Journal of Systems Architecture 68 (2016) 17–37

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

An OpenCL software compilation framework targeting an SoC-FPGA VLIW chip multiprocessor

Samuel J. Parker, Vassilios A. Chouliaras*

Wolfson School, Loughborough University, Loughborough, LE11 3TU, UK

ARTICLE INFO

Article history:
Received 12 October 2015
Revised 2 June 2016
Accepted 5 June 2016
Available online 15 June 2016

Keywords:
OpenCL
FPGA
Heterogeneous computing
Multi-core
Compilation

ABSTRACT

Modern systems-on-chip augment their baseline CPU with coprocessors and accelerators to increase overall computational capability and power efficiency, and thus have evolved into heterogeneous multi-core systems. Several languages have been developed to enable this paradigm shift, including CUDA and OpenCL. This paper discusses a unified compilation environment to enable heterogeneous system design through the use of OpenCL and a highly configurable VLIW Chip Multiprocessor architecture known as the LE1. An LLVM compilation framework was researched and a prototype developed to enable the execution of OpenCL applications on a number of hardware configurations of the LE1 CMP. The presented OpenCL framework fully automates the compilation flow and supports work-item coalescing which better maps onto the ILP processor cores of the LE1 architecture. This paper discusses in detail both the software stack and target hardware architecture and evaluates the scalability of the proposed framework by running 12 industry-standard OpenCL benchmarks drawn from the AMD SDK and the Rodinia suites. The benchmarks are executed on 40 LE1 configurations with 10 implemented on an SoC-FPGA and the remaining on a cycle-accurate simulator. Across 12 OpenCL benchmarks results demonstrate near-linear wall-clock performance improvement of 1.8 × (using 2 dual-issue cores), up to 5.2 × (using 8 dual-issue cores) and on one case, super-linear improvement of 8.4 × (FixOffset kernel, 8 dual-issue cores). The number of OpenCL benchmarks evaluated makes this study one of the most complete in the literature.

1. Introduction

State-of-the-art silicon technology nodes empowered VLSI designers to integrate complex functionality on a single chip with such advanced Systems-on-Chip (SoC) incorporating multiple diverse (Heterogeneous) processing engines and connected via numerous, high bandwidth, point-to-point links. These engines are supplied with data by hundreds of local memory blocks under the control of Direct Memory Access (DMA) engines. On this bespoke computing substrate there is the implicit requirement that millions of lines of both legacy and new application code will run efficiently with both software and hardware components expected to be delivered to market under very tight deadlines. Further complications such as the substantial non-recurrent costs involved and verification closure at tape-out make state-of-the-art SoC design inaccessible to all but the largest of organizations.

In parallel, industry witnesses a revolution in performance and capability of Field-Programmable Gate Arrays (FPGAs) with the leading vendors (Xilinx and Altera, now Intel) consistently delivering high capacity programmable silicon incorporating hundreds of embedded (hard-wired) blocks. These include memory controllers, DSPs, clocking infrastructure, high-throughput interfaces (PCIe) and networking capability (Interlaken), supported by very high speed differential I/O (SERDES). The vendors supply a wealth of silicon intellectual property (IP) such as soft processors and more recently, high-value hardened IP (ARM A9 SMP subsystem in the Zynq [1] and Cyclone V SoC device families respectively), advanced interconnect (AXI4) and a number of other blocks covering every conceivable application. What is even more noteworthy is that this rich ecosystem, along with proprietary Electronic Design Automation (EDA) tools is provided for (nearly) free to the FPGA silicon customers. To address the design and verification bottleneck of very complex current (28 nm) and expected (16/14 nm) SoC-FPGAs vendors increasingly embrace a software-centric design approach based on Electronic System Level Methodologies (ESL). Potentially disruptive ESL technologies such as *Behavioural Synthesis* (AutoESL [2] from Xilinx and C2H from Altera and very recently tools such as SDSoC/SDAccel and AOCL [3] respectively) seem to be displacing established Register-Transfer-Level (RTL) methodologies when targeting these latest devices.

* Corresponding author.
E-mail address: v.a.chouliaras@lboro.ac.uk (V.A. Chouliaras).

http://dx.doi.org/10.1016/j.sysarc.2016.06.003
1383-7621/

With the introduction of General-Purpose Graphics Programming Units (GPGPUs) and the release of the proprietary CUDA API [4] from NVIDIA, a trend towards the universal use of such devices in a number of market segments (spanning the continuum from High Performance Computing (HPC, [5]), Desktop and all the way to embedded and mobile computing) is emerging. The Open Compute Language (OpenCL, [6]) was proposed as an open standard API for general-purpose computing across CPUs, GPGPUs and other accelerators in response to CUDAs performance advantage on NVIDIA hardware. This was standardized by the Khronos Group and nowadays, OpenCL drivers are offered by all the major graphic processor designers such as AMD, Intel, and Qualcomm. [1] Unlike CUDA, OpenCL is target agnostic and this has enabled the emergence of an ecosystem around not only GPGPUs but also CPUs and FPGAs as will be discussed in Section 2.

This research is motivated by the ever-increasing adoption of the Single Instruction Multiple Thread (SIMT) processing paradigm (via OpenCL) for advanced FPGA design and this paper presents an automated compilation framework that enables parallel computation, through the execution of OpenCL kernels[2] on a configurable VLIW Chip Multiprocessor (CMP) [7,8]. The LE1 architecture (Section 3.1) is both configurable and extensible and is designed for embedded DSP applications on FPGA and standard-cell silicon. The researched software framework is in the form of a *user-space driver* which encompasses an LLVM-based compiler back-end as well as a source-to-source transformer that modifies the OpenCL kernels to execute more effectively on the LE1. A high-level view of the researched software/hardware framework is shown in Fig. 1. From the figure, inputs to the framework are the kernel and the machine description (machine.xml) which specifies micro-architectural parameters of the LE1 CMP instance. The kernel is transformed and compiled with a custom LLVM back-end developed for the LE1 resulting in a number of assembly (.s) files. These are combined into two binaries (iram.h, dram.h) with the instruction stream and the initialized data section loaded onto the processor via the API (executing on the ARM host). The final executable is loaded onto the FPGA target via the Xilinx Microprocessor Debugger (xmd) tool. At the same time, the tool-chain is used to validate the LE1 CMP at Register Transfer Level (RTL) using the flow depicted in the bottom half of Fig. 1.

The framework is capable of targeting many hardware configurations (as specified in the machine.xml) and executes OpenCL kernels both on the LE1 CMP, mapped onto a Zynq z7045 device (Xilinx zc706 development board), as well as on a highly cycle-accurate simulator. We evaluate the scalability of our approach using 12 OpenCL benchmarks from the AMD[3] and Rodinia [9] benchmark suites (Section 4.1.1), across 40 machine configurations, making this the largest OpenCL study reported in academic literature to date.

The paper is organized as follows: Section 2 presents the background, state-of-the-art and motivation behind this research. The proposed software/hardware approach is introduced in Section 3 and the detailed methodologies are discussed in Section 4. Section 5 presents the execution results from applying our framework on the chosen OpenCL benchmarks and includes a thorough discussion of our findings. Section 6 draws conclusions on the efficiency of our solution and this paper concludes with a number of suggestions for future software and hardware improvements in Section 7.

2. Motivation and background

2.1. Motivation

The majority of accelerators currently used are deeply multi-threaded, many-core systems such as GPGPUs and Intel's MIC architecture. GPGPUs offer higher performance and energy-efficiency compared to commodity x86 CPUs. However, the US Department of Energy has identified custom designs and the use of co-design as very important in producing even more efficient computers [10]. Co-design can be used to create application specific instruction set extensions for deeply-embedded configurable processors and optimize the design of heterogeneous multi-core SoCs for more efficient computing [11]; a key reason for its use is that the GPGPU execution model is not suitable for all types of problems. The latter relies on an implicit SIMD execution model (SIMT) where the concurrent execution of hundreds of threads is used to mask stalls and long latency operations. GPGPUs achieve maximum throughput when the executing threads maintain the same program counter (PC), allowing the single-issued instruction to execute with different data across hundreds of data-paths. This also allows threads to issue memory operations with high spatial locality resulting in data traffic optimization in the memory hierarchy. These constraints have little effect on highly-regular graphic shader programs, but throughput can dramatically decrease in the presence of control-flow with bespoke solutions proposed to alleviate thread divergence [12,13]. System designers have looked into building systems with many cores that are not multi-threaded [14,15], but this approach still does not address the fact that not all problems can be solved effectively in the same manner.

FPGAs, by virtue of their user programmability and dense floating point performance (Altera *Arria10* and *Stratix10* families), are in a unique position of being adopted as universal OpenCL targets and previous generations of these devices have been shown to be faster than GPGPUs for some algorithms [16]. There are, however, major barriers in their widespread adoption as accelerators relating to the skill-set required to design, optimize and verify designs as well as long FPGA tools compilation times (hours to day/s). The latter makes them completely unsuitable for runtime OpenCL kernel compilation which is one of the cornerstones of the OpenCL API. High-level synthesis (HLS) has addressed these issues to a degree by offering higher levels of abstraction with more commonly used languages, such as C, SystemC and more recently, OpenCL. This does not address the issue of place and route time, and for absolute performance VLSI engineers still design at RT level.

The main motivation behind the researched software framework is the need to offer a fully programmable compute engine as a solution between fixed many-core systems such as the Intel MIC and the very fine-grained control when targeting SoC-FPGAs, while eliminating branch-divergence through source-transformation and ILP compilation and alleviating the substantial FPGA place-and-route runtimes. This is achieved through our core contributions which include the instantiation on the SoC-FPGA of the LE1 CMP and the subsequent on-line compilation of OpenCL kernels by our framework targeting the LE1 silicon. We also note that the LE1 is a capable MIMD accelerator, can easily accommodate shared-memory programming models such as OpenMP and POSIX Threads (PThreads) [17] and due to the proposed source transformation/compilation flow (Section 4), it does not suffer software-incurred performance inefficiencies due to thread divergence. The

[1] Officially conformant devices:http://www.khronos.org/conformance/adopters/conformant-products#opencl

[2] An OpenCL kernel is a function executed by multiple processing elements on a 1D/2D/3D application space. Kernels are C-based and their arguments are augmented with memory space specifiers (private, local and global). OpenCL enables the execution of hundreds/thousands of such functions across multiple processing elements (PEs) resulting in substantial performance improvement compared to the sequential version of the application. Kernels are grouped into 'Work-groups' (WG) and multiple such work-groups constitute a Compute Unit (CU).

[3] http://developer.amd.com/tools-and-sdks/opencl-zone/amd-accelerated-parallel-processing-app-sdk