Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 330 (2016) 5–25

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

JIT-Based Cost Analysis for Dynamic Program Transformations

John Magnus Morton[1,4] Patrick Maier[2,4] Phil Trinder[3,4]

School of Computing Science
Univeristy of Glasgow
UK

Abstract

Tracing JIT compilation generates units of compilation that are easy to analyse and are known to execute frequently. The AJITPar project investigates whether the information in JIT traces can be used to dynamically transform programs for a specific parallel architecture. Hence a lightweight cost model is required for JIT traces.
This paper presents the design and implementation of a system for extracting JIT trace information from the Pycket JIT compiler. We define three increasingly parametric cost models for Pycket traces. We determine the best weights for the cost model parameters using linear regression. We evaluate the effectiveness of the cost models for predicting the relative costs of transformed programs.

Keywords: Cost Model, JIT Compiler, Program Transformation, Skeleton, Parallelism

1 Introduction

The general purpose hardware landscape is dominated by parallel architectures — multicores, manycores, clusters, etc. Writing performant parallel code is non-trivial for a fixed architecture, yet it is much harder if the target architecture is not known in advance, or if the code is meant to be portable across a range of architectures. Existing approaches to address this problem of *performance portability*, e.g. OpenCL [21], offer device abstraction yet retain a rather low-level programming model typically intended for a specific problem domain, e.g. for numerical data-parallel problems.

There is less language support for multiple architectures in other domains. For example symbolic computations, like combinatorial searches or computational algebra, often exhibit large degrees of parallelism but the parallelism is *irregular*:

[1] Email: j.morton.2@research.gla.ac.uk
[2] Email: Patrick.Maier@glasgow.ac.uk
[3] Email: Phil.Trinder@glasgow.ac.uk
[4] This work is funded by UK EPSRC grant AJITPar (EP/L000687/1).

http://dx.doi.org/10.1016/j.entcs.2016.12.012

the number and size of parallel tasks is unpredictable, and parallel tasks are often created dynamically and at high rates [28].

The *Adaptive Just-in-Time Parallelism* (AJITPar) project [2] investigates a novel approach to deliver *portable parallel performance* for programs with irregular parallelism across a range of architectures by combining declarative task parallelism with dynamic scheduling and dynamic program transformation. Specifically, AJITPar proposes to adapt task granularity to suit the architecture by transforming tasks at runtime, thus varying the amount of parallelism depending on the architecture. To facilitate dynamic transformations, AJITPar will leverage the dynamic features of the Racket language and its recent trace-based JIT compiler, Pycket [13,10].

Dynamic task scheduling and task transformation both benefit from predicted task runtimes. This paper investigates how to construct lightweight cost models for JIT traces. A JIT *trace* is simply a linear path through the program control flow graph that the compiler has identified as being executed often. We hypothesize that even very simple cost models can yield sufficiently accurate predictions as traces have very restricted control flow, and we only require to compare the *relative* costs of pre- and post-transformed expressions.

The main contributions in this paper are as follows. We have designed and implemented a system for extracting JIT trace information from the Pycket JIT compiler (Section 3). We have defined 3 cost models for JIT traces, ranging from very simple to parametric, and we have used an regression analysis over the Pycket benchmark suite to automatically tune the architecture-specific cost model parameters (Section 4). We have shown that the tuned cost model can be used to accurately predict the relative execution times of transformed programs (Section 5).

2 Related Work

2.1 AJITPar

The Adaptive Just-In-Time Parallelisation (AJITPar) project [2] aims to investigate a novel approach to deliver *portable parallel performance* for programs with irregular parallelism across a range of architectures. The approach proposed combines declarative parallelism with Just In Time (JIT) compilation, dynamic scheduling, and dynamic transformation. The project aims to investigate the performance portability potential of an *Adaptive Skeletons* (AS) library based on task graphs, and an associated parallel execution framework that dynamically schedules and adaptively transforms the task graphs. We express common patterns of parallelism as a relatively standard set of algorithmic skeletons [17], with associated transformations. Dynamic transformations, in particular, rely on the ability to dynamically compile code, which is the primary reason for basing the framework on a JIT compiler. Moreover, a trace-based JIT compiler can deliver estimates of task granularity by dynamic profiling and/or dynamic trace cost analysis, and these can be exploited by the dynamic scheduler. A trace-based JIT-compiled functional language was chosen as functional programs are easy to transform; dynamic compilation allows a wider range of transformations including ones depending on runtime information;