Advanced Hardware Optimization Algorithms for High Performance Queue Processor Architecture

Hiroki Hoshino s1130198

Abstract

Instruction level parallelism (ILP) is important to improve performance of general processors. ILP allows the instruction of a sequential program to be executed in parallel. However aggressive optimization of the compiler and some bigger hardware mechanisms are needed to find and exploit ILP.

In this research the queue based instruction set architecture is used. This architecture offers an attractive option in the design of embedded systems. Instructions based on queue machine are generated using level order traversal that allows us to find all available parallelism in programs. Thus the hardware executes instructions in parallel with little effort.

Some optimization and design issues for queue processor architecture (QC) have been proposed. This processor implemented the offset references, the memory extension instruction, the pipelined structure and the floating point execution unit. However the optimized QC cannot reuse data in the queue register. Also, if the queue register is full of available data, there is the critical problem that the processor cannot execute any more. This research describes the solution of problems such as the reusing data problem and the queue register overflow problem.

1 Introduction and research motivation

The architecture based on general RISC architecture extracts instruction level parallelism by using various methods. For example, a super-scalar machine has the reorder buffer, and the register renaming hardware, to execute instructions in parallel to get better performance. Also, a VLIW machine gets better performance by optimizing the program with its compiler. However, extracting instruction level parallelism from the program needs the complex hardware and the complex compiler techniques.

Now we use a queue machine. The program for the queue machine is generated using a breadth first algorithm. This algorithm means the instructions within the same level are independent and the program has high instruction parallelism without any special techniques. This feature is called natural instruction level instruction (Natural ILP).

The RISC machine uses the random access register to save the intermediate result. A given instruction for the RISC machine needs to specify the operand register number. However the queue machine has the queue register, which is called QREG, to save the intermediate result. This register is the first-in first-out (FIFO) register. The QREG has two pointers. One is the queue head (QH), which indicates the register address of reading data. The other is the queue tail (QT), which indicates the register address of writing data. These pointer values are calculated in the hardware, so a given instruction does not need to specify operands. As a result, the instruction width can be shorter and the program size can be smaller.

2 Related work to queue computing

We earlier proposed QC-2 [4], which is based on the parallel queue processor (PQP) [1–3]. This QC-2 improved with various optimization and design issues. A given instruction can read the second operand from QH added to the offset position. The offset field is only 8-bit of width. The covop instruction is added in order to extend the offset field to 16-bit. Also, QC-2 has six pipeline stages to be able to execute in high speed. Moreover, the floating point calculation unit is added to QC-2. By those optimization, QC-2 improved the performance, with 13% higher speed than PQP.

However, this QC-2 is not implemented with the structure of the reusing data mechanism. Also there is limit to the QREG size. When the QREG is full, QREG cannot store the data any more. In the next section, the solution of the above problems are described.

3 Optimization techniques

The earlier design processor, QC-2, can not reuse the data saved in QREG. Then the new pointer is added to QREG. This pointer allows the data to be able to be reused. Also new pointer control instructions are designed to execute various programs as much as possible.

And the number of QREG of QC-2 is 256. If the registers are required more than 256 to execute some program, the overflow program occurs. The solution to this
problem is proposed and hardware implementation is described in this paper.

### 3.1 Reusing data algorithm

The previous design of QREG has two pointers. One is the queue head (QH). This pointer keeps the address of the register where the data is read. Another one is the queue tail (QT), this pointer keeps the address of the register where the intermediate result is written. Each register has the ready bit, whose width is 1-bit. When the ready bit is 1, this means its register value is valid. If the ready bit is 0, its register value is invalid, so that data cannot be used. The ready bit is set to 1 when the data is written into its register and the ready bit is set as 0 when the data is read.

The data is always consumed from the register indicated by QH and the result is always generated in the register indicated by QT. The ready bit of consumed register is set at 0 and the ready bit of produced register is set to 1. Thus if the data is consumed once, this data cannot be used again because the ready bit is 0.

#### 3.1.1 Live queue head

Now the new third pointer is considered. It is called live queue head (LQH). This pointer allows the earlier consumed data to be used again. Even if the data between QH and LQH is used once, the registers ready bit is not reset to 0. So these data are still available, and can be reused. When the data needs to be reused, before this data is consumed, the queue pointer control instruction is added.

#### 3.1.2 Queue pointer control instructions (QPCI)

There are two types of QPCI. One is \texttt{stplqh} instruction. This instruction makes LQH stop in its current position. If the instructions following this \texttt{stplqh} instruction is executed, the ready bit of the read register is not reset to 0.

Another one is \texttt{autlqh} instruction. This instruction makes LQH move automatically from the current LQH added to the offset position. Also, this instruction resets the ready bit of the register between current LQH and the next LQH.

Figure 1. shows an example of LQH instruction execution. In state 2, the \texttt{stplqh} instruction is executed to stop LQH in this position because the data c is needed for the next \texttt{div} instruction. While LQH is stopping, the ready ready bit is not reset to 0. So until the \texttt{autlqh} instruction is executed, the ready bit is still 1 even if the instruction reads the register data. The data c does not disappear in state 3, and can be used in the next state 4 \texttt{div} instruction. Now that the data c is not needed, so \texttt{autlqh} instruction make LQH move automatically and the

![Diagram of QPCI](image_url)
3.2 Queue overflow handling algorithm

The number of QREG is infinite in considering the queue computation model. However, the resource is finite, so there are limits. If the QT address has the same address of LQH, the data indicated by LQH needs to be reused and cannot be overwritten. So the program cannot execute in this processor any more. This problem is called the queue overflow problem (QO). QREG is full of available data. To solve this QO problem, another queue register is organized. QC-3 can use this queue register when the QO happens. This queue register is called extended queue register (EXQREG). In addition, all programs executed in QC-3 do not need a lot of registers. When the program can be executed without using EXQREG, the power of EXQREG can turn off and QC-3 can be expected to consume low power.

3.2.1 Needed register size

There are some benchmarks such as butterfly8, LU decomposition, prefix computing and newton polynomial to research the needed register size. The max of number of QREG is shown in table 1.

According to this table, the program of Newton polynomial only needs the number of register more than 16. The other benchmark programs can be executed in 16 QREG size. So I decided the number of QREG is 16 and the number of EXQREG is 16. This configuration means that only when the Newton benchmark program
is executed, EXQREG is used.

3.2.2 Queue overflow algorithm overview

QREG and EXQREG have QH, QT and LQH, in particular, the QH, QT and LQH of EXQREG are called EXQREG.QH, EXQREG.QT and EXQREG.LQH respectively. The destination and source address are decided which pointer value is appropriate, by whether QO has happened or not. Figure 2 shows the way of solving the QO problem. There are 4 case of the queue pointer behavior as following.

In Figure 2(a), the next QT and LQH indicate the same place of register. Then the QO problem happens. At this time, QT of EXQREG, EXQREG.QT is adopted as the next QT. Also the position of QO has happened is saved into the QO_Start register.

While the QO is happening, if the next QH and QO_Start indicate the same address, QH goes to EXQREG. Figure 2(b) shows this behavior. Also if the next LQH and QO_Start indicate the same address, LQH goes to EXQREG in the same way as QH.

In case of QT of EXQREG being bigger than 16, EXQREG is over. QT returns to QREG indicated by QO_Start. Figure 2(c) shows this behavior.

Also, in case of QH of EXQREG becoming bigger than 16, EXQREG is over. QH returns to QREG indicated by QO_Start. LQH works with same way of this QH behavior. Then QO is canceled when the LQH comes back to QREG. Figure 2(d) shows this QH behavior.

3.2.3 Implementation

The queue pointers are calculated in queue computation unit (QCU). The ready bit control is calculated in execution unit (EU). Thus above algorithms are implemented into QCU and EU. The source code is designed in Verilog-HDL.

4 Evaluation result

4.1 Benchmark analysis and result

The whole QC-3 is simulated by Verilog-XL tool. The four benchmarks are designed for evaluation. The benchmarks are butterfly8, LU decomposition, prefix computing and Newton polynomial. The butterfly benchmark is the simplified First-Fourier-Transfer benchmark, which has a lot of instruction level parallelism. The benchmark of LU decomposition and prefix computing has also many parallelisms. The prefix computing and Newton polynomial benchmarks have a lot of times of reusing data. As the previous section described, only the Newton polynomial program occurs with the queue overflow. Table 2 shows the characteristics and simulation result of cycle time in each benchmark. From the table, the program size with the queue pointer control instruction (QPCI) is bigger than one without the QPCI. However these programs can not execute without the QPCI in QC-3, the reusing data mechanism can not be used without the QPCI. The average increasing rate of the program size with the QPCI is 42%.

The simulation cycle of butterfly benchmark is most same as one of Newton benchmark. However the number of line for both benchmarks is different. From this analysis, the simulation cycle does not increase with the number of lines. This result comes from the autlqh instruction execution. If the interval of the autlqh instruction insertion is short, the stall occurs a lot of time. The butterfly includes many autlqh instruction with short interval. That’s way, the simulation cycle becomes longer.

### Table 2: Benchmark analysis and simulation cycle

<table>
<thead>
<tr>
<th>Bench.</th>
<th>Lines</th>
<th>Program size (byte)</th>
<th>Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>with QPCI</td>
<td>without QPCI</td>
</tr>
<tr>
<td>Butterfly8</td>
<td>70</td>
<td>140</td>
<td>92</td>
</tr>
<tr>
<td>LU 3x3</td>
<td>63</td>
<td>126</td>
<td>80</td>
</tr>
<tr>
<td>Prefix 8</td>
<td>51</td>
<td>102</td>
<td>78</td>
</tr>
<tr>
<td>Newton 5</td>
<td>82</td>
<td>164</td>
<td>108</td>
</tr>
</tbody>
</table>

4.2 Complexity evaluation

The complexity of each module as well as the whole QC-3 are given as the number of adaptive look-up table (ALUT). The tool of QuarusII version 8.0 is used to compile the whole QC-3. The target device is EP4SGX290FH29C4, Stratix IV family, and design is optimized for balanced optimization guided by a properly implemented constraint table. Main optimization
techniques are found in the queue computation unit and the control unit. From the results shown in table 3, the area of them in QC-3 is bigger than QC-2. However the total ALUTs number of the QC-3 is smaller than QC-2. This result comes from the address width calculation. The QC-2 has a queue register, whose size is 256, and the address width is 8-bit. For the comparison with QC-2 and QC-3, the QC-2 register size is only changed to 16 while the address bit width is kept at 8-bit. On the other hand the QC-3 address bit width is 4-bit. So the number of ALUT in the QC-2 is bigger than the QC-3.

Table 3: Complexity analysis with ALUTs

<table>
<thead>
<tr>
<th>Description</th>
<th>QC-3</th>
<th>QC-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction fetch unit</td>
<td>911</td>
<td>874</td>
</tr>
<tr>
<td>Instruction decode unit</td>
<td>965</td>
<td>929</td>
</tr>
<tr>
<td>Queue computation unit</td>
<td>1386</td>
<td>1290</td>
</tr>
<tr>
<td>Barrier queue unit</td>
<td>6600</td>
<td>7674</td>
</tr>
<tr>
<td>Issue unit</td>
<td>1837</td>
<td>2367</td>
</tr>
<tr>
<td>execution unit</td>
<td>100156</td>
<td>99996</td>
</tr>
<tr>
<td>Queue-register unit</td>
<td>28706</td>
<td>30659</td>
</tr>
<tr>
<td>Memory access</td>
<td>332</td>
<td>333</td>
</tr>
<tr>
<td>Control unit</td>
<td>356</td>
<td>86</td>
</tr>
<tr>
<td>Total ALUTs number</td>
<td>141449</td>
<td>144208</td>
</tr>
</tbody>
</table>

5 Conclusion

The LQH mechanism was implemented to reuse data earlier consumed. The QPCIs are designed to control the LQH pointer and the ready bit. Also the mechanism of the QO problem solution is implemented to be execute the program needed more register than QREG. The QC-2 with the 16 QREG can not execute all of the benchmark. On the other hand, the QC-3 with the 16 QREG can execute all of the benchmark with the 16 EXQREG.

The program size increases about 42% in average. The simulation cycle becomes bigger, which is effected by QPCI insertion interval. Also the area of QCU and CU becomes bigger by this modification and the area of the whole QC-3 processor becomes smaller than the QC-2. However more programs can be executed in this QC-3 processor.

6 Future work

Queue processors are developed for use in embedded system. To get better performance, this processor was designed with pipeline technique. However this optimization make the processor area bigger. So, combining the barrier unit and the issue unit together allows the area to be smaller. In future work, combining these units together is needed to improve QC-3.

Moreover, the interrupt support is not implemented in this QC-3. So implementation of the interrupt problem will be implemented in future work.

References


