An Implementation Method of HEVC Dataflow Graph Based on Reconfigurable Processer

Yun Zhu*[1], Chuanzhan Hu*, Lin Jiang†, Xubang Shen††
* Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: zhuyun@xupt.edu.cn, 616545219@qq.com
†Xi'an University of Science and Technology, Xi'an, China
E-mail: jianglin@xust.edu.cn
††Xi'an Microelectronic Technology Research Institute, Xi'an, China
E-mail: shenxubang@163.net

***Abstract*— Aiming at the problems of low resource utilization and long algorithm execution time caused by unreasonable task division when complex applications in high-efficiency video coding are mapped on a reconfigurable processor. This paper proposes a method for implementing a dataflow graph (DFG) of HEVC algorithm based on a reconfigurable processor. According to analyzed the data dependence and parallelism between the algorithms, the loop optimization is performed on the HEVC part with more loops and the algorithm DFG is constructed. This paper mainly analyzes the mapping of smaller-scale DFG to different processing units in the array processor. The experimental results show that compared with the serial scheme, the speedup ratio can reach up to 7.4x, the processing element utilization can be increased by up to 80%, and the execution time can be reduced by up to 86%.**

Keywords—HEVC, Loop optimization , DFG, Parallel mapping

I. INTRODUCTION

The realization of video coding technology relies on the most basic mathematical operations, loop jumps, and memory operations, so corresponding software and hardware support is required. High-efficiency video coding (HEVC) has the characteristics of high computational complexity and multiple cycles, and has high requirements for real-time coding. For such a high-complexity calculation, only algorithm optimization at the software level can no longer meet the needs of real-time video coding. Although traditional general-purpose processors are flexible, their overall performance is not as good as Application Specific Integrated Circuit (ASIC). ASIC has the highest performance-to-power ratio and the smallest footprint, but it has a longer development cycle and poor flexibility[1]. However, the dynamically programmable and reconfigurable array processor has the characteristics of high parallelism and flexible programming[2], which can meet real-time requirements and effectively improve coding efficiency.

When traditional manual mapping of high-efficiency video coding algorithms, existing computing resources cannot be fully utilized, and some resources are idle during calculation. Without fully considering the characteristics of the array structure, the traditional manual mapping of the HEVC algorithm has frequent data interactions and long calculation time[3]. In addition, the existing computing resources cannot be fully utilized, some resources are idle during the calculation process, and the system performance is mediocre [4][5].

Dataflow graph (DFG) graphically depicts the process of dataflow and processing in the system. Each node corresponds to a kind of operation, and each edge corresponds to the input and output between nodes[6][7]. The dependence of data is represented by the directed edge of DFG. Each processing element in the array processor can be regarded as an operator. The structure of the input, calculation, and output nodes of the DFG graph corresponds to the structure of the input buffer, output buffer, and processing element of the array processor. Therefore, each connection relationship between the nodes can be converted to the connection relationship between the array operation nodes, giving full play to the high parallelism of the array processor[8].

Based on the array processor developed by the project team, this paper proposes a mapping method based on the data flow graph. By analyzing the data dependence and parallelism in the HEVC algorithm, a data flow graph is constructed, and a smaller-scale data flow graph is mapped through nodes.

The remainder of the paper is organized as follows: Section II analyzes the data dependence and parallelism of the HEVC algorithm, and introduces the hardware implementation plat. Section III optimizes the loop part of the HEVC algorithm, and then, a DFG is constructed for the optimized HEVC algorithm. In Section IV presents the implementation results and analyzes in detail. Finally, Section V summarizes the full text.

II. RELATED WORKS

A. Reconfigurable array processor

The array processor used in the experiment is composed of 1,024 processor elements (PEs) in the form of adjacent interconnection. Figure 1 shows part of the 4×4 PEs, which is based on the H-Tree reconstruction mechanism (HRM), layer configuration network and a global controller. The PE is

[1] Corresponding author.

implemented by a four-stage pipeline of extraction, decoding, execution and write-back. The array processor logically divides the 4×4 PEs into processing element group (PEG). Global network communication mechanism based on local shared memory. The data interaction in the cluster adopts adjacent interconnection and shared storage. The on-chip network uses inter-cluster communication. HRM provides a solution for realizing the dynamic reconstruction of different algorithms. The global controller determines the operation mode and selects the appropriate function of one or more PEs, and then unicasts the reconstruction configuration information to the HRM.

The mapping of the HEVC algorithm in this paper is implemented on this array structure.

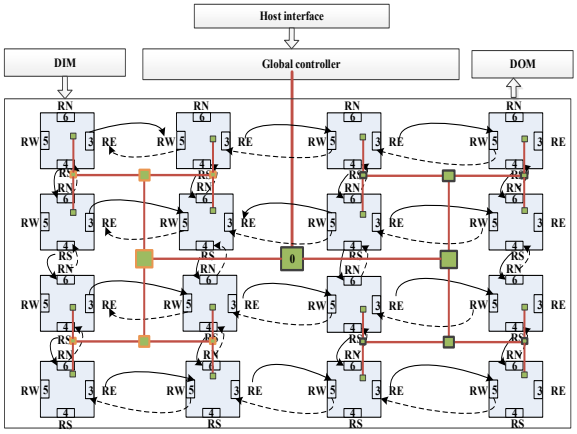


Fig.1 Reconfigurable array processor

B. *Analysis of Data Parallelism of HEVC*

The HEVC coding framework is shown in Figure 2. The prediction process is mainly divided into intra-frame prediction and inter-frame prediction. The image value of the coded block is predicted by the pixel information around the coded block. In order to run high-performance HEVC on the array processor, first perform a parallel analysis of the core algorithms in HEVC.

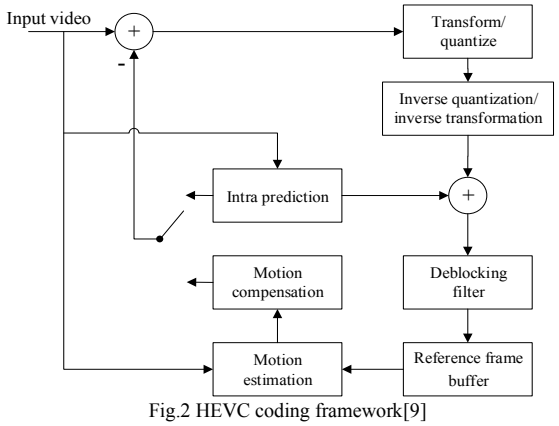


Fig.2 HEVC coding framework[9]

Compared with the H.264 algorithm, the HEVC algorithm adds a flexible quadtree division[10], and uses the coding tree unit (CTU) to replace the H.264 macro block. The image is flexibly divided into coding block sizes from 64×64 to 4×4, the prediction modes of the intra prediction algorithm are increased from 13 to 35 prediction modes, and the motion compensation filter with higher tap coefficients and more precise is used between frames. But it brings higher coding complexity.

In the HEVC standard, a 64×64 coded block is divided into 849 prediction units (PU). The intra prediction algorithm selects the optimal mode after traversing 35 prediction modes for prediction units of different sizes. There is no data dependency between each mode. So, the calculation of each prediction mode in the intra prediction algorithm can be calculated in parallel.

The sub-pixel interpolation calculation of inter-frame motion estimation is for pixels at different positions. The reference block pixel values and interpolation calculation formulas used are different. The larger the PU block, the more interpolation pixels that need to be calculated, and the more complicated the sub-pixel positions that need to be processed. The PE in the array processor has parallel computing capability and can complete the interpolation calculation of multiple pixels in the same clock cycle.

III. DFG MAPPING FOR HEVC

A. *Pretreatment: Loop optimization*

The longest running time of HEVC encoding is the loop number and loop calculation process. Before constructing DFG, the loop body in the algorithm should be optimized first. There are three ways to optimize loops: loop unrolling, loop separation, and loop tiling. Different algorithms select the corresponding loop optimization method respectively. The three methods are described below.

1) Loop unrolling

Loop unrolling refers to unrolling part of the loop. The iteration is expanded to each statement in the loop body, which can reduce the number of loops[11][12]. Fully unrolling the loop can maximize the parallelism of operators, reduce conditional judgments at the end of each iteration, concentrate more operations together, perform unified calculations, and map this to a reconfigurable array, which can reduce the number of times to reconfigure the array.

Taking the SAD algorithm in the HEVC as an example, the specific calculation is shown in formula (1). Where $f_k(m, n)$ is the pixel value of the brightness block in the current frame, and $f_{k-1}(m+i, n+j)$ is the pixel value of the brightness block in the reference frame.

$$SAD_{(i,\ j)} = \sum_{i=1}^{m}\sum_{j=1}^{n} | f_k(m,\ n) - f_{k-1}(m+i,\ n+j) | \qquad (1)$$

In the process of comparing the SAD value, the 64×64 coded block is split into 256 4×4 prediction blocks. This article uses the smallest block 4×4 as an example to implement the SAD algorithm. Figure 3 (a) shows the SAD algorithm assembly code. Every time the value of SAD is

calculated, it is necessary to determine whether the calculation is completed with 16 pixels, and the algorithm is still executed serially after the determination is completed. The original code loop body of the SAD algorithm contains only one subtraction and one absolute value operator operation, only 2 of the 16 PEs are working, and the remaining 14 PEs are in idle state, the actual utilization of PE is only 12.5 %.

Unrolling the calculation process of 16 cycles in the source code, different PEs can perform SAD calculations at the same time, that is, the absolute difference between the current frame and the reference frame of 16 pixels can be calculated in the same clock cycle. The code after loop unrolling is shown in Figure 3(b), 8 of the 16 PEs work at the same time, and the PE utilization rate is 50%. If the SAD value of an 8×8 code block is calculated, 16 PEs can be calculated at the same time, and the PE utilization rate can reach 100%.

```
ADDI R2,R0,#300,#PE00
ADDI R6,R0,#300,#PE01
ADDI R7,R0,#0
ADDI R1,R0,#16
JISUAN: LD R3,R2
        LD R11,R6
        SUB R13,R3,R2
        ABS R13,R13
        ADD R7,R13
        ADDI R2,R2,#1
        ADDI R6,R6,#1
        SUBI R1,R1,#1
        BNE R1,R0,JISUAN
STOP:NOP
```

(a) Source code

```
ADDI R2,R0,#300,#PE00
ADDI R6,R0,#300,#PE01
LD R3,R2
LD R11,R6
SUB R13,R3,R2
ABS R13,R13
ADDI R2,R2,#1
ADDI R6,R6,#1
STOP:NOP
```

(b) after loop unrolling

Fig.3 SAD loop unrolling implementation

During the execution of the HEVC, many algorithms require loop unrolling. Taking an 8×8 coding block as an example, Table 1 shows the comparison of coding time before and after loop unrolling of some HEVC algorithms.

Table 1 Comparison of execution time before and after loop unrolling of some HEVC algorithms

Algorithms	Before loop unrolling		After loop unrolling	
	Number of loops	Execution time （ns）	Number of loops	Execution time （ns）
Read operation	64	7510	8	1038
Store operation	64	14038	8	1754
DC mode selection	100	47213	15	7083
Planar mode selection	192	39238	24	4905
Image reconstruction	64	7935	8	991

2) Loop separation

In order to improve the parallelism of calculation, the algorithm is optimized by the method of loop separation. Only configuration information needs to be generated between sub-loops. During the process of loop separation, the execution order of each basic block will be changed at the same time.

$$dcValue = (\sum_{x=1}^{N} R_{(x,0)} + \sum_{y=1}^{N} R_{(0,y)} + N) >> (\log_2(N)+1) \qquad (2)$$

This paper takes 8×8 coding block size as an example to optimize the calculation process of *dcValue* in DC mode. The calculation of the predicted value *dcValue* is shown in formula (2). The main operation is a cyclic accumulation operation, and the number of accumulations is related to the current prediction block size. As shown in Figure 4, (a) and (b) are the codes before and after the loop separation. After loop separation, each time the algorithm is executed, the judgment for each row (column) is reduced. Only the current row or column needs to be judged, and different rows and columns can be executed on different PEs, which effectively improves degree of parallelism of the calculation. If the loops are completely separated and implemented on a reconfigurable processor, a total of 16 loops in different rows and columns are executed in 16 PEs, and the parallelism can be increased by 8 times.

```
for(i=0;i<8;i++)
  {
     sum=0;
     N=8;
     for(j=0;j<8;j++)
     {
        sum+=a[i][j]
     }
  }
   sum=sum+8;
   sum=sum>>4;
```

（a）Code before loop separation

```
for (i=0;i<8;i++)
    {
       sum1=0;
       sum1+=a[i];
    }
for(j=0;j<8;j++)
    {
       sum2=0;
       sum2+=b[j];
    }
    sum=0;
    sum=sum1+sum2+8;
    sum=sum>4;
```

（b）Code after loop separation

Fig.4 Loop separation comparison of DC mode selection

3) Loop tiling

Loop tiling can be used for imperfect nested loops. By adding conditional statements to move the outer layer operations to the inner layer, this increases the number of operations in the loop body and turns multiple nested loops into single-layer loops, as shown in Figure 5. Using the loop tiling method, both the inner loop and the outer loop can be parallelized and accelerated on the array processor.

```
for (i=0;i<N;i++)
   {
      sum=0;
      for(j=0;j<M;j++)
      {
         sum+=a[i][j];
         b[i][j]=sum;
      }
   }
```

(a) Nested loop code

```
for(n=0;n<N*M; n++)
   {
      i=n/N;
      j=n%M;
      if(j==0)
         sum=0;
         sum+=a[i][j];
      if (j==m-1)
         b[i][j]=sum;
   }
```

(b) Loop tiling

Fig. 5 Example of looping tiling code

Taking the Planar prediction mode as an example, it is mainly suitable for images with smoother texture, and the main calculation lies in the solution of $P^{H}_{(x,y)}$ and $P^{V}_{(x,y)}$. $P^{H}_{(x,y)}$ predicts the pixel value of each row of the prediction block in the horizontal direction, and traverses each pixel value of the first row in turn. When processing an 8×8 prediction block, it is necessary to loop 8 times to perform prediction. $P^{V}_{(x,y)}$ predicts the pixel value of each column, and it still needs to loop 8 times to predict. Planar algorithm mapping is loop nested execution, and loop tiling is needed to improve

calculation efficiency. Flatten the loop of inner calculation $P^{H}_{(x,y)}$ and the loop of outer calculation $P^{V}_{(x,y)}$, turning multiple nested loops into single-layer loops. After unfolding the single-layer loop, it is mapped to different PEs of the array processor for calculation.

B. DFG construction of HEVC algorithm

Loop optimization can reduce the conditional judgment at the end of each iteration, can effectively reduce the number of calculations for judgment and another branch of the array processor, reduce the number of reconfigurations of the array, and can also make full use of the computing units in the reconfigurable array. DFG is a direct expression of the loop body, so for loop parallelism in high-level languages, DFG is usually used for algorithm implementation.

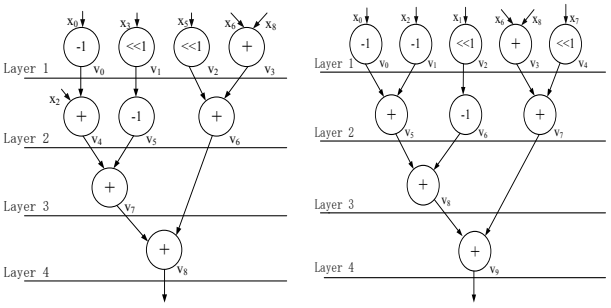
HEVC includes intra prediction, image reconstruction, integer motion estimation, fractional motion estimation, motion compensation, quantization and dequantization, and deblocking filtering algorithms. This article mainly analyzes the realization of parallel mapping when the nodes of the DFG are less than 16 on the 16 processing elements of a processing element cluster. When the number of nodes in the DFG is greater than 16, the DFG needs to be re-divided, which is beyond the scope of this article. Therefore, this article mainly studies the construction and hardware implementation of DFGs of Sobel operator, SAD algorithm, and matrix multiplication.

Through the statistics of the CU division results under different test sequences under the official software of HM16.0, the average probability of an 8×8 code block is 67%. In this paper, an 8×8 code block is used to construct a DFG to realize the algorithm.

1) DFG construction of Sobel operator

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ 1 & 0 & 2 \end{pmatrix} * A_x \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \quad (3)$$

$$G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * A_x \begin{pmatrix} x_0 & x_1 & x_2 \\ x_3 & x_4 & x_5 \\ x_6 & x_7 & x_8 \end{pmatrix} \quad (4)$$



(a) G_X (b) G_Y

Fig.6 DFG of Sobel operator

The Sobel operator is mainly used for edge detection. The calculations of the filter coefficients in the horizontal direction and the filter coefficients in the vertical direction are shown in equation (3), (4) respectively.

Expand Sobel's filtering formulas separately, there are many constant operations, such as, and so on. The dataflow diagram of the Sobel operator in the horizontal and vertical directions is shown in Figure 6.

2) DFG construction of SAD algorithm

In the previous section, the SAD algorithm was optimized for loop unrolling. The code after loop unrolling can unroll the cumulative calculation of each row and each column. In this paper, the smallest divided block 4×4 is used as an example to implement the SAD algorithm, and the loop calculation operation is expanded to obtain the SAD dataflow diagram, as shown in Figure 7. Among them, a0, b0, c0, and d0 represent original pixels, and a1, b1, c1, and d1 represent reference pixels.

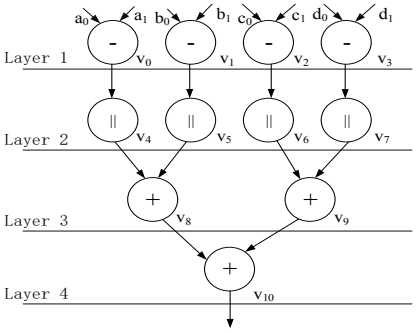


Fig.7 DFG of SAD

3) DFG construction of matrix multiplication

Matrix multiplication is mostly used in multimedia applications. Scalar replacement is used in the calculation of reconfigurable array processors. After scalar replacement, array elements can be effectively reused and memory access operations can be concentrated, thereby improving memory access efficiency. The numbers in the array are sequentially stored in the fixed address of the data storage, and after the calculation is completed, the temporary variable corresponding to the array element is written back to the array element. The DFG of matrix multiplication is shown in Figure 9. In 4×4 matrix multiplication, each element of the output matrix requires 4 multiplication operations and 3 addition operations. A total of 64 multiplications and 48 additions are required, that is, a total of 112 DFG operation nodes are required. The algorithm requires 32 input data (16 a and b matrices each) and 16 output data. Where a[0], a[1], a[2], a[3], b[0], b[1], b[2], b[3] respectively represent the first row of the first matrix And the first row of the second matrix.

```
for(i=0;i<4;i++)
{
    c[i][0] = a[i][0] * b[0][0] + a[i][1] * b[1][0] + a[i][2] * b[2][0] + a[i][3] * b[3][0];
    c[i][1] = a[i][0] * b[0][1] + a[i][1] * b[1][1] + a[i][2] * b[2][1] + a[i][3] * b[3][1];
    c[i][2] = a[i][0] * b[0][2] + a[i][1] * b[1][2] + a[i][2] * b[2][2] + a[i][3] * b[3][2];
    c[i][3] = a[i][0] * b[0][3] + a[i][1] * b[1][3] + a[i][2] * b[2][3] + a[i][3] * b[3][3];
}
```

Fig.8 4×4 matrix multiplication core code

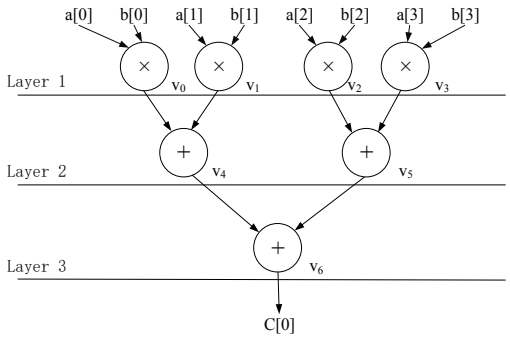
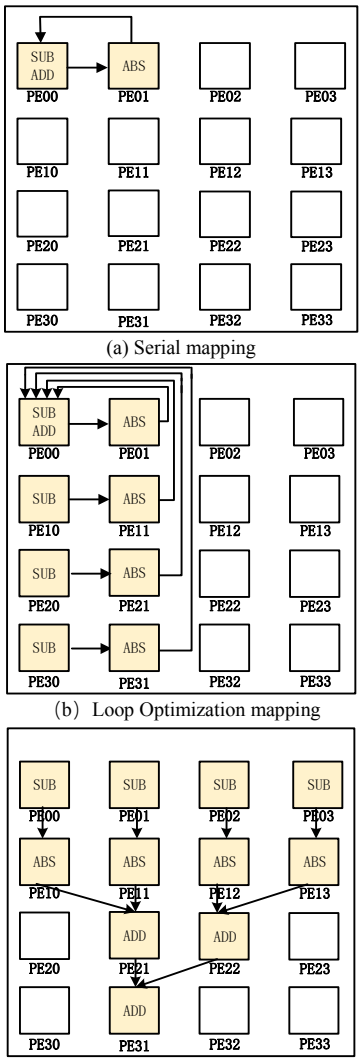


Fig.9 DFG of matrix multiplication

C. *Mapping*

Taking the 4×4 coding block size as an example, Figure 10 shows three different mapping schemes of the SAD algorithm on the array processor. Figure 10.(a) is the mapping scheme for serial implementation. The original code loop body of the SAD algorithm contains only one subtraction and one absolute value operator operation. Only 2 of the 16 PEs are working, and the remaining 14 are The PEs are all in an idle state. Figure 10.(b) is the parallel mapping scheme of the SAD algorithm after loop unrolling optimization. #PE00, #PE01, #PE02, #PE03 calculate the difference operation of 4 lines of code blocks respectively, #PE10, #PE11, #PE12, #PE13 respectively perform absolute value operations. In this case, there are 8 PEs working at the same time. Figure 10.(c) is a parallel mapping scheme based on the DFG of SAD. According to Figure 7, the operation in each node in the DFG is mapped to a PE of the array processor. In this case, there are 11 PEs working at the same time. Therefore, compared with serial implementation and parallel implementation after loop optimization, the DFG mapping method allows more PEs to work at the same time, which greatly improves the operating efficiency of the array processor.



(a) Serial mapping

(b) Loop Optimization mapping

(c) DFG mapping

Fig.10 Three mapping schemes of SAD algorithm on the array processor

Table 2 Performance analysis of Sobel operator, SAD and Matrix Multiplication

	Sobel Operator			SAD			Matrix Multiplication		
	Serial	Loop Optimization	DFG	Serial	Loop Optimization	DFG	Serial	Loop Optimization	DFG
Total execution time (ms)	1804.15	902.097	243.81	4.151	1.437	0.798	3.236	1.341	0.709
Cycle execution time (ms)	1665.52	460.97	110.94	3.570	0.332	0.160	2.880	0.439	0.192
Loop ratio (%)	92.3	51.1	45.5	86	22.4	20	89	32.7	27
PE utilization rate (%)	12.5	50	62.5	12.5	50	68.75	12.5	43.75	93.75

Table 3 Comparison of execution time of different test sequences

	Sobel Operation (ms)			SAD (ns)			Matrix Multiplication (ns)		
	Serial	Loop Optimization	DFG	Serial	Loop Optimization	DFG	Serial	Loop Optimization	DFG
carphone_qcif (176×144)	1804.15	902.097	243.81	4151347	1437004.8	798336	3236064	1341204	709632
bridge-far_qcif (352×288)	1762.3	879.15	238.61	4151295	1436497.2	797941	3236075	1341172	708937
foreman_qcif (1280×720)	1759.22	881.52	235.25	4151332	1436853.5	798219	3235956	1341216	709657
highway_qcif (1920×1080)	1767.24	884.62	239.01	4151352	1437121.5	798461	3236112	1341147	709841
Average	1773.23	886.84	239.17	4151332.3	1436869.18	798239.25	3236051.75	1341184.75	709516.75

IV. Experimental Results

A. Performance comparison of three implementation methods

As shown in Table 2, the experimental results such as the execution time and PE usage rate of the three algorithms after different implementations are counted. Compared with the serial implementation, the loop time of DFG implementation is reduced 94.06% on average, the execution time of DFG implementation is reduced by 81.79% on average, the speedup ratio can reach 7.4x, the average speedup is 5.72x, and the PE utilization rate is increased by 62.5% on average. Compared with the implementation of loop optimization, the loop time of DFG implementation is reduced by an average of 61.33%, and the execution time is reduced 54.61%, the average acceleration ratio was 2.4x, and the PE utilization rate increased by 27.08%.

Table 3 shows the execution time statistics of Sobel operator, SAD and matrix multiplication algorithm under different test sequences after using three implementation methods. The experimental data shows that the execution time of the DFG method is reduced by 81.77% and 54.86% on average, compared with the serial implementation method and after loop optimization.

B. Performance analysis comparison

Table 4 shows the performance comparison and analysis of the SAD algorithm after it is implemented on hardware. Literature [13] proposed a pipeline design for SAD parallel processing, which can process code blocks from 4×4 to 64×64 in size. The proposed SAD calculation is applicable to various search algorithms in motion estimation. Literature [14] proposed an effective parallel pipeline SAD structure, which realizes coding block grouping and hardware resource sharing, can process coding blocks from 4×4 to 64×64 in size, and can realize the calculation of flexible block division in HEVC. Although the hardware operating frequency of the SAD algorithm implemented in this paper is lower than that of literature [13] and [14], its execution delay is reduced by 7.2% compared to literature [13]. The architecture proposed in [14] requires 2048 cycles to execute a 64×64 encoding block, and each clock cycle can only process 2 pixels. The SAD algorithm based on DFG proposed in this paper takes the smallest partition 4×4 as an example. The coding unit with the largest size of 64×64 can be divided into 256 4×4 prediction blocks. Each 4×4 SAD value is independent of each other and can be implemented in parallel. It takes 8 clock cycles to realize an 8×8 coding block, and each clock cycle processes 8 pixels, which is 6 pixels more than [14], which greatly reduces the execution time.

Table 4 Performance analysis of SAD algorithm

	[13]	[14]	This paper
Implementation platform	Virtex-5	Artix-7	Virtex-6
Maximum frequency （MHZ）	475.21	498.2	120.72
LUTs	14761	25072	31730
Execution delay（ns）	44.19	32.06	41
Number of pixels （pixel）	-	4096	64
Calculation time （cycles）	-	2048	8
Throughput （pixel/cycle）	-	2	8

V. Conclusions

In this paper, through analysis of the data dependence and parallelism of the HEVC algorithm, after optimizing the loop algorithm that takes up more execution time, a DFG of the HEVC algorithm is constructed. The algorithm with a smaller DFG is simulated and verified. The experimental results show that compared with serial implementation and parallel implementation after loop optimization, the PE utilization rate of Sobel operator DFG is increased by 80% and 20%, respectively. The execution time was reduced by 86.51% and 73.03% respectively. Compared with serial implementation and parallel implementation after loop optimization, the PE

utilization rate of SAD algorithm DFG is increased by 66.67% and 33.33% respectively, the execution time is reduced by 80.7% and 44.45% respectively. Compared with the serial implementation and the parallel implementation of the loop optimization, the DFG implementation of the matrix multiplication algorithm has increased the utilization rate of PE by 86.67% and 53.33%, and the execution time has been reduced by 78.07% and 47.1% respectively.

In order to obtain high-performance HEVC encoding, this paper only analyzes the mapping implementation of smaller DFGs. Later we will continue to study how to divide the DFG with a large number of nodes on the array processor.

ACKNOWLEDGMENT

This research is supported by the National Key Research and Development Project of China (No. 2020AAA0104603), and the National Natural Science Foundation of China (No.61834005, 61772417, 61802304, 61602377, 61874087, 61634004), and the Shaanxi province key R&D plan (NO.2021GY-029).

REFERENCES

[1] Chen S, Huang J, Xu X, et al. “Integrated optimization of partitioning, scheduling, and floorplanning for partially dynamically reconfigurable systems”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018, 39(1): 199-212.

[2] Yun Z, Jiang L, Wang S, et al. “Design of reconfigurable array processor for multimedia application”. *Multimedia Tools & Applications*, 2018, 77(3):3639-3657.

[3] Kalali E, Hamzaoglu I. “An Approximate HEVC Intra Angular Prediction Hardware”. *IEEE Access*, 2019, 8: 2599-2607.

[4] Kou M, Gu J, Wei S, et al. “TAEM: fast transfer-aware effective loop mapping for heterogeneous resources on CGRA”.*2020 57th ACM/IEEE Design Automation Conference (DAC), San Francisco, USA*. IEEE, 2020: 1-6.

[5] Akbari O, Kamal M, Afzali-Kusha A, et al. “X-CGRA: An energy-efficient approximate coarse-grained reconfigurable architecture”. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2019, 39(10): 2558-2571.

[6] Wu C, Deng C, Liu L, et al. “A multi-objective model oriented mapping approach for NoC-based computing systems”. *IEEE Transactions on Parallel and Distributed Systems*, 2016, 28(3): 662-676.

[7] Kullu P, Tosun S. “MARM-GA: mapping applications to reconfigurable mesh using genetic algorithm”. *2019 22nd Euromicro Conference on Digital System Design (DSD). Kallithea, Greece,* IEEE, 2019: 13-18.

[8] Yin S , Liu D , Sun L , et al. “DFGNet: Mapping dataflow graph onto CGRA by a deep learning approach”, *2017IEEEInternationalSymposium on Circuits and Systems (ISCAS). Baltimore, MD, USA,* IEEE, 2017:1-4

[9] G. J. Sullivan, J. Ohm, W. Han and T.Wiegand, “Overview of the High Efficiency Video Coding (HEVC) Standard”. *IEEE Transactions on Circuits and Systems for Video Technology*, 2012, 22(12): 1649-1668.

[10] Bin Li, Gary J. Sullivan, Jizheng Xu. “Compression performance of high efficiency video coding (HEVC) working draft 4”. *2012 IEEE International Symposium on Circuits and Systems*, 2012, 13(21): 886 - 889

[11] Zeng L, Xu C, Li R. “Partition and Scheduling of the Mixed-Criticality Tasks Based on Probability”. *IEEE Access*, 2019, 7: 87837-87848.

[12] Wang Z, Tang Q, Guo B, et al. “Resource Partitioning and Application Scheduling with Module Merging on Dynamically and Partially Reconfigurable FPGAs”. *Electronics*, 2020, 9(9): 1461.

[13] Joshi A M, Ansari M S, Sahu C. “VLSI Architecture of High Speed SAD for High Efficiency Video Coding (HEVC) Encoder”. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2018:1-4.

[14] Nagaraju M, Gupta S K, Bhadauria V, et al. “Design and Implementation of an Efficient Mixed Parallel-Pipeline SAD Architecture for HEVC Motion Estimation”. *Advances in VLSI, Communication, and Signal Processing*, *Springer, Singapore*, 2021, 605-621.