Enabling GPU Software Developers to Optimize Their Applications - The LPGPU^2 Approach

Ben Juurlink, Jan Lucas, Nadjib Mammeri
{b.juurlink, j.lucas, mammeri}@tu-berlin.de
TU Berlin, Einsteinufer 17, Berlin Germany, 10587

Georgios Keramidas, Katerina Pontzolkova, Ignacio Aransay, Chrysa Kokkala
{g.keramidas, k.pontzolkova, i.aransay, c.kokkala}@think-silicon.com
Think Silicon, Patras Science Park, Rion Achaias Greece, 26504

Martyn Bliss
martyn.bliss@samsung.com
Samsung Electronics, Communications House, Staines UK

Andrew Richards
andrew@codeplay.com
Codeplay, Argyle House, Edinburgh Scotland

***Abstract*—Low-power GPUs have become ubiquitous, they can be found in domains ranging from wearable and mobile computing to automotive systems. With this ubiquity has come a wider range of applications exploiting low-power GPUs, placing ever increasing demands on the expected performance and power efficiency of the devices. The LPGPU^2 project is an EU-funded, Innovation Action, 30-month-project targeting to develop an analysis and visualization framework that enables GPU application developers to improve the performance and power consumption of their applications. To this end, the project follows a holistic approach. First, several applications (use cases) are being developed for or ported to low-power GPUs. These applications will be optimized using the tooling framework in the last phase of the project. In addition, power measurement devices and power models are devised that are 10x more accurate than the state of the art. The ultimate goal of the project is to promote open vendor-neutral standards via the Khronos group. This paper briefly reports on the achievements made in the first phase of the project (till month 18) and focuses on the progress made in applications; in power measurement, estimation, and modelling; and in the analysis and visualization tool suite.**

***Index Terms*—GPUs, Low Power, Embedded Computing, Power Modelling, Performance Counters, Microbenchmarks, Visualization, API Interposer, Data Collection**

I. Introduction

Consumers today expect to be able to carry around a supercomputer that has detailed graphical displays, is easy to use and lasts for several days on a single battery charge. Not only that, users also expect their devices to observe, hear and understand the world around them. Such devices range from smartphones that can understand human spoken requests all the way to self-driving cars. Delivering such capabilities requires very power efficient Graphics Processing Units (GPUs).

The strict power limitation of battery-powered devices means that these demands cannot be met by hardware improvements alone and the software must better exploit the available resources. Unfortunately, GPU application developers are hindered when creating low-power GPU software by the quality of current performance and power analysis tools. To give an example, we are currently evaluating the performance of the Vulkan API [1] and comparing it to that of CUDA [2], OpenCL [3] and OpenGL [4]. Vulkan is a lightweight API that provides more control to the programmer of when and where work happens through queues and command buffers, and is therefore expected to have a low API overhead and probably perform better compared to other frameworks. As a first step, we have developed CUDA, OpenCL, OpenGL and Vulkan versions of a simple vector addition kernel. The execution time results on two platforms AMD RX560 and NVIDIA GTX1050Ti are shown in Figure 1. Contrary to our expectations, the OpenCL and CUDA versions performed better than the Vulkan implementation on both platforms. To investigate this, we needed to have access to performance analysis tools on both platforms. However, currently there are no performance analysis tools supporting Vulkan on either platform. We had to resort to analyzing the generated assembly code in order to explain the differences in performance. It is worth mentioning that this is even worse on mobile platforms. There is a lack of performance analysis tools and often access to disassemblers and such low level tools is not provided. This is clearly not a viable solution, not just because not all GPU programmers are hardware experts but it also hinders their efficiency and productivity.

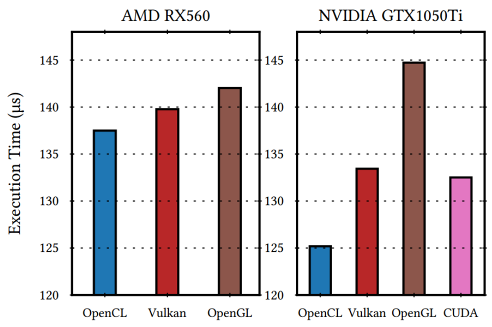

Fig. 1. Vector addition kernel results

To address this problem, the LPGPU^2 consortium has come together to work on aiding developers in creating software

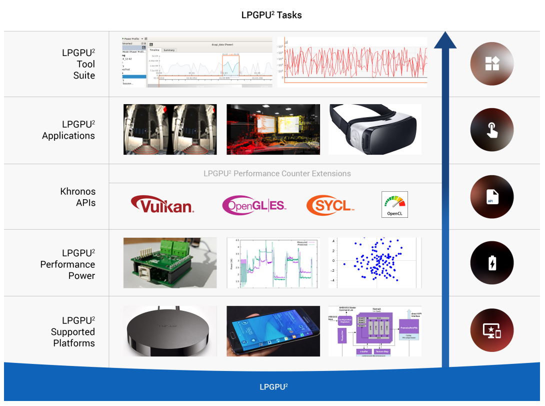

Fig. 2. General diagram of the LPGPU^2 framework

for low-power GPUs. The overall objective of the LPGPU^2 project is to develop a performance and power visualization and analysis tool suite enabling GPU software developers to analyze and optimize their applications. The framework or tool suite is based on AMD's CodeXL [5] and will support multiple open APIs such as OpenCL, Vulkan, OpenGL and SYCL [6] as well as multiple GPU platforms. To better visualise the framework, Figure 2 depicts a general overview of the different components addressed by the LPGPU^2 project. In addition, LPGPU^2 has the following objectives.

- To evaluate the tool suite on applications using the aforementioned APIs on low-power GPUs. Some of these applications have to be developed for or ported to low-power GPUs.
- To optimize these applications using the LPGPU^2 tool suite.
- To develop power models and power measurement devices that will aid in estimating power consumption as well as performance.
- To devise and promote open standards maximising the project's impact. The LPGPU^2 consortium promotes open and vendor-neutral performance and power counters standards via the Khronos group.

The LPGPU^2 consortium consists of one university (Berlin University of Technology, TUB) and four companies (Think Silicon, Codeplay, Samsung, and Spin Digital). TUB has leading experts in power measurement, estimation, and modeling as well as GPU architecture. TUB also acts as the coordinator of the project. The companies in the consortium are world leaders in power-efficient GPU design (Think Silicon), GPU and compute tools (Codeplay), graphics standards and applications (Samsung), and video codecs as well as media players (Spin Digital).

This paper is organized as follows. In Section II we describe the progress made in low-power GPU applications. Section III explains how we measure, estimate, and model the power consumption of embedded low-power GPUs. Section IV briefly describes the tool suite, which will mainly be developed in the second half of the project. Finally, Section V draws some conclusions and describes the work that is currently being carried out.

II. Applications

As part of LPGPU^2 several applications are being developed and ported to low-power GPU platforms. These applications combine graphics and compute parts, and require the use of existing and emerging APIs such as OpenGL ES^{TM} [4], OpenCL^{TM} [3], [7], and Vulkan^{TM} [1], in order to deliver the required performance and functionality. These applications will be used as benchmarks for the power and performance analysis tool being developed as part of the project. Figure 2 illustrates the role of the applications in LPGPU^2 .

The applications under consideration are important commercially for individual partners in the area of their core businesses:

- Spin Digital develops video codecs for the next generation of ultra high quality media. In LPGPU^2 , Spin Digital is developing a media player based on the H.265 video codec. The main contribution has been the development of a high performance and high quality multi-API video rendering engine. When combined together with the H.265 video decoder, the new video rendering engine allows the creation of state-of-the art media playback applications in areas such as UHD TV, Virtual Reality (VR), and large screen display.
- Samsung graphics team in the UK is responsible for Android mobile graphics in their platforms. Their applications are related to Virtual Reality (VR), Augmented Reality (AR), and font rendering. Any output from this work will help in improving Samsungs' mobile graphics platform where their mobile phones could be used for VR using GearVR, their mobile camera can used for AR and their mobile phone screen could be used for displaying text using font rendering.
- To realize the possibilities of computational photography, applications must access the hardware at low level. This would allow to control individual ISP blocks, as well as send and receive detailed information from the hardware, including exposure settings, flash, focus point, timestamps, and raw image sensor data. Using the GPU for post-camera processing is a promising direction since the bulk of the memory traffic can be eliminated. Think Silicon has built three different implementations of a set of ISP algorithms (in C, in NEMA|GFX, and in Vulkan). An ISP demonstrator based on NEMA|GFX [8], a proprietary low-level graphics API of Think Silicon, was developed and presented at industrial exhibitions with the goal of exploiting commercial opportunities of executing ISP and post-camera processing algorithms on embedded GPUs.
- TensorFlow^{TM} [9], is an artificial intelligence framework that can be used for executing machine learning algorithms. While a computation expressed using TensorFlow can be executed across heterogeneous systems, support has so far been limited to NVIDIA$^{®}$ processors using

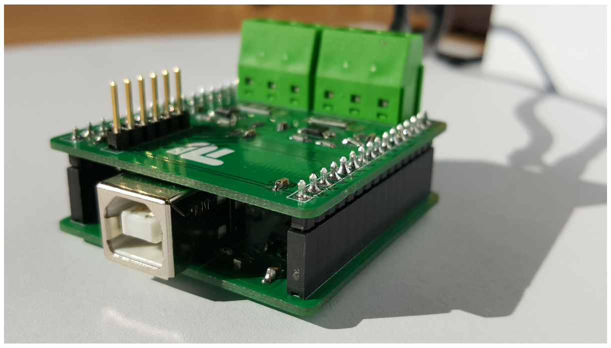

Fig. 3. LPGPU^2 power measurement testbed

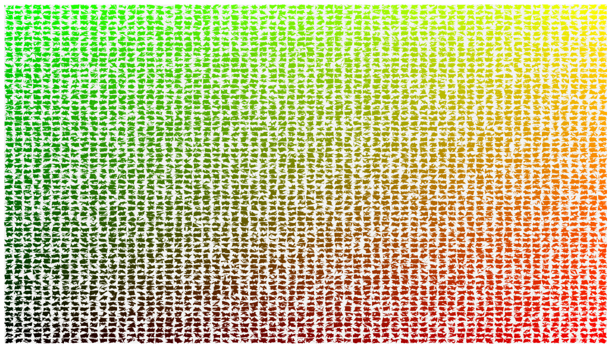

Fig. 4. GLBench microbenchmark

CUDA® [2]. In order to enable developers to access a wider range of processors, we are working to bring support for OpenCL devices to the TensorFlow framework using SYCL. OpenCL is a framework for writing programs that execute across heterogeneous platforms, and SYCL is a royalty-free, cross-platform C++ abstraction layer that builds on the underlying concepts, portability and efficiency of OpenCL, while adding the ease-of-use and flexibility of modern C++14. Parallelization is also important from a processing and power management perspective. Since tensors are n-dimensional vectors, having access to parallelization of the TensorFlow code is important not just at the training stage, but also when performing inference on new data sets.

III. Power Measurement, Estimation & Modeling

Another large area of the project is the measurement, estimation and modeling of the power consumption of embedded, low power GPUs, as well as the SoCs that employ these GPUs. The derived power models will be integrated in the LPGPU^2 toolchain and will act as a valuable mean to locate the most power consuming parts of the executed applications.

A. LPGPU2 Power Measurement Testbed

In order to verify and calibrate our power models, the LPGPU^2 project also developed its own power measurement testbed for embedded SoCs. The first LPGPU project also used a power measurement testbed, but relied on a CotS USB DAQ and custom signal processing circuits. While this setup worked, it was cumbersome to use due to several issues: closed source drivers prevented the use of regular up to date Linux distributions, sample rate and resolution could be improved and the wiring between custom signal conditioning was prone to loose contacts. The LPGPU^2 power measurement testbed, shown in Figure 3, improves upon the old testbed: the complete circuit, firmware and host software was designed as part of the project, sample rate and resolution was improved and new software was written to support measurements of embedded Android based platforms.

B. Data-Dependent Power Consumption

The switching activity of CMOS circuits depends on the processed data. As CMOS dynamic power depends on the switching activity, this also influences the energy consumption. Our initial experiments showed a large influence of data values on the energy consumption of commercial GPUs. In one experiment the power consumption increased by 65%, when changing the processed data without changing the number of executed instructions or memory accesses patterns. Conventional architectural power models for GPUs do not consider the influence of data on power. The LPGPU^2 project developed a novel power model for GPU ALUs, that takes the data-dependence into account [10]. By considering data dependent metrics such as hamming distances, the power consumption of the data path could be predicted with 85.6% smaller errors than previous models.

C. Microbenchmarks

A microbenchmarking suite was developed to automatically characterize the power and performance characteristics of each platform. This suite consists of several different benchmarking applications. CPUBench is responsible for stressing the CPU under various load conditions. CLBench is used to stress the compute part of the GPU with OpenCL test kernels. GLBench is a Android applications that employs OpenGL ES 3.1 to stress the vertex and pixel shaders of the GPU. A screenshot from GLBench stressing the vertex shaders is shown in Figure 4. Each applications is able to execute several different microbenchmarks. CLBench as well as GLBench can be configured via flexible configuration scripts. New microbenchmarks can be added by simply writing a new script with the kernels and options describing how these kernels should be executed. These microbenchmarks are executed on the device to discover the influence of the different performance counters on the power consumption.

D. Android Power Model

One result of the project is the development of a flexible power model for Android based systems. The power model collects performance counters suitable for power estimation. The microbenchmarks described in the previous section are used to calibrate the power model. After the calibration has been performed for a platform, power consumption can be predicted from the performance counters without requiring extra measurement hardware. An overview of this setup is

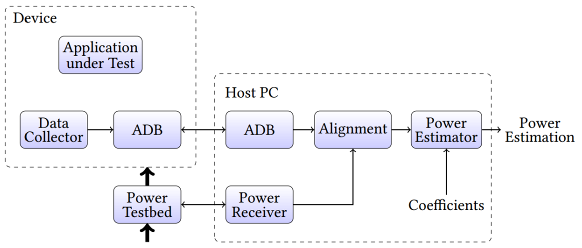

Fig. 5. Overview of the power model and evaluation platform

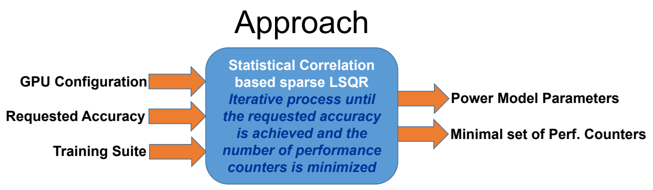
Approach
GPU Configuration
Requested Accuracy
Training Suite
Statistical Correlation based sparse LSQR
Iterative process until the requested accuracy is achieved and the number of performance counters is minimized
Power Model Parameters
Minimal set of Perf. Counters

Fig. 6. Power model methodology verified at Nema GPUs

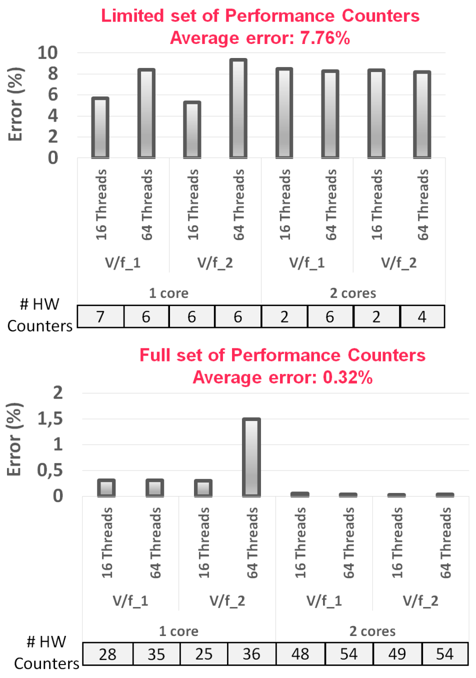

Fig. 7. Accuracy of the power model when a restricted set (top graph) and the full set of the monitoring counters (bottom graph) is used

shown in Figure 5. Android Debug Bridge (ADB) is used as communication channel between a lightweight performance counter collection software running on the device and the power model running on a host PC. A special alignment procedure is used to ensure that performance counter data and measured power data are synchronized.

E. GPU Power Model Verified at Netlist-level

While the previous power model relied on microbenchmarks and public architectural information, we also wanted to determine how accurate power estimations can be achieved, when full knowledge of the architecture is available. Most importantly, how the power estimations can be improved when the performance monitoring hardware is built in tandem with the power model. To this end, a fully parameterized power model is created and a methodology for selecting the suitable set of hardware performance counters is developed. The proposed methodology attempts to reduce the set of required hardware counters for a given upper bound or maximum error in estimating the power consumption of embedded GPUs. The outcome of this activity is validated in Think Silicon GPUs and in particular in 3D multi-threaded and multi-core NEMA—t GPU [11] assuming various configurations (altering the number of GPU hardware threads, the number of cores), process technologies (TMSC@40nm and FDSOI@25nm), and operating configurations (two voltage/frequency levels).

The calibration and validation of the power model is performed based on fine-grain, netlist-level, power measurements using the IC power compiler tools of Synopsis. Figure 6 depicts a high level overview of the derived methodology. The inputs of the power model are: i) the GPU configuration, ii) the requested accuracy of the power model, and iii) the training suite (the testbench suite of Think Silicon is used for this purpose). The outputs of the model are: i) the model parameters (i.e., the weight factors of a parameterized equation which takes also as input the performance counter values), ii) the minimal set of required performance counters for the input accuracy. The heart of the power model is a statistical correlation model based on a least square linear regression (LSQR) algorithm [12].

The development of the power model is divided into two main phases: the training phase and the validation phase. During the training phase, the initial power model is created using the testbenches and the netlist-level power measurements as input. The validation phase includes an iterative phase in which the number of selected hardware performance counters is progressively reduced based on a try-and-error algorithm until the predefined, input accuracy is achieved. The experimental results of the validation phase are illustrated in Figure 7 in which 13 ISP/post-camera processing computation kernels are used (developed also as part of the LPGPU^2 project).

More specifically, Figure 7 shows the reported error assuming two scenarios. In the first scenario (top graph), the target is to minimize the number of monitoring counters (thus the HW and run-time overheads) by setting an upper bound in the power estimations to 25% (maximum error across all ISP applications). In the second scenario, the goal is to minimize the output error without imposing any restrictions in the number of monitoring counters. The grey boxes in the bottom of both graphs in Figure 7 illustrate the number of counters utilized in all studied GPU configurations (juxtaposed on the x-axis). As the graphs illustrate, moving from the restricted set of counters to the full set, the average error is significantly

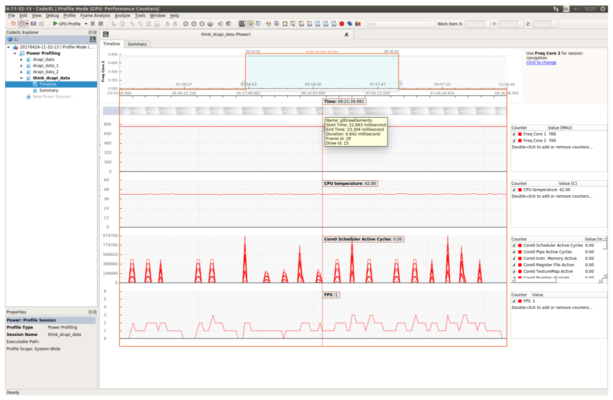

Fig. 8. CodeXL displaying data captured from Think Silicon's Nema GPU

decreased (from 7.76% to 0.32%).

IV. Tool Suite

The tool suite provides the infrastructure and mechanisms to allow the visualization of collected API traces along with hardware counter data (performance & power). Figure 8 shows a representative example of the collected data that is visualized in the project's tool. Figure 9 depicts a overview of the data processing architecture of the system. The main components of the toolsuite are: CodeXL visualization part, Collector, Interposer, and DC API.

A. CodeXL

As part of LPGPU^2 , the CodeXL toolsuite is heavily extended to support the needs of the project as well as to provide the required visualization and processing capabilities. More specifically the database schema is modified to offer additional types of data, simultaneous timeline and counter views, and visualisation of OpenGL ES tracing. CodeXL is also extended to support the visualization of CPU (along with the GPU) performance counters. Due to the flexible data processing architecture and the use of standard CodeXL databases as the de facto data storage format, we are able to collect and visualize data from devices that are not supported natively by CodeXL. In the case of LPGPU^2 this allows support for Android based mobile devices such as phones.

B. DC API (Data Collection API)

DC API is a graphics and hardware counter vendor neutral API that developed within the project. It is designed to to be located below the application API (e.g., OpenGL ES, EGL, Vulkan, NEMA—GFX etc.) and access the performance and power counters in a standardized fashion. DC API interacts with the Shim module (see below) that enables it to read data for each enabled counter and then store it for visualization in a later phase. Between the counter sampling, it remains dormant until the next sample period.

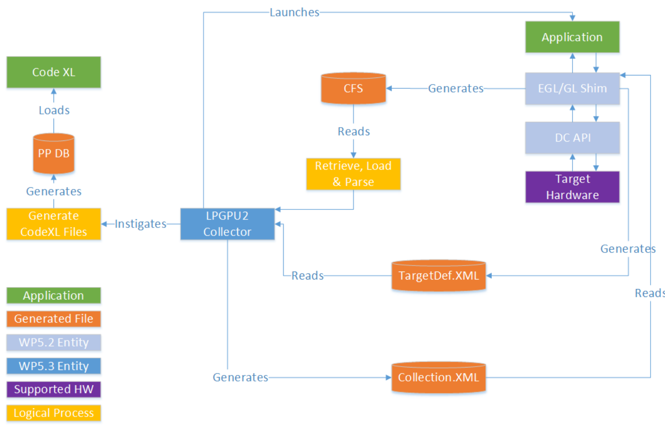

Fig. 9. Data processing architecture

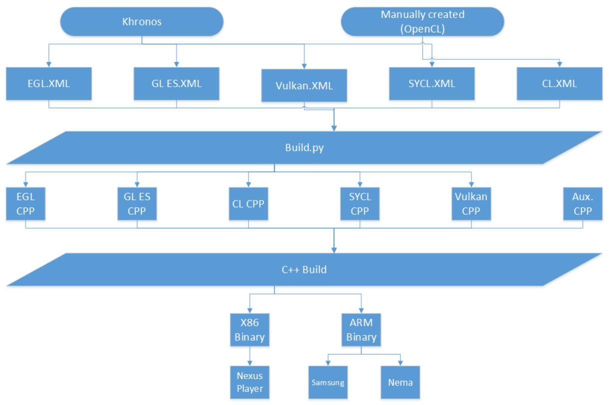

Fig. 10. Generation of the Shim from XML API description files

C. Shim

The Shim is a C++ based intermediate layer that in inserted between the application that we wish to collect data for and the underlying graphics API. This module is generated from the Khronos API definitions and it is designed to perform various actions such as: tracing function names, execution times, monitoring state etc. (Figure 10).
The Shim functionality can be used in two ways that largely depend on the version of the target Android platform and other system parameters such as root or unrooted access to the device, various permissions issues, vendor specific Android customizations etc. These are:

- Symlinked global interposition: a global shim that tracks the functions that are called system-wise
- Combined APK build: under this operation, the necessary binary components are bundled into the APK of the application that we wish to collect data for, while the application is also modified to explicitly load the Shim libraries.

The API definitions that are currently supported by our framework are provided in Table I.

API	Version	Notes
EGL	1.5	EGL implementation is quite complex due to the existence of the eglGetProcAddress function which is used to provide the address of other functions within the binary implementation. To overcome this, it is required to re-routed around the Shim implementation to avoid infinite recursion.
OpenGL ES	3.2	GL implementation is complex due to the existence of the glGetProcAddress function which is used to provide the address of other functions within the binary implementation. To overcome this, it is required to re-routed around the Shim implementation to avoid infinite recursion.
OpenCL	1.1	An XML definition file is created manually for this API from the public specifications.
Vulkan	1.0	Additional support is needed when supporting Vulkan to perform layer registration which is needed to interact correctly with the Vulkan loader. This is not required for the other APIs.
Nema	1.0	An XML definition is created manually for NEMA—GFX API of Think Silicon. In addition, extra tracing information of NEMA—GFX API calls, as performed by the SurfaceFlinger of Android, is inserted into the database.

TABLE I
APIS AND VERSIONS SUPPORTED BY THE TOOL

D. Collector

The collector is the main module written in python and its main responsibility is to coordinate all the operations and data collections. The roles of the collector are:

- Start and stop data collection
- Consume raw data files generated from data collection and generate the CodeXL database
- Manage auxiliary collection files

E. Intermediate File Size Characterization

In order to minimize the workload on the target device, it is crucial to identify the optimal size for the file chunks that are created on the target during the data collection process. As a result, a number of stopwatch tests were performed to determine the ideal file chunks size. The results of our experiments can be seen in Table II. In these experiments both the file chunks size and the generation frequency are altered.

V. CONCLUSIONS AND FUTURE WORK

The LPGPU^2 project aims to enable GPU programmers with a framework that allows them to locate and improve the most power consuming and performance critical parts of their graphics as well as GPGPU embedded applications. This paper has outlined how the project addresses these problems from different perspectives; from power models and tool development to applications and algorithm optimizations. The LPGPU^2 consortium aims to deliver to the open source community a toolchain encompassing a GUI interface, APIs, interposer, and data collectors. The toolchain will equip GPU application developers with the required infrastructure and mechanisms allowing the visualization and interaction with the collected data (API traces along with hardware counters) to enable them to improve the performance and power consumption of their applications. At the lower level, highly accurate counter-based power models are being developed and calibrated for various low power platforms. The success of the project is ensured by a wide range of applications that are being developed or extended as part of the project and will be optimized by the LPGPU^2 framework. These applications are: font rendering, augmented reality, virtual reality, ISP algorithms, deep learning using the Tensorflow framework as well as an H.265 codec and a new high-performance video rendering engine.

Approximate file size	Target Generation (Hz)	Target Generation (KB/s)	Host Retrieval (Hz)—	Host Retrieval (KB/s)
1KB	31	31	12.5	12.5
2KB	15	30	12.5	25
5KB	6	30	6	30
10KB	2.8	28	2.8	28
25KB	1.16	29	1.16	29
50KB	0.6	30	0.6	30
100KB	0.3	30	0.3	30

TABLE II
DATA GENERATION AND RETRIEVAL

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 688759.

REFERENCES

[1] Khronos, "Vulkan Specification," 2016. [Online]. Available: https://www.khronos.org/registry/vulkan/specs/1.0/html/vkspec.html
[2] NVIDIA, "NVIDIA CUDA C Programming Guide v7.5," 2015. [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
[3] Khronos, "OpenCL Specification," 2009. [Online]. Available: https://www.khronos.org/opencl/
[4] J. Kessenich, D. Baldwin, and R. Rost, "The OpenGL ® Shading Language," *Language*, vol. 1, pp. 1–29, 2010. [Online]. Available: http://www.opengl.org/documentation/specs/
[5] AMD, "CodeXL: A comprehensive tool suite that enables developers to harness the benefits of cpus, gpus and apus," 2016. [Online]. Available: https://github.com/GPUOpen-Tools/CodeXL
[6] Khronos, "SYCL Specification," 2015. [Online]. Available: https://www.khronos.org/registry/SYCL/specs/sycl-1.2.pdf
[7] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science and Engineering*, vol. 12, no. 3, pp. 66–72, 2010.
[8] T. Silicon, "Nema-gfx API," 2016. [Online]. Available: http://think-silicon.com/products/software/nemagfx-api/
[9] M. e. a. Abadi, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/
[10] J. Lucas and B. Juurlink, "ALUPower: Data Dependent Power Consumption in GPUs," in *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sept 2016, pp. 95–104.
[11] T. Silicon, "3D Nema-t GPU," 2016. [Online]. Available: http://think-silicon.com/products/hardware/nema-tiny/
[12] P. Rousseeuw, "Least Median of Squares Regression," *Journal of the American Statistical Association*, vol. 79, no. 388, pp. 871–880, 1984.