MANCHESTER
1824
The University of Manchester

Optimising Dynamic Binary Modification Across ARM Microarchitectures

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
    - microarchitectural simulation
    - program analysis
    - memory error detection and debugging
- main limitation: it introduces overheads
  - in particular, runtime performance overhead

MANCHESTER
1824
The University of Manchester

Example: Memory access instrumentation using DBM

```
$ mambo_mtrace /usr/bin/whoami 2> ./memory_trace
cosmin
$ cat memory_trace
w: 0x7ff5ec1080 16
w: 0x7ff5ec10a8 8
w: 0x7ff5ec10f8 8
w: 0x7ff5ec1568 8
w: 0x7ff5ec10d0 8
w: 0x7ff5ec1140 8
w: 0x7ff5ec1148 8
w: 0x7ff5ec1150 8
[...]
```

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	Geomean overhead[1]	Worst case overhead[1]
Valgrind	>200%	>5000%

Table: Performance of DBM tools for ARM

[1] Compared to native execution, on SPEC CPU2006 running on an APM X-C1
[2] *MAMBO: a low-overhead dynamic binary modification tool for ARM.* ACM Transactions on Architecture and Code Optimization (TACO) 2016.

MANCHESTER 1824
The University of Manchester

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	Geomean overhead[1]	Worst case overhead[1]
MAMBO-baseline[2]	26%	165%
Valgrind	>200%	>5000%

Table: Performance of DBM tools for ARM

[1] Compared to native execution, on SPEC CPU2006 running on an APM X-C1
[2] *MAMBO: a low-overhead dynamic binary modification tool for ARM.* ACM Transactions on Architecture and Code Optimization (TACO) 2016.

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	Geomean overhead[1]	Worst case overhead[1]
MAMBO-baseline[2]	26%	165%
DynamoRIO	34%	159%
Valgrind	>200%	>5000%

Table: Performance of DBM tools for ARM

[1] Compared to native execution, on SPEC CPU2006 running on an APM X-C1

[2] *MAMBO: a low-overhead dynamic binary modification tool for ARM.* ACM Transactions on Architecture and Code Optimization (TACO) 2016.

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	Geomean overhead[1]	Worst case overhead[1]
MAMBO-opt	12%	66%
MAMBO-baseline[2]	26%	165%
DynamoRIO	34%	159%
Valgrind	>200%	>5000%

Table: Performance of DBM tools for ARM

[1] Compared to native execution, on SPEC CPU2006 running on an APM X-C1
[2] *MAMBO: a low-overhead dynamic binary modification tool for ARM.* ACM Transactions on Architecture and Code Optimization (TACO) 2016.

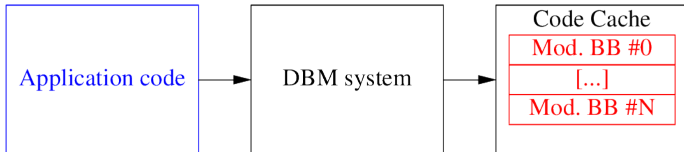
Working principles of DBM

The DBM system scans the application code and copies it to a software code cache:

- it transforms the code to maintain correctness & control
- organised in basic blocks
  - single-entry and single-exit regions
- all application code runs from the code cache
- it enables doing other modifications
  - by plugins via an API
- think JIT (re)compilation for native code

MANCHESTER 1824
The University of Manchester

The code cache



MANCHESTER 1824
The University of Manchester

Optimisations

- Aim: tweak the generated code to better match the processor microarchitecture
- Optimisation for the processor's frontend:
  - Hot code traces
- Indirect branches:
  - branches which have a dynamic target (register or memory)
  - the translation needs to perform a source PC (SPC) to translated PC (TPC) lookup for each execution
  - major source of runtime overhead
  - returns: Hardware-assisted return address prediction
  - generic: Adaptive Indirect Branch Inlining (AIBI)

MANCHESTER 1824
The University of Manchester

HW-assisted return address prediction [1]

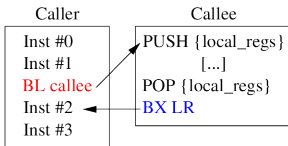


Figure: The original function call

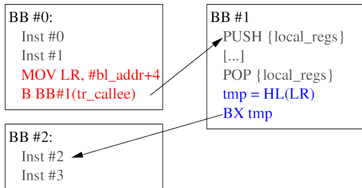


Figure: Typical translation of the function call

MANCHESTER 1824
The University of Manchester

HW-assisted return address prediction [2]

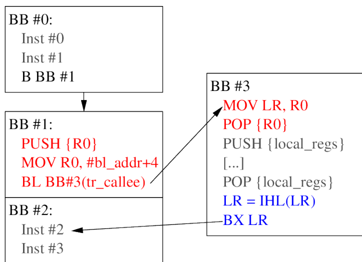


Figure: Translation for HW-assisted return address prediction

- use of call and return instructions preserved
- translations of the call and predicted return in adjacent BBs

The University of Manchester

Adaptive indirect branch inlining [1]

- SPC-to-TPC lookup usually done with a hash table lookup

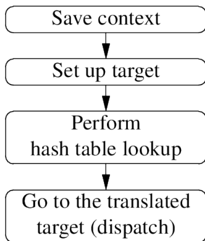


Figure: Translation of an indirect branch with hash table lookup

MANCHESTER 1824
The University of Manchester

Adaptive indirect branch inlining [2]

- some degree of temporal locality
  - can be exploited to predict the TPC
    - however, the prediction must be verified
    - and it must be relatively cheap to update
- AIBI: locally cache predicted SPC and TPC
  - compare target to cached SPC
  - if equal, branch to cached TPC
  - otherwise, perform hash table lookup and update the cached prediction

MANCHESTER 1824
The University of Manchester

Adaptive indirect branch inlining [3]

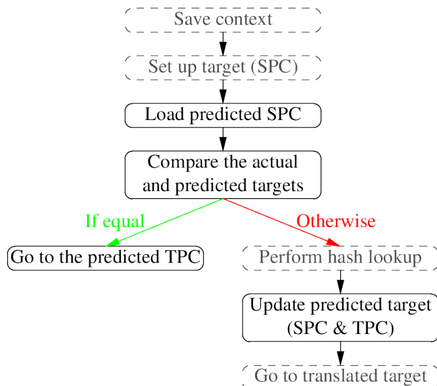


Figure: Translation of an indirect branch with AIBI

MANCHESTER
1824
The University of Manchester

Experimental setup

- SPEC CPU2006
- Paper: 5 ARM computers running GNU/Linux:

System	ODROID-XU3	ODROID-X2	Tronsmart R28	Jetson TK1	APM X-C1
SoC	Exynos 5422	Exynos 4412 Prime	Rockchip RK3288	NVIDIA T124	APM883208
Core	Cortex-A7	Cortex-A9	Cortex-A17	Cortex-A15	X-Gene 1
Frequency	1.4 GHz	1.7 GHz	1.6 GHz	2.3 GHz	2.4 GHz
L2 cache size	512 KiB	1 MiB	1 MiB	2 MiB	256 KiB
L3 cache size	N/A	N/A	N/A	N/A	8 MiB
L1i line length	32	32	64	64	64
L1d line length	64	32	64	64	64
L2 line length	64	32	64	64	64
L1d TLB	10	32	32	32(R) + 32(W)	20
L1i TLB	10	32	32	32	10
L2 TLB	256	132	1024	512	1024
IB predictor	previous[1]	previous	previous	adaptive	adaptive
OOO	N	Y, 2-issue	Y, 2-issue	Y, 3-issue	Y, 4-issue
Pipeline len	8	8-11	10-12	15	15

Experimental setup

- SPEC CPU2006
- APM X-C1
  - APM X-Gene1
  - 2.4 GHz
  - 4-issue out-of-order
  - 15 stage integer pipeline
  - Ubuntu 14.04 LTS, Linux 4.2

MANCHESTER 1824
The University of Manchester

Evaluation [1]

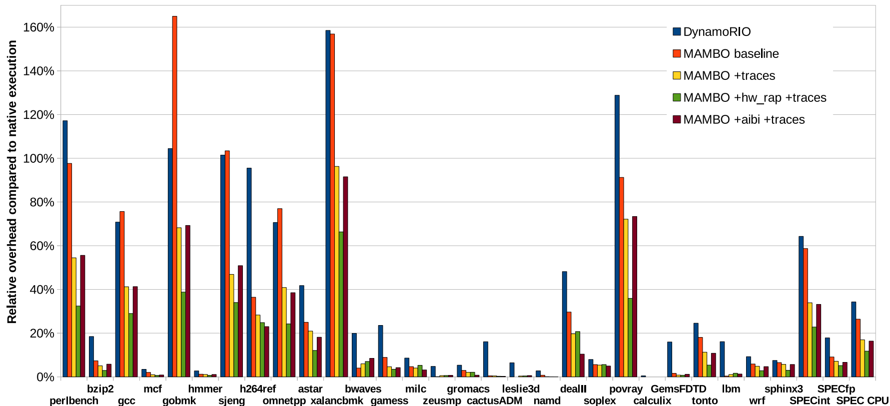


Figure: Relative overhead for SPEC CPU2006 on the APM X-C1 system

MANCHESTER 1824
The University of Manchester

Evaluation [2]

		SPEC CPU overhead	
Hardware platform	MAMBO config.	Geomean	Worst case
ODROID-XU3 (LITTLE)	baseline	26%	136%
in-order Cortex-A7	+traces	21%	75%
	+hw_rap +traces	19%	69%
ODROID-X2	baseline	30%	141%
OOO Cortex-A9	+traces	17%	71%
	+aibi + traces	15%	66%
Tronsmart R28	baseline	29%	159%
OOO Cortex-A17	+traces	17%	75%
	+aibi +traces	16%	67%
Jetson TK1	baseline	35%	177%
OOO Cortex-A15	+traces	23%	100%
	+hw_rap +traces	21%	80%
APM X-C1	baseline	26%	165%
OOO X-Gene1	+traces	17%	96%
	+hw_rap +traces	12%	66%

MANCHESTER 1824
The University of Manchester

Summary

- DBM is a powerful technique for microarchitectural simulation, program analysis and debugging
  - however, it introduces a performance overhead
- 3 optimisations:
  - Hardware-assisted return address prediction
  - Adaptive indirect branch inlining
  - Traces
- Worst case overhead significantly reduced:
  - from 136-177% to 66% - 80%
- Implemented for MAMBO:
  `https://github.com/beehive-lab/mambo`

MANCHESTER 1824
The University of Manchester

MAMBO

- Fast DBM implementation for ARM (AArch32 and AArch64)
- Runs on GNU/Linux
- Open source, Apache 2.0 license
  - `https://github.com/beehive-lab/mambo`
- contributions are welcomed
  - bug reports & patches
  - sample plugins
  - feedback on the API
- *Cosmin Gorgovan, Amanieu d'Antras, and Mikel Luján. MAMBO: A Low-Overhead Dynamic Binary Modification Tool for ARM. TACO, Article 14 (April 2016)*