Accelerating Graphic Rendering on Programmable RISC-V GPUs

Blaise Tine, Varun Saxena, Santosh Srivatsan, Joshua R. Simpson, Fadi Alzammar, Liam Paul Cooper, Sam Jijina, Swetha Rajagopalan, Tejaswini Anand Kumar, Jeff Young, Hyesoon Kim
Abstract

Graphics rendering remains one of the most compute-intensive and memory-bound applications of GPUs and has been driving their push for performance and energy efficiency since its inception. Early GPU architectures focused only on accelerating graphics rendering and implemented dedicated a fixed-function rendering units. Today’s GPUs have become more programmable to address the complexity and diversity of modern graphics workloads while still accelerating several components of the graphics pipeline in fixed-function hardware.

Generalizing the GPU microarchitecture and implement some of its graphics hardware blocks in software can save area that can be used to expand the generic pipeline, especially in mobile systems-on-chips environments where power and area is scarce.

In this work, we propose a RISC-V-based hybrid GPU architecture that accelerates the graphics pipeline without paying the cost of a full hardware graphics pipeline. We evaluated the design on an Altera Arria 10 FPGA running at 200 MHz.
Motivations

GPIO Acceleration for edge computing
- GPU has many applications
  - e.g. Graphics, ML, Crypto, graphs, etc.
- General pipeline optimization improve all applications
- Specialized components improve single applications
- Fixed-function area is mainly for graphics
- Can we trade graphics area for more cores?
Motivations (2)

Research in GPU Hardware Architecture

- Graphics hardware research beyond simulation
- Full stack open-source framework
- RISC-V ISA extension for graphics
- Open-source Vulkan software stack
GPU Framework Overview

- Graphics Applications
  - 3D & 2D content
- Vulkan Runtime
  - Graphics APIs
- Vortex Vulkan Render
  - Shader compiler
  - Kernel scheduling
- GPU Processor
  - Rasterizer
  - Render Output
  - Core cluster
3D Graphics Pipeline Stages

<table>
<thead>
<tr>
<th>Back-end (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Vertex shading</td>
</tr>
<tr>
<td>• Primitive assembly</td>
</tr>
<tr>
<td>• Clipping</td>
</tr>
<tr>
<td>• Culling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Front-end (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Triangle setup</td>
</tr>
<tr>
<td>• Rasterization</td>
</tr>
<tr>
<td>• Interpolation</td>
</tr>
<tr>
<td>• Early-Z</td>
</tr>
<tr>
<td>• Pixel shading</td>
</tr>
<tr>
<td>• Depth/Stencil</td>
</tr>
<tr>
<td>• Blending</td>
</tr>
</tbody>
</table>
Hybrid 3D graphics Pipeline Stages

**Back-end (CPU)**
- Vertex shading
- Primitive assembly
- Clipping
- Culling
- Triangle setup

**Front-end (GPU)**
- Rasterization
- Interpolation
- Early-Z
- Pixel shading
- Depth/Stencil
- Blending
Graphics Hardware Microarchitecture

- **Command Processor**
  - CPU-GPU communication

- **DCRs**
  - Configuration registers
  - CPU driven

- **Raster Unit**
  - Triangle rasterizer
  - Tile-based

- **ROP Unit**
  - Depth/Stencil
  - Blending
  - Logic Op

- **Texture Unit**
  - Texture sampling
Rasterizer Unit

- **Raster Slices**
  - Tile Evaluators
  - Block Evaluators
  - Quad Evaluators

- **Memory Unit**
- **Configuration Registers**
- **Raster Cache**
Render Output Unit

<table>
<thead>
<tr>
<th>Rop Slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Depth Stencil</td>
</tr>
<tr>
<td>• Blending</td>
</tr>
<tr>
<td>• Logic Op</td>
</tr>
<tr>
<td>• Memory Unit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROP Cache</td>
</tr>
</tbody>
</table>

![Diagram showing Render Output Unit components]

- Core 0
- Core 1
- Core 2
- Core n
- Input Crossbar
- ROP Slice(s)
- ROP Unit
- ROP Cache
- Deadlock Handler
- Depth/Stencil Unit
- Blend Unit
- Logic Op
- ROP Slice

- DCRs (Data Control Registers)
Vulkan Software Render

Vulkan Runtime Render
- Vulkan runtime API
- JIT compilation on host CPU
- Shader to RISC-V compilation
- Graphics extension Calls
- Kernel scheduling
Evaluation

FPGA Setup

- Intel Arria 10 FPGA
- Up to 8 cores (128 threads)
- Caches: 16KB I\$, 16KB D$
- Shared memory: 16 KB
- Memory bandwidth: 16 GB/s
- Fmax 200 MHz

Evaluated 3D Demos
Evaluation

Performance
- Texture FillRate: 3.2 GT/s
- Pixel FillRate: 1.6 GT/s
- Avg frame rate: ~10 fps
- Scaling across apps
Relative area cost in single-core configuration

Texture Unit: 13%
Raster Unit: 11%
Rop Unit: 8%
References

| Website:  | https://vortex.cc.gatech.edu |
| Github:   | https://github.com/vortexgpgpu/vortex |

Thank you!