Opportunities for XXL Datamining

Marc Snir
Outline

- How a Supercomputer looks like in > 2010
- What it takes to run a DM code on such a platform
- How DM can help supercomputing
Large NSF Funded Supercomputers beyond 2010

- One Petascale platform -- Blue Waters at NCSA, U Illinois
  - Sustained performance: petaflop range
  - Memory: petabyte range
  - Disk: 10’s petabytes
  - Archival storage: exabyte range

- Multiple 1/4 scale platforms at various universities

- Available to NSF-funded “grand challenge” teams on a competitive basis

- **My talk:** What it takes to mine data at such scale
- **Your job:** Think big
The Uniprocessor Crisis

- Manufacturers cannot increase clock rate anymore (power problem)
- Computer architects have run out of productive ideas on how to use more transistors to increase single thread performance
  - Diminishing return on caches
  - Diminishing return on instruction-level parallelism

⇒ Increased processor performance will come only from the increase on number of cores per chip

Petascale = 250K -- 1M threads

Need algorithms with massive levels of parallelism
Average # Processors Top 500 System
Mileage is Less than Advertised

Instruction per cycle, frequent item mining

(M Wei)
It’s the Memory, Stupid

PC Balance
(word operands from memory per flop)

Seem stuck at ~ 1:10 ratio

(source McAlpin)
The Memory Wall and Palliatives

- The problem
  - Memory bandwidth is limited (cost)
  - Compilers cannot issue enough concurrent loads to fill the memory pipe
  - Compilers cannot issue loads early enough to avoid stalls
- Solutions
  - Multicore and vector operations -- to fill the pipe
    Simultaneous multithreading -- to tolerate latency
  - Need even higher levels of parallelism!
Solutions to the Memory Wall

- Caching and **locality**
  - **Need algorithms with good locality**
- Split communication
  - Memory prefetch (local memory)
  - Put/get (remote memory)
  - **Need programmed communication** (not necessarily message-passing)

- **N.B.**: Computer power is essentially free; you pay for storing and moving data
  - Accelerators (GPUs, FPGAs, Cell processors) enhance a non-critical resource, and will often have a negligible impact on overall performance
Load Balancing

- Problem: Amount of computation in DM kernels heavily data dependent -- work partitioning results in load imbalances
- Hard solution: develop good work predictors and do explicit, static load balancing
- Easy solution: use system with task virtualization and dynamic task migration
  - E.g., AMPI (Kale, http://charm.cs.uiuc.edu/) -- scalable, negligible (often negative) overheads
  - Overhead of task migration is few seconds, at worse
    - Parallel file system is shared all
  - Task virtualization essential for modularity and ease of programming
Code Tuning

- Is essential when using a Petascale system
  - 1 hour = $5K - $10K
- Is data dependent (more so with Datamining than with many applications)
- Is platform dependent
Relative Performance of Frequent Item Mining Codes is Input Dependent

three algorithms on six real-world datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>LCM</th>
<th>FP-Growth</th>
<th>Eclat</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess (s=0.18)</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>pumsb (s=0.45)</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>bms-wv-1 (s=0.0005)</td>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>accidents (s=0.16)</td>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>webdocs (s=0.1)</td>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

FIMI workshop needs some thinking…

(C Jiang PhD thesis)
A Schematic View of Performance Tuning

1. Algorithm selection
   (LCM, FP_Growth, Eclat, …)

2. Implementation selection
   (choice of data structures…)

3. Automatic tuning
   (compiler, runtime)

All three should be platform and data dependent
Algorithm Selection is a Classification Problem

- Generated datasets
- Input set generator
- Empirical evaluation
- Alg. pool
- Labeled features
- SVM learning

Training stage

- Input
- Feature
- SVM learning
- Platform specific SVM model
- predict
- Alg.
- Run Alg.

Execution stage

- Can be solved using supervised ML
- Smart part: choice of feature vector that can be computed fast and “works”
Results: Average execution time

- The predicted algorithm is close to optimal (12.5% worse)
- The predicted algorithm is significantly better than LCM (65.3%)
Selected features

- **Size**
  - The number of ‘1’s in the bit matrix

- **Density**
  - Number of ‘1’s divided by number of cells

- **Height**:
  - \(1 - \text{support threshold} / \text{density}\)
  - An estimate of how much room for the support to decrease to the threshold

- **Similarity**:
  - How similar transactions are to each other

Example:

\[
\begin{array}{cccccc}
1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
\end{array}
\]

\[\text{Size}=18\]
\[\text{Density} = 18/30 = 0.6\]
\[\text{Height} = 1-s/density = 1 - 0.2/0.6 = 2/3\]
Implementation Selection

- We represent implementation choices via *tuning patterns*—descriptions of solutions to common software performance optimization problems that are applicable to multiple algorithms
  - Lexicographic ordering
  - Aggregation
  - Compaction
  - Wave-front prefetch
  - Tiling for sparse arrays
  - SIMDization
- Probably need richer ontology (relations, constraints, expert knowledge)

- Classification problem: select best set of tuning patterns
  - Used SVM; GA probably more appropriate
- Good speedup (up to 2.1)
- ALL does not always win!
- Optimal set of tuning patterns is machine and data dependent
Prediction results – LCM

Number of times that each code version is the fastest

- Prediction close to “optimal” (oracle)
- Prediction overhead is negligible

M Wei, PhD thesis, ICDE07
Summary

- Main obstacle to petascale datamining is dreaming of grand challenges that need it

- Petascale datamining requires tuned code
  - Node performance (locality) + scalability

- Should develop tunable code generators to adapt to platform and data
  - Need good training sets!

- Code tuning is a very interesting classification problem
Questions?
Similarity definition

- "Similarity": how similar transactions are to each other

- "Normalized hamming distance" (pair-wise similarity):
  - Given two transactions, their “normalized hamming distance” is the number of differences divided by the total number of unique ones.
  - Example:

    | T1 | T2 |
    |----|----|
    | 1 1 0 1 0 1 |
    | 0 1 0 0 1 1 |

    Difference = 3,
    Therefore, distance(T1,T2) = 3/5 = 0.6

    the number of unique ones is 5
Similarity feature definition

- Normalized hamming distance defines pair-wise distance, but we need a global measure of similarity among all transactions.

- Approach – “Average linkage clustering”
  - Start with $n$ transactions, each as a cluster
  - Merge the two closest into one new cluster
  - Repeat merging until one cluster left.

- “Similarity” = average value of the n-1 clustering distances
Prediction results on real-world datasets

accidents

chess

webdocs

pumsb

Pumsb_star

connect
Prediction results on real-world datasets

- mushroom
- retail
- kosarak

BMS-POS

BMS-WebView1

BMS-WebView2

www.informatics.uiuc.edu
Using synthetic data for training

- IBM Quest dataset Generator
  - Widely used in data mining research

- Problem:
  - The generated dataset is not representative of real-world data

![Synthetic datasets](chart1)

![Real-world datasets](chart2)

Best algorithm
Item frequency curve

![Graph showing item frequency curve with normalized support and items in decreasing frequency order with labels connect, mushroom, T10I4D100K, and T40I10D100K.]
Using modified IBM generator to produce algorithm variability

Modified Synthetic datasets

Real-world datasets
Lexicographic ordering of transactions

- **Preprocess** original database by reordering transactions in lexicographic order
  - Alphabet: items in descending frequency order

- Improves locality of accesses (LCM & FP_Growth); reduces computation (Eclat)
- Overhead of lexicographic ordering
Lexicographic ordering in LCM

- Spatial locality of traversal is improved (fewer jumps)
  - Locality improved for most frequent items
  - Order mostly preserved for projected databases
    - Ordering overhead amortized over multiple traversals
Lexicographic ordering in Eclat

- Range reduction reduces computation

Transactions for \{a, c\}

Mark first 1

Mark last 1

1s becomes contiguous for c

Rows are switched after lexicographic ordering
Lexicographic ordering – project() in FP-Growth

- Tree is constructed by inserting transactions from the original database one by one
- Lexicographic ordering improve the temporal locality for insertions.

<table>
<thead>
<tr>
<th>tid</th>
<th>transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{c, f, a}</td>
</tr>
<tr>
<td>1</td>
<td>{c, f, b}</td>
</tr>
<tr>
<td>2</td>
<td>{c, f, a}</td>
</tr>
<tr>
<td>3</td>
<td>{d, e}</td>
</tr>
<tr>
<td>4</td>
<td>{c, f, a, b, d, e}</td>
</tr>
</tbody>
</table>

What’s next?

<table>
<thead>
<tr>
<th>tid</th>
<th>transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{c, f, a}</td>
</tr>
<tr>
<td>1</td>
<td>{c, f, a}</td>
</tr>
</tbody>
</table>

(1) Pseudo node
(2) Already in the cache
Lexicographic ordering – `project()` in FP-Growth

- Access pattern: From an intermediate node to root
- More (parent, child) pairs are contiguous in the memory – better spatial locality

![Diagram showing lexicographic ordering and its implications on memory access patterns.](image)
Wave-front prefetch

Array of short linked lists
- Prefetch pointers from different linked-lists in one iteration
  ✓ Hides memory latency
  ✗ Increases register pressure

Can be used even if lists are of different length
Tiling (LCM)

- Improves temporal locality
- Slightly increases instruction count and memory pressure
Programming patterns applied

<table>
<thead>
<tr>
<th>Patterns</th>
<th>LCM</th>
<th>Eclat</th>
<th>FP-Growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexicographic ordering</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Aggregation</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td>Compaction</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td>Pointer prefetching</td>
<td>N/A</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td>Tiling</td>
<td>✓</td>
<td>N/A</td>
<td>□</td>
</tr>
<tr>
<td>Software prefetch</td>
<td>✓</td>
<td>N/A</td>
<td>✓</td>
</tr>
<tr>
<td>SIMDization</td>
<td>N/A</td>
<td>✓</td>
<td>N/A</td>
</tr>
</tbody>
</table>