Design and implementation of a conflict-free memory accessing technique for FFT on multicore VLIW DSP

Hong Ye1a), Naijie Gu1, Xiaoci Zhang1, and Chuanwen Lin2b)
1 School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China
2 Department of Computer Science and Technology, Hefei University, Hefei Anhui 230601, China
a) yehong91@mail.ustc.edu.cn
b) lincw11@foxmail.com

Abstract: With the extensive application of signal processing, FFT on DSP processors evoke to be the trend. Aiming at solving the conflicts in memory accessing of these applications, an address generation technique called Mod-N Address is presented this paper. This scheme avoids the influence and conflicts of accessing to one specific bank at the same time. Instead, the data accessing pressure had been shared evenly into an independence block. Furthermore, a novel conflict-free memory-addressing scheme called Modulo-Block Scheduling (MBS) is proposed to ensure the continuous operation. We prove that the solution could achieve conflict-free accessing without extra time overhead and the unit had been implemented on our platform HXDSP104x. From the results, we can see that proposed unit achieves a 1.4 speed-up compared with the previous scheme, and experimental evaluations also show that the structure outperforms by an average of 8% to 54% compared with the conventional methods in performance.

Keywords: FFT, DSP, SIMD, conflict-free, Modulo addressing

Classification: Integrated circuits

References

[1] J. Eyre and J. Bier: “The evolution of DSP processors,” IEEE Signal Process. Mag. 17 (2000) 43 (DOI: 10.1109/79.826411).
[2] IEEE P802.11n/D4.00, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications; Amendment 4: Enhancements for Higher Throughput (2008).
[3] IEEE 802.16 Working Group. Part 16: Air interface for fixed and mobile broadband wireless access systems. IEEE P802. 16e/D8 (2005).
[4] 3GPP TR 25.814 v7.1.0. Physical Layer Aspects for Evolved UTRA (2006).
[5] M. Bahtat, et al.: “Instruction scheduling heuristic for an efficient FFT in VLIW processors with balanced resource usage,” EURASIP J. Adv. Signal Process. 2016 (2016) 38 (DOI: 10.1186/s13634-016-0336-0).
[6] M. Garrido, et al.: “A 4096-point radix-4 memory-based FFT using DSP slices,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 25 (2017) 375 (DOI: 10.1109/TVLSI.2016.2573708).
1 Introduction

The fast Fourier transform (FFT) has been widely used in the digital signal processing area, such as digital video broadcasting (DVB), Digital media broad-
casting (DMB), Digital audio broadcasting (DAB). In IEEE 802.11a/g/n [2], for example, 64 and 128-point FFT operations are commonly used. A computation sizes from 128 to 2048-point FFT are required in IEEE 802.16e [3] and 3GPP-LTE [4]. To meet with the ever-growing desire of high-performance and low-latency computation, the study of high throughput and conflict-free FFT obviously become an interesting field.

Common FFT processors can be classified into pipelined [6, 15] and memory-based architectures [14, 16, 17, 18, 20]. The tradeoff between hardware overhead and speed is the main criterion for selection. Specialized FFT processors implementations on ASIC or FPGA can provide high data throughput rate but lack flexibility. Meanwhile, general purpose processors offer flexibility but cannot meet the requirement of real-time computation. To solve the defects mentioned above, researchers have proposed DSP to achieve the tradeoff between flexibility and performance. Different from general processors, DSP separate the code with data segments which would require different addressing modes or bus fetching approaches [1]. What’s more, VLIW and SIMD are introduced in DSP processors to exploit the inherent parallelism and computational efficiency. Therefore, DSP processors have been widely used for processing FFT applications.

Modern multi-core DSP processors, such as Texas Instruments TMS320C6x [21], Analog Devices TigerSHARC [7], FreeScale StarCore MSC8x, are capable of vector operations and can be realized by the special instruction set. Utilizing these techniques, they can make maximum use of the processor’s underlying hardware and minimize the amount of memory space required. Multi-core DSP can also provide the required level of computational power for FFT applications in different types of parallelism which are referred to as Instruction-level Parallelism (ILP) and Data-level Parallelism (DLP).

In FFT applications, data read from memory, processed by PEs and stored back in memory, and this process repeats iteratively until all the stages of the FFT algorithm are calculated [6]. Because memory access is the major cause of data non-parallelization and power dissipated, a conflict-free memory addressing scheme is important for DSPs. Some Continuous-Flow FFT schemes are proposed in [14, 17], but Mixed-Radix FFT is unfit for DSP processors due to its complex implementation. An efficient modulo like FFT scheduling called instruction scheduling heuristic is proposed in [5], which took into account the hardware resources and cache structure. In [19], a Memory-Addressing Algorithm for solving Spatial and Temporal Conflicts was proposed, which adopts merged-bank memory to lower the area requirement and power dissipation of memory-based FFT processors. However, above-mentioned approaches only make sense in a particular platform, which can not apply to generic accessing conflicts.

In this paper, we present a memory-addressing technique called mod-N address to solve the bank conflicts problem for DSP processors. Mod-N address unit can adjust the sequence of data between stages and make conflict-free accessing with a cross addressing pattern. The proposed structure is achieved on a Multi-core DSP and can effectively deal with conflict-free memory address by decreasing the probability when reading/writing multi-group data from the same bank simultaneously. The addition of these hardware units not only requires little changes to the
instruction set architecture but also contributes to the significant performance improvement by removing conflicts and reducing the overhead cycles. Furthermore, a novel conflict-free memory-addressing scheme, named Modulo-Block Scheduling (MBS), is proposed in this paper to ensure the continuous operation of the FFT applications. The proposed structure and scheme has been implemented on our processor HXDSP104x, and shown a significant accelerate rate compared with the previous work.

The rest of this paper is organized as follows. Section II presents a brief overview of the typical multi-media algorithm FFT and the conflicts in this paper. The mainly Mod-N address scheme will be described in section III. Section IV and V describes the platform and MBS for solving the problem above in detail. The performance evaluation results and comparisons will be discussed in Section VI. Finally, Section VII provides the conclusions.

2 Background

2.1 FFT algorithm

FFT algorithms, proposed by Cooley and Tukey [8] in 1965, can be divided into many small series processed with radix-2/4/8 or other radix-2’ algorithms. With the development of the research, mixed-radix FFT algorithms [10], split-radix FFT algorithms [11] and even Prime Factor algorithm had been proposed. To improve the throughput and performance with a tradeoff between the resource consumption and hardware, high-radix FFT and parallel processing can be utilized. Although these novels FFT algorithms and mixed FFT algorithms may have low asymptotic complexity, most of them need additional complex addressing schemes and storage strategy which may decrease memory access efficiency for a SIMD DSP architecture [12].

As Radix-8 algorithms may be intricate for the mainstream SIMD patterns, Radix-4 and Radix-2 algorithms show a better regularity for SIMD memory access. Nevertheless, when the FFT input is in its input order, the output order will be arranged in the so-called digit-reversed. Radix-2 FFT schemes present a more straightforward re-ordering process than others and can be implemented easier for hardware directly. What’s more, because the multiplications in last two stages in the Radix-4 algorithm has no regularity for SIMD technique, and a Radix-4 algorithm can only perform the power of 4 points FFT, we prefer Radix-2 DIF FFT algorithm in this paper.

2.2 Conflicts in SIMD DSP

Due to the feature of high parallelism on FFT, SIMD architecture is adopted and dramatically increases the performance by manipulating data streams with a single instruction. Fig. 1 shows the structure chart of a 4-SIMD-FFT. Because of the sophisticated recursive address generation and bit-reversed addressing, the execute multiple instances of the same operation cannot be implemented in a SIMD instruction. This situation would cause concurrent data access constraints and collisions.
In this paper, $[n]$ is represented as the serial number in memory. When dealing with FFT algorithm, $P[n]$ is denoted by the number of FFT order. Notations $\text{load}([n_1],[n_2],\ldots)$, $\text{save}([n_1],[n_2],\ldots)$ are used as reading/writing from the memory in the specific number. The number of clusters controlled by a single instruction can be defined as the SIMD width $r$. Memory is usually divided into $r$ banks to provide sufficient bandwidth which supports data retrieval for each FUs. In SIMD execution model, assuming $a$ and $b$ are the initial address and offset address of a SIMD instruction, the real address sequence $F$ can be expressed as:

$$F = \{a + i \times b \mid 0 \leq i < r \}$$

where $i$ is a integer and $r$ addresses can be accessed simultaneously. The specific address $f_k$ will locate in the memory bank $i$ when:

$$i \equiv f_k \ (mod \ r)$$

To make the real address sequence $F$ locate in different banks without collision, each address should be notarized not to be placed in the same memory bank. Which can be expressed as:

$$\forall i \forall j : f_i \neq f_j \iff f_i \neq f_j \ (mod \ r)$$

Generally speaking, the SIMD operation would access memory in continuous data. The address space of any two adjacent memory operations in the sorted sequence must be equal to the size of the data they access. However, as the offset addresses mentioned above are calculated by specific functions flexibility, register bank contentions occurred because the obtain from the algorithms are uncertain. Memory conflicts thus come out when managing non-aligned and irregular access operations as the different clusters may access the same bank in an instruction. Hence additional instructions and extra overhead cycles will be needed to eliminate the conflicts. What’s more, the data exchange between each cluster is another problem for a DSP processor.
3 Methods for Mod-N addressing scheme

3.1 Modulo operation strategy

As we know, DLP is crucial for a processor in the form of SIMD extension. While in memory access, assuming the initial address, offset address, update address are \(a, b, c\) respectively. Thus in a \(r\)-width SIMD instruction, the sequence of offset addresses can be expressed as:

\[
0, b, 2b, \ldots, (r - 1)b
\]

(4)

And the continuous accessing addresses are:

\[
a + 0, a + b, a + 2b, \ldots, a + (r - 1)b
\]

(5)

The address update unit then updates the initial address to be: \(a + c\) in preparation for the next accessing.

Initially, we take offset addresses into account separately or assume the initial address \(a\) to be 0. As we know from [13], the sequence:

\[
0 \mod r, b \mod r, 2b \mod r, \ldots, (r - 1)b \mod r
\]

(6)

consist of precisely \(d\) copies of the \(r/d\) numbers

\[
0, d, 2d, \ldots, r - d
\]

(7)

in some order, where \(d = \gcd(b, r)\).

For example, when \(b = 10\) and \(r = 8\), we have \(d = 2\), and the numbers are

\[
0, 2, 4, 6, 0, 2, 4, 6
\]

(8)

If the numbers refer to the sequence of accessing banks, only bank with ID: 0, 2, 4, 6 are be accessed, and bank conflicts undoubtedly occur in this example because specific banks would be accessed more than one time simultaneously.

Therefore, when \(d = \gcd(b, r) = 1\), the sequence consist of precisely 1 copies of the \(r\) numbers

\[
1, 2, 3, \ldots, r - 1
\]

(9)

Hence,

\[
a + 0, a + b, a + 2b, \ldots, a + (r - 1)b
\]

(10)

consist of precisely 1 copies of the \(r\) numbers. In this occasion, each bank all be accessed for one time in a most optimal way.

Thus in mod-\(N\) address pattern, if \(\gcd(b, r) = 1\) (such as \(b = r + 1\)), the \(r\) continuous instructions could make a full-scale access in a \(N \times r\) region. Hence address \(f_i\) and \(f_j\) generated from the same sequence \(F\) will never belong to a same bank as \(f_i \neq f_j \pmod{r}\) in any case. What’s more, the data in each bank can be sent to specific registers in different clusters flexibly in this structure, it’s an outstanding idea for conflict-free memory accessing.

3.2 Mod-\(N\) address unit

In common SIMD accessing, processor conduct a load/save operation with the input operands of the base and the index register.

Due to the enormous overhead of eliminating conflicts and arranging such access, we adopt a flexible access unit called Mod-\(N\) memory addressing unit which
is shown in Fig. 2. The structure can adjust the order of priority in a minimum accessing cycles and strategy for a conflict-free addressing mapping.

When accessing memory in Mod-N addressing pattern, the address will be divided into two parts by $\log_2 N$. The addition is made in upper address part and lower portion by address generator separately. Commonly, $N$ equal to the number of banks or the SIMD width $r$. As there is no carried-over between these parts, the high-order of different addresses formed by a SIMD instruction is the same. When the carry bit came out, instead of accessing the corresponding memory, the mod-$N$ addressing will turn back to the top of the regions. Thus we can make full-scale access in each region in strategy periodically. It also has excellent effects in reading/writing from different storage units in array memory frequently. Because the data in registers can be sent flexibly and the bank will be accessed in diagonal flip, there is no bank confliction in this pattern. Thus if we generate instructions in specific strategy, there are no conflicts in memory access. As it just like the modulo operation in mathematical calculation, we called it Mod-$N$ addressing.

As there is no carry bit between the lower part and upper part, all the bank could also be accessed once in one instruction either. Hence, the instantaneous address arithmetic unit (IAAU) can figure out the real addresses from each bank without conflicts. In mod-$N$ address pattern, rotate reading is more frequently used instead of sequential reading and the data read/write from memory could be flexible.

4 Platform

Mod-$N$ address unit has been implemented on a DSP processor called HXDSP104x. It is the second generation DSP processor of HXDSP in a 16 nm technology, and the power consumption consumes only 10 w with the area of 30 mm * 30 mm, which can provide up to 134 GFLOPS. The memory capacity of the processor is 105 Mbit, and the communication bandwidth can be 252 Gbps. HXDSP104x is a 32 bit static super-scalar DSP targeting computation intensive data-parallel applications such as signal processing, multi-media applications. The summary of the architectural model is shown in Fig. 3.

This DSP has 16 transmitting slots in each VLIW packet with 1 GHz of CPU frequency inside. HXDSP104x consists of four element operation Macros (clusters) for execution. Each of them contains 8 ALU and MUL, 4 SHF, 1 SPU and a 128 registers data. The clusters are connected to six 1 MB blocks. Each block contains 8
banks which adopt an 8-way SIMD for parallel computation. The data will be shared and transmitted in a 512 bit Asymmetric full-duplex bus for reading/writing. And the proposed architecture Mod-N addressing unit showed in Fig. 2 is specialized hardware upgraded on the address generator in the processor.

![Fig. 3. Platform of HXDSP104x processor](image)

5 Design and implementation of FFT with mod-N address

5.1 Instruction set

The instruction sets in Table I below mainly describe the basic operation for a SIMD architecture. To illustrate the situation, we assuming the multi-cluster SIMD width to be 4 and each cluster indicated as $x, y, z, t$.

{Macro} $Rs + 1 : Rs = [Um + = Un, Uk]$ is a load instruction, in which $Um$ and $Un$ denote initial address and offset address. Update address is represented as $Uk$, and Macro denotes the specific SIMD clusters. Meanwhile, $s, m, n$ and $k$ are the register numbers of the destination operand and the two source operands. In this way, data addressed by $Um$ and $Un$ are loaded to the specified registers named $Rs$ on the specified clusters.

| Instruction set | {Macro} $Rs + 1 : Rs = [Um + = Un, Uk]$ |
|-----------------|-----------------------------------------|
| Vector load/store | $[Um + = Un, Uk] = {Macro} Rs + 1 : Rs$ |
| Vector addition | $[Macro] Rs = Rm + Rn$ |
| Vector subtraction | $[Macro] Rs = Rm - Rn$ |
| Vector multiplication | $[Macro] Rs = Rm \times Rn$ |
| Mod-N address load/store | $m[Un + = Um, Uk] = {Macro} xRayRbzRctRd$ |
| | $[Macro] xRayRbzRctRd = m[Un + = Um, Uk]$ |

Other than general accessing instructions, Instruction for Mod-N address could appoint specific clusters and the number of registers flexibly. In these Instructions, a
processor will access the particular registers appointed by $Ra$, $Rb$, $Rc$, $Rd$ from each cluster successively. Address generator will figure out the real addresses calculated by a modulo operation.

5.2 Modulo-block scheduling

Here we take a 4-SIMD $P$-point FFT algorithm for example. As we can see from Fig. 1, data can be read and sent to 4 different clusters in sequential order and calculated in the first $\log_2 P - 2$ stages respectively. However, because data calculated in the last two stages belong to neighboring banks, accessing conflicts may be caused as those data cannot be sent to the same cluster in one cycle.

In common SIMD accessing, data from each bank can be read/write for once in an instruction. In this case, memory would be accessed in a row and sent to the different clusters. However, when the algorithm and the instruction need to send these data into the same cluster, memory conflicts and pipeline stuck inevitably occurred. In order to realize the conflict-free accessing efficiently with maximum throughput, a scheme called Modulo-block scheduling (MBS) was proposed. In this occasion, we would access a sequential row of the memory instead of a specific one and make parts of access sequence deferment by adjusting the offset address in an accessing section. As mentioned in 3.1, each bank of the memory can also be read for once in instruction if we follow specific rules.

In Modulo-block scheduling, It takes $n$ continuous instructions to adjust the order for an $N$-bank DSP by mod-$N$ address. The main procedure illustrated in Fig. 4. Each column corresponds to one memory bank and shows how the data are stored in the memory when bank conflicts occurred. As the data of real part and imaginary part in the same point can be stored in neighboring banks, the width $N$ is 8, and we can assume the $n$ to be 4. Thus the offset address can be $9 = 8 + 1$.

At the first instruction operation, instead of $load([0, 1], [2, 3], [4, 5], [6, 7])$ and send them to the row 0, we can $load([0, 1], [10, 11], [20, 21], [30, 31])$ and send consecutive two of them to the sequential row in main diagonal line. Secondly, Point([8, 9], [18, 19], [28, 29], [6, 7]) would be transferred and sent to the memory. Notice that in the Mod-N address pattern, the last group of point is $[6, 7]$ instead of $[38, 39]$ by modulus operation. The data would be written back in the second
diagonal line, and \( P[6,7] \) would be written to the first column in modulus operation. After that, \( P[16,17],[26,27],[4,5],[14,15] \) would be extracted and written back in the third diagonal and \( P[4,5], P[14,15] \) would be written to the top column in modulus operation. Finally, we would \( \text{load}([24,25],[2,3],[12,13],[22,23]) \) and written back. We can see that Point([10,1],[2,3],[4,5],[6,7]) had been read in the continuous instructions instead of a individual instruction, they can be sent to the same cluster conflict-free.

In this case, we can realize the conflict-free modulo access. A matrix transposition has been mapped in a small section when the data write back to the memory. Thus the data could be sent to the clusters fluently as the data has been adjusted to the different banks. As this scheme can execute the process conflict-free within a group of specific instructions, it has same throughput capacity as common address pattern. We called it Modulo-Block Scheduling (MBS).

6 Performance evaluation

We implemented the Mod-N address unit on the HXDSP104x DSP processor and its corresponding simulator. A good feature of the proposed architecture is its scalability for every size. Any size of FFT can be mapped to the DSP conflict-free, which improved the transfer efficiency significantly.

As we can see from the Fig. 5, performance comparisons of FFT cycles between proposed scheme and previous algorithm had been presented. The central ordinate on the left represents the computation cycles, and the deputy ordinate right side means the speedups above. When the size is smaller than 256, the comparative result exhibits a progressive increase in speed-up. It mainly because the overhead of solving conflicts is relatively lower in small size and then the speedup eventually reached a stable level. When the FFT size is larger than 2048, the memory contention becomes seriousness with size as Mod-N address had shown a more prominent role in conflict-free access.

Fig. 6 shows the performance comparisons between TMS320C6678 and our processor. Texas Instrument's TMS320C6678 is one of the primary examples in this field, which is an eight-core DSP and built on TI's innovative KeyStone™ architecture [22]. Our processor could achieve a speed-up more than 5.13 compared with C6678 in eight-core pattern with Mod-N Address. The speedups may decrease at 3.35 to 2.22 due to the restricted to memory size and transmission bandwidth. Note that although TMS320C6678 produced good grades, our architecture outper-

![Fig. 5. Comparison with and without Mod-N address](image-url)
forms outstanding cycles in comparison. We can reach a better speed-up at 6.56 to 8.79 compared to the 1k- to 4k-FFT implemented on TMS320C6678.

Fig. 7 summarizes the comparison of several memory-based FFT processors with their respective addressing schemes. In the performance comparisons here and all above, data samples were the single-precision complex floating point, with imaginary parts in odd indexes and real parts in even ones. Therefore, The 8 banks of HXDSP104x would be divided into 4 group. The picture shows that our proposed addressing scheme outperform more than 54% in computation cycles compared to [14, 17, 15] when the FFT size is larger than 2048. The cycle costs may be a little higher compared with [16, 18] when the size of FFT is small, it is just because the arithmetic units could not be made full used in such scale. However, the performance is also 8% better than [16, 18] when the point is larger.
than 512. As this comparison are counted between our general DSPs and specialized FFT processors, which prove our addressing scheme is hardware-efficient in both structure and memories.

7 Conclusion

The design and implementation of memory address unit for conflict-free FFT are introduced in this paper. This scheme avoids the influence and conflicts of accessing to one specific bank at the same time. Instead, the data accessing pressure had been shared evenly into an independence block, which is achieved conflict-free accessing. Considering that it has no difference from ordinary transmission in the speed of the transmission process in an entire block, it has an excellent effect on recursive functions like FFT. This scheme can also be used for solving other bank conflict problems like matrix transposition and FIR. We will explore its role in future works.