STATE UNIVERSITY
OF BANGLADESH

Application of Parallel Computing Models for Image Processing in Smartphone Devices

M. Asif Hossain

A thesis Presented for the
Parallel Processing and Distributed
System Lab

CSE-0410
Department of Computer Science and Engineering
14th September 2021

Acknowledgements

For guidance, support, and feedback throughout this Thesis, I would like to express my sincere gratitude to my Course Advisor **Khan Md. Hasib.**

Abstract

Today powerful parallel computer architectures empower numerous application areas in personal computing and consumer electronics and parallel computation is an established mainstay in personal mobile devices (PMD) and smartphones. During the last fourteen years, PMDs and smartphones have been equipped with increasingly powerful parallel computation architectures (CPU+GPU) enabling rich gaming, photography, and multimedia experiences ultimately general-purpose parallel computation through application programming interfaces.
This study views the current status of parallel computing and parallel programming, and specifically, its application and practices of digital image processing applied in the domain of Mobile Systems (MS), Personal Mobile Devices (PMD), and smartphones. The application of parallel computing and -programming has become more common today with the changing user-application requirements and with the increased requirements of sustained high-performance applications and functionality. Furthermore, the paradigm shift of data consumption in personal computing towards PMD and mobile devices is under increased interest. The history of parallel computation in MS, PMD, and is a relatively new topic in academia and industry. The literature study revealed that while there is a good amount of new application-specific research emerging in this domain, the foundations of dominant and common parallel programming paradigms in the area of MS, PMD smartphones are still moving targets.

Contents

Chapter 1

Introduction

The need for accelerated information processing and computation through scalable parallel execution has existed for several decades. The development and evolution of computer architectures have made parallel computation more accessible and cost-effective than ever before. The amount of created and stored information has been rapidly increasing in many areas of human life, in science and engineering, industry, medicine, and entertainment. Different types of requirements drive the development and application of parallel processing. In science and engineering, High-Performance Computing (HPC) and Massively Parallel Processing (MPP) systems comprise numerous processing units, often consisting of hundreds and even millions of processing cores to perform the required application execution simultaneously. Modern Numerical Weather Prediction (NWP), complex simulation models in astronomy and medicine employ massively distributed parallel processing applications utilizing enormous communication- and storage resources. Consumer-level multimedia and graphics computation, smartphones, and tables often require real-time parallelism of multiple concurrent processing units to accomplish the tasks of visual content processing we are accustomed to media interfaces today (Grama, 2003). Today, a smartphone or a tablet has become the sole personal computer for many people. Information browsing and acquisition, information consumption and processing, collaboration, and sharing are increasingly conducted with smartphones and portable tablets. The ever-changing landscape of services and applications people use every day is transforming the applications and their processing and performance requirements. For example, Mobile Augmented Reality (MAR) has been a trending topic among mobile application developers in recent years.

MAR adds a segment of mobile applications that were not possible to implement until recent developments in mobile device hardware and software technology. For example, already today furniture manufacturers provide mobile applications that allow users with a smartphone camera to augment their home spaces with furniture or lightning fixtures in real-time to help their customers to make purchase decisions. Similarly, a cosmetics company provides a smartphone app where users can apply artificial make-up on one's face in real-time, still making it look like it was real (Wired, 2017; Modiface, 2017). In the landscape of emerging new applications, the increasing efficiency and processing requirements push the future of computing towards parallelism. In specific computation setups, the new applications, modern games, and scientific- and engineering applications have requirements that are constrained by the CPU-only performance. A new general computing model has emerged where the CPU works together with the graphics processing unit (GPU). In these parallel configurations, typically containing several multi-core processing units (e.g. CPU and GPU), can reside potential in responding to increased computation requirements (Owens, Houston, Luebke, Green, Stone, Phillips, 2008)

1.1 Purpose and Motivation of the Study

The motivation for the study emerged from the author's personal interests in image processing and the ambition to build a real-time video processing application for Apple iOS-platform iPhone, tablets, and Android smartphones, tablets. Application of digital image processing for video effects processing typically requires massive computation power as the information required to be processed is vast. Apple iOS Software Development Kit (SDK), Android Software Development Kit (SDK), Apple and Android devices have included a powerful programmable GPU. Those advanced technologies come with some of these complex abstractions and the author's main interest is to break through these abstractions.

Chapter 2

Prior Research

The following chapters will create a brief outlook on the topics of this review. First, parallel computing and parallel programming models are presented followed by their applicability on current mobile systems and personal mobile devices. Furthermore, digital image processing is briefly described.

2.1 Parallel Computing and Programming Model in MS and PMDs

Many embedded devices and Personal Mobile Devices (PMDs) today are energy-efficient computers hosting multi-core CPUs and GPUs with integrated peripheral devices, cameras, sensors, and rich wireless connectivity. PMD parallel computing environments share many of the same principles as those applied in desktop and server systems. A typical PMD is powered from a rechargeable battery power source and the main system components including processing units, memory, and peripheral devices are typically coupled inside a single physical device. Furthermore, there are several constraining differences between the two environments. For example, a mobile device has a battery power source, meaning the power provided for all the system components is only a fraction of what is available on a desktop computer. Mainly due to this fundamental power constraint, processor cores' operating speeds, memory bus width, and -speed typically fall greatly behind those available in a desktop computer (Singhal, Yoo, Choi, Park, 2010). Modern PMD platforms support many types of parallelism on the hardware- and programming levels. To reach the additional processing power provided by the mobile het-

erogeneous system, programming access to the device GPU is required. The GPU hardware programming interfaces on common PMD platforms, such as Android and iOS, are mainly supporting graphics programming language-oriented models (Singhal, Yoo, Choi, Park, 2010).

2.2 Digital Image Processing

The digital image is a bounded, two-dimensional numerical representation of a real-world image. This discrete image has been constructed into pixels (individual points in the two-dimensional plane). In a digital image, each pixel has a discrete value of intensity or gray level, which defines the information value of the pixel. Digital image processing means the processing of (these) digital images utilizing computer programs and algorithms.
Digital image processing is a vast science and practice extending beyond human senses, capable of applying methods into discrete data captured on almost the entire electromagnetic spectrum from gamma- to radio waves. Digital image processing is often discussed with other image processing sciences such as digital image analysis and computer vision. One of the first practical applications of digital image processing was employed by U.S. space exploration in the mid-1960s.
The available computing power and development of image processing algorithms brought the science into practical significance; distorted images from space probes could be recovered and meaningful information could be restored by means of digital image processing. Today, the science of digital image processing is vast and is providing invaluable meaning to almost any area of industry, science, and human life (Gonzalez & Woods, 2017)

Gonzalez & Woods (2017) group the application of digital image processing into ten sub-categories:

1. Image acquisition
2. Image enhancement 10
3. Image restoration
4. Wavelets and other image transforms
5. Color image processing

6. Compression and watermarking
7. Morphological processing
8. Image segmentation
9. Feature extraction
10. Image pattern classification

2.3 Scope of the Research

The purpose was to build a real-time video effect processing application that exhaustively utilizes the GPGPU programming model on Apple iOS and Android environments. While the review of available literature seeks to be broad, specific narrowing criteria and selection is applied to limit the scope of literature to serve those aforementioned theses.

The research question for this study is:

RQ1: What parallel computing architectures and parallel programming models are available now on mobile systems and personal mobile devices to apply complex CPU-GPU computing?

Chapter 3

Research Method

The research method for this study Systematic Search and Review consisted of searches available that are combined strengths of critical review with a comprehensive search process. Typically addresses broad questions to produce the best evidence synthesis.
The appraisal may or may not include quality assessment and synthesis are minimal narrative, a tabular summary of studies. The analysis will consist of What is known; practice recommendations. What remains unknown; uncertainty around findings, recommendations for future research.

3.1 Sources of Literature

1. IEEE Xplore Digital Library: To finding correct information, direct access to the IEEE Xplore interface will be used.

2. ACM Digital Library: Same as IEEE Xplore, to find out accurate information, direct access to the ACM Digital Library interface.

3. Google Scholar Search: If the two aforementioned failed to give results, Google Scholar Will be able to find unofficial links/scans/photographs of documents of interest.

4. The State University of Bangladesh Library and moodle.sub.edu.bd: SUB Library has many research papers and books which will help in this thesis and moodle.sub.edu.bd has all content for the all courses such as PDF books, previous semesters works, course guidelines, and all courses videos.

5. Google Internet Search: Google is the fastest source to find some bits of initial information on some specific topic.

3.2 Literature Types and Time Period

The main source of literature was from scientific publisher organizations such as IEEE and ACM. However, to broaden the sources of knowledge, information from practice (e.g. programming libraries, manufacturer's references) was included in the search process. Using multivocal literature.
Within the subject of the study, the time period studied consisted of the literature from the 1960s to the present year 2017. As a limitation, research papers concentrating on parallel image processing on mobile platforms were limited to the period of years 2013-2017.

3.3 Search Process and Keywords

Literature searches were conducted using the information systems listed keywords. Since the topic categories were vast, the search process loosely followed a systematic mapping study protocol as presented by Kitchen-ham, Budgen, and Brereton (2014). The main keywords used in the search were: parallel computing, parallel programming, image processing, signal processing, graphics processing unit, GPU, GPGPU, mobile systems, personal mobile devices. Appropriate available Boolean operators (AND, NOT) were often used in search engines to narrow down the results.

Chapter 4

Parallel Image Processing

Parallel computing refers to the process of breaking down larger problems into smaller, independent, often similar parts that can be executed simultaneously by multiple processors communicating via shared memory, the results of which are combined upon completion as part of an overall algorithm. The primary goal of parallel computing is to increase available computation power for faster application processing and problem solving.
The result of the study will accurately determine how parallel computing are implanting in smartphones devices.

4.1 Levels of Parallel Programming Models

The number of various parallel programming models developed through the last forty decades is extensive. The existing models can be collapsed into a coarse grouping as displayed in Figure. In model (a), the level of parallel programming abstraction is the highest, and the model hides the complexity of the parallelization at the hardware level. The model requires the least parallel programming and the gained parallelism is limited to specific hardware introduced parallelisms such as superscalar execution and SMT. Model (b) exploits parallelism at the programming library- and compiler level thus no specific parallel programming is required by the programmer. This model is typically used when programs are restructured and recompiled in pursue of additional performance by parallelism. Figure x (c) depicts a model where direct HW parallelism is hidden but its vast parallel computational functionality is provided through extensive 22 programming APIs. This model

is used e.g. in GPGPU programming especially in mobile devices. The last one, figure x (d), depicts the most powerful parallel programming model. The model emphasizes the programmer's control over massive parallel computation by exploiting efficient programming libraries and tools. Examples of such libraries include MPI, CUDA, and POSIX threading (Hwu et al, 2007)

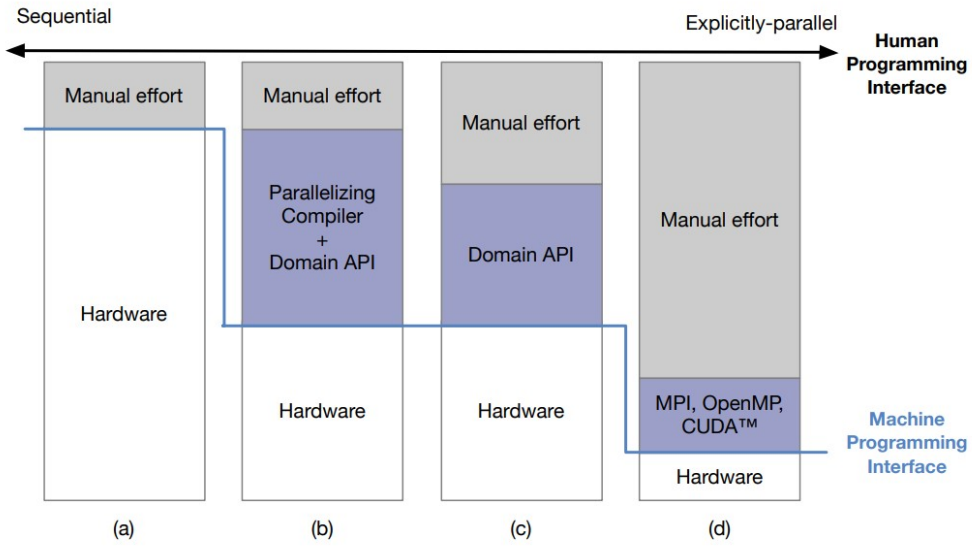

Figure: Parallel programming models grouped by parallel programming required by a developer: (a) Hardware introduced parallelism (b) Model with parallelizing compilers and domain API. (c) Model with extensive functional domain API (d) Explicit model, close to hardware with great control over parallelism. Figure adapted from Hwu et al (2007).

When choosing the right programming model for the task, the business domain and application of the parallel programming, available options, and human and computing resources can drive the choice of an appropriate programming model. For example, implementing a distributed program over a global network favors properties from another viewpoint in comparison to designing parallel algorithms for a desktop computer image processing ap-

plication. There are specific desired properties that help to assess a specific model's ability to match to a specific task or a project. McCool et al (2012) enlists the following properties important:

- Performance: The programming model should be performance-predictable to scale up to larger system sizes.
- Productivity: The model should contain sufficient tools to build performant software and monitor and debug its problems efficiently. It should provide enough productive functionality in terms of algorithms and libraries.
- Portability: To maximize re-use of existing software the model must be portable across variable hardware systems now and into the future.

4.2 Parallel Computing in Embedded and Mobile Systems

The GPU inside a mobile device can be used for a multitude of operations such as computation-intensive tasks such as game graphics handling and image processing. Regardless of the active research done in the area of parallel programming in the last decades the mixture of available programming models on mobile platforms is vast and no common, fitting high-level parallel programming solutions are available to program the new heterogeneous systems. While some research exists in integrating OpenCL programming language and its runtime environment with Android OS, yet many of the platforms, and specifically their chipset suppliers mainly support mobile parallel programming through the Open Graphics Language Embedded System (OpenGL ES) programming interface (Ross, Richie, Park, Shires & Pollock, 2014).

4.3 Parallel Programming models in PMDs

Today the most common application programming interface (API) for mobile parallel computing is Khronos Group (2016) OpenGL ES 4.1 (or higher) and many device manufacturers implement support for OpenGL ES into their

devices. Using this API programmers can write their own user programs, or kernels, using specified OpenGL ES shading language. Within these user programs, programmers typically implement the functionality that is executed in parallel tasks by the graphics hardware. (Singhal et al, 2010). Current popular PMD programming platforms such as Google Android and Apple iOS support explicit parallel programming models via threading by platform supported libraries and frameworks, and a selection of general-purpose compute- and graphics programming languages. The following sections briefly describe those additional languages and models available in the respective platforms.

4.3.1 OpenGL ES

OpenGL ES is a graphics programming standard, and it is by far the dominant graphics programming language for mobile- and embedded devices. The standard implies a shared memory programming model and defines the programming API used to manage the OpenGL ES program contexts and execution, and the OpenGL ES Shading Language used to write the shader programs.
By design, it is a data-parallel shared memory programming model. OpenGL ES (= Embedded System) is designed for low power systems such as mobile phones, handheld consoles, media devices, and vehicles. OpenGL ES allows programmers to harness the power of underlying GPU for complex 2D and 3D data computation using effective data-parallel SIMD computation. The GPU is accessed through rich interface API and programming of vertex- and fragment shaders. The shader programming language resembles C-language and is highly portable across device platforms due to its strict standard. The rendering pass of the OpenGL ES is two-phase as illustrated in Figure. First, a vertex shader is typically used to transform properties, lightning, etc. of the 3D objects before primitive assembly and rasterization to a 2D plane. Finally, the fragment shader program colors the rasterized object using e.g. configured data or a texture from memory (Apple, 2017; Munshi, Ginsburg & Shreiner, 2009).

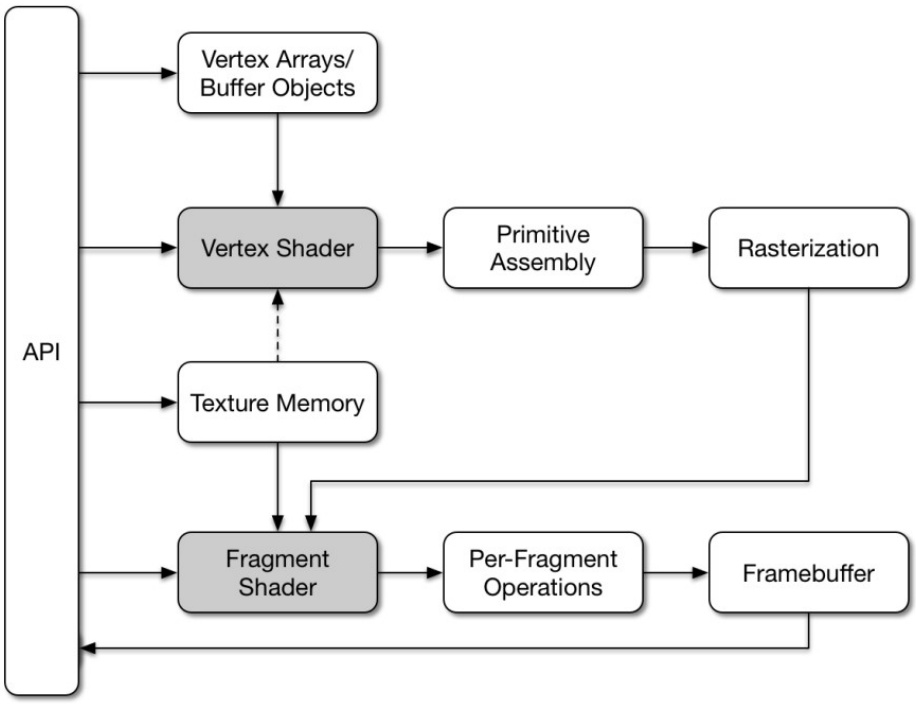

Figure: OpenGL ES 2.0 processing overview (adopted from Munshi, Ginsburg & Shreiner, 2008).

4.3.2 Apple Metal

Apple Metal1 is a low-level graphics and compute API for device GPU on Apple's macOS-, iOS- and tvOS-platforms. It claims to provide a high-level shared-memory programming model through general applicability such as available in OpenCL programming language. Similar to OpenGL ES, it provides programmable user programs (kernels) and efficient management API. The used programming language with Metal is C++. Being integrated into Apple's operating systems.

4.3.3 Google Renderscript

Google Renderscript is an Android operating system high-performance computation framework targeted for data-parallel, intensive computation tasks such as image processing, audio processing, and computer vision. Render-

script runtime environment shares and offloads the computation-intensive tasks to available device processing units (CPU cores, GPU cores). Much like in OpenCL, computation is accessed through programmable kernels that can be written in C99-compliant programming language, and through execution management API which is available in Java, C++, and Renderscript native language.
On April 2021 Google announced that with Android 12.0 they will be deprecating their RenderScript APIs. Moving forward Android developers should primarily target the Vulkan API for high-performance computing needs.

4.4 Parallel Image Processing on Mobile Systems

A number of experimental studies have been conducted on parallel CPU-GPU image processing on mobile platforms. Trending research has included topics such as exploration and performance comparison of various computing setups on mobile hardware such as programming models, parallel CPU-GPU programming setup against sequential CPU-only and -CPU-GPU configurations, and generic applicability of the new approach.

Thabet, Mahmoudi & Bedoui (2014) conducted a review on image processing on mobile devices viewing into serial- and parallel image processing on mobile devices. The parallel image processing methods identified were threading and CPU-GPU parallelism using OpenGL ES and OpenCL. A brief discussion summary of trend differences of that study and will be introduced in chapter 2, while the following sections assess other relevant developments in mobile parallel processing.

Baek, Lee & Choi (2015) studied CPU-only and parallel CPU-GPU task allocation and configurations (CPU-GPU sequential, CPU-GPU parallel) on various image processing tasks using OpenGL ES 2.0. In their experimental setup CPU and GPU were allocated to process different tasks in parallel. First, the CPU decoded an H.264 image frame followed by a format conversion for the GPU processor. GPU then continued to perform image processing tasks such as grayscale conversion and canny edge detection. In a sequential setup, GPU did not start the work before the CPU had finished

the current frame. This caused unnecessary idling on both sides of the processing ends – CPU and GPU. In a parallel setup, the authors implemented a double-buffering between CPU and GPU. When CPU had finished the conversion task, it could start the next conversion immediately after passing the buffer to GPU through API calls. Vice versa, GPU as a faster processor, did not have to wait for CPU but a new frame was available with shorter idle time than on unbuffered setups.

In another study, Baek et al (2013) compared the performance of three computation configurations, CPU-only, CPU-GPU sequential, and CPU-GPU parallel, in augmented reality (AR) feature extraction-description task using OpenGL ES 2.0. The steps required to accomplish the process were: 1) Image down-sampling 2) Grayscale conversion 3) Canny edge detection 4) Labeling 5) Contour detection 6) Rectangle check. Furthermore, in CPU-GPU tasks there was an extra image format conversion between steps 3 and 4 performed by the CPU. For CPU-GPU parallel configuration a memory buffer solution was implemented to reduce idle times for both processors. The experimental results indicated that CPU-GPU parallel processing configuration was more efficient and faster than the other two configurations in comparison on all tested image sizes (640x480, 1280x720, and 1920x1080 pixels) and the performance gain increased dramatically the more computation was required with larger image sizes.

OpenGL ES 2.0 shader programming and the split paradigm between CPU and GPU has been identified as difficult for designers and programmers. Semmo, Drschmid, Trapp, Dllner & Pasewaldt (2016) presented research where an XML-based OpenGL ES-based shader effect setup and configuration framework was designed and implemented. Using the framework programmers could create complex state-of-the-art stylization effects such as oil-painting- and water-coloring effects consisting of multiple, simpler, and sequentially chained effects. The implemented framework was deployable on multiple platforms such as Android, iOS, and WebCL capable of interoperating the XML descriptions of the OpenGL ES shaders. In the same study, a case study revealed that the user experience on the quality of the achieved effects and filters using the framework effects was perceived as good. Furthermore, the developer-experience of the framework among the students was perceived as good, easing up the burden of writing OpenGL ES shaders. Developers could better focus on designing the effects and rapid prototyp-

ing. However, for real-time purposes, the implemented framework was not yet providing good enough results.

The performance capabilities of new emerging programming models OpenCL and RenderScript were studied by Kim & Kim (2016) on Windows and Android platforms. In the study, the performance of the two programming models was compared using common image processing routines: matrix multiplication and transpose function on both platforms. The authors conclude, that Renderscript is able to utilize better the available computing cores and was more efficient in terms of computations speed. On the Windows platform, OpenCL outpaced Renderscript being almost 10-times faster. On Android devices, Renderscript was 3-to-27 times faster than OpenCL depending on the Android device. Furthermore, the authors noted, that Renderscript programming was easier and relied more on its engine's and operating system's capability to automatically share the computation load between computing cores.

In a study by Hewener & Tretbar (2015) authors demonstrated a software-based Mobile Ultrasound Plane Wave Beamforming system implemented on Apple iPad Air 2 tablet device using Apple Metal framework. The authors implemented a demonstrative software-based beamforming reconstruction solution using Apple Metal computing API. The beamforming system consisted of a compute command encoder built with Apple Metal. The system was processing high-framerate ultrafast ultrasound input data from 128 channels at a 40 MHz sampling rate. While the reconstruction and visualization could be computed with fewer frames the computation requirement was still vast. With the system, authors were able to demonstrate that a mobile software-based beamforming systems can effectively reduce development costs of ultrasound devices when the specialized hardware-based solution can be transferred to parallel computation using consumer electronics.

OpenCL is a promising portable computing standard also for mobile systems. With its more general-purpose approach to programming, harnessing it to a programmable library could even more reduce the learning curve of parallel application in programming. Cavus et al (2014) presented a study where authors implemented an OpenCL-based image processing library (TRABZ-10) on a test mobile system equipped with heterogeneous CPU+GPU architecture. The library implementation was benchmarked against reputable

OpenCV computer vision library. In the setup, OpenCV was harnessed to run its serial computing routines on the test system CPU, while the processing library was devised to run on GPU only. The implemented and tested routines included the commonly used operations in image processing; matrix calculations, filtering, morphological operations, transformations, geometric transformations, color conversions and feature detection. The results indicated, that most operations executed by the TRABZ-10 library were significantly faster than those executed by OpenCV. On some operations, speed-up gain was up to 8-times that of compared to OpenCV.

Chapter 5

Discussion

This literature review revealed important knowledge on the state of research of parallel computation on mobile platforms and PMDs. There is a good amount of studies on various, generic topics that provide important knowledge of parallel computation properties in those contexts. However, the PMD- and mobile system industry and its technology development and practices are rapidly developing areas and it seems that the science community interest has not yet reached to cover in-depth studies on how parallel computation power could be harnessed to empower user-contexts in the best possible ways in specific use-case contexts.

An interesting area of research in PMDs is the benefits of parallel computation power in user-tasks included in popular use-cases and applications, in terms of effects on user productivity and -experience. Image- and video processing applications are a popular category in PMDs. These applications often include impressive photography- and cinematic effects that require complex, novel solutions and intensive multi-pass computation on a single image frame. Investigating these complex computation setups can give more precise and more concise answers to parallel computation in PMDs. This research viewpoint, specifically on the Apple iOS platform, will be part of the author's upcoming Master's Thesis, continuing on the results of this thesis.

Another viewpoint to parallel processing in PMDs is Software Engineering (SE) practices required for parallel computing, specifically the programming models and potential assistive frameworks that could ease up the development. While there are numerous programming languages and -libraries and

tools in the server- and desktop environments addressing parallel computing challenges, there are only a few common and portable available on PMD platforms. Are the existing models of parallel programming on current PMDs sufficient, or optimal for a designer- or programmer productivity? For example, what are the experiences on the OpenGL ES platform programming model on the Apple iOS platform – is it perceived as too technical thus hindering creativity and productivity? To overcome these problems, there are some promising open source programming libraries and -frameworks available on Apple iOS-platform. These libraries try to hide the laborious and repetitive setup and state maintenance programming typically required with OpenGL ES thus releasing developers and designers to concentrate more on the application design problems.

Chapter 6

Conclusion

Parallel computation is a mainstay in Computer Science and Software Engineering. In many areas of scientific- and personal computing, the increased computation performance will come through progressions in parallelism. CPU-GPU parallelism is a well-established practice on personal- and server computing, but a relatively new phenomenon on mobile systems. For many consumers, Personal Mobile Devices (PMD) is the new Personal Computers (PC), and computer usage-paradigm shift towards PMD is emerging, yet an evident process.

Based on recent research, there is a potential to be explored in parallel computing in these device settings, to reach towards more higher-level portable solutions. That said, portable devices are targeted for media consumption including cameras and video. During recent years, while mobile systems hardware has strongly shifted towards parallelism with the introduction of heterogeneous computing architectures, the practices, and technology in mobile parallel programming models are still in search of optimal practices and improvements in current solutions. The current explicit programming models on popular PMD platforms look primitive in comparison to the state-of-the-art available in desktop and server computing. Mobile platforms surely continue to evolve towards parallelism in programming with industry and communities pushing additional and improved solutions for these platforms.

In future research, will look into constructive research of image processing and video rendering using parallel computation by implanting different processor architectures on Linux, Windows, macOS, IOS, and Android platforms. And also include the answer on how meaningful are the efforts put into parallel programming solutions available.

Chapter 7

References

Abts, D., & Felderman, B. (2012). A guided tour of data-center networking. Communications of the ACM, 55(6), 44-51. doi:10.1145/2184319.2184335

Almasi, G.M., Gottlieb, A. (1994). Highly Parallel Computing. Redwood City, CA. The Benjamin/Cummings Publishing Company, Inc.

Bauer, P., Thorpe, A., Brunet, G. (2015). The quiet revolution of numerical weather prediction. Nature, 525(7567), 47-55. doi:10.1038/nature14956

Blake, G., Dreslinski, R. G., Mudge, T., Flautner, K. (2010).Evolution of thread level parallelism in desktop applications. Paper presented at the Proceedings - International Symposium on Computer Architecture, 302-313.doi:10.1145/1815961.1816000

Cavus, M., Sumerkan, H. D., Simsek, O. S., Hassan, H., Yaglikci, A. G., Ergin, O. (2014). GPU based parallel image processing library for embedded systems. Paper presented at the VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications, 1 234-24

Culler, D. E., Singh, J. P., Gupta, A. (1999). Parallel computer architecture: a hardware/software approach. San Francisco, CA: Morgan Kaufman Publishers, Inc.

Flynn, M. J. (1972). Some computer organizations and their effectiveness. IEEE Transactions on Computers, C-21(9), 948-960. doi:10.1109/TC.1972.5009071

Diaz, J., Muoz-Caro, C., Nio, A. (2012). A survey of parallel programming models and tools in the multi and many-core era. IEEE Transactions on Parallel and Distributed Systems, 23(8), 1369-1386. doi:10.1109/TPDS.2011.30

Grama, A. (2003). Introduction to parallel computing. London: Pearson Education.

Hennessy, J. L., Patterson, D. A. (2011). Computer Architecture: A quantitative approach. Waltham, MA: Elsevier, Inc.

Internet Live Stats. Internet Live Stats. http://www.internetlivestats.com/

Kim, S. K., Kim, S. -. (2016). Comparison of OpenCL and RenderScript for mobile devices. Multimedia Tools and Applications, 75(22)

Kim, W., Voss, M. (2011). Multicore desktop programming with intel threading building blocks. IEEE Software, 28(1), 23-31. doi:10.1109/MS.2011.12

Kitchenham, B. A., Budgen, D., Pearl Brereton, O. (2011). Using mapping studies as the basis for further research - A participant-observer case study. Information and Software Technology, 53(6), 638-651. doi:10.1016/j.infsof.2010.12.011

Krisnamurthy, E.V. (1989). Parallel Processing. Singapore: Addison-Wesley Publishing Company, Inc.

Limet, S., Smari, W. W., Spalazzi, L. (2015). High-performance computing: To boldly go where no human has gone before. Concurrency Computation, 27(13), 3145-3165. doi:10.1002/cpe.346