Design Space Exploration of SDR Vector Processor for 5G Micro Base Stations

Wei Chen¹, Dake Liu¹, (Senior Member, IEEE), Yong Bai¹, (Senior Member, IEEE) and Rifei Yang¹
¹ State Key Laboratory of Marine Resource Utilization in South China Sea, School of Information and Communication Engineering, Hainan University, Haikou 570228, China

Corresponding author: Dake Liu (e-mail: liu.dake@outlook.com).

ABSTRACT This paper studies the design requirements and challenges of SDR (Software-Defined Radio) vector processors for 5G micro base stations. Pareto principle reflects the rule of "vital few and trivial many", which states that 80% of consequences stem from 20% of causes. Since 20% of the instructions account for about 80% of the running time of the micro base station processor, it is essential to speed up the 20% instructions as the complex vector operations consuming most of the runtime. This paper proposes instruction fusion strategy and black-box acceleration strategy to speed up the kernel function of the micro base station algorithms. The experimental results show that the instruction fusion strategy can make the performance improvement ratio reach up to 17% while running the BDTI/ EEMBC benchmark, and the black-box acceleration strategy can make the performance improvement ratio reach about 5% while running the kernel of matrix inversion. In addition, our SIMD micro architecture is designed as a vector processor to eliminate the extra costs when implementing kernel micro scheduling. This paper provides a reference for the hardware implementation of 5G micro base stations with low cost and low-power consumption.

INDEX TERMS 5G Micro Base Stations; Software-Defined Radio; Vector Processor; Instruction Space

I. INTRODUCTION

5G micro base stations are mostly used in indoor and vertical application scenarios. The focus of our research is to design a baseband architecture with high performance, low cost, and low power. Digital baseband operations include the complex vector operations, bit operations, low precision soft bit operations, and their program flow and parameter configuration. For macro base stations, the number of antennas is at least 64, and correspondingly the number of layers is 8 and baseband bandwidth is larger than 100MHz. Its MCS (Modulation and Coding Scheme) of uplink is 256QAM and MCS of downlink is 64QAM. For micro base stations, the number of antennas is not more than 4, and correspondingly the number of layers is 4 or 2, and the baseband bandwidth is 100MHz. Its MCS is 256QAM. Thus, for micro base stations, the main code behavior is its small matrix size, except for the same requirements as macro stations, such as the complex matrix operations, function operations, and the complicated data type. Redundant costs caused by the computing and addressing of small size matrix is a major challenge. Therefore, it is necessary to explore the SIMD micro architecture and vector instruction set, which is suitable for small size matrix computing and can support both subcarrier parallel and data parallel. On the other hand, there is a trivial difference between the macro station and the micro station when running bit and soft bit algorithms. There is thus no need for special discussions on bit and soft bit computing.

In industry, the vector processors for 5G base stations are all based on frameworks of general vector processing and are suitable for ultra-long and ultra-large matrices. Penta G is a 5G baseband IP platform for mobile devices and base stations developed by CEVA [1]. For 5G EMBB scenarios, Penta G provides optimized software and hardware architecture supporting data rate up to 10Gbps. In 2018, CEVA launched a digital signal processor: CEVA-XC12 [2], as an IP device for efficient implementation of 5G, Gigabit LTE, MU-MIMO WiFi and other Gigabit modems. In academia, there are hardware accelerators designed for each function in the baseband physical layer. According to specific scenarios, they can be divided into programmable accelerators that support multiple algorithms [3] and dedicated accelerators for specific
algorithms [4]. The above-mentioned 5G baseband processors are suitable for the traditional baseband algorithms of macro stations and the operations of large size matrix. However, in a micro base station, the typical feature or opportunity of baseband algorithms is its predictable small matrix sizes. The feature of the small size matrix gives opportunities to allocate matrix data in register file to reduce the cost of data accessing. The typical matrix size of these algorithms is $4 \times 4$ with complex variables of $16b+16b$. This paper focuses on exploring the SIMD micro architecture and instruction set suitable for micro base stations with $4 \times 4$ or smaller matrices to minimize the latency and accessing cost.

There are recognized vector processor researches for 3G and 4G [5,6]. The amount of 4G calculations is one-tenth of the amount of 5G calculations (where the bandwidth is one-fifth, and the time slot is one-half), so 4G does not require such high-performance design for domain specific. In addition, 3G is usually implemented by ASIC. In order to meet the performance requirements of 5G, the baseband of 5G micro base station needs to meet the requirements of low power consumption, more flexible configuration, and shorter computation delays. Hence we adopt the design method based on ASIP (Application Specific Instruction-Set Processor) to design vector processor for 5G micro stations.

The parallel use of execution units is usually out-of-order (OoO) execution hardware in modern CPUs. But the execution cycle count of OoO architecture may not be a fixed value, impacting cycle accurate requirement. OoO is for general computing and our scenario runs rather predictable algorithms. It thus not be necessary to use such an advanced OoO architecture. In addition, the power consumed by reservation station and reorder buffer in OoO architecture is very large. Hence that such an advanced OoO architecture don’t meet the requirements of real-time and lower cost for the baseband of 5G micro base stations. The research on the hardware implementation of 5G micro base station with low cost and low-power consumption is still at the initial stage. In academia and industry, there are some deficiencies or even gaps in this research at present. For micro base stations, this paper takes the lead in comprehensively researching the design of dedicated baseband symbol processor with the motivation of approaching the performance limit. This paper discusses the instruction set of processors from the application scope and algorithm coverage range, and discusses the parallel architecture of processing from the performance requirements. It has certain academic and commercial values. The research in this paper provides the guidance for the hardware implementation of 5G micro base stations in the future, which has certain significance. Scholars from Beijing Institute of Technology [7,8] have studied the design of baseband ASIC/ASIP of micro base stations for deployment in 5G UDN (Ultra Dense Networking). This paper firstly proposes the instruction fusion strategy and black-box acceleration strategy, which can achieve high processing speed and low processing delay required. This paper further proposes the SIMD micro architecture of SDR (Software Defined Radio) vector processor in 5G micro base stations and the scheme of pipeline scheduling, which is used as the solution of digital baseband system in 5G physical layer. The SIMD micro architecture and the pipeline scheduling scheme can accelerate the core algorithms of baseband physical layer with $4 \times 4$ or smaller matrices in micro base stations, while ensuring the flexibility of the architecture, reconfigurability, high throughput, low latency, low silicon cost and low-power consumption. Finally, based on the above proposed acceleration strategies and micro architecture, the vector instruction set is designed. The experimental results show that the vector processor designed in this paper can execute the kernel functions of micro base station baseband with high efficiency and low cost.

II. ACCELERATION STRATEGY

Pareto principle points out that for an application domain, we can define and decompose an instruction set to obtain a subset. 10%-20% instructions take up 80-90% of the running time. The small amount of instructions are mostly used to execute the kernel function in innermost loops. By focusing on and accelerating these instructions, we can achieve several times performance improvement with limited increase in cost.

In the baseband processing flow of micro base stations, the frequently used algorithms include channel estimation, channel equalization and beamforming algorithms, etc. The frequently used DSP kernel algorithms include the various matrix inversion algorithms, Wiener filter and other interpolation algorithms, etc. The matrix inversion algorithms mainly include the matrix inversions based on LU decomposition, Cholesky decomposition, SQRD (Sorted QR Decomposition), and SVD (Singular Value Decomposition). We focus on acceleration of kernels of core symbol algorithms that are used in baseband. According to Pareto principle, the kernel functions are extracted from the frequently used baseband algorithms, and the acceleration strategy is formulated in this section.

This paper uses the design method of ASIP based on multi-step arithmetic fusion to achieve the programming flexibility and sufficient performance through software-hardware co-design, thus greatly reducing the design costs and prolonging the life cycle of products. In order to extract the kernel functions that need to be accelerated mostly, this paper firstly profile algorithms of the baseband in micro base stations, and analyze the similarities and differences of the calculation and control parts. Then, we analyze the operation and control flow of the kernel functions to formulate the acceleration strategies: (1) instruction fusion; (2) black-box acceleration; (3) SIMD micro architecture.

A. INSTRUCTION FUSION

For the baseband processor of the micro base stations, this paper adopts the strategy of instruction fusion to fuse multiple instructions that appear jointly in the baseband algorithm many times in sequence into one instruction to execute. At the same time, the instructions before fusion are reserved to ensure
the function coverage of instruction set. The strategy makes full use of the existing data path hardware of the processor to fuse multi-step operations to form an instruction, and add as little hardware as possible to maximize the performance over cost ratio. In addition, it reduces the running time of the algorithm, and reduces the number of register accesses and the control cost.

1) ARITHMETIC OPERATIONS BASED ON COMPLEX VARIABLES

General purpose processors use a software subroutine to emulate a step of complex data computing. We add complex data type and corresponding arithmetic operations. For example, six steps of complex multiplication are fused into one step operation, and two steps of complex addition/saturation are fused into one step operation, and at the same time, the frequently-used modulus operation of complex numbers is fused into one step operation. For a 4 by 4 complex vector dot product, the fusion strategy gets the performance improvement factor of this case to be about 2.2, which is equivalent to a 14% increase in the performance of the entire PUSCH (Physical Uplink Shared Channel).

2) MULTIPLY-ACCUMULATE INSTRUCTION OF COMPLEX VARIABLES AND ITS DEEP FUSION

The traditional real data MAC (Multiply-Accumulate) operation has been widely used. In a DSP (Digital Signal Processing) processor for real data, this kind of instruction is named reduce. Micro base stations need a lot of complex data MAC operations and their deformations with high complexity. Their core operations contain similar operations, including MAC, conjugate MAC, MAC with result truncation, rounding, and saturation. Accordingly, the strategy of instruction fusion fuses multiple multiplication micro-instructions, conjugate micro-instructions, addition micro-instructions and saturation micro-instructions into one configurable MAC instruction. Before the MAC operation, the instruction completes conditional conjugation and scaling operations. After the MAC operation, the instruction completes truncation, rounding, and saturation processing. The saturation processing needs extra control flow and comparison instructions. If the conditional saturation processing is not accelerated, the efficiency can be seriously reduced. The MAC instruction designed in this paper fuses the saturation processing into the reconfiguration instruction after multiply-accumulate of complex data. Thus, at least seven complex variable instructions are fused into one instruction without increasing arithmetic hardware.

3) MATRIX ACCELERATION INSTRUCTION

Matrix manipulation is essential baseband operation with sufficient complexity. A special feature of micro base stations is its matrix sizes. The size is not more than 4×4 and it can be decomposed into the 2×2 matrix solutions. Another special feature is that all operation data of the small size matrix operation can be stored and allocated in register file, which can reduce the data access cost. These are the main opportunity in micro base stations for cost reduction and performance improvement. The strategy of instruction fusion is to fuse multiple micro-instructions into 2×2 matrix instructions. For example, 2×2 matrix multiplication is a high frequently used operation in the baseband algorithm of micro base stations. If the data has been arranged neatly in the register, we can implement 2×2 matrix multiplication in one instruction, which needs 8 multiplication instructions, 4 addition instructions, or 12 micro instructions. For a 2×2 matrix multiplication, we use one acceleration instruction to replace 12 micro instructions, our method reduce 11 instructions. Hence the acceleration instruction gets the performance improvement factor to be about 1.09 (the improvement ratio is 9% while running the kernel of matrix multiplication). In addition, the operation of 2×2 matrix multiplied by 2×2 diagonal matrix (formula is as follows) appears more frequently because a matrix with higher size can be divided to multi 2×2 matrices. We can implement this operation in one instruction. The acceleration instruction only needs one clock to complete this operation, and the unaccelerated SIMD instructions need four clocks to complete this operation, hence this acceleration instruction can save three clocks.

\[
\begin{bmatrix}
    a & b \\
    c & d
\end{bmatrix} \times \begin{bmatrix}
    e & 0 \\
    0 & f
\end{bmatrix} = \begin{bmatrix}
    a \times e & b \times f \\
    c \times e & d \times f
\end{bmatrix}
\] (1)

\[
\begin{bmatrix}
    a & b \\
    c & d
\end{bmatrix} = \begin{bmatrix}
    a & e \\
    c & e
\end{bmatrix} \times \begin{bmatrix}
    1 & 0 \\
    0 & 1
\end{bmatrix}
\]

The 4×4 matrix inversion can be simply implemented based on analytical solution of 2×2 matrix inversion. We consider a 4×4 matrix R that can be partitioned as:

\[
R = \begin{bmatrix}
    A & C \\
    C^H & B
\end{bmatrix}
\] (2)

where A, B and C are the 2×2 matrices. R is a Hermitian positive definite matrix and using the Banachiewicz formula for the inverse of a partitioned matrix [9], \(R^{-1}\) can be computed as

\[
R^{-1} = \begin{bmatrix}
    T^{-1} & -T^{-1}CB^{-1} \\
    (-T^{-1}CB^{-1})^H & B^{-1} + B^{-1}C^HT^{-1}CB^{-1}
\end{bmatrix}
\] (3)

where T is Schur complement and is defined as

\[
T = A - CBC^H
\] (4)

\[
T^{-1}\)corresponds to

\[
T^{-1} = \begin{bmatrix}
    t_{11} & t_{12}^{-1} \\
    t_{21} & t_{22}
\end{bmatrix} = \frac{1}{\det(T)} \begin{bmatrix}
    t_{22} & -t_{12} \\
    -t_{21} & t_{11}
\end{bmatrix} = \frac{1}{\det(T)} \tilde{T}
\] (5)

Paper [10] modify the Banachiewicz formula for matrix inversion to avoid division operation. In the same way, \(B^{-1}\) can be written as follows

\[
B^{-1} = \begin{bmatrix}
    b_{11} & b_{12} \\
    b_{21} & b_{22}
\end{bmatrix}^{-1} = \frac{1}{\det(B)} \begin{bmatrix}
    b_{22} & -b_{12} \\
    -b_{21} & b_{11}
\end{bmatrix} = \frac{1}{\det(B)} \tilde{B}
\] (6)

Then, considering the following change of variable

\[
T \rightarrow \det(B) T
\] (7)

and matrix T can be rewritten as follows

\[
T = \det(B) A - CBC^H
\] (8)

Formula (3) can be written as

\[
R^{-1} = \frac{1}{\det(B) \times \det(T)} \begin{bmatrix}
    \det(B) \tilde{T} & -\det(B) \tilde{T} C \tilde{B} \\
    (-\det(B) \tilde{T} CB)^H & \det(T) \tilde{T} B + \tilde{B} C^H \tilde{T} CB
\end{bmatrix}
\] (9)

The operation of (5) can be implemented by using the SIMD data path and Taylor hardware (see the section B), so that we can implement the operation of 2×2 matrix inversion in one instruction. The operations of the 2×2 matrix inversion
formula (5) can be implemented in the architecture of this vector processor, hence the code of this formula is adapted to hardware as the architectural code. Code transform adapting to our vector processor architecture are called architectural code. For the \(4 \times 4\) matrix inversion, experimental results indicate that the results from the architectural code is equal to results from the original matrix inversion algorithms, such as the matrix inversions based on LU decomposition, Cholesky decomposition, SQRD and etc. Table I shows the comparison of the truncation error between the architectural code and original algorithm using 16b data path. The matrices of the experimental data set are all \(4 \times 4\) matrices selected from small to large. Considering the convenience of experiment, we define the meaning of “small matrix” and “large matrix” as: If each row vector module of matrix \(A\) is all larger than each row vector module of matrix \(B\), and each column vector module of matrix \(A\) is all larger than each column vector module of matrix \(B\), then matrix \(A\) is relatively large, and matrix \(B\) is relatively small. The experimental results show that the truncation error of both the architectural code and original algorithm are almost the same, and this architectural code does not introduce additional accuracy loss. It proves that the Banachiewicz algorithm (5) is suitable for the architecture of this vector processor. For \(4 \times 4\) matrix inversion based on different algorithm (LU, SQRD, Cholesky), table II shows the comparison of the cycle counts between that using original method and that using the Banachiewicz formula (5). The experimental results show that this acceleration strategy make the cycle count reduction ratio reach about 58% while running the kernel of matrix inversion.

### TABLE I

| Experimental data set | Truncation error based on original algorithm | Truncation error based on the architectural code |
|-----------------------|---------------------------------------------|-----------------------------------------------|
| 100 matrices          | \(10^5\)                                     | \(10^6\)                                     |
| 200 matrices          | \(10^4\)                                     | \(10^5\)                                     |
| 300 matrices          | \(10^4\)                                     | \(10^5\)                                     |

In addition, we also design other matrix acceleration instructions, such as real number multiplied by \(2 \times 2\) complex matrix, solving the determinant of \(2 \times 2\) complex matrix, \(2 \times 2\) complex matrix multiplied by the conjugate of complex matrix, etc. When \(2 \times 2\) complex matrix multiplied by the conjugate of complex matrix, these special matrix acceleration instructions get the performance improvement factor to be about 1.11 (the improvement ratio is about 11% while running the kernel of complex matrix multiplied by the conjugate of complex matrix), because we can only use 2 acceleration instructions to complete the calculation, the general purpose processors need at least 20 instructions, our method reduce 18 instructions. The SIMD data path of the vector processor in this paper satisfies the requirements of these instructions for the data path. We can use the existing data path hardware of the processor to fuse multi-step operations to form an instruction, so that the matrix acceleration instructions generated by the fusion operation can achieve maximum performance acceleration.

The remaining acceleration instructions based on the strategy of instruction fusion also include vector addition and subtraction instruction, vector dot multiplication instruction, vector sorting instruction, and load/store vector data instruction, etc.

The main difference between an ASIP and a general-purpose processor is the acceleration in an application domain. For general computing, the application mix has been formalized and specified using general benchmarks SPEC (Standard Performance Evaluation Corporation) [11]. Adapt to our research, we select BDTI (Berkeley Design Technology Incorporation, a company supplying benchmarks of DSP processors to the DSP market) OFDM receiver benchmark [12] as the object of comparison. We also select the EEMBC (EDN Embedded Microprocessor Benchmark Consortium, another source of knowledge about DSP benchmark) TeleBench benchmark [13] as another object of comparison. Table III shows the relative comparison between the performance value of our vector processor based on instruction fusion strategy and the average value of performance according to BDTI/ EEMBC benchmark. The performance measurement is based on \(4 \times 4\) complex data matrix. The experimental results show that for our vector processor, the average performance improvement factor of these kernels is about 2.4 to 2.7 (the average improvement ratio is about 15% to 17% while running the BDTI/ EEMBC benchmark).

### TABLE III

| Kernel            | Performance improvement factor |
|-------------------|--------------------------------|
| Dot product cfp16 | 2.3-2.6 times                 |
| Modulus of vector cfp16 | 2.7-2.8 times                |
| Matrix mul cfp16  | 2.6-2.8 times                 |
| Vector reciprocal cfp16 | 2.3-2.9 times               |
| Vector min/max cfp16 | 2.5-3.0 times                |
| Matrix conjugate mul cfp16 | 2.1-2.4 times            |
| FIR cfp16         | 2.0-2.3 times                 |
| Overall average   | 2.4-2.7 times                 |

\(cfp16\) (for complex-valued 16+16-bit floating point)

### B. BLACK-BOX ACCELERATION

According to the data flow of the formalized algorithm, the strategy of black-box acceleration in this paper is to group the accelerated operations or data transferring set for the most frequently used arithmetic to form a new data path or part of the data path. This data path is completely different from the
The data path of the general SIMD and cannot be integrated into the SIMD, so it is called a black box. Taylor series hardware is one black-box acceleration method, and this paper mainly takes Taylor hardware as an example. At present, there are many methods to implement the reciprocal. For example, Newton-Raphson method is used in the paper [14]. This method requires relatively less storage resources, but if the result needs to achieve a certain precision, multi-level iteration is needed, which consumes more computing resources. The SRT(4) algorithm is also used to implement the reciprocal in the paper [15], which consumes much resources, including multiple integer subtractions and shifts, and there is not much room to improve this operating frequency. In view of the above deficiencies, such as large resource consumption, low execution efficiency and slow speed, this paper uses the strategy of black-box acceleration to optimize this operation, that is to use the Taylor series to implement the reciprocal. Meanwhile, Taylor is also used to implement the square root, tangent, arc tangent, sine, cosine, Zadoff-Chu sequence, and the corresponding acceleration instructions are designed. This paper uses the sixth-order Taylor to implement these operations, and the expanded Taylor series expression is:

\[ Y = a_0 + a_1 \cdot x + a_2 \cdot x^2 + a_3 \cdot x^3 + a_4 \cdot x^4 + a_5 \cdot x^5 + a_6 \cdot x^6 \]  

where \( a_0, a_1, \ldots, a_6 \) are polynomial estimation coefficients associated with Taylor series hardware. We align the operators that cannot be integrated into the SIMD data path, which include the first calculation of \( a_1 \cdot x \) and \( x^2 \), the second calculation of \( x^3 \) and \( x^4 \), and the third calculation of \( x^5 \) and \( x^6 \), and then these aligned preprocessing operations are formed as the output of the black-box. The fourth calculation is to use the SIMD data path to complete the MAC operation. In this paper, the hardware of SIMD data path is reused as much as possible to ensure that the formed black box hardware is minimal. At the same time, in firmware, the Taylor operation is fine tuned into repeat mode for running multiple times, which can hide extra three pipelines of the preprocessing operation in the Taylor, and shorten the clock cycle of Taylor operation to 1 or 2 clocks. The whole process makes the black-box hardware the least but also achieves the maximum acceleration.

The data path of Taylor operation is shown in Figure 1. The SIMD data path of this vector processor consists of 16 multipliers and 14 adders. In this paper, part of Taylor computing hardware (purple, left) is shared with the SIMD computing hardware (green, right) to reduce the cost.

![FIGURE 1. The data path of Taylor operation](image)

Table IV shows the hardware overhead of parallel Taylor operation with shared hardware, as well as the cycle counts and result error of each operation within the specified value range. For the operations outside this value range, this paper uses different polynomial estimation coefficients to calculate the Taylor expansion. The experimental results show that the strategy of black-box acceleration can reduce the cycle counts from five clocks to one or two clocks. Within the specified value range accepted by the corresponding baseband algorithms, the result error of each Taylor series is 0.006% on average. We use the strategy of black-box acceleration to reduce the overall hardware overheads of the processor by maximizing the sharing of SIMD hardware, and reduce the clock overhead.

The performance of the Taylor operation

| Operation          | The hardware overhead shared with SIMD | The cycle counts after sharing | Result error |
|--------------------|----------------------------------------|-------------------------------|--------------|
| Reciprocal         | Four parallel addition operations can share 8 multipliers and 12 adders at once (8/32768<<x<<2^m1) | 1-2 clocks                    | 0.0056%      |
| Square root        | Three parallel square root operations can share 12 multipliers and 14 adders at once (8/32768<<x<<2^m1) | 1-2 clocks                    | 0.0052%      |
| Tangent            | Seven parallel tangent operations can share 14 multipliers and 14 adders at once (x<<π/2,001) | 1 clock                       | 0.0063%      |
| Arc Tangent        | Seven parallel arc tangent operations can share 14 multipliers and 14 adders at once (x<<π/2,001) | 1 clock                       | 0.0061%      |
| Sine               | Seven parallel sine operations can share 14 multipliers and 14 adders at once (x<<π/2,001) | 1 clock                       | 0.0069%      |
| Cosine             | Seven parallel cosine operations can share 14 multipliers and 14 adders at once (x<<π/2,001) | 1 clock                       | 0.0065%      |
| Zadoff-Chu         | Two parallel Zadoff-Chu sequence operations can share 10 multipliers and 12 adders at once (sequence length \( N_s \) ≈ 2048) | 2 clocks                      | 0.0071%      |

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3119292, IEEE Access
For the operation of the reciprocal, Table V shows the cycle counts comparison between Taylor and the division algorithm running in software. In addition, our Taylor method based on black-box acceleration strategy is rather general, which can run Horner method or Estrin method. Based on Horner method or Estrin method, the polynomial of Taylor is converted to nested forms, that is addition/subtraction operation and multiplication operation are executed repeatedly in turn. The addition/subtraction operation can be implemented by using the first-level adder in data path of Taylor operation, and the multiplication operation can be implemented by using the first-level multiplier in data path of Taylor operation, and the following iterative operations reuse these adders and multipliers in turn until the polynomial is solved. Taylor operation is high frequently used operation in running baseband symbol algorithm. We often solve the reciprocal/square root/tangent/arc tangent/sine/cosine for the same complex variable. If we modify codes and collect a Taylor algorithm sequentially running together, and these operations can share the result of preprocessing operation in the Taylor hardware. We tune Taylor operation into repeat mode for running multiple times, which can hide extra three pipelines of the preprocessing operation in the Taylor hardware. When Taylor operations are executed multiple times, the average clock counts can reach 1 to 2 clocks. Meanwhile, when the kernel micro scheduling is used for multi subcarriers, the Taylor execution can be collected together and the proposed method does not consume extra clock cycles. It can be performed as running a single clock instruction. The performance measure is based on prolog cost and epilog cost. Before and after the calculation of the main part of the kernel function, there may be the calculations of non-holonomic vectors, and the data flow structure is triangular, which is called prolog and epilog. This part of the extra cost is recorded as the prolog cost and epilog cost. When the number of consecutive Taylor operations is not large enough, the main cost comes from prolog and epilog, hence under this circumstances, the cycle count of Taylor operation is 5 clocks. For the matrix inversion, we compare our Taylor division method with iterative division method [14,15], the overall average performance improvement factor of matrix inversion kernel is about 1.05 (the average improvement ratio is about 5% while running the kernel of matrix inversion). Through comparison (as shown in Table V), the method in this paper has the lowest hardware overhead and the least clock overhead under the same accuracy, and the advantage of black-box acceleration strategy is obvious.

| Algorithm   | Hardware overhead | Cycle counts |
|-------------|------------------|--------------|
| Taylor      | 3 multipliers (expect the hardware shared with SIMD) | 1-2 clocks |

Table V: The Comparison of Overhead Between Taylor and the Division Algorithm Running in Software

C. MICRO ARCHITECTURE

The SIMD micro architecture of this vector processor consists of control logic (green), a memory subsystem (red), and a data path (blue) as shown in Figure 2. The control logic is composed of PC_FSM (PC Finite State Machine), DMA (Direct Memory Access), PM (Program Memory) and Inst_dec (Instruction Decoder). The memory subsystem is composed of AGU (Address Generation Unit), data memory, register file, RP (Read Permutation Network) and WP (Write Permutation Network). The data path handles the baseband algorithms targeted in this paper in parallel. The outputs of the data path will be transferred to the WP and then written to data memory or register file. The read permutation network and write permutation network are designed for data reordering to ensure that vector data can be obtained in the required order of data in parallel. The SIMD processor model is shown in Figure 3. The vector length of SIMD processor is 4. The SIMD processor contains 4 groups of vector register files, each vector register file size is 512 bits, the total vector register files size is 2048 bits. The number of LD/ST units is 1, because we have designed a dedicated processor for fast Fourier transform (FFT) algorithm, FFT module is not included in our vector processor, a LD/ST unit is enough for our processor. The supported data types include 16bit real number, 32bit real number, 16bit complex number and 32bit complex number.

The acceleration instructions generated by the instruction infusion and black-box acceleration strategies will increase the complexity of the pipeline. In order to solve this problem and make the acceleration instructions can be realized in pipelined
modules to approach the efficiency limit of the data path and the memory bandwidth. As shown in Figure 4. The vector processor contains at most 10 pipeline stages. Firstly, one instruction will be read out from the PM during IF (Instruction fetch). Then, during ID (Instruction Decoding), the fetched instruction will be decoded and the address of the operand will be generated. During Mem (Memory access), the source operand will be acquired and transferred to the read permutation network. During Perm (Permutation), the operands will be reordered in the read permutation network according to the requirements of the implemented algorithm, and then the reordered operands are sent to the data path. According to the requirements of different instructions, the data path of the vector processor consumes 1 to 5 pipeline stages. The 1 to 3 pipeline stages correspond to the data path of Taylor hardware, and the 4 to 5 pipeline stages correspond to the data path of SIMD. In order to ensure that the data path can work normally, we design some logic for buffering control signals and destination operands, control signals are used to select which pipeline stage of the data path should be executed. During the pipeline stage WB (Write Back Results), the results will be stored. We can map algorithms to the proposed architecture to speed up the calculation. For the 2x2 matrix inversion algorithm (5), we firstly compute \( \det(T) \), that is to compute \( t_{11} \times t_{22} - t_{21} \times t_{12} \). Mapping to the corresponding hardware of SIMD architecture, this operation is implemented by using the hardware of SIMD data path, which include the multipliers, the first-level adder and the second-level adder. Then, we compute \( 1/\det(T) \), this reciprocal operation is implemented by using the Taylor hardware and the hardware of SIMD data path. Finally, we compute \( \frac{1}{\det(T)} \begin{bmatrix} t_{22} & -t_{12} \\ -t_{21} & t_{11} \end{bmatrix} \), that is to compute complex scalar multiplied by complex vector, this operation is implemented by using the multipliers and the first-level adder in SIMD data path. The functional configuration of the first-level adder includes: addition, subtraction, arithmetic negation, minimum value, maximum value, left shift, right shift, absolute value, positive and negative, left pass-through and right pass-through, hence the first-level adder is used to transform \( 1/\det(T) \times t_{12} \) and \( 1/\det(T) \times t_{11} \) into \( -1/\det(T) \times t_{12} \) and \( -1/\det(T) \times t_{11} \). Our proposed architecture can maximize the utilization rate of SIMD data path hardware, and the different operations of kernel algorithms can be implemented in the SIMD data path, which can reduce the overall hardware overheads of the processor by maximizing the sharing of SIMD data path hardware. Meanwhile, we propose instruction fusion and black-box accelerate instructions. We demonstrate the execution for one OFDM (Orthogonal Frequency Division Multiplexing) subcarrier with \( 4 \times 4 \) matrix, and multi subcarriers can be executed in parallel by expanding the parallelization of the SIMD. In addition, the parallel access mode of our SIMD micro architecture hides the time overhead of load/store, and our pipeline scheduling scheme of vector processor can let the different operations realized in pipeline to approach the efficiency limit of the data path. Hence when the algorithm is mapped to our architecture, the optimization results is obvious.

III. INSTRUCTION SET

Table VI summarizes the basic calculation mode and the average overhead of operators in the kernel functions of all
core symbol processing algorithms. Based on the analysis of the above overhead of operators and the acceleration instructions generated by the above acceleration strategies, the instruction set of the vector processor designed in this paper are shown in the table VII.

TABLE VI

| Scalar operators/ ratio | Vector operators/ ratio |
|------------------------|------------------------|
| Multiply / 10.5%        | Vector add scalar / 0.6%|
| Division / 10%          | Vector multiply scalar / 8%|
| Square root / 4%        | Vector dot product / 2%|
| Compare / 3%            | Vector modulus square / 3%|
| Look up table / 0.6%   | Vector modulo square accumulation / 8%|
| Absolute value / 0.4%  | Matrix multiplication / 3%|
| Take symbol / 0.9%     | Vector multiplication accumulation / 16%|
| Trigonometric function / 1% | Vector min / max / 2% |

IV. THE LIMIT OF SIMD MICRO ARCHITECTURE

In this paper, the limit of SIMD micro architecture of vector processor in micro base stations is defined as: in an ideal case, a parallel hardware architecture with a parallelism of N should read 2N operands in each clock cycle. Then the data path performs the calculation and consumes the fetched operands to produce effective output. The limit was defined by David Kuck [16]. Using this definition, in micro base stations, the minimum number of clocks of the baseband algorithm in parallel architecture can be deduced theoretically, which can be used as the reference standard for performance evaluation.

The limit number of clock cycles of these algorithms deduced from the above theory is used as the evaluation standard of the best implementation performance (0 extra cost). The experimental results are shown in the table VIII.

The limit number of clock cycles of the kernel function defined in this table is calculated under the condition that the data can be accessed in parallel without additional control overhead for each computation. The actual execution cycle cannot reach the defined limit due to the extra cost, which includes the extra cost of the data path, addressing and control. The extra cost of pipeline is caused by the vector data dependencies that exist in the pipeline of data path, which only includes the costs generated during the calculation of the kernel function. The addressing register needs to be configured before calculation, which called the cost of addressing preparation. If these costs cannot be hidden in pipelined parallel execution of the parallel architecture during the operation, the extra cost will be generated. The extra cost of control includes jumps, etc. The extra cost of prolog and epilog is recorded as the pro-epi-log cost. In order to approach the theoretical limit, this paper uses the proposed scheme of efficient pipeline scheduling and the acceleration strategies of instruction fusion and black-box to eliminate these extra costs. The ratio of extra costs is calculated by the formula: (actual execution cycle - theoretical limit) / actual execution cycle. According to the experience of skilled engineers, the ratio of extra costs of general DSP can be more than 300%, while this paper can reach 17.8% on average (the lowest can reach 16%).

The experimental results show that the performance difference between the vector processor in this paper and the limits of definition is small. It proves that this vector processor can achieve high performance for the small size kernel function of micro base stations in the field of wireless communication. In the future, we will continue to shorten the distance from the limits of definition, which mean that we will continue to approach to the limit of SIMD micro architecture of vector processor in micro base stations [16].

For CEVA-XC12 processor, the cycle count of matrix inversion module is about 40.1% of symbol processing while running the kernel of data symbols demodulation [17].

TABLE VII

| Instruction classification | The function of the instruction |
|---------------------------|--------------------------------|
| Real/complex vector operation | Vector addition / subtraction / dot product / multiplication accumulation / conjugate multiplication and accumulation / modulus square / 2×2 matrix inversion and other 2×2 matrix operations, etc. |
| Scalar operation | Scalar addition / subtraction / multiplication / absolute value / comparison, etc. |
| Get max / min value | Taking the max/ min value of the vector |
| Data reordering | Vector reordering |
| Special acceleration | Special matrix acceleration / Taylor / LUT / linear detection / interpolation and fitting |
| Data transferring | Data transferring between SRAM / register files, scalar/vector data transferring |
| Control | No operation, loop and jump, etc. |
| Data type conversion | 16/32bit data conversion, fixed-point / floating-point data conversion |

TABLE VIII

| Kernel function | Scale | Extra cost | Execution cycle | The ratio of extra costs |
|-----------------|-------|------------|-----------------|--------------------------|
|                 |       | Pipeline   | Address preparation | Control | Pro-epi-log | Theoretical limit | Actual cycle |                           |
| Matrix inversion based on SVD | 4×4 | 1 clock | 2 clocks | 2 clocks | 1 clock | 29 clocks | 35 clocks | 17%|
| Matrix inversion based on ASORD and other functions | 4×4 | 1 clock | 3 clocks | 1 clock | 1 clock | 27 clocks | 33 clocks | 18%|
| Matrix inversion based on Cholesky decomposition | 4×4 | 1 clock | 2 clocks | 1 clock | 1 clock | 25 clocks | 30 clocks | 16%|
| MMSE (no including the call of Cholesky) | 4×4 | 1 clock | 3 clocks | 1 clock | - | 20 clocks | 25 clocks | 20%|
same matrix inversion algorithm, the vector processor of this paper uses the above acceleration strategies and efficient pipeline scheduling scheme to ensure the highest efficiency without NOP (No Operation), which reduces the cycle count of matrix inversion operation about 20.4%. Compared with CEVA-XC12, the cycle count of matrix inversion in this paper is reduced to a certain degree, it proves that this vector processor has higher performance.

V. EVALUATE THE HARDWARE OVERHEAD
We have implemented our proposed design in a 28nm CMOS technology. The gate count of our vector processor is 194k, and the gate count of the extra hardware is 32K. The acceleration strategy of instruction fusion makes the performance improvement ratio reach up to 17% while running the BDTI/EEMBC benchmark. The acceleration strategy of black-box can make the performance improvement ratio reach about 5% while running the kernel of matrix inversion. The proposed scheme of pipeline scheduling makes the vector instructions can be realized in pipelined modules to approach the efficiency limit of the data path. We compare the overall performance of the vector processor in this paper and the other processors designed in related papers (as shown in the table IX). The experimental result shows that the NHE (Normalized Hardware Efficiency) of this vector processor is higher than other processors, which proves that the processor of this paper is better than other processors in the balance of the performance and hardware overhead. Power consumption analysis is very difficult because of insufficient information. For other processors, we don't know the size of SRAM and register files, and the internal precision and DSP data precision during iteration, and whether the bus power consumption has been calculated. Listed information items are usually not published, or the published information is incomplete. Hence we cannot make a strict comparison. The power consumption of our vector processor is 61 mW, the power consumption is measured on 0.9V supply voltage at the clock frequency of 500MHz. By analyzing the power consumption in these literatures [22,23,24,25,26,27], we find that our power consumption is lower than the power consumption in these literatures. All this proves that this vector processor can reduce both the power consumption and overall overhead to a certain extent on the basis of meeting the functional, performance, flexibility and latency requirements of 5G micro base stations. This paper provides a more flexible acceleration scheme with high speed and low cost for micro base stations.

In this paper, we firstly analyze the operation flow of core symbol processing algorithms, and compare the similarities and differences of the calculation and control parts, so as to extract the kernel function which needs to be accelerated most. Then we propose the strategy of instruction fusion to form acceleration instruction set to accelerate the extracted kernel function, our designed acceleration instruction set provides efficient acceleration support for all core baseband symbol algorithms. Some other DSP processors accelerate instructions for a kind of baseband algorithm, which may not be able to accelerate another kind of baseband algorithm. Meanwhile, we make full use of the existing data path hardware of the processor to fuse multi-step operations to form acceleration instructions, and we add trivial hardware to maximize the performance over cost ratio. In addition, it reduces the running time, and reduces the number of register accesses and the control cost. Some other DSP processors add some hardware to form some acceleration instructions, which increase extra hardware overhead. In addition, we propose the strategy of black-box acceleration is to group the accelerated operations or data transferring set for the most frequently used algorithms to form a new data path or part of the data path. This data path is completely different from the data path of the general SIMD. For example, we make full use of data path hardware to form the data path of Taylor operation (only adding the Taylor hardware for preprocessing), which can implement the reciprocal/ square root/ tangent/ arc tangent/ sine/ cosine/ Zadoff-Chu sequence operations. We add black-box acceleration to reduce the overall hardware overheads of the vector processor by maximizing the sharing of SIMD hardware. The arithmetic unit of some other DSP processors provides a high-efficiency acceleration support for a kind of operation, which may become unnecessary redundant overhead for another kind of operation. Moreover, we propose the efficient and flexible SIMD micro architecture, which can make full use of the parallelism of the data path and support the parallel access mode to hide the time overhead of load/store. In addition, we propose a scheme of pipeline scheduling which makes the execution of vector instructions in pipelined modules to approach the efficiency limit of the data path. Some other DSP processors cause unnecessary idle or wait in the program pipeline, this additional overhead is mainly due to the data dependence of the algorithm itself, or the control dependence or data dependence caused by the unreasonable program/instruction arrangement. Above all, we propose innovative and specific design strategies for the instruction set, SIMD data path, SIMD micro architecture, and pipeline scheduling scheme, the experimental results indicate that our method can achieve the goal of reducing additional overhead and approaching the performance limit of SIMD micro architecture.

VI. CONCLUSION
Micro base stations play a very important role in the construction of 5G network. Currently, in academia and industry, there is a lack of the research on the hardware implementation of 5G micro base stations with low cost and low-power consumption. With the above motivation to reduce the redundancy overhead and improve the reuse efficiency of hardware for approaching the theoretical limit as much as possible, we propose some innovative and effective method: the acceleration strategies of instruction fusion and black-box, the efficient pipeline scheduling scheme, the efficient SIMD micro architecture. Firstly, we use instruction fusion strategy
to accelerate some existing SIMD instructions in the present major vector processors’ instruction set. The experimental results show that these acceleration instructions make the performance improvement ratio reach up to 17% while running the BDTI/ EEMBC benchmark. Then, we use black-box acceleration strategy to form Taylor instructions, which are unique in the present major baseband processors’ instruction set. The experimental results show that for a kernel of matrix inversion, Taylor instructions can make the performance improvement ratio of matrix inversion kernel reach about 5% by making full use of data path hardware. In addition, our proposed scheme of pipeline scheduling makes the vector instructions can be realized in pipelined modules to approach the efficiency limit of the data path. Moreover, we propose the efficient and flexible SIMD micro architecture, which can support the parallel access mode to hide the time overhead of load/store. The experimental results indicate that the hardware redundancy ratio of our vector processor can reach 17.8% on average (the lowest can reach 16%), and the NHE (Normalized Hardware Efficiency) of this vector processor is significantly higher than the baseband processors of other references, and our power consumption is lower than the baseband processors of other references. All these prove that when this vector processor executes the kernel functions of micro base station baseband, the performance is greatly improved and the extra cost of symbol processing is greatly reduced. This paper provides the guidance for the hardware implementation of 5G micro base stations with low cost and low-power consumption, which has certain significance.

REFERENCES

[1] Mike, Demler, “Ceva PentaG Adds AI to 5G Baseband,” Microprocessor Rep., vol. 32, no. 3, pp. 18-20, Mar. 2018.

[2] David, Kanter, “Ceva Telegraphs 5G Intent with XC12,” Microprocessor Rep., vol. 31, no. 4, pp. 25-27, Apr. 2017.

[3] Chen X, Minwegen A, Hussain S B, et al, “Flexible, Efficient Multimode MIMO Detection by Using Reconfigurable ASIC,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 10, pp. 2173-2186, Oct. 2015.

[4] Y. Lee, C. Wang, and Y. Huang, “A Hybrid RF/Baseband Precoding Processor Based on Parallel-Index-Selection Matrix-Inversion-Bypass Simultaneous Orthogonal Matching Pursuit for Millimeter Wave MIMO Systems,” IEEE Trans. Signal Process., vol. 63, no. 2, pp. 305-317, Jan. 2015.

[5] Wu D, Melfh C, Simko M. “Implementation Aspects of Channel Estimation for 3GPP LTE Terminals,” in Proc. Ear. Wireless. Sustain. Wireless Technologies, Apr. 2011., pp. 440–444.

[6] R. Fasthuber, P. Raghavan, L. Van der Perre and F. Cathoor, “A Scalable MIMO Detector Processor withNear-ASIC Energy Efficiency,” IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 23, no. 10, pp. 1973-1986, Oct. 2015.

[7] Cai Z, Liu D, “Baseband design for 5G UDN base stations: Methods and implementation,” China Commun., vol. 14, no. 5, p. 59-77, May. 2017.

[8] W. Qiao, D. Liu and S. Liu, “QFEC ASIC: A Flexible Quad-Mode FEC ASIC for Polar, LDPC, Turbo, and Convolutional Code Decoding,” IEEE Access, vol. 6, pp. 72189-72200, Nov. 2018.

[9] Meyer, Carl, Matrix analysis and applied linear algebra. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2000.

[10] Safari A, Fakhriaie S M, Abbassfar A, “Algorithm and FPGA implementation of interpolation-based soft output MMSE MIMO detector for 3GPP LTE,” IET Commun., vol. 8, no. 4, pp. 492-499, Mar. 2014.

[11] SPEC. The Standard Performance Evaluation, Accessed: Jun. 2021. [online]. Available: http://www.spec.org/.

[12] BDTI. bdit-ofdm-receiver-benchmark. Accessed: Jun. 2016. [online]. Available: https://www.bdti.com/services/bditi-ofdm-receiver-bench mark.

[13] EEMBC. TeleBench. Accessed: Feb. 2020. [online]. Available: http://www.eembc.org/telebench/.

[14] M. Joldes, J. Muller and V. Popescu, “On the computation of the reciprocal of floating point expansions using an adapted Newton-Raphson iteration,” in Proc. IEEE ASAP, Jun. 2014, pp. 63-67.

[15] Chen J, Peng Y, Lei Y, et al, “Design of double-precision floating-point division and square root based on SRT algorithm,” in Proc. IEEE CIST, Jan. 2014, pp. 61-67.

[16] D. J. Kuck, “What Is Good Parallel Performance? And How Do We Get It?,” IEEE Comput. Sci. Eng., vol. 3, no. 1, pp. 81-85, Feb. 1996.

[17] CEVA. CEVA-XC12 5G gNB Architecture for small cell. Accessed: Aug. 2018. [online]. Available: https://www.ceva-dsp.com/product/ceva-xc12.

[18] Lee L, Chen C A, Chen C E, et al, “Square-Root Generalized Eigenvalue Decomposition Processor for Leakage-Based Multi-User MIMO Precoding with Multi-Antenna Users,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 66, no. 6, pp. 2382-2393, June. 2019.

[19] C.-A. Chen, C.-E. Chen, and Y.-H. Huang, “Forward-substitution-based generalized eigenvalue decomposition processor for MIMO precoding,” in Proc. IEEE SIPS, Oct. 2017, pp. 1–4.

[20] Y.-C. Tsai, C.-E. Chen, and C.-H. Yang, “A flexible geometric mean decomposition processor for MIMO communication systems,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 64, no. 2, pp. 446-456, Feb. 2017.

[21] C.-F. Liao, F.-C. Lan, J.-W. Jiang, and Y.-H. Huang, “A 576-mbit/s 64-OFDM 4-4 MIMO precoding processor with lattice reduction,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 61, no. 2, pp. 95–99, Feb. 2014.
[22] Z. Liang et al., "A 3.85-Gb/s 8 × 8 Soft-Output MIMO Detector with Lattice-Reduction-Aided Channel Preprocessing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 2, pp. 307-320, Feb. 2021.

[23] Z. Yan, G. He, Y. Ren, W. He, J. Jiang, and Z. Mao, “Design and implementation of flexible dual-mode soft-output MIMO detector with channel preprocessing,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 11, pp. 2706–2717, Nov. 2015.

[24] C. Zhang, L. Liu, D. Markovic, and V. Owall, “A heterogeneous reconfigurable cell array for MIMO signal processing,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 62, no. 3, pp. 733–742, Mar. 2015.

[25] K. Chen, Y. Hwang and Y. Liao, "VLSI Design of a High Throughput Hybrid Precoding Processor for Wireless MIMO Systems," *IEEE Access*, vol. 7, pp. 85925-85936, Jun. 2019.

[26] R. Damodaran et al., “A 1.25GHz 0.8W C66x DSP Core in 40nm CMOS,” *in Proc. IEEE VLSID*, Jan. 2012, pp. 286–291.

[27] Y.-Y. Lee, C.-H. Wang, and Y.-H. Huang, “A hybrid RF/baseband precoding processor based on parallel-index-selection matrix-inversion-bypass simultaneous orthogonal matching pursuit for millimeter wave MIMO systems,” *IEEE Trans. Signal Process.*, vol. 63, no. 2, pp. 305–317, Jan. 2015.