Computers and Electrical Engineering 000 (2016) 1–21

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

Potential and methods for embedding dynamic offloading decisions into application code

Gavin Vaz, Heinrich Riebler[*], Tobias Kenter, Christian Plessl

Department of Computer Science, Paderborn University, 33098 Paderborn, Germany

ARTICLE INFO

Article history:
Received 10 June 2015
Revised 28 April 2016
Accepted 28 April 2016
Available online xxx

Keywords:
Runtime system
Runtime decision
Hotspot offloading
Convey HC-1
LLVM
SCEV

ABSTRACT

A broad spectrum of applications can be accelerated by offloading computation intensive parts to reconfigurable hardware. However, to achieve speedups, the number of loop iterations (trip count) needs to be sufficiently large to amortize offloading overheads. Trip counts are frequently not known at compile time, but only at runtime just before entering a loop. Therefore, we propose to generate code for both the CPU and the coprocessor, and defer the offloading decision to the application runtime. We demonstrate how a toolflow, based on the LLVM compiler framework, can automatically embed dynamic offloading decisions into the application code. We perform in-depth static and dynamic analysis of popular benchmarks, which confirm the general potential of such an approach. We also propose to optimize the offloading process by decoupling the runtime decision from the loop execution (decision slack). The feasibility of our approach is demonstrated by a toolflow that automatically identifies suitable data-parallel loops and generates code for the FPGA coprocessor of a Convey HC-1. We evaluate the integrated toolflow with representative loops executed for different input data sizes.

1. Introduction

With the rise of heterogeneous computing using accelerators in mobile and general-purpose systems and the advent of Field Programmable Gate Arrays (FPGAs) in the data center [1], the question of optimal application partitioning across the HW/SW boundary is as important as ever. As FPGAs become increasingly accessible through techniques like high-level synthesis or overlay architectures, fast and automated yet accurate techniques are required to determine which parts of an application can benefit from being offloaded to an FPGA accelerator. Traditional methods have relied either on static code analysis or on profiling data to characterize suitable application hotspots. If the application's behavior cannot be statically determined at compile time or if it is heavily input data or parameter dependent, partitioning decisions need to be taken heuristically, possibly leaving significant optimization potential on the table.

Therefore we propose to defer the actual decision to offload a particular code section to an accelerator to program runtime, where dynamic information about the execution is available, for example, the loop trip count for data-dependent hot loops. We propose to automatically insert code that performs these runtime decisions into the application code. During a static partitioning process at compile time, code for both targets (CPU and FPGA accelerator) is generated. At runtime, before some potentially acceleratable code section is entered, the amount of computation and the size of data to be worked

[*] Corresponding author.
E-mail addresses: gavin.vaz@uni-paderborn.de (G. Vaz), heinrich.riebler@uni-paderborn.de, heinrich.riebler@upb.de (H. Riebler), kenter@uni-paderborn.de (T. Kenter), christian.plessl@uni-paderborn.de (C. Plessl).

http://dx.doi.org/10.1016/j.compeleceng.2016.04.021
0045-7906/© 2016 Elsevier Ltd. All rights reserved.

on can be determined in many cases and can be used for making better offloading decisions than static analysis only. For this purpose, we build a toolflow based on the LLVM compiler infrastructure [2], which allows us to target all applications that can be compiled to, or are distributed in the LLVM intermediate representation (LLVM IR).

In this work we investigate the possible benefits of offloading decisions at runtime in three connected experiments.

First, we evaluate how often common compute workloads offer an opportunity for runtime decisions under the assumption that offloaded code typically corresponds to loops or loop nests. For this evaluation we let our LLVM based toolflow *statically* (offline) analyze the code of three real-world benchmark suites and determine which loops can be precisely characterized at runtime (RT), but not at compile time (CT). It turns out that this is the case for a considerable fraction of all detected loops. However, to have a reasonable fraction of runtime dependent loops gives us only a necessary precondition for the usefulness of our approach. A runtime depended loop serves only as a good offloading candidate if, in addition, the execution frequency is sufficiently large. Therefore we also present a *dynamic* (online) evaluation of the application behavior where we analyze the execution frequencies of the different types of loop (nests) in more detail. We show that of all the detected loops not only are a considerably fraction runtime dependent, but that many of those loops also impose large execution frequencies. We also evaluate the overheads that are inserted into the applications taking offloading decisions at runtime.

Second, after showing the usefulness of runtime decisions for the offloading process we propose a new approach of rearranging or moving the decision point within the application. For a loop to be characterized as runtime decidable, all the information required to take a runtime decision needs to be available at least right before the beginning of the loop (nest). We show that for most real-world applications from various benchmark suites, this information is available much before its actual usage in the loop. Hence, the decision could be taken earlier and the possible coprocessor execution could be prepared in the timespan gained moving the decision up. We created an LLVM based toolflow which is able to detect and characterize these code movement opportunities. We present different cases where this information can be exploited to enhance the runtime decision and improve the offloading process.

Third, we evaluate the benefit of runtime decisions with an actual toolflow targeting the reconfigurable Convey HC-1 compute platform. The integrated toolflow can automatically identify loops suitable for vectorization and generate efficient code for the host CPU and a vector coprocessor implemented as an FPGA overlay. We have extended this toolflow to analyze for detected loops whether their dynamic offloading decision should be taken at runtime and insert the required decision into the application code. When offloading decisions are deferred to runtime, data movement between CPU and coprocessor also needs to be organized at runtime as well. Alongside our runtime decisions we generate code for proper data movement and again profit from analysis that uses runtime information to determine which amount of data needs to be transferred.

We apply our toolflow to a set of different, runtime-dependent loops with different nesting structures and evaluate the performance for different data dimensions. Our results show that runtime decisions can improve the overall performance of the system as compared to always executing vectorizable code either on the CPU or the coprocessor.

This paper extends our previous conference publication [3] with its static analysis of the potential of runtime decisions in real-world benchmarks by an in-depth dynamic analysis of these applications. We also propose for the first time the concept of decision slack and present a first static analysis of its potential. Finally, we provide additional details of our case-study and extend our coverage of related work.

The remainder of this paper is structured as follows. In Section 2 we discuss related work, in particular other approaches of systems with offloading decisions at runtime and related work targeting the Convey HC-1. In Section 3 we present our approach of inserting the offloading decision right into the code of the application and show an analysis of common benchmark suites and how widely applicable runtime decisions are. In Section 4 we introduce alternative insertion points for runtime decision and discuss our approach to prepone the decision to improve the offloading process. This new approach is evaluated in the same section with real-world applications. In Section 5 we present the integrated toolflow with runtime decisions and code generation for a reconfigurable coprocessor and describe the data migration strategy implemented for this platform. Then, we compare the performance of special aspects and the overall integrated toolflow in Section 6, and finally draw a conclusion in Section 7.

2. Related work

In this section, we describe related work for the topics covered in this paper. First, we present background information on static HW/SW partitioning and describe other efforts in building systems with dynamic HW/SW partitioning. Then we discuss related research on data migration at runtime and complete this section with other research effort in characterization of common benchmark suites.

2.1. Static HW/SW partitioning

One important challenge in the acceleration of applications with the help of specialized hardware is to decide which parts of the application shall be executed in hardware. To tackle this HW/SW partitioning problem, researchers proposed exact methods and heuristics that can be used for automated partitioning toolflows. Apart from the design space exploration strategies, the methods also differ in the granularity of the partitioning objects that are considered. Depending on the target architecture, methods work at the level of functions, loops, basic blocks and even instructions.