Current Optics and Photonics
Vol. 2, No. 5, October 2018, pp. 453-459

ISSN: 2508-7266(Print) / ISSN: 2508-7274(Online)
DOI: https://doi.org/10.3807/COPP.2018.2.5.453

Heterogeneous Computation on Mobile Processor for Real-time Signal Processing and Visualization of Optical Coherence Tomography Images

Jaehong Aum[1], Ji-hyun Kim[2], Sunghee Dong[2], and Jichai Jeong[2]*

[1]Department of Computer and Radio Communication Engineering, Korea University, Seoul 02841, Korea
[2]Department Brain and Cognitive Engineering, Korea University, Seoul 02841, Korea

(Received June 28, 2018 : revised August 9, 2018 : accepted September 4, 2018)

We have developed a high-performance signal-processing and image-rendering heterogeneous computation system for optical coherence tomography (OCT) on mobile processor. In this paper, we reveal it by demonstrating real-time OCT image processing using a Snapdragon 800 mobile processor, with the introduction of a heterogeneous image visualization architecture (HIVA) to accelerate the signal-processing and image-visualization procedures. HIVA has been designed to maximize the computational performances of a mobile processor by using a native language compiler, which targets mobile processor, to directly access mobile-processor computing resources and the open computing language (OpenCL) for heterogeneous computation. The developed mobile image processing platform requires only 25 ms to produce an OCT image from 512 × 1024 OCT data. This is 617 times faster than the naïve approach without HIVA, which requires more than 15 s. The developed platform can produce 40 OCT images per second, to facilitate real-time mobile OCT image visualization. We believe this study would facilitate the development of portable diagnostic image visualization with medical imaging modality, which requires computationally expensive procedures, using a mobile processor.

Keywords : Biomedical imaging, Optical coherence tomography, Biophotonics
OCIS codes : (170.3880) Medical and biological imaging; (110.4500) Optical coherence tomography; (110.4155) Multiframe image processing

I. INTRODUCTION

Optical coherence tomography (OCT) is an interferometric imaging modality that is frequently used for clinical and preclinical applications in dermatology, ophthalmology, and otolaryngology [1-5]. It can produce high-resolution tomograms at the few-micrometers scale with noninvasive scanning. However, OCT requires complex signal processing that demands intensive computing to produce a tomographic image, which is potentially a bottleneck to increasing the imaging rate of OCT, and limits its utility when computing resources are limited. Consequently, many solutions have been proposed for processing OCT data in real time using high-performance computing tools, such as a multi-CPU connected system [6], field-programmable gate arrays (FPGAs) [7], and compute unified device architecture (CUDA) -assisted high-performance graphics processing units (GPUs) [8-11].

In recent times, various portable medical diagnostic devices have been proposed [12-16]. Such devices can be utilized in numerous situations, such as emergency onsite cases, home monitoring of medical status, and providing healthcare systems for developing countries. Developing a portable OCT device for such purposes would also be beneficial. It can be applied in tracking the progress of glaucoma, diabetic macular edema, and other ophthalmic or dermatologic diseases, for instance [12, 17-19]. However, we expect there will be several shortcomings of portable OCT devices versus a conventional OCT system, such as degraded resolution, low signal-to-noise ratio, low frame rate, etc. One of our major concerns is low frame rate. Conventional high-performance processing devices are relatively bulky and consume much power, which limits

*Corresponding author: jcj@korea.ac.kr, ORCID 0000-0003-0225-8534
Color versions of one or more of the figures in this paper are available online.

their usage for portable devices. On the other hand, a processing device that can be installed in a mobile device for its smaller size and low power consumption shows low computational performance. This may demand an excessively long time to obtain OCT image visualizations. The critical hurdles here are developing a mobile image processing platform for OCT and utilizing it for practical applications. Currently, it is difficult to implement OCT hardware using mobile devices, except for the optics. In the near future, we may expect to implement OCT hardware including OCT-control and image-processing parts on mobile devices. Before implementing OCT hardware on mobile devices aside from the optics part, we would like to identify the limit steps in image processing that cause the process to take a long time.

A system-on-a-chip (SoC) is an integrated circuit that contains all components in a single chip. A SoC typically contains a memory block, microprocessor, and data I/O interface. Owing to their low power consumption and miniaturize size, SoCs are commonly used in mobile electronic devices. Further, high-performance SoCs called mobile processor have been developed for smartphones, to process multimedia content. The microprocessor of a mobile chip consists of multiple processing units, such as a central processing unit (CPU), digital signal processor (DSP), and GPU to process multimedia content, as shown in Fig. 1. The computational performance of each processing unit is not significant, but we believe utilizing multiple processing units in a parallel computation scheme would greatly increase performance. To the best of our knowledge, an OCT image processing that solely runs on a mobile processor has never been investigated. Our primary interest is to develop a fully optimized scheme for processing raw OCT image data and visualizing those images in real time on a mobile device. Furthermore, we investigate the possibility of developing a real-time OCT image platform using a mobile processor.

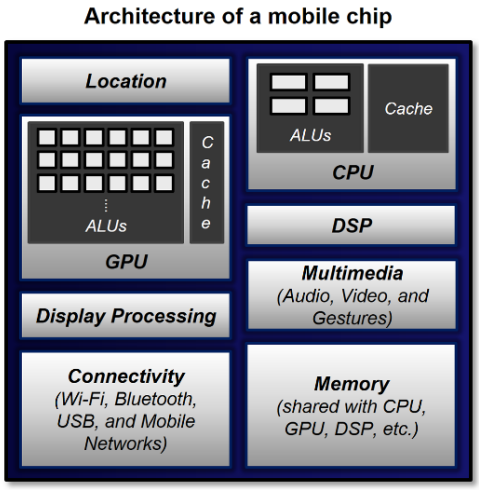

FIG. 1. Architecture of a Snapdragon 800 mobile processor. The mobile processor consists of multiple processing units, data I/O interface, geometric location-tracking system, and a shared memory block.

In this paper, we present a mobile image-processing platform developed for mobile processor with a fully optimized processing scheme that processes OCT data and renders images. The image-processing platform uses a Snapdragon 800 mobile processor installed in a Samsung SHV-E330S smartphone. It was developed as an Android application for compatibility with the Android OS that dominates the smartphone market (about 85% market share in 2017 [20]). Without optimization for high-speed computation, the mobile devices required more than 15 s to produce an OCT image from 512 × 1024 OCT raw data, which is too slow to be applied to real-time visualization. Consequently, to increase its computation speed we introduced an accelerated processing system, which we call HIVA. Initially we employed a native development kit (NDK), which facilitates direct access to the computing resources of the device by compiling the native language into machine-language code targeting the mobile processor. With the NDK adopted, the time to produce an OCT image was reduced to 1,160 ms, which represents a 13.3-fold increase in computing performance. Subsequently, for further acceleration of the devices we adopted the open computing language (OpenCL), which facilitates heterogeneous computation using multiple processing units of a mobile processor. By using precomputed data to avoid unnecessary recomputation, full utilization of multiple cores of the GPU, and maximization of data access throughput with consideration of coalesced memory access, the OpenCL processing scheme was optimized. With the implementation of our proposed mobile platform, the time to produce an OCT image was further reduced to 25 ms, an additional 46.4-fold acceleration. This allows an imaging rate of 40 frames per second, as achieved by real-time visualization. Compared to the initial version, which took 15 s to produce an OCT image, our proposed method shows 617-fold acceleration.

II. METHODS

We introduce HIVA for the purpose of maximizing the computational performance of a mobile processor, which runs on the Android OS, when it processes raw OCT-signal data and renders the produced tomographic images as an Android application for an OCT imaging system. HIVA accelerates the processing speed by skipping unnecessary, heavily delaying procedures while executing functions in runtime, with an alternative method for executing functions with a native-language-coded system. It exploits multiple processing units of a mobile processor with high-performance parallel computation to minimize the time consumed for the OCT signal processing tasks, which demand heavy computational overhead.

2.1. Overview of High-performance Computation on Android Applications

Android applications are developed using Java, an interpreted language that executes in a virtual machine. It does not allow direct access to device resources. These features facilitate program compatibility and stability, which are critical for Android applications running on various devices. However, algorithms written in Java execute more slowly than algorithms written in native languages such as C/C++, which execute their instructions on the processor directly, as described in Fig. 2. The performance gap between an interpreted language and a native language becomes severe when it comes to computationally intensive work. To overcome the performance issues associated with Java, a native language compiler that produces machine code that runs on mobile processor is needed. Furthermore, many C/C++ libraries and APIs that enable high-performance computation on a device exploiting multiple processing units, such as open multiprocessing (OpenMP), open graphics library (OpenGL), and OpenCL, can be adopted.

OpenMP facilitates multithreaded computation with a simple modification of existing code. This multithreaded computation accelerates computing speed by utilizing multiple CPU cores concurrently. With OpenMP, acceleration of twofold or threefold can generally be expected. However, computing performance is limited when only a CPU is used. OpenGL and OpenCL allow parallel computation involving a GPU, which is a high-performance parallel-computing-optimized processing unit. With the aid of GPUs, a huge acceleration in OCT data-processing speed can be expected, as the OCT data-processing task is highly amenable to parallel computation. OpenGL was originally developed to process graphics tasks, while OpenCL was developed for application to parallel computing across heterogeneous platforms consisting of CPU, GPU, DSP, etc., to perform general tasks rather than graphics tasks. Unlike OpenMP, adopting OpenGL or OpenCL is complex; however, the adoption promises much higher computational performance.

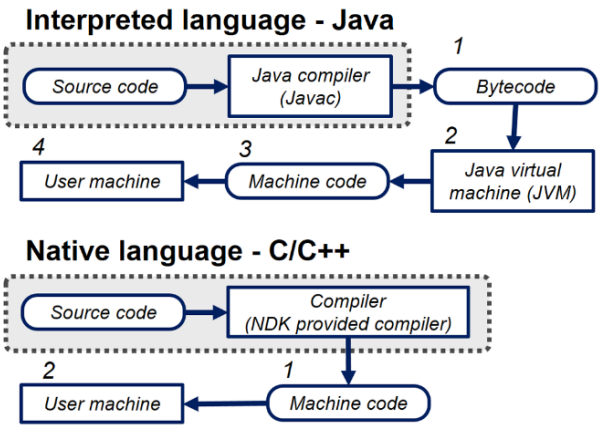

FIG. 2. Illustration of the steps from source codes, written in native language and interpreted language respectively, to be executed on a user machine. Gray boxes indicate the steps performed before runtime. A native-language-written program needs fewer steps to be executed in runtime.

2.2. Heterogeneous Computation Scheme for Concurrent Utilization of Multiple Processing Units from a Mobile Processor

To develop HIVA, we used the following development tools: Android Studio 1.4, Android SDK 5.0.1, Android NDK 11.0.0, Gradle 2.8, OpenGLes 2.0, OpenCL 1.1, and Java 1.7. Android Studio provides an integrated development environment, Gradle is a build tool for Android applications, and OpenGLes is a version of OpenGL that is targeted at mobile devices. We developed the main structure of the application, including the user interface and data I/O system, in Java. However, the sections dealing with intensive computing tasks, such as OCT signal processing and image rendering, were written in C/C++ with the aid of the NDK. NDK is a set of compilers that compiles native-language source code into machine code, targeting various types of mobile processing devices. With the NDK, we can compile C/C++ code into machine code that can run directly on the processor of a mobile chip. However, the C/C++ -written and -compiled codes cannot be called directly, unless a Java method calls them, since an Android application should be written in Java. For the purpose of calling the machine code with a Java method, we defined in Java the native methods associated with the image-rendering and OCT-signal-processing tasks, as shown in Fig. 3. The native methods were incomplete methods, without implementation code. We wrote the implementation of the methods as

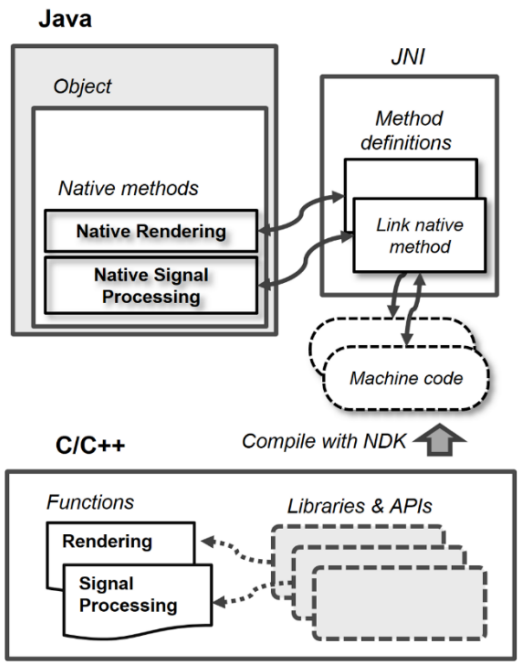

FIG. 3. Illustration of the relationship between the Java-coded methods and C/C++ -coded functions associated with NDK and JNI.

C/C++ functions that are then compiled into machine code. The separate definition and implementation for each method are then linked through the Java native interface (JNI). When the native Java methods are called, JNI calls the corresponding natively coded functions and passes arguments between them. In this manner, we can execute C/C++ machine code by calling a Java method in an Android application.

For those parts of programs that are not specifically to be executed on a single-core CPU, we adopt OpenGL and OpenCL over OpenMP to exploit the computing power of the GPU contained in a mobile processor. To initialize OpenGL to render OCT images, we allocate a two-dimensional (2D) buffer for each OCT image. In addition to writing basic GPU rendering tasks via shader code, we wrote modules for adjusting the contrast and brightness levels of the image manually, to maximize the visibility of the sample obtained from an OCT image. In contrast to OpenGL, OpenCL has a relatively complex initialization process. First, we select the processor we wish to use with the OpenCL API, which in our case is the GPU. Next, we allocate GPU buffers to save and process OCT data. OpenCL defines functions called *kernel functions*, to run on GPUs. However, prior to launching these kernel functions it is necessary to compile their associated code into the machine code of the GPU, using the OpenCL compiler. OCT-signal-processing tasks are written as kernel functions and compiled into machine code.

Figure 4 illustrates the schematic for our real-time mobile-device OCT system. First, we load OCT raw data from the Java side, and call the native method to process OCT data from the C/C++ side. When the method is called for the first time, OpenGL and OpenCL are initialized prior to signal processing and image rendering. To produce OCT images from OCT raw data, we use *k*-space resampling with linear interpolation, removal of DC signal, and IFFT, as we did in a previous study, in which CUDA was used to develop a real-time OCT system [8]. Prior to launching the kernel functions from OpenCL, the data must be transferred to the OpenCL device buffer, which is the GPU buffer in our case. While launching the kernel functions,

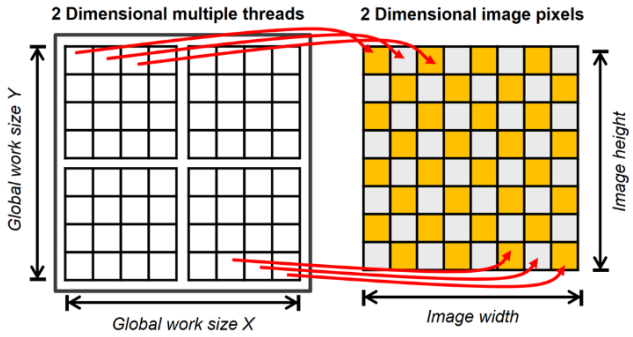

FIG. 5. One-to-one matching relation between two-dimensional multiple threads and two-dimensional-image pixel data.

OCT signal processing is performed using the GPU. In this process we create two-dimensional multiple threads, which are dedicated one-to-one to the same number of each OCT image's pixels, to process each pixel's data in parallel using OpenCL, as shown in Fig. 5. In this manner we can maximally utilize hundreds of processing cores, keeping a minimal number of cores idling. To avoid unnecessary repeated computation, we use precomputed information about the OCT signal, such as the DC spectrum and interpolation position indices. The dataset of the information was organized and saved in one-dimensional buffers in serial order, to satisfy the coalesced memory-access condition when we access the dataset [21, 22]. The processed data from the GPU computation are then transferred to the vertex buffer, which holds the OCT image for rendering on a display, using OpenGL. The image rendering via OpenGL is performed asynchronously, regardless of the signal processing task.

III. RESULTS

We used a Samsung SHV-E330S smartphone to verify the efficacy of our system. This model smartphone has a Qualcomm Snapdragon 800 mobile processor, which comprises a 2.26 GHz four-core CPU and a GPU containing

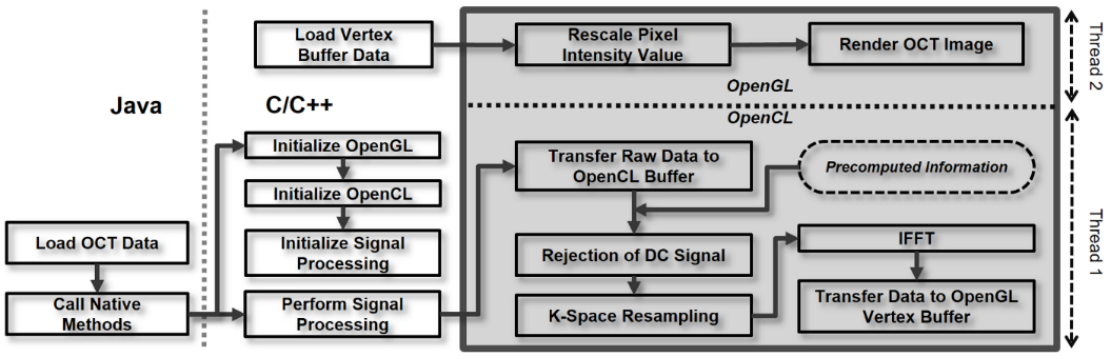

FIG. 4. Schematic of the proposed OCT signal processing and image rendering, fully optimized for high-speed computation on mobile processor.

128 arithmetic logic units (ALUs). 2 GB of LPDDR3 RAM with 800 MHz memory frequency is installed, which is shared by the CPU, GPU, and other processing units. The OS version used was Android 5.0.1, and the applications were built targeting this same Android OS version. We subsequently measured the processing time to produce an OCT image from raw OCT data with our proposed system. Identical OCT-signal-processing systems were also developed with different methods, and the processing times of those systems measured, to compare computational speeds. The sample OCT data used were 512 × 1024 2D data, obtained by scanning an African clawed frog (*Xenopus laevis*) with a fiber-based spectral-domain OCT (SD-OCT) [9]. The original bit depth of the data was 12 bits in the form of unsigned integers, and then we converted it into 32-bit floating numbers for numerical stability while processing the data. The size of the raw OCT datafile for a single OCT image was 2 MB.

Table 1 shows the processing times required to produce the OCT images from raw OCT data under the various processing systems. For this test, we preloaded the data in memory and fetched it every time for a new OCT image. The produced images were displayed on the 1080 × 1920-resolution supported AMOLED display installed on the smartphone. First, the Java-coded system required almost 15 s to produce an OCT image. With the help of the NDK, a C/C++ -coded system was developed that took 1160 ms, approximately 13 times as fast as the Java system. Further, when OpenCL was adopted for our proposed system, processing took only 25 ms, which is a 617-fold increase in speed compared to the Java-coded system. This result proves that our system is capable of producing 40 OCT images per second, to realize real-time imaging. The OCT images produced with our proposed system are presented in Fig. 6.

Accelerating the computing speed with OpenCL using GPU computation is relatively complicated. OpenMP, another tool for accelerating computing speed, is easier to apply. For comparison, we applied OpenMP to our system and subsequently measured the processing time of the system. With the application, the processing speed increased approximately fourfold, taking 258 ms to produce an OCT image by utilizing multiple cores of the CPU, compared to the single-core case that took 1160 ms. Adopting OpenMP is simpler than adopting OpenCL, resulting in improved computational performance. We consider applying OpenMP instead of OpenCL with GPU computation, when the GPU is fully dedicated to other processing tasks, or the system does not require heavy computation.

In Table 2 we present the computation time for each step of producing an OCT image, to investigate the performance gain we achieved in each step by applying our heterogenous computation system on a mobile processor. The most significant performance gain was observed in the IFFT step, which has a complexity of $O(n \log(n))$, while the other steps are $O(n)$, where n is the data size. Compared to the single-core case, the GPU case showed about 180-fold increase in computing speed. On the other hand, the GPU case required data-transfer steps between host and device before GPU computation, which was unnecessary for the CPU-only case. Unlike desktop GPUs, a mobile chip's GPU does not possess its own physical memory, which is often referred to as device memory, but it shares a unified physical memory with the CPU. However, it is still necessary to transfer data between host and device for GPU computation, because the host and device are not physically separated, but virtually divided. The data-transfer step from host to device took about 1.7 ms, only 6.7% of the total processing time of 25.1 ms. We also measured the processing time using a desktop GPU Nvidia GTX680

TABLE 1. Comparison of the speed at which an OCT image is produced, with processing systems under different conditions

Type of system	Java coded system	C/C++ coded system		HIVA
Library used	-	-	OpenMP	OpenCL
Processor (number of utilized ALUs)	CPU (1 core)	CPU (1 core)	CPU (4 cores)	GPU (128 cores)
Processing time	15,423 ms	1160 ms	258 ms	25 ms
Relative speed	1	13.30	59.78	616.92

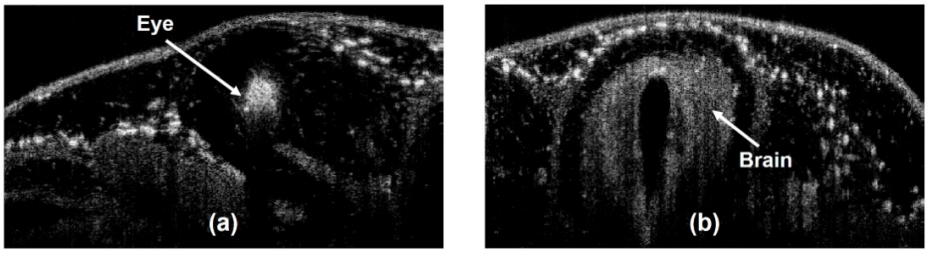

FIG. 6. Our proposed technique produced OCT sample images taken from the (a) eye and (b) brain regions of an African clawed frog (*Xenopus laevis*). These images were rendered on a display panel from SHV-E330S and captured by a capturing application.

TABLE 2. Comparison of the processing time for each step to produce an OCT image, with and without a multicore parallel processing scheme

Type	Mobile processor (Snapdragon 800)			Desktop GPU (Nvidia GTX680)
	CPU (1 core)	CPU (4 cores)	GPU (128 cores)	GPU (1536 cores)
Data transfer (host to device)	-	-	1.7 ms	1.56 ms
DC elimination	34.8 ms	14.1 ms	5.2 ms	0.12 ms
K-space resampling	119.3 ms	32.3 ms	6.1 ms	0.57 ms
IFFT	993.7 ms	205.9 ms	5.5 ms	0.17 ms
Copying produced image to pixel buffer	12.2 ms	5.6 ms	6.6 ms	0.46 ms
Total processing time	1160.0 ms	257.9 ms	25.1 ms	4.67 ms

for the same processing steps, using CUDA. The estimated total processing time using the desktop GPU was 4.67 ms, which is about 5 times as fast as HIVA.

IV. DISCUSSION

In the case of a desktop GPU, the data-transfer step took about 1.56 ms, taking a significant amount of the total processing time (33.4%). In fact, the data-transfer throughput of the desktop GPU was 5.82 GB/s, while the mobile processor's was 4.51 GB/s, showing little difference. For a desktop GPU, it is critical to minimize data-transfer time, which required several optimization techniques to be introduced, such as using page-locked memory with direct-memory-access units (DMA), asynchronous data streaming, etc. However, it was not a critical issue for the mobile processor, taking only 6.7% of total processing time in our case, without employing such methods.

We have tested our system with the raw OCT data preloaded into the memory. For real applications, raw data should be transferred from an OCT hardware device to the memory. However, it may not be a critical concern, since there are some high-bandwidth interfaces for real-time data transfer. For instance, SATA III, USB 3.0, and USB 2.0 support 768, 625, and 60 MB/s, respectively. Since our raw-datafile size for a single OCT image was 2 MB (768 kB before conversion to 32-bit floating variable), 80 frames of OCT data can be theoretically transferred per second even using just USB 2.0, which would be enough for our system.

We have demonstrated a significant acceleration in OCT image production on a mobile processor. However, we believe further improvement of the system is possible. There exists latency when we launch each OpenCL kernel function for GPU computation. Generally, the latency is much higher than the latency observed when launching common C/C++ -coded functions. We also observed the high latency by estimating the elapsed time for launching a simple OpenCL kernel function that performs element-wise addition in the GPU; it took about 4.9 ms. Since most of the steps took between 5 and 7 ms, the latency can be considered a critical bottleneck of the system. We expect that further improvement of the processing speed can be realized by integrating the multiple kernel functions into a single kernel function that is carefully designed concerning synchronization of multiple threads. We leave this for our future work, including the development of a real-time-imaging, fully portable OCT device.

V. CONCLUSION

In this paper we developed a high-performance signal-processing and image-rendering heterogeneous computation system for optical coherence tomography (OCT) on mobile processor, with the introduction of HIVA. Over the past decade, various high-performance processing units have been adopted to facilitate the development of real-time OCT systems. However, to the best of our knowledge this is the first system to employ a mobile processor to process OCT data in real time. The mobile platform was developed as an Android application running on a smartphone (Samsung SHV-E330S) with a Snapdragon 800 mobile processor. The initial OCT-data-processing scheme without HIVA required approximately 15 s to produce a 512×1024 OCT image from raw OCT data. NDK was adopted to reduce the processing time to 1160 ms with direct access to the computing resources of the mobile processor. Furthermore, we performed heterogeneous computation using OpenCL along with full optimization of the processing scheme, focusing on maximum utilization of the mobile processor's processing resources. This resulted in further reduction of the processing time to 25 ms. The processing-time reduction allows us to realize real-time systems able to process 40 OCT images per second. Our developed technique exhibited a 617-fold increase in computation speed, compared to the 15 s required by the unoptimized system. This proves that it is possible to develop real-time OCT image visualization using a single mobile processor, if it has a fully optimized processing system that utilizes heterogeneous computation.

ACKNOWLEDGMENT

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF), funded by the Ministry of Education under Grant NRF-2015R1D1A1A01056746.

REFERENCES

1. E. Gibson, M. Young, M. V. Sarunic, and M. F. Beg, "Optic nerve head registration via hemispherical surface and volume registration," IEEE Trans. Biomed. Eng. **57**, 2592-2595 (2010).
2. Y.-C. Ahn, Y.-G. Chae, S. S. Hwang, B.-K. Chun, M. H. Jung, S. J. Nam, H.-Y. Lee, J. M. Chung6, C. Oak, and E.-K. Park, "*In vivo* optical coherence tomography imaging of the mesothelium using developed window models," J. Opt. Soc. Korea **19**, 69-73 (2015).
3. R. Kafieh, H. Rabbani, F. Hajizadeh, and M. Ommani, "An accurate multimodal 3-D vessel segmentation method based on brightness variations on OCT layers and curvelet domain fundus image analysis," IEEE Trans. Biomed. Eng. **60**, 2815-2823 (2013).
4. P. Li, X. Yin, L. Shi, A. Liu, S. Rugonyi, and R. K. Wang, "Measurement of strain and strain rate in embryonic chick heart in vivo using spectral domain optical coherence tomography," IEEE Trans. Biomed. Eng. **58**, 2333-2338 (2011).
5. N. H. Cho, U. Jung, H. I. Kwon, H. Jeong, and J. Kim, Development of SD-OCT for imaging the *in vivo* human tympanic membrane," J. Opt. Soc. Korea **15**, 74-77 (2011).
6. G. Liu, J. Zhang, L. Yu, T. Xie, and Z. Chen, "Real-time polarization-sensitive optical coherence tomography data processing with parallel computing," Appl. Opt. **48**, 6365-6370 (2009).
7. T. E. Ustun, N. V. Iftimia, R. D. Ferguson, and D. X. Hammer, "Real-time processing for Fourier domain optical coherence tomography using a field programmable gate array," Rev. Sci. Instrum. **79**, 114301 (2008).
8. J.-H. Kim, J. Aum, J.-H. Han, and J. Jeong, "Optimization of compute unified device architecture for real-time ultrahigh-resolution optical coherence tomography," Opt. Commun. **334**, 308-313 (2015).
9. N. H. Cho, U. Jung, S. Kim, W. Jung, J. Oh, H. W. Kang, and J. Kim, "High speed SD-OCT system using GPU accelerated mode for *in vivo* human eye imaging," J. Opt. Soc. Korea **17**, 68-72 (2013).
10. Y. Watanabe and T. Itagaki, "Real-time display on Fourier domain optical coherence tomography system using a graphics processing unit," J. Biomed. Opt. **14**, 060506-060506-3 (2009).
11. Y. Huang, X. Liu, and J. U. Kang, "Real-time 3D and 4D Fourier domain Doppler optical coherence tomography based on dual graphics processing units," Biomed. Opt. Express **3**, 2162-2174 (2012).
12. W. Jung, J. Kim, M. Jeon, E. J. Chaney, C. N. Stewart, and S. A. Boppart, "Handheld optical coherence tomography scanner for primary care diagnostics," IEEE Trans. Biomed. Eng. **58**, 741-744 (2011).
13. S. Dong, K. Guo, P. Nanda, R. Shiradkar, and G. Zheng, "FPscope: a field-portable high-resolution microscope using a cellphone lens," Biomed. Opt. Express **5**, 3305-3310 (2014).
14. T. Li, M. Duan, K. Li, G. Yu, and Z. Ruan, "Bedside monitoring of patients with shock using a portable spatially-resolved near-infrared spectroscopy," Biomed. Opt. Express **6**, 3431-3436 (2015).
15. K. Daoudi, P. J. van den Berg, O. Rabot, A. Kohl, S. Tisserand, P. Brands, and W. Steenbergen, "Handheld probe integrating laser diode and ultrasound transducer array for ultrasound/photoacoustic dual modality imaging," Opt. Express **22**, 26365-26374 (2014).
16. D. D. Mehta, N. T. Nazir, R. G. Trohman, and A. S. Volgman, "Single-lead portable ECG devices: Perceptions and clinical accuracy compared to conventional cardiac monitoring," J. Electrocardiol. **48**, 710-716 (2015).
17. X. Xu, L. Yu, and Z. Chen, "Effect of erythrocyte aggregation on hematocrit measurement using spectral-domain optical coherence tomography," IEEE Trans. Biomed. Eng. **55**, 2753-2758 (2008).
18. S. Lee, E. Lebed, M. V. Sarunic, and M. F. Beg, "Exact surface registration of retinal surfaces from 3-D optical coherence tomography images," IEEE Trans. Biomed. Eng. **62**, 609-617 (2015).
19. F. Atry, S. Frye, T. J. Richner, S. K. Brodnick, A. Soehartono, J. Williams, and R. Pashaie, "Monitoring cerebral hemodynamics following optogenetic stimulation via optical coherence tomography," IEEE Trans. Biomed. Eng. **62**, 766-773 (2015).
20. Smartphone OS Market Share, 2017 Q1, https://www.idc.com/promo/smartphone-market-share/os (2017).
21. G. Chen, X. Shen, B. Wu, and D. Li, "Optimizing data placement on GPU memory: A portable approach," IEEE Trans. Comput. **99**, 1-1 (2016).
22. R. Melo, G. Falcao, and J. P. Barreto, "Real-time HD image distortion correction in heterogeneous parallel computing systems using efficient memory access patterns," J. Real Time Image Process. **11**, 83-91 (2016).