A Low-Cost Memory Remapping Scheme for Address Bus Protection

Lan Gao*, Jun Yang†, Marek Chrobak‡,
Youtao Zhang§, San Nguyen‡, Hsien-Hsin S. Lee†

*University of California, Riverside
†University of Pittsburgh
‡Georgia Institute of Technology

Outline

- Background & Motivation
  - Secure Processor Model
  - Address Information Leakage
- Previous Address Bus Protection Solutions
  - The HIDE Scheme
  - The Shuffle Scheme
- Our Low-Cost Address Permutation Scheme
- Performance Evaluation
- Conclusion

Secure Processor Model

[Image: Diagram of a secure processor model with a processor chip and various components like CPU, RAM, and network interface.]

Address Information Leakage

[X. Zhuang, ASPLOS 2004]

| Address Sequence |  \(abc\ abc\ abc\ ...\) |
| CFG Hint        | Loop          |
| "naive"         | \(m\)         |
| "square root"   | \(m + 2\sqrt{m}\) |
| "hierarchical"  | \(O(\log^2 r)\) |

Oblivious Memory Access

- The idea: [Cedad Goldreich et al.]
  - Replace each memory access by a sequence of redundant accesses
  - Satisfactory from a theoretical perspective
- Overhead:

<table>
<thead>
<tr>
<th>Memory</th>
<th>(m)</th>
<th>(m + 2\sqrt{m})</th>
<th>(O(\log^2 r))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Runtime</td>
<td>(r \cdot m)</td>
<td>(O(\sqrt{m}))</td>
<td>(O(\log^2 r))</td>
</tr>
</tbody>
</table>
The HIDE Cache

- The Idea: break the correlation between repeated addresses [Xiaolong Zhang et al. ASPLOS 2004]
  - Permute the address space at suitable intervals
  - Permute blocks within a "chunk"
- How: Lock and Permute
  - Lock a block in the cache
  - A new read from memory
  - A dirty block since last permutation
  - Permute a chunk when replacing a locked block

HIDE Cache: An Example

- CPU Sequence: Read 0, 1, 2, 3, 8, 0, Write 1, 3, Read 9
- Block 1 relocated twice before it's replaced
- 50% brought on-chip when C0 is permuted

Increased Memory Accesses

- Breakdown of memory traffic for different chunk sizes

The Shuffle Buffer

- The Idea: dynamic control flow obfuscation [X. Zhang et al., CASES 2004]
  - Relocate a block if they appeared on the bus once
- How: Random Swap
  - Any newly read block is inserted into a shuffle buffer
  - A buffered block is written back to the address of the newly read block
  - Only read/write access pairs are observed on the address bus
Shuffle Buffer: An Example

<table>
<thead>
<tr>
<th>Accesses</th>
<th>Shuffle Buffer</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td></td>
<td>Page 0</td>
</tr>
<tr>
<td>0,1,2,3</td>
<td>0 1 2 3 4 5 6 7 8 9 15</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0 1 2 3 4 5 6 7 8 9 15</td>
<td></td>
</tr>
<tr>
<td>1,3</td>
<td>0 1 2 3 4 5 6 7 2 9 15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0 1 8 2 4 5 6 7 2 9 15</td>
<td></td>
</tr>
<tr>
<td>Finish</td>
<td>0 9 8 3 4 5 6 7 2 1 15</td>
<td></td>
</tr>
</tbody>
</table>

Increased Page Faults

Page Fault Curve for gcc

Outline

- Background & Motivation
  - Secure Processor Model
  - Address Information Leakage
- Previous Address Bus Protection Solutions
  - The HIDE Scheme
  - The Shuffle Scheme
- Our Low-Cost Address Permutation Scheme
- Performance Evaluation
- Conclusion

Our Scheme

- Goals:
  - Avoid wasteful memory traffic
    - Eliminate wasteful permutations
  - Avoid wasteful reads/writes in each permutation
  - Preserve locality and keep the page fault rate low
- How: RR Block Permutation
  - Permute only on-chip blocks of the same chunk
  - Permute only when an RR (Recently Read) block is to be replaced

RR Block Permutation Overview

RR Block Permutation: An Example

Events:

Initialy load x,y,z from memory
read miss replace x permutation write hit read miss replace y

Cache:

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mem:

m[a1]=x
m[a2]=y
m[a3]=z
m[a4]=x
m[a5]=y
m[a6]=z
m[b1]=x
m[b2]=y
m[b3]=z
m[b4]=x
m[b5]=y
m[b6]=z

Only x is written back
Only y is written back
### The Search Algorithm

--- Permute Sufficient Number of Blocks

1. Replace block A in Page \( P_k \)
2. \( C_k = \# \) of accessed blocks in page \( P_k \)
3. \( \sum C_i = \# \) of accessed blocks in sector

- **YES**
  - \( C_k < 128 \)
  - Read \((128 - C_k)\) blocks on-chip

- **NO**
  - \( \sum C_i < 128 \)
  - Search Page \( P_k \)’s neighbor until 128 blocks are found

- **NO**
  - Permit the 128 selected blocks

### Security Strength

- Between two permutations, all addresses on the bus are different
- The easiest case: A block being mapped to the \( n \)th writeback \( \rightarrow \times \frac{1}{128} \times \frac{1}{128} \)
- It becomes more difficult to make a correct guess with these uncertainties:
  - No clear indication when a permutation happens
  - No fixed set of on-chip blocks that participate in a permutation

### Outline

- Background & Motivation
  - Secure Processor Model
  - Address Information Leakage
- Previous Address Bus Protection Solutions
  - The HIDE Scheme
  - The Shuffle Scheme
- Our Low-Cost Address Permutation Scheme
- Performance Evaluation
- Conclusion
Experiment Environment

- **Tools**
  - Simplescalar Toolset 3.0
  - SPEC2K benchmarks
- **Configuration**
  - **Cache**
    - Separate L1 I- and D-cache: 8K, 32B line
    - Integrated L2 Cache: 1M, 32B line
    - Chunk Size: 8K, 16K, 32K, 64K
  - **Other Settings**
    - Page Settings: 4KB, perfect LRU repl policy
    - Perfect auxiliary on-chip storage for all schemes

Memory Traffic Comparison

Page Faults Comparison

Conclusion

- Proposed an efficient address permutation scheme to combat the information leakage on the address bus
- Tackled two main problems of the previous schemes:
  - The excessive memory traffic in the HIDE scheme
  - The increased page faults in the Shuffle scheme
- **Preliminary experiments:**
  - Reduce the memory traffic in HIDE from 12X to 1.88X
  - Keep the page fault rate as low as the base settings

Thanks

Questions?