Speculative High-Level Synthesis of Instruction Set Processors

Jean-Michel Gorius
Univ Rennes

Steven Derrien, Simon Rokicki
CAIRN Team, Univ Rennes, IRISA, Inria

COLQ – M2 SIF Colloquium
June 30th, 2021
High-Level Synthesis of Instruction Set Processors

A processor with a given Instruction Set (ISA) can be implemented in many ways

Low energy

High performance

Processor Design Landscape

Micro-coded processor

Single issue pipeline

Multiple issue pipeline

Out-of-Order pipelined processor

Decode and Rename

ROB

Low energy

High performance

Takeaway

Need for customizable architecture for heterogeneous embedded applications
Customizing Instruction Set Processors


Customizing hardware
Early work for Application-Specific Instruction Set Processor design using Domain-Specific Languages
[Cloutier and Thomas, 1993; Huang and Despain, 1993]

Previous approaches still rely on low-level specifications of the hardware

Our work
Automatic synthesis of pipelined processors from a behavioral description
High-Level Synthesis

Synthesizing circuits from an algorithmic specification

Efficient design synthesis for regular access patterns and computations

```c
int X[N], Y[N];
int tmp, res = 0;
#pragma HLS mult = 1, adder = 1
for(int i = 0; i < N; ++i) {
    tmp = X[i] * Y[i];
    res += tmp;
}
```
Problem Statement

Research Question

How to make pipelined processor design amenable to HLS design flows?

```
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```
while (1) {
  ir = fetch(pc);
  op, rd, wr = decode(ir);
  if (op == BR) {
    pc = branch_target(op);
  } else {
    pc = pc + 4;
    switch(op) {
      case NEG:
        arg = regs[rd];
        regs[wr] = neg(arg);
        break;
      case ADDPC: // ...
      case LOAD: // ...
        // ...
    }
  }
}
Loop Pipelining and Static Scheduling

```plaintext
instr = fetch(pc);
if(resolve_branch(instr))
  pc = branch_target(instr);
else
  pc += 1;
```

**Simplified ISS**

```
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>instr = fetch(pc)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>instr = fetch(pc)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resolve_branch(instr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resolve_branch(instr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pc += 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pc = branch_target(instr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pc = branch_target(instr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pc = branch_target(instr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>B3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Loop Pipelining and Static Scheduling

```
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```

Simplified ISS
Simplified ISS

```plaintext
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```
Loop Pipelining and Static Scheduling

```c
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```

**Simplified ISS**

**Inefficient design**

Need to complete an entire iteration to know the next value of PC
Speculative High-Level Synthesis

```
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```

Simplified ISS

---


Speculative High-Level Synthesis

\[
\text{instr} = \text{fetch}(pc); \\
\text{if} (\text{resolve_branch(instr)}) \\
\quad \text{pc} = \text{branch_target(instr)}; \\
\text{else} \\
\quad \text{pc} += 1;
\]

Simplified ISS


Speculative High-Level Synthesis

\[
\text{instr} = \text{fetch}(\text{pc}); \\
\text{if}(\text{resolve_branch}(\text{instr})) \\
\quad \text{pc} = \text{branch_target}(\text{instr}); \\
\text{else} \\
\quad \text{pc} += 1;
\]

Simplified ISS


Speculative High-Level Synthesis

```
instr = fetch(pc);
if(resolve_branch(instr))
    pc = branch_target(instr);
else
    pc += 1;
```

**Simplified ISS**


Speculative High-Level Synthesis

```c
instr = fetch(pc);
if(resolve_branch(instr))
  pc = branch_target(instr);
else
  pc += 1;
```

**Simplified ISS**


Speculative High-Level Synthesis [Josipovic'2019, Derrien'2020]

\[
\text{instr} = \text{fetch}(\text{pc}); \\
\text{if}(\text{resolve_branch} (\text{instr})) \\
\quad \text{pc} = \text{branch_target}(\text{instr}); \\
\text{else} \\
\quad \text{pc} += 1;
\]

Simplified ISS

Takeaway

Pipelined processor design requires speculation

Speculative HLS

- Maximal ILP and resource utilization
- What to speculate on? How to speculate?


Speculative Loop Pipelining

Source-to-source transformations
- Transform the input code to expose potential speculations

Speculation support
- Decouple control logic from datapath
- HLS toolchain infers the speculative structure

Limitations
- Lack of support for intertwined speculations

Reminder: Mapping an ISS to Hardware

```
while (1) {
    ir = fetch(pc);
    op, rd, wr = decode(ir);
    if (op == BR) {
        pc = branch_target(op);
    } else {
        pc = pc + 4;
        switch(op) {
            case NEG:
                arg = regs[rd];
                regs[wr] = neg(arg);
                break;
            case ADDPC: // ...
            case LOAD: // ...
                // ...
            }
    }
}
```
Speculating on the Value of PC

```
while(1) {
    ir = fetch(pc);
    op, op0, imm = decode(ir);
    if(op == BR)
        pc = imm;
    else if(op == CONDBR)
        pc = exec(op, op0, imm);
    else
        pc = pc + 4;
}
```
Multi-Conditional Speculation

Key Idea
Speculatively select paths of increasing length
while (1) {
    // ...
    case NEG:
        if (rd == prev_wr)
            arg = ex2;
        else
            arg = regs[rd];
        regs[wr] = ex2
            = neg(arg);
        prev_wr = wr;
        break;
    // ...
}
Chained Speculation

Key Idea
Restart everything if Ca misspeculates, but restart only y if Cb misspeculates
while (1) {
    // ...
    case NEG:
        if (rd == prev_wr &&
            ex1_data_avail(rd))
            arg = ex1;
        else {
            if (rd == prev_wr)
                arg = ex2;
            else
                arg = regs[rd];
        }
    ex1 = exec1(arg);
    set_ex1_data_avail(rd);
    ex2 = exec2(ex1);
    regs[wr] = ex2;
    prev_wr = wr;
    break;
    // ...
}
Current Status

Problem Characterization

Taxonomy of speculation patterns

Implementation Status

- Transformation passes to make speculative patterns emerge
- Initial validation infrastructure for multiple speculations
Conclusion

Compiling Speculative Hardware
Represent speculation at the compiler level, manipulate hardware in the compiler

Need for New Tools
We cannot afford to continue wasting precious transistors and need fast and reliable prototyping tools

Compiling Speculative Hardware
Represent speculation at the compiler level, manipulate hardware in the compiler

Computer-Aided Design
Easily explore tradeoffs between different pipeline depths, speculation configurations
Backup
Application-Specific Instruction Set Processors

```c
if(IN.pRn == WB.IN.pRn) && (WB.IN.writeback_code == WRITEBACK_WRITERN)
{
    // bypass the value from WB
    OUT.op_2 = WB.IN.result;
}
else
{
    // read the value from the register file
    TpRn = IN.pRn;
    OUT.op_2 = R[TpRn.ExtractToLong(0,4)];
}
```

Example LISATek code [Klemm et al., 2007]


Multi-Path Speculation
Exploring Speculation Patterns
Transforming Speculation Patterns

\[ \gamma \rightarrow \gamma(y_{Fa}) \rightarrow \gamma(y_{Sa}) \rightarrow \gamma(y) \rightarrow \gamma \]

\( \Delta_1 = 5 \text{ cycles} \)

\( \Delta_2 = 10 \text{ cycles} \)

\( \gamma \)-node fusion and rebalancing