MaxSim: A Simulator Platform for Managed Applications

DOI:
10.1109/ISPASS.2017.7975286

Document Version
Accepted author manuscript

Citation for published version (APA):
Rodchenko, A., Kotselidis, C., Nisbet, A., Pop, A., & Luján, M. (2017). MaxSim: A Simulator Platform for Managed Applications. In *IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS 2017* https://doi.org/10.1109/ISPASS.2017.7975286

Published in:
IEEE International Symposium on Performance Analysis of Systems and Software - ISPASS 2017

MaxSim: A Simulation Platform for Managed Applications

Andrey Rodchenko, Christos Kotselidis, Andy Nisbet, Antoniu Pop, and Mikel Luján
School of Computer Science, The University of Manchester, UK
Email: {Andrey.Rodchenko, Christos.Kotselidis, Andy.Nisbet, Antoniu.Pop, Mikel.Lujan}@manchester.ac.uk

***Abstract*—Managed applications, written in programming languages such as Java, C# and others, represent a significant share of workloads in the mobile, desktop, and server domains. Microarchitectural timing simulation of such workloads is useful for characterization and performance analysis, of both hardware and software, as well as for research and development of novel hardware extensions.**

This paper introduces MaxSim, a simulation platform based on the Maxine VM, the ZSim simulator, and the McPAT modeling framework. MaxSim is able to simulate fast and accurately managed workloads running on top of Maxine VM and its capabilities are showcased with novel simulation techniques for: 1) low-intrusive microarchitectural profiling via pointer tagging on the x86-64 platforms, 2) modeling of hardware extensions related, but not limited to, tagged pointers, and 3) modeling of complex software changes via address-space morphing.

Low-intrusive microarchitectural profiling is achieved by utilizing tagged pointers to collect type- and allocation-site- related hardware events. Furthermore, MaxSim allows, through a novel technique called address space morphing, the easy modeling of complex object layout transformations. Finally, through the co-designed capabilities of MaxSim, novel hardware extensions can be implemented and evaluated.

We showcase MaxSim's capabilities by simulating the whole set of the DaCapo-9.12-bach benchmarks in less than a day while performing an up-to-date microarchitectural power and performance characterization. Furthermore, we demonstrate a hardware/software co-designed optimization that performs dynamic load elimination for array length retrieval achieving up to 14% L1 data cache loads reduction and up to 4% dynamic energy reduction. MaxSim is available at https://github.com/arodchen/MaxSim released as free software.

I. Introduction

Managed Runtime Environments (MRE) have been widely adopted in a variety of computing domains ranging from mobile phones to enterprise servers. Managed languages, and Java in particular, have been utilized not only in application and middleware domains but also in system programming for the development of research prototypes such as the Maxine *Virtual Machine* (VM) [1], Jikes RVM [2], the Singularity operating system [3], the Graal compiler [4], and the Truffle [5] *Abstract Syntax Tree* (AST) interpreter.

The end of single-core scaling [6], [7] makes the achievement of further energy and performance improvements, solely by enhancements in *Hardware* (HW), an extremely challenging task. A way to address this challenge is to design domain-specific HW extensions for certain *Software* (SW) tasks in general, and for managed workloads in particular. In order to design HW extensions that address distinctive features of managed workloads, such as object orientation and *Garbage Collection* (GC), a specialized simulation platform is necessitated to improve research productivity. Such a platform must enable close integration of a fast and accurate microarchitectural simulator and a modern MRE, while providing a feedback loop between these two components. In this paper we present MaxSim: a simulation platform targeting managed applications.

MaxSim, in contrast to previous efforts, allows fast, accurate, and low-intrusive performance analysis of managed workloads by employing a novel pointer tagging scheme. Fast, accurate, and low-intrusive performance analysis is typically performed by utilization of HW counters [8], [9], which has three main limitations. First, the frequent accesses to HW counters can introduce performance overheads. Second, the association of collected events with high-level information related to managed workloads can be limited [10]. Finally, HW counters are not always portable between architectures and may not be complete for arbitrary purposes. Also in MaxSim, the simulator has an awareness of the VM, so it is able to distinguish what code is being executed (GC, non-GC) and what data is being accessed (thread local storage, stack, heap, code cache, native).

In detail, this paper contributes the following:

- **MaxSim – a novel experimental platform for HW/SW co-design exploration** on the basis of the state-of-the-art Maxine VM, the ZSim microarchitectural simulator [11], and the McPAT power, area, and timing modeling framework [12].
- **A novel pointer tagging scheme in x86-64 architectures** that is based on *Dynamic Binary Translation* (DBT) that: 1) allows the fast, accurate, and low-intrusive fine-grain microarchitectural profiling of managed workloads, and 2) enables the implementation of HW/SW co-designed optimizations, such as hardware-assisted retrieval of array lengths encoded in object pointers. In addition, the collected profiling information can be also loaded back to the Maxine VM, creating a full feedback loop between the simulator and the VM.
- **A novel address space morphing technique** for simulating complex software changes regarding object layout transformations such as fields expansion, contraction and reordering.

The techniques, implemented in MaxSim and described in this paper, are applicable to other simulators and runtime systems. However, the selection of the state-of-the-art Maxine VM and ZSim simulator provides a unique combination of research productivity, accuracy and speed of simulation.

The paper is organized as follows: Section II presents the background and describes the key components of MaxSim. It also presents the validation of ZSim on the DaCapo-9.12-bach benchmarks [13] executed by Maxine. Section III describes the MaxSim platform and introduces the novel simulation and optimization techniques. Section IV presents the use cases of the proposed platform. Finally, Section V presents the related work, while Section VI summarizes this paper. The experimental platform presented in this paper is available at https://github.com/arodchen/MaxSim released under the GPLv2 free software license.

II. Background

This section provides a comparison of the different research VMs and simulation techniques. It mainly focuses on the Maxine VM and the ZSim simulator, since they are the two main components of the introduced MaxSim platform.

A. Research VMs

MREs are complex SW systems typically consisting of a baseline compiler or an interpreter, an optimizing compiler, GC algorithms, facilities for thread synchronization, exception handling, deoptimization, and other functionalities. All the aforementioned components of a VM have been extensively studied by the research community. Ideally, a VM should be designed in such a way to allow the plug-in of different modules extending its research and optimization capabilities. Unfortunately, this is not always feasible since high performance and high degrees of modularity are two aspects that counteract each other. In order to achieve high performance, VMs are optimized across the components sacrificing modularity.

To that end, VMs broadly fall into two categories: production quality and research VMs. Production quality VMs such as the HotSpot JVM [14] can achieve high performance at the expense of limited experimentation capabilities due to the lack of modularity. On the contrary, research VMs such as the Jikes RVM [2] and Maxine VM [1], compared in Table I, although do not reach the performance goals of the HotSpot VM, offer higher degrees of freedom and enable higher productivity due to their modular design.

Maxine VM was chosen instead of Jikes RVM for the following reasons:

1) It supports the widely-adopted x86-64 architecture.
2) It is compatible with the JDK7 Class Libraries and can run to completion the full set of the DaCapo-9.12-bach [13], SPECjvm2008 [15], pjbb2005 [16] and other benchmarks.
3) It supports the Graal [4] optimizing compiler, which is the next-generation optimizing compiler of HotSpot JVM.

Research VM	**ISAs**	**Class Libraries**	**Support of Other Languages**
Jikes RVM	PowerPC, IA-32	Apache Harmony GNU Classpath	-
Maxine VM	x86-64, ARMv7	JDK 7	+ (via Graal and Truffle)

TABLE I: Research VMs comparison.

4) It supports the Truffle [5] optimizing AST interpreter, that allows the execution of other languages, apart from Java, such as JavaScript, R, Ruby, and others.

B. Maxine VM

The main design goals of Maxine VM are modularity and increased research productivity. Maxine VM consists of a number of interchangeable modules that are accessed through module interfaces, which are called *schemes*. The schemes describe heap and GC functionalities, object layouts, locking facilities, and other aspects of VMs.

In order to assess the performance of Maxine VM, we compare it against production-quality VMs. To that end, the Maxine VM[1] with its two optimizing compilers, C1X and Graal customized for Maxine[2], is compared against the production-quality HotSpot VM with its two optimizing compilers, C2 (ver. 1.8.0.25) and Graal[3]. The performance comparison of four `VM-compiler-version` triplets on the DaCapo-9.12-bach benchmarks[4] is presented in Figure 1, where performance is relative to `HotSpot-C2-1.8.0.25`. Whiskers represent 95% confidence intervals. As depicted in Figure 1, the performance of `HotSpot-Graal-21075` is comparable to `HotSpot-C2-1.8.0.25`, while the performance of the research-oriented `Maxine-Graal-8810.11558` and `Maxine-C1X-8810.11558` is 57% and 53% of `HotSpot-C2-1.8.0.25`, which is considered to be satisfactory for research purposes [1].

As already mentioned, the Maxine VM has two optimizing compilers, namely C1X and Graal. Theoretically, if the Maxine VM is optimized across its modules, its peak performance with Graal should be on-par with that of the HotSpot VM with the same compiler. From the performance results presented in Figure 1 we can see that `Maxine-Graal-8810.11558` is around 8% faster than `Maxine-C1X-8810.11558` in geomean. However, since C1X is much less complex than Graal and has much lower compilation times, C1X has been selected as the optimizing compiler of MaxSim. Regarding the baseline compiler, the T1X template compiler of the Maxine VM has been used in MaxSim.

C. Timing Simulation Techniques

Microarchitectural simulation presents a number of challenges that define trade-offs between simulation speed, simulation accuracy, and engineering efforts required to modify

[1]https://github.com/arodchen/maxine rev.8810

[2]https://github.com/arodchen/graal rev.11558

[3]http://hg.openjdk.java.net/graal/graal-compiler rev.21075

[4]`eclipse` is not present, as it did not pass on `Maxine-Graal-8810.11558`

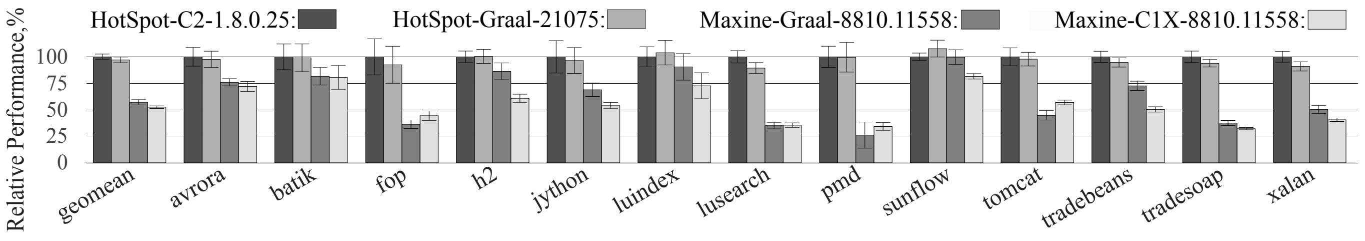


Fig. 1: Performance of different `VM-compiler-version` triplets relative to `HotSpot-C2-1.8.0.25` (higher is better).

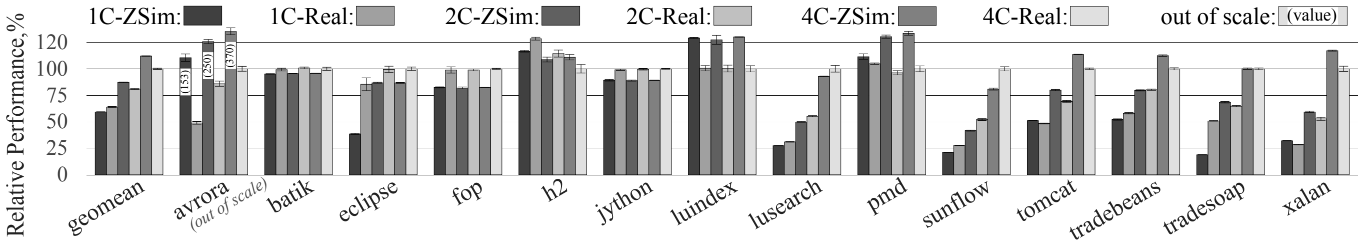


Fig. 2: Validation of different simulated HW configurations `*-ZSim` against real system configurations `*-Real`. The depicted performances are relative to `4C-Real` (higher is better).

or implement new HW timing models. FPGA-based simulators [17], [18], [19], [20] are the fastest, but their implementation or extension requires substantial engineering efforts.

SW-based simulators are easier to maintain than the FPGA-based ones, since they do not require special HW. SW-based simulators can be subdivided into two groups: full-system and user-level. The state-of-the-art open-source full-system simulators [21], [22] are more complex and, typically, slower (higher simulation times) than user-level ones. The benefit of using a full-system simulation is the extra accuracy achieved since more components of the computing stack are simulated. However, for workloads[5] that spend the vast majority of their time in user-level code, this is not the case.

The user-level SW-based simulators, such as [23], [24], [11], provide the best research trade-offs in terms of accuracy, simulation speed, and engineering effort sacrificing the ability to simulate the kernel code. From the currently available user-level simulators, only ZSim [11] allows the execution of arbitrary managed workloads via lightweight user-level virtualization. For that reason, ZSim was the simulator of choice for MaxSim.

D. ZSim Simulator

ZSim is an execution-driven simulator based on the Pin [25] dynamic binary instrumentation and modification tool. One of the design goals of this simulator is scalability, which is achieved via the "bound-weave" simulation parallelization technique. With minor modifications to its user-level virtualization and scheduling techniques[6], ZSim was capable of simulating the full set of the DaCapo benchmarks executed by the Maxine VM with the C1X optimizing compiler. The parameters of the simulated systems referenced in this paper are described in Table II. The configurations `1C`, `2C`, and `4C` represent the Intel Nehalem microarchitecture with 1, 2, and 4 enabled cores respectively. Furthermore, `1CQ` represents the 1-core CPU with just a **Q**uarter of the 8MB *Last Level Cache* (LLC). We use that configuration in order to simulate the case when only a quarter of the available `4C` resources is available to the workload (if the LLC could be partitioned).

Name		`1C`	`2C`	`4C`	`1CQ`
Cores	type	x86-64 Nehalem OOO core at 2.66 GHz			
	total	4			1
	enabled	**1**	**2**	**4**	**1**
Prefetchers		disabled			
L1I Caches		32KB, 4-way, LRU, 3-cycle latency			
L1D Caches		32KB, 8-way, LRU, 4-cycle latency			
L2 Caches		256KB, 8-way, LRU, 6-cycle latency			
L3 Cache	type	16-way, hashed, 30-cycle latency			
	size	8MB			2MB
Memory Controller		1, 3 DDR3 channels, 47-cycle latency			
DRAM		3GB, DDR3-1066, 1GB DIMM per channel			

TABLE II: ZSim configurations.

We validated ZSim against a real system with the results presented in Figure 2. The performance of the simulated models `1C-ZSim`, `2C-ZSim`, `4C-ZSim` is validated against the performance of the real systems `1C-Real`, `2C-Real`, `4C-Real` respectively, where `Real` represents an Intel Core i7 920 (Bloomfiled) CPU based on the Nehalem microarchitecture. The performance shown is relative to the `4C-Real` configuration. Whiskers represent 95% confidence intervals. It can be seen that the difference in geomean execution times between the real platform and the simulated models is from 8% to 12%, which is in alignment with the ZSim original validation [11]. Furthermore, the performance scalability pattern (from 1 core to 4 cores) of the simulated models is consistent with the real system. However, two major inconsistencies were observed. Firstly, the execution times of

[5]The DaCapo benchmarks with the exception of `avrora` are such an example.

[6]https://github.com/arodchen/zsim rev.102

eclipse and tradesoap on the one-core model 1C-ZSim were more than two times greater than the real system 1C-Real. This is due to the different thread scheduling algorithms used: on the real system a *Completely Fair Scheduling* (CFS) [26] scheme is employed while on ZSim a simple round-robin scheduling is used. Secondly, the avrora test on the Maxine VM spends more than half of its execution time in the Linux kernel on *-Real configurations. However, ZSim is capable of simulating only user-level code, significantly over-estimating avrora's performance.

III. MaxSim: A Simulation Platform for Managed Applications

In this section, we describe, in detail, the novel features of MaxSim along with its capabilities. These features are: 1) pointer tagging that can be used for light-weight object-based microarchitectural profiling and/or HW/SW co-designed optimizations, 2) integration with the McPAT framework for power and energy estimations, and 3) the address space morphing technique allowing the easy modeling and performance/power estimation of complex object layout transformations.

A. Pointer Tagging

A pointer tag is a number of bits of an address which are ignored during memory access operations. In general, the main use cases of tagged pointers are: 1) capability-based addressing [27], [28] and security [29], [30], which can also require tagged memory, and 2) storage of type information [31], [32], [33]. The shift from 32-bit to 64-bit architectures enables 16 exabytes of memory to be addressable, a number which significantly exceeds the amount of memory needed for applications targeting these architectures. This fact motivated the support for tagged pointers in modern commodity architectures: AArch64 with 8-bit pointer tags [34] and Sparc M7 with up to 32-bit pointer tags [35].

Although x86-64 architectures do not currently support tagged pointers [36] (Sect. 3.3.7.1), the virtual addressing is currently limited to 48 bits[7] with the high 16 bits replicating bit 47. MaxSim exploits these high 16 bits on x86-64 architectures, to encode extra information that can be interpreted during simulation for various purposes. The main use case is the assignment of extra information to an object via its pointer. This extra information can regard either associations with high-level language features (Section III-A1) or other metadata for HW/SW co-designed optimizations (Section III-A2).

Typically, associating extra information with objects poses a trade-off between extra required memory and access time. Figure 3 presents three options for the storage of object metadata. The first option is "in object storage", where the metadata is stored inside an object in an intrusive manner which also increases memory footprint. The second option is "associative array storage" which requires both extra space and lookup time to retrieve metadata. To that end, if metadata is accessed read-mainly and frequently (on every memory access

[7]In the upcoming version of the architecture, the virtual addressing will be extended to 57 bits [37].

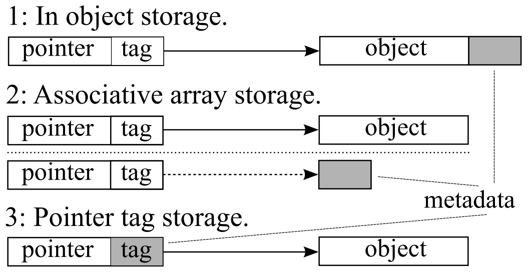


Fig. 3: Different options for object metadata storage.

operation) and the amount of metadata to be stored can fit in 16 bits, "pointer tag storage" is preferable which is the third option. Encoding metadata into the available 16 bits of an object's address saves memory bandwidth and reduces access latency.

To enable tagged pointers support in MaxSim, the following three invariants must be preserved in Maxine VM:

1) All pointers to the same object must be tagged with the same tag.
2) When a field inside an object is accessed, [tag:base + (index * scale) + disp] addressing mode must be used, where base points to the beginning of the object and (index * scale) + disp represents an offset (later on, this will be referred to as [tag:base + offset]).
3) An object pointer tag is immutable between any following adjacent points in an object's lifetime: object allocation, initialization, and evacuation during GC.

The first invariant allows the comparison of tagged pointers without extensive VM modifications, while the second invariant allows an accessed object's class field to be identified using this canonical form. The third invariant implies that the pointer tag can only be changed in certain places, where all pointers to an object to be tagged are accessible without a full scan of all objects. All live objects are untagged during a stop-the-world VM operation when switching to the ZSim fast forwarding mode. During the ZSim fast forwarding mode, execution happens without simulation and extensive binary modification/instrumentation at near-native speed until entering the next *Region Of Interest* (ROI) for simulation. Untagged object pointers are tagged back during the stop-the-world VM operation when entering the next ROI and switching back from the fast forwarding to the normal simulation mode.

Finally, ZSim simulation is based on the Pin dynamic binary instrumentation and modification tool, and pointers' tag detection and untagging is performed via the API shown in Figure 4. To summarize, pointer tagging allows to: 1) perform light-weight object-based microarchitectural profiling and, 2) perform a number of HW/SW co-designed optimizations by encoding data in tagged pointers.

1) Light-weight Object-based Microarchitectural Profiling: Simulation-based profiling, an important technique in performance analysis, is one of the key features of MaxSim. In order to enable this functionality, it is essential to bind mi-

```
// Returns true if instruction operand is a memory reference.
BOOL INS_OperandIsMemory(INS ins, UINT32 n);

// Returns base, index, scale, displacement
// (address = base + disp + index * scale).
REG INS_OperandMemoryBaseReg(INS ins, UINT32 n);
REG INS_OperandMemoryIndexReg(INS ins, UINT32 n);
UINT32 INS_OperandMemoryScale(INS ins, UINT32 n);
INT64 INS_OperandMemoryDisplacement(INS ins, UINT32 n);

// Rewrites memory operand to reference the virtual
// memory location contained in the given register.
VOID INS_RewriteMemoryOperand(INS ins, UINT32 memIndex,
      REG reg);
```

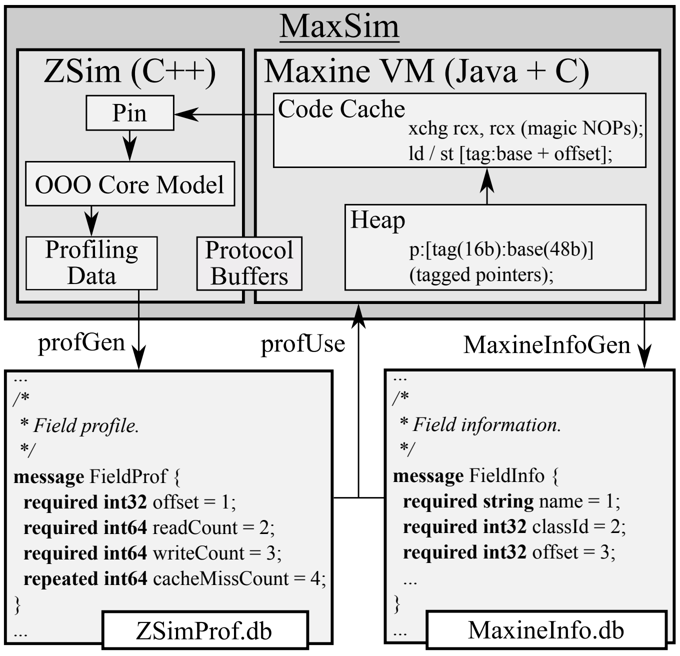
Fig. 4: Pin API for tag pointers retrieval and untagging.

croarchitectural events with high-level language information. This binding is achieved via the pointer tagging mechanism described in the previous section.

The Maxine VM assigns a tag to a pointer, and ZSim collects events related to this tag during memory access operations. MaxSim currently supports several implementations of language information association with object pointers among which are: `ClassIdTagging` and `AllocationSiteIdTagging`. `ClassIdTagging` assigns object class IDs to all object pointers allowing the association of microarchitectural events per class. A class ID is a compact unsigned integer representing the class of an object and is usually stored in the class information object which is accessible via a pointer stored in an object's header. By storing class ID in the pointer tag, we manage to save two load operations at the expense of untagging and tag retrieval, which are two and one shift operations, respectively. `AllocationSiteIdTagging` assigns allocation site IDs to object pointers. An allocation site ID is a compact unsigned integer representing a pair of an allocation site estimation of an object and a class ID. Allocation site IDs are requested from ZSim via magic NOP operations [38], which have NOP semantics during non-simulated execution. On each allocation site ID request, ZSim returns a compact ID, which is associated with an allocated object's class ID and an allocation site estimation using stack trace estimation in ZSim. Stack trace estimation is performed using per-thread circular buffers by pushing return addresses on function calls and popping them on function returns.

The state-of-the-art techniques to associate allocation sites with objects usually require either hashing [39] or storage of extra information in or adjacent to objects [40]. Such techniques introduce noticeable overheads and interference with a normal workload execution. In comparison with the aforementioned techniques, the proposed technique is much less intrusive, as it takes just a few lightweight operations during an object allocation to set a tag.

Figure 5 shows the integration scheme of MaxSim and the flow of profiling information between its components. The profiling information is stored in the Protocol Buffers



```
...
/*
 * Field profile.
 */
message FieldProf {
  required int32 offset = 1;
  required int64 readCount = 2;
  required int64 writeCount = 3;
  repeated int64 cacheMissCount = 4;
}
...
```

```
...
/*
 * Field information.
 */
message FieldInfo {
  required string name = 1;
  required int32 classId = 2;
  required int32 offset = 3;
  ...
}
...
```

Fig. 5: Handling of profiling information in MaxSim.

format [41], and it consists of two parts. The first part, stored in the `ZSimProf.db` file, contains microarchitectural events collected by ZSim. Examples of such events (memory accesses and cache misses) related to a class field are shown in Figure 5. The second part, stored in the `MaxineInfo.db` file, contains information necessary to bind collected events to high-level language information. In Figure 5, for example, field information (name, class ID, offset) is represented. In case there are several ROIs during the same simulation, several `ZSimProf.db` files and a single `MaxineInfo.db` file will be generated.

The profiling is performed during memory access operations, and collected events are associated with triplets of an instruction pointer, a pointer tag, and a memory address offset. Allocation site IDs are reported from ZSim to Maxine via magic NOPs with an allocation site ID stored in the `rcx` register by ZSim.

The detailed collected information can later be uploaded to Maxine VM to guide optimizations, or it can be printed in a textual format. The snippet of the textual output is presented in Figure 6. In this example, `AllocationSiteIdTagging` was active, and the profiling information is shown for objects of `HashMap$Entry[]` class allocated during a call to `HashMap.<init>` method at offset `354` (in the constructor of HashMap). In total, `1` object of `88` bytes and `19` objects of `152` bytes were allocated reaching a total allocation footprint of `2976` bytes. Furthermore, `983` memory accesses were performed with `11` L3 cache read misses and `7` L3 cache write misses. At offset `80`, `33` reads and `9` writes were performed with `9` L3 cache read misses. Finally, all `9` misses at offset `80` occurred at offset `107` of method `HashMap.put`. The presented tagged-based profiling scheme is especially useful

```
// Memory access profiling.
java.util.HashMap$Entry[](...)@
   [java.util.HashMap.<init>(...)+354(...)]
      (asi:0 mf:2976(s:152(19) s:88(1)) ac:983 ... l3rm:11 l3wm:7):
         (o:80 r:33 w:9 ... l3rm:9 l3wm:0)

// L3 cache miss reads profiling.
[java.util.HashMap.put(Object, Object)+107(...)]
   (m:9 asi:0 ol:80 oh:80)
```

Fig. 6: Snippet of profiling information textual output.

for profiling object-oriented SW in which objects can be relocated (*e.g.* copying garbage collection), as pointer tags preserve objects' identities for profiling.

2) HW/SW Co-designed Optimizations Enabled by Tagged Pointers: The presence of available bits, when tagged pointers are enabled, creates a number of HW/SW co-designed optimization opportunities. It is possible to encode some information related to an object in a pointer tag and to extend functionality of memory access operations via a tag for performance/power optimizations or security enhancements. An example of such an optimization is related to array length encoding in tags and is one of the use-cases of this paper. Its evaluation is presented in Section IV-B.

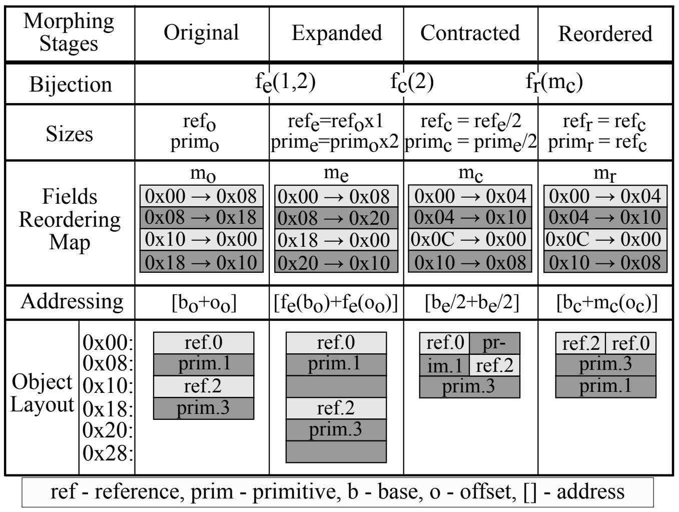
B. Integration with the McPAT Framework

To be able to perform energy estimations, we integrated the energy estimation model (which uses McPAT) from the Sniper simulator [42] for the same microarchitecture simulated by ZSim. Conversion of microarchitectural events from the ZSim to Sniper format was adopted from the ZSim-NVMain simulator [43]. The modeling tool required the collection of a number of extra microarchitectural events in ZSim such as the number of predicted branches and floating point microoperations.

C. Simulator/VM Co-operative Address Space Morphing

For many managed languages in general, and for Java in particular, layouts of objects in memory are not specified and depend on the VM implementation. Changing layouts of objects can improve cache locality and decrease memory footprint. However, such transformations are difficult to implement without adding extra complexity or breaking the modularity of a VM. MaxSim implements a novel address space morphing technique to perform simulation of complex object layout transformations, specifically fields expansion, contraction, and reordering.

As shown in Figure 7, the proposed technique is a co-operative multi-stage object layout transformation. Furthermore, it leverages the flexibility of Maxine VM to expand object fields and the ability of ZSim to remap memory addresses during simulation. Thus, in order to perform fields reordering and contraction by a factor of `N`, the following three stages are performed: 1) all fields except from those to be contracted are expanded by a factor of `N` by Maxine VM, 2) ZSim contracts the heap by a factor of `N` via address



Morphing Stages		Original	Expanded	Contracted	Reordered
Bijection			$f_e(1,2)$	$f_c(2)$	$f_r(m_c)$
Sizes		ref_o $prim_o$	ref_e=ref_ox1 $prim_e$=$prim_o$x2	ref_c = ref_e/2 $prim_c$ = $prim_e$/2	ref_r = ref_c $prim_r$ = ref_c
Fields Reordering Map		m_o 0x00 → 0x08 0x08 → 0x18 0x10 → 0x00 0x18 → 0x10	m_e 0x00 → 0x08 0x08 → 0x20 0x18 → 0x00 0x20 → 0x10	m_c 0x00 → 0x04 0x04 → 0x10 0x0C → 0x00 0x10 → 0x08	m_r 0x00 → 0x04 0x04 → 0x10 0x0C → 0x00 0x10 → 0x08
Addressing		$[b_o+o_o]$	$[f_e(b_o)+f_e(o_o)]$	$[b_e/2+b_e/2]$	$[b_c+m_c(o_c)]$
Object Layout	0x00: 0x08: 0x10: 0x18: 0x20: 0x28:	ref.0 prim.1 ref.2 prim.3	ref.0 prim.1 ref.2 prim.3 	ref.0 pr- im.1 ref.2 prim.3	ref.2 ref.0 prim.3 prim.1

ref - reference, prim - primitive, b - base, o - offset, [] - address

Fig. 7: Example of address space morphing in MaxSim.

space remapping, and 3) ZSim remaps the offsets of the fields according to the provided reordering map.

In the example of Figure 7, the original object layout has two reference fields, `ref.0` and `ref.2`, and two primitive fields, `prim.1` and `prim.3` (the leftmost object layout). During simulation, it is morphed in three stages to the new layout (the rightmost object layout) which results in its fields being reordered, as described by the m_o reordering map, and its references being contracted by a factor of `2`. In order to perform such transformations, four parameters to three bijections are provided. The first bijection f_e from the `Original` to the `Expanded` space takes two arguments: `1` - expansion factor for references, `2` - expansion factor for primitives. The transformation defined by this bijection is performed via changing layouts of objects in Maxine VM. The fields reordering map m_o is also modified according to this bijection. The second bijection f_c from the `Expanded` to the `Contracted` space takes the contraction factor as its argument. This transformation is performed in ZSim by dividing by `2` bases and offsets of memory access operations to objects. Furthermore, the fields reordering map m_e is modified by dividing by `2` all to-offsets. The third bijection f_r from the `Contracted` to the `Reordered` space takes the reordering map m_c from the `Contracted` stage and performs fields reordering according to this map resulting in the simulation of the desired layout. Heap and thread-local allocation buffer sizes are also doubled in Maxine VM on the `Expanded` stage.

Another issue that should be considered during simulator/VM co-operative address space morphing is expanded objects copying and initialization. After expanding primitives twice in Maxine VM, it will take twice as many dynamic instructions to perform copying or initialization than it would take in the case of the final layout presented in the example. This issue is handled via filtering during simulation of execution of object copying and initialization which happens in a loop. In this loop, every second iteration is omitted from the timing simulation. The indication that loop filtering should be

```
static void setWords( Pointer p, int numWords, Word val) {
  // loop prologue
  zsimMagicOp( FILTER_LOOP_BEGIN, p);
  for ( int i = 0; i < numWords; i++ ) {
    p.writeWord( i * WORD_SIZE, val);
  }
  // loop epilogue
  zsimMagicOp( FILER_LOOP_END);
}
```

Fig. 8: Example of loop iterations filtering.

enabled or disabled is performed by the VM via magic NOP operation in the loop's prologue and epilogue respectively. An example of such loop, with filtered iterations, is shown in Figure 8.

In order to validate our proposed address space morphing simulation technique, the following experiment was performed. Both references and primitives of heap objects were expanded twice in the Maxine VM via the bijection $\texttt{f}_e\texttt{(2,2)}$. During simulation in ZSim, memory accesses to expanded fields are projected back to original unexpanded address space (by contracting twice) via the bijection $\texttt{f}_c\texttt{(2)}$, thus simulating the original object layout[8]. The execution times were compared to the simulation of the original object layout, and the measured execution time geomean difference was less than 1% for the DaCapo benchmarks validating the proposed technique.

The simulation of objects' fields reordering transformation via address space morphing is driven by a configuration file passed to MaxSim in the Protocol Buffers format, presented in Figure 9. Fields reordering is described by the `typeDesc` of the type to be simulated having a different layout. The objects to be simulated as having an alternative layout are tagged by a `transTag`. On memory accesses to objects tagged by a `transTag`, address remapping is done during simulation by using an associative array represented by `fieldOffsetRemapPairs`, replacing matching `fromOffset` by `toOffset` during simulation. This technique allows fast experimentation with various objects layouts. It also allows to have different layouts of objects of a superclass and its subclasses so that the same field can have different offsets in them.

Expansion and contraction of references and primitives via address space morphing allow simulating ordinary object pointers compression [44] in MaxSim. An example of another transformation which could be implemented and simulated via the presented technique is a replacement of precisions and sizes of certain fields to different ones (long to int or double to float). To summarize, address space morphing allows to evaluate the performance impact of changing the order and/or size of fields.

[8]In this experiment no fields reordering was performed.

```
// Fields offset remapping pair.
message FieldOffsetRemapPair {
    required int32 fromOffset = 1;
    required int32 toOffset = 2;
}

// Data transformation information.
message DataTransInfo {
  required string typeDesc = 1;
  required int32 transTag = 2;
  repeated FieldOffsetRemapPair fieldOffsetRemapPairs = 3;
}
```

Fig. 9: Configuration file in the Protocol Buffer format driving fields reordering transformation simulation.

IV. Use Cases

This section will present two use cases of MaxSim. The first one regards the microarchitectural characterization of the Dacapo benchmarks. The second use-case showcases simulation of the architectural extensions related to the retrieval of array lengths stored in pointer tags.

A. Characterization of the DaCapo Benchmarks

MaxSim was able to simulate the whole set of the Dacapo-9.12-bach benchmarks in less than a day, with the results depicted in Figures 10 and 11. During the characterization we used two of the configurations of Table II: `1CQ` and `4C`. Figure 10 shows the L2 and L3 *Load Cache Misses Per Kilo Instruction* (LCMPKI) for both configurations. As shown, the majority of the Dacapo benchmarks are not cache-miss-intensive, which corresponds with the previous findings [45]. Figure 11 contains the information on *Instructions Per Clock* (IPC) and *Consumed Power* (CP). The geomean IPC is close to 1.4, while the CP is between 10 and 60 watts depending on the configuration. Hatched parts of the bars in Figures 10 and 11 represent parts of the presented metrics related to *Garbage Collection* (GC).

B. Evaluation of the HW/SW Co-designed Optimization Related to Array Length Encoding into Array Object Pointers' Tags

Implementations of managed languages associate array lengths with array objects allowing them to perform array bound checks at runtime. A common way of storing an array length is inside an array object at some constant offset from a base pointer. In Maxine VM, array lengths are stored at offset `0x10` of an array object and can be in the range of $[0; 2^{31}-1]$.

Having 16-bit pointer tags, it is possible to store a range of array lengths $[0; 2^{16} - 2]$. The value $2^{16} - 1$ serves as a *Not an Array Length* (NaAL) indicator. The retrieval of an array length can be performed via the method shown in Figure 13. In our evaluation, this code is emitted in seven instructions of 25 bytes size with an average execution height of 5.5 instructions. On the contrary, the baseline scheme utilizes just one instruction of three bytes size.

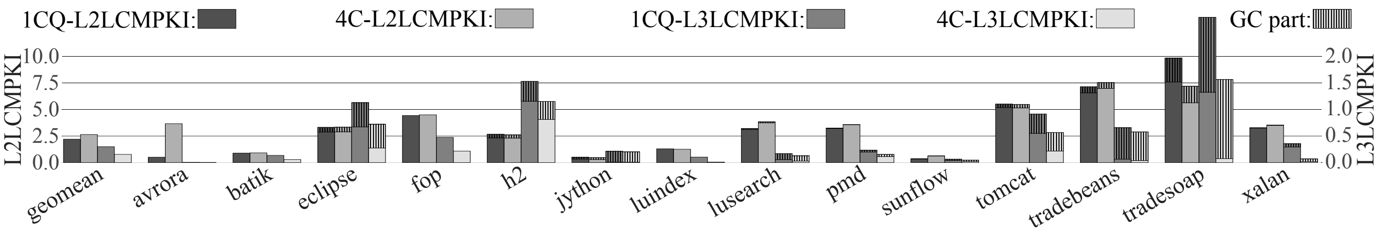


Fig. 10: L2 and L3 Load Cache Misses Per Kilo Instruction (LCMPKI) on the DaCapo-9.12-bach benchmarks on MaxSim.

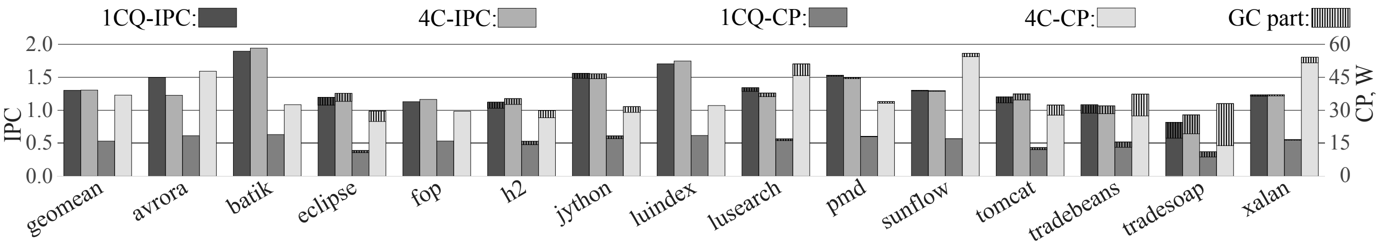


Fig. 11: Instructions Per Clock (IPC) and Consumed Power (CP) on the DaCapo-9.12-bach benchmarks on MaxSim.

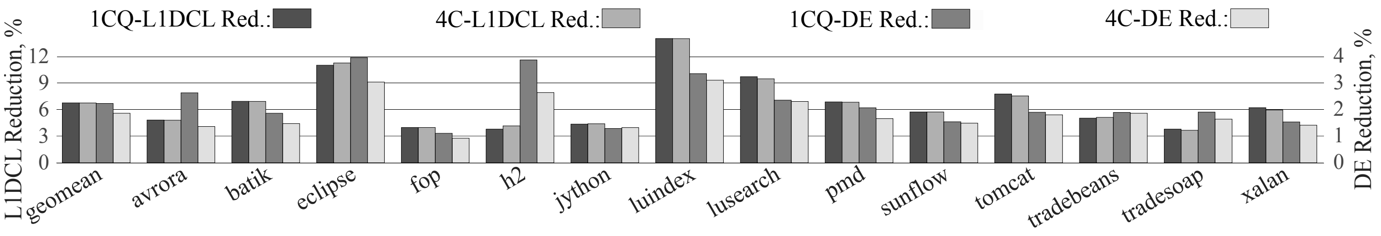


Fig. 12: L1 Data Cache Loads (L1DCL) and Dynamic Energy (DE) Reductions on the DaCapo-9.12-bach benchmarks after employing the HW/SW co-designed optimization related to array length tagging.

```
// Retrieving array length.
int retrieveArrayLength( Address_t objectAddress) {
   TAG_t tag = extractTAG( objectAddress);
   if ( tag != NaAL ) {
      return (int) tag;
   }
   return ∗ (int ∗) (objectAddress + 0x10));
}
```

Fig. 13: Array length retrieval with tagged pointers.

In order to perform the whole code snippet in just one instruction, corresponding to the last return statement in Figure 13, we propose the HW extension shown in Figure 14. The presented HW extension relies on the invariant, preserved by the VM, that the array length field of an array object is always accessed via a `[tag:base+offset]` addressing mode. Furthermore, the `ArrayLengthTagging` scheme has to be enabled. In this scheme, all non-array objects and arrays with lengths greater than $2^{16} - 2$ are tagged with the `NaAL` tag, while all the other array objects are tagged with their lengths. Thus, when an array length is accessed, the aforementioned tagged address pattern can be identified by the *Address Generation Unit* (AGU) in the proposed HW extension. Upon detecting an access to an array length field,

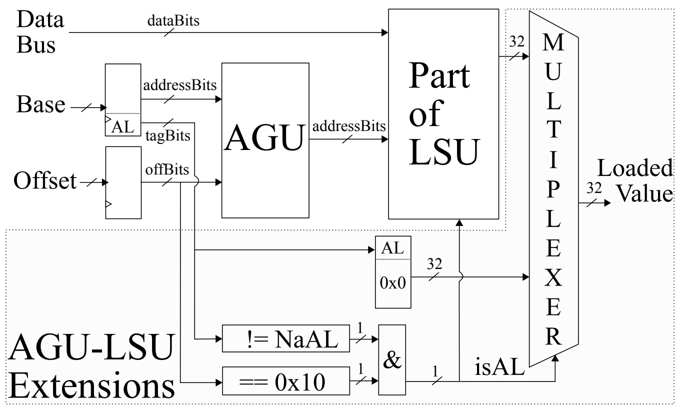


Fig. 14: Extensions to Address Generation Unit (AGU) and Load Store Unit (LSU) for array length retrieval from tagged pointers.

which is also encoded in a pointer tag, the `isAL` signal is set. Consequently, the value `AL` from the tag bypasses the *Load-Store Unit* (LSU) on its way to a consumer.

The values of the matching offset (`0x10`) and the matching tag (`NaAL`) for the presented AGU extension can be fixed or variable. In the latter case, these values can be set via a control register, making this scheme more general. If an array length is loaded from a pointer tag then we assume one cycle latency, which we model in the ZSim simulator.

We evaluate the proposed HW/SW co-designed optimization on the DaCapo-9.12-bach benchmarks on the `1CQ` and `4C` ZSim models of Table II. Figure 12 presents the results for *L1 Data Cache Loads* (L1DCL) and *Dynamic Energy* (DE) reductions. Although no significant performance gains were observed, the proposed technique resulted in up to 4% and 2% geomean dynamic energy reduction, and up to 14% and 7% geomean L1 data cache loads reduction.

V. Related Work

The closest platform [46] allowing user-level simulation of managed workloads is based on the Sniper multicore simulator [23] and the Jikes RVM [2]. The main limitation of this platform against MaxSim, is that it only supports 32-bit Jikes RVM and is not capable of running the full set of the DaCapo benchmarks. Regarding the simulator, Sniper uses the instruction-window centric *Out-Of-Order* (OOO) core model [47] with an average relative error of 11% for single-core and 21% for eight-core simulations on the SPLASH-2 benchmarks [48]. It is very close to ZSim's average relative error, which on a selection of tests from PARSEC [49], SPLASH-2, and SPEC OMP2001 [50] is 10% for single-core and 11% for six-core simulations. The tandem of Sniper and Jikes was used to explore a number of HW/SW co-designed techniques. These techniques improve memory bandwidth and reduce power and energy consumption by preventing write backs of cache lines containing parts of dead objects and by preventing fetches-on-writes while initializing cache lines containing parts of newly allocated objects with zeros [51].

The platform described in [52] is based on the Hotspot JVM and the full-system Simics simulator [53]. It does not require any changes to the Hotspot JVM and it can be very helpful in non-disruptive simulation-based performance analysis. It has high visibility of the Java high-level information (with the exception of thread and stack state). The design goal of that platform was to decouple it as much as possible from the concrete JVM implementation via a clear interface. Our platform, on the contrary, followed the co-design approach of the VM and the simulator development to facilitate extra functionality.

The ZSim simulator is written in C++, and communication of high-level information with the Maxine VM happens via Protocol Buffers. If the simulator was written in Java, the communication between the two components could have happened via reflection. The simulator called Tejas [54] is written in Java and can run on any platform the Java VM can execute. However, it has two limitations: firstly, it is a trace-driven simulator, and secondly, it uses an intermediate virtual ISA, which can introduce inaccuracy.

The *Virtual Performance Analyzer* (VPA) framework [55] follows the approach of partial selective simulation of HW-SW interaction and uses a cycle-approximate model. The motivation of this approach is the observation that I/O operations are sensitive to delays, and a simulation speed above 10 MIPS should be preserved not to alter the behavior of the program. The ZSim simulator solves this problem via the lightweight user-level virtualization technique, achieving for the OOO model an average simulation speed of 12 MIPS (in our experiments).

Introspection of target agnostic JIT compilation in the Smalltalk VM on top of gem5 [56] was shown to be useful for debugging and power/performance analysis. However, gem5 has a low simulation speed of 200 KIPS. Moreover, with Graal [4] and Truffle [5] it could be possible to run Smalltalk and other managed languages on the presented platform in future.

VI. Conclusion

In this paper, we have presented MaxSim: a novel and open-source experimental platform for HW/SW co-design research and characterization of managed workloads. MaxSim is based on the state-of-the-art Maxine VM, the ZSim microarchitectural simulator, and the McPAT power, area, and timing modeling framework. MaxSim features the simulation of 16-bit-tagged pointers, which were utilized for: 1) low-intrusive memory access profiling, 2) tagged pointers modeling on x86-64 architectures, and 3) experimenting with novel HW/SW co-designed optimizations by extending semantics of memory access operations via pointer tagging. In addition, the address-space morphing technique was presented, which allows modeling and simulation of complex software changes, such as compressed object pointers optimization and other data layout transformations. We showcased MaxSim's capabilities by: 1) performing an up-to-date microarchitectural characterization of the full set of the Dacapo benchmarks in less than a day, and 2) presenting a novel HW/SW co-designed optimization that performs dynamic load elimination for array length retrieval achieving up to 14% L1 data cache loads reduction and up to 4% dynamic energy reduction. MaxSim is available at https://github.com/arodchen/MaxSim released as free software.

Acknowledgements

This work is partially supported by EPSRC grants PAMELA EP/K008730/1, DOME EP/J016330/1, and EU Horizon 2020 ACTiCLOUD 732366 grant. A. Rodchenko is funded by a Microsoft Research PhD Scholarship, A. Pop is funded by a Royal Academy of Engineering Research Fellowship, and M. Luján is funded by a Royal Society University Research Fellowship.

References

[1] C. Wimmer, M. Haupt, M. L. Van De Vanter, M. Jordan, L. Daynès, and D. Simon, "Maxine: An approachable virtual machine for, and in, Java," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 30:1–20:24, Jan. 2013.

[2] B. Alpern, C. R. Attanasio, A. Cocchi, D. Lieber, S. Smith, T. Ngo, J. J. Barton, S. F. Hummel, J. C. Sheperd, and M. Mergen, "Implementing Jalapeño in Java," in *Proceedings of the 14th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 1999, pp. 314–324.

[3] J. Larus and G. Hunt, "The Singularity system," *Communications of the ACM*, vol. 53, no. 8, pp. 72–79, Aug. 2010.

[4] "OpenJDK: Graal project," http://openjdk.java.net/projects/graal/, 2016, [Online; last accessed 10-Mar-2017].

[5] C. Wimmer and T. Würthinger, "Truffle: A self-optimizing runtime system," in *Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity (SPLASH)*, 2012, pp. 13–14.

[6] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, pp. 114–117, April 1965.

[7] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct 1974.

[8] P. F. Sweeney, M. Hauswirth, B. Cahoon, P. Cheng, A. Diwan, D. Grove, and M. Hind, "Using hardware performance monitors to understand the behavior of Java applications," in *Proceedings of the 3rd Conference on Virtual Machine Research And Technology Symposium*, 2004, p. 5.

[9] M. Hauswirth, P. F. Sweeney, A. Diwan, and M. Hind, "Vertical profiling: Understanding the behavior of object-oriented applications," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2004, pp. 251–269.

[10] A. Georges, D. Buytaert, L. Eeckhout, and K. De Bosschere, "Method-level phase behavior in Java workloads," in *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 2004, pp. 270–287.

[11] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, 2013, pp. 475–486.

[12] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 1, pp. 5:1–5:29, Apr. 2013.

[13] S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khang, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks: Java benchmarking development and analysis," in *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA)*, 2006, pp. 169–190.

[14] M. Paleczny, C. Vick, and C. Click, "The Java Hotspot$^{\text{TM}}$server compiler," in *Proceedings of the 1st Java Virtual Machine Research and Technology Symposium*, 2001, pp. 1–12.

[15] "SPECjvm2008 benchmarks," http://www.spec.org/jvm2008, 2008, [Online; last accessed 10-Mar-2017].

[16] "pjbb2005," http://users.cecs.anu.edu.au/~steveb/research/research-infrastructure/pjbb2005, 2005, [Online; last accessed 10-Mar-2017].

[17] M. Pellauer, M. Adler, M. A. Kinsy, A. Parashar, and J. S. Emer, "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing." in *Proceedings of the 17th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2011, pp. 406–417.

[18] A. Khan, M. Vijayaraghavan, S. Boyd-Wickizer, and Arvind, "Fast and cycle-accurate modeling of a multicore processor," in *Proceedings of the 2012 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2012, pp. 178–187.

[19] D. Chiou, D. Sunwoo, J. Kim, N. A. Patil, W. Reinhart, D. E. Johnson, J. Keefe, and H. Angepat, "FPGA-accelerated simulation technologies (FAST): Fast, full-system, cycle-accurate simulators," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2007, pp. 249–261.

[20] E. S. Chung, M. K. Papamichael, E. Nurvitadhi, J. C. Hoe, K. Mai, and B. Falsafi, "ProtoFlex: Towards scalable, full-system multiprocessor simulations using FPGAs," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 2, no. 2, pp. 15:1–15:32, Jun. 2009.

[21] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[22] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSS: A full system simulator for multicore x86 CPUs," in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011, pp. 1050–1055.

[23] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011, pp. 52:1–52:12.

[24] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *Proceedings of the 16th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2005, pp. 190–200.

[26] "CFS scheduler," https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt, 2014, [Online; last accessed 10-Mar-2017].

[27] R. S. Fabry, "Capability-based addressing," *Communications of the ACM*, vol. 17, no. 7, pp. 403–412, Jul. 1974.

[28] H. M. Levy, *Capability-Based Computer Systems*, 1984. [Online]. Available: http://homes.cs.washington.edu/~levy/capabook/

[29] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: A flexible information flow architecture for software security," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA)*, 2007, pp. 482–493.

[30] J. R. Crandall and F. T. Chong, "Minos: Control data attack prevention orthogonal to memory model," in *Proceedings of the 37th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2004, pp. 221–232.

[31] E. I. Organick, *Computer System Organization: The B5700/B6700 Series (ACM Monograph Series)*. Academic Press, Inc., 1973.

[32] B. Babayan, "E2K technology and implementation," in *Proceedings from the 6th International Euro-Par Conference on Parallel Processing (Euro-Par)*, 2000, pp. 18–21.

[33] M. E. Houdek, F. G. Soltis, and R. L. Hoffman, "IBM System/38 support for capability-based addressing," in *Proceedings of the 8th Annual Symposium on Computer Architecture (ISCA)*, 1981, pp. 341–348.

[34] "ARM Cortex-A series programmer's guide for ARMv8-A," http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf, 2015, [Online; last accessed 10-Mar-2017].

[35] K. Aingaran, S. Jairath, G. Konstadinidis, S. Leung, P. Loewenstein, C. McAllister, S. Phillips, Z. Radovic, R. Sivaramakrishnan, D. Smentek, and T. Wicki, "M7: Oracle's next-generation Sparc processor," *IEEE Micro*, vol. 35, no. 2, pp. 36–45, Mar 2015.

[36] "Intel 64 and IA-32 architectures software developers manual. volume 1: Basic architecture," http://download.intel.com/design/processor/manuals/253665.pdf, 2011, [Online; last accessed 10-Mar-2017].

[37] "5-level paging and 5-level EPT white paper," https://software.intel.com/sites/default/files/managed/2b/80/5-level_paging_white_paper.pdf, 2016, [Online; last accessed 10-Mar-2017].

[38] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, pp. 92–99, Nov. 2005.

[39] R. Odaira, K. Ogata, K. Kawachiya, T. Onodera, and T. Nakatani, "Efficient runtime tracking of allocation sites in Java," in *Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*, 2010, pp. 109–120.

[40] D. Clifford, H. Payer, M. Stanton, and B. L. Titzer, "Memento mori: Dynamic allocation-site-based optimizations," in *Proceedings of the 2015 International Symposium on Memory Management (ISMM)*, 2015, pp. 105–117.

[41] "Protocol Buffers - Google's data interchange format (ver. 2.6.1)," https://developers.google.com/protocol-buffers/, 2014, [Online; last accessed 10-Mar-2017].

[42] W. Heirman, S. Sarkar, T. E. Carlson, I. Hur, and L. Eeckhout, "Power-aware multi-core simulation for early design stage hardware/software co-optimization," in *Proceedings of the 21st International Conference*

on Parallel Architectures and Compilation Techniques (PACT), 2012, pp. 3–12.

[43] A. Armejach, A. Cristal, and O. S. Unsal, "Tidy cache: Improving data placement in die-stacked DRAM caches," in *Proceedings of the 2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2015, pp. 65–73.

[44] A. R. Adl-Tabatabai, J. Bharadwaj, M. Cierniak, M. Eng, J. Fang, B. T. Lewis, B. R. Murphy, and J. M. Stichnoth, "Improving 64-bit Java IPF performance by compressing heap references," in *Proceedings of the International Symposium on Code Generation and Optimization (CGO)*, 2004, pp. 100–110.

[45] H. Inoue and T. Nakatani, "Identifying the sources of cache misses in Java programs without relying on hardware counters," in *Proceedings of the 2012 International Symposium on Memory Management (ISMM)*, 2012, pp. 133–142.

[46] "Jikes–Sniper page in Sniper online documentation," http://snipersim.org/w/Jikes, 2014, [Online; last accessed 10-Mar-2017].

[47] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 11, no. 3, pp. 28:1–28:25, Aug. 2014.

[48] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA)*, 1995, pp. 24–36.

[49] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[50] V. Aslot, M. J. Domeika, R. Eigenmann, G. Gaertner, W. B. Jones, and B. Parady, "SPEComp: A new benchmark suite for measuring parallel computer performance," in *Proceedings of the International Workshop on OpenMP Applications and Tools: OpenMP Shared Memory Parallel Programming (WOMPAT)*, 2001, pp. 1–10.

[51] J. B. Sartor, W. Heirman, S. M. Blackburn, L. Eeckhout, and K. S. McKinley, "Cooperative cache scrubbing," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation (PACT)*, 2014, pp. 15–26.

[52] G. Wright, P. McGachey, E. Gunadi, and M. Wolczko, "Introspection of a Java^{TM}virtual machine under simulation," Tech. Rep., 2006.

[53] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[54] S. R. Sarangi, R. Kalayappan, P. Kallurkar, S. Goel, and E. Peter, "Tejas: A java based versatile micro-architectural simulator," in *25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Sept 2015, pp. 47–54.

[55] C.-H. Tu, H.-H. Hsu, J.-H. Chen, C.-H. Chen, and S.-H. Hung, "Performance and power profiling for emulated Android systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 19, no. 2, pp. 10:1–10:25, Mar. 2014.

[56] B. Shingarov, "Live introspection of target-agnostic JIT in simulation," in *Proceedings of the International Workshop on Smalltalk Technologies (IWST)*, 2015, pp. 5:1–5:9.