An IDE for Reconfigurable Video Array Processor

Rong Yang*, Xiaoyan Xie*, Miaomiao Chai *, Lin Fang *, Wanqi He*, Jingtao Sun*
*Xi'an University of Posts and Telecommunications, Xi'an, China
E-mail: xxy@xupt.edu.cn Tel: +86-29-15389268117

Abstract— **Integrated development environment (IDE) is one of the key points to construct software ecological of reconfigurable array processor (RAP) chips. However, transplanting from conventional IDE is a daunting task, because of the complexity of high-level behavior description in front-end and special spatial-temporal instructions bind with hardware, such as branch prediction, out-of-order execution, SIMD parallelism. Therefore, we propose a hierarchical IDE design method. At the front-end, the static back slicing is introduced to deconstruct the abstract semantics of high-level language (HLLs) into relatively fixed operations and simple structure. So that the spatial-temporal features are easy to be peel out. At the bottom, the machine instruction sets are encapsulated into instruction groups（IGs）. The semantic abstraction level of hardware description is enhanced. Physical hardware details are separated from the Intermediate Representation（IR），the scalability is brought out. Finally, an IDE is developed by this method, for high efficiency video coding (HEVC) algorithm mapping. The testing results show that the efficiency of algorithm development is greatly improved while maintaining the same coding quality.**

Keywords— **IDE; static back slicing; instruction groups; Reconfigurable Array Processor; HEVC**

I. INTRODUCTION

In real time vision and video encoding, the power cost and computing efficiency of hardware acceleration become the bottleneck of edge computing, especially conventional processor based on von Neumann architecture, such as CPU, GPU, DSP etc. The accelerator based on application specific integrated circuit (ASIC) can provide high computing energy efficiency with low power consumption. But it couldn't effectively deal with the demand of developing cycle and fast iterative caused by algorithm variability. In decade years, coarse grained reconfigurable array (CGRA） is more and more recognized in industry and academia. The CGRA offers the promise of substantially accelerating computations through the concurrent nature of hardware structures and the ability of these architectures for hardware customization [1]. It has gradually become the first choice of video coding or neural network accelerators. But reconfigurable array present numerous challenges to the average software programmer, as they expose a hardware-oriented computation model where programmers must also assume the role of hardware designers. Existing operating system, compiler and IDE，suitable for universal stored-program (Von Neumann Type) computing models, could not give full play to features and true capabilities of CGRA[2]. Bad software ecology and inherent programming habits of programmer cause higher time costs of learning, developing, maintenance and iteration, even no appreciated on-chip resource utilization [3]. The new IDE toolkit has become impending in industry, which could pad the gap between abstract semantic of software and parallel computing models of hardware.

The core of IDE is compiler, translating HLLs into binary machine code which can be recognized by specific processor. Compiler design is a very fussy and formidable task. The syntax characteristics analyzed of HLLs, instructions and memory structure and the standardization of generated code are all involved. Although some mature tools had been developed for syntax/morphology analysis or code generation, there still needs heavy works to regular expression modified, HLLs logic split and physical details mapping etc. Therefore, we propose a hierarchical IDE design method. At the front-end, task-oriented static back slicing is used to replace the lexical analysis and syntax parsing. The abstract semantics of HLLs can be deconstructed into relatively fixed operations and simple structure. So that the patio-temporal features are easy to be peel out. At the bottom, the machine instruction sets are encapsulated into several instruction groups. The semantic abstraction level of hardware description is enhanced in back-end. Resource allocation relative mapping rules are defined in Intermediate representation (IR) to mapping program pieces to instruction groups. Physical details are separated from the back-end，the portability is brought out. Based on the DPRAP, a RAP developed by author's team, an IDE for HEVC algorithm mapping is developed. The hierarchical design enhances the portability for different hardware platform.

II. RELATED WORKS

Traditional HLLs are stored-program, the von Neumann manner adopted by CPU and GPU. The computing model relies on an implicit sequential consistency. The storage in is abstracted in a single large virtual address space with uniform access time. In CGRA, every thread must be provisioned with resources statically. The hardware execution is data flow driven. In the early compilers, branch prediction, out-of-order execution, SIMD parallelism are all not fully considered. The trigger of computing advantages of CGRA is translating the temporal sequential logic to tempo-spatial parallel logic. So, researchers explored some schemes with the help of high-level synthesis (HLS) joint with hardware prototype. In [3], a C based ROCCC compiler is designed to compile HLLs into FPGAs, a possibility for loop transformations and optimizations. Baxter [4] develops a parallel toolset to provide a series of matrix operation application program interfaces (APIs), by software layer virtualizing Maxwell [5] components.

Bosshart [6] described a specific HLL and compiler for abstract forwarding model of programmable switches. Specially designed register file, mentioned in [7], is more compatible for searching space, it makes loop nests compiling more efficiently. The CGRA simulator proposed by [8], providing the HDLs mapping from multithread concurrent tasks through an optimizing compiler. All above efforts tries raising the abstraction level of HDLs to that of HLLs, by using the hardware/ software codesign theories. The research is also focused on the hardware coupled loop-unrolling, loop iteration and loop-merging. The scalability and portability are not being actively considered. Another trying is to rebuild CGRA compiler from LLVM [9], an open source compiler framework developed by [10], made computationally intensive applications working well on reconfigurable arrays. Traditionally, LLVM is used to be front-end. As such it has access to many software optimizations. However, hardware and software programing paradigms are inherently different, so we cannot expect all optimizations of LLVM to work seamlessly for HLLs.

Almost all above works focuses on IR code generation and loops optimization, excluding the front-end rebuilt. In fact, the syntax tree generated from front-end is a kind of fine-grained, ordering coherence intermediate code, after lexical and syntactic analysis. It is not available for paralleling instructions of CGRA. Complex computing segmentation, data allocation, data flow control and other strategies need to be added. And the back-end and IR remain a daunting to programmers. Recently, program slicing [11] is especially act out in software analysis and testing. It decomposes the program according to the dependency inside the software. After the lexical and syntactic analysis, program slicing also continue to extract semantics according to different observation points and generate control flow diagrams and source code fragments. Different from fine-grained symbol table and syntax tree, figured out from front-end, these code fragments are regular and single, the spatial-temporal features are clearer. Then, IR code conversion rules are simpler.

To generate target machine code has indivisible relationship with hardware machine code, which is core of the back-end. The optimization strategy for hardware structure is also indispensable to maximize the parallelism of array processors. Which will bring huge works to the reuse and migration of compilers. If there defines an abstraction layer with simple semantics, to encapsulate machine instructions and hardware structure, between the hardware and the back-end. Such a modify may be reduce the dependence of back-end on hardware platform and improve the portability of compiler.

III. THE IDE DESIGN OF CGRA

A. The CGRA Oriented Compilation Framework

According to above thinking, the CGRA oriented compilation flows can be constructed as in Fig.1. Instead of simple lexical and syntactic analysis in normal front-end, slicing tool is used to perform lexical analysis, semantic extraction, Loops optimizing, fragmentation etc. By this way, the output of front-end will be source code fragments and corresponding control flow diagrams. According to the control flow diagram, the computation mode maybe suitable for spatial or temporal. Temporal refers to the splitting of computations in sections to be executed by time-sharing the processing elements (PEs), whereas spatial refers to the splitting of the computations in sections to be executed on multiple PEs. Which are supported with one or both in many of CGRA. The temporal mode can be mapped into pipeline controlled by instruction flow, the spatial mode can be controlled by dada flow. This partitioning and the corresponding data partitioning are guided by specific performance metrics, such as the estimated execution time and power dissipation within IR.

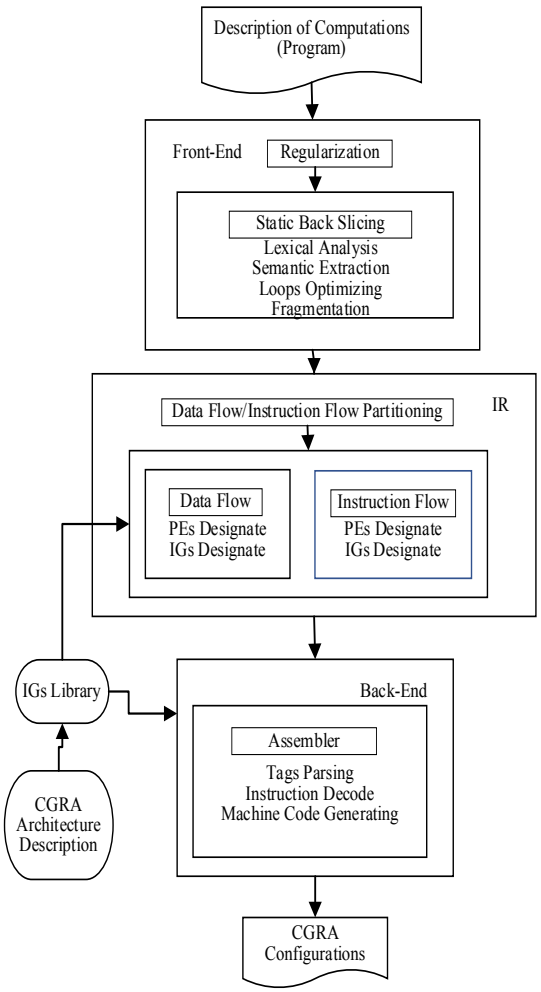


Fig.1 The CGRA oriented compilation flows

Although the structural characteristics of computation are more obvious handled by such a front-end, it is still difficult to mapping it to machine codes. So as the last one thought discussed above, we encapsulate machine codes into instruction groups (IGs), according to customized special application. What exposes to the back-end and IR is abstraction semantics instead of machine code with physical structure

features. So, the IR just need to judge the type of temporal or spatial for the computations of slice, which are mapped to the corresponding IGs of the CGRA. Such design of IR is very simple.

At the back end, it also shows loose coupling with hardware details. The assembler only translates IGs to machine codes by calling IGs library interface, according to tag parsing. And then, the CGRA configuration file is generating. The CGRA will load the configuration file and perform the computations.

B. The IDE Architecture

The compilation flows given in Fig.1 are integrated into the IDE, as shown in Fig.2. This IDE can be used to programming for HEVC algorithms.

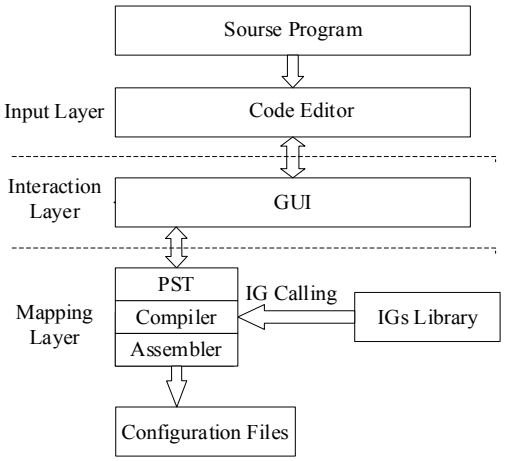


Fig.2 IDE framework

The GUI provides visual design components for programmer, contains the source program and the video data stream. With Code Editor, C syntax codes can be inputs and compiling to IGs. In which the code completion and input tracking functions for HEVC algorithm mapping are involved in. Code completion is designed to complete the functions, methods and keywords that users may use according to part of their input. Input tracking can prompt if it exists spelling or grammatical errors. PST is an embedded program slicing tool. The mapping between program slicing segments and IGS is performed within compiler. The assembler translates IGs to configuration file in machine code form, which can be loaded, recognized and distributed by controller of DPRAP. In the system design, the disassembly module is provided to disassemble the binary machine code into assembly instructions, which is convenient for programmers to check and modify errors.

C. IGs for HEVC

By accumulation of long-term HEVC algorithm mapping works, we summarize up the common operations, shown in Table 1.

Table 1 IGs for HEVC

Type	IG Name	Function
Conrtrol Flow	HandSend	PE Handshake of Sender
	HandRecv	PE Handshake of Receiver
	Load	Data Loading
	Store	Data Storing
	Routing	Remote PE Routing
	Add	Addition
	Sub	Subtraction
	Mul	Multiplication
	Compare	Data Comparing
Data Flow	MatrixAdd	Matrix Summation
	MatrixSub	Matrix Subtraction
	MatrixMul	Matrix Multiplication
	MatrixTrans	Matrix Transposition
	SAD	Sum of Absolute Difference
	MatrixT	Matrix Inversion
	MatrixCh	Matrix Blocking

Each IG provides parameters corresponding functional requirements. Each parameter will be mapped to the fixed position during translation. According to the functions, there are two type of IGs. As shown in Table 1. The Data flow IG is responsible for data flow computations. The control flow IG is responsible for communication between PEs and external memory. The translation process from API to IG is shown in Fig. 3. The IG's expressions are named in API. When scanned by IR of compiler, they are mapped into IGs library, and exported the corresponding assembly codes.

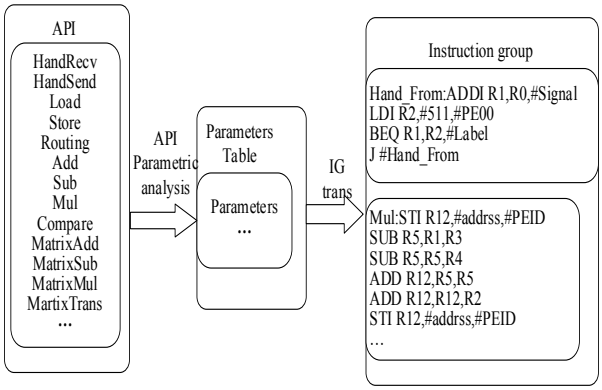


Fig.3 Translation process from API to IG

IV. TESTING AND VERIFICATION

A. The Hardware Platform Supported for IDE

The IDE indicated in this issue is to assist the works of HEVC algorithms mapping on the DPRAP, a CGRA proposed by our project team. Which It can support the calculations of MPEG, HEVC, the other video coding algorithms and convolution and pooling in neural networks. As shown in Fig.4,

the Global Controller (GC) is the kernel of reconfigurable mechanism in DPRAP. Which provides the host interface (HI) to IDE, maintains the controlling to processor array composed with PE groups (PEG). Each PEG has 16 light-core PEs. Communications among adjacent PEs are realized directly through four shared registers, lies in east, south, west and north (RE, RS, RW and RN). PEG communicated with each other by the routers. The GC receives the configuration instruction sequences from the bus of HI and reconfigures the functions of PEs by changing the instruction in the memory. The H-Tree network and mask-based addressing makes configuration instruction distribution and calling simplifier. These designs make the function of PEs reconfigured in runtime and flexible changing between SIMD and MIMD modes.

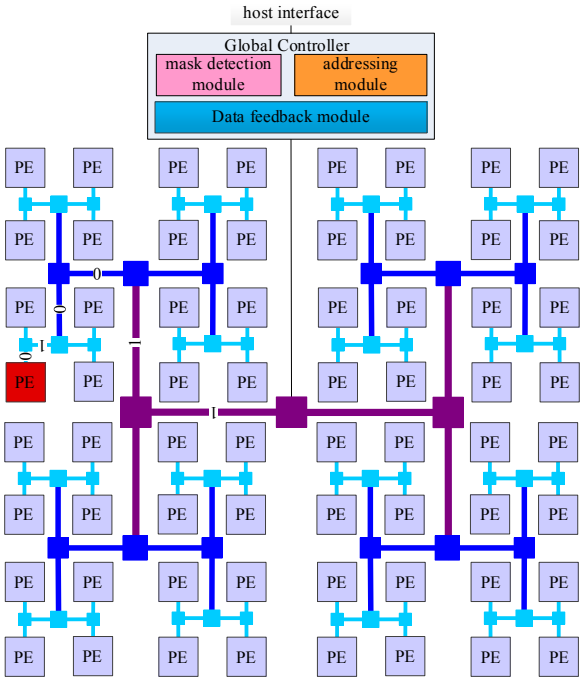


Fig.4 The functional diagram of DPRAP

B. *The Software Tools of IDE*

The QT is selected to support the GUI of IDE, The DPRAP is simulated with Modelsim. The data interaction between IDE and Modelsim use the inter-process communication function of QT. The HLL codes are input to the GUI, then sliced and interpreted into assembly instructions through compiler. The assembly instructions are translated into binary machine code through assembler. The functions of machine codes are simulated by Modelsim. Figure 5 shows the interface and running results of the IDE. The window 1 shows the HLLs input. The window 2 shows the slicing results. IGI calling results is shown in window 3, and binary codes is compiled in windows 4.

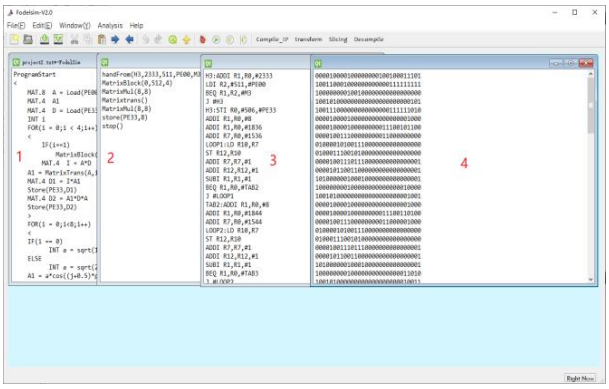


Fig.5 The interface display of IDE

C. *The slicing verifications*

In standards of HEVC, the positive transformation phase of transform quantization is referred to (1) to (3).

$$F = AfA^T \tag{1}$$

$$A(i,j) = c(i)\cos\left[\frac{(j+0.5)\pi}{N}i\right] \tag{2}$$

$$c(i) = \begin{cases} \sqrt{\frac{1}{N}}, i = 0 \\ \sqrt{\frac{2}{N}}, i \neq 0 \end{cases} \tag{3}$$

Where A is the transfer matrix, f is the pixels matrix, F is the coefficient matrix of forward converter, N is the dimension of code block, i and j are horizontal and vertical frequencies of 2D-wave respectively. They are described by HLLs as shown in Fig. 6(a).

Given a slicing criteria, $C = (13, A1)$, it means that the interesting point is on line 13 of program. To the slice of variable $A1$, the static backs slicing is to extract all sentences that affect variable $A1$ before reaching the interesting point. As this interpretation, the slicing result is shown as Fig.6(b).

```
1  ProgramStart
2  <
3      MAT.8  A = Load(PE00,0)
4      MAT.4  A1
5      MAT.4  D = Load(PE33,0)
6      INT i
7      FOR(i = 0;i < 4;i++)
8      <
9          IF(i==1)
10             MatrixBlock(A,4,i)
11         MAT.4  I = A*D
12         A1 = MatrixTrans(A,i)
13         MAT.4 D1 = I*A1
14         Store(PE33,D1)
15         MAT.4 D2 = A1*D*A
16         Store(PE33,D2)
17     >
18     FOR(i = 0;i<8;i++)
19     <
20         IF(i == 0)
21             INT a = sqrt(1/4);
22         ELSE
23             INT a = sqrt(2/4);
24          A1 = a*cos((j+0.5)*pi*i/4)
25     >
26 >
27 stop()
```

(a) HLLs description of positive transformation

```
MAT.8  A = Load(PE00,0)
MAT.4  A1
INT i
FOR(i = 0;i < 4;i++)
<
    IF(i==1)
        MatrixBlock(A,4)
    A1 = MatrixTrans(A,i)
>
```

(b) Results of slicing criterion C
Fig.6 HLLs description and slicing results of positive transformation

Comparing Fig.6(a) to (b), it is clearly that the sentences unrelated to the output have been deleted for sliced segment. In slicing results, the operations before the interesting point is to generate the transfer matrix and calculate the converter. According to the semantics, the horizontal and vertical frequencies of 2D-wave and the size of the pixels block can be translated to IG calling.

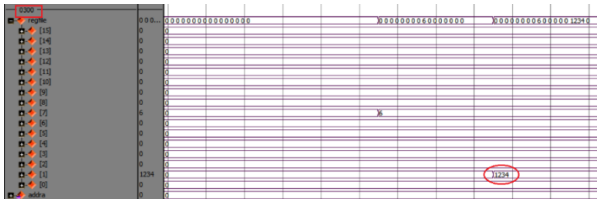
D. IG mapping verifications

The IGs for handshake of are used for communication between PEs in same PE group. The handshake signal is sent and received through adjacent interconnection registers. So, IGs defined for handshake are two types. The API function of handshake IG used for sender is defined as $HandSend(PEID, Signal)$, there $PEID$ is the PE ID of receiver, $Signal$ is the handshake signal. The API function of handshake IG used for receiver is defined as $HandRecv(HandFrom, Signal, addrstart, PEID, Frist_{Label})$, there $HandFom$ is the tag of Precursor algorithm, $,PEID$ is the PE ID of sender, $addrstart$ is the start address for loading data, $Frist_{Label}$ is the entry position of jumping. For example, if it wants to shake hands between PE00 and PE01, then operations for sender and receiver PE can be instantiated $HandSend(PE01,1234)$ and $HandRecv(1234,511,PE00,Table)$. The compiled results of which are shown as in Fig.7 (a) and (b) respectively. When these codes are assembled and distributed to DPRAP, the waveform is shown as Fig.8(a) and (b).

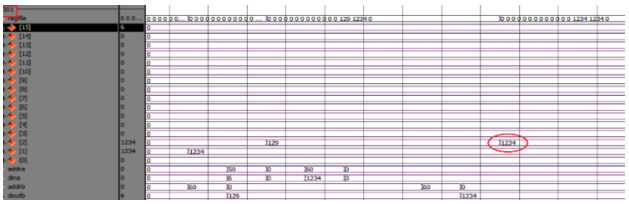
```
ADDI R7,R0,#6
STI R7,#50,#PE01
ADDI R1,R0,#1234
STI R1,#60,#PE01
```

```
ADDI R1,R0,#1234
LOOPS:LDI R2,#60,#PE01
BNE R1,R2,#LOOPS
LDI R15,#50,#PE01
```

(a) Compiled sender (b) Compiled receiver
Fig.7 The compiled results of handshake IG



(a) Simulation results of sender IG calling



(b) Simulation results of receiver IG calling
Fig.8 Simulation results of handshake of PEs

E. HEVC Testing

Using the IDE developed by this paper, we map the intra-coding of HEVC on DPRAP. We select 6 indicators to evaluate the video coding quality of HEVC testing model (HM) and our works. The testing sequences Container_qcif and Foreman_qcif are used to compare the coding qualities between HM and our works, results are shown in Table 2.

Table 2 Coding qualities comparison

Testing Sequences	Indicator	HM	Our works	△
Container _qcif	PSRN/dB	34.25	35.42	1.17
	SSIM	0.9991	0.9992	0.0001
	KBlur	0.999	0.998	-0.001
	MSE	142	142	0
	MAE	136	136	0
Foreman _qcif	PSRN/dB	33.98	38.64	4.66
	SSIM	0.9984	0.9985	0.0001
	KBlur	0.998	0.997	-0.001
	MSE	195	195	0
	MAE	194	189	-5

The peak signal-to-noise ratio (PSNR) of a I-frame indicating the impact of noise introduced during coding. The larger the PSNR value, the better the image quality, the smaller the noise introduced in the coding. The structural similarity (SSIM) makes up for the perceptual characteristics of human visual recognition. The closer it is to the constant 1, the better the reaction quality. The Kblur evaluates the difference between the original pixel and the coded pixel. The mean absolute error (MAE) calculates the average of absolute difference of original pixels and the coded pixels. The mean squared error (MSE) sums the square of the difference between original pixels and the coded pixels, it is more sensitive to outlier. △ is the difference of HM and our encoder. From Table 2, it is easy to see that the △ value of SSIM, KBlur, MAE and MSE are small enough to be almost negligible. The PSNR of our works are all superior to HM. But the workload and difficulties of algorithm migrating is decreased largely.

V. CONCLUSIONS

In this paper, we focus on the trouble of transplanting compiler to parallel CGRA. Thinking of the complexity of

high-level behavior description in front-end, we introduce the static back slicing tools to deconstruct the abstract semantics of HLLs into relatively fixed operations and simple structure. So that the spatial-temporal features are easy to be peel out, and tasks distributed on CGRA are more optimal. Aiming at special spatial-temporal instructions are bind with hardware architecture tightly, such as branch prediction, out-of-order execution, SIMD parallelism. It makes compiler migrated hardly. We encapsulate the machine instruction sets of common operations into IGs. Resource allocation relative mapping rules are defined in Intermediate representation (IR). Physical details are separated from the back-end，the portability is brought out. So, mapping program pieces to instruction groups made code generation no longer associated with hardware instructions. Oriented to the dynamic programmable reconfigurable array processor (DPRAP), developed by author's team, an IDE for high efficiency video coding (HEVC) algorithm is developed by this method. The testing results show that the slicing makes the compiler simpler, algorithm mapping on DPRAP is more convenient. If the CGRA platform is changed, there only needs to rebuild IGs library and its corresponding API.

VI. Acknowledgment

This research is supported by the National Natural Science Foundation of China (61802304, 61602377, 61874087), and the Shaanxi province key R&D plan (NO.2021GY-029, 2021KW-16).

D. References

[1] Tu F, Yin S, Ouyang P, et al. Reconfigurable Architecture for Neural Approximation in Multimedia Computing[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2019, 29(3), pp. 892-906.

[2] Cardoso J M P, Diniz P C, Weinhardt M. Compiling for reconfigurable computing: A survey[J]. Acm Computing Surveys, 2010,42(4), pp. 1-65.

[3] Windh S, Ma X, Halstead R J, et al. High-Level Language Tools for Reconfigurable Computing[J]. Proceedings of the IEEE, 2015, 103(3), pp.390-408.

[4] R. Baxter et al., "The FPGA High-Performance Computing Alliance Parallel Toolkit," Second NASA/ESA Conference on Adaptive Hardware and Systems (AHS 2007), 2007, pp. 301-310.

[5] Baxter R, Booth S, Bull M, et al. Maxwell - a 64 FPGA supercomputer[J]. Engineering Letters, 2008, 16(3), pp.287-294.

[6] Bosshart P, Daly D, Gibb G, et al. P4: Programming protocol-independent packet processors. ACM SIGCOMM Computer Communication Review, 2014,44(3), pp.87−95.

[7] WZ. Yin, ZY. Zhao, ZG. Mao, Q. Wang, WG, Sheng. A fast and efficient compilation framework for coarse-grained reconfigurable architecture [J]. Microelectronics and Computer,2019,36(08):45-48+53.

[8] YP. Niu. Research on Co-Verification Technology of Software and Hardware in Reconfigurable Multi-core System [D]. Hefei University of Technology,2020.

[9] Sun Y,Zhang Y,Qian J.Program Slicing Method of LLVM IR Based on Information-Flow Analysis[C]// 2019 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC). IEEE, 2020: 166-172.

[10] Lattner C, Adve V. The LLVM Compiler Framework and Infrastructure Tutorial[J]. Lecture Notes in Computer Science, 2005, 3602:15-16.

[11] XJ. Zhang. Program slicing technology research and slicing scheme design [D]. University of Electronic Science and Technology,2017.