Hierarchical NUMA

Anshuman Khandual

khandual@linux.vnet.ibm.com
Linux Technology Centre
IBM
Contents

- NUMA Systems
- NUMA in Memory Management
- Coherent Device Memory
- Existing CDM Solutions
- Existing NUMA Design Limitations
- Need for Attribute based NUMA
- Hierarchical NUMA Proposal
- Conclusion
• NUMA Systems

• NUMA in Memory Management

• Coherent Device Memory

• Existing CDM Solutions

• Existing NUMA Design Limitations

• Need for Attribute based NUMA

• Hierarchical NUMA Proposal

• Conclusion
Why NUMA

- Some workloads benefit from a single application image running on a big NUMA system
- Accessing an extensive set of processor cores and large shared memory for it's working set
- Require a **large working memory set** to be processed in **multiple chunks** through **multiple threads**
- Primarily **enterprise** and **HPC** workloads fall in this category

  For example, the following class of applications fall in that category

  - In Memory DB
  - ERP
  - CRM
  - Business Intelligence
  - Scientific Research
  - Virtualization

Example of some commercially available NUMA systems

- IBM POWER8 E880
- Dell PowerEdge R930
- HP DL980
- Supermicro SYS-7088B-TR4FT
• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory
• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
NUMA Properties

- Systems implement NUMA through various types of NUMA interconnects
- NUMA interconnects can be with or without any data routers
- From CPU's perspective, distance is just access latency
- Latency depends on NUMA interconnect, number of hops, system bus speed etc
- Represented as a **distance** factor, how far is the memory from a given CPU
- Platform firmware must provide `node_distance[MAX_NUMNODES][MAX_NUMNODES]`
- Arch code processes ACPI (X86) or DT (PowerPC) to fetch distance information from firmware
- Arch code needs to export `node_distance(a, b)` for the core kernel to use every where else

NUMA initialization on POWER

```c
node_distance()
    __node_distance()
        distance_lookup_table[MAX_NUMNODES][MAX_DISTANCE_REF_POINTS]
        associativity(distance_ref_points[])
```

NUMA initialization on X86

```c
node_distance()
    __node_distance()
        numa_distance[]
```
NUMA Memory Representation

Arch NUMA Initialization

Platform Firmware

Memory Hardware

Scanning DT or ACPI

Probing the hardware

Memblock

Buddy system

Node 0
Node 1
Node 2
Node ..
Node N

PGlist data

PGlist data

PGlist data

PGlist data

Zone movable

Zone normal

Zone dma32

Zone dma

Free area[0] → free list

Free area[n] → free list

Migrate unmoveable

Migrate movable

Migrate reclaimable

Migrate pctypes

Migrate highatomic

Migrate cma

Page

Page

Page

Page

Page

Page

Page

Page

Page

Page
• NUMA Systems

• NUMA in Memory Management

• Coherent Device Memory

• Existing CDM Solutions

• Existing NUMA Design Limitations

• Need for Attribute based NUMA

• Hierarchical NUMA Proposal

• Conclusion
Coherent Device Memory (CDM)

- Non system RAM memory which can be accessed coherently from the CPU
- They are similar to system RAM in many characteristics but are still different
- They are comparable in size with system RAM
- They offer specialized functions for applications and drivers to use
- They may be accessed by other forms of device compute

**Existing CDM devices**

- POWER9 processor accessing NVIDIA Volta GPU over NVLink2
- Persistent Memory (NVDIMM) on both PowerPC and X86 systems
- Intel MCDRAM is a High Bandwidth Memory (HBM) on the chip

**Different DRAM**

- Existing systems can have DDR2, DDR3 and DDR4, some times in the same system
- DDR5 will be coming in two forms (low power format and standard one)
- They can have different latency, bandwidth and power consumption properties
• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory

• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
ZONE_DEVICE

- Created a brand new zone for persistent memory or NVDIMM
- Developed to represent NVDIMM or persistent memory inside the kernel
- It helps support DAX file system on those devices
- Persistent memory is fast enough, page cache layer is not required
- These memory can have their own **struct pages**, giving it access to some core MM features
- It attempts to integrate NVDIMM in the kernel while still keeping its own specializations
Heterogeneous Memory Management (HMM)

- Developed to represent **unaddressable** device memory as coherent.
- Creates a new memory type in ZONE_DEVICE for **unaddressable** memory.
- Applicable to device memory which does not support coherency, hence kernel provides coherency.
- Kernel achieves the coherency with mirrored page tables and on demand migrations during access.
- It helps map completely **unaddressable** memory in user space as well.

![Diagram of memory management](image-url)
HMM CDM (Addressable Coherent Memory)

- Modified version of HMM to support devices with coherent memory
- Creates yet another memory type for coherent device memory in ZONE_DEVICE
- Drops the mirrored page table as the hardware already has the coherency support
- Migration interfaces changed to accommodate this new coherent ZONE_DEVICE memory type
CDM NUMA

- Represents CDM as NUMA node to achieve seamless integration
- Device memory remains in ZONE_MOVABLE to avoid kernel allocation into CDM
- Implicit allocation from user space should be avoided to the CDM
- Zonelists (FALLBACK and NOFALLBACK) format changed to achieve the desired isolation
- Applications can have CDM memory through mmap(MPOL_BIND, ……) interface explicitly
- mbind() implementation got changed to walk new zonelists containing CDM zones
Intel MCDRAM

- Multi Channel DRAM (MCDRAM)
- High Bandwidth Memory (~4x compared to DRAM) but low capacity (Upto 16GB)
- Packaged inside Xeon Phi X200 chip (Knights Landing)
- Can be configured as a third level cache or as a distinct NUMA node
- Data placement on MCDRAM happens through numactl (mbind), memkind and AutoHBW libraries
- This signifies how application can use specialized memory for it's advantage

Further Reading

• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory
• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
Existing NUMA Limitations (Representing CDM)

- Distance, hence latency is not the only memory attribute
- Does not recognize other properties like bandwidth, reliability, power consumption etc
- There can be different types of memory at the same distance (may be connected by different interfaces)
- So what should be the properties of memory?

- Latency (Speed of random access)
- Bandwidth (Speed of contiguous data stream processing)
- Reliability (Probable rate of memory failure)
- Power consumption (Power consumption in holding the data in memory)
- Density (Compact allocation, power saving, anything else)
• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory
• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
Need for Attribute Based NUMA

- All existing solutions attempt to make core MM understand the very concept of special memory
- ZONE_DEVICE, HMM, HMM CDM
  - Does not provide explicit user interface for CDM allocation
  - Driver managed and hidden from userspace
  - Being on ZONE_DEVICE, it is already isolated from standard allocation paths
- NUMA CDM
  - Provides explicit user interface for CDM allocation
  - Does invasive changes in kernel to achieve isolation, reclaim and compaction
- For any future CDM memory, attempts will be made to change core kernel to suit its particular needs
- Core MM does not differentiate between **various types** of memory with **different attributes**
- Hence, there is a need for unified representation of different kinds of memory in the kernel
- Traditional distance based NUMA has to change to accommodate these new kinds of memory
• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory
• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
Hierarchical NUMA Proposal

- Changes to physical memory representation
  - Changes to memblock
  - Changes to buddy System
- Changes to virtual memory representation
  - New system calls
  - Changes to VMA
  - Changes to memory policy
- Changes to memory allocation
  - New GFP flags
  - Changes to __alloc_pages_nodemask() 
  - Changes to get_pages_from_freelist() 
  - Changes to __rmqueue_smallest() 
  - New zone_to_area_mattr() function 
- Changes to other memory functions
Physical Memory Representation

Cumulative mattr value is the key for the RB Tree
Memblock

struct memblock_region {
    phys_addr_t base;
    phys_addr_t size;
    unsigned long flags; /* VMA flags */
    unsigned long mattr; /* Memory attribute flags */
#ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
    int nid;
#endif
};

MEM_LATENCY_MASK 0x000000000000000F    /* Memory latency */
MEM_BANDWIDTH_MASK 0x00000000000000F0    /* Memory bandwidth */
MEM_RELIABILITY_MASK 0x0000000000000F00    /* Memory reliability */
MEM_POWERVALUE_MASK 0x000000000000F000    /* Memory power consumption */
MEM_DEVICECOMP_MASK 0x00000000000F0000    /* Memory device compute */
MEM_DENSITY_MASK 0x0000000000F00000    /* Memory density */

MEM_LATENCY_SHIFT 0
MEM_BANDWIDTH_SHIFT 4
MEM_RELIABILITY_SHIFT 8
MEM_POWERVALUE_SHIFT 12
MEM_DEVICECOMP_SHIFT 16
MEM_DENSITY_SHIFT 20
Buddy System (PGLIST_DATA)

```
struct cdm_node {
    struct rb_node node; /* Attached to rb_cdm */
    unsigned long mattr; /* Memblock attributes */
    struct free_area free_area[MAX_ORDER]; /* CDM buddy system */
}
```

```
struct zone {
    struct free_area free_area[MAX_ORDER]; /* Contains DRAM */
    struct rb_root rb_cdm; /* Contains CDM */
    unsigned long nr_cdm; /* Total CDM nodes on the zone */
    struct cdm_node *cdm_ind[MAX_PROPERTY][MAX_CDM]; /* Search cache */
    struct cdm_node *cdm_cum[MAX_CDM]; /* Search cache */
}
```

- zone \( \rightarrow \) \((rb\_cdm, \text{nr}\_cdm, \text{cdm}\_\text{ind}, \text{cdm}\_\text{cum})\) gets updated during boot and memory hotplug

- Contiguous memory\_regions need to have same memory attribute to be a single CDM node

- A zone can contain multiple CDM nodes with different properties forming a RB tree

- Cumulative value of ‘mattr’ is the key while inserting the node into RB tree

- Cumulative value adds up all memory attribute values for the CDM

- CDM nodes remain balanced in the RB tree and forms a 'hierarchical' system
Virtual Memory Representation

New System Calls

- get_mempolicy_mattr
- set_mempolicy_mattr
- mmap_attr
- mbind_attr
- madvise_mattr
- move_pages_mattr

mempolicy

task_struct

mm_struct

VMA

vm_mattr

mattr

mempolicy

Applicable mempolicy

Influences memory allocation for the process
Virtual Memory Area (VMA)

```c
struct vma_area_struct {
    ...
    unsigned long vm_flags; /* Existing VMA flags */
    unsigned long vm_mattr; /* Contains memory attribute flags */
    ...
}
```

<table>
<thead>
<tr>
<th>Mask</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_LATENCY_MASK</td>
<td>0x000000000000000F</td>
<td>Memory latency</td>
</tr>
<tr>
<td>VM_BANDWIDTH_MASK</td>
<td>0x000000000000000F0</td>
<td>Memory bandwidth</td>
</tr>
<tr>
<td>VM_RELIABILITY_MASK</td>
<td>0x000000000000000F00</td>
<td>Memory reliability</td>
</tr>
<tr>
<td>VM_POWERVALUE_MASK</td>
<td>0x000000000000000F000</td>
<td>Memory power consumption</td>
</tr>
<tr>
<td>VMDEVICECOMP_MASK</td>
<td>0x000000000000000F0000</td>
<td>Memory device compute</td>
</tr>
<tr>
<td>VM_DENSITY_MASK</td>
<td>0x000000000000000F000000</td>
<td>Memory density</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Shift</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM_LATENCY_SHIFT</td>
<td>0</td>
</tr>
<tr>
<td>VM_BANDWIDTH_SHIFT</td>
<td>4</td>
</tr>
<tr>
<td>VM_RELIABILITY_SHIFT</td>
<td>8</td>
</tr>
<tr>
<td>VM_POWERVALUE_SHIFT</td>
<td>12</td>
</tr>
<tr>
<td>VMDEVICECOMP_SHIFT</td>
<td>16</td>
</tr>
<tr>
<td>VM_DENSITY_SHIFT</td>
<td>20</td>
</tr>
</tbody>
</table>

- VMA memory attributes look similar to that of memblock? They are but
- VMA memory attributes will be ABI where as memblock attributes can change later
Memory Policy

```c
struct mempolicy {
    atomic_t refcnt;
    unsigned short mode;
    unsigned short flags;
    unsigned long mattr; /* Contains VMA mattr flags */
    union {
        short preferred_node;
        nodemask_t nodes;
    } v;
    union {
        nodemask_t cpuset_mems_allowed;
        nodemask_t user_nodemask;
    } w;
};
```

- Memory policy structure can belong to either to a VMA or to the entire task
- Why copy VMA flags into memory policy? It improves performance during page fault
- During each page fault, mmap_sem need not be held to read vma→vm_mattr
# GFP Flags

<table>
<thead>
<tr>
<th>LATENCY</th>
<th>BANDWIDTH</th>
<th>RELIABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>__GFP_LATENCY_1</td>
<td>__GFP_BANDWIDTH_1</td>
<td>__GFP_RELIABILITY_1</td>
</tr>
<tr>
<td>__GFP_LATENCY_2</td>
<td>__GFP_BANDWIDTH_2</td>
<td>__GFP_RELIABILITY_2</td>
</tr>
<tr>
<td>__GFP_LATENCY_3</td>
<td>__GFP_BANDWIDTH_3</td>
<td>__GFP_RELIABILITY_3</td>
</tr>
<tr>
<td>__GFP_LATENCY_4</td>
<td>__GFP_BANDWIDTH_4</td>
<td>__GFP_RELIABILITY_4</td>
</tr>
<tr>
<td>__GFP_LATENCY_5</td>
<td>__GFP_BANDWIDTH_5</td>
<td>__GFP_RELIABILITY_5</td>
</tr>
<tr>
<td>__GFP_LATENCY_6</td>
<td>__GFP_BANDWIDTH_6</td>
<td>__GFP_RELIABILITY_6</td>
</tr>
<tr>
<td>__GFP_LATENCY_7</td>
<td>__GFP_BANDWIDTH_7</td>
<td>__GFP_RELIABILITY_7</td>
</tr>
<tr>
<td>__GFP_LATENCY_8</td>
<td>__GFP_BANDWIDTH_8</td>
<td>__GFP_RELIABILITY_8</td>
</tr>
<tr>
<td>__GFP_LATENCY_9</td>
<td>__GFP_BANDWIDTH_9</td>
<td>__GFP_RELIABILITY_9</td>
</tr>
<tr>
<td>__GFP_LATENCY_10</td>
<td>__GFP_BANDWIDTH_10</td>
<td>__GFP_RELIABILITY_10</td>
</tr>
<tr>
<td>__GFP_LATENCY_11</td>
<td>__GFP_BANDWIDTH_11</td>
<td>__GFP_RELIABILITY_11</td>
</tr>
<tr>
<td>__GFP_LATENCY_12</td>
<td>__GFP_BANDWIDTH_12</td>
<td>__GFP_RELIABILITY_12</td>
</tr>
<tr>
<td>__GFP_LATENCY_13</td>
<td>__GFP_BANDWIDTH_13</td>
<td>__GFP_RELIABILITY_13</td>
</tr>
<tr>
<td>__GFP_LATENCY_14</td>
<td>__GFP_BANDWIDTH_14</td>
<td>__GFP_RELIABILITY_14</td>
</tr>
<tr>
<td>__GFP_LATENCY_15</td>
<td>__GFP_BANDWIDTH_15</td>
<td>__GFP_RELIABILITY_15</td>
</tr>
<tr>
<td>__GFP_LATENCY_16</td>
<td>__GFP_BANDWIDTH_16</td>
<td>__GFP_RELIABILITY_16</td>
</tr>
</tbody>
</table>
GFP Flags (Cont..)

<table>
<thead>
<tr>
<th>POWER VALUE</th>
<th>DEVICE COMPUTE</th>
<th>DENSITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>__GFP_POWERVALUE_1</td>
<td>__GFP_DEVICECOMP_1</td>
<td>__GFP_DENSITY_1</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_2</td>
<td>__GFP_DEVICECOMP_2</td>
<td>__GFP_DENSITY_2</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_3</td>
<td>__GFP_DEVICECOMP_3</td>
<td>__GFP_DENSITY_3</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_4</td>
<td>__GFP_DEVICECOMP_4</td>
<td>__GFP_DENSITY_4</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_5</td>
<td>__GFP_DEVICECOMP_5</td>
<td>__GFP_DENSITY_5</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_6</td>
<td>__GFP_DEVICECOMP_6</td>
<td>__GFP_DENSITY_6</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_7</td>
<td>__GFP_DEVICECOMP_7</td>
<td>__GFP_DENSITY_7</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_8</td>
<td>__GFP_DEVICECOMP_8</td>
<td>__GFP_DENSITY_8</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_9</td>
<td>__GFP DEVICECOMP_9</td>
<td>__GFP_DENSITY_9</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_10</td>
<td>__GFP_DEVICECOMP_10</td>
<td>__GFP_DENSITY_10</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_11</td>
<td>__GFP_DEVICECOMP_11</td>
<td>__GFP_DENSITY_11</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_12</td>
<td>__GFP_DEVICECOMP_12</td>
<td>__GFP_DENSITY_12</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_13</td>
<td>__GFP_DEVICECOMP_13</td>
<td>__GFP_DENSITY_13</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_14</td>
<td>__GFP_DEVICECOMP_14</td>
<td>__GFP_DENSITY_14</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_15</td>
<td>__GFP_DEVICECOMP_15</td>
<td>__GFP_DENSITY_15</td>
</tr>
<tr>
<td>__GFP_POWERVALUE_16</td>
<td>__GFP_DEVICECOMP_16</td>
<td>__GFP_DENSITY_16</td>
</tr>
</tbody>
</table>
Memory Allocation

User Allocation

GFP_HIGHUSER_MOVABLE

alloc_pages_vma

gfp_flags |= mem_attr_to_gfp(pol->mattr)

Kernel Allocation

Any GFP flag as required

alloc_pages

struct alloc_context {
    struct zonelist *zonelist;
    nodemask_t *nodemask;
    struct zoneref *preferred_zoneref;
    int migratetype;
    enum zone_type high_pageidx;
    bool spread_dirty_pages;
    unsigned long mattr;
};

ac->mattr = gfp_flags & GFP_MATTR_MASK

gfp_flags &= ~GFP_MATTR_MASK

get_page_from_freelist

ac->mattr goes into all down functions here onwards

gfp_flags now does not have memory attributes
Memory Allocation (Cont..)

\[\text{ac} \rightarrow \text{mattr}\]

get_page_from_freelist

rmqueue

\[\text{__rmqueue}\]

\[\text{__rmqueue\_smallest}\]

If (mattr) \rightarrow \text{then CDM}

\[\text{area} = \&(\text{zone}\rightarrow\text{free\_area}[\text{current\_order}])\]

If (!mattr) \rightarrow \text{then DRAM}

\[\text{area} = \text{zone\_to\_area\_mattr}(\text{zone}, \text{mattr}, \text{current\_order})\]

\[\text{page} = \text{list\_first\_entry\_or\_null}(\text{area}\rightarrow\text{free\_list}[\text{migratetype}], \text{struct page}, \text{lru})\]
Memory Allocation (Cont..)

area = zone_to_area_mattr(zone, mattr, current_order)

- If mattr contains a single attribute, match the exact or closest of the same attribute
- If mattr contains multiple attributes, match the exact or closest match for cumulative value
Memory Attribute Matching

Matching may not be straight forward

There may not be a straight match

Requested memory attributes

CDM  CDM  CDM  CDM

Matching may not be straight forward

Single Attribute Request

Multiple Attribute Request

All Attribute Request

Exact or closest match on single attribute

Walk RB tree again

Exact or closest match on cumulative attribute for applicable attributes as per request

Exact or closest match on cumulative attribute

Refer zone → cdm_ind[ ][ ]

Refer zone → cdm_cum[ ][ ]
Other Memory Functions

Memory allocation sub functions

- Compaction
  - struct compact_control must contain memory attributes
  - zone_to_area_mattr() is called to find appropriate free_area[order] in a given zone

- Reclaim
  - struct scan_control must contain memory attributes
  - zone_to_area_mattr() is called to find appropriate free_area[order] in a given zone
• NUMA Systems
• NUMA in Memory Management
• Coherent Device Memory
• Existing CDM Solutions
• Existing NUMA Design Limitations
• Need for Attribute based NUMA
• Hierarchical NUMA Proposal
• Conclusion
Conclusion

- There may be other possible solutions with some pros and cons
  - CDM and DRAM zones with memory attributes
  - CDM and DRAM nodes (NODE_DATA) with memory attributes
  - CDM and DRAM pageblocks (memory attribute types in line with migration types)

- The proposed idea here may not be the best possible solution
- The community should debate this proposal to reach at an optimal solution
- Industry trends suggest, going forward CDM will be provided my multiple vendors and OEMs
- Looking forward to build interest in the community to work towards finding a solution
  - Linux kernel should be ready! Memory HW technologies are coming in fast :)
Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of the employers (IBM Corporation).

- IBM and IBM (Logo) are trademarks or registered trademarks of International Business Machines in United States and/or other countries.

- Linux is a registered trademark of Linux Torvalds.

- Other company, product and service names may be trademarks or service marks of others.
References

- http://www-03.ibm.com/systems/in/power/hardware/e880/

- http://www-03.ibm.com/systems/in/resources/systems_power_hardware_e880_580x326.png
- https://www.nextplatform.com/2015/05/04/ibm-scales-up-power8-iron-targets-in-memory/
Acronyms

VMA  Virtual Memory Area
PFN  Page Frame Number
PGDAT  PGLIST_DATA

SMP  Symmetric Multi Processing
NUMA  Non Uniform Memory Access
HNUMA  Hierarchical NUMA
DRAM  Dynamic Random Access Memory
CDM  Coherent Device Memory

HPC  High Performance Computing
ERP  Enterprise Resource Planning
CRM  Customer Resource Management
RB  Red Black (Tree)
Thank You
Backup
NUMA Evolution
Symmetric Multi Processing (SMP)

- Memory & compute intensive workloads benefit with **single OS image** on **big SMP systems**
- They utilize all cores & memory bandwidth with **shared data structures**
- They include enterprise apps like DB, CRM and HPC
- There has always been a need for bigger SMP systems for these kind of applications
- Building bigger SMP system with **Single System Bus** is complex and expensive
- **Single System Bus** design also has scalability limitations
- Then how computer system can scale up?

![Diagram of Standard SMP system](image-url)
Non Uniform Memory Access (NUMA)

- **NUMA** is a **design choice** to avoid further **complicating** (expensive) **Single System Bus** design.
- **Single System Bus** SMP design always had **scalability problems**.
- **NUMA** is a **logical follow up** from Single Bus SMP design to accommodate more cores and memory.
- **NUMA** has **more than one system bus** connected through high speed **NUMA inter-connects**.
Existing NUMA Data Structures
Memblock

- CONFIG_HAVE_MEMBLOCK_NODE_MAP decides if memblock has **node information**
- Memblocks are **basic blocks of memory regions** received from platform firmware
- memblock_add() adds a memory region into the system
- memblock_add_node() or memblock_set_node() adds **node information** into the memblock
- From memblock, NUMA information flows up the stack in the kernel

```c
struct memblock_region {
    phys_addr_t base;
    phys_addr_t size;
    unsigned long flags;
    ifdef CONFIG_HAVE_MEMBLOCK_NODE_MAP
        int nid;
    endif
};
```
Memory Section

- Contains mem_map information along with pageblock flags
- `mem_section→section_mem_map` contains memmap along with pfn
- `mem_section→pageblock_flags` contain flags for pageblock_nr_pages blocks
- Pageblock flag information is stored in `zone→pageblock_flags` (CONFIG_SPARSEMEM)

```c
struct mem_section {
    /*
    * This is, logically, a pointer to an array of struct
    * pages. However, it is stored with some other magic.
    * (see sparse.c::sparse_init_one_section())
    *
    * Additionally during early boot we encode node id of
    * the location of the section here to guide allocation.
    * (see sparse.c::memory_present())
    *
    * Making it a UL at least makes someone do a cast
    * before using it wrong.
    */
    unsigned long section_mem_map;

    /* See declaration of similar field in struct zone */
    unsigned long *pageblock_flags;

    #ifdef CONFIG_PAGE_EXTENSION
    /*
    * If SPARSEMEM, pgdat doesn't have page_ext pointer. We use
    * section. (see page_ext.h about this.)
    */
    struct page_ext *page_ext;
    unsigned long pad;
    #endif

    /* WARNING: mem_section must be a power-of-2 in size for the
    * calculation and use of SECTION_ROOT_MASK to make sense.
    */
};
```
Page

- page → flags contains NUMA information as well
- page_to_pfn() and pfn_to_page() mapping has to be fast (used in hot paths)
- If NODE_NOT_IN_PAGE_FLAGS is defined, page flags will not have NUMA information
- Instead it will be fetched from mem_section based table (which is always initialized)
- But page flags based page → nid direct mapping is always faster

```c
#ifdef NODE_NOT_IN_PAGE_FLAGS
extern int page_to_nid(const struct page *page);
#else
static inline int page_to_nid(const struct page *page)
{
    return (page->flags >> NODES_PGSHIFT) & NODES_MASK;
}
#endif

int page_to_nid(const struct page *page)
{
    return section_to_node_table[page_to_section(page)];
}
EXPORT_SYMBOL(page_to_nid);
```
Pglist_data

- NUMA organization of memory starts at `struct pglist_data` (every NUMA node got one)
- Always fetched by `NODE_DATA(nid)` from the core MM
- Contains all zones `struct zone node_zones[MAX_NR_ZONES]`
- Contains both the zonelists `struct zonelist node_zonelists[MAX_ZONELISTS]`
- On POWER
  - `struct pglist_data *node_data[MAX_NUMNODES] → NODE_DATA(node)`
  - `initmem_init() → setup_node_data()`
- On X86
  - `struct pglist_data *node_data[MAX_NUMNODES] → NODE_DATA(node)`
  - `numa_register_memblks() → alloc_node_data()`
Zonelist

- Zonelists are used during memory allocation
- Helps in navigating NUMA for any allocation, should happen either for a node strictly or with fall backs
- Part of struct pglist_data (pgdat → node_zonelists[MAX_ZONELISTS])
- ZONELIST_FALLBACK contains zones of all nodes with a fallback order
- ZONELIST_NOFALLBACK contains zones of the same node, accessed only through __GFP_THISNODE

```c
/*
 * This struct contains information about a zone in a zonelist. It is stored
 * here to avoid dereferences into large structures and lookups of tables
 */
struct zoneref {
    struct zone *zone; /* Pointer to actual zone */
    int zone_idx;     /* zone_idx(zoneref->zone) */
};

/*
 * One allocation request operates on a zonelist. A zonelist
 * is a list of zones, the first one is the 'goal' of the
 * allocation, the other zones are fallback zones, in decreasing
 * priority.
 * To speed the reading of the zonelist, the zonerefs contain the zone index
 * of the entry being read. Helper functions to access information given
 * a struct zoneref are
 * zonelist_zone() - Return the struct zone * for an entry in zonerefs
 * zonelist_zone_idx() - Return the index of the zone for an entry
 * zonelist_node_idx() - Return the index of the node for an entry
 */
struct zonelist {
    struct zoneref _zonerefs[MAX_ZONES_PER_ZONELIST + 1];
};
```
Memory NUMA Representation

NODE 0  NODE 1  NODE 2  NODE ..  NODE N
pglist_data  pglist_data  pglist_data  ............  pglist_data

ZONE_MOVABLE  ZONE_NORMAL  ZONE_DMA32  ZONE_DMA

free_area[0]  free_list
free_area[1]  free_list
free_area[2]  free_list
free_area[n]  free_list
free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page

ZONE_DMA

free_area[1]  free_list
free_area[2]  free_list
free_area[n]  free_list
free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page

ZONE_NORMAL

free_area[1]  free_list
free_area[2]  free_list
free_area[n]  free_list
free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page

ZONE_DMA32

free_area[1]  free_list
free_area[2]  free_list
free_area[n]  free_list
free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page

free_area[2]  free_list
free_area[n]  free_list
free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page

free_area[MAX_ORDER]  free_list

MIGRATE_UNMOVEABLE  page  page  ......  page
MIGRATE_MOVEABLE  page  page  ......  page
MIGRATE_RECLAIMABLE  page  page  ......  page
MIGRATE_PCTYPES  page  page  ......  page
MIGRATE_HIGHATOMIC  page  page  ......  page
MIGRATE_CMA  page  page  ......  page
Sample Commercial NUMA Systems
IBM POWER8 E880
DELL Power Edge R930
HP DL980
Supermicro 7088B-TR4FT

SuperServer SYS-7088B-TR4FT
(Angled View – System)

KVM Port
Power Button
1 of 8 CPU Modules
Memory Fault LED

12x 2.5" Hot-Swap SAS3/SATA3 Drive Bays
(2 storage modules)

© Super Micro Computer, Inc. Information in this document is subject to change without notice.
Intel Xeon E7400 NUMA Diagram
Relevant Community Discussions

NUMA CDM
- https://lkml.org/lkml/2016/10/24/19 (RFC V1)
- https://lkml.org/lkml/2016/11/22/339 (RFC V1.2)
- https://lkml.org/lkml/2017/1/29/198 (RFC V2)
- https://lkml.org/lkml/2017/2/8/329 (PATCH V1)
- https://lwn.net/Articles/720380/ (RFC from Balbir Singh)

LSFMM 2017
- https://lwn.net/Articles/717601/ (HMM and CDM)

HMM CDM
- https://lkml.org/lkml/2017/4/7/638 (RFC V1)
- https://lwn.net/Articles/725412/ (PATCH V2)
- https://lwn.net/Articles/727114/ (PATCH V3)
- https://lwn.net/Articles/727692/ (PATCH V4)

HMM
- https://lwn.net/Articles/726691/ (PATCH V24)
- https://lwn.net/Articles/731259/ (PATCH V25)