A Parallel 1-D FFT Implementation Method for Multi-core Vector Processors

Zhong Liu(✉) and Xi Tian

College of Computer, National University of Defense Technology, Changsha 410073, China
zhongliu@nudt.edu.cn

Abstract. This paper presents an efficient parallel 1-D FFT implementation method based on the architecture features of multi-core vector processor. It divides the parallel computation of large-point 1-D FFT into the (n-m)-level parallel FFT computation and M-point parallel FFT computation according to the number of data points M that can be accommodated in the global cache (GC). The parallel FFT computation for each stage are performed using a shared DDR data method in (n-m)-level FFT computation. In the M-point parallel FFT computation, a parallel FFT computation method based on the matrix Fourier algorithm is designed, it converts the original M-point 1-D FFT computation into a 2-D FFT computation, and achieves parallel FFT computation using a shared GC data method, which avoids multiple data transfers between GC and AM and reduces data transmission overhead. Merge Column FFT computation with factor matrix multiplication and column FFT computation results in the AM, which further reduces the number of data transfer between AM and GC, and can significantly improve the efficiency of M-point FFT computation. The experimental results on Matrix show that the average speedup of the single-core single-precision 1-D FFT is 8.26 times and the average speedup of the dual-core single-precision 1-D FFT is 6.78 times compared with the TMS320C6678 with the same frequency.

Keywords: Multi-core vector processors · Large-point 1-D Fast Fourier Transform · Matrix Fourier algorithm · Parallel

Discrete Fourier Transform (DFT) is one of the most important algorithms for scientific computing, especially in the field of signal processing systems, such as radar signal processing, underwater acoustic signal processing, spectrum analysis, video image algorithm, speech recognition, etc. With the increasingly prominent problems of power consumption and heat dissipation, energy consumption has gradually become an increasingly important factor affecting high-performance computing systems, and the architecture of the processor is moving toward multi-core, many-core, heterogeneous GPUs, embedded DSPs, etc. How to improve the computational performance of DFT for novel architecture has been a research hotspot [1–7]. The fast Fourier transform (FFT) algorithm proposed by Cooley and Turkey [8] significantly reduces the

Supported by the National Natural Science Foundation of China under Grant No. 61572025.

computational complexity of the N-point DFT algorithm from the original $O(N^2)$ to $O(N\log_2 N)$. It can achieve high FFT computation performance if the FFT computation data can be stored in the on-chip memory of processors. On the other hand, because the FFT computation data of the same level is not reusable, the computation data needs to be processed multiple times in the processor's storage to complete the computing, thereby greatly reducing the computation performance of the FFT. Therefore, the computational performance optimization of large-point FFTs is very dependent on the layout and migration methods of the data, and it needs to be performed according to the processor architecture characteristics. Goedecker [9] and Karner [10] study how to use FMA instructions to reduce the number of multiply-add operations to optimize FFT computation performance for processors that provide FMA instructions. Liu [11] proposed a vectorization method using FMA to accelerate FFT calculations for FMA structured vector processors. HE [12] proposed a large point FFT optimization method for GPU architecture. FFTW [13] is a widely used FFT math library on general-purpose CPU platforms. It has good portability and can search for optimal FFT implementation based on processor architecture features. Daisuke [14, 15] studied the 1-D parallel FFT implementation of distributed storage. Jongsoo [16] proposed a 1-D FFT implementation method with low communication overhead for Intel's Xeon Phi coprocessor.

1 Matrix Architecture

As shown in Fig. 1, Matrix is a high-performance multi-core vector processor designed for high-density computing. It is designed as Very Long Instruction Word (VLIW) architecture and includes a Scalar Processing Unit (SPU) and a Vector Processing Unit (VPU). The SPU is responsible for scalar task computing and flow control, and the VPU is responsible for vector computing. Matrix includes a complex multi-level

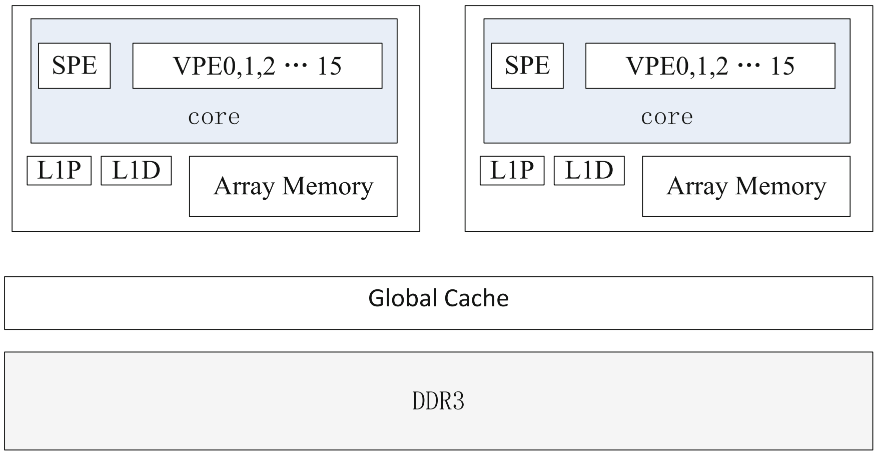

Fig. 1. Matrix multi-core block diagram

storage structure such as scalar register file, vector register file, scalar Level 1 cache, large-capacity on-chip vector array memory (AM), multi-core shared global cache (GC), and external shared DDR3. Matrix provides Fused Multiply-Add (FMA) instructions and its peak performance of single core is up to 100GFLOPS at 1 GHz. It can not effectively unleash Matrix's VLIW pipeline structure, vector computing and multi-level storage characteristics using traditional FFT algorithm, and results in low computing performance. Based on the architectural features of Matrix, this paper presents an efficient large-point parallel one-dimensional FFT implementation method, which can significantly improve the performance of large-point one-dimensional FFT on Matrix.

2 Parallel 1-D FFT Computing Method

The basic principle of the Decimation-In-Frequency (DIF) radix-2 FFT algorithm is as follows. Let x0, …. , xN − 1 be N = 2n points complex numbers. The 1-D DFT is defined by the formula:

$$X(k) = \sum_{i=0}^{N-1} x(i) W_N^{ik} \quad (\mathrm{k} = 0, \ldots\ldots, \mathrm{N} - 1)$$

Where 0 $\leq$ k < n, $W_N^{ik} = e^{-j(2\pi/N)ik}(j = \sqrt{-1})$ (known as the twiddle factor). Decompose the output sequence X(k) into two sequences by parity, then

$$\begin{cases} X(2k) = \sum\limits_{i=0}^{N/2-1} (x(i) + x(i+N/2) W_{N/2}^{ik} \\ X(2k+1) = \sum\limits_{i=0}^{N/2-1} ((x(i) - x(i+N/2)) W_N^i) W_{N/2}^{ik} \end{cases} \tag{1}$$

After N/2 butterfly computation as shown in Fig. 2, the N-point DFT is decomposed into two N/2-point DFT. This process can continue until N/2 2-point DFT computation.

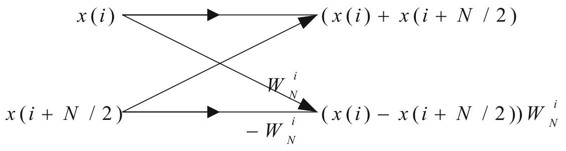

Fig. 2. Butterfly operation

In the above-mentioned one-dimensional FFT computation method, for each stage of FFT computation, N point data needs to be transferred from the off-chip DDR memory to the on-chip vector array memory. After the computation, the computation result is transferred from the on-chip vector array memory to the off-chip DDR memory. In the next stage of FFT computation, repeat the above process until all stages of FFT computation are completed. For large-point FFT, at each stage of FFT computation, because the total computation data far exceeds the capacity of the global cache (GC), the data in the GC is continuously replaced, but there is no hit, and the data transmission time cannot be reduced. The $N = 2^n$ point FFT computation, including n-level FFT butterfly unit calculation, requires 2n data transmissions back and forth. The data transmission time is very expensive; the data transmission time is much longer than the computation time, which results in low overall FFT computation efficiency.

Assume that the global cache (GC) can accommodate $M = 2^m$ point FFT computation data. After the (n-m)-level FFT butterfly unit computation, instead of computing all the FFT butterfly units step by step, 2^{n-m} M-point FFT computations are computed in sequence. In each M-point FFT computation, since the data is completely cached in the GC, it is not necessary to fetch computation data from off-chip DDR memories, which significantly reduces the data transfer time. It can effectively reduce the computation time of the M-point FFT, thus greatly improving the overall FFT computation efficiency. Therefore, the parallel computing of large-point 1-D FFT is divided into parallel computing of the (n-m)-level FFT and parallel computing of the M-point FFT.

2.1 (n-m)-Level Parallel FFT Computation Method

In the (n-m)-level FFT computation, each stage of FFT computation needs to transfer N-point data from the off-chip DDR memory to the on-chip vector array memory AM. Since the computation data are all one-time consumption, the data cached in the GC cannot be reused, and the test data indicates that the transmission time spent for transmitting data from the DDR to the AM is greater than the total computation time of the butterfly computation of each level. So parallel FFT computation is performed using a shared DDR data method, which can reduce the total computation time in the (n-m)-level FFT computation. As shown in Fig. 3, taking the first-stage FFT computation as an example, the core 0 computes the butterfly computation of the first and third 1/4-section data and the core 1 computes the butterfly computation of the 2nd and 4th 1/4-section data. After each stage is computed, the cores synchronize once to ensure data consistency. In the (n-m)-level FFT computation, the amount of data read each time is significantly greater than the length of the vector. Therefore, the data blocks computed by the butterfly unit are all continuously accessed in DDR, which improves the efficiency of DDR data access. The data transmitted to the AM is also a continuous data block, which facilitates vector data access and is easy to be vectorized.

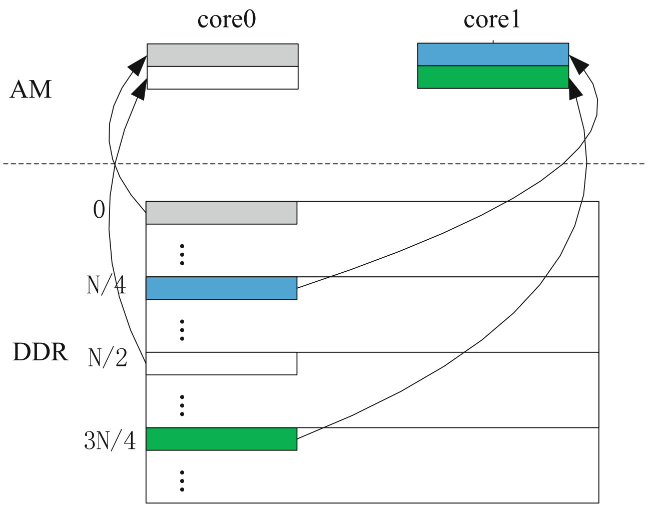

Fig. 3. Parallel 1-level FFT computing based on sharing DDR data

2.2 m-Level Parallel FFT Computation Method

After the (n-m)-level FFT butterfly computation, 2^{n-m} M-point FFT computations need to be computed in turn. In each M-point FFT computation, although the computation data of the M-point FFT can be completely accommodated in the GC, it cannot be fully accommodated in the vector array storage AM, and which requires 2 m data transmission processes. In order to facilitate vector computation and continuous data block transmission to improve FFT computation efficiency, it is still necessary to modify the original computation method. For this reason, the FFT vectorization computation method based on the matrix Fourier algorithm is designed, and the original M-point 1-D FFT is converted into a two-dimensional FFT computation. This method can significantly improve the computational efficiency of the M-point 1-D FFT.

Let M = RS, R = 2r, S = 2 s. The sequence x(i) is grouped into R subsequences of length S, that is, 1-D sequence x(i) is converted into a 2-D array sequence of the following form:

$$\begin{bmatrix} x(0) & x(1) & \cdots & x(S-1) \\ x(S) & x(S+1) & \cdots & x(2S-1) \\ \vdots & \vdots & \cdots & \vdots \\ x((R-1)S) & x((R-1)S+1) & \cdots & x(RS-1) \end{bmatrix}$$

Let i and k be mapped as follows:

$$\begin{cases} i = Si_1 + i_2, & \begin{cases} 0 \le i_1 \le R-1 \\ 0 \le i_2 \le S-1 \end{cases} \\ k = k_1 + Rk_2, & \begin{cases} 0 \le k_1 \le R-1 \\ 0 \le k_2 \le S-1 \end{cases} \end{cases}$$

Then X(k) can be transformed as follows:

$$\begin{aligned} X(k) &= X(k_1 + RK_2) \\ &= \sum_{i_2=0}^{S-1} \sum_{i_1=0}^{R-1} x(Si_1 + i_2) W_M^{(k_1 + Rk_2)\left(S_{i_1} + i_2\right)} \\ &= \sum_{i_2=0}^{S-1} \left\{ \left[\sum_{i_1=0}^{R-1} x(Si_1 + i_2) W_R^{k_1 i_2} \right] W_M^{k_1 i_1} \right\} W_S^{k_2 i_2} \end{aligned} \tag{2}$$

As can be seen from the above equation, the 1-D FFT computation of M-point can be converted into a computation similar to 2-D FFT. That is, first compute the S R-point column FFT computation by column, then multiply the result of the computation with a factor matrix, then compute the R S-point row FFT computation by row. Where, the computation data of R-point column FFT and S-point row FFT can be accommodated in the AM. It only need to design a suitable data layout method, convenient vector data access and computation, and it can achieve efficient computation efficiency.

In the M-point FFT computation, the parallel FFT computation is performed using the shared GC data method. As shown in Fig. 4, the first computation is the S R-point column FFT computation by column. The core 0 computes the first half of the S/2 R-point column FFT computation, and the core 1 computes the second half of the S/2 R-point column FFT computation. The cores synchronize once before performing row FFT computation to ensure that all data has completed column FFT computation.

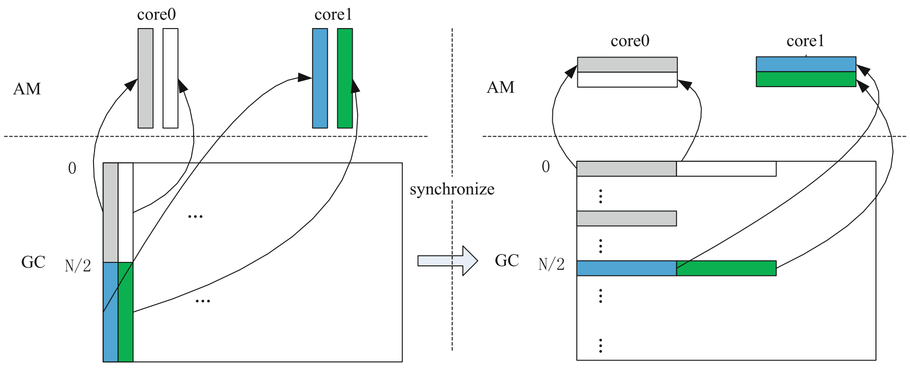

Fig. 4. Parallel M-points FFT computing based on sharing GC data

If directly performing computation according to the above formula (2), you need to complete all column FFT computation before multiplying the result data with a factor matrix. In order to avoid multiple data transmission between GC and AM and reduce data transmission overhead, after the column FFT computation in the AM is computed, the column FFT computation result in this section is multiplied by the factor matrix. It makes the column FFT computation and the column FFT computation result and the factor matrix multiplication only need one traversal computation, and no longer requires multiple data transmission between AM and GC, which improves the computation efficiency of the M-point FFT.

To further overlap the computation time into the data transmission time and reduce the total computation time, a computation method based on the DMA double buffering mechanism is adopted in each stage of the above-mentioned FFT computation, column FFT and row FFT computation.

2.3 FMA Optimized Buttery Computation

The FMA instruction provided by Matrix can further improve the computational efficiency of the FFT butterfly computation. Assume that the two inputs of a butterfly computation of the DIF radix-2 FFT are A and B respectively, the outputs are Y1 and Y2 respectively, the butterfly factor is W, and the subscripts r and i respectively represent the real and imaginary parts of the complex, according to the formula (1), there are

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ W & -W \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} 1 & \\ & W \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} A \\ B \end{bmatrix}$$

Expand the formula:

$$\begin{bmatrix} Y_{1i} \\ Y_{1r} \\ Y_{2i} \\ Y_{2r} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & W_r & 0 \\ 0 & 0 & 0 & W_r \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & -W_i/W_r & -1 & W_i/W_r \\ W_i/W_r & 1 & -W_i/W_r & -1 \end{bmatrix} \begin{bmatrix} A_r \\ A_i \\ B_r \\ B_i \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & W_r & 0 \\ 0 & 0 & 0 & W_r \end{bmatrix} \begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & -1 & W_i/W_r \\ 0 & 0 & -W_i/W_r & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} A_r \\ A_i \\ B_r \\ B_i \end{bmatrix}$$

Derive from the above formula, a butterfly computation requires 4 fusion multiply-add operations, 2 real multiplications and 2 real additions, and a total of 8 floating-point operations. It reduces 2 floating point operations compared to traditional FFT butterfly computation.

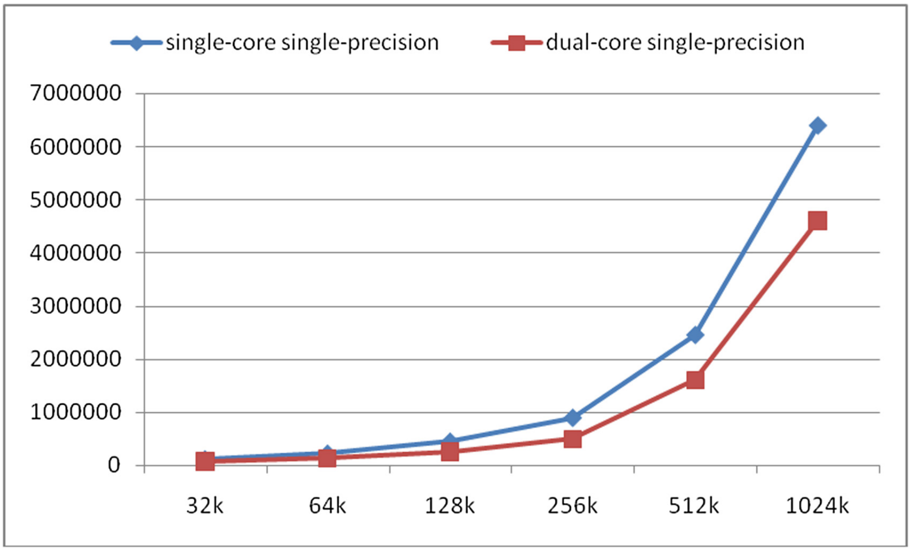

Fig. 5. The computation performance of single-precision 1-D FFT in matrix

3 Performance Testing and Analysis

The parallel 1-D FFT algorithm proposed in this paper was tested and analyzed in the RTL-level test environment of the multi-core vector processor Matrix. The system frequency of Matrix in the test is 1 GHz, the frequency of DDR3 is 1600 MHz, and the single and dual core peak performances are 100GFLOPS and 200GFLOPS, respectively. The single-core and dual-core peak performance of the same frequency TMS320C6678 tested at the same time is 16GFLOPS and 32GFLOPS, respectively. Statistical experimental data is averaged over multiple experiments.

First, as shown in Fig. 5, single-core and dual-core performance of a single-precision 32k, 64k, 128k, 256k, 512k, and 1024k-point radix-2 FFT is tested on Matrix respectively. The speedups from 32k to 1024k are 1.61, 1.69, 1.74, 1.8, 1.53, 1.39, respectively, and the average speedup is 1.63. As shown in Fig. 6, single-core and dual-core performance of a double-precision 32k, 64k, 128k, 256k, 512k, and 1024k-point radix-2 FFT is tested on Matrix respectively. The speedups from 32k to 1024k are 1.66, 1.72, 1.67, 1.48, 1.35, 1.26, and the average speedup is 1.52. It can be seen that when the computed data exceeds the GC capacity (single-precision 256k, double-precision 128k), the computation performance and speedup decreased significantly because of the limited DDR bandwidth.

Second, we compared the single- and dual-core single-precision 1-D FFT performance of the Matrix and TMS320C6678, respectively. As shown in Figs. 7 and 8, the test data shows that the computation time of the single-core single-precision 32k points radix-2 FFT is only 0.11 ms, and the corresponding performance of the TMS320C6678 at the same frequency is 0.915 ms. The performance ratios from 32k to 1024k are 8.29, 8.54, 9.26, 9.85, 7.61, 6.03, respectively, and the average ratio is 8.26.

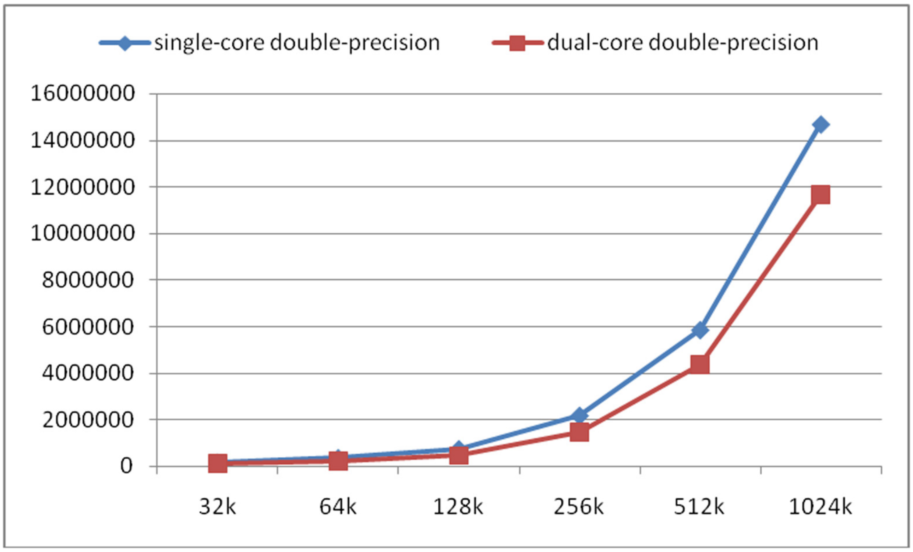

Fig. 6. The computation performance of double-precision 1-D FFT in matrix

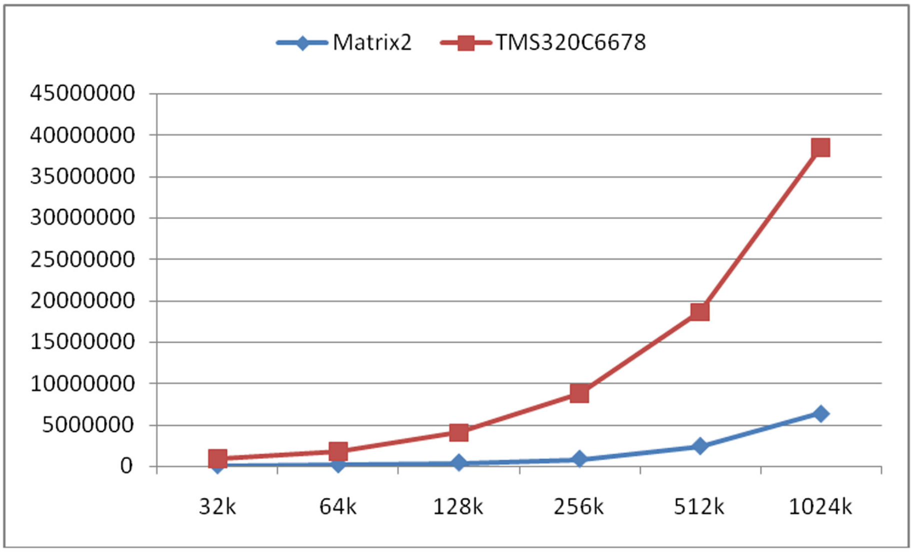

Fig. 7. Performance comparison of single-core single-precision 1-D FFT on matrix and TMS320C6678

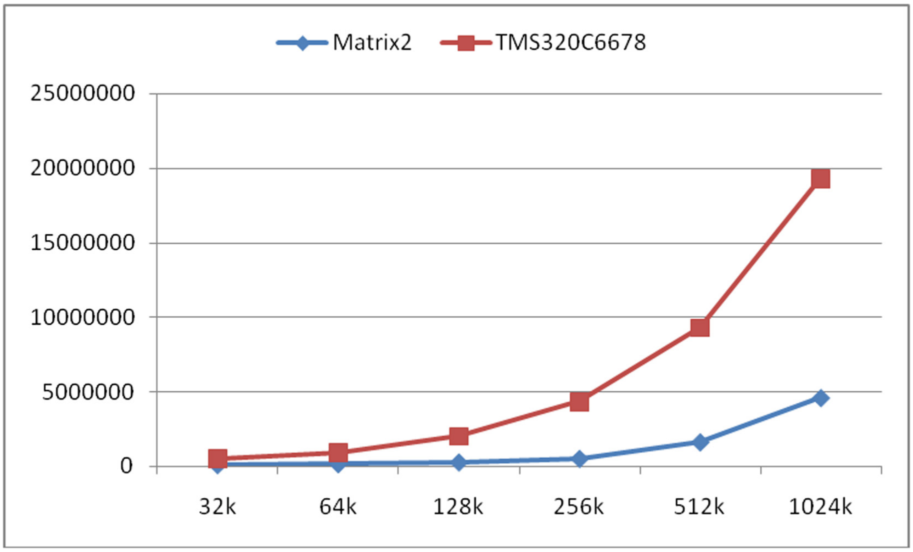

Fig. 8. Performance comparison of dual-core single-precision 1-D FFT on matrix and TMS320C6678

The computation time of the dual-core single-precision 32k points radix-2 FFT is only 0.0686 ms, and the corresponding performance of the TMS320C6678 at the same frequency is 0.478 ms. The performance ratios from 32k to 1024k are 6.97, 7.18, 7.86, 8.72, 5.78, and 4.19, respectively and the average ratio is 6.78. However, the peak performance ratio of the two is 6.25. It shows that the proposed 1-D FFT vector and parallel algorithm have higher computational efficiency and efficiently exploit the computing performance of Matrix.

4 Conclusions

With the rapid development of microprocessor technology, the architecture is becoming more and more novel and complex. The computation performance optimization of large-point FFT depends more on the mining of processor architecture features. The computation performance of large-point FFT is not only related to the processor's peak computational performance, but more importantly depends on the data storage layout and migration method. This paper proposes a large-point parallel 1-D FFT implementation method based on matrix Fourier algorithm for the independently developed vector processor Matrix. Experimental results show that the proposed parallel 1-D FFT implementation method based on multi-core vector processors has significant advantages. It can efficiently exploit the computing performance of multi-core vector processors. Compared with the TMS320C6678 with the same frequency, the average speedup of the single-core single-precision 1-D FFT is 8.26 times, and the average speedup of the dual-core single-precision 1-D FFT is 6.78 times.

References

1. Franchetti, F., Puschel, M., Voronenko, Y., Chellappa, S., Moura, J.M.: Discrete fourier transform on multicore. Signal Process. Mag. IEEE **26**, 90–102 (2009)
2. Gu, L., Siegel, J., Li, X.: Using GPUs to compute large out-of-card FFTs. In: Proceedings of the International Conference on Supercomputing, pp. 255–264. ACM (2011)
3. Pekurovsky, D.: P3DFFT: a framework for parallel computations of Fourier transforms in three dimensions. SIAM J. Sci. Comput. **34**, 192–209 (2012)
4. Pippig, M.: PFFT: an extension of FFTW to massively parallel architectures. SIAM J. Sci. Comput. **35**, 213–236 (2013)
5. Takahashi, D.: Implementation of parallel 1-D FFT on GPU clusters. In: 2013 IEEE 16th International Conference on Computational Science and Engineering (CSE), pp. 174–180, December 2013
6. Tang, P.T.P., Park, J., Kim, D., Petrov, V.: A framework for low-communication 1-D FFT. Sci. Program. **21**, 181–195 (2013)
7. Wang, E., Zhang, Q., Shen, B., Zhang, G., Lu, X., Wu, Q., Wang, Y.: Intel math kernel library. High-Performance Computing on the Intel® Xeon Phi™, pp. 167–188. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06486-4_7
8. Cooley, J.W., Turkey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
9. Goedecker, S.: Fast Radix 2, 3, 4, and 5 kernels for fast Fourier Transformations on computers with overlapping multiply-add instructions. SIAM J. Sci. Comput. **18**(6), 1605–1611 (1997)
10. Karner, H., Auer, M., Ueberhuber, C.W.: Multiply-add optimized FFT kernels. Math. Model. Methods Appl. Sci. **11**(01), 105–117 (2001)
11. Liu, Z., Chen, H., Xiang, H.V.: Vectorization of accelerating fast fourier transform computation based on fused multiply-add instruction. J. Natl. Univ. Def. Technol. **37**(2), 72–78 (2015)
12. HE, T., Zhu, D.: Design and implementation of large-point 1D FFT on GPU. Comput. Eng. Sci. **35**(11), 34–41 (2013)
13. Frigo, M., Johnson, S.G.: The design and implementation of FFTW. Proc. IEEE **93**(2), 216–231 (2005)
14. Takahashi, D.: A parallel 1-D FFT algorithm for the Hitachi SR8000. Parallel Comput. **29**(6), 679–690 (2003)
15. Takahashi, D., Uno, A., Yokokawa, M.: An implementation of Parallel 1-D FFT on the K computer. Int. Conf. High Perform. Comput. Commun. **248**(4), 344–350 (2012)
16. Park, J., Bikshandi, G., Vaidyanathan, K., Tang, P.T.P., Dubey, P., Kim, D.: Tera-scale 1D FFT with low communication algorithm and Intel® Xeon Phi™ coprocessors. In: Proceedings of SC13: International Conference for High Performance Computing, Networking, Storage and Analysis, vol. 31, no. 12, p. 34. ACM (2013)