Hardware Supported Permission Checks
On Persistent Objects for Performance
and Programmability

Tiancong Wang, Sakthikumaran Sambasivam, James Tuck

twang14@ncsu.edu or jtuck@ncsu.edu
NVM Programming
NVM Programming

• Non-Volatile Memory is attractive
  • Byte addressable persistent storage
NVM Programming

- Non-Volatile Memory is attractive
  - Byte addressable persistent storage
- Direct access (DAX)
NVM Programming

- Non-Volatile Memory is attractive
  - Byte addressable persistent storage
- Direct access (DAX)
  - Treat NVM as many persistent regions, a.k.a. pools
NVM Programming

- Non-Volatile Memory is attractive
  - Byte addressable persistent storage
- Direct access (DAX)
  - Treat NVM as many persistent regions, a.k.a. pools
  - Pools are mapped to address space to create persistent data structure
NVM Programming

- Non-Volatile Memory is attractive
  - Byte addressable persistent storage
- Direct access (DAX)
  - Treat NVM as many persistent regions, a.k.a. pools
  - Pools are mapped to address space to create persistent data structure
  - Use permanent pointers for creating linked data structures
NVM Programming

- Non-Volatile Memory is attractive
  - Byte addressable persistent storage
- Direct access (DAX)
  - Treat NVM as many persistent regions, a.k.a. pools
  - Pools are mapped to address space to create persistent data structure
  - Use permanent pointers for creating linked data structures
  - A permanent pointer can point to an object within the pool, or across pools
Problem

Program 1
Address Space

stack

heap

BSS
data
text

Pool 1
Pool 2

NVMM
Problem

• Another program wants to access the same data structure
Another program wants to access the same data structure

★ Is the permanent pointer still valid?
Problem

- Another program wants to access the same data structure
  ★ Is the permanent pointer still valid?
- Need to **map** the pools before accessing it.
Another program wants to access the same data structure

★ Is the permanent pointer still valid?

Need to map the pools before accessing it.

Not all the programs have same permissions; need to check the permission before mapping
Problem

- Another program wants to access the same data structure
  ★ Is the permanent pointer still valid?
- Need to map the pools before accessing it.
- Not all the programs have same permissions; need to check the permission before mapping
  ★ We need to validate on ALL permanent pointers before dereferencing, to make sure the pool is checked and mapped
Problem

• Another program wants to access the same data structure
  ★ Is the permanent pointer still valid?
• Need to map the pools before accessing it.
• Not all the programs have same permissions; need to check the permission before mapping
  ★ We need to validate on ALL permanent pointers before dereferencing, to make sure the pool is checked and mapped
Solution

• System Persistent Object Table (SPOT)
  • Similar to page table, BUT is a system table.
  • Permission check: stores permission information
  • Mapping: stores mapping of physical addresses
• Programmers no longer need to reason about permanent pointers as long as the program has permissions
Contributions
Contributions

• Point out permanent pointers can point to unmapped regions; reasoning about them is a new programming burden
Contributions

• Point out permanent pointers can point to unmapped regions; reasoning about them is a new programming burden

• Hardware can efficiently check permission and map pools when dereferencing any permanent address.
Contributions

• Point out permanent pointers can point to unmapped regions; reasoning about them is a new programming burden.

• **Hardware** can efficiently check permission and map pools when dereferencing **any** permanent address.

• We study and compare different design choices that provide permission checks in hardware and software.
Contributions

• Point out permanent pointers can point to unmapped regions; reasoning about them is a new programming burden

• **Hardware** can efficiently check permission and map pools when dereferencing **any** permanent address.

• We study and compare different design choices that provide permission checks in hardware and software

• Evaluate on 6 microbenchmarks and 2 applications
Contributions

• Point out permanent pointers can point to unmapped regions; reasoning about them is a new programming burden
• **Hardware** can efficiently check permission and map pools when dereferencing *any* permanent address.
• We study and compare different design choices that provide permission checks in hardware and software
• Evaluate on 6 microbenchmarks and 2 applications
• Our design offers a 2.9x speedup on average for microbenchmarks and 1.4x and 1.8x speedup on TPC-C and Vacation
Outline

• Introduction
• Motivation - More details of permission check and pool mapping
• Implementation - Hardware supported permission check (SPOT)
• Result - Performance on out-of-order processors
• Summary
MOTIVATION

• Permanent address: OID
• Where’s programming burden and performance overhead
• Idea of automatic permission check
Motivation - ObjectIDs
Motivation - ObjectIDs

- Permanent address:
  ObjectId, (Pool ID, offset)
Motivation - ObjectIDs

- **Permanent address:** ObjectID, (Pool ID, offset)
- **Translation:** Address = BaseAddressLookup(Pool #) + offset

<table>
<thead>
<tr>
<th>Pool ID</th>
<th>Base address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>5678</td>
<td>0x12345678</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Motivation - ObjectIDs

- **Permanent address**: ObjectID, (Pool ID, offset)
- **Translation**: Address = BaseAddressLookup(Pool #) + offset

<table>
<thead>
<tr>
<th>Pool ID</th>
<th>Base address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>5678</td>
<td>0x12345678</td>
</tr>
</tbody>
</table>

Object ID | Offset
---|---
5678 | 

Hash Table search

Translated address
Motivation - ObjectIDs

- **Permanent address**: ObjectID, (Pool ID, offset)
- **Translation**: Address = BaseAddressLookup(Pool #) + offset
- Translation can be optimized through either hardware or software support in prior works

<table>
<thead>
<tr>
<th>Pool ID</th>
<th>Base address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234</td>
<td>0xDEADBEEF</td>
</tr>
<tr>
<td>5678</td>
<td>0x12345678</td>
</tr>
</tbody>
</table>

Hash Table search

Translated address
Motivation - Programming burden
Motivation - Programming burden

oid1 = head(1,10)
Motivation - Programming burden

oid1 = head
check oid1(1,10) Check&Map Pool 1

Address Space

Pool 1

Pool 2

stack

heap

BSS
data
text
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1 \( (1, 10) \)
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
(1, 20) = load from addr1
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2 (1,20)
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2

Pool 1

Pool 2

Address Space

stack

Pool 1

Pool 2

heap

BSS
data
text

 oid2 = (1, 20)
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
(2, 20) = load from addr2
Motivation - Programming burden

oid1 = head
check oid1
 addr1 = translate oid1
oid2 = load from addr1
check oid2
 addr2 = translate oid2
oid3 = load from addr2
check oid3 (2, 20)  Check & Map Pool 2

Address Space

Pool 1

Pool 2

stack
data
BSS
heap
text
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3 (2,20)
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3
load from addr3

(2, 20)
Motivation - Programming burden

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3
load from addr3
Routine to access on ObjectID:

- Validation
- Translation
- Load/store data from it

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3
load from addr3
Motivation - Performance Overhead

- Each validation adds overhead:
  - Check if the pool is mapped or not
  - If not,
    - Check permissions
      - Permission per user
      - Use Linux file permission: read, write, executable on the owner, group and others: e.g. 644
    - If the user has permission, map the pool to the address space, similar to memory-mapped files (kernel-level operation)
  - Otherwise, abort program immediately
Motivation - Performance Overhead

• Each validation adds overhead:
  • Check if the pool is mapped or not
  • If not,
    • Check permissions
      • Permission per user
      • Use Linux file permission: read, write, executable on the
        owner, group and others: e.g. 644
    • If the user has permission, map the pool to the address space,
      similar to memory-mapped files (kernel-level operation)
    • Otherwise, abort program immediately

1. Redundant check on pools
2. Check permissions
3. Context switch
Prior works - Leave it to programmers

oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3
load from addr3
Prior works - Leave it to programmers

- Prior works solve this problem by leaving to programmers

```lisp
oid1 = head
check oid1
addr1 = translate oid1
oid2 = load from addr1
check oid2
addr2 = translate oid2
oid3 = load from addr2
check oid3
addr3 = translate oid3
load from addr3
```
Prior works - Leave it to programmers

- Prior works solve this problem by leaving it to programmers
- Programmers have to check & map the pools with names at the beginning of the program -> eliminates performance overhead

\[
\begin{align*}
oid1 &= \text{head} \\
\text{check oid1} \\
addr1 &= \text{translate oid1} \\
oid2 &= \text{load from addr1} \\
\text{check oid2} \\
addr2 &= \text{translate oid2} \\
oid3 &= \text{load from addr2} \\
\text{check oid3} \\
addr3 &= \text{translate oid3} \\
\text{load from addr3}
\end{align*}
\]
Prior works - Leave it to programmers

- Prior works solve this problem by leaving it to programmers.
- Programmers have to check and map the pools with names **at the beginning** of the program -> eliminates performance overhead.

\[
\begin{align*}
\text{oid1} &= \text{head} \\
\text{addr1} &= \text{translate oid1} \\
\text{oid2} &= \text{load from addr1} \\
\text{addr2} &= \text{translate oid2} \\
\text{oid3} &= \text{load from addr2} \\
\text{addr3} &= \text{translate oid3} \\
\text{load from addr3}
\end{align*}
\]
Prior works - Leave it to programmers

- Prior works solve this problem by leaving to programmers
- Programmers have to check & map the pools with names at the beginning of the program -> eliminates performance overhead

check-and-map “pool 1”
check-and-map “pool 2”
...
oid1 = head
addr1 = translate oid1
oid2 = load from addr1
addr2 = translate oid2
oid3 = load from addr2
addr3 = translate oid3
load from addr3
Prior works - Leave it to programmers

• Prior works solve this problem by leaving to programmers

• Programmers have to check & map the pools with names **at the beginning** of the program -> eliminates performance overhead

• Programming burden remains:
  • Programmers need to know what pools to open
  • Possible source of bugs

```python
oid1 = head
addr1 = translate oid1
oid2 = load from addr1
addr2 = translate oid2
oid3 = load from addr2
addr3 = translate oid3
load from addr3
```
Another approach - Automatic Permission Check
Another approach - Automatic Permission Check

• Automatic validation on all ObjectIDs
Another approach - Automatic Permission Check

• Automatic validation on all ObjectIDs
  + No need for programmers to reason about validation of ObjectIDs; all happen in hardware
  - Incur overhead to check permissions on all ObjectIDs
Another approach - Automatic Permission Check

• Automatic validation on all ObjectIDs
  + No need for programmers to reason about validation of ObjectIDs; all happen in hardware
  - Incur overhead to check permissions on all ObjectIDs

➡ Only check permission when a pool misses in translation → remove redundant checks on already mapped pools
Another approach - Automatic Permission Check

- Automatic validation on all ObjectIDs
  + No need for programmers to reason about validation of ObjectIDs; all happen in hardware
  - Incur overhead to check permissions on all ObjectIDs
  ➪ Only check permission when a pool misses in translation → remove redundant checks on already mapped pools
  - Overhead to map a pool during translation (context switch + update to OS data structures)
Another approach - Automatic Permission Check

• Automatic validation on all ObjectIDs
  + No need for programmers to reason about validation of ObjectIDs; all happen in hardware
  - Incur overhead to check permissions on all ObjectIDs
  ➪ Only check permission when a pool misses in translation → remove redundant checks on already mapped pools
  - Overhead to map a pool during translation (context switch + update to OS data structures)
  ➪ Hardware maps pools using their physical address, avoids OS interaction
IMPLEMENTATION

- Overview of SPOT
- Design details of SPOT
SPOT Overview

- Persistent Object Lookaside Buffer (POLB)
- System Persistent Object Table (SPOT)
- System Memory

Translated physical address to access cache
SPOT Overview

• Built on prior work of hardware-supported translations
SPOT Overview

- Built on prior work of hardware-supported translations
- New instructions: nvld and nvst
SPOT Overview

- Built on prior work of hardware-supported translations
- New instructions: nvld and nvst
- Small buffer near the core for translation: POLB, similar to TLB
SPOT Overview

- Built on prior work of hardware-supported translations
- New instructions: `nvld` and `nvst`
- Small buffer near the core for translation: POLB, similar to TLB
- **System Persistent Object Table (SPOT)**
  - ★One SPOT in whole system, because pool ID is unique
SPOT Overview

- Built on prior work of hardware-supported translations
- New instructions: nvld and nvst
- Small buffer near the core for translation: POLB, similar to TLB
- **System Persistent Object Table (SPOT)**
  - ★ One SPOT in whole system, because pool ID is unique
  - ★ Holds translation information of all pools (physical addresses)
SPOT Overview

• Built on prior work of hardware-supported translations
• New instructions: nvld and nvst
• Small buffer near the core for translation: POLB, similar to TLB
• **System Persistent Object Table (SPOT)**
  ★ One SPOT in whole system, because pool ID is unique
  ★ Holds translation information of all pools (physical addresses)
  ★ Holds permissions of all pools (UNIX file permissions)
SPOT Walk

- When executing NVLD, Pool ID is searched in POLB

**Permanent Storage**
- System Persistent Object Table (POT)
- Persistent Object Look-aside Buffer (POLB)

**NVLD**
- Pool ID
- Offset
SPOT Walk

- When executing NVLD, Pool ID is searched in POLB
- A pool entry misses in POLB = a pool is unmapped in current process
SPOT Walk

- When executing NVLD, Pool ID is searched in POLB
- A pool entry misses in POLB = a pool is unmapped in current process
- Perform SPOT Walk:

<table>
<thead>
<tr>
<th>NVLD</th>
<th>Pool ID</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>POLB</td>
<td>Pool ID</td>
<td>Phys. addr.</td>
</tr>
<tr>
<td>Core</td>
<td>SPOT Walk</td>
<td></td>
</tr>
<tr>
<td>Permanent Storage</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SPOT Walk

- When executing NVLD, Pool ID is searched in POLB
- A pool entry misses in POLB = a pool is unmapped in current process
- Perform SPOT Walk:
  1. Check permissions
SPOT Walk

• When executing NVLD, Pool ID is searched in POLB
• A pool entry misses in POLB = a pool is unmapped in current process
• Perform SPOT Walk:
  1. Check permissions
  2. Map the pool, by moving the physical address to POLB (no need for context switch)
SPOT Walk

• When executing NVLD, Pool ID is searched in POLB
• A pool entry misses in POLB = a pool is unmapped in current process
• Perform SPOT Walk:
  1. Check permissions
  2. Map the pool, by moving the physical address to POLB (no need for context switch)
Permission Check Logic

- Hierarchical design of SPOT; one level of SPOT table has entries with permission info.
- SPOT entry stores the owner and group of the pool, along with the Owner Permission (OP), Group Permission (GP), and other permission (P).
- Need OS to set protected registers of the running program.

<table>
<thead>
<tr>
<th>SPOT Entry</th>
<th>Addr. of physical page frame</th>
<th>OP</th>
<th>GP</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 52</td>
<td>127</td>
<td>96</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Owner ID</td>
<td>Group ID</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Protected register(s)

- Running user ID
- Primary* group ID

* Considering a user might belong to multiple groups, select one of the group.
Permission Check Logic

- First User ID is compared against Owner ID

```
<table>
<thead>
<tr>
<th>SPOT Entry</th>
<th>Addr. of physical page frame</th>
<th>Owner ID</th>
<th>Group ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>96</td>
<td>64</td>
<td>64</td>
</tr>
</tbody>
</table>
```

Running user ID
Permission Check Logic

- First User ID is compared against Owner ID

```
<table>
<thead>
<tr>
<th>SPOT Entry with permission info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr. of physical page frame</td>
</tr>
<tr>
<td>OP GP P</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Owner ID</th>
<th>Group ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>96</td>
</tr>
</tbody>
</table>

Running user ID

Owner ID

match?
Permission Check Logic

- First User ID is compared against Owner ID
- If matched, check the Owner Permission (OP) to determine the read/write permission of the pool
Permission Check Logic

- First User ID is compared against Owner ID
- If matched, check the Owner Permission (OP) to determine the read/write permission of the pool

<table>
<thead>
<tr>
<th>SPOT Entry with permission info</th>
<th>Addr. of physical page frame</th>
<th>OP</th>
<th>GP</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>127</td>
<td>96</td>
<td>64</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

User and owner matched

\[ \text{OP} = 00 \] ?

Yes

No permission, abort program
Permission Check Logic

- First User ID is compared against Owner ID
- If matched, check the Owner Permission (OP) to determine the read/write permission of the pool

User and owner matched

- $OP = 00$ ? Yes
  - No permission, abort program
- $OP = 1x$ ? Allow write
- $OP = 01$ ? Read-only
Permission Check Logic

- First User ID is compared against Owner ID
- If User ID not matched, check the primary group ID
Permission Check Logic

- First User ID is compared against Owner ID
- If User ID not matched, check the primary group ID
Permission Check Logic

- First User ID is compared against Owner ID
- If user ID not matched, check the primary group ID
- If matched, check the Group Permission (GP) to determine the read/write permission of the pool

<table>
<thead>
<tr>
<th>Addr. of physical page frame</th>
<th>127</th>
<th>96</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP</td>
<td>GP</td>
<td>P</td>
<td></td>
</tr>
</tbody>
</table>

Primary group ID and Group ID matched
Permission Check Logic

- First User ID is compared against Owner ID
- If user ID not matched, check the primary group ID
- If matched, check the Group Permission (GP) to determine the read/write permission of the pool
Permission Check Logic

- First User ID is compared against Owner ID
- If user ID not matched, check the primary group ID
- If matched, check the Group Permission (GP) to determine the read/write permission of the pool

*Because the user and the owner might belong to other groups.*
Permission Check Logic

- First User ID is compared against Owner ID
- If user ID not matched, check the primary group ID
- If matched, check the Group Permission (GP) to determine the read/write permission of the pool

Primary group ID and Group ID matched

- \[ = 00 \] ? Yes Trap to OS*
- \[ = 1x \] ? Allow write
- \[ = 01 \] ? Read-only

*Because the user and the owner might belong to other groups.
More in the paper

• If user ID and group ID both doesn’t match, check Other Permission bits (P).
• Hierarchical SPOT walk
• Permission information and translation information stored at different granularities
EVALUATIONS

- Methodology - Benchmarks and different designs
- Overall Performance on out-of-order processors
Methodology - Benchmarks

Three pool usage patterns for each microbenchmark:

- **ALL**: all nodes in one pool
- **EACH**: each node in different pool (1000~10000 pools)
- **RANDOM**: nodes are randomly scattered in 32 pools

Benchmarks: **Linked List**, Red-Black Tree, B-Tree, B+ Tree, Binary Search Tree, String Position Swap and TPC-C and vacation

* Each color represent one pool
Methodology - Comparisons

- All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwT-Base</td>
<td>Software translation</td>
<td>Separate check before translation</td>
</tr>
<tr>
<td>SwT-Opt</td>
<td>Software translation</td>
<td>Check permission during translation</td>
</tr>
<tr>
<td>HwT</td>
<td>Hardware translation</td>
<td>Separate check before translation</td>
</tr>
<tr>
<td>HwT+SPOT</td>
<td>Hardware translation</td>
<td>Check permission during translation</td>
</tr>
</tbody>
</table>
Methodology - Comparisons

• All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th></th>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwT-Base</td>
<td>Software translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>SwT-Opt</td>
<td>Software translation</td>
<td>Check permission during translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT</td>
<td>Hardware translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT+SPOT</td>
<td>Hardware translation</td>
<td>Check permission during translation</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Methodology - Comparisons

- All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwT-Base</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>SwT-Opt</td>
<td>Check permission during translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT+SPOT</td>
<td>Check permission during translation</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Methodology - Comparisons

- All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwT-Base</td>
<td>Software translation</td>
<td>Separate check before translation</td>
</tr>
<tr>
<td>SwT-Opt</td>
<td>Software translation</td>
<td>Check permission during translation</td>
</tr>
<tr>
<td>HwT</td>
<td>Hardware translation</td>
<td>Separate check before translation</td>
</tr>
<tr>
<td>HwT+SPOT</td>
<td>Hardware translation</td>
<td>Check permission during translation</td>
</tr>
</tbody>
</table>
Methodology - Comparisons

- All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwT-Base</td>
<td>Software translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>SwT-Opt</td>
<td>Software translation</td>
<td>Check permission during translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT</td>
<td>Hardware translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td>HwT+SPOT</td>
<td>Hardware translation</td>
<td>Check permission during translation</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Methodology - Comparisons

• All assume programmers don’t open the pools with names; **need permission check on all ObjectIDs**

<table>
<thead>
<tr>
<th></th>
<th>Translation Scheme</th>
<th>Permission Check</th>
<th>Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SwT-Base</strong></td>
<td>Software translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td><strong>SwT-Opt</strong></td>
<td>Software translation</td>
<td>Check permission during translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td><strong>HwT</strong></td>
<td>Hardware translation</td>
<td>Separate check before translation</td>
<td>Trap to OS</td>
</tr>
<tr>
<td><strong>HwT+SPOT</strong></td>
<td>Hardware translation</td>
<td>Check permission during translation</td>
<td>Hardware</td>
</tr>
</tbody>
</table>
Results - Out-of-order

<table>
<thead>
<tr>
<th>Bench</th>
<th>SwT-Opt</th>
<th>HwT</th>
<th>HwT+SPOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>EACH</td>
<td>1</td>
<td>1.5</td>
<td>2.5</td>
</tr>
<tr>
<td>RANDOM</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

* Normalized to SwT-Base
Results - Out-of-order

- No big impact on ALL

* Normalized to SwT-Base
Results - Out-of-order

- No big impact on ALL
- HwT is worse than SwT-Opt because of unnecessary checks
• No big impact on ALL
• HwT is worse than SwT-Opt because of unnecessary checks
• HwT+SPOT is the best, especially in RANDOM (2.9x)
Other Results

• Comparison between performances on in-order and out-of-order processors
• Sensitivity analysis on page size
• Storage overhead of SPOT
Summary

• Permanent pointers can point to unmapped regions, and reasoning about them is a new programming burden
• System Persistent Object Table (SPOT) holds all created pools for fast permission checking and as backing store for translation
• Evaluate permission checking and translation implementations in both software and hardware and evaluate them
• Show compelling 2.9x speedup on RANDOM workloads and 1.4x and 1.8x speedup on TPC-C and vacation
THANK YOU!

Questions?