RADIX-4 FFT IMPLEMENTATION USING SIMD MULTIMEDIA INSTRUCTIONS

Kouhei Nadehara, Takashi Miyazaki and Ichiro Kuroda

C&C Media Research Laboratories, NEC Corporation
4-1-1, Miyazaki, Miyamae-ku, Kawasaki 216-8555, Japan
{nade,miyazaki,kuroda}@ccm.CL.NEC.co.jp

ABSTRACT

In this paper, a fast radix-4 complex FFT implementation using 4-parallel SIMD instructions is presented. Four radix-4 butterflies are calculated in parallel at all stages by loading consecutive 4 elements into a register. At the last stage, every 4 elements is packed into a register and calculated in parallel. This regular data flow enables higher parallelism and an overhead reduction in data format conversion. The implementation result on the V830R processor, which has a 4-parallel SIMD-type multimedia instruction set, achieves practical performance quite competitive with high-end parallel DSPs. Multiply-accumulate instructions with symmetrical rounding introduced to the V830R processor are effective to maintain FFT accuracy.

1. INTRODUCTION

The Fourier transform is a method to convert the time domain input signals into the frequency domain. In the digital signal processing field, the discrete Fourier transform (DFT) of discrete-time signals is widely used for spectrum analysis, voice recognition, and fast computation of block filters etc. The fast Fourier transform (FFT) is employed in actual implementations because the DFT is very computationally intensive in theory [1].

As the FFT is a very multiply intensive algorithm, digital signal processors (DSPs), which integrate a hardware multiplier on a single chip, have been employed for the faster FFT. Recently, many general purpose microprocessors have introduced a hardware multiplier and associated signal processing instructions to enhance host-based signal processing capabilities such as software compression and decompression of audio and video signals [2]. Furthermore, some microprocessors have introduced "single instruction stream-multiple data streams (SIMD)" instructions for higher multimedia signal processing performance, which performs parallel calculation on packed data [3, 4, 5]. In these processors, for example, one SIMD instruction can deal with four halfword (16-bit) operands stored in 64-bit registers in parallel.

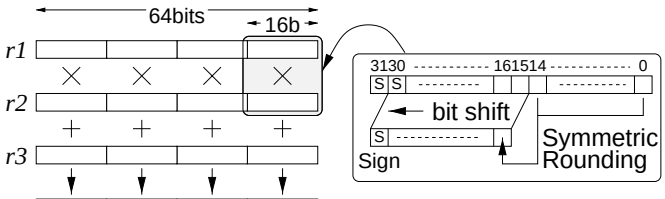
Some microprocessor vendors have released the FFT implementations on their SIMD instruction sets. Intel has shown an example of a radix-2 complex FFT implementation using their MMX instruction set in the application note [6]. It stores both real and imaginary parts in a single register. However, this irregular data structure leads to data format conversion overheads and a limitation in parallelism. Sun Microsystems provides a multimedia library including radix-2 and radix-4 FFT implementations using their VIS instruction set, but its source codes and performance evaluation has not been disclosed [7].

In this paper, a fast decimation-in-frequency radix-4 complex FFT algorithm for general purpose processors with SIMD instructions, and the implementation example on the V830R processor have been presented. In this algorithm, all butterflies can be calculated in 4-parallel by introducing a separate procedure for the last butterfly stage. In addition, it is shown that multiply-accumulate instructions with symmetric rounding greatly reduce computational errors.

2. PROCESSOR ARCHITECTURE

The V830R processor is an embedded 32-bit RISC for low-power, low-cost multimedia systems [5]. This processor is comprised of a 32-bit integer pipeline and a 64-bit multimedia coprocessor. This processor has two-way superscalar instruction issue logic. A multimedia instruction and a following integer instruction can be executed simultaneously.

The coprocessor performs SIMD-parallel operations on packed data in thirty-two 64-bit multimedia registers. For example, the *vmacr* instruction performs four 16-bit multiply-accumulate (MAC) operations simultaneously (Figure 1 (a)). Calculations are performed between operands at the same halfword positions in different registers. Therefore, the SIMD parallelism is applicable to 4 sets



(a) *"vmacr r1, r2, r3"*

(b) Multiplication with Symmetric Rounding

Figure 1: SIMD Multiply-Accumulate Instruction Example.

of data which follow exactly the same signal flow.

Unlike conventional microprocessors, the V830R processor has introduced DSP-style MAC instructions as shown in Figure 1 (b) to prevent error accumulation. A 16-bit multiplication results in a 32-bit value, but most significant two bits basically have the same sign (S) bits. Therefore, bit 31 is removed as redundant to gain accuracy by 1-bit. In addition, lower 15-bit is not truncated but rounded symmetrically to zero. When these operations cause an overflow, the result is saturated.

3. RADIX-4 FFT

The radix-4 FFT is less computationally demanding than the radix-2 FFT. The radix-4 FFT is also faster than the split-radix FFT on processors with 1-clock throughput MAC instructions such as V830R, because data address calculations in the radix-4 FFT are simpler [8].

In the radix-4 FFT, the N-point DFT is first decomposed into four N/4-point DFTs. Then, each N/4-point DFT is decomposed again into four N/16-point DFTs. These decompositions are applied recursively until each DFT size is reduced to 4. A 4-point DFT is called a "radix-4 butterfly," and is calculated directly following the signal flow shown in Figure 2. In this figure, "W" means a complex coefficient called a "twiddle factor." In later sections, a simplified notation shown in Figure 3 is used for a radix-4 butterfly to focus on input and output of the butterfly.

The radix-4 FFT reduces computational loads dramatically. In N-point FFT computational complexity increases proportional to N^2 in theory. In contrast, the N-point radix-4 FFT comprises of $log_4 N$ stages, each of which includes N/4 butterflies. Therefore, computational complexity is proportional to $Nlog_4 N$ in the radix-4 FFT.

Radix-4 FFT is highly data parallel. At each stage, N/4 butterflies can be calculated independently. However, due to restrictions of the SIMD parallelism, it should be determined carefully which butterflies to be calculated in parallel.

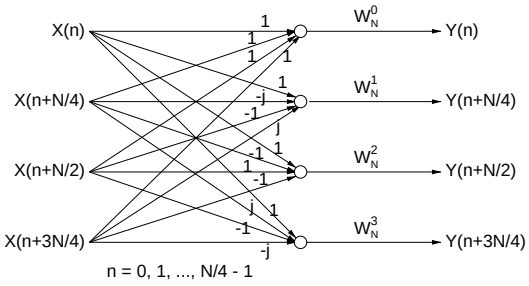


Figure 2: Radix-4 Butterfly Signal Flow.

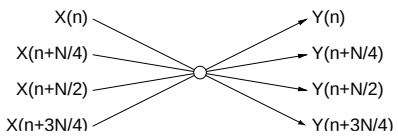


Figure 3: Simplified Radix-4 Butterfly.

4. SIMD PARALLEL IMPLEMENTATION

Parallelization of the radix-4 complex FFT is shown, taking the 64-point, 16-bit fixed-point FFT as an example. This FFT comprises of 3 stages of radix-4 butterflies.

The data structure in the proposed implementation is shown in Figure 4. The real part (Xr) and the imaginary part (Xi) of input X are stored in separate arrays to load four consecutive real or imaginary values to a 64-bit register with a single load instruction.

4.1. The First and Second Stages

The proposed fast radix-4 FFT implementation deals with consecutive 4 butterflies in parallel by loading consecutive 4 elements in a coprocessor register at stages except the last.

Figure 5 shows an example of 4-way parallelization at the 1st stage of the 64-point radix-4 FFT. In the 64-point FFT, distances between elements referred by a butterfly at the 1st stage is 16 (N/4 in Figure 2). For example, the 1st butterfly requires X[0], X[16], X[32] and X[48], and the 2nd butterfly requires X[1], X[17], X[33] and X[49], and so on. Therefore, if elements X[0∼3], X[16∼19], X[32∼35] and X[48∼51] are packed into coprocessor registers, 4 butterflies shown by the solid lines in Figure 5, which follow the same signal flow, can be dealt in parallel using SIMD instructions at the 1st iteration.

Outputs of 16 butterflies at the 1st stage connect to inputs of 16 butterflies at the 2nd stage. At the 2nd stage, distances between elements referred by a butterfly is 4. Therefore, if elements X[0∼3], X[4∼7], X[8∼11] and X[12∼15] are packed into coprocessor registers, 4 butterflies can be performed in parallel using SIMD instructions.

In the data structure shown in Figure 4, consecutive 4 real or imaginary elements necessary in a parallel calculation of 4 butterflies can be accessed with a single load instruction. In contrast, as twiddle factors may not be stored in consecutive addresses, they must be packed into a register using 4 load and format conversion instructions. Therefore, sine and cosine values for twiddle factors are stored in pairs to access a real and an imaginary part of the required twiddle factor with a single load instruction.

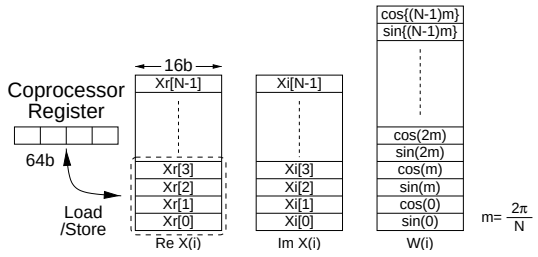


Figure 4: Data Structure for Inputs and Twiddle Factors.

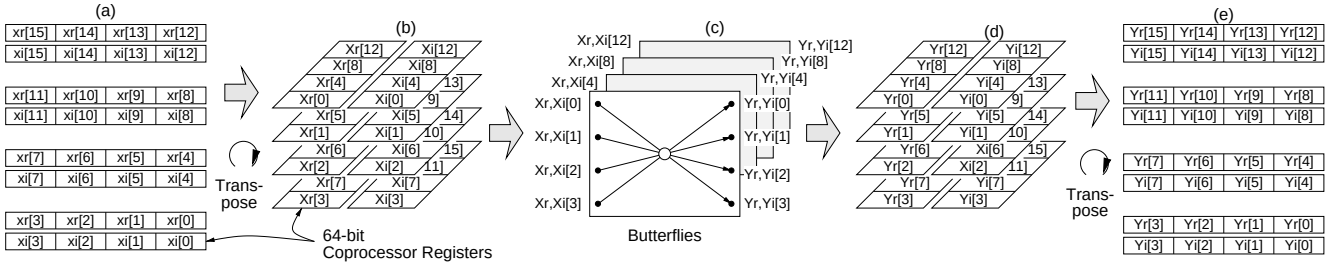


Figure 6: Parallelization of the last Stage.

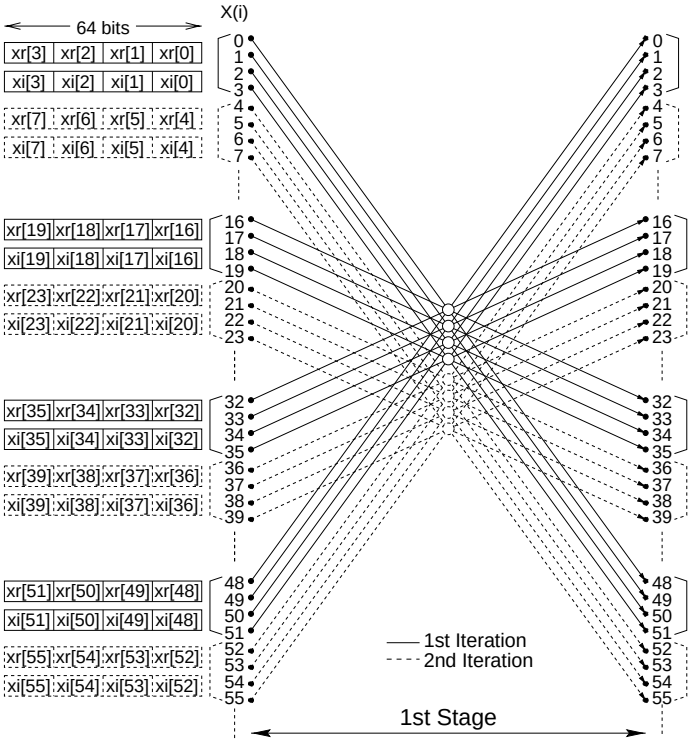


Figure 5: Parallelization of the 1st Stage.

4.2. The Last Stage

At the last stage, the parallelization method is different from previous stages, because a radix-4 butterfly should be performed between 4 consecutive elements. These 4 consecutive elements should be loaded to different registers, because the processor does not provide instructions to perform calculations between operands at different halfword positions in the same register.

As shown in Figure 6 (b), 4 real or imaginary elements should be packed into a coprocessor register from every 4 elements in the array Xr or Xi. For example, X[0], X[4], X[8] and X[12] should be stored in a single register. This format conversion from memory image (Figure 6 (a)) is equivalent to a transpose of a 4×4 matrix. In the V830R processor, 16 elements packed in 4 registers can be transposed easily with 8 format conversion instructions.

Table 1: Parallel Radix-4 FFT Performance on the V830R.

Point	Inst. (Para.)	Clocks	Time
16	223 (50%)	168	0.8 μs
64	1,219 (62%)	839	4.2 μs
256	6,141 (66%)	4,093	20.5 μs
1024	29,481 (68%)	19,257	96.3 μs

Table 2: Performance Comparison of the 256-point FFT.

Processor	MHz	Algorithm	Clocks	Time
V830R/AV	200	Radix-4	4,093	20.5 μs
PentiumII	233	Radix-2	5,522	23.7 μs
TMS320C62x	200	Radix-2	4,225	21.1 μs
TMS320C62x	200	Radix-4	2,763	13.8 μs

After the transposition, 4 butterflies are calculated in parallel with SIMD instructions. In the last stage, twiddle factors are all 1, therefore butterflies are calculated with SIMD additions and subtractions. Outputs of butterflies should be transposed again to be stored in the array Xr or Xi.

5. RESULTS

The parallel radix-4 FFT is implemented on the V830 processor in assembly language. Instructions are scheduled to exploit the two-way superscalar capability of the processor. While calculating butterflies with SIMD instructions at the 64-bit coprocessor, data addresses which will be accessed in the next iteration are calculated in advance at the free 32-bit integer pipeline.

5.1. Performance

Performance of the proposed parallel radix-4 FFT implemented on V830R is shown in Table 1. It does not include cache miss penalties, because FFT software and its working data set can fit in internal caches. For example, 256-point complex radix-4 FFT can be performed in 20.5μs on the 200-MHz V830R processor. Parallel execution ratio is as high as 66%. This shows that a simple superscalar mechanism contributes to a considerable speedup.

Table 2 shows a performance comparison between conventional and proposed implementations. Clock count is reduced by 35% when compared with a conventional radix-2

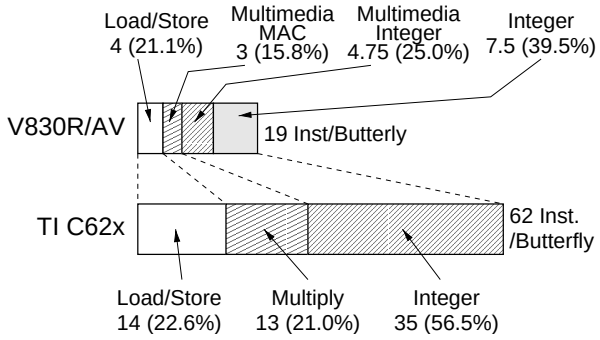


Figure 7: Radix-4 Butterfly Instruction Counts.

FFT on the Intel Pentium II processor [9]. The conventional algorithm is based on radix-2, and employs an irregular data structure which stores two sets of real and imaginary part into a single multimedia register. This data structure seems to be suited to the MMX MAC which performs $X1 \times Y1 + X2 \times Y2$, but has a bad influence on performance due to data format conversion overheads.

Texas Instruments TMS320C62x is a latest high-end DSP with a 8-way "multiple-instruction streams, multiple-data streams (MIMD)"-parallel capability [10]. When compared with this 8-way parallel DSP, the radix-4 FFT on V830R is slightly faster than radix-2 on TMS320C62x, but slower than radix-4 on TMS320C62x by 32% [11] (Table 2).

The main difference in execution times between TMS320-C6x and V830R comes from the access time of twiddle factors. TMS320C6x can access twiddle factors in background thanks to its high parallelism, but V830R has to spent 18.1% of the total execution time to access and pack twiddle factors into the SIMD format.

Figure 7 shows a comparison of average instruction counts to perform a radix-4 butterfly. It shows SIMD parallelism works very effective on the radix-4 butterfly, as V830R, which has 4-way parallel SIMD instructions, reduces the instruction count less than one third of the TMS320C62x. In contrast, because TMS320C62x has more flexible parallelism, these processors result in similar radix-4 butterfly execution times. Therefore an average calculation time of each butterfly is similar.

Therefore, it is shown that the multimedia coprocessor enables the V830R general purpose microprocessor to have a quite competitive signal processing performance with high-end DSPs.

5.2. Precision

The MAC instructions with symmetrical rounding introduced to V830R are very effective in reducing computational errors. The errors are evaluated by comparing results calculated by fixed-point MACs with truncation or rounding to double-precision floating point emulation.

	MAE	MSE
Rounding	4.8	19.5
Truncation	8.7	145.0

Table 3: Errors in 256-point FFT.

Table 3 shows mean absolute errors (MAEs) and mean square errors (MSEs) in 256-point complex radix-4 FFT for random numbers between –128 and 128. This table shows MSEs has been reduced to less than one seventh by introducing DSP-style MAC instructions with rounding to a general purpose processor.

6. CONCLUSION

A fast radix-4 complex FFT implementation suitable for microprocessors with a 4-parallel SIMD instruction set has been proposed. The implementation loads consecutive 4 real or imaginary elements into a register at stages except the last. At the last stage, every 4 elements in a real or an imaginary array are loaded into registers by a matrix transposition. These data structure enable the processor to deal with 4 butterflies in parallel at all stages. The algorithm implemented on the V830R processor reduces clock count by 35% in the case of 256-point FFT, when compared with a conventional implementation.

7. REFERENCES

[1] C. S. Burrus and T. W. Perks, "DFT/FFT and Convolution Algorithms," Wiley Interscience, New York, 1985

[2] K. Nadehara, I. Kuroda, M. Daito and T. Nakayama, "Low-Power Multimedia RISC," IEEE MICRO, Vol. 15, No. 6, pp.20–29, Dec. 1995.

[3] Linley Gwennap, "Intel's MMX Speeds Multimedia," Microprocessor Report, Vol. 10, No. 3, pp. 1, 6–10, Micro Design Resources, Mar. 5, 1996.

[4] Marc Tremblay et al., "VIS Speeds New Media Processing," IEEE MICRO, pp. 10–20, Aug. 1996.

[5] K. Suzuki, T. Arai, K. Nadehara and I. Kuroda, "V830R/AV: Embedded multimedia RISC processor," IEEE MICRO, Vol. 18, No. 2, pp. 36-47, Apr. 1998

[6] "Using MMX Instructions to Perform Complex 16-bit FFT," Intel Application Note AP-555, Order No. 243040-001, Mar. 1996

[7] "mediaLib User's Manual," Sun Microsystems, Part No. 802-7799-04, Oct. 1997

[8] R. Meyer and K Schwarz, "FFT Implementation on DSP-Chips — Theory and Practice," Proc. ICASSP 1990, pp. 1503–1506

[9] "Signal Processing Library Performance Specifications," ⟨URL:http://developer.intel.com/design/perftool/-PERFLIBST/spl/sp4spec.htm⟩

[10] "TMS320C62x/C67x CPU and Instruction Set Reference Guide" Texas Instruments, Literature No. SPRU189C, Mar. 1998

[11] "TMS320C62x Assembly Benchmarks," ⟨URL:http://www.ti.com/sc/docs/dsps/products/c6000/-c62x/benchmk.htm⟩