Vertex Shader Design

Lan-Da Van (范倫達), Ph. D.
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan
Summer, 2017
Source from:

A 155-mW 50-Mvertices/s Graphics Processor With Fixed-Point Programmable Vertex Shader for Mobile Applications

Outline

• Introduction
• System Architecture
• System Operations
  – Data Transfer Flow
  – Dual Operations
• Programmable SIMD Vertex Shader
  – Internal Architecture
  – Pipeline Structure
  – Fixed-Point Graphics Processing
  – SIMD Multiply Engine
Introduction (1/2)

• For wireless applications, the low power consumption is the most important issue because of limited battery lifetime.

• The reduced instruction set computer (RISC) architecture are widely used as the main platforms for wireless applications because of their high MIPS/watt.

• However, these low-power RISC platforms have very limited system resources in terms of computation power and memory bandwidth.
Moreover, since users are watching 3-D graphics images on a small screen very closely to their eyes, the graphics processor must generate high quality of graphics images with advanced graphics algorithms. These increase image quality and/or lower memory bandwidth usage, and thus also power consumption.

In the previous works, the graphics processors did not integrate vertex shader and showed lack of processing parallelism for streaming graphics data.

In this work, we designed and implemented a graphics processor with programmable fixed-point single-instruction-multiple-data (SIMD) vertex shader for mobile applications.
System Architecture (1/2)

- Block diagram of the graphics processor
System Architecture (2/2)

• The vertex shader is implemented as an ARM-10 co-processer and processes all per-vertex operations such as geometry transformation and lighting calculation.
• The primitive assembly such as clipping and culling is also performed by the vertex shader in collaboration with the RISC processor.
• The rendering engine is responsible for the rasterization and the per-pixel operations such as pixel blending and texture mapping.
• The rendering engine instruction is composed of transformed vertex coordinates, texture coordinates and lit vertex color.
• The PFS (Programmable Frequency Synthesizer) reduces the dynamic power consumption of the chip by clock gating and frequency scaling (skip here!).
A. Data Transfer Flow

- The whole system performance of graphics hardware depends not only on the performance of the individual hardware acceleration blocks but also on the communication cycles for transferring the graphics data between external memory and the graphics hardware.
In this data flow, hardware-instruction path and graphics-data path are separated from each other to improve the streaming processing within allowed memory bandwidth.
B. Dual Operations

- (a) **Tightly Coupled Co-processor** (TCC) state
- (b) **Parallel Processor** (PP) state
• (a) **Tightly Coupled Co-processor (TCC) state**
  
  They do not affect the memory and registers unless the arithmetic flags (negative, zero, carry out and overflow) of the ARM-10 processor satisfy a condition specified in the instructions.

  In the vertex shader, SIMD control flags such as arithmetic flags, saturation, overflow and underflow are updated after execution of every SIMD data processing instructions and can be moved to program status register (PSR) of the ARM-10 processor.

  The general SIMD instructions such as arithmetic and movement operations are implemented in the TCC state, performing clipping and back-face culling operations in 3-D graphics pipeline.
• (b) **Parallel Processor (PP) state**
  
  – In this state, the vertex shader behaves like an independent processor and it does not need any control from the ARM-10 processor.
  
  – The PP state has a separate graphics instruction set different from the general SIMD instructions of the TCC state.
System Operations (6/7)

- Processor for synchronization. The vertex shader executes the independent vertex program codes while the ARM-10 processor performs its main application program or enters even into cache miss.

- Various user-defined vertex processing operations such as geometry transformation and lighting calculations can be performed for the current vertex input while next vertex data is fetched from the ARM-10 processor.

- To maintain the communication protocol of the ARM-10 co-processor interface, the vertex shader drives co-processor busy (CPbusy) signal to the ARM-10 processor in the PP state, blocking next co-processor instruction from the ARM-10.
The graphics instruction set is the subset of the general SIMD instructions with graphics extensions such as source swizziling and write-masks.

In the PP state, there are more register file sets that can be used as input operands of instructions in programmer’s model.

The general SIMD instructions in the TCC state can also accelerate various multimedia functions beyond 3-D graphics such as MPEG-4 video.
Programmable SIMD Vertex Shader (1/4)

Graphics/matrix constants
• In the control part, there is a 2 kB code memory that stores vertex program codes of graphics instructions.

• **Vertex program control unit (VPCTRL)** issues the graphics instructions without control of the ARM-10 processor.

• The contents of **control register** determine its operating state.

• The two operating states—the TCC state and the PP state—share all of the hardware blocks except instruction fetch units.
Programmable SIMD Vertex Shader (3/4)

- Fixed-point vector unit is responsible for all SIMD arithmetic operations such as addition and multiplication.
- Special function unit (SFU) is responsible for reciprocal (RCP) and reciprocal square root (RSQ) operations.
- Display buffer, implemented as a 32 kB synchronous SRAM, stores graphics constants such as transformation matrix, lighting parameters and lookup table entries.
  - Ex. of lighting parameters: normal vectors, view vectors, light vectors
• For streaming graphics processing, the vertex shader contains multiple register files:
  – **Vertex Input Registers (VIR)**
    • Hold the vertex attributes such as position and normal vector
    • Feed into the fixed-point SIMD datapath
  – **Vertex Output Registers (VOR)**
    • There are three output vertex register files for caching of vertex data in the primitive assembly and only one of them is accessible in the vertex program
  – **Vertex General SIMD Registers (VGR)**
    • Store temporary results during vertex program execution
Pipeline Structure (1/2)
- For programmable shading, operands of the SRAM display buffer and the SIMD register files are accessed at the same time in the decode stage.
- In the execution stage, there are three separated pipelines:
  - SIMD arithmetic-and-logic (ALU) pipeline
  - SIMD multiply pipeline
  - SFU pipeline
- To reduce the design complexity, register forwarding logic between pipeline stages is used only in the general SIMD register file.
• $Q_{m.n}$ represents the format of a fixed point number, where ‘$m$’ represents the number of bits used for integer part and ‘$n$’ represents for that of fraction part.
In this work, fixed-point number representation is used instead of floating-point number format.

Simple integer datapath of fixed-point unit can achieve higher clock frequency while consuming less power than floating-point unit, yielding total energy reduction.

To improve the usefulness of the fixed-point arithmetic operations, it is designed such that hardware status registers automatically indicate the overflows and the underflows occurred in the multiplications of two fixed-point numbers.

These status registers can be used to check errors in the fixed-point arithmetic without extra cycle penalties and degradation of SIMD parallelism.
Fixed-Point Graphics Processing (3/3)

- Block diagram of SIMD ALU
Since multiplication-equivalent instructions spend most of time in graphics operations, the throughput of fixed-point MAC operations is designed as a single cycle. In addition, fast 4-cycle matrix transformation (TRFM) is implemented.

- MUL x
- MAC y
- MAC z
- MAC w
However, fixed-point MUL and MAC operations require two cycle integer multiplications and two cycle integer additions, leading to 4-cycle latency.

To resolve data dependency between these MUL and MAC operations, it is allowed that intermediate value of the integer multipliers can be bypassed to accumulator input of the integer adders in the SIMD multiply engine.
By this scheme, the graphics processor shows 50 Mvertices/s peak graphics performance for parallel projection at 200 MHz operating frequency.
• Hardware architecture of single fixed-point multiplier unit in the SIMD multiply engine
Instruction-level Power Management
Rendering Engine and Clock Gating
Performance Results (1/3)

### TABLE 1
**CHARACTERISTICS OF THE GRAPHICS PROCESSOR**

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Technology</td>
<td>0.18 um 6-Metal CMOS</td>
</tr>
<tr>
<td>Power Supply</td>
<td>1.8V (core), 3.3V (I/O)</td>
</tr>
<tr>
<td>Transistor Counts</td>
<td>2M Logic, 96K SRAM</td>
</tr>
<tr>
<td>Die Size</td>
<td>4.8mm x 4.8mm (core), 6.0mm x 6.0mm (chip)</td>
</tr>
<tr>
<td>Operating Frequency</td>
<td>Fast: ~200MHz/50MHz, Normal: ~100MHz/25MHz, Slow: ~50MHz/12.5MHz</td>
</tr>
<tr>
<td>Power Consumption</td>
<td>&lt;155mW</td>
</tr>
<tr>
<td>Package</td>
<td>256 pin PBGA</td>
</tr>
</tbody>
</table>

**General**
- 1000MIPS (ARM and vertex shader)
- 80MFLOPS (software emulation)

**Geometry**
- 50M vertices/s
- (Geometry transformation)

**Rendering**
- 50M pixels/s, 200M texels/s
- (Bilinear MIPMAP filtered pixel)

**Full 3D Pipeline**
- 3.6M polygons/s (sustaining)
- (Including full OpenGL lighting, clip check and texturing)

**Programmability**
- Vertex program version 1.1 compatible

**Screen Resolution**
- up to 512 x 512 pixels

**Triangle Setup**
- Hardware-accelerated triangle setup engine

**Shading**
- Gouraud / Flat

**Texture Mapping**
- Point/Bilinear MIPMAP filtering

**Antialiasing**
- x2, x4
Performance Results (2/3)
Performance Results (3/3)
Conclusion

• A low-power graphics processor is designed and implemented co-processor architecture with dual operations.
  – Fixed-point graphics processing
  – Instruction-level power management of the vertex shader
  – Pixel-level clock gating of the rendering engine