Developing Bare-Metal GPGPU Drivers From Scratch

What prevents scientists from developing own GPGPU drivers?

Marcel Lütke Dreimann
Universität Osnabrück
Germany
mluetkedreim@uos.de

Daniel Kessener
Universität Osnabrück
Germany
dkessener@uos.de

ABSTRACT

Most of modern computers use *Graphic Processing Units (GPUs)* as an additional source of computing power. However, using GPUs in bare-metal research operating systems comes with some challenges. Existing drivers for Linux or Windows are complex and cannot be used for without much effort. Documentation of modern GPUs is often missing or incomplete and drivers are incomprehensive or closed source. This paper tries to explain what prevents scientists from creating their own GPU drivers. Additionally, it gives an overview about GPGPU driver development for GPUs from different manufacturers and shows some challenges. Nevertheless, we have ambitiously started an undertaking to develop our own driver from scratch. To some extend this was successful, but with many problems on the way.

KEYWORDS

GPGPU, bare-metal driver, GPU documentation, challenges

1 INTRODUCTION

Present research operating systems like the MxKernel are developed to run well on heterogeneous hardware [8]. The hardware platform can consist of many-core CPUs, GPUs and *Field Programmable Gate Arrays (FPGAs)*. Because research operating systems are designed to run on bare hardware, a driver for each hardware unit is required. The use of GPUs with their GPGPU functionality is especially interesting, because GPUs can achieve more computing power than traditional CPUs [9]. GPU drivers for Linux and Windows already exist and are mostly developed by the manufacturers themselfes. However, these drivers cannot be easily used in custom operating systems. The driver code is large compared to other device drivers and contains a lot of code, which is not needed for GPGPU functionality. Additionally, a custom driver allows up for more flexibility and a better integration into novel operating system architectures. Furthermore, new concepts for GPU tasks and different strategies to better exploit heterogenity could be researched. A closer look into the existing Linux drivers will be taken in Chapter 3.

Because we did not find any research driver or scientific work dealing with developing own GPU drivers, there has to be a cause that prevents science from doing so. This paper tries to identify some of the causes. Therefore, we will show that:

(1) GPUs evolve more quickly than research teams could develop drivers for the new hardware
(2) the hardware documentation of the manufacturers is not sufficient for driver development
(3) a GPGPU driver for Intel IGPs is feasible in less than 2000 lines of code, which is about 0.2% of the open source Linux driver lines of code

2 RELATED WORK

In the scientific field, no research operating system we are aware of can use GPUs natively. And most scientific works focus on GPU virtualization instead of native usage (see [5] and [11]). Other works extend already existing drivers and allow GPUs to access memory over a network connection [3]. [10] and [6] deal with more advanced integration of GPUs into Linux. Additionally, [1] analyzed the existing Linux drivers and [12] developed their own tool chain for cracking and reassembling GPU binaries. In contrast to these papers, [2] deals with a hardware implementation of a GPGPU.

FGBS '21, March 11–12, 2021, Wiesbaden, Germany

https://doi.org/10.18420/fgbs2021f-03

3 GPGPU DRIVERS ON LINUX

This chapter shows how the Linux operating system manages to use GPUs for executing GPGPU applications. We will analyze drivers for GPUs from different manufacturers and show how large their codebase is. The three manufacturers we compare are *Intel*, *AMD* and *Nvidia*. Intels GPUs are mostly integrated into Intel desktop processors and are therefore less complex and less powerful. However, Intel plans to build dedicated GPUs with more computing power, especially for the GPGPU field[1]. AMD and Nvidia build mostly dedicated GPUs with more power and complexity.

All these manufacturers have their own drivers and software packages to run GPGPU applications on Linux. However, not all of them are open source and can be used as additional documentation for a custom driver. Figure 1 shows the open source drivers and components which are required to run *OpenCL* applications on hardware. *i915*, *amdgpu* and *nouveau* are the kernel modules for Intel, AMD and Nvidia GPUs respectively, and have direct access to the hardware. They are responsible for printing the kernels graphics output to the screen and therefore also initialize the hardware. The kernel's graphics output is printed via the *Direct Render Manager* (DRM) and *Translation Table Manager* (TTM) kernel module. Running GPGPU applications is not part of the graphics driver and needs additional software. For Intel GPUs this is the *NEO* driver, which consist of three software components: The *compute* driver is the main driver for running GPGPU applications. The *Gmmlib* is responsible for memory management and the *Intel Graphics Compiler* (IGC) generates binary code from OpenCL code. AMD has a similar software package called *ROCm*. Nvidia has no open source compute driver and can only run GPGPU applications with the closed source driver. If the user wants to run a GPGPU application, the OpenCL API calls the computing driver of the chosen hardware unit, which communicates with the hardware via the kernel module.

The lines of code of these components are listed in Figure 2 and 3. These figures also show a subdivision into comments, headers and C/C++ code. The comments are measured without licensing information. Note that not all the code is necessary to understand a GPGPU driver. Graphics driver code and compiler code are less important for a custom driver, because the custom driver can load a binary image without compiling OpenCL code at runtime. The Intel driver package is small compared to AMD's driver package with a about 1.2 million lines of code. The amdgpu driver with ROCm has the largest codebase with about 33.5 million lines of code. However, these drivers contain additional parts that do not contribute to the complexity of the driver. amdgpu has about 2.1 million lines of code in the header files, where most of them only define register identifiers and bit locations of these registers. The ROCm driver contains tools and libraries such as the *llvm* project. The nouveau driver for Nvidia GPUs is the smallest driver. This can be justified by the smaller feature set the driver offers (see nouveau feature matrix[2]). An open source compute driver for Nvidia GPUs is missing. Nevertheless, we compared text segments of the driver binaries. The results suggest that the closed source Nvidia driver does have an even larger codebase than amdgpu or i915 (see table 1).

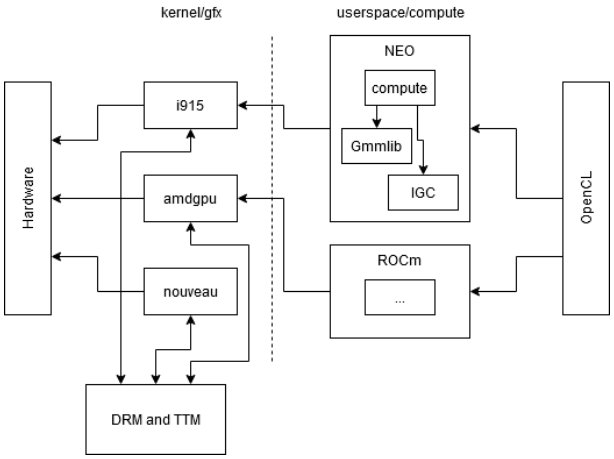

Figure 1: Linux open source drivers

kernel module	text size in byte
i915[3]	1.862.293
amdgpu[3]	4.211.273
nvidia 450.102.04	19.385.754

Table 1: Comparision of text segments of different kernel modules.

4 HARDWARE DOCUMENTATION

The documentation is the first place to look for detailed information about GPUs. Therefore this chapter gives an

[1] https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/iris-xe-max-graphics-product-brief.pdf

[2] https://nouveau.freedesktop.org/FeatureMatrix.html

[3] Ubuntu 20.10, kernel version 5.8.0-41

[4] All Numbers calculated by the cloc tool on 16.01.2021 0:00

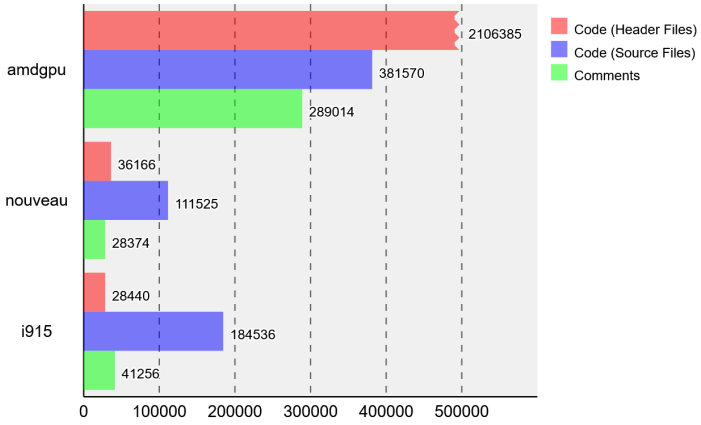

Figure 2: Source code comparison of kernel modules from AMD(amdgpu), Nvidia(nouveau) and Intel(i915)[4]

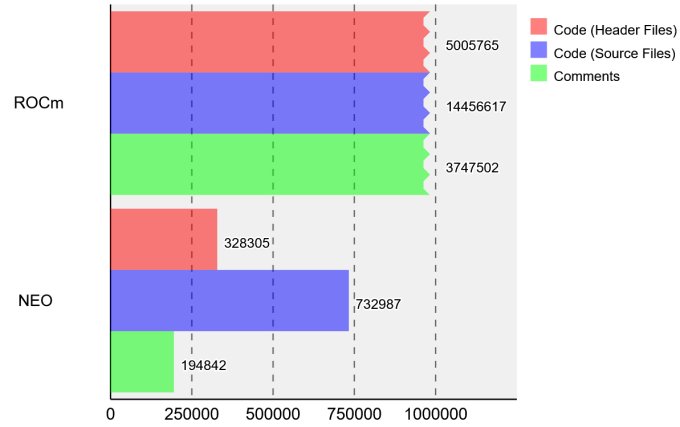

Figure 3: Source code comparison of compute drivers from AMD(ROCm) and Intel(NEO)[4]

overview of documentation for Intel, AMD and Nvidia GPUs. Additionally, some alternative resources are listed. At last, we check whether these resources are sufficient for writing a custom driver.

Intels GPU documentation can be found on the *Intel Open Source* Website[5]. There, we can find a collection of PDF files for most processor platforms. Every processor platform has its own integrated GPU. However, multiple processor platforms share the same generation of GPUs. Therefore some processor platforms only have a configuration chapter. This chapter contains information needed to access the GPU. Everything else is kept the same and can be found in the processor platform documentation, where the GPU generation was first introduced. As a whole, the most recent *Ice Lake* platform contains 14 different volumes. Each volume describes a different part of the GPU. For instance, the first volume lists all instructions, registers and structures of the GPU. Following volumes deal with the GPUs memory or different engines. The GPGPU functionality is part of the rendering engine and can be found in *Volume 9: Render Engine*. In total, the 7020 pages of documentation look quite detailed. Nevertheless, there are some issues with the documentation, as we show in Chapter 6.

AMD published their GPU documentation on their website[6]. It can be found under the heading: *Open GPU Documentation*. There, we can find some *Reference Guides* and one *Programming Guide*. The documents are dated to the year 2013 and older. Therefore, no documentation of up-to-date GPUs can be found. Reference Guides contain register definitions and the Programming Guide contains additional information of the R5xx GPU generation. As a whole, we would have to use the Programming Guide (dated to June 2010) and one of the Reference Guides to write a driver for a modern GPU. The overall concept of a GPU might not have changed a lot since 2010, but the register definitions can not be used. In total, the Programming Guide contains 288 pages, which does not have enough detail to write a custom driver. Necessary parts of the GPU such as the GPUs memory are not explained. Therefore we can say that writing a custom driver with just this information is nearly impossible. However, the website also lists an email address, where driver developers can ask for help. We used this opportunity and got useful answers, but more detailed documentation does not seem to exist. Additionally, AMD has a second website called *GPU Open*[7], but this website only offers documentation for APIs or GPU instruction sets.

Nvidias documentation can be found in one of their Git repositories[8]. There, we can find a lot of text files and C++ headers. They mostly contain a list of values and some explanation. Some text files also includes figures created by text symbols. It seems like the *Ampere*, *Turing* and *Volta* architectures are documented, but some parts of the repository are unordered, and we do not know to which architecture they belong. The Ampere architecture is Nvidias newest generation of GPUs. Documentation for computing GPUs, like the Nvidia Quadro seems to be missing. Because of the quality of the documentation and the lack of detail, it seems like creating a custom driver for Nvidia cards is a difficult task - if not entirely impossible.

[5]https://01.org/linuxgraphics/documentation

[6]https://developer.amd.com/resources/developer-guides-manuals/

[7]https://gpuopen.com/documentation/

[8]https://github.com/NVIDIA/open-gpu-doc

In addition to the official resources, some alternative resources are useful too. For example, the *Rendering-pipeline* website[9] has a good summary of all documentation available. Furthermore, the *X.Org Foundation* documentation [10] has listed some of the official, but also some additional documents.

5 CHALLENGES OF GPU DRIVER PROGRAMMING

This chapter shows some challenges of developing a bare metal GPGPU driver. Most of the problems during development are related to problems with GPU documentation, which was reviewed in Chapter 4. The challenges can be summarized as following:

(1) hardware as a black box
(2) lack of information
(3) complexity of hardware and open source drivers
(4) speed of GPU development

The first challenge comes with the hardware itself. The feedback of the hardware device is limited. It can be seen like a black box, where we do not know what it does. For instance, the GPU could wait for a specific driver code to finish or it could hang and stop working if the driver code did something wrong. Outside the black box we do not know what happened and what error may have occurred. To look into the black box some kind of hardware debugger would be required. However, we are not aware of any publicly available hardware debuggers. Furthermore, the documentation provided by the manufacturer is not sufficient for writing a driver (see Chapter 4,). Therefore, open source drivers are necessary to analyze and understand. Furthermore, dedicated GPUs specifically are complex and the open source driver code is complex too. This is the case because a GPU is nearly its own computer, plugged into a computer. The device has its own memory, processing unit with instruction set and IO ports. Even more complexity appears, when the open source driver supports multiple GPUs of different GPU generations. This leads to a lot of hardware-specific code. For example, there are multiple files in the amdgpu driver, which have the same name, but end with a different version identifier. Each file belongs to a part of the GPU driver and is intended for specific generations of AMD GPUs. In addition, the kernel modules contain a lot of code for the graphics part of the GPU. This code is not necessary for a GPGPU driver. However, it is hard to distinguish which code is needed for graphics and which code is part of the initialization and is therefore also important for the GPGPU driver. In some cases graphics code is indeed required for GPGPU applications to run, because graphics and computing is not completely separated. As seen in Chapter 3 there are no pure compute drivers. The compute drivers depend on the graphics drivers. The last and maybe biggest challenge comes with the development speed of GPUs. Especially Nvidia and AMD release multiple GPUs every year[11]. If research prototype drivers want to stay relevant, they have to support state of the art GPUs. As we saw earlier in this paragraph, driver development can be a hard task and therefore needs its time. But if GPUs are released that frequently, it is hard for scientific teams to keep up with their drivers. The amdgpu driver has about 670.000 lines of code changes in their driver every year[12]. Intels i915 driver has about 130.000 lines of code changes[12]. And these are only changes in the kernel modules. The total amount of changes with ROCm and NEO are even higher. With an estimation of 10.000 lines of code a scientist can write every year, it would need 67 scientists to keep up with the driver development for an AMD GPUs kernel module. For most scientific working groups this will not be possible to do.

6 EXAMPLE: INTEL GPGPU DRIVER (VS. AMD GPGPU DRIVER)

This chapter shows some issues by example. Additionally, we present the results of our driver projects. At first, we will look at the driver for the integrated Intel GPUs of the *Intel UHD 600* generation and at last we investigate a driver for the *AMD Polaris* generation.

Intel

The Intel driver was part of a bachelor thesis [4]. Therefore the driver was developed by one person in a time of approximately 3 months.
The first problem occurred already at the beginning of the development. The GPU is in a sleep state by default to reduce the power consumption if it is not used. One of the first things the driver has to do, is to wake the GPU up. This step is crucial for using the GPU, because without waking it up, the GPU will not react to any register changes. The documentation however does not explain this step. A trace of the i915 kernel module suggested writing some values into special GPU registers. These GPU registers are called *Force Wake Registers* and are responsible for waking up the GPU. Not only is this step not mentioned in the documentation, even the list of

[9] http://renderingpipeline.com/graphics-literature/low-level-gpu-documentation/
[10] https://www.x.org/docs/
[11] https://vintage3d.org/dbn.php
[12] Average value calculated with git diff over the years 2015 to 2021.

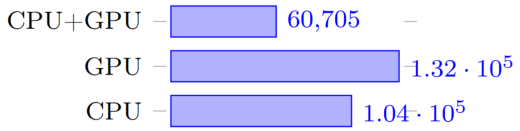

Figure 4: Example makespan for processing units in μs
Source: *[7]*

registers omits these important registers. With just the information of the documentation it would be impossible to start using the GPU. In older documentation, we found the Force Wake Registers in the register list, which suggests that the registers were either removed or forgotten in the newer documentation. There are more examples of this kind of issue, but the required information is sometimes also missing in older documentation. This shows what kind of problem can occur and how erroneous the Intel documentation is. Nevertheless, in retrospect after the AMD driver development, the Intel documentation is still the best documentation for GPUs out there.
At the end of the bachelor thesis, we managed to have a working driver[13] with less than 2000 lines of code. However, it was way slower than the Linux driver. This was fixed after the bachelor thesis. Again, it was a missing step of initialization, because another register was missing in the documentation. Now, the driver can compete with the Linux driver. [7] used this driver to evaluate a design concept for parallel data processing on heterogeneous hardware. The results suggest, that a combination of CPU and GPU computing power can improve the performance in computing tasks (see figure 4). Nevertheless, the driver is still not perfect. Some programs do not run and result in a GPU hang. It takes further work to investigate and fix these issues.

AMD

The AMD driver was part of a students project and was developed by a single person again, like the Intel driver. However, this driver had about one year of development time. Because we already developed the Intel driver, some parts of the AMD driver were similar and therefore easier to implement. Nevertheless, the AMD GPU is a dedicated GPU and has a lot more complexity.
The documentation issues are way more problematic than in the case of the Intel driver. Where the Intel documentation misses some registers or details, the AMD documentation misses complete chapters. It states nothing concerning the access to the GPUs memory or any type of initialization. Therefore, we had to use the open source driver as the main source of information. However, as we saw in Chapter 3, the open source driver has about double the size of the Intel driver. Additionally, the PCI part of the driver is more complicated, because the GPU is connected to a different PCI bridge than the CPU. Furthermore, access to the GPUs memory also has to deal with the different PCI bridge and PCI Express functions. Further, the GPU needs a special initialization executed by an *Atom BIOS*. The BIOS can be found in the *Expansion ROM* field of the PCI header. The BIOS however, is no executable code. It has to be interpreted by the driver. At this point, a complete interpreter was necessary. We decided to port this part of the amdgpu driver. Nevertheless, to get the Atom BIOS running it took about one month. After the execution of the Atom BIOS the screen becomes black, because the BIOS seems to disable some of the textmode parts of the GPU.
The current state of the driver is still stuck in the initialization phase of the GPU and the compute part is still missing, yet. AMDs friendly email support gave us important hints, but we still have not had the breakthrough.

7 SUMMARY AND OUTLOOK

To sum up, the main cause for the lack of GPU driver developement in the scientific community is related to the fast development of GPUs. Because of the mostly poor documentation and the hardware complexity it takes a lot of time and effort to develop even a special purpose driver. Integrated Intel GPUs age not as fast as dedicated GPUs from AMD or Nvidia. Furthermore, Intels documentation is the most useful and the GPU itself is not as complex as a dedicated one. Therefore a driver for Intel GPUs is feasible and may be also extendable for newer generations. Dedicated GPUs, on the other hand, age very fast. Even if we could develop a working driver, it is likely to be outdated as soon as it is finshed. At first, our Intel and AMD drivers have to be developed further and issues have to be solved. A possible solution could be an open source GPGPU driver with a simple interface, so that a research operating system can use the driver. Furthermore, Intel plans to release dedicated *XE* GPUs[14] with focus on GPGPU computing power. This could be a good GPU for research if Intel provides documentation in the same detail like their integrated GPU documentation. An alternative to the Intel XE

[13] Driver source can be found at: https://ess.cs.uos.de/git/software/uos-intel-gpgpu

[14] https://newsroom.intel.com/wp-content/uploads/sites/11/2020/11/SC20-Keynote-Slides.pdf

GPU could be an open source GPGPU like the MIAOW project, that offers an open source RTL implementation of a GPGPU (see [2]). Our hope is to draw more attention to this topic, so that GPU manufacturer will show more interest in good hardware documentation or even publish additional open source implementations, that are easy to use in research operating systems.

Acknowledgements The work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) under grant no. SP 968/9-2.

REFERENCES

[1] David M Airlie and Open Source Contractor. 2006. Open Source Graphic Drivers-They Don't Kill Kittens. In *Proceedings of the Linux Symposium*, Vol. 1. 19–26.

[2] Raghuraman Balasubramanian, Vinay Gangadhar, Ziliang Guo, Chen-Han Ho, Cherin Joseph, Jaikrishnan Menon, Mario Paulo Drumond, Robin Paul, Sharath Prasad, Pradip Valathol, and Karthikeyan Sankaralingam. 2015. Enabling GPGPU Low-Level Hardware Explorations with MIAOW: An Open-Source RTL Implementation of a GPGPU. *ACM Trans. Archit. Code Optim.* 12, 2, Article 21 (June 2015), 25 pages. https://doi.org/10.1145/2764908

[3] Feras Daoud, Amir Watad, and Mark Silberstein. 2016. GPUrdma: GPU-Side Library for High Performance Networking from GPU Kernels. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers* (Kyoto, Japan) *(ROSS '16)*. Association for Computing Machinery, New York, NY, USA, Article 6, 8 pages. https://doi.org/10.1145/2931088.2931091

[4] Marcel Lütke Dreimann. 2019. Ein Treiber für die native Codeausführung auf Intel GPUs für den MxKernel. (2019).

[5] José Duato, Francisco D. Igual, Rafael Mayo, Antonio J. Peña, Enrique S. Quintana-Ortí, and Federico Silla. 2010. An Efficient Implementation of GPU Virtualization in High Performance Clusters. In *Euro-Par 2009 – Parallel Processing Workshops*, Hai-Xiang Lin, Michael Alexander, Martti Forsell, Andreas Knüpfer, Radu Prodan, Leonel Sousa, and Achim Streit (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 385–394.

[6] Shinpei Kato, Michael McThrow, Carlos Maltzahn, and Scott Brandt. 2012. Gdev: First-Class GPU Resource Management in the Operating System. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, Boston, MA, 401–412. https://www.usenix.org/conference/atc12/technical-sessions/presentation/kato

[7] Michael Müller, Thomas Leich, Thilo Pionteck, Gunter Saake, Jens Teubner, and Olaf Spinczyk. 2020. He..ro DB: A Concept for Parallel Data Processing on Heterogeneous Hardware. 82–96.

[8] Michael Müller and Olaf Spinczyk. 2019. Mxkernel: rethinking operating system architecture for many-core hardware. In *9th Workshop on Systems for Multi-core and Heterogenous Architectures*.

[9] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. GPU Computing. *Proc. IEEE* 96, 5 (2008), 879–899. https://doi.org/10.1109/JPROC.2008.917757

[10] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray, and Emmett Witchel. 2011. PTask: Operating System Abstractions to Manage GPUs as Compute Devices. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 233–248. https://doi.org/10.1145/2043556.2043579

[11] A. J. Younge, J. P. Walters, S. Crago, and G. C. Fox. 2014. Evaluating GPU Passthrough in Xen for High Performance Cloud Computing. In *2014 IEEE International Parallel Distributed Processing Symposium Workshops*. 852–859. https://doi.org/10.1109/IPDPSW.2014.97

[12] Xiuxia Zhang, Guangming Tan, Shuangbai Xue, Jiajia Li, Keren Zhou, and Mingyu Chen. 2017. Understanding the GPU Microarchitecture to Achieve Bare-Metal Performance Tuning. *SIGPLAN Not.* 52, 8 (Jan. 2017), 31–43. https://doi.org/10.1145/3155284.3018755