MediaTek CorePilot 2.0™

Heterogeneous Computing Technology

Delivering extreme compute performance with maximum power efficiency

In July 2013, MediaTek delivered the industry's first mobile system on a chip with Heterogeneous Multi-Processing. The MT8135 chipset for Android tablets features CorePilot 1.0 technology that maximizes performance and power saving with interactive power management, adaptive thermal management and advanced scheduler algorithms. In 2015, MediaTek introduced CorePilot 2.0, the evolution of CorePilot 1.0, with more advanced heterogeneous computing technology to support not only symmetric CPU cores, asymmetric big/little CPU cores scheduling, but also cooperation with CPU and GPU. By adding support to GPU, CorePilot 2.0 can provide better user experience by providing higher performance with lower power consumption.

Table of Contents

Introduction

With ever-increasing array of applications in the mobile device environment, better user experience, balance between powerful computing capability, and efficient power/thermal management has become a significant challenge for modern mobile device SoCs. The challenge is to meet user's performance need without device overheating and rapid battery drain. Homogeneous computing resources (e.g. CPU) are technically not always up to this task. The key solution to this technical issue is to take advantage of the heterogeneous computing resources in modern mobile device SoC.

Heterogeneous computing resources could be multi-core CPUs probably with big/little topologies and other types of computing units such as a GPU (graphic computing unit). Based on heterogeneous architecture design, CPU big/little cores and GPU in mobile device excel at managing different types of workloads, as shown in the example below. With image blurring process, commonly used in image processing related applications (such as Portrait Play), offloading from CPU to GPU can reduce up to 50% power consumption under similar performance.

Table 1. Comparison of Heterogeneous Computing Resources in Mobile Device SoC

	CPU big-core	CPU little-core	GPU
Performance	high	Moderate	High
Power consumption	high	Low	Low
Ideal cases	task-parallel compute-intensive tasks	routine light-weight tasks	data-parallel compute-intensive tasks
Example	browser scrolling or game	email or audio playing	face beautification or multi-media processing

As shown in Figure 1, CorePilot™ 2.0, the evolution of CorePilot 1.0, now supports not only symmetric CPU cores, asymmetric big/little CPU cores scheduling, but also cooperation with CPU and GPU. By adding support to GPU, we believe CorePilot 2.0 can provide better user experience by providing higher performance with lower power consumption. In CorePilot 2.0, this can be fulfilled by utilizing these heterogeneous computing resources in two ways. First one is to dispatch tasks to the suitable heterogeneous computing resources. For example, routine light-weight tasks should be executed on the CPU little cores instead of the CPU big cores, since the little cores can suit this kind of performance need. Another example is to execute the latency-oriented tasks, such as browser scrolling, on the CPU big cores to reduce the response time. The second way is to balance the load between the heterogeneous computing resources. For example, when the CPU little cores are busy, some tasks can be migrated to the CPU big cores or GPU (if the task is programmed by OpenCL) to balance the whole system load.

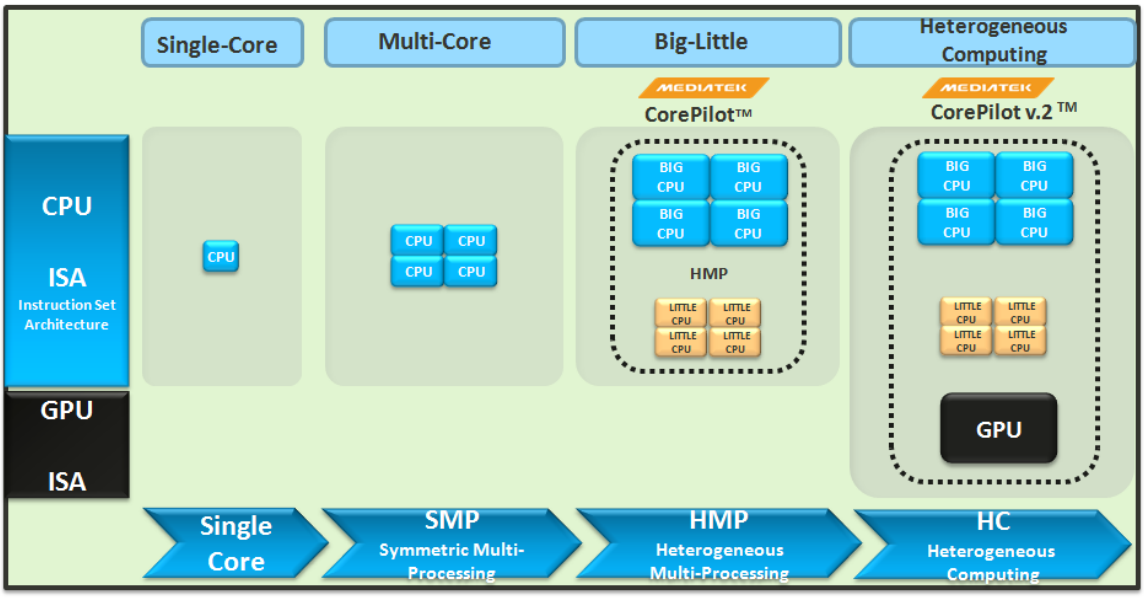

Figure 1. Evolution of MediaTek CorePilot

MediaTek, as a leading SoC-design company, not only provides series of high quality SoCs for mobile devices with heterogeneous computing resources, but also introduces the heterogeneous computation technology, CorePilot 2.0, to utilize the on-chip heterogeneous computing resources, thereby improving user experience. The details of CorePilot 2.0 are described in below. The big/little CPU scheduler, introduced on CorePilot 1.0 in 2013, is first described briefly, followed by the Device Fusion technology--CPU/GPU. Finally, we provide case studies to demonstrate the benefit of CorePilot 2.0.

MediaTek CorePilot — Heterogeneous Multi-Processing Technology

ARM brought heterogeneous computing to mobile SoC designers with big.LITTLE™ architecture. ARM big.LITTLE combines high-performance “big” CPUs with energy efficient “LITTLE” CPUs on the same SoC to reduce energy consumption (and thereby preserve battery power), while still delivering peak performance.

Since both CPUs are architecturally compatible, workloads can be allocated to each CPU, on demand, to suit performance needs. High intensity tasks such as games are allocated to the big CPUs, for example, while less demanding tasks such as email and audio playback are allocated to the “LITTLE” CPUs. The load balancing happens quickly enough to be completely transparent to the user and can reduce energy consumption by up to 70% for common tasks.

Table 2. Comparison of Different Heterogeneous Computing Models

	Cluster Migration	CPU Migration	Heterogeneous Multi-Processing (HMP)
Switching Granularity	low – cluster of core	medium – pair of big/little cores	high – each single core
Maximum Performance	medium – all “big”	medium – all “big”	high – all “big” + all “little”
Average Power Saving	Low	Medium	High

There are three different software models for implementing heterogeneous computing with the ARM big.LITTLE architecture: Cluster Migration, CPU Migration and Heterogeneous Multi-Processing. The comparison of these different heterogeneous computing models is shown in the above in

Table 2. When compared with the other two models, the Heterogeneous Multi-Processing (HMP), adopted by MediaTek Corepilot, shows outstanding performance and power saving.

Based on open-source HMP technology derived from Linaro (http://www.linaro.org), CorePilot maximizes the performance and power-saving potential of heterogeneous computing with interactive power management, adaptive thermal management and advanced scheduler algorithms. The overview of CorePilot is shown in Figure 2 below. As shown in Table 3 CorePilot’s Interactive Power Manager reduces the amount of power and heat generated by CPU via two main modules. The Dynamic Voltage and Frequency Scaling module automatically adjusts the frequency and voltage of CPUs on-the-fly, while the CPU Hot Plug module switches CPUs on and off on demand.

The thermal limits of a mobile device can be exceeded when its CPU or GPU runs at peak performance. This, in turn, can be detrimental to the user experience. Adaptive Thermal Management (ATM) addresses this by monitoring device temperatures and dynamically adjusting the power budget to keep them within a specified range, while minimizing the impact on performance. Compared with traditional dynamic power management designs with fixed temperature points for thermal throttling, ATM can achieve a 10% performance increase.

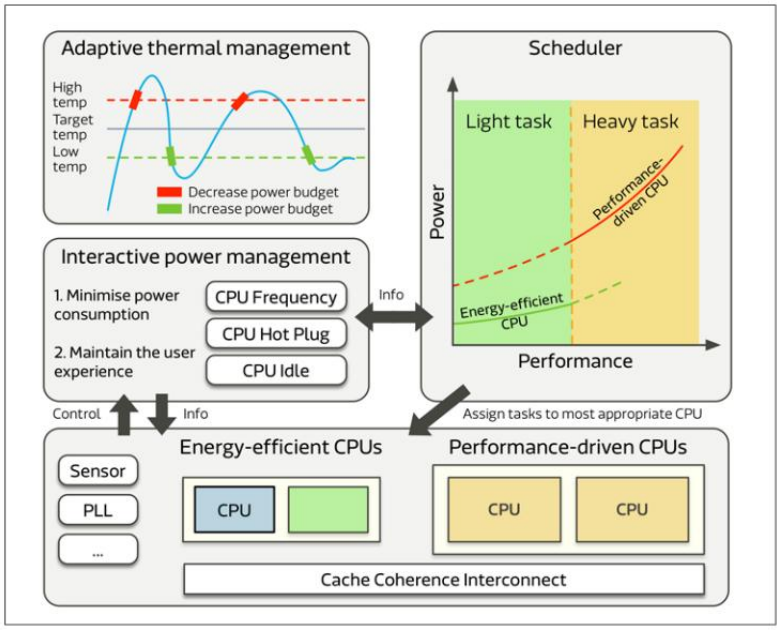

Figure 2. Overview of CorePilot

Table 3. Key Components of Interactive Power Management

Dynamic Voltage & Frequency Scaling	**Traditional symmetric multi-processors apply a unified Dynamic Voltage and Frequency Scaling (DVFS) policy to all CPUs. CorePilot's Interactive Power Management applies different DVFS policies to 'big' and 'LITTLE' cores to maximize power and thermal efficiency**
CPU Hot Plug	Interactive Power Management monitors CPU load and seamlessly switches cores on or off to save power or to increase performance. CPUs can also be switched off with non-CPU-bound tasks to reduce power consumption.

Performance is the usual goal for operating system scheduling, and technology has evolved over time accordingly. With Symmetric Multi-Processing (SMP), the Completely Fair Scheduler (CFS) is currently the most common scheduling algorithm and it distributes the workload equally among CPU cores. With Heterogeneous Multi-Processing, however, CFS can result in performance degradation, since tasks are not efficiently matched to CPU core capabilities. MediaTek CorePilot, on the other hand, delivers a true heterogeneous compute model by using a scheduling algorithm that assigns tasks to two different schedulers, according to their priority — the Heterogeneous Multi-Processing (HMP) scheduler and Real-Time (RT) scheduler. The HMP scheduler is responsible for assigning normal-priority tasks to the big/little CPU clusters and performs four main functions, shown in Table 3. The RT scheduler assigns high-priority real-time tasks that require a fast CPU response to the big/little cluster. The RT scheduler has priority over the HMP scheduler and MediaTek has further modified its design so

that the highest priority tasks are assigned to performance-driven CPUs. Lesser priority real-time tasks are then assigned to other available CPUs.

Table 4. Key Components of the MediaTek HMP Scheduler

Load tracking	**By tracking the status of each task, the HMP scheduler determines which task is heavy and which task is relatively light.**
CPU Capacity Estimation	The HMP Scheduler is aware of the available compute capacity of each processor in the big/little clusters, and so is able to make the most appropriate scheduling decisions.
Intelligent Load-Balancing	Load tracking and CPU capacity estimation are used in concert for rapid load balancing – assigning and reassigning tasks to performance-driven or energy-efficient CPUs, as required.
Task Packing	The HMP scheduler consolidates as many light-load tasks as possible and matches them to the most appropriate CPUs. CPUs without active tasks can then be switched off via CPU Hot Plug, or put into an idle state.

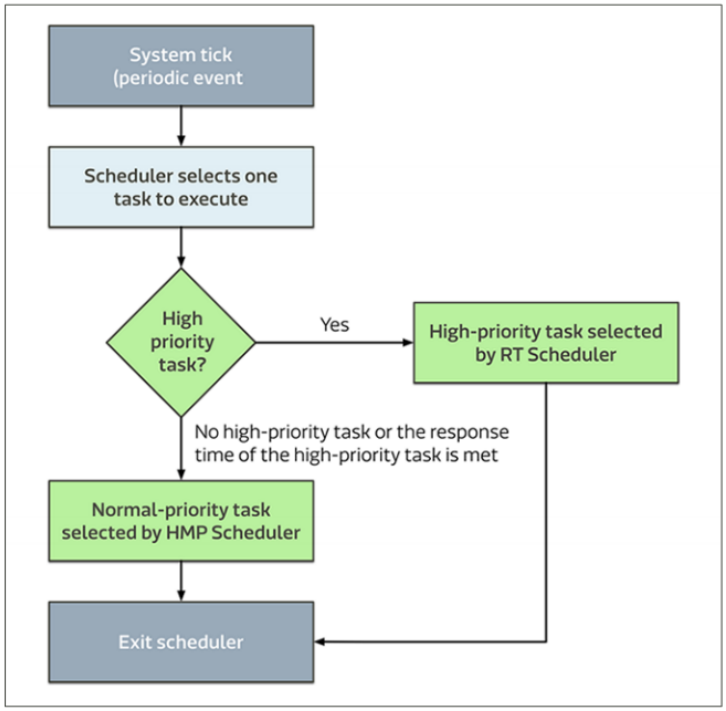

Figure 3. Process Flow for the HMP and RT Schedulers

MediaTek Device Fusion — Heterogeneous Computing with a Fused CPU+GPU Device

Recent findings suggest that using CPU and GPU together is a more efficient way for computing when compared with using CPU only, because different types of computing resources may

better suit different workloads. For example, the CPU is generally good at control-intensive workloads while GPU performs well at compute-intensive ones. OpenCL (Open Computing Language), a popular open standardized computing platform for heterogeneous computing, is designed to serve as the common high-level language for exploitation of computing resources. The OpenCL program can be executed on every device, including the mobile device which supports the OpenCL standard APIs.

CorePilot 2.0, including Device Fusion technology, can intelligently exploit the heterogeneous computing resources (i.e. CPU and GPU) to improve efficiency and performance. Via the Device Fusion, OpenCL programs can be well dispatched to on-chip CPU and GPU to improve efficiency and performance.

Currently, it is challenging for programmers to write programs that can effectively utilize heterogeneous devices (i.e. GPU and CPU) to achieve high performance/efficiency computing. Anticipating device computing capacity and program affinity (i.e. preference to device) for a specific program is not straightforward. With incorrect anticipation, performance could be degraded due to improper dispatching of jobs (of a program) to device(s).

To address these problems, CorePilot 2.0 introduces the intelligent technology—Device Fusion, which can efficiently execute OpenCL programs by fusing GPU and CPU computing capabilities. As shown in Figure 4 below, Device Fusion is able to flexibly dispatch each kernel (functional part) of an OpenCL program to the most suitable device. Moreover, for throughput-oriented programs, Device Fusion provides an infrastructure which can automatically maintain parallel execution on GPU and CPU. By using the Device Fusion, programmers can focus on algorithm development and obtain performance improvements without being affected by platform issues.

Device Fusion can dispatch OpenCL programs to the most suitable device or to both devices for parallel execution.

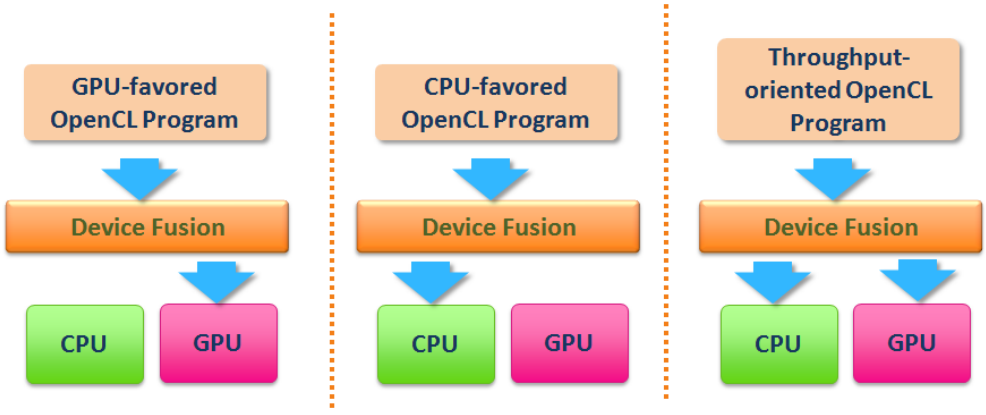

Figure 4. Dispatch Options in Device Fusion

Problem of executing OpenCL program in mobile SoC

Although OpenCL programs can be executed on all OpenCL-supported devices, the performance may not meet the programmer's expectation. The reasons are as follows.

The program affinity (i.e. preference to device) for a specific program is not easy to predict. The affinity of a program could be affected by algorithm design, implementation or (and) architecture of the target device. A program running on the un-preferred device results in lower performance when comparing to the preferred device. For example, two different implementations of the same object detection algorithm, *violaJones_NAIVE* and *violaJones_LWS32*, shows different program affinities, leading to large performance difference when executing on CPU and GPU, as shown in the Figure 3. From the figure, executing *violaJones_LWS32* (which prefers GPU) on the CPU results in severe performance degradation compared to the GPU. The *violaJones_NAIVE* shows the opposite results – CPU has better performance.

The figure below shows the execution time of two OpenCL kernel, violaJones_NAIVE and violaJones_LWS32, under CPU and GPU. Different implementations of the same algorithm lead to dramatic performance differences on CPU and GPU.

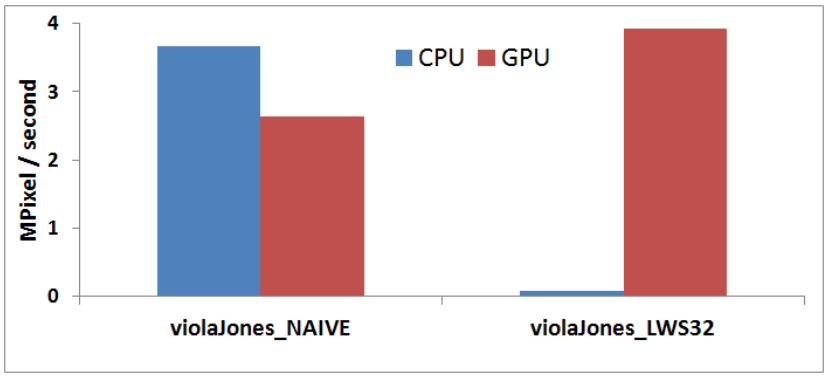

Figure 5. Performance Differences on CPU and GPU

The computing capability of the target devices, where the programs will be executed, may not be anticipated easily by programmers. Even if the programmers can focus on specific CPU and GPU as the target devices, the computing capability of CPU and GPU still cannot be anticipated in a straightforward way. This is because under the mobile device environment, the computing ability of CPU and GPU is often reduced due to power budget or the thermal constraint. Once the program is dispatched to a device with lower computing capability (below anticipation), the anticipated performance cannot be achieved. Such condition worsens for the throughput-oriented programs to be executed in parallel on both CPU and GPU.

For example, the performance of parallel execution of the program "Juliaset" is shown in Figure 6 below. The y-axis indicates the performance of parallel execution. The x-axis shows the ratio *n*, where *n* is the ratio of jobs dispatching to GPU and 1-*n* is the ratio of jobs dispatching to CPU. As

seen in the figure, the maximum performance occurs when the ratio is 0.56, indicating GPU and CPU handles 56% and 44% (i.e. 100% - 56%) of the jobs, respectively. With incorrect anticipation to CPU/GPU computing capability (e.g. given the ratio less than 0.5), performance of parallel execution could be lower than that of only dispatching to GPU.

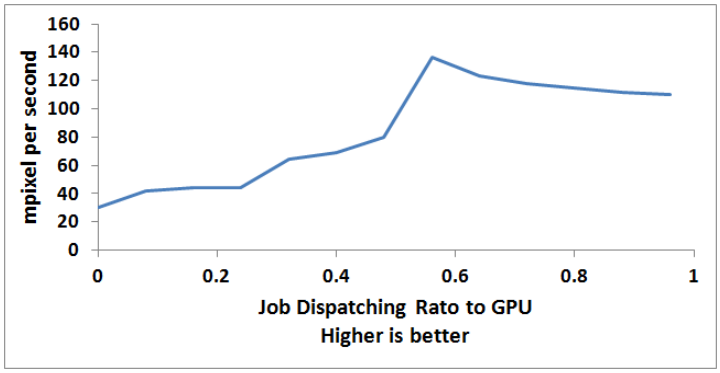

Figure 6. Performance of Parallel Executing Juliaset with Different Dispatch Ratios

The final reason the computing capacity may not be anticipated by programmers is the additional overhead for parallel execution. For programs aiming to be executed in parallel, enabling data sharing and synchronization incurs additional overhead, such as cache synchronization. Such overheads typically are hardware-dependent. Once the overheads outweigh the benefits of parallel execution, the performance could be below anticipation. Unfortunately, it is not easy for programmers to evaluate if parallel execution is beneficial by considering the overhead.

MediaTek Device Fusion

The Device Fusion is presented as a virtual OpenCL device. The virtual device, emulated on top of the CPU and GPU devices1, is compliant with the standard OpenCL API so that the existing OpenCL applications can easily take advantage of the Device Fusion without any modification. The overview of Device Fusion is described in Figure 7 below.

[1] Note that, the standalone OpenCL CPU and GPU devices are still available.

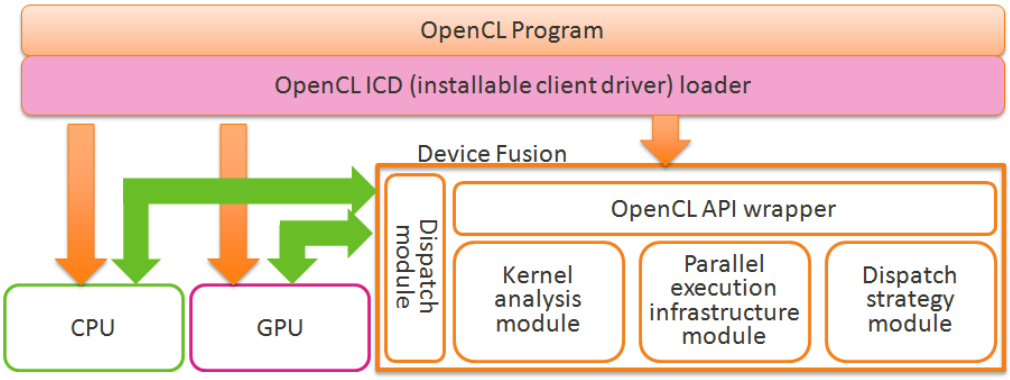

Figure 7. Overview of Device Fusion

The virtual device includes three modules: kernel analysis module, parallel execution infrastructure module, and dispatch strategy module. When an OpenCL program is assigned to the virtual device, each kernel of the program will be first analyzed to collect necessary information, such as if it is worthwhile for the kernel can to be executed in parallel. If the parallel execution is allowed, the parallel execution infrastructure module is invoked to prepare the infrastructure, such as data sharing and synchronization. Finally, the dispatch strategy module determines the number of jobs (i.e. work-items of the OpenCL program) of the kernel to CPU and GPU, respectively, according to the internal load-balancing algorithm. However, if the parallel execution is not available, the virtual device tries to dispatch the entire kernel to the most suitable device utilizing the advice of the dispatch strategy module.

Case Studies Using CorePilot 2.0

Super Resolution

The first case study to demonstrate the benefit of CorePilot 2.0 is SuperResolution. SuperResolution is an image-processing algorithm which can enhance image resolution and extract significant detail from the original image. SuperResolution requires significant computation to extract the detail. In SuperResolution, the process is divided into several stages: *find_neighbor*, *col_upsample*, *row_upsample*, *sub* and *blur2*. Each stage is implemented as an OpenCL kernel. In this case study, three different image resolutions: 1MPixel, 2MPixel and 4MPixel are enlarged by 2.25x by using GPU only, CPU only and Device Fusion (in CorePilot 2.0), respectively. The execution time of each kernel is shown in below.

As shown in Figure 8 below, Device Fusion outperforms both GPU and CPU in three different resolutions by up to 55% and 59%, respectively. The outperformance is the result of the parallel execution of the major kernel, *find_neighbor*, and the dispatch the other kernels to the most suitable device. The excellence of the Device Fusion can be shown in this case study: it can intelligently determine the best dispatch way for real-world algorithms, achieving high efficiency/performance.

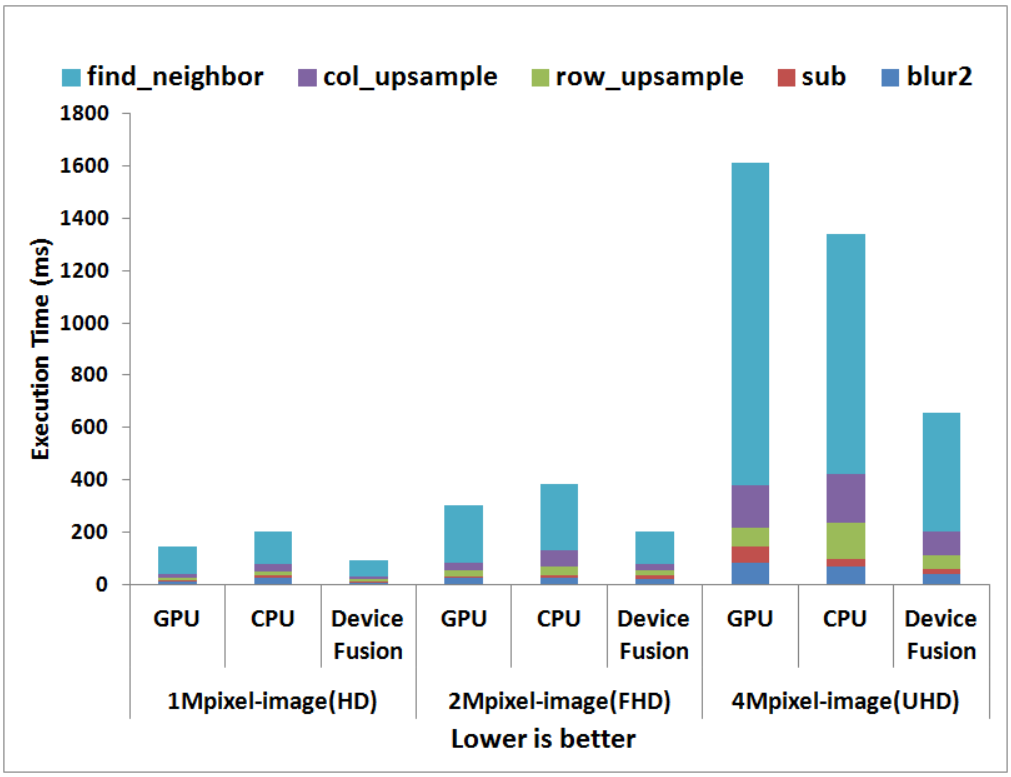

Figure 8. Execution Times Breakdown of Each Stage in SuperResolution under GPU, CPU and Device Fusion

Face Detection

Face detection is a frequently used method to recognize people's faces from an image. It is broadly used in mobile devices when taking pictures or device unlocking. The flow of face detection is shown in Figure 7 below. When the face detection program is executed via CorePilot 2.0, each function will be dispatched to the suitable device by the Device Fusion. For example, the function "IntegralStep1" is dispatched to GPU and the function "ViolaJones" is dispatched to CPU. For the functions dispatched to CPU, they are scheduled by the HMP CPU scheduler included in CorePilot. The performance and energy consumption results are also shown in Figure 9. According to the figure, Device Fusion leads to performance improvement by 22% and 146% when compared with CPU only and GPU only, respectively. This is because the CorePilot 2.0 can dispatch jobs to the most suitable heterogeneous computing devices. In addition to performance improvement, CorePilot 2.0 can reduce the energy consumption of face detection by 17% and 18% when compared with CPU only and GPU only, respectively. Such achievement is positive to user experience due to the performance improvement and energy consumption reduction.

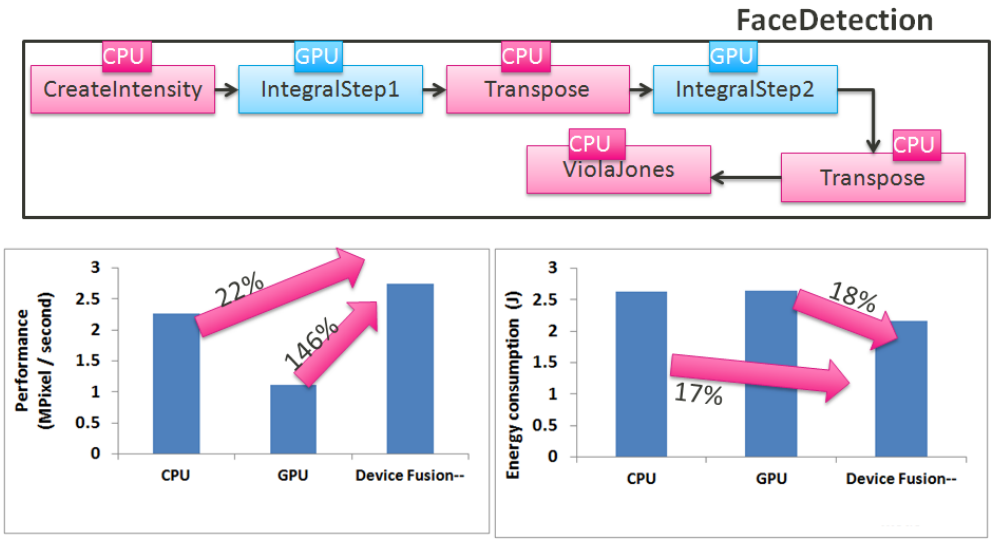

Figure 9. Flow of Face Detection and Performance/Energy Consumption Results Comparison

Conclusion

CorePilot 1.0 efficient management to symmetric or asymmetric multi-core CPUs to achieve load balance and better performance. The enhanced CorePilot 2.0, the CorePilot 1.0 successor `, cant both manage CPU cores, but also efficiently manage both CPUs and GPU in modern mobile device SoCs. CorePilot 2.0 includes Device Fusion technology that provides efficient computing by dispatching workloads programmed by OpenCL languages to the suitable device or to both devices. Since CPU and GPU expert at different workloads, CorePilot 2.0 can utilize both CPU and GPU to run various workloads to achieve better user experience by providing higher performance with lower power consumption.

In this paper, two multi-media case studies are shown to demonstrate the benefit of using CorePilot 2.0. Using both CPU and GPU in CorePilot 2.0 achieves up to 146% performance improvement when comparing to using CPU-only or GPU only. While achieving performance improvement, CorePilot 2.0 also leads to lower energy consumption by up to 18% when comparing to using CPU-only or GPU only.