Performance Analysis of KDD Applications using Hardware Event Counters

CAP Theme 2


Peter Christen and Adam Czezowski

Peter.Christen@anu.edu.au
Adam.Czezowski@anu.edu.au

6 February 2002
1 Overview

- A Short Introduction to KDD (or Data Mining)
- Three KDD Applications used for Performance Analysis
  - Decision Tree Induction (C4.5)
  - Market Basket Analysis (APRIORI)
  - Predictive Model (ADDFIT)
- Performance Analysis
- Hardware Performance Counters
  - Libraries (PAPI, PCL, libcpc)
  - libcpc Example Program
  - UltraSPARC III Hardware Events
- Experiments and Results
- Conclusions and Outlook
2 **A short Introduction to KDD**

- Analysis of massive and complex data collections (with Giga- and Terabytes, some even Petabytes, and hundreds of attributes)
- Discovery of previously unknown information (data driven exploration)
- Modelling of the data (predict future behaviour using available and historic data)
- Data analysis can not be done manually
- KDD applications include
  - Customer profiling and segmentation, E-Commerce and E-Business
  - Market basket analysis, fraud detection
  - Improvement of health services
  - Analysis of Human Genomic Data
  - Web and text mining

*Many organisations are data rich but information poor*
3 **KDD Techniques and Technologies**

- Techniques: Clustering, classification, neural and Bayesian networks, predictive modelling, association rules, genetic algorithms, etc.
- KDD became possible with powerful (multiprocessor) computers, and large (automatic) data collection and storage
- Efficient and scalable (with data size and complexity) techniques and algorithms are needed
- KDD is multi-disciplinary, using technologies from
  - Databases
  - Machine learning
  - Applied statistics
  - Pattern recognition
  - Computational mathematics
  - High-performance computing
  - Visualisation
4 Decision Tree Induction (C4.5)

- Ross Quinlan, University of New South Wales, 1993
- Given a data set with records (e.g. SQL table), where each record has the same attributes
- Build a classification model of the data (classify records into different classes)
- Data set is split into training set (used to build the decision tree) and test set (to verify the quality of the tree)
- Data structure: Recursive tree (not restricted to binary trees)
- Example:

<table>
<thead>
<tr>
<th>age</th>
<th>student</th>
<th>credit_rating</th>
<th>buys_computer?</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>yes</td>
<td>excellent</td>
<td>yes</td>
</tr>
<tr>
<td>42</td>
<td>yes</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>29</td>
<td>no</td>
<td>excellent</td>
<td>no</td>
</tr>
<tr>
<td>34</td>
<td>no</td>
<td>fair</td>
<td>yes</td>
</tr>
<tr>
<td>27</td>
<td>no</td>
<td>fair</td>
<td>?</td>
</tr>
<tr>
<td>34</td>
<td>yes</td>
<td>excellent</td>
<td>?</td>
</tr>
</tbody>
</table>

![Decision Tree Example](image)
5 Association Rule Induction (APRIORI)

- R. Agrawal, T. Imielinski and A. Swami, 1993
- Popular for Market Basket Analysis
  (trying to find what products customers frequently buy together)
- Given a data set with transactions (can have variable length)
- The task is to (1) find frequent large item sets and then (2) build rules from these item sets
- Data structures: Prefix trees, hash tables
- Example:

<table>
<thead>
<tr>
<th>TID</th>
<th>List of item_IDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T100</td>
<td>I1, I2, I5</td>
</tr>
<tr>
<td>T200</td>
<td>I2, I4</td>
</tr>
<tr>
<td>T300</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T400</td>
<td>I1, I2, I4</td>
</tr>
<tr>
<td>T500</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T600</td>
<td>I2, I3</td>
</tr>
<tr>
<td>T700</td>
<td>I1, I3</td>
</tr>
<tr>
<td>T800</td>
<td>I1, I2, I3, I5</td>
</tr>
<tr>
<td>T900</td>
<td>I1, I2, I3</td>
</tr>
</tbody>
</table>

If I1 and I2 → I5
If I1 and I5 → I2
If I2 and I5 → I1
If I1 → I2
If I1 → I3
6 Additive Models (ADDFIT)

- **ANU Data Mining Group, 2000**

- Build a predictive model of the data with additive functions
  \[ f(x_1, \ldots, x_d) = f_0 + f_1(x_1) + \ldots + f_d(x_d) \]

- **Two steps**
  1. Assemble dense symmetric linear system from data
  2. Solve linear system sequential or in parallel

- Assembly is data dependent and results in irregular memory access

- **Advantages**
  - Linear scalable with dimensionality of the data (number of attributes)
  - Input data set has to be read only once
  - Size of the linear system is independent of the input data
    (only depends on the model)

- Data structure: Symmetric dense linear system
Characteristics of KDD Applications

- Usually access input data (on disk because of its size) several times (ADDFIT only accesses data once)
- Build dynamic and recursive data structures
  - Hash tables
  - Linked lists
  - Trees
- Size of data structures is data dependent (often not linear scalable with input data)
- Data structure access is data dependent (irregular)
- Complex core routines (large instruction foot-prints)

Many KDD applications have irregular memory access patterns and therefore result in sub-optimal performance
8 Performance Analysis

- Modern processors and computer systems are becoming more and more complex
  - Longer pipelines
  - Multiple functional units and multiple instruction issued per cycle
  - Speculative branch predictions
  - Several cache levels
  - Symmetric multiprocessing
- There is an increasing gap between CPU and memory access speed
- Many of today’s complex applications require large amounts of memory (many functions and large data sizes)
- CPU caches are only useful (efficient) when many data items or instructions can be accessed directly from the cache (locality)

*Understanding program behaviour is important to achieve good efficiency and high performance*
9 Performance Analysis Methods

- Profiling: Information about where your program spent its time and which functions called which other functions while it was executing.

- Monitoring system utilisation with commands like: `ps`, `top`, `iostat`, `vmstat`, `kstat`, `cpustat`, `cputrack`, `har`, `pmap`, etc.

- Simulation: Possibility to modify hardware parameters.

- Hardware counters
  - Most modern microprocessors have hardware event counter registers.
  - Possibility to count various hardware events.
  - Control and access through library calls.
  - Easy to instrument source code.
  - Possible to analyse only parts of the code (e.g. computational core routines).
  - Possible to analyse programs with short run times.
Performance Counter Libraries

- Solaris / UltraSPARC
  - The UltraSPARC I, II and III processors have two on-chip hardware counter registers that allow runtime measurements of various hardware events
  - Solaris provides access to these through the libcpc(3LIB) library

- Platform independent libraries
  - PAPI (Performance Application Programming Interface)
    http://icl.cs.utk.edu/projects/papi/
  - PCL (Performance Counter Library)
    http://www.kfa-juelich.de/zam/PCL/

Both PAPI and PCL specify a standard for accessing hardware performance counters available on most modern microprocessors

- Various vendor specific libraries for other processors (including Intel Pentium, PowerPC, MIPS and Alpha) and operating systems
11 Some UltraSPARC III Events

- **MIPS (Million Instructions Per Second)**
  \[
  \text{instr\_cnt} / \text{tick\_cnt} * \text{clock\_freq}
  \]

- **FLOPS (Floating-Point Instructions Per Second)**
  \[
  (\text{fa\_pipe\_completion} + \text{fm\_pipe\_completion}) / \text{tick\_cnt} * \text{clock}
  \]

- **CPI (Cycles Per Instruction)**
  \[
  \text{cycle\_cnt} / \text{instr\_cnt}
  \]

- **Address bus utilisation**
  \[
  (\text{ec\_misses} + \text{ec\_wb}) / (\text{tick\_cnt} * \text{bus\_clock} / \text{cpu\_clock})
  \]

- **Data-Cache miss rate**
  \[
  (\text{dc\_rd\_miss} + \text{dc\_wr\_miss}) / (\text{dc\_rd} + \text{dc\_wr})
  \]

- **Instruction-TLB misses**
  \[
  \text{itlb\_miss} / \text{instr\_cnt}
  \]

*More useful measures are possible, based on 66 UltraSPARC III hardware events*
**libcpc Code Instrumenting**

- Use `#include <libcpc.h>` to include library
- Use `cpc_access()` and `cpc_version()` to check version and accessibility of counters
- Use `cpc_getcpuver()` to get counter configuration
- Use `cpc_strtoevent()` to initialise `cpc_event_t` data structure and fill it with events (given as string)
- Use `cpc_bind_event()` to bind an initialised `cpc_event_t` structure to the calling process
- Use `cpc_take_sample()` to sample counters as desired
- Use `cpc_rele()` to release when done
- Compile with `-lcpc` flag
13  **libcpc Example Program**

```c
#include <libcpc.h>

int cpc_cpuver;

cpc_event_t cpc_event, start, stop;
char *cpc_arg="pic0=cycle_cnt, pic1=instr_cnt";

cpc_cpuver = cpc_getcpuver();
cpc_strtoevent(cpc_cpuver, cpc_arg, &cpc_event);
cpc_bind_event(&cpc_event, 0);

cpc_take_sample(&start);

    /* ... add your code to analyse here ... */

    cpc_take_sample(&stop);

printf("cycle_cnt: %lld, instr_cnt: %lld\n",
    (stop.ce_pic[0]-start.ce_pic[0]), (stop.ce_pic[1]-start.ce_pic[1]));
```

---

CAP Workshop 02  KDD Performance  14 DCS, ANU
### Characteristics of Our Test Programs

<table>
<thead>
<tr>
<th>Program</th>
<th>BLAS (SUNPERF)</th>
<th>ADDFIT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>small</td>
<td>medium</td>
</tr>
<tr>
<td>Data</td>
<td>$209 \times 209$</td>
<td>$660 \times 660$</td>
</tr>
<tr>
<td></td>
<td>104,858 records</td>
<td>209,715 records</td>
</tr>
<tr>
<td>Run time</td>
<td>0.075 sec</td>
<td>1.255 sec</td>
</tr>
<tr>
<td>Iterations</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Heap size</td>
<td>1,024 KB</td>
<td>10,240 KB</td>
</tr>
<tr>
<td>User code</td>
<td>98.54%</td>
<td>99.16%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>APRIORI</th>
<th>C4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>small</td>
<td>large</td>
</tr>
<tr>
<td>Data</td>
<td>T5I4D10K with 10,000 records</td>
<td>T10I8D1000K with 1,000,000 records</td>
</tr>
<tr>
<td>Run time</td>
<td>3.1 sec</td>
<td>42 sec</td>
</tr>
<tr>
<td>Iterations</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>Heap size</td>
<td>19,776 KB</td>
<td>70,512 KB</td>
</tr>
<tr>
<td>User code</td>
<td>97.98%</td>
<td>98.53%</td>
</tr>
</tbody>
</table>
First phase is loading data from files
Second phase is computing frequent item sets and decision tree
Measured with `pmap` using a Python script (for filtering output)
ADDFIT (like BLAS matrix-matrix multiplication) allocates all memory in one block at beginning
MIPS and MFLOPS Measurements

- **MIPS**: Million Instructions Per Second (correspond directly to Instructions Per Cycle)
- **MFLOPS**: Million Floating-Point Instructions Per Second
- **KDD applications are not dominated by floating-point instructions**
- **For KDD applications smaller input data sets result in higher MIPS rate**
Data- and Instruction-Cache Miss Rates

- Instruction-Cache miss rate is much smaller than Data-Cache miss rate.
- Both Data- and Instruction-Cache miss rates are much smaller in user mode than in system (kernel) mode.
- While the Data-Cache miss rate is increasing with larger data sets, the Instruction-Cache miss rate is decreasing.
L2-Cache and Data-TLB Miss Rates

- System (kernel) Level-2 miss rates are much higher than user miss rates (with the exception of C4.5 with the large data set)
- Instruction-TLB miss rates (not shown here) are all smaller than 0.02%
- High miss rates both for Level-2 as well as Data-TLB for C4.5 with the large data set are because of the sorting of entire categorical attributes (using recursive quicksort)
Conclusions and Outlook

- Performance analysis is important to
  - understand characteristics of modern complex applications
  - find bottlenecks both in software (application as well as operating system) and hardware (processor and memory system)
  - improve efficiency and performance of high-performance computer systems

- Hardware counters are a good tool for performance analysis, but it is
  - easy to drown in numbers (many possible measurements)
  - sometimes hard to understand the meaning of the results
  - important to consider side effects from other running programs and the operating system

- Our future research directions
  - Analyse more KDD applications
  - Do analysis on a Primepower SMP system (ANU Supercomputing Facility)
  - Extend analysis to parallel SMP codes