Consolidating representations of the physical memory

Mike Rapoport
<rppt@linux.ibm.com>

This project has received funding from the *European Union's Horizon 2020 research and innovation programme* under grant agreement No 825377

How many end-of-memory variables you have, grandma!

a comment in x86::setup_arch()

Simple cases

- Flat memory
- No memory hotplug
- No kexec support

```
csky, h8300, hexagon, microblaze,
nds32, nios2, openrisc, um
```

Adding complexity

- SPARSEMEM
- Holes in flat memory map
- kexec
- Memory hot(un)plug

Holes in memory map

- Holes in physical memory do not have memory map
  - Requires custom `pfn_valid()`
- arc uses old good `min_pfn` and `max_pfn`
- m68k has virtually contiguous direct mapping
  - `pfn_valid()` when `virt_addr_valid()`
- arm relies on memblock
  - `pfn_valid()` when `memblock_is_memory()`
  - Slow with crazy ACPI memory layouts

kexec

- Relies on resource tree to find free memory
  - “System RAM” is not quite IOMEM
- For !x86 “System RAM” may contain firmware blobs
- Duplicated code in `arch/` to register memory resources
  - Can easily go out of sync with memblock and memory map

Memory hot(un)plug

- Per architecture representation of hot(un)plugable regions
- Xarray of `memory_block`'s
  - Until recently `memory_block`'s were only accessible via memory device hierarchy
- Duplicated updates of data structures
  - Register resource
  - Create memory block
  - Add memblock

Existing representation

- “System RAM” subtree in iomem_resource
- memblock if `ARCH_KEEP_MEMBLOCK`
- memory_blocks if `MEMORY_HOTPLUG`
- Architecture specific data structures
  - `min_low_pfn, max_low_pfn, min_high_pfn, max_high_pfn`
  - `m68k::m68k_memory, parisc::pmem_ranges`
  - `x86::e820, x86::numa_meminfo, powerpc::drmem_lmb`

Physical memory

- A collection of contiguous memory banks
  - Up to x86's first megabyte madness
- A bank
  - Spawns a fixed address range
  - Belongs to a NUMA node
  - May be hot(un)plugged
- Nodes may have hotplug ranges
  - Empty on boot

Kernel view of physical memory

- Firmware supplies memory description
  - Physical address ranges
  - Ranges used by the firmware
    - Some cannot be mapped in kernel page tables
  - Unusable memory, e.g. because of HW errors
- Free and used memory ranges may or may not intersect
  - device tree vs e820

Physical memory model

- Representation of the memory bank
  - Address range
  - Attributes
    - Hotpluggable
    - Mapping is prohibited
    - Onlining controls
  - NUMA node
  - struct device for memory hotplug

Physical memory model

- Representation of the occupied memory
  - Address range
  - Attributes
    - Firmware defined type: ACPI tables, EfiRuntimeServicesData, ...
    - Reservation type: unusable, firmware, kernel
    - Mapping is prohibited?
  - NUMA node?

Physical memory model

- A collection to glue memory bank and reserved memory representations
- Implementation alternatives:
  - Completely new module
  - Based on resource tree
  - Based on memblock

Resource tree

- Is “System RAM” an IOMEM resource?
  - `struct resource` defined in `include/linux/ioport.h`
- `IORESOURCE_BITS` do not reflect required attributes
- Not supported on all architectures
- Traversals include actual IOMEM resources, burning cycles for nothing
- Resource requests model is too strict

Memblock

- Used by all architectures
- Allows adding and reserving memory from the very start
  - Up to reasonable limits
- Comparable in performance with the resource tree
- `struct memblock_region` has most of the necessary bits

```
struct memblock_region {
        phys_addr_t base;
        phys_addr_t size;
        enum memblock_flags flags;
#ifdef CONFIG_NUMA
        int nid;
#endif
+#ifdef CONFIG_MEMORY_HOTPLUG
+         struct device *dev;
+#endif
};
```

```
enum memblock_flags {
        /* No special request */
        MEMBLOCK_NONE       = 0x0,
        /* hotpluggable region */
        MEMBLOCK_HOTPLUG    = 0x1,
        /* mirrored region */
        MEMBLOCK_MIRROR     = 0x2,
        /* don't add to direct map */
        MEMBLOCK_NOMAP      = 0x4,
+        /* unusable */
+        MEMBLOCK_UNUSABLE   = 0x8,
+        /* used by firmware */
+        MEMBLOCK_FIRMWARE   = 0x10,
};
```

Gaps

- No locking
- `memblock_remove` may fail
- Perceived as an allocator
  - Maybe rename `memblock_alloc` back to `bootmem_alloc`?
- x86 has gaps in memblock integration since 2.6

Immediate steps

- Move “System RAM” setup to memblock
  - https://lore.kernel.org/all/20210531122959.23499-1-rppt@kernel.org
- Add flags for reserved regions
- Make boundary between memory representation and boot time allocator clearer

Longer term

- Ensure `memblock_remove` does not fail
- Sort out inconsistencies between architectures and generic code
- Remove redundant arch-specific data

Far fetched

- Convert user visible ABIs to use memblock as “baking store”
  - /sys/devices/system/memory
  - /sys/firmware/memory
  - /proc/iomem?
- Enable ARCH_KEEP_MEMBLOCK on architectures supporting memory hotplug

So, what am I missing?

Thank you!