Transparent Superpages Support for FreeBSD on ARM

Zbigniew Bodek
zbb@semihalf.com
zbb@freebsd.org

17.05.2014 Ottawa
Presentation outline

- Virtual Memory
  - principles of operation
  - drawbacks
- Introduction to Superpages
  - basic concepts
  - implementation for ARM
- Validation and benchmarking
- Future work
Virtual Memory

Virtual Address Space

Additional level of indirection

Physical Address Space

MMU

VM subsystem

NAND

HDD

OTHER
Virtual Memory

Superpages already there

Our work focused here

Intelligent system resources management
Operates on hardware-specific stuff
Virtual Memory

- Accessing memory on ARM

[Diagram showing the MMU and TLB interaction, with annotations for CPU, VA, MMU, TLB, and PA with labels VA - Virtual Address, PA - Physical Address, and Hardware managed TLB.]
Virtual Memory

- Accessing memory on ARM
Virtual Memory

- Accessing memory on ARM

- Small TLB (speed)
- 4 KB page sizes (low fragmentation factor)
Virtual Memory

- Accessing memory on ARM

- L1 Table
- pmap
- pm_l2
- L2 Table

2 memory accesses on every TLB miss
Virtual Memory

- Accessing memory on ARM
- Limitations
  - Small TLBs (due to speed restrictions)
  - 4 KB page size
    - to maintain dense granulation and hence small fragmentation factor

SMALL TLB COVERAGE
Virtual Memory

- Accessing memory on ARM
- How to overcome?
  - Enlarge TLB?
  - Use bigger pages?
    - Allow user to decide which page size to use?
Introduction to Superpages

- Superpages technique overcomes this issue
- Reducing TLB misses

VA – Virtual Address
PA – Physical Address

CPU ➔ MMU ➔ MEM

MMU

TLB

Base page
Superpage
Base page
Introduction to Superpages

- Reservation-based allocation
Introduction to **Superpages**

- Reservation-based allocation

```
sys/arm/include/vmparam.h
```

- **VM_NRESERVLEVEL** - specifies a number of promotion levels enabled for the architecture. Effectively this indicates how many superpage sizes are used.

- **VM_LEVEL_{X}_ORDER** - for each reservation level this parameter determines how many base pages fully populate the related reservation level.
Introduction to **Superpages**

- Reservation-based allocation

```plaintext
sys/arm/include/vmparam.h

VM_NRESERVLEVEL - 1 (one superpage size will be used)

VM_LEVEL_0_ORDER - 8 (superpage will consist of 256 (1 << 8) base pages)
```
Implementation for ARMv6/v7

- Introduced support for machine-dependent portion of Superpages mechanism
  - promotion - pmap_promote_section()
  - demotion - pmap_demote_section()
  - creation - pmap_enter_section()
  - removal - pmap_remove_section()
  - shared mappings management - pmap_pv_promote/demote_section()
  - other modifications of the pmap(9) module
Implementation for ARMv6/v7

- Introduced support for machine-dependent portion of Superpages mechanism

Virtual Address Space
- Continuous in VA and PA spaces
- Identical attributes
- Identical access permissions

Physical Address Space

```
vms_reserv_level_iffullpop()
l2b_occupancy: 256
```
Implementation for ARMv6/v7

- Summarize general functionalities
  - Superpage creation
    1. Check for contiguity & attributes consistency
    2. Allocate & set up single PV entry for the superpage
    3. Create a 1MB section mapping (don’t deallocate L2)
    4. Cache + TLB maintenance (invalidate old data)
Implementation for ARMv6/v7

- Summarize general functionalities
  - Superpage creation
    - Promotion or direct mapping
    - Preferred read-only mappings (minimize disc traffic)
    - Contiguity (PA/VA) and attributes check required
    - Corresponding L2 table (and l2_bucket) preserved
    - Single PV entry for entire superpage area
Implementation for ARMv6/v7

- Summarize general functionalities
- Superpage creation

L1 Table (page directory)

VA to PA

Change L1 descriptor to a section mapping

Stash the L2 table for later (i.e. demotion)
Implementation for ARMv6/v7

- Summarize general functionalities
  - Superpage removal
    - Demote superpage when:
      - Changing attributes of the base page within
      - Paging out the base page
      - Write attempt to RO superpage
    - Remove superpage when:
      - The address map region to remove is at least superpage size
      - Quick recreation of the L2 table is not possible
Transparent Superpages Support for FreeBSD on ARM

Implementation for ARMv6/v7

- Summarize general functionalities
- Superpage removal
  - During demotion:
    - Recall old L2 table
      - recreate if there is none
      - fix-up if it is obsolete
    - Fix-up L1 table accordingly
    - Recreate PV entries basing on the superpage PV entry
Implementation for ARMv6/v7

- Summarize general functionalities
  - Superpage removal
    - During demotion:
      - L1 Table (page directory)
      - PD entry
      - Change back L1 descriptor to a valid page directory entry
      - Recall / recreate / fix-up L2 table if possible
Implementation for ARMv6/v7

- Introduced support for machine-dependent portion of Superpages mechanism
  - Support for two page sizes
    - 4 KB small page (base page)
    - 1 MB section (superpage)
  - One superpage instead of 256 base pages
    - Less TLB misses
    - Shorter translation table walk
Validation and benchmarking

- Test tools
  - GUPS (Giga Updates Per Second)
  - LMbench (STREAM)
  - Self-hosted world build
  - forkbomb
  - Hardware performance counters

- Test platform
  - Armada XP (quad core ARMv7)
Validation and benchmarking

CPU Time [s]

Updates [bn/s]

<table>
<thead>
<tr>
<th>Mmap reread %</th>
<th>Bcopy (libc) %</th>
<th>Bcopy (hand) %</th>
<th>Mem read %</th>
<th>Mem write %</th>
<th>Rand mem latency %</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.26</td>
<td>2.29</td>
<td>3.37</td>
<td>2.2</td>
<td>8.44</td>
<td>37.85</td>
</tr>
</tbody>
</table>

GCC
- 6h 36min
- 5h 14min

CLANG
- 6h 16min
- 6h 15min
Validation and benchmarking

- HW performance counters
  - Per-CPU TLB miss counter
  - Per-CPU cycles counter
- Goals:
  - Measure/estimate TLB miss penalty
  - Check TLB miss reduction due to superpages
Validation and benchmarking

- **Test plan**
  - Allocate
    - \(2 \times (\text{TLB size}) \times (\text{superpage size})\)
    - memory region
  - `realloc()`
  - `asm volatile("mcr p15, 0, %0, c9, c14, 0::r"(1));`
  - Enable user access to PMU registers. Has to be done in supervisor mode (for example kernel module)
  - Configure counter to watch particular event.
    - Only TLB miss is relevant.
    - CPU cycles are watched by a dedicated counter
Validation and benchmarking

- **Test plan**
  1. Allocate $2 \times (\text{TLB size}) \times (\text{superpage size})$ memory region
  2. Touch all 4KB pages: Addr: [0 : end]
     - Prefault all pages in the range
  3. Touch 64 pages with 1MB interval: Addr: [0 : (TLB size) \times (superpage size)]
  4. Disable PMU counters
  5. Enable PMU counters
  6. Get CPU cycles count and TLB miss count
Validation and benchmarking

- Test plan

  X - CPU cycles recorded during the test
  Y - CPU cycles for all loop iterations without TLB miss
  T - Number of all TLB misses

\[
CPM = \frac{(X - Y)}{T}
\]
Validation and benchmarking

Test results

- Cyc/TLB miss
- TLB miss number

<table>
<thead>
<tr>
<th>Cyc/TLB miss</th>
<th>TLB miss nb.</th>
</tr>
</thead>
<tbody>
<tr>
<td>157</td>
<td>32882</td>
</tr>
<tr>
<td>60</td>
<td>193</td>
</tr>
</tbody>
</table>

256 iterations through 64MB area
What's next?

- Support for 64 KB pages
  - Further performance improvement
  - More applications can use superpages
- Enable superpages by default \( sp\text{-}enabled = 1 \)
- Move all status flags from PV to PTE
  - Less overhead on promotion failure
  - Faster page management
References

- Project’s wiki page
  http://wiki.freebsd.org/ARMSuperpage

- Paper
  http://semihalf.com/download.html
Acknowledgments

- Special thanks go to:
  - Grzegorz Bernacki
  - Alan Cox

- Project mentors and sponsors:
  - Rafał Jaworowski & Bartłomiej Sięka (www.semihalf.com)
  - The FreeBSD Foundation (www.freebsdfoundation.org)
Any questions?