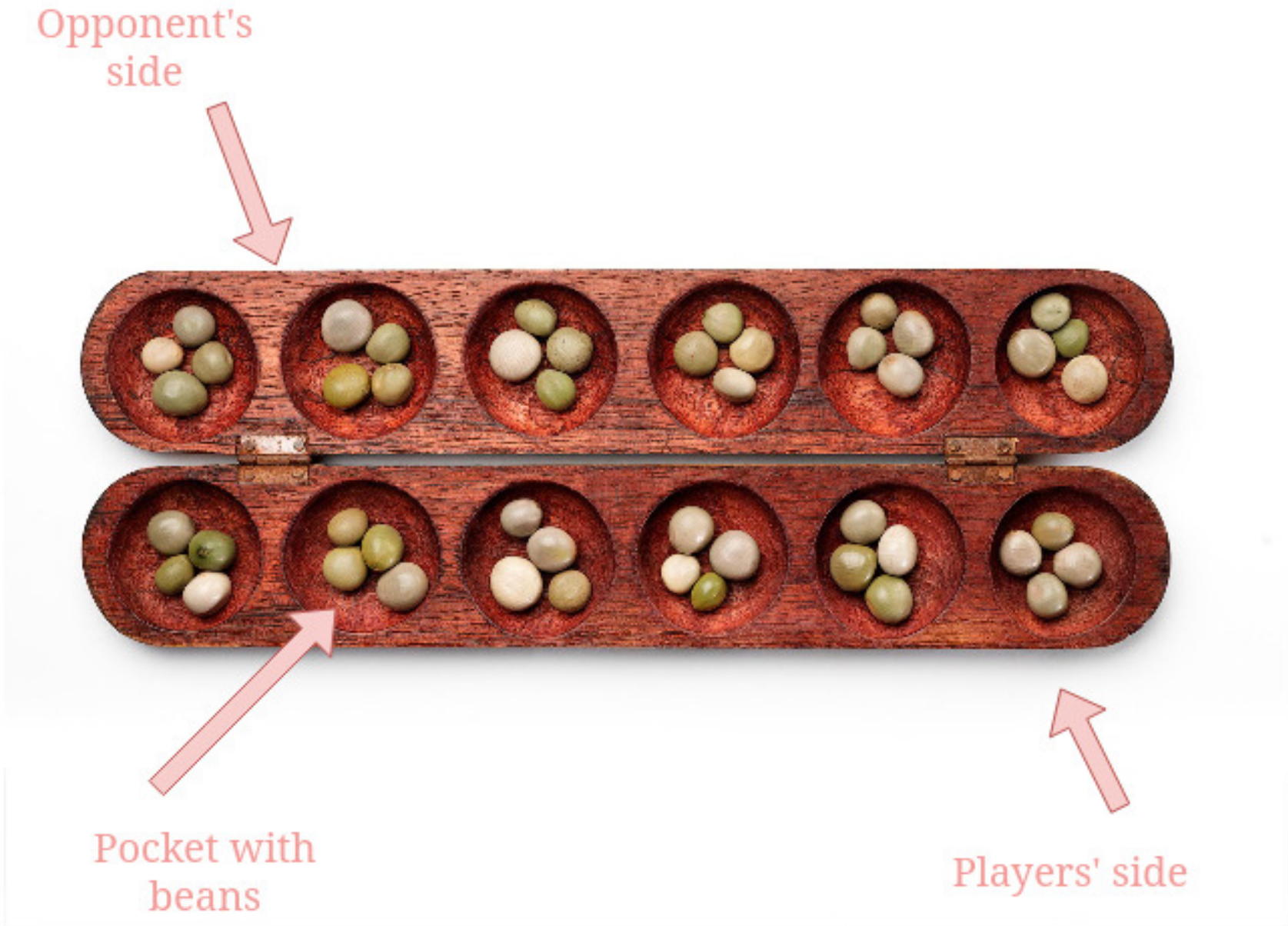
Oware - Optimization

Alexander Graf - 01429203
Christoph Hochrainer - 01429786
Anton Oellerer - 01429853

Introduction

Oware Rules[1]



[1] https://de.wikipedia.org/wiki/Oware

Project

Composed from two programs:

- turn
- **comp**

Goal

Optimize comp

- cycles
- branchmisses
- time

Our Comp Structure

comp variants:

- opt_oware vs oware
- **alphabeta_search:**
- openmp_search vs table_search

Optimizations

- Alpha-Beta Search
- Oware Micro Optimizations
- Profiling
- OpenMP
- Transposition table

Alpha-Beta Search

- Extends min-max search with pruning
- Finds optimal move for the specified depth
- Skips subtrees which can not yield better results
- Disadvantage: harder to parallelize

Oware Micro Optimizations

- Seeding done via loop from 0 to houses
- Increase of seeds per house is calculated once
- Manual loop unrolling
- Capturing is done in subsequent loop
- + Constant amount of loops
- - Divisions, Modulo operations
- - Branches

Profiling

- GCC Profiling [2]
- Profile program to gather information
- Compile the program with optimization according to the information
- + Optimized compiler optimizations
- - Needs to be run before it can be recompiled
- - Cannot deal well with parallelization
- - Optimized for the benchmark program

[2] https://gcc.gnu.org/onlinedocs/gcc/Cross-profiling.html

OpenMP

- Parallelize loops in the search implementation
- - No pruning at top level
- + All searches are parallel

Transposition table

- Array of fixed size
- Saves best move per board layout (+ depth)
- Using zobrist[3] hash of the board
- Update entry on deeper depth
- Good reusability in one turn
- Hard to reuse over multiple turns

[3] https://en.wikipedia.org/wiki/Zobrist_hashing

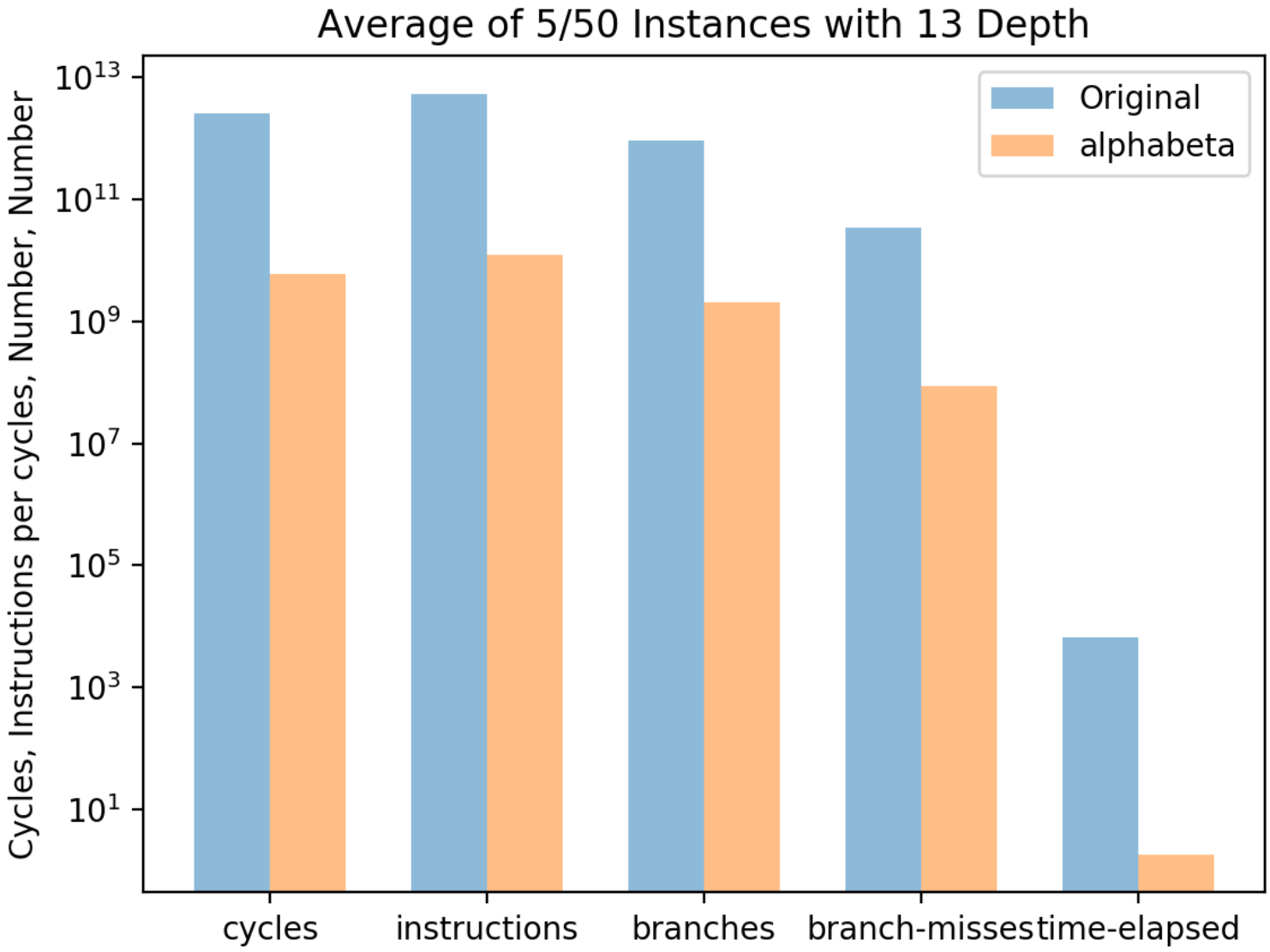
Testing and Benchmarks

- Automated generated oware instances
- Tests can be reproduced with same seed
- Tests compare to original program
- Automated via scripts
- Codequality via warning flags

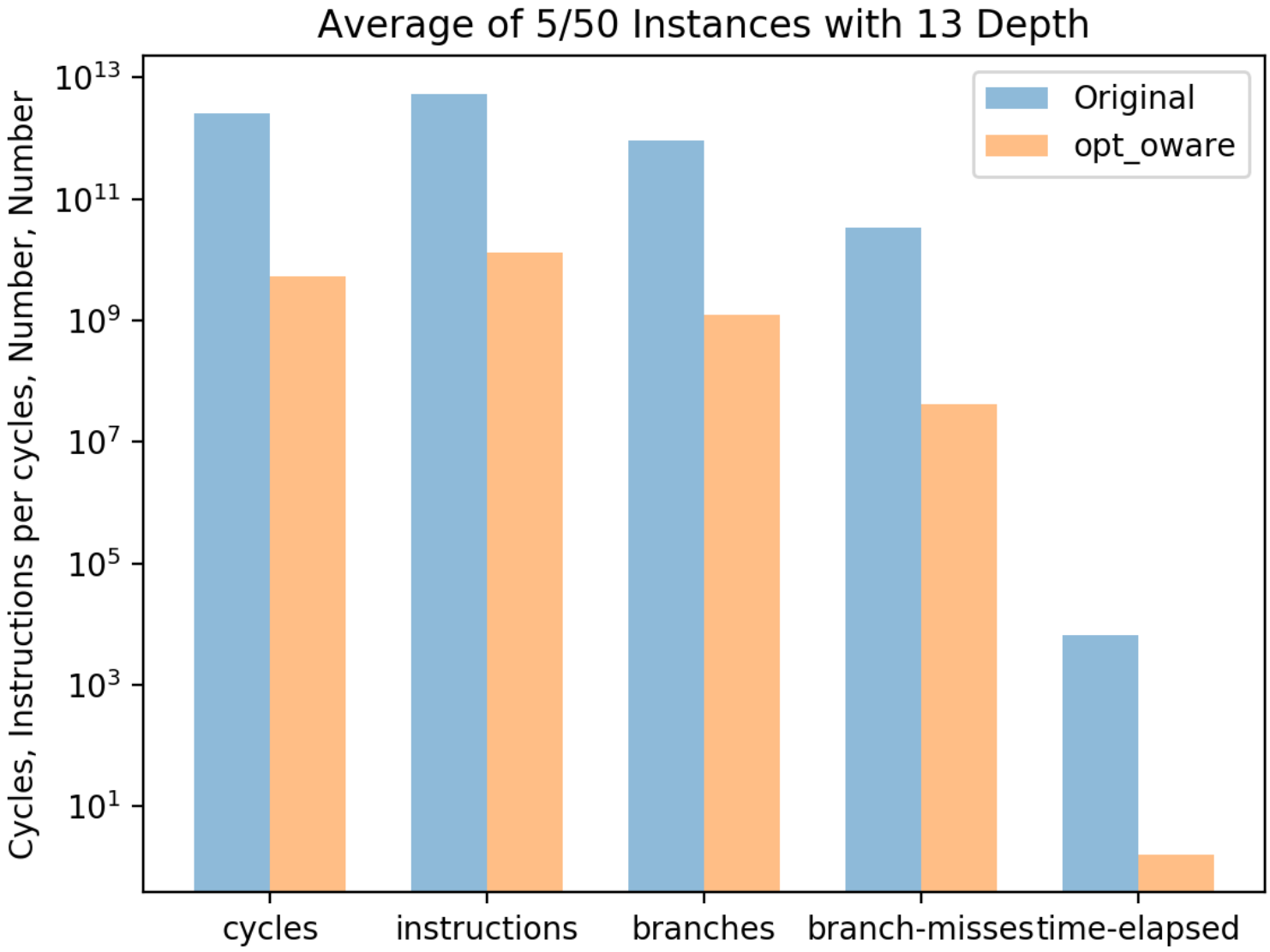
Benchmarks - Setup

- **Number of test instances:** 50
- **OS Version:** Debian 9, Linux Kernel version 4.9.0
- **GCC Version:** 6.3.0
- **CPU:** Intel(R) Xeon(R) CPU E31220 @ 3.10GHz (4 cores)
- **Number of test instances:** 50
- **Compilation flags:** -O3

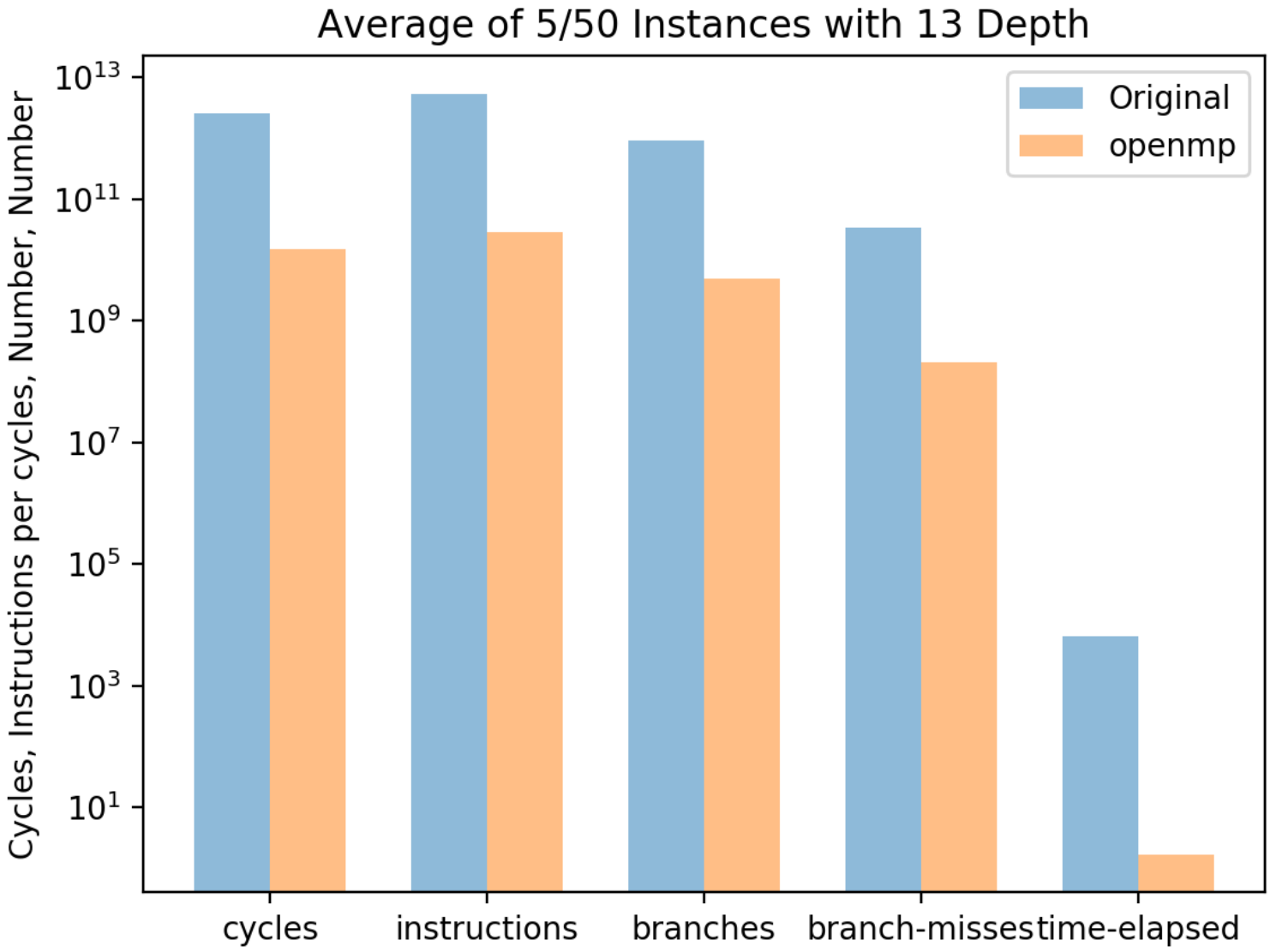
Benchmarks - Original/Alpha-Beta



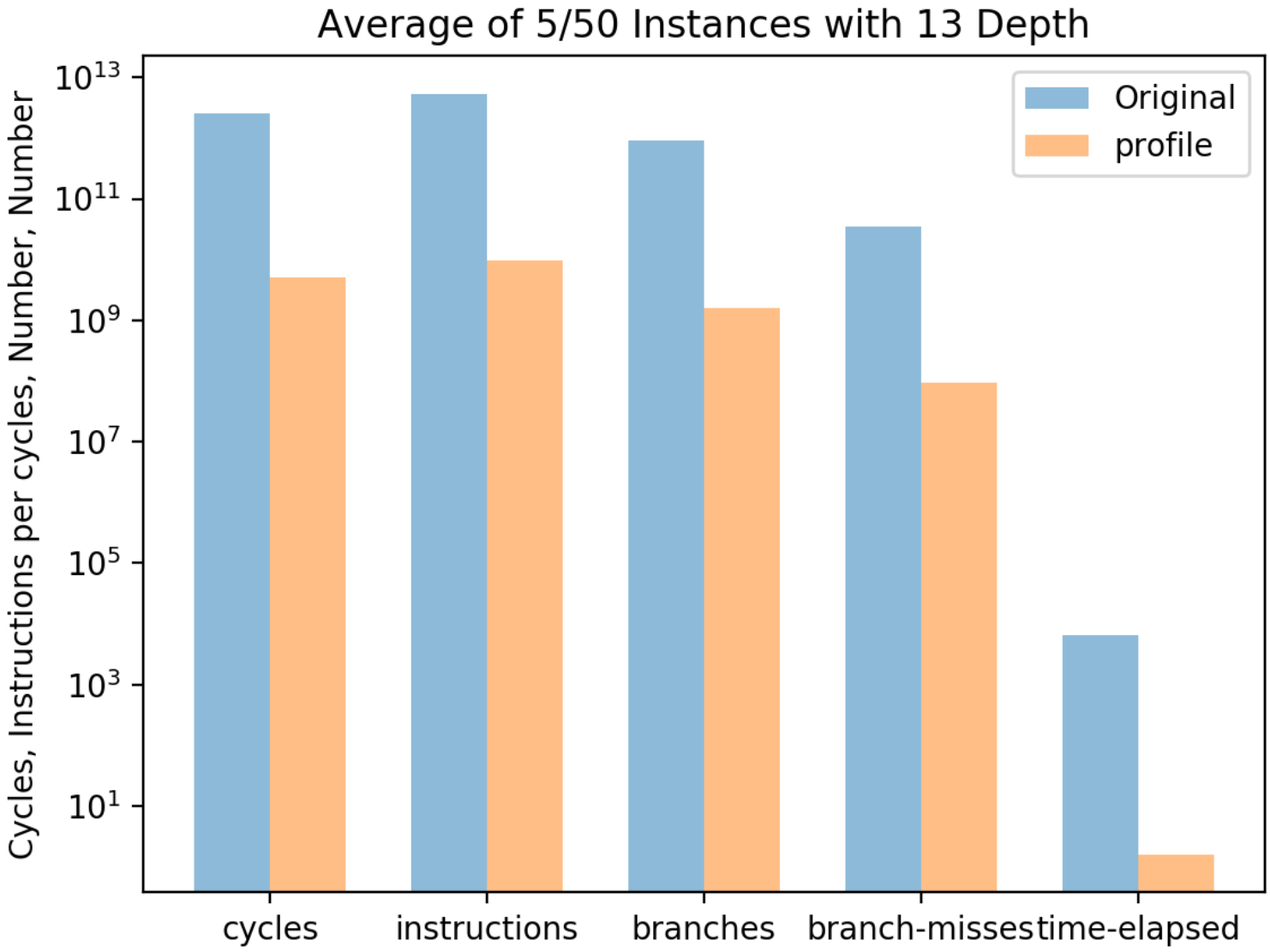
Benchmarks - Original/Opt-oware



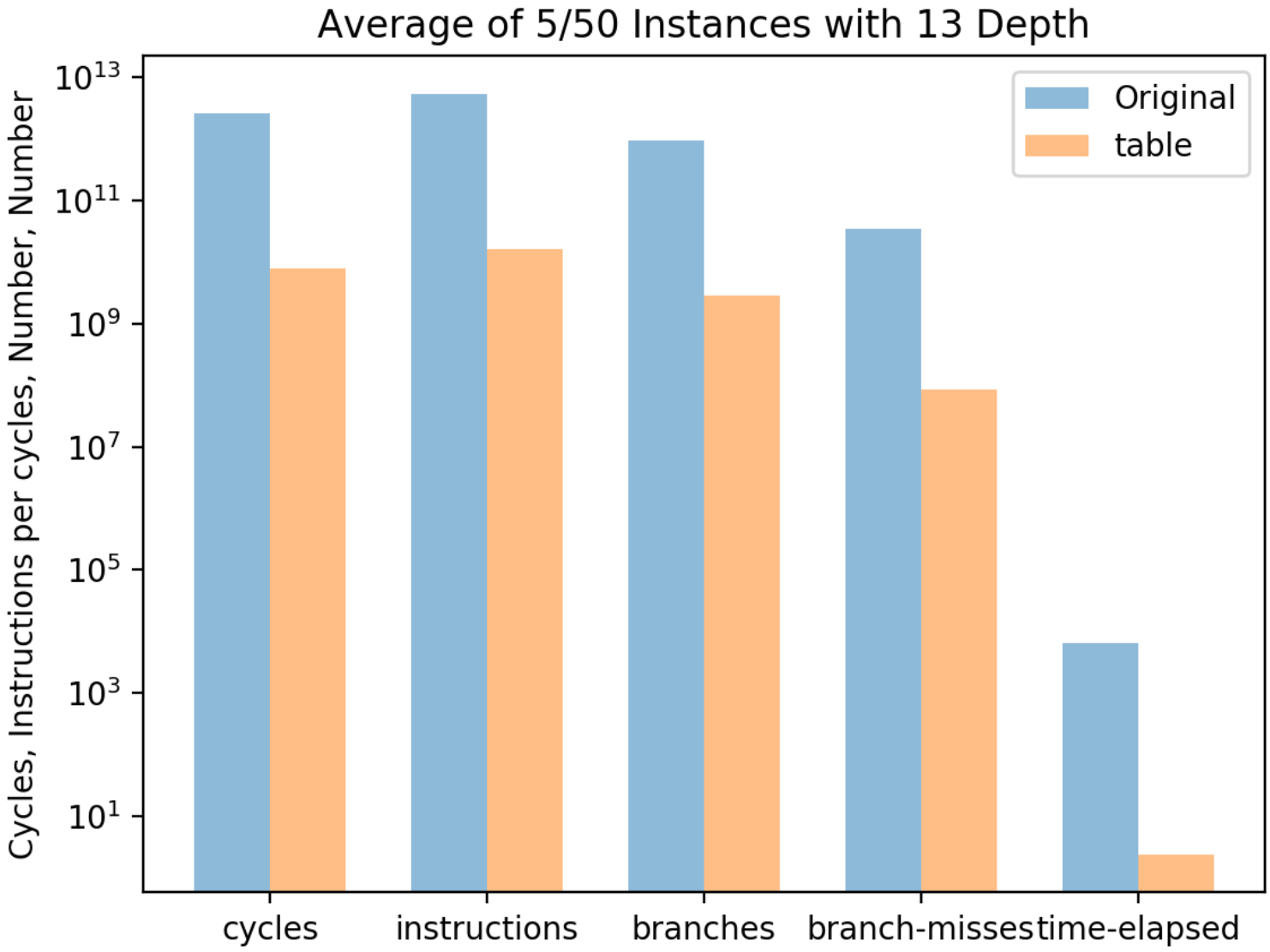
Benchmarks - Alpha-Beta/OpenMP



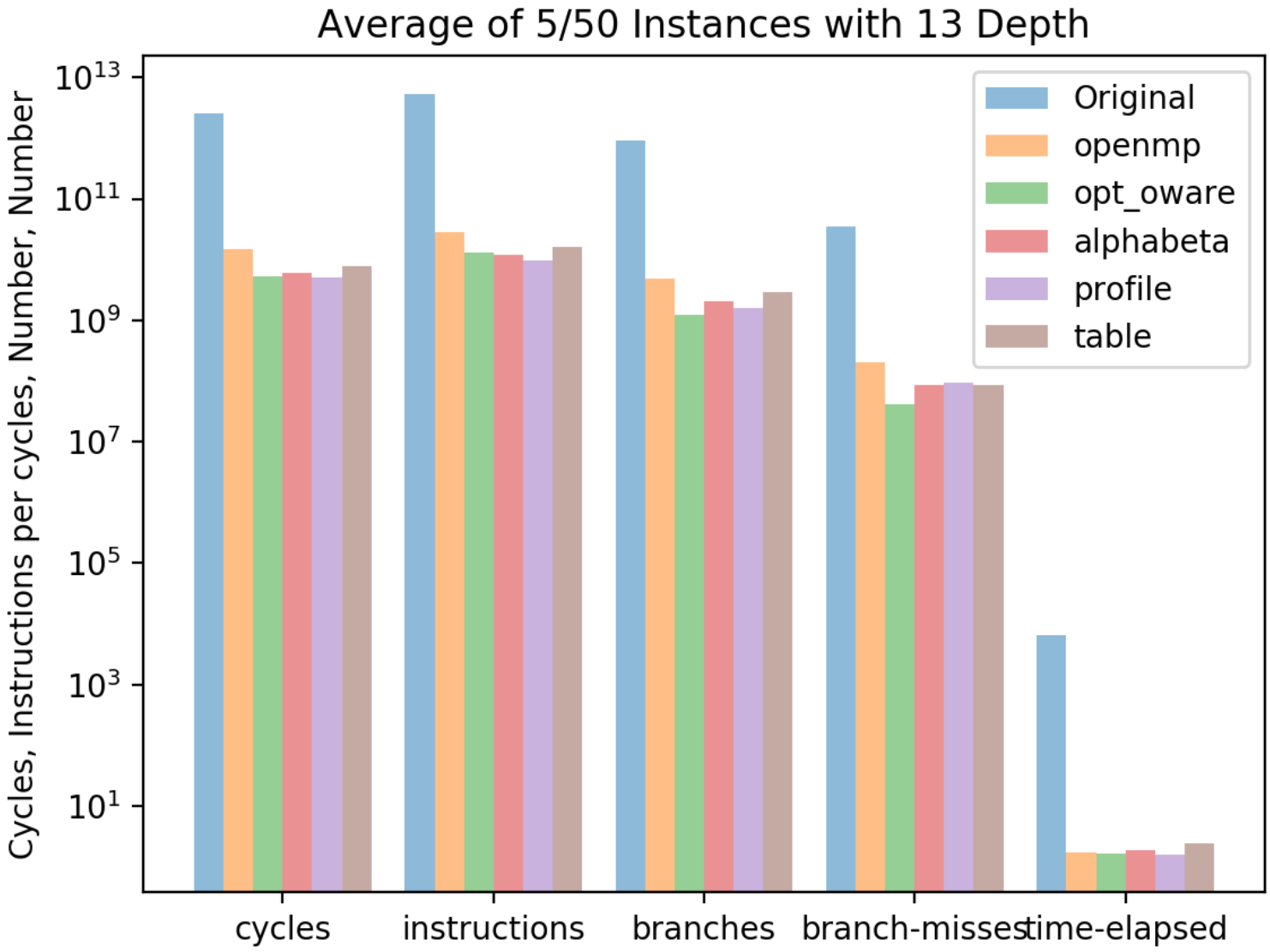
Benchmarks - Alpha-Beta/Profiling



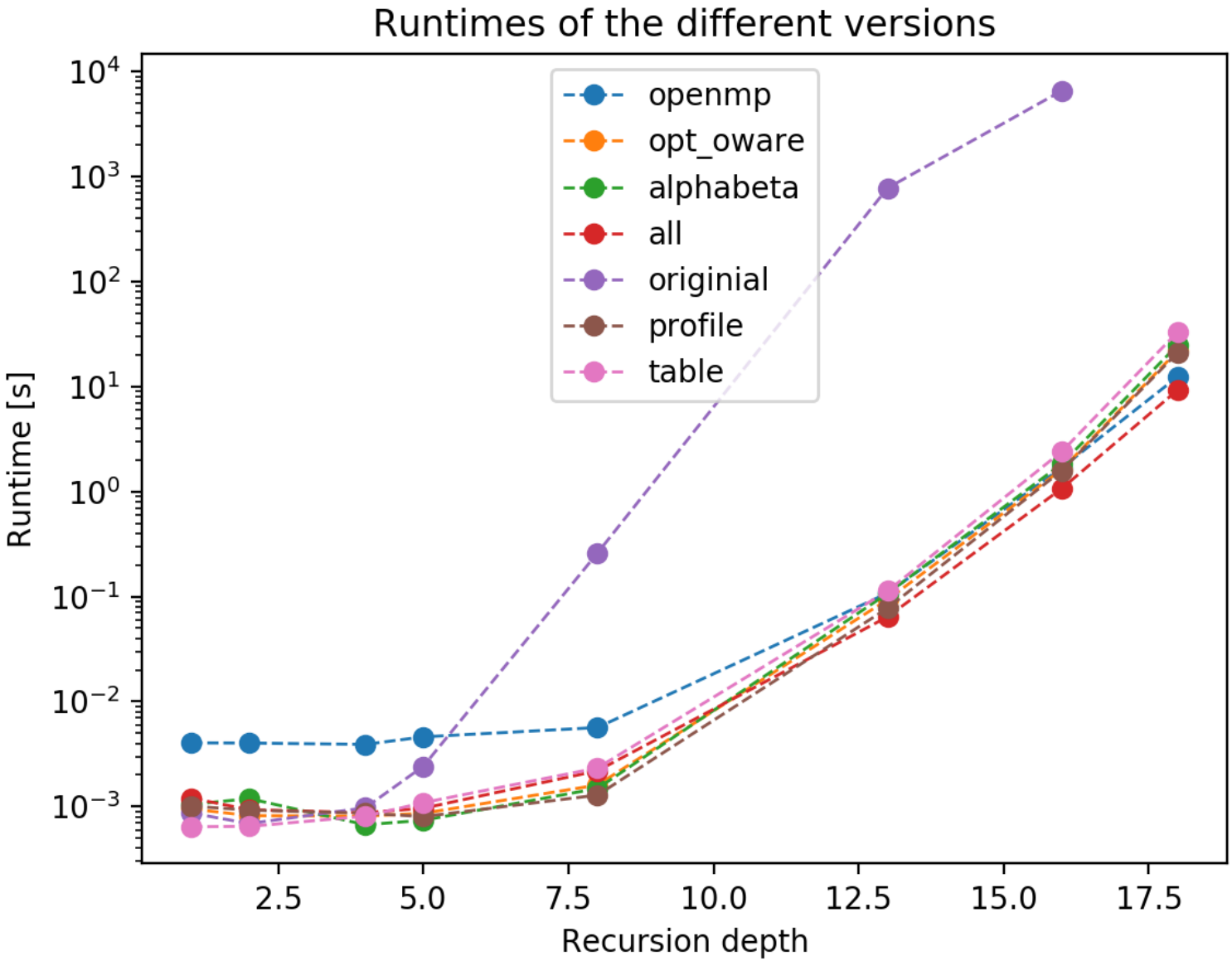
Benchmarks - Alpha-Beta/Table



Benchmarks - Collected Stats



Benchmarks - Scaling



Benchmarks - Overview

optimization	cycles	instructions	instruction/cycle	branches	branch-misses	misses/branches	time
alphabeta	6,013,453,676	12,170,956,782	2.02	2,070,344,913	87,272,090	4.22%	1.832
openmp	14,712,533,778	28,813,657,365	1.96	4,878,496,142	202,890,056	4.16%	1.685
opt_oware	5,296,750,490	12,837,914,926	2.42	1,217,572,091	42,045,841	3.45%	1.622
profile	5,106,006,248	9,759,299,468	1.91	1,606,655,027	92,851,070	5.78%	1.582
table	7,867,361,553	15,984,091,676	2.03	2,840,728,195	86,013,593	3.03%	2.402

Further insights

- *MFLOPs/s:* 0.0001
- *L3 Data Volume (Gbytes):* 0.1941
- *Arithmetic Intensity:* 4.3E-05 FLOPs/Byte
- *Bandwidth:* 11.16 GB/s
- *Attained:* 0.00048 GB/s (0.02
- $\rightarrow$Biggest amount of time spent on integer operations
- $\rightarrow$Most of the tree seems to stay inside of the CPU memory

Conclusion

- Transposition table seems to have too few hits (more research needed)
- OpenMP overhead only pays off after some time
- Alpha-Beta pruning very important
- Profiling most useful micro optmiziation
- Manual loop unrolling very useful
- Branching is expensive
- (Small) memcopys seem to be pretty cheap