Energy Efficiency Analysis of Compiler Optimizations on the SPEC CPU 2017 Benchmark Suite

Norbert Schmitt, James Bucek, Klaus-Dieter Lange and Samuel Kounev

April 24, 2020

SPEC, the SPEC logo and the names SPECrate, SPECspeed, SPEC CPU, SPECpower, and PTDaemon are registered trademarks of the Standard Performance Evaluation Corporation, reprint with permission.

11th ACM/SPEC International Conference on Performance Engineering (ICPE), April 22-24

https://se.informatik.uni-wuerzburg.de/
Motivation

- More services are placed in the cloud
- Leading to an increasing amount and larger data centers
- Data centers need an estimated 140 billion kWh annually by 2020¹

¹ J. Whitney and P. Delforge, „Data center efficiency assessment,“ NRDC, August 2014.
Motivation

- Cloud data centers can be made more efficient
  - Intelligently placing or consolidating services
  - Minimize resources through auto-scaling while satisfying performance demand
- Hardware can be made more efficient
  - Dynamic voltage and frequency scaling
  - Different C-States
- Software controls the hardware
- Running software has an influence on the energy efficiency of the complete system
- Different, but functionally identical software can have a different energy efficiency while the performance does not change

---


Contribution

- A first look at which factors make the software susceptible to compiler optimizations
  - Programming Language
  - Application domain
- Based on the SPEC CPU 2017 benchmark suite
SPEC CPU 2017

- SPEC CPU 2017 benchmark suite is compute-intensive
  - Different code and problem sizes
    - C, C++, and Fortran, covering multiple programming paradigms
    - 1000 to 1.5 million lines of code
  - Stressing CPU, memory, and compiler
- Defined run and reporting rules for good repeatability
- 43 benchmarks organized in four suites
  - SPECspeed Integer and Floating-Point: Time required to process one unit of work
  - SPECrate Integer and Floating-Point: Work per unit of time (Throughput)
- Each of the four suites produces two metrics
  - Base: Each programming language, or combination, must use identical compiler settings
  - Peak: Each benchmark can use different compiler settings
Evaluation

- Relative comparison of base and peak values of the SPECrate Integer suite
- Benchmarks with identical compiler settings for base and peak runs are excluded
- Optimizing for performance can increase energy efficiency
- Example 500.perlbench_r
  - 15% reduction in runtime
  - 5.5% better energy efficiency
Evaluation

- 23 benchmarks in SPECrate Integer and Floating-Point suites in total:
  - 7 excluded due to identical compiler settings
  - 16 benchmarks listed
  - 3 benchmarks implemented and counting towards two languages
- Can C-like languages be better optimized for energy efficiency?

<table>
<thead>
<tr>
<th>Language</th>
<th>EE improved</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>8</td>
<td>8</td>
<td>100%</td>
</tr>
<tr>
<td>C++</td>
<td>6</td>
<td>7</td>
<td>85.7%</td>
</tr>
<tr>
<td>Fortran</td>
<td>1</td>
<td>4</td>
<td>25%</td>
</tr>
</tbody>
</table>
Evaluation

- $H_0$: C-like and functional languages are equally likely to show better energy efficiency
- $H_0$ must be rejected at the 5% level
- $H_0$ can not be rejected at the 1% level
- Possible reasons
  1. Compiler allows fewer optimizations for Fortran programs
  2. Functional programming provides an already energy-efficient programming style
  3. Results are outliers

<table>
<thead>
<tr>
<th>Language</th>
<th>EE improved</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>8</td>
<td>8</td>
<td>100%</td>
</tr>
<tr>
<td>C++</td>
<td>6</td>
<td>7</td>
<td>85.7%</td>
</tr>
<tr>
<td>Fortran</td>
<td>1</td>
<td>4</td>
<td>25%</td>
</tr>
</tbody>
</table>

Fisher's exact test contingency table

<table>
<thead>
<tr>
<th>Language</th>
<th>Energy Efficiency Improved</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>C-like</td>
<td>14</td>
</tr>
<tr>
<td>Functional</td>
<td>1</td>
</tr>
<tr>
<td>Sum</td>
<td>15</td>
</tr>
</tbody>
</table>

Motivation ➔ Contribution ➔ SPEC CPU 2017 ➔ Evaluation ➔ Conclusion
Evaluation

- 23 benchmarks in SPECrate Integer and Floating-Point suites in total:
  - 7 excluded due to identical compiler settings
  - 16 benchmarks listed
- Benchmarks were grouped into four application domains

<table>
<thead>
<tr>
<th>App. Domain</th>
<th>EE improved</th>
<th>Total</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language Transformation</td>
<td>2</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>Modelling and Simulation</td>
<td>3</td>
<td>7</td>
<td>42.8%</td>
</tr>
<tr>
<td>Artificial Intelligence</td>
<td>1</td>
<td>1</td>
<td>100%</td>
</tr>
<tr>
<td>Others</td>
<td>6</td>
<td>6</td>
<td>100%</td>
</tr>
</tbody>
</table>

Motivation ➔ Contribution ➔ SPEC CPU 2017 ➔ Evaluation ➔ Conclusion
Data centers consume large amounts of energy

Use SPEC CPU 2017 benchmark suite to

- Check if the compiler settings influence the benchmarks in terms of energy efficiency
- See if the programming language is responsible for the improvement
- See if the application domain is responsible for the improvement

Comparison of programming languages show promising results that C-like languages can be easier optimized

Application domain show nondistinctive results

Further measurements on a broader set of software are necessary
Thank You!

https://se.informatik.uni-wuerzburg.de/
norbert.schmitt@uni-wuerzburg.de