Vectorized Code Scheduling Method for the FFT Algorithm in VLIW Architecture

Te-Shin Yang and Jih-Ching Chiu

Department of Electrical Engineering
National Sun Yat-Sen University, Kaohsiung, 804, Taiwan.
Email:m9031690@student.nsysu.edu.tw

Abstract
The high degree ILP techniques are frequently used in the DSP design particularly the VLIW techniques in the present day. The paper presents a method that eliminates the vector change effect for the vectorized FFT code scheduling. This method involves two instruction groups: Circular Queue Index Register (CQI) setting instructions and Conditional Load instructions. These can achieve the goals of the optimized vectorized code scheduling for the FFT algorithm.

Keywords: vectorized code scheduling, instruction level parallelism (ILP), software pipelining, Very Long Instruction Word (VLIW), vector change

1. Instruction
Because of the increasing complexity of embedded applications, recent high performance processors have depended on Instruction Level Parallelism (ILP) to approach high execution speed. Very Long Instruction Word (VLIW) is one style of processor design that tries to achieve high level ILP by executing long instruction word, which is composed of many instructions [7]. In the DSP applications, the Fast Fourier Transform (FFT) is a high computing complexity algorithm and is usually used [5]. This paper presents a new technology, the vectorized code in a VLIW architecture [2] [3], which is used to the optimized FFT algorithm in low level code scheduling. [4]

2. The FFT algorithm with Software pipelining method
The Sande-Tukey FFT (ST-FFT) is similar to the Cooley-Tukey FFT instead of performing the summations over the frequency index [1]. The ST-FFT algorithm is often used in the FFT applications to support high degree of computational parallelism. The algorithm consists of M= log2(N) stages and bit-reversing of the output sequence. The C program of the first stage is show as Figure 1. N means the N-point FFT processing. The loop body can be extracted as Figure 2.

```
CN= N/2;
for(i=0;i<CN; i++){
    X1[i]=X[i] + X[i+CN];
    X1[i+CN]=( X[i] – X[i+CN] )*W[i];
}
```

Figure 1. FFT program of the first stage

```
X1 [ i ] = X [ i ] + X [ i + CN ]
X1 [ i + CN ] = ( X [ i ] – X [ i + CN ] ) * W [ i ]
```

Figure 2. Loop body of the FFT program

The assembler program of the Loop body can be expended as Figure 3.

```
Load r0 , VXa
Load r1 , VXb
Load r4 , VW
r2=r0+r1
r3=(r0 - r1)*r4
Store r2 , VX
Store r3 , VY
```

Figure 3. Expended program of the loop body

The Sandy-Tukey butterfly (Figure 4) presents the loop body of the ST-FFT algorithm and can be computed with eight instructions as Figure 3. The
ST-butterfly graph shows that two point input data can generate two results.

Figure 4. The Sandy-Tukey butterfly

Considering the data dependencies between instructions, the loop body can be parallelized with the software pipelining to achieve high level ILP as Figure 5. If a software pipelining step can compute a column composed of eight instructions, each stage needs \( \frac{N}{2} + 3 \) steps with the optimal software pipelining scheduling [6].

The loop body of the N-point FFT processing will need \( (\log_2 N) \times \left( \frac{N}{2} + 3 \right) \) steps with optimal software pipelining scheduling. The VLIW word format is like Figure 6, which is composed of eight instructions. One software pipelining step can be included in an instruction word.

<table>
<thead>
<tr>
<th>Load</th>
<th>Load</th>
<th>Load</th>
<th>Store</th>
<th>Store</th>
<th>ALU</th>
<th>ALU</th>
<th>Mul</th>
</tr>
</thead>
</table>

Figure 6. VLIW word format

3. Vectorized code scheduling

The FFT algorithm needs a special data fetching function to support data for the FFT butterfly algorithm processing immediately. In section 2, a VILW instruction word can complete a 2-point ST-butterfly processing. When processing the N-point FFT algorithm and N is a large number, the VLIW architecture has performance bottlenecks due to correctly and immediately computing data addresses. In the vectorized computation opinions, a regular sequential data stream which is known as a vector operand. The number of vector operands is different in each FFT computing network stage and is proportional to the stage level. Figure 7 shows the signal-flow graph of the 16-point FFT network. Assuming that all input data(X) and coefficient (W) are located in the data memory. The vector operands of the first stage are only one vector pair, a vector is x₀-x₇ and another is x₈-x₁₅. At the second stage, the four vectors are x₀-x₃, x₈-x₁₁, x₄-x₇ and x₁₂-x₁₄. At the third stage, eight vectors are x₀-x₁, x₄-x₅, x₈-x₉, x₁₂-x₁₃, x₂-x₃, x₆-x₇, x₁₀-x₁₁ and x₁₄-x₁₅. The last stage is a special case; one vector is the odd data and another vector is the even data. When executing a N-point FFT processing, the L stage will be divided into \( 2^L \) vectors and \( 2^{L-1} \) vector pairs. When a vector computing “changes” to the next vector computing, it needs to set up a new vector base addresses which are used to indicate the vector start entries. This will cause some computing overhead, called vector change effect. The “vector change effect” is proportional to the stage level of the FFT algorithm. Its relation of the computing penalty is as follows:

\[
\text{Vector change effect penalty} \propto 2^{L-1}
\]

(L is the stage level.)

If at the start of every stage it needs one step to set the initial base address, each stage will need \( \left( \frac{N}{2} + 3 + 1 \right) \) steps for execution. The loop body of the N-point FFT will need \( (\log_2 N) \times \left( \frac{N}{2} + 4 \right) \) steps.

\[ N/2+3 \text{ Steps} \Rightarrow N/2+3 \text{ Cycles, if one step needs one cycle.} \]

Figure 5. Parallelization with Software Pipelining
steps to be executed. The L stage has $2^{L-1}$ vector pairs and $2^{L-1}-1$ vector changes. It means that the L stage needs $(N/2 + 4 + 2^{L-1} - 1)$ steps to be executed and get N-point results. The loop body of the N-point FFT will need $\sum_{L=1}^{\log_2 N-1} (N/2 + 3 + 2^{L-1}) + (N/2 + 4)$ steps. The vector change effect penalty is $\sum_{L=1}^{\log_2 N-1} (2^{L-1} - 1)$ steps.

To reduce the overhead of vector change, the organization of the vectorized code is proposed to integrate all discrete vectors into single vector for the ST-butterfly. Each stage is executed at only one vector pair in these methods. Two kinds of new instruction groups will be added to the instruction set, as follows:

1. Circular Queue Index Register (CQI) setting instructions:
   
   SET CQI, CQInit

2. Conditional Load instruction:
   
   LOADRZ [Rn++k], <Rd>

### 3.1 Circular Queue Index Register setting instructions

CQI is a special circular queue register for the vectorized instruction codes. The instruction, “SET CQI, CQInit”, is used to set the initial value for CQI. The special circular queue register “CQI” will be decreased with 1 when executing the instruction “LOADRZ”. In generally, the flag RZ is equal to zero. If the content of CQI is zero, the flag RZ will be set and CQI is restored with the initial value CQInit. The value of CQInit is defined by the vector range that can be accessed contiguously. The initial value of CQI is equalized as following equation for each stage:

$$CQInit = \frac{N}{2^{\text{stage level}(L)}} \quad (1 < L < \log_2 N - 1)$$

When executing a 16-point FFT processing, the CQInit value in the first stage is 8, the second stage is 4, the third stage is 2 and the last stage is 1.

### 3.2 Conditional load instruction

The instruction “LOADRZ [Rn++k], <Rd>” is a conditional index load instruction. Its behavior is as Figure 8. If the flag RZ is not set, it will load the memory address [Rn+k] to Rd. If the flag RZ is set, it will load the memory address [Rn+CQInit].

![Figure 8. Conditional load behavior](image)

When CQInit = 4, the vector address generator will follow the steps as Figure 9.
1. Set \( R_n = \) address of data \( x_0 \) and \( CQI_{init} = 4 \)

2. Load \( x_0 \), \( CQI = 4-1 = 3 \); \( RZ = 0 \)

3. Load \( x_1 \), \( CQI = 3-1 = 2 \); \( RZ = 0 \)

4. Load \( x_2 \), \( CQI = 2-1 = 1 \); \( RZ = 0 \)

5. Load \( x_3 \), \( CQI = 1-1 = 0 \); \( RZ = 1 \)

6. Load \( x_7 \), \( CQI = CQI_{init} - 1 = 3 \), \( RZ = 0 \)

\[ \sum_{l=1}^{N-1} \left( \frac{N}{2} + 3 + 2^{l-1} \right) + \left( \frac{N}{2} + 4 \right) \] steps to be executed.

**4. Conclusion**

In this paper, we proposed a method that eliminates the vector change effect for the vectorized FFT code scheduling with adding the Circular Queue Index Register setting instructions and Conditional load instruction. In this way, we can achieve the goals of the optimized vectorized code scheduling for the FFT algorithm.

**5. Reference**


