Dynamic Binary Rewriting and Migration for Shared-ISA Asymmetric processors

Giorgis Georgakoudis
University of Thessaly
ggeorgakoudis@gmail.com

Spyros Lalis
University of Thessaly
lalis@inf.uth.gr

Dimitrios S. Nikolopoulos
Queen’s University of Belfast
d.nikolopoulos@qub.ac.uk

1 Introduction

Shared-ISA asymmetric, multicore architectures address both issues of programmability and performance customization, observed in current asymmetric designs of single-ISA and disjoint ISA platforms. In shared-ISA architectures, the system consists of baseline and performance enhanced (PE) cores, implementing overlapping ISAs. Higher performance is achievable by PE instructions while also there is some degree of binary compatibility regarding baseline instructions.

2 Prototype platform

A shared-ISA asymmetric platform on an FPGA using configurable Microblaze soft-cores in two different configurations:

- A minimal core type, implementing the basic ISA
- A PE core type, configured to include additional hardware units that extend the basic ISA with PE instructions

Our contributions:

- A dynamic binary rewriting method, implemented as an OS service, which enables code portability with performance enhancement and allows code migration among cores with different performance capabilities.
- An evaluation of rewriting on our FPGA hardware using benchmarks from the SPEC CPU2006 and Rodinia benchmark suites.
- A case study of multiprogram workloads where we devise a scheduling policy, relying on thread migrations, to minimize a workload’s average turnaround time.

3 Dynamic binary rewriting and migration

Application code is required to be initially compiled for the baseline ISA. Binary rewriting is invoked dynamically, as an OS service, to rewrite thread code:

- On thread creation, when a thread is scheduled to run on a PE core
- On thread migration, provided the thread is migrating to a different core type

The rewriting needs no other input than the binary itself and the target processor. At thread creation, we build the call graph to identify executable code. Patching for execution on a PE core involves identifying calls to SW emulation routines and instruction pattern, replacing them with equivalent PE, hardware instructions and performing only intra-procedural relocations with re-linking to remove superfluous nops. De-patching, when migrating to a baseline core, restores a thread’s code by reversing modifications done.

4 Evaluation

Single program measurements

We measure average normalized turnaround time for a number of SPEC CPU2006 and Rodinia benchmarks, in three different modes:

- Baseline: code is compiled for the baseline ISA and executed on a PE core. We denote rew as the speedup value obtained by rewriting overhead, whereas rew+ohd denotes the (lower) speedup when including overhead. Benchmarks are categorized in three different classes according to speedup achieved: high, medium and low.

Multi-program measurements

We implement an octo-core Microblaze configuration, fully-subscribed when deploying multi-program workloads. There are three types of workloads consisting of benchmarks belonging to different speedup classes: high-low, high-med and med-low.

We measure the average normalized turnaround time for each of those workloads in the following configurations:

- BASE is the lower performance bound: all cores are baseline ones
- UNF, MIG and ORAC have heterogeneous hardware of four PE and four baseline cores where initial thread mapping and migration ability vary.
- PE-REWR and PE-STATIC have all PE cores. In PE-REWR benchmarks are rewritten for PE instructions, while for PE-STATIC code is statically targeted.

5 Conclusions and future work

Conclusions

- Binary rewriting is a feasible method for enabling portability and performance enhancement at the same time.
- Rewritten code performs closely to slightly worse compared with non-portable, statically targeted code for PE instructions.
- Rewriting enables migrations which empower the scheduler to achieve better thread-to-core mappings, subject to higher level goals, as our multi-program case study shows for minimizing a workload’s average turnaround time.

Future work

- Augment binary rewriting with more complex ISA and architectural state transformations.
- Explore binary rewriting techniques for other micro-architectural asymmetries, either ISA-transparent or ISA-intrusive.
- Investigate runtime profiling, estimation methods and scheduling policies for shared-ISA asymmetric platforms.

This work has been partially supported by the European Commission under the I-CORES project (FP7 MCF-IRG Contract #224759)