A Design of a 8-Thread Graphics Processor Unit with Variable-Length Instructions

Kwang-Yeob Lee, Second Jae-Chang Kwak, *Seokyeong University*

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.

Index Terms— 3D Graphics Accelerator, OpenGL ES 2.0, Shader, Multi-thread, Variable Length Instruction.

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 an instruction and the next instruction is long enough to ignore stalls caused with instruction dependence.

In VLIW architecture, unnecessary memory is occupied to store instructions, because of long length instruction format. In order to solve this memory waste problem, a variable length instruction format is proposed. Instructions should be able to cooperate each other to support a variable length instruction format. 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.

Manuscript received March 18, 2008; revised August 18, 2008. Kwang-Yeob Lee[1], Jae-Chang Kwak[2], [1] Dept. of Computer Engineering, Seokyeong University, [2]Dept. of Computer Science, Seokyeong University, Corresponding Author Seo-kyeong Univ. Jeongneung 4-dong, Seongbuk-gu, Seoul, Korea

II. The Proposed Architecture

2.1 Multi-thread processing

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

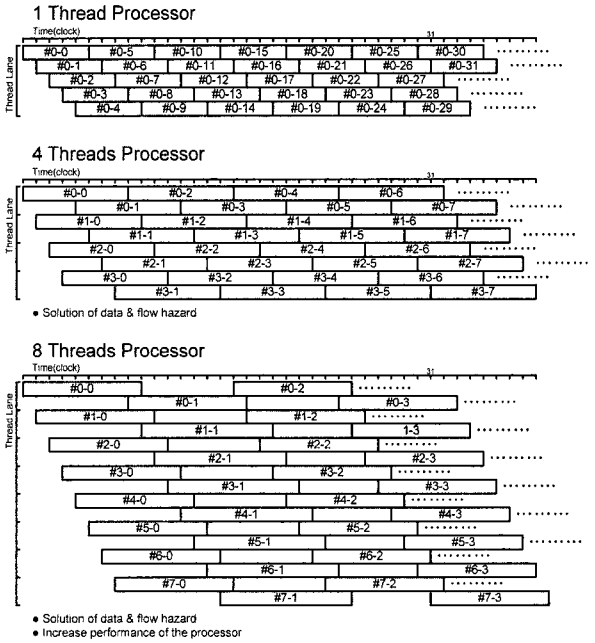


Fig. 1 Multi-thread Processing

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.

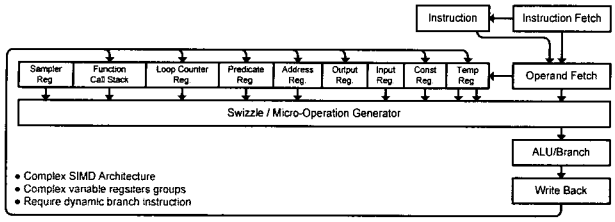


Fig. 2 SIMD(Single Instruction Multiful Data) architecture

Figure 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.

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.

This variable length instruction format can simplify the instruction unit, can improve performance by shorter implementation of complex operations, and can save memory space for long programs

2.3 Dual Phase Architecture

Another problem of SIMD architecture is the waste of various register groups. For example, in Shader 3.0 API of DirectX 9.0, roles of each proposed register group seems to be not overlapped each other as shown in Fig. 3.

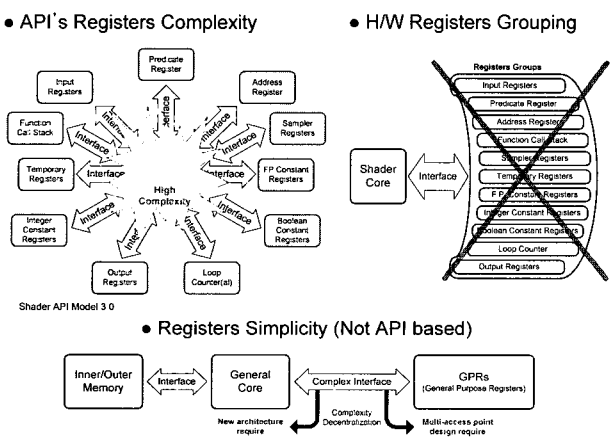


Fig. 3 Registers set requirement of shader model 3.0

But in the point of view of a hardware design, this fact can be considered that various input/output ports are required to many registers. It shows that the hardware design based on the requirements of API as shown in Fig. 3 is inefficient. This problem can be solved simply by designing only one register group.

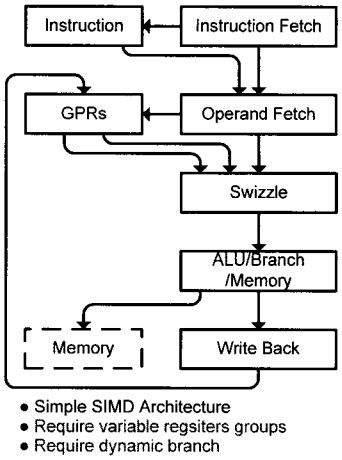


Fig. 4 Simple SIMD architecture

As shown in Fig.4, the design of only one GPRs(General Purpose Registers) can simplify the complexity of registers of a processor. The processor with this design can be simpler than the process supporting all registers of API. There are some limitations in using only one GPRs. In DirectX Shader API, even though every registers are used at the same time, parts of register groups are combined to process a complex expression. For the processor with one GPRs, the complex expression must be divided into many instructions in a macro form, and this can degrade the performance. This degradation is a trade-off for the simplification of the processor.

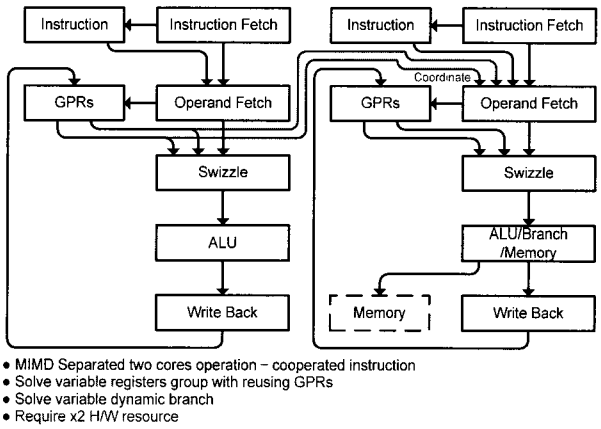


Fig. 5 Cooperated SIMD architecture

Two cooperated processors architecture can be suggested to improve the utilization of registers under the simplification of the processor, as shown in Fig. 5. For the performance improvement, the coordination is connected from GPRs to Operand Fetch among two processors.

With this, it will enlarge registers' scalability and more complex registers operations can be performed even though processor has just one GPRs. Since this suggestion requires twice hardware, now the next step is necessary to integrate two processors into one processor.

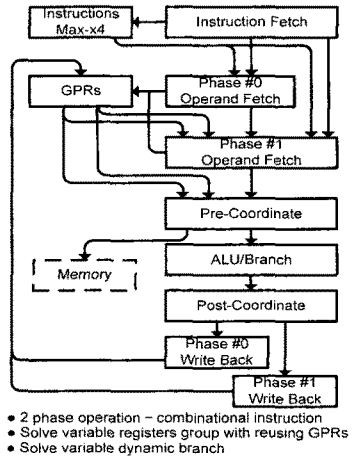


Fig. 6 Dual Phase SIMD architecture

Fig. 6 shows Dual Phase SIMD architecture by removing unnecessary modules and integrating of overlapping modules. This is a new one processor architecture with one GPRs and shared ALU module, whose functionality is identical to two processor architecture shown in Fig. 5.

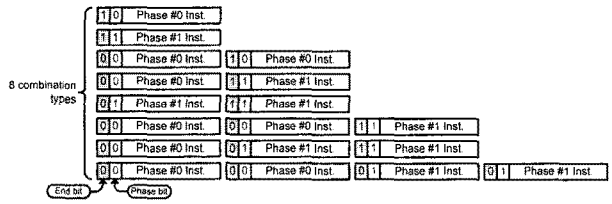


Fig. 7 Phase bit - Dual phase architecture

Fig. 7 represents dual phase instructions for the Dual Phase SIMD architecture. Thousands of instruction combinations are possible implemented by VL-IW structure. These instruction can be executed in maximum two phases, and each phase can contain maximum two instruction fragments. For this representation, additional two bits are located at the upper MSB, as a phase bit and an end bit.

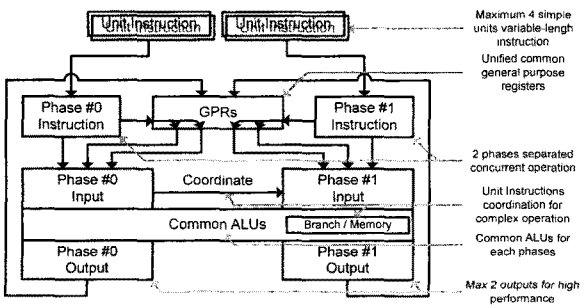


Fig. 8 Dual phase architecture

Fig. 8 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.

The phase #1 instruction can combine instructions for compare, branch, compute, or memory access without any extra exclusive hardware. The phase #1 instruction can execute more complex instructions by receiving the expression from phase #0.

2.4 Combination of instruction fragments

The limitations of this instruction structure will be explained, before showing examples how these complex instructions can be represented. Instructions can be combined with maximum 4 instruction fragments and can perform thousands of operations. Since each phase is shared to maximize the efficiency of ALU, so there are exclusive paring rules as shown in Fig. 9.

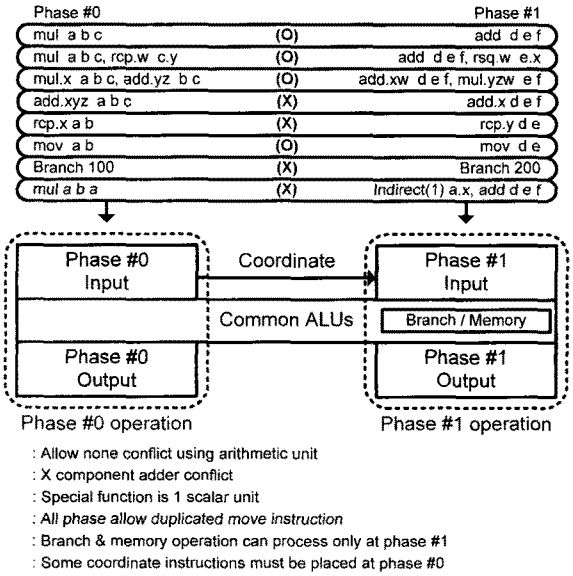


Fig. 9 Exclusive pairing rules

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.

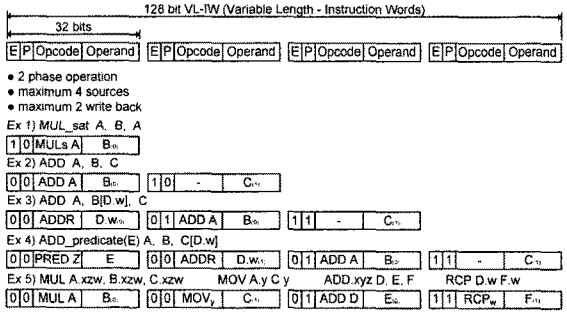


Fig. 10 Arithmetic operation with dual phase VL-IW

Fig. 10 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 presents various arithmetic instruction fragments can be combined into one instruction. This combined instruction can be executed faster.

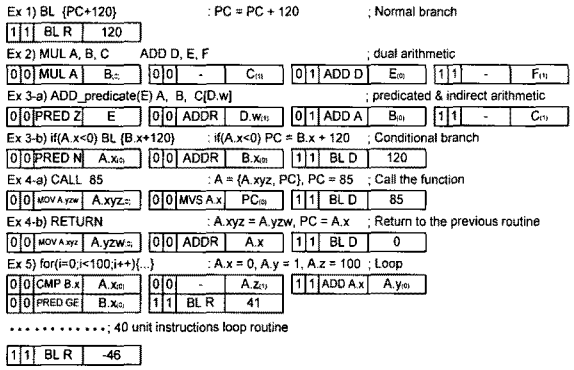


Fig. 11 Branch and looping operation with dual phase VL-IW

Fig. 11 shows a part of representations of arithmetic expressions. As shown in Example 1, the branch instruction exists, but there is no compare branch instruction, such as je or jne, in the x86 system. These instructions can be represented by 'predicate' instruction of phase #0, as shown in Example 3. Call/return instruction can be represented by branch instruction and 'swizzle' instruction of phase #0. Also other instructions, such as indirect branch, nested branch, comparison branch, and looping functions, can be represented in the similar way. This architecture doesn't have an extra instruction penalty.

III. 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. 2.

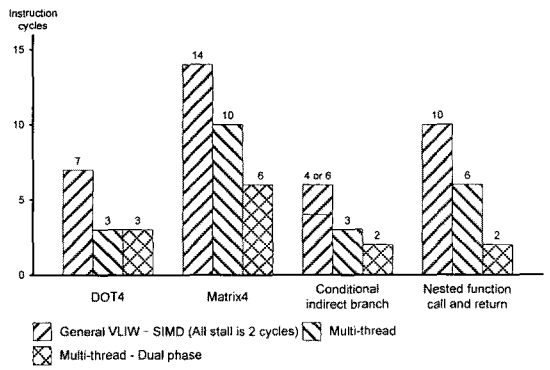


Fig. 12 Architectures' performance comparison

Fig. 2 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.

This multi-thread with dual phase operation and variable length instruction words architecture can provide an efficient performance improvement of ALU. It also can be applicable to the multimedia acceleration. This architecture has an optimal processor architecture with the minimal resources.

ACKNOWLEDGMENT

This work was supported by "Nano IP/SoC Innovative Promotion Group" and "Ministry of Knowledge Economy System IC 2010 Project".

REFERENCES

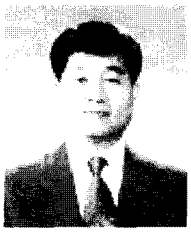
[1] Liza Fireman, "The Complexity of SIMD Alignment" *Technion – Computer Science Department – M.Sc. Thesis MSC – 2006.*

[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.*

Kwang Yeob Lee
studied electronics engineering at Sogang University and Yonsei University from 1979 to 1987. In 1994 he received the Ph.D from the Yonsei University. From 1989 to 1995, he was with Hyundai Electronics as a designer of System LSI. During that time, he was responsible for the design of microcontroller. In 1995, he joined the Department of Computer Engineering , Seokyeong University. His research interests include Embedded System, Mobile 3D Graphics Accelerator, SoC Design.



Jae Chang Kwak
received the B.S degree from Yonsei University in 1983. He received the M.S and Ph.D degrees in computer science from the University of Iowa in 1989 and 1993, respectively. He is currently an Professor of Computer Science at Seokyeong University. His main interests are Network Traffic control, Realtime Scheduling, Embedded System, Mobile Graphics System.