Big Iron for Big Data: An Unnatural Alliance?

Steve Plimpton
Sandia National Labs

Salishan Conference on High-Speed Computing
April 2012
Programming model:
- SS: polish same code for years, tune its performance
- BD: quick & dirty or agile, morph tools for data
- BD: less emphasis on performance, more on programmer effort
Big data analytics (BD) versus scientific simulations (SS)

- **Programming model:**
  - SS: polish same code for years, tune its performance
  - BD: quick & dirty or agile, morph tools for data
  - BD: less emphasis on performance, more on programmer effort

- **Computational load:**
  - SS: computation heavy (not memory or I/O bound), in-core
  - BD: computation light (often I/O bound), often out-of-core
Big data analytics (BD) versus scientific simulations (SS)

- **Programming model:**
  - SS: polish same code for years, tune its performance
  - BD: quick & dirty or agile, morph tools for data
  - BD: less emphasis on performance, more on programmer effort

- **Computational load:**
  - SS: computation heavy (not memory or I/O bound), in-core
  - BD: computation light (often I/O bound), often out-of-core

- **Computational structure:**
  - SS: either structured or lots of effort to load-balance
  - BD: often dramatically unstructured (power-law graphs)
  - BD: less emphasis on enforcing/exploiting data locality

- **Data usage & ownership:**
  - SS: big data is an output, not an input
  - BD: big data is an input, maybe an output
  - BD: data may be sensitive or proprietary
  - BD: compute where data is, "own" the data
Big data analytics (BD) versus scientific simulations (SS)

- **Programming model:**
  - SS: polish same code for years, tune its performance
  - BD: quick & dirty or agile, morph tools for data
  - BD: less emphasis on performance, more on programmer effort

- **Computational load:**
  - SS: computation heavy (not memory or I/O bound), in-core
  - BD: computation light (often I/O bound), often out-of-core

- **Computational structure:**
  - SS: either structured or lots of effort to load-balance
  - BD: often dramatically unstructured (power-law graphs)
  - BD: less emphasis on enforcing/exploiting data locality

- **Data usage & ownership:**
  - SS: big data is an output, not an input
  - BD: big data is an input, maybe an output
  - BD: data may be sensitive or proprietary
  - BD: compute where data is, “own” the data
Olympic metric for price: gold, silver, bronze
Olympic metric for **price**: gold, silver, bronze

**Gold-plated** solution:

- top-10 machine (petascale, ORNL Jaguar)
- **$100M**, 224K cores
- custom Spider parallel file system (Lustre)
  - not NFS, tuned for big-chunk read/writes
  - 10 Pbytes, 13K disks
  - 240 GB/sec, IOPs $= \sim 1M$
- data itself not valuable (regenerate by simulation)
- code & CPU time to create it is valuable
Olympic metric for **price**: gold, aluminum, bronze

**Aluminum-plated** solution:

- top-500 machine (bioinformatics cluster at Columbia U)
- **$2.5M**, 4000 cores using data 24/7
- clustered commercial NAS (Isilon, Panasas, ...)
  - NFS, scalable capacity, 1B small files
  - 1 Pbyte, 1000 disks
  - 20 Gb/sec, 500K IOPs
- data generated externally (> Moore’s and Kryder’s laws)
- data is highly valuable, must be easy to manage/backup
Olympic metric for *price*: gold, aluminum, plywood

**Plywood-plated** solution:

- Google, Facebook, Twitter = racks of cheap cores and disks
- *minimal $\$$*, could not care less about top-500 and LINPACK
- 1+ disk/core:
  - scalable capacity
  - 100 Pbytes, 100K disks (made this up)
  - 10 TB/sec, 20M IOPs
- my summer vacation photos may be valuable
- much of data is not valuable (continually refreshed)
## Bottom line

<table>
<thead>
<tr>
<th>Medal</th>
<th>$/PByte</th>
<th>TB/core</th>
<th>PB/Pflop</th>
<th>IOPs/PB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold</strong></td>
<td>$10M</td>
<td>0.044</td>
<td>5</td>
<td>(~100K)</td>
</tr>
<tr>
<td><strong>Aluminum</strong></td>
<td>$2.5M</td>
<td>0.25</td>
<td>40</td>
<td>500K</td>
</tr>
<tr>
<td><strong>Plywood</strong></td>
<td>$0.3M</td>
<td>1+</td>
<td>100+</td>
<td>(~200K)</td>
</tr>
</tbody>
</table>

Big data computing done on aluminum and plywood. No one wants to pay gold prices to do big data computing. Don’t want to pay for compute speed & interconnect. Do want to pay for storage capacity and I/O capability.

Additional issues:
- Run where data is produced & stored (local vs center)
- Data accessibility (add, delete, backup)
### Bottom line

<table>
<thead>
<tr>
<th>Medal:</th>
<th>$/PByte</th>
<th>TB/core</th>
<th>PB/Pflop</th>
<th>IOPs/PB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold:</strong></td>
<td>$10M</td>
<td>0.044</td>
<td>5</td>
<td>$\sim 100K$</td>
</tr>
<tr>
<td><strong>Aluminum:</strong></td>
<td>$2.5M</td>
<td>0.25</td>
<td>40</td>
<td>500K</td>
</tr>
<tr>
<td><strong>Plywood:</strong></td>
<td>$0.3M</td>
<td>1+</td>
<td>100+</td>
<td>$\sim 200K$</td>
</tr>
</tbody>
</table>

- Big data computing done on aluminum and plywood
- No one wants to pay *gold prices* to do big data computing
- Don’t want to pay for compute speed & interconnect
- Do want to pay for storage capacity and I/O capability
### Bottom line

<table>
<thead>
<tr>
<th>Medal</th>
<th>$/PByte</th>
<th>TB/core</th>
<th>PB/Pflop</th>
<th>IOPs/PB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Gold</strong></td>
<td>$10M</td>
<td>0.044</td>
<td>5</td>
<td>~100K</td>
</tr>
<tr>
<td><strong>Aluminum</strong></td>
<td>$2.5M</td>
<td>0.25</td>
<td>40</td>
<td>500K</td>
</tr>
<tr>
<td><strong>Plywood</strong></td>
<td>$0.3M</td>
<td>1+</td>
<td>100+</td>
<td>~200K</td>
</tr>
</tbody>
</table>

- Big data computing done on aluminum and plywood
- No one wants to pay gold prices to do big data computing
- Don’t want to pay for compute speed & interconnect
- Do want to pay for storage capacity and I/O capability
- **Additional issues:**
  - run where data is produced & stored (local vs center)
  - data accessibility (add, delete, backup)
MapReduce for scientific data


- 1M atoms, 100M snapshots ⇒ 3 Pbytes
- Stats on where each atom traveled
  - near-approach to docking site
  - membrane crossings
MapReduce for scientific data


- 1M atoms, 100M snapshots ⇒ 3 Pbytes
- Stats on where each atom traveled
  - near-approach to docking site
  - membrane crossings

- Data is stored exactly wrong for this analysis
- MapReduce solution:
  1. **map**: read snapshot, emit key = ID; value = (time, xyz)
  2. **communicate**: aggregate all values with same ID
  3. **reduce**: order the values, perform analysis
MapReduce for scientific data


- 1M atoms, 100M snapshots $\Rightarrow$ 3 Pbytes
- Stats on where each atom traveled
  - near-approach to docking site
  - membrane crossings

- Data is stored exactly wrong for this analysis
- MapReduce solution:
  1. **map**: read snapshot, emit key = ID; value = (time, xyz)
  2. **communicate**: aggregate all values with same ID
  3. **reduce**: order the values, perform analysis

- **Key point**: extremely parallel comp + MPI_All2all comm
Why is MapReduce attractive?

- **Plus:**
  - write only the code that only you can write
  - write zero parallel code (no parallel debugging)
  - out-of-core for free

- **Plus/minus (features!):**
  - ignore data locality
  - load balance thru random distribution
    - key hashing = slow global address space
  - maximize communication (all2all)

- **Minus:**
  - have to re-cast your algorithm as a MapReduce
Why is MapReduce attractive?

**Plus:**
- write only the code that only you can write
- write zero parallel code (no parallel debugging)
- out-of-core for free

**Plus/minus (features!):**
- ignore data locality
- load balance thru random distribution
  - key hashing = slow global address space
- maximize communication (all2all)

**Minus:**
- have to re-cast your algorithm as a MapReduce

Matches big data programming model:
**minimal human effort**, not maximal performance
Hadoop:
- HDFS, fault tolerance
- extra big-data goodies (BigTable, etc)
- no one runs it on gold-plated platforms
MapReduce software

- Hadoop:
  - HDFS, fault tolerance
  - extra big-data goodies (BigTable, etc)
  - no one runs it on gold-plated platforms

  - MapReduce on top of MPI
  - Lightweight, portable, C++ library with C API
  - Out-of-core on big iron if each proc can write scratch files
  - No HDFS (parallel file system with data redundancy)
  - No fault-tolerance (blame it on MPI)
What could you do with MapReduce at Petascale?

- Post-simulation analysis of big data output
- Graph algorithms:
  - vertex ranking via PageRank (460)
  - connected components (250)
  - triangle enumeration (260)
  - single-source shortest path (240)
  - sub-graph isomorphism (430)

Matrix operations:
- matrix-vector multiply (PageRank kernel)
- tall-skinny QR (D Gleich, P Constantine)

Simulation data \implies cheaper surrogate model
- 500M x 100 dense matrix \implies 30 min on 256 plywood cores

Machine learning: classification, clustering, ...

Win the TeraSort benchmark
What could you do with MapReduce at Petascale?

- Post-simulation analysis of big data output
- Graph algorithms:
  - vertex ranking via PageRank (460)
  - connected components (250)
  - triangle enumeration (260)
  - single-source shortest path (240)
  - sub-graph isomorphism (430)
- Matrix operations:
  - matrix-vector multiply (PageRank kernel)
  - tall-skinny QR (D Gleich, P Constantine)
    - simulation data ⇒ cheaper surrogate model
    - 500M x 100 dense matrix ⇒ 30 min on 256 plywood cores
- Machine learning: classification, clustering, ...
- Win the TeraSort benchmark
No free lunch: PageRank (matvec) performance

Cray XT3 (gold), 1/4 billion edge highly sparse, irregular matrix

- MapReduce communicates matrix elements
- But recall: load-balance, out-of-core for free
Sub-graph isomorphism for data mining

- Data mining, **needle-in-haystack** anomaly search
- Huge graph with **colored vertices, edges** (labels)
- **SGI** = find all occurrences of small target graph
Sub-graph isomorphism for data mining

- Data mining, **needle-in-haystack** anomaly search
- Huge graph with **colored vertices, edges** (labels)
- **SGI** = find all occurrences of small target graph

![Diagram showing sub-graph isomorphism](image)

<table>
<thead>
<tr>
<th>L</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Sub-graph isomorphism for data mining

- Data mining, **needle-in-haystack** anomaly search
- Huge graph with **colored vertices, edges** (labels)
- **SGI** = find all occurrences of small target graph

Example: 18 Tbytes $\Rightarrow$ 107 B edges $\Rightarrow$ 573 K matches in 55 minutes on 256 plywood cores
Streaming data

- Continuous, real-time I/O
- Stream = small datums at high rate
- **Resource-constrained processing:**
  - only see datums once
  - compute/datum < stream rate
  - only store state that fits in memory
  - age/expire data
- Pipeline model is attractive:
  - datums flow thru compute kernels
  - hook kernels together to perform analysis
  - split stream to enable shared or distributed-memory parallelism
Streaming software

- IBM InfoSphere (commercial)
- Twitter Storm (open-source)

  - Parallel Harness for Informatic Stream Hashing
  - phish swim in a stream
  - runs on top of MPI or sockets (zeromQ)
Streaming software

- IBM InfoSphere (commercial)
- Twitter Storm (open-source)

  - Parallel Harness for Informatic Stream Hashing
  - phish swim in a stream
  - runs on top of MPI or sockets (zeroMQ)

Key point: **zillions of small messages** flowing thru processes
PHISH net for real-time simulation data analysis

- Gold vs aluminum vs plywood:
  - Most streaming is high data rate, low computation
  - Mismatch: continuous streaming versus batch jobs
  - Could couple to simulation for "steering"

```
running simulation

snapshots

Map

Scatter
Scatter
Scatter
....
Scatter

Reduce

IDs

Analyze
Analyze
Analyze
....
Analyze

Trigger

Stats
```
Big Iron and Big Data: An Unnatural Alliance?

Samuel Johnson (1709-1784):

Using big iron + MPI for big data ... is like a dog walking on his hind legs. It is not done well; but you are surprised to find it done at all.
Big Iron and Big Data: An Unnatural Alliance?

Samuel Johnson (1709-1784):

Using big iron + MPI for big data ... is like a dog walking on his hind legs. It is not done well; but you are surprised to find it done at all.

**Unnatural MPI:**

- ignoring data locality
- all2all (MapReduce)
- tiny messages (streaming)
- lots of I/O
If you want that dog to walk at Petascale ...

**Data analytics** for informatics problems (e.g. MapReduce):
- fast all2all for data movement (bisection bandwidth)
- fast I/O rate to parallel disks for out-of-core

**Streaming algorithms** for informatics problems:
- high throughput (zillions of small messages)
- also low latency

**Both** need:
- fast I/O (bandwidth + IOPs), ideally a disk per node
- hi-speed access to external world (source of data)

*Caveat:* have ignored fault tolerance. Hadoop and Twitter Storm have it, MPI does not.
If you want that dog to walk at Petascale ...

Data analytics for informatics problems (e.g. MapReduce):
- fast all2all for data movement (bisection bandwidth)
- fast I/O rate to parallel disks for out-of-core

Streaming algorithms for informatics problems:
- high throughput (zillions of small messages)
- also low latency

Both need:
- fast I/O (bandwidth + IOPs), ideally a disk per node
- hi-speed access to external world (source of data)

Caveat: have ignored fault tolerance
- Hadoop and Twitter Storm have it
- MPI does not
Thanks & links

Sandia collaborators:
- Karen Devine (MR-MPI)
- Tim Shead (PHISH)
- Todd Plantenga, Jon Berry, Cindy Phillips (graph algorithms)

Open-source packages (BSD license):
- [http://mapreduce.sandia.gov](http://mapreduce.sandia.gov) (MapReduce-MPI)

Papers: