Fine-grained Benchmark Subsetting for System Selection

Pablo de Oliveira Castro, Y. Kashnikov, C. Akel, M. Popov, W. Jalby

University of Versailles – Exascale Computing Research

CGO February 2014
Motivation

- Find system with the best performance on a set of applications?
- Reduce the cost of benchmarking
Key Idea

- Applications have redundancies
  - Similar code called multiple times
  - Similar code used in different applications
- Detect redundancies and keep only one representative
Previous Approaches

Remove similar applications

Joshi, Phansalkar, Eeckhout

Remove similar instruction blocks

Simpoint: Sherwood, Perelman, Calder
What can be improved?

Application subsetting
- Coarse grained: less similarity, less accuracy

Instruction block subsetting
- Not portable, requires a simulator
- Cannot evaluate compilers
Source Code Subsetting

- Subset fine-grained source code fragments
  - Fine grained
  - Can be recompiled and executed on multiple architectures
- Codelets
Our Approach

**Step A:** Detect codelets

**Step B:** Build profile on a reference system

Applications

- SP
- BT

Codelet Finder (CAPS)

Static & Dynamic Profiling

- Maqao
- Likwid

Vectorization ratio

- FLOPS
- Cache Misses
- ...

f1, f2, f3

SP codelets

BT codelets

SP

BT
Our Approach

Step C: Cluster similar codelets

Step D: Extract representative set

Step E: Benchmark representatives
Breaking the Application into Codelets

- **Codelet**: source code fragment
  - Functions: too big, mixes different computation patterns
  - Innerloops: too small, hard to warmup and to measure
  - Outerloops (sweetspot)

- Capture most of the performance in HPC applications

![Graph showing the percentage of execution time captured by codelets for NAS Serial Benchmarks.](image-url)
Profiling and Clustering

- Automatically group similar codelets
  - Profile codelets on a reference system
  - Memory/Cache bandwidth, Instruction mix, Vectorization, ...
- Cluster codelets using feature distance
- We expect that:
  - Clusters capture similar computation patterns
  - Clusters react similarly to architecture change
Clustering NR Codelets

<table>
<thead>
<tr>
<th>Codelet</th>
<th>Computation Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>toeplz_1</td>
<td>DP: 2 simultaneous reductions</td>
</tr>
<tr>
<td>rstrct_29</td>
<td>DP: MG Laplacian fine to coarse mesh transition</td>
</tr>
<tr>
<td>mprove_8</td>
<td>MP: Dense Matrix x vector product</td>
</tr>
<tr>
<td>toeplz_4</td>
<td>DP: Vector multiply in asc./desc. order</td>
</tr>
<tr>
<td>realft_4</td>
<td>DP: FFT butterfly computation</td>
</tr>
<tr>
<td>toeplz_3</td>
<td>DP: 3 simultaneous reductions</td>
</tr>
<tr>
<td>svbkbs_3</td>
<td>SP: Dense Matrix x vector product</td>
</tr>
<tr>
<td>lop_13</td>
<td>DP: Laplacian finite difference constant coefficients</td>
</tr>
<tr>
<td>toeplz_2</td>
<td>DP: Vector multiply element wise in asc./desc. order</td>
</tr>
<tr>
<td>four1_2</td>
<td>MP: First step FFT</td>
</tr>
<tr>
<td>tridag_2</td>
<td>DP: First order recurrence</td>
</tr>
<tr>
<td>tridag_1</td>
<td>DP: First order recurrence</td>
</tr>
<tr>
<td>ludcmp_4</td>
<td>SP: Dot product over lower half square matrix</td>
</tr>
<tr>
<td>hqr_15</td>
<td>SP: Addition on the diagonal elements of a matrix</td>
</tr>
<tr>
<td>relax2_26</td>
<td>DP: Red Black Sweeps Laplacian operator</td>
</tr>
<tr>
<td>svdcmp_14</td>
<td>DP: Vector divide element wise</td>
</tr>
<tr>
<td>svdcmp_13</td>
<td>DP: Norm + Vector divide</td>
</tr>
<tr>
<td>hqr_13</td>
<td>DP: Sum of the absolute values of a matrix column</td>
</tr>
<tr>
<td>hqr_12_sq</td>
<td>SP: Sum of a square matrix</td>
</tr>
<tr>
<td>jacobi_5</td>
<td>SP: Sum of the upper half of a square matrix</td>
</tr>
<tr>
<td>hqr_12</td>
<td>SP: Sum of the lower half of a square matrix</td>
</tr>
<tr>
<td>svdcmp_11</td>
<td>DP: Multiplying a matrix row by a scalar</td>
</tr>
<tr>
<td>elmhes_11</td>
<td>DP: Linear combination of matrix rows</td>
</tr>
<tr>
<td>mprove_9</td>
<td>DP: Substracting a vector with a vector</td>
</tr>
<tr>
<td>matadd_16</td>
<td>DP: Sum of two square matrices element wise</td>
</tr>
<tr>
<td>svdcmp_6</td>
<td>DP: Sum of the absolute values of a matrix row</td>
</tr>
<tr>
<td>elmhes_10</td>
<td>DP: Linear combination of matrix columns</td>
</tr>
<tr>
<td>balanc_3</td>
<td>DP: Vector multiply element wise</td>
</tr>
</tbody>
</table>
Clustering NR Codelets

<table>
<thead>
<tr>
<th>Codelet</th>
<th>Computation Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>toeplz_1</td>
<td></td>
</tr>
<tr>
<td>rstrct_29</td>
<td></td>
</tr>
<tr>
<td>mprove_8</td>
<td></td>
</tr>
<tr>
<td>toeplz_4</td>
<td></td>
</tr>
<tr>
<td>realf_4</td>
<td></td>
</tr>
<tr>
<td>toeplz_3</td>
<td></td>
</tr>
<tr>
<td>svbkbsb_3</td>
<td></td>
</tr>
<tr>
<td>lop_13</td>
<td></td>
</tr>
<tr>
<td>toeplz_2</td>
<td></td>
</tr>
<tr>
<td>four1_2</td>
<td></td>
</tr>
<tr>
<td>tridag_2</td>
<td></td>
</tr>
<tr>
<td>tridag_1</td>
<td></td>
</tr>
<tr>
<td>ludcmp_4</td>
<td></td>
</tr>
<tr>
<td>hqr_15</td>
<td></td>
</tr>
<tr>
<td>relax2_26</td>
<td></td>
</tr>
<tr>
<td>svdcmp_14</td>
<td>DP: Vector divide element wise</td>
</tr>
<tr>
<td>svdcmp_13</td>
<td>DP: Norm + Vector divide</td>
</tr>
<tr>
<td>hqr_13</td>
<td>DP: Sum of the absolute values of a matrix column</td>
</tr>
<tr>
<td>hqr_12_sq</td>
<td>SP: Sum of a square matrix</td>
</tr>
<tr>
<td>jacobi_5</td>
<td>SP: Sum of the upper half of a square matrix</td>
</tr>
<tr>
<td>hqr_12</td>
<td>SP: Sum of the lower half of a square matrix</td>
</tr>
<tr>
<td>svdcmp_11</td>
<td></td>
</tr>
<tr>
<td>elmhes_11</td>
<td></td>
</tr>
<tr>
<td>mprove_9</td>
<td></td>
</tr>
<tr>
<td>matadd_16</td>
<td></td>
</tr>
<tr>
<td>svdcmp_6</td>
<td></td>
</tr>
<tr>
<td>elmhes_10</td>
<td></td>
</tr>
<tr>
<td>balanc_3</td>
<td></td>
</tr>
</tbody>
</table>

Similar computation patterns

Long Latency Operations

Reduction Sums
Capturing Architecture Change

**Nehalem (Ref)**
Freq: 1.86 GHz
LLC: 12 MB

- LU/erhs.f: 49
- FT/appft.f: 45

**Cluster A**: triple-nested high latency operations (div and exp)
- BT/rhs.f: 266
- SP/rhs.f: 275

**Core 2**
→ 2.93 GHz
→ 3 MB

- Faster operations:
  - Lower values indicate faster performance
- Slower operations:
  - Higher values indicate slower performance

**Atom**
→ 1.66 GHz
→ 1 MB

- Slower operations:
  - Higher values indicate slower performance

---

Reference Target

---

+ Reference
× Target
Same Cluster = Same Speedup

**Nehalem (Ref)**
Freq: 1.86 GHz
LLC: 12 MB

- LU/erhs.f : 49
- FT/appft.f : 45

**Cluster A:** triple-nested high latency operations (div and exp)

- BT/rhs.f : 266
- SP/rhs.f : 275

**Core 2**
→ 2.93 GHz
→ 3 MB

**Atom**
→ 1.66 GHz
→ 1 MB

Cluster B: stencil on five planes (memory bound)
Representative Selection

Choose central codelet as representative

- Prediction model: Codelets from the same cluster have the same speedup when changing architectures
Representative Extraction: Codelet Finder

- Extract representatives as standalone microbenchmarks
- Can be recompiled and run outside of the original application

C or Fortran Application

```
for (i = 0; i < N ; i ++) {
    for (j = 0; j < N ; j ++) {
        a[i][j] += b[i]*c[j];
    }
}
```

Core Dump

Extracted Codelet

```
for (i = 0; i < N ; i ++) {
    for (j = 0; j < N ; j ++) {
        a[i][j] += b[i]*c[j];
    }
}
```

Is Source-code Isolation Viable for Performance Characterization? *PSTI’13*
Validation

- Trained and selected feature set on Numerical Recipes + Atom + Sandy Bridge
- Validated approach on NAS Serial and a new architecture, Core 2

<table>
<thead>
<tr>
<th>Reference</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>Atom</td>
</tr>
<tr>
<td>CPU</td>
<td>L5609</td>
</tr>
<tr>
<td>Frequency (GHz)</td>
<td>1.86</td>
</tr>
<tr>
<td>Cores</td>
<td>4</td>
</tr>
<tr>
<td>L1 cache (KB)</td>
<td>4×64</td>
</tr>
<tr>
<td>L2 cache (KB)</td>
<td>4×256</td>
</tr>
<tr>
<td>L3 cache (MB)</td>
<td>12</td>
</tr>
<tr>
<td>Ram (GB)</td>
<td>8</td>
</tr>
</tbody>
</table>

Table: Test architectures.
NAS results

- 18 representatives
- 23 times faster benchmark
- 5.8% median error
Tradeoff Reduction / Accuracy (NAS)

More clusters:
- \(\uparrow\) accuracy
- \(\uparrow\) benchmarking cost

Automatically select good tradeoff using Elbow method
Overall results (NAS)

- Accurately evaluate architectures
- Choose the best architecture-benchmark pairs
Conclusion

- Take advantage of source loops redundancies to reduce benchmarking time
  - Generate portable compressed benchmarks
  - Accurate (< 10%) and Faster (> ×23)

- Applications
  - System Selection (this)
  - Fast compiler performance regression tests
  - Iterative Compilation

  - data and analysis code available as a reproducible IPython notebook
Thanks for your attention!
Feature Selection

- Genetic Algorithm: find best set of features on Numerical Recipes + Atom + Sandy Bridge
- The feature set is still among the best on NAS

![Graph showing comparison of feature sets across different scenarios]
Reduction Factor Breakdown

<table>
<thead>
<tr>
<th>Reduction</th>
<th>Total</th>
<th>Reduced invocations</th>
<th>Clustering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atom</td>
<td>44.3</td>
<td>×12</td>
<td>×3.7</td>
</tr>
<tr>
<td>Core 2</td>
<td>24.7</td>
<td>×8.7</td>
<td>×2.8</td>
</tr>
<tr>
<td>Sandy Bridge</td>
<td>22.5</td>
<td>×6.3</td>
<td>×3.6</td>
</tr>
</tbody>
</table>

**Table**: Benchmarking reduction factor breakdown with 18 representatives.
NAS: regular codes.
- Only 19% of codelets have different behavior across invocations.
- Detect *ill-behaved codelets*. Exclude them from representatives.

SPEC: different working set per invocation.
- Ongoing: Cluster codelets across working sets
Across Applications Similarities

Subsetting – Across Applications – Per Application

Atom

Core 2

Sandy Bridge

Number of clusters

Median % error

Subsetting

Across Applications

Per Application

Number of clusters
Profiling Features

Performance counters per codelet

4 dynamic features

FLOPS
L2 Bandwidth
L3 Miss Rate
Mem Bandwidth

Likwid

4 dynamic features

Bytes Stored / cycle
Stalls
Estimated IPC
Number of DIV
Number of SD
Pressure in P1
Ratio ADD+SUB/MUL
Vectorization (FP/FP+INT/INT)

Maqao

8 static features

Static dissassembly and analysis

SP codelets

8 static features

Static dissassembly and analysis