.
    - ★ The winner for the cluster is moved to a specific location;
    - ★ The flag for that location is set to `1`
    - ★ And now we're ready for subsequent rounds.

Valiant's algorithm, 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 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

Valiant's Algorithm, the remaining steps

- On step k, we have N/m_k elements left.
- On step m_k is the "sparsity" of the problem – i.e. the number of processors per remaining element.
- We can make groups of $2m_k + 1$ elements, and have

$$\begin{aligned} m_k(2m_k + 1) &= \frac{(2m_k+1)((2m_k+1)-1)}{2} \\ &= \binom{2m_k + 1}{2} \end{aligned}$$

  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.
- Therefore, $m_{k+1} = 2m_k^2 + m_k$.
  - 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.
  - See slide 32 to see the details of the first few rounds.

Valiant's Algorithm, run-time

- The sparsity is roughly 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.
- See slide 33 for the details.

Take-home message from Valiant's algorithm

- The PRAM model is simple, and elegant, and many clever algorithms have been designed based on the PRAM model.
- It is also physically unrealistic:
  - As shown on slide 8, logic gates have bounded fan-in and fan-out.
  - Implementing the processor to memory interconnect requires a logic network of depth $\Omega(\log P)$.
  - Therefore, access time must be $\Omega(\log P)$.
  - Each step of the PRAM must take $\Omega(\log P)$ physical time.
- Valiant's $O(\log \log N)$ algorithms takes $O(\log N \log \log N)$ physical time
  - It's slower than doing a simple reduce.
  - And it uses lots of communication – think of all those λ penalties!
  - But it's very clever. ☺
- Valiant understood this and pointed these issues in his paper.
  - But there has still be extensive research on PRAM algorithms.
  - It's an elegant model, what can I say?

Summary

- Simplified CTA reminds us that communication is expensive, but it doesn't explicitly charge for bandwidth.
- LogP accounts for bandwidth, but doesn't recognize that all bandwidth is not the same:
  - Communicating with an immediate neighbour is generally much cheaper than communicating with a distant machine.
  - Otherwise stated, the bisection bandwidth for real machines is generally much less than the per-machine bandwidth times the number of machines.
    - We can't have everyone talk at once at full bandwidth.
    - logP uses the bisection bandwidth – this is conservative, but it doesn't recognize the advantages of local communication.
- 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 heterogeneity of today's parallel computers:
    - multi-core on chip, faster communication between processors on the same board than across boards, etc.
- We'll use CTA because it's simple.
  - 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.

Preview

October 12: Energy, Power, and Time
October 13: Homework 3 released
October 15: Sorting Networks
October 17: The 0-1 Principle
October 18: HW 3 earlybird (11:59pm).
October 19: Midterm Review Homework: HW3 due: **12 noon**.
October 22: Midterm
October 24-26: Sorting (second half)
October 29-November 30: Data Parallelism with CUDA

Review

- 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 might 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.
- What does the '**g**' stand for in "logP"?

For further reading

- [Valiant1975] Leslie G. Valiant, "Parallelism in Comparison Problems," *SIAM Journal of Computing*, vol. 4, no. 3, pp. 348–355, (Sept. 1975).
- [Fortune1979] Steven Fortune and James Wyllie, "Parallelism in Random Access Machines," *Proceeding of the* 11th *ACM Symposium on Theory of Computing* (STOC'79), pp. 114–118, May 1978.
- [Snyder1986] Lawrence Snyder, "Type architectures, shared memory, and the corollary of modest potential", *Annual review of computer science*, vol. 1, no. 1, pp. 289–317, 1986.
- [Culler1993] David Culler, Richard Karp, *et al.*, "LogP: towards a realistic model of parallel computation," *ACM SIGPLAN Notices*, vol. 28, no. 7, pp. 1–12, (July 1993).

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.

Valiant Details

round	values remaining	group size	processors per group
1	N	$2 * 1 + 1 = 3$	$3 = 3$ choose 2
2	$\frac{N}{3}$	$2 * 3 + 1 = 7$	$3 * 7 = 21 = 7$ choose 2
3	$\frac{1}{7}\frac{N}{3} = \frac{N}{21}$	$2 * 21 + 1 = 43$	$21 * 43 = 903 = 43$ choose 2
4	$\frac{1}{43}\frac{N}{21} = \frac{N}{903}$	$2 * 903 + 1 = 1,807$	$903 * 1,807 = 1,631,721 = 1807$ choose 2
...	...	...	...
k	$\frac{N}{m_k}$	$2m_k + 1$	$m_k(2m_k + 1) = (2m_k + 1)$ choose 2
$k+1$	$\frac{1}{2m_k+1}$ $= \frac{N}{m_k}\frac{N}{m_k(2m_k+1)}$ $= \frac{N}{m_{k+1}}$	$2m_{k+1} + 1$	$m_{k+1}(2m_{k+1} + 1) = (2m_{k+1} + 1)$ choose 2

- m_k is the "sparsity" at round k:

$$\begin{aligned} m_1 &= 1 \\ m_{k+1} &= m_k(2m_k + 1) \end{aligned}$$

- 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 $k \geq 3$, $m_k > 2^{2^{k-1}}$.
- Therefore, if $N \geq 2$, $k > \log\log(N) + 1 \Rightarrow m_k > N$.

Let's solve the run-time recurrence

- For Valiant's algorithm. Let $m_0 = 3$ denote the sparsity at the first step.
- $m_{k+1} = 2m_k^2 + m_k$
  - $\log_2 m_{k+1} = \log_2(2m_k^2 + m_k)$
  - $2\log_2 m_k + 1 < \log_2 m_{k+1} < 2\log_2 m_k + 1 + \alpha/m_k$; where $\alpha = \log_2(e)/2$.
  - $2^k \log_2 m_0 + 2^k - 1 < \log_2 m_k < 2^k m_0 + (5/4)(2^k - 1)$; because $m_k \geq 3$, $\log_2(e)/6 = 0.240449\ldots < 1/4$.
  - $(1 + \log_2 3)2^k - 1 < \log_2 m_k < ((5/4) + \log_2 3)2^k - (5/4)$; because $m_0 = 3$.
- We want to find k such that $m_k \geq N$. It is sufficient if
  - $(1 + \log_2 3)2^k - 1 > \log_2 N$
  - $2^k > (\log_2 N + 1)/(1 + \log_2 3)$
  - $k > \log_2[(\log_2 N + 1)/(1 + \log_2 3)]$
  - For $N > 2$, $(\log_2 N + 1)/(1 + \log_2 3) < \log_2 N$.
- For $N > 2$, let $k = \log_2 \log_2 N$. We have shown that $m_k > N$.
  - Valiant's algorithm takes $O(\log \log N)$ rounds.
  - Each round takes constant time on a CRCW PRAM.
  - $\therefore$ Valiant's algorithm takes $O(\log \log N)$ time on a CRCW PRAM.