Caching in on the TLB

Samyuktha Subramanian

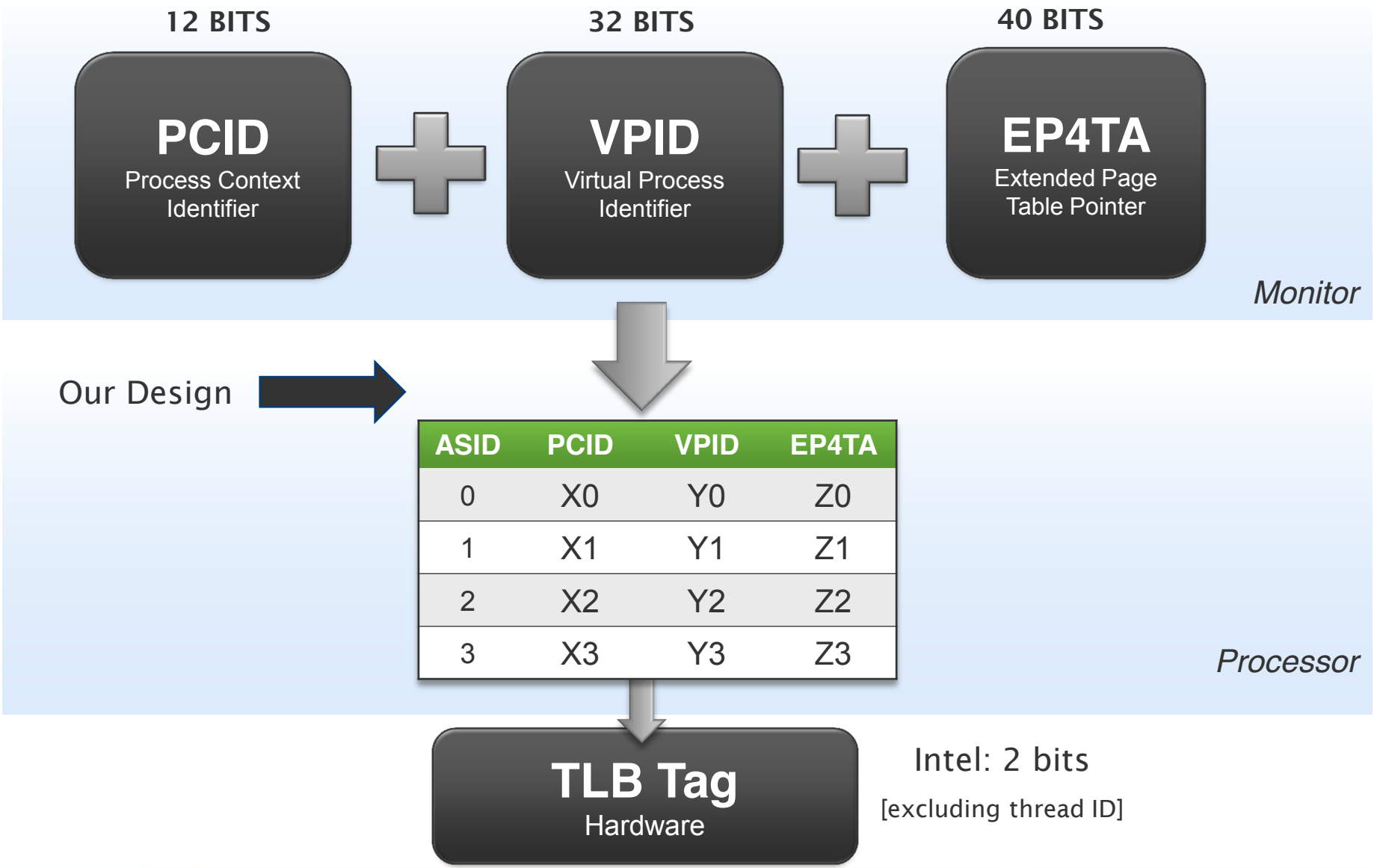
Mentor: Jim Mattson

Manager: Giri Rashiyamani

Outline

- Background
- Motivation
- Design Abstraction
- Implementation
  - Comparison
  - System Architecture
  - Some Specifics
- Results
- Challenges
- Applications

Background

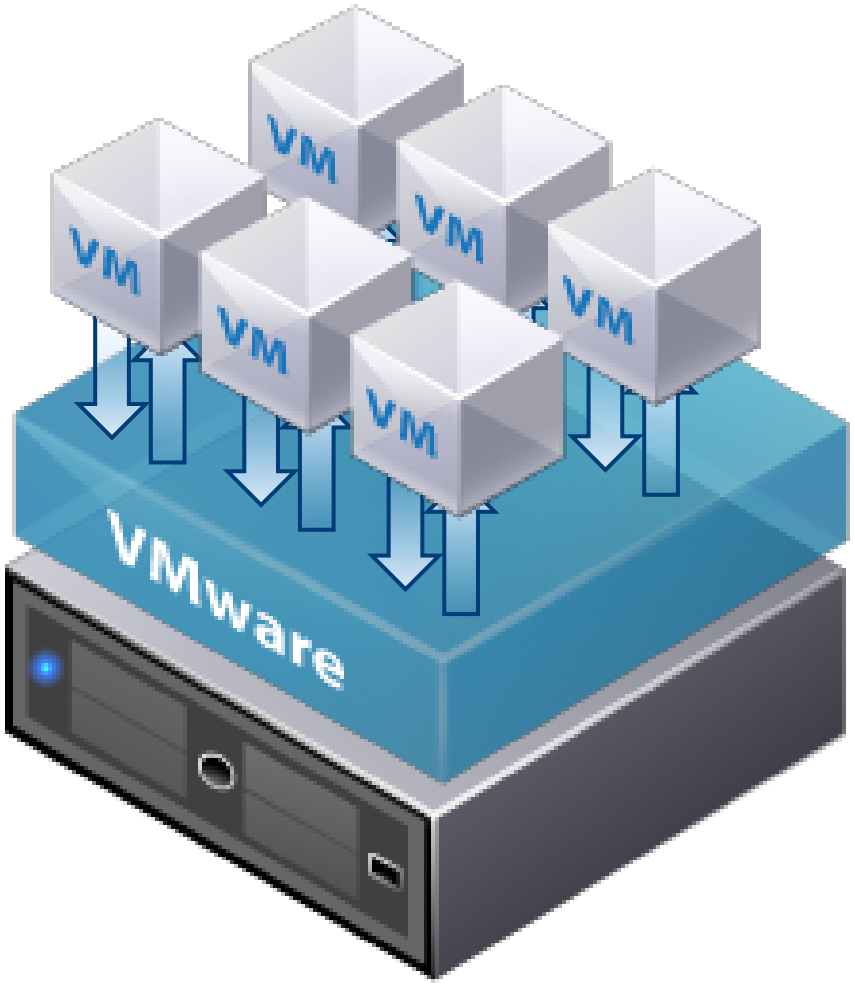


Motivation

	Pre-Haswell	Haswell
Size of the STLB	512 B	1024 B
Support: Large Pages	2 MB cached only in L1 DTLB (32 B)	STLB can now cache these pages, 32x size increase

STLB: 4 way associative *STLB: 8 way associative*

Design Abstraction



Objective

Host **multiple address spaces** from

- Virtual Machines
- Virtual Processors

Managing the TLB

Implementation

Implementation Comparisons

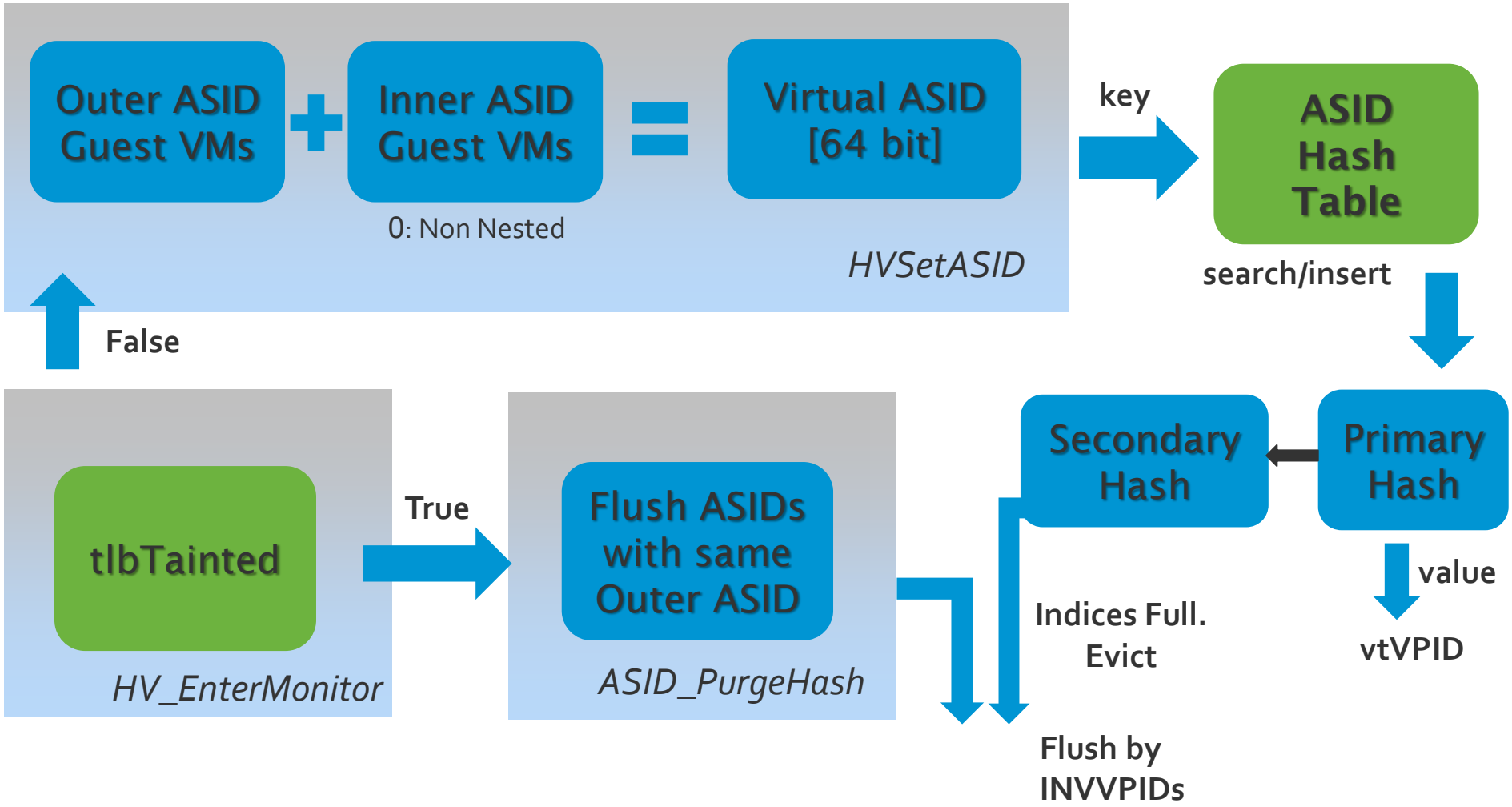
Present

- **ASID Assignment**
  - VPID 9 to all guest worlds
  - VPID 1-8 to nested guests
- **TLB Flushes**
  - pcpuTainted = TRUE
  - Set when
    - Another VMM world was run before on this physical CPU
    - This VMM world was run on another physical CPU before this
- **Summary: TLB Flushed**
  - on EVERY context switch between VMM worlds

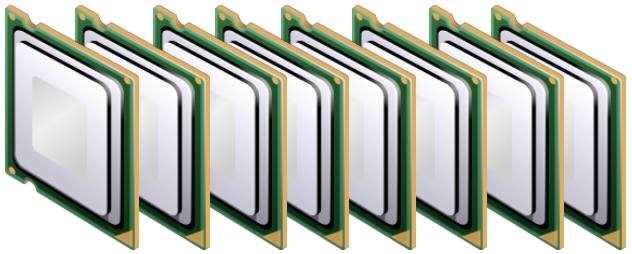
Proposed

- **ASID Assignment**
  - Unique VPID to ALL guest worlds
- **TLB Flushes**
  - tlbTainted = TRUE
  - Set when
    - Old ASID claimed by another ASID in the intervening time
    - This VMM world was run on another physical CPU before this
- **Summary: TLB Flushed**
  - when a TLB shoot-down may have occurred since the last time the world was scheduled

System Architecture



Data Structure Ownership



Physical Processors
[MAX_PCPUS]

ASID Hash Tables
[MAX_PCPUS]

structure ***pcpuASIDTable***

is a VMM⟷VMX shared structure

pointer
PerPCPUASIDTable

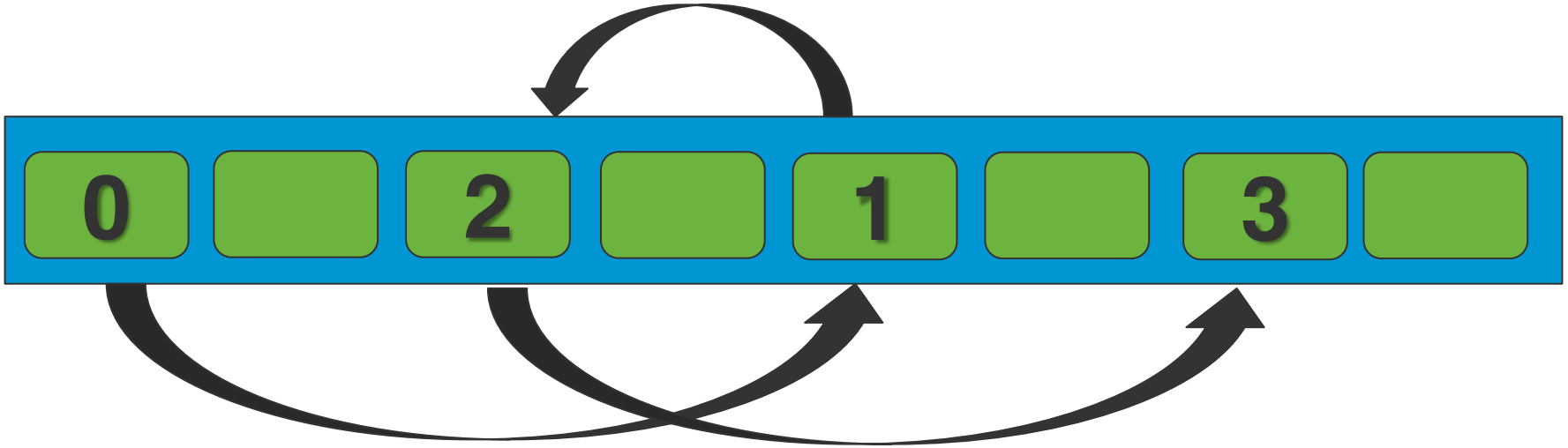
- **Initialization in *WorldSharedDataInit***
  - Monitor grabs a pointer to the structure [kernel]
- **Also copy the current value of *worldID***
  - This is used for calculating the outer ASID

Primary Hash Function

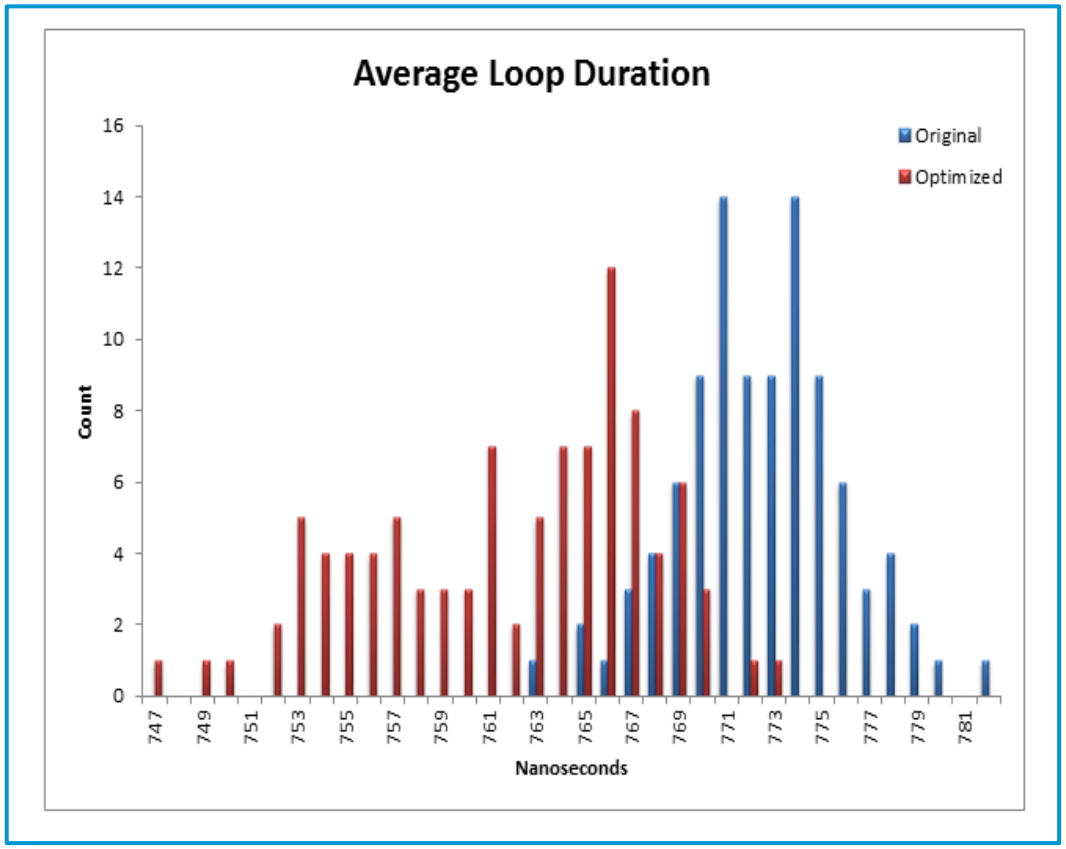
- **Requirements**
  - Primary Function

    Even distribution of consecutive ASIDs in the hash table
  - Secondary Function

    Avoid evicting a consecutive ID if the primary index is full



Results

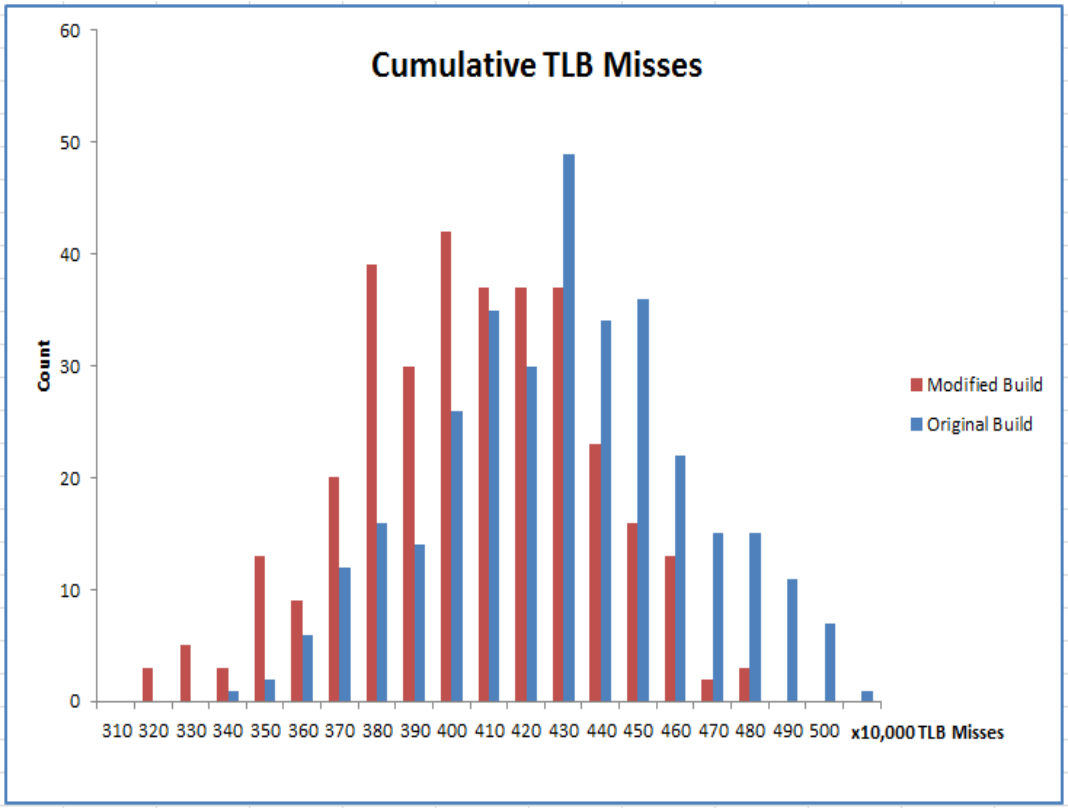


Custom Micro Benchmark

- Measures time to run a small piece of iterative code
- About a **1.4** % decrease in the number of cycles
- Mean
  - Original Build: **772** ns
  - Modified Build: **761** ns

- Inner Loop touches **512** pages
- Outer Loop runs **100, 000** times
- This runs on one VM, second VM wakes every millisecond
- Both: **1** VCPU wide, constrained to run on the same physical CPU.

Results (Cont.)



Kernel Compile Benchmark

- About a **9.5** % decrease in the number of TLB misses
- Mean (Entire Run)
  - Original Build: **428.2** thousand misses.
  - Modified Build: **386.4** thousand misses.

- Data collected simultaneously from 2 virtual machines
- Width of each virtual machine: 8 vCPUs; width of the ESX Haswell Machine: 8 pCPUs
- Time: **0.03** seconds. Not a TLB intensive benchmark

Challenges

- Hash Function Design
- Managing the Hash Table: Kernel + Monitor
- Access to the Scratch Address Space: Not always true
- Intervening TLB Shoot-down
  - World scheduled on another physical CPU before it was scheduled on this CPU
- Concurrency
  - Two worlds simultaneously trying to get the same ASID

Possible Application

- **Limitation**
  - Calculating the World Switch Rate to get an increase in performance of **1%**:

    Cost of an STLB Miss: **30** cycles

    STLB Size: **1024** B

    Savings = Cost x Size x Frequency

    Frequency should be **780** switches/s.
- **Possible Applications**
  - Windows VM on a 1 kHz timer

    Running a Flash player or Multimedia

    Competing with a VM running an app that has a significant TLB footprint.

Questions?