The Panfrost Driver

BY RUSLAN BUKIN

The mature operating system and the new graphics driver for the arm64 platform gets us ready to run FreeBSD on high-end server/desktop systems, research-platforms, embedded environments, or even current smartphones/personal devices.

Every tech giant is considering moving—or is already moving—their hardware to arm64 architecture. Google just announced the Tensor chip for their new Pixel 6 devices; Amazon built AWS Graviton processors; Facebook is testing massive arm64 server deployment and powers Oculus AR/VR glasses with an arm64 SoC. The Apple M1 SoC debuted in Macbooks. Many Android devices and Google Chromebooks are now based on arm64.

At the same time, the arm64 port for FreeBSD has been promoted to a Tier 1 platform and a few arm64 server-solutions are now available from different vendors. Some people in the FreeBSD community are already using arm64 machines as FreeBSD desktops.

The largest ARM market is still smartphones (the ARM share is nearly 100%) and personal devices. These are based on a variety of ARM-based SoCs with a choice of peripherals and network connectivity options included (5G, etc.). But the selection of Mobile GPUs the SoCs are based on is limited: Vivante, Broadcom VideoCore, Nvidia Tegra, Qualcomm Adreno, Imagination PowerVR, ARM Mali and the graphics included for the Apple M1 SoC.

FreeBSD does not support any of these GPUs, but work is in progress on one of them, so, let's talk about the ARM Mali GPU.

The Mali GPU

The Mali GPU was first launched in 2005 by Falanx Microsystems A/S (a Norwegian company)—a spin-off from a research project of the Norwegian University of Science and Technology, and then acquired by ARM in 2006. Since 2007, ARM has developed several generations/micro-architectures of Mali: Utgard, Midgard, Bifrost and a very new Valhall. The Utgard GPU (Mali 4xx) was a GPU of the armv7 era, but it is still available on a newer low-end arm64 systems. ARM claims it is the world's most shipped mobile GPU ever. Midgard (Mali Txxx) and Bifrost (Mali Gxx) are newer GPUs that are often included with an arm64-based smartphone or tablet.

The Mali GPU has several threads intended for different operations. For example, the Midgard GPU uses complex tiling architecture. It has three threads responsible for vertex shaders; a tiler that sorts triangles into tilers and passes down to a fragment shader; and the fragment shader that processes final rasterization over passed-in tiles (i.e. writes to the framebuffer).

The Mali GPU supports Vulcan 1.2, OpenGL ES 3.2, OpenCL 1.2 (Midgard architecture), OpenCL 2.0 (Bifrost architecture), and OpenCL 2.1 (Valhall architecture). SPIR-V is also supported by Mali GPUs.

Software

ARM writes its own proprietary software drivers in-house for these GPUs. These are binary blobs for Linux. Binary blobs are not available for FreeBSD and using them on Linux ties you to a specific version of the Linux distribution/kernel and display subsystem. That restricts your ability to use your machine as you see fit and undermines the philosophy of free and open-source software. It also provides no way to debug the code and adds security issues. It disallows kernel hackers from using the latest development kernel on their desktop machines. But the good news is that there are two open-source reverse-engineered (RE) drivers for Mali GPU now available: Lima and Panfrost.

A GPU Open-source Driver

The graphics stack consists of several components, mainly the `userspace` graphics driver and the kernel part of it, but also the direct rendering module (DRM), kernel mode setting (KMS) and `libdrm`.

The Userspace Part (the Mesa Library)

A graphics driver runs in userspace and makes the translation between the Graphics API (OpenGL, OpenCL, etc) to the native sequence of instructions (opcodes) in each GPU ISA.

The driver composes job chains that have to be executed on GPU. A job is a sequence of instructions—a bunch of abstract memory buffers, including temporary buffers and main data. A job also comes with a set of dependencies (called 'fences') that must be met before a job can be put on a GPU thread to run. For example, a job could depend on completion of another job. A few to a few hundred jobs per second are executed on the GPU during the graphics workload. A job is composed by the graphics driver, which is a part of the Mesa Gallium library.

The Lima Driver

You may have heard about Lima—an open-source reverse-engineered driver for Mali. It is made for Utgard micro-architecture. It targets OpenGL ES 2.0 (embedded) as well as OpenGL 2.1 (desktop). It can't do newer APIs like OpenGL ES 3.2 or OpenCL/Vulkan due to Utgard hardware limitations. Initially developed by Luc Verhaegen at the beginning of 2012, this work was sponsored by Codethink. The project was abandoned in 2013 and stagnated for some time, before it was resumed in June 2017 by Qiang Yi (a Software Engineer at AMD).

Qiang Yi begun integrating the driver into the Mesa library, and, as of March 2019, the driver was fully merged. The Lima driver is supported on various armv7 boards and Chromebooks. Although the Lima could probably be built on FreeBSD with no issues, we have not written the kernel part for it.

The Panfrost Driver

A new project covers support for Midgard and Bifrost—newer GPU architectures that are included in an enormous number of arm64 systems. It was made by tracing ARM's proprietary userspace driver and open-source kernel drivers.

The original effort began with the Lima driver to support Midgard under the name Tamil, but source code was never released. A new project with the codename Chai was founded by Alyssa Rosenzweig—a mathematics student at the University of Toronto. Alyssa's goal was to write a free software driver for Midgard GPUs, also known as the Mali Txxx series.

Alyssa did most of the Midgard reverse-engineering and driver development. A separate project, BiOpenly, began an effort to reverse-engineer the Bifrost, and then project Chai was merged with BiOpenly under the name Panfrost. Later, Alyssa joined Collabora Ltd to continue work on Panfrost.

Unlike Lima, Panfrost was done in partnership with ARM. It was initially (and almost entirely) done through a self-funded effort by the Panfrost community and Collabora Ltd, but at some point, ARM Ltd joined the effort to make the solution optimal. ARM provided the documentation and helped upstream the Mali kernel driver to Linux.

Build Instructions on FreeBSD

What follows is an example for how to build the Panfrost userspace driver under FreeBSD:

```
git clone https://gitlab.freedesktop.org/mesa/mesa
mkdir build && cd build
meson .. . -Degl=enabled -Dgles1=enabled -Dgles2=enabled -Ddri-drivers= -Dvulkan-drivers=
-Dgallium-drivers=panfrost,kmsro
ninja
```

The Panfrost Kernel Driver

I began working on the Panfrost kernel driver with the goal of creating the first-ever mobile GPU support for the FreeBSD arm64 platform. The GPU is a memory-mapped device like many other peripherals on arm64 SoC. The kernel driver of the graphics stack provides an interface (a 'glue') to the OS and to the actual GPU hardware. It is the only part of the graphics stack that directly accesses GPU memory-mapped registers. It is responsible for GPU power management, hardware initialization, GEM objects management, GPU threads management, MMU configuration, GPU interrupts and MMU pagefault handling. The GPU kernel driver accepts jobs from userspace, allocates buffers, sets up MMU mappings, puts jobs to run on GPU and controls the overall operation of GPU.

As the Panfrost driver supports various GPU models and versions, it is responsible for setting up quirks and workarounds for any hardware issues related to a particular model.

As well as interfacing to DRM and handling its requests, a GPU kernel driver provides an ioctl interface for its userspace part. The ioctl layer is mainly used to submit jobs for processing on GPU, managing buffer objects (BO), and MMU mappings management.

IOCTL calls

A set of ioctl calls has been implemented in the kernel driver:

1. `panfrost_ioctl_submit` – used to submit a job to the kernel driver.
2. `panfrost_ioctl_wait_bo` – waits for a completion of the last `panfrost_ioctl_submit` on a buffer object (BO). This is needed for the cases where multiple processes are rendering to a BO and you want to wait for all rendering to be completed.
3. `panfrost_ioctl_create_bo` – creates Panfrost BO.
4. `panfrost_ioctl_mmap_bo` – used to map Panfrost BOs to userspace. This doesn't perform any mmap but returns an offset to use in an mmap on the DRM device node.
5. `panfrost_ioctl_get_param` – allows the userspace driver to read information about the hardware, such as GPU model, version, features present, etc.
6. `panfrost_ioctl_get_bo_offset` – returns an offset for the BO in the GPU address space.
7. `panfrost_ioctl_madvise` – gives advice to the kernel in the case where buffer contents (backed pages) could be temporarily released back to OS to save memory.

The MMU

The Mali GPU has an internal MMU, and so all the buffers with which the GPU operates are virtually addressed. The format of MMU page tables is mostly compatible with arm64 CPU MMU. There are a few differences, however, that have been discovered during this work:

- The page type bit of a PTE entry is different from the CPU MMU.
- The access flag bit is inverted.
- Some of the permission bits are inconsistent.
- The control registers and TLB maintenance paradigm are wildly different.

Before we put a job on the GPU, we have to setup MMU mappings, and once the work has completed, we must tear them down. The MMU support for the Mali GPU for FreeBSD has been developed, reviewed, and already committed to the base tree. It is part of the arm64 IOMMU framework (see `sys/arm64/iommu directory`). Initially, and similarily to the ARM System MMU (SMMU), the GPU MMU management routines were introduced into the arm64's CPU page management (pmap) code. Later people started complaining that maintaining SMMU, GPU and CPU MMU in the same pmap slows down the development process significally, so we have made a decision to split out SMMU and GPU routines to a new pmap specifically made for IOMMU (`iommu_pmap.c`).

KMS

The Mali GPU only processes rendering. It processes data in memory and puts it back to memory. It does not feature display controllers and has no idea how to output any data to a monitor. Generally, graphics support of an arm64 platform consists of various hardware components:

- The video display processor that operates with hardware windows (planes). At least two planes should be registered—the main window plane and a cursor plane. This device blends them when we move the cursor on the screen (it puts one layer on top of another in hardware) and provides the final data to output to the Display Interface Controller.
- The display interface controller which outputs data to the display on the physical layer. Interfaces like HDMI, DisplayPort, LVDS, MIPI DSI are usually implemented in this device.
- Peripheral devices like I2C/SPI. An i2c controller is usually required to read information about the connected HDMI monitor (its resolution, capabilities, etc.) or to configure a MIPI display.
- The GPU itself.

Along with a set of drivers for these devices, FreeBSD should support the clock framework for a given platform, as a set of clock domains must be configured to transmit data to the display.

The KMS (kernel-mode setting) is a part of DRM. It allows the user to configure the outputs when multiple hardware planes are registered as well as when multiple resolution/connectors options are available.

The DRM

The Direct Rendering Manager (DRM) is a framework in Linux and FreeBSD for managing Graphic Processing Units (GPUs). It is designed to support complex GPUs providing unified access to hardware. It is also responsible for memory management, DMA and interrupt handling.

DRM exposes the API (a set of ioctls) allowing userspace programs to send commands/data to GPU and to configure kernel mode setting (KMS). libdrm provides a set of wrappers around drm ioctl calls to simplify the API for user applications.

We currently support two DRM/KMS projects:

1. drm-kmod and LinuxKPI

DRM appeared in FreeBSD as a '`drm-kmod`' port in 2001 to support 3D graphics cards at that time (also known as `drm1`). From 2010-2012 Konstantin Belousov developed support for the new Intel HD Graphics GPUs introducing new DRM support in FreeBSD base (`drm2`).

A new project, `drm-next`, started around 2017 to support unmodified DRM. With frequent updates upstream, it became difficult to support `drm2`—a ported version of graphical drivers—as it required going through the driver and changing Linux-KPI to FreeBSD-KPI on every driver update.

DRM has been developed in a Linux kernel environment, so it uses the Linux kernel API. To use DRM code 'as is' in FreeBSD, we must translate its calls to native FreeBSD primitives/APIs. In this new '`drm-next`' project it is done by using the LinuxKPI framework. LinuxKPI was originally made for storage devices and networking adapter drivers by Mellanox Inc. and others, but now it is also required for importing graphics drivers from Linux. Due to licensing issues, they can't be part of the FreeBSD kernel (only as a module).

LKPI makes it easier to bring Linux DRM drivers to FreeBSD with minimal modifications and keep them up to date against Linux upstream. Later in 2018, the `drm1` and `drm2` were moved from the kernel to the ports tree as `drm-legacy-kmod` port; while `drm-next port` took the original name '`drm-kmod`'.

This DRM/KMS driver is available not only on x86 with support for most modern Intel HD GPUs, but on powerpc and arm64 platforms as well, targeting PCI-e based AMD/Radeon graphics cards.

I have been using this DRM on the ARM Neoverse N1 (arm64) server with a Radeon PCIe card for several months and it is working just fine.

Installing the driver is as easy as:

```
cd /usr/ports/graphics/drm-kmod
make install
```

2. drm-subtree & DRMKPI

A new project, 'drm-subtree', to support DRM on FDT-based arm and arm64 systems has been started by Emmanuel Vadot and Michal Meloun, with the goal of moving the DRM back to the kernel.

On FDT-based platforms like arm and arm64 the situation with DRM is a bit different: we want to keep DRM in the kernel, as it better suits embedded environments and we don't want to import arm graphics drivers from Linux. One reason is that the frameworks for FDT-based resources (gpio/pinmux/clock/regulator) are different between Linux and FreeBSD and that could make the LinuxKPI abstraction very complex. Also, drivers need to be synced with DTS files that we regularly import from Linux.

This project includes a new DRMKPI framework that is a bit of a stripped-down version of LinuxKPI—but for DRM-only. In DRMKPI, most unneeded storage/networking code—like `netdevice/sysfs` code—is removed, eliminating most fields from Linux's `task_struct` with the goal of eliminating `task_struct` itself, too.

At https://github.com/evadot/drm-subtree you can find a working set of DRM/KMS drivers and instructions for Allwinner, Nvidia Tegra and Rockchip SoCs that allow the display to work under FreeBSD.

A Target Platform for Panfrost

The FreeBSD Panfrost kernel driver has been developed for the Morello Board — a collaboration between ARM and Cambridge Computer Laboratory to build the first CHERI-enabled ASIC. This work is sponsored by UKRI with support from ARM Ltd.

The CHERI CPU that is included with this board is based on ARM Neoverse N1 SoC — the first server-class architecture from ARM, the Cortex-successor. While the Morello Board SoC will include the Bifrost GPU, the N1 development platform has no GPU included, which means we can't use it for GPU driver development.

The rk3399 board used for Panfrost kernel driver development.

The Morello Board is currently in its last stages of manufacturing and will be available later this year. In its absence right now, we have chosen Rockchip RK3399 as the target platform for Panfrost kernel driver development. The RK3399 is a mature and well supported FreeBSD platform with lots of llow-cost single-board computers available on the market. It includes a Mali Midgard GPU, HDMI controller with Full-HD, 4K support and even a PCI-e controller. It is an interesting platform because it can run entirely using free software including the GPU drivers.

The Result

The project started in November 2020, and it took around 6 months for me to have the Mali GPU working. Most of the time was spent writing Rockchip DRM drivers (video engine, display controller) and making bugfixes in peripheral drivers. The most difficult part of the Panfrost driver was figuring out the MMU unit behavior, its page tables format and cache-management operations. I also had to add missing functionality to the drm-subtree (to import the DRM scheduler and its dependencies). Thanks to the people on the bsdmips and panfrost IRC channels for help and moral support!

Quick tests demonstrate good performance and an overall quick response of the desktop graphical environment. The RK3399 has two video display processors: Big and Little. While Big features 4K resolution, the support was added for the Little targeting Full-HD. Both Wayland and Xorg operate well with minor issues like memory leakages (and work is in progress to fix it).

I was able to run a 2D accelerated desktop, browsers and 3D mesa demos like `glxgears(1)` and `geartrain(1)`: the HDMI monitor max refresh rate (60 FPS) was easily reached.

I'm looking forward to switching my main desktop to the Morello Board and to using my Panfrost driver daily. The Panfrost driver for FreeBSD has been put up for review on github: https://github.com/evadot/drm-subtree.

RUSLAN BUKIN is a Senior Research Software Engineer at the University of Cambridge Computer Laboratory. His main area of interest is machine-dependent software. He has led several projects including the port of FreeBSD to RISC-V architecture, IOMMU support for arm64 and Intel SGX support for amd64. Aside from FreeBSD, he has developed the MDEPX embedded kernel used in research projects. In his free time he enjoys cycling across the UK.