Red Hat
Enterprise Linux

malloc for everyone

Jérôme Glisse

Red Hat

CPU IS NOT ALONE

New, highly parallel, workload:
- AI model training and model execution
- Image processing (recognition, enhancement, ...)
- Math large dataset (matrices, vectors, ...)
- Signal processing (FFT, ...)

GPU, FPGA, DSP can be 100x faster, or more, than CPU

LIBRARY AND MODULARITY

Increase in program modularity:

- Use more and more library (BLAS, math, FFT, ...)
- Versatile program pipeline (enable/disable block)
- Many input sources (disk, network, sensor, ...)

Programmers no longer control all the code, they rely more and more on ready to use building blocks.

DATA COMING AND GOING EVERYWHERE

Processors (CPU, GPU, FPGA, ...) work on data from/to various devices:

- Network
- Storage
- Sensors

...

Having CPU passing data from input/output device to other processors (GPU, FPGA, ...) is a bottleneck !

DATA-FLOW EVERYWHERE

Computation pipeline can interleave various processors (CPU, GPU, FPGA, ...)

Having CPU being a middleman between each processor is a bottleneck !

WHAT IS AN ADDRESS SPACE ?

An address space is a mapping between virtual address (pointers in your program) and physical memory.

Each programs on your computer have their own CPU address space (fork and exec).

Each CPUs threads of a program share a common address space (pthread).

malloc()/free() is what manage the address space (mmap/munmap from kernel perspective)

DEVICE ADDRESS SPACE

Each device can have its own address space manage through device specific API (DeviceAPIMalloc)

For flat dataset (matrix, vector, image, ...) this is not too hard (DeviceAPIMalloc follow by DeviceAPIMemcopy).

It is hard and error prone for complex dataset (any kind of graph, collections, ...) have to re-allocate into device address each node and re-build the pointers graph.

DEVICE ADDRESS SPACE GRAPH EXAMPLE

Duplicating a list into a device address space:

```
device_prev_entry = NULL;
list_for_each_entry(entry, head, list) {
   struct mystruct *device_entry = deviceAPIMalloc();
   deviceMemcopy(device_entry->data, entry->data);
   device_entry->next = NULL;
   if (device_prev_entry) {
      device_prev_entry->next = deviceAPIPointer(device_entry);
   }
   device_prev_entry = device_entry;
}
```

SAME ADDRESS SPACE FOR EVERYONE

Would it not be easier if all compute device (GPU, FPGA, ...) had the same address space ?

It would mean all compute device would be able to access any CPU valid pointers.

Yes it is easier !

NOT ALL PHYSICAL MEMORY IS EQUAL

Multiple type of physical memory:

- Main memory (DDR DIMM reasonably fast 60GB/s)
- HBM (faster than a roadrunner 200GB/s to 1000GB/s)
- Device memory (200GB/s to 1000GB/s local access)

System bus (PCIE for instance) can be a bottleneck.

AN OPTIMIZATION PROBLEM
NUMA IS BACK AND HE IS ANGRY

When a processor (CPU, GPU, FPGA, ...) works on a dataset (range of virtual address) you want to **use the fastest physical memory** to back the range if virtual address.

A dataset can be worked on by different processors (CPU, GPU, ...) one after the other. What is the fastest memory for the first processor might not be for the second.

NUMA all over again !

MIGRATION FOR EVERYONE

To use the fastest memory means we have to migrate from on type of physical memory to another (for a range of virtual address).

For multi-sockets CPUs, ie NUMA system, we have an API:

- migrate_pages()
- move_pages()
- mbind()

MIGRATE_PAGES() THE HAMMER

```
migrate_pages(int pid, unsigned long maxnode,
              const unsigned long *old_nodes,
              const unsigned long *new_nodes)
```

Move all pages in a process to another set of nodes.

Big hammer, not what we want, we can have different processors (CPU, GPU, ...) working concurrently on different dataset (range of virtual address).

MOVE_PAGES() THE CHERRY PICKER

```
move_pages(int pid, unsigned long count, void **pages,
                  const int *nodes, int *status, int flags)
```

Move individual pages of a process to another node, one by one … that’s a lot of pages for giga bytes dataset (1GB/4KB = 2^18 = 262144)

Cherry picking one page at a time is highly flexible but also highly intensive (building pages and nodes array).

Not the most efficient way.

MBIND()

```
mbind(void *addr, unsigned long len, int mode,
        const unsigned long *nodemask,
        unsigned long maxnode, unsigned flags)
```

Set memory policy for a memory range. The nodemask points to a bit mask of nodes containing up to maxnode bits. If bits N is set then it means you want to use memory on node N. Multiple bits can be set so that kernel can use multiple nodes.

Almost it ...

A HANDFUL OF NODES

Each time we migrate something we are usualy targeting a handful of possible physical memory and with an order preference ie first try to use the fastest, if you run out fallback to the second one, ...

A nodemask is wasteful (highly sparse ie many zero bits) and lack ordering.

NODE IS NOT JUST ONE TYPE OF MEMORY

Using node for migration target fails to capture the node complexity. Each node can have multiple different types of physical memory (HBM, DDR DIMM, Persistent DIMM, ...).

So instead of node, what we want is to be able to specify which physical memory directly.

DREAMING OF

```
pmbind(void *addr, unsigned long len, int mode,
            const unsigned long *physmemids,
            unsigned long maxids, unsigned flags)
```

Bind a range of virtual address [addr, addr+len] to the ordered list of physical memory whose identifient are pass in physmemids.

Can replace mbind() as a more versatile solution.

IDENTITY CRISIS

We need to be able to identify physical memory (including device memory) in a system with an id.

Sadly /sys/devices/system/memory/ is a wreck (one id per 128MB section on x86-64)

Existing programs do have expectations from memory in /sys/devices/system/memory/ and device memory can not full-fill those expectation (cache coherency, atomics, ...).

NEW ORDER

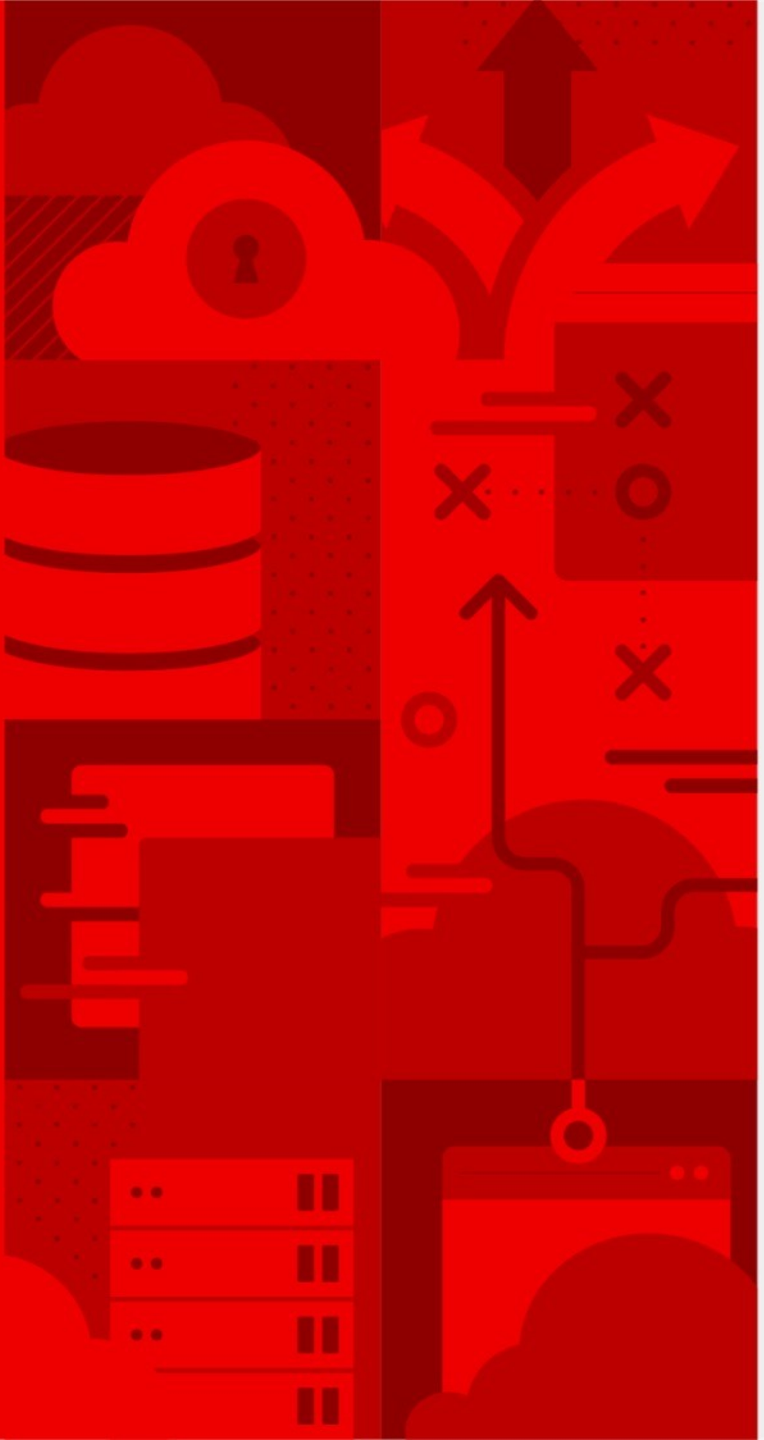
A new memory device in driver model.

One id for each physical memory with same characteristics:
- One id for all DDR DIMM in a node
- Each node can have multiple id for different memory
- Each device can have multiple id for different memory

All in /sys/devices/system/memoryids/ with:
- link to node (even device are attached to a node)
- link to device (if the memory belong to a device)
- info (local: bandwidth, latency, ...)

SPEAK NOW

Who is against ?



Thank you

Red Hat is the world’s leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/user/RedHatVideos

twitter.com/RedHat

Red Hat