[MemOS/Challenges: 1st week] Introduction & Overview

2021. 8. 11

Baik Song An

ETRI

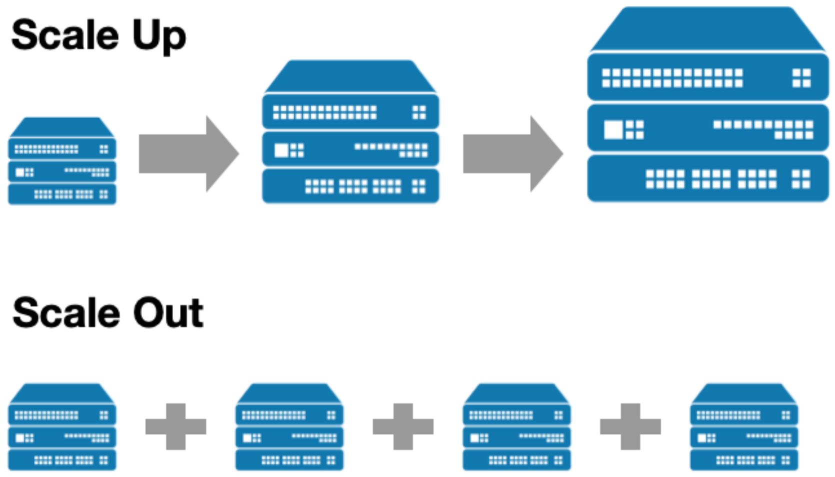
Contents

- GiantVM overview
- What to run on GiantVM? (NUMA-aware programming)
- How to fix GiantVM (and others)? (On-going issues)

GiantVM Overview

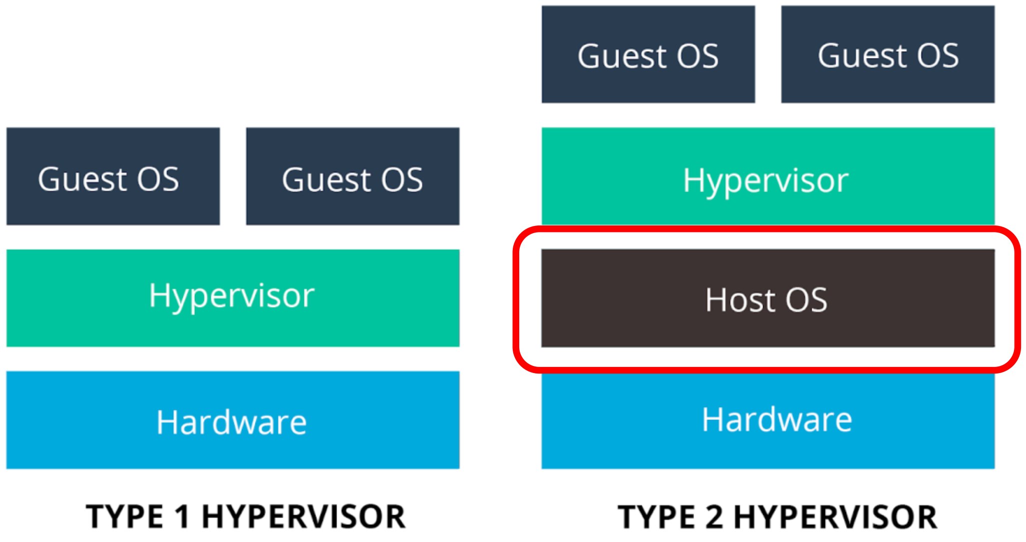
Motivation

- The era of 'big data': explosive growth of data to be processed
  - Paradigm shift: scale-up → scale-out
  - Mapreduce, Spark, …
- The limit of scale-out model
  - Complicated programming model
  - Runtime overheads
- Dilemma ☹
  - Scale-out: complicated programming
  - Scale-up: huge HW cost



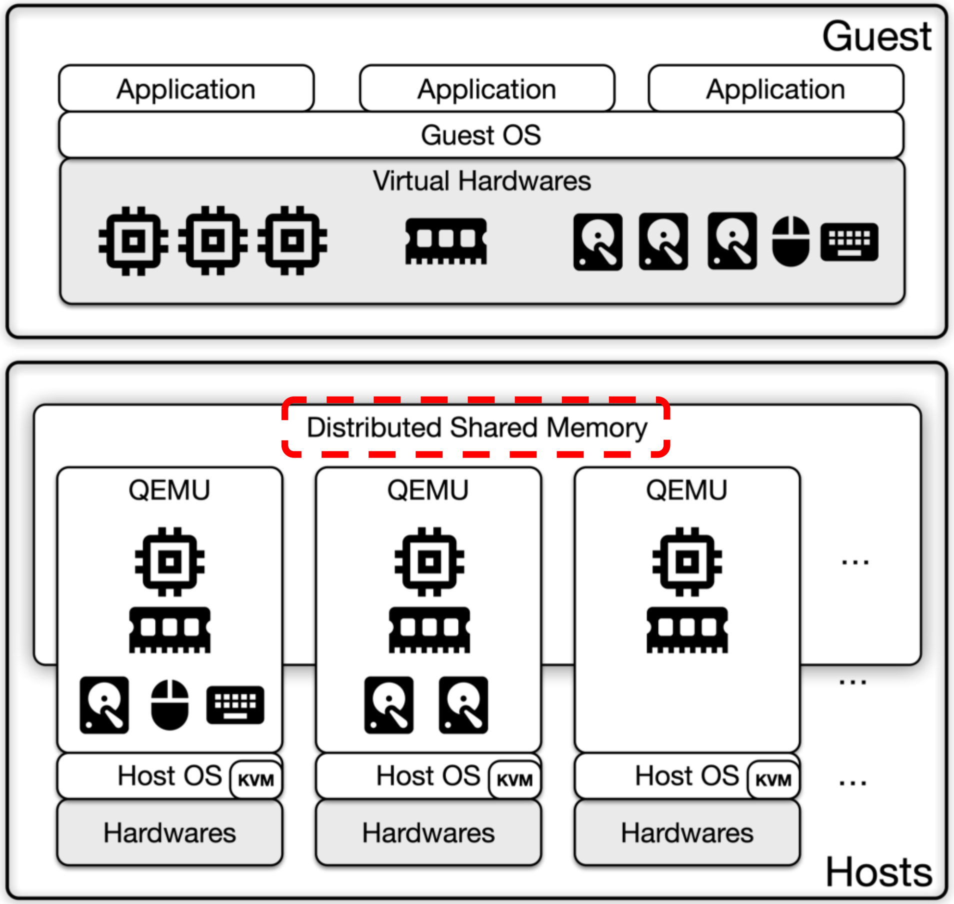
Motivation (Cont'd)

- Single System Image (SSI)
  - Running single OS on a cluster
  - Huge engineering costs for compatibility (POSIX, etc.)
  - Porting close-source device drivers is not possible
- **Many-to-one (multi-node) virtualization**
  - Combination of virtualization & SSI
  - Efficient resource management through HW abstraction
  - (Full virtualization) No need for OS modification
- Open-source, type-2 hypervisor
  - Based on QEMU/KVM
  - Using the existing OS as a hypervisor, with no need for scratch building



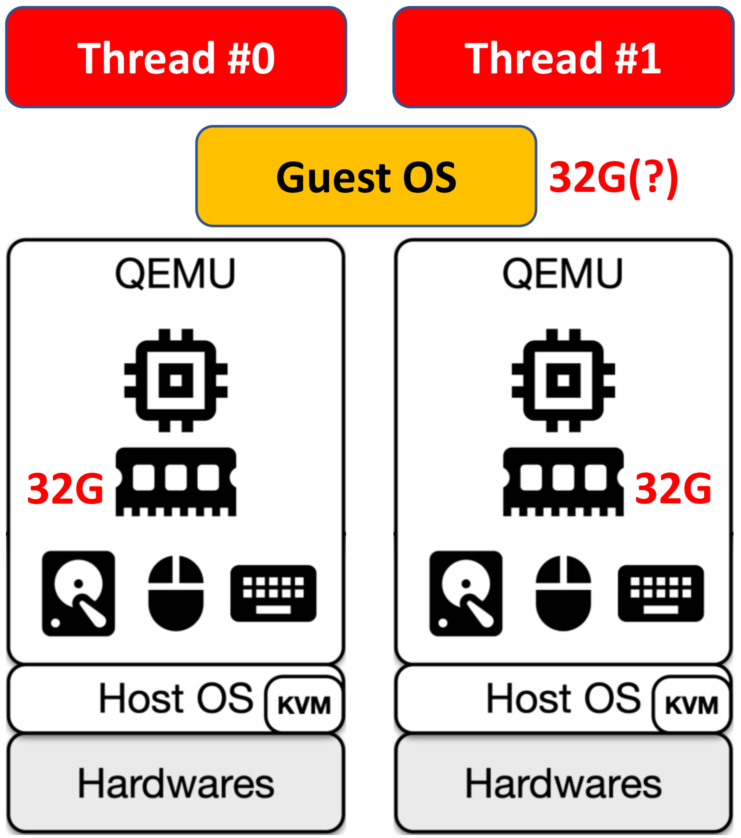
GiantVM: Overview

- Open-source distributed hypervisor
  - “GiantVM: a type-II hypervisor implementing many-to-one virtualization”, VEE ‘20
- Many-to-one virtualization
  - Similar to ScaleMP/TidalScale (commercial)
- (Tentative) **base platform for MemOS**
- Type-2 hypervisor (QEMU/KVM)
  - QEMU: Remote vCPU, I/O
  - KVM: Distributed Shared Memory (DSM)

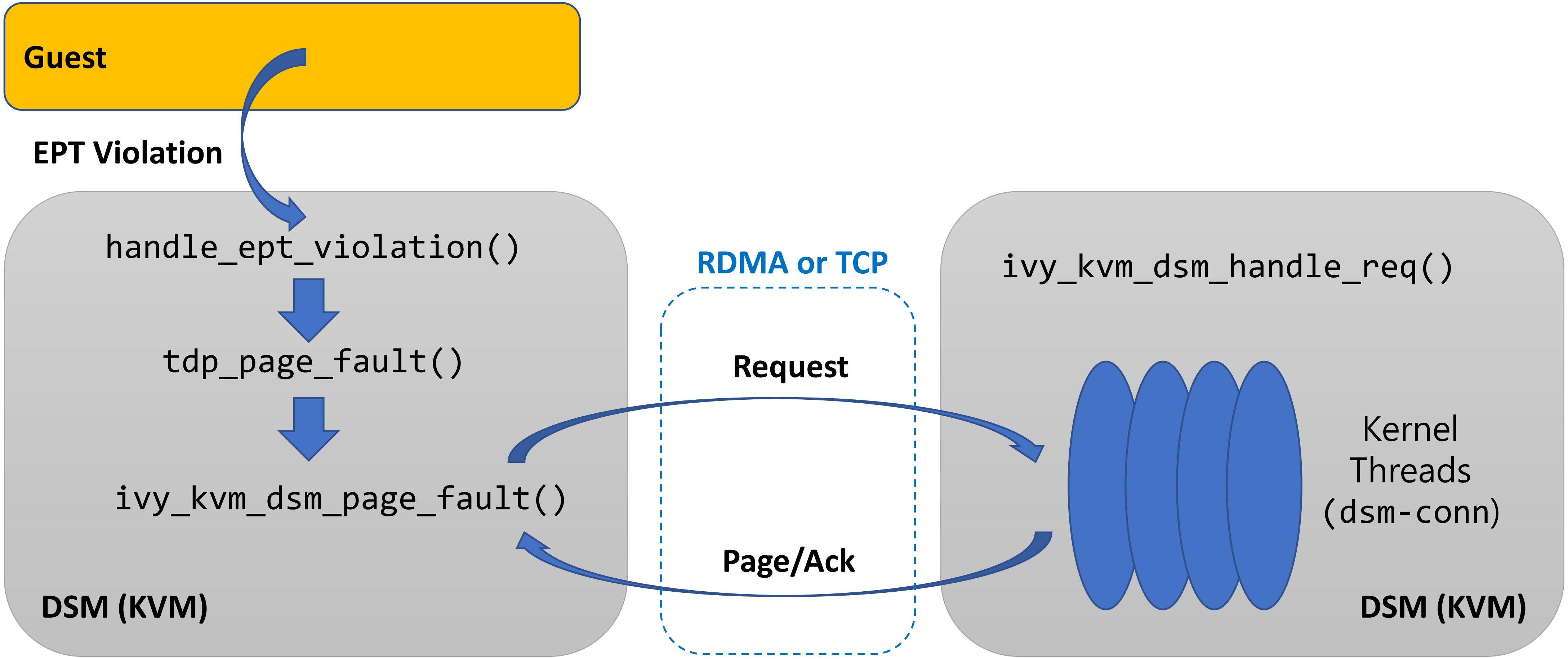


DSM in GiantVM

- Implemented in KVM
- Based on Ivy DSM
  - “Memory Coherence in Shared Virtual Memory Systems” (PODC ’86)
- Memory size seen by the Guest OS
  - The memory size exported by **one local QEMU instance**
  - With DSM, each local QEMU instance sees its own memory & coordinates with each other



Basic Mechanism of DSM



Page Fault Handling

- Sequences

```
tdp_page_fault()
  kvm_dsm_vcpu_acquire_page()
    __kvm_dsm_acquire_page()
      kvm_dsm_page_fault()
        ivy_kvm_dsm_page_fault()
```

ivy_kvm_dsm_page_fault()

```
ivy_kvm_dsm_page_fault()
{
   if (write) {
      if (dsm_is_owner) {         /* self */
         kvm_dsm_invalidate();
      }
      else {
         owner = dsm_get_prob_owner();
         if (initialized && dsm_id == 0) {
            dsm_set_prob_owner(dsm_id);
            dsm_change_state(OWNER | MODIFIED);
            dsm_add_to_copyset(dsm_id);
            goto out;
         }
         kvm_dsm_fetch(owner, &req, &resp);
         kvm_dsm_invalidate(&resp.inv_copyset, owner);
      }
      dsm_clear_copyset();
      dsm_add_to_copyset(dsm_id);
      if (!dsm_is_owner)
         __kvm_write_guest_page();
      dsm_set_prob_owner(dsm_id);
      dsm_change_state(OWNER_MODIFIED);
   }
```

```
   else {  /* read */
      owner = dsm_get_prob_owner();
      if (initialized && dsm_id == 0) {
         dsm_set_prob_owner(dsm_id);
         dsm_change_state(OWNER | SHARED);
         dsm_add_to_copyset(dsm_id);
         goto out;
      }
      kvm_dsm_fetch(owner, &req, &resp);
      memcpy(dsm_get_copyset(), &resp.inv_copyset);
      dsm_add_to_copyset(dsm_id);
      __kvm_write_guest_page();
      dsm_set_prob_owner(dsm_id);          /* different
from orig. Ivy */
      dsm_change_state(OWNER | SHARED);
   }

out:
   …
   return;
}
```

Request Handling

- Sequences

`kvm_dsm_init()`

Kernel Thread: `kvm-dsm/%d`

`kvm_dsm_threadfn()`

Kernel Thread: `dsm-conn/%d:%d`

`kvm_dsm_handle_req()`

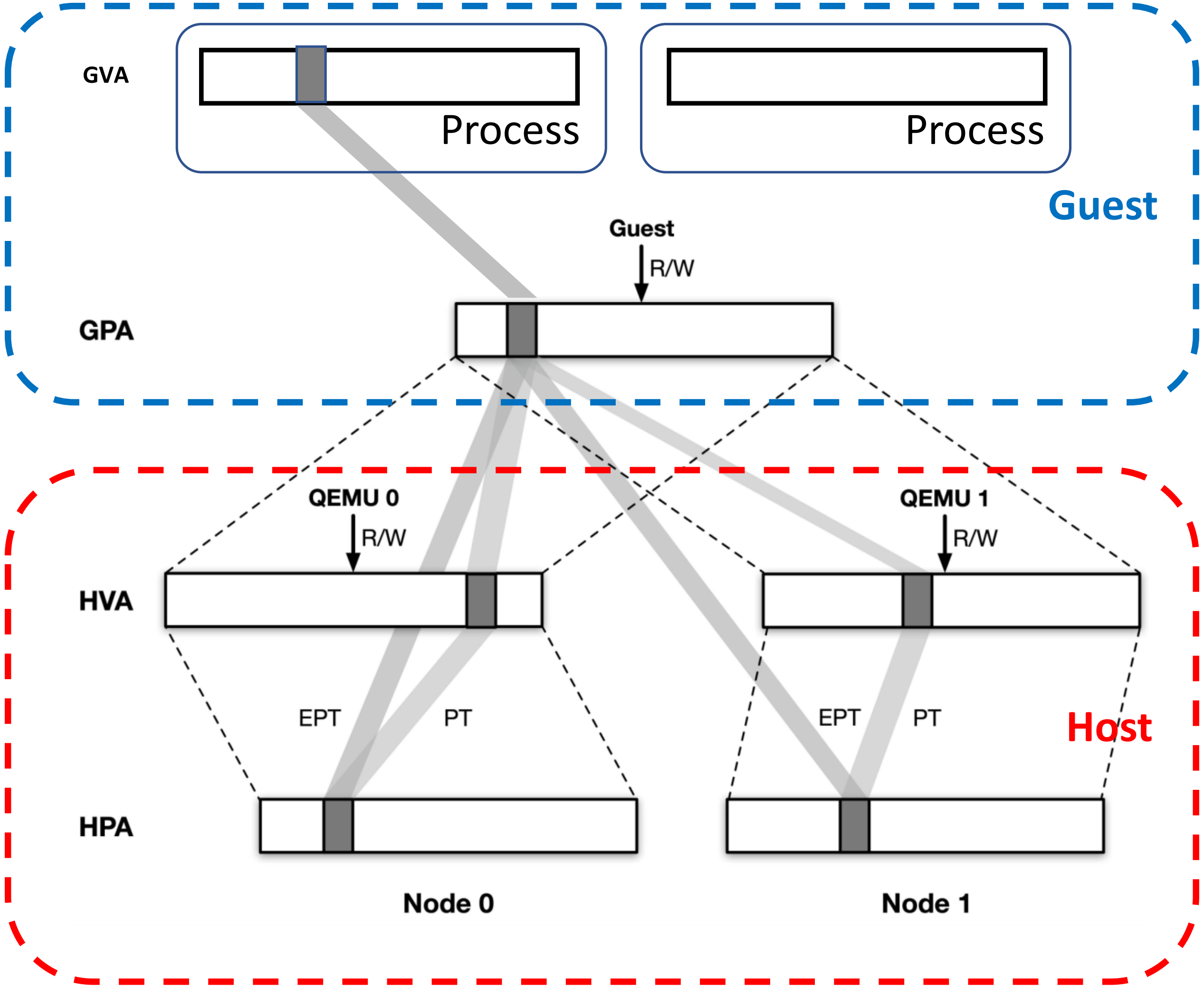
`ivy_kvm_dsm_handle_req()`

ivy_kvm_dsm_handle_req()

```
ivy_kvm_dsm_handle_req()
{
   while (1) {
      len = network.ops.receive(conn_sock, &req);  /* receive a request message */
      switch (req.req_type) {
      case DSM_REQ_INVALIDATE:
         ret = dsm_handle_invalidate_req();
         break;
      case DSM_REQ_WRITE:
         ret = dsm_handle_write_req();
         break;
      case DSM_REQ_READ:
         ret = dsm_handle_read_req();
         break;
   }
   …
   return;
}
```

Address Spaces in GiantVM

- gva
  - Guest **virtual** address
  - Assigned per process in GVM
- gfn
  - Guest **frame** number (GPA)
  - System-wide unique
- vfn
  - **Virtual** frame number (HVA)
  - Node-wide unique
- hfn
  - **HW** frame number (HPA)



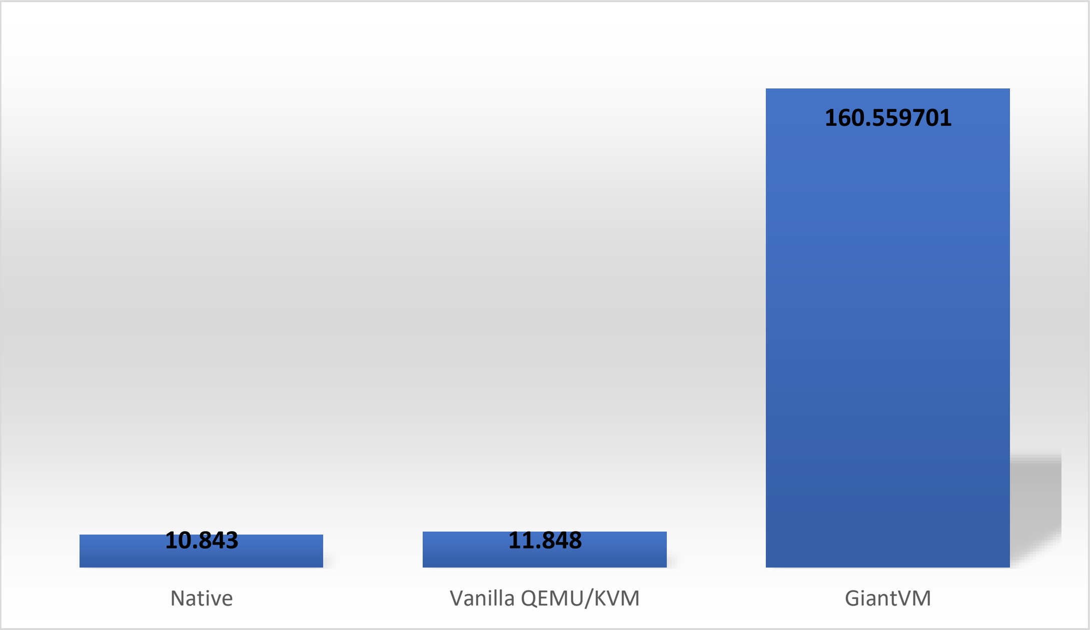
Performance Issues in GiantVM

- Serious performance degradation with multi-thread parallel workloads ☹
  - Especially when multiple threads concurrently access shared memory
- Reason for overheads?
  - Guest kernel
  - Host kernel (DSM in KVM)

Components	Description	Samples
Router	Interrupt & I/O router	1.06%
QEMU	QEMU except Router	0.97%
GuestOS	Guest OS (non-root mode)	35.44%
DSM	DSM page fault handler	43.75%
Others	Kernel part except DSM	18.89%

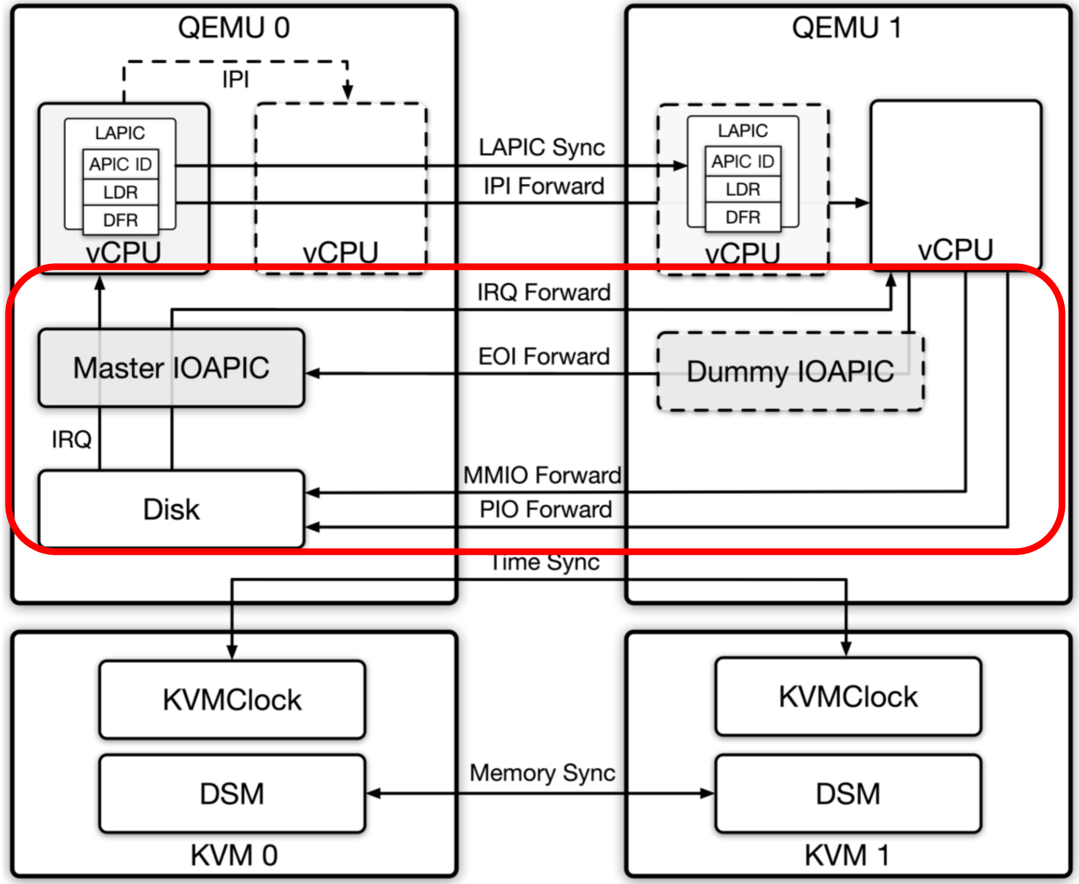
(Source: GiantVM, VEE '20)

- 24 threads, memcpy() 4GB per thread
- Exec. Time (lower is better)



I/O in GiantVM

- Only the I/O devices in a master QEMU(QEMU0) are visible
  - All PIO/MMIO are accessed through the master
  - Causes bottleneck

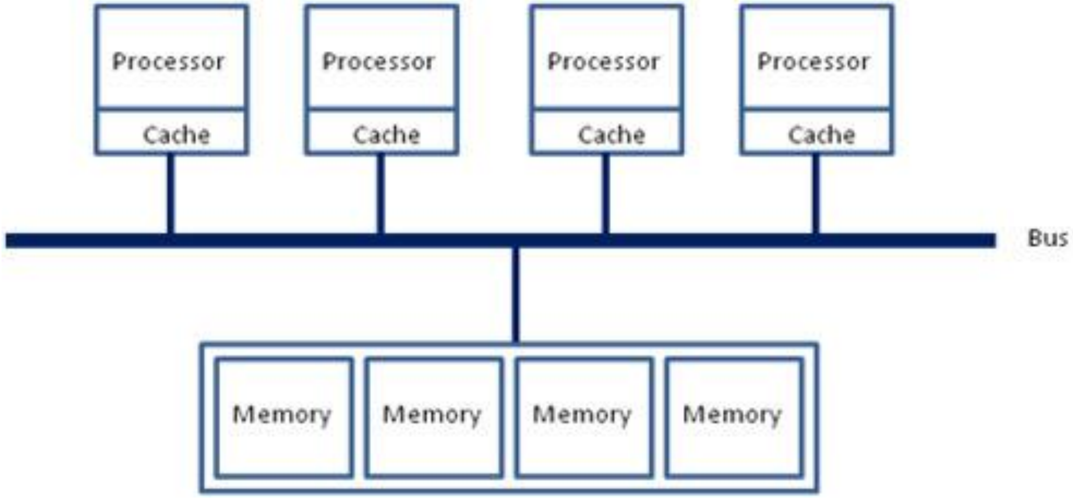


What to run on GiantVM?

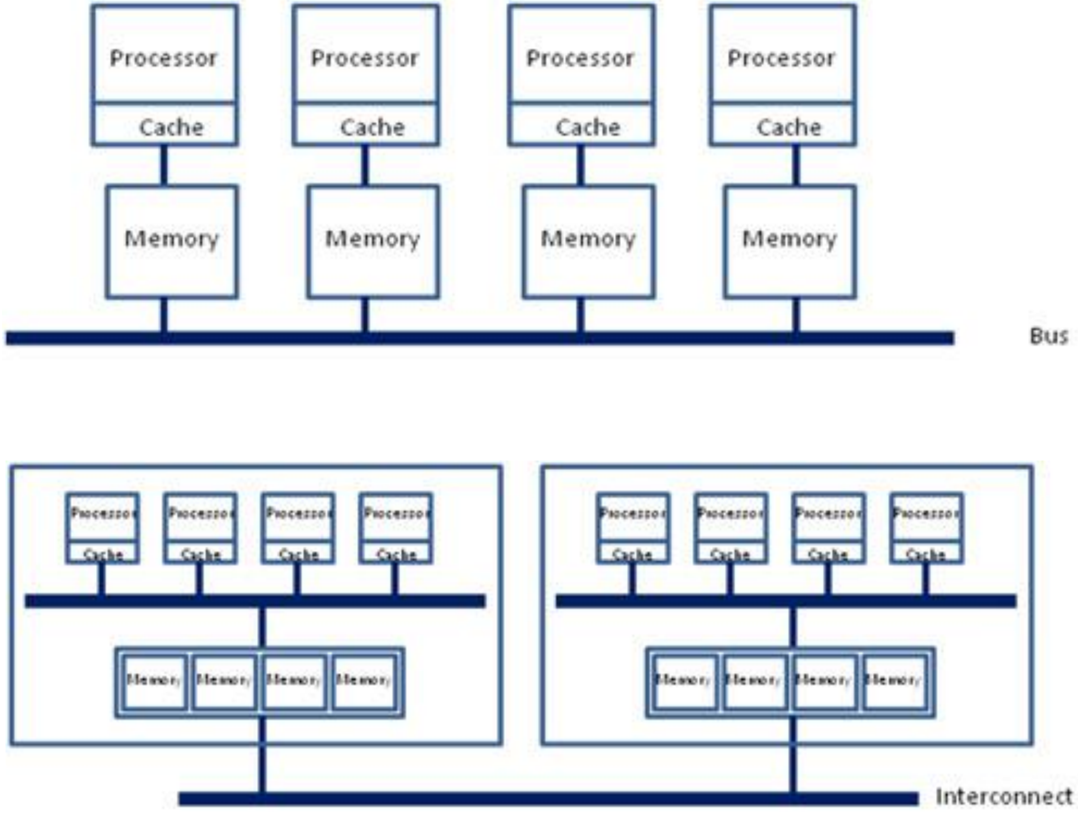
NUMA-aware programming

NUMA (Non-Uniform Memory Access)

- UMA



- NUMA



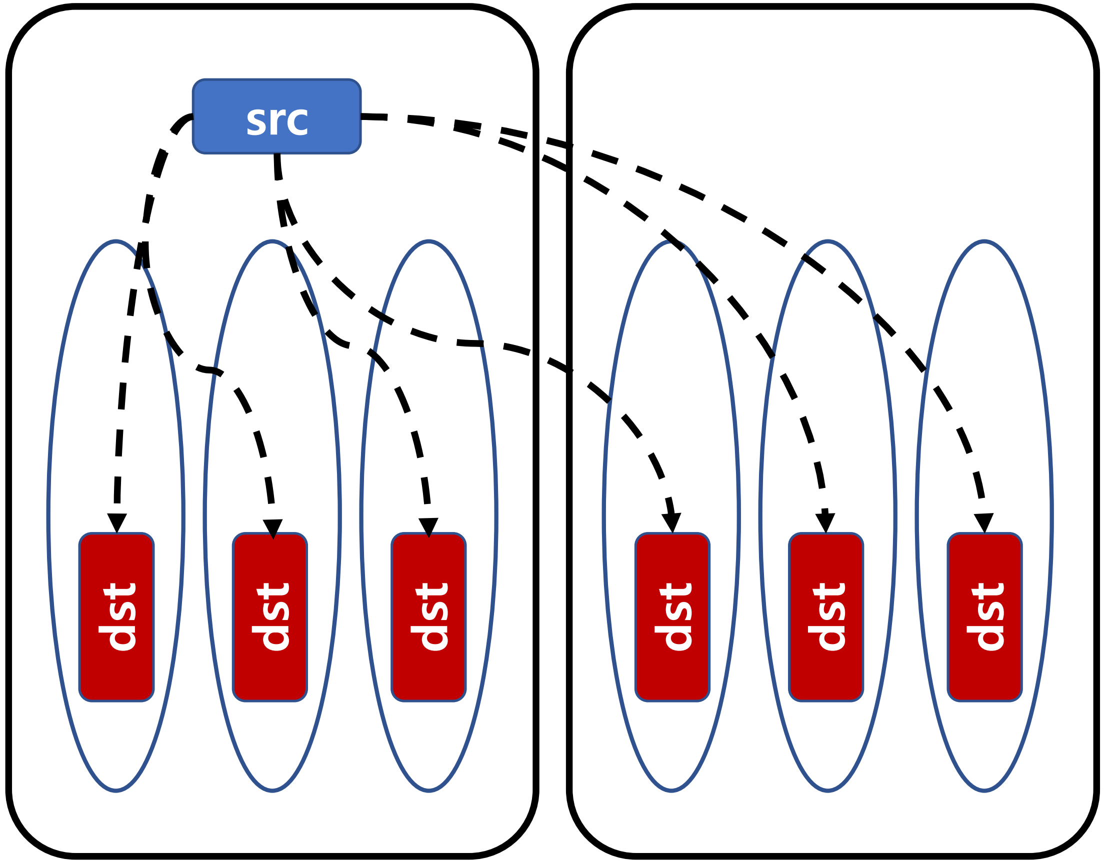
NUMA-aware Programming

1. Processor Affinity
   - Thread migration to another NUMA node
     - Need to fetch the data from the previous node (overheads)
   - Pinning a thread to a specific core, or limiting the migration to intra-node cores

2. Data Placement with Explicit Memory Allocation Directives
   - Example) libnuma library in Linux
   - Users explicitly designate a NUMA node for memory allocation
     - numa_alloc_onnode()

Example: memcpy() microbench

- In-house multithread memcpy() microbenchmark
- Per-thread src -> dst memcpy
- One source in primary node is shared by all worker threads
- All src/dst memory are allocated with numa_alloc_onnode()
- Each thread is pinned to its corresponding vCPU



How to fix GiantVM (and others)?

On-going issues

On-going Issues with GiantVM

- TCP support
  - The default inter-node networking mechanism is RDMA
  - TCP is supported, but unstable in both performance & stability
- Guest memory size
  - 1 + 1 = 1, 1 + 1 + 1 = 1, 1 + 1 + 1 + 1 = 1, …
  - Requires a major overhaul of the current DSM
- I/O bottleneck
  - Previously stated
- AMD CPU support
  - Currently, only Intel CPUs are supported

On-going Issues with GiantVM (Cont'd)

- DSM performance
  - Remember, the InfiniBand is slower than UPI. Deal with it.
  - But, it's tooooooo slow! ☹

1. Page prefetching
   - Fetching multiple pages at a time from a remote node
   - Would be beneficial, at least for sequential accesss

2. Asynchronous page fault
   - Currently, all DSM page faults are handled synchronously
   - Asynchronous handling allows vCPUs to do other tasks instead of blocking

On-going Issues with GiantVM (Cont'd)

3. Guest page cache optimization
   - A major cause of filesystem overheads
   - Page caches are shared by multiple nodes through the slow DSM

4. What else?

On-going Issues with Workloads

- Workloads tested so far
  - In-house memcpy() microbench (previously stated)
  - Some HPC workloads (based on floating-point matrix multiplication)
  - Parallel benchmark (PARSEC, NPB)
  - Manycore OS benchmark (MOSBENCH, vbench)
- Target workload
  - (Tentatively) HPC applications (SpMV, etc.)
  - Not decided yet
- Once the target workload is decided, some optimization should be done
  - Adding NUMA-awareness, etc.

Thank you!