Hybrid CPU/FPGA Performance Models

Angshuman Parashar⋆ Michael Adler⋆ Michael Pellauer† Joel Emer⋆†

⋆ VSSAD
Intel Corporation
Hudson, MA 01749
{angshuman.parashar, michael.adler, joel.emer}@intel.com

† CSAIL
Massachusetts Institute of Technology
Cambridge, MA 02139
{pellauer, emer}@csail.mit.edu

Introduction

Pipeline parallelism is an inherent characteristic of structural performance models, which makes them well-suited for mapping onto FPGA substrates. This observation, along with the increasing difficulty being faced by traditional software performance models in simulating future processor designs at reasonable speeds, has fueled a flurry of research activity in recent years on FPGA-based emulation and performance modeling [6, 4, 5, 1].

Implementing detailed industry-grade FPGA-based models and deploying them on simulation farms, however, is a non-trivial task. Such models typically require hundreds of man-hours of development effort, and deploying the models involves providing them access to network-mounted benchmark traces and analysis tools. These tasks are harder to accomplish for FPGA-based designs due to the absence of standardized abstractions that have traditionally been available to software programmers, such as file systems, device drivers and communication protocols.

Due to these reasons, *hybrid* performance models – with the simulator application partitioned between hardware implemented on an FPGA and software running on a host CPU – are more interesting for real-world applications [4, 5, 1]. The host software can not only handle tasks such as loading the benchmark and printing statistics, but also assist in performance modeling by simulating uncommon events like disk accesses. Regular server systems can be augmented with off-the-shelf add-on FPGA boards to create platforms where such hybrid infrastructures can be hosted.

We present in this paper a new framework for building hybrid performance models, with a focus on providing an adaptable and easy-to-use interface to the end user (the model developer). The key component of this framework is a layered protocol stack used for communicating between the FPGA and the host CPU, which exposes a client-server request-response interface to the user. We provide tools to automatically generate typed client and server stubs based on user-specified service interfaces. These stubs provide a convenient abstraction to model writers, enabling them to communicate between the FPGA and the CPU without having to concern themselves with the details of the underlying physical platform. The layers in our protocol stack have been carefully designed to maximize code reuse across multiple FPGA/CPU physical platforms while exposing the same uniform interface to the end user.

Remote Request/Response

Overview

We propose a protocol called *Remote Request/Response* (RRR) that model developers can use to communicate between the FPGA and CPU partitions of a hybrid performance model. RRR is similar to Remote Procedure Calls (RPC) [2] in that *servers* export their functionalities to the outside world in the form of *services* that *clients* can connect to and make use of. In our RRR paradigm, a client addresses a server using a static, well-known service name. The client makes a request to the server by calling a method that is part of the service interface exported the server. The server processes the request, and can optionally send a response back to the client.

Specification Language

RRR service interfaces are described by the model developer using a custom specification language. An example service specification is shown in Figure 1. In this example, we define a service called ISA_EMULATOR, which has a server each on the CPU and on the FPGA. The *"fpga"* server exports the method *UpdateRegister* and allows the *"cpu"* client to make requests to it. The *"cpu"* server exports the methods *Sync* and *Emulate*, and allows the *"fpga"* client to make requests to it.

```
service ISA_EMULATOR
{
   server fpga <- cpu {
       method UpdateRegister(in REGINFO rinfo);
   };

   server cpu <- fpga {
       method Sync(in REGINFO rinfo);
       method Emulate(in IINFO iinfo,
                      out IADDR newPc);
   };
};
```

Figure 1: An Example RRR Service Specification.

Stub Generation

The set of RRR service specifications belonging to a hybrid model are collected together during the model's build process and parsed by a script which generates (a) a set of unique serviceIDs, one for each service in the model, and (b) a set of client and server *stubs*, which abstract away the details of the communication channels in the RRR runtime subsystem and provide convenient typed interfaces to the end-user client and server code. A sample FPGA-side client stub interface generated in Bluespec System Verilog (BSV) [3] is shown in Figure 2. Invoking the method `makeRequest_Emulate()` in this example will result in the generation of a message that travels through the RRR communication layers (described in the next section), culminating in the invocation of a method in the corresponding software server stub on the CPU.

Implementation

The RRR communication subsystem is implemented as a set of layered interfaces that were designed to maximize portability and code

```
interface ClientStub_ISA_EMULATOR;
   method Action makeRequest_Emulate(IINFO iinfo);
   method ActionValue#(IADDR) getResponse_Emulate();
endinterface
```

Figure 2: An Example Auto-generated Client Stub.

reuse across different physical platforms on which FPGAs could be used as compute nodes. This layered hierarchy is shown in Figure 3. The protocol operates as follows:

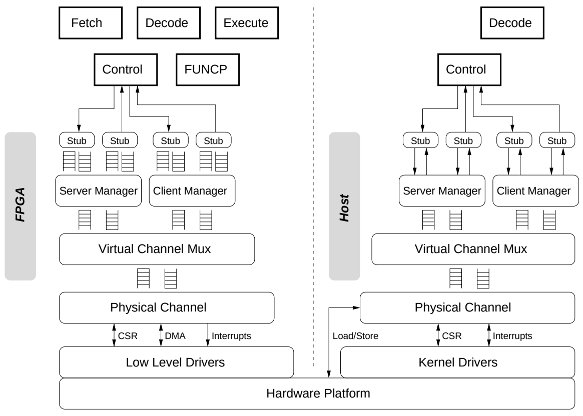

Figure 3: RRR Protocol Stack.

1. The lowest layer in the hierarchy comprises of the low-level drivers used to drive the physical medium (such as a bus) that can be used to communicate between the FPGA and the host processor.

2. Using these drivers, we build a full-duplex ordered FIFO abstraction between the two compute nodes. We call this the Physical Channel. This abstraction can be implemented in multiple ways using the functionalities provided by the set of platform device drivers. For example, our current implementation uses a set of coherent CSRs to create a pair of circular FIFOs. The drivers and the Physical Channel are the only platform-dependent layers in the hierarchy.

3. Next, we multiplex the Physical Channel into multiple full-duplex Virtual Channels. Messages traversing each virtual channel are ordered with respect to other messages in the same channel, but no ordering is guaranteed across channels.

4. The next layer is occupied by a Client Manager (which is responsible for arbitrating between multiple client requests, and dispatching responses to the appropriate client) and a Server Manager (which dispatches requests to the appropriate server, and arbitrates responses). An RRR call path is established by connecting a client manager one end of virtual channel and a server manager to the other end.

5. Automatically-generated RRR stubs talk to the client and server managers to handle requests and responses.

Applications: HAsim Hybrid Modules

We developed the RRR framework as a part of the HAsim [6] simulation infrastructure. For hybrid HAsim models, we partition logic between the CPU and the FPGA at the granularity of a *module*. Models are created from a collection of pure-software (CPU), pure-hardware (FPGA) and hybrid hardware/software modules. A pure software module interacts with the software side of a hybrid module and a pure hardware module interacts with its hardware side. Internally, the hybrid module communicates with its counterpart residing on the opposite compute node via RRR, using an interface that it declares in an RRR specification file. Figure 4 shows an example.

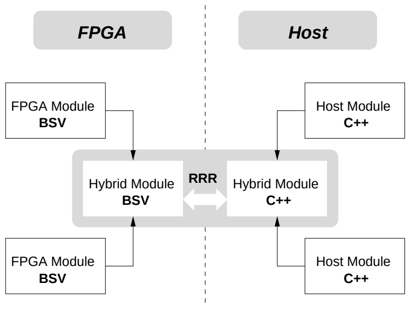

Figure 4: A HAsim Hybrid Module.

RRR has been used extensively in several hybrid HAsim performance models. CPU-to-FPGA RRR services are generally used for controlling the state of the simulation and for invalidating or updating stale state cached on the FPGA. FPGA-to-CPU services are used for printing events and statistics, triggering assertions when the FPGA encounters an error, emulating system calls, and obtaining access to host system memory. Several of these functionalities (such as control, assertions and memory access) could prove useful for general-purpose hybrid applications as well.

Concluding Remarks

We presented a general, flexible, cross-platform infrastructure for building hybrid CPU/FPGA applications. Taking inspiration from Remote Procedure Calls, our communication framework allows developers to create modules that communicate using user-defined interfaces and types via automatically-generated client and server stubs. We are actively using the infrastructure for the development of hybrid performance models as part of the HAsim project, but we believe our tools and techniques are general enough to be useful for a wider variety of FPGA-based applications.

References

[1] Arvind, K. Asanovic, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, D. Patterson, J. Rabaey, and J. Wawrzynek. Ramp: Research accelerator for multiple processors - a community vision for a shared experimental parallel hw/sw platform. Technical report, 2005.

[2] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, 1984.

[3] Bluespec, Inc. Bluespec system verilog reference guide. *http://www.bluespec.com/*, 2007.

[4] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat. Fpga-accelerated simulation technologies (fast): Fast, full-system, cycle-accurate simulators. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, 2007.

[5] E. S. Chung, E. Nurvitadhi, J. C. Hoe, B. Falsafi, and K. Mai. A complexity-effective architecture for accelerating full-system multiprocessor simulations using fpgas. In *FPGA '08: Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, 2008.

[6] M. Pellauer, M. Vijayaraghavan, M. Adler, Arvind, and J. Emer. Quick performance models quickly: Timing-directed simulation on fpgas. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2008.