Asymmetric NUMA:

Multiple-memory management for the rest of us

Paul Mundt
paul.mundt@renesas.com

Magnus Damm
damm@igel.co.jp

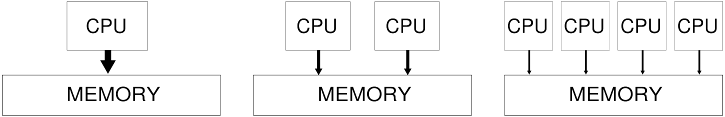
Renesas Technology

ELC-Europe 2007

Outline

Outline

Uniprocessor and Beyond



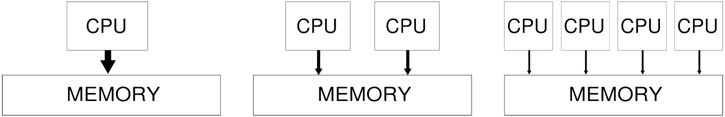
Uniprocessor System (UP):

- A single processor has all memory bus bandwidth to itself.

Symmetric Multi-Processor System (SMP):

- Multiple processors *share* memory bus bandwidth.

Uniprocessor and Beyond



Uniprocessor System (UP):

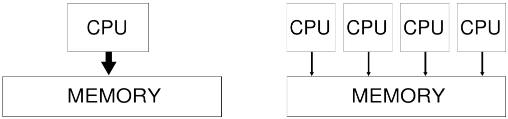
- A single processor has all memory bus bandwidth to itself.

Symmetric Multi-Processor System (SMP):

- Multiple processors *share* memory bus bandwidth.

Memory bus bandwidth becomes a bottleneck for large systems.

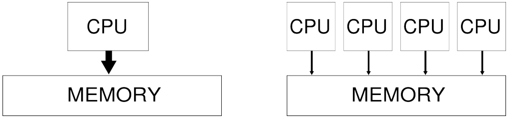
From UMA to NUMA



Uniform Memory Access (UMA)

- Uniprocessor or Symmetric Multi-Processor configurations.
- Memory is accessed through a high-speed local bus.

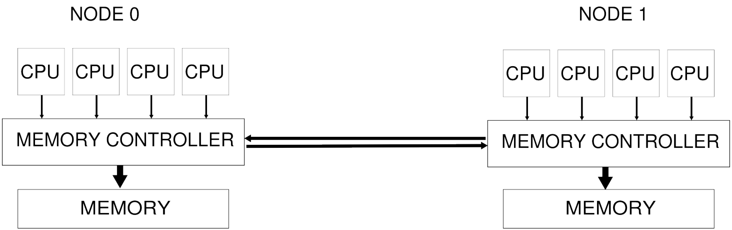
From UMA to NUMA



Uniform Memory Access (UMA)

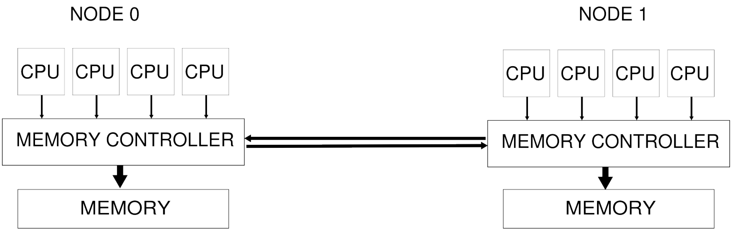
- Uniprocessor or Symmetric Multi-Processor configurations.
- Memory is accessed through a high-speed local bus.

Memory access speed is constant for all processors and memories.



Non-Uniform Memory Access (NUMA):

- A NUMA system is divided into multiple *nodes*.
- Each node is equipped with zero or more processors.
- Local memory may exist on the same node as the processor.
- Remote memory access is provided through interconnects.



Non-Uniform Memory Access (NUMA):

- A NUMA system is divided into multiple *nodes*.
- Each node is equipped with zero or more processors.
- Local memory may exist on the same node as the processor.
- Remote memory access is provided through interconnects.

With NUMA the access speed for memory varies with the *distance* between processors and memory regions.

Symmetric and Asymmetric NUMA

Symmetric NUMA:

- Equal amounts of memory is assigned to each node.

Symmetric and Asymmetric NUMA

Symmetric NUMA:

- Equal amounts of memory is assigned to each node.

Asymmetric NUMA:

- The amount of memory for each node may vary greatly.
- Nodes may be equipped with memory but without processors.
- Memoryless nodes - processor-only configurations.

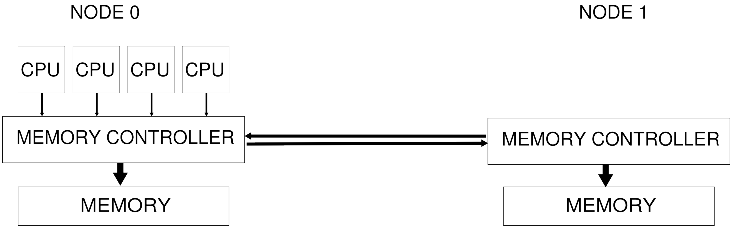
Symmetric and Asymmetric NUMA

Symmetric NUMA:

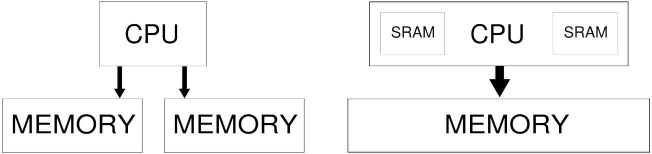
- Equal amounts of memory is assigned to each node.

Asymmetric NUMA:

- The amount of memory for each node may vary greatly.
- Nodes may be equipped with memory but without processors.
- Memoryless nodes - processor-only configurations.



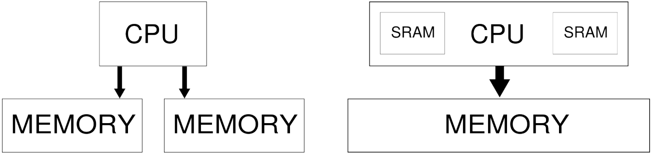
NUMA for Embedded Systems



The existing NUMA interfaces in Linux are...

- architecture-independent.
- well-established.

NUMA for Embedded Systems



The existing NUMA interfaces in Linux are...

- architecture-independent.
- well-established.

...and they allow us to:

- Select node memory to back file data with.
- Select node for process memory ranges.

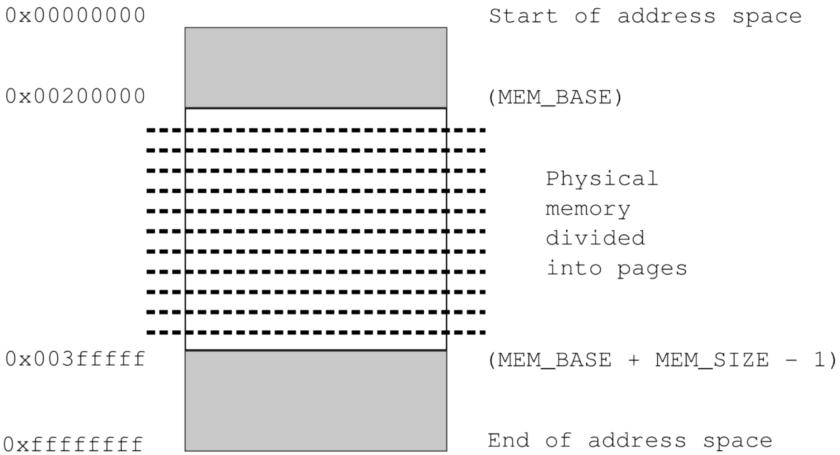
NUMA for Embedded Systems

“...superh is starting to use NUMA now, due to varying access times of various sorts of memory. one can envisage other embedded setups doing that”

- Andrew Morton, on linux-mm

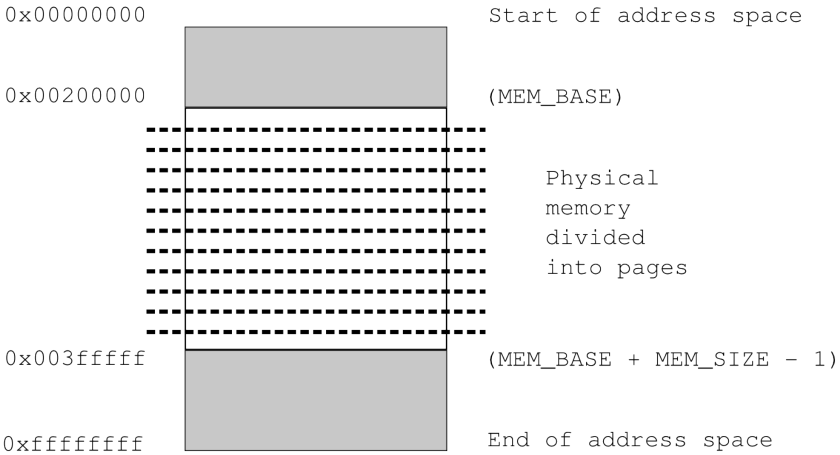
Outline

Physical Memory



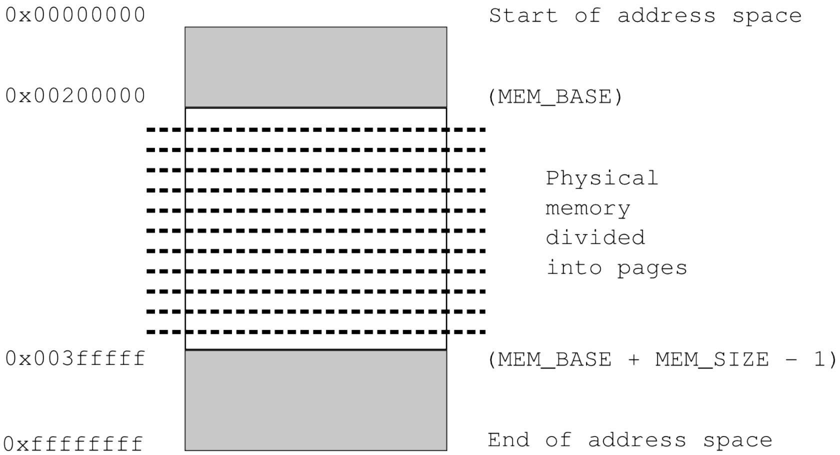
struct page mem_map[]

Physical Memory



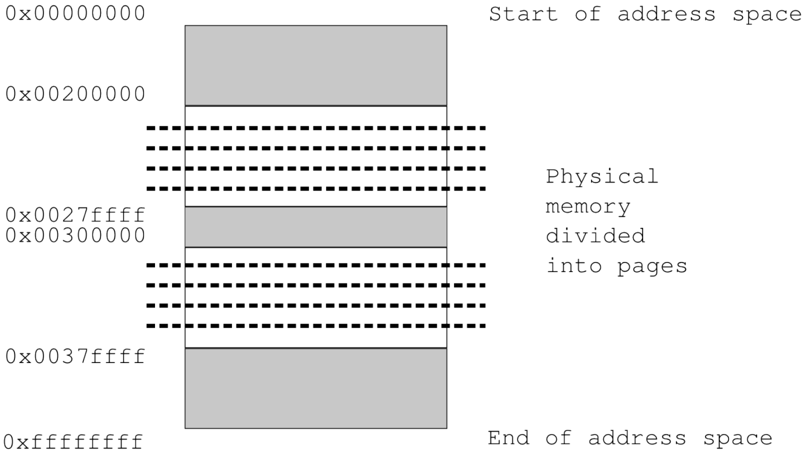
PFN - Page Frame Number

Memory Models



One contiguous range of memory: `CONFIG_FLATMEM`

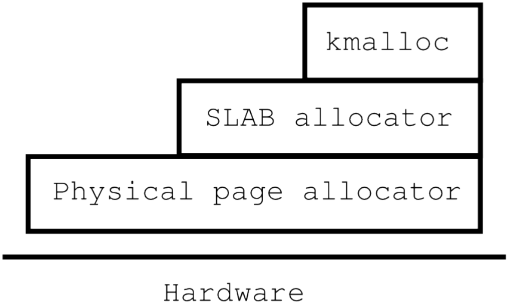
Memory Models



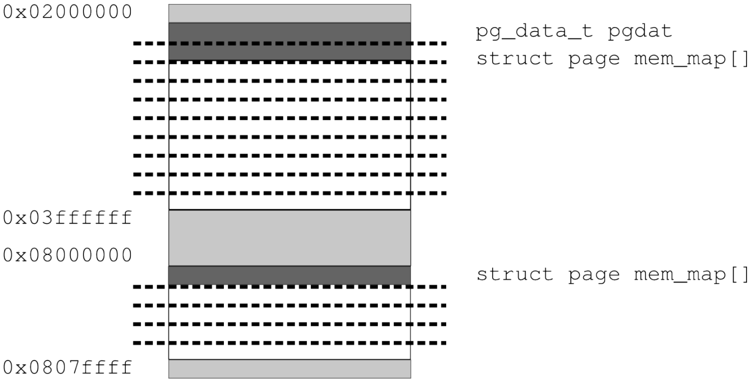
More than one contiguous range of memory:

`CONFIG_SPARSEMEM` or `CONFIG_DISCONTIGMEM`

Physically Contiguous Allocators



Physical Page Allocator



Physical Page Allocator

Binary Buddy Allocator Algorithm

```
struct page *alloc_pages(gfp_t, unsigned int)
void __free_pages(struct page *, unsigned int)
```

Order N allocations:

- Order 0 -> 1 page.
- Order 1 -> 2 pages.
- Order 2 -> 4 pages.
- ...
- Order N -> 2 ^N pages.

SLAB Allocator

A caching object-based allocator.

Create and destroy a cache of objects:

```
struct kmem_cache *kmem_cache_create(...)
void kmem_cache_destroy(struct kmem_cache *)
```

Allocate and free objects using the cache:

```
void *kmem_cache_alloc(struct kmem_cache *, gfp_t))
void kmem_cache_free(struct kmem_cache *, void *)
```

SLAB Allocator

A caching object-based allocator.

Create and destroy a cache of objects:

```
struct kmem_cache *kmem_cache_create(...)
void kmem_cache_destroy(struct kmem_cache *)
```

Allocate and free objects using the cache:

```
void *kmem_cache_alloc(struct kmem_cache *, gfp_t))
void kmem_cache_free(struct kmem_cache *, void *)
```

`/proc/slabinfo` provides statistics.

SLAB Allocators (2.6.23)

SLAB (1996)

- linux/mm/slab.c, ~4500 lines, `CONFIG_SLAB`
- Default allocator, per-cpu and per-node data.

SLOB (2003)

- linux/mm/slob.c, ~600 lines, `CONFIG_SLOB`
- Simple and small, but with single free list.

SLUB (2007)

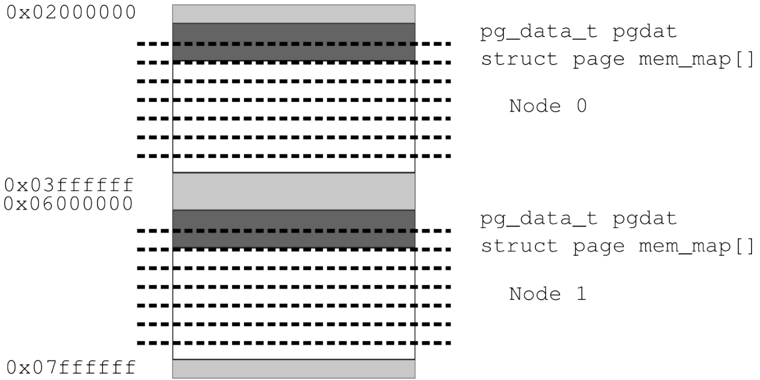
- linux/mm/slub.c, ~4100 lines, `CONFIG_SLUB`
- Unqueued allocator, minimizes cache line usage.

kmalloc and kfree

A thin multi-purpose layer on top of the SLAB allocator.

```
void *kmalloc(size_t, gfp_t)
void kfree(const void *)
```

Multiple Nodes (CONFIG_NUMA=y)



CONFIG_SPARSEMEM or CONFIG_DISCONTIGMEM required.

Asymmetric NUMA

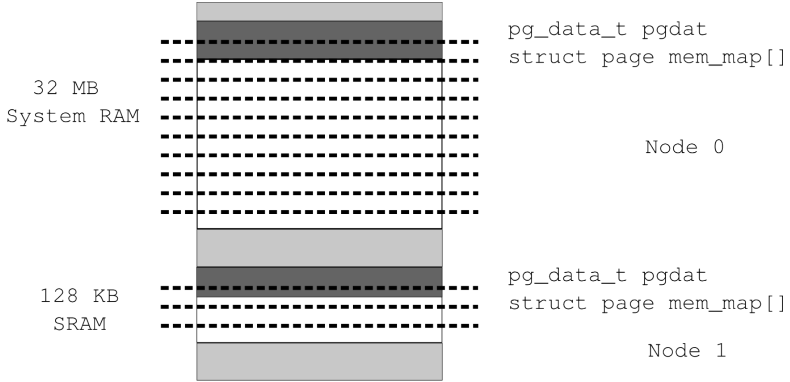
Physical Page Allocator

- Kernel allocates from all nodes during initialization.
  - 16MB cut-off added.
- Kernel defaults to node-local allocations during run time.
  - Use node 0 for System RAM.

SLAB Allocators

- SLAB
- SLUB
  - Requires patches to exclude small nodes.
- SLOB
  - Low overhead, preferred SLAB Allocator for Asymmetric NUMA.
  - Primitive locking - possible performance issues for SMP systems.

Asymmetric NUMA



Asymmetric NUMA - Overhead

```
/ # cat /sys/devices/system/node/node1/meminfo

Node 1 MemTotal:  128 kB
Node 1 MemFree:  72 kB
Node 1 MemUsed:  56 kB
Node 1 Active:  0 kB
...
```

Outline

What are Memory Policies?

Memory Policies control the behavior of the memory allocator.

What are Memory Policies?

Memory Policies control the behavior of the memory allocator.

They allow us to adjust...

- Per-process memory allocation policy.
- Memory allocation policies for ranges of process memory.
- Memory Policies for file systems such as tmpfs.

What are Memory Policies?

Memory Policies control the behavior of the memory allocator.

They allow us to adjust...

- Per-process memory allocation policy.
- Memory allocation policies for ranges of process memory.
- Memory Policies for file systems such as tmpfs.

Using Memory Policies we can...

- Select which nodes to allocate from.
- Chose between optimizing for latency or bandwidth.
- Allow or disallow fallback allocation from other nodes.

Memory Policies - Modes

MPOL_DEFAULT

- Prioritize local node over remote ones.

MPOL_BIND

- Allocate from specified nodes *only*, one by one.

MPOL_INTERLEAVE

- Spread out allocations over specified nodes.

MPOL_PREFERRED

- Allocate from specified nodes, one by one.

Memory Policies

Per-process control of memory allocations:

```
int set_mempolicy(int mode, unsigned long *nodemask,
unsigned long maxnode)
```

Control ranges of process memory:

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

Memory Policies

Per-process control of memory allocations:

```
int set_mempolicy(int mode, unsigned long *nodemask,
unsigned long maxnode)
```

Control ranges of process memory:

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

man 2 set_mempolicy
man 2 mbind
man 3 numa - libnuma by Andi Kleen

tmpfs and cpusets

tmpfs is a file system which keeps all files in memory. Mount options can be used to select memory policy.

```
mount -t tmpfs -o mpol=bind:0,2 tmpfs /mytmpfs
```

See linux/Documentation/filesystems/tmpfs.txt for more information.

tmpfs and cpusets

tmpfs is a file system which keeps all files in memory. Mount options can be used to select memory policy.

```
mount -t tmpfs -o mpol=bind:0,2 tmpfs /mytmpfs
```

See linux/Documentation/filesystems/tmpfs.txt for more information.

cpusets is kernel mechanism that assigns processes to a subset of all available processors and memory nodes.

More information available in linux/Documentation/cpusets.txt

Summary

- Asymmetric NUMA brings NUMA to the embedded space.
- Well-established interfaces outweights the added overhead.