Improving LLVM-Generated Code Size for X86 Processors

David Kreitzer
Zia Ansari
Intel Corporation
Introduction

- Code Size is often ignored
- *But it is still important!*
Introduction

Intel® Quark™ Microcontroller D2000

- 32-bit Pentium® instruction set
- No native floating-point support
Introduction

How does LLVM code size look for Intel® Quark™ D2000 Microcontroller?
Methodology

- Team: Zia Ansari, David Kreitzer, Michael Kuperstein, Andrey Turetskiy, Anton Nadolski
- What metric to use?
- Gap analysis
- Evaluation and implementation
Individual Findings

- For each finding
  - Explain the opportunity with an example
  - Give the impact on C/C++ SPEC 2000
  - Report the status of the fix in LLVM trunk
Fixed Inefficient Immediate Encodings

- Use 1-byte imm encoding of “imm op mem” instructions
- 0.22% code size reduction
- Committed as rL241152 on Jul 1, 2015 (http://reviews.llvm.org/D10766)

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>BINARY ENCODING</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmpl $0x1, global</td>
<td>83 3d 00 00 00 00 00 01</td>
</tr>
<tr>
<td>cmpl $0x1, global</td>
<td>81 3d 00 00 00 00 00 00 00 01</td>
</tr>
</tbody>
</table>
Fixed Poor Floating-Point Compares under -msoft-float

\[
\text{\texttt{\%cmp = fcmp ule double \%a, \%b = !fcmp ogt double \%a, \%b}}
\]

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{_ledf2(a, b) &lt;= 0</td>
<td></td>
</tr>
</tbody>
</table>

- Affects all targets with soft float emulation
- 0.79% code size reduction
- Committed rL242280 on Jul 15, 2015 ([http://reviews.llvm.org/D10804](http://reviews.llvm.org/D10804))
Avoid Promoting to 32-bit CMP

- rL195496 promotes compares to avoid partial reg writes
- 0.78% code size reduction from reverting rL195496

<table>
<thead>
<tr>
<th>BEFORE (12 BYTES)</th>
<th>AFTER (8 BYTES)</th>
<th>BETTER (9 BYTES)</th>
</tr>
</thead>
<tbody>
<tr>
<td>movsbl 8(%esp), %eaxmovsbl 4(%esp), %ecxcmpl %eax, %ecx</td>
<td>movb 4(%esp), %alcmpb 8(%esp), %al</td>
<td>movzbl 4(%esp), %eaxcmpb 8(%esp), %al</td>
</tr>
</tbody>
</table>

- We are working on a more comprehensive fix for the partial register write issue: rL260572 on Feb 11, 2016 (http://reviews.llvm.org/D17032)
- We hope to eventually revert r195496
Tuning Loop Rotation

C code: for (i = 0; i < n; ++i) { ... }

Basic lowering:

```
i = 0;
while (true) {
    if (i >= n) break;
    ...
    ++i;
}
```

Loop trip count = n + 1

- 0.75% code size reduction from -rotation-max-header-size=2
- We plan to tune this specifically for Quark

After loop rotation:

```
i = 0;
if (i < n) {
    while (true) {
        ...
        ++i;
        if (i >= n) break;
    }
}
```

Loop trip count = n

Additional condition - header duplication. More complicated condition - bigger code size increase.
## X86 Call Frame Optimization

### Instruction Binary Encoding

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl $3, 12(%esp)</td>
<td>c7 44 24 0c 03 00 00 00 (8)</td>
</tr>
<tr>
<td>movl $2, 8(%esp)</td>
<td>c7 44 24 08 02 00 00 00 (8)</td>
</tr>
<tr>
<td>movl $1, 4(%esp)</td>
<td>c7 44 24 04 01 00 00 00 (8)</td>
</tr>
<tr>
<td>movl $.str, (%esp)</td>
<td>c7 04 24 00 00 00 00 00 (7)</td>
</tr>
<tr>
<td>call _printf</td>
<td>e8 00 00 00 00 (5)</td>
</tr>
</tbody>
</table>

- PUSH encodes smaller than MOV, especially PUSH imm8
- 4.2% code size reduction, larger for non-MCU targets

### Committed as rL223757, rL227752, rL244729 on Aug 12, 2015

### Stack Layout Optimization

- Order stack objects to maximize use of 1-byte offsets
- Large objects can hog space
- 1.2% code size reduction
- Committed as rL260917 on Feb 15, 2016 ([http://reviews.llvm.org/D15393](http://reviews.llvm.org/D15393))

#### X86 STACK FRAME

<table>
<thead>
<tr>
<th>Return Address</th>
<th>Local symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>%esp+128→</td>
<td>Local symbols</td>
</tr>
<tr>
<td>%esp+124→</td>
<td>Local symbols</td>
</tr>
<tr>
<td>%esp→</td>
<td>Local symbols</td>
</tr>
</tbody>
</table>

#### INSTRUCTION

<table>
<thead>
<tr>
<th>INSTRUCTION</th>
<th>BINARY ENCODING</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl (%esp), %eax</td>
<td>8b 04 24</td>
</tr>
<tr>
<td>movl 124(%esp), %eax</td>
<td>8b 84 24 7c</td>
</tr>
<tr>
<td>movl 128(%esp), %eax</td>
<td>8b 84 24 80 00 00 00</td>
</tr>
</tbody>
</table>
**Fixed Excessive Duplication of Immediates**

- Merge common immediates within a basic block
- 1.5% code size reduction
- Committed rL244601 on Aug 11, 2015 ([http://reviews.llvm.org/D11363](http://reviews.llvm.org/D11363))

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Binary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>movl $0x3, mem1</td>
<td>c7 05 00 00 00 00 00 03 00 00 00 (10)</td>
</tr>
<tr>
<td>movl $0x3, mem2</td>
<td>c7 05 00 00 00 00 00 03 00 00 00 (10)</td>
</tr>
<tr>
<td>movl $0x3, %eax</td>
<td>b8 03 00 00 00 (5)</td>
</tr>
<tr>
<td>movl %eax, mem1</td>
<td>a3 00 00 00 00 (5)</td>
</tr>
<tr>
<td>movl %eax, mem2</td>
<td>a3 00 00 00 00 (5)</td>
</tr>
</tbody>
</table>
Miscellaneous Optimizations

- Inlining Heuristics
  - Threshold adjustments / soft-float heuristics / ODR
  - Still tuning / Not upstreamed yet

- LEA Optimizations
  - Reuse LEAs in subsequent memrefs and LEAs
    Committed rL254712 on Dec 4, 2015 (http://reviews.llvm.org/D13294)
    Committed rL257589 on Jan 13, 2016 (http://reviews.llvm.org/D13295)

- ~ 1% Code size reduction
"In terms of Chromium code size, this reduced the size of one of our main .dlls by 447 kB, which is a significant chunk!" – HansW
Performance Results

• How is Performance affected under -Os?
  • MCU Benchmarks : No regressions.
  • Big Core Benchmarks : No significant regressions (> 1%)
  • “…time is improved by a whopping 1.6%! … a very respectable improvement for general-purpose backend tuning!” – SeanS
  • … following up on 1 O3 flto test-suite performance regression

Large Code Size Reduction with Minimal Performance Downside
Other Findings and Future Work

- Tune and upstream inlining and loop rotation improvements
- Move -Os optimizations to -O2 where appropriate
- Improve -Os for non-Microcontroller targets
- Many other findings, e.g.
  - Merge identical (or nearly identical) functions
  - Tune frame pointer elimination
  - Optimize memcpy/memset for size
Questions?
Backup
Experimental Data Collection

- Experimental option to enable/disable each optimization
- Public patch based on recent LLVM trunk: (http://reviews.llvm.org/D18098)
- Details are in the patch summary
Tuning Function Inlining

- // Threshold to use when optsize is specified
  const int OptSizeThreshold = 75; ← change to 15

- Relax inlining penalty given to soft-float instructions
  - Loads and Stores shouldn’t require library calls

- Give bonus to hasLinkOnceODRLinkage with single use
  - Definition can be removed in object . . . Hope for no more uses

- 1.1% code size reduction

- Currently being tuned before upstreaming
LEA Optimization

<table>
<thead>
<tr>
<th>BEFORE</th>
<th>AFTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>leal arr+4(%edi,%edi,2), %eax</td>
<td>leal arr+4(%edi,%edi,2), %eax</td>
</tr>
<tr>
<td>movl arr(%edi,%edi,2), %edx</td>
<td>movl -4(%eax), %edx</td>
</tr>
<tr>
<td>leal arr+8(%edi,%edi,2), %ecx</td>
<td>movl $111, (%eax)</td>
</tr>
<tr>
<td>movl $111, (%eax)</td>
<td>movl $222, 4(%eax)</td>
</tr>
</tbody>
</table>

- Removes redundant LEAs (-Oz only)
  - 0.09% code size reduction
  - Committed rL257589 on Jan 13, 2016 ([http://reviews.llvm.org/D13295](http://reviews.llvm.org/D13295))

- Removes redundant address recalculations
  - 0.15% code size reduction
  - Committed rL254712 on Dec 4, 2015 ([http://reviews.llvm.org/D13294](http://reviews.llvm.org/D13294))