MANCHESTER 1824
The University of Manchester

Dynamic Binary Instrumentation and Modification with MAMBO

Cosmin Gorgovan
Guillermo Callaghan
Mikel Luján

School of Computer Science
University of Manchester

Defining key terms

- Dynamic - at runtime
- Binary - at the level of native code
- Dynamic Binary Modification (DBM)
  - altering applications at runtime, at the native code level
- (Software) Instrumentation
  - *the transformation of a program into its own measurement tool*
- Dynamic Binary Instrumentation (DBI)
  - DBM, when the modification consists of adding instrumentation code
- DBM / DBI system
  - software runtime implementing DBM / DBI

Example uses of DBM/DBI

- microarchitectural simulation
  - *Sniper Multi-Core Simulator, APTSim* (MAMBO-based)*
- cache simulation
  - *Valgrind Cachegrind, drcachesim, MAMBO cachesim*
- program analysis
  - *Valgrind Callgrind*
- memory error detection / debugging
  - *Valgrind Memcheck, Dr. Memory*

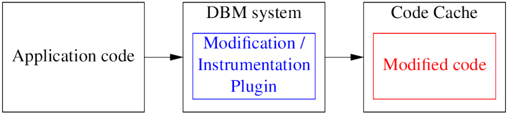
* *John Mawer, Oscar Palomar, Cosmin Gorgovan, Andy Nisbet, Will Toms, and Mikel Luján. 2017. The Potential of Dynamic Binary Modification and CPU-FPGA SoCs for Simulation. FCCM, 2017*

MANCHESTER 1824
The University of Manchester

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
- 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
- this introduces overheads
    - in particular a performance overhead



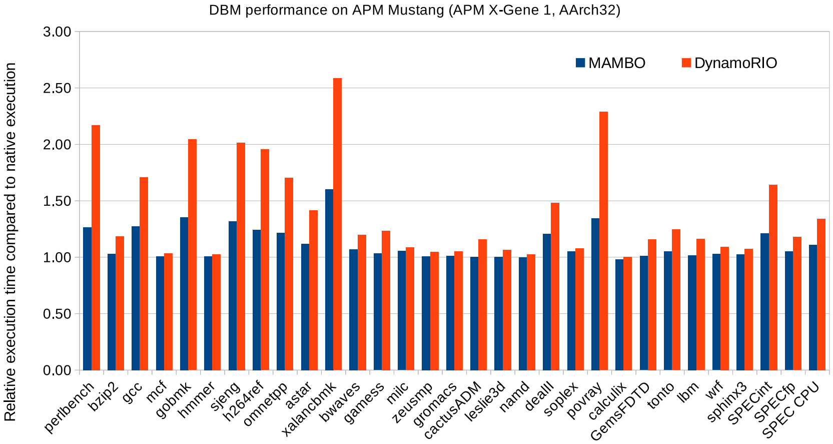
MAMBO

- Fast DBM implementation for Arm (AArch32 and AArch64)
- API for modification and instrumentation *plugins*
- 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
- VM image:
  `https://github.com/beehive-lab/mambo-vm`

Why MAMBO?

- small codebase: 16 kLoC (core) + 1.3kLoC (sample plugins)
- good compatibility with applications (and improving)
- allows analysis of app. code at the machine code level
  - useful for microarchitectural analysis and simulation
- the API allows trading off between performance, portability and ease of development
- good performance
  - the lowest base overhead among the DBM systems for Arm
  - good performance scaling for multithreaded applications

Why MAMBO? Low overhead



The MAMBO API

- Event-driven programming model
- Plugins typically handle:
  - Code analysis
  - Code generation, modification or instrumentation
  - Runtime event handling
- Functionality to implement common tasks with architecture-independent code
  - write-once for A32, T32, A64
- Allows access to the raw machine code
  - advanced code analysis
  - highly optimised code generation
- Multithreaded scaling by minimising synchronisation

Plugins distributed with MAMBO

- branch_count - dynamic execution counters for each type of branch (direct, indirect and returns)
- cachesim - configurable cache hierarchy simulator
- mtrace - records memory access traces
- soft_div - dynamically replaces AArch32 hardware divide instructions with an emulation routine
- **upcoming**: memcheck - detects & reports memory usage errors (e.g. buffer overflows, double frees)

Summary

- MAMBO - DBM implementation for Arm with low overhead and a small codebase
- *Cosmin Gorgovan, Amanieu d'Antras, Mikel Luján: MAMBO: A Low-Overhead Dynamic Binary Modification Tool for ARM. TACO 13(1): 14:1-14:26 (2016)*
- *Cosmin Gorgovan, Amanieu d'Antras, Mikel Luján: Optimising Dynamic Binary Modification Across ARM Microarchitectures. ICPE 2018: 28-39*
- `https://github.com/beehive-lab/mambo`
  - open source code, including plugins (Apache 2.0)
  - open access papers
  - API tutorial slides