SPEED UP CREATION OF A VM WITH PASSTHROUGH GPU

LIANG LI, DIDI CHUXING

OCT 2020

滴滴

滴滴云

AGENDA

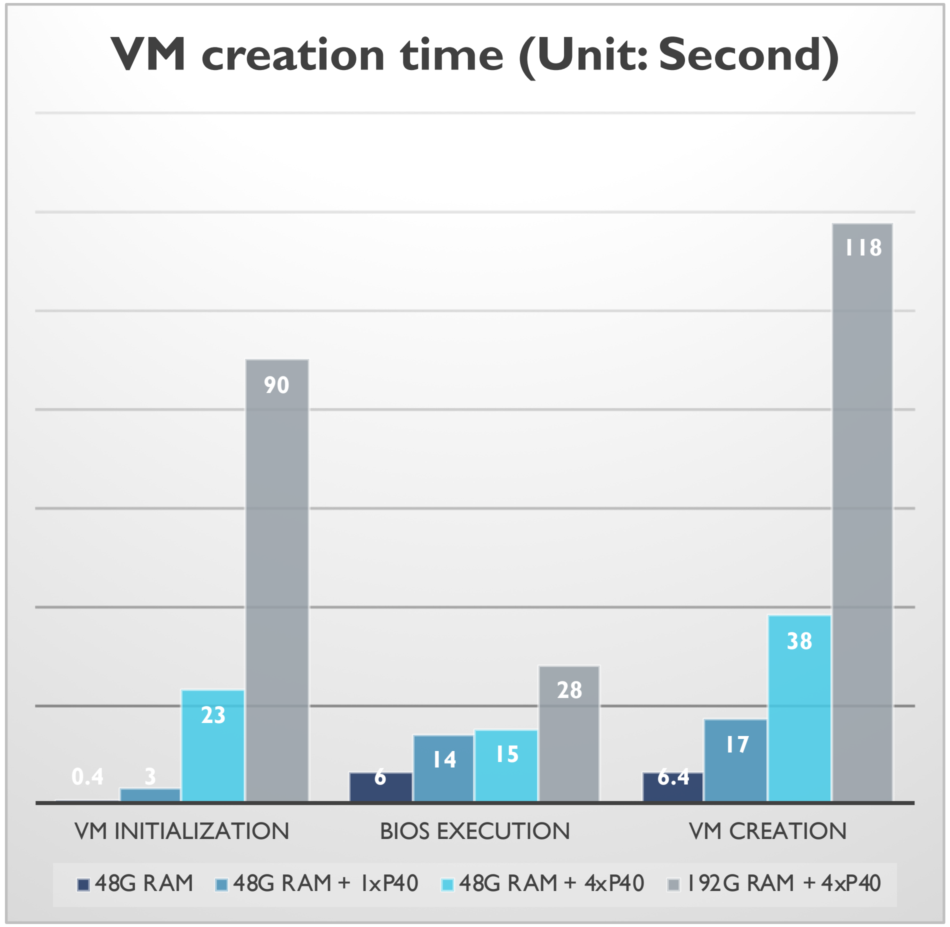
- Background
- Issues
- Solutions
- Effect of optimizations
- Conclusion

BACK GROUND

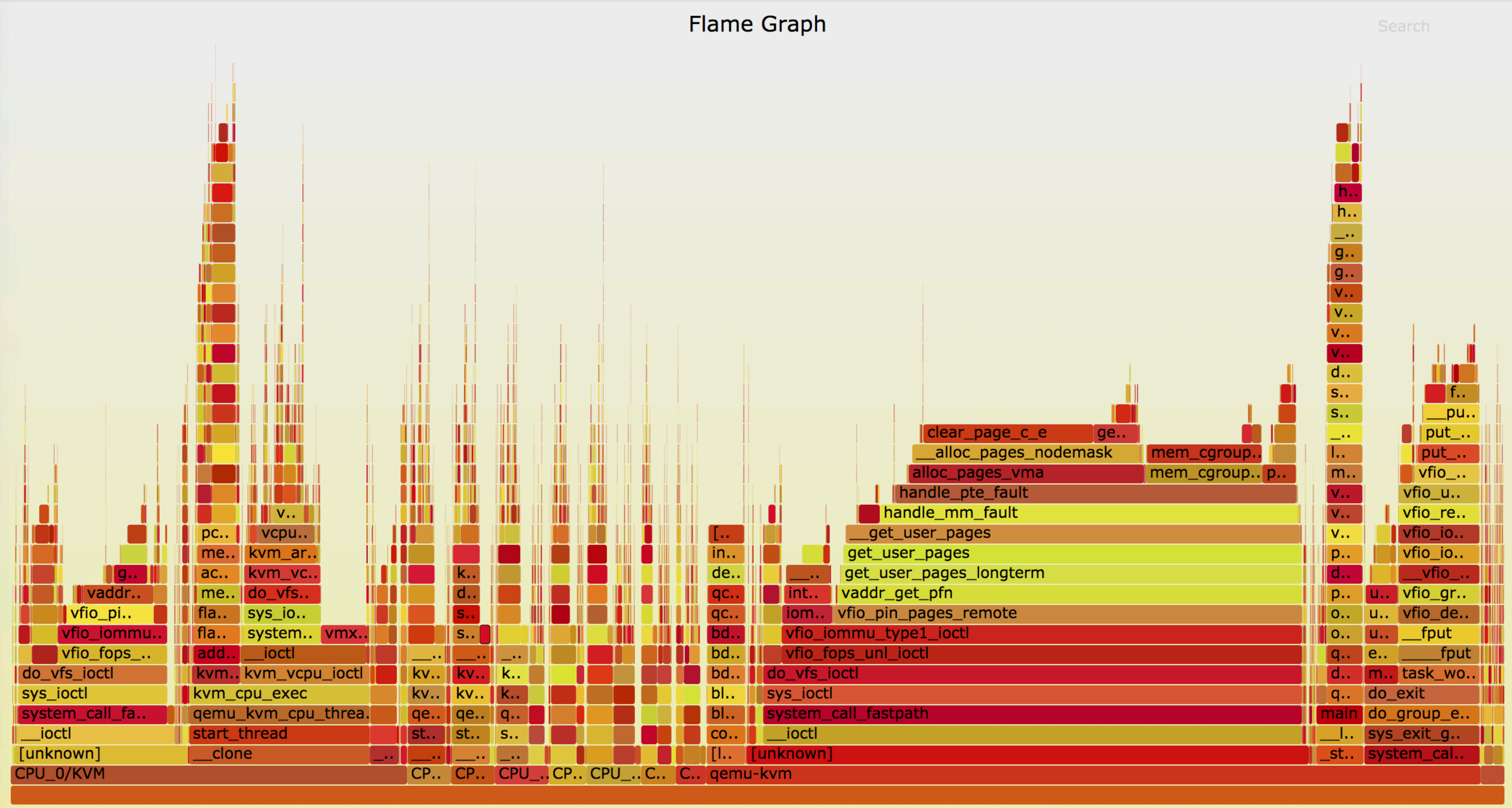
- CPU VM instance
  - A virtual machine without any passthrough device
- GPU VM instance
  - A virtual machine with one or more passthrough GPU cards
- Creation time of a VM instances with the 256G RAM
  - CPU VM instance： several Seconds
  - GPU VM instance： several Minutes
- Impact of a long VM creation time
  - Poor user experience
  - Computing resources waste

BASE LINE DATA

- Some definitions
  - VM Creation time
    - The time interval between QEMU process start to execute and guest kernel start to run
  - VM initialization time
    - The time interval between QEMU process start to execute and VCPU start to run
  - BIOS execution time
    - The time interval between VCPU start to run and the first guest kernel log is printed
- Factors affect GPU VM creation time
  - RAM size of VM
  - Type of GPU card
  - Count of GPU cards



WHAT SLOW DOWN CREATION OF A GPU VM INSTANCE



WHAT SLOW DOWN CREATION OF A GPU VM INSTANCE

- Function ` vfio_pin_pages_remote` is slow
- Repeated VFIO DMA map (unmap) for the same IOVA area
- PCI device reset
- KVM management meta data initialization
- Other miscellaneous configuration

SPEED UP VFIO_PIN_PAGES_REMOTE

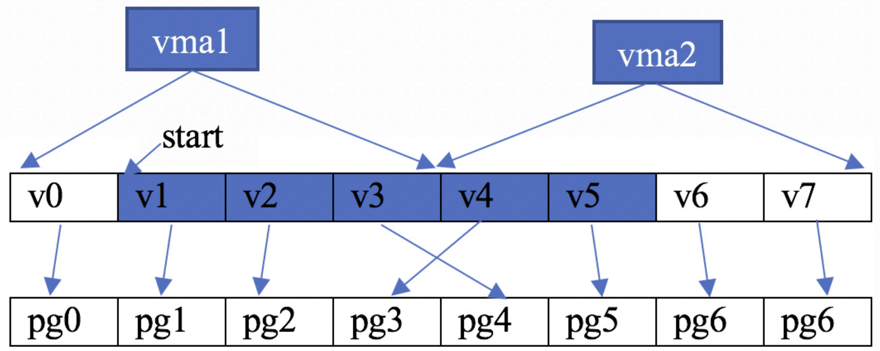
- Why ` vfio_pin_pages_remote` is slow ?
  - Zero out physical memory when allocating pages is time consuming
    - solution： Pre zero out free pages
  - Page per page process is inefficiency because of too many page table accessing
    - solution： Pin memory in bulk

PRE ZERO OUT FREE PAGES

- **Details of implementation**
  - Based on `free page reporting`
  - Zero out operation is done in a kernel worker thread
  - Set PG_zero flag in struct page when page content is zeroed out
  - Check PG_zero flag first if zero out is needed， skip zero out operation when set
  - PG_zero flag is cleared and zero out worker is woken up when a page is freed
  - RFC patch set: https://lore.kernel.org/lkml/20200412090728.GA19572@open-light-1.localdomain/

PIN MEMORY IN BULK

- **Details of implementation**
  - Add a new get_user_ct_page() and a new 'get_user_ct_pages_longterm()' to kernel mm
  - New semantics
    - Try to pin the specified pages in the same VMA
    - Return information about a bulk of physical continues memory
  - Use 'get_user_ct_page' and 'get_user_ct_pages_longterm' in `vaddr_get_pfn` to pin a bulk of memory
  - Use huge page will take more benefits



```
ret = get_user_pages_longterm (start, 5, write, force, pages, vmas);
```

pages	vmas	ret
pg1	vma1	5
pg2	vma1	
pg4	vma2	
pg3	vma2	
pg5	vma2	

```
ret = get_user_ct_pages_longterm (start, 5, write, force, pages, vmas);
```

pages	vmas	ret
pg1	vma1	2

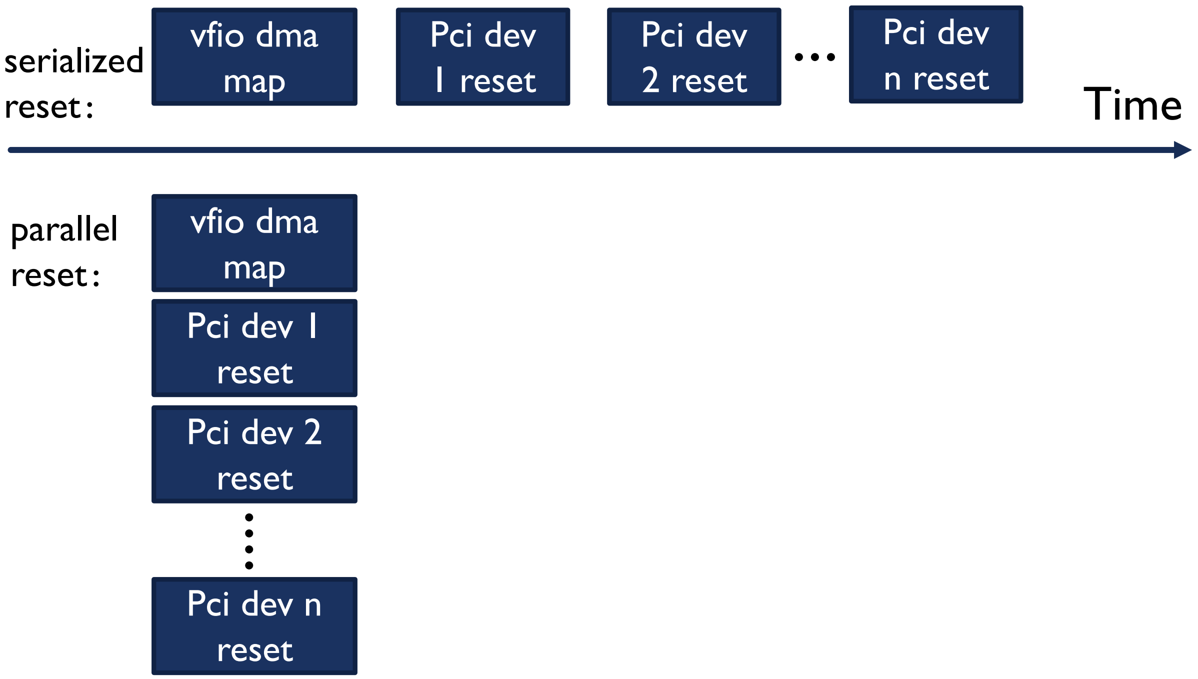
MAKE VFIO DMA MAP MORE EFFICIENT

- Current issues
  - Repeated vfio dma map and vfio dma unmap for the same IOVA area
  - Updating a part of an mapped IOVA area need to unmap the whole IOVA first and then redo the map
- Solutions
  - Manage VFIO DMA MAP in user space
    - Maintain VFIO DMA MAP IOVA information in QEMU
    - Do not do VFIO_IOMMU_UNMAP_DMA ioctl in vfio_dma_unmap， only do it when map a conflict IOVA area
    - Left the cleanup work which did by VFIO_IOMMU_UNMAP_DMA ioctl to kernel when QEMU process terminates
  - Split IOVA area which contain the address of 0x100000 before vfio dma map
    - One part is below 0x100000, which will be remapped
    - One part is above 0x100000, which keeps unchanged if the lower part get updated

PCI RESET OPTIMIZATION

- Current issues
  - One PCI device reset takes about 1 second
  - The same PCI device was reset twice during VM creation
    - One is in `qemu_system_reset`
    - Another in VFIO_GROUP_GET_DEVICE_FD ioctl
  - PCI devices reset operations are serialized

- Solutions
  - Remove the redundant PCI device reset
  - Do PCI device reset operations in parallel when there are multiple passthrough devices
  - Make PCI device reset in parallel with vfio dma map



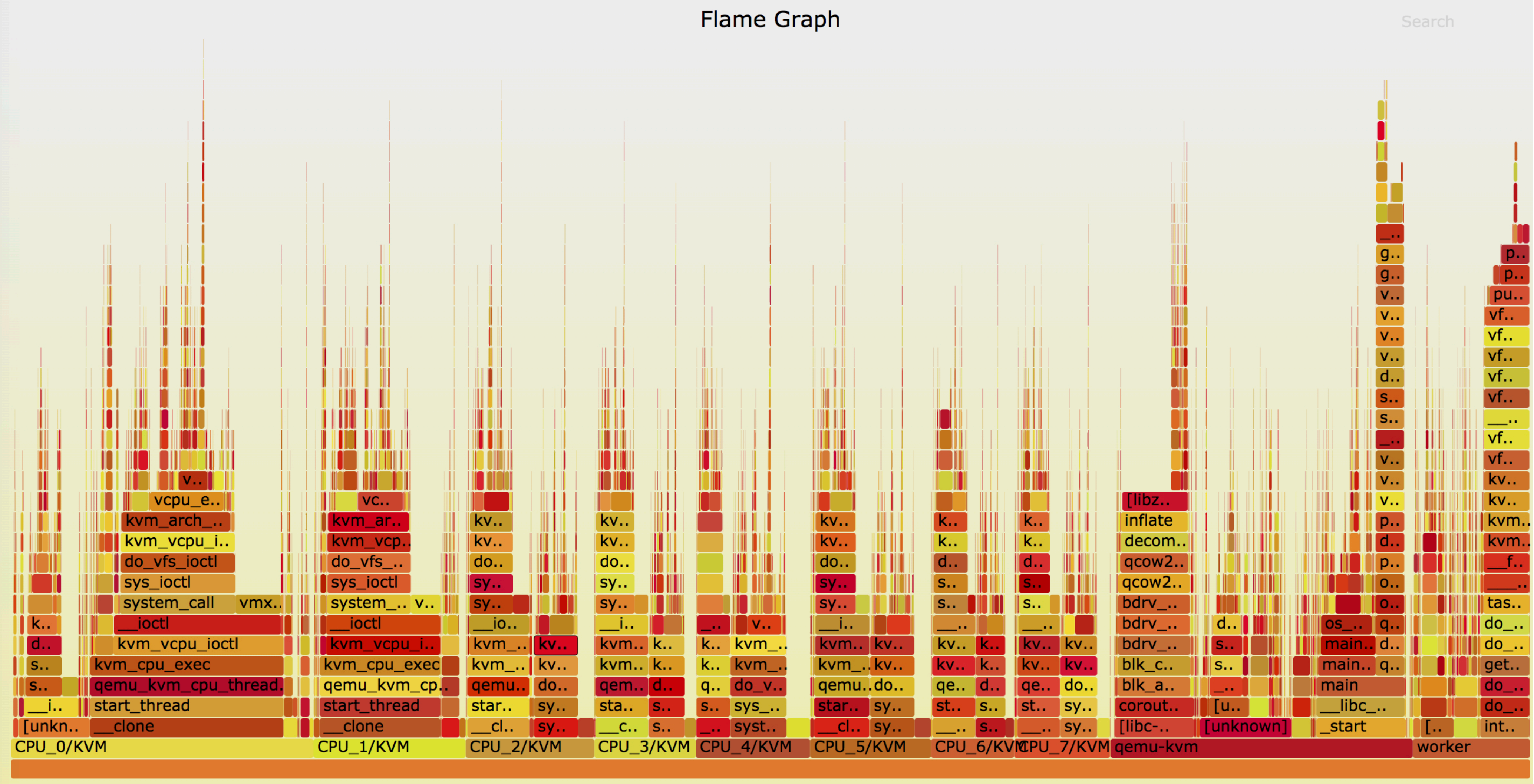
KVM META DATA OPTIMIZATION

- Current issues
  - Dirty page bitmap initialization for PCI device MMIO is time consuming and useless
  - EPT entries’ D bit should be set when PML is enabled, rmap traversal is time consuming
- Solutions
  - Skip PCI MMIO dirty page log related processing
  - Make the rmap traversal more efficient
    - Count the effective rmap, skip rmap traversal if the memory slot has no effective rmap items

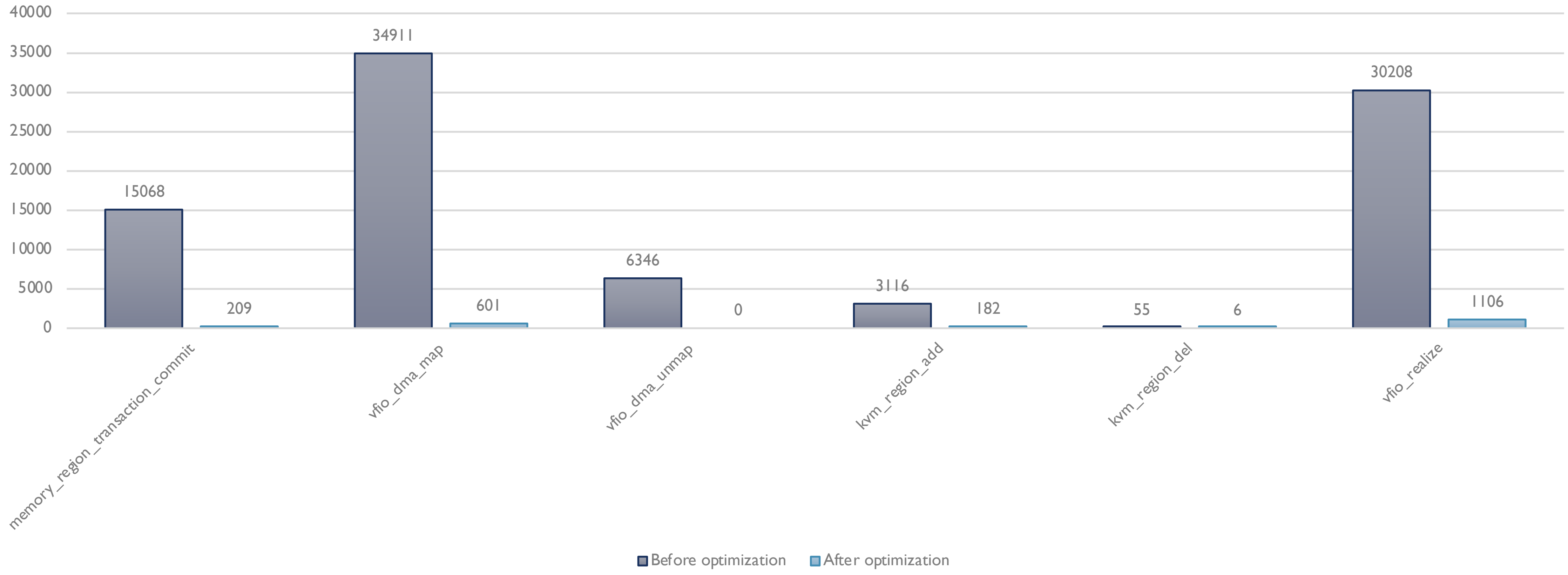
CONFIGURATION OPTIMIZATION

- Current issues
  - BIOS boot menu has 2.5 seconds of timeout by default
  - Guest grub has user defined timeout
  - Improper NUMA strategy will slow down page allocation
- Solutions
  - Disable boot menu
  - Change guest grub timeout to 0
  - Be careful with the NUMA memory policy

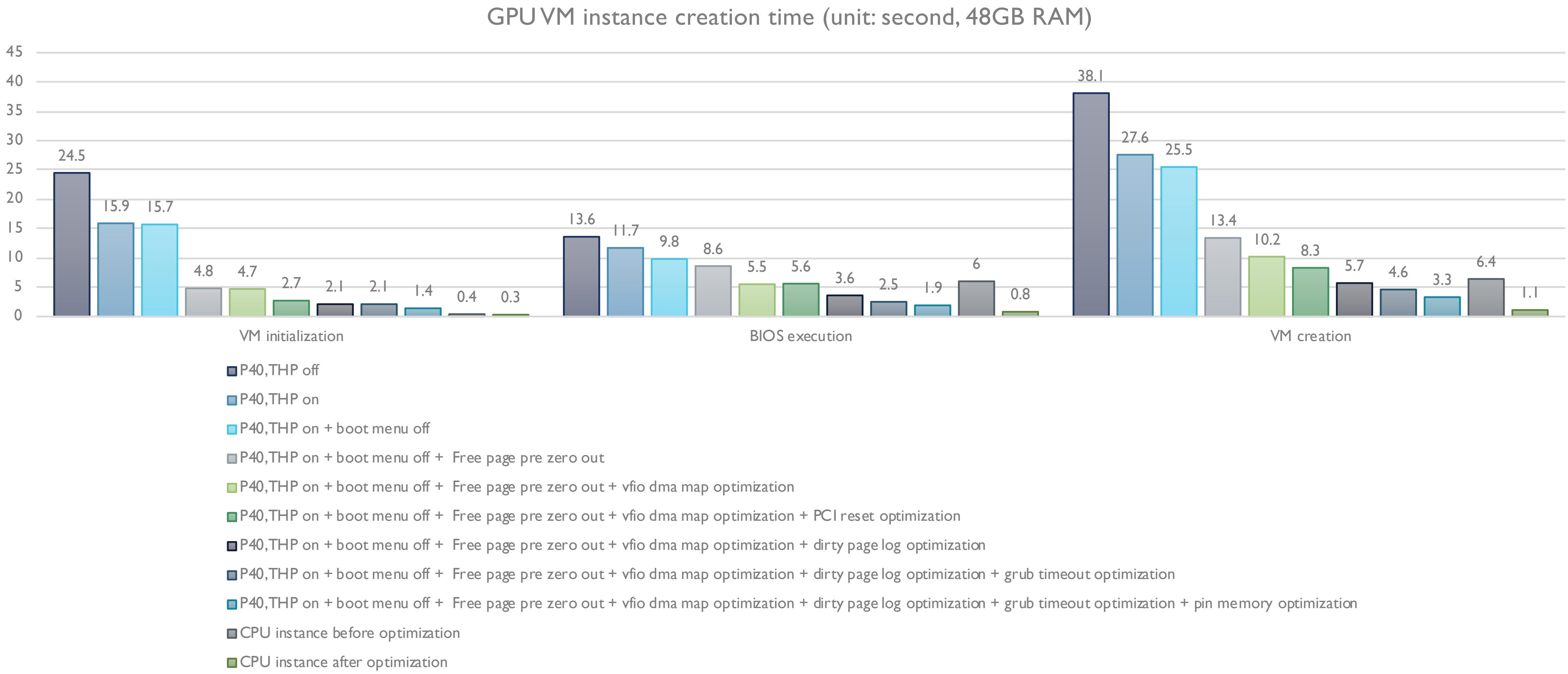
EFFECT OF OPTIMIZATION



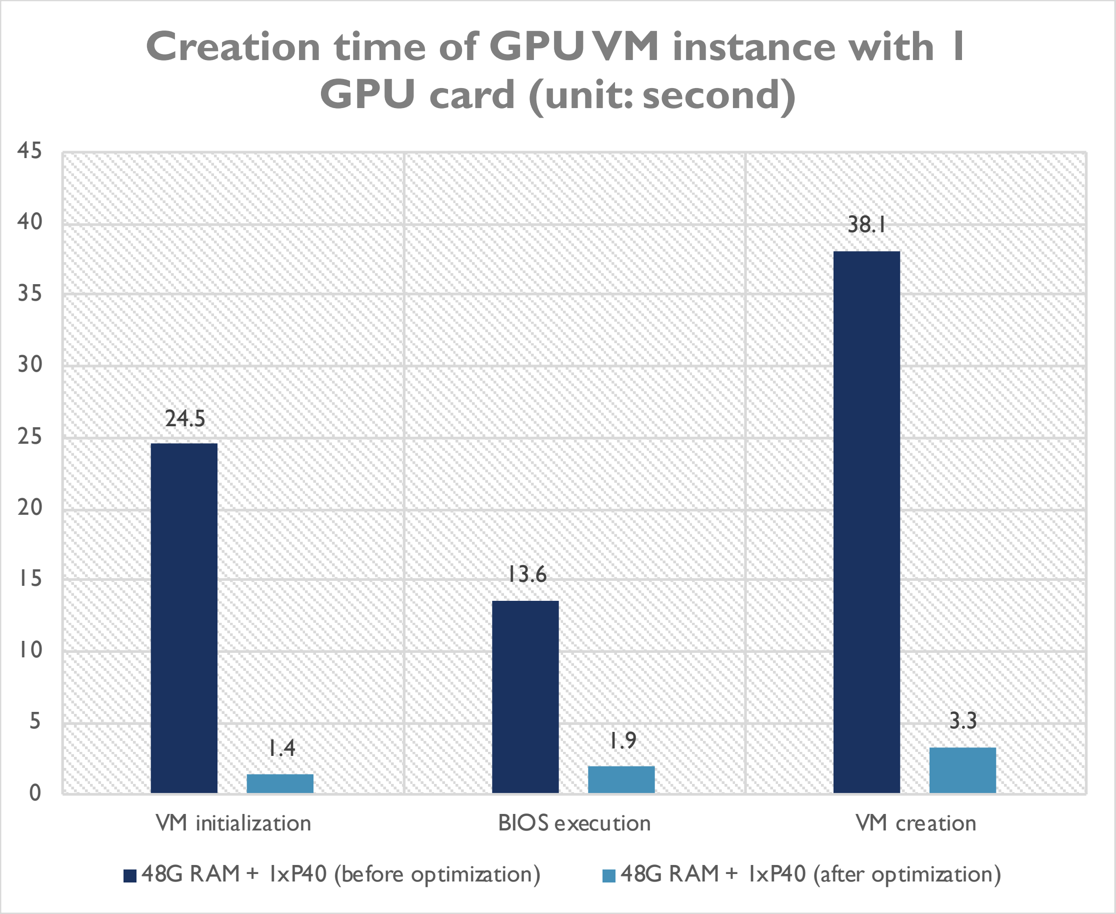
EFFECT OF OPTIMIZATION

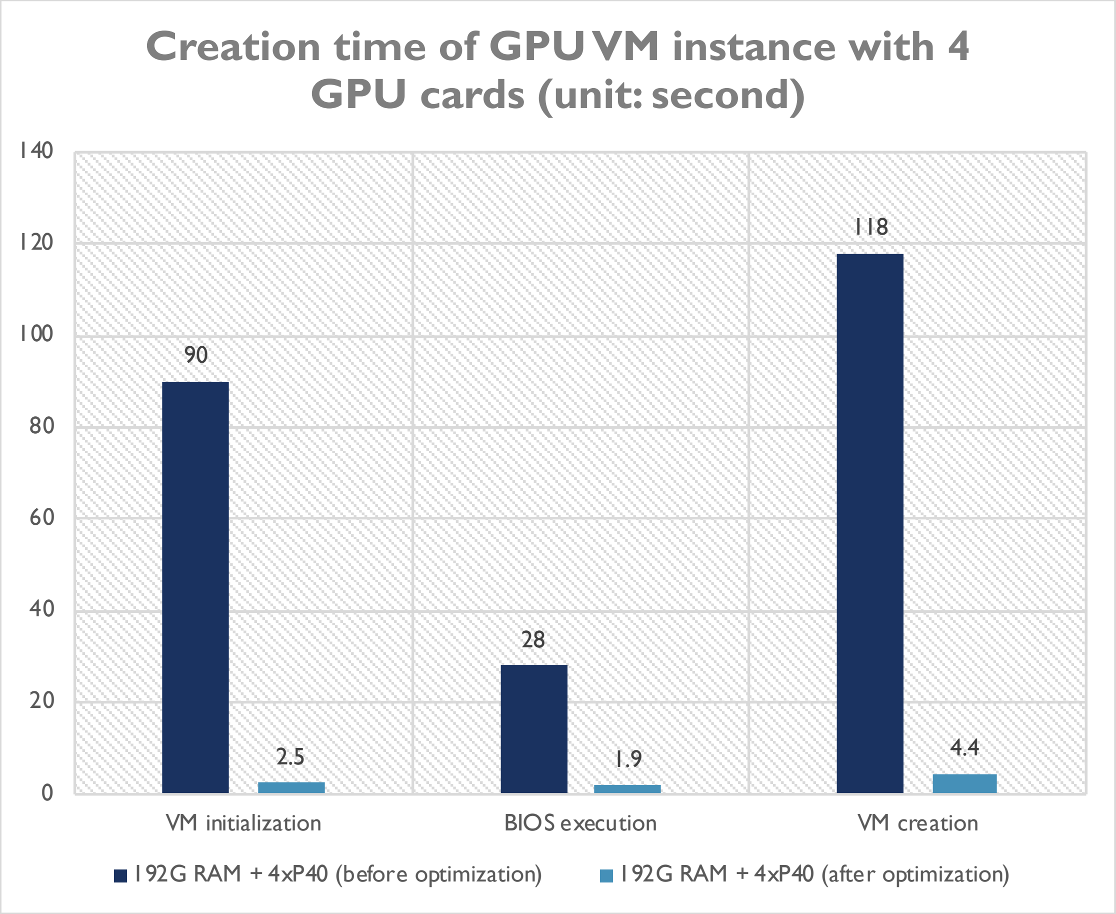

Main QEMU functions accumulated time cost comparison(Unit: ms)

EFFECT OF OPTIMIZATION



EFFECT OF OPTIMIZATION





CONCLUSION

- All the optimizations are not limited to GPU passthrough device, they apply to other PCI passthrough devices too.
- Limitations of pre zero out free page
  - Hugetlb fs can’t always benefit from current implementation
  - Page allocation speed remains the same when pages were not zeroed out in time
- Pros
  - Transparent to guest
  - DMA operation in BIOS stage can be handled correctly
- TODO
  - About GPU VM creation time, there is some room for further improvement
  - Linux memory management can be improved for device passthrough scenario
  - Contribute our work to upstream

THANKS

Liang Li <liliang324@gmail.com>

滴滴

滴滴云