Computers and Electrical Engineering 40 (2014) 1199–1214

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Fast description and synthesis of control-dominant circuits ☆

Marc-André Daigneault, Jean Pierre David *

Department of Electrical Engineering, Ecole Polytechnique de Montreal, Montreal, Canada

ARTICLE INFO

Article history:
Available online 22 March 2014

ABSTRACT

General purpose processors, graphics processing units (GPUs) and field-programmable gate-arrays (FPGAs) compete and collaborate to offer ever increasing performances. Nevertheless, despite fruitful decades of research, FPGA are still a lot more difficult to exploit than processor-based approaches. It is today possible to automatically map C/C++/SystemC algorithms into circuits. However, exploiting fine grain parallelism for control dominant applications is still reserved to highly specialized people in hardware design. This paper presents the application of our synchronized-transfer-level hardware design methodology to the implementation of pipelined floating point operators. The methodology builds on a hardware description language for which the designer manages dynamic connections between data token sources and sinks. A compiler automates the generation and the optimization of the synchronization logic, whose low-level complexity is thus hidden to the designer. Applied to the design of a floating-point matrix multiplication hardware accelerator, the proposed methodology leads to similar computing performances than the dedicated designs reported in the literature but within shorter design times (hours instead of days), simpler source code and no need for advanced hardware design skills.

1. Introduction

A famous debate between Gene Amdahl and Daniel Slotnick on the feasability of parallel computing dates back to as far as 1967 [1,2]. Nowadays, as the performance of single-threaded processors have stopped following Moore's Law, multi-core processors have become a commodity, and the spectrum of high-performance parallel computing devices has never been so colourful. Initially aiming at delivering jaw dropping 3D graphics, the Graphics Processing Units (GPUs) present in state-of-the-art video cards are increasingly used in the most advanced scientific applications, offering performances in the order of teraflops [3]. Field-Programmable Gate-Arrays (FPGAs) can also leverage on the billions of transistors that are made available through state of the art integrated circuit (IC) fabrication processes, and are still riding on – what's left of – Moore's Law. Having evolved from glue-logic to processing devices of their own, modern high-end FPGAs include hundreds of hard DSP and RAM memory blocks, tens of thousands of registers, sometimes hard-processors, and have thus gradually narrowed the gap separating their performance with those of ASICs [4]. Unlike the ASIC however, an FPGA can be reconfigured in a matter of minutes to implement any hardware design (*fitting* the device), possibly including configurable processors and their programs.

Recently, the Basic Linear Algebra Subroutines (BLAS) library has been implemented on three different platforms [5]: an Intel Core 2 Duo E8500 processor, an Nvidia Tesla C1060 GPU, and a Virtex5 FPGA (inside a BEE3 platform). Results show that

☆ Reviews processed and recommended for publication to Editor-in-Chief by Associate Editor Dr. Rene Cumplido.
* Corresponding author.
E-mail address: JPDavid@polymtl.ca (J.P. David).

http://dx.doi.org/10.1016/j.compeleceng.2014.02.011
0045-7906/© 2014 Elsevier Ltd. All rights reserved.

the FPGA implementation provides the best energy efficiency in terms of GFLOPS per Watt, while the GPU is the most power hungry. In the field of matrix multiplication, high-performance GPU implementations will deliver as much as 393 GFLOPS [6]. The FPGA implementations of matrix multiplication proposed in [7–13] can deliver up to 29.8 GFLOPS. As the world goes more mobile and becomes more conscious of its impact on the environment, energy efficiency can be a strong driving force to the adoption of FPGAs for processing and co-processing units [14]. Nevertheless, while concurrent programming is becoming the de-facto approach to fully-exploit the potential of state of the art programmable ICs, programming and debugging hardware remains inherently harder than software.

Despite important advances in the field of high-level C-synthesis, most hardware designs are still described with Register-Transfer-Level (RTL) languages such as VHDL and Verilog. However, such abstraction level is simply too low to handle the millions and billions of transistors available in modern ICs. In this work, we consider a beyond-RTL abstraction level that considers assignments as transfers between data-synchronized sources and sinks, instead of simple register transfers. Using implicit predefined synchronization interfaces similar to Xilinx's AXI4-Stream and Altera's Avalon Streaming interfaces, such *Synchronized-Transfer-Level* (STL) abstraction frees the programmer from the tedious task of handling and specifying low-level synchronization control logic. Data transfers occur between connected source and sink when both are ready to send/receive.

Our hardware description language (HDL) builds on the CASM language [15], which proposes that Finite State Machines (FSMs) handle dynamic connections between data token sources and sinks. The proposed STL abstraction adds a constraint programming paradigm to allow the specification of logical rules constraining the authorization of data transfers over different connections [16]. Thanks to those authorization rules, it is possible to describe concisely behaviors such as synchronization, arbitration and constrained scheduling between concurrent connections. Nevertheless, the dynamic connection of interfaces supporting back-pressure (ready-to-receive signals) is confronted to the pitfall of induced combinational loops, resulting in unpredictable behavior. The proposed methodology addresses this issue too.

This paper illustrates how our STL description and synthesis methodology can help a designer to quickly implement the matrix multiplication application, which involves the design of a state-of-the-art floating-point accumulator that relies on the delayed-buffering (DB) method [17]. At the featured abstraction level, such design requires only a few state machines handling constrained connections between data-driven operators. Most of the hardware complexity is implicit and automatically handled by our compiler. The number of transfers that are authorized at each clock cycle is also optimized, taking into account the dependencies between the transfers and the constraints specified by the designer. The compiler is particularly helpful at optimizing cyclic dependencies, preventing the generation of combinatorial loops. The full description of our pipelined implementation of the matrix multiplication circuit is simple enough to be implemented in hours instead of days, without any advanced knowledge of hardware design. Nevertheless, the performances obtained are similar to the ones reported by experts for the dedicated implementations already cited.

The paper is organized as follows: Section 2 discusses related work in the field of high level hardware synthesis. Section 3 presents the main features of the STL design methodology such as predefined synchronization protocols and interfaces, transfer authorization rules, and the handling of combinational loops. Section 4 presents the application of the featured methodology to the design of a floating-point matrix multiplication hardware accelerator, supporting matrix sizes up to 1024×1024, and of a high-performance floating-point accumulator. Section 5 presents and discusses the results obtained for the FPGA implementation targeting both Virtex-V and Stratix-III devices. Section 6 concludes this work.

2. Related work

Proposing useful abstractions for beyond-RTL hardware description languages remains an open challenge. The design of modern digital hardware applications involves the interconnection of hundreds and more components, ranging from simple registers to complex multi-core devices. Despite decades of research and development, C/C++/SystemC high-level synthesis, while most beneficial to system-level design and verification and for fast prototyping [18–20], has yet to deliver efficient hardware synthesis that could replace RTL methodologies [21]. For instance, these approaches work well when the control flow of the application is statically determined, but are less appropriate to efficiently support and handle highly data-dependent control-flows. It has also been pointed-out that the C language may not be well suited for the description of concurrent hardware designs [22]. Some issues such as the ability to describe structure are considerably alleviated with support for SystemC, but C/C++ remains highly sequential and automatic parallelism extraction has its limits, pointer analysis and aliasing being an notable one.

The use of logical implication rules to constrain data transfer authorizations over synchronized connections in our STL methodology is somehow akin to the use of atomic guarded actions that is proposed in BlueSpec SystemVerilog [23]. BlueSpec features operation-centric semantics [24], and lets the user specify behaviors as a collection of atomic guarded actions, referred to as rules. When a predicate is true, all the actions guarded by that predicate are executed concurrently. It is also possible to compose these rules to produce new behaviors, such that two sets of guarded actions can execute atomically, logically one after the other, when both predicates are simultaneously true. Yet, the concept of constraint rule remains different from that of a guarded action rule in the sense that the former represents necessary rules, while the later represent necessary and sufficient rules for an action (transfer) to happen. At the application level, the work in [7] presents the design of a matrix multiplication hardware accelerator using BlueSpec high-level synthesis HDL. The resulting FPGA implementation on a Virtex-II