Abstraction Library To Simplify Application FPGA Configuration And Control

Craig P. Steffen
Innovative Systems Laboratory
National Center for Supercomputing Applications
University of Illinois at Urbana/Champaign
csteffen@ncsa.uiuc.edu

NCSA's Innovative Systems Laboratory[1] (ISL) has created an FPGA bitfile and interface metadata storage and control library to manage bitfiles for FPGA computing. The initial version of this system has been created to control a Nallatech H101 PCI-X Virtex-4 HPC accelerator card, but the framework is expandable and can encompass multiple execution environment with with modular upgrades.

In 2008 and 2009, the ISL has created a template library to abstract away some of Nallatech[2]'s FUSE library calls from the applications programmer. This library is internally referred to as the NCSA Nallatech Interface, or Nnalli library. The functions of the nnalli library hide the bookkeeping aspects necessary for use with the direct FUSE interface functions and abstract anything not needed by the programmer into the library's internal state. These functions make the Nallatch FPGA control system look more like a mathematical library to the applications programmer, who then calls initialize routines once, and functions that move data into and out of the FPGA memory space, and control configuration and execution of the FPGA processor. The Nnalli libarary is a convenience tool, saving the programmer from bookkeping tasks required by the FUSE library. Functions in the Nnalli library include (with arguments left out for brevity):

```
nnalli_init_next_card()
nnalli_run()
nnalli_wait()
nnalli_write_integer_scalar_args()
nnalli_write_4byte_values()
nnalli_write_8byte_values()
```

Craig Steffen taught a one-week course on reconfigurable programming at the Center for High Performance Computing at Cape Town, South Africa, at the invitation of Mike Inngs of the University of Cape Town, in December of 2008. The first generation Nnalli library, as currently used by ISL internally, we the framework for teachign the class. Parallel FPGA programming was the focus of the course; the nnalli library provided the codeing framework so the students could concentrate on aspects of parallelization. However, the process of setting up a new piece of code for the library (three different major problem types were attempted through the week) was an obstacle to the students getting things set up quickly.

The central issue with this approach, which is a broader one for accelerator technologies used in large application codes, is that the bitfiles are opaque entities, and the libraries to control, configure, and move data to and from are separate from the application. Somehow, an association must be made when the program runs to attach the proper bitfile and link it into the program. The FUSE library and Nnalli build system does this by compiling the Nallatech's network.h file into one of the object files that controls the FPGA. This has two specific problems. One is that the location of the file is compiled into the executable. If the program is moved onto another computer, the information must be updated and the file compiled again. The second problem is that each version of the executable has one and only one version of the FPGA processor image locked into it. One executable cannot be used to compare different solutions to a problem, or to use two different FPGA processing solutions.

This explanation of our work is neither meant to hold up Nallatech as a good nor bad example of interface software. The Nnalli libraries were specifically developed work within Nallatech's software design system. Our new libraries are also being developed for running Nallatech H101 cards, but we will definitely be exapanding this work to encompass other hardware systems. This paper is staying with discussing building software infrastructure for use with Nallatech's hardware in an effort to stay with a specific example of a hardware and software stack that controls an accelerator system, as an example of possible other such systems.

Our current work is a system that improves the above-mentioned problems. We have created a container file format with associated programs and libraries that embeds a bitfile into a larger ACcelerator File Format file (ACFF file) that also contains information about what FPGA device the bitfile configures, and contains in a machine readable format all the associations between buffer names and FUSE-style addresses. That is, it contains an opaque executable image (bitfile) and the run-time information to connect that library to the running application. The user points the application at the proper bitfile at runtime. The run-time library (that's compiled into the executable) unpacks the bitfile and the other metadata from the .acff file. The bitfile is loaded into the FPGA using FUSE as normal (except that its location is determined at application run time) and the meta-data is used to initialize the FUSE libraries so that the data is transported to the right place in the

FPGA.

Embedding the bitfile is obviously not a new idea. SRC Computers Inc.[3], of Colorado Springs, Colorado, have always had their bitfiles integrated with their executables. Their purpose is to have a tightly-integrated invisible software-FPGA interface, and their interface does that well. Our goal with the acff libraries is to enable one application (that's not hardware specific) to be able to use different, perhaps new hardware easily without recompiling or re-tooling.

This library was created for convenience of moving the bitfiles around and being able to test new images against old images directly. However, the implications of this approach are far more profound. This file storage system and support libraries create and enforce a method of dividing execution of an application between what gets processed in the main CPU and what can be processed on an accelerator. Formalizing the dividing line around the accelerator modularizes this approach and leads to software that can use differnet accelerators. The current software can use one set of compiled software but have the calculations performed by different FPGA images. Once we create container types for different types of FPGA images, the same executable code can operate using totally different technologies enabled in the container class, without the application knowing the difference.

The idea of this work comes out of very recent discussions with Mike Inggs and students at CHPC in South Africa about streamlining software compiling for FPGA acceleration. This code is very new, and is being developed day by day. The short-term to-do list (much of which will running in some form before July) will include: a machine-computable description of the algorithm and input and output data channels to verify that the image in the embedded file is appropriate for the application that is calling it a repository for .acff files so that an application running on a machine can search for appropriate helper code a test harness that will take a piece of application code with the appropriate acff library calls and test a grouping of accelerator files and coallate the test results

The acff libraries are built to be expandable. The basic library, which will largely remain static, contains functions to pack and unpack acff files. For each hardware/software stack combination (H101 + nnalli + FUSE, for instance) someone must write the packing and unpacking routines for the bitfiles and necessary metadata. From then on, porting a new application merely means modifying the application to be accelerated to have the proper acff library calls to pack the data together and initiate execution of the accelerating processing.

REFERENCES

[1] 2009, the Innovative Systems Lab is a group with the National Center for Supercomputing Applications, which is a division of the University of Illinois at Urbana/Champaign. [Online]. Available: http://www.ncsa.illinois.edu/AboutUs/Directorates/ISL.html

[2] 2009, Nallatech Inc., Glasgow, Scotland. [Online]. Available: http://www.nallatech.com

[3] 2009, SRC Computers LLC, Colorado Springs, Colorado, USA. [Online]. Available: http://srccomp.com/