ACCFS – Operating System Integration of Computational Accelerators Using a VFS Approach

Andreas Heinig[1], Jochen Strunk[1], Wolfgang Rehm[1], and Heiko Schick[2,⋆]

[1] Chemnitz University of Technology,
Str. der Nationen 62, 09126 Chemnitz, Germany
{heandr,sjoc,rehm}@cs.tu-chemnitz.de
http://www.tu-chemnitz.de/cs/ra/projects/accfs/
[2] IBM Deutschland Research & Development GmbH
Schönaicher Str. 220, 71032 Böblingen, Germany
schickhj@de.ibm.com

Abstract. For a number of applications integrating specialized computational accelerators into a general-purpose computing environment yields more performance per watt and per dollar than a pure multi-core approach. In contrast to fully application-specific hybrid solutions we offer the advantage to maintain traditional programming models and development environments to a certain extent. In this paper we introduce an open generic operating system interface concept what we call Accelerator File System (ACCFS) for integrating application accelerators into Linux based platforms. By describing the proposed concepts and interface we contribute to a broader discussion of this challenging topic.

1 Introduction

The usage of reconfigurable hardware becomes more and more an important theme in research and industry. Especially FPGAs (e.g. Xilinx Virtex, Altera Stratix, DRC.) can speedup a variety of applications. Woods et al. [1] gained a speedup of more than 50 compared with a CPU when accelerating a Quasi-Monte Carlo Simulation.

An emerging type of computational accelerators are Programmable Graphic Processors. The leading vendors for graphic boards recently presented special techniques to accelerate applications with their chips. Massive parallel GPGPUs (General Purpose Graphics Processing Units), like the Nvidia Tesla or the AMD FireStream chips, enable huge speedups. For example, a "deformable image registration" calculation reaches a speedup by factor 34 [2].

Another platform is the IBM Cell/B.E. processor which was developed in a cooperation of IBM, Toshiba and Sony. On this processor multiple independent

⋆ The ACCFS project is performed in collaboration with the Center of Advanced Study IBM Böblingen, Development and Research, Germany.

J. Becker et al. (Eds.): ARC 2009, LNCS 5453, pp. 374–379, 2009.

vector processors called SPUs (Synergistic Processing Units) are built around a 64-bit PowerPC core (PPE).

One of the big challenges when using such "accelerators" is the efficient integration into a HPC system environment and the definition of appropriate programming models as well. In this paper we address the operating system integration by defining an open generalized interface. Therefore, we evolve basic concepts in Sections 3. The implementation called Accelerator File System (ACCFS) is described in Section 4. In Section 5 some accelerator examples are mentioned. Section 6 discusses further work and summarizes the results.

2 Related Work

To integrate a FPGA into the operating system several solutions exist. One is the extension of Linux with hardware processes (e.g. BORPH [3]) or hardware threads (e.g. ReconOS [4]). BORPH uses conventional UNIX IPC mechanisms to communicate. Inter-thread communication is mapped to FIFOs. The hardware threads introduced by ReconOS can use the normal operating system services to interact with the environment. However, both solutions only concentrate on FPGA integration without providing an open interface for easy integration of additional hardware by using clearly exported structures. Also, these models lack performance when they are implemented on platforms where the processor that is running the operating system is not an integral part of the FPGA itself. The main drawback is the deep modification of the Linux kernel. Thus porting to another kernel version becomes more complicated and sometimes impossible without patching when major kernel parts are changed in a new revision.

Another approach for FPGA integration is XVFS (Xilinx Virtual File System) presented by Donlin et al. [5]. Here a virtual file system is used to directly modify the low-level parts of the FPGA. XVFS represents all heterogeneous resources in a file system hierarchy. Thus, it is possible to use standard `read` or `write` system calls to get or manipulate the configuration of the FPGA. Furthermore, access to every LUT, BRAM and routing information is granted. However, we do not need such a detailed view as it is necessary for real-time or embedded system designers. In our scope the FPGA is a computational accelerator where we abstract the functionalities. Only the communication mechanisms message passing and direct memory access together with the (re)configuration are needed. This will enable us to establish a generalized accelerator model which is beneficial for library and application programmers. When accelerating an application, the same communication and usage models can be adopted even if different accelerator types are used.

If we look at graphic cards, we will see another integration approach named CUDA [6]. A C-like language is used to program Nvidia graphic chips. CUDA enforces a functional offload programming model where the compute kernels are off-loaded to the GPGPU. The graphic card is exported to the user-space via a character device. Data exchange is enabled through DMA transfers supported by a library. It is not possible to map the memory into the application address space. This restricts the achievable performance in a variety of communication

patterns. A second drawback is the usage of the `ioctl` kernel interface. It is not possible to provide an open generic interface when using such `ioctl`s, because no standards exist how to define these.

A. Bergmann developed SPUFS (Synergistic Processing Unit File System) [7], which is used to integrate the SPUs of the Cell/B.E. processor into the Linux environment. The concepts of SPUFS are the virtualization of the SPU and the virtual file system context access. A context is mapped into a virtual file system such that it appears as a directory containing special files. File system operations on these files effect the communication with a SPU.

The main advantage of both, SPUFS and XVFS, is the clean interface without the usage of `ioctl`s. Additionally no modifications of existing kernel structures are necessary. Thus, we base our solution on these concepts.

3 Accelerator Integration Concepts

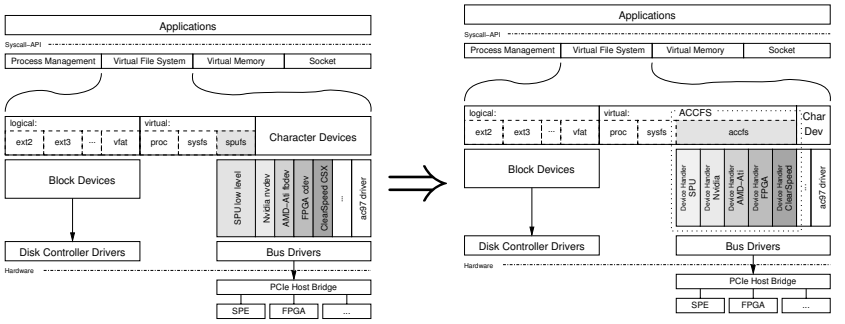
Virtualization. To virtualize the accelerators we abstract the *physical accelerator* with an *accelerator context*. The context is the operational data set of the accelerator. It includes all necessary information to describe the current hardware state in such a way that the operation can be interrupted and resumed later without data loss. In a multi-user/thread/process environment, like Linux, virtualization optimizes the resource usage of the accelerators. Contexts which do not make use of the hardware at a given time are not scheduled on the physical accelerator.

Generic Interface. We propose a virtual file system (VFS) to interface the accelerator. This offers the advantage of a complete `ioctl` free implementation because every functionality can be exported through a dedicated file. Each context can be bounded on a directory inside the VFS. The accelerator is accessible through a set of files supporting the POSIX file operations. To include reconfigurable hardware as well, this file set has to be dynamically exported.

Separation. The integration of new accelerators will be much easier when splitting the functionalities in a part which handles the common abstraction layers and a part managing the low-level hardware access.

Host initiated DMA (Direct Memory Access). To interact with the accelerator several methods are feasible. One is simple memory mapped IO with standard load/store machine instructions. In this memory access method the host is the active part who issues a read or write for every memory access. Another method is DMA-bulk transfer. Here the accelerator needs a DMA unit capable of moving the data. In cases where the accelerator is able to initiate these transfers by itself, the DMA unit has to handle virtual memory managing issues, too. However, not every accelerator supports virtual memory. For this reason we propose host initiated DMA, where the host setups the memory management unit and initialize the data transfer. The actual data movement is done asynchronously by the accelerator.

Asynchronous Context Execution. This concept eases the software development because multi-threading is not required when using multiple accelerator units. Every context runs asynchronously to the host system.



(a) **Without ACCFS**

(b) **ACCFS: Common Generalized Interface**

Fig. 1. ACCFS - Layered Structure

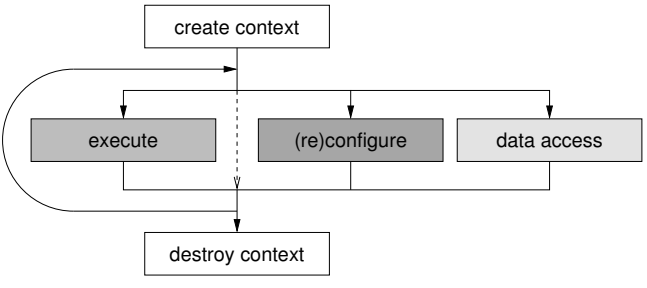


Fig. 2. Flow Diagram: ACCFS Interface Usage

4 ACCFS - Implementation

The concepts described in Section 3 are implemented in ACCFS (Accelerator File System). ACCFS is an open generic system interface based on SPUFS. It is designed to integrate different accelerator types into the Linux operating system. The goal of ACCFS is to replace the different character device based interfaces (cf. Figure 1a) with a generic file system based interface (cf. Figure 1b).

The whole dotted block in Figure 1b shows ACCFS which is divided into two parts. Part one, labeled "accfs", is the file system driver. Its task is to handle the common stuff equally belonging to all accelerators. Included are VFS management, context creation, setting up DMA transfers, and providing the user interface. Last but not least it handles the integration of the second part the "device handlers". Therefore, a special interface called "Vendor Interface" is implemented.

4.1 User Interface / VFS

ACCFS implements the flow diagram shown in Figure 2.

Before the application can use an accelerator a context has to be created by invoking the `acc_create` system call. The desired accelerator type is selected with the arguments *vendor_id* and *device_id*. We chose this extra system call

Table 1. ACCFS Context

File	Description	File	Description
regs	register file	memory/	exported memories (directory)
[io]box	in/out-bound mailbox	status	execution status and synchronization
[io]box_stat	in/out-box status	config	device configuration

because `mkdir` can not convey so much arguments. Creating a special file was also not an option, because ACCFS exports only the data related to a context. A special control file however is not context specific. After the successful context creation a new directory is created and the belonging file handle is returned. The context is destroyed when closing this handle. The `acc_run` system call is used to start the execution of the accelerator. The accelerator (program) is executed asynchronously, meaning that `acc_run` returns immediately without waiting for finished execution.

The communication with the accelerator is performed with the normal POSIX `read` or `write` system calls on the files exported by ACCFS. Table 1 shows an overview of the file set. If a file is visible/accessible depends on the device handler that can dynamically enable or disable the entries depending on the capabilities of the device.

A detailed view of the user interface can be found on the ACCFS project page `http://www.tu-chemnitz.de/cs/ra/projects/accfs`.

4.2 Vendor Interface

The vendor interface consists of a couple of functions exported by *accfs* and a structure called *accfs_vendor* which have to be registered by each device handler. This structure contains callback functions which are invoked by *accfs* instead of the generic routines. For example, if the vendor needs to constrain the register file access because the hardware can only write to 32bit boundaries, the device handler has to declare the `regs_write` function. Inside this function the correct alignment can be checked.

5 Accelerator Support

Currently we are working on Cell/B.E. integration. Hereby a QS21 Cell Blade is coupled with a Intel Xeon System via PCIe. We have already established a heterogeneous coupling over Ethernet. There the RSPUFS (Remote SPUFS)[8] implementation made the SPUs accessible on the AMD Opteron system.

As another proof of concept we have implemented an accelerator for arithmetic operations on a HyperTransport FPGA card [9] based on a Xilinx Virtex-4. An Iwill DK8-HTX motherboard with two Opteron processors is used as host system. To get the FPGA running as a HT cave device we replaced the pre-installed BIOS with a modified LinuxBIOS version. Our first test results looks promising. We are able to exchange reconfigurable modules (currently a pattern matcher and a Mersenne Twister) during run-time with the help of ACCFS. A detailed

description of our solution will be published in a paper which is already submitted under the name "Run-Time Reconfiguration for HyperTransport coupled FPGAs using ACCFS".

The virtualization of the FPGA is currently not implemented. However, we plan to realize a cooperative scheduling model, where a module running on the FPGA is requested to step in a kind of a "stop" state. When reached, the module will be removed after saving the block RAM. This means all information inside the block RAM must be sufficient to continue the operation after restore.

6 Conclusion

In this paper we have presented aspects of the concepts and the implementation of ACCFS. These concepts build the grounding of our approach to establish an open generalized accelerator interface. We have analyzed both reconfigurable and non-reconfigurable hardware to define a common set of interface functions. This interface supports the direct access of an accelerator through register file, memory or via mail boxing. Those functionalities are reflected in files inside the VFS exported by ACCFS. The file system calls `open`, `close`, `read`, `write`, and `mmap` are building the user interface. With the exception of two new system calls we do not modify any parts of the Linux kernel. This enables us to provide ACCFS support also for future kernel releases.

Further research will focus on the complete integration of the Cell/B.E. and FPGAs. We are also planning to port ACCFS to host systems other than x86.

References

1. Woods, N.A., VanCourt, T.: FPGA Acceleration of Quasi-Monte Carlo in Finance. In: Proceedings of FPL, pp. 335–340. IEEE, Heidelberg (2008)
2. Samant, S.S., Xia, J., Muyan-Ozcelik, P., Owens, J.D.: High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy. Medical Physics 35(8), 3546–3553 (2008)
3. So, H.K.-H., Brodersen, R.: A unified hardware/software runtime environment for FPGAbased reconfigurable computers using BORPH. Trans. on Embedded Computing Systems 7(2), 1–28 (2008)
4. Lübbers, E., Planner, M.: ReconOS: An RTOS Supporting Hard-and Software Threads. In: Proceedings of FPL, pp. 441–446. IEEE, Amsterdam (2007)
5. Donlin, A., Lysaght, P., Blodget, B., Troeger, G.: A Virtual File System for Dynamically Reconfigurable FPGAs. In: Proceedings of FPL, pp. 1127–1129 (2004)
6. NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide. Santa Clara, CA, Tech. Rep, version 1.1 (November 2007)
7. Bergmann, A.: The Cell Processor Programming Model. IBM Corporation, Tech. Rep. (June 2005)
8. Heinig, A., Oertel, R., Strunk, J., Rehm, W., Schick, H.: Generalizing the SPUFS concept - a case study towards a common accelerator interface. In: Proceedings of MRSC, Belfast, April 1-3 (2008)
9. Nüssle, M., Fröning, H., Giese, A., Litz, H., Slogsnat, D., Brüning, U.: A Hypertransport based low-latency reconfigurable testbed for message-passing developments. In: Proceedings of KiCC 2007 (2007)