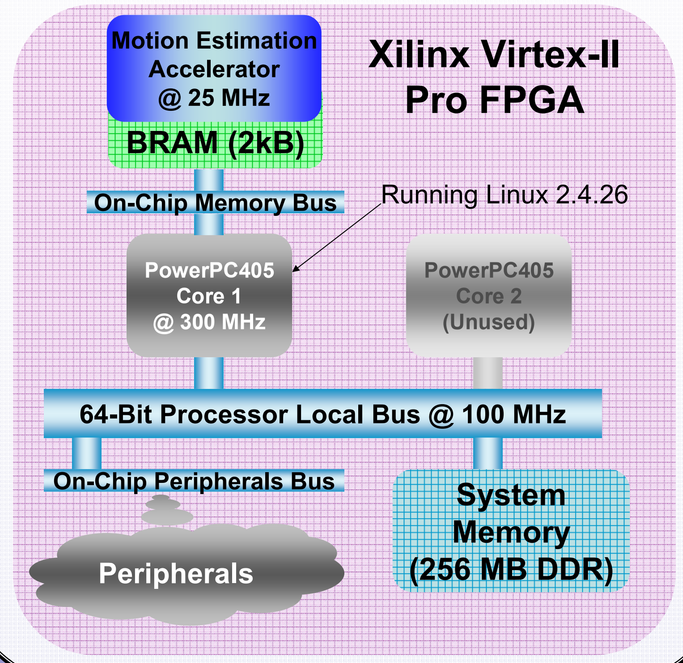
A Dynamic Library Interface to Reconfigurable Accelerators

John H. Kelm, Isaac Gelado, Kuangwei Hwang, Steven S. Lumetta, Nacho Navarro, Dan Burke, Wen-mei Hwu
{jkelm2,igelado,khwang3,steve,nacho,drburke,hwu}@crhc.uiuc.edu
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign

Introduction

Reconfigurable computing is a mechanism to exploit the inherent parallelism present in HPC applications with greater efficiency and functional density than general purpose processors. We present a programming model for reasoning about hardware accelerators as a reconfigurable platform integrated with HPC applications. Examples of hardware accelerators and protection mechanisms running on a commodity reconfigurable platform are presented.

Current Platform



Accelerator Model

• Beyond FPGA-Based Designs

The limited clock frequency of current reconfigurable systems (*i.e.,* FPGAs) when compared to current microprocessors requires reconfigurable systems to exploit high-levels of parallelism to provide demonstrable speedup. *Current* reconfigurable offerings are an artifact of the *current* market forces that shape reconfigurable hardware designs—a market not focused on highly integrated reconfigurable computing.

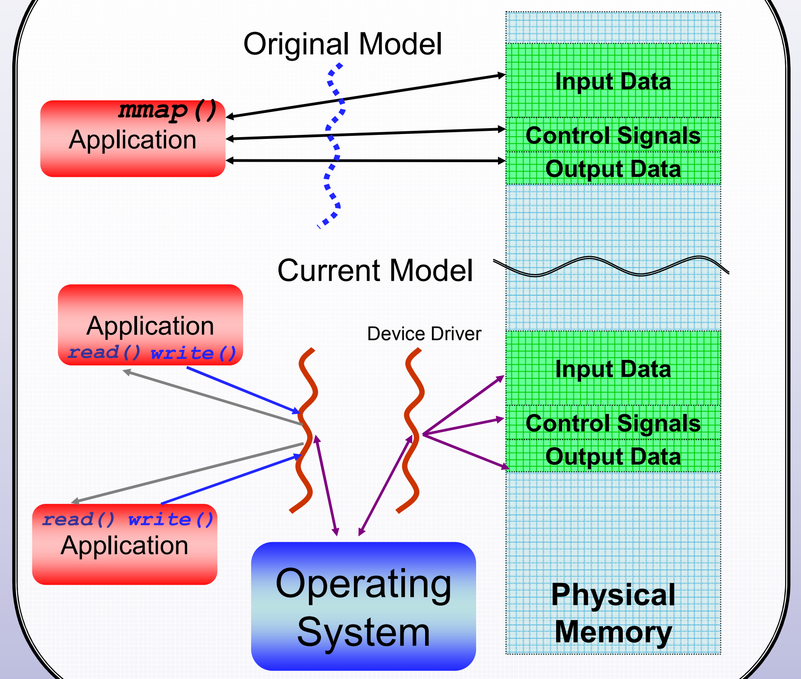
Integration and Applicability

Integrating some amount of reconfigurable logic with high-speed conventional processing cores on a single die has the potential to overcome limited off-chip bandwidth and longer off-chip latencies for I/O-bound accelerators. By providing quick access to accelerators, taking advantage of finer-grained and latency-sensitive kernels of execution becomes possible. The ability to exploit a finer-grained form of parallelism (*i.e.,* code segments amenable to being encapsulated in a library call) is crucial to expanding the applicability and underscoring the relevance of reconfigurable computing to the HPC community. Moreover, giving applications the ability to incorporate more fine-grained hardware acceleration can provide more speedup for applications that can already take advantage of accelerators exploiting coarse-grained parallelism.

• Programming Model

The hardware interface provided to the accelerators is as important as the application interface—the connection between the two being the main thrust of our work. Debugging is of key concern to developers who already have the onerous task of developing complex applications in a multithreaded/clustered computing environment. Encapsulating the implementation of the reconfigurable resources in library calls allows the systems designers to change how the underlying hardware will accelerate the application. Furthermore, the library interface provides a recognizable programming model for application developers to facilitate easier integration of hardware accelerators into their HPC applications.

Interface Model



Protection:

A First Order Concern

• The Need For Protection

Hardware accelerators need not be merely passive components that operate only on data pushed into their local store by their master application. Taking the passive approach to interfacing with accelerators reduces concurrency and the potential to utilize faster, low-latency access to data. However, by providing greater capabilities in our model of hardware accelerators we must realize the dangers of malicious applications through accelerators by proxy gaining greater access to data than would otherwise be permissible acting in software alone. Buggy or malicious accelerators that are given carte blanche to access memory (or even hardware resources) could have dire consequences in terms of stability and security of the system.

• Implementing Hardware Protection

In an attempt to gain greater understanding of protection in a reconfigurable computing environment we have developed a hardware mechanism to provide a protective barrier between system memory and hardware accelerators. The accelerator memory management unit (MMU) utilizes a translation look-aside buffer (TLB) to be filled by the operating system and in coordination with the controlling application for the accelerator.

• Protection Models

The accelerators are not restricted to the memory protection scheme enforced by the conventional cores' architecture and the host operating system. There are many research avenues we are currently exploring in the area of protection models for hardware accelerators that can aid in providing a secure, yet fast memory and device protection model.

• The Benefits of Translation

A further benefit of protection checking is the ability to do translation of virtual addresses (understood by the user applications) into physical addresses (needed to access devices directly over the bus) in parallel. The need for accelerators to operate using the virtual address space of the controlling application becomes clear when it attempts to access pointer-based data structures. Our goal is to free application developers from the marshalling of data into linear arrays to be accessible by hardware accelerators (and incurring the concomitant latency) .

Current Applications

• Motion Estimation Accelerator

We have implemented a hardware accelerator to speedup motion estimation—a key component for digital video compression. Currently, the motion estimator is accessed using the character device driver interface that Unix systems programmers are accustomed to. Such a model places the responsibility of allocating the accelerator onto the operating system. This model allows multiple users to access the same device in-turn and overlap one thread's execution of non-accelerated code with another thread blocking on a call to the accelerator providing greater concurrency in addition to the benefits of hardware acceleration.

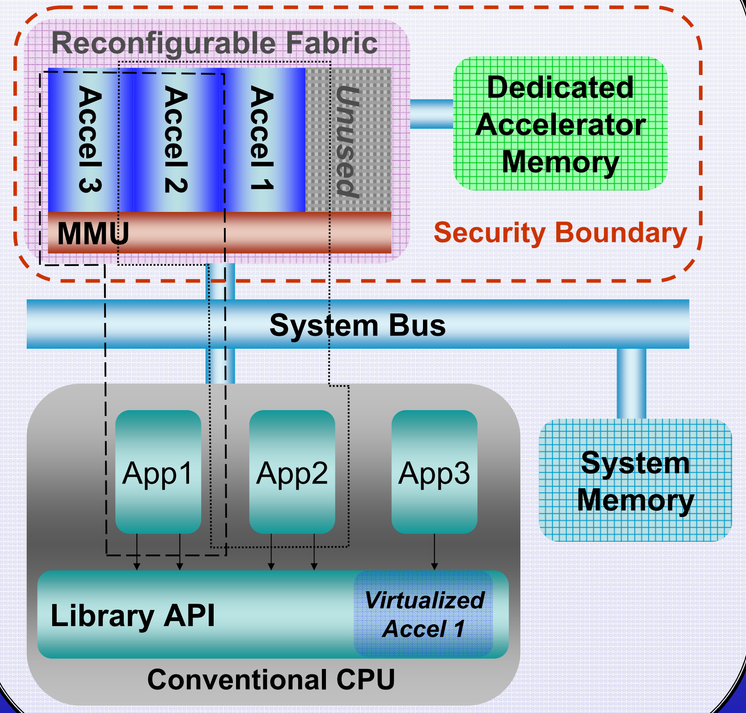
• Audio Dithering Accelerator

We have developed a stereo audio linear dithering accelerator. This accelerator is used by audio applications, like MP3 players. The accelerator implements two buffers to store the samples from the left and right channels and another buffer where the processed audio is stored. These buffers are mapped into the application virtual address space, so it can directly write and read data from them. This model is very close to the one used by exokernels: the operating system is only in charge of granting the access to the accelerator; it is up to the application to correctly use it. One application can share an accelerator with another one by explicitly granting the access to it.

•Clustered HPC Applications

Currently we are exploring hardware acceleration in a clustered environment to provide HPC applications with greater performance and scalability than is possible with a purely conventional architecture.

Proposed Platform



Virtualization:

Separating Application Interface from Implementation

Software vs. Hardware Execution

What metrics should be used to decide whether a section of code should be run on the hardware accelerator or on the conventional core? Having a hardware accelerator/software library call abstraction for interfacing applications with hardware accelerators opens the door to questions regarding how to best utilize the resources available.

• Priority and Resource Sharing

There is a fixed amount of reconfigurable area available to the system, but multiple threads representing different phases of application behavior that may want to take advantage of these resources. To speedup an HPC application, the operating system must provide mechanisms to not only share the accelerators efficiently, but safely while taking into account the priority of the subtasks that request usage.

• Phased Application Behavior

Many HPC applications have a phased behavior that can be managed with runtime reconfiguration of accelerators. However, it may not be advantageous to reconfigure hardware in the event of a phase change. Profiling or hardware feedback can be used by the operating system and compiler to judge what to map into the reconfigurable substrate.

• Evolving Reconfigurable Architectures

The abstraction provided by a library interface between HPC applications and hardware resources has the benefit of allowing the reconfigurable substrate to evolve. Current applications should reap the benefits of faster, more numerous reconfigurable and hardwired resources that will inevitably become available to the library and accelerator developers. In an atmosphere where no reconfigurable architecture standards exist, research efforts must remain flexible enough in their designs to allow for changing hardware models. Virtualization of resources through software execution and hardware mapping techniques is key to providing portability and backward/forward compatibility of reconfigurable hardware accelerators.

Debugging

• New HW/SW Development Tools

Current HDL development tools used to create accelerators are not designed with hardware accelerators in mind nor for use by HPC application developers. By providing a defined interface with which to communicate with accelerators it now becomes possible to build tools that allow for rapid development and debugging by the applications developers that may not be hardware-design savvy.

• Side-Effects of HW Accelerators

When an HPC application crashes, what happens to its accelerator? Debugging an accelerator is not as easy as loading its core file a debugger. How much of that state is visible to the running application for the purpose of debugging the software application? Is the problem with the application actually a flaw in the accelerator? These are some of the questions we hope to answer with our current research foci.

• Accelerator Interface Debugging

The interface provided by wrapping calls to a hardware accelerator in library function calls allows for concrete reasoning with respect to error conditions as opposed to more integrated accelerator approaches. Furthermore, by incorporating virtualization of the hardware resources directly into the programming model, allows application development without access to incomplete or prohibitively expensive reconfigurable platforms to develop applications incorporating hardware accelerators.