GPhys (*G*uest *Phys*ical Memory Map)

Rohit Jain

Agenda

What is GPhys?

GPhys Functions

GPhys Interface

Implementation Details (bulk of the time spent here)

What is GPhys?

```
/*
 * GPhys -- Maintain a linear address space analogous to the guest physical
 * address space.
 */
```

Accessing memory at guest physical address pa with GPhys:

- `int *ptr = GPhys_PA2PtrTraced(pa); /* Very fast! */`
- `Log("Integer at address %lx is %d.\n", pa, *ptr);`

Without GPhys:

- `BPN bpn = PA_2_BPN(pa);`
- `MPN mpn = BusMem_TranslateBPN(bpn, …);`
- `int *ptr = KSEG_GetPtrFromMPN(mpn, …); /* Could be expensive */`
- `Log("Integer at address %lx is %d.\n", pa, *ptr);`
- `KSEG_ReleasePtr(ptr);`
- `BusMem_ReleaseUnpinnedMPN(&mpn);`

GPhys Functions

1. **Facility to access memory at a given guest physical address.**
   - Fast compared to KSEG (due to more dedicated address space)
   - Easy to perform an access with or without respecting monitor traces.
2. **Serves as the page tables for nested page table (NPT) and extended page table (EPT) hardware.**
3. **Serves as a cache of MPNs for guest memory in the monitor.**
4. **Provides capability for observing guest memory access behavior.**
   - Tracking accessed and dirty pages (for fast checkpointing, vmotion and possibly HEAT based optimizations)
   - Tracking hotly accessed 2M regions (for upgrading to large pages)

GPhys Interface

1. Guest memory access.

- GPhys_PA2PtrTraced
- GPhys_PPN2PtrNoTrace
- WITH_MAPPED_[PINNED_]PPN_NOTRACE_DO
- GPhys_GuestPgWlk / GPhys_GetPDPTE
- GPhys_TryReadWritePhys

2. Validation

- GPhys_Validate
- GPhys_HandleFault (implicit validation through access)
- GPhys_ValidateWithMPN
- GPhys_TryValidateBPN (eager validation)
  - When removing a trace or when a 2M region is no longer sampled.

GPhys Interface

3. Invalidation

- GPhys_InvalidateMappings
- GPhys_InvalidateMappingRange
- GPhys_InvalidateBPNToLargeFlushReq
- GPhys_UpdateTraces
- GPhys_(Open|Process)[Large]FlushReq

4. Numa Migration

- GPhys_NumaMigrate
- GPhys_NumaMigrateOnVCPU
- GPhys_IncNumaMigrate

GPhys Interface

5. Large page scanning

- GPhys_WSCalc
- GPhysTreeWalk (called from GPhys_1HzCall)

6. Cache of guest memory MPNs

- GPhys_LookupMPN
- GPhys_ValidateWithMPN

Multiple page table trees, all mapping guest memory

- For the VMM accessing guest memory
  - Traced tree (GPHYS_TREE_TRACED): Honors traces
  - Notrace tree (GPHYS_TREE_NOTRACE): Provides guest memory access without trace protection

- For EPT and NPT
  - The tree that hardware NPT/EPT pointer points to (GPHYS_TREE_HWMMU)
  - The guest memory accesses in HV mode go through this tree.
  - Honors traces.

Quotas and Sharing

- Each tree gets an overhead memory quota assigned to it at initialization.
- The HWMMU tree gets 100% quota by default because the guest is expected to work through all of its memory.
- The TRACED and NOTRACE trees get quotas that are determined by various factors:
  - HWMMU or SWMMU?
  - Nested VM or non-nested VM?
  - EPT (Intel) or NPT (AMD)?

Quotas and Sharing (cont.)

- Traced Tree
  - This tree is completely shared with the HWMMU tree when running a non-nested VM on AMD with NPT.
  - Cannot share with the EPT tree because the formats are different.
  - Cannot share with the NPT tree when running a nested VM because the NPT tree is traced, causing NPFs on every access in the monitor.
  - Quotas:
    - When shared with the HWMMU tree, 0%.
    - When not shared with the HWMMU tree, 20%.
    - When running with SWMMU, 100%.

Quotas and Sharing (cont.)

- Notrace Tree
  - Opportunistically shared with the traced tree at the L1 level.
    - An avail bit in the L2Es is used to mark the L1s that are shared.
  - Sharing possible only when the 2M region covered by the L1 table is not traced.
    - Sharing is broken when a trace is installed in the 2M region.
  - Quotas:
    - 5% when using HWMMU.
    - 100% when using SWMMU.

Overall Overhead Memory Consumption

- Let N be the overhead of a page-table tree mapping entire guest physical memory.
- AMD HWMMU, non nested: 1.05 * N
- AMD HWMMU, nested VM: 1.25 * N
- Intel HWMMU: 1.25 * N + a small EPTMap overhead
- Intel/AMD SWMMU: 2 * N

- GPhys trees are shared among all VCPUs, so the overhead memory consumption is independent of the number of VCPUs.

Evictions (when the quota fills up)

- GPhysEvictPT called
- Maintains a running L2E pointer.
- Scans through L2Es, clearing A bit until:
  1. An L2E is found with PS bit not set and A bit already cleared.
  2. A threshold number (100) of L2Es are scanned.
- The L2E is invalidated and the corresponding L1 page table is freed.
- If no valid L2Es remain in the L2 table, the L2 table is also freed.
- Since a global TLB flush is required, evictions are batched (up to 10 L1 tables at a time).

Mapping the gphys trees in the monitor

- Trees in x86 format
  - Utilize the ScratchAS recursive mapping framework
    - ScratchAS_AttachL4E
    - ScratchAS_GetPTEPtr
    - ScratchAS_PgWalk
    - ScratchAS_GetLastPTE

- Intel HWMMU (EPT) tree
  - Mapped using EPTMap
    - GPhysMapEPT/GPhysUnmapEPT/GPhysRemapEPT
    - GPhysEPTPgWalk
    - GPhysGetLastEPTE

The ScratchAS recursive mapping framework

- One of the L4Es in the scratchAS points back to the L4 table.
- The recursion effectively maps PTEs at all levels in all page-table trees mapped in the scratchAS.
  - Use ScratchAS_GetPTEPtr(l4off, level, offset)
  - The function is a simple arithmetic manipulation of the three arguments, so it is very fast.
  - For ex, Accessing the l2e entry mapping guest physical address “pa” in the gphys traced tree:

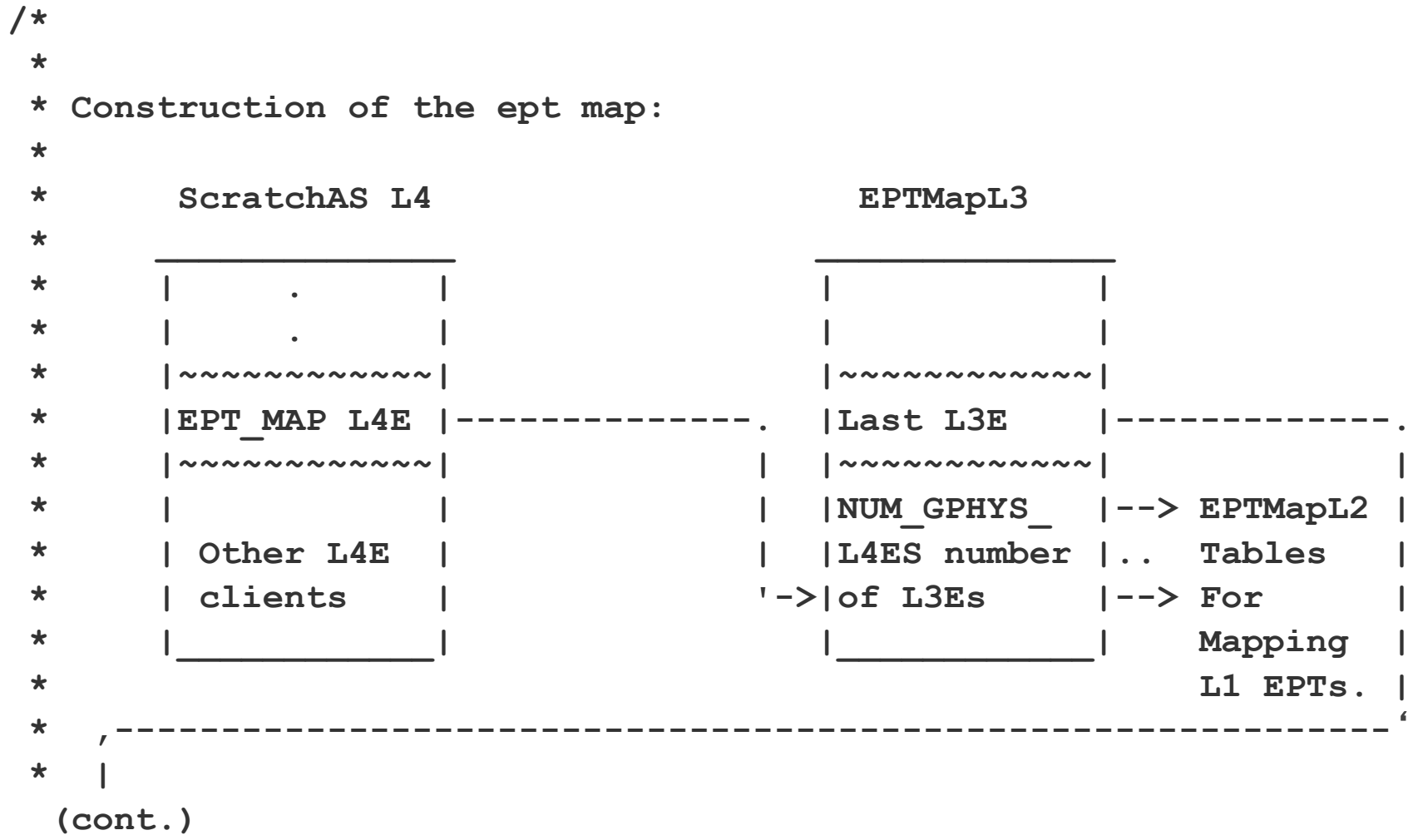
```
VM_L2E *l2e =
  ScratchAS_GetPTEPtr(gPhysTracedL4Off, PT_LEVEL_2, pa);
```

- GPhysGetPTEPtr uses ScratchAS_GetPTEPtr for accessing trees in x86 format (all trees except Intel HWMMU tree).

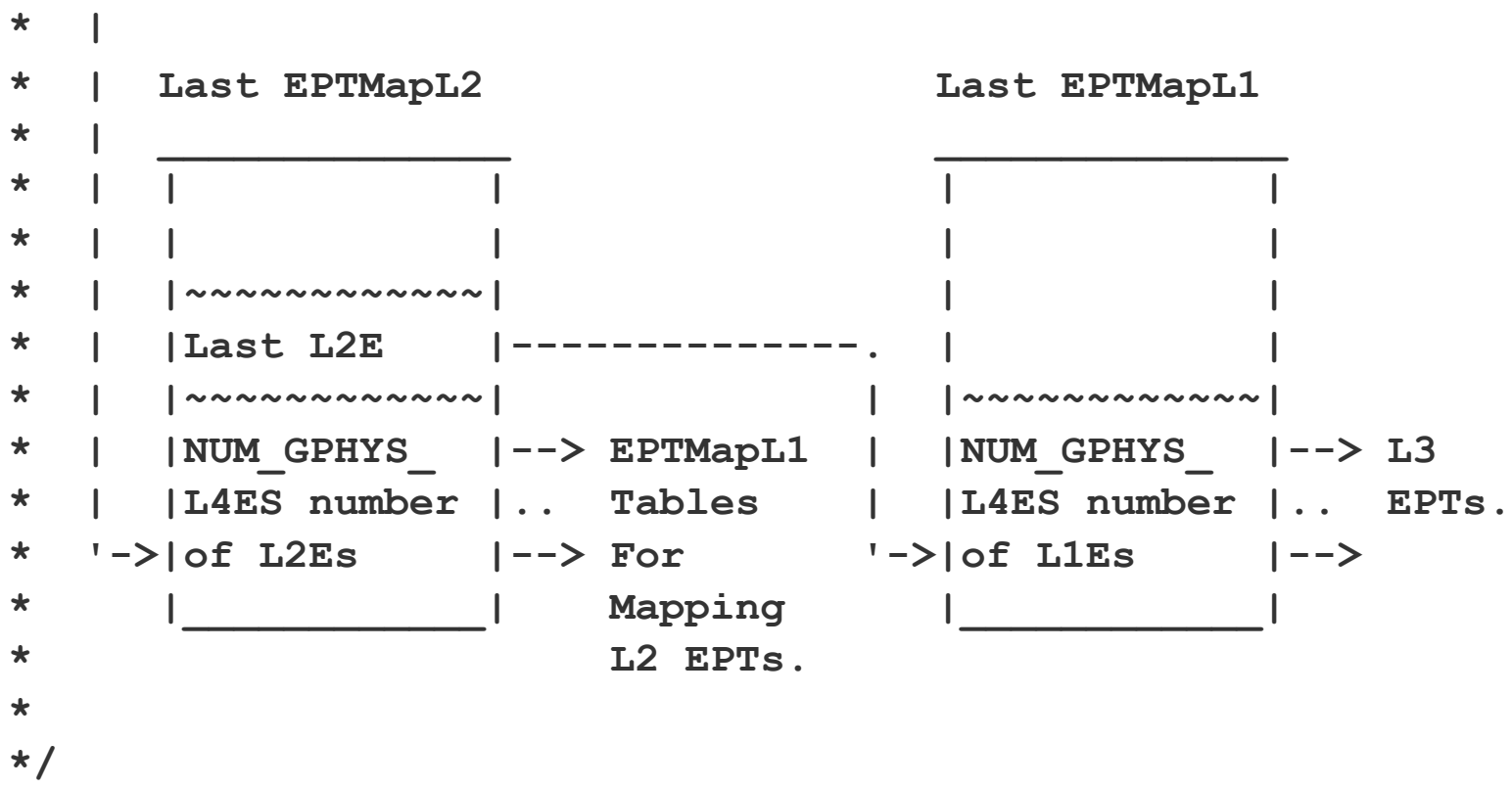
The EPTMap framework

- Explicitly creates the mappings which are created implicitly by the recursive framework.
- Recursive mappings map all L4Es followed by all L3Es followed by all L2Es and finally all L1Es.
- EPTMap maps in the reverse order – all L1Es followed by all L2Es followed by all L3Es.
- Currently, the EPTMap uses its own L4E, but it uses only NUM_GPHYS_L4ES (4) L3Es in that L3 table.
- Accesses through EPTMap should be as fast as accesses through the recursive mappings for x86 style page tables.
  - No need to use KSEG to map EPT tables.

The EPTMap framework



Implementation Details (cont)



The EPTMap framework (cont.)

- The mappings effectively create an array of EPT PTEs (called EPTMap), starting with EPT L1Es, followed by EPT L2Es, followed by EPT L3Es.
- gPhysEPTMapLayout[level] provides an index into this array where EPT PTEs for “level” start.
- GPhysEPTMapIdx(level, pa) provides an index into the array for the EPT PTE at level “level” mapping physical address “pa”.
- GPhysGetPTEPtr for the EPT tree becomes:

```
GPhysPTE *pte = (GPhysPTE *)SCRATCHAS_EPT_MAP_BASELA +
                            GPhysEPTMapIdx(level, pa);
```

- The page tables which create the EPT map are called EPTMapPTs. When talking about a specific level, they are referred to as EPTMapLx, where x is the level.

Validation

- All GPhys trees start out empty.
- Validation happens only when:
  - A GPhys pointer is accessed causing a page fault (through GPhys_HandleFault).
    - GPhys_PA2PtrTraced, etc. do not validate the tree!
  - The guest accesses memory in HWMMU mode and the corresponding GPhys PTE is not already validated.
    - The monitor gets a nested page fault (#NPF)
    - HV exit code calls GPhys_Validate.
  - The monitor removes a trace on a page causing a GPhys PTE to be eagerly validated with new permissions. (GPhys_UpdateTraces)
  - The monitor “unsamples” a page causing a 2M region to be eligible for upgrade to a large page.
  - BusMem validation eagerly validates a GPhys PTE in the HWMMU mode to use it as a cache of MPNs. (GPhys_ValidateWithMPN)

Validation flow

- We are given a guest physical address that needs to be validated.
- PA_2_BPN gives us the “bus page number” (BPN).
- BusMem_TranslateBusFrame gives us the MPN.
  - This function is first called without a lock (to avoid holding a lock across a potential Platform_LockGuestPage call).
  - The busmem lock is then grabbed using BusMemTryLock(). If that fails, the lock is acquired and BusMem_TranslateBusFrame is called again.
- The rest of the validation happens under the protection of the busmem lock (in GPhysValidateWork).
- Determine if we can map the page in GPhys:
  - GPhys is not used in SMM mode.
  - The APIC pages cannot be mapped in GPhys if they are mapped at different physical addresses in different VCPUs.
    - “gPhysOnAndSingleGlobalAPICPPN” tracks this.

Validation flow (cont.)

- For each GPhys tree
  - Determine if the tree needs to be validated.
    - When using software MMU, only validate the tree being accessed (traced or notrace).
    - When running on Intel with hardware MMU, we skip validating the traced and notrace trees for guest accesses in HV mode.
  - Determine if the access is “too traced”
    - For guest accesses in HV mode or monitor’s traced tree accesses, bail out if the trace permissions do not allow the access.
    - This avoids an infinite page-fault loop.
  - GPhysPageWalk() returns a pointer to the last level of the tree which needs to be validated, allocating page tables along the way if needed.
    - Evicts page tables (GPhysEvictPT) if the quota is exceeded.
  - The last level PTE is populated with the given MPN and permissions.

Invalidation

- Happens when:
  - Physmem mappings change (e.g. A20 flip)
    - GPhys_InvalidateMappings and GPhys_InvalidateMappingRange.
  - Traces are installed (GPhys_UpdateTraces).
  - A page is zapped in busmem (GPhys_InvalidateBPNToLargeFlushReq).
  - Evictions

Invalidation Flow

- GPhysProtect is the single point of entry for all invalidations except evictions and GPhys_InvalidateMappings.
- Uses a reverse physmem lookup to get the physical address(es) from the given BPN (bus address).
  - We implement a cache of reverse lookup for the mainmem region.
- Breaks sharing between traced and notrace trees when installing a trace.
- For each tree, obtains the PTE pointer using GPhysGetLastPTE and downgrades the permissions on that PTE as needed.
- Adds the physical address to the “flush” array, which tracks the physical address mappings that need to be flushed from the TLB.
- GPhys_InvalidateMappings and protecting the buserror BPN require the big hammer – GPhysFreeAll.

GPhysFreeAll

- Throws away all GPhys page tables and starts from scratch.
- Used when there are too many pages to be invalidated.
  - Primarily for handling the buserror BPN, which can map to a large number of physical addresses.
- Tricky because the guest is running in parallel on other VCPUs!
- Flow:
  1. Create new gphys skeleton trees (including EPTMap).
     - This may require evicting some page tables to make space.
  2. Crosscall all VCPUs to flush their TLBs and attach the new page tables in the scratchAS.
  3. Free the old page table trees.

Numa Migration

- Remaps all gphys overhead memory to the new NUMA node when the VMM is NUMA-migrated.
- GPhys_NumaMigrate
  - Remaps all L3 page tables to the new node (GPhysNumaMigrateL3)
  - Remaps the EPTMap page tables when running with Intel+HWMMU.
  - Does not remap the rest of the gphys page-tables.
    - May be too big to migrate in one shot (think of monster VMs!)
    - Sets up state for incremental numa migration of these page tables.
- GPhys_NumaMigrateOnVCPU
  - Maps the new L3 tables in each VCPU.
  - Sets up the per-vcpu incremental numa-migration state.

Incremental Numa Migration

- Happens every second through GPhys_IncNumaMigrate.
- gPhysNumaMigrate.l3Off maintains a shared, running L3E pointer.
- gPhysNumaMigrate.pendingL3Es maintains the remaining L2 trees to be migrated.
- Each VCPU grabs work by atomically fetching and incrementing gPhysNumaMigrate.l3Off.
  - Calls GPhysNumaMigrateL1 for each L2E in the table, and finally GPhysNumaMigrateL2 for the L2 table itself.
- Currently, 800k cycles per second allocated to incrementally numa migrate both busmem frames and gphys page tables.

GPhys as a cache of MPNs

- The HWMMU tree has a 100% quota by default.
  - Few evictions, if any.
  - Invalidations only on rare occasions
    - Zaps, physmem remaps, read-traces.
  - PTEs contain MPNs!
- When using HWMMU, we do not store the MPN in the busmem frames, saving 4 bytes per page.
  - BusMem asks GPhys for the MPN when needed using GPhys_LookupMPN.
    - If the MPN is not found in GPhys, Platform_BPN2MPN is called to retrieve the MPN from the platform.
    - The platform call is cheap on hosted because the pframes are mapped in the monitor.
  - BusMem eagerly validates GPhys during its translate routine to make the caching more effective. It calls GPhys_ValidateWithMPN.

GPhys as a cache of MPNs (cont.)

- GPhys_LookupMPN flow
  - Works only for main memory pages
    - Reverse physmem lookup cache only works for main memory.
    - Device pages are often too traced.
  - Performs a reverse physmem lookup (BPN->PPN).
  - Performs a lock-free walk of the HWMMU tree.
    - Can get a page-fault if a page-table is invalidated during the walk.
    - On a #PF, deals with it by returning failure.
- GPhys_ValidateWithMPN flow
  - Same as GPhys_Validate, except, it skips the BusMem_TranslateBusFrame step and goes directly to GPhysValidateWork.
  - Noticed a problem (validating the wrong tree when using EPT!)

Observing guest memory accesses

- When using HWMMU, all guest memory accesses in HV mode go through the HWMMU tree.
- Can use accessed and dirty bits in the HWMMU tree (AMD only for now!) to observe guest memory accesses and mutations.
  - GPhys_WSCalc clears A-bits on PTEs and counts the number of set A bits per 2M region the next second to identify candidates for large pages.
    - The clearing and counting pattern repeats every two seconds through GPhys_1HzCall.
    - Talk to Ravi!
  - SMP FT clears A and D bits on PTEs, uses A bits to hierarchically scan the page-tables during a checkpoint and uses the D bits to identify memory pages with mutations.
    - Talk to Wei!

Reading guest memory “safely”

- Some clients want to read guest memory without invoking GPhys validation. e.g. GPhys_PageWalk(), PhysMemReadWritePhysical().
- GPhys_TryReadWritePhys() provides this interface.
  - Performs guest memory access through a GPhys pointer.
  - On a fault, sets the value to zero and returns failure.
  - GPhys_PageWalk bails naturally because a value of zero implies a not-present page-fault.
  - PhysMemReadWritePhysical retries with KSEG if this routine fails.

That's it!

Questions?