Performance Diagnosis through Classification of Computation Bursts to Known Computational Kernel Behavior

Kevin Huck, Juan González, Judit Gimenez, Jesús Labarta

Dagstuhl Seminar 10181: Program Development for Extreme-Scale Computing
6 May, 2010
Disclaimer...

- This talk represents work (barely) in progress
- Still emerging from the “back of the envelope” stage
- Still performing background research
- Don't quite know if it will work (yet)
Problem Outline

- Given performance counters for an application, how does a user know if a function is “slow”?
- Need *something* to compare to the results
- How does it map to the *potential* of the machine?
- What is the best performance the user can expect?
- Is the application maximizing the potential?
- Can we recommend a treatment?
- Can we use classification to predict performance?
- Performance counters are useful for relative comparisons (parametric studies)
Performance “Check-up”

Application → Collect vital Statistics (symptoms)

Is the application (patient) sick?

Yes: Recommend treatment

No...

...but what is “sick”?
PEPC example...

<table>
<thead>
<tr>
<th>Code Region</th>
<th>% Total</th>
<th>PAPI_TOT_INS</th>
<th>PAPI_TOT_CYC</th>
<th>PAPI_L2_DCM</th>
<th>PAPI_L2_DCA</th>
<th>PAPI_TLB_DM</th>
<th>PAPI_L1_DCM</th>
<th>PAPI_L1_DCA</th>
<th>PAPI_FP_OPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1</td>
<td>39.4</td>
<td>26223145</td>
<td>354122681</td>
<td>1514683</td>
<td>2851495</td>
<td>34411</td>
<td>1490606</td>
<td>9647690</td>
<td>45</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>23.3</td>
<td>71297209</td>
<td>65564277</td>
<td>37537</td>
<td>1449918</td>
<td>1949</td>
<td>850972</td>
<td>40775978</td>
<td>35748232</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>12.6</td>
<td>38435449</td>
<td>42834201</td>
<td>48784</td>
<td>1014255</td>
<td>328260</td>
<td>767939</td>
<td>20731647</td>
<td>2030650</td>
</tr>
<tr>
<td>Cluster 4</td>
<td>9.1</td>
<td>31953051</td>
<td>153824100</td>
<td>674451</td>
<td>1681282</td>
<td>20921</td>
<td>666080</td>
<td>12137954</td>
<td>80628</td>
</tr>
<tr>
<td>Cluster 5</td>
<td>8.3</td>
<td>18541689</td>
<td>149067192</td>
<td>600684</td>
<td>1545183</td>
<td>19163</td>
<td>580653</td>
<td>6551879</td>
<td>5</td>
</tr>
<tr>
<td>Cluster 6</td>
<td>2.5</td>
<td>30529581</td>
<td>47530237</td>
<td>49763</td>
<td>768252</td>
<td>226950</td>
<td>517571</td>
<td>17484268</td>
<td>1523558</td>
</tr>
<tr>
<td>Cluster 7</td>
<td>2.2</td>
<td>17509002</td>
<td>48068912</td>
<td>33544</td>
<td>101143</td>
<td>323</td>
<td>66602</td>
<td>4811809</td>
<td>0</td>
</tr>
</tbody>
</table>

seems low...

seems high...

...but what is normal? Can it be better?
Framework Overview

- **Training:**
  - Using synthetic kernels, collect training data on target system
  - Cover as much as the performance space as (reasonably) possible
  - Construct classifiers based on training data

- **Testing**
  - Collect performance data from real application
  - Classify application code regions (i.e. computation bursts)
  - For significant code regions, interpret classifications and provide tuning suggestions based on expert system rules, transformations

- Previous efforts require simulation environment for measuring cache reuse distance – “Quantifying Locality in The Memory Access Patterns of HPC Applications”, Weinberg et al., SC 2005

- Mapping of application behavior to known benchmarks for prediction on new hardware – “Performance Projection of HPC Applications Using SPEC CFP2006 Benchmarks”, Sharkawi et al., IPDPS 2009
Prototype Tool Environment

• Training:
  • PAPI http://icl.cs.utk.edu/papi/

• Collection & Testing:
  • Extrae 2.0 (MPITrace) http://www.bsc.es/paraver
  • PAPI
  • Clustering – “Automatic Detection of Parallel Applications Computation Phases”, González et al., IPDPS 2009
  • PerfExplorer / Weka
  • Paraver http://www.bsc.es/paraver
Synthetic Kernel Example: Apex-Map

- “Apex-MAP: A Global Data Access Benchmark to Analyze HPC Systems and Parallel Programming Paradigms” - Strohmaier and Shan, SC'05
- Synthetic benchmark to explore temporal, spatial locality extents
- Support for single node performance and distributed memory performance (using MPI, UPC, or SHMEM)
- We are using 10 temporal locality parameters, 10 spatial locality parameters, 7 data sizes, yielding 700 training instances per target platform
- Collect PAPI hardware counters for each (multiple runs for counter groups)

**Height = cycles per access**
Apex-Map algorithm (simplified)

- Temporal locality parameter alpha (1.0 to 0.0, 1.0 = random access, 0.0 = the same address)
- Spatial locality parameter L (1 to 64k)
- Memory block size M

```c
// warmup and measured time
for (i = 0; i < times + extra; i++) {
    // create the array of indexes
    initIndexArray(length);
    // iterate over indexes
    for (j = 0; j < length; j++) {
        // read L bytes and compute
        for (L = 0; L < maxL; L++) {
            W0 = W0 + c0 * (data0[ind[j] + L]);
        }
    }
}
```

```
// initIndexArray:
for (j = 0; j < length; j++)
    ind[j] = (pow(drand48(), 1/K) * (M/L - 1)) * L
```
Classification Strategies – Marenostrum

- Classify temporal locality
  - TOT_INS/TOT_CYC, L1_DCM/L1_DCA, L3_DCM/L1_DCA, and TLB_DM/L1_DCA ratios

- Classify spatial locality
  - L1_DCM/L1_DCA, L3_DCM/L1_DCA, and TLB_DM/L1_DCA ratios

- Ratios provide “normalization”

- Other counters (appear to) have no correlation with changes in spatial/temporal parameters*

*subject to change for native counters
Benchmark Patients

• Flux (image temporal filter kernel) – USAF / UM
  • 4 matrices allocated as 1D vectors with size width*height
  • Iterate over pixels and add filter value to each one
  • Original version – poor performance on PPC970MP
  • Flux blocked – size of inner loops sized to fit in the cache
  • Flux prefetched – unfuse inner loop to improve cache prefetch behavior

• 2D Matrix multiply
  • outMatrix[i][j] += inMatrix1[i][k] * inMatrix2[k][j];
  • naïve method – outer to inner: i, j, k
  • inner loop reordering – outer to inner: i, k, j
Classification Results of Synthetic Patients

Spatial locality improvement yields bigger IPC improvement

Surface: Apex-Map
Points: different data sizes
Memory Hierarchy Behavior – NOT direct map!
Architecture differences

- **Marenostrum (PPC970MP 2.3GHz):**
  - Blocking method is faster than default
  - Prefetch method is faster than blocking
  - For *this* machine, prefetch method is best

- **Jaguar (AMD Opteron 2435 (Istanbul) 2.6GHz):**
  - For *this* machine, blocking is faster than prefetch!
  - Improvement in temporal locality has greater effect on IPC than change in spatial locality
  - Analysis of AMD cache behavior just starting, so no full understanding of this yet
PEPC test

• “x-large” case run on 8k processors of Jaguar
  • Only 128 were merged by front-end mpi2prv
• TOT_CYC and TOT_INS collected on all processes, other counters sampled across time
• Computation bursts are clustered by IPC and TOT_INS values
• All other counters extrapolated from cluster results – “Performance Data Extrapolation in Parallel Codes”, González (not yet submitted)
PEPC Trace in Paraver: x-large on Jaguar

1 iteration: 11.5 seconds

Cluster ID

Functions

Useful IPC

IPC: 0.05 to 2.48
Green = low IPC, blue = high IPC
PEPC Trace in Paraver: x-large on Jaguar

DBSCAN (Eps=0.02, MinPoints=4)
Trace 'pepc-8k-burst-split.chop1-clustersNEW.prv'

- NOISE
- Cluster 1
- Cluster 2
- Cluster 3
- Cluster 4
- Cluster 5
- Cluster 6
- Cluster 7
- Cluster 8
- Cluster 9
- Cluster 10
- Cluster 11
- Cluster 12
- Cluster 13
- Cluster 14
- Cluster 15
- Cluster 16

Graph showing performance metrics with clusters and markers for different data points.
Cluster 1 has most time, lowest CPI – but not because of memory accesses

Cluster 1 = end of PEPC_FIELDS, Cluster 2 = begin of TREE_DOMAIN, Cluster 3 = PEPC_FIELDS in between TREE_WALK calls, Cluster 4 = TREE_ALLOCATE & TREE_BUILD, Cluster 5 = First phase of TREE_WALK, Cluster 6 = TREE_FILL, Cluster 7 = TREE_PROPERTIES, Cluster 8 = TREE_PROPERTIES in between TREE_UTIL::SORT_I calls
Scale (a.k.a Bernd's Questions)

- Preparations:
  - Synthetic benchmark collection (~5m45s on XT5)
  - no instrumentation necessary*
- Program startup: static link or LD_PRELOAD
- Actual measurement: bursts mode, on-line mode
- Finalization: 8k files, 128 per directory, 1.6GB
- Post-processing:
  - Trace merging & clustering
    - 128 traces of 8192 process run merged (12s) and clustered (~1m20s):
      15MB trace from 1.6GB of collected data
  - Analysis can be automated (?)
    - Many manual steps at the moment, some supervision

* ...except when it is.
Problems with Classifications

- Which counters represent performance space?
  - Target dependent
  - Inter-related parameters
- Counter “normalization” method(s)
- Weights of “normalized” counters
  - Classifiers work with absolute values
- Which classifier(s) to use?
- Problems with over-training
- Native counters or PAPI preset?
- Multicore influence on behavior
Other Synthetic Kernels

- “OMI4papps: Optimisation, Modeling and Implementation for Highly Parallel Applications” - Weinberg, et al., Leibniz Supercomputer Centre Report
  - Modified Apex-MAP: adds compute-intensive loop for driving up FLOP/cycle to system max, and stride behavior parameter
- MetBench: computer architecture stress benchmarks (BSC)
  - Characterization of POWER5/POWER6 hardware priorities between SMT threads
    - http://www.bsc.es/caos
- Concurrency benchmarks – MPI and OpenMP
- GPGPU?
  - SHOC

Project Link: http://www.konwihr.uni-erlangen.de/projekte/laufende-projekte/omi4papps-de.shtml
Future Work / Considerations

- Further investigate current & previous work for inspiration/discouragement
- Find synthetic benchmarks for computation intensity, concurrency, specialized hardware
- Determine answers to questions on previous slide
- Possibly reduce class space
  - Good, normal, bad
- Collect / develop diagnosis rules
  - Could require in-depth understanding of counter behavior for each platform
  - Integrate existing rules, i.e. APART
- Code transformations – Active Harmony
  http://www.dyninst.org/harmony/
Thanks

Comments / questions?

For more information:

kevin.huck@bsc.es
juan.gonzalez@bsc.es

http://www.bsc.es/paraver