IR-Level vs. Machine-Level If-Conversion for Predicated Architectures

Alexander Jordan, Nikolai Kim, Andreas Krall
{ajordan,kim,andi}@complang.tuwien.ac.at

Institute of Computer Languages
University of Technology Vienna

ODES-10: Optimizations for DSP and Embedded Systems

1This work is supported by the Austrian Science Fund (FWF) under contract P21842, Optimal Code Generation for Explicitly Parallel Processors.
Outline

Implementation

- Two iterative If-Conversion variants
- Different levels of code abstraction
- Tail duplication on the machine level
- Implemented within LLVM 2.9
- Targeting TI’s TMS320C64X DSP

Evaluation

- Multi-Issue-Multi-Cluster compiler setup
- Profile information vs. profile estimation
- Based on a cycle accurate simulator
- MiBench, DSPStone, Olden, BenchmarkGames tests
If-Conversion: Introduction

### Basics
- Requires hardware support
- Removes conditional branches
- Naturally enlarges scheduling scope
- Possible application on different levels
- Increases register pressure

### Idea
- Turn control dependences into data dependences
- Favor frequent, penalize less frequent regions
- Eliminate conditional branches, merge basic blocks
Target architecture

Texas Instruments TMS320C64X
- Clustered VLIW architecture, 2 clusters
- 4 functional units, 32 GP registers per cluster
- 3 predicate registers per cluster
- SIMD subset, full predication support
void foo (int z, int val) {
    int x = val;
    if (z > 10) x = x + 1;
    return x * x;
}
Iterative If-Conversion

Process bottom-up, repeat until a threshold is reached.

Repeat steps

- Identify basic convertible patterns
- Fold predicates and convert identified patterns
Code abstraction: machine level (ML)

Features

- Straightforward implementation
- Exact target specific information (instructions, latencies)
- Precedes register allocation, manipulates virtual registers
- Utilizes extra passes to generate precise profile information
- Integrates a tail duplication mechanism
Code abstraction: intermediate representation (IR)

Features
- Early SSA-transformation, implementation more involved
- Target information not present, thus modeled inexactly
- Instrumentation and profile loading provided by LLVM
- Can be combined with LLVM-IR optimizations

Alexander Jordan, Nikolai Kim, Andreas Krall (TU Wien)
## Conversion profitability

### Estimation details
- Statically adjustable for different optimization levels
- Conservative register pressure estimation
- Geared toward branch elimination, no knowledge about scheduling

\[ C: \text{candidate region containing} \ n \ \text{basic blocks} \]
\[ \text{cycles, weight: execution time, frequency for a basic block} \]

\[
Cost(C) = \text{branch\_cost}(C) + \sum_{i=1}^{n} \text{cycles}(Block_i) \times weight_i
\]

\[
Cost_{conv}(C) = \text{cycles} \left( \bigcup_{i=1}^{n} Block_i \right) \times weight_C
\]
### Performance evaluation: statistic

**Benchmark Baseline** (kCycles)
**IR-ifconv prof. (kCycles)**
**ML-ifconv prof. (kCycles)**
**ML-ifconv estim. (kCycles)**
**IR-speedup prof. (%)**
**ML-speedup prof. (%)**
**ML-speedup estim. (%)**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Baseline (kCycles)</th>
<th>IR-ifconv prof. (kCycles)</th>
<th>ML-ifconv prof. (kCycles)</th>
<th>ML-ifconv estim. (kCycles)</th>
<th>IR-speedup prof. (%)</th>
<th>ML-speedup prof. (%)</th>
<th>ML-speedup estim. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple-bubblesort</td>
<td>1.70</td>
<td>1.38</td>
<td>1.55</td>
<td>1.55</td>
<td>18.46</td>
<td>8.37</td>
<td>8.37</td>
</tr>
<tr>
<td>Simple-quicksort</td>
<td>1.41</td>
<td>1.17</td>
<td>1.26</td>
<td>1.26</td>
<td>16.83</td>
<td>11.01</td>
<td>10.44</td>
</tr>
<tr>
<td>DSP-fir2dim</td>
<td>6.20</td>
<td>5.50</td>
<td>5.66</td>
<td>5.33</td>
<td>11.26</td>
<td>8.76</td>
<td>13.92</td>
</tr>
<tr>
<td>DSP-lms</td>
<td>1.22</td>
<td>1.16</td>
<td>0.98</td>
<td>0.98</td>
<td>5.16</td>
<td>19.98</td>
<td>19.98</td>
</tr>
<tr>
<td>DSP-fft</td>
<td>90.07</td>
<td>89.42</td>
<td>89.07</td>
<td>89.41</td>
<td>0.72</td>
<td>1.11</td>
<td>0.73</td>
</tr>
<tr>
<td>DSP-matrix1</td>
<td>32.84</td>
<td>26.19</td>
<td>24.62</td>
<td>24.62</td>
<td>20.26</td>
<td>25.02</td>
<td>25.02</td>
</tr>
<tr>
<td>DSP-matrix2</td>
<td>29.28</td>
<td>22.79</td>
<td>24.09</td>
<td>23.90</td>
<td>22.18</td>
<td>17.72</td>
<td>18.38</td>
</tr>
<tr>
<td>DSP-n_complex_updates</td>
<td>1.42</td>
<td>1.42</td>
<td>1.30</td>
<td>1.30</td>
<td>0.00</td>
<td>8.64</td>
<td>8.64</td>
</tr>
<tr>
<td>DSP-n_real_updates</td>
<td>1.05</td>
<td>0.69</td>
<td>0.92</td>
<td>0.92</td>
<td>34.09</td>
<td>12.63</td>
<td>12.63</td>
</tr>
<tr>
<td>DSP-startup</td>
<td>7.26</td>
<td>5.44</td>
<td>5.68</td>
<td>5.68</td>
<td>25.08</td>
<td>21.80</td>
<td>21.80</td>
</tr>
<tr>
<td>MB-adpcm</td>
<td>1136.39</td>
<td>526.81</td>
<td>599.57</td>
<td>599.57</td>
<td>53.64</td>
<td>47.24</td>
<td>47.24</td>
</tr>
<tr>
<td>MB-basicmath</td>
<td>1919.05</td>
<td>1757.54</td>
<td>1919.05</td>
<td>1919.05</td>
<td>8.42</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>MB-bitcount</td>
<td>1002.55</td>
<td>820.11</td>
<td>810.59</td>
<td>810.59</td>
<td>18.20</td>
<td>19.15</td>
<td>19.15</td>
</tr>
<tr>
<td>MB-blowfish</td>
<td>393.38</td>
<td>395.25</td>
<td>370.44</td>
<td>370.44</td>
<td>-0.48</td>
<td>5.83</td>
<td>5.83</td>
</tr>
<tr>
<td>MB-CRC32</td>
<td>7376.18</td>
<td>7376.18</td>
<td>5310.91</td>
<td>5310.91</td>
<td>0.00</td>
<td>28.00</td>
<td>28.00</td>
</tr>
<tr>
<td>MB-dijkstra</td>
<td>87441.65</td>
<td>71757.59</td>
<td>79975.22</td>
<td>82901.63</td>
<td>17.94</td>
<td>5.19</td>
<td>5.19</td>
</tr>
<tr>
<td>MB-FFT</td>
<td>49896.15</td>
<td>50350.01</td>
<td>49910.29</td>
<td>49910.29</td>
<td>-0.91</td>
<td>-0.03</td>
<td>-0.03</td>
</tr>
<tr>
<td>MB-gsort</td>
<td>57888.96</td>
<td>56059.27</td>
<td>57788.96</td>
<td>57788.96</td>
<td>3.16</td>
<td>0.17</td>
<td>0.17</td>
</tr>
<tr>
<td>MB-stringsearch</td>
<td>5312.74</td>
<td>5304.64</td>
<td>4568.07</td>
<td>5246.85</td>
<td>14.02</td>
<td>1.24</td>
<td>1.24</td>
</tr>
<tr>
<td>MB-susan</td>
<td>51545.54</td>
<td>45264.09</td>
<td>43007.53</td>
<td>43007.53</td>
<td>12.19</td>
<td>16.56</td>
<td>16.56</td>
</tr>
<tr>
<td>Olden-bisort</td>
<td>48526.77</td>
<td>46456.66</td>
<td>45792.90</td>
<td>45792.90</td>
<td>4.27</td>
<td>5.63</td>
<td>5.63</td>
</tr>
<tr>
<td>BG-fannkuch</td>
<td>238.99</td>
<td>234.09</td>
<td>220.21</td>
<td>214.24</td>
<td>2.05</td>
<td>7.86</td>
<td>10.35</td>
</tr>
<tr>
<td>BG-nsieve-bits</td>
<td>32078.41</td>
<td>26262.03</td>
<td>24194.72</td>
<td>24194.72</td>
<td>18.13</td>
<td>24.58</td>
<td>20.71</td>
</tr>
</tbody>
</table>

**Average**

|                | 13.82  | 14.38  | 13.85  |

*Figure: Detailed comparison statistic*
Performance evaluation: ML vs. IR

Figure: If-Conversion: ML vs. IR
Conclusions

Machine level If-Conversion...
- performs slightly better when compared to the IR-variant
- more traditional, easy to implement and to adapt
- performs equally well in absence of profile information
- exposes a more predictable optimization behavior
- is less flexible due to the machine pass pipeline

LLVM-IR If-Conversion...
- also achieves a substantial speedup
- requires more implementational effort
- preserves SSA-properties of the code
- is combinable with existing SSA-transformations
- is therefore also subject to phase ordering issues
Thank you for being my audience!