Mondrian Memory Protection

Emmett Witchel
Josh Cates
Krste Asanović

MIT Lab for Computer Science
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.
Currently, Protection Is Not Provided

- Plug-ins need access to different, small data structures.
  - Word level protection at word boundaries.
- Placing every possibly shared data on its own page is very difficult.
  - Some data structures imposed by hardware.
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
  - Similar to mprotect.

![Diagram showing Mondrian Memory Protection](image-url)
Word Level Protection Is Not New

• Segmentation is a traditional solution.
  o + Provides word-level protection.
  o - Explicit segment registers [B5000, x86]
  o - Non-linear addressing

• Capability based machines.
  o + Fine-grained sharing.
  o - Revocation difficult [System/38, M-machine].
  o - Different protection for different domains via shared capability is hard.
MMP is a New 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.
  - MMP does not have tagged pointers.

- MMP is all the fun of segmentation without the headaches.
There’s No Free Lunch

- MMP requires extra memory to store permissions tables.
  - Good engineering keeps tables small.

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

- Let's look at the table in memory.
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.

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

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

0-256KB no perm.

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.
MMP Implementation — PLB

- Let's look at the PLB.
PLB Requirements

• The PLB caches protection table entries tagged by Domain-ID.
  o Like a TLB but without translation.
  o Like a TLB but variable ranges, not just page sizes.
PLB Permissions Check Flow

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

- **Addr Regs**
  - +

- **PLB**
  - Tag
  - Perm Tab. Ent.
  - PD-ID
  - Hit?

- **Read/Write**
  - CK
  - OK
  - Fault

- **Access Perm. Table in Memory**

- **PC checked for execute permissions.**
PLB Requirements

- PLB task—index permissions data from different sized memory chunks.
  - Loads from different addresses can get permissions information from different levels in the table.

\[
\text{D} \quad \text{VS.} \quad \text{1}^{\text{st}} \text{ level or 2}^{\text{nd}} \text{ level}
\]
Protection Look aside Buffer (PLB)

- PLB index implemented by ternary CAM.
  - Like superpages in a TLB, but protection superpages are easy for OS—they don’t require lots of contiguous physical memory.
  - PLB index limited to power-of-two size.

<table>
<thead>
<tr>
<th>Tag (26 bits)</th>
<th>Perm. Table Ent.</th>
<th>PD-ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x07 XX XX</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>0x09 87 XX</td>
<td>D</td>
<td>0</td>
</tr>
<tr>
<td>0x09 20 58</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

- The compressed format has intermediate number of don’t-care bits, and non power-of-two sized regions.
MMP Implementation — Sidecars

- Let's look at the sidecars.

CPU
- Domain ID (PD-ID)
- Perm. Table Base

Sidecarts
- Protection
- Lookaside Buffer

Memory
- Permissions Table

Refill

Refill
Register Sidecars

- Sidecars allow permissions checks without accessing the PLB (register level cache).
  - Base, bounds and permissions information in sidecar.
  - Lower access energy for sidecar than PLB.

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

- PC has its own sidecar.

Instruction

OP | RS | IMM

Address Regs

Sidecar Regs
- Base
- Bound
- Perm

Base ≥ Addr. ≥ Bound

Read/Write

CK

Yes

Read/Write

OK

Fault

Access PLB
Coarse-Grained Evaluation

- Coarse-grained protection equivalent to current UNIX semantics (text, ro-data, data, bss, stack).
  - One protection domain.

- Application mix from SPEC2000, SPEC95, Java, Media bench, and Olden.
  - Compiled with gcc -O3 (egcs-1.0.3)
  - Address traces fed to MMP simulator.
## Coarse-Grained Protection Results

<table>
<thead>
<tr>
<th></th>
<th>60 Entry PLB</th>
<th>60 Entry TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. to MMP tables / Application refs</td>
<td>0.00–0.56%</td>
<td>0.00–2.59%</td>
</tr>
<tr>
<td>Table size / App. data</td>
<td>0.04–0.62%</td>
<td>0.02–0.22%</td>
</tr>
<tr>
<td>Sidecar miss rate</td>
<td>1–40% (12%)</td>
<td>--</td>
</tr>
</tbody>
</table>

- Comparison with TLB is just for scale, a TLB is still useful with MMP.
  - MMP is 2 bits of protection, not 4 bytes of translation + protection.
Fine-Grained Evaluation

- Fine-grained protection: Every malloc-ed region goes in its own protection region with inaccessible header words between regions.
  - malloc library is protected subsystem.

- Very demanding evaluation, almost worst case.
  - Protected subsystems will likely not have to export every region malloc-ed.
  - Functionality similar to purify.
# Fine-Grained Protection Results

<table>
<thead>
<tr>
<th></th>
<th>60 Entry PLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. to MMP tables</td>
<td>0.0–7.5% (0.1–19%)</td>
</tr>
<tr>
<td>Application refs</td>
<td>0.4–8.3%</td>
</tr>
<tr>
<td>Table size / App. data</td>
<td>0.6–11.0%</td>
</tr>
<tr>
<td>Table references</td>
<td></td>
</tr>
<tr>
<td>eliminated by sidecars</td>
<td></td>
</tr>
</tbody>
</table>

- Time and space overheads very small.
  - Results include table updates.
  - Minimal cache disturbance (study in paper).
  - Sidecar helps eliminate table references.
  - Paper compares different entry formats.
MMP Timeline With Translation

- MMP can add an offset to the VA, providing translation.
  - Protection check happens on pre-translated address.
  - Address generation is 3-to-1 add on critical path.
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.
Implementing Translation

- **MMP entry format** is flexible, allowing additional pointer types.
  - Pointer to permissions and byte-level translation offset.

- **Translation information** held in sidecar.
MMP Networking Results

- Simulated a zero-copy networking implementation that uses unmodified read system call.
  - Web client receiving 500KB.

- 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.
Related Work

• Capabilities [Dennis65, IBM AS400].
• Domain Pages [Koldinger ASPLOS92].
• Guarded pointers [Carter ASPLOS94].
• Guarded page tables [Liedke 94].
• IP longest prefix match [Waldvogel TOCS 01].
Possible Applications

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

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

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

• Implement C++ const.

• Use meta-data for cache coherence.

• Make each function its own protection domain.
  o Buffer overrun much more difficult.
Conclusion

- 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).