Heterogeneous Computing with a Fused CPU+GPU Device

MediaTek White Paper

January 2015

1 Introduction

Heterogeneous computing technology, based on OpenCL (Open Computing Language), intelligently fuses GPU and CPU computing capability for high performance/efficiency computation. In the mobile SoC environment with SoC, heterogeneous computing technology exploits different on-chip computing resources to achieve high performance and efficient computing. This high performance/efficient computing can be achieved by executing the program on both CPU and GPU in parallel, or by dispatching the program to the most suitable device.

Recent findings suggest that Heterogeneous Computing is a more efficient way for computing when compared to homogeneous computing because different types of computing resources may better suit different workloads. For example, the CPU is generally good at control-intensive workloads while GPU at compute-intensive ones. Therefore, it is critical to intelligently exploit the heterogeneous computing resources to improve efficiency and performance.

However, it is challenging for programmers to write programs that can effectively utilize heterogeneous devices 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, Mediatek has introduced the heterogeneous computing platform **Device Fusion**, which can efficiently execute OpenCL programs by fusing GPU and CPU computing capability. As shown in Figure 1 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.

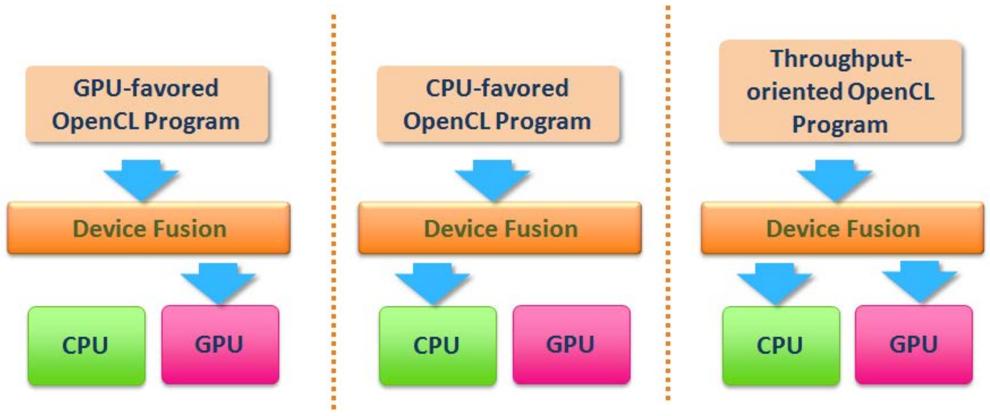
GPU-favored OpenCL Program
Device Fusion
CPU
GPU
CPU-favored OpenCL Program
Device Fusion
CPU
GPU
Throughput-oriented OpenCL Program
Device Fusion
CPU
GPU

Figure 1. Dispatch Options in Device Fusion

Two real-world case studies (large-size matrix multiplication and SuperResolution) and one popular OpenCL benchmark, CompuBenchCL, are utilized here to show the high performance/efficiency of the Device Fusion. Experiment results show that under the CompuBenchCL benchmark, Device Fusion outperforms GPU-alone and CPU-alone solutions by 20% and 151% in geometric mean, respectively. The results of large-size matrix multiplication shows the Device Fusion can provide better performance than GPU and CPU by up to 50% and 319%, respectively. Finally, the results of SuperResolution can also be improved by 55% and 59% when compared to GPU and CPU.

Heterogeneous Computing

Enabled by the advance of SoC manufacture capability, various Heterogeneous Computing resources are available on a single chip. Heterogeneous Computing refers to the computation of using computing resources with dissimilar instruction set architectures (ISAs). For example, a typical smartphone nowadays may be equipped with a CPU of more than 4 cores and GPU of more than 2 cores. Furthermore, it is expected that the number of CPU and GPU cores will continuously growing and more heterogeneous computing resources (e.g. DSP) will be incorporated into a single chip

OpenCL

OpenCL, a popular open standardized computing platform for heterogeneous computing, is designed to serve as the common high-level language for exploitation of heterogeneous computing resources. The OpenCL program can be executed on every device that supports the OpenCL standard APIs. For portability, an OpenCL kernel function is Just-In-Time (JIT) compiled into the corresponding ISAs for execution, after it is dispatched to an OpenCL-supported device.

2 The Problem

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 2. 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.

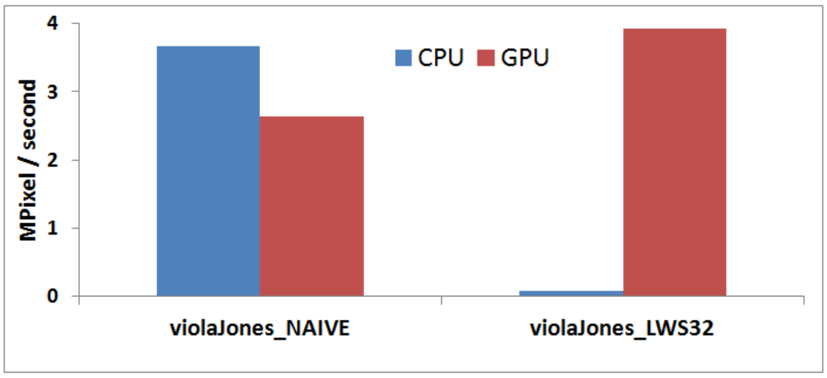

Figure 2. Performance Differences on CPU and GPU

- The computing capability of the target devices, where the programs are going to be executed on, 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 3 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.

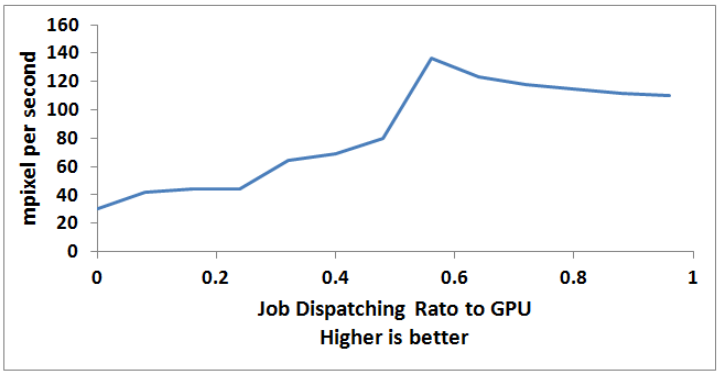

Figure 3. Performance of Parallel Executing Juliaset with Different Dispatch Ratios

- The final reason 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.

3 Solution

The Device Fusion is presented as a virtual OpenCL device. The virtual device, emulated on top of the CPU and GPU devices[1], 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 4 below.

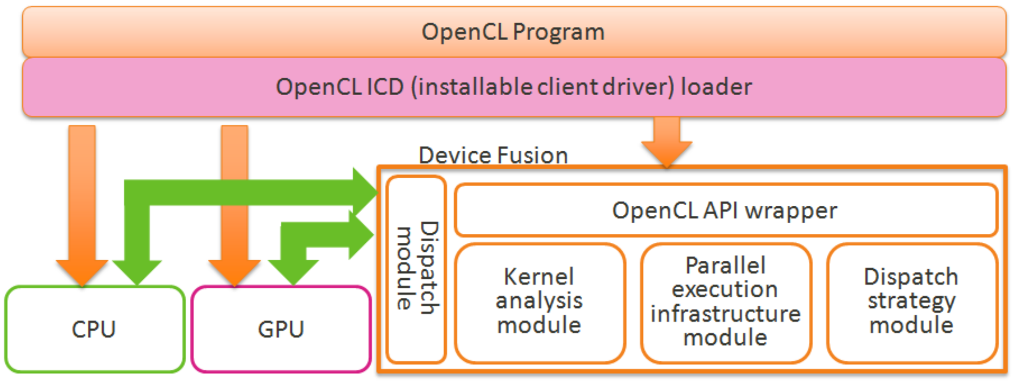

Figure 4. Overview of Device Fusion

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.

Experiment Results

Two real-world case studies and one popular OpenCL benchmark, CompuBenchCL, are utilized here to show the high performance/efficiency of the Device Fusion. The two real-

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

world case studies are large-size matrix multiplication and SuperResolution, an image-enhancement algorithm.

Case Study: Matrix Multiplication

Matrix multiplication is commonly used in many implementations of modern algorithms, such as object recognition and image processing. The execution times of a matrix multiplication implementation under GPU, CPU and DS are shown in Figure 5 below. Device Fusion outperforms GPU and CPU by 50% and 319%, respectively. This is the result of the parallel execution. Specifically, 72% and 28% work-items, intelligently determined by the internal algorithm of the Dispatch Strategy module, are dispatched to GPU and CPU.

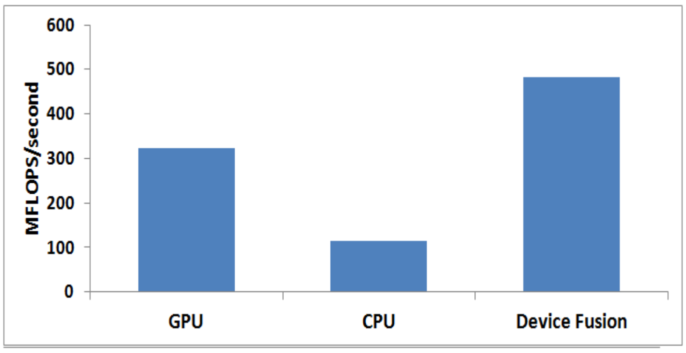

Figure 5. Performance of Matrix Multiplication under GPU, CPU and Device Fusion

Case Study: 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 this 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, CPU and Device Fusion, respectively. The execution time of each kernel is shown in Figure 6 below. From the figure, 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.

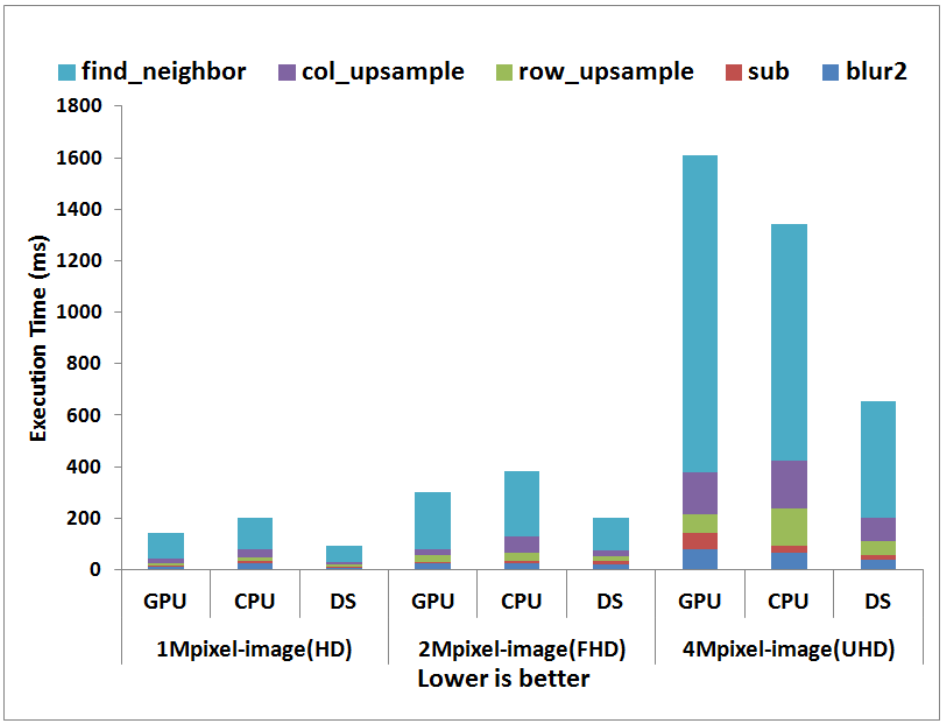

Figure 6. Execution Times Breakdown of Each Stage in SuperResolution under GPU, CPU and Device Fusion

CompuBenchCL

CompuBenchCL, a popular OpenCL benchmark, provides multiple comprehensive test items to measure the computing capability of the OpenCL device. Specifically, physics simulation, graphic computing, image processing and computer vision are included. The CompuBenchCL performance results are shown in the Figure 7 below.

The experiment results demonstrate that under the benchmark CompuBenchCL the Device Fusion can efficiently run each test item, leading to outperformance when compared with GPU and CPU by 20% and 151%, respectively, in geometric means. Similarly to SuperResolution, the performance improvement is the result of the dispatch to the correct device(s). This indicates the Device Fusion can used to improve performance of various scenarios by using OpenCL.

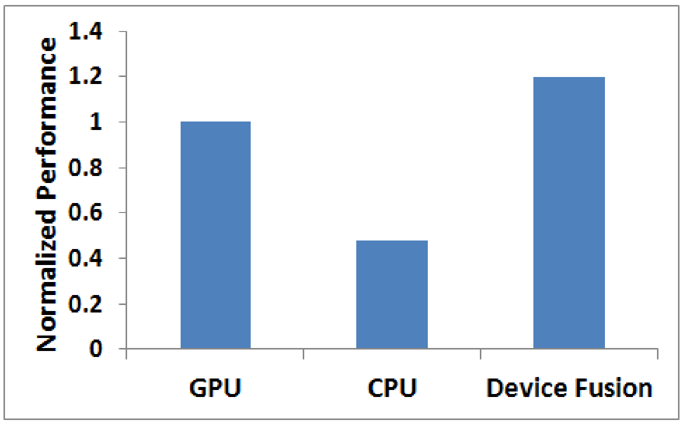

Figure 7.CompuBenchCL Performance Results