**EMULATED GPGPU KERNELS**

**A STUDY INTO PERFORMANCE**

Eric Nilsson

Blekinge Institute of Technology

Karlskrona

Sweden

E-mail: EricNNilsson@gmail.com

**KEYWORDS**

GPGPU, GPU, Simulation, Emulation, DirectCompute, WARP.

**ABSTRACT**

Emulation of *GPU*-kernels on the *CPU* is common for the purposes of debugging or profiling, but also in situations where hardware implementations are unavailable, such as server-side applications (see Microsoft’s reference on *WARP* (Microsoft, 2013)). When performing such emulation, one may experience severe performance loss due to the architectural differences in-between the two target platforms. This study examines the performance of such emulated kernels, by analyzing three driver types from the *DirectX*-framework using *DirectCompute*, amongst which is the high speed software rasterizer Microsoft *WARP*. The performance of *WARP* is compared to that of traditional *GPU* hardware acceleration and the standard driver for software rasterization - the Reference Device Driver.

Results demonstrate a major performance boost when compared to that of software rasterization using the Reference Device Driver. Thus, this study reasons that performance losses traditionally obstructing *GPU*-kernel emulation may be amended sufficiently by technologies, such as Microsoft *WARP*, to the degree that such emulation can be considered viable for use in retail applications.

As such, the hypothesis suggested in this document may be summarized as follows:

The performance-impaired emulation of a GPU-kernel may be amended by using high speed software rasterizers, such as Microsoft WARP.

**INTRODUCTION**

When developing *GPU*-kernels, relatively small modifications in code may induce large deviations in performance due to massively parallelized instruction sets and architectural differences in-between on-chip hardware (See Performance Considerations by Kirk & Hwu (Kirk & Hwu, 2010; ch. 6) for an analyis on the volatility of *GPGPU*-performance). The increased utilization of complex *GPU*-kernels has brought forth the need of more extensive debugging- and profiling-possibilities involving access of data that may be hard to retrieve from hardware. Therefore, it may be desirable for the developer to be able to view the data being computed on the graphics card - a possibility often limited in terms of *GPGPU*-technologies, possibly due to architectural differences in-between chip manufacturers.

A preferred solution to this problem has been to emulate such *GPU*-kernels on the *CPU* (see Microsoft's reference on *DirectX* Driver Types (Microsoft, 2013)), such as described by Kerr et al. (Hwu, 2011; p. 416-419) concerning the implementation of the *GPU Ocelot* compilation framework (see *GPU Ocelot* (GPU Ocelot, 2013)), often in exchange for substantial performance-losses. Other resons to emulate *GPU*-kernels may concern pre-silicon development - that is, development for hardware not yet existant, when hardware is busy, or otherwize unavailable (see Microsoft’s reference on *WARP* (Microsoft, 2013)). This study comprises an investigation into the performance of software emulation of hardware accelerated *GPGPU*-kernels, by the means of analyzing several software rasterizers.

Furthermore, this material concerns inquiry into the *DirectCompute*-framework on the *Windows*-platform, analyzing the performance of a *GPGPU*-kernel on-chip, using the *DirectX* standard software rasterizer, and utilizing the *DirectX* 11.1-addition Microsoft *WARP*-technology (*Windows Advanced Rasterization Platform*) - which promises high-speed software emulation (see Microsoft’s reference on *WARP* (Microsoft, 2013)).

This article concludes that the performance losses inflicted by such emulation may be reduced enough for that emulation to be considered feasable for retail use, as originally proposed by Microsoft (see Microsoft’s reference on *WARP* (Microsoft, 2013)). Thus, this study proposes use of Microsoft *WARP*-technology in industry rasterization if graphics hardware is unavailable, not sufficient, or busy.

As such, the study concerns the fields of simulation, emulation and *GPU*-technologies - respectively, with the purpose of facilitating debugging and profiling of *GPU*-kernels, whilst maintaining acceptable performance.

The remainder of this document presents the method and process to acquire the data used, the technologies using which it has been acquired, the results in and of their own, the conclusions based off the results and finally; the author's personal reflections surrounding future work in the area along with propositions of further study.

**CONTRIBUTION**

**Object of Study**

Due to the nature of the experiment, it is important that the Object of Study is measurable and deterministic. For the purpose of this study, the data computed corresponds to a square matrix multiplication, since such an operation is highly parallelizable, as demonstrated by Kirk & Hwu (Kirk & Hwu, 2010; ch. 3). Additionally, verifying the result of such an operation is trivial, and potential losses in computational precision may be computed from the expected data-type. The operation was therefore considered suitable for the purpose of this experiment.

The dimension *200x200* was selected for the respective matrices as it is big enough to execute efficiently on the *GPU*, but yet reasonably sized to keep measurements collected by the means of emulation comparable to the hardware accelerated reference case.

Throughout this material, the compiled data will be referred to as *AB=C*, and thus make out the Object of this experiment.

**Method of Study**

In order to establish the Object of Study, *AB=C*, the experiment utilizes *GPGPU*-kernels using which the result is calculated, in it’s entirety, in aforementioned kernel using some target Subject (see Section Subjects of Study). The experiment is devised of the following approximate steps in order to compile the Object of Study *AB=C*:

1. Randomize two matrices *A & B* using desired data-type.
2. Establish the product-matrix *AB=Ref*. The resulting matrix will be used as a reference matrix to verify the final result.
3. Start a synchronized high-precision timer.
4. Dispatch *GPU*-kernel calculating the product matrix *AB=C*.
5. Stop the timer once the kernel has finished execution.
6. Establish possible deviation in-between resulting matrix *C* and previously established matrix *Ref*.

The process described above make out the Method of this study.

**Subjects of Study**

The *GPU*-kernels described in Section Method of Study are comprised of *HLSL*-syntax and are compiled & executed using Microsoft *DirectCompute*. These kernels are run using three types of acceleration technologies with varying strategies of *GPU*-kernel emulation. These are comprised of the following:

*Hardware-Acceleration (GPU)*

The execution of a *DirectCompute*-kernel on a graphics card. This is the common case, and involves no emulation of the kernel. Thus, this case will act as a reference for the emulated Subjects.

During hardware acceleration on the *GPU*, one may expect high performance.

*Software Rasterization (CPU)*

The emulated execution of a *DirectCompute*-kernel using the *DirectX* Reference Device Driver. The Reference Driver was developed for the purpose of testing and debugging, and is - although it does support some *CPU*-optimizations - not intended to be used in retail applications (see Microsoft’s reference on *DirectX* Driver Types (Microsoft, 2013)).

As the Reference Driver is designed for the purpose of accuracy, rather than speed, one may expect poor performance.

*Windows Advanced Rasterization Platform (CPU)*

The emulated execution of a *DirectCompute*-kernel using a special software rasterizer devised by Microsoft in their latest revision of the *DirectX*-framework. The driver is based off the *DirectX* Reference Driver and uses thread pooling to distribute tasks efficiently on the *CPU*, along with grouping execution in batches for optimum performance. Microsoft describes *WARP* as a high-performance software rasterizer, and recommends using the driver for retail applications, such as casual games (see Microsoft’s reference on *WARP* (Microsoft, 2013)).

As there are, as of yet, few studies performed on Microsoft *WARP*; expectations are unclear, but the driver is expected to perform better than standard software rasterization.

Theese three *DirectX* driver-types make out the Subjects of this study.

**Kernels**

In addition to the Subject drivers mentioned in Section Subjects of Study, two kernels with varying level of optimization are examined. The kernels have been implemented in accordance to the *CUDA*-kernels as described by Kirk & Hwu (Kirk & Hwu, 2010; p. 67, p. 87). These kernels are presented below.

*Matrix mult. w. Thread Blocks*

A kernel producing *AB=C* from two given matrices, writing back *C* for further analysis. The kernel is executed with one thread for each element in the square matrices, and likewize each produce a lone element of the resulting matrix.

Execution is performed in blocks of *16x16* threads since this was the block dimension, out of samples *8*, *16*, and *32*, that performed optimally whilst hardware accelerated on the system described in Section Equipment.

This kernel will be referred to as the Basic Kernel throughout this material.

*Matrix mult. w. Thread Blocks & Shared Memory*

Similar to the previous kernel, but further optimized to utilize *shared memory* in order to reduce time-consuming reading of *global memory*, as presented by Kirk & Hwu (Kirk & Hwu, 2010; p. 77-93).

Stratton et al. (Stratton, Stone, & Hwu, 2008; p. 1-3) instructs that the *CUDA* *GPGPU*-model may be applied onto multicore *CPU*s, including locality-wize execution of logical thread-blocks (all threads in a block limited to a single core), with the utilization of *local*- and *shared*-memory approximately corresponding to a core's *L1*-cache. Hence, the kernel is presented as a scenario due to the preconditions of *WARP* - stating that a kernel optimized for *GPU*-execution is likewize optimized for execution with *WARP* (see Microsoft’s reference on *WARP* (Microsoft, 2013)). Thus, we investigate a more optimized kernel to see whether or not this behaviour may be replicated in the experiment.

This kernel will be referred to as the Tiled Kernel throughout this material.

Furthermore, aforementioned kernels both support integer- and floating-point precision. These kernels are attached, in their entirety, under Section APPENDIX.

**Tools**

The experiment process has been subdivided into three major components, all of which use Microsoft *Visual Studio 2012* for compilation. These are presented below.

*matrixgen*

Denotes a utility developed to generate matrices of different dimensions and data-types. Furthermore, *matrixgen* compiles the reference matrix *Ref* used when comparing the result returned from the *DirectCompute*-dispatch described in the next paragraph.

*matrixgen* is written in C++ and utilizes C++ AMP to generate and multiply matrices *A* & *B* into product matrix *Ref*. In order to achieve random values in a C++ AMP-kernel the solution includes the random number generator-library C++ AMP RNG. As *matrixgen* utilizes Microsoft C++ AMP-technology, *Windows 7* or later is required.

*experiment*

Making out the primary component of the study, *experiment* uses *DirectCompute* to compile the product matrix *C* from the matrices *A* & *B* generated by *matrixgen*. The application outputs data surrounding the execution of said kernel, such as it's execution time in milliseconds, to an intermediate file.

*experiment* is written in C++ with it's respective *DirectCompute*-kernels written in *HLSL*-syntax. As *experiment* is developed using the *Windows 8 SDK*, *Windows 8.0* or later is required. Furthermore, *experiment* requires a *DirectX 11.0*- or *DirectX 11.1*}-compatible graphics card.

*analytics*

*analytics* is a utility developed to compose data surrounding possible precisional deviations in-between matrices *C* & *Ref*. *analytics* compiles the minimum- and maximum-deviation encountered, as well as to calculate the standard deviation of said precisional deviation. In turn, *analytics* outputs this information to an intermediate file.

*analytics* is written in C++.

These three applications are, in turn, run as subprocesses in a script specifying the various configurations and number of times to run each program. This script, written in Python, then compiles the assorted results of these applications and outputs a range of files suitably formatted for interpretation by *Gnuplot*.

The source code manufactured for the sake of this study is freely available via an online Git repository (Nilsson, 2013), along with a guide on how to compile and run the solution in order to replicate the experiment. Furthermore, the complete results collected and used throughout this study is also available for download, and may be acquired for further analysis.

**Equipment**

The results presented in this study is gathered from experiments performed on a system with the following specifications:

**CPU** Intel Q9550 Quad Core 2.83GHz

**GPU** ATI Radeon HD 5800

**OS** Windows 8.0

This system setup was selected for use, for the purpose of this study, as Microsoft claims that the *WARP* driver performs best on modern quad-core *CPU*s (see Microsoft’s reference on *WARP* (Microsoft, 2013)).

**Process of Study**

For the purpose of this study, the Object of Study - being *200x200* matrices - were randomized with numbers in-between zero and ten. The product matrix of these matrices was then computed 100 times for each configuration. In this way, the Basic Kernel was run with each Subject of Study - being Hardware Acceleration, Software Rasterization and *WARP* - respectively, likewize as with the Tiled Kernel. For each execution, data surrounding the dispatch-time of each kernel (meaning the time taken to execute corresponding kernel, regardless of program initialization.) was garnered along with precision-wize deviational data.

The process described above was then repeated for integer- and floating point-precision.

The measurements gathered from these executions make out the Results presented in this study.

**Results**

Based off the average of the collected execution times described in Section Process of Study; results indicate an improvement in the performance of the kernels when using Microsoft *WARP*, compared to the performance of the *DirectX* Reference Device Driver from which *WARP* is derived. Table 1 demonstrates a performance gain with the Hardware Accelerated Subject when utilizing *shared memory* amongst blocks; with varying results for the other Subjects - either increasing or decreasing execution time (see Table 2).

Table 1: Average execution time in milliseconds of a *200x200* integer matrix multiplication.



Table 2: Average execution time in milliseconds of a *200x200* floating point matrix multiplication.



Using both integer- and floating point-precision, the performance of *WARP* is impaired by the kernel utilizing *shared memory* according to the data presented in Tables 1 & 2. In Microsoft's guide on *WARP* (see Microsoft’s reference on *WARP* (Microsoft, 2013)), the author claims that an application, if tuned to run efficiently on hardware, will run efficiently on *WARP* - and vice versa. However, the data collected for this study rather indicates an increase in execution time of *23.9%*, independant of precision, even though the same kernel accelerates the hardware accelerated Subject by roughly *70%*. The floating point scenario of this effect is visualized in Figure 1.

Figure 1: *WARP* execution time with float-precision for the Basic- and Tiled-kernel, with visualized mean and standard deviations. Values outside of their respective standard deviations have been clipped for the sake of clarity.



Integer precision calculation showed no sign of precisonal loss whatsoever. Meanwhile, all floating point-experiments experienced an equal average loss of computational precision. However, the data collected indicates no divergence in the precisional loss of respective Subject. The average deviations in precision experienced equally with each configuration, in relation to the *Ref*-matrix descibed in Section Method of Study, are presented in Table 3.

Table 3: Average precisional deviations in floating point-operations for all Subjects.



These results are expected as *WARP* conforms to the precision requirements of the *Direct3D 10*- and *10.1*-specification (see Microsoft’s reference on *WARP* (Microsoft, 2013)). See Microsoft's documentation on Floating-point Rules (Microsoft, 2013) for more information surrounding floating point-precision in the *Direct3D*-framework.

**CONCLUSION**

Based off the results presented in Section Results, using *WARP* to emulate the kernels presented in this study has magnitudes greater performance than if one were to apply the *DirectX* Reference Device Driver in the same manner. Hence, if one were to compare the execution of *WARP* & the Reference Device Driver side-by-side, and assume the same area of application, *WARP* is superior in terms of execution time - assuming the same preconditions as those presented in this material.

However, keeping in mind the major performance improvements offered by Microsoft *WARP*, it is important to consider that the two may be appropriate for different purposes. The *DirectX* Reference Device Driver is primarily proposed by Microsoft as a debugging-/pre-silicon-development tool, whereas *WARP* is intended for use in a broader sense (see Microsoft’s reference on *WARP* (Microsoft, 2013)) - such as to render graphics for casual games - in addition to debugging and error-profiling purposes. This calls for further inquiry into what the flaws of using Microsoft *WARP* may be.

In conclusion; this study proposes, pursuant to the established performance of Microsoft *WARP*, that *WARP*-technologies are feasable for extended use in applications - for purposes other than debugging and profiling.

**Future Work**

For the sake of brevity, the author suggests complementary elaboration into double precision calculations in *DirectCompute*-kernels, with the intent of examining whether or not the Subjects detailed in Section Subjects of Study may have differentiating effects on computational precision.

Additionally, the use of more complex kernels in coagency with the *WARP*-driver should be examined in order to study the effects of more demanding emulation on the *CPU*.

**APPENDIX**

**Matrix mult. w. Blocks**

****

**Matrix mult. w. Blocks & Shared Memory**



**REFERENCES**

Hwu, W.-m. W. (2011). *GPU Computing Gems Jade Edition* (1st ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Kirk, D. B., & Hwu, W.-m. W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach* (1st ed.). San Francisco, CA, USA: {Morgan Kaufmann Publishers Inc.

Stratton, J. A., Stone, S. S., & Hwu, W.-m. W. (2008). MCUDA: An Efficient Implementation of CUDA Kernels for Multi-core CPUs. (J. N. Amaral, Ed.) *Languages and Compilers for Parallel Computing*, 16-30.

**WEB REFERENCES**

GPU Ocelot. (2013). *GPU Ocelot*. Retrieved 10 21, 2013, from http://b.gatech.edu/1a489L1

Microsoft. (2013, 11 26). *Microsoft's reference on DirectX Driver Types*. Retrieved 10 21, 2013, from MSDN: http://bit.ly/1b63NiX

Microsoft. (2013, 11 16). *Microsoft's reference on Floating-Point Rules*. Retrieved 11 16, 2013, from MSDN: http://bit.ly/1bOQnZu

Microsoft. (2013, 11 16). *Microsoft's reference on WARP.* Retrieved 10 21, 2013, from MSDN: http://bit.ly/19hiwrZ

Nilsson, E. (2013). *dv2549*. Retrieved 10 21, 2013, from Github: http://bit.ly/1c4wd20