FreeBSD TLB shootdown enhancement in Azure

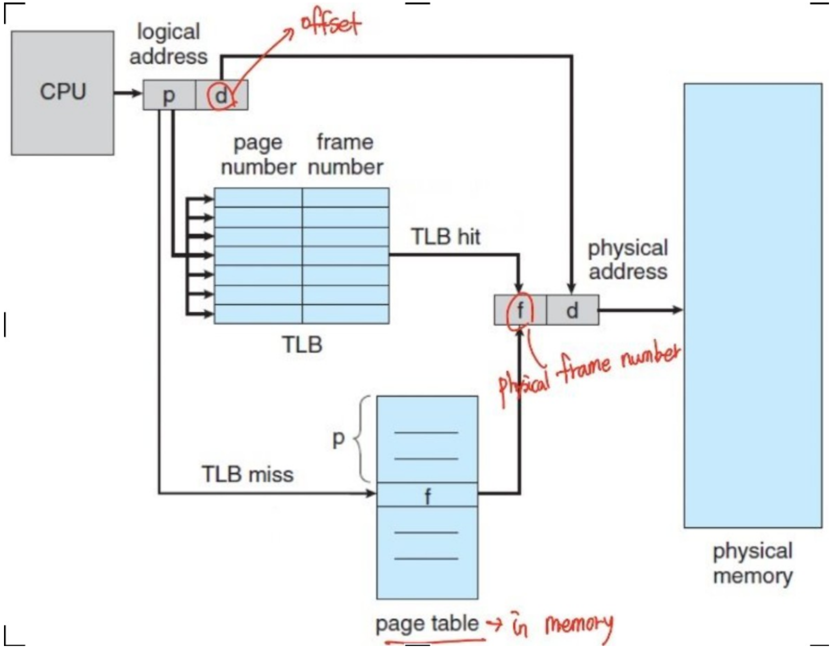
- Souradeep Chakrabarti
- Wei Hu

FreeBSD in Azure

- On June 8, 2016, a standard FreeBSD 10.3 image was published into the Azure Marketplace. Microsoft published the image working as part of the FreeBSD community and in collaboration with the FreeBSD Foundation.
- Over the year Microsoft has worked with FreeBSD community to enable support for different h/w and Azure Hyper-V features in FreeBSD.
- Currently Azure supports FreeBSD in
  - Gen2:
    - amd64
    - arm64
  - Gen1:
    - amd64

TLB – Translation Lookaside Buffer (x86)

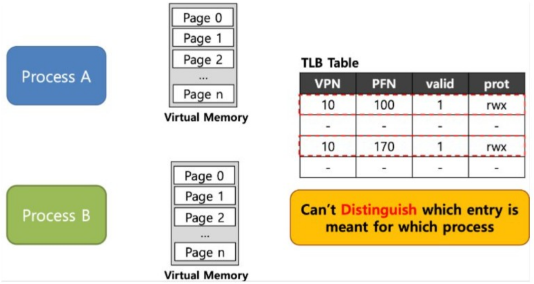
- On chip address translation cache stores recent virtual to physical memory translations.
- Translation read from page table and stored in TLB entries for following hit to improvement performance.
- Per-core cache. Every core has its own TLB.



TLB flush and shootdown

Flush local TLB entries

- Context switch - update CR3



- Page table updates – INVLPG
- Performance impact

- TLB Shootdown – actions of one core causing the TLB to be flushed on other cores
  - Send IPIs to remote cores for TLB shootdown requests.
  - Wait till all cores complete the flushes
  - More performance impact

TLB shootdown

- TLB flushing is a complex and costly operation in a large scale multiprocessor system.
- TLB flushing in a multiprocessor system uses IPIs to notify the other processors.
- On a native OS, the IPI delivery is handled completely in the CPU execution hardware.
- In virutal machine it involves multiple context switches and memory accesses, which increase the overhead and complexity of the TLB flushing.
- For a VM, the TLB flush IPI should be emulated by the hypervisor, which alone knows the vCPU to pCPU mapping that is needed for IPI delivery.
- TLB flush IPI also requires the IPI sender to wait until all receivers acknowledge the flush operation. If one of the IPI receiving vCPUs is delayed in being scheduled by the Hyper-V, the sender vCPU would have to wait longer until the TLB flush IPI is acknowledged.

Solution Idea : Hypercall

Hypercall – Interface for communication with the hypervisor - The hypercall interface accommodates access to the optimizations provided by the hypervisor.

To use Hyper-V hypercalls to offload the TLB invalidation synchronization between all target processors.

Keep the local TLB shootdown as is, and offload the remote TLB shootdowns to Hyper-V hypercalls.

Implementation : main requirement

- Hyper-V guest enlightments
- Send requests (hypercalls) to Hyper-V, let host flush TLB

Implementation : changes

- Refactoring existing remote TLB shootdown, which happens using smp_targeted_tlb_shootdown().
- Introduction of Hyper-V specific function hyperv_vm_tlb_flush().
- After Hyper-V is initialized, transfer the tlb shootdown from native to hyperv_vm_tlb_flush.
- Introduction of repetative Hyper-V hypercall mechanism for processor count more than 64.
- Creating new functions and interfaces to integrate the new hypercalls: HVCALL_FLUSH_VIRTUAL_ADDRESS_SPACE and HVCALL_FLUSH_VIRTUAL_ADDRESS_LIST.
- Based on different invl_op_codes, three different approach were taken for remote TLB shootdown.

Implementation

- Mainly in two commits

- Bec000c9c1ef409989685bb03ff0532907befb4a

  Refactor the existing tlb shootdown code.

```
smp_targeted_tlb_shootdown_t smp_targeted_tlb_shootdown = &smp_targeted_tlb_shootdown_native;
```

- 2b887687edc25bb4553f0d8a1183f454a85d413d

  Call Hyper-V tlb flush routine if guest is running on Hyper-V

```
smp_targeted_tlb_shootdown = &hyperv_vm_tlb_flush;
```

- Hypercall to tlb shootdown

```
        flush->flags |= HV_FLUSH_NON_GLOBAL_MAPPINGS_ONLY;
        status = hypercall_do_md(HVCALL_FLUSH_VIRTUAL_ADDRESS_SPACE,
            (uint64_t)flush, (uint64_t)NULL);
```

Result:Perf Numbers - IPI vs Hypercall

Cloud	CPU	VM SKU	Cost	TLB	Average (microseconds)
Azure	Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz (2793.44-MHz K8-class CPU)	E48ds v5 (48 vcpus, 384 GiB memory)	$3.45/hr	IPI	27
				hypercall	14.7

Perf Numbers - Intel vs AMD

Cloud	CPU	VM SKU	Cost	TLB	Average (microseconds)
Azure	Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz (2793.44-MHz K8-class CPU)	E48ds v5 (48 vcpus, 384 GiB memory)	$3.45/hr	IPI	27
				hypercall	14.7
Azure	AMD EPYC 7763 64-Core Processor (2550-MHz measured)	E48as v5 (48 vcpus, 384 GiB memory)	$2.71/hr	IPI	40.1
				hypercall	25

FreeBSD 15.0 guests (non-debug build), numbers taken from total FreeBSD kernel build with -j100 build option

Perf Numbers - 48 vs 16 vCPUs

Cloud	CPU	VM SKU	Cost	TLB	Average (microseconds)
Azure	Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz (2793.44-MHz K8-class CPU)	E48ds v5 (48 vcpus, 384 GiB memory)	$3.45/hr	IPI	27
				hypercall	14.7
Azure	Intel(R) Xeon(R) Platinum 8473C (2100-MHz measured)	E16s v5 (16 vcpus, 128 GiB memory)	$1/hr	IPI	12
				hypercall	7.7

FreeBSD 15.0 guests (non-debug build), numbers taken from total FreeBSD kernel build with -j100 build option.

Perf Numbers - Azure vs AWS

Cloud	CPU	VM SKU	Cost	TLB	Average (microseconds)
Azure	Intel(R) Xeon(R) Platinum 8370C CPU @ 2.80GHz (2793.44-MHz K8-class CPU)	E48ds v5 (48 vcpus, 384 GiB memory)	$3.45/hr	IPI	27
				hypercall	14.7
AWS	Intel(R) Xeon(R) Platinum 8275CL CPU @ 3.00GHz (3000.00-MHz K8-class CPU)	c5d.12xlarge (48 vcpus, 96 GB memory)	$2.3/hr	IPI	25.3

FreeBSD 15.0 guests (non-debug build), numbers taken from total FreeBSD kernel build with -j100 build option.

Challenges

- The performance are quite visibile when doing the micro level test, but not in macro level.
- A generic para-virtualization framework, to offload IPI’s to Hyper-V.

Ref

- https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/manage/manage-hyper-v-scheduler-types
- https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/tlfs/tlfs
- O. Kilic, S. Doddamani, A. Bhat, H. Bagdi and K. Gopalan, "Overcoming Virtualization Overheads for Large-vCPU Virtual Machines," 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, USA, 2018, pp. 369-380, doi: 10.1109/MASCOTS.2018.00042. keywords: {Virtual machine monitors;Virtualization;Program processors;Schedules;Scheduling;Emulation;Hardware;Virtualization;Virtual Machine;Virtual CPU;Scheduling},