Smartphones

Editor: Nayeem Islam ■ Qualcomm ■ nayeem.islam@gmail.com

The Rise of Mobile Visual Computing Systems

Kayvon Fatahalian, *Carnegie Mellon University*

In the past decade, the growing demand for compute-intensive *visual computing applications* (applications that synthesize, manipulate, and interpret images and videos) has spurred the development of mobile graphics and image-processing architectures that achieve exceptional levels of energy efficiency. Today, hundreds of millions of smartphones operate within a power budget of only a few watts while featuring GPUs offering close to a quarter teraflop of compute capability. These energy-optimized mobile GPUs, whose performance now exceeds that of older-generation game consoles (such as the XBox 360 or Playstation 3), render complex 3D scenes at high frame rates for high-pixel-density displays.

The programmable execution units in mobile GPUs, together with image signal processors (ISPs) and multicore mobile CPUs, also form a platform for rapid innovation in consumer digital photography. High-frame-rate video, high-dynamic-range imaging, and interactive photo-stitching on 1080p and higher video streams are now common features on many devices. Perhaps even more important, in the coming years, these systems will increasingly be tasked with not only generating and manipulating images but also interpreting them. Applications relying on high-performance, energy-efficient computer vision and image understanding capabilities are becoming increasingly prevalent on consumer smartphones and in a broad array of "always on" embedded platforms used in automotive, robotics, security, and smart-city sensing scenarios.

The world's demand for increasingly capable mobile visual computing applications running on a range of low-energy computing platforms shows no sign of slowing down for the foreseeable future. Therefore, it's helpful for engineers to understand the technologies and solutions used to deliver efficient platforms today, and to look ahead at the challenges these emerging workloads will present to system architects in the years to come.

The world's demand for increasingly capable mobile visual computing applications running on a range of low-energy computing platforms shows no sign of slowing down.

WHAT IS MOBILE VISUAL COMPUTING?

Not long ago, mobile graphics engineers struggled to meet the challenges of delivering responsive user interfaces for multitouch devices, supporting simple OpenGL-based 3D graphics, and playing back a feature-length HD movie on a single battery charge. Modern mobile computing applications present a significantly more demanding set of requirements.

High-Resolution 3D Graphics

Games developed for smartphones and tablets now feature elaborate 3D scenes with complex geometry, materials, and lighting. These applications require nearly the same set of GPU functionality as AAA game titles developed for desktop-graphics APIs (such as OpenGL 4 or Direct3D 11). As a result, the difference between mobile and desktop 3D graphics is now largely one of performance, not critical feature set, as evidenced by popular game engine frameworks (such as Unity and Unreal Engine) targeting both desktop and mobile platforms with the same tools.

Although 3D graphics workloads involve many operations that are best carried out by fixed-function processing (texture mapping, tessellation, rasterization, and surface occlusion), the most expensive operations are application-programmable "shading" operations that evaluate the color of each screen pixel. These operations, which simulate the physics of light reflecting off the surface visible in each pixel, are highly data-parallel and must be carried out using floating-point arithmetic (although half-precision floating-point operations are often suitable).

Computational Photography

Capturing high-quality photographs and videos is an essential feature of any modern smartphone. Many camera applications rely on *computational photography* methods, which digitally

Published by the IEEE CS ■ 1536-1268/16/$33.00 © 2016 IEEE

manipulate image sensor output to obtain higher quality photographs or to synthesize new photographs that can't be acquired directly due to the physical limitations of a smartphone camera's optical system. For example, modern smartphones perform sophisticated image enhancements (including denoising, white-balance, and contrast enhancement) on raw sensor output to produce images that rival the quality of low-end digital single-lens reflex (DSLR) cameras. Applications also manipulate and combine multiple shots to simulate DSLR effects (such as lens defocus blur), synthesize high-dynamic range images, remove the effects of camera shake, automatically stitch images together to create panoramas, and even help photographers capture the right moment by selecting the best shot in a burst-mode sequence.

In all cases, these operations must complete in near real time to ensure a high-quality user experience. For efficiency, it's often sufficient to perform many computational photography operations using half-precision floating-point or low-precision fixed-point operations.

Real-Time Computer Vision and Image Understanding

Rapid recent advances in computer vision algorithms are making it possible for computers to intelligently interpret the contents of images and videos. Soon, applications will be able to track and identify people, objects, or potential obstacles in the camera's environment, detect (even predict) human activities, and reconstruct the 3D geometry of a dynamic scene. These image-understanding operations dwarf the sophistication of today's simple image analysis tasks (such as reading QR codes or identifying books from photographs of their covers), and they will be key functionalities expected in mobile devices in the coming years.

Computer vision workloads involve a range of operations, from low-level image-processing tasks, such as feature extraction and motion estimation, to compute-intensive operations, such as the evaluation of deep neural networks and classifiers. These operations must proceed in real time on video streams and over extended recording durations.

HETEROGENEOUS PARALLEL PROCESSING

A modern system on chip (SoC) executes visual computing tasks using a heterogeneous collection of processing resources. Many of these processing resources are specialized to achieve high performance per watt for a specific class of workloads, giving application developers a choice of what resource to use for a particular task. A sketch of a typical system architecture is given in Figure 1.

This system features a multicore mobile CPU supporting throughput-oriented short-vector instructions (such as ARM's 128-bit NEON instructions). The system also has a GPU featuring both 3D-rendering specific logic blocks and a collection of application-programmable cores. While the CPU's cores are well suited for instruction streams with complex control flow, GPU cores provide high floating-point instruction throughput for data-parallel computations. Shading computations for OpenGL-based 3D graphics are the principle workload executed on the GPU's cores, but these processors can also serve as a platform for high-performance pixel manipulation in computational photography and computer vision applications.

The compute capability of the programmable CPU and GPU cores is augmented by an ISP, which is designed to efficiently execute the pipeline of image-processing operations that convert raw image sensor output into common YUV or RGB image formats (demosaicing, dead-pixel correction, and so on). Although ISP blocks have traditionally supported only limited programmability (if at all), increasing sophistication and diversity in image-processing algorithms is now motivating more versatile, programmable ISP designs. For example, Qualcomm's Snapdragon

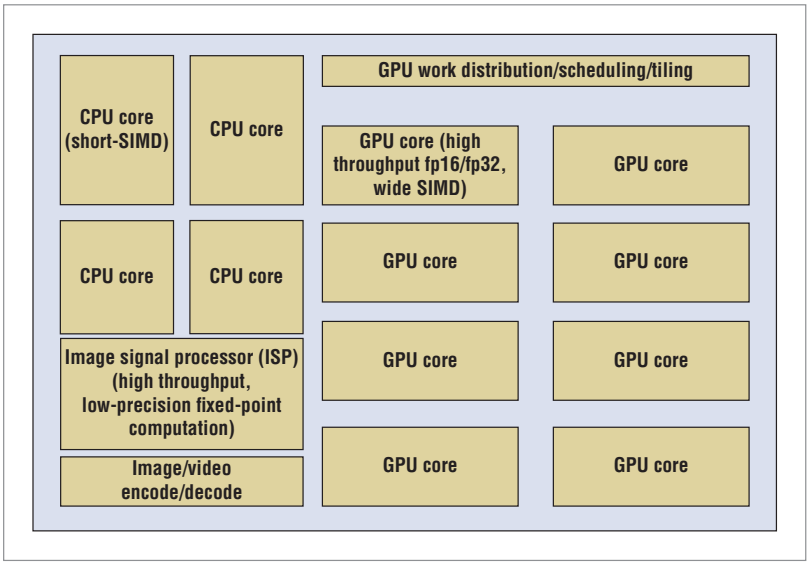

Figure 1. System on chip (SoC) processing resources used by visual computing apps. A modern SoC features a heterogeneous collection of processing resources, which present application and runtime system developers with the challenge of determining which resources most efficiently execute different visual computing workloads. GPU core designs achieve high throughput by aggressively employing single-instruction multiple-data (SIMD) processing.

820 SoC provides a programmable ISP (Hexagon 680) capable of executing integer vector instructions specific to high-efficiency image processing. In addition to CPU, GPU, and ISP blocks, modern SoCs also retain specialized hardware engines for image and video compression and decompression tasks (JPG, H.264, and so on).

MOBILE GPU PROGRAMMABLE RESOURCES

A significant fraction of the compute capability in a modern SoC lies within the programmable cores of the GPU. In many high-end smartphones, these highly versatile but throughput-optimized processing units provide over a quarter teraflop of single-precision floating-point performance. Although low-level architectural details vary across vendors (and are maintained as trade secrets by some), modern mobile GPU core designs all share several key characteristics.

Wide SIMD Execution

GPU core designs achieve high throughput by aggressively employing single-instruction multiple-data (SIMD) processing to pack cores densely with arithmetic logic units (ALUs). Modern designs range from 4-wide SIMD execution (ARM Mali) to 16-wide (Imagination PowerVR) and 32-wide (NVIDIA Tegra) configurations. Designs also mix wide SIMD execution with limited superscalar (or very long instruction word) execution to achieve additional parallelism. For example, an Imagination 7-Series GPU core decodes up to two single-precision (fp32) instructions per clock and executes each on a 16-wide group of SIMD ALUs. Similarly, each of the two cores in the NVIDIA Tegra X1 GPU execute up to four 32-wide fp32 instructions per clock.

Accelerated Half-Precision Floating Point

In contrast to desktop GPU designs, mobile GPU cores place heavy emphasis on support for energy-efficient, half-precision floating-point arithmetic (fp16). Many designs can perform fp16 operations at twice the throughput of single-precision operations. As a result, using half-precision instructions has both performance and energy-efficiency benefits. Mobile application developers are heavily encouraged to use lower-precision fp16 instructions when they're sufficient for an application's needs.

Multithreaded Execution

To compute the color of output pixels, shading computations in the OpenGL graphics pipeline access data stored in large DRAM-resident buffers called *textures*. Mobile GPUs employ traditional caching mechanisms and on-chip static RAMs to reduce memory traffic as much as possible, but like their desktop GPU counterparts, they also use a large degree of hardware multithreading to avoid ALU stalls by hiding the latency of off-chip data access.

Table 1 summarizes the architectural features and peak compute capability of several recent mobile GPU designs and compares these designs with that of a high-end discrete GPU (NVIDIA GTX 980) as well as the GPU in the XBox 360 gaming console. (The Imagination and ARM mobile GPUs are representative of parts in the iPhone 6s and Galaxy 6S smartphones, respectively.) Peak throughput for the mobile GPUs is computed using a 650 MHz clock, although actual clock rates in shipping devices can dynamically vary well below or above this estimate. The compute capabilities of the mobile GPUs exceed that of the XBox 360, and in several designs, fp16 performance exceeds a half a teraflop. The NVIDIA Tegra X1 GPU is targeted at higher end mobile devices, such as gaming tablets or vehicular systems. When running at 1 GHz (a plausible clock rate in a more energy-plentiful computing environment, such as a vehicle), the GPU can deliver over 1 Tflop of fp16 performance.

TABLE 1
Mobile GPUs employ multicore, single-instruction multiple-data (SIMD) designs to deliver high-peak floating-point throughput. Many designs attain even higher peak throughput for half-precision floating-point (fp16) operations.

	SIMD width	Arithmetic logic units/core (fp32 multiply-add)	Cores/ GPU	Gflops (fp32/fp16)
Imagination GT7600	16	32	6	250/500*
PowerVR	4	20 + 2 dot†	8	176/166*
ARM Mali T760MP8	32	128	2	332/664*
NVIDIA Tegra X1 NVIDIA GeForce GTX 980 (discrete)	32	128	16	4612/ — (1.1 GHz)
Xbox 360 ATI Xenos (console)	16	80	3	240/ — (500 MHz)

* Gflops for mobile GPUs computed using 650 MHz core clock
† Mali ALUs: not multiply-add ALUs—2 pipelines/core × (5 mul + 5 add + fp32 vec4 dot product) per pipeline

BANDWIDTH-EFFICIENT 3D GRAPHICS

Implementations of the OpenGL 3D graphics pipeline on mobile GPUs serve as telling examples of the extent to which visual computing tasks must be reoptimized for energy efficiency. These implementations reflect a key design principle of energy-optimized system design: reduce or eliminate off-chip data access whenever possible. The pursuit of bandwidth-efficient rendering in the mobile setting has led to notable differences in graphics pipeline implementation between mobile and desktop GPUs.

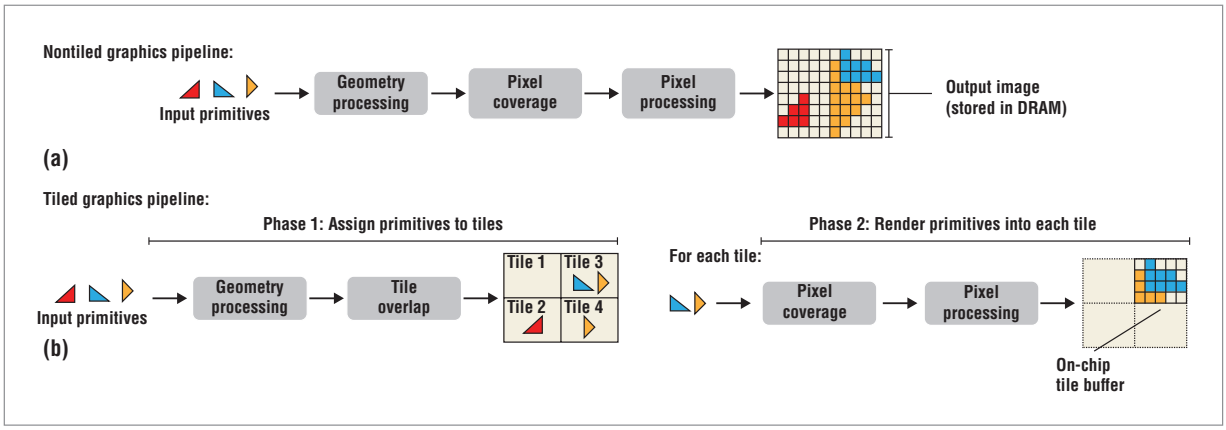

Figure 2. An (a) untiled and (b) tiled graphics pipeline. Tiled rendering systems achieve bandwidth-efficient (and therefore energy-efficient) operation by reorganizing the rendering computation into a two-phase process. Primitives are first sorted into tiles by the screen region they overlap. Then, the geometry in each tile is rendered on a per-tile basis. By restructuring the rendering computation to proceed in tile order, rather than primitive order, tiled rendering systems improve the locality of access to pixel data, significantly reducing the bandwidth required to render an image.

Tiled Rendering to Reduce Memory Bandwidth

The most notable difference between current desktop and mobile GPU implementations is the use of bandwidth-preserving *tiled rendering* techniques in mobile GPU designs. (Imagination, ARM, and Qualcomm GPUs all employ forms of tiled rendering.) As illustrated in Figure 2a, in a nontiled rendering pipeline implementation, the GPU immediately processes geometry provided by the application. The GPU will load a geometric primitive (such as a triangle) from memory, execute the entire OpenGL rendering pipeline on this primitive, and then potentially update pixels in the output image. While access to input geometry data is bandwidth efficient (primitives are only loaded from memory once), updating output image pixels can incur a high bandwidth cost, because a high-resolution output image is too large to remain resident in local, on-chip storage.

In contrast, tiled rendering systems (Figure 2b) reorganize the rendering pipeline into two phases that reduce memory traffic by increasing the temporal locality of accesses to output image pixels. The first phase of rendering partitions the screen into disjoint regions called *tiles* and sorts scene primitives according to the tiles they overlap. In the second phase, each tile is independently rendered using only the scene primitives determined (in phase 1) to be visible in that region of the screen.

The advantage of tiled rendering is that tiles can be sized to ensure all pixel data for a tile remains resident in on-chip storage (for example, small tile sizes, such as 16 × 16 or 32 × 32 pixels, are common). As a result, when rendering a tile, all updates to pixels in the tile (for the entire scene's worth of geometry) can be serviced without incurring off-chip memory traffic. Only when a tile has been fully rendered must final pixel values be transferred to memory for subsequent display.

Tiled rendering incurs the overhead of two phases of computation and must store intermediate results (per-tile primitive lists) to memory between phases. However, because current mobile graphics workloads feature only a modest number of scene primitives and are rendered to extremely high pixel count displays, improving the locality of access to pixel data often yields significant energy-efficiency benefits. Evolution of mobile 3D graphics workloads toward increasingly high geometric complexity scenes will require mobile GPU architects to reevaluate the efficiency of their current tiling methods. For example, Qualcomm's Adreno mobile GPUs already features the ability to dynamically select between tiled and non-tiled rendering methods based on characteristics of the 3D rendering workload.

Hardware-Accelerated Data Compression

Mobile GPUs also reduce the bandwidth requirements of 3D graphics using extensive hardware support for data compression. While all GPUs (both desktop and mobile) contain hardware for compressing texture and output image pixel data prior to transfer to or from main memory, advanced techniques that achieve higher compression ratios, such as ARM's Adaptive Scalable Texture Compression (ASTC), have been aggressively developed and adopted by mobile GPU vendors. Data-compression hardware is not only present in the GPU but also in display hardware. It's common for rendered images to be stored and transferred to the display in a compressed form to save bandwidth, and then to be decompressed directly by the display hardware.

Previous Page | Contents | Zoom in | Zoom out | Front Cover | Search Issue | Next Page

SMARTPHONES

Aggressive Discarding of Rendering Work

Mobile GPUs further conserve memory bandwidth and reduce overall energy consumption by seeking to perform as little work as possible to render an image. Mobile GPUs aggressively discard primitives (or pieces of primitives) from the rendering pipeline when it's determined that they won't contribute to the final image. For example, it's wasteful for a GPU to compute per-pixel lighting and shading computations (and access off-chip texture data required by these computations) for an object that is later determined to be occluded by another object in the scene.

One popular technique for avoiding nearly all unnecessary per-pixel shading computations is to defer shading computations until after the visibility of all scene geometry in a tile (or screen region) is known. In desktop graphics applications, these deferred shading optimizations are typically implemented by applications on top of the OpenGL pipeline implementation. On mobile platforms, they're often directly accelerated by GPU hardware, because it's efficient to perform deferred shading as part of tiled OpenGL pipeline implementations. Mobile GPUs also perform additional work-elimination optimizations such as skipping updates to pixels that don't change from frame to frame (for example, ARM's memory transaction elimination).

A NEED FOR NEW PROGRAMMING ABSTRACTIONS

Mobile GPU architects have succeeded at aggressively optimizing the implementation of mobile 3D rendering pipelines, because abstractions for describing 3D graphics computations are well established, stable, and standardized across the industry. Today, a major question facing application developers and system implementers is how to establish similar, unifying high-level abstractions for a broader set of visual computing applications, such as image processing, computational photography, and computer vision.

One recent success is the Halide language, originally developed at MIT, which enables image-processing algorithms to be described concisely using high-level functional abstractions. Halide greatly simplifies the process of mapping (or "scheduling") an image-processing algorithm onto the multicore and SIMD execution resources of modern processors, such as CPUs and mobile GPUs. Halide is now in use at Google to author high-performance implementations of computational photography applications, such as the HDR+ application in hundreds of millions of Android smartphones.

The design and success of shading languages for 3D graphics has inspired interest in creating new data-parallel programming languages that simplify the process of running code on a mobile GPU's programmable cores. Kronos's OpenCL, Apple's Metal, and Android's Renderscript all seek to provide data-parallel—but not 3D-rendering specific—applications access to high-throughput GPU processing.

At the other end of the spectrum, SoC vendors continue to expose the compute capabilities of their platforms through domain-specific libraries. For example, Qualcomm's FastCV or NVIDIA's VisionWorks provide applications with heavily optimized image processing and computer vision kernels. Deep neural-network implementations (key kernels used by many modern computer vision algorithms) are also provided to applications as black-box libraries, such as NVIDIA's CUDA Deep Neural Network (cuDNN) library.

As both the compute resources on an SoC and the complexity and workload diversity of visual computing applications continues to grow, it will become increasingly challenging for mobile application developers to harness the power and efficiency of these systems. For example, future programmable ISPs will likely join CPUs and GPUs as a third type of programmable compute unit available on most platforms, further complicating the challenge facing application developers (or parallel runtime developers) of selecting the best execution platform for a particular visual computing task. Future applications will surely involve the integration of several classes of workloads, requiring tight synchronization and data transfer between the different system components. New domain-specific programming frameworks, tailored specifically to the needs of specific areas of visual computing (as opposed to general, heterogeneous parallel computing languages), are the most likely approach to successfully meeting the goals of programmer productivity and high-efficiency execution in these contexts.

HARDWARE TRENDS

Until recently, mobile hardware architects have focused on increasing system efficiency to achieve higher performance and deliver functionality present in traditional computing platforms (for example, 64-bit addressing, virtualization support, and feature parity in 3D graphics). In the near future, interest will likely shift toward tighter integration of the powerful compute engines on the SoC—a unique challenge that hasn't already been addressed in the desktop setting. For example, ongoing efforts to facilitate CPU and GPU communication via a single address space might be extended to address coherence issues and grow to include additional IP blocks such as a programmable ISP. Efficient hardware support for synchronizing these heterogeneous units and enabling fine-grained communication through on-chip buffers (rather than off-chip memory) will also be of high interest.

Another vector of future innovation will explore new types of specialized compute engines for the SoC. The programmable ISP for image processing and computational photography is one clear direction for additional specialization. Recently, Imagination announced plans to include hardware units that accelerate ray-tracing operations (in addition to traditional OpenGL graphics operations) in future

PowerVR GPUs. Also, growing interest in always-on speech recognition and computer vision is motivating many research efforts to design dedicated hardware that accelerates deep neural network evaluation.

In the coming decade, emerging media platforms, such as virtual reality and augmented reality systems, will place immense demands on mobile 3D graphics systems to synthesize high-resolution images. Computational photography applications will continue to provide more intelligent and rich ways for consumers to capture and share life events. Cameras are rapidly becoming commonplace in vehicles to help drivers (both human and autonomous) interpret and safely navigate the world. Organizations, both public and private, are using extended duration video capture to provide security (via security cameras and body-mounted police cameras) and to measure and understand the behavior of the world (urban governments seek to use image analysis to count traffic, measure pollution, find available parking spots, and more efficiently manage cities).

In all of these cases, visual computing tasks serve as a driving force for making high-throughput, low-energy computing ubiquitous in the real world. This domain will continue to operationalize as much computing capability as architects can provide, and will push the limits of high-efficiency mobile computing for years to come.

Kayvon Fatahalian is an assistant professor of computer science at Carnegie Mellon University. Contact him at kayvonf@cs.carnegie.edu.

Selected CS articles and columns are also available for free at http://ComputingNow.computer.org.