Models of Computation for Massive Data

Jeff M. Phillips

August 28, 2013
Outline

Sequential:
- External Memory / (I/O)-Efficient
- Streaming

Parallel:
- PRAM and BSP
- MapReduce
- GP-GPU
- Distributed Computing

![Graph showing runtime vs. data size]
RAM Model

RAM model (Von Neumann Architecture):

- CPU and Memory
- CPU Operations (+, −, *, …) constant time
- Data stored as *words*, not *bits*.
- **Read, Write** take constant time.
Today’s Reality

What your computer actually looks like:

- 3+ layers of memory hierarchy.
- Small number of CPUs.

Many variations!
RAM Model

RAM model (Von Neumann Architecture):

- CPU and Memory
- CPU Operations (+, −, *, ...) constant time
- Data stored as words, not bits.
- Read, Write take constant time.
External Memory Model

- $N =$ size of problem instance
- $B =$ size of disk block
- $M =$ number of items that fits in Memory
- $T =$ number of items in output
- $I/O =$ block move between Memory and Disk
External Memory Model

- $N =$ size of problem instance
- $B =$ size of disk block
- $M =$ number of items that fits in Memory
- $T =$ number of items in output
- $I/O =$ block move between Memory and Disk

Advanced Data Structures: Sorting, Searching
Streaming Model

CPU makes "one pass" on data

- Ordered set $A = \langle a_1, a_2, \ldots, a_m \rangle$
- Each $a_i \in [n]$, size $\log n$
- Compute $f(A)$ or maintain $f(A_i)$ for $A_i = \langle a_1, a_2, \ldots, a_i \rangle$.
- Space restricted to $S = O(poly(\log m, \log n))$.
- Updates $O(poly(S))$ for each $a_i$. 
CPU makes "one pass" on data

- Ordered set \( A = \langle a_1, a_2, \ldots, a_m \rangle \)
- Each \( a_i \in [n] \), size \( \log n \)
- Compute \( f(A) \) or maintain \( f(A_i) \) for \( A_i = \langle a_1, a_2, \ldots, a_i \rangle \).
- Space restricted to \( S = O(\text{poly}(\log m, \log n)) \).
- Updates \( O(\text{poly}(S)) \) for each \( a_i \).

Advanced Algorithms: Approximate, Randomized
PRAM

Many ($p$) processors. Access shared memory:

- EREW: Exclusive Read Exclusive Write
- CREW: Concurrent Read Exclusive Write
- CRCW: Concurrent Read Concurrent Write

Simple model, but has shortcomings...
...such as Synchronization.
PRAM

Many \((p)\) processors. Access shared memory:

- EREW: Exclusive Read
  Exclusive Write
- CREW: Concurrent Read
  Exclusive Write
- CRCW: Concurrent Read
  Concurrent Write

Simple model, but has shortcomings...
...such as Synchronization.

Advanced Algorithms
Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Procedes in Rounds:

1. Compute: Each processor computes on its own Data: \( w_i \).
2. Synchronize: Each processor sends messages to others:
   \[ s_i = \text{MessSize} \times \text{CommCost} \. \]
3. Barrier: All processors wait until others done.

Runtime: \( \max w_i + \max s_i \)

Pro: Captures Parallelism and Synchronization
Con: Ignores Locality.
Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Proceeds in Rounds:

1. **Compute**: Each processor computes on its own Data: $w_i$.
2. **Synchronize**: Each processor sends messages to others:
   $s_i = \text{MessSize} \times \text{CommCost}$.
3. **Barrier**: All processors wait until others done.

Runtime: $\max w_i + \max s_i$

**Pro**: Captures Parallelism and Synchronization
**Con**: Ignores Locality.
Bulk Synchronous Parallel

Each Processor has its own Memory
Parallelism Proceeds in Rounds:

1. Compute: Each processor computes on its own Data: $w_i$.

2. Synchronize: Each processor sends messages to others:
   $s_i = \text{MessSize} \times \text{CommCost}$.

3. Barrier: All processors wait until others done.

Runtime: $\max w_i + \max s_i$

Pro: Captures Parallelism and Synchronization
Con: Ignores Locality.
MapReduce

Each Processor has full hard drive, data items $\langle \text{KEY, VALUE} \rangle$. Parallelism Proceeds in Rounds:

- **Map**: assigns items to processor by KEY.
- **Reduce**: processes all items using VALUE. Usually combines many items with same KEY.

**Repeat** M+R a constant number of times, often only one round.

- Optional post-processing step.

**Pro**: Robust (duplication) and simple. Can harness Locality
**Con**: Somewhat restrictive model
**MapReduce**

Each Processor has full hard drive, data items \(< \text{KEY, VALUE} >\).

Parallelism Proceeds in Rounds:

- **Map**: assigns items to processor by \text{KEY}.
- **Reduce**: processes all items using \text{VALUE}. Usually combines many items with same \text{KEY}.

**Repeat** \(M+R\) a constant number of times, often only one round.

- Optional post-processing step.

Pro: Robust (duplication) and simple. Can harness Locality

Con: Somewhat restrictive model

**Advanced Algorithms**
General Purpose GPU

Massive parallelism on your desktop. Uses Graphics Processing Unit. Designed for efficient video rasterizing. Each processor corresponds to pixel $p$

- depth buffer:
  $$D(p) = \min_i ||x - w_i||$$
- color buffer:
  $$C(p) = \sum_i \alpha_i \chi_i$$
- ...

Con: Somewhat restrictive model, hierarchy. Small memory.
Distributed Computing

Many small slow processors with data. Communication very expensive.

- Report to base station
- Merge tree
- Unorganized (peer-to-peer)

Data collection or Distribution
Distributed Computing

Many small slow processors with data. Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

Data collection or Distribution
Distributed Computing

Many small slow processors with data. Communication very expensive.

- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

Data collection or Distribution
Distributed Computing

Many small slow processors with data. Communication very expensive.
- Report to *base station*
- Merge tree
- Unorganized (peer-to-peer)

Data collection or Distribution

Advanced Algorithms: Approximate, Randomized
Themes

What are course goals?
  ▶ How to analyze algorithms in each model
  ▶ Taste of how to use each model
  ▶ When to use each model
Themes

What are course goals?
▶ How to analyze algorithms in each model
▶ Taste of how to use each model
▶ When to use each model

Work Plan:
▶ 1-3 weeks each model.
  ▶ Background and Model.
  ▶ Example algorithms analysis in each model.

<table>
<thead>
<tr>
<th>I/O</th>
<th>Stream</th>
<th>Parallel</th>
<th>MapReduce</th>
<th>GPU</th>
<th>Distributed</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Class Work

1 Credit Students:
- Attend Class. (some Fridays less important)
- Ask Questions.
- If above lacking, may have quizzes.
- Scribing Notes, Video-taping Lectures, or Giving Lectures.

3 Credit Students:
*Must also do a project!*
- Project Proposal (Aug 30).
  Approved or Rejected by Sept 4.
- Presentations (Dec 11 or 13).
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  - TM: $O(n)$
  - VNA: $O(n)$

- Sorting:
  - TM: $O(n^2)$
  - VNA: $O(n \log n)$

- Searching:
  - TM: $O(n)$
  - VNA: $O(\log n)$
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):
  - TM: $O(n)$
  - VNA: $O(n)$
- Sorting:
  - TM: $O(n^2)$
  - VNA: $O(n \log n)$
- Searching:
  - TM: $O(n)$
  - VNA: $O(\log n)$
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):

  - \(\mathcal{O}(n)\)
  - \(\mathcal{O}(n \log n)\)
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):
  - TM: $O(n)$  
  - VNA: $O(n)$
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):
  - TM: $O(n)$  VNA: $O(n)$
- Sorting:
Sequential Review

Turing Machines (Alan Turing 1936)

- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)

- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?

- Scanning (max):
  - TM: $O(n)$
  - VNA: $O(n)$

- Sorting:
  - TM: $O(n^2)$
  - VNA: $O(n \log n)$
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):
  TM: $O(n)$  VNA: $O(n)$
- Sorting:
  TM: $O(n^2)$  VNA: $O(n \log n)$
- Searching:
Sequential Review

Turing Machines (Alan Turing 1936)
- Single Tape: MoveL, MoveR, read, write
- each constant time
- content pointer memory, infinite tape (memory)

Von Neumann Architecture (Von Neumann + Eckert + Mauchly 1945)
- based on ENIAC
- CPU + Memory (RAM): read, write, op = constant time

How fast are the following?
- Scanning (max):
  - TM: $O(n)$  
  - VNA: $O(n)$
- Sorting:
  - TM: $O(n^2)$  
  - VNA: $O(n \log n)$
- Searching:
  - TM: $O(n)$  
  - VNA: $O(\log n)$
## Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n)</th>
<th>10</th>
<th>10^2</th>
<th>10^3</th>
<th>10^4</th>
<th>10^5</th>
<th>10^6</th>
<th>10^7</th>
<th>10^8</th>
<th>10^9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.000007</td>
</tr>
</tbody>
</table>
Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n = )</th>
<th>(10)</th>
<th>(10^2)</th>
<th>(10^3)</th>
<th>(10^4)</th>
<th>(10^5)</th>
<th>(10^6)</th>
<th>(10^7)</th>
<th>(10^8)</th>
<th>(10^9)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.00001</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>&gt;15</td>
</tr>
</tbody>
</table>
Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n = )</th>
<th>10</th>
<th>10²</th>
<th>10³</th>
<th>10⁴</th>
<th>10⁵</th>
<th>10⁶</th>
<th>10⁷</th>
<th>10⁸</th>
<th>10⁹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.001871</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>&gt;15 yrs</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td></td>
</tr>
</tbody>
</table>
Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n = )</th>
<th>10</th>
<th>10²</th>
<th>10³</th>
<th>10⁴</th>
<th>10⁵</th>
<th>10⁶</th>
<th>10⁷</th>
<th>10⁸</th>
<th>10⁹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.000007</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>&gt;15</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td></td>
</tr>
<tr>
<td>Bubble</td>
<td>0.000003</td>
<td>0.000105</td>
<td>0.007848</td>
<td>0.812912</td>
<td>83.12960</td>
<td>~2 hour</td>
<td>~9 days</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n =)</th>
<th>10</th>
<th>10^2</th>
<th>10^3</th>
<th>10^4</th>
<th>10^5</th>
<th>10^6</th>
<th>10^7</th>
<th>10^8</th>
<th>10^9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.000007</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>&gt;15 min</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td>-</td>
</tr>
<tr>
<td>Bubble</td>
<td>0.000003</td>
<td>0.000105</td>
<td>0.007848</td>
<td>0.812912</td>
<td>83.12960</td>
<td>∼2 hour</td>
<td>∼9 days</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

Complexity Theory:

- \(\text{LOG: } \text{poly } \log(n) = \log^c n \) (... need to load data)
Asymptotics

How large (in seconds) is:

- Searching ($\log n$)
- Max ($n$)
- Merge-Sort ($n \log n$)
- Bubble-Sort ($n^2$) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>$n =$</th>
<th>10</th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
<th>$10^8$</th>
<th>$10^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.001871</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>$&gt;15$</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td>-</td>
</tr>
<tr>
<td>Bubble</td>
<td>0.000003</td>
<td>0.000105</td>
<td>0.007848</td>
<td>0.812912</td>
<td>83.12960</td>
<td>$\sim2$ hour</td>
<td>$\sim9$ days</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Complexity Theory:

- **LOG**: poly log($n$) = $\log^c n$ (... need to load data)
- **P**: poly($n$) = $n^c$ (many cool algorithms)
Asymptotics

How large (in seconds) is:

- Searching \((\log n)\)
- Max \((n)\)
- Merge-Sort \((n \log n)\)
- Bubble-Sort \((n^2)\) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>(n=)</th>
<th>10</th>
<th>10²</th>
<th>10³</th>
<th>10⁴</th>
<th>10⁵</th>
<th>10⁶</th>
<th>10⁷</th>
<th>10⁸</th>
<th>10⁹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.000009</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.0000048</td>
<td>0.00000387</td>
<td>0.00000398</td>
<td>0.003987</td>
<td>0.040698</td>
<td>9.193978</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td>-</td>
</tr>
<tr>
<td>Bubble</td>
<td>0.000003</td>
<td>0.000105</td>
<td>0.007848</td>
<td>0.812912</td>
<td>83.12960</td>
<td>~2 hour</td>
<td>~9 days</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Complexity Theory:

- \(\text{LOG} \): \(\text{poly} \log(n) = \log^c n\) (... need to load data)
- \(\text{P} \): \(\text{poly}(n) = n^c\) (many cool algorithms)
- \(\text{EXP} \): \(\exp(n) = c^n\) (usually hopeless ... but 0.00001\(^n\) not bad)
Asymptotics

How large (in seconds) is:

- Searching (log $n$)
- Max ($n$)
- Merge-Sort ($n \log n$)
- Bubble-Sort ($n^2$) ... or Dynamic Programming

<table>
<thead>
<tr>
<th>$n =$</th>
<th>10</th>
<th>$10^2$</th>
<th>$10^3$</th>
<th>$10^4$</th>
<th>$10^5$</th>
<th>$10^6$</th>
<th>$10^7$</th>
<th>$10^8$</th>
<th>$10^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000001</td>
<td>0.000002</td>
<td>0.000002</td>
<td>0.000007</td>
<td>0.000002</td>
</tr>
<tr>
<td>Max</td>
<td>0.000003</td>
<td>0.000005</td>
<td>0.000006</td>
<td>0.000048</td>
<td>0.000387</td>
<td>0.003988</td>
<td>0.040698</td>
<td>9.193987</td>
<td>$&gt;15$</td>
</tr>
<tr>
<td>Merge</td>
<td>0.000005</td>
<td>0.000030</td>
<td>0.000200</td>
<td>0.002698</td>
<td>0.029566</td>
<td>0.484016</td>
<td>7.833908</td>
<td>137.9388</td>
<td>-</td>
</tr>
<tr>
<td>Bubble</td>
<td>0.000003</td>
<td>0.000105</td>
<td>0.007848</td>
<td>0.812912</td>
<td>83.12960</td>
<td>83.12960</td>
<td>83.12960</td>
<td>83.12960</td>
<td>-</td>
</tr>
</tbody>
</table>

Complexity Theory:

- LOG: poly log($n$) = log$^c n$ (... need to load data)
- P : poly($n$) = $n^c$ (many cool algorithms)
- EXP: exp($n$) = $c^n$ (usually hopeless ... but 0.00001$^n$ not bad)
- NP: verify solution in P, find solution conjectured EXP (If EXP number parallel machines, then in P time)
Data Group

Data Group Meeting
Thursdays @ 12:15-1:30pm in LCR
(to be confirmed)

http://datagroup.cs.utah.edu/dbgroup.php