A Multi-thread Processor Architecture With Dual Phase Variable-Length Instructions

Hyung-Ki Jeong[1] and Kwang-Yeob Lee[2], Jae-Chang Kwak[3]
[1,2]Dept. of Computer Engineering, Seokyeong University
[3]Dept. of Computer Science, Seokyeong University
407-ho, bukak building Seo-kyeong Univ. Jeongneung 4-dong, Seongbuk-gu, Seoul, Rep. of Korea
Tel/Fax : +82-2-940-7240
E-mail: [1]clonexking@skuniv.ac.kr

Abstract: Most of multimedia processors for 2D/3D graphics acceleration use a lot of integer/floating point arithmetic units. We present a new architecture with an efficient ALU, built in a smaller chip size. It reduces instruction cycles significantly based on a foundation of multi-thread operation, variable length instruction words, dual phase operation, and phase instruction's coordination. We can decrease the number of instruction cycles up to 50%, and can achieve twice better performance.

1. Introduction

For high performance of the multimedia software or the other API's work, we need a processor on the most popular architectures of SIMD[1] and VLIW[2](Very Long Instruction Words) domain. These processors have many integer/floating point ALUs and special functions (reciprocal, reciprocal square root…). They use many micro-operation bit-fields on instruction to process them concurrently. But too long instruction may cause a waste of instruction memory size.

We are applying new architectures on a next generation processor. One of them is a multi-thread operation. It doesn't need to handle with branch, data and control hazard. Because the term between of an instruction and a next instruction is long enough to ignore stalls caused with instruction dependence.

The other ways are reducing the wasteful instruction size and make fragmentations of VLIW instruction to VL-IW (Variable Length Instruction Words) architecture for effective phase instruction operation and cooperation. A dual phase operation leads to efficient use of ALUs and to shorter instruction cycles with phase cooperation.

New architectures are implemented based on shader model 3.0 for 3D acceleration. The shader[3] model 3.0 is a part of Micro-soft's DirectX 9.0 API[4]. It can accelerate the OpenVG API for 2D vector graphics. The new architectures can be applicable to 2D or 3D graphics.

2. The Proposed Architecture

2.1 Multi-thread processing

Multi-thread is a technique to improve the processor's performance with minimum resources.

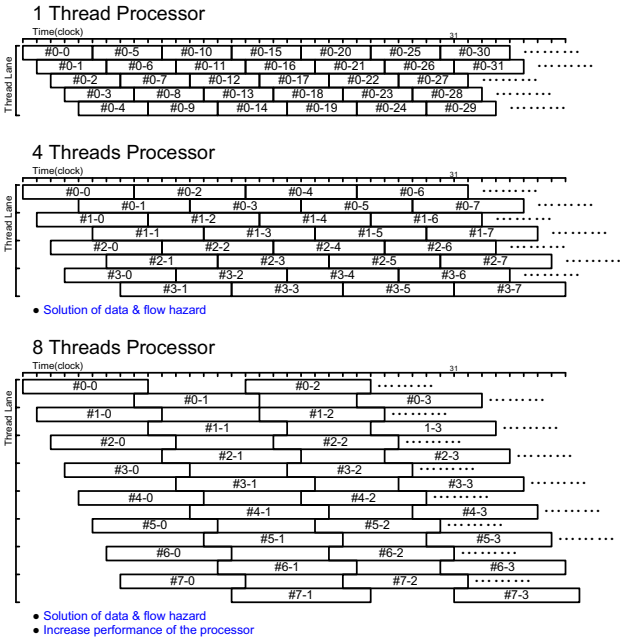


Fig. 1. Multi-thread Processing

Fig. 1 shows the simple round-robin multi-thread execution. A processor can get rid of the dependency between instructions in this way, so it eliminates data or branch hazard. A handling module for the hazard stall cases is removed, but thread status registers are required for additional thread counts. If the processor has enough threads, processor pipelines can be extended to improve operation speed without data/branch hazards, because multi-thread can loose the instruction's dependency that can be ignored.

2.2 VL-IW(Variable Length Instruction Words) Architecture

VLIW have a micro-operation implementation structure, which consists of micro-operations with long instruction length to control the arithmetic/control unit on SIMD. VLIW's instruction includes many micro-operations, and it has a fixed long length. Although just a few micro-operations are needed, full instruction length must be used. It wastes too many instruction fields. So instruction format must be reconfigured based on the execution frequency of micro-operations.

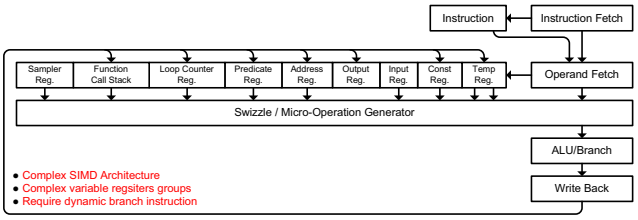


Fig. 2. SIMD(Single Instruction Multiful Data) architecture

Fig. 2 shows SIMD architecture that represents recent API shader model 3.0. For the architecture, many registers and instruction bit-fields are needed to indicate register groups clearly. In our previous research, we designed the instruction field for this architecture. Minimum 64 bit instruction fields were needed for shader model 1.0, and 102 bit instruction fields were needed for shader model 3.0. But most instruction fields are used very rarely. It is too wasteful to apply on the media processor.

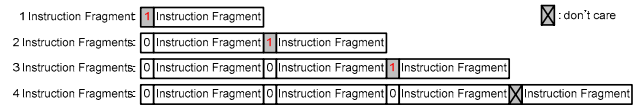


Fig. 3. End bit - VL-IW(Variable Length Instruction Words)

To resolve the waste problem of instruction fields, we divide the instruction format to maximum 4 small structured 32 bit instruction fragments. Each instruction fragment has one micro-operation field, one destination and one source operand field. We can construct thousands of instruction fragment combinations by defining multiplex micro-operations and sources. Fig. 3 shows combination methods with an end bit of instruction fragment's MSB. All combined instruction fragments can be executed concurrently.

2.3 Dual Phase Architecture

Another issue of SIMD is a waste of various registers groups. If you see the API at first glance, it should be drawn a line between many registers groups.

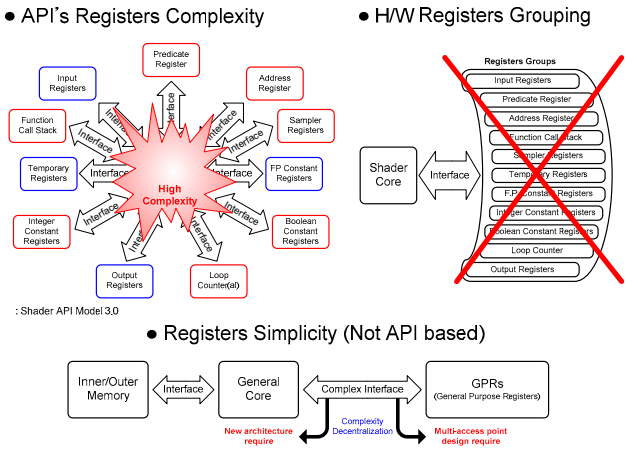


Fig. 4. Registers set requirement of shader model 3.0

Fig. 4 shows how registers groups can be simplified on example DirectX 9.0 API's shader model 3.0 architecture. As shown in Fig. 2, it is configured for API's registers requirements, but too many hardware resources are used to fit for API's requirements. Ironically it simply can be solved as against that. The solution is having only one registers group.

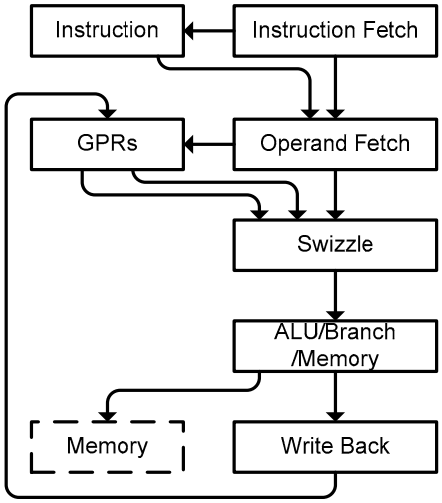


Fig. 5. Simple SIMD architecture

Fig. 5 shows that all registers groups merge to single GPRs(General Purpose Registers). In this way, the processor can be simpler than a full API supported processor. But it is restricted to the usage of a single register. The restriction of registers utilization can't support the API's requirements, or many macro instructions could be needed to implement each API domain's instructions.

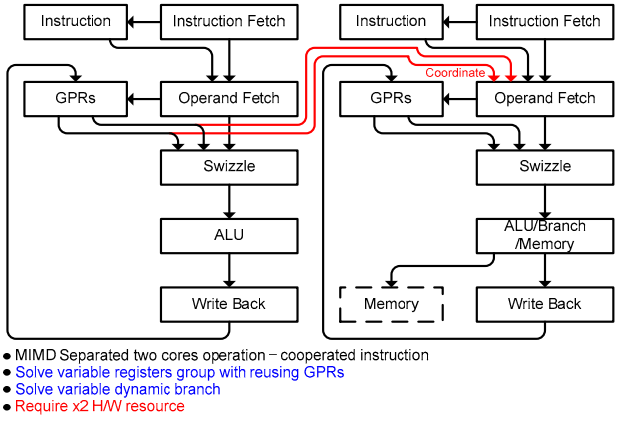


Fig. 6. Cooperated SIMD architecture

We must find the way to solve the side effect of simplified GPRs structure. We can imagine simply two processors with GPRs as shown in Fig. 6. Let assume that one processor can coordinate source operands of the other processor. With this, it will enlarge registers' scalability and more complex registers operations can be performed even though processor has just one GPR. But it requires twice chip size. So a unification process of two processors is needed for the overlapped hardware modules.

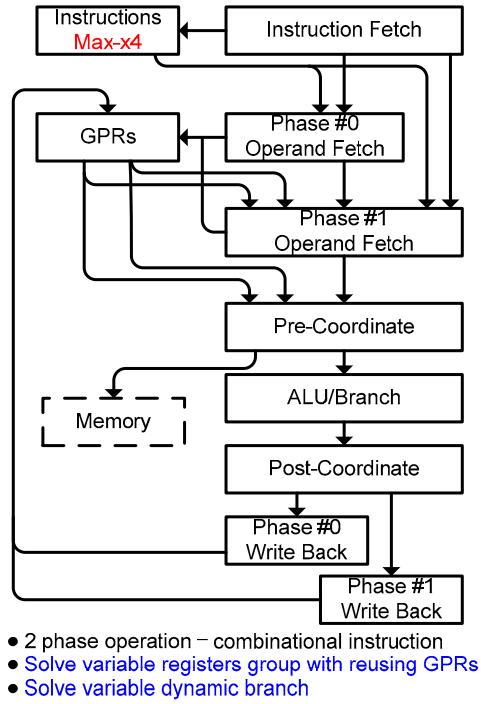


Fig. 7. Dual Phase SIMD architecture

Fig. 7 shows the exclusion of the overlapped modules. Two processors' architecture is merged to share one registers module and ALUs from Fig. 6. We called it 'dual phase architecture'.

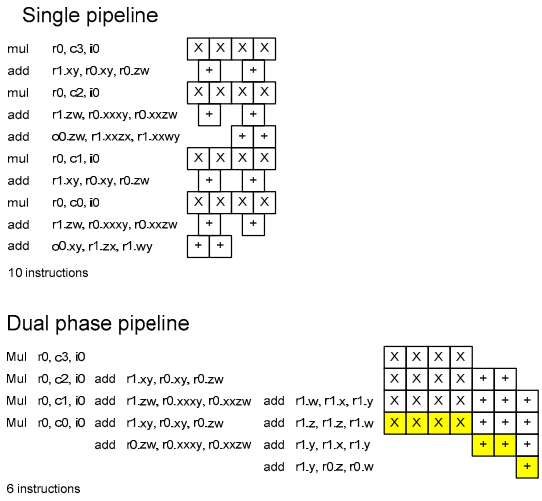


Fig. 8. ALU utilization of Dual phase pipeline

Fig. 8 shows an example of dual phase operation's efficiency through the matrix operation.

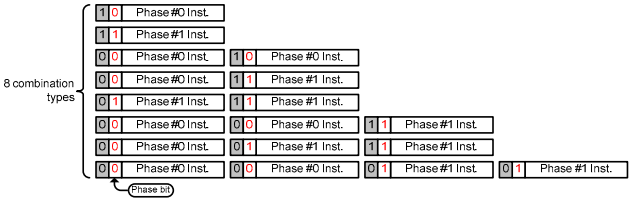


Fig. 9. Phase bit - Dual phase architecture

Fig. 9 illustrates combinations of dual phase instructions that can generate thousands of instruction fragments' combinations with an additional phase bit to distinguish phase operations. Each phase can combine and take maximum 2 instruction fragments.

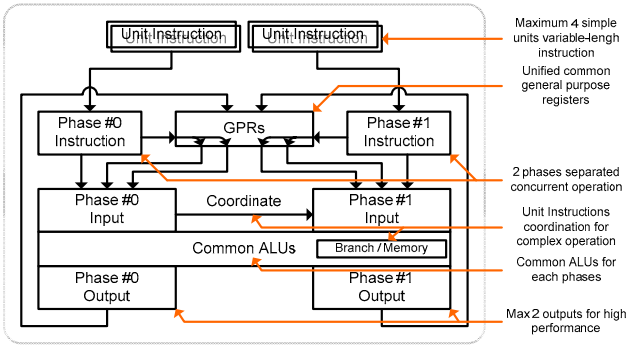
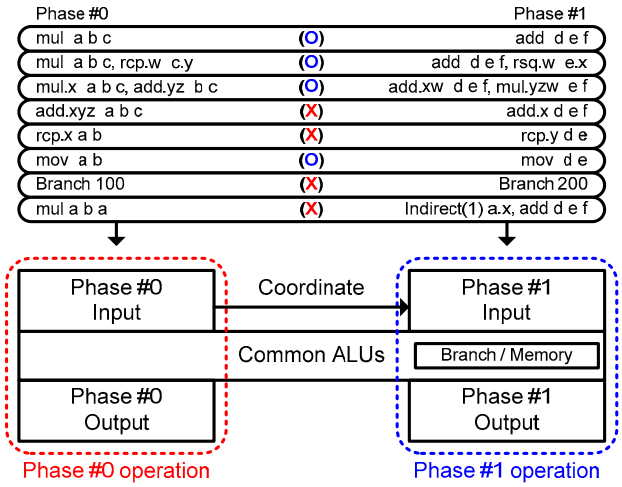


Fig. 10. Dual phase architecture

Fig. 10 shows a comprehensive architecture of all presented architectures up to now. It is organized into two phases. It shares the GPRs and ALUs. It can process maximum 2 instruction fragments per a phase. Each phase can designate maximum 2 sources and 1 destination. The phase #1 instruction can coordinate by phase #0 instruction. Therefore, phase #1 instruction can handle complex operations, such as a branch or memory operations, without any modules to support these operations.

2.4 Combination of instruction fragments

Instructions can be combined with maximum 4 instruction fragments. They can perform thousands of operations. Since each phase is shared to maximize the efficiency of ALU, there are exclusive paring rules as shown in Fig. 11.



: Allow none conflict using arithmetic unit
: X component adder conflict
: Special function is 1 scalar unit
: All phase allow duplicated move instruction
: Branch & memory operation can process only at phase #1
: Some coordinate instructions must be placed at phase #0

Fig. 11. Exclusive pairing rules

For example, the use of same arithmetic units on the same component at each phase is prohibited. The other cases of branch or memory instruction fragments can be performed only on phase #1.

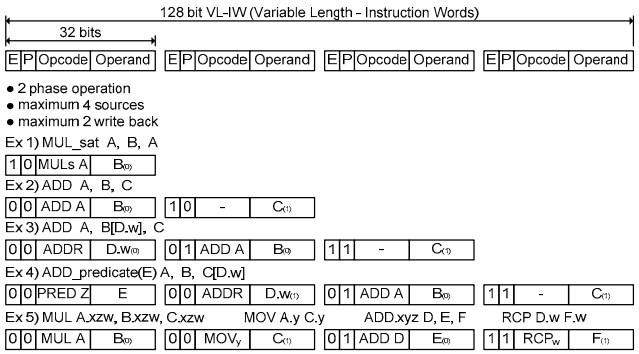


Fig. 12. Arithmetic operation with dual phase VL-IW

Fig. 12 shows some examples how the proposed architecture uses ALUs efficiently. Example 1&2 present that it makes shorter implementations of general arithmetic operation with small instruction fragments. Example 3&4 present complex arithmetic operations coordinated by phase #0 instruction. Example 5 shows many arithmetic operations can be performed concurrently.

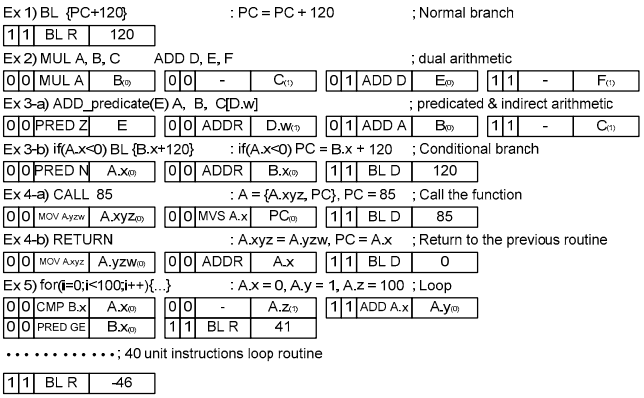


Fig. 13. Branch and looping operation with dual phase VL-IW

Fig. 13 shows examples of indirect branch, nested branch, comparison branch, and looping functions with minimized instruction counts. Among the above functions, all branch instructions can be performed with one instruction.

3. Conclusion

We present a new architecture for using efficient ALUs with dual phase variable length instruction method. With the multi-thread dual phase architecture, we can decrease the instruction implementation cycles length up to 50% , and the performance is minimum 200% better than a generic SIMD architecture as shown in Fig. 14.

Fig. 14 shows the performance comparison of generic SIMD, multi-thread, and multi-thread with dual phase architecture, using some functions and branch operations on the 2D and 3D graphics accelerated processor.

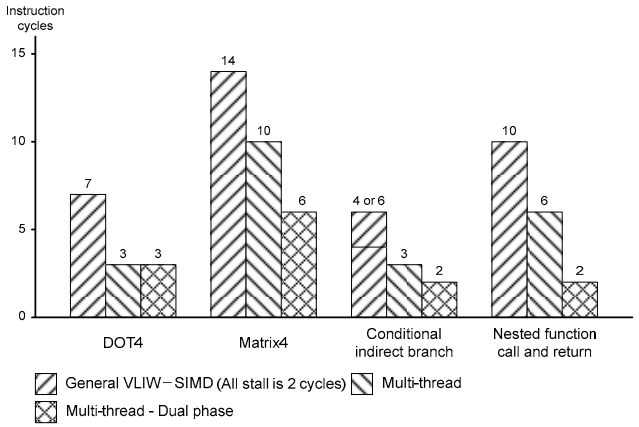


Fig. 14. Architectures' performance comparision

We can conclude that the multi-thread with dual phase operation and variable length instruction words architecture is suitable to sensitive processors for efficient use of ALUs.

References

[1] Liza Fireman, "The Complexity of SIMD Alignment" *Technion – Computer Science Department – M.Sc. Thesis MSC – 2006.*

[2] Mauricio Breternitz, Jr., "Compilation, Architectural Support, and Evaluation of SIMD Graphics Pipeline Programs on a General-Purpose CPU" *Proceedings of the 12th international conference on parallel architectures and compilation techniques.*

[3] H.K. Jeong, "Design of 3D Graphics Geometry Accelerator using the Programmable Vertex Shader" *ITC-CSCC 2006.*

[4] James C. Lelterman, "Learn Vertex and Pixel Shader Programming with DirectX9" *Wordware Publishing, Inc. 2004.*