An Effective Task Scheduling Scheme for Multi-core Tile based Rendering GPU

Won-Jong Lee, Seok-Yoon Jung, Shi-Hwa Lee
Embedded Multimedia Group, System Architecture Lab.
SAIT, SAMSUNG ELECTRONICS Co., Ltd.

HIGH-PERFORMANCE GRAPHICS
PARIS, FRANCE JUNE 25-27, 2012

Motivation

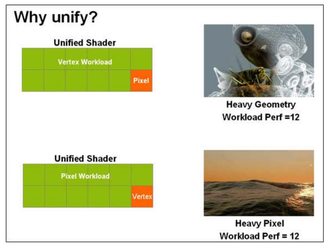
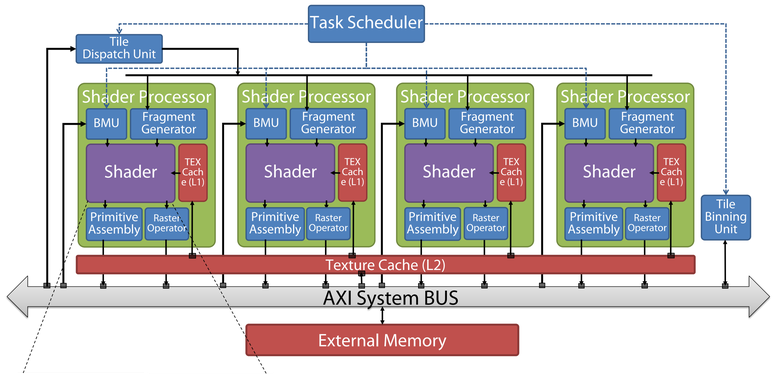
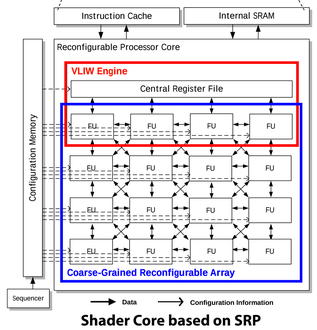


Image courtesy NVIDIA

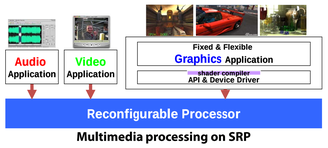
- Modern GPUs have employed unified shaders, which allow more flexible use of the graphics rendering hardware. This trend has recently been realized in mobile GPUs [1][2][3].
- For the best GPU performance, the utilization of unified shader cores and fixed-function hardware should be maximized while maintaining load balancing among multiple cores.
- In order to satisfy this requirement, we propose an effective task-scheduling scheme, which has the following two key features: 1) a shader-interleaving scheme that can hide latencies by exclusively executing different kernels, and 2) a task slot-based dynamic load-balancing scheme that evenly distributes workloads to the multiple cores.

Multi-core Unified Shader Architecture





Shader Core based on SRP

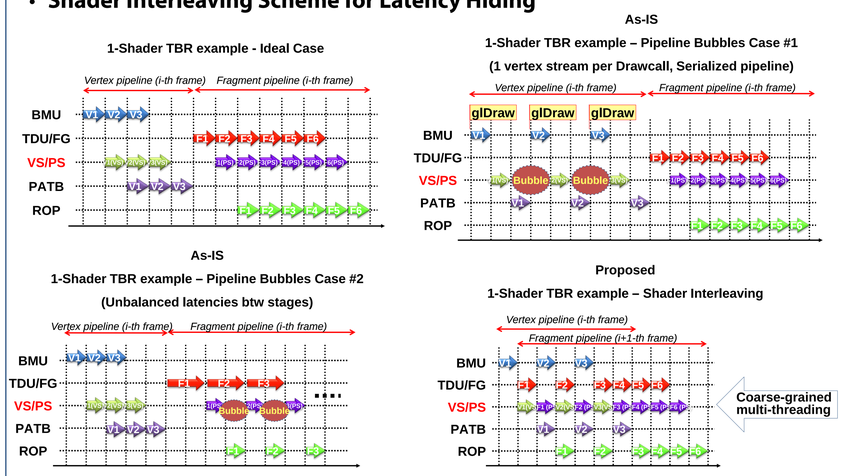


Multimedia processing on SRP

- Our multi-core unified shader architecture, which is an improvement over our previous work, the SRP (Samsung Reconfigurable Processor)-based GPU [4].
- Multiple unified shaders can be easily implemented as the SRP is reconfigurable. Each core includes vertex and pixel processing hardware in order to handle both types of data. Specifically, these units are batch management units (BMUs), primitive assembly (PA) units, tile dispatch units (TDUs), fragment generators (FGs), and raster operators (ROPs). The tile-binning unit (TBU) is individually equipped for parallel processing.
- The task scheduler (TS) is a newly designed unit responsible for dynamic load balancing and shader interleaving

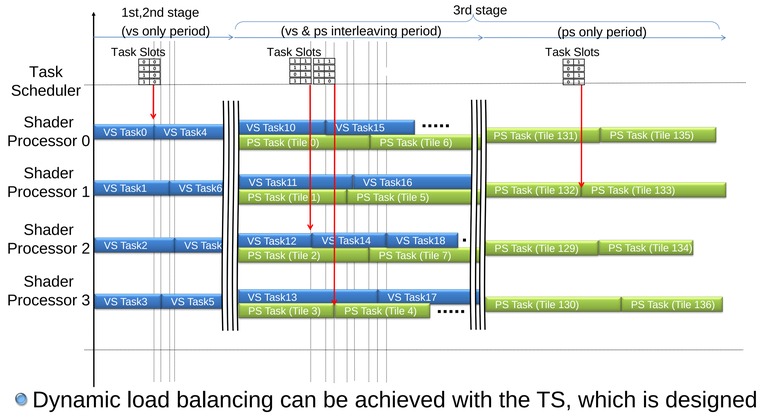
Latency Hiding and Load Balancing

- **Shader Interleaving Scheme for Latency Hiding**



- The GPU drives the fixed-function hardware and the shader cores to execute in a fully pipelined manner. Unfortunately, this can cause pipeline bubbles because of inconsistent latencies between the hardware and the shader stage. If the kernel running on the shader core finishes before the hardware stage, the shader core remains idle until the next stream arrives.
- In order to hide these latencies, we propose a *shader interleaving* that allows different kinds of kernels to run exclusively. When a certain kernel (e.g., the vertex shader) becomes idle, the other kernel (e.g., the pixel shader) is immediately assigned to the shader core and executed without any delays. This can greatly improve overall throughput, which in turn leads to faster execution.

- **Task Scheduler for Dynamic Load Balancing**



- Dynamic load balancing can be achieved with the TS, which is designed for creating, scheduling, and assigning tasks to the shader cores. The TS first accepts graphics commands from hosts, generates the tasks on a per unit (e.g., a drawcall or a tile) basis, and then schedules them to the idle cores.
- *Task slots* indicating the status of each core are defined and used to select a target core in the TS. The TS also takes into consideration the fact that different tasks (e.g., vertex, pixel) should be interleaved as much as possible for hiding latency. Above figure shows the operational flow of the scheduled tasks executing on four unified shaders. Scheduled tasks are processed in parallel in-between frames, which leads to dynamic load balancing, as shown in the figure.

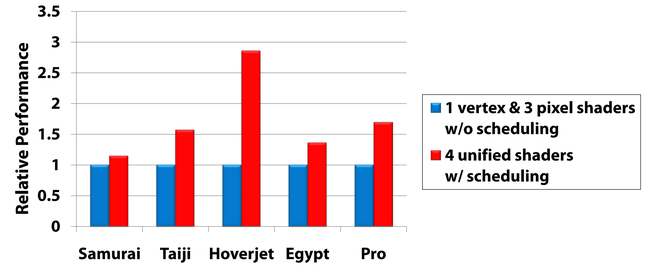
Results

- **Implementation**

- The prototype GPU, including the proposed scheduler, was verified and evaluated by cycle-accurate simulation, RTL simulation, and FPGA targeting. The GPU was synthesized for a Xilinx Vertex6 FPGA board running at 25Mhz.

- **Benchmarks**
  - Rightware's 3DMark Mobile ES [5] and GLBenchmark [6] were used as test benchmarks. Various benchmarks having different workload characteristics (e.g., Taiji and Egypt: pixel-intensive, Hoverjet: vertex-intensive) were selected for testing unified shader efficiency.

- **Performance Evaluation**



- Above figure compares the performance of our GPU (4 unified shaders) with that of a GPU with non-unified shaders (1 vertex and 3 pixel shaders) [4] and having the same number of cores. Our GPU, which adopts a scheduling scheme, is 1.2 to 2.8 times faster, thanks to the effective combination of latency hiding and dynamic load balancing.
- For better performance, we are currently developing a more accurate scheduling algorithm that supports multi-threading. Finally, it is expected that our GPU, including the proposed scheduler, will be a core intellectual property for future application processors.

References

[1] Imagination, PowerVR SGX series, http://www.imgtec.com/powervr/sgx_series5.asp (2012)
[2] ARM, Mali T658, http://www.arm.com/products/multimedia/mali-graphics-hardware/mali-t658.php (2012)
[3] Qualcomm, Adreno series, https://developer.qualcomm.com/discover/chipsets-and-modems/adreno (2012)
[4] W.-J. Lee et al, "A Scalable GPU Architecture based on Dynamically Embedded Reconfigurable Processor," HPG 2011: ACM Conference on High-Performance Graphics, (Poster), Vancouver, Canada (2011).
[5] RightWare, benchmarking software, http://www.rightware.com/en/Benchmarking+Software (2011)
[6] GLBenchmark, http://www.glbenchmark.com (2011)

joe.w.lee@samsung.com