Specializing Code to New Architectures via Dynamic Adaptive Recompilation

Dhanrajbir Singh Hira[1], Quinn Pham[1], Tyler Gobran[1], João P. L. de Carvalho[1], Nemanja Ivanovic[2], Christopher Barton[2], and José Nelson Amaral[1]

University of Alberta[1] and IBM Canada[2]

Introduction

Applications distributed as binaries need to be recompiled to utilize novel instructions introduced in new microprocessors. However, applications are generally distributed as binaries compiled using a subset of instructions common to a family of processors. DAR is a dynamic adaptive recompilation approach which recompiles segments of a program to benefit from target-specific specialization and other improvements in the compiler. DAR generates a fat binary which includes the IR of program segments that may benefit from recompilation.

Proof-of-Concept Implementation

A proof-of-concept implementation of DAR was built using the framework of the LLVM project.

- Functions are selected for recompilation using a C function attribute.
- A transformation pass extracts marked functions and inserts a trampoline into the original function to handle dynamic dispatch logic.
- A JIT runtime library built on top of ORC JIT is used to dynamically compile the extracted IR.
- LLD was extended to create sections in the binary that store IR of marked functions and other DAR metadata.

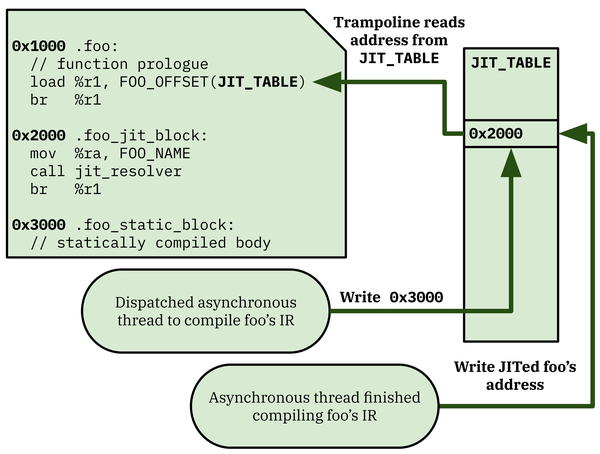

Figure 1: Diagram of the JIT call trampoline.

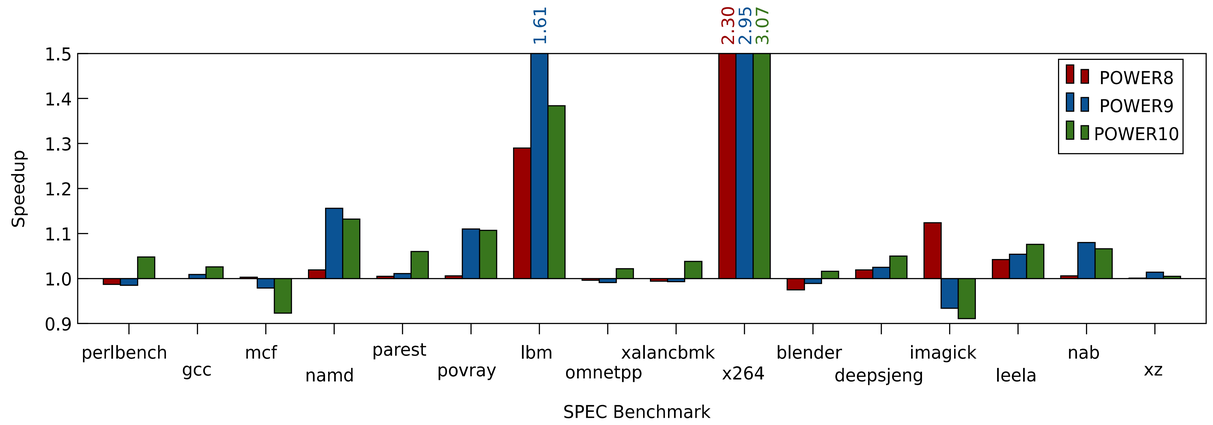

Figure 2: Speedup for sub-targets with respect to Power 7 sub-target baseline.

Performance Gains

Specializing less than **1%** of the functions in x264 achieves **2.29x** speedup.

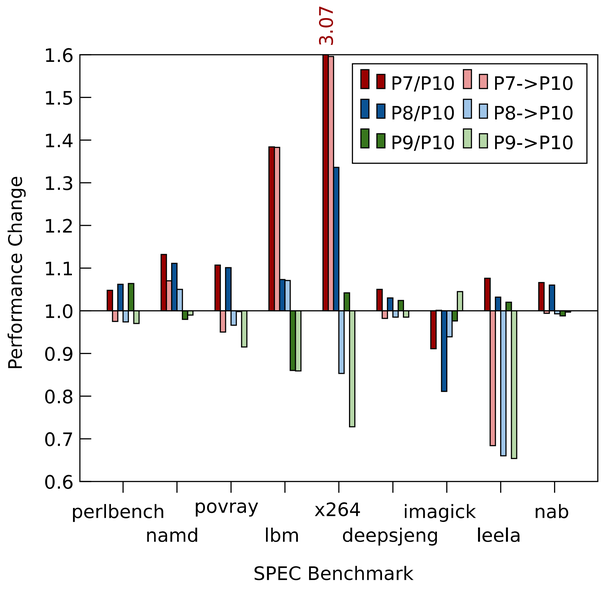

Figure 3: Performance change when the whole benchmark is specialized to Power 10 (PX/PY) and when selected functions are specialized to Power 10 (PX->PY).

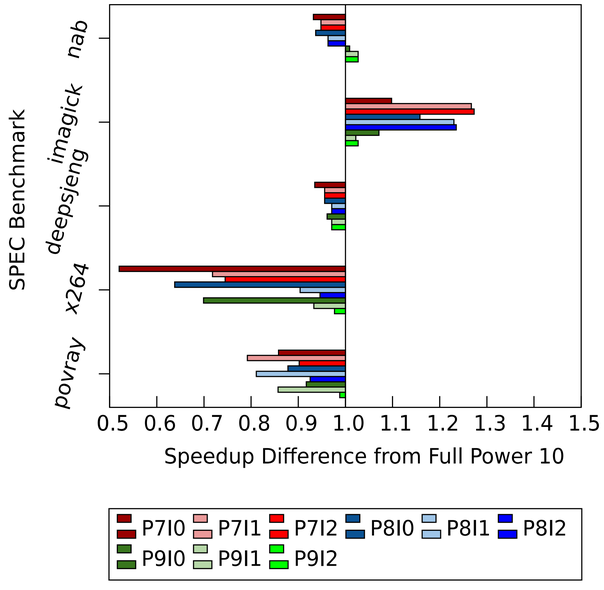

Figure 4: Performance when selected functions are specialized to Power 10 compared to when the whole benchmark is specialized to Power 10. I# indicates the number of inlining iterations performed.

Preliminary Experimentation

Determining whether DAR should be fully developed, we performed experiments using the CPU SPEC 2017 benchmarks to answer the following questions.

- **Is there performance to be gained by targeting newer sub-targets?** Compare binaries compiled statically for different sub-targets (Figure 2).
- **Can we attain similar performance by compiling only a subset of functions for a new sub-target?** Manually select "important" functions to be compiled statically for a new sub-target (Figure 3).
- **Can we improve performance by enabling inlining into the functions compiled for a new sub-target?** Iteratively select additional functions within the call graph of "important" functions to be compiled statically for a new sub-target (Figure 4).

Selecting Functions at Compile Time

In the preliminary experimentation, functions were selected manually using profile information. However, DAR would need to statically select functions to be dynamically recompiled. Selecting appropriate functions is important as every function which does not speedup adds to the overhead of the system. One idea is to gather information about the target features that are queried during the compilation of a function. Functions that do not query target specific features during their compilation are not likely to benefit from specialization. LLVM exposes target information through its TargetTransformInfo (TTI) interface. We implemented a system that records the TTI methods called during the compilation of each function. The goal is to build a model which can predict whether a function will benefit from specialization by looking at the TTI methods called during its compilation.

Contact Information

dhanrajb@ualberta.ca
qpham@ualberta.ca

UNIVERSITY OF ALBERTA