ACACES A Decoupled Access/Execute Architecture for Mobile GPUs

Jose-Maria Arnau[*,1],
Joan-Manuel Parcerisa[*,1],
Polychronis Xekalakis[†,2],

[*] *Computer Architecture Department, Universitat Politecnica de Catalunya*
[†] *Intel Barcelona Research Center, Intel Labs Barcelona*

ABSTRACT

Smartphones are emerging as one of the fastest growing markets, providing enhanced capabilities every few months. However, supporting these hardware/software improvements comes at the cost of reducing the operating time per battery charge. The GPU is only left with a shrinking fraction of the power budget, but the trend towards better screens will inevitably lead to a higher demand for improved graphics rendering. In this paper, we focus on improving the energy efficiency of the GPU. We show that the main bottleneck for graphical applications are the pixel/texture caches and that traditional techniques for hiding memory latency (prefetching, multithreading) do not work well or come at a high energy cost. We thus propose the migration of GPU designs towards the decoupled access/execute concept. Furthermore, we reduce bandwidth usage by exploiting inter-core data sharing. The end design achieves 93% of the performance of a heavily multithreaded GPU while providing 34% energy savings.

1 Introduction

The fast growing of the smartphones market is fueled by the end-user demand for a continuously improved mobile computing experience. Current smartphones excel in providing a real web browsing experience, allow for video and picture editing and support a plethora of 3D games. However, supporting all these capabilities comes at a high energy cost, which in turn results in fairly short operating times per battery charge. Figure 1a shows the energy consumed by the main components of a smartphone along with the capacity of the battery over the past years. Most of the energy is consumed today by the communication

[1]E-mail: {jarnau,jmanel}@ac.upc.edu
[2]E-mail: polychronis.xekalakis@intel.com

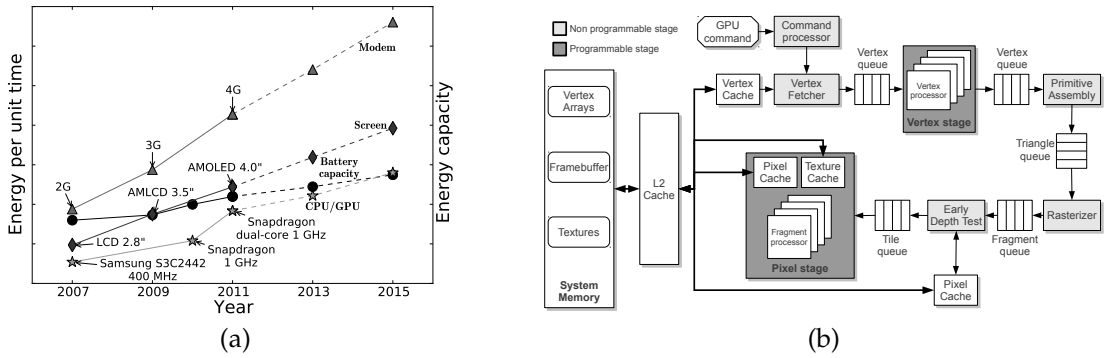
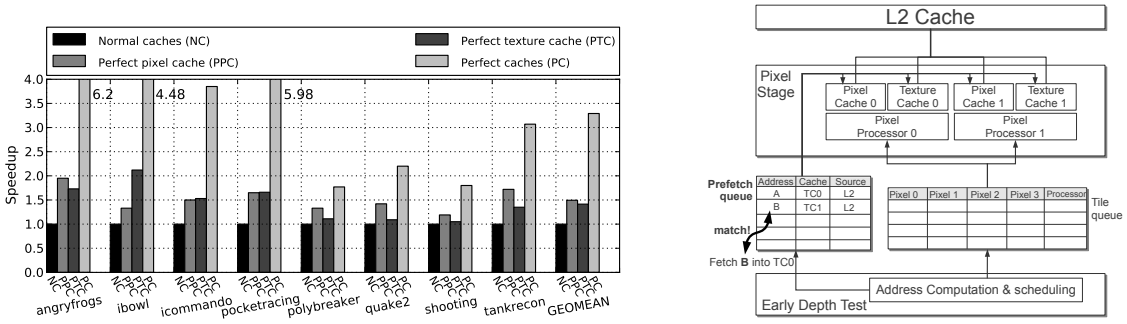


Figure 1: (a) Energy required by the main components of a smartphone and the battery capacity over the past years. (b) Microarchitecture of a GPU similar to the one in the NVIDIA Tegra 2 chipset.

system and the screen, and there is a growing disparity between the energy required and the energy available. The GPU is only left with a small fraction of the power budget, however, the trend towards better screens will lead to a higher demand for improved graphics rendering, that is, higher performing GPUs. In this paper we focus on improving the energy efficiency of the GPU. Due to the highly parallel nature of GPU applications , we believe that the main complication that will have to be dealt with is how to bridge the memory gap in an energy efficient manner. We show that the main bottleneck for graphical applications are the pixel/texture caches and that traditional techniques to hide the memory latency, such as prefetching or multithreading, do not work well or come at a high energy cost. We thus propose to employ a Decoupled Access/Execute (DAE) [Smi82] design for the pixel processors of a mobile GPU, and we introduce a mechanism to exploit inter-core data sharing at a low energy cost on top of the DAE architecture. By using commercial Android applications we demonstrate that our system achieves 93% of the performance of a highly threaded GPU, but providing 34% energy savings. The remainder of this paper is organized as follows: section 2 provides background on the baseline GPU and section 3 describes the DAE system and presents the experimental results.

2 Background

The mobile GPU assumed throughout this paper closely tracks the ultra-low power GeForce GPU in the NVIDIA Tegra 2 chipset (figure 1b). In this GPU, the application OpenGL ES stream is first converted by the OpenGL ES driver to a sequence of GPU commands. When the *Command processor* receives a command, it installs the vertex and pixel programs and triggers the GPU pipeline so the input batch of vertices is processed. First, vertices are transformed and shaded in the vertex processors, based on the programmed by the user vertex program. Next, the resulting triangles are rasterized, i. e., converted to a set of pixels. Finally, pixels are processed in the pixel processors based on the programmed by the user pixel program. A vertex/pixel processor consists on a simple in-order pipeline and Simultaneous Multithreading is employed to boost performance by interleaving the execution of groups of threads or **warps**. Regarding the memory hierarchy, the GPU includes several specialized first level caches (vertex, pixel, and texture caches) and a second level cache on-chip.



(a) (b)

Figure 2: (a) The memory hierarchy is one of the main bottlenecks in a mobile GPU. (b) Decoupled Access/Execute architecture for the pixel processors of a mobile GPU.

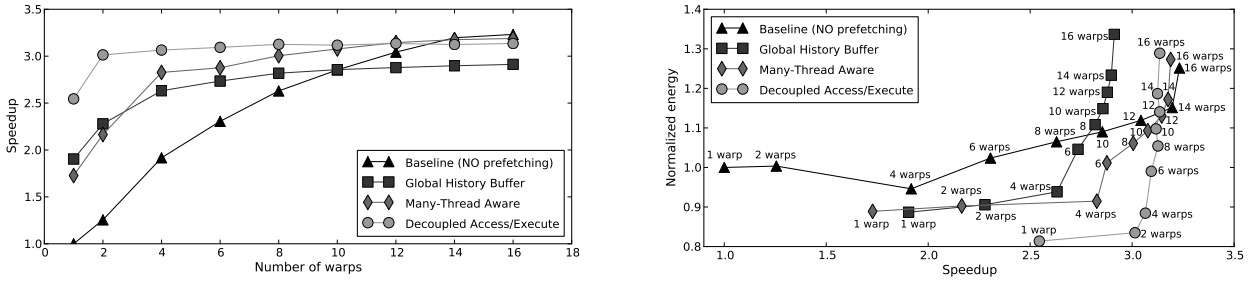
3 Decoupled Access/Execute on a Mobile GPU

We have evaluated the behavior of the memory hierarchy of a mobile GPU, like the one shown in figure 1b, by using 8 commercial Android games that provide a representative mixture of mobile graphical applications. We run Android on top of an emulator, QEMU, and we have instrumented the OpenGL ES Android driver so we obtain information about the rendering process (instructions and memory addresses) that we feed to our trace-driven cycle-accurate GPU simulator. The results for a single-threaded GPU are shown in figure 2a. If no other mechanism than the caches is employed to hide the memory latency, there is still ample room to improve the performance of the memory hierarchy (3.29x average speedup when using perfect caches).

Multithreading is a very effective technique to hide the memory latency and desktop GPUs rely on aggressive multithreading to keep the functional units busy. However, we find it to be power-hungry as shown in figure 3b. Although a GPU with 16 warps/core achieves an average speedup of 3.23x (close to the 3.29x offered by perfect caches), it also increases energy consumption by 25% on average due to the huge size of the main register file that is needed to keep the state of all the simultaneous threads.

We have also evaluated several **hardware prefetchers** on a mobile GPU: the Global History Buffer (GHB) prefetcher [NS04] and the Many-Thread Aware prefetcher [LLKV10] (specifically designed for GPUs). The results obtained by these state-of-the-art prefetchers are shown in figure 3. When using just prefetching without multithreading (1 warp/core), the hardware prefetchers provide an average speedup of up to 1.9x, which is far from the 3.29x speedup achieved by the perfect caches so there is still ample room for improvement.

Since the traditional techniques to hide memory latency do not work well or come at a high energy cost, we propose the migration of GPU designs towards the DAE concept as illustrated in figure 2b. The proposed scheme issues prefetch requests for each pixel so all the necessary information is fetched into the caches while the pixels are waiting in the tile queue. The available per-pixel information is employed to compute the addresses of the cache lines that are needed to process the pixel, these addresses are inserted into the prefetch queue and subsequently the corresponding prefetch requests are sent to the target texture/pixel caches. By the time a pixel is dispatched to the pixel processor, the data required to process it are usually available in the pixel and texture caches so that almost all processor cache accesses hit. Notice that the prefetch requests are based on computed addresses rather than predicted



(a) (b)

Figure 3: (a) Speedups for different prefetchers and different number of warps. (b) Normalized energy vs speedup, the DAE system provides the most energy-efficient solution.

addresses, so higher accuracy than hardware prefetchers is achieved. Since pixel programs are typically free of *loss of decoupling events* [APX12], most of the required information can be prefetched into the caches. The DAE system can be further improved by exploiting **inter-core data sharing**. We have observed that, in our set of workloads, 66.3% of the cache misses can be satisfied by the pixel or the texture cache of another pixel processor. We propose to employ the prefetch queue to capture part of this data reusage. More specifically, we include an additional field into the prefetch queue, called *Source*, that will hold the predicted alternative location of the block. Each time a new prefetch request is inserted into the prefetch queue, the addresses of all the queued requests are associatively compared with the new address. If there is a match, the identifier of the matching cache is recorded in the *Source* field of the new entry. Otherwise, the identifier of the L2 cache is recorded. When the prefetch request is dispatched to its target cache, the *Source* field is used by the cache controller to redirect the request in case of a cache miss. Allowing for remote L1 requests saves bandwidth to the L2 cache and provides energy savings since accessing a small L1 cache requires less energy than accessing the bigger L2 cache. Figure 3 shows the results obtained by this DAE design. Compared to the hardware prefetchers (no multithreading, 1 warp/core), the DAE system is 33% faster and provides 9% energy savings. Compared to aggressive multithreading, a DAE system with 2 warps/core achieves nearly the same performance (93%) of a bigger GPU with 16 warps/core, but it provides 34% energy savings. Therefore, the DAE system provides the most energy efficient solution to hide the memory latency, since it achieves a performance close to a system with perfect caches at the lowest energy budget.

References

[APX12] J.-M. Arnau, J.-M. Parcerisa, and P. Xekalakis. Boosting mobile gpu performance with a decoupled access/execute fragment processor. ISCA '12, 2012.

[LLKV10] J. Lee, N. B. Lakshminarayana, H. Kim, and R. Vuduc. Many-thread aware prefetching mechanisms for gpgpu applications. MICRO '43, 2010.

[NS04] K. J. Nesbit and J. E. Smith. Data cache prefetching using a global history buffer. HPCA '04, 2004.

[Smi82] J. E. Smith. Decoupled access/execute computer architectures. ISCA '82, 1982.