MaxSim: A Simulation Platform for Managed Applications

Open-source: https://github.com/beehive-lab/MaxSim

Andrey Rodchenko, Christos Kotselidis,
Andy Nisbet, Antoniu Pop, Mikel Lujan

Advanced Processor Technologies Group,
School Of Computer Science,
The University of Manchester
Overview

- What simulation platform for managed applications is needed and why?
- VM Selection Justification: Maxine VM
- Simulator Selection Justification: ZSim
- MaxSim: Overview and Features
- Use Cases: Characterization, Profiling, and HW/SW Co-design
- Conclusion
What simulation platform for managed applications is needed and why?

TIOBE Programming Community Index (March 2017)


Source: www.tiobe.com
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

<table>
<thead>
<tr>
<th>// Example of a class.</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Foo {</td>
</tr>
<tr>
<td>public long bar;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>// Source code example.</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>// Allocation site.</td>
</tr>
<tr>
<td>Object obj = new Foo();</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>// GC can happen.</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>// Type introspection.</td>
</tr>
<tr>
<td>if (obj instanceof Foo) {</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
- Memory

- reference
- primitive
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

```
// Example of a class.
class Foo {
   public long bar;
}

// Source code example.
{
   // Allocation site.
   Object obj = new Foo();
   ...
   // GC can happen.
   ...
   // Type introspection.
   if (obj instanceof Foo) {
      ...
   }
```

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata

- reference
- primitive
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

```java
// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }
}
```

| Memory | Memory
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80</td>
<td>0xb8</td>
</tr>
</tbody>
</table>

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata

```
// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }

```

---

```
Memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>Code Cache</td>
</tr>
<tr>
<td>0x78</td>
<td></td>
</tr>
<tr>
<td>0x80</td>
<td>Class Information</td>
</tr>
<tr>
<td>0xb8</td>
<td></td>
</tr>
</tbody>
</table>

- reference
- primitive
```

---

Class Information

- 0x80
- 0xb8
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

```java
// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }
}
```

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

**Specific Characteristics of Managed Applications**

```java
// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }
}
```

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata

```java
// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }
}
```
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata

// Example of a class.
class Foo {
    public long bar;
}

// Source code example.
{
    // Allocation site.
    Object obj = new Foo();
    ...
    // GC can happen.
    ...
    // Type introspection.
    if (obj instanceof Foo) {
        ...
    }
}
What simulation platform for managed applications is needed and why?

Specific Characteristics of Managed Applications

- Distributed in the verifiable bytecode format
- Automatic memory management
- JIT compilation and interpretation
- Object orientation and associated metadata
What simulation platform for managed applications is needed and why?

Support for Tagged Pointers

• An option for object metadata storage

Support in commodity 64-bit architectures

➢ AArch64: [tag:8b | pointer:48b]
➢ SPARC M7: [tag:8b | pointer:48b] - [tag:32b | pointer:32b]
➢ x86-64: [signExtension | pointer:(48b|57b)]
What simulation platform for managed applications is needed and why?

Support for Tagged Pointers

- An option for object metadata storage

Support in commodity 64-bit architectures

- AArch64: [tag:8b | pointer:48b]
- SPARC M7: [tag:8b | pointer:48b] - [tag:32b | pointer:32b]
- x86-64: [signExtension | pointer:(48b|57b)]
What simulation platform for managed applications is needed and why?

Support for Tagged Pointers

- An option for object metadata storage

![Tagged Pointer and Object Diagram]

- Support in commodity 64-bit architectures
  - AArch64: [tag:8b | pointer:48b]
  - SPARC M7: [tag:8b | pointer:48b] - [tag:32b | pointer:32b]
  - x86-64: [signExtension | pointer:(48b|57b)]
What simulation platform for managed applications is needed and why?

Support for Tagged Pointers

- An option for object metadata storage

```
Tagged Pointer
- reference
- primitive
- tag
- metadata
```

- Support in commodity 64-bit architectures
  - AArch64: [tag:8b | pointer:48b]
  - SPARC M7: [tag:8b | pointer:48b] - [tag:32b | pointer:32b]
  - x86-64: [signExtension | pointer:(48b|57b)]
What simulation platform for managed applications is needed and why?

Design Goals

• Productivity for research
  ➢ VM modularity and support of other languages
  ➢ High simulation speed (DaCapo benchmarks in one day on a single PC)

• Awareness of the VM in the simulator

• Advanced features
  ➢ Support of tagged 64-bit pointers
  ➢ Ability to experiment with different object layouts
  ➢ Ability to perform power and energy modeling
VM Selection Justification: Maxine VM

Maxine VM\(^1\): A Platform for Research in VM Technology

- Mostly written in Java, with a substrate written in C
- Modular design: schemes for object layouts, object references, heap and GC, thread synchronization, etc.
- Compilers: T1X (O0), C1X (O1), Graal (O2)
  - Graal supports other languages via Truffle (JavaScript, R, Ruby, others)
- Target ISAs: x86-64, ARMv7
- Class library: JDK 7

VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine Inspector: Integrated Debugging Support
VM Selection Justification: Maxine VM

Maxine VM: Performance Comparison Against Hotspot VM

- Maxine VM performance is \(~59\%\) of the highly optimized Hotspot VM
- Graal (O2) compiler delivers 8\% better performance than C1X (O1)
VM Selection Justification: Maxine VM

Maxine VM: Performance Comparison Against Hotspot VM

- Maxine VM performance is ~59% of the highly optimized Hotspot VM

- Graal (O2) compiler delivers 8% better performance than C1X (O1)
VM Selection Justification: Maxine VM

Maxine VM: Performance Comparison Against Hotspot VM

- Maxine VM performance is ~59% of the highly optimized Hotspot VM
- Graal (O2) compiler delivers 8% better performance than C1X (O1)
Simulator Selection Justification: ZSim

ZSim¹: Fast and Accurate Microarchitectural Simulation

- x86-64 execution-driven timing simulator based on Pin
- Bound-weave technique for scalable simulation
- Lightweight user-level virtualization
- Comparison with open simulators supporting managed applications

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Engine</th>
<th>Full-System</th>
<th>Simulation Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>gem5</td>
<td>Emulation</td>
<td>yes</td>
<td>~100-300 KIPS</td>
</tr>
<tr>
<td>Sniper *</td>
<td>DBT</td>
<td>no</td>
<td>~1-3 MIPS</td>
</tr>
<tr>
<td>ZSim</td>
<td>DBT</td>
<td>no</td>
<td>~7-20 MIPS</td>
</tr>
</tbody>
</table>


* Sniper can simulate DaCapo benchmarks on 32-bit Jikes RVM only.
Simulator Selection Justification: ZSim

ZSim Validation: DaCapo on Maxine VM

- 100% pass rate and ~10% geomean simulation error at ~12 MIPS

- Inconsistencies:
  - eclipse, tradesoap (1C-*): Round Robin vs CFS scheduling
  -avrora: spends more than 50% of execution in the kernel
Simulator Selection Justification: ZSim

ZSim Validation: DaCapo on Maxine VM

- 100% pass rate and ~10% geometric mean simulation error at ~12 MIPS

- Inconsistencies:
  - eclipse, tradesoap (1C-*): Round Robin vs CFS scheduling
  - avrora: spends more than 50% of execution in the kernel
Simulator Selection Justification: ZSim

ZSim Validation: DaCapo on Maxine VM

- 100% pass rate and ~10% geomean simulation error at ~12 MIPS

- Inconsistencies:
  - eclipse, tradesoap (1C-*): Round Robin vs CFS scheduling
  - avrora: spends more than 50% of execution in the kernel
MaxSim: Overview and Features

Maxine-ZSim Integration Scheme

<table>
<thead>
<tr>
<th>MaxSim</th>
<th>Maxine VM (Java + C)</th>
<th>ZSim (C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Code Cache</td>
<td></td>
</tr>
<tr>
<td>p:[tag(16b):base(48b)];</td>
<td>ld / st [tag:base + offset];</td>
<td></td>
</tr>
<tr>
<td>Protocol Buffer Messages</td>
<td>xchg rcx, rcx;</td>
<td></td>
</tr>
</tbody>
</table>

- Magic NOPs
  - Simulation control
  - VM awareness
  - Sending/receiving protocol buffer messages

- Protocol Buffer Messages
  - Interface definition
  - Configuration
  - Profile serialization

- Tagged Pointers
  - VM awareness
  - Profiling
MaxSim: Overview and Features

Maxine-ZSim Integration Scheme

MaxSim

<table>
<thead>
<tr>
<th>Maxine VM (Java + C)</th>
<th>ZSim (C++)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>Code Cache</td>
</tr>
<tr>
<td>p:[tag(16b):base(48b)];</td>
<td>ld / st [tag:base + offset]; xchg rcx, rcx;</td>
</tr>
<tr>
<td>Protocol Buffer</td>
<td></td>
</tr>
<tr>
<td>Messages</td>
<td></td>
</tr>
</tbody>
</table>

- Protocol Buffer Messages
  - Interface definition
  - Configuration
  - Profile serialization

- Magic NOPs
  - Simulation control
  - VM awareness
  - Sending/receiving protocol buffer messages

- Tagged Pointers
  - VM awareness
  - Profiling
MaxSim: Overview and Features

Maxine-ZSim Integration Scheme

- Protocol Buffer Messages
  - Interface definition
  - Configuration
  - Profile serialization

- Magic NOPs
  - Simulation control
  - VM awareness
  - Sending/receiving protocol buffer messages

- Tagged Pointers
  - VM awareness
  - Profiling

MaxSim

Maxine VM (Java + C)

Heap
p:[tag(16b):base(48b)];

Code Cache
ld / st [tag:base + offset];
xchg rcx, rcx;

ZSim (C++)

Protocol Buffer Messages
MaxSim: Overview and Features

Maxine-ZSim Integration Scheme

MaxSim

Maxine VM (Java + C)

- Heap
  - `p:[tag(16b):base(48b)];`
- Code Cache
  - `ld / st [tag:base + offset];`
  - `xchg rcx, rcx;`

ZSim (C++)

- Protocol Buffer Messages
- Magic NOPs
  - Simulation control
  - VM awareness
  - Sending/receiving protocol buffer messages
- Tagged Pointers
  - VM awareness
  - Profiling

- Protocol Buffer Messages
  - Interface definition
  - Configuration
  - Profile serialization
MaxSim: Overview and Features

VM Awareness in the Simulator

- VM memory regions
  - Stack
  - TLS
  - Heap
  - Code cache
  - Native code
  - Others

- VM operations
  - Garbage collection
  - Object allocation

- Object binding
  - To its class
  - To its allocation site
MaxSim: Overview and Features

Pointer Tagging

● Two types of pointer tagging are supported
  ➢ Class ID tagging
  ➢ Allocation site ID tagging

● Tagging/untagging of all pointers at arbitrary places of execution
  ➢ Enables simulation fast-forwarding

● After tagging the following properties are preserved:
  ➢ Pointers to the same object are tagged with the same tag
  ➢ Tags are immutable between an allocation and a garbage collection
  ➢ Objects are accessed using \([\text{tag:base + offset}]\) addressing mode
MaxSim: Overview and Features

Address Space Morphing

- Motivation: easy experimentation with object layouts without adding extra complexity or breaking modularity of Maxine VM

- Supports two object layout transformations

<table>
<thead>
<tr>
<th>Transformation</th>
<th>Before</th>
<th>After Each</th>
<th>After Both</th>
</tr>
</thead>
</table>
| Fields reordering               | class String {
|                                 | char value[];
|                                 | long hash; }             |                          |                          |
|                                 | 0x00 value               | 0x00 value                |                          |
|                                 | 0x08 CIP                 | 0x08 CIP                  |                          |
|                                 | 0x10 hash                | 0x10 hash                 |                          |
|                                 | 0x18 <reserved>          | 0x18 <reserved>           |                          |
|                                 |                         |                          |                          |
| Object pointers compression     | 0x00 CIP                 | 0x00 CIP                  | 0x00 value               |
|                                 | 0x08 <reserved>          | 0x08 <reserved>           | 0x08 CIP                 |
|                                 | 0x10 value               | 0x10 value                | 0x10 hash                |
|                                 | 0x18 hash                | 0x18 hash                 | <reserved>               |

- Makes use of two properties of MaxSim
  - Flexibility of Maxine VM to expand object fields
  - Ability of ZSim to remap memory addresses

[Diagram showing object layouts before and after transformations]
MaxSim: Overview and Features

Address Space Morphing

• Motivation: easy experimentation with object layouts without adding extra complexity or breaking modularity of Maxine VM

• Supports two object layout transformations

  ✓ Fields reordering
  ✓ Object pointers compression

• Makes use of two properties of MaxSim

  ✓ Flexibility of Maxine VM to expand object fields
  ✓ Ability of ZSim to remap memory addresses
MaxSim: Overview and Features

Address Space Morphing

- Motivation: easy experimentation with object layouts without adding extra complexity or breaking modularity of Maxine VM

- Supports two object layout transformations

  - Fields reordering
  - Object pointers compression

- Makes use of two properties of MaxSim
  - Flexibility of Maxine VM to expand object fields
  - Ability of ZSim to remap memory addresses

### Address Space Morphing Comparison

<table>
<thead>
<tr>
<th>Before</th>
<th>After each</th>
<th>After both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td></td>
<td>0x08</td>
<td>0x08</td>
</tr>
<tr>
<td></td>
<td>0x10</td>
<td>0x10</td>
</tr>
<tr>
<td></td>
<td>0x18</td>
<td>0x18</td>
</tr>
<tr>
<td>class String {</td>
<td>value</td>
<td>CIP</td>
</tr>
<tr>
<td>char value[];</td>
<td></td>
<td></td>
</tr>
<tr>
<td>long hash; }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CIP</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>&lt;reserved&gt;</td>
<td>0x08</td>
<td>0x08</td>
</tr>
<tr>
<td>value</td>
<td>0x10</td>
<td>0x10</td>
</tr>
<tr>
<td>hash</td>
<td>0x18</td>
<td>0x18</td>
</tr>
<tr>
<td>CIP</td>
<td>0x00</td>
<td>0x00</td>
</tr>
<tr>
<td>&lt;reserved&gt;</td>
<td>0x08</td>
<td>0x08</td>
</tr>
<tr>
<td>value</td>
<td>0x10</td>
<td>0x10</td>
</tr>
<tr>
<td>hash</td>
<td>0x18</td>
<td>0x18</td>
</tr>
</tbody>
</table>

- reference
- primitive
MaxSim: Overview and Features

Address Space Morphing

• Motivation: easy experimentation with object layouts without adding extra complexity or breaking modularity of Maxine VM

• Supports two object layout transformations
  
  ▶ Fields reordering
  
  ▶ Object pointers compression

• Makes use of two properties of MaxSim
  
  ▶ Flexibility of Maxine VM to expand object fields
  
  ▶ Ability of ZSim to remap memory addresses
MaxSim: Overview and Features

Address Space Morphing

• Motivation: easy experimentation with object layouts without adding extra complexity or breaking modularity of Maxine VM

• Supports two object layout transformations

<table>
<thead>
<tr>
<th>before</th>
<th>after each</th>
<th>after both</th>
</tr>
</thead>
</table>
  | Fields reordering | class String {
  |         |     char value[];
  |         |     long hash;
  |         | } |
  | CIP     | 0x00 value | 0x00 value |
  | <reserved> | 0x08 CIP | 0x08 CIP |
  | hash    | 0x10 hash | 0x10 hash |
  |          | 0x18 <reserved> | 0x18 <reserved> |

• Makes use of two properties of MaxSim

  | Flexibility of Maxine VM to expand object fields
  | Ability of ZSim to remap memory addresses
MaxSim: Overview and Features

Stages of Address Space Morphing

\[ f (1:2) \]

### Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>ref.0</th>
<th>prim.1</th>
<th>ref.2</th>
<th>prim.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x18</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>ref.2</th>
<th>ref.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Addressing

\[ [b_o + o_o] \]

### Fields Reordering Map

- reference
- primitive

<table>
<thead>
<tr>
<th>Address</th>
<th>Reorder</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x08</td>
</tr>
<tr>
<td>0x08</td>
<td>0x18</td>
</tr>
<tr>
<td>0x10</td>
<td>0x00</td>
</tr>
<tr>
<td>0x18</td>
<td>0x10</td>
</tr>
</tbody>
</table>
MaxSim: Overview and Features

Stages of Address Space Morphing

\[ f_e(1,2) - \text{expansion} \]

in Maxine VM

### Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>ref.0</td>
</tr>
<tr>
<td>0x08</td>
<td>prim.1</td>
</tr>
<tr>
<td>0x10</td>
<td>ref.2</td>
</tr>
<tr>
<td>0x18</td>
<td>prim.3</td>
</tr>
</tbody>
</table>

### Addressing

\[ [b_o + o_o] \]

### Fields Reordering Map

- reference
- primitive

\[ m_o \]
MaxSim: Overview and Features

Stages of Address Space Morphing

\( f_e(1,2) \) - expansion

in Maxine VM

Layout

```
0x00  ref.0
0x08  prim.1
0x10  ref.2
0x18  prim.3

0x00  ref.0
0x08  prim.1
0x10  ref.2
0x18  prim.3
0x20
0x28
```

Addressing

\([b_0 + o_0]\)

Fields Reordering Map

```
0x00→0x08
0x08→0x18
0x10→0x00
0x18→0x10
```
MaxSim: Overview and Features

Stages of Address Space Morphing

\( f_e(1,2) \) - expansion

in Maxine VM

Layout

<table>
<thead>
<tr>
<th>0x00</th>
<th>ref.0</th>
<th>0x00</th>
<th>ref.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x08</td>
<td>prim.1</td>
<td>0x08</td>
<td>prim.1</td>
</tr>
<tr>
<td>0x10</td>
<td>ref.2</td>
<td>0x10</td>
<td>ref.2</td>
</tr>
<tr>
<td>0x18</td>
<td>prim.3</td>
<td>0x18</td>
<td>prim.3</td>
</tr>
</tbody>
</table>

Addressing

\[[b_0 + o_0] \quad [f_e(b_0) + f_e(o_0)]\]

Fields Reordering Map

\[m_o\]
MaxSim: Overview and Features

Stages of Address Space Morphing

\( f_e(1,2) \) - expansion

in Maxine VM

### Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>ref.0</th>
<th>0x00</th>
<th>prim.1</th>
<th>0x08</th>
<th>ref.2</th>
<th>0x10</th>
<th>prim.3</th>
<th>0x18</th>
<th>ref.0</th>
<th>0x00</th>
<th>prim.1</th>
<th>0x08</th>
<th>ref.2</th>
<th>0x10</th>
<th>prim.3</th>
<th>0x18</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td></td>
<td>ref.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ref.0</td>
<td>0x00</td>
<td></td>
<td>0x08</td>
<td>prim.1</td>
<td>0x08</td>
<td></td>
<td>0x20</td>
</tr>
<tr>
<td>0x08</td>
<td></td>
<td>prim.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>prim.1</td>
<td>0x08</td>
<td></td>
<td>0x10</td>
<td></td>
<td>0x10</td>
<td></td>
<td>0x10</td>
</tr>
<tr>
<td>0x10</td>
<td></td>
<td>ref.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ref.2</td>
<td>0x10</td>
<td></td>
<td>0x18</td>
<td></td>
<td>0x18</td>
<td></td>
<td>0x18</td>
</tr>
<tr>
<td>0x18</td>
<td></td>
<td></td>
<td>ref.2</td>
<td></td>
<td></td>
<td></td>
<td>ref.2</td>
<td></td>
<td></td>
<td>0x18</td>
<td></td>
<td>0x20</td>
<td></td>
<td>0x20</td>
<td></td>
<td>0x20</td>
</tr>
</tbody>
</table>

### Addressing

\[ [b_0 + o_0] \quad \quad [f_e(b_0) + f_e(o_0)] \]

### Fields Reordering Map

\[
\begin{align*}
m_o & : 0x00 \rightarrow 0x08 \\
& \quad 0x08 \rightarrow 0x18 \\
& \quad 0x10 \rightarrow 0x00 \\
& \quad 0x18 \rightarrow 0x10 \\
m_e & : 0x00 \rightarrow 0x08 \\
& \quad 0x08 \rightarrow 0x20 \\
& \quad 0x18 \rightarrow 0x00 \\
& \quad 0x20 \rightarrow 0x10 
\end{align*}
\]
MaxSim: Overview and Features

Stages of Address Space Morphing

$f_e(1,2)$ - expansion

in Maxine VM

$f_c(2)$ - contraction

in ZSim

### Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>0x00</th>
<th>0x08</th>
<th>0x10</th>
<th>0x18</th>
</tr>
</thead>
<tbody>
<tr>
<td>ref.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prim.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ref.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prim.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Addressing

$[b_0 + o_0]$  $[f_e(b_0) + f_e(o_0)]$

### Fields Reordering Map

<table>
<thead>
<tr>
<th>$m_o$</th>
<th>0x00→0x08</th>
<th>0x08→0x18</th>
<th>0x10→0x00</th>
<th>0x18→0x10</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_e$</td>
<td>0x00→0x08</td>
<td>0x08→0x20</td>
<td>0x18→0x00</td>
<td>0x20→0x10</td>
</tr>
</tbody>
</table>

- reference
- primitive
MaxSim: Overview and Features

Stages of Address Space Morphing

- $f_e(1,2)$ - expansion
  - in Maxine VM

- $f_c(2)$ - contraction
  - in ZSim

### Layout

<table>
<thead>
<tr>
<th>Address in Maxine VM</th>
<th>Address in ZSim</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00 → ref.0</td>
<td>0x00 → ref.0</td>
</tr>
<tr>
<td>0x08 → prim.1</td>
<td>0x08 → prim.1</td>
</tr>
<tr>
<td>0x10 → ref.2</td>
<td>0x10 → ref.2</td>
</tr>
<tr>
<td>0x18 → prim.3</td>
<td>0x18 → prim.3</td>
</tr>
</tbody>
</table>

### Addressing

- $[b_o + o_o]$
- $[f_e(b_o) + f_e(o_o)]$
- $[b_e/2 + o_e/2]$

### Fields Reordering Map

- $m_o$
- $m_e$
MaxSim: Overview and Features

Stages of Address Space Morphing

\( f_e(1,2) \) - expansion \( f_c(2) \) - contraction

in Maxine VM in ZSim

Layout

<table>
<thead>
<tr>
<th>ref.0</th>
<th>prim.1</th>
<th>ref.1</th>
<th>prim.2</th>
<th>ref.0</th>
<th>m.1</th>
<th>ref.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0x08</td>
<td>0x10</td>
<td>0x18</td>
<td>0x00</td>
<td>0x08</td>
<td>0x10</td>
</tr>
<tr>
<td>prim.3</td>
<td>ref.2</td>
<td>prim.3</td>
<td>ref.2</td>
<td>prim.3</td>
<td>ref.2</td>
<td>prim.3</td>
</tr>
<tr>
<td>0x18</td>
<td>0x20</td>
<td>0x28</td>
<td>0x00</td>
<td>0x20</td>
<td>0x18</td>
<td>0x00</td>
</tr>
</tbody>
</table>

Addressing

\[ [b_o + o_o] \quad [f_e(b_o) + f_e(o_o)] \quad [b_e/2 + o_e/2] \]

Fields Reordering Map

\[
\begin{align*}
0x00 &\rightarrow 0x08 \\
0x08 &\rightarrow 0x18 \\
0x10 &\rightarrow 0x00 \\
0x18 &\rightarrow 0x10 \\
0x00 &\rightarrow 0x08 \\
0x08 &\rightarrow 0x20 \\
0x18 &\rightarrow 0x00 \\
0x20 &\rightarrow 0x10
\end{align*}
\]

- reference
- primitive
MaxSim: Overview and Features

Stages of Address Space Morphing

- $f_e(1,2)$ - expansion (in Maxine VM)
- $f_c(2)$ - contraction (in ZSim)

**Addressing**

\[
\begin{align*}
[b_0+o_0] & \quad [f_e(b_0) + f_e(o_0)] & \quad [b_e/2 + o_e/2]
\end{align*}
\]

**Fields Reordering Map**

- $b_0$ and $o_0$ reordering in Maxine VM
- $b_e$ and $o_e$ reordering in ZSim

- Reference and primitive fields are reordered as shown in the table.
MaxSim: Overview and Features

Stages of Address Space Morphing

- $f_e(1,2)$ - expansion
  - in Maxine VM
  - Layout
  - Addressing
  - Fields Reordering Map

- $f_c(2)$ - contraction
  - in ZSim
  - Layout
  - Addressing

- $f_r(m_c)$ - reordering
  - in ZSim
  - Layout
  - Addressing
  - Fields Reordering Map
MaxSim: Overview and Features

Stages of Address Space Morphing

- $f_e(1,2)$ - expansion
- $f_c(2)$ - contraction
- $f_r(m_c)$ - reordering

**in Maxine VM**

**in ZSim**

### Layout

<table>
<thead>
<tr>
<th>Address</th>
<th>Maxine VM</th>
<th>ZSim</th>
<th>ZSim</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>ref.0</td>
<td>ref.0</td>
<td>ref.0</td>
</tr>
<tr>
<td>0x08</td>
<td>prim.1</td>
<td>prim.1</td>
<td>prim.1</td>
</tr>
<tr>
<td>0x10</td>
<td>ref.2</td>
<td>ref.2</td>
<td>ref.2</td>
</tr>
<tr>
<td>0x18</td>
<td>prim.3</td>
<td>prim.3</td>
<td>prim.3</td>
</tr>
</tbody>
</table>

### Addressing

- $[b_o + o_o]$
- $[f_e(b_o) + f_e(o_o)]$
- $[b_e/2 + o_e/2]$
- $[b_c + m_c(o_c)]$

### Fields Reordering Map

- $m_o$
- $m_e$
- $m_c$

- reference
- primitive
MaxSim: Overview and Features

Stages of Address Space Morphing

- $f_e(1,2)$ - expansion
  - in Maxine VM

- $f_c(2)$ - contraction
  - in ZSim

- $f_r(m_c)$ - reordering
  - in ZSim

**Layout**

<table>
<thead>
<tr>
<th>0x00</th>
<th>ref.0</th>
<th>0x00</th>
<th>ref.0</th>
<th>0x00</th>
<th>ref.2</th>
<th>0x08</th>
<th>ref.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x08</td>
<td>prim.1</td>
<td>0x08</td>
<td>prim.1</td>
<td>0x08</td>
<td>prim.1</td>
<td>0x08</td>
<td>prim.1</td>
</tr>
<tr>
<td>0x10</td>
<td>ref.2</td>
<td>0x10</td>
<td>ref.2</td>
<td>0x10</td>
<td>ref.2</td>
<td>0x10</td>
<td>ref.2</td>
</tr>
<tr>
<td>0x18</td>
<td>prim.3</td>
<td>0x18</td>
<td>prim.3</td>
<td>0x18</td>
<td>prim.3</td>
<td>0x18</td>
<td>prim.3</td>
</tr>
</tbody>
</table>

**Addressing**

- $[b_o + o_o]$  
- $[f_e(b_o) + f_e(o_o)]$  
- $[b_e/2 + o_e/2]$  
- $[b_c + m_c(o_c)]$

**Fields Reordering Map**

- $m_o$
  - 0x00→0x08
  - 0x08→0x18
  - 0x10→0x00
  - 0x18→0x10

- $m_e$
  - 0x00→0x08
  - 0x08→0x20
  - 0x18→0x00

- $m_c$
  - 0x00→0x04
  - 0x04→0x10
  - 0x0C→0x00
  - 0x18→0x08
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

• Simulation filtering of copying and initialization

```
// Loop used for initialization.
void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) {
        p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
}
```

• Special cases for fast simulation
  ➢ Array of primitives and code cache objects are handled differently

• Validation
  ➢ References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  ➢ Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

• Simulation filtering of copying and initialization
  \[ f_c(2) – \text{contraction} \]
  in ZSim

• Special cases for fast simulation
  ➢ Array of primitives and code cache objects are handled differently

• Validation
  ➢ References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  ➢ Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization
  - \( f_c(2) \) – contraction in ZSim

```
// Loop used for initialization.
void setWords(Pointer p, int n) {
  ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
  for (int i = 0; i < n; i++) {
    p.writeWord(i, 0);
  }
  ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
}
```

- Special cases for fast simulation
  - Array of primitives and code cache objects are handled differently

- Validation
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization
  
  ```c
  // Loop used for initialization.
  void setWords(Pointer p, int n) { 
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) { // i = 0
      p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
  }
  ```

- Special cases for fast simulation
  
  - Array of primitives and code cache objects are handled differently

- Validation
  
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization
  
  \[
  f_c(2) \text{ – contraction in ZSim}
  \]

  ![Diagram showing address space morphing]

- Special cases for fast simulation
  
  - Array of primitives and code cache objects are handled differently

- Validation
  
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout

```c
// Loop used for initialization.
void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) {
        // i = 1
        p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
}
```
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization

```c
// Loop used for initialization.
void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) { // i = 2
        p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
}
```

- Special cases for fast simulation
  - Array of primitives and code cache objects are handled differently

- Validation
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization

- Special cases for fast simulation
  - Array of primitives and code cache objects are handled differently

- Validation
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

• Simulation filtering of copying and initialization

  \[
  f_c(2) \text{ – contraction in ZSim}
  \]

  
  ```c
  // Loop used for initialization.
  void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) { // i = 4
      p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
  }
  ```

  ```
  \begin{array}{c}
  \text{0x00} \\
  \text{0x08} \\
  \text{0x10} \\
  \text{0x18} \\
  \text{0x20} \\
  \text{0x28} \\
  \end{array}
  \begin{array}{c}
  0 \\
  0 \\
  0 \\
  0 \\
  0 \\
  \end{array}
  \begin{array}{c}
  \text{0x00} \\
  \text{0x08} \\
  \text{0xa0} \\
  \text{0x10} \\
  \text{0x18} \\
  \text{0x20} \\
  \end{array}
  \begin{array}{c}
  0 \\
  0 \\
  0 \\
  0 \\
  0 \\
  \end{array}
  \end{array}
  
  \]

• Special cases for fast simulation

  ➢ Array of primitives and code cache objects are handled differently

• Validation

  ➢ References and primitives were expanded twice in Maxine VM and contracted twice in ZSim

  ➢ Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

• Simulation filtering of copying and initialization

```c
// Loop used for initialization.
void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) {   // i = 5
        p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
}
```

• Special cases for fast simulation

  ➔ Array of primitives and code cache objects are handled differently

• Validation

  ➔ References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  ➔ Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

• Simulation filtering of copying and initialization
  
  // Loop used for initialization.
  void setWords(Pointer p, int n) {
    ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
    for (int i = 0; i < n; i++) {
      p.writeWord(i, 0);
    }
    ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
  }

  \[ f_c(2) \] – contraction
  in ZSim

  [b + o_e]   [b_e/2 + o_e/2]

  0x00 0 0x00 0
  0x08 0 0x08 0
  0x10 0 0x10 0
  0x18 0 0 0
  0x20 0 0 0
  0x28 0 0 0

• Special cases for fast simulation
  
  ➢ Array of primitives and code cache objects are handled differently

• Validation
  
  ➢ References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  ➢ Less than 1% difference in comparison with the original layout
MaxSim: Overview and Features

Address Space Morphing: Special Cases and Validation

- Simulation filtering of copying and initialization
  ```c
  // Loop used for initialization.
  void setWords(Pointer p, int n) {
      ZSIM_MAGIC_NOP(BEGIN_LOOP_FILTERING);
      for (int i = 0; i < n; i++) {
          p.writeWord(i, 0);
      }
      ZSIM_MAGIC_NOP(END_LOOP_FILTERING);
  }
  ```

- Special cases for fast simulation
  - Array of primitives and code cache objects are handled differently

- Validation
  - References and primitives were expanded twice in Maxine VM and contracted twice in ZSim
  - Less than 1% difference in comparison with the original layout
MaxSim: Use Cases

DaCapo Tomcat Characterization

1 Core 2MB LLC: ♦
4 Cores 8MB LLC: ■
GC part: □

Instructions per Clock

L3 Load Cache Misses per Kilo Instruction

Consumed Power

L2 Load Cache Misses per Kilo Instruction
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

<table>
<thead>
<tr>
<th>char[]</th>
<th>(i:43</th>
<th>mf:57163720</th>
<th>s:56(200337)</th>
<th>r2m:722499</th>
<th>w2m:158200</th>
<th>r3m:108784</th>
<th>w3m:7723</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(o:16</td>
<td>f:.35</td>
<td>r:18602074</td>
<td>w:759093</td>
<td>r2m:596449</td>
<td>w2m:62251</td>
<td>r3m:80211</td>
<td>w3m:161</td>
</tr>
</tbody>
</table>
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```plaintext
char[](C)(i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
...
(o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
...
### MaxSim: Use Cases

**Analysis of L2 Cache Misses via Profiling**

MaxSim output of class profiling information

<table>
<thead>
<tr>
<th>char[]</th>
<th>i: 43</th>
<th>mf: 57163720</th>
<th>s: 56(200337)</th>
<th>r2m: 722499</th>
<th>w2m: 158200</th>
<th>r3m: 108784</th>
<th>w3m: 7723</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>o: 16</td>
<td>f: 0.35</td>
<td>r: 18602074</td>
<td>w: 759093</td>
<td>r2m: 596449</td>
<td>w2m: 62251</td>
<td>r3m: 80211</td>
</tr>
</tbody>
</table>

...
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```plaintext
char[](i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):

(o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
```
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[](C)(i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
   ...
   (0:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
   ...
```
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[][](i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
  (o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
```

Maxsim output of cache miss site profiling information

```
```
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[]([C](i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):

(o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
```

Maxsim output of cache miss site profiling information

```
```
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[]([C](i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
...
(o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
...
```

Maxsim output of cache miss site profiling information

```
...```

MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[][] (C)(i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
  (o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
```

Maxsim output of cache miss site profiling information

```
... [java.lang.String.equals(Object)+108(k:I bci:23)](m:539629 i:43 ol:16 oh:16)
...```

String.class bytecode

<table>
<thead>
<tr>
<th>bci</th>
<th>Instr.</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>getfield</td>
<td>980</td>
</tr>
<tr>
<td>23</td>
<td>arraylength</td>
<td>980</td>
</tr>
<tr>
<td>24</td>
<td>istore_3</td>
<td>980</td>
</tr>
</tbody>
</table>
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[][] (i:43) mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
... (o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
```

Maxsim output of cache miss site profiling information

```
```

String.class bytecode

<table>
<thead>
<tr>
<th>bci</th>
<th>Instr.</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td><em>getField</em></td>
<td>980</td>
</tr>
<tr>
<td>23</td>
<td><em>arraylength</em></td>
<td>980</td>
</tr>
<tr>
<td>24</td>
<td><em>istore_3</em></td>
<td>980</td>
</tr>
</tbody>
</table>
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[] ([C] (i: 43) mf: 57163720 (s: 56(200337) ...) r2m: 722499 w2m: 158200 r3m: 108784 w3m: 7723):

(o: 16 f: .35 r: 18602074 w: 759093 r2m: 596449 w2m: 62251 r3m: 80211 w3m: 161)
```

Maxsim output of cache miss site profiling information

```
String.class bytecode

<table>
<thead>
<tr>
<th>bci</th>
<th>Instr.</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>getfield</td>
<td>980</td>
</tr>
<tr>
<td>23</td>
<td>arraylength</td>
<td>980</td>
</tr>
<tr>
<td>24</td>
<td>istore_3</td>
<td>980</td>
</tr>
</tbody>
</table>
```

String.java source code

```
974  public boolean equals(Object anObject) {
975    if (this == anObject) {
976        return true;
977    }
978    if (anObject instanceof String) {
979        String anotherString = (String) anObject;
980        int n = value.length;
981        if (n != anotherString.value.length)
982            return false;
983    }
```
MaxSim: Use Cases

Analysis of L2 Cache Misses via Profiling

MaxSim output of class profiling information

```
char[]([C](i:43 mf:57163720 (s:56(200337) ... r2m:722499 w2m:158200 r3m:108784 w3m:7723):
  (o:16 f:.35 r:18602074 w:759093 r2m:596449 w2m:62251 r3m:80211 w3m:161)
...
```

Maxsim output of cache miss site profiling information

```
...
```

String.class bytecode

```
<table>
<thead>
<tr>
<th>bci</th>
<th>Instr.</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>getfield</td>
<td>980</td>
</tr>
<tr>
<td>23</td>
<td>arraylength</td>
<td>980</td>
</tr>
<tr>
<td>24</td>
<td>istore_3</td>
<td>980</td>
</tr>
</tbody>
</table>
```

String.java source code

```
974     public boolean equals(Object anObject) {
975         if (this == anObject) {
976             return true;
977         }
978         if (anObject instanceof String) {
979             String anotherString = (String) anObject;
980             int n = value.length;
981             if (n != anotherString.value.length)
982                 return false;
983         }
984     }
```
Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0xFFFFF is Not an Array Length (NaAL) indicator.
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

• Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0xFFFFF is Not an Array Length (NaAL) indicator

• Array length retrieval in software

Source code

```
inline int retrieveArrayLength(Pointer_t objectPointer)
{
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return * ((int *) (objectPointer + 0x10));
}
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFE], when 0xFFFF is Not an Array Length (NaAL) indicator.

- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer) {
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *((int *)((objectPointer + 0x10));
}
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

• Having 16-bit-tagged pointers it is possible to store a range of array lengths \([0;0xFFFFE]\), when 0xFFFFF is Not an Array Length (NaAL) indicator

• Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer) {
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *((int *) (objectPointer + 0x10));
}
```
Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0xFFFFF is Not an Array Length (NaAL) indicator.

- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer)
{
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return * ((int *) (objectPointer + 0x10));
}
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFE], when 0xFFFF is Not an Array Length (NaAL) indicator

- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer) {
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *(int *)((objectPointer + 0x10));
}
```
Storing Array Length in a Pointer Tag

• Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0XFFFFF is Not an Array Length (NaAL) indicator

• Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer)
{
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *((int *) (objectPointer + 0x10));
}
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0xFFFFF is Not an Array Length (NaAL) indicator
- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer) {
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *((int *) (objectPointer + 0x10));
}
```

- Dynamic execution height of 4.5 instructions of 19 bytes
- Originally 1 instruction of 4 bytes

x86-64 assembler

```assembly
// objectAddress in %rdi
movq %rdi, %rax
shrq $48, %rax
cmpq $65535, %rax
jne .L1
movq 16(%rdi), %rax
.L1:
// array length in %rax
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFFE], when 0xFFFFF is Not an Array Length (NaAL) indicator

- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer) {
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *(int*) (objectPointer + 0x10);
}
```

- Dynamic execution height of 4.5 instructions of 19 bytes
- Originally 1 instruction of 4 bytes

x86-64 assembler

```
// objectAddress in %rdi
movq %rdi, %rax
shrq $48, %rax
cmpq $65535, %rax
ejne .L1
movq 16(%rdi), %rax
.L1:
// array length in %rax
```

Tagged pointer

Array object

```text
len
len
len
```

```text
0x00
0x08
0x10
```
MaxSim: Use Cases

Storing Array Length in a Pointer Tag

- Having 16-bit-tagged pointers it is possible to store a range of array lengths [0;0xFFFE], when 0xFFFF is Not an Array Length (NaAL) indicator.

- Array length retrieval in software

Source code

```c
inline int retrieveArrayLength(Pointer_t objectPointer)
{
    TAG_t tag = extractTAG(objectPointer);
    if (tag != NaAL) {
        return (int) tag;
    }
    return *(int *)((int *) (objectPointer + 0x10));
}
```

x86-64 assembler

```assembly
// objectAddress in %rdi
movq %rdi, %rax
shrq $48, %rax
cmpq $65535, %rax
jne .L1

movq 16(%rdi), %rax

.L1:
// array length in %rax
```

- Dynamic execution height of 4.5 instructions of 19 bytes
- Originally 1 instruction of 4 bytes
HW-Assisted Array Length Retrieval from Tagged Pointers

MaxSim: Use Cases

AGU

Data Bus
Base
Offset

addressBits

tagBits

offBits

Part of LSU

dataBits

addressBits

addressBits

32

32

32

0x0

MU

MT

LP

LEX

0x0

20
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return * (( CIP_t *) (objectAddress + 0x10));
}
```

// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return * (( CIP_t *) (objectAddress + 0x10));
}
```

```
// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
```
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return *(CIP_t *)((objectAddress + 0x10));
}
```

```
// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
```
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return * (( CIP_t *) (objectAddress + 0x10));
}
```

```
// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
```
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return * (( CIP_t *) (objectAddress + 0x10));
}
```

// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return *((CIP_t *)(objectAddress + 0x10));
}
```

```
// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
```
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return *( (CIP_t *) (objectAddress + 0x10));
}
```

// objectAddress in %rdi
movq 16(%rdi), %rax
// array length in %rax
MaxSim: Use Cases

HW-Assisted Array Length Retrieval from Tagged Pointers

- Array length retrieval in one instruction

```c
inline int retrieveArrayLength(Address_t objectAddress) {
    return *((CIP_t*)((int)objectAddress + 0x10));
}
```

```assembly
movq 16(%rdi), %rax // array length in %rax
```
Evaluation of HW-Assisted Array Length Retrieval

- L1 Data Cache Loads Reduction (~6% in geomean) on DaCapo Benchmarks
  - 1 Core 2MB LLC:
  - 4 Cores 8MB LLC:

- Dynamic Energy Reduction (~2% in geomean) on DaCapo Benchmarks
  - 1 Core 2MB LLC:
  - 4 Cores 8MB LLC:
MaxSim: Use Cases

Evaluation of HW-Assisted Array Length Retrieval

L1 Data Cache Loads Reduction (~6% in geomean) on DaCapo Benchmarks

Dynamic Energy Reduction (~2% in geomean) on DaCapo Benchmarks
MaxSim: Use Cases

Evaluation of HW-Assisted Array Length Retrieval

L1 Data Cache Loads Reduction (~6% in geomean) on DaCapo Benchmarks

Dynamic Energy Reduction (~2% in geomean) on DaCapo Benchmarks
Conclusion

• Novel simulation platform for managed applications
  ▶ Based of the state-of-the art VM and simulator
  ▶ Awareness of the VM in the simulator
  ▶ Simulation of 16-bit tagged pointers on x86-64
  ▶ Low-overhead memory access profiling
  ▶ Address-space morphing technique

• Use cases
  ▶ Workload characterization and profiling
  ▶ HW/SW co-design and exploration of architectural specialization for managed applications
  ▶ Easy experimentation with object layout transformations

• Open-source platform is available at:
  https://github.com/beehive-lab/MaxSim