WHITE PAPER

Take GPU Processing Power Beyond Graphics with Mali GPU Computing

Roberto Mijat
Visual Computing Marketing Manager
August 2012

Introduction

Modern processor and SoC architectures endorse parallelism as a pathway to get more performance more efficiently. GPUs deliver superior computational power for massive data-parallel workloads. Modern GPUs are becoming increasingly programmable and can be used for general purpose processing. Frameworks such as OpenCL™ and Android™ Renderscript enable this. In order to achieve uncompromised features support and performance you need a processor specifically designed for general purpose computation. After an introduction to the technology and how it is enabled, this presentation will explore design considerations of the ARM Mali-T600 series of GPUs that make them the perfect fit for GPU Computing.

The rise of parallel computation

Parallelism is at the core of modern processor architecture design: it enables increased processing performance and efficiency. Superscalar CPUs implement instruction level parallelism (ILP). Single Instruction Multiple Data (SIMD) architectures enable faster computation of vector data. Simultaneous multithreading (SMT) is used to mitigate memory latency overheads. Multi-core SMP can provide significant performance uplift and energy savings by executing multiple threads/programs in parallel. SoC designers combine diverse accelerators together on the same die sharing a unified bus matrix. All these technologies enable increased performance and more efficient computation, by doing things in parallel. They are all well established techniques in modern computing.

Portability and complexity

Today's computing platforms are complex heterogeneous systems (HMP). For example the Samsung® Exynos Quad SoC, which is at the heart of the award winning Samsung Galaxy S III smartphone, includes: an ARM Cortex™-A9 quad-core CPU implementing VPF and 128-bit NEON™ Advanced SIMD, a quad-core Mali-400 MP 2D/3D graphics processor, a JPEG hardware codec, a multi-format video hardware codec and a cryptography engine.

Programming approaches for each processor (CPU, GPU, ISP, DSP etc) are all different. Optimizing code for a selected accelerator requires specialized expertise. Code written for one accelerator is typically non portable to other architectures. This leads to a suboptimal utilization of the platform's processing potential. Writing parallel code that scales is also very difficult, and has proven illusive for most applications in the mobile industry today.

GPUs: Moving beyond graphics

Early GPUs were specifically designed to implement graphics programming languages such as OpenGL®. Whilst this meant that OpenGL applications/operations would typically achieve good performance, it also meant that programmers were limited to the fixed functionality expressed by the API. To address this limitation, GPU implementers made the pixel processor in the GPU programmable (via small programs called shaders). Over time, to handle increasing shader complexity, the GPU processing elements were redesigned to support more generalized mathematical, logic and flow control operations.

Enabling GPU Computing: Introduction to OpenCL

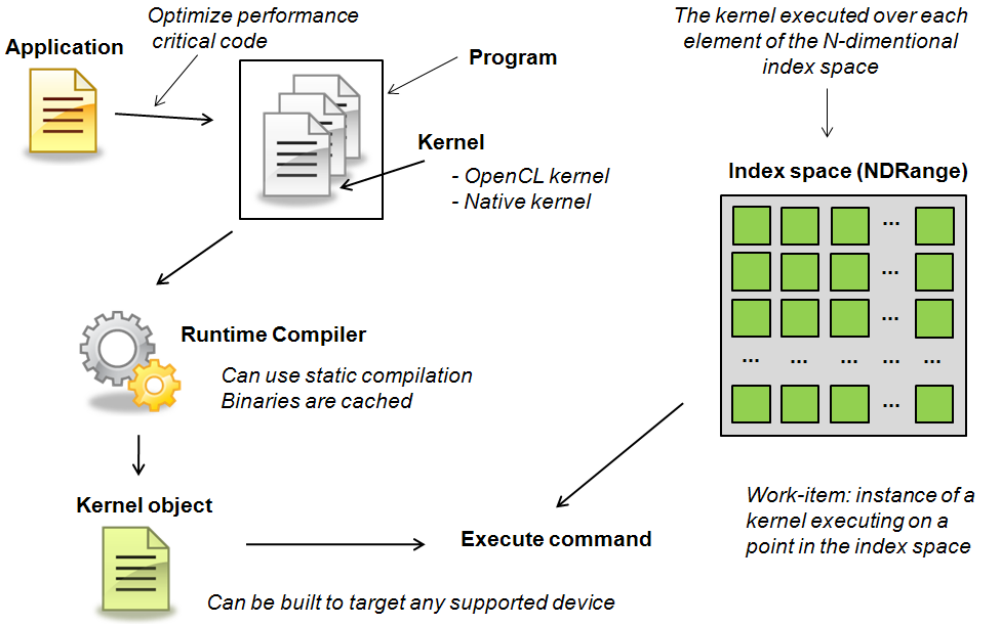
OpenCL (Open Compute Language) provides a solution that enables easier, better, portable programming of heterogeneous parallel processing systems and unleashes the computational power of GPUs needed by emerging workloads. OpenCL creates a foundation layer for a parallel computing ecosystem and takes graphics processing power beyond graphics. It is defined by the Khronos Group, and it is a royalty-free open standard, interoperable with existing APIs.

The OpenCL framework includes:

- A framework (compiler, runtime, libraries) to enable general purpose parallel computing
- OpenCL C, a computing language portable across heterogeneous processing platforms (a superset of a subset of C99, removing pointers and recursion but adding vector data types and other parallel computing features)

- An API to define and control (interrogate and configure) the platform and coordinate parallel computation across processors.

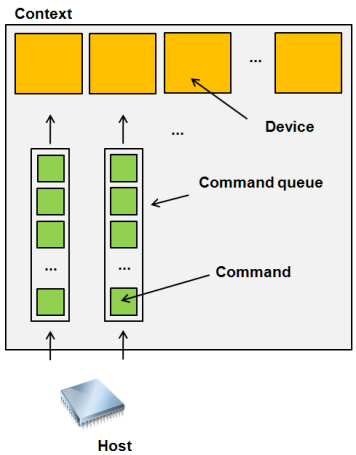
The developer will identify performance critical areas in its application and rewrite them using the OpenCL C language and API. An OpenCL C function is known as *kernel*. Kernels and supporting code are consolidated into *programs*, equivalent in principle to DLLs.



OpenCL implements a control-slave architecture, where the *host* processor (on which the application runs) offloads work to a computing resource. When a kernel is submitted for execution by the host, an *index space* is defined. The index space represents the set of data that the kernel will be applied to. It can have 1, 2 or 3 dimension (hence the name of *NDRange*, or N-dimensional range). The instance of a kernel executing on an individual entry in the index space takes the name of *work-item*. Work items can be grouped into *work-groups*, which will execute on a single *compute unit*.

Kernels can be compiled ahead of time and stored in the application as binaries, or JIT-compiled on the device, in which case the kernel code will be embedded in the application as source (or a suitable intermediate representation). The kernel can be compiled to execute on any of the supported devices in the platform.

The application developer defines a *context* of execution, which is the environment the OpenCL C kernels execute in. The context includes the list of target devices, associated command queues, the memory accessible by the devices and its properties. Using the API, the application can queue commands such as: execution of kernel objects, moving of memory between host and processing plane, synchronization to enforce ordered execution between commands, events to be triggered or waited upon, and execution barriers.



OpenCL enables general purpose computing to be carried out on the GPU. The ARM Mali-T600 series of GPUs has been specifically designed for general purpose GPU computing, and an OpenCL 1.1. Full Profile DDK is available from ARM.

More information of OpenCL can be found on the Khronos website.

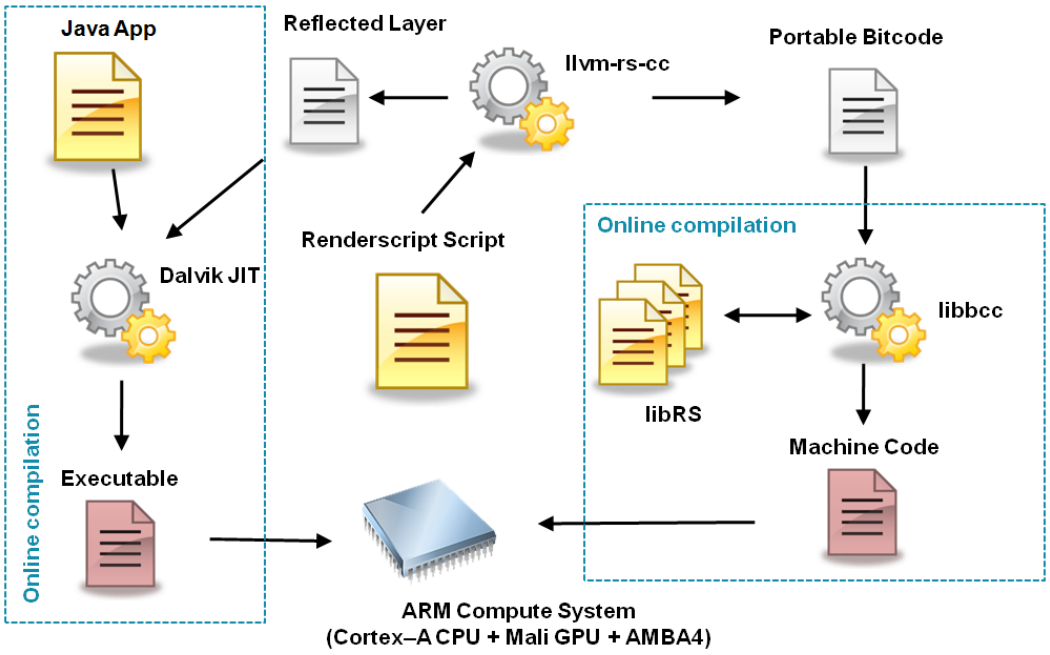
Android Renderscript

Renderscript is a high performance computation API for Android. It has been officially introduced in Honeycomb.
Renderscript complements existing Android APIs by adding:

- A compute API for parallel processing similar to CUDA/OpenCL
- A scripting language based on C99 supporting vector data types (called ScriptC)

Earlier versions of Renderscript included an experimental graphics engine component. This has been deprecated since Android 4.1 Jelly Bean.

Like OpenCL, Renderscript implements a cross-platform control-slave architecture with runtime compilation. The majority of the application will be written using the Dalvik APIs as usual, whilst performance critical code – or code more suitable for parallel execution – will be identified and rewritten using the ScriptC language.



A key design consideration of Renderscript is performance portability: the API is designed so that a script should show good performance across all devices instead of peak performance for one device at the expense of others (naturally, intensive data parallel algorithms will continue to be more suitable for acceleration by the GPU). The compilation infrastructure is based around LLVM. A first stage of compilation is performed offline: portable bitcode is generated as well as all the necessary glue code to enable visibility of the script's data and functions from the Java application (the *reflected layer*). The APK package will include the Java application and associated files, assets and so forth, plus the RenderScript

portable binary. When Dalvik JIT-compiles the application, the intermediate bitcode is also compiled for the target processor. The compiled bitcode will be cached to speed up future loading of the application, and re-compiled only if the scripts are updated. This split enables aggressive machine-independent optimization to be carried out offline, therefore making the online JIT compilation lighter-weight and more suitable for energy-limited battery-powered mobile devices.

Up until Android 4.1, Renderscript is only enabled to target the CPU (with VFP/NEON). In the near future, this will be extended to target other accelerator, such as the GPUs.

ARM Mali-T600 series of GPUs: Designed for GPU Computing

To achieve optimal general purpose computational throughput you need a purposely designed processor, such as the Mali-T600 series of GPUs from ARM. These are designed to integrate the graphics and compute functionalities together, optimizing interoperation between the two both at hardware and software driver levels.

ARM Mali-T600 GPUs are designed to work with the latest version (4) of the AMBA (Advanced Microcontroller Bus Architecture) which feature Cache Coherent Interconnect (CCI). Data shared between processors in the system, a natural occurrence in heterogeneous computing, no longer requires costly (cycles and joules) synchronization via external memory and explicit cache maintenance operations. All of this is now performed in hardware, and is enabled transparently inside the drivers provided by ARM. In addition to reduced memory traffic, CCI avoids superfluous sharing of data: only data genuinely requested by another master is transferred to it, to the granularity of a cache line. No need to flush a whole buffer or data structure anymore.

Computing frameworks like Renderscript and OpenCL introduce significant additional requirements for precision and support of mathematical functions. In addition to satisfy IEEE 754 precision requirements for single and double floating point, Mali-T600 GPUs implement the majority of these mathematical operations directly in hardware. In fact over 60% of floating point functions defined by the OpenCL specification is hardware accelerated (most trigonometric functions, power and exponent, square root and division) and all of them meet IEEE 754 precision requirements. Over 70% of integer operations are also implemented in hardware. Mali-T600 GPUs natively supports 64-bit integer data types, something not common in competing architectures. Barriers and atomics are also implemented in hardware. In essence, the vast majority of operations take place in a single cycle (or a few cycles max). This provides an immense step-up in performance for general purpose computation if compared to current generation of GPUs not purposely designed for it.

There is more. As well as task management and event dependencies being optimized in hardware, task dependency coordination is entirely designed into the hardware job manager unit. The software driver responsibility is reduced to handing over the workload to the GPU: all scheduling, prioritization and run-time synchronization take place transparently, behind the scenes.

Typically GPUs are designed to favor throughput over latency. Mali-T600 GPUs treat generic memory load/stores as first-class operations with proper latency tolerance.

Typically developers use a blend of APIs during development. The Mali software driver infrastructure is tightly integrated and optimized. All APIs of the Mali software stack architecture share the same high-level API objects, the same address space, the same queues, dependencies and events. This approach reduce code footprint and significantly increase performance. Data structures are shared between APIs and devices, to avoid unnecessary memory copies.

Use cases

In addition to the many scientific, academic, industrial and financial use cases, there is a wide variety of applications where general purpose GPU computing brings great benefits. Examples include:

- Computational Photography and Computer Vision: compensating the limitation of the hardware sensor, image stabilization, HDR compensation, face and smile recognition, image editing, filters, landmark & context recognition, superimposition of information
- Multimedia: post-processing, motion vectors, transcoding, super-scaling, 2D-3D conversion
- Stream Data Processing: deep packet inspection, antivirus, encryption, compression, data analytics
- UIs, Gaming and 3D Modelling: voice recognition, gesture recognition, physics, AI, photorealistic ray tracing, modelling
- Augmented Reality
- And many many more!

GPU computing can be used for any computationally intensive task, but will be most efficient where parallelism can be exploited (either parallelism within the task, or where multiple tasks can be executed simultaneously).

Conclusion

Modern processor and SoC architectures endorse parallelism as a pathway to get more performance more efficiently. GPUs deliver superior computational power for massive data-parallel workloads. Modern GPUs are becoming increasingly programmable and can be used for general purpose processing. OpenCL and Renderscript enable this technology providing easier, better programming of heterogeneous parallel compute systems and unleashing the computational power of GPUs needed by emerging workloads.

To achieve optimal general purpose computational throughput you need a purposely designed GPU, such as the Mali-T600 series of GPUs from ARM. The ARM Mali-T600 series of GPUs is designed to integrate the graphics and compute functionalities together, optimizing interoperation between the two and delivering market leading 3D graphics and general purpose parallel computation.

For more information: ***gpucompute-info@arm.com***.