Initial Evaluation of Multiple RISC ISAs using the Embench™ Benchmark Suite

What is the Cost of Simplicity?

David Patterson (UC Berkeley), Jeremy Bennett (Embecosm), Palmer Dabbelt (SiFive), Cesare Garlati (Hex Five Security), and Ofer Shinaar (Western Digital)

Tuesday, December 10, 2019
State of Benchmarks for IoT/Embedded Computers

- Billions of Internet of Things (IoT) devices shipped soon
- Still no high quality, widely reported benchmark for embedded computers

Yunsup Lee, SiFive CTO, Keynote address “Opportunities and Challenges of Building Silicon in the Cloud” 12/5/18 RISC-V Summit:

“... the benchmark scores are 4.9 CoreMarks/MHz and 2.5 DMIPS/MHz. I’m saying this in front of Dave [Patterson], who doesn’t really like Dhrystone or CoreMark as benchmarks. Sorry. This is the industry standard benchmark I learned.”

It’s past time to apologize
Embench

- Group from Academia and Industry Developing
  - We want your help to evolve Embench: info@embench.org
- Free
- Easy to Port
- Suite of ≈20 Real Programs (vs 1 Synthetic Program)
- Geometric Mean & Geometric Standard Deviation of Ratios to Reference Platform (PULP RI5CY core)
- Also Report Code Size, Context Switch Time, and Interrupt Latency
  - Necessary for embedded IoT devices yet novel part of formal benchmark
- Sustaining Organization involving Academia and Industry to Evolve over Time
  - FOSSi Foundation: The Free and Open Source Silicon Foundation
- Follows Agile Benchmark Development Philosophy: Versions 0.5, 0.6, …
- Given current state of widely reported benchmarks for embedded computing, we believe Embench—even the 0.5 version—will be a big help to the IoT field
<table>
<thead>
<tr>
<th>Name</th>
<th>Comments</th>
<th>Original Source</th>
<th>C LOC</th>
<th>code size</th>
<th>data size</th>
<th>time (ms)</th>
<th>branch</th>
<th>memory</th>
<th>compute</th>
</tr>
</thead>
<tbody>
<tr>
<td>aha-mont64</td>
<td>Montgomery multiplication</td>
<td>AHA</td>
<td>162</td>
<td>1,052</td>
<td>0</td>
<td>4,000</td>
<td>low</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>crc32</td>
<td>CRC error checking 32b</td>
<td>MiBench</td>
<td>101</td>
<td>230</td>
<td>1,024</td>
<td>4,013</td>
<td>high</td>
<td>med</td>
<td>low</td>
</tr>
<tr>
<td>cubic</td>
<td>Cubic root solver</td>
<td>MiBench</td>
<td>125</td>
<td>2,472</td>
<td>0</td>
<td>4,140</td>
<td>low</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>edn</td>
<td>More general filter</td>
<td>WCET</td>
<td>285</td>
<td>1,452</td>
<td>1,600</td>
<td>3,984</td>
<td>low</td>
<td>high</td>
<td>med</td>
</tr>
<tr>
<td>huffbench</td>
<td>Compress/Decompress</td>
<td>Scott Ladd</td>
<td>309</td>
<td>1,628</td>
<td>1,004</td>
<td>4,109</td>
<td>med</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>matmult-int</td>
<td>Integer matrix multiply</td>
<td>WCET</td>
<td>175</td>
<td>420</td>
<td>1,600</td>
<td>4,020</td>
<td>med</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>minver</td>
<td>Matrix inversion</td>
<td>WCET</td>
<td>187</td>
<td>1,076</td>
<td>144</td>
<td>4,036</td>
<td>high</td>
<td>low</td>
<td>med</td>
</tr>
<tr>
<td>nbody</td>
<td>Satellite N body, large data</td>
<td>CLBG</td>
<td>172</td>
<td>708</td>
<td>640</td>
<td>3,774</td>
<td>med</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>nettle-aes</td>
<td>Encrypt/decrypt</td>
<td>Nettle</td>
<td>1,018</td>
<td>2,880</td>
<td>10,566</td>
<td>3,988</td>
<td>med</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>nettle-sha256</td>
<td>Cryptographic hash</td>
<td>Nettle</td>
<td>349</td>
<td>5,564</td>
<td>536</td>
<td>4,000</td>
<td>low</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>nsichneu</td>
<td>Large - Petri net</td>
<td>WCET</td>
<td>2,676</td>
<td>15,042</td>
<td>0</td>
<td>4,001</td>
<td>med</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>picojpeg</td>
<td>JPEG</td>
<td>MiBench2</td>
<td>2,182</td>
<td>8,036</td>
<td>1,196</td>
<td>3,748</td>
<td>med</td>
<td>med</td>
<td>high</td>
</tr>
<tr>
<td>qrduino</td>
<td>QR codes</td>
<td>Github</td>
<td>936</td>
<td>6,074</td>
<td>1,540</td>
<td>4,210</td>
<td>low</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>sglib-combined</td>
<td>Simple Generic Library for C</td>
<td>SGLIB</td>
<td>1,844</td>
<td>2,324</td>
<td>800</td>
<td>4,028</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>slre</td>
<td>Regex</td>
<td>SLRE</td>
<td>506</td>
<td>2,428</td>
<td>126</td>
<td>3,994</td>
<td>high</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>st</td>
<td>Statistics</td>
<td>WCET</td>
<td>117</td>
<td>880</td>
<td>0</td>
<td>4,151</td>
<td>med</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>statemate</td>
<td>State machine (car window)</td>
<td>C-LAB</td>
<td>1,301</td>
<td>3,692</td>
<td>64</td>
<td>4,000</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
<tr>
<td>ud</td>
<td>LUD composition Int</td>
<td>WCET</td>
<td>95</td>
<td>702</td>
<td>0</td>
<td>4,002</td>
<td>med</td>
<td>low</td>
<td>high</td>
</tr>
<tr>
<td>wikisort</td>
<td>Merge sort</td>
<td>Github</td>
<td>866</td>
<td>4,214</td>
<td>3236</td>
<td>4,226</td>
<td>med</td>
<td>med</td>
<td>med</td>
</tr>
<tr>
<td>Directory</td>
<td>Description</td>
<td>Date</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>baseline-data</td>
<td>Py build (#9)</td>
<td>3 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>config</td>
<td>Py build (#9)</td>
<td>3 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>doc</td>
<td>Note that Embench is a trademark (#28)</td>
<td>12 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pylib</td>
<td>Ensure we use at least Python 3.6. (#25)</td>
<td>26 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>src</td>
<td>Use __int128 for 64 x 64 -&gt; 128 bit multiplication if available (#19)</td>
<td>15 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>support</td>
<td>Fix several errors in the places where floating point is used.</td>
<td>27 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.gitignore</td>
<td>Py build (#9)</td>
<td>3 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUTHORS</td>
<td>Initial commit of the new repository.</td>
<td>6 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COPYING</td>
<td>Initial commit of the new repository.</td>
<td>6 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ChangeLog</td>
<td>Remove initialization of new empty dictionary. (#13)</td>
<td>27 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSTALL</td>
<td>Update documentation and convert to Markdown (#27)</td>
<td>15 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEWS</td>
<td>Clean up a couple of annoyances.</td>
<td>6 months ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>README.md</td>
<td>Note that Embench is a trademark (#28)</td>
<td>12 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>benchmark_size.py</td>
<td>Ensure we use at least Python 3.6. (#25)</td>
<td>26 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>benchmark_speed.py</td>
<td>Ensure we use at least Python 3.6. (#25)</td>
<td>26 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>build_all.py</td>
<td>Ensure we use at least Python 3.6. (#25)</td>
<td>26 days ago</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What affects code size for embedded apps?

- **Instruction Set Architecture**
  - ARM vs ARC vs RISC-V vs AVR vs ...
  - Which extensions included:
    - ARM: v7, Thumb2, ...
    - RISC-V: RV32, RV64, M (Multiple/divide), C (compress), ...

- **Compiler**
  - Open (GCC, LLVM) and Proprietary (Embacosm, IAR, …)
  - Which optimizations included: Loop unrolling, inlining procedures, minimize code size, …
  - How fast are compilers improving?
  - Older ISAs likely have more mature and better compilers?

- **Libraries**
  - Open (GCC, LLVM) and Proprietary (IAR, Sega, …)

- **Embench excludes libraries as they can swamp code size for embedded benchmark**
Impact of optimizations of GCC on RISC-V: Speed

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Higher is faster)

![Graph showing the impact of optimizations on RISC-V speed]

- O0
- O1
- O2
- O3
- Os
- O3
- Unrolling, inlining
- Some inlining
- -msave
- -mreturn

- Optimization: Speed

GEOMEAN vs. Optimization: Speed

- 0.48
- 0.93
- 1.01
- 1.10
- 1.10
- 1.20
- 0.91
- 0.90
Impact of optimizations: Speed with Geo Std Dev

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Higher is faster)
Impact of optimizations of GCC on RISC-V: Code Size

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Lower is smaller)
Impact of optimizations: Code Size with Geo Std Dev

- RI5CY RV32IMC GCC 10.0.0 18-Nov-19 (Lower is smaller)
Instruction Set Observations

- `msave-mrestore` invokes functions to save and restore registers at procedure entry and exit instead of inline code of stores and loads
  - ISA Alternative would be Store Multiple instruction and Load Multiple instruction
- Reduces code size another 7%
- But also reduces performance 10%
Benchmarking Lessons?

1. Must show code size with performance so as to get meaningful results
2. Importance of Geo Standard Deviation as well as Geo Mean

E.g., O3 with loop unrolling and procedure inlining worthwhile?
1X to 1.5X (1.2X Geo Mean) faster programs but
2X to 5X (3X Geo Mean) bigger programs
Impact of ISA of GCC on RISC-V: Speed

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave-mrestore
- RI5CY RV32I GCC 10.0.0 (Higher is faster)
Impact of optimizations: Speed with Geo Std Dev

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave -mrestore
- RI5CY RV32I GCC 10.0.0 18-Nov-19 (Higher is faster)
Impact of ISA of GCC on RISC-V: Code Size

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave-mrestore
- RI5CY RV32 GCC 10.0.0 (Lower is smaller)
Impact of optimizations: Code Size with Geo Std Dev

- Add M (Multiply/Divide), C (Compress) -O2 vs -Os -msave -mrestore
- RI5CY RV32I GCC 10.0.0 18-Nov-19 (Lower is smaller)
Benchmarking/RISC-V Lessons?

1. Multiply/Divide (RV32IM) improves performance 1.5-1.7X over integer baseline (RV32I) and reduces code size 3% to 6%.
2. Compress has no impact on performance, reduces code size 1.4-1.5X.
3. -msave-mrestore reduce performance 10%, code size 1.25X over GCC -O2.
   ○ As opposed to 10% code size over GCC -Os.
4. Integer only has widely varying performance (Geo Std Dev 2.5)
Code Size over GCC Versions (ARM M4, RV32IMC)

GCC -Os (-msave-mrestore) dates and versions

<table>
<thead>
<tr>
<th>Date</th>
<th>ARM Cortex M4</th>
<th>RISCY RV32IMC</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/3/16 (4.9.4)</td>
<td>0.92</td>
<td></td>
</tr>
<tr>
<td>10/10/17 (5.5)</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>10/26/18 (6.5)</td>
<td>0.91</td>
<td></td>
</tr>
<tr>
<td>12/6/18 (7.5)</td>
<td>0.91</td>
<td>1.02</td>
</tr>
<tr>
<td>2/22/19 (8.3)</td>
<td>0.91</td>
<td>1.01</td>
</tr>
<tr>
<td>8/12/19 (9.2)</td>
<td>0.91</td>
<td>1.01</td>
</tr>
<tr>
<td>11/18/19 (10.0)</td>
<td>0.91</td>
<td>0.98</td>
</tr>
</tbody>
</table>
Code Size by Architecture: GCC 10.0.0 -O2 & -Os

- Atmel AVR ATmega64 (8-bit RISC microcontroller)
  - Licensed core AVR ISA from Atmel (acquired by Microchip Technology in 2016)
- ARC EM (32-bit RISC style ISA) not including compressed instructions
  - Licensed core from Synopsys
- RI5CY RV32IMC (32-bit RISC-V with compressed instructions)
  - Popular open source core from ETH Zurich: low power targeting high energy efficiencies
- ARM Cortex M4 (32-bit ARM with Thumb2 compressed instructions)
  - Popular licensed core from ARM
Code Size by Architecture: GCC 10.0.0 -O2 & -Os

**Code Size GCC -O2 (10.0.0) Lower is smaller code**

- AVR: 5.0
- ARC: 4.0
- RI5CY: 3.0
- ARM: 2.0

**Code Size GCC -Os (10.0.0) Lower is smaller code**

- AVR: 2.0
- ARC: 1.0
- RI5CY: 0.0
- ARM: 0.0

(-msave-mrestore for RI5CY)
RISC-V E option (for embedded)

- Only 16 registers for tiniest processors: 0.4X perf, 1.7-2.0X larger program
Lots more to explore with Embench

- More compilers: LLVM, IAR, …
  - And more optimizations
- More architectures: MIPS, Tensilica, ARMv8, RV64I, …
  - And more instruction extensions: bit manipulation, vector, floating point, …
- More processors: ARM M7, M33, M24, RISC-V Rocket, BOOM, …
- Context switch times
- In later versions of Embench: Interrupt Latency, Floating Point programs
- Collect and publish results on Embench.org web site
- Want to help? Email info@embench.org
Related Talks by Embench Members next 2 sessions

- **GCC Compiler: Code Size Density**
  - Nidal Faour and Ofer Shinaar, Western Digital
  - 1:20pm - 1:40pm today, Grand Ballroom 220-C (We’re in 220-A now)
  - Using small test cases derived from real scenarios when comparing the RISCV to other Cores

- **Open Source Compiler Tool Chains for RISC-V:**
  - **Past, Present and Future**
  - Jeremy Bennett, Embecosm
  - 1:50 pm - 2:10pm today, Grand Ballroom 220-C (We’re in 220-A now)
  - Using Embench to explore more features and compilers (e.g., LLVM, RISC-V Bit extension)
Conclusions

● Code size and performance should be linked for embedded benchmarks
  ○ Loop unrolling and procedure inlining can triple code size
● RISC-V M extension improves performance 1.5-1.7X and code size 3%-6%
● Using GCC and Embench, RV32IMC code much smaller than AVR
● ARM Thumb2 smaller than RV32IMC, but within one standard deviation
● In past year RISC-V GCC getting better at code size, ARM GCC stable/mature
● For GCC 10.0 compiler, adding Load Multiple/Store Multiple might add 10% performance with same code size over -msave-mrestore optimizations at cost of more complex ISA implementation
● We believe Embench 0.5 suite is already an improvement over single synthetic programs Dhrystone and CoreMark
● Let us know if you’d like to help: Email info@embench.org