Efficiently Synthesizing Lowest Cost Rewrite Rules for Instruction Selection

Ross Daly
Create Domain Specific Accelerators by *Specializing* Processors

Image Processing CGRA

ML Training CGRA

Large Language Model CGRA??
Each Version Requires a Custom Compiler
Processing Element (PE) Design Space Exploration

- For a domain: Large space of possible PE designs
- Generate 100s of PEs
- Benchmark and compare each PE
- Requires a working compiler per PE!
Ideal: Generate Compiler

Specification of PE

Specification of IR

Generate

Application

Compiler

Lower to IR

Instruction Selection

Scheduling + Allocation

Assembly

CGRA Executable
My Work: Generate Instruction Selection
Rewrite Rules

Specification of PE

Specification of IR

Generate

Rewrite Rules

Instruction Selection

Scheduling + Allocation

Assembly

CGRA Executable
Instruction Selection
(i.e. ‘Mapping’)

• Instruction Selection
Task:
• Translate intermediate instructions (IR) to architecture-specific instructions (ISA)
• Translation must preserve equivalence
• Produce optimal ISA code
## Rewrite Rules

<table>
<thead>
<tr>
<th>IR Pattern</th>
<th>ISA Pattern</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir.add(x,y)</td>
<td>isa.add(x,y)</td>
<td>1</td>
</tr>
<tr>
<td>ir.sub(x,y)</td>
<td>isa.sub(y,x)</td>
<td>1</td>
</tr>
<tr>
<td>ir.mul(x,y)</td>
<td>isa.mul(x,y)</td>
<td>1</td>
</tr>
<tr>
<td>ir.add(x, ir.mul(y,z))</td>
<td>isa.mac(x,y,z)</td>
<td>1</td>
</tr>
</tbody>
</table>

### IR Code

\[
\begin{align*}
R0 & := \text{ir.mul}(x, y) \\
R1 & := \text{ir.add}(R0, x) \\
R2 & := \text{ir.sub}(R1, y)
\end{align*}
\]

### ISA Code

\[
\begin{align*}
R0 & := \text{isa.mac}(x, y, x) \\
R2 & := \text{isa.sub}(R0, x)
\end{align*}
\]

### Instruction Selection

The selection process involves translating IR instructions into ISA instructions, considering both performance and architectural constraints.
Rewrite Rules

<table>
<thead>
<tr>
<th>IR Pattern</th>
<th>ISA Pattern</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ir.add(x, y)</code></td>
<td><code>isa.add(x, y)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.sub(x, y)</code></td>
<td><code>isa.sub(y, x)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.mul(x, y)</code></td>
<td><code>isa.mul(x, y)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.add(x, </code></td>
<td><code>isa.mac(x, y, x)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.mul(y, z)</code>)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IR Code**

\[
\begin{align*}
R0 & := \text{ir.mul}(x, y) \\
R1 & := \text{ir.add}(R0, x) \\
R2 & := \text{ir.sub}(R1, y)
\end{align*}
\]

**ISA Code**

\[
\begin{align*}
R0 & := \text{isa.mac}(x, y, x) \\
R2 & := \text{isa.sub}(R0, x)
\end{align*}
\]

**Instruction Selection**
### Rewrite Rules

<table>
<thead>
<tr>
<th>IR Pattern</th>
<th>ISA Pattern</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>ir.add(x,y)</td>
<td>isa.add(x,y)</td>
<td>1</td>
</tr>
<tr>
<td>ir.sub(x,y)</td>
<td>isa.sub(y,x)</td>
<td>1</td>
</tr>
<tr>
<td>ir.mul(x,y)</td>
<td>isa.mul(x,y)</td>
<td>1</td>
</tr>
<tr>
<td>ir.add(x, ir.mul(y,z))</td>
<td>isa.mac(x,y,z)</td>
<td>1</td>
</tr>
</tbody>
</table>

**IR Code**

- $R0 := \text{ir.mul}(x, y)$
- $R1 := \text{ir.add}(R0, x)$
- $R2 := \text{ir.sub}(R1, y)$

**Instruction Selection**

**ISA Code**

- $R0 := \text{isa.mac}(x, y, x)$
- $R2 := \text{isa.sub}(R0, x)$
### Rewrite Rules

<table>
<thead>
<tr>
<th>IR Pattern</th>
<th>ISA Pattern</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ir.add(x, y)</code></td>
<td><code>isa.add(x, y)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.sub(x, y)</code></td>
<td><code>isa.sub(y, x)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.mul(x, y)</code></td>
<td><code>isa.mul(x, y)</code></td>
<td>1</td>
</tr>
<tr>
<td><code>ir.add(x, ir.mul(y, z))</code></td>
<td><code>isa.mac(x, y, z)</code></td>
<td>1</td>
</tr>
</tbody>
</table>

**IR Code**

\[
R0 := ir.\text{mul}(x, y) \\
R1 := ir.\text{add}(R0, x) \\
R2 := ir.\text{sub}(R1, y)
\]

**Instruction Selection**

**ISA Code**

\[
R0 := \text{isa.mac}(x, y, x) \\
R2 := \text{isa.sub}(R0, x)
\]
Large Diversity in Possible PEs

- ISA for the PE can be CISC-like compared to the IR
  - E.g. PE specialization using APEX Design Space Exploration
  - Requires "many-to-1" rewrite rules
    - Each IR pattern has multiple instructions
Large Diversity in Possible PEs

- ISA for the PE can be RISC-like compared to the IR
- Requires “1-to-many” rewrite rules
- Each ISA pattern can has multiple instructions
A rewrite rule generation tool must be capable of generating ‘many-to-many’ rewrite rules
How? Use Program Synthesis!
Program Synthesis

Program Specification

Program Synthesis

0| input x : BV[16]
1| input y : BV[16]
2| input z : BV[16]
3| r0 = ir.mul(y,z)
4| r1 = ir.add(x,r0)
5| return r1
How to Synthesize a Many-to-Many Rewrite Rule?

- Search for two patterns (IRPat, ISAPat)

- Such that:
  - IRPat is composed of 1 or more IR instructions
  - ISAPat is composed of 1 or more ISA instructions
  - IRPat is functionally equivalent to ISAPat

Many-to-Many Rewrite Rule

\[
\text{IRPat} \equiv \equiv \text{ISAPat}
\]
Prior Work on Instruction Selection Rewrite Rule Synthesis

- Buchwald et al. Synthesizing an Instruction Selection Rule Library from Semantic Specifications
  - Uses Component-Based Program Synthesis to synthesize ‘many-to-1’ rewrite rules.
    - IR pattern with one or more instruction (many)
    - ISA pattern with single instruction (1)
Generalized Component-Based Program Synthesis (GCBPS)

M IR-Instructions → Generalized CBPS → IR-Program (size M)

N ISA-Instructions → Generalized CBPS → ISA-Program (size N)

New Program Synthesis Technique!
Generalized Component-Based Program Synthesis (GCBPS)

\[ \phi_{synth} := \exists L^a, L^b. \forall I^a, I^b, O^a, O^b, W^a, W^b \]
\[ \psi_{wfp}(L^a) \land \psi_{wfp}(L^b) \land \]
\[ ((\psi_{prog}(L^a, I^a, O^a, W^a) \land \psi_{prog}(L^b, I^b, O^b, W^b)) \implies ((\land_i I_i^a = I_i^b) \implies O^a = O^b)) \]
Solved using Counter Example Guided Inductive Synthesis (CEGIS)

\[ \phi_{synth} := \exists L^a, L^b. \forall I^a, I^b, O^a, O^b, W^a, W^b \]
\[ \psi_{wf_f}(L^a) \land \psi_{wf_f}(L^b) \land \]
\[ ((\psi_{prog}(L^a, I^a, O^a, W^a) \land \psi_{prog}(L^b, I^b, O^b, W^b)) \implies \]
\[ ((\land_i I^a_i = I^b_i) \implies O^a = O^b) \]
Given a Satisfying Solution

IR-Program
(size M)

ISA-Program
(size N)
Programs are Interpreted as Patterns

IR-Program (size M) -> IR-Pattern (size M)

ISA-Program (size N) -> ISA-Pattern (size N)
Pair of Patterns Interpreted as an Instruction Selection Rewrite Rule

Many-to-Many Rewrite Rule

IR-Pattern (size M) \(\rightarrow\) ISA-Pattern (size N)
Complete: Synthesizing All Rewrite Rules
Complete

• Why? Having a complete set of rewrite rules will improve the code quality of instruction selection!

• Iterative Algorithm (inspired by Buchwald et al.):
  • Iterate over all combinations of instructions
    • Iterate over all possible number of inputs
      • Find all satisfying solutions to GCBPS
### Number of Synthesized Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>Number of instructions in ISA Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17 71 X X</td>
</tr>
<tr>
<td>2</td>
<td>89 3945 X X</td>
</tr>
</tbody>
</table>
### Number of Synthesized Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17</td>
<td>71</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>89</td>
<td>3945</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Timed Out!
Problem: Are These the Same Rule?

0| input x : BV[16]  
1| input y : BV[16]  
2| t0 = ir.sub(x,y)  
3| return t0

0| input x : BV[16]  
1| input y : BV[16]  
2| t0 = isa.neg(y)  
3| t1 = isa.add(x,t0)  
4| return t1

0| input x : BV[16]  
1| input y : BV[16]  
2| t0 = ir.sub(y,x)  
3| return t0

0| input x : BV[16]  
1| input y : BV[16]  
2| t0 = isa.neg(x)  
3| t1 = isa.add(y,t0)  
4| return t1
Yes! But x and y are Swapped

0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(x, y)
3| return t0

0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(y)
3| t1 = isa.add(x, t0)
4| return t1

0| input x : BV[16]
1| input y : BV[16]
2| t0 = ir.sub(y, x)
3| return t0

0| input x : BV[16]
1| input y : BV[16]
2| t0 = isa.neg(x)
3| t1 = isa.add(y, t0)
4| return t1
Duplicates

- Different kinds of duplicates
  - Input variable permutations
  - Commutativity of instructions
  - Same-kind instruction
  - Topological sorts of data dependency graph
Preventing Synthesis of Duplicates

- **Key idea**: Don’t use program synthesis to enumerate duplicates!

- Formally define *equivalence relations* for rewrite rules

- Updated Algorithm: Given a new rewrite rule:
  - All duplicates are programmatically enumerated and excluded from search space using that rules’ equivalence class.
## Number of Unique Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>51</td>
<td>959</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>78</td>
<td>1004</td>
<td>19278</td>
<td>X</td>
</tr>
</tbody>
</table>
# Number of Unique Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>Number of instructions in ISA Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>78</td>
</tr>
</tbody>
</table>

Does not time out!
### Number of Unique Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9</td>
<td>51</td>
<td>959</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>78</td>
<td><strong>1004</strong></td>
<td>19278</td>
<td>X</td>
</tr>
</tbody>
</table>

75% of synthesized of previous rewrite rules were duplicates! (3945 -> 1004)
Speedup by Removing Duplicates

10x Speedup!

- (Dup + Comp)
- Same-Kind Narrowing
- Commutative Narrowing
- Dup Exclusion
- (Dup + Comp) Exclusion
- All Optimizations

Cumulative SAT Time (min)

Number of Unique Rules
Problem: Different Rules for Same IR Pattern?

$\begin{align*}
0&| \text{ input } x : \text{ BV}[16] \\
1&| \text{ input } y : \text{ BV}[16] \\
2&| \text{ t0 = ir.add}(x,y) \\
3&| \text{return t0}
\end{align*}$

$\begin{align*}
0&| \text{ input } x : \text{ BV}[16] \\
1&| \text{ input } y : \text{ BV}[16] \\
2&| \text{ t0 = isa.add}(x,y) \\
3&| \text{return t1}
\end{align*}$

$\begin{align*}
0&| \text{ input } x : \text{ BV}[16] \\
1&| \text{ input } y : \text{ BV}[16] \\
2&| \text{ t0 = ir.add}(x,y) \\
3&| \text{return t0}
\end{align*}$

$\begin{align*}
0&| \text{ input } x : \text{ BV}[16] \\
1&| \text{ input } y : \text{ BV}[16] \\
2&| \text{ t0 = isa.add}(x,0) \\
3&| \text{ t1 = isa.add}(t0,y) \\
4&| \text{return t1}
\end{align*}$
High Cost Rules

- There are many valid rewrite rules for the same IR pattern
- Only one will be the lowest cost for a given cost metric
- All higher cost rules will *never* be useful in instruction selection
Preventing Synthesis of High Cost Rewrite Rules

- **Key idea**: Don’t use program synthesis to enumerate high cost rules!

- **Key Idea**: Knowing instruction selection cost metric at synthesis time allows preventing synthesis of higher cost rules

- Try all lower cost ISA instruction combinations before higher cost ones
  - Guarantees the first rule found is lowest-cost
  - Exclude IR patterns with a lowest cost rule from search space
### Number of Lowest-Cost Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in IR Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>112</td>
<td>69</td>
<td>9</td>
</tr>
</tbody>
</table>
Number of Lowest-Cost Many-to-Many Rewrite Rules

<table>
<thead>
<tr>
<th>Number of instructions in ISA Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>112</td>
<td>69</td>
<td>9</td>
</tr>
</tbody>
</table>

Does not time out!
# Number of Lowest-Cost Many-to-Many Rewrite Rules

99.6% of unique rewrite rules are high cost!! (19278 -> 69)

<table>
<thead>
<tr>
<th>Number of instructions in ISA Pattern</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of instructions in IR Pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>112</td>
<td>69</td>
<td>9</td>
</tr>
</tbody>
</table>
Speedup by Removing High Cost

Cumulative SAT Time (min)

Number of Lowest Cost Rules

92x Speedup!
### Different Cost Metrics

Synthesize Different Rules

<table>
<thead>
<tr>
<th>ISA</th>
<th>Rewrite Rule Size (IR, ISA)</th>
<th>Unique (CS)</th>
<th>Unique (E)</th>
<th>Common</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>(2, 5)</td>
<td>109</td>
<td>145</td>
<td>82</td>
</tr>
<tr>
<td>1b</td>
<td>(2, 4)</td>
<td>147</td>
<td>182</td>
<td>105</td>
</tr>
<tr>
<td>2</td>
<td>(3, 2)</td>
<td>104</td>
<td>171</td>
<td>1099</td>
</tr>
</tbody>
</table>

**CS:** Code Size  
**E:** Energy
Summary

• New program syntheses technique to synthesize many-to-many rewrite rules for Instruction Selection

• Iterative algorithm to generate complete set of many-to-many rewrite rules

• Only unique rules using equivalence relations
  • Resulting in large speedups

• Only lowest-cost rules using a cost metric
  • Resulting in further large speedups
A Necessary Step in Automatically Generating Application Compilers
Questions?
Backup