The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar1,2, Pratyush Patel3, Minesh Patel1, Konstantinos Kanellopoulos1, Saugata Ghose4, Rachata Ausavarungnirun5, Geraldo F. Oliveira1, Jonathan Appavoo6, Vivek Seshadri7, Onur Mutlu1,4

1 Motivation

Applications

Cannot adapt efficiently

Virtual Memory

managed by the operating system

Hardware

• Modern computing systems continue to diversify with respect to system architecture, memory technologies, and applications’ memory needs
• Continually adapting the conventional virtual memory framework to each possible system configuration is challenging

2 Example Challenges of the Conventional Virtual Memory Framework

1. Page tables need to be shared between, and understood by both the hardware and the OS, resulting in rigid page table structures
   - Challenging to implement the page table flexibility that applications can benefit from

2. In virtual machines, both the guest and host OS perform address translation, resulting in an extra level of indirection
   - Challenging to perform computation in virtualized environments, efficiently

3. OS that defines and manages the address mapping, has low visibility into fine-grained runtime memory behavior information
   - Challenging in heterogeneous memories to make timely migration decisions based on quickly changing memory access patterns or other dynamic behavior

3 Our Goal

Design an alternative virtual memory framework that

• Efficiently and flexibly supports increasingly diverse system configurations
• Provides the key features of conventional virtual memory framework while eliminating its key inefficiencies

4 Virtual Block Interface (VBI)

Key idea:

Delegate physical memory management to dedicated hardware in the memory controller

Guiding Principles:

1. Size virtual address spaces appropriately for processes
   - Mitigates translation overheads of unnecessarily large address spaces
2. Decouple address translation from access protection
   - Delays address translation until necessary to access memory
   - Enables the flexibility of managing translation and protection using separate structures
3. Communicate data semantics to the hardware
   - Enables intelligent resource management

5 VBI Design Overview

• Globally-visible VBI address space
  - Consists of a set of virtual blocks (VBs) of different sizes
  - Provides system-wide unique VBI addresses
  - VBI addresses are directly used to access on-chip caches
  - Pros: Enables inherently virtual caches

• All VBs are visible to all processes
  - OS controls which processes access which VBs
  - Each process has its own permissions (read/write/execute) when attaching to a VB
  - OS maintains a list of VBs attached to each process used to perform permission checks
  - Processes map each semantically meaningful unit of information to a separate VB
    - e.g., a data structure, a shared library
  - Memory management is delegated to the Memory Translation Layer (MTL) in the memory controller
    - Address translation and physical memory allocation
    - Translation structures are not shared with the OS
    - Per-VB translation structure tuned to the VB’s characteristics
  - Pros: many benefits, including
    - Address translation overhead for the processes running inside a virtual machine, is no different than the processes running natively on system
    - Enabling flexible translation structures

6 Optimizations Enabled by VBI

Naturally enabled by VBI and not easily attainable before:

• Appropriately sized process address space
• Flexible address translation structures
• Communicating data semantics to the hardware
• Inherently virtual caches
• Eliminating 2D page walks in virtual machines
• Delayed physical memory allocation
• Early memory reservation mechanism

7 Example Use Case: Address Translation

- Native: applications run natively on an x86-64 system
- Virtual: applications run inside a virtual machine (accelerated using 2D page walk cache [Bhardwaj+, ASPLOS’08])
- Perfect TLB: an unrealistic version of Native with no translation overhead
- VBI-Full: VBI with all the optimizations that it enables

Key Takeaways

1. VBI-full improves the performance, by 2.4X on average compared to Native
2. VBI-full enables significant performance improvement in virtualized environments (4.3X on average compared to Virtual)
3. VBI-full outperforms Perfect TLB (by 49%) on average as the optimizations that VBI enables are not limited to only reducing the address translation overhead

8 Conclusion

VBI is a promising new virtual memory framework

• Can enable several important optimizations
• Increases design flexibility for virtual memory
• A new direction for future work in novel virtual memory frameworks

0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0

Speedup

- Virtual
- Perfect TLB
- VBI-Full