Advanced Optimization and Design Issues of a 32-bit Embedded Processor Based on Produced Order Queue Computation Model

Hiroki Hoshino, Ben A. Abderazek, Kenichi Kuroda
Adaptive System Laboratory,
School of Computer Science and Engineering, The University of Aizu, Japan

Outline

• Background
• Our proposal
• Queue computation overview
• System architecture
• Result
• Conclusion
Outline

• Background
• Our proposal
• Queue computation overview
• System architecture
• Result
• Conclusion

Background

• Instruction level parallelism (ILP)
  – The key to improve the performance
  – Allow the instructions to be executed in parallel
  – To exploit this, there are two main schemes
    * Superscalar scheme
      Complex hardware techniques are needed such as register renaming, wide issue buffer and reorder buffer
    * VLIW scheme
      Complex compiler techniques are needed
Outline

• Background
• Our proposal
• Queue computation overview
• System architecture
• Result
• Conclusion

Our proposal

• A produced order queue machine
  – The intermediate data is written into a circular queue register (QREG)
  – A given instruction implicitly reads its operands from a head of the queue register and the executed result is written into a tail of the queue register
  – Characteristics
    ➢ Grouped ILP
    ➢ Simple hardware
    ➢ Low power
    ➢ Short code size
Outline

- Background
- Our proposal
- Queue computation overview
- System architecture
- Design methodology
- Result
- Conclusion

Queue computation overview

Equation

\[ e = ab/c \]
\[ f = ab(c+d) \]

Data flow graph

Program

\[ \text{ld a} \]
\[ \text{ld b} \]
\[ \text{ld c} \]
\[ \text{ld d} \]

\[ \text{div -2} \]
\[ \text{autlq} +2 \]
\[ \text{mul -1} \]
\[ \text{st e} \]
\[ \text{mul} \]
\[ \text{st f} \]
\[ \text{stplq} \]
\[ \text{add} \]

Queue Register

\[ a \]
\[ b \]
\[ c \]
\[ d \]
\[ a*b \]
\[ c+d \]
\[ a*b/c \]
\[ ab(c+d) \]
Grouped ILP

Using breadth first algorithm

The instructions within same level are independent

Program generation from complex DAG

DAG of real part of FFT
Outline

- Background
- Our proposal
- Queue computation overview
- System architecture
- Result
- Conclusion

System architecture

- Instruction set architecture for Queue Core is fixed 16-bit
- Queue Core fetches 4 instructions per cycle
- There are 6 pipeline stages
- Queue computation unit (QCU) executes the addresses of operand or result
- Queue Core has a circular queue register
**Circular queue register structure**

- Queue register has 3 pointers
  - Queue Head (QH)
    - The position of taking data
  - Queue Tail (QT)
    - The position of being written data
  - Live Queue Head (LQH)
    - Keeping used data to reuse
    - Data between QH and LQH cannot be overwritten

---

**Dynamic address calculation for parallel execution**

- CN: consumed number
- PN: produced number

QH and QT calculation in QCU

QH0 and QT0 for next cycle
### Renew pointer mechanism

For correct execution, renew mechanism is needed:

- Branch
- Interrupt
- Exception

<table>
<thead>
<tr>
<th>cycle</th>
<th>Inst</th>
<th>QH</th>
<th>QH2</th>
<th>QT</th>
<th>LQH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>add</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>com</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>beq</td>
<td>6</td>
<td>-</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>add</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>sub</td>
<td>7</td>
<td>6</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>mul</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>st</td>
<td>11</td>
<td>-</td>
<td>-</td>
<td>11</td>
</tr>
</tbody>
</table>

Example of pointers for branch case

---

### Memory access optimization

- Offset field is only 8-bit
  - Whole memory cannot be accessed
- To extend the operand field, the Queue Core uses “covop” instruction

Example of displacement extension hardware mechanism
Outline

- Background
- Our proposal
- Queue computation overview
- System architecture
- Design methodology
- Result
- Conclusion

Code analysis result

<table>
<thead>
<tr>
<th></th>
<th>MIPS16</th>
<th>ARM</th>
<th>Thumb</th>
<th>x86</th>
<th>QC</th>
</tr>
</thead>
<tbody>
<tr>
<td>H.263</td>
<td>58.00</td>
<td>83.66</td>
<td>80.35</td>
<td>57.20</td>
<td>41.34</td>
</tr>
<tr>
<td>MPEG2</td>
<td>53.09</td>
<td>78.40</td>
<td>69.99</td>
<td>53.22</td>
<td>36.75</td>
</tr>
<tr>
<td>Susan</td>
<td>47.34</td>
<td>80.48</td>
<td>77.54</td>
<td>46.66</td>
<td>35.12</td>
</tr>
<tr>
<td>AES</td>
<td>51.27</td>
<td>86.67</td>
<td>69.59</td>
<td>44.62</td>
<td>34.93</td>
</tr>
<tr>
<td>Blowfish</td>
<td>54.59</td>
<td>86.38</td>
<td>82.76</td>
<td>57.45</td>
<td>45.49</td>
</tr>
<tr>
<td>FFT</td>
<td>58.09</td>
<td>100.74</td>
<td>92.54</td>
<td>46.27</td>
<td>36.77</td>
</tr>
<tr>
<td>Average</td>
<td>53.73</td>
<td>86.05</td>
<td>78.79</td>
<td>50.9</td>
<td>36.77</td>
</tr>
</tbody>
</table>

Normalized code sizes
Comparison result

<table>
<thead>
<tr>
<th></th>
<th>PQP</th>
<th>SH-2</th>
<th>ARM7</th>
<th>LEON2</th>
<th>Micro Blaze</th>
<th>QC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed (SPD) in MHz</td>
<td>22.5</td>
<td>15.3</td>
<td>25.2</td>
<td>27.5</td>
<td>26.7</td>
<td>25.5</td>
</tr>
<tr>
<td>Speed (ARA) in MHz</td>
<td>21.5</td>
<td>14.1</td>
<td>24.5</td>
<td>26.7</td>
<td>26.7</td>
<td>24.2</td>
</tr>
<tr>
<td>Average Power in mW</td>
<td>120</td>
<td>187.5</td>
<td>22</td>
<td>458</td>
<td>135</td>
<td>90</td>
</tr>
</tbody>
</table>

Speed and power consumption comparisons for various Synthesizable CPU cores over speed(SPD) and area(ARA) optimizations.
The target is Stratix FPGA device(EP1S25F1020). The speed is given in MHz.

Outline

- Background
- Our proposal
- Queue computation overview
- System architecture
- Design methodology
- Result
- Conclusion
Conclusion

- A detail of Queue Core architecture
- Optimization techniques
  - Memory extend
  - Offset reference
  - Queue computation unit (dynamic calculation)
- The program size for Queue Core is small
  - about 30% smaller than MIPS
  - about 50% smaller than ARM7
- The Queue Core shows better performance for both area and speed optimizations
  - about 40% higher speed than SH-2

Future work

- Queue overflow handling optimization
  - The Queue Core cannot execute correct result in limited register size
- Further optimizations in the Issue and barrier units are needed
Thank you for listening!

---

**Queue Core Features**

- **Advantage point**
  - At the compiler point
    - Natural ILP -> Grouped ILP
      - no complex compiler is needed
    - Scheduling is simple
  - At the hardware point
    - No complex hardware is needed
      - No register renaming
      - No reorder buffer
      - No wide issue buffer

- **Disadvantage point**
  - ISA is 16-bit width
  - Offset field is 8-bit
    - Need extending field
      - HW is bigger
  - Queue is not popular
    - What application is appropriate for QC