Mondrian Memory Protection

Presenter:
Mohammad Moghimi
CSE Dept.
UCSD

Most of the slides are taken from:
http://groups.csail.mit.edu/cag/scale/papers/mmp-asplos2002-slides.ppt
Software Has Needs

• Plug-ins have won as the extensible system model.
  □ Fast & data sharing is convenient.

• Software is written for a model not directly supported by current hardware and OSes.
  □ No protection.
Mondrian Memory Protection

- Single address space split into multiple protection domains.
- A domain owns a region of the address space and can export privileges to another domain.
Memory Protection Requirements

• Small:
  □ Sharing granularity can be smaller than a page.

• Different:
  □ Different protection domains may have different permissions on the same memory piece

• Revoke:
  □ The protection domain that owns a memory region can revoke permissions of other protection domains on that region
MMP is a Solution

• Segmentation semantics without the problems.
  □ MMP provides fine-grained protection and data sharing.
  □ MMP uses linear addressing.
  □ MMP is compatible with existing ISAs
  □ MMP has no segment registers.
  □ MMP has easy perm. Revocation.
There’s No Free Lunch

• **MMP** requires extra memory to store permissions tables.
  - Good engineering keeps tables small.
  - $< 8\%$ memory allocation
  - $< 9\%$ memory accesses

• **MMP** requires CPU & memory system resources to access tables.
  - Good engineering provides an effective cache for permissions information so table access is infrequent.
Memory Access Timeline

- VA - constructed by processor.
- LA - post segmentation.
- PA - post TLB translation.
• **MMP checks virtual addresses.**

  - Protection check only needs to happen before instruction graduation (not in critical path).
MMP Implementation — Tables

**CPU**
- Domain ID (PD-ID)
- Perm. Table Base
- Protection Lookaside Buffer

**Memory**
- Permissions Table

Refill
Permission Table Requirements

- Entries should be compact.
  - 2 bits of permissions data per word (none, read-only, read-write, execute-read).

- Should represent different sized regions efficiently.
  - Any number of words at a word boundary.

- solution:
  - Organized like a hierarchical page table (trie).

<table>
<thead>
<tr>
<th>Perm Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>no perm</td>
</tr>
<tr>
<td>01</td>
<td>read-only</td>
</tr>
<tr>
<td>10</td>
<td>read-write</td>
</tr>
<tr>
<td>11</td>
<td>execute-read</td>
</tr>
</tbody>
</table>
Representing Large Regions Efficiently

- Upper level entries are typed, enabling large entries.

1st level
256KB sub-blocks

2nd level
256B sub-blocks

3rd level – 4B sub-blk

2 bits per sub-block
Representing Large Regions Efficiently

- Upper level entries are typed, enabling large entries.

1\textsuperscript{st} level
256KB sub-blocks

2\textsuperscript{nd} level
256B sub-blocks

3\textsuperscript{rd} level – 4B sub-blk

2 bits per sub-block
Compressing The Entry Format

- Most words have same perm. as neighbor.
  - Compressed entries represent longer, overlapping regions.
  - Compressed entries are the same size, but represent more information.
Register Sidecars

• Sidecars allow permissions checks without accessing the PLB (register level cache).
  - Has base, bounds and permissions information.
  - Cheaper than going to PLB.

• Increased hit rate with compressed entry format because non power-of-two sized regions are not fully indexed by PLB.
Sidecar Permissions Check Flow

- PC has its own sidecar.

### Diagram Explanation

- **Instruction**
  - OP
  - RS
  - IMM

- **Sidecar Regs**
  - Base
  - Bound
  - Perm

- **Base ≥ Addr. ≥ Bound**

- **Read/Write**

- **CK**

- **Access PLB**
  - OK
  - Fault

- **Fault**

- **Base Bound Perm**

- **Sidecar Permissions Check Flow**
  - **Has its own sidecar.**
MMP Timeline With Translation

- **MMP** can add an offset to the **VA**, providing translation.
  - Protection check happens on pre-translated address.
Why Translation?

- Implement zero-copy networking.
- Translation lets memory discontiguous in one domain appear contiguous in another.
- No cache aliasing problem, translation before cache access.
MMP Networking Results

• Simulated a zero-copy networking implementation that uses unmodified read system call.

• Eliminates 52% of memory references relative to a copying implementation.
  - Win includes references to update and read the permissions tables.
  - 46% of reference time saved.
Possible Applications

• Safe kernel modules.
  □ Safe plug-ins for apache and web browsers.

• Eliminate memory copying from kernel calls.
  □ Provide specialized kernel entry points.

• Support millions of threads, each with a tiny stack.

• Implement C++ const.

• Make each function its own protection domain.
  □ Buffer overrun much more difficult.
• Fine-grained protection is the solution for safe, extensible systems.

• Fine-grained protection can be provided efficiently.

• Mondrian Memory Protection will enable more robust software.
  - It matches the way we think about code.
  - It can be adopted incrementally (e.g., 1st just change malloc library).
Questions?
MMP’s Performance

• **Coarse Grained Protection**: <1% extra memory access and <1% memory usage

• **Fine Grained Protection**: <8% extra memory access and <8% memory usage

• **Speed**: <12% added to execution cycle

• **Reference:**
  - Emmett Witchel’s PhD Thesis:
Nooks Architecture: references


Crashes Today

User Program

Kernel

Driver

User Program
Crashes Today

User Program

User Program

Kernel

Driver
Crashes Today

User Program

User Program

Kernel

Driver
Vision

User Program

User Program

Kernel

Driver
Vision

User Program

User Program

Kernel

Driver
Nooks Isolation Manager (NIM)
Isolation - Memory

User Program

Kernel

Driver
Stack
Heap

User Program

Lightweight Kernel Protection Domains
Isolation - Control Transfer

User Program

User Program

Kernel

Driver

eXtension Procedure Call

XPC

XPC
Isolation - Data Access

User Program

User Program

Kernel

Driver

Copy-in / Copy-out
Nooks Isolation Manager (NIM)
Nooks layer are shaded
MMP vs. Nooks

- MMP is faster
- MMP needs HW support
- Nooks is easier to implement (only sw)
- Both are designed for faulty codes and not malicious codes
- Nooks is only designed for kernel extensions (e.g. drivers) but MMP is more general
- MMP is language independent while Nooks is more language specific (mostly C)
- MMP support fine granularity memory protection
Another reference

• Tarun Chopra, Memory Protection Scheme: Nooks Architecture vs. Mondrian Protection Scheme
Thanks