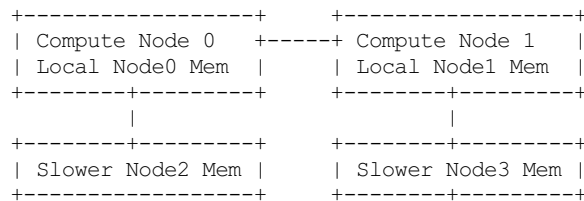# NUMA Locality

Some platforms may have multiple types of memory attached to a compute node. These disparate memory ranges may share some characteristics, such as CPU cache coherence, but may have different performance. For example, different media types and buses affect bandwidth and latency.

A system supports such heterogeneous memory by grouping each memory type under different domains, or "nodes", based on locality and performance characteristics. Some memory may share the same node as a CPU, and others are provided as memory only nodes. While memory only nodes do not provide CPUs, they may still be local to one or more compute nodes relative to other nodes. The following diagram shows one such example of two compute nodes with local memory and a memory only node for each of compute node:

```
+------------------+       +------------------+
| Compute Node 0   +-----+ Compute Node 1   |
| Local Node0 Mem  |       | Local Node1 Mem  |
+--------+---------+       +--------+---------+
         |                          |
+--------+---------+       +--------+---------+
| Slower Node2 Mem |       | Slower Node3 Mem |
+------------------+       +--------+---------+
```

A "memory initiator" is a node containing one or more devices such as CPUs or separate memory I/O devices that can initiate memory requests. A "memory target" is a node containing one or more physical address ranges accessible from one or more memory initiators.

When multiple memory initiators exist, they may not all have the same performance when accessing a given memory target. Each initiator-target pair may be organized into different ranked access classes to represent this relationship. The highest performing initiator to a given target is considered to be one of that target's local initiators, and given the highest access class, 0. Any given target may have one or more local initiators, and any given initiator may have multiple local memory targets.

To aid applications matching memory targets with their initiators, the kernel provides symlinks to each other. The following example lists the relationship for the access class "0" memory initiators and targets:

```
# symlinks -v /sys/devices/system/node/nodeX/access0/targets/
relative: /sys/devices/system/node/nodeX/access0/targets/nodeY -> ../../nodeY

# symlinks -v /sys/devices/system/node/nodeY/access0/initiators/
relative: /sys/devices/system/node/nodeY/access0/initiators/nodeX -> ../../nodeX
```

A memory initiator may have multiple memory targets in the same access class. The target memory's initiators in a given class indicate the nodes' access characteristics share the same performance relative to other linked initiator nodes. Each target within an initiator's access class, though, do not necessarily perform the same as each other.

The access class "1" is used to allow differentiation between initiators that are CPUs and hence suitable for generic task scheduling, and IO initiators such as GPUs and NICs. Unlike access class 0, only nodes containing CPUs are considered.

NUMA Performance

Applications may wish to consider which node they want their memory to be allocated from based on the node's performance characteristics. If the system provides these attributes, the kernel exports them under the node sysfs hierarchy by appending the attributes directory under the memory node's access class 0 initiators as follows:

```
/sys/devices/system/node/nodeY/access0/initiators/
```

These attributes apply only when accessed from nodes that have the are linked under the this access's initiators.

The performance characteristics the kernel provides for the local initiators are exported are as follows:

```
# tree -P "read*|write*" /sys/devices/system/node/nodeY/access0/initiators/
/sys/devices/system/node/nodeY/access0/initiators/
|-- read_bandwidth
|-- read_latency
|-- write_bandwidth
`-- write_latency
```

The bandwidth attributes are provided in MiB/second.

The latency attributes are provided in nanoseconds.

The values reported here correspond to the rated latency and bandwidth for the platform.

Access class 1 takes the same form but only includes values for CPU to memory activity.

NUMA Cache

System memory may be constructed in a hierarchy of elements with various performance characteristics in order to provide large address space of slower performing memory cached by a smaller higher performing memory. The system physical addresses memory initiators are aware of are provided by the last memory level in the hierarchy. The system meanwhile uses higher performing memory to transparently cache access to progressively slower levels.

The term "far memory" is used to denote the last level memory in the hierarchy. Each increasing cache level provides higher performing initiator access, and the term "near memory" represents the fastest cache provided by the system.

This numbering is different than CPU caches where the cache level (ex: L1, L2, L3) uses the CPU-side view where each increased level is lower performing. In contrast, the memory cache level is centric to the last level memory, so the higher numbered cache level corresponds to memory nearer to the CPU, and further from far memory.

The memory-side caches are not directly addressable by software. When software accesses a system address, the system will return it from the near memory cache if it is present. If it is not present, the system accesses the next level of memory until there is either a hit in that cache level, or it reaches far memory.

An application does not need to know about caching attributes in order to use the system. Software may optionally query the memory cache attributes in order to maximize the performance out of such a setup. If the system provides a way for the kernel to discover this information, for example with ACPI HMAT (Heterogeneous Memory Attribute Table), the kernel will append these attributes to the NUMA node memory target.

When the kernel first registers a memory cache with a node, the kernel will create the following directory:

```
/sys/devices/system/node/nodeX/memory_side_cache/
```

If that directory is not present, the system either does not provide a memory-side cache, or that information is not accessible to the kernel.

The attributes for each level of cache is provided under its cache level index:

```
/sys/devices/system/node/nodeX/memory_side_cache/indexA/
/sys/devices/system/node/nodeX/memory_side_cache/indexB/
/sys/devices/system/node/nodeX/memory_side_cache/indexC/
```

Each cache level's directory provides its attributes. For example, the following shows a single cache level and the attributes available for software to query:

```
# tree /sys/devices/system/node/node0/memory_side_cache/
/sys/devices/system/node/node0/memory_side_cache/
|-- index1
|   |-- indexing
|   |-- line_size
|   |-- size
|   `-- write_policy
```

The "indexing" will be 0 if it is a direct-mapped cache, and non-zero for any other indexed based, multi-way associativity.

The "line_size" is the number of bytes accessed from the next cache level on a miss.

The "size" is the number of bytes provided by this cache level.

The "write_policy" will be 0 for write-back, and non-zero for write-through caching.

See Also

[1] https://www.uefi.org/sites/default/files/resources/ACPI_6_2.pdf - Section 5.2.27