A programming model for developing application specific dataflow machines on FPGAs

A programming model for developing Application Specific Dataflow Machines on FPGAs

Nick Brown
EPCC at the University of Edinburgh
Edinburgh, UK
n.brown@epcc.ed.ac.uk

I. Introduction and background

FPGAs have become popular in many fields but are yet to gain wide acceptance in High Performance Computing (HPC) for accelerating scientific or engineering simulations. Whilst there are numerous on-going activities exploring the role of FPGAs for such workloads, often using HLS which enables programming in C or C++, significant challenges remain for scientific software developers to achieve performance with reconfigurable architectures. The underlying issue is that HLS presents a Von Neumann based programming model which is inappropriate for FPGAs, resulting in a significant disconnect between the semantics of existing HLS-based languages, and how experienced FPGA programmers must write their dataflow codes to best exploit the hardware.

It is our hypothesis that abstractions which are entirely suited towards dataflow, and without any Von Neumann baggage, will more effectively enable the programming of high performance codes on FPGAs in a productive manner.

II. Application Specific Dataflow Machine (ASDM)

Like how the imperative language programmer has a Von Neumann execution model, the FPGA programmer should have an explicit dataflow execution model provided to them by the language. We propose an *Application Specific Dataflow Machine (ASDM)* which is focused around building custom computing machines on an application by application basis. In this model programmers organise their code as a set of concurrently running *filters*, each consuming streams of input data and, based upon their code, transforming this to a stream of output data. The language and its associated abstractions should present this model to the programmer as the first-class concern so that they naturally encode their algorithms in such a form to ensure that the only way of writing correct dataflow code is also one that is fast-by-construction.

III. Lucent: A vehicle for dataflow abstractions

We have developed the Lucent dataflow language which, following the declarative paradigm, enables programmers to abstractly structure their algorithm to follow the ASDM execution model and suit dataflow. Based around ideas first presented in Lucid [1], the declarative nature of our language means that there is a significant amount of high level information upon which the compiler can operate to infer most appropriate implementation choices. Furthermore, making time a first class concern is also highly beneficial as this focuses the programmer in considering the evolution of data from one point in time to the next.

```
1  external filter mykernel:int(a:int) where:
2      mykernel=if (a < 10) then 1 fby 2 else 0 fi
```

Listing 1: Lucent example illustrating key ASDM concepts for a simple filter which streams out computed data

Listing 1 briefly illustrates our approach where *mykernel* is a single dataflow filter, with an input and output. There is no such thing as a variable in Lucent (as that indicates data at rest, and at rest it is not making progress). Consequently *a* is an input streams of integers, and *mykernel* is an output stream of type integer. The expression at line 2 declares that the value of the output stream at any point in time (what we call a *time quantum*) is 1 followed by 2 if the value of the *a* stream at the specific time quantum is less than ten, otherwise the value streamed out is 0. As an example of considering the progression of time, the followed by (*fby* operator) declares that a value is followed by another value next time quantum, for instance *myseq:int = 0 fby 1 fby 2* defines the *myseq* sequence to be the values 0, 1, and 2 respectively as Lucent time progresses (and 2 infinitely after this point). Our compiler currently supports the Xilinx Alveo family of FPGAs and there are numerous additional abstractions and constructs in Lucent which brevity means we are unable to highlight in this extended abstract but can be explored with documentation and examples at the repository, ***https://github.com/mesham/lucent***.

IV. Summary

Our approach is to provide the ASDM execution model which enables programmers to conveniently map their algorithms to the dataflow paradigm and consider, as a first class concern, the progression of time, consumption of data and generation of results. Using abstractions presented in the Lucent declarative language, there is a rich amount of information which can then be used by the compiler to make appropriate choices when mapping to the underlying hardware.

References

[1] LUCID, the dataflow programming language, W. W. Wadge and E. A. Ashcroft. Academic Press Professional, Inc., USA. 1985.