Operating System Interfaces to Reconfigurable Systems

John H. Kelm, Isaac Gelado, Kuangwei Hwang, Dan Burke,
Sain-Zee Ueng, Nacho Navarro, Steve Lumetta, Wen-mei Hwu
Soft Systems

Annual Symposium
September 2006

Overview

We present a hybrid CPU with accelerator computing model. The prototype platform runs the GNU/Linux operating system on the embedded PowerPC of the Xilinx Virtex-II Pro FPGA. We demonstrate two methods for interfacing reconfigurable hardware accelerators with common applications running on the processor: the *direct access model* and the *indirect access model*.

The direct access method enables applications to exploit platform-specific features and incur little overhead when accessing accelerators after initialization. The indirect access method provides a greater degree of abstraction between the application developer and the reconfigurable system. Three motivating examples are explored, all of which make use of the services provided by a standard operating system while also being executed partially in the reconfigurable fabric of the FPGA. Furthermore, we experiment with an accelerator framework for connecting accelerators to our system via well-defined interfaces.

Motivations

- **Increased Power Efficiency**
  - Power efficient performance for mobile multimedia workloads and cluster computing.
- **Use of Standard Operating Systems**
  - Target applications take advantage of the services provided by the OS
    - Examples: file systems, device drivers, networking support, standard C libraries
  - Applications also have kernels of execution accelerated in the reconfigurable fabric.
- **Reduced Design Complexity**
  - Application developers focus on the application development accessing accelerators via library call like interfaces.
  - Hardware designers focus on accelerator development by mapping accelerators into a well-defined accelerator framework.
  - This work provide the means to connect these two sets of interfaces.
- **Exploit Many Levels of Granularity**
  - Flexible interface methods for enabling reconfigurable hardware to effectively accelerate the spectrum of fine-grained to coarse grained parallelism.
- **Code Reuse and Portability**
  - An accelerator is designed once and is used by multiple applications.
  - It is portable across any architecture that supports the framework.
  - Library call abstraction allows accelerated applications to run on non-reconfigurable systems of the same architecture family without modification.

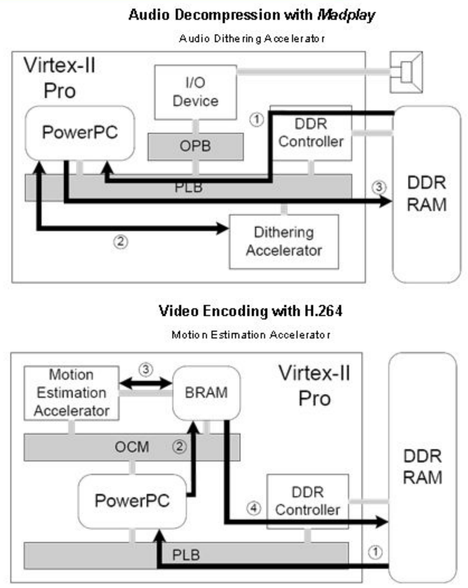
Future Work

- **Shared Accelerators**
  - What user/application/thread obtains access? Is there preemption?
  - Can there be a "context switch" on the accelerator?
  - Is access like a transaction? Is there state to be saved?
- **Protection Boundaries**
  - Debugging in a heterogeneous hardware/software environment can be a difficult.
  - Accelerators can have bugs too.
  - Malicious accelerators could be present.
  - Users could use benign, but unrestrained accelerators dangerously.
  - How is protected access to the accelerators provided? What is the granularity? How is the OS/Application involved?
- **Application-to-Framework Mapping**
  - How does a human programmer (or better yet, a compiler) map high-level programs into our hybrid model effectively and efficiently?
  - What sorts of applications can be accelerated using our framework and interfaces?
- **Increased Concurrency**
  - Overlap accelerator computation with CPU computation and CPU/accelerator communication.
  - Low overhead accelerator/application thread communication using Symmetric Multi-Threading (SMT)-like approaches.

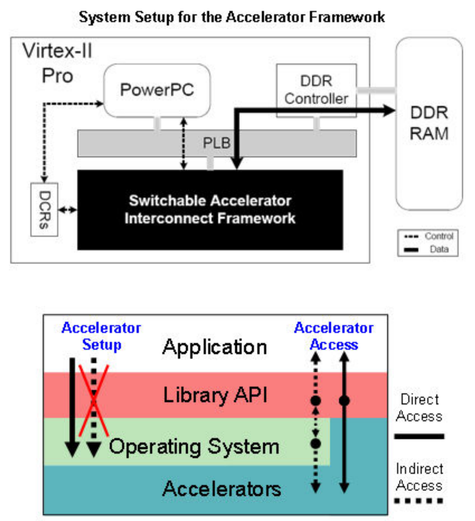
Direct Access Model

- Interface hardware accelerators based on the `mmap()` system call.
- The accelerator is attached to a system bus that is accessible from the processor via *physical addresses*.
- The OS provides each accelerated application with a set of pointers inside the *virtual addresses* space of the application.
- **The OS sets up the means to access the device directly and gets out of the way.**

The figures show the flow of data through the accelerators using two motivating example applications running on Linux.



Indirect Access Model

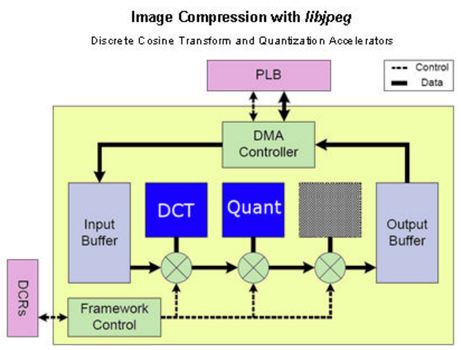


- An abstraction offering a simple, consistent interface to the application developer.
- Operating system intervention is required on every transaction between the accelerator and the application.
- The OS can allocate accelerator access and virtualize (i.e., execute the would be accelerated call on the CPU) guided by system policy and the dynamic behavior of the system.
- Access to protected resources (e.g., DMA and DCRs on the PPC).
- Takes advantage of high throughput burst DMA transfers across the PLB.
- Using DCRs to exchange control information with the accelerator allows for concurrent data and control exchanges with the accelerators.

Accelerator Framework

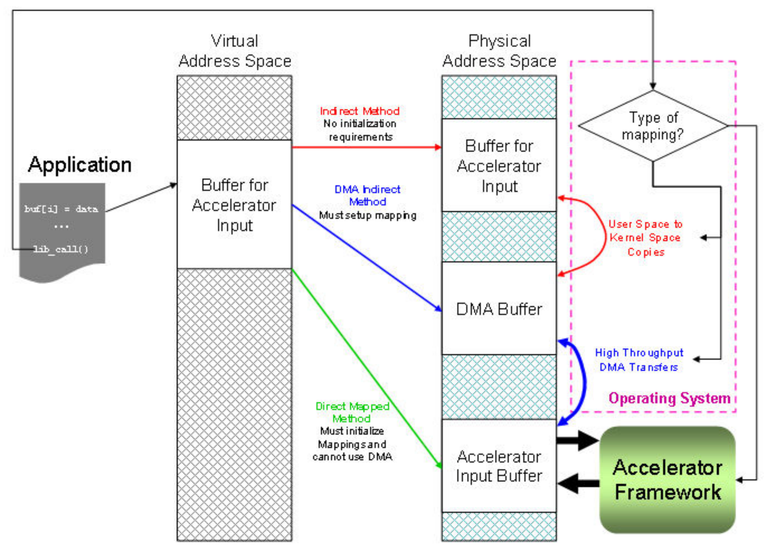
Switchable Interconnect Framework Goals:

- Provide infrastructure for reconfigurable accelerator development.
- Framework integrated with the direct and indirect access models.
- Reduce design and debugging times by reducing the complexity of the accelerator to system interface.
- Increase portability by providing a consistent, well-defined interface from both the application developer's and accelerator designer's points of view.
- Explore ways to optimize calls to the accelerator.
- Enable time multiplexing of resident accelerators and a means to electronically isolate accelerators for runtime reconfiguration.



The accelerator framework provides well-defined connections between accelerators using a switched interconnect. The framework would implement the buffers, controllers and interconnects as embedded logic to reduce the strain on place and route compared to an unrestrained design. Furthermore, it would take advantage of the high speed, high efficiency and high density of hardwired logic.

Interface Dataflow



Results

Action	Overhead cycles	Overhead μseconds
System Call Overhead	1,853	6.18
DMA Setup	549	1.83
DMA Transfers	448	1.49
Accelerator Execution	987	3.29
Cache Coherence	348	1.16
Data Copies	1060	3.53
Total Time	5,244	17.5

Table 1. Overheads Associated with the Indirect Access Model (JPEG)

Channeling all accesses to the reconfigurable accelerators through the OS allows for precise control over the resources by the system, but at a cost. Table 1 shows that ~19% of the accelerator call time is spent executing on the accelerator. The system call overhead represents the biggest challenge to enabling low overhead accesses using the indirect model, but the cost is not immutable. The more accelerator/OS interaction that can be encapsulated in a single call, the lower the amortized cost of the system call.

Total time per call		
Full Search (HW)	96,480	322
Diamond Search (SW)	174,911	583
Full Search (SW)	639,925	2,133

Accelerator Action	Overhead cycles	Overhead μseconds
`open` System Call	34,000	113.3
`mmap` System Call	24,000	80.0
Data Marshaling+Write	90,600	302
Initiation	180	0.600
Accelerator Computation	5,400	18
Read From Accelerator	300	1

Table 2. Motion Estimation Call Breakdown

Table 2 shows the number of cycles associated with each action taken by the ME accelerator during each call and the one time cost incurred when setting up the accelerator. The time spent doing ME in hardware is less than 1% that of doing full search in software, however the overhead associated with making the call must be accounted for. ME serves as an example of course acceleration.

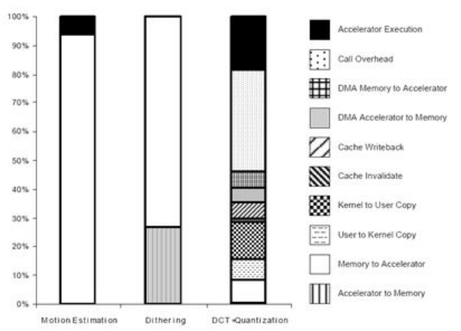


Figure 3. Time Spent in Each Component of a Call to the Three Example Accelerators

We compare of the time spent in each action of the direct access method (ME and dithering) and those of the indirect access method (DCT+Quantization).

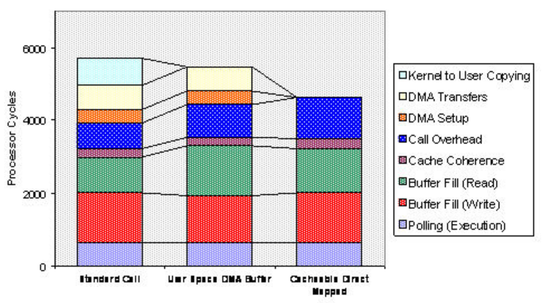


Figure 4. Indirect Access Model with JPEG

A standard copy moves data from a user space buffer to a kernel buffer requiring no initialization prior to accessing the accelerator. User space DMA buffers allow the application to write directly to a DMA enabled buffer mapped into the application's virtual address space. The direct mapped buffer forgoes DMA and writes directly to the accelerator input buffer through the data cache. However, these two methods incur costly initialization, mapping and unmapping.