Research on the realization and optimization of FFTs in ARMv8 platform

Qi Du1,*，Hui Huang2
1College of Computer Science, National University of Defense Technology, Changsha, China
2HuNan Provincial Institute of Science and Technology Information, Changsha, China
*Corresponding author e-mail: 674163792@qq.com

Abstract. Fast Fourier transform (FFT) is a basic algorithm in digital signal processing and has been widely used in engineering, science and mathematical calculation. With the wide application of ARMv8 processor, it is more and more important to study FFT algorithm to realize high performance on ARMv8 platform. In the current open source FFT library, FFTW packages are mainly suitable for ARMv8 platform, while FFTW and FFTs packages are mainly suitable for ARMv7 platform. This paper implements and optimizes FFTs software package on ARMv8 platform, making it suitable for ARMv8 processor. The experimental results show that the performance of FFTs is improved by about 18% compared with FFTW in ARMv8 platform.

1. Introduction

At present, in the field of digital signal processing, DFT (Discrete Fourier Transform) is one of the most used the Discrete Transform algorithm, and FFT (Fast Fourier Transform) algorithm is a Fast algorithm of DFT algorithm, is widely used in digital communication, image processing, radar signal processing, aerospace and other fields[1].

In recent years, in the field of High Performance Computing (HPC), the computational performance and power consumption ratio and the performance-to-cost ratio have attracted much attention. ARM's ARMv8 architecture with its compared with traditional high performance processor in the whole of their own performance with low power consumption and higher superiority has been in the field of high performance computing, such as NVIDIA ARM IP core was used to construct the prototypes system, Barcelona is super center built Tididabo proof-of-concept system [2], the system contains 256 computing nodes, each compute node has two ARM kernel, and the whole system peak of 512 Gflops double-precision floating-point operation theory.

At present, there are relatively few FFT algorithm libraries on ARMv8 platform. However, with the wide application of ARMv8 processor, it is increasingly important to study FFT algorithm to achieve high performance on ARMv8 platform. In the current open source FFT library, FFTW[3] software package is mainly suitable for ARMv8 platform, while FFTW and FFTs[4] software packages are mainly suitable for ARMv7 platform.

In this paper, FFTs software package is implemented and optimized on ARMv8 platform to make it suitable for ARMv8 processor. The experimental results show that the performance of FFTs is improved by about 18% compared with FFTW in ARMv8 platform. The main contributions of this paper are as follows:

(1) This paper introduces the NEON intrinsic instruction function and assembly instruction related
to SIMD of ARMv8 into FFTs, which improves the performance of FFT algorithm in ARMv8 platform;
(2) FFTs of ARMv8 platform was implemented and its performance was compared with FFTW.

2. Background

2.1. FFT algorithm principle

At present, FFT algorithms are relatively good, including Cooley-Tukey algorithm, prim-factor algorithm, Rader algorithm, split-radix algorithm, etc. [5-6]. This paper mainly introduces Cooley-Tukey FFT algorithm. In 1965, Cooley and Tukey [7] published the butterfly algorithm, using the branching strategy to reduce the DFT calculation time from the original $O(N^2)$ to $O(N\log_2 N)$, where $N$ is the number of calculated data points.

In the field of digital signal processing, the Fourier transform of continuous time signal $x(t)$ can be expressed as:

$$x(w) = \int_{-\infty}^{\infty} x(t) e^{-jwt} \, dt \quad (1)$$

The spectrum of continuous signal $x(t)$ can be calculated by using discrete signal $x(n)$,

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \quad k = 0,1,\ldots,N-1 \quad (2)$$

Where, $X(k)$ is the spectrum of time series $x(n)$, and $W_N = e^{-j\frac{2\pi}{N}}$ is the rotation factor. For the time-domain sampling value of $N$ points, $N$ spectral points can be obtained through the calculation of formula (2), which is the discrete Fourier transform (DFT).

2.2. Introduction to FFTW and FFTs

The full name of FFTW is the fast Fourier Transform in the West, which is a standard C language assembly for fast calculation of discrete Fourier Transform. It can calculate one-dimensional or multidimensional real and complex data and discrete Fourier Transform of arbitrary scale, and supports input of arbitrary length and multidimensional data [5], which solves the problem of large FFT operation.

FFTW decompositions sequences of length $N$ into short sequences of length $N1$ and $N2$ for calculation. These short sequences are computed by a series of highly optimized C code snippets called solvers, which are independent methods of solving transformations. Solvers are managed and combined by the planner to perform the calculations. A combination is called a plan, which is a data structure that contains the calculated change path and Pointers to all input and output arrays.

For the same length $N$, there are many ways to decompose it. Different combination paths have different effects on the operation speed, and their performance will vary under different machines and platforms. Therefore, it is necessary to determine which combination has the best effect. FFTW selects the plan with the fastest execution speed by evaluating various combinations. Record this plan, and use only the corresponding plan for every transformation of the same length in the future, so that FFTW can adapt to the optimal operating environment [8].

FFTs, also known as the Fastest Fourier Transform in the South, was developed by Anthony Blake of Waikato university in New Zealand in 2012 as an open source project on GitHub. FFTs supports x86, x64, arm and other platforms, but currently does not support Aarch64. The last commit was on June 18, 2017.

The call flow of FFTW and FFTs is basically similar. Table 1 shows the basic call flow of one-dimensional complex DFT.
3. Development platform and compiler

The development platform is an ARMv8 64-bit processor with a frequency of 2.4GHz and 64GB memory. ARMv8 64-bit processor supports FMA (Fused-Multiply-and-Add) operation, the peak value of single core double precision operation is 9.6Gflops, and the peak value of single core single precision operation is 19.2GFlops.

Each CPU core has 32 128-bit floating-point vector registers, which can only enjoy L1 data Cache of 32KB and L1 instruction Cache of 32KB. Every two CPU cores share a L2 Cache with a size of 256KB, which is composed of a module with a total of 4 modules connected by high-speed interconnection. All modules share the 8MB L3 Cache through high-speed interconnection. The high speed interconnection provides access to memory through two DDR memory controllers. The relationship between L1, L2, and L3 caches is Inclusive.

The compiler is GNU C compiler GCC, version 8.3.0. This version provides the ”-march=armv8-a” compilation option for software compilation for the armv8 64-bit processor platform. On the ARMv8 64-bit processor platform, GCC8.3.0 provides support for NEON intrinsic instruction functions, including single-precision and double-precision floating-point operations and other intrinsic instruction functions.

4. Optimization of FFTs on ARMv8 platform

According to discrete Fourier transform (DFT) formula (2), the essence of Fourier transform is matrix vector multiplication (GEMV) between rotation factor matrix and input vector, namely

\[
\begin{bmatrix}
X(0) \\
X(1) \\
\vdots \\
X(N-2) \\
X(N-1)
\end{bmatrix} = 
\begin{bmatrix}
W_0^0 & W_0^0 & \ldots & W_0^0 & W_0^0 \\
W_1^0 & W_1^1 & \ldots & W_1^{N-2} & W_1^{N-1} \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
W_{N-2}^0 & W_{N-2}^{N-2} & \ldots & W_{N-2}^{(N-2)(N-2)} & W_{N-2}^{(N-2)(N-1)} \\
W_{N-1}^0 & W_{N-1}^{N-1} & \ldots & W_{N-1}^{(N-2)(N-1)} & W_{N-1}^{(N-1)(N-1)}
\end{bmatrix} 
\begin{bmatrix}
x(0) \\
x(1) \\
\vdots \\
x(N-2) \\
x(N-1)
\end{bmatrix}
\]

We know that matrix vector multiplication (GEMV) is very suitable for improving computing performance through SIMD instructions [9].

In GCC8.3.0, ”-mfpu=neon” compilation option is provided for the support of neon intrinsic instruction, in which the header file is arm_neon.h, the single-precision data type is float32x4_t, and the double-precision data type is float64x2_t. The NENO intrinsic instruction related to floating-point operation mainly includes:

1. single-precision instruction function: vaddq_f32, vsubq_f32, vmulq_f32, vmlaq_f32, vmlsq_f32, vst1_f32, vget_high_f32, vget_low_f32, vld1q_f32, vcombine_f32, etc.

2. double precision instruction function: vaddq_f64, vsubq_f64, vmulq_f64, vmlaq_f64, vmlsq_f64, vst1_f64, vget_high_f64, vget_low_f64, vld1q_f64, vcombine_f64, etc.

In general, the implementation of Cooley-Tukey FFT algorithm involves two key calculations: rotation factor generation and butterfly calculation.

In FFTs, when the length of the input signal sequence is less than 32, the generation of the rotation
factor is based on the look-up table method. The values of each phase are put into the code in a hard-coded form to reduce the generation time of the rotation factor; when the length of the input sequence is greater than 32, combined with the symmetry and periodicity of the rotation factor, the calculated rotation factor is put into a mapping index for the butterfly calculation.

In the part of butterfly calculation, this paper does not modify the algorithm of FFTs itself, but only replaces the ARMv7 instruction in the original FFTs package by introducing the NEON intrinsic instruction function of ARMv8 and assembly instruction, and modifies the name of the corresponding register. The main differences between ARMv8 and ARMv7 are shown in table 2.

| Table 2. Main differences between ARMv8 and ARMv7 |
|---------------------------------------------------|
| **Characteristics**                              | **ARMv8**         | **ARMv7**          |
| Instruction set                                  | Aarch64 and Aarch32 | A32 and T16        |
| Supported address length                         | 128               | 64                 |
| Number of general registers                      | 31                | 15                 |
| NEON                                               | default           | optional           |
| Number of SIMD registers                         | 32                | 32                 |

In FFTs, files related to the ARMv7 platform are codegen_arm.h, macros-neon.h, neon.s, neon_static.s, etc. By referring to the official document of Arm *® Architecture Reference Manual Armv8, for armv8-a Architecture profile* [10], replace the ARMv7 instruction and register name in the above file with the corresponding Armv8 instruction and register name. For example, replace vadd.f32 in neon.s with vadd.f64, vsub.f32 with vsub.f64, and so on.

5. Performance test, precision comparison and analysis

Based on FFTs, this paper presents FFT for Armv8 processor. Table 3 shows the detailed configuration of Armv8 64-bit multi-core processor test platform.

| Table 3. Test platform configuration |
|--------------------------------------|
| **processor**                        | ARMv8 64bits(2.4GHz) |
| **RAM**                              | 64 GB               |
| **operating system**                 | Ubuntu, kernel versions 4.19.0 |
| **compiler**                         | GCC 8.3.0           |
| **Compiler Options**                 | -march=armv8-a -mfp=neon |
| **FFTW versions**                    | 3.3.8               |

In this paper, the test results of FFTW library are taken as the benchmark. In order to evaluate the accuracy and performance of FFTs under different calculation scales, the length of input data \( N \) is selected from 2 to 134,217,728 for testing, and the input data is random Numbers in [-1,1] interval. Table 4~ Table 6 are the results of C2C calculation of test data for different dimensions of FFTW library. In the table, \( N \) represents data size, SP represents single-precision performance, and DP represents double-precision performance, the performance unit is MFlops.

| Table 4. FFTW one-dimensional test results |
|--------------------------------------------|
| \( N \) | SP     | DP     | \( N \) | SP     | DP     | \( N \) | SP     | DP     |
| 2     | 856.768 | 381.42 | 1024   | 6833.28 | 4893.99 | 524288 | 3580.22 | 2457.63 |
| 4     | 2741.28 | 1267.72| 2048   | 6435.52 | 4316.31 | 1048576 | 3428.70 | 2390.12 |
| 8     | 4296.80 | 3164.79| 4096   | 5882.24 | 3835.86 | 2097152 | 3373.58 | 2332.35 |
| 16    | 5294.24 | 4638.75| 8192   | 5141.12 | 3391.23 | 4194304 | 3203.26 | 2236.67 |
| 32    | 4761.92 | 5271.06| 16384  | 5123.20 | 3422.98 | 8388608 | 3068.74 | 2145.98 |
| 64    | 5230.88 | 5279.25| 32768  | 4756.64 | 3265.33 | 134217728 | 2906.97 | 1901.30 |
| 128   | 5719.68 | 5229.82| 65536  | 4702.56 | 3391.23 | 2679.22 | 1885.32 |
| 256   | 6316.16 | 5021.64| 131072 | 4096.32 | 2683.05 | 134217728 | 2906.97 | 1901.30 |
| 512   | 7265.60 | 5194.09| 262144 | 3683.20 | 2505.58 | 134217728 | 2679.22 | 1885.32 |
Table 5. FFTW two-dimensional test results

| N   | SP    | DP    | N   | SP    | DP    |
|-----|-------|-------|-----|-------|-------|
| 4x4 | 5699.92 | 3141.75 | 256x256 | 6121.95 | 3735.36 |
| 4x8 | 6624.57 | 4537.64 | 256x512 | 5902.27 | 3485.60 |
| 8x8 | 7642.40 | 5384.28 | 512x512 | 5633.28 | 2993.76 |
| 8x16 | 9733.91 | 6107.94 | 512x1024 | 5076.95 | 2762.56 |
| 16x16 | 9774.31 | 6212.19 | 1024x1024 | 4818.12 | 2590.88 |
| 16x32 | 9732.75 | 6121.27 | 1024x2048 | 4598.22 | 2478.13 |
| 32x32 | 8975.25 | 5501.99 | 2048x2048 | 4406.88 | 2403.89 |
| 32x64 | 8731.30 | 5697.21 | 2048x4096 | 4278.75 | 2363.93 |
| 64x64 | 7771.39 | 5194.75 | 4096x4096 | 4198.23 | 2310.53 |
| 64x128 | 6887.61 | 4558.72 | 4096x8192 | 3953.93 | 2208.90 |
| 128x128 | 6960.55 | 4311.21 | 8192x8192 | 3770.22 | 2090.67 |
| 128x256 | 6809.93 | 3811.04 | 8192x16384 | 3564.43 | 1987.33 |

Table 6. FFTW three-dimensional test results

| N   | SP    | DP    | N   | SP    | DP    |
|-----|-------|-------|-----|-------|-------|
| 4x4x4 | 6249.96 | 4144.45 | 32x64x64 | 6439.29 | 3997.21 |
| 4x4x8 | 7525.14 | 5088.32 | 64x64x64 | 6248.61 | 3731.56 |
| 4x8x8 | 7837.08 | 5823.89 | 64x64x128 | 5522.78 | 3167.52 |
| 8x8x8 | 8119.81 | 6340.29 | 64x128x128 | 4856.64 | 2620.32 |
| 8x8x16 | 7964.69 | 6237.56 | 128x128x128 | 5072.53 | 2379.78 |
| 8x16x16 | 7609.58 | 6001.77 | 128x128x256 | 4747.70 | 2308.32 |
| 16x16x16 | 7218.73 | 5743.25 | 128x256x256 | 4594.23 | 2297.13 |
| 16x16x32 | 6801.36 | 5200.23 | 256x256x256 | 4301.63 | 2267.89 |
| 16x32x32 | 6958.28 | 5006.57 | 256x512x512 | 4417.15 | 2208.47 |
| 32x32x32 | 7097.68 | 4808.87 | 256x512x512 | 4265.16 | 2145.83 |
| 32x32x64 | 6433.08 | 4201.89 | 512x512x512 | 3921.57 | 2007.51 |

5.1. Performance test

FIG. 1 show the performance diagrams of C2C calculation at one-dimensional input scales with single precision. From this figure, we can see that the single-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 18.35%. FIG. 1 show the performance diagrams of C2C calculation at one-dimensional input scales with double precision. From this figure, we can see that the double-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 7.34%.
FIG. 2 show the performance diagrams of C2C calculation at two-dimensional input scales with single precision. From this figure, we can see that the single-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 11.35%. FIG. 2 show the performance diagrams of C2C calculation at two-dimensional input scales with double precision. From this figure, we can see that the double-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 6.87%.

![Figure 2. Two-dimensional test results](image)

FIG. 3 show the performance diagrams of C2C calculation at three-dimensional input scales with single precision. From this figure, we can see that the single-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 11.73%. FIG. 3 show the performance diagrams of C2C calculation at three-dimensional input scales with double precision. From this figure, we can see that the double-precision performance of FFTs is generally higher than FFTW, with a maximum improvement of 3.82%.

![Figure 3. Three-dimensional test results](image)

5.2. Accuracy comparison

To compute the accuracy of an FFT, we compare the transform of uniform pseudorandom numbers in [-0.5,0.5) with the output of an arbitrary-precision arithmetic FFT (with > 40 decimal places of accuracy)[11]. The same random inputs are used for all transforms of the same size and type.

5.2.1. Comparison of single-precision FFTs accuracy. As shown in table 7, for single-precision C2C calculation, the accuracy of FFTW and FFTs calculation results is on the order of $10^{-7}$ if the one-dimensional input scale and data are consistent.
Table 7. Precision of single precision calculation results

| N   | L1 norm | L2 norm | Infinite norm |
|-----|---------|---------|---------------|
| 2   | 6.32e-08 | 5.84e-08 | 4.84e-08 |
| 4   | 3.51e-08 | 3.62e-08 | 4.91e-08 |
| 8   | 4.20e-08 | 4.56e-08 | 6.25e-08 |
| 16  | 7.11e-08 | 7.29e-08 | 6.80e-08 |
| 32  | 6.13e-08 | 6.07e-08 | 6.56e-08 |
| 64  | 7.98e-08 | 8.86e-08 | 1.36e-07 |
| 128 | 7.43e-08 | 7.66e-08 | 7.45e-08 |
| 256 | 7.45e-08 | 8.15e-08 | 8.76e-08 |
| 512 | 1.02e-07 | 1.07e-07 | 1.19e-07 |
| 1024| 1.13e-07 | 1.14e-07 | 1.34e-07 |
| 2048| 1.07e-07 | 1.09e-07 | 1.22e-07 |
| 4096| 1.17e-07 | 1.16e-07 | 1.21e-07 |
| 8192| 1.13e-07 | 1.16e-07 | 1.31e-07 |
| 16384| 1.25e-07 | 1.26e-07 | 1.34e-07 |
| 32768| 1.35e-07 | 1.37e-07 | 1.53e-07 |
| 65536| 1.41e-07 | 1.42e-07 | 1.55e-07 |
| 131072| 1.47e-07 | 1.48e-07 | 1.71e-07 |
| 262144| 1.57e-07 | 1.59e-07 | 2.20e-07 |

5.2.2. Double precision FFTs precision comparison. As shown in Table 8, for double-precision C2C calculation, the accuracy of FFTW and FFTs calculation results is on the order of $10^{-16}$ if the one-dimensional input scale and data are consistent.

Table 8. Precision of double precision calculation results

| N   | L1 norm | L2 norm | Infinite norm |
|-----|---------|---------|---------------|
| 2   | 0.00e+00 | 0.00e+00 | 0.00e+00 |
| 4   | 3.31e-44 | 3.90e-44 | 3.81e-44 |
| 8   | 3.67e-17 | 5.14e-17 | 6.09e-17 |
| 16  | 8.11e-17 | 9.35e-17 | 1.24e-16 |
| 32  | 9.49e-17 | 1.13e-16 | 1.18e-16 |
| 64  | 1.66e-16 | 1.37e-16 | 1.37e-16 |
| 128 | 1.28e-16 | 1.35e-16 | 1.46e-16 |
| 256 | 1.57e-16 | 1.59e-16 | 1.52e-16 |
| 512 | 1.54e-16 | 1.61e-16 | 1.62e-16 |
| 1024| 1.78e-16 | 1.81e-16 | 2.22e-16 |
| 2048| 1.89e-16 | 1.92e-16 | 2.27e-16 |
| 4096| 2.03e-16 | 2.03e-16 | 2.93e-16 |
| 8192| 2.15e-16 | 2.18e-16 | 2.65e-16 |
| 16384| 2.24e-16 | 2.27e-16 | 2.30e-16 |
| 32768| 2.34e-16 | 2.36e-16 | 2.41e-16 |
| 65536| 2.52e-16 | 2.54e-16 | 3.02e-16 |
| 131072| 2.58e-16 | 2.60e-16 | 3.18e-16 |
| 262144| 2.69e-16 | 2.71e-16 | 3.00e-16 |
6. Conclusion
Based on FFTs, this paper implements a version on ARMv8 platform, and compares its performance and accuracy with FFTW library. FFTs has higher performance than FFTW.

7. Acknowledgments
Thanks to my girlfriend Hui Huang for her support and encouragement!

References
[1] Kalamkar D D, Trzaskoz J D, Sridharan S, et al. High Performance Non-uniform FFT on Modern X86-based Multicore Systems[C]//Parallel & Distributed Processing Symposium. IEEE, 2012.
[2] Nikola R, Pall C, et al. Building Supercomputers from Mobile Processors[C], edaWorkshop13. 2013.
[3] Information on http://www.fftw.org
[4] Information on https://github.com/anthonix/ffts
[5] Frigo M, Johnson S G. The design and implementation of FFTW3. Proceedings of the IEEE, 2005, 93(2): 216-231
[6] Frigo M. A fast Fourier transform compiler// Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implementation. Atlanta, USA, 1999, 34(5): 169-180
[7] Cooley J W, Tukey J W. An algorithm for the machine calculation of complex Fourier series[J]. Mathematics of Computation, 1965, 19(90): 297-301.
[8] Frigo M, Johnson S G. FFTW: An Adaptive Software Architecture for the FFT[A]//Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing[C].1998.
[9] Weizhi Xu , Zhiyong Liu , Jun Wu, etc. Auto-Tuning GEMV on Many-Core GPU[A]// Parallel and Distributed Systems (ICPADS), 2012 IEEE 18th International Conference on Parallel and Distributed Systems[C].2012.
[10] Information on https://static.docs.arm.com/ddi0487/ea/DDI0487E_a_armv8_arm.pdf?_ga=2.126363085.1768605964.1576481267-390961851.1576481267
[11] Information on http://www.fftw.org/accuracy/method.html