Chinese Journal of Electronics
Vol.32, No.5, Sept. 2023

Vector Memory-Access Shuffle Fused Instructions for FFT-Like Algorithms

LIU Sheng, YUAN Bo, GUO Yang, SUN Haiyan, and JIANG Zekun

(*College of Computer, National University of Defense Technology, Changsha 410073, China*)

Abstract — The shuffle operations are the bottleneck when mapping the FFT-like algorithms to the vector single instruction multiple data (SIMD) architectures. We propose six (three pairs) innovative vector memory-access shuffle fused instructions, which have been proved mathematically. Combined with the proposed modified binary-exchange method, the innovative instructions can efficiently address the bottleneck problem for decimation-in-frequency or decimation-in-time (DIF/DIT) radix-2/4 FFT-like algorithms, reach a performance improvement by 17.9%–111.2% and reduce the code size by 5.4%–39.8%. In addition, the proposed instructions fit some hybrid-radix FFTs and are suitable for the terms of the initial or result data placement for general algorithms. The software and hardware costs of the proposed instructions are moderate.

Key words — Vector memory-access shuffle, Fused instructions, Vector SIMD, FFT-like algorithms.

I. Introduction

The fast Fourier transform (FFT) [1] is a basic algorithm for solving engineering and scientific problems and a universal benchmark in many processor domains (e.g., embedded system, scientific computing, and general-purpose desktop). Designing and executing FFT through processors efficiently and quickly is a common problem for processor designers (at the hardware level) and programmers (at the software level). Although a dedicated FFT hardware accelerator is a standard solution [2], this task is accomplished by writing programs in programmable processors more commonly.

Currently, vector single instruction multiple data (SIMD) structures are becoming popular in mainstream processors for their high power efficiency features. We counted the new processors released at the HotChips conference in 2015 and 2016 and found that nearly 70% of them adopted vector SIMD architecture to improve power efficiency. Thus, this architecture has become widely available outside the boundaries of CPUs, DSPs, and GPUs.

FFT algorithm has two parallel mapping forms, the binary-exchange (BE) and the 2D transpose (TR) [3], both of them can be mapped in vector SIMD structures. The TR is generally used to transform large FFT algorithms into small FFT algorithms in row and column directions, suitable for larger-scale FFT algorithms. The BE is suitable for smaller-scale FFTs (suitable for computing smaller-scale FFTs after the TR transformation) and is more widely used and can be used as the basis for the TR. This paper focuses on the BE consequently. When the BE is used to map the FFT-like algorithm on the vector SIMD structure, the first few rounds of the decimation-in-time (DIT) method or the last few rounds of the decimation-in-frequency (DIF) method need to use a shuffle unit to adjust the original or resultant data before and after each butterfly operation. The different algorithm radix, the different extraction methods, and the differences in fixed/float-point, single/double-precision of the data being processed lead to the need for shuffle data of FFT algorithm in the vector SIMD structure being more complex, which can quickly become the bottleneck of the system.

In fact, the algorithms like Viterbi decoding, discrete cosine transform, discrete sine transform, Hartley transform, etc., have similar memory-access and shuffle requirements as FFT in SIMD processor mapping, except the specific butterfly operation is different from that of the FFT algorithm. Since the technology proposed in this paper mainly solves the problems of memory-access and shuffle of these algorithms, which are not related to the specific butterfly operation and have general applicability to FFT-like algorithms, we

Manuscript Received Nov. 17, 2021; Accepted Dec. 24, 2021. This work was supported by the National Natural Science Foundation of China (61602493, 61502508).
 DOI:10.23919/cje.2021.00.401

will not intentionally distinguish between FFT and FFT-like algorithms in the following description.

In this work, we proposes six (three pairs) vector instructions with memory-access and shuffle based on mathematical proofs innovatively, fusing some functions of the two components of traditional SIMD processors, shuffle and vector memory-access. We also proposes a not-in-place storage binary exchange algorithm (BE_NIP), halting the number of data shuffles in the traditional in-place storage-based binary exchange algorithm and unifying the shuffle pattern. The combination of the two techniques can completely hide the communication between processing elements (PEs) of common types of FFT algorithms (e.g., DIT/DIF, radix-2/4, partial hybrid radix) on vector SIMD processors. Experimental results show that the proposed technique can improve the performance of FFT algorithms by 17.9%–111.2% and reduce the code size by 5.4%–39.8%. In addition, the proposed technique applies to the problem of initial or result data placement for general-purpose algorithms and has the advantages of the moderate overhead in hardware and software.

The rest of this paper is organized as follows. Section II presents the related research work. Section III analyzes the mapping process of FFT-like algorithms on vector SIMD structures. Section IV elaborates the proposed mapping method, vector memory-access shuffle fused instructions and binary swap with not-in-place storage. Section V evaluate the effectiveness of the proposed technique through experiments and finally Section VI gives a summary of the whole paper.

II. Related Work

The fused multiply-add instruction is the most well known instruction fusion technique. This instruction completes multiplication and addition operations with one instruction, fully exploiting the characteristics of multiply-add pairs in algorithms such as matrix multiplication, and has become almost standard in DSPs and widely used in CPUs.

Unaligned access is, in a sense, a fusion of memory-access instructions and shift instructions. Philips' TM3270 processor supports for 32-bit data unaligned accesses and does not require additional hardware execution clocks [4]. However, since there is only one set of access ports, single unaligned access cross a cache line can result in at most two cache misses. The TMS320C64x processor family from TI, USA, supports unaligned access to 32-bit and 64-bit data, but single unaligned access results in one of the dual access instruction slots not working [5]. The DSP from ADI, USA, uses additional data alignment buffer logic to implement unaligned access [6] efficiently.

Several previous works have fused memory-access and shuffle instructions functionally to accelerate FFT-like algorithms. Reference [7] proposed VHALFUP and VHALFDN instructions that can reduce some of the shuffle operations in the mapping of FFT algorithms, but only the time-domain radix-2 FFT algorithm is considered, and an additional MOV operation is required to store the intermediate results of the two instructions. The VEXC instruction proposed in [8] eliminates the extra MOV operation in [7]. However, it adds an extra register write port, and considers only the time-domain radix-2 FFT algorithm, thus has no acceleration effect on other types of FFT algorithms. Although these two methods can bring partial efficiency improvement to the FFT, the improvement is limited because the number of times and types required by the shuffle operations of the method is still very large.

The gather-scatter instruction [9], which has appeared in processors in recent years, reads or writes data in parallel in different address sequences by setting up several different address-generating units. This instruction is, in a way, also a technique for memory-access and shuffle fusion, but it will contain a large number of buffers and crossbar networks in hardware because of the address conflict problem that must be considered, and the vast hardware overhead makes the bandwidth it can support is limited, while the scalability of the instruction proposed in this paper is better. In terms of implementation complexity, the hardware overhead of the gather-scatter instruction increases into $O(P^2)$ as the vector width P grows, and in this paper it increases into $O(P)$. In Vision P6, the latest DSP from Tensilica, the bandwidth of gather-scatter is only half of the bandwidth of a typical vector memory-access instruction due to hardware complexity [10]. Thus the instructions proposed in this paper and the gather-scatter instructions can co-exist without any mutual inclusion relationship.

III. Mapping Analysis of FFT on Vector SIMD Structure

1. Vector SIMD structure and FFT algorithm

Fig.1 gives an abstract model of the vector SIMD architecture. P (P is a positive integer power of 2) processing elements execute the same instructions under the control of a fetch and dispatch unit. Each PE contains a private operation unit (usually several computing components, such as multiplier, shifter, arithmetic unit) and a vector register file (VRF). Register-level data exchange between PEs is achieved through shuffle unit, mostly crossbar structures [11], [12]. Vector memory (VM) consisted of M memory bodies provides

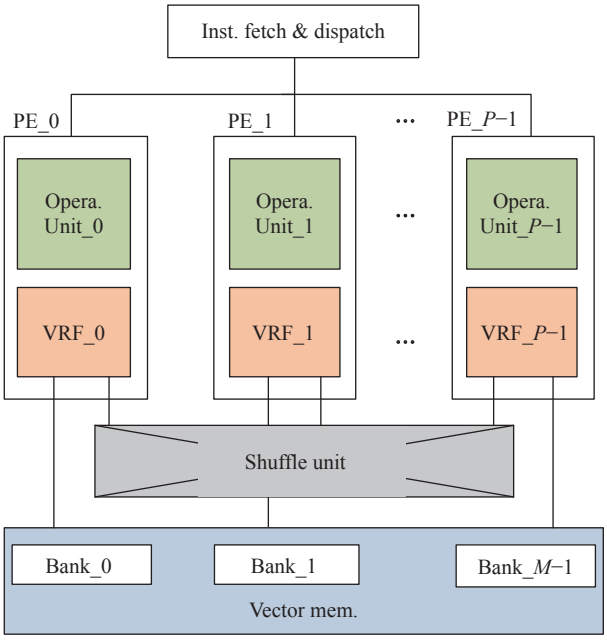


Fig. 1. Vector SIMD structure.

high bandwidth supply to VRFs.

In 1965, Cooley and Tukey invented the FFT algorithm [1] based on the mathematical characteristics of the discrete Fourier transform (DFT), which can reduce its computational complexity from $O(N^2)$ to $O(N\log_2 N)$. For the radix-2 FFT algorithm, which requires $\log_2 N$ levels to complete, the FFT algorithm can be divided into DIT and DIF, with the same results but slightly different processing: the operational flow graph of DIT FFT is basically reversed from that of DIF FFT, and its butterfly operation is multiplication followed by addition. In contrast, the DIF FFT is addition followed by multiplication.

The basic idea of the radix-n FFT algorithm is to divide the N points sequence into n N/n points sub-DFTs by ordinal number and decompose them in this way each time until the DFT is n point. Theoretically, a larger number of radixs can further reduce the number of operations, such as the radix-4 FFT algorithm has reduced the number of operations compared with the radix-2, and the number of iterations is half of the radix-2 FFT, which can further speed up the DFT operation. In addition to the radix-2 and radix-4 FFTs, which are the most widely used FFT algorithms, there are mixed-radix FFTs, where the last few levels are 2 or 4, occupying a relatively large proportion. For example, the 1200-point FFT in 3GPP LTE protocol, 1200 can be decomposed into $5\times5\times3\times2\times2\times2\times2$, that is, radix-5, 3, and 2 FFT operations can be performed respectively. If the number of points to be processed is not an integer power of 2 or 4, it can be transformed into a radix-2 or 4 FFT algorithm by using the zero-padding operation and then perform the zero-suppression operation at the end of the operation.

The input and output data, intermediate results, and butterfly factors in the FFT algorithm are generally complex numbers. According to the data representation of the real (or imaginary) part of the complex number, such as 32-bit (64-bit) fixed-point, single (double) precision floating-point, the FFT algorithm can be abbreviated such as 32-bit (64-bit) fixed-point FFT, single (double) precision floating-point FFT. There are two storage forms for complex data (not only including FFT algorithm): 1) The real and imaginary parts of the same element are stored next to each other, as shown in Fig.2 (a); 2) The real parts of all elements are together, and the imaginary parts are together, as shown in Fig.2(b). Obviously, the former storage method is more intuitive. However, when the machine word width of the SIMD processor and the width of the real/imaginary part of the complex data to be processed are the same (such as for a 32-bit processor, the real/imaginary part of the complex data are both 32 bits, for a 64-bit processor the real/imaginary part of the complex data are both 64 bits), for the former storage method, when these data are moved to the VRF, the real and imaginary data will be in the same register. However, the user wants the real and imaginary data to be in different registers (such as when $P=4$, the real part is in VR0, the imaginary part is in VR1) to facilitate the next operation. This will introduce additional shuffle operations after the traditional vector memory-access operations, reducing the program's execution efficiency.

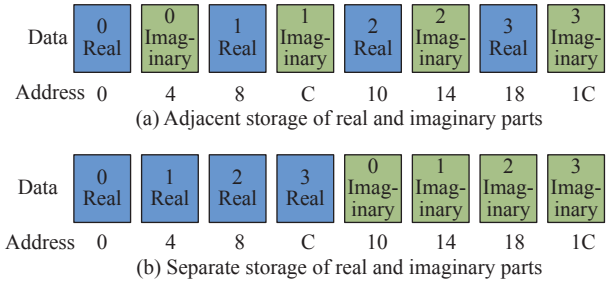


Fig. 2. Two ways to store complex data.

2. BE_IP mapping method and analysis of shuffle operation

Next, this paper analyzes the BE method and the shuffle operation, using the DIF radix-2 FFT algorithm as an example. It should be noted that the bit-reversal operation at the end of the operation is generally accomplished by a scalar unit or DMA that supports bit-reversal addressing and is not discussed too much here.

The execution of the DIF radix-2 FFT algorithm using the BE method in the vector SIMD structure is shown as follows [13].

1) Determine the number of iteration levels L and

shuffle levels K according to the length N of the DIF FFT transform and the number P of PEs in the vector SIMD, obviously $L = \log_2 N$. N is much larger than P in general, so $K = \log_2 P$. Furthermore, prepare the butterfly factor in advance.

2) Allocate the storage area. Load the data to be computed into the first area of the VM and the butterfly factor into the second area.

3) Take a batch of data to be operated and butterfly factors according to the computing power of the vector SIMD processor.

4) Determine whether the current round belongs to the previous $L-K$ round. If not, it goes to step 5). Otherwise, it goes to step 7).

5) Perform a shuffle operation with a butterfly operation and go to step 6).

6) Shuffle the results, store them back in the original storage location and go to step 8).

7) Perform butterfly operations and store the results to the original location.

8) Determine whether the current round is the end of the operation or not. If yes, go to step 9), otherwise go to step 3).

9) Determine whether the current round is equal to L or not. If not, increase the current round by one and go to step 3). Otherwise, end the operation.

From the above mapping process, we can see that the BE mapping method uses an in-place computation mechanism, and the method of using the same memory cell to store the same butterfly operation input and output data is called in-place operation [3], [13]. In-place computation saves memory cells in a sense and makes the FFT algorithm fast and straightforward. However, this method also results in the need to shuffle the data before the post-K-level butterfly operation. Moreover, after the calculation, the data needs to be shuffled back and saved to the VM. In this paper, we abbreviate the BE mapping method using in-place operations as the BE_IP mapping method.

In this paper, referring to the method of describing the acceleration ratio and efficiency of the FFT algorithm in reference [3], we assume that the total time spent by the DIF radix-2 FFT in vector SIMD using the BE_IP method is T_P. If a complex multiplication and a complex addition take time t_C, the acceleration ratio is $S = t_C \times N \times (\log_2 N)/T_P$, and the efficiency is $E = S/P = \frac{t_C \times N \times \log_2 N}{P \times T_P}$, then $T_P = \frac{t_C \times N \times \log_2 N}{P \times E}$. Considering that the shuffle network uses the crossover network with two inputs and one output, it takes two operations to complete the shuffle of $2P$ elements in each round of the BE method, and it needs four operations to shuffle the elements back again. Let the time spent for one shuffle operation be t_S, then the total time spent for the shuffle operation of the N-point FFT using the BE_IP method is $T_S = 4t_S \times (N/2P) \times \log_2 P$. To sum up, the proportion of the shuffle operation in the whole BE_IP mapping method is as follows:

$$\alpha = \frac{T_S}{T_P} = \frac{2t_S \times E \times \log_2 P}{t_C \times \log_2 N} \tag{1}$$

Considering the typical case, select $P = 16$, $t_S = 1$, and regard the complex multiplication and addition operation as pipeline ultimately, i.e., $t_C = 1$. For the three cases of efficiency $E = 0.30$, 0.35, and 0.40, the value α obtained according to formula (1) are shown in Fig.3, which shows that 1) The proportion of the shuffle operation in the whole BE_IP algorithm is more significant, between 17% and 32%; 2) As the number of FFT points increases, the proportion of the shuffle operation decreases, but the decrease is gradually becoming smaller.

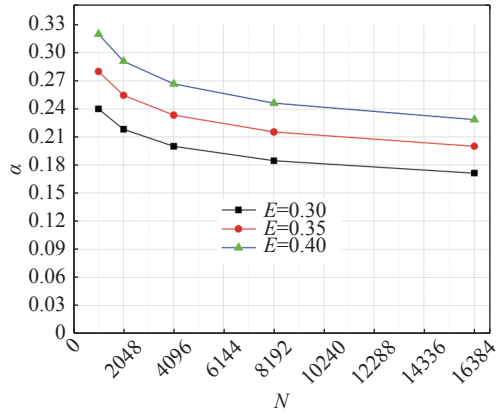


Fig. 3. Weight of the shuffle operation in BE_IP mapping method.

Excessive shuffle operations impact the performance of the FFT algorithm, mainly reflected in three aspects: 1) The bandwidth of the shuffle unit is limited. Crossbar is the most common implementation in the shuffle unit of SIMD processors due to its flexibility. However, the hardware overhead limit makes the crossbar implementation mostly use one vector input and one vector output, and the actual effective bandwidth is low. 2) There is an overhead in setting up the shuffle mode. Although the shuffle unit with crossbar can provide a variety of shuffle modes, it needs to set up and call different shuffle modes when in use, which introduces additional overhead. 3) The instruction issue slot is limited. In a typical vector SIMD processor, the multiplication unit, addition unit, and memory-access unit occupy the central part of the instruction issue slot, and there is usually only one shuffle unit, which may share an instruction slot with an addition or multiplication unit. This limitation of the instruction issue slot will reduce the execution efficiency of algorithms such as FFT.

It is essential to reduce the number of shuffle algorithms and the overhead of shuffle operations to improve the efficiency of the BE_IP algorithm due to the weight of shuffle operation in it. In addition, in the BE_IP mapping method of DIF FFT, each level in the post-K-level butterfly operation requires different shuffle patterns, which makes the programmer need to expand the post-K-level during programming, increasing program code size.

IV. Main Work

This section proposes the corresponding shuffle function in need for the radix-2 DIF FFT algorithm and proves its properties. Then gives BE mapping methods for not-in-place storage and proposes the corresponding vector memory-access fused instructions. And proposes six (three pairs) sets of vector memory-access fused instructions considering the cases of different radix (radix-2/4), different domains (DIF/DIT), and different element widths (element widths greater than, equal to, and less than the processor machine word widths).

1. The modulo shuffle function

Definition 1 If A and B are set $\{0,1,2,\ldots,2P-1\}$, then the modulo shuffle $f(x)$ is a function from A to B and $f(x)=\begin{cases}2x\%(2P-1), & 0\le x\le 2P-2\\ 2P-1, & x=2P-1\end{cases}$, P is a positive integer power of 2, and "%" denotes the modulo operation. Fig.4 shows the change in position of the data according to the modulo shuffle when $P=8$.

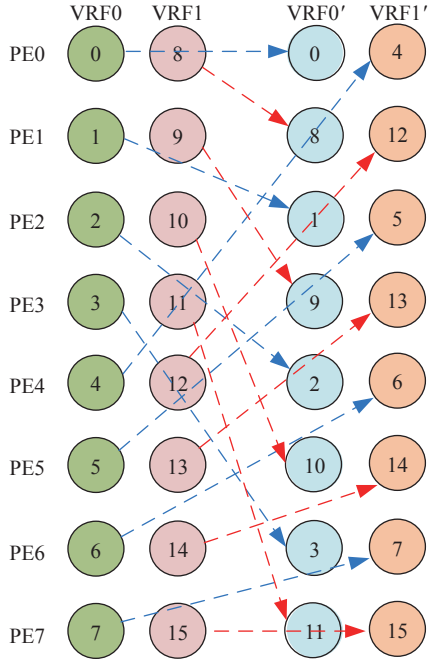


Fig. 4. The effect of modulo shuffle at P=8.

Property 1 $f(x)$ is a bijective function.

Proof 1a) First, prove that $f(x)$ is an injection. That is, prove that if $x_2\neq x_1$, then $f(x_2)\neq f(x_1)$.

i) When $x_2=2P-1$ and $x_1\neq 2P-1$, that is $0\le x_1\le 2P-2$, then $f(x_2)=2P-1$, $f(x_1)=2x_1\%(2P-1)$). Obviously $f(x_2)\neq f(x_1)$;

ii) When $x_1=2P-1$ and $x_2\neq 2P-1$, the case is similar to i);

iii) When $x_2\neq x_1$, $0\le x_1\le 2P-2$, $0\le x_2\le 2P-2$, use the converse method. Suppose that $f(x_2)=f(x_1)$, then $2x_1\%(2P-1)=2x_2\%(2P-1)$, i.e., $2(x_2-x_1)=K\times(2P-1)$. K is an integer. Since $0\le x_1\le 2P-2$ and $0\le x_2\le 2P-2$, $-2(2P-2)\le 2(x_2-x_1)\le 2(2P-2)$. K can only be -1, 0, or 1. Since $2(x_2-x_1)$ is an even number, and $K\times(2P-1)$ is an odd number when $K=\pm 1$, K cannot be ± 1 and can only be 0. Moreover, when $K=0$, obviously $x_2=x_1$, which contradicts the premise. So $f(x)$ is an injection.

1b) Second, prove that $f(x)$ is a surjection. That is, prove that for $\forall y\in B$, there exists $\exists x$ makes $f(x)=y$.

i) When $y=2P-1$, it is evident that $\exists x=2P-1$, $f(x)=y$ is satisfied;

ii) When $0\le y\le 2P-2$, it is obvious that when y is even, $\exists x=y/2$ satisfies $f(x)=y$; when y is odd, $\exists x=(2P-1+y)/2$ satisfies $f(x)=y$.

According to i) and ii) of 1b), i.e., for $\forall y\in B$, there exists $\exists x$ makes $f(x)=y$ so that $f(x)$ is a surjection.

From 1a) and 1b), the function $f(x)$ is both injection and surjection and thus is a bijection. The proof is over.

Property 2 When $x_1\le x_2$, then case i) $x_1,x_2\in\{0,1,2,\ldots,P-1\}$; case ii) $x_1,x_2\in\{P,P+1,P+2,\ldots,2P-1\}$, if any one of them is satisfied, then $f(x_2-x_1)=f(x_2)-f(x_1)$.

Proof 2a) Consider case i), when $x_1\le x_2$ and $x_1,x_2\in\{0,1,2,\ldots,P-1\}$, obviously $0\le x_1\le x_2\le P-1$. It is easy to obtain that $0\le 2x_1\le 2P-2$, $0\le 2x_2\le 2P-2$, $0\le x_2-x_1\le P-1$, and $0\le 2(x_2-x_1)\le 2P-2$. From Definition 1, we know that $f(x_2)=2x_2\%(2P-1)=2x_2$, and $f(x_1)=2x_1\%(2P-1)=2x_1$, hence $f(x_2)-f(x_1)=2(x_2-x_1)$. Moreover, $f(x_2-x_1)=(2(x_2-x_1))\%(2P-1)=2(x_2-x_1)$, so $f(x_2)-f(x_1)=f(x_2-x_1)$.

2b) The prove of case ii) is similar to that of case i) and will not be described in detail here. The proof is completed.

Property 3 Let $K=\log_2 P$, then $f^{K+1}(x)=x$.

Proof 3a) When $x=2P-1$, it is evident that $f^{K+1}(2P-1)=2P-1$. $f^{K+1}(x)=x$, at this point.

3b) When $0\le x\le 2P-1$, it is easy to know $0\le f^t(x)\le 2P-2$ by the bijection property of $f(x)$. Obviously $f^1(x)=2^1x\%(2P-1)$, $f^2(x)=(2f(x))\%(2P-1)=(2x\%(2P-1)+2x\%(2P-1))\%(2P-1)$. By the linearity theorem of congruence, we can get that $f^2(x)=(2x+2x)\%(2P-1)=(2^2x)\%(2P-1)$. Suppose that $f^t(x)=$

$2^t\%(2P-1)$, $2\le t\le K+1$, then $f^{t+1}(x)=2f^t(x)\%(2P-1)=(2^tx\%(2P-1)+2^tx\%(2P-1))\%(2P-1)$. By the linear property of congruence, we can get that $f^{t+1}(x)=(2^tx+2^tx)\%(2P-1)=2^{t+1}x\%(2P-1)$. In summary, it is known by mathematical induction that $f^{K+1}(x)=(2^{K+1}x)\%(2P-1)$. Obviously $2^{K+1}=2^{\log_2 P+1}=2P$, furthermore, by the linear property of congruence, we can get that $f^{K+1}(x)=(2Px)\%(2P-1)=((2P\%(2P-1))*(x\%(2P-1)))\%(2P-1)=x\%(2P-1)=x$. The proof is over.

According to the definition of modulo shuffle function $f(x)$ and the three corresponding property theorems, we can get the following conclusions:

1) From Property 1, after modulo shuffling a set of elements with $2P$ length, these $2P$ elements just swap their positions, and there will not be a situation where one element appears twice or more after the shuffle and the other does not appear.

2) From Property 2, if the distance of two elements in first (or last) P elements is $P/2, P/4, P/8, \ldots, 1$, their distance will become $P, P/2, P/4, \ldots, 2$ after the modulo shuffle. In vector SIMD architecture, two elements with the distance of P will be in one PE's VRF after vector memory-access, and they can do butterfly operations directly. Therefore, through the first modulo shuffle, two elements with the distance of $P/2$ in the first (or last) half P elements can do butterfly operations directly. This distance for direct butterfly operation becomes $P/4$ in the second modulo shuffle,..., and reaches P/P (i.e., 1) after $\log_2 P$ times of modulo shuffle. The above process is matched with the radix-2 DIF FFT.

3) From Property 3, for a totle of $2P$ elements, after $\log_2 P+1$ times of modulo shuffle, their location will recover to their initial value. In other words, after the first $\log_2 P$ modulo shuffle, arbitrary two elements with the distance of 1 in the first (or last) P elements can do butterfly operations directly, and then after another modulo shuffle, the sequence can be restored to the original position.

2. BE_NIP mapping method

We found that the in-place storage mechanism results in the excessive shuffle operations included in the BE_IP method. This paper proposes the not-in-place storage BE method (BE_NIP) method in conjunction with the modulo shuffle function $f(x)$ proposed in the previous subsection. The proposed method is shown in **Algorithm 1**.

1) The process of the first $L-K-1$ level operation is identical to the BE_IP algorithm.

2) After the butterfly operation is completed for the first set of data at the $K+1$ level to last, modulo shuffle the data in the VRF0 and VRF1 where the results are stored and then store the data in the VRF0 and VRF1 into VM. Process all the data of the K+1 level to last according to this method.

3) At the K level to last, the data is read from VM and performed butterfly operation directly, then modulo shuffle the data of VRF0 and VRF1 where the results are stored, and store the data in the VRF0 and VRF1 into VM. Process all the data of the K levels to last according to this method.

4) Complete the butterfly operation from the $K-1$ level to the last level according to the method in 3).

Algorithm 1 The BE_NIP mapping method

Input: data $x[N-1:0]$; butterfly factors.
Output: result data x.
for $i=0$ to $\log_2 N-\log_2 P-2$ do
  for $j=0$ to $N/2P-1$ do
    vector load to get current butterfly factors;
    vector load to get $x[2P*j+2P-1:2P*j]$;
  do P pairs of butterfly operations;
  vector load to put $x[2P*j+2P-1:2P*j]$;
  end for $j=0$ to $N/2P-1$ do
end for $i=0$ to $\log_2 N-\log_2 P-2$ do
for $i=0$ to $\log_2 P$ do
  for $j=0$ to $N/2P-1$ do
    vector load to get current butterfly factors;
    vector load to get $x[2P*j+2P-1:2P*j]$;
    do P pairs of butterfly operations;
    do $f(x)$ **shuffle** to the $x[2P*j+2P-1:2P*j]$
    vector load to put $x[2P*j+2P-1:2P*j]$;
  end for $j=0$ to $N/2P-1$ do
end for $i=0$ to $\log_2 P$ do

Obviously, the BE_NIP algorithm can reduce the overhead of shuffle in two aspects: 1) The shuffle required before and after the butterfly operation in the BEA_IP algorithm is reduced to the shuffle only before the butterfly operation, and no shuffle is required after the butterfly operation, i.e., the number of mixing is reduced by half. 2) The mode of shuffle is different in each of the post-K levels in the BEA_IP algorithm, while the mode of shuffle in the BEA_NIP is exactly the same (all modulo shuffling $f(x)$). On the one hand, it can reduce the overhead caused by switching the shuffle modes, and on the other hand, it can make the code of the FFT algorithm in the post-K level identical and reduce the code size.

3. Vector memory-access shuffle fused instruction

The BE_NIP mapping method proposed above can reduce the number of shuffle requests in the FFT algorithm to a certain extent and improve the execution efficiency. However, the shuffle request is still existing

in the BE_NIP method. And there is still a certain amount of overhead in the implementation of the most potent shuffle unit based on fully associative network, which is limited by the hardware overhead and the register write port. In this paper, we fuse shuffle operations into the vector memory-access unit to solve this problem. On the one hand, it enables the shuffle operation to take advantage of the high bandwidth of the datapath among the instruction issue slot of the vector memory-access unit, VM and VRF. On the other hand, since vector memory-access instructions supporting different modulo reassembly methods can be selected directly, the user will not need to set up the shuffle mode.

Definition VSTDWMX (vector store double word modulo X type) instruction Decode the address to access the VRF according to the instruction, read the double word data from the VRF, and modulo shuffling the data according to the $f(x)$, and then store it into the memory according to the target address.

Fig.5 illustrates the diagram of the effect of the VSTDWMX instruction execution when $P = 8$. It is obvious that the VSTDWMX instruction can directly replace line 14 and 15 in the BE_NIP algorithm. That is, the modulo shuffle operation in the BE_NIP algorithm can share the instruction slot and high bandwidth of the vector memory-access unit and can be completely hidden in the memory-access instruction.

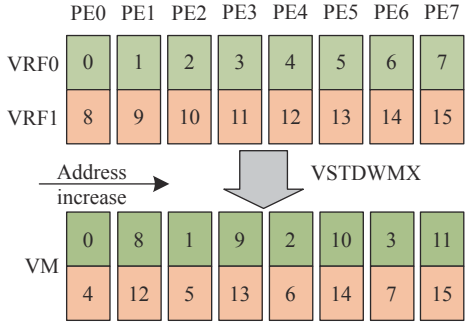


Fig. 5. The effect of VSTDWMX instruction execution when $P = 8$.

The above VSTDWMX instruction is proposed only when considering the radix-2 DIF FFT and the bit width of the data and the register are the same. This problem becomes complicated after considering various factors such as radix-2/4, DIF/DIT, the real part of the complex element being processed is 32/64-bit, and the processor is 32/64-bit. Fortunately, we found that adding three dedicated instructions to each normal vector load and store double-word access can satisfy all the above requirements. As shown in Table 1, VSTDWMX and VLDDWMX are a pair of instructions, mainly for the radix-2 DIF/DIT FFT algorithm. VSTDWMY and VLDDWMY are a pair of instructions, mainly for the radix-4 DIF/DIT FFT algorithm. And VSTDWMZ and VLDDWMZ are a pair, primarily to solve the problem that when the width of the FFT element is larger than the processor's register, adjusting the data position at the beginning or before the end of the operation. Specifically, VSTDWMX and VLDDWMX instructions are muxed inside the data shuffled by one-way vector double-word access; And VSTDWMY, VLDDWMY, VSTDWMZ, and VLDDWMZ instructions are muxed between the data shuffled by two-way vector double-word. The third column of Table 1 gives the proposed shuffle functions corresponding to the vector memory-access shuffle instructions, which have similar properties to the shuffle functions corresponding to VSTDWMX. The proof process is identical, so they are not repeated here.

In addition, using the proposed VSTDWMX and VLDDWMX instructions can complete the conversion between the two forms of complex data storage illustrated in Fig.2, which will accelerate the problem of placing the initial or result data of the general-purpose algorithm.

V. Analysis and Review

1. Performance optimization

1) Experimental setup

The experimental platform of this paper is selected from the subject's clock-accurate instruction set simulator FT-Matrix-Sim [12]. Briefly, FT-Matrix adopts a very long instruction word structure with parallel scalar and vector processing units. The instructions of scalar and vector units are dispatched by a common fetch and dispatch unit. The vector unit contains three MACs, one ALU/BP, and two Load/Store instruction issue slots, and the VPE performs register-level data exchange through the shuffle unit. The specific parameters are shown in Table 2.

In order to evaluate various approaches to implement the FFT algorithm, we add the VHALFUP and VHALFDN instructions in FT-Matrix-Sim regarding [7] and the VEXC from [8]. In addition, the FT-Matrix includes the shuffle unit and the vector memory-access shuffle fused instructions proposed above. For the FFT algorithm with same parameters, the following five schemes are implemented in assembly language. BASE indicates that use BE_IP algorithm and the shuffle instructions. VHALFUP/DN indicates that use BE_IP algorithm as well as VHALFUP and VHALFDN instructions. VEXC indicates that use BE_IP algorithm and VEXC instructions. NIP indicates that use the BE_NIP algorithm and shuffle instructions. And NIP_VLSSF indicates that use BE_NIP algorithm and the proposed vector memory-access shuffle instruction.

Table 1. The six vector memory-access shuffle fused instructions

Instruction	Explanation	Corresponding modulo shuffle function	Two-way vector memory access instruction participation	The application range
VSTDWMX	Vector memory access shuffle store instruction Type X	$f(x)=\begin{cases}2x\%(2P-1), & 0\le x\le 2P-2\\ 2P-1, & x=2P-1\end{cases}$	Two ways can be executed separately or simultaneously	The round radix-2 DIF FFT needs to be shuffled, part of the radix-4 DIF FFT when the number of PE is not an integer power of 4
VSTDWMY	Vector memory access shuffle store instruction Type Y	$f(x)=\begin{cases}4x\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	The round radix-4 DIF FFT needs to be shuffled
VSTDWMZ	Vector memory access shuffle store instruction Type Z	$f(x)=\begin{cases}2x\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	Adjustment of data position at the end of the operation when the element width is larger than the register width of the processor
VLDDWMX	Vector memory access shuffle load instruction Type X	$f(x)=\begin{cases}Px\%(2P-1), & 0\le x\le 2P-2\\ 2P-1, & x=2P-1\end{cases}$	Two ways can be executed separately or simultaneously	The round radix-2 DIF FFT needs to be shuffled, part of the radix-4 DIF FFT when the number of PE is not an integer power of 4
VLDDWMY	Vector memory access shuffle load instruction Type Y	$f(x)=\begin{cases}Px\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	The round radix-4 DIF FFT needs to be shuffled
VLDDWMZ	Vector memory access shuffle load instruction Type Z	$f(x)=\begin{cases}2Px\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	Adjustment of data position at the end of the operation when the element width is larger than the register width of the processor
VLDDWMX	Vector memory access shuffle load instruction Type X	$f(x)=\begin{cases}Px\%(2P-1), & 0\le x\le 2P-2\\ 2P-1, & x=2P-1\end{cases}$	Two ways can be executed separately or simultaneously	The round radix-2 DIF FFT needs to be shuffled, part of the radix-4 DIF FFT when the number of PE is not an integer power of 4
VLDDWMY	Vector memory access shuffle load instruction Type Y	$f(x)=\begin{cases}Px\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	The round radix-4 DIF FFT needs to be shuffled
VLDDWMZ	Vector memory access shuffle load instruction Type Z	$f(x)=\begin{cases}2Px\%(4P-1), & 0\le x\le 4P-2\\ 4P-1, & x=4P-1\end{cases}$	Two ways need to be executed simultaneously	Adjustment of data position at the end of the operation when the element width is larger than the register width of the processor

All FFT algorithms do not include bit reverse processing, and we disregard the overhead caused by all instruction misses (miss will become hit in multiple consecutive simulations). We also do not consider the overhead introduced by the configuration of the shuffle mode, the occupation of general-purpose registers during the shuffle operation, etc. The MOV operation required after the VHALFUP/DN operation is ignored (the MOV instruction can be provided from multiple instruction slots). The butterfly factors are calculated in advance and scheduled as required by the various algorithms described above. The processed elements are stored in the VM using a crossover of real and imaginary parts. All the above are optimized in manual assembly implementation by using various means such as software pipeline and loop unfold. At the same time, we consider using floating MAC unit in the iteration cycle as much as possible to improve the utilization of MAC. As the butterfly factors in the first stage of DIF FFT (or the last stage of DIT FFT) are all 1, there is no need to access memory to obtain the butterfly factor and perform multiply operations, which are optimized by using a dedicated iteration period and software pipelining methods. In the evaluation of the first four implementations, we use the same iteration period as the NIP_VLSSF algorithm.

Table 2. Specific parameters of the simulation platform

Name	Specific parameter value
Number of PE	4, 8, 16 (baseline), 32
VM Capacity	512 KB
Processor machine word length	64 bits
Shuffle unit	Crossover network implementation; two inputs and one output (baseline); two cycles pipeline (baseline) or one cycle completion (PE counts less than or equal to 8)
Instruction issue slot design	3 MAC (perform floating multiply-add, multiply or add) units, 1 BP (branch) unit, 2 vector LS units
Multiplier	4 stage pipeline, supports double floating or single floating SIMD
Adder	3 stage pipeline, supports double floating or single floating SIMD
MAC unit	6 stage pipeline, support double floating or single floating SIMD
Execution cycles of vector memory-access instruction	VLoad class instruction: 8 cycles pipeline. VStore class instruction: 4 cycles pipeline

2) Performance improvement of common FFT algorithms

Table 3 gives the normalized speedup ratio for five implementations of the FFT computation in radix-2/4 and DIF/DIT for 1024 and 4096 points single/double floating complex number. The VHALFUP/DN method proposed in reference [7] and the VEXC in [8] are only applicable to the radix-2 FFT algorithm but not to the radix-4. The NIP proposed in this paper and the further NIP_VLSSF can achieve performance improvement in both radix-2 and radix-4 FFT algorithms. In addition, the VHALFUP/DN brings a limited speedup, which is mainly due to the vital function of the BASE algorithm in the evaluation environment. Further, in the radix-2 FFT, the speedup of the NIP is greater than that of the VHALFUP/DN and smaller than VEXC. In contrast, the performance of the NIP_VLSSF is optimal. Compared with the BASE, the performance improvement of the proposed NIP is between 17.3% and 33.1%, and the NIP_VLSSF is between 52.8% and 97.1%. Compared with the best performing method VEXC in the previous reference, the performance improvement of the NIP_VLSSF is between 17.9% and 37.1%. In the radix-4 FFT, the performance improvement of the proposed NIP is between 13.5% and 27.8% compared with the BASE and the NIP_VLSSF is between 84.3% and 111.2%.

In addition, Table 3 also shows the statistics of the 1536-point single floating DIF algorithm, and since $1536 = 3 \times 2^9$, the same technique as the radix-2 FFT can be used for the last four stage of the rounds that need to be shuffled. Compared with the previous best method VEXC, the performance is improved by 21.6%.

By analyzing the parity rows in Table 3, the proposed NIP_VLSSF brings little difference in the speedup of the two FFT extraction algorithms, DIT and DIF, which is mainly due to the fact that the computation effort and the shuffle need are almost the same for these two extraction methods. For single and double floating FFT with same points, Table 3 shows that the performance speedup of the NIP_VLSSF is higher for double floating FFT. That is because the double floating FFT contains a slightly larger ratio of the shuffle operation. In addition, since the double floating numbers are stored consecutively in real and imaginary be-

Table 3. The performance improved in the common FFT algorithm

Number of points	Radix-2/Radix-4	Single/Double precision	DIF/DIT	Normalized acceleration ratio					NIP_VLSSF/M Radix-2:M=VEXC; Radix-4:M=BASE
				BASE	VHALF (UP/DN)	VEXC	NIP	NIP_VLSSF	
1024	Radix-2	Single	DIF	1	1.03	1.30	1.17	1.53	117.92%
1024	Radix-2	Single	DIT	1	1.03	1.28	1.17	1.54	120.09%
1024	Radix-2	Double	DIF	1	1.05	1.34	1.26	1.70	126.88%
1024	Radix-2	Double	DIT	1	1.04	1.36	1.24	1.70	124.76%
1024	Radix-4	Single	DIF	1	–	–	1.14	1.84	184.34%
1024	Radix-4	Single	DIT	1	–	–	1.14	1.93	192.88%
1024	Radix-4	Double	DIF	1	–	–	1.27	2.08	207.66%
1024	Radix-4	Double	DIT	1	–	–	1.28	2.11	211.16%
4096	Radix-2	Single	DIF	1	1.04	1.42	1.23	1.80	126.74%
4096	Radix-2	Single	DIT	1	1.04	1.40	1.24	1.84	131.34%
4096	Radix-2	Double	DIF	1	1.06	1.44	1.33	1.97	137.10%
4096	Radix-2	Double	DIT	1	1.04	1.46	1.30	1.95	133.77%
4096	Radix-4	Single	DIF	1	–	–	1.14	1.93	192.58%
4096	Radix-4	Single	DIT	1	–	–	1.14	1.97	196.73%
4096	Radix-4	Double	DIF	1	–	–	1.26	1.99	198.63%
4096	Radix-4	Double	DIT	1	–	–	1.22	2.01	201.46%
1536	Radix-3,Radix-2	Single	DIF	1	1.04	1.29	1.22	1.57	121.62%

fore computation, the NIP_VLSSF can solve this problem efficiently. For the same points, accuracy, and extraction method, Table 3 shows that the NIP_VLSSF has a higher speedup ratio for the radix-4 FFT algorithm. Although the radix-4 FFT requires fewer rounds of shuffle than the radix-2, the number of radix-4 butterfly operations that need to be shuffled is much more than that of radix-2, so the NIP_VLSSF can play a more significant role.

3) Effect of FFT points

Fig.6 illustrates the normalized speedup of single-precision floating-point complex radix-2 DIF FFT at different point sizes. On the one hand, it can be seen from equation (1) in Section III.2 that the share of the shuffle operations in the overall operation decrease as the FFT size increases. On the other hand, as the FFT points increase, the efficiency of the FFT mapping will show a slow increase until a smooth trend (e.g., the proportion of filling and emptying decreases gradually when using software pipeline). The two aspects determine that the speedup of the proposed NIP_VLSSF shows a trend of increasing, then decreasing, and then stabilizing with the increasing of FFT size.

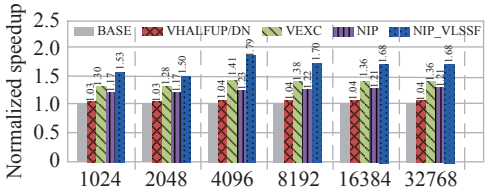
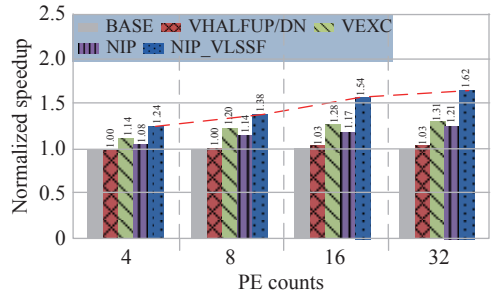


Fig. 6. Normalized speedup ratio of single-precision floating-point complex radix-2 DIF FFT at different scales.

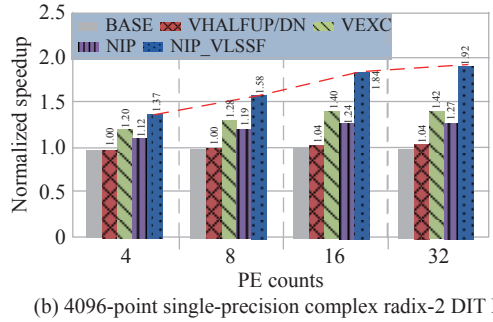
As illustrated in Fig.6, when the radix-2 DIF FFT size ranges from 1024 to 4096, the speedup of the proposed NIP and NIP_VLSSF increase from 1.17/1.53 to 1.23/1.79, compared with BASE. And as the FFT size ranges from 4096 to 32768, the speedup decrease to 1.21/1.68, then keep steady. We also found the same pattern in other FFT types. In addition, when the FFT points continue to increase, and the data and butterfly factors capacity exceeds the VM capacity, the performance of FFT will be affected by more factors, such as the performance of the Cache system, the storage bandwidth of DDR, etc.

4) Effect of the PE number

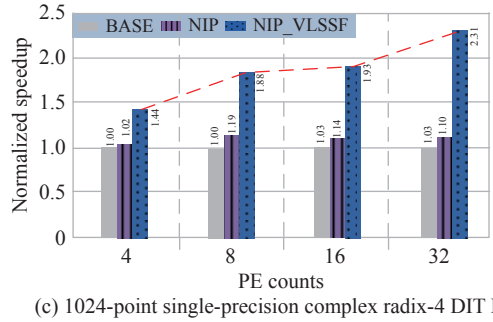
The normalized speedup of various implementations of the 1024-point and 4096-point single floating radix-2 and radix-4 DIT FFT with different PE counts are presented respectively in Fig.7. The rounds number that needs to be shuffled by the FFT increases with the PE counts. The proposed method mainly reduces the cost caused by shuffle operation. Thus, the performance improvement increases with the rise of the PE counts.



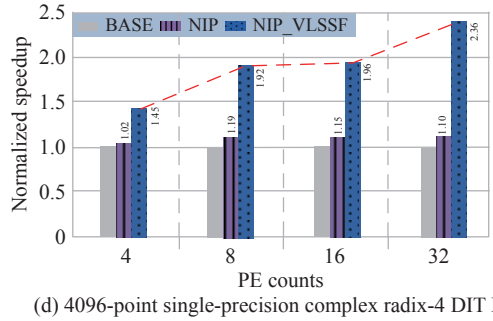
(a) 1024-point single-precision complex radix-2 DIT FFT



(b) 4096-point single-precision complex radix-2 DIT FFT



(c) 1024-point single-precision complex radix-4 DIT FFT



(d) 4096-point single-precision complex radix-4 DIT FFT

Fig. 7. Normalized speedup ratio of various methods with different PE number.

The performance speedup increases almost linearly with the PE counts, as shown in Fig.7(a) and (b). For example, for 1024 point radix-2 DIT FFT, compared with BASE, the NIP_VLSSF can bring a speedup of 1.24, 1.38, 1.54, and 1.62 as the PE count of 4, 8, 16, and 32. Morever, for 4096 point radix-2 DIT FFT, the speedup is even higher (1.37, 1.58, 1.84, and 1.92). The speedup of the traditional VEXC also increases with the PE counts, but they are lower than those of the

NIP_VLSSF.

Unlike the radix-2 FFT, in the radix-4 FFT, with the PE counts increasing, the speedup of NIP_VLSSF increases in a stepwise manner. The main reason is the number of rounds to be shuffled is 2 when PE counts are 8 and 16. And the rounds are 1 and 3 for PE count of 4 and 32. Thus, the speedup of the NIP_VLSSF does not vary much when PE counts are 8 and 16. As shown in Fig.7(c) and (d), the speedup of the NIP_VLSSF are 1.44, 1.88, 1.93, and 2.31 for 1024-point radix-4 DIT FFT when PE counts are 4, 8, 16, and 32, respectively, and 1.45 for 4096-point radix-4 DIT FFT. The VHALFUP/DN and VEXC do not work for the radix-4 FFT, while the NIP, which does not use vector memory-access shuffle instructions, provides a worse speedup than the NIP_VLSSF.

2. Code compression

Smaller code size means less instruction cache miss, and the NIP and NIP_VLSSF proposed in this paper can significantly reduce the code size of FFT. Traditional BEA_IP, including those accelerated by the methods VHALFUP/DN and VEXC, uses different sub modes of VHALFUP/DN or VEXC and shuffle instructions in the different shuffling rounds. So the assembly code cannot be unified for all rounds that need to be shuffled but need to be expanded, making the code size too long. Contrarily, the NIP and NIP_VLSSF proposed in this paper have the same shuffle instructions or memory-access shuffle fused instructions in the rounds that need to be shuffled, so the code can be unified in the shuffled rounds, and the code size can be significantly reduced. As shown in Fig.8, the code compression ratio of the NIP_VLSSF ranges from 18.4% to 39.8% for PE counts from 8 to 32 in 4096-point radix-2 single floating DIF FFT, compared with the VEXC. For 4096-point radix-4 single floating DIF FFT, the code compression ratio of the NIP_VLSSF ranges from 5.4% to 24.2% compared with the BASE.

3. Hardware overhead

The instructions proposed in this paper introduce some hardware overhead. First, the added vector memory-access shuffle fused instruction will occupy 2 bits of instruction coding space. Second, two additional registers are needed at each stage of the vector memory-access pipeline to record and pass the type of instructions. When implementing data shuffle in the vector memory-access pipeline, there will only be four types of data reassembly (including the case of no data reassembly) for vector Load instructions (or vector store). So two 1/4 MUXs are added at the address calculation and data return stage, each of which drives $4P \times T$ bits (suppose the vector SIMD processor has two vector memory-access units, each of which can issue a double-word vector memory-access instruction per cycle. P is the PE count, and T is the processor's machine word length). From the timing overhead point of view, the added 1/4 MUX has a small overhead and is negligible.

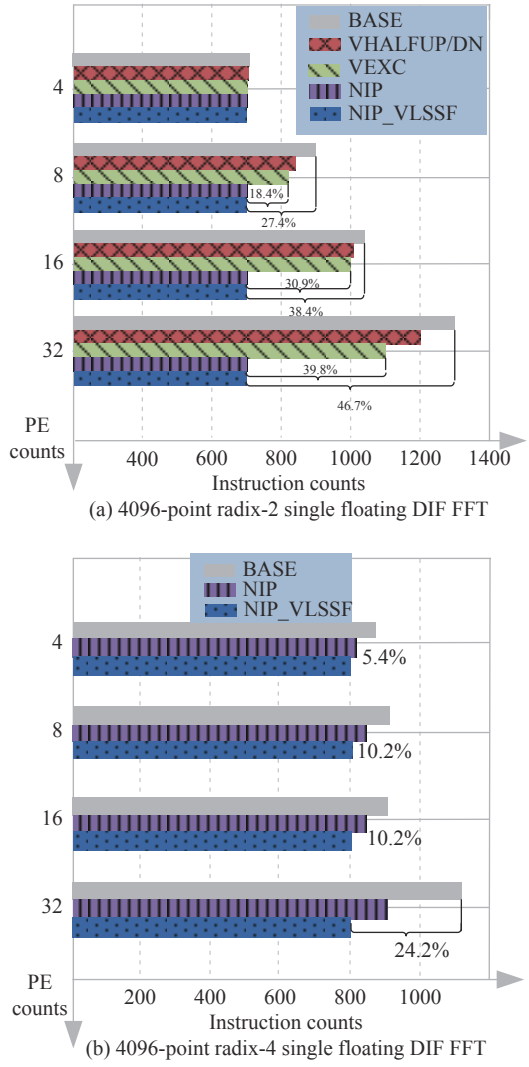


Fig. 8. Code size of FFT algorithm with different PE count.

4. Comprehensive comparison

In terms of hardware and timing overhead, the BASE has no additional hardware requirements other than the use of shuffle unit, and therefore the hardware overhead is minimal. The VHALFUP/DN requires further specialized instructions, but the overhead is not significant. The VEXC requires two reads and two writes ports to the VRF, which is often assigned to the BP unit (usually, the BP unit contains only one register file write port), so the VEXC adds a write port to VRF. Therefore, it has a higher timing overhead. The NIP_VLSSF shares some resources of the memory-access unit, and the additional hardware is mainly the MUX and a small number of flip-flops, so the hardware overhead is reasonable.

In terms of applicability, the VHALFUP/DN and

VEXC can only accelerate the radix-2 FFT algorithm, while the NIP_VLSSF has good acceleration for both radix-2 and radix-4 FFT and can be used for other algorithms with real and imaginary cross-storage. Thus, the applicability is optimal.

In summary, the NIP_VLSSF can effectively improve the execution efficiency of standard FFT algorithms, reduce the code size, have moderate hardware overhead, and is highly applicable. It has obvious advantages over the current solutions.

VI. Summary

In this paper, we propose a class of vector memory-access shuffle fused instructions for FFT-like algorithms and vector SIMD structures, which fuse some functions of data shuffle and vector memory-access parts of traditional SIMD processors. Together with the proposed binary swap mapping method of non-in-place storage with a modulo shuffle, it can completely hide the communication overhead between PEs and efficiently solve the data access and data replacement problems faced by FFT-like algorithms in vector SIMD structures. Thus has important theoretical and engineering significance.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol.19, no.90, pp.297–301, 1965.

[2] W. Hussain, F. Garzia, T. Ahonen, *et al.*, "Designing fast Fourier transform accelerators for orthogonal frequency-division multiplexing systems," *Journal of Signal Processing Systems*, vol.69, no.2, pp.161–171, 2012.

[3] A. Gupta and V. Kumar, "The scalability of FFT on parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol.4, no.8, pp.922–932, 1993.

[4] J. W. Van De Waerdt, S. Vassiliadis, S. Das, *et al.*, "The TM3270 media-processor," in *Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, Barcelona, Spain, pp.331–342, 2005.

[5] Texas Instruments, "Tms320c64x/C64x+ DSP CPU and instruction set reference guide, Texas Instruments User manual SPRU732C,"
Available at: *https://www.ti.com/lit/ug/spru732j/spru732j.pdf*, 2015

[6] J. Fridman, "Data alignment for sub-word parallelism in DSP," in *Proceedings of IEEE Workshop on Signal Processing Systems. SiPS 99. Design and Implementation (Cat. No. 99TH8461)*, Taipei, China, pp.251–260, 1999.

[7] R. Thomas, *An Architectural Performance Study of the Fast Fourier Transform on Vector IRAM*. University of California, Berkeley, Berkeley, CA, USA, pp.16–35, 2000.

[8] K. Zhang, S. M. Chen, S. Liu, *et al.*, "Accelerating the data shuffle operations for FFT algorithms on SIMD DSPs," in *Proceedings of the 9th IEEE International Conference on ASIC*, Xiamen, China, pp.740–743, 2011.

[9] B. S. He, N. K. Govindaraju, Q. Luo, *et al.*, "Efficient gather and scatter operations on graphics processors," in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, Reno, NV, USA, 2007.

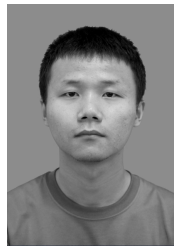
[10] G. Efland, S. Parikh, H. Sanghavi, *et al.*, "High performance DSP for vision, imaging and neural networks," in *2016 IEEE Hot Chips 28 Symposium (HCS)*, Cupertino, CA, USA, pp.1–30, 2016.

[11] M. Woh, S. Seo, S. Mahlke, *et al.*, "AnySP: Anytime anywhere anyway signal processing," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, Austin, TX, USA, pp.128–139, 2009.

[12] S. M. Chen, Y. H. Wang, S. Liu, *et al.*, "FT-Matrix: A coordination-aware architecture for signal processing," *IEEE Micro*, vol.34, no.6, pp.64–73, 2014.

[13] J. H. Huang, "*Design and implementation of vectorized FFTs on the YHFT-Matrix*," *Master Thesis*, Nat. Univ. Defense Technol., Changsha, China, 2012. (in Chinese)

LIU Sheng was born in 1984. He is a Ph.D. and Assistant Research Fellow in National University of Defense Technology. His main research interests include microprocessor architecture and VLSI design.
(Email: liusheng83@nudt.edu.cn)



YUAN Bo was born in 1996. He is a Ph.D. candidate of electronic science and technology. His main research interests include microprocessor architecture and VLSI design.
(Email: yuanbo18@nudt.edu.cn)

GUO Yang (corresponding author) was born in 1971. He is a Ph.D. and Research Fellow. His main research interests include low power VLSI circuit, microprocessor design & verification, and electronic design automation (EDA) for VLSI circuit.
(Email: guoyang@nudt.edu.cn)

SUN Haiyan was born in 1976. She is a Ph.D. and Associate Research Fellow in National University of Defense Technology. Her main research interests include compilation, programming model, and embedded application.
(Email: helen@nudt.edu.cn)

JIANG Zekun was born in 1996. He received the M.E. degree from Chongqing University of Posts and Telecommunications. His main research interests include microprocessor architecture. (Email: jzk61010@hotmail.com)