Two Unified Shaders Design

Lan-Da Van (范倫達), Ph. D.
Department of Computer Science
National Chiao Tung University
Taiwan, R.O.C.
Fall, 2018
Source from:

A Dual-Shader 3-D Graphics Processor With Fast 4-D Vector Inner Product Units and Power-Aware Texture Cache

Outline

• Introduction

• System Architecture
  – Unified Shader Model
  – Parallel Processing Level
  – Unified Processor Architecture
  – Exploration of Optimal Datapath

• Unified Shader Core Implementation
  – 4-D Vector Inner Product Unit
  – Configurable Texture Cache

• Implementation Results

• Conclusion
Introduction

• Unified shaders act as vertex shader and pixel shader: used as vertex and pixel processing components of the entire processor, respectively.

• PC graphics architecture: Due to many cores, shaders perform the vertex and pixel operations in parallel to maximize the throughput. However,….

• Mobile device: Due to limitations in silicon area and power consumption, each shader should produce sufficient processing capability within limited power consumption.

• The speed of floating-point computation is the most important issues to determine the overall 3D graphics processor performance. => Instruction Stall
Motivation (1/2)

• Maximize the parallel processing ability within limited silicon area, where main concept:
  • Data level parallelism
  • Instruction level parallelism
  • Task level parallelism

• where architecture features:
  • Four-way SIMD
  • Four threads
  • Two-way VLIW
  • Two cores
Motivation (2/2)

• The overall performance and power of the processor can be determined by vertex transformation capability in vertex processing and power efficient texture operations in pixel processing.
  • Propose a 4-D vector inner product unit.
  • Propose an efficient scheduled shader communication unit (SCU) and configurable texture cache to reduce overall power consumption.

• Performance:
  • 143 Mvertices/s and 2.3 Gtexels/s
  • OpenGL ES 2.0
Unified Shader Model (1/2)
Unified Shader Model (2/2)

• Vertex program transforms and lights input vertex data.

• After gathering these intermediate vertices, a fragment generator makes triangles by groups of three vertices and interpolates pixel colors inside the triangles.

• Pixel program modified the generated pixels by texturing or blending in a pixel program.
Parallel Processing Levels

• Vertex and pixel data level parallelism
  – Between a vertex and a vertex
  – Between a pixel and a pixel

• Shader instruction level parallelism
  – Inside a vertex program
  – Inside a pixel program

• Vertex and pixel task level parallelism
  – Between vertex and pixel programs
Unified Shader Architecture (1/3)
Unified Shader Architecture (2/3)

- Main functional blocks: Two programmable unified shaders, a host interface, a shader communication unit (SCU), a shader scheduler, and a texture cache.
- A host interface unit fetches vertex and pixel data from external memory into a SCU, and sends out the output data of shaders.
- The SCU stores vertex and pixel data, and distributes them to two shaders by control signals from a scheduler.
Unified Shader Architecture (3/3)

- Shaders execute geometry and rendering operations, and write back final results to the SCU.
- The unified shader core is designed to perform both vertex program and pixel program.
- The texture unit, by texture instructions, generates texture address, fetches texture data, and executes texture filtering operations.
- Each shader has its own L1 texture cache, and shares an L2 texture cache.
Computing Resources for Parallel Processing (1/4)

One Shader with 2-way VLIW
Computing Resources for Parallel Processing (2/4)

- Upper eight arithmetic logic units (ALUs) indicate one unified shader, and lower eight represent another shader. These two shaders operate asynchronously, and they communicate using SCU.
- Each shader has two-way VLIW architecture, and each VLIW datapath is composed of two four-way SIMD datapath.
- Since the graphics data are vector colors or vector positions, such as RGBA and XYZW, SIMD datapath receives these four scalar inputs at the same time for parallel processing.
The vertex and pixel register files (VPRF) are storage elements connected to SIMD datapath.

One VPRF has four separated thread registers, and they store different vertex or pixel data. It means that four input data share one SIMD ALU at the same time. Since an SIMD ALU can process only one data at a time, we use a time-multiplexed method to process four input data.

Four data are processed by the single SIMD processor in an interleaved method called as multithreaded method [6].
Computing Resources for Parallel Processing (4/4)

- Multiple data are processed at the same time because there are no dependencies between series of four input data.
- The parallelism of shader instructions is used in the synchronous two-way VLIW architecture.
- Two asynchronous shaders can use the parallelism between vertex and pixel programs. Two shaders can perform independent tasks or same tasks simultaneously.
Processing Sequences (a) SIMD Architecture (b) This work (1/2)

Fig. 3. Processing sequences. (a) SIMD architecture. (b) This work.
Processing Sequences (a) SIMD Architecture (b) This work (2/2)

• In the conventional SIMD architecture, shader instructions are performed sequentially, and only one vertex or pixel can be processed.
• The proposed configuration, on the other hand, can handle eight input data and four programs in parallel. Using multithreaded architecture, since four data V1, V2, V3, V4 are processed in an interleaved order, it hides the latency of the long latency floating-point instructions.
### Example

**Vertex Program**
Grayscale in Table I

| Inst 0 | DP4 | r1.x, v2, c10 |
| Inst 1 | DP4 | r1.y, v2, c11 |
| Inst 2 | DP4 | r1.z, v2, c12 |
| Inst 3 | Nrm | r0.xyz, r1 |
| Inst 4 | Mul | r1.z, r0.z, c15.x |

**Data dependency**

**Complex function**

**Idle data path caused by dependency**

---

### eVLIW [6]

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Slot 1</th>
<th>Slot 2</th>
<th>Slot 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ Inst 0 ]</td>
<td>[ Inst 1 ]</td>
<td>[ Inst 2 ]</td>
<td>[ ]</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
<td>[ Inst 4 ]</td>
</tr>
</tbody>
</table>

**Proposed 2-way VLIW Architecture**

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Slot 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ Inst 0 ]</td>
<td>[ Inst 1 ]</td>
</tr>
<tr>
<td>[ Inst 2 ]</td>
<td>[ Inst 3 ]</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>[ Inst 3 ]</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>[ Inst 4 ]</td>
</tr>
</tbody>
</table>

---

### Without Multi-Threaded Architecture

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ Inst 2 ]</td>
<td>Vertex 0</td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>NOP</td>
<td>NOP</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>Vertex 0</td>
</tr>
</tbody>
</table>

**Data path stall caused by dependency**

---

### With Multi-Threaded Architecture

<table>
<thead>
<tr>
<th>Slot 0</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ Inst 2 ]</td>
<td>Vertex 0</td>
</tr>
<tr>
<td>[ Inst 2 ]</td>
<td>Vertex 1</td>
</tr>
<tr>
<td>[ Inst 2 ]</td>
<td>Vertex 2</td>
</tr>
<tr>
<td>[ Inst 2 ]</td>
<td>Vertex 3</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>Vertex 0</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>Vertex 1</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>Vertex 2</td>
</tr>
<tr>
<td>[ Inst 3 ]</td>
<td>Vertex 3</td>
</tr>
</tbody>
</table>
# Exploration of Optimal Datapath (2/3)

- Four-way SIMD
- Four-threaded
- Two-way VLIW
- Two-cores

<table>
<thead>
<tr>
<th>Test benches</th>
<th>$N_{\text{inst}}$ (VS)</th>
<th>$N_{\text{inst}}$ (PS)</th>
<th>$N_{\text{DP}}$ (VS)</th>
<th>$N_{\text{DP}}$ (PS)</th>
<th>eVLIW[10] utilization (VS) (%)</th>
<th>eVLIW[10] utilization (PS) (%)</th>
<th>Proposed Utilization (VS) (%)</th>
<th>Proposed Utilization (PS) (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grisaille</td>
<td>34</td>
<td>13</td>
<td>14</td>
<td>2</td>
<td>71</td>
<td>36</td>
<td>98</td>
<td>72</td>
</tr>
<tr>
<td>ThinFilm</td>
<td>32</td>
<td>5</td>
<td>14</td>
<td>0</td>
<td>48</td>
<td>42</td>
<td>83</td>
<td>83</td>
</tr>
<tr>
<td>EdgeFuzz</td>
<td>25</td>
<td>38</td>
<td>16</td>
<td>4</td>
<td>63</td>
<td>49</td>
<td>93</td>
<td>89</td>
</tr>
<tr>
<td>DebugTexCoord</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>50</td>
<td>50</td>
<td>75</td>
<td>50</td>
</tr>
<tr>
<td>PlasticD</td>
<td>26</td>
<td>24</td>
<td>16</td>
<td>3</td>
<td>80</td>
<td>43</td>
<td>100</td>
<td>86</td>
</tr>
<tr>
<td>Lambert</td>
<td>32</td>
<td>10</td>
<td>18</td>
<td>1</td>
<td>77</td>
<td>40</td>
<td>94.4</td>
<td>80</td>
</tr>
<tr>
<td>Reflection</td>
<td>26</td>
<td>27</td>
<td>17</td>
<td>4</td>
<td>67.5</td>
<td>50</td>
<td>96.4</td>
<td>92</td>
</tr>
<tr>
<td>Debugspace</td>
<td>37</td>
<td>9</td>
<td>18</td>
<td>0</td>
<td>69</td>
<td>32</td>
<td>95.6</td>
<td>64</td>
</tr>
<tr>
<td>DOOM3</td>
<td>30</td>
<td>23</td>
<td>18</td>
<td>3</td>
<td>50</td>
<td>35</td>
<td>84.2</td>
<td>70.6</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>27.5</strong></td>
<td><strong>16.6</strong></td>
<td><strong>15</strong></td>
<td><strong>1.88</strong></td>
<td><strong>64</strong></td>
<td><strong>41.9</strong></td>
<td><strong>91.1</strong></td>
<td><strong>76.4</strong></td>
</tr>
</tbody>
</table>
• Two-way VLIW, two-shader system is more efficient in performance than four-way VLIW, one-shader system
  – the four-way VLIW architecture cannot fully utilize the instruction level parallelism.
  – Table I (on page 19) shows that the only 42% datapath are used in the pixel operation. In the two-shader system, datapath can be almost fully utilized.
• At the case of more divided shader which is a simply duplicated four SIMD shader system, the datapath utilization is saturated, and area and control overhead is increased.
The proposed SCU uses shared memory to provide efficient data transmission among shaders, and splits it into four banks.

Controlled by a scheduler to reduce power consumption.

The SCU is composed of four-banked SRAMs and a switch network.

A scheduler for shader control is located in the switch network.

Two unified shaders can access the memory banks through the switch network.
Shader Communication Unit (3/4)

shared memory共為4個bank，每個bank有兩種模式：shading mode和I/O mode。
I/O mode時，可以在I/O bus和SCU之間傳輸vertex或pixel；
shading mode時，可以在SCU和shader之間傳輸vertex或pixel。

一開始bank 0~2存放vertex，
bank 3為空。

bank 0為I/O mode，因為shader已經處理完vertex，因為輸出到I/O bus。
bank 1為shading mode，shader (US1)還沒處理完。
bank 2為shading mode，將vertex送給shader (US0)處理。
bank 3為idle，不做任何事，因為只有兩個shader。

bank 0為shading mode，將vertex送給shader (US0)處理。
bank 1為shading mode，將vertex送給shader (US1)處理。
bank 2為idle，不做任何事，因為只有兩個shader。
bank 3為I/O mode，從I/O bus傳進vertex。

Fig. 5. Shader communication unit. (a) Shader communication unit architecture. (b) Processing sequence.
In this system, the scheduling obeys the following rules.

The shader can access any bank if all banks are filled with either vertex or pixel. A vertex bank is marked by V, and a pixel bank is marked by P.

If some banks are filled with pixels, they have a higher priority to be accessed by the shader because the pixel output is the final results of operations.
Unified Shader Architecture
Proposed Floating-Point 4-D Vector Inner Product Unit (1/2)
Proposed Floating-Point 4-D Vector Inner Product Unit (2/2)
Configurable texture cache. (a) Overall architecture. (b) Cache management Circuit. (c) Bank selection by additional index fields
Performance Evaluation (1/3)

Fig. 10. Performance evaluation. (a) Architectural performance comparison. (b) Chip test comparison.
Performance Evaluation (2/3)

- JSSC 08 [7]:
  - Matrix Operation
  - Arithmetic Operation
  - Nop from dependency

- JSSC 07 [6]:
  - Matrix
  - Arithmetic

- ISSCC 07 [5]:
  - Matrix
  - Arithmetic
  - Nop

- Four-threaded Dual-Core Dual-issue VLIW with Fast Arithmetic Unit:
  - Matrix
  - Arithmetic

- Fixed & Floating point & 100Mhz

- Floating point & 100Mhz

- Logarithmic number system & 200Mhz

- Floating point & 100Mhz

- This Work

- Floating point & 143Mhz

- latency / vertex (ns)
Performance Evaluation (3/3)

Normalized P-D Product

- Grisaille
- ThinFilm
- EdgeFuzz
- DebugTexCoords
- PlasticD
- Lambert
- Reflection
- DebugSpaces
- DOOM3
- Average

Legend:
- SIMD [7]
- SIMD-Logarithmic number system [5]
- Expanded VLIW [6]
- Proposed system

Values:
- Grisaille: 0.52
- ThinFilm: 0.92
- EdgeFuzz: 0.61
- DebugTexCoords: 0.39
- PlasticD: 1.0
- Lambert: 0.8
- Reflection: 0.7
- DebugSpaces: 0.6
- DOOM3: 0.5
- Average: 0.7
Comparison Results (1/2)

![Diagram showing comparison results with Peak Performance and Power metrics. The diagram includes bars for JSSC07 [6], ISSCC07 [5], JSSC08 [7], and This Work with values 4.5, 4.9, 0.79, and 7.72 respectively. There is a 58% improvement indicated.]

*1) \[
\frac{\text{Peak Performance}}{\text{Power}} = \frac{\text{Mvertices/s} \cdot \text{Mtexels/s}}{\text{mW (geometry)} \cdot \text{mW (rendering & texture)}}
\]

*2) We assumed that JSSC07 [6] has the same rendering engine with this work.
## Comparison Results (2/2)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation Frequency</td>
<td>200/50</td>
<td>100</td>
<td>100</td>
<td>143</td>
</tr>
<tr>
<td>(Vertex/Pixel) (Mhz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Consumption</td>
<td>86.8/66.2</td>
<td>157 (Vertex only)</td>
<td>68 (Both)</td>
<td>367 (full operation) 282/151</td>
</tr>
<tr>
<td>(Vertex/Pixel) (mW)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peak Performance</td>
<td>141/200</td>
<td>120 (Vertex only)</td>
<td>9.1/400</td>
<td>143/2300</td>
</tr>
<tr>
<td>(Mvertices/s / Mtexels/s)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmability</td>
<td>Vertex</td>
<td>Vertex</td>
<td>Vertex/Pixel</td>
<td>Vertex/Pixel Shading</td>
</tr>
<tr>
<td></td>
<td>Shading</td>
<td>Shading</td>
<td>Shading</td>
<td></td>
</tr>
<tr>
<td>Process Technology</td>
<td>0.18μm</td>
<td>0.18μm</td>
<td>0.13μm</td>
<td>0.18μm</td>
</tr>
<tr>
<td>Datapath Type</td>
<td>logarithmic</td>
<td>Float</td>
<td>Fixed/Float</td>
<td>Float</td>
</tr>
<tr>
<td>Main Features</td>
<td>Logarithmic datapath &amp; DVFS</td>
<td>Extended- VLIW architecture</td>
<td>Pixel-vertex multithreading &amp; light engine</td>
<td>2way-VLIW &amp; 2-Core 4D vector datapath Power aware texture</td>
</tr>
</tbody>
</table>
## Chip Implementation

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Technology</td>
<td>0.18um CMOS</td>
</tr>
<tr>
<td>Power Supply</td>
<td>1.8V</td>
</tr>
<tr>
<td>Chip Size</td>
<td>4.5mm x 4.5mm</td>
</tr>
<tr>
<td>Operating Frequency</td>
<td>143 Mhz</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>367mW</td>
</tr>
<tr>
<td>Transistor counts</td>
<td>3.4M logic, 14KB on-chip SRAM</td>
</tr>
<tr>
<td>Supported Shader Standard</td>
<td>OpenGL ES 2.0 Shader Model 3.0</td>
</tr>
<tr>
<td>Performance</td>
<td></td>
</tr>
<tr>
<td>Vertex Shading</td>
<td>143Mvertices/s (Geometry Transformation)</td>
</tr>
<tr>
<td>Pixel Shading</td>
<td>2.3Gtexels/s (Trilinear Filtering)</td>
</tr>
</tbody>
</table>
Measurement Results

<table>
<thead>
<tr>
<th></th>
<th>64mW</th>
<th>110mW</th>
<th>108mW</th>
<th>85mW</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCU</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shader 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shader 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Texture Unit</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Power breakdown

Measured waveform by logic analyzer
Conclusion

• A 3-D graphics processor based on dual-core dual-issue VLIW architecture and shader core optimization techniques is proposed.

• To achieve both high performance and low power, we used four key approaches:
  – A two-core and two-issue system for high datapath utilization
  – An shader communication system for scalability and power reduction
  – An IEEE 754 compliant floating point (FP) arithmetic unit with reduced delay for fast matrix multiplication
  – A configurable texture cache to reduce the energy consumption.