HMM: GUP NO MORE !
XDC 2018

Jérôme Glisse

HETEROGENEOUS COMPUTING
CPU is dead, long live the CPU

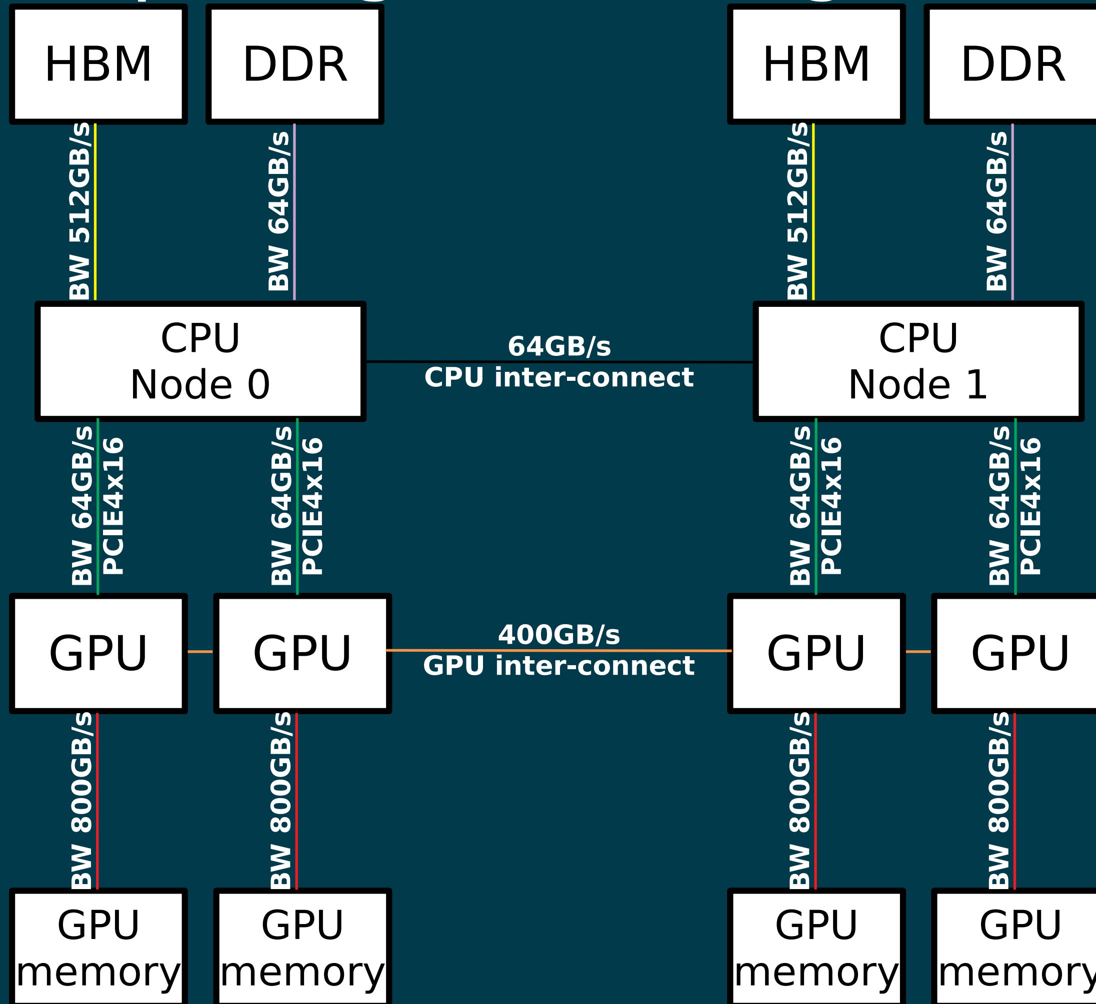
Heterogeneous computing is back, one device for each workload:

- GPUs for massively parallel workload
- Accelerators for specific workload (encryption, compression, IA, ...)
- FPGA for more specific workload

CPU is no longer at the center of everything and

- Device have they own local fast memory (GPU, FPGA, ...)
- System topology is more complex than just CPU at a center
- Hierarchy of memory (HBM, DDR, NUMA)

MEMORY HIERARCHY
Computing is nothing without data



redhat.

EVERYTHING IN THE RIGHT PLACE
One place for all no longer work

For good performance dataset must be placed the closest to the compute unit (CPU, GPU, FPGA, ...). This is a hard problem:

- Complex topology, hard to always pick best place
- Some memory not big enough for whole dataset
- Dataset can be use concurrently by multiple unit (CPU, GPU, FPGA, ...)
- Lifetime of use, dataset can first be use on one unit than on another
- Sys-admin resource management (can means migration to make room)

redhat.

BACKWARD COMPATIBILITY
Have to look back ...

Can not break existing application:

- Allow library to use new memory without updating application
- Must not break existing application expectation
- Allow CPU atomic operation to work
- Using device memory should be transparent to the application

Not all the inter-connect are equal

- PCIE can not allow CPU access to device memory (no atomic)
- Need CAPI or CCIX for CPU access to device memory
- PCIE need special kernel handling for device memory to be use without breaking existing application

redhat.

SPLIT ADDRESS SPACE DILEMMA
Why mirroring ?

One address space for CPU and one address space for GPU:

- GPU driver built around memory object like GEM object
- GPU address not always expose to userspace (depends on driver API)
- Have to copy data explicitly between the two
- Creating complex data structure like list, tree, … cumber-stone
- Hard to maintain a complex data structure synchronize on CPU and GPU
- Break programming language system and memory model

redhat.

SAME ADDRESS SPACE SOLUTION
It is a virtual address !

Same address space for CPU and GPU:

- GPU can use same data pointer as CPU
- Complex data structure work out of the box
- No explicit copy
- Preserve programming language system and memory model
- Can transparently use GPU to execute portion of a program

HOW IT LOOKS LIKE

Same address space an example

From:

```
typedef struct {
        void *prev;
        long gpu_prev;
        void *next;
        long gpu_next;
} list_t;

list_add(list_t *entry, list_t *head) {
        entry→prev = head;
        entry→next = head→next;
        entry→gpu_prev = gpu_ptr(head);
        entry→gpu_next = head→gpu_next;
        head→next→prev = entry;
        head→next→gpu_prev = gpu_ptr(entry);
        head→next = entry;
        head→gpu_next = gpu_ptr(entry);
}
```

To:

```
typedef struct {
        void *prev;
        void *next;
} list_t;

list_add(list_t *entry, list_t *head) {
        entry→prev = head;
        entry→next = head→next;
        head→next→prev = entry;
        head→next = entry;
}
```


redhat.

GUP (get_user_pages)
It is silly talk

- GUP original use case was direct IO (archaeologist agree)
- Driver thought it was some magical calls which:
  - Pin a virtual address to page
  - Allow driver to easily access program address space
  - Allow driver and device to work directly on program memory
  - Bullet proof it does everything for you …

IT DOES NOT GUARANTY ANY OF THE ABOVE DRIVER ASSUMPTIONS !

redhat.

GUP (get_user_pages)
What is it for real ?

Code is my witness ! GUP contract:

- Find page backing a virtual address at instant T and increment its refcount

Nothing else ! This means:

- By the time GUP returns to its caller the virtual address might be back by a different page (for real !)
- GUP does not magically protect you from fork(), truncate(), ...
- GUP does not synchronize with CPU page table update
- Virtual address might point to a different page at <u>any</u> times !

redhat.

HMM
Heterogeneous Memory Management

It is what GUP is not, a toolbox with many tools in it:

- Army Swiss knife for driver to work with program address space !
- The one stop for all driver mm needs !
- Mirror program address space onto a device
- Help synchronizing program page table with device page table
- Use device memory transparently to back range of virtual address
- Isolate driver from mm changes
  - When mm changes, HMM is updated and the API it expose to the driver stays the same as much as possible
- Isolate mm from drivers
  - MM coders do no need to go modify each device drivers, only need to update HMM code and try to maintain its API

redhat.

HMM WHY ?
It is all relative

Isolate glorious mm coders from pesky drivers coders

Or

Isolate glorious drivers coders from pesky mm coders

This is relativity 101 for you ...

redhat.

HOW THE MAGIC HAPPENS
Behind the curtains

Hardware solution:

- PCIE:
  - ATS (address translation service) PASID (process address space id)
  - No support for device memory
- CAPI (cache coherent protocol for accelerator PowerPC)
  - Very similar to PCIE ATS/PASID
  - Support device memory
- CCIX
  - Very similar to PCIE ATS/PASID
  - Can support device memory

Software solution:

- Can transparently use GPU to execute portion of a program
- Support for device memory even on PCIE
- Can be mix with hardware solution

PRECIOUS DEVICE MEMORY
Don’t miss on device memory

You want to use device memory:

- Bandwidth (800GB/s – 1TB/s versus 32GB/s PCIE4x16)
- Latency (PCIE up to ms versus ns for local memory)
- GPU atomic operation to it local memory much more efficient
- Layers: GPU→IOMMU→Physical Memory

redhat.

PCIE IT IS A GOOD SINK
PCIE what is wrong ?

Problems with PCIE:

- CPU atomic access to PCIE BAR is undefined (see PCIE specification)
- No cache coherency for CPU access (think multiple cores)
- PCIE BAR can not always expose all memory (mostly solve now)
- PCIE is a packet protocol and latency is to be expected

redhat.

HMM HARDWARE REQUIREMENTS
Magic has its limits

Mirror requirements:

- GPU support page fault if no physical memory backing a virtual address
- Update to GPU page table at any time, either:
  - Asynchronous GPU page table update
  - Easy and quick GPU preemption to update page table
- GPU supports at least same numbers of bits as CPU (48-bit or 57-bit)

Mix hardware support (like ATS/PASID) requirements:

- GPU page table with per page select of hardware path (ATS/ID)

HMM device memory requirements:

- Never pin to device memory (always allow migration back to main memory)

redhat.

HMM A SOFTWARE SOLUTION
Workaround lazy hardware engineer

HMM toolbox features:

- Mirror process address space (synchronize GPU page table with CPU one)
- Register device memory to create struct page for it
- Migrate helper to migrate range of virtual address
- One stop for all mm needs
- More to come ...

redhat.

HMM HOW IT WORKS
For the curious

How to mirror CPU page table:

- Use mmu notifier to track change to CPU page table
- Call back to update GPU page table
- Synchronize snapshot helpers with mmu notifications

How to expose device memory:

- Use struct page to minimize changes to core mm linux kernel
- Use special swap entry in CPU page table for device memory
- CPU access to device memory fault as if it was swapped to disk

MEMORY PLACEMENT
Automatic and explicit

- Automatic placement is easiest for application hardest for kernel
- Explicit memory placement for maximum performance and fine tuning

redhat.

AUTOMATIC MEMORY PLACEMENT
Nothing is simple

Automatic placement challenges:

- Monitoring program memory access pattern
- Cost of monitoring and handling automatic placement
- Avoid over migration ie spending too much time moving things around
- From too aggressive automatic migration, to too slow
- For competitive device access, which device to favor
- The more complex the topology the harder it is
- Heuristics different for every class of devices

redhat.

EXPLICIT MEMORY PLACEMENT
Application controlling where is what

Explicit placement:

- Application knows best:
  - What will happen
  - What is the expected memory access pattern
- No monitoring
- Programmers must spend extra time and effort
- Programmers can not always predict their program pattern

redhat.

HBIND: EXPLICIT PLACEMENT API
New kernel API for explicit placement

Few issues with mbind():

- mbind() is NUMA centric, everything in a node is at the same level
- mbind() use bitmap so select node, hard to add device as node
- mbind() is CPU centric

Heterogeneous bind hbind() intends to address mbind shortcomings:

- Depends on new memory enumeration in sysfs
- Can mix/select CPU and/or device memory for a range of virtual address
- Can support memory hierarchy inside a node:
  - From HBM for CPU (fastest but relatively small)
  - To main memory (fast but relatively big)
  - To non volatile memory (slower but extremely big)

redhat.

MEMORY PLACEMENT
What to do ?

Allow both, automatic and explicit memory placement, to co-exist !
Programmers that can predict access pattern and want to spend the extra time to do explicit management should be allow. Automatic placement should be use for everyone else to try to maximize performance.

Still leaves questions:

- Disable automatic for range that have an explicit policy ?

HMM: MORE THINGS TO COME
What is missing for you ?

Things coming up soon in a kerne near you:

- Generic page write protection:
  - Faster device atomic to memory (through regular PCIE memory write)
  - Duplicate memory (across GPU and main memory, across GPUs)
- Seamless peer to peer (with ACS) for device memory
- Add an helper for get_user_pages_fast()
- Integrate with DMA/IOMMU to share resources
- Support hugetlb fs ? Does anyone care about that ?

redhat.

HMM THE SUMMARY

Heterogeneous Memory Management

HMM is a toolbox for heterogeneous memory management

- Mirror process address space:
  - Same virtual address point to same memory on both devices and CPUs
  - Keep devices and CPUs virtual address to physical mapping in sync
- Migrate helpers to move memory:
  - Move a range of virtual address to given physical memory
  - Allow migration to and from device memory
  - Policy left to device driver or application
- Support PCIE system (allow to use PCIE device memory):
  - Hiding the device memory as swap on PCIE
  - CPU access fault (like if memory was swap to disk)
  - Automatically migrate memory back on CPU access
- Peer to peer between devices:
  - NiC to GPU or GPU to NiC
  - ...

redhat.

THANK YOU

plus.google.com/+RedHat

facebook.com/redhatinc

linkedin.com/company/red-hat

twitter.com/RedHat

youtube.com/user/RedHatVideos