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.

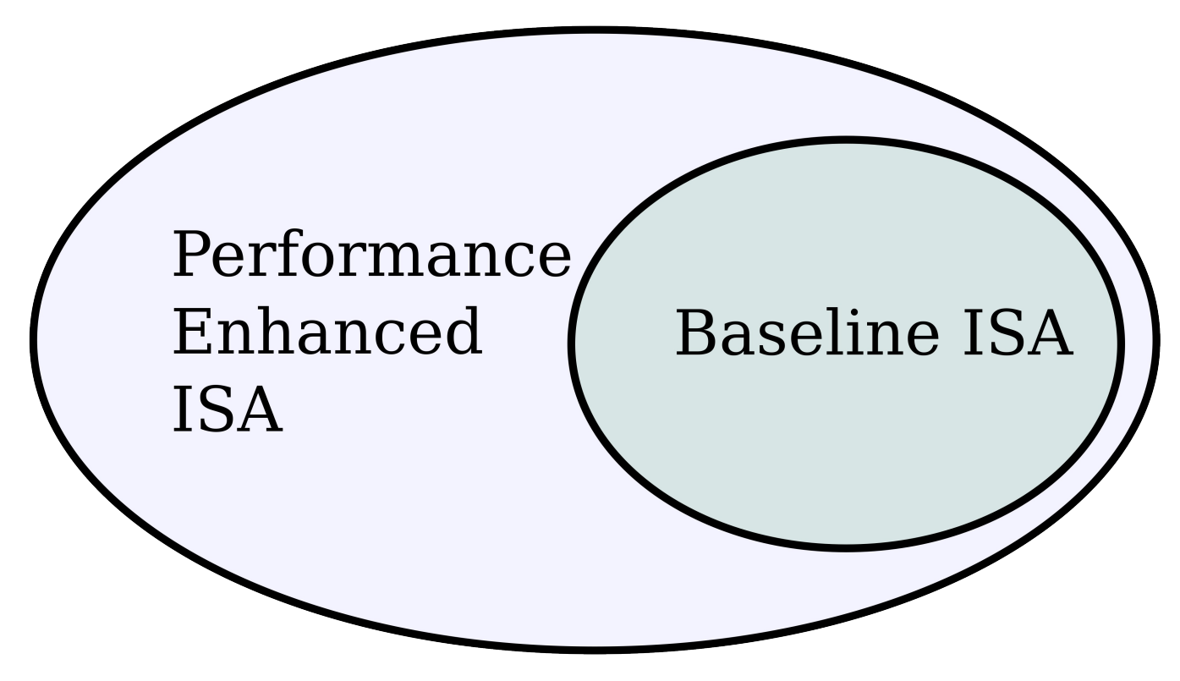


Fig. 1 Baseline ISA is a subset of PE ISA

Motivation: Hardware asymmetry should be handled transparently to the programmer with the goal to achieve code portability and performance enhancement simultaneously. Portability with enhancement enables speedup driven code migration, empowering the scheduler to realize higher-level efficiency and performance goals.

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

Code compiled for the baseline ISA is portable across both core types but lacks performance enhancements whereas code compiled for the PE core achieves better performance at the cost of portability.

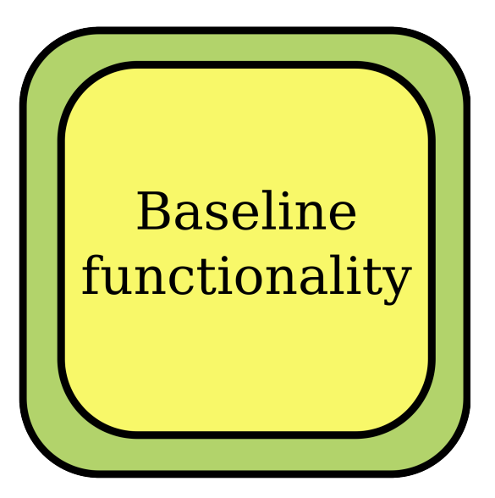


Fig. 2 Baseline core

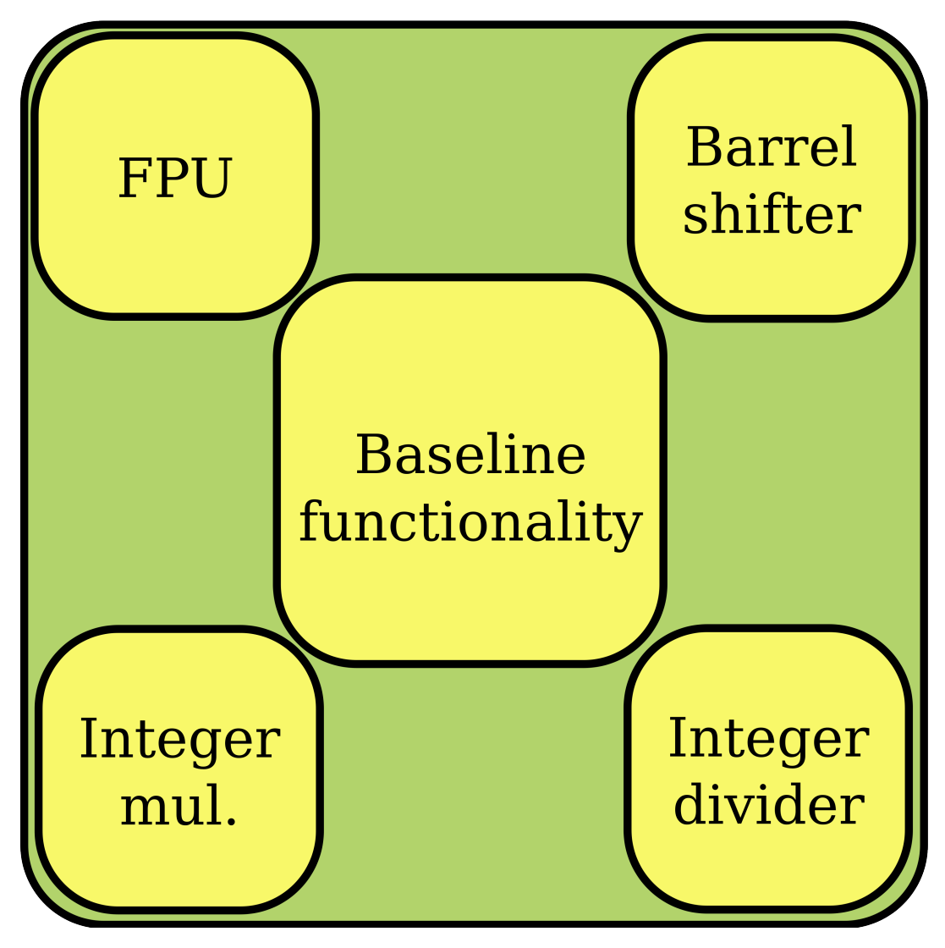


Fig. 3 Performance enhanced core

Our contributions:

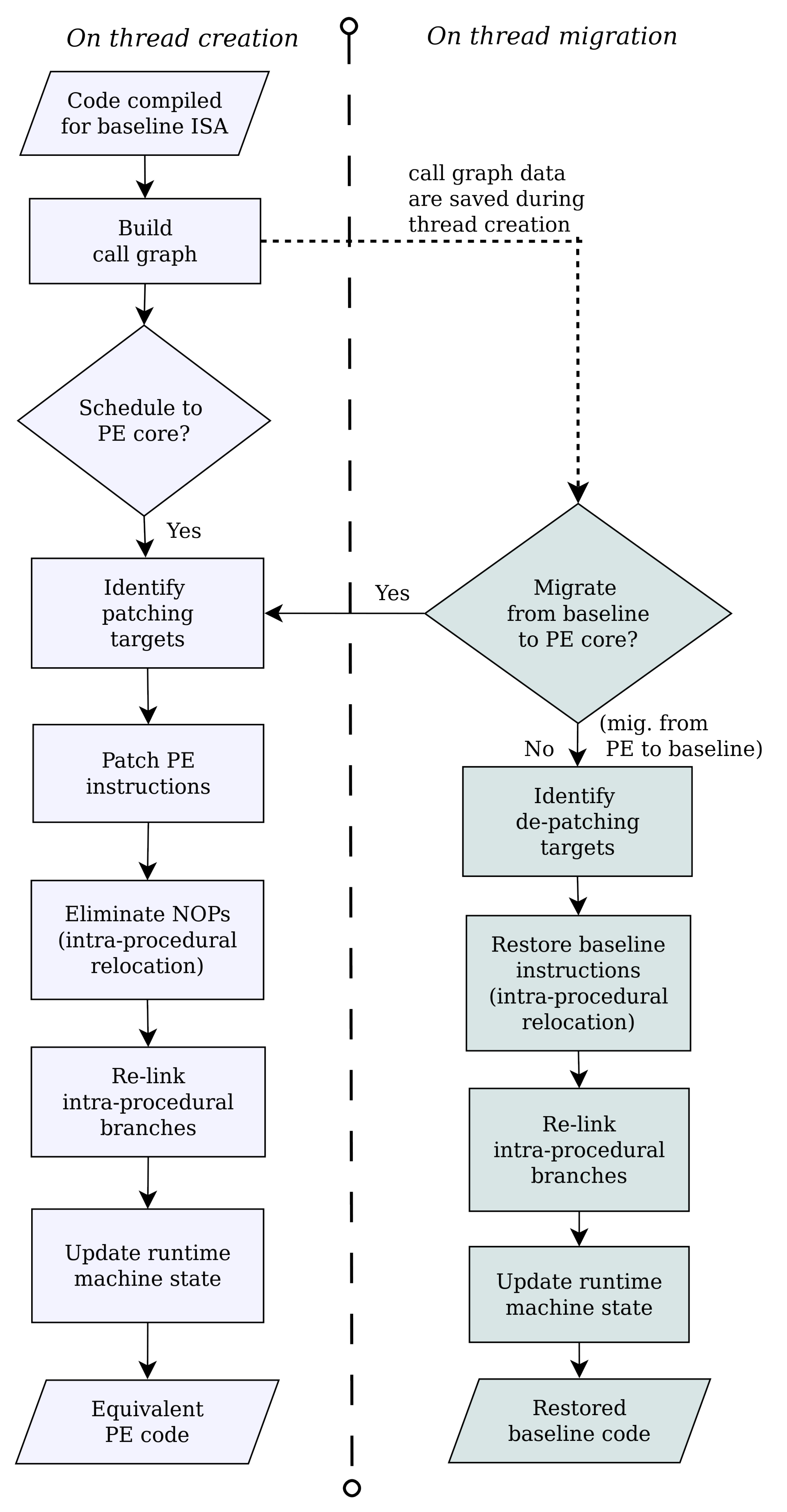
- A **dynamic binary rewriting** method, implemented as an OS service, which enables code portability with performance enhancement and allows code migration among cores with different performance capabilities.
- An evaluation of rewriting on our FPGA hardware using benchmarks from the SPEC CPU2006 and 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 different core type

The rewriter needs no other input than the binary itself and the target processor. At thread creation, we build the thread's 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.


On thread creation
On thread migration
Code compiled for baseline ISA
Build call graph
call graph data are saved during thread creation
Schedule to PE core?
Yes
Identify patching targets
Patch PE instructions
Eliminate NOPs (intra-procedural relocation)
Re-link intra-procedural branches
Update runtime machine state
Equivalent PE code
Migrate from baseline to PE core?
Yes
No
(mig. from PE to baseline)
Identify de-patching targets
Restore baseline instructions (intra-procedural relocation)
Re-link intra-procedural branches
Update runtime machine state
Restored baseline code

We devise a scheduling policy to reduce a multiprogram workload's average turnaround time following the observation that certain threads benefit more when executing on a PE core. Policy implementation relies on:

- *Online profiling* to identify execution hotspots
- *Speedup estimation,* approximating thread speedup
- *Thread migration,* enabled by binary rewriting

After a sampling period, threads with higher speedup estimate migrate to PE cores, while lower speedup are swapped to baseline cores.

4 Evaluation

Single program measurements

We measure *execution time* for a number of SPEC CPU2006 and Rodinia benchmarks, in three different modes:

- Baseline: code is compiled for the baseline ISA executed on a baseline core
- Rewriting: code is compiled for the baseline ISA but executed on a PE core, thus patched by the rewriter to make use of PE instructions
- Static: code is compiled statically for the PE ISA executing to a PE core being non-portable but performance enhanced

We plot *speedup* obtained by rewriting and statically targeted code, denoted as *static*, against baseline. We denote *rewr* as the speedup value obtained omitting rewriting overhead, whereas *rewr+ohd* denotes the (lower) speedup when including overhead. Benchmarks are categorized in three different classes according to speedup achieved: *high*, *medium* and *low.*

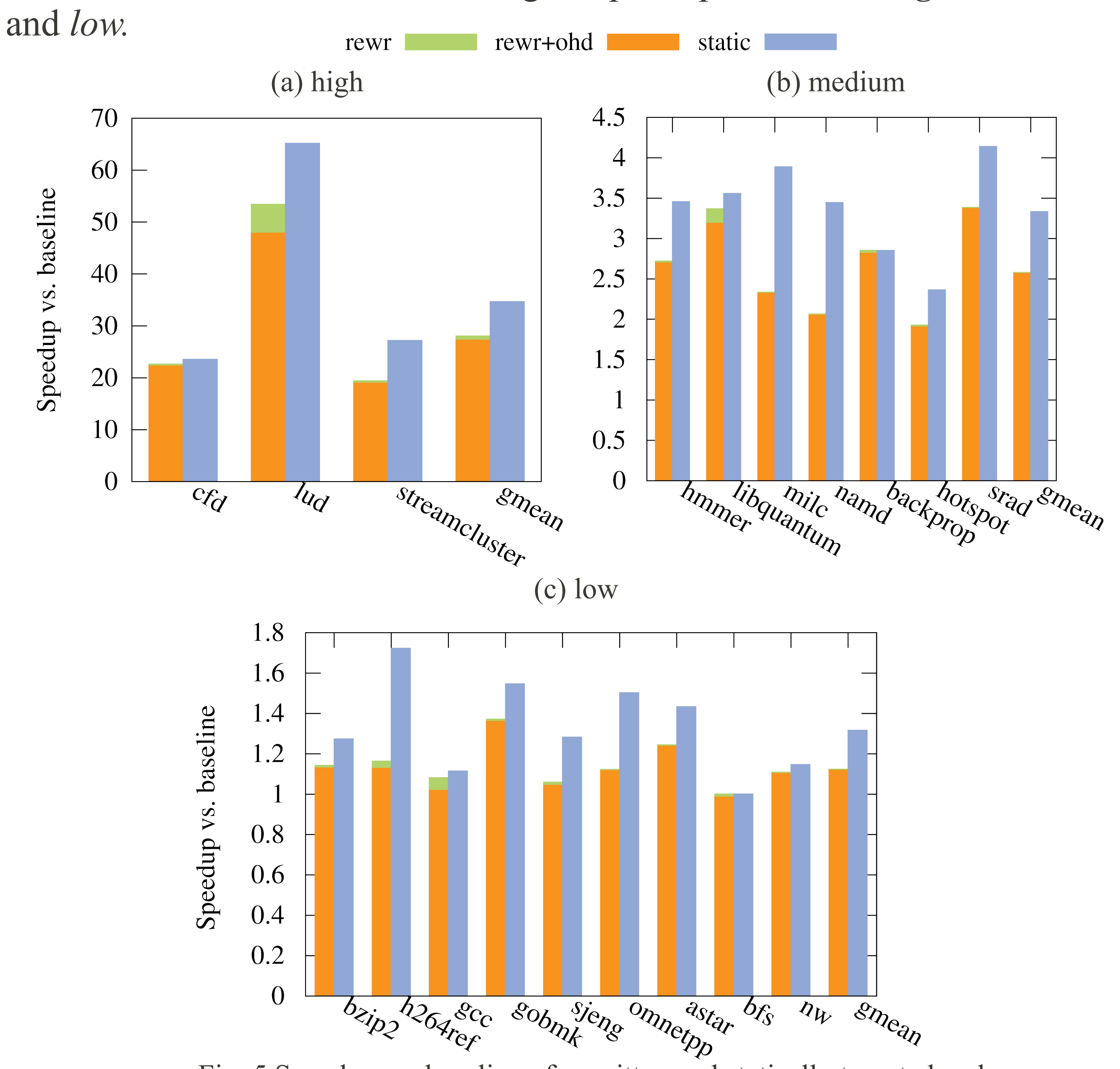


Fig. 5 Speedup vs. baseline of rewritten and statically targeted code

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.*

The plot shows 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

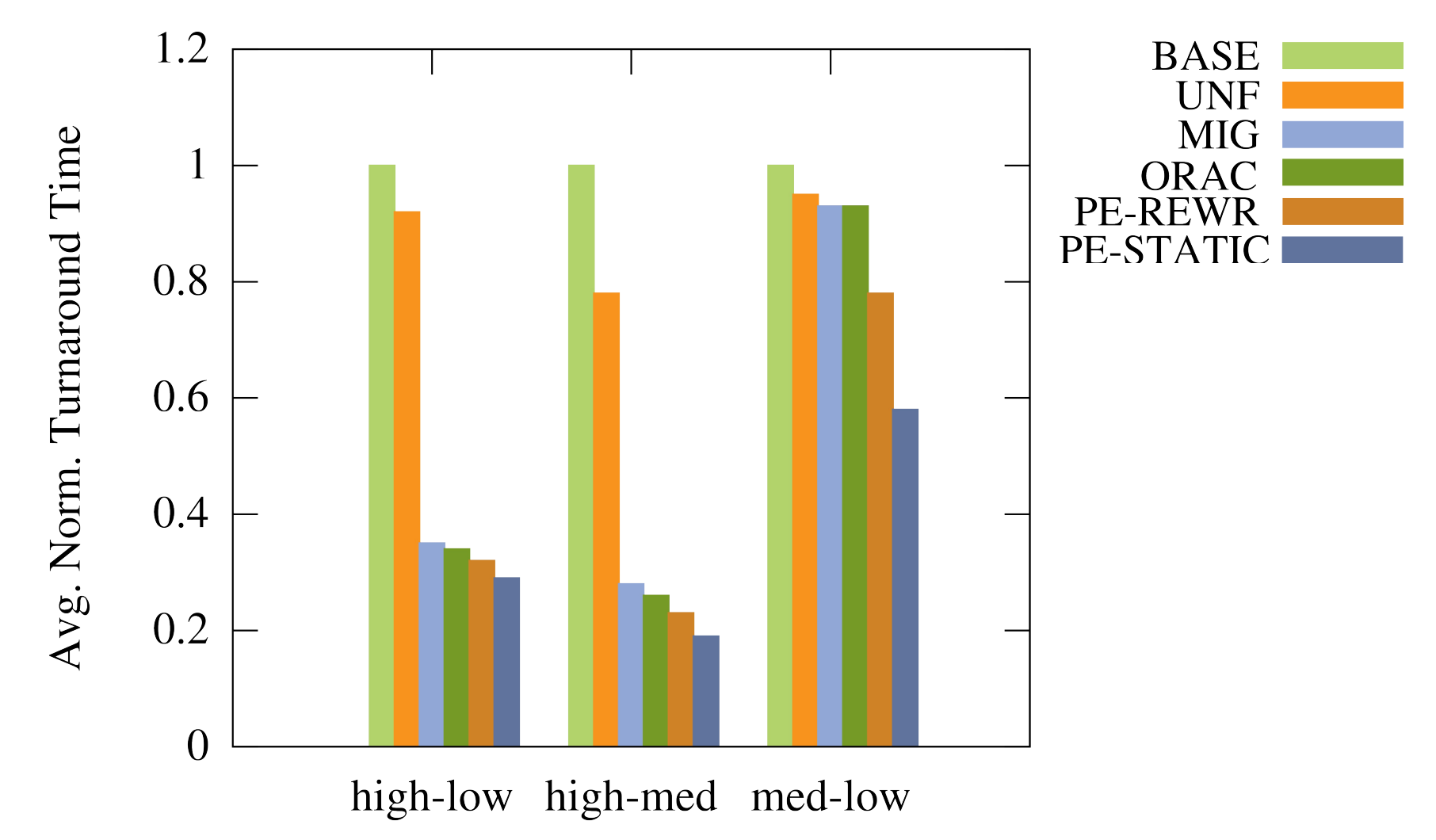


Fig. 6 ANTT for various modes

5 Conclusions and future work

Conclusions

- Binary rewriting is a feasible method for enabling portability and performance enhancement at the same time
- Rewritten code perform 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)