

MAMBO: A Low Overhead Dynamic Binary Modification Tool for ARM

.....

Cosmin Gorgovan, Amanieu d'Antras & Mikel Luján
School of Computer Science
University of Manchester

What is Dynamic Binary Modification (DBM)?

- software technique for altering applications at runtime
 - working on machine code
 - transparently to the applications
 - many uses: virtualization, instrumentation, translation
- main limitation: it introduces overheads
 - in particular, runtime performance overhead

Motivation

- previous DBM optimisation research and existing low-overhead DBM tools focused on x86/x86-64
- DBM performance on ARM lagged behind state of the art

Tool	Overhead ¹	Maintained	Source code
Valgrind	226%	Y	GPL
QEMU	1907%	Y	GPL

Table: DBM tools for ARM (2013)

¹ Compared to native execution, on SPEC CPU

MAMBO

- open source DBM tool for ARM (Apache 2.0 license)
 - <https://github.com/bee-hive-lab/mambo>
 - released together with the paper
- designed as a research system
 - small code size (fewer than 10,000 LoC)
 - relaxed transparency reduces implementation complexity
- good performance
 - low runtime performance overhead
 - multithread scaling

Overall design

- all binary code is *scanned* and *translated* before execution to
 - enable arbitrary modification via a plugin API
 - maintain control of execution
 - run correctly from the code cache
- the modified code is stored in a software *code cache*
- the code cache is organised in single-entry and single-exit units - *basic blocks* (BBs)
- data is still accessed in the original locations

Code cache & metadata

Application code

```

0x8000:
MOV R0, #0
ADD R1, R1, #1
BL 0x8081
  
```

Translated code

```

bb_x:
MOV R0, #0
ADD R1, R1, #1
MOVW LR, #((spc+4) & 0xFFFF)
MOVT LR, #((spc+4) >> 16)
B translation_of(0x8081)
  
```

Hash table

SPC	TPC
Source Program Counter	Translated Program Counter
0x8000	bb_x

- BL - Branch-with-Link - call
- B - Branch

Indirect branches: main source of overhead

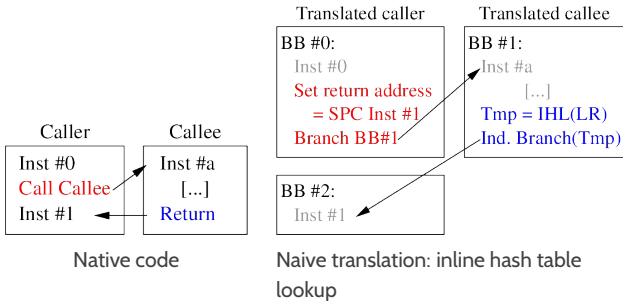
- types:
 - generic indirect branches
 - returns
 - table branches
- their target address is:
 - not available at code scanning time
 - dynamic, it changes across executions of the branch
- source program counter (SPC) to translated program counter (TPC) **lookup done for each execution**

Inline hash table lookup

Hash table (HT) lookup routine used for mapping SPCs to TPCs

- the baseline translation for indirect branches
- uses linear probing
 - to improve hardware data cache locality
- the least significant X bits of SPC are used as the key; $X = 19$
- collision in the HT = branch mispredictions at lookup
 - therefore, a low fill rate HT tends to perform better

Return instruction translation

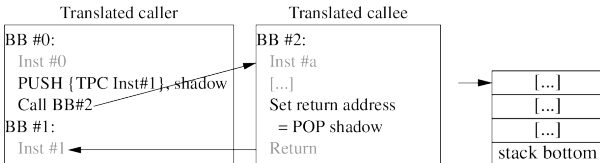


IHL() - inline hash table lookup: 10-20 instructions

- code expansion
- eliminating hardware return address prediction by replacing
 - the direct call with a regular direct branch
 - the return with a regular indirect branch

Low overhead return address prediction

- shorter critical path by using a shadow return address stack
- enables hardware return address prediction

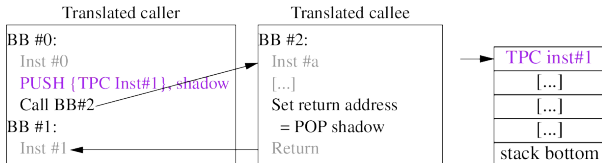


Low overhead return address prediction
(simplified)

The shadow
return address
stack

Low overhead return address prediction

- shorter critical path by using a shadow return address stack
- enables hardware return address prediction

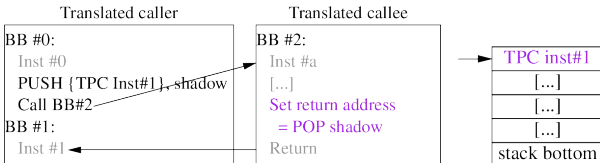


Low overhead return address prediction
(simplified)

The shadow
return address
stack

Low overhead return address prediction

- shorter critical path by using a shadow return address stack
- enables hardware return address prediction

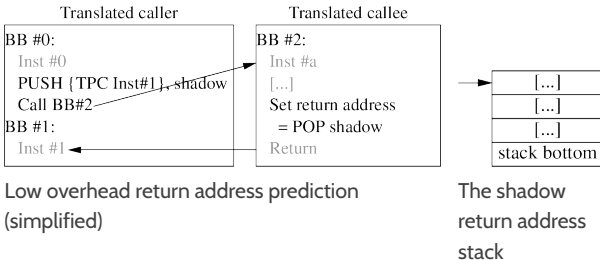


Low overhead return address prediction
(simplified)

The shadow
return address
stack

Low overhead return address prediction

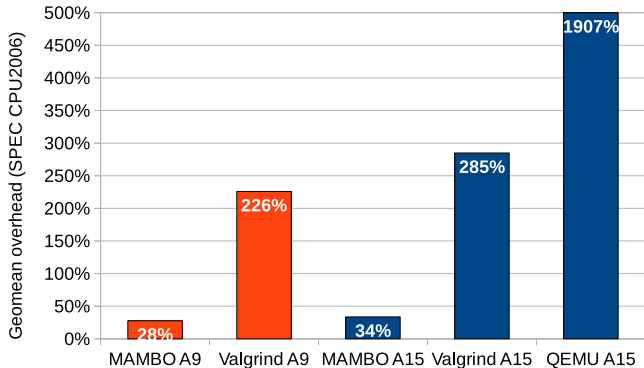
- shorter critical path by using a shadow return address stack
- enables hardware return address prediction



Incorrect return address predictions

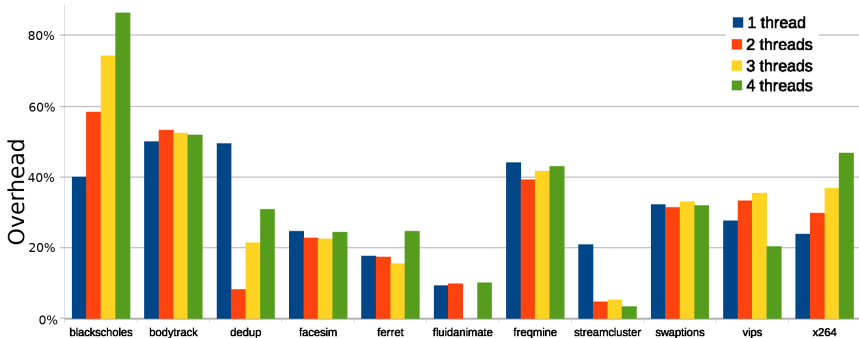
- procedures are generally expected to return to the caller
- but not always:
 - exceptions
 - *longjmp*
 - direct modification of the return address
- please refer to the paper for a complete description of misprediction handling

Single thread overhead (SPEC CPU2006)



- A9: Cortex-A9 (ODROID-X2)
- A15: Cortex-A15 (Jetson TK1)

Multithreaded overhead (PARSEC 3.0) - A15







Geomean overhead (1, 2, 3, 4 threads): 30%, 27%, 32%, 32%

Summary of the paper

- MAMBO is a low overhead, open source DBM tool
- the main source of overhead:
 - the translated indirect branches
- optimisations for:
 - generic indirect branches - efficient inline hash table lookup
 - returns - shadow return address stack, HW branch prediction
 - table branches - space-efficient table branch linking
- overhead on SPEC CPU2006: 28% (A9), 34% (A15)
 - compared to Valgrind: 226% (A9), 285% (A15)

<https://github.com/beehive-lab/mambo>

 syscalls.c	Move syscall handling to a separate file	8 days ago
 traces.c	Handle uncond_imm_arm branches in traces	3 months ago
 util.S	Rename the *.s files to *.S	8 days ago
 util.h	Initial public commit	9 months ago

README.md

MAMBO: A Low-Overhead Dynamic Binary Modification Tool for ARM

News:

- We will present the TACO paper at [HiPEAC 2017](#), on 25th of January.

Publications:

- Cosmin Gorgovan, Amanieu d'Antras, and Mikel Luján. 2016. MAMBO: A low-overhead dynamic binary modification tool for ARM. *ACM Trans. Archit. Code Optim.* 13, 1, Article 14 (April 2016). If you use MAMBO for your research, please cite this paper.

Note that the version of MAMBO published in this repository is newer and has significantly lower overhead than the one used in the paper, mostly due to the implementation of traces. If you want to reproduce the results in the paper, please get in touch.