Application Development for Hybrid Pipelined Systems

Mark A. Franklin*, Patrick Crowley*, Roger D. Chamberlain*, Jeremy Buhler*, and James H. Buckley†
*Department of Computer Science and Engineering
†Department of Physics
Washington University in St. Louis
jbf@wustl.edu, pcrowley@wustl.edu, roger@wustl.edu, jbuhler@wustl.edu, buckley@wustl.edu

Introduction

Many embedded data processing applications can be characterized by their pipelined structure. The output from one stage of the computation is forwarded to the input of a successor computational stage. Examples include signal processing, biosequence search, encryption, and others. The input data might originate from signal acquisition subsystems or large, disk-based data stores [1,2].

For performance reasons, it is often desirable to deploy these pipelined applications on physically pipelined computing infrastructures. In addition, the computing resources themselves can have a hybrid structure, with a combination of general-purpose processors and reconfigurable logic (e.g., FPGAs) available.

Given the availability of a hybrid computing infrastructure to enable the execution of a pipelined application, we now must consider what the available and appropriate deployment options are. Mapping the application to the computational resources and choosing the "best" resource for each computational pipeline stage are decisions that must be guided by good performance estimates of significant metrics (e.g., throughput, latency).

Exploring the Design Space

Figure 1 illustrates the type of design questions that we address. Across the top of the figure is an application that consists of three pipelined computational stages (1 → 3). Across the bottom of the figure is a pair of computing resources (compute platforms 1 and 2). The figure illustrates application stage 1 being mapped to compute platform 1, application stage 3 being mapped to compute platform 2, and a question as to whether application stage 2 should be mapped to compute platform 1 or 2.

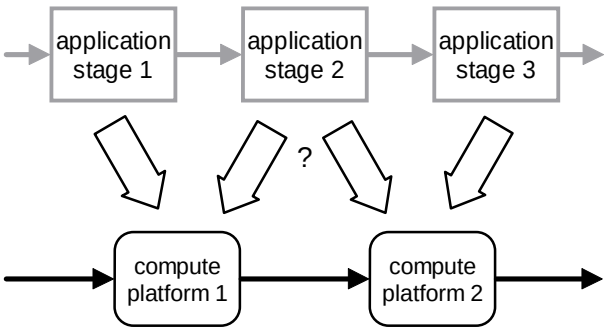

Figure 1: Mapping application to architecture

This work supported by NSF grants CCF-0427794 and DBI–0237902 and NIH/NGHRI grant 1 R42 HG003225–01.

While the figure illustrates a particular design question, a full design problem has many such questions. For example, what technology should be used for compute platform 1 (e.g., processor or reconfigurable logic)? How does this choice impact the mapping question for application stage 2?

Development Environment

We are designing a development environment for pipelined applications that supports a number of hybrid computing platforms. Our target platforms include traditional general-purpose processors (e.g., x86, PowerPC, ARM), chip multiprocessors (e.g., Intel IXP network processor, AMD multi-core Opteron), and reconfigurable logic (e.g., FPGAs).

Application development for a hybrid compute platform starts with an executable specification developed in C/C++ that indicates the algorithm for each application stage and the movement of information between stages. Algorithms for particular application stages are developed using traditional MPI-style `send()` and `rcve()` calls to deliver data between stages of the pipeline. This executable specification is, in fact, directly deployable on a traditional parallel processor. It also serves as the reference system (i.e., the "golden model") for algorithmic verification and validation efforts on the subsequent candidate designs.

For an application pipeline stage that is to be deployed within an FPGA, it is necessary to develop the individual stage's algorithm using a language appropriate for the platform (e.g., VHDL, Verilog, or SystemC). With some chip multiprocessor platforms, specialized algorithm development is also likely to be required. In all cases, however, data into and out of the stage is handled via a common interface specification [2].

By having the data communications in and out of each stage conform to a standard interface specification, the infrastructure provided as part of the development environment can:

(1) connect the application pipeline stages together, independent of which computing platform each stage runs on;

(2) deliver data output from one stage to the appropriate downstream stage; and

(3) empirically measure the performance of individual pipeline stages as well as the application as a whole.

In addition, the infrastructure enables one to build a library of algorithms that can be readily reused.

Emphasis on Performance Evaluation

Given that performance is the primary motivating factor when using hybrid computational platforms, the lack of consideration given to evaluating performance in the vast majority of application development environments is appalling. We take the position that performance evaluation, either of a constructed system on which empirical measurements can be taken or of a candidate system on which performance estimates must be used, is a primary design consideration, second only to correctness, that should be richly supported in the development environment.

In our system, performance evaluation is being supported at a number of levels. Analytic models have been developed that not only predict which stage(s) is(are) a performance bottleneck, but also provide quantitative assessment of stage-to-stage queuing requirements (an often ignored resource that can be critical to overall performance).

Provided simulation models are of two forms. One is a traditional discrete-event simulation toolset that has been designed to simplify the model development process for the pipelined applications of interest. The second is a co-simulation infrastructure that enables developers to concurrently execute the general-purpose processor deployed pipeline stages and simulate (at the RTL level) the reconfigurable hardware deployed pipeline stages.

Finally, the development infrastructure's ability to automatically integrate and deploy application pipelines onto actual hardware platforms dramatically eases the effort required to widen the scope of the design space explored via direct execution and empirical evaluation.

Initial Applications

We are testing our application development environment as it is being built using a set of real-world pipelined applications. These applications have been chosen to be representative of the general class of pipelined applications of interest and have intentionally been selected from a diverse group of user communities.

Our first application consists of signal processing tasks associated with the VERITAS gamma-ray telescope. This NSF/DOE funded project consists of an array of 12 m atmospheric Cherenkov telescopes for gamma-ray astronomy at 50 GeV to 50 TeV energies [3]. The data set consists of images of Cherenkov light flashes from gamma-ray initiated electromagnetic showers.

The algorithmic requirements for this application include a signal processing pipeline that is quite common to many other applications. Our initial pipeline stages include an FFT, a low-pass filter, a matched filter, all followed by an IFFT. These computations are applied to the time series signal from each image pixel. Following this, the resulting individual pixel signals are aggregated and pattern detection is performed searching for signature images.

The second application in the set is 3DES encryption. The 3DES application is a pipeline that consists of 3 copies of the Data Encryption Standard (DES) kernel [4]. In addition to the macro-level pipelining just described, the kernel itself consists of a series of computation stages that can be pipelined.

The third application in the set is the HMMER program for protein sequence analysis using profile hidden Markov models. Here, the primary computation is in the form of a dynamic programming problem [5].

The final application is also from the field of biocomputation. BLAST, the Basic Local Alignment Search Tool [6], is the most widely used software for rapidly comparing a query sequence to a biosequence database. As shown in Figure 2, BLAST is functionally organized as a pipeline with three stages: word matching, ungapped extension, and gapped extension. The inputs to this pipeline are a query sequence and a sequence database. Each stage of BLAST's pipeline implements progressively more sophisticated and more expensive computations to identify biologically meaningful similarities between query and database.

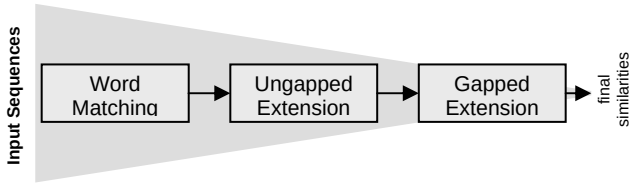

Figure 2: The BLAST application pipeline

Conclusions

We describe an application development environment for pipelined hybrid systems with an emphasis on making performance evaluation a primary design consideration. The development environment supports a number of computational platforms, including general-purpose processors, chip multiprocessors, and reconfigurable logic. The development environment is being tested using a set of real-world, production applications.

References

[1] R. D. Chamberlain, R. K. Cytron, M. A. Franklin, and R. S. Indeck, "The *Mercury* System: Exploiting Truly Fast Hardware for Data Search," in *Proc. of Int'l Workshop on Storage Network Architecture and Parallel I/Os*, September 2003, pp. 65-72.

[2] M. A. Franklin, R. D. Chamberlain, M. Henrichs, B. Shands, and J. White, "An Architecture for Fast Processing of Large Unstructured Data Sets," in *Proc. of 22nd Int'l Conf. on Computer Design*, October 2004, pp. 280-287.

[3] T. C. Weekes et al., "VERITAS: the Very Energetic Radiation Imaging Telescope Array System," *Astroparticle Physics*, **17**:221-243, 2002.

[4] ANSI X9.52-1998, *Triple Data Encryption Algorithm Modes of Operation.*

[5] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, 1998.

[6] S. F. Altschul et al., "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs," *Nucleic Acids Research*, **25**:3389-3402, 1997.