Compiling for Automatically Generated Instruction Set Extensions

Alastair Murray and Björn Franke
Institute for Computing Systems Architecture,
School of Informatics, University of Edinburgh

CGO ’12: April 2nd, 2012
Automated Processor Synthesis:

- Targeted at the embedded domain.

- Automatically generate an application specific processor.
  - Extend a baseline processor.

- Automatically find specialised extension instructions.

- Automatically generate a simulator and compiler for this processor.
PASTA Work-flow

1. Application
2. Profiling Data
3. Extension Instruction Generator
4. Extension Unit
5. Baseline Processor
6. Extended Processor
PASTA Work-flow
PASTA Work-flow
Compilation for Extension Instructions

• Current instruction set extension methodologies either:
  – Generate a standard tree-based instruction matcher, or,
  – Modify the original supplied code directly.

• Tree-based instruction matchers can not handle graph-shaped extension instructions.

• Modifying the original code prevents changes.

• Existing graph-based instruction matching techniques target small instructions.
Motivation

data = input;
coef = coefficient;
sum = 0.0;
for (i = 0; i < 8; i++) {
    # term1 = data++;
    # term2 = coef++;
    sum += term1 * term2;
}
*output = sum;

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Size of Extension Instructions

Number of Nodes in Instruction

Number of Instructions
Gcc Passes

![Diagram of Gcc Passes]

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Example

data = input;
coef = coefficient;
sum = 0.0;
for (i = 0; i < 8; i++) {
    # term1 = *data++;
    # term2 = *coef++;
    sum += term1 * term2;
}
*output = sum;

...
Example

\[
\begin{align*}
\text{term1} &= \textit{*data}++; \\
\text{term2} &= \textit{*coef}++; \\
\text{sum} &= \text{term1} \times \text{term2};
\end{align*}
\]
Example

```
# term1 = *data++;
# term2 = *coef++;
sum += term1 * term2;

... (70 more lines cut)
```

```
term1_72 = *input_10;
data_73 = input_10 + 4B;
term2_74 = *coefficient_12;
coef_75 = coefficient_12 + 4B;
D.1968_76 = term1_72 * term2_74;
sum_77 = D.1968_76 + 0.0;
term1_85 = *data_73;
data_86 = data_73 + 4B;
term2_87 = *coef_75;
coef_88 = coef_75 + 4B;
D.1968_89 = term1_85 * term2_87;
sum_90 = sum_77 + D.1968_89;
term1_98 = *data_86;
data_99 = data_86 + 4B;
... (70 more lines cut)
```
Example

term1_72 = *input_10;
data_73 = input_10 + 4B;
term2_74 = *coefficient_12;
coef_75 = coefficient_12 + 4B;
D.1968_76 = term1_72 * term2_74;
sum_77 = D.1968_76 + 0.0;
term1_85 = *data_73;
data_86 = data_73 + 4B;
term2_87 = *coef_75;
coef_88 = coef_75 + 4B;
D.1968_89 = term1_85 * term2_87;
sum_90 = sum_77 + D.1968_89;
term1_98 = *data_86;
data_99 = data_86 + 4B;
... (70 more lines cut)
Example
Example
Example

```c
map_ise_const_tmp.44.11 = 4B + 0;
__asm__ ( "vext001vr02, vr03 = vr01, vr00;"
    : "=a36" data_73, "=a37" coef_75,
      "=a38" data_86, "=a39" coef_88,
      "=a40" data_99, "=a41" coef_101,
      "=a42" data_112, "=a43" coef_114
    : "a32" input_10,
      "a33" map_ise_const_tmp.44.11,
      "a34" coefficient_12);
coef_127 = coef_114 + 4B;
coef_140 = coef_127 + 4B;
coef_153 = coef_140 + 4B;
term2.19 = *coef_153;
data_125 = data_112 + 4B;
data_138 = data_125 + 4B;
```
Evaluation Methodology

- **ISEGEN** algorithm used to generate extension instructions.

- Programs run on cycle-accurate simulator.
  - Simulator has been verified against hardware implementation.
  - Assume floating point hardware exists, for evaluation purposes

- 179 Benchmarks from:
  - EEMBC
  - UTDSP
  - DspSTONE
  - SNURT
  - Reference Cryptography implementations
Results

Speed-up from extension instructions

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
### Results (Retargeting)

<table>
<thead>
<tr>
<th>Function</th>
<th>Directly using extension instructions</th>
<th>Re-targeting extension instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>UTDSP addcm arrays</td>
<td>1.10</td>
<td>1.10</td>
</tr>
<tr>
<td>UTDSP compress arrays</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>UTDSP edge detect arrays</td>
<td>1.30</td>
<td>1.30</td>
</tr>
<tr>
<td>UTDSP ift 1024 arrays</td>
<td>1.40</td>
<td>1.40</td>
</tr>
<tr>
<td>UTDSP fir 32 1 arrays</td>
<td>1.50</td>
<td>1.50</td>
</tr>
<tr>
<td>UTDSP histogram arrays</td>
<td>1.60</td>
<td>1.60</td>
</tr>
<tr>
<td>UTDSP latnrm 32 64 arrays</td>
<td>1.70</td>
<td>1.70</td>
</tr>
<tr>
<td>UTDSP latnrm 8 1 arrays</td>
<td>1.80</td>
<td>1.80</td>
</tr>
<tr>
<td>UTDSP lmsfir 32 64 arrays</td>
<td>1.90</td>
<td>1.90</td>
</tr>
<tr>
<td>UTDSP lmsfir 8 1 arrays</td>
<td>2.00</td>
<td>2.00</td>
</tr>
<tr>
<td>UTDSP mult 10 10 arrays</td>
<td>1.10</td>
<td>1.10</td>
</tr>
<tr>
<td>UTDSP mult 4 4 arrays</td>
<td>1.20</td>
<td>1.20</td>
</tr>
<tr>
<td>UTDSP spectral arrays</td>
<td>1.30</td>
<td>1.30</td>
</tr>
</tbody>
</table>

**AVERAGE**: 1.20

**Speed-up**

- Directly using extension instructions
- Re-targeting extension instructions
Timings

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Conclusions

- A high-level pass focussing on extension instructions can:
  - Leave conventional instructions to the mature back-end.
  - Employ computationally expensive algorithms where they will help.
  - Maintain an acceptable run-time.
  - Achieve an average speed-up of 1.26 across 179 benchmarks.

- Future work:
  - Integrate the use of this compiler into instruction set extension design space exploration.
  - Re-design the slowest parts of the matching algorithms.
## Time Spent in Sub-passes

<table>
<thead>
<tr>
<th>Task</th>
<th>Time (s)</th>
<th>Time %</th>
<th>Sub-Task</th>
<th>Time (s)</th>
<th>Time %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build IRS</td>
<td>27.49</td>
<td>0.9%</td>
<td>Parse XML Instr's</td>
<td>9.33</td>
<td>0.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Build DFG</td>
<td>5.09</td>
<td>0.2%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Clean BB graphs</td>
<td>7.97</td>
<td>0.3%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Build VF2 graphs</td>
<td>5.1</td>
<td>0.2%</td>
</tr>
<tr>
<td>Mapping</td>
<td>2,904.3</td>
<td>96.1%</td>
<td>VF2 Algorithm</td>
<td>1,482.6</td>
<td>49.1%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Node Comparison</td>
<td>1,384.9</td>
<td>45.8%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Viability Checking</td>
<td>33.07</td>
<td>1.1%</td>
</tr>
<tr>
<td>Register Allocation</td>
<td>6.92</td>
<td>0.2%</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIMPLE Modification</td>
<td>83.43</td>
<td>2.8%</td>
<td>Scheduling</td>
<td>82.95</td>
<td>2.7%</td>
</tr>
</tbody>
</table>

Timings are from the summation of all 179 benchmarks.
Timings (Normalised)

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Hardware Sizes

- 12 in + 8 out ISEs
- 1.25 Cycle input cost ISEs
- 4 in + 4 out ISEs

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Match Validity

Vin 0
- rB
- rA
- rC
- rD

Vin 1
- rE
- rF
- rG
- rH

Vin 2
- rL
- rK
- rJ
- rI

Vout 0
- rM
- rN
- rO
- rP

Vout 1
- rT
- rS
- rQ
- rR
Varying Register Port Contraints
Tuning for Small Extension Instructions

![Bar chart showing speed-up for various benchmarks with different cycles]

- Speed-up: 0.00 Cycles, 0.50 Cycles, 1.00 Cycles, 1.25 Cycles, 2.1 Cycles
- Benchmarks: crypto aes, dot product fixed, dspstone dot product fixed, etc.

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Embedded Constants

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Scalar Registers

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Permuation Units

- Full register allocation
- Register allocation without permutations
- Full register allocation + 2-step permutations

Graph showing speed-up for various benchmarks:
- crypto aes
- dspstone dot product
- eembc1 automotive a2time01
- eembc1 automotive iirfilt
- eembc1 networking pkflow
- eembc1 office dither01
- eembc1 telecom viterb00
- eembc1 consumer cjpeg
- eembc1 networking pktflow
- eembc1 office dither01
- eembc2 consumer mp4decode
- eembc2 consumer mpeg2dec
- eembc2 networking tcp
- SNURT qurt
- UTDSP compress ptrs
- UTDSP mult 4 4 ptrs
- GEO-MEAN
- AVERAGE
- FULL GEO-MEAN
- FULL AVERAGE

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012
Register Allocation

No register allocation
Standard register allocation
Bad register allocation

Speed-up

A. Murray & B. Franke  Compiling for Automatically Generated Instruction Set Extensions  April 2nd, 2012