Software “PPA” Metrics
Results from Real-World Security Applications
Joe Circello
Fellow, Chief MCU Core Platform & Security Architect
NXP Semiconductors, N.V.
Software Power, Performance, Area (PPA) Metrics

• Tradeoffs between traditional hardware PPA metrics are well-known in SoC design
• In this work, we “redefine” PPA into a software context for microcontroller applications
  – $P =$ relative power in terms of memory (code + data) access rates
  – $P =$ relative performance in terms of execution cycles and dynamic instruction path
  – $A =$ relative static code size
• This work will further limit its focus here to be a study of relative static code sizes
  – For constrained MCU compute environments, static code size is a key parameter
    ▪ Non-volatile code memory (flash, ROM) are typically relatively large components and represent a significant amount of dynamic power dissipation
• Applications studied involve “black-box” security subsystems with processing elements
  – Two production security software releases: i.MX & “CryptoLib” for MCU devices
    ▪ Original implementations use Arm Cortex-Mx cores as processing elements
      ▪ Retarget to the RISC-V ISA for this study with \(-\text{\texttt{march = RV32IMC}}\)
      ▪ Explore various compiler optimization levels, e.g., -Os, -O2, -O3

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
More details on the Software Packages

• Software package #1 from i.MX
  – ~100K lines of source C code
  – Arm Cortex-M4 (v7M) GCC vs. RISC-V GCC
  – For both packages:
    ▪ Measuring code sizes of .text + .rodata sections
    ▪ Starting point is the Arm GCC -Os image
  – Compiler optimization levels: -Os, -O2, -O3 plus -msave-restore for RISC-V
  – arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors 6-2017-q2-update) 6.3.1 20170620 (release) [ARM/embedded-6-branch revision 249437]
  – riscv32-unknown-elf-gcc (GCC) 7.1.1 20170509 (march=RV32IMC)

• Software package #2 from CryptoLibrary
  – ~40K lines of source C code
  – Arm Cortex-M4 (v7M) GCC vs. RISC-V GCC and armcc CM4 (v7M) vs. RISC-V GCC
  – arm-none-eabi-gcc (GNU Tools for ARM Embedded Processors) 4.8.4 20140725 (release) [ARM/embedded-4.8-branch revision 213147]
  – ARM Compiler 5.06 update 5 (build 528)
  – riscv32-unknown-elf-gcc (GCC) 7.1.1 20170509 (march=RV32IMC)

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
Static Code Size Data from Software Package #1

![SW Package #1 Static Code Size Data](image)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-Os</td>
<td>89856</td>
<td>103048</td>
<td>1.15</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>-Os/msave-restore</td>
<td>99420</td>
<td>1.11</td>
<td>1.00</td>
<td>0.96</td>
<td></td>
</tr>
<tr>
<td>-O2</td>
<td>99052</td>
<td>110682</td>
<td>1.12</td>
<td>1.10</td>
<td>1.07</td>
</tr>
<tr>
<td>-O2/msave-restore</td>
<td>107092</td>
<td>1.08</td>
<td>1.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-O3</td>
<td>114844</td>
<td>131860</td>
<td>1.15</td>
<td>1.28</td>
<td>1.28</td>
</tr>
<tr>
<td>-O3/msave-restore</td>
<td>127860</td>
<td>1.11</td>
<td>1.24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RV32IMC GCC vs. CM4 GCC Code Size Comparison

Relative code size calculated for each compiler option

*Smaller relative code size is better*

Relative code size range [1.08 – 1.15] at the upper end of expectations

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
RV32IMC GCC vs. {CM4 GCC, CM4 Armcc} Comparisons
Relative code size calculated for each compiler option
Armcc options include -Ospace and –execute-only flags
Smaller relative code size is better

Relative code size range [1.29 – 1.52] considerably beyond the upper end of expectations
More Observations from SW #2 and Next Steps

• Attributes of software package #2
  – Highly-modular, relatively-short functions
  – Indirect memory pointer references within data structures
  – APIs with large numbers of parameters
  – Relatively-high data memory access rate per instruction
  – Code written using “secure coding” guidelines
  – Side-by-side review of compiler outputs show similar instruction sequences
    ▪ Inherent differences in instruction lengths seem to account for most of size
  – Belief that these factors contribute to (much) larger-than-expected RV32IMC size
• Next steps...
  – Continue static code analysis in other “production code” application areas
  – Generate runtime analysis metrics for Power and Performance
    ▪ Memory access rates per instruction (instruction fetches, data loads & stores)
    ▪ Stack memory footprint
• Acknowledgements: Antoine Dambre (SW #1) and Pim Vullers (SW #2)

#RISCVSUMMIT | tmt.knect365.com/risc-v-summit/
SECURE CONNECTIONS FOR A SMARTER WORLD