CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. (2015)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3501

SPECIAL ISSUE PAPER

Type-driven data layouts for improved vectorisation

Artjoms Šinkarovs[*,†] and Sven-Bodo Scholz

School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, EH14 4AS, UK

SUMMARY

Vector instructions of modern CPUs are crucially important for the performance of compute-intensive algorithms. Auto-vectorisation often fails because of an unfortunate choice of data layout by the programmer. This paper proposes a data layout inference for auto-vectorisation that identifies layout transformations that convert single instruction, multiple data-unfavourable layouts of data structures into favourable ones. We present a type system for layout transformations, and we sketch an inference algorithm for it. Finally, we present some initial performance figures for the impact of the inferred layout transformations. They show that non-intuitive layouts that are inferred through our system can have a vast performance impact on compute intensive programs. Copyright © 2015 John Wiley & Sons, Ltd.

Received 23 December 2013; Revised 13 January 2015; Accepted 10 March 2015

KEY WORDS: vectorisation; data parallelism; hpc

1. INTRODUCTION

In the last decade, vectorisation became an important research topic again, as most of the modern CPUs grant vectorisation capabilities by means of single instruction, multiple data (SIMD) extensions [1]. Classical research into auto-vectorisation focuses on the optimisation of loop nestings [2]. Data-independent operations within such loop nestings are identified; the loop-nestings as well as the order of operations within the loop nestings are reorganised to match pre-defined vectorisation patterns, typically sequences of identical arithmetic operations within loops. For vectorisation to be effective, such subsequent operations need to work on data that are adjacent in memory. Otherwise, loading/storing overheads in most cases outweigh any possible performance gains from using SIMD operations. Furthermore, vectorisation only yields a substantial benefit if it can be applied within loop nestings, preferably within the innermost loops. As a consequence, classical auto-vectorisation fails to deliver substantial performance improvements whenever loop nestings cannot be re-arranged to match the layout of the data structures that are being computed on.

In this paper, we propose a novel approach towards program vectorisation: rather than focusing on a reorganisation of loop nestings, we suggest a reorganisation of data layouts to enable vectorisations. Key to this approach is a program analysis for inferring suitable data layouts. Based on this inference, a subsequent program transformation followed by classical auto-vectorisation achieve the overall goal.

Data layout inference comes with several challenges: most importantly, we have to make sure that any new layout can be accommodated by means of a semantics preserving program transformation. Languages such as C give guarantees on how data are being stored in memory that we need to

*Correspondence to: Artjoms Šinkarovs, School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, EH14 4AS, UK.

†E-mail: A.Sinkarovs@macs.hw.ac.uk