Journal of Systems Architecture 60 (2014) 592–614

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

A framework for post-silicon realization of arbitrary instruction extensions on reconfigurable data-paths

Saptarsi Das [a,*], Kavitha Madhu [a], Madhav Krishna [a], Nalesh Sivanandan [a], Farhad Merchant [a], Santhi Natarajan [a], Ipsita Biswas [a], Adithya Pulli [c], S.K. Nandy [a], Ranjani Narayan [b]

[a] CAD Laboratory, Indian Institute of Science, Bangalore, India
[b] Morphing Machines Pvt. Ltd, Bangalore 560055, India
[c] T.U. Delft, Delft, Netherlands

ARTICLE INFO

Article history:
Received 10 January 2013
Received in revised form 15 June 2014
Accepted 18 June 2014
Available online 8 July 2014

Keywords:
Reconfigurable computing
Instruction extension
Architecture exploration
Hardware data-path
Application acceleration

ABSTRACT

In this paper we present a framework for realizing arbitrary instruction set extensions (IE) that are identified post-silicon. The proposed framework has two components viz., an IE synthesis methodology and the architecture of a reconfigurable data-path for realization of the such IEs. The IE synthesis methodology ensures maximal utilization of resources on the reconfigurable data-path. In this context we present the techniques used to realize IEs for applications that demand high throughput or those that must process data streams. The reconfigurable hardware called HyperCell comprises a reconfigurable execution fabric. The fabric is a collection of interconnected compute units. A typical use case of HyperCell is where it acts as a co-processor with a host and accelerates execution of IEs that are defined post-silicon. We demonstrate the effectiveness of our approach by evaluating the performance of some well-known integer kernels that are realized as IEs on HyperCell. Our methodology for realizing IEs through HyperCells permits overlapping of potentially all memory transactions with computations. We show significant improvement in performance for streaming applications over general purpose processor based solutions, by fully pipelining the data-path.

1. Introduction

Extensible processors [1,2] are a class of processors that can be configured and/or extended. Extensible processors are key components of MPSoCs. MPSoCs are increasingly being used for embedded applications that demand both high performance and throughput in small form factor low power devices. One of the methods to achieve this is to accelerate application kernels by executing them as Instruction Extensions (IEs) on custom data-paths. An IE is a bundle of basic instructions that is executed as a single macro operation with multiple inputs and multiple outputs (MIMO) [3]. The relative ordering of basic instructions that constitute an IE is fixed and is not elaborated upon in the definition of the IE. IEs are used to replace code sequences that are either hot-spots or sequential bottlenecks. IEs also offer significant savings in energy consumption over pure software implementation using general purpose processors by eliminating fetch, decode and memory/register-file transactions for the individual basic instructions that constitute the IEs. Broadly, there are two ways in which IEs are realized:

- IEs are determined for a domain of applications through extensive application profiling and analysis of hot-spots. The fixed ordering of basic instructions that comprise an IE necessitates fixed hardware data-path for its implementation. Such fixed data-paths are realized as Custom Functional Units (CFUs) and/or custom data-paths [1,4]. Some of the prominent techniques for automatic synthesis of application-specific IEs are presented in [5–8]. High NRE cost associated with the realization of such CFUs cannot be avoided. This drawback renders this approach inapplicable in scenarios where flexibility in terms of application scalability, interoperability are desirable objectives.
- IEs may be defined post-silicon. This entails synthesis of MIMO macro operations on the same hardware data-path. Since IEs originate from different application domains system designers may conduct design-space exploration with various choices of

* Corresponding author.
E-mail addresses: sdas@cadl.iisc.ernet.in (S. Das), kavitha@cadl.iisc.ernet.in (K. Madhu), madhav@cadl.iisc.ernet.in (M. Krishna), nalesh@cadl.iisc.ernet.in (N. Sivanandan), farhad@cadl.iisc.ernet.in (F. Merchant), santhi@cadl.iisc.ernet.in (S. Natarajan), ipsita@cadl.iisc.ernet.in (I. Biswas), a.pulli@student.tudelft.nl (A. Pulli), nandy@serc.iisc.ernet.in (S.K. Nandy), ranjani.narayan@morphingmachines.com (R. Narayan).

http://dx.doi.org/10.1016/j.sysarc.2014.06.002
1383-7621/

IEs without having to redesign hardware data-paths for each individual choice of IEs. Thus post-Silicon definition of data-paths for IEs can mitigate the high NRE costs associated with CFUs and enable support for application scalability and interoperability without compromising on performance. This approach can be classified into two categories.

- Data-paths for IEs are synthesized from their Hardware Description Language (HDL) descriptions on FPGA like substrates. Examples of this approach include Molen Polymorphic Processor [9], Convey Hybrid-Core Computer [10]. One major shortcoming of this approach is the necessity of HDL programming to specify the IEs.
- Data-paths for IEs are synthesized from High Level Language (HLL) descriptions. Platforms such as Reconfigurable Custom Functional Unit (RCFU) proposed by Murakami et al. [11,12] and the Dyser-fabric [13] based architecture of accelerator proposed by Benson et al. [14] use HLL based description of IEs.

From the above classification it becomes clear that post-Silicon IEs synthesized using HLLs are going to be part of *future-proof* embedded solutions due to their flexibility and ease of use.

Implementation of such IEs require a reconfigurable hardware data-path supported by an automatic IE synthesis. Note that reconfigurable data-paths are usually designed as 2-D structures comprising array of FUs/ALUs and some form of programmable interconnect. The constituent basic instructions in an IE are realized on the FUs/ALUs. Communication between basic instructions resulting from the relative ordering of basic instructions pertaining to different IEs are realized using the programmable interconnect. In literature, there exists a large number of Coarse Grain Reconfigurable Architectures [15–17]. In the present exposition we discuss only those coarse grain structures that are used to synthesize IEs [11,12,14]. These solutions do offer reduced energy footprint for compute-intensive applications through IEs. However, achieving high throughput necessitates execution of IEs by staging many dynamic instances of the same IE in a pipeline. Note that there may exist dependences across dynamic instances of an IE. Such dependences prevent initiation of dependent instances of IEs in close succession. However, this problem may be circumvented by interleaving execution of different instances of IEs that are independent. HyperCell, as a data-path element for IEs as presented in this paper supports a micro-architecture for pipelined execution of IEs. In the following we quantitatively establish the benefits of pipelined executions of IEs.

Consider a DFG $G(V,E)$[1] corresponding to an IE. Let us assume that the DFG has n vertices corresponding to n elementary (dyadic/monadic) instructions and the critical path[2] delay is τ units. In Table 1 we present a comparison of execution time of the DFG $G(V,E)$ on three different platforms. When implemented on a single FU/ALU, the execution time[3] of the DFG is n, the number of vertices in the DFG. When implemented on a linear row of w FUs, the execution time has a lower bound of $\frac{n}{w}$. Note that in the linear row of FUs there are w FUs available in the data-path and we achieve up-to w-times speedup over a single FU. Therefore improvement in performance is commensurate with the increase in hardware complexity of the data-path. On the other hand, as we migrate to a reconfigurable hardware comprising array of n FUs, the improvement in performance is incommensurate with increase in hardware complexity. Implementation of the DFG $G(V,E)$ on such a reconfigurable hardware data-path incurs execution time proportional to τ. Therefore, with an n-times increase in number of FUs we achieve only a n/τ-times improvement in performance. This is largely due to the fact that at any instant of time, only a small subset of resources are utilized for computation while the remaining resources starve for data. We refer to this aforementioned phenomenon of incommensurate improvement in speedup with increase in hardware complexity as *Speedup-gap*. Speedup-gap can be defined as the ratio of relative hardware complexity and speedup.

Note that the relative hardware complexity is measured by comparing the number of ALUs/FUs in different platforms. We make an important observation from Table 1. The speedup-gap encountered in implementing IEs on 2-D reconfigurable data-path is caused by long-latency sequential paths in the DFGs of the IEs. Bridging the speedup-gap between reconfigurable data-paths and general purpose processor based implementations require exploitation of both spatial and temporal parallelism to increase hardware utilization of the reconfigurable data-paths to 100%. Realization of systolic algorithms on systolic architectures are well known examples of architectures that exploit both spatial and temporal parallelism in applications [18]. However it should be noted that systolic architectures are not very suitable for realizing data-paths for IEs of diverse nature. If we can imitate the behaviour of systolic architecture by supplying the reconfigurable hardware with different sets of data corresponding to different instances of G every cycle, then the problem of speedup-gap can be alleviated. In that case, the effect of the long latency sequential parts of an application get amortized through pipelining of many dynamic instances of G. If N instances of the graph are executed exploiting both spatial and temporal parallelism, the total execution time on hardware is $\tau + (N-1)$. Execution time on linear FU array is $\frac{Nn}{w}$ and on single FU is Nn. Therefore speedup achieved through hardware implementation of G with respect to single FU is, $\frac{nN}{\tau+N-1}$ which becomes close to n, when N becomes very large. The speedup is proportional to the number of vertices of DFG realized on hardware. Similarly speedup with respect to linear FU array of issue-width w is $\frac{Nn}{w(\tau+N-1)} \simeq \frac{n}{w}$, which is the ratio of number of FUs/ALUs in the reconfigurable hardware data-path and the linear FU array. In general, if the initiation interval of dynamic instances of the IE is Δ cycles[4], then the total execution time of N dynamic instances is $\tau + (N-1)\Delta$. Thus the speedup-gap with respect to single FU can be quantified as $\frac{n(\tau+(N-1)\Delta)}{nN}$ and the speedup-gap with respect to linear FU array can be quantified as $\frac{n(\tau+(N-1)\Delta)w}{wnN}$. For large enough N, the speedup-gap approaches Δ, the initiation interval. Therefore minimization of speedup-gap necessitates execution of many dynamic instances of the IE in a pipelined manner and minimization of initiation interval of the successive dynamic instances.

Executing many dynamic instances of an IE in a pipelined manner is akin to executing kernels on fixed-function hardware, where the operations remain fixed in space and input sets corresponding to different dynamic instances are streamed into the hardware. Streaming kernels[5] such as FFT, FIR are naturally the most suitable candidates for execution on such hardware structures comprising large number of FUs/ALUs since they can be easily expressed as successive dynamic instances of the same DFG. However, applications with very high degree of parallelism can also be partitioned into parallel kernels which in turn can be realized as IEs in a streaming

[1] Since each IE contains a set of basic instructions, the IE can be represented as a Data Flow Graph $G(V,E)$, where the set of vertices V in the graph represents the set of basic instructions and the set of edges E represents set of dependences between the said basic instructions.

[2] Critical path of a data flow graph is the longest path from one of the root nodes to one of the leaf nodes.

[3] In this example we assume all the operations to be of same latency. This simplifying assumption is made to keep the comparison simple. A more realistic example is presented in Table 3.

[4] The latency of schedule between successive dynamic instances of the same IE is the initiation interval [19].

[5] Streaming kernels are characterized by four major characteristics, viz. self-containment and independence of kernels, static nature of the computation groups, explicit definition of communication and limited lifetime of the stream data [20].