Multi-Core Memory Hierarchies

Lecture 12: Modern Memory Management

Tong Dong

http://mprc.pku.edu.cn/~tongdong/MMH
Virtual Memory

- Average Access Time

\[ T_{L1} = (1 - m_{L1})T_{L1}^{hit} + m_{L1}T_{L2} + m_{TLB}T_{Tablewalking} \]
Virtual Memory

- Programmer productivity, security and memory utilization
- Page and Segmentation (Reach, Range, etc.)

![Diagram of Virtual Memory](image)
Reducing TLB Miss Rate and Miss Penalty

- **Uniprocessor TLB**
  - TLB size and associativity
  - Multilevel hierarchies
  - Super pages
  - Prefetching
  - Virtual-address Cache (HPCA’16, ISCA’16)

- **CMP and Multi-core TLB**
  - Cooperative TLB Prefetching (ASPLOS’10)
  - Synergistic TLB (MICRO’10)
  - Shared Last-level TLB (HPCA’11)
Classical Virtualization

- VMM Essential Characteristics:
  - Fidelity. Software on the VMM executes identically to its execution on hardware, barring timing effects.
  - Performance. An overwhelming majority of guest instructions are executed by the hardware without the intervention of the VMM.
  - Safety. The VMM manages all hardware resources.

- VMM implementation style: Trap-and-Emulate
  - Classically virtualizable

- The most important ideas:
  - De-privileging
  - Primary and shadow structures
  - Memory traces
    - Page-protection of the shadow structures

- Software Virtualization: Binary Translation (BT)
- Hardware MMU Support: TLB
Operations intercepted by the hypervisor in a virtualized system could consume thousands of cycles of overhead to trap the condition, exit the guest, emulate the operation in the hypervisor, and return to the guest.

**AMD Nested Paging**
- If a guest page walk has n levels and a nested page walk has m levels, a 2D walk requires \( nm + n + m \) page entry references.

**Shadow page table for Virtualization**

**AMD Opteron Page Walk Cache (PWC) : Caching Page Entries**

---

**Table 1.** TLB miss frequency, latency, and performance impact

<table>
<thead>
<tr>
<th>Instruction and Data Translations</th>
<th>TLB Misses (Per 100K Inst.)</th>
<th>Walk Latency 2D/Native</th>
<th>Perfect TLB Opportunity Native</th>
<th>2D</th>
</tr>
</thead>
<tbody>
<tr>
<td>MiscServer</td>
<td>294.3</td>
<td>4.01X</td>
<td>14.0%</td>
<td>75.7%</td>
</tr>
<tr>
<td>WebServer</td>
<td>129.0</td>
<td>3.90X</td>
<td>4.7%</td>
<td>44.4%</td>
</tr>
<tr>
<td>JavaServer</td>
<td>257.0</td>
<td>3.91X</td>
<td>13.5%</td>
<td>89.0%</td>
</tr>
<tr>
<td>IntCPU</td>
<td>70.4</td>
<td>4.57X</td>
<td>11.4%</td>
<td>48.6%</td>
</tr>
<tr>
<td>FpCpu</td>
<td>18.2</td>
<td>4.43X</td>
<td>5.7%</td>
<td>27.5%</td>
</tr>
</tbody>
</table>

---

*Figure 1.* (a) Standard x86 page walk. (b) Two-dimensional page walk. Italics indicate column and row names; notations such as \( \{ nL, gPA \} \) and \( \{ gL, t \} \) indicate entries in the indicated columns and rows.

---

Two-dimensional (2D) Page Walking Cache

Figure 3. Percentage of all unique page entries for each 2D page with reference in MiscServer

Figure 4. Spatial locality of \( (n_{L_1}, gPA) \) page entries

Figure 5. 2D page walk caching designs

NT: Nested TLB
Translation Cache ISCA 2010

○ TLB Miss: 5-14% to 50% impact on performance
  – AMD’s Page Walk Cache stores page table entries from any level of the tree, resembling in Data Cache.
  – Intel’s Paging-Structure Caches implements distinct caches for each level of the tree, resembling in TLB.

○ Caching Page Walking
  – Unified Page Table Cache (UPTC)
  – Split Page Table Cache (SPTC)
  – Unified Translation Cache (UTC)
  – Split Translation Cache (STC)
  – Translation-Path Cache (TPC)

○ Translation Cache: store partial translations and allow the page walk hardware to skip one or more levels of the page table. The best!

○ Radix tables cause up to 20% fewer total memory accesses and up to 400% fewer DRAM accesses than hash-based tables (the inverted page table) because of the locality in virtual address use.

○ Coalescing and sharing MMU Cache (MICRO 2013)
Translation Cache  ISCA 2010

<table>
<thead>
<tr>
<th>Base Location</th>
<th>Index</th>
<th>Next Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>125</td>
<td>0ae</td>
<td>508</td>
</tr>
<tr>
<td>042</td>
<td>00c</td>
<td>125</td>
</tr>
<tr>
<td>613</td>
<td>0b9</td>
<td>042</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 3: An example of the contents of a UPTC. Each entry is tagged with the address of a page table entry, consisting of the 40-bit physical page number of the page table page and a 9-bit index into it. The entry then provides a 40-bit physical page number for the next lower level page table page. (Only 12 bits of the physical page numbers are shown, for simplicity.)

<table>
<thead>
<tr>
<th>L2 entries</th>
<th>Base Location</th>
<th>Index</th>
<th>Next Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>125</td>
<td>0ae</td>
<td>508</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L3 entries</th>
<th>Base Location</th>
<th>Index</th>
<th>Next Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>042</td>
<td>00c</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>L4 entries</th>
<th>Base Location</th>
<th>Index</th>
<th>Next Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>613</td>
<td>0b9</td>
<td>042</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 4: An example of the contents of a SPTC. Each entry holds the same tag and data as in the UPTC.

<table>
<thead>
<tr>
<th>L4 index</th>
<th>L3 index</th>
<th>L2 index</th>
<th>Next Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b9</td>
<td>00c</td>
<td>0ae</td>
<td>508</td>
</tr>
<tr>
<td>0b9</td>
<td>00c</td>
<td>xx</td>
<td>125</td>
</tr>
<tr>
<td>0b9</td>
<td>xx</td>
<td>xx</td>
<td>042</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 6: An example of the contents of a UTC. An “xx” means “don’t care”.

<table>
<thead>
<tr>
<th>L4 index</th>
<th>L3 index</th>
<th>L2 index</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b9</td>
<td>00c</td>
<td>0ae</td>
<td>042</td>
<td>125</td>
<td>508</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 7: An example of the contents of the TPC after the virtual address (0b9, 00c, 0ae, 0c2, 016) is walked. The TPC holds three 9 bit indices, as the translation caches do, but all three 40-bit physical page numbers are stored for all three page table levels.
FreeBSD’s reservation-based physical memory allocator

- After a program uses every 4KB page within an entire 2MB region of virtual memory, that contiguous region is promoted to a large page.
- Consecutive virtual pages will also be consecutive physical pages

SpecTLB

- Exploits the contiguity and alignment generated by a reservation-based physical memory allocator to interpolate address translations based on the physical address of nearby pages.
- SpecTLB can remove 40% of DRAM accesses; Prediction accuracy is exceeding 99%; only needs moderate size, tens for high hit ratio.

| Table 1: Memory accesses to the page table in different phases of the nested page walk for an average of all benchmarks examined. The table is split into those TLB misses where diagonal speculation was not attempted or failed and those where diagonal speculation was successful. |
|---------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|
|                                | nL4 | gL4 | nL3 | gL3 | nL2 | gL2 | nL1 | gL1 |
| No diagonal speculation        | L2 accesses | 0.005 | 0.109 | 0.066 | 0.149 | 0.105 | 0.359 | 0.142 | 0.372 | 0.199 |
|                                | DRAM accesses | 0.000 | 0.048 | 0.033 | 0.075 | 0.049 | 0.161 | 0.142 | 0.372 | 0.199 |
| After diagonal speculation     | L2 accesses | 0.000 | 0.029 | 0.028 | 0.033 | 0.031 | 0.044 | 0.057 | 0.946 | 0.946 |
|                                | DRAM accesses | 0.000 | 0.028 | 0.028 | 0.028 | 0.029 | 0.031 | 0.033 | 0.197 | 0.046 |

Speculative Error Overhead?
Basic OS memory allocation mechanisms such as buddy allocators and memory compaction naturally assign contiguous physical pages to contiguous virtual pages.

Coalesced Large-Reach TLBs (CoLT), which leverage this intermediate contiguity to coalesce multiple virtual-to-physical page translations into single TLB entries.

CoLT implementations eliminate 40% to 58% of TLB misses on average, improving performance by 14%.
For the majority of their address space, big-memory workloads use read-write permission on most pages, are provisioned not to swap, and rarely benefit from the full flexibility of page-based virtual memory.

Mapping part of a process’s linear virtual address space with a direct segment, while page mapping the rest of the virtual address space.

Figure 3. Virtual Address and Physical Address layout with a primary region. Narrow rectangles represent pages.

Figure 5. DTLB miss overheads when scaling up GPUs.

Coalesced and Shared Memory Management Unit Caches to Accelerate TLB Miss Handling

A hardware/software coalescing technique
- Low-overhead operating system (OS) page table allocation mechanisms that maximize coalescing opportunity
- Complementary enhancements to standard MMU caches to detect and exploit coalescing patterns
- Replacing per-core MMU caches with a unified MMU cache shared by all cores

A. Bhattacharjee. 2013. Large-reach memory management unit caches. MICRO-46. (49)
Observation: many more translations exhibit “clustered” spatial locality in which a group or cluster of nearby virtual pages map to a similarly clustered set of physical pages.

A multi-granular TLB organization that significantly increases its effective reach and reduces miss rates substantially while requiring no additional OS support.

Clustered TLB HPCA 2014

B. Pham, et al. 2014. Increasing TLB reach by exploiting clustering in page translations. HPCA-20. (47)
Super-Page

- x86-64
  - 4KB, 2MB, 1GB
- Sparc T4
  - 8KB, 64KB
  - 4MB, 256MB
  - 2GB
- ARM
  - 1KB Tiny page
  - 4KB, 64KB
  - 1MB, 16MB
- ~50% performance Improvement by using Superpage. [CAN’07]
GTSM, or gap tolerant sequential mapping, that allows superpages to be formed even in the presence of retired physical pages.

Page Retirement is used to avoid uncorrectable errors in memory.
  - Memory Errors, Power, Non-volatile Memory

GTMS, construct superpages from non-contiguous physical memory. By utilizing a block selection bitmap, a superpage is mapped to multiple equal-sized small memory blocks (i.e., physical pages) instead of a single large contiguous memory block.
A set of commercial and scale-out applications exhibit significant use of superpages and suffer from the fixed and small superpage TLB structures.

Prediction-guided multi-grain TLB design (vs. Skewed TLB)
- Uses a superpage prediction mechanism to avoid multiple lookups in the common case.

Page Size Prediction

---

Fig. 3. (a) PC-based and (b) Register-Value based Page Size Predictors

Fig. 4. Multigrain Indexing (512 entries, 8-way associative TLB) with 4 supported page sizes, 6 set-index bits.
RMM: Redundant Memory Mappings ISCA 2015

- RMM adds a redundant mapping, in addition to page tables, that provides a more efficient representation of translation information for ranges of pages that are both physically and virtually contiguous.

- **Range translation** maps a contiguous virtual address range to contiguous physical pages, and uses BASE, LIMIT, and OFFSET values to perform translation of an arbitrary sized range.

- Software managed range table to map virtual ranges to physical ranges and a hardware range TLB in parallel with the last-level page TLB to accelerate their address translation.

- Extend the OS’s default lazy demand page allocation strategy to perform eager paging. **Eager paging** instantiates pages in physical memory at allocation request time, rather than at first-access time as with demand paging.
RMM: Redundant Memory Mappings ISCA 2015

<table>
<thead>
<tr>
<th>Transparent to application</th>
<th>Kernel support</th>
<th>Hardware support</th>
<th># of entries</th>
<th>Maximum reach per entry</th>
<th>Application domain</th>
<th>No size-alignment restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multipage Mappings [47, 39, 38]</td>
<td>✓</td>
<td>X</td>
<td>512</td>
<td>32 KB to 16 MB</td>
<td>any</td>
<td>X</td>
</tr>
<tr>
<td>Transparent Huge Pages [6, 36]</td>
<td>✓</td>
<td>✓</td>
<td>32</td>
<td>2 MB</td>
<td>any</td>
<td>X</td>
</tr>
<tr>
<td>libhugetlbfs [1]</td>
<td>X</td>
<td>✓</td>
<td>4</td>
<td>1 GB</td>
<td>big memory</td>
<td>X</td>
</tr>
<tr>
<td>Direct segments [10]</td>
<td>X</td>
<td>✓</td>
<td>1</td>
<td>unlimited</td>
<td>big memory</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Redundant Memory Mappings**
✓ ✓ ✓ N unlimited any ✓

**Table 1: Comparison of Redundant Memory Mappings with previous approaches for reducing virtual memory overhead.**

**Figure 3:** RMM hardware support consists primarily of a range TLB that is accessed in parallel with the last-level page TLB.
Virtualization software often splinters guest operating system (OS) large pages into small system physical pages.

Observation: the act of splintering a large page is usually performed to achieve finer-grained memory management rather than to fundamentally alter virtual or physical address spaces.

Generalized Large-page Utilization Enhancements (GLUE)
- GLUE augments standard TLBs to store information that identifies these contiguous, aligned, but splintered regions.
- GLUE then uses TLB speculation to identify the constituent translations.
- Small system physical pages are speculated by interpolating around the information stored about a single speculative large-page translation in the TLB.

Figure 5: The mechanics of TLB speculation. We show the case when we speculate from the 2MB L1 TLB.

Figure 6: Timelines for (a) speculating from the 2MB L1 TLB correctly, and verifying this in the L2 TLB; (b) mis-speculating from the 2MB L1 TLB, and verifying this in the L2 TLB; (c) speculating from the 2MB L1 TLB correctly, and verifying with a page table walk; (d) mis-speculating from the 2MB L1 TLB, and verifying with a page table walk; (e) speculating from the L2 TLB correctly, and verifying with a page table walk; and (f) mis-speculating from the L2 TLB, and verifying with a page table walk.

B. Pham, et al. 2015. Large pages and lightweight memory management in virtualized environments: can you have it both ways? MICRO-48. (28)
Base Naïve Paging
Nested Paging
Shadow Paging
Agile Paging

<table>
<thead>
<tr>
<th></th>
<th>Base Native</th>
<th>Nested Paging</th>
<th>Shadow Paging</th>
<th>Agile Paging</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB hit</td>
<td>fast (VA⇒PA)</td>
<td>fast (gVA⇒hPA)</td>
<td>fast (gVA⇒hPA)</td>
<td>fast (gVA⇒hPA)</td>
</tr>
<tr>
<td>Max. memory access on TLB miss</td>
<td>4</td>
<td>24</td>
<td>4</td>
<td>~ (4–5) avg.</td>
</tr>
<tr>
<td>Page table updates</td>
<td>fast direct</td>
<td>fast direct</td>
<td>slow mediated by VMM</td>
<td>fast direct</td>
</tr>
<tr>
<td>Hardware support</td>
<td>1D page walk</td>
<td>2D+1D page walk</td>
<td>1D page walk</td>
<td>2D+1D page walk with switching</td>
</tr>
</tbody>
</table>

Agile Paging: Exceeding the Best of Nested and Shadow Paging. ISCA-43. (10)
Agile Paging  ISCA 2016

- SHSP, Selective hardware software paging (VEE’11)
- Key intuition: most of the updates to a hierarchical page table occur at the lower levels or leaves of the page table.
- Agile paging allows virtualized page walk to
  - Start with the shadow paging for stable upper levels of page table and
  - Allows switching in the same page walk to nested paging for lower levels of page table which receive frequent updates.
  - Agile paging has three architectural page table pointers in hardware: one each for shadow, guest, and host page tables.
  - The shadow page table needs to logically support a new switching bit per page table entry. To allow fine grain switching from shadow paging to nested paging on any entry at any level of guest page table.

<table>
<thead>
<tr>
<th>Switch Level Mem. accesses</th>
<th>Shadow</th>
<th>L4</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
<th>Nested</th>
<th>Avg.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>8</td>
<td>12</td>
<td>16</td>
<td>20</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>memcached</td>
<td>88.2%</td>
<td>4.5%</td>
<td>7.3%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.76</td>
</tr>
<tr>
<td>canneal</td>
<td>94.7%</td>
<td>4.6%</td>
<td>0.7%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.24</td>
</tr>
<tr>
<td>astar</td>
<td>92.3%</td>
<td>7.5%</td>
<td>0.2%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.32</td>
</tr>
<tr>
<td>gcc</td>
<td>81.6%</td>
<td>11.7%</td>
<td>6.7%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>5.00</td>
</tr>
<tr>
<td>graph500</td>
<td>99.8%</td>
<td>0.2%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.01</td>
</tr>
<tr>
<td>mcf</td>
<td>99.1%</td>
<td>0.9%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.04</td>
</tr>
<tr>
<td>tigr</td>
<td>88.3%</td>
<td>7.6%</td>
<td>3.1%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.51</td>
</tr>
<tr>
<td>dedup</td>
<td>91.4%</td>
<td>2.2%</td>
<td>6.4%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>4.60</td>
</tr>
</tbody>
</table>
MIX TLB \textit{ASPLOS 2017}

- Good TLB Design
  - Good Performance and simple Implementation
  - Energy Efficient: set-associative TLB
  - Supporting multiple page size, multi-indexing problem

- MIX TLB
  - Set-indexing scheme for small pages
  - Use bits within the superpage page offset to select a TLB set which means that a superpage is mapped to multiple (potentially all) TLB sets, an operation we refer to as \textit{mirroring}.
  - Detect these adjacent superpages and \textit{coalesce} them into the same TLB entry.

\textbf{Figure 3.} Superpage B lookup and fill for split versus MIX TLBs.

In non-uniform memory systems, using large pages often leads to performance degradation, or allocating large chunks of memory becomes more difficult due to memory fragmentation.

Hybrid coalescing, a HW-SW hybrid translation architecture.
- The operating system encodes memory contiguity information in a subset of page table entries, called anchor entries.
- The most important benefit of hybrid coalescing is its ability to change the coverage of the anchor entry dynamically, reflecting the current allocation contiguity status.
POM-TLB, a very large level-3 TLB that is Part of Memory (DRAM)
- Caching TLB entries is more effective and beneficial than caching page table entries.
- Partitioned TLB: POM-TLB\textsubscript{small} (4KB) and POM-TLB\textsubscript{large} (2MB)
- Page Size Predictor
- Cache Bypass Predictor
- Cache Replacement
- Cache Coherent

\[ \text{Addr}_{\text{POM-TLB}} (VA) = ((VA \oplus VM\_ID) \gg 6) \& ((1 << \log_2 (N)) - 1) \times 64 + \text{Base\_Addr}_{\text{POM-TLB}} \]
Frequent virtual machine context switches are common, resulting in increased TLB miss rates (often, by over 5X if contexts are doubled).

CSALT mitigates data cache contention caused by conflicts between data and translation entries by employing a novel TLB-Aware Cache Partitioning scheme.

Figure 4: CSALT System Architecture

Figure 6: CSALT Overall Flowchart

Memory Management for GPU

- Shared virtual memory between CU and GPU.
  - nVidia Unified Virtual Addressing (UVA) requires special allocation and pinned memory pages
- GPU require up to 100s unique translations in a single cycle!
- Findings:
  - A per GPU CU L1 TLB should be placed after the coalescing hardware and scratchpad memory
  - A shared highly-threaded page table walker
  - High TLB miss ratio (~29%). Employing a page walk cache to reduce TLB miss penalty

---

J. Power, D. Mark Hill and David A. Wood. 2014. Supporting x86-64 address translation for 100s of GPU lanes. HPCA-20. (51)
Memory Management for GPU

- **Configuration uses 16 KB of storage.**

<table>
<thead>
<tr>
<th></th>
<th>Per-CU TLB entries</th>
<th>Highly-threaded page table walker</th>
<th>Page walk cache size</th>
<th>Shared L2 TLB entries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ideal MMU</strong></td>
<td>Infinite</td>
<td>Infinite</td>
<td>Infinite</td>
<td>None</td>
</tr>
<tr>
<td><strong>Section 4</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design 0</td>
<td>N/A: Per-lane MMUs</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design 1</td>
<td>128</td>
<td>Per-CU walkers</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Design 2</td>
<td>128</td>
<td>Yes (32-way)</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Design 3</td>
<td>64</td>
<td>Yes (32-way)</td>
<td>8 KB</td>
<td>None</td>
</tr>
<tr>
<td><strong>Section 6</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared L2</td>
<td>64</td>
<td>Yes (32-way)</td>
<td>None</td>
<td>1024</td>
</tr>
<tr>
<td>Shared L2 &amp; PWC</td>
<td>32</td>
<td>Yes (32-way)</td>
<td>8 KB</td>
<td>512</td>
</tr>
<tr>
<td>Ideal PWC</td>
<td>64</td>
<td>Yes (32-way)</td>
<td>Infinite</td>
<td>None</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>1 cycle</td>
<td>20 cycles</td>
<td>8 cycles</td>
<td>20 cycles</td>
</tr>
</tbody>
</table>

Figure 9: Details of the highly-threaded page table walker
Memory Management for GPU

- The Impact of Warp Scheduling on MMU Design
- TLB-aware CCWS (TA-CCWS)
  - VTAs for TLB and Cache Misses
- TLB conscious warp scheduling (TCWS)
  - VTAs just for TLB Misses
- TLB-aware Thread Block Compaction

**Figure 1.** Our approach embeds a TLB and PTW per shader core so that all caches become physically-addressed.

**Figure 2.** Compared to a baseline architecture without TLBs, speedup of naive, 3-ported TLBs per shader core, with and without cache-conscious wavefront scheduling, with and without thread block compaction. Naive TLBs degrade performance in every case.

B. Pichai, et al. 2014. Architectural support for address translation on GPUs: designing memory management units for CPU/GPUs with unified address spaces. ASPLOS-19. (61)
Mosaic: GPU MMU for multiple page sizes, MICRO 2017

- GPU MMU fundamental challenges:
  - (1) the address translation challenge, and
  - (2) the demand paging challenge

- Observation: GPGPU applications present an opportunity to support multiple page sizes without costly data migration, as the applications allocate a large number of base pages at once

- Mosaic uses base pages to transfer data over the system I/O bus, and allocates physical memory in a way that
  - (1) preserves base page contiguity and
  - (2) ensures that a large page frame contains pages from only a single memory protection domain.

- During data transfer, this mechanism enables the GPU to transfer only the base pages that are needed by the application over the system I/O bus, keeping demand paging overhead low.
  - (1) Contiguity Conserving Allocation (CoCoA)
  - (2) In-Place Coalescer
  - (3) Contiguity-Aware Compaction (CAC)

Mosaic: GPU MMU for multiple page sizes, MICRO 2017

Figure 6: Coalescing timeline for (a) GPU-MMU baseline and for (b) Mosaic.
Unified virtual address space between the host CPU cores and customized accelerators can largely improve the programmability. Prior studies either assume an infinite-sized TLB and zero page walk latency, or rely on a slow IOMMU which penalizes the overall system performance.

To support bulk transfers of consecutive data between the scratchpad memory of customized accelerators and the memory system, we present a relatively small private TLB design.

For the effects of the widely used data tiling techniques, we design a shared level-two TLB to serve private TLB misses on common virtual pages.

Simply leveraging the host per-core MMU for efficient page walk handling.

---

1. **Inefficient TLB Support.** TLBs are not specialized to provide low-latency and capture page locality.

2. **High Page Walk Latency.** On an IOTLB miss, four main memory accesses are required to walk the page table.
Summary

- Memory Management Units for Multi-/Many-Core
  - Page Walking
  - TLB Size

- TLB Cache and Translation Cache
  - Nested Page Table

- Multiple Page Sizes Support

- TLB Coalescing and Splinting

- Range-based Address Translation

- Hardware Support for Shadow Page Table

- Memory Management for GPUs and Accelerators