NUMA -1- (ARM64 initialization)

2019-04-19 (http://jake.dothome.co.kr/numa-1/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

<kernel v5.10>

NUMA -1- (ARM64 initialization)

arm64_numa_init()

arch/arm64/mm/numa.c

```
/**
 * arm64_numa_init - Initialize NUMA
 *
 * Try each configured NUMA initialization method until one succeeds.  T
he
 * last fallback is dummy single node config encomapssing whole memory.
 */
```

```
void __init arm64_numa_init(void)
{
        if (!numa_off) {
                if (!acpi_disabled && !numa_init(arm64_acpi_numa_init))
                        return;
                if (acpi_disabled && !numa_init(of_numa_init))
                        return;
        }

        numa_init(dummy_numa_init);
}
```

On an ARM64 system, attempt to configure a NUMA system through the ACPI table or device tree. If there is no such configuration, it performs a dummy numa initialization with a single node.

numa_init()

arch/arm64/mm/numa.c

```
static int __init numa_init(int (*init_func)(void))
{
        int ret;

        nodes_clear(numa_nodes_parsed);
        nodes_clear(node_possible_map);
        nodes_clear(node_online_map);

        ret = numa_alloc_distance();
        if (ret < 0)
                return ret;

        ret = init_func();
        if (ret < 0)
                goto out_free_distance;

17          if (nodes_empty(numa_nodes_parsed)) {
18                  pr_info("No NUMA configuration found\n");
19                  ret = -EINVAL;
20                  goto out_free_distance;
21          }
22
23          ret = numa_register_nodes();
24          if (ret < 0)
25                  goto out_free_distance;
26
27          setup_node_to_cpumask_map();
28
29          return 0;
30  out_free_distance:
31          numa_free_distance();
32          return ret;
33  }
```

Initialize the NUMA system.

- In line 5~7 of the code, initialize all bitmaps that manage the NUMA state.
- In code lines 9~11, allocate as many numa_disatance[] arrays as you need and specify the initial values.
- Run the function @init_func received as an argument in line 13~15 of the code to set it.
- If there is no node information in line 17~21, it prints the message "No NUMA configuration found" and exits.
- In lines 23~25 of the code, assign the node data (struct pglist_data) and set the default node information.
- In line 27 of code, set the node-> cpu map.

As shown below, if the NUMA setting cannot be found through ACPI or the device tree, you will see the message "No NUMA configuration found" output. And it has been fallback, and through the dymmy NUMA initialization routine it says "Faking a node at..." You can also see the message being printed.

```
[    0.000000] NUMA: No NUMA configuration found
[    0.000000] NUMA: Faking a node at [mem 0x0000000040000000-0x000000007fffffff]
[    0.000000] NUMA: NODE_DATA [mem 0x7ddf4840-0x7ddf5fff]
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x0000000040000000-0x000000007fffffff]
[    0.000000]   Normal   empty
[    0.000000] Movable zone start for each node
[    0.000000] Early memory node ranges
[    0.000000]   node   0: [mem 0x0000000040000000-0x000000007fffffff]
[    0.000000] Initmem setup node 0 [mem 0x0000000040000000-0x000000007fffffff]
```

numa_alloc_distance()

arch/arm64/mm/numa.c

```
static int __init numa_alloc_distance(void)
{
        size_t size;
        u64 phys;
        int i, j;

07         size = nr_node_ids * nr_node_ids * sizeof(numa_distance[0]);
08         phys = memblock_find_in_range(0, PFN_PHYS(max_pfn),
09                                       size, PAGE_SIZE);
10         if (WARN_ON(!phys))
11                 return -ENOMEM;
12 
13         memblock_reserve(phys, size);
14 
15         numa_distance = __va(phys);
16         numa_distance_cnt = nr_node_ids;
17 
18         /* fill with the default distances */
19         for (i = 0; i < numa_distance_cnt; i++)
20                 for (j = 0; j < numa_distance_cnt; j++)
21                         numa_distance[i * numa_distance_cnt + j] = i == j ?
22                                 LOCAL_DISTANCE : REMOTE_DISTANCE;
23 
24         pr_debug("Initialized distance table, cnt=%d\n", numa_distance_cnt);
25 
26         return 0;
27 }
```

Number of nodes * Create and allocate an array of numa_distance[] for the number of nodes.

- In line 7 of code, use the number of nodes * the number of nodes as the size to be assigned.
- In line 8~11 of the code, we find out the space equal to the size of the empty space of the memblock on a page-by-page basis.
- In line 13 of code, reserve the size area to the memblock.
- Convert the physical address assigned in line 15~16 to a virtual address, assign it to a numa_distance, and specify the number of nodes.
- In lines 19~22 of code, numa_distance[] logically expresses a double array. The default numa_distance[] value is LOCAL_DISTANCE(10) only if it means the same node, and REMOTE_DISTANCE(20) is otherwise specified.
    - e.g. if nr_node_ids=2, look at numa_distance[].
        - numa_distance[0] = 10
        - numa_distance[1] = 20
        - numa_distance[2] = 20
        - numa_distance[3] = 10

numa_register_nodes()

arch/arm64/mm/numa.c

```
static int __init numa_register_nodes(void)
{
        int nid;
        struct memblock_region *mblk;

        /* Check that valid nid is set to memblks */
        for_each_mem_region(mblk) {
                int mblk_nid = memblock_get_region_node(mblk);

                if (mblk_nid == NUMA_NO_NODE || mblk_nid >= MAX_NUMNODES) {
                        pr_warn("Warning: invalid memblk node %d [mem %#010Lx-%#010Lx]\n",
12                                 mblk_nid, mblk->base,
13                                 mblk->base + mblk->size - 1);
14                         return -EINVAL;
15                 }
16         }
17 
18         /* Finally register nodes. */
19         for_each_node_mask(nid, numa_nodes_parsed) {
20                 unsigned long start_pfn, end_pfn;
21 
22                 get_pfn_range_for_nid(nid, &start_pfn, &end_pfn);
23                 setup_node_data(nid, start_pfn, end_pfn);
24                 node_set_online(nid);
25         }
26 
27         /* Setup online nodes to actual nodes*/
28         node_possible_map = numa_nodes_parsed;
29 
30         return 0;
31 }
```

Assign node data (struct pglist_data) and set the default node information.

- In code lines 7~16, if no node is specified in the memblock, it will print a warning message and return an error -EINVAL value.
- In line 19~25 of the code, traverse the nodes, assign node data (struct pglist_data), set the basic information of the node, and change the node status to online.
- In line 28 of code, specify the online nodes in the possible map.

setup_node_to_cpumask_map()

arm64/mm/numa.c

```
/*
 * Allocate node_to_cpumask_map based on number of available nodes
 * Requires node_possible_map to be valid.
 *
 * Note: cpumask_of_node() is not valid until after this is done.
 * (Use CONFIG_DEBUG_PER_CPU_MAPS to check this.)
 */
```

```
static void __init setup_node_to_cpumask_map(void)
{
        int node;

        /* setup nr_node_ids if not done yet */
        if (nr_node_ids == MAX_NUMNODES)
                setup_nr_node_ids();

        /* allocate and clear the mapping */
        for (node = 0; node < nr_node_ids; node++) {
                alloc_bootmem_cpumask_var(&node_to_cpumask_map[node]);
                cpumask_clear(node_to_cpumask_map[node]);
        }

        /* cpumask_of_node() will now work */
        pr_debug("Node to cpumask map for %u nodes\n", nr_node_ids);
}
```

Node - > sets the CPU map.

- In line 6~7 of the code, specify the number of possible nodes if the nr_node_ids has not yet been set.
- In line 10~13 of code, it iterates through the number of nodes, allocates and initializes the node_to_cpumask_map[node] bitmap.

dummy_numa_init()

arch/arm64/mm/numa.c

```
/**
 * dummy_numa_init - Fallback dummy NUMA init
 *
 * Used if there's no underlying NUMA architecture, NUMA initialization
 * fails, or NUMA is disabled on the command line.
 *
 * Must online at least one node (node 0) and add memory blocks that cover all
 * allowed memory. It is unlikely that this function fails.
 */
```

```
01 static int __init dummy_numa_init(void)
02 {
03         phys_addr_t start = memblock_start_of_DRAM();
04         phys_addr_t end = memblock_end_of_DRAM();
05         int ret;
06
07         if (numa_off)
08                 pr_info("NUMA disabled\n"); /* Forced off on command line. */
09         pr_info("Faking a node at [mem %#018Lx-%#018Lx]\n", start, end - 1);
10
11         ret = numa_add_memblk(0, start, end);
12         if (ret) {
13                 pr_err("NUMA init failed\n");
14                 return ret;
15         }
16
17         numa_off = true;
18         return 0;
19 }
```

If there is no NUMA setting, initialize it with a dummy NUMA configuration consisting of one node.

- If the kernel parameter "numa=off" is specified in code lines 7~8, it will output the message "NUMA disabled".
- In line 9 of the code, "Faking a node at..." Outputs a single memory information via message information.
- In lines 11~15 of the code, specify node 0 for both memory memblocks.
- On line 17 of the code, NUMA indicates that it is disabled.

numa_add_memblk()

arch/arm64/mm/numa.c

```
/**
 * numa_add_memblk() - Set node id to memblk
 * @nid: NUMA node ID of the new memblk
 * @start: Start address of the new memblk
 * @end:  End address of the new memblk
 *
 * RETURNS:
 * 0 on success, -errno on failure.
 */
```

```
int __init numa_add_memblk(int nid, u64 start, u64 end)
{
        int ret;

        ret = memblock_set_node(start, (end - start), &memblock.memory, nid);
        if (ret < 0) {
                pr_err("memblock [0x%llx - 0x%llx] failed to add on node %d\n",
                        start, (end - 1), nid);
                return ret;
        }

        node_set(nid, numa_nodes_parsed);
        return ret;
}
```

Specify the node ID in the memblock area and set the current node in the numa_nodes_parsed bitmap.

"numa" Early Kernel Parameters

numa_parse_early_param()

arch/arm64/mm/numa.c

```
static __init int numa_parse_early_param(char *opt)
{
        if (!opt)
                return -EINVAL;
        if (str_has_prefix(opt, "off"))
                numa_off = true;

        return 0;
}
early_param("numa", numa_parse_early_param);
```

If the "numa=off" kernel parameter is specified, assign true to the global variable numa_off.

Initializing the NUMA system via the device tree

Refer to the system configuration with four NUMA nodes as follows.

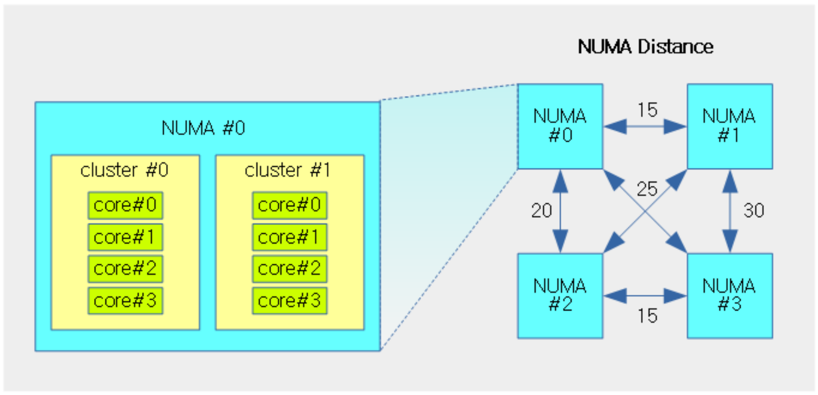
- Each node consists of 4 clusters, for a total of 16 clusters.
- Each cluster has four cores, for a total of 4 cores.
- Since the numa id of the memory node is recognized and passed on by the UEFI firmware, only the information of node 0 is recorded below.

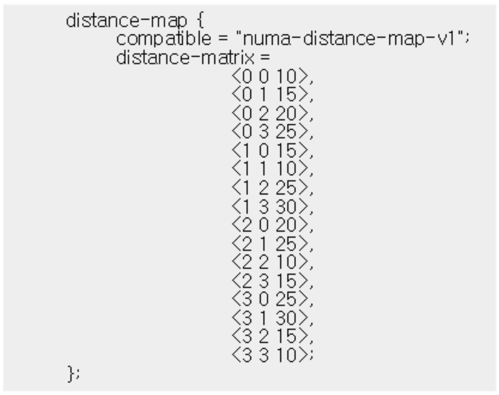
```
#include "hip07.dtsi"

/ {
        model = "Hisilicon Hip07 D05 Development Board";
        compatible = "hisilicon,hip07-d05";

        /* the mem node will be updated by UEFI. */
        memory@0 {
                device_type = "memory";
                reg = <0x0 0x00000000 0x0 0x40000000>;
                numa-node-id = <0>;
        };

        distance-map {
                compatible = "numa-distance-map-v1";
                distance-matrix = <0 0 10>,
                                  <0 1 15>,
                                  <0 2 20>,
                                  <0 3 25>,
                                  <1 0 15>,
                                  <1 1 10>,
                                  <1 2 25>,
                                  <1 3 30>,
                                  <2 0 20>,
                                  <2 1 25>,
                                  <2 2 10>,
                                  <2 3 15>,
                                  <3 0 25>,
                                  <3 1 30>,
                                  <3 2 15>,
                                  <3 3 10>;
```

arch/arm64/boot/dts/hisilicon/hip07.dtsi

```
        cpu-map {
                cluster0 {
                        core0 {
                                cpu = <&cpu0>;
                        };
                        core1 {
                                cpu = <&cpu1>;
                        };
                        core2 {
                                cpu = <&cpu2>;
                        };
                        core3 {
                                cpu = <&cpu3>;
                        };
                };

                ...

                cluster15 {
                        core0 {
                                cpu = <&cpu60>;
                        };
                        core1 {
                                cpu = <&cpu61>;
                        };
                        core2 {
                                cpu = <&cpu62>;
                        };
                        core3 {
                                cpu = <&cpu63>;
                        };
                };
        };

        cpu0: cpu@10000 {
                device_type = "cpu";
                compatible = "arm,cortex-a72", "arm,armv8";
                reg = <0x10000>;
                enable-method = "psci";
                next-level-cache = <&cluster0_l2>;
                numa-node-id = <0>;
        };

        ...

        cpu63: cpu@70303 {
                device_type = "cpu";
                compatible = "arm,cortex-a72", "arm,armv8";
                reg = <0x70303>;
                enable-method = "psci";
                next-level-cache = <&cluster15_l2>;
                numa-node-id = <3>;
        };

        cluster0_l2: l2-cache0 {
                compatible = "cache";
        };

        ...

        cluster15_l2: l2-cache15 {
                compatible = "cache";
        };
};
```

The following figure shows the distances of the NUMA system.





```
distance-map {
        compatible = "numa-distance-map-v1";
        distance-matrix =
                        <0 0 10>,
                        <0 1 15>,
                        <0 2 20>,
                        <0 3 25>,
                        <1 0 15>,
                        <1 1 10>,
                        <1 2 25>,
                        <1 3 30>,
                        <2 0 20>,
                        <2 1 25>,
                        <2 2 10>,
                        <2 3 15>,
                        <3 0 25>,
                        <3 1 30>,
                        <3 2 15>,
                        <3 3 10>;
};
```

(http://jake.dothome.co.kr/wp-content/uploads/2019/04/numa-distance-1.png)

of_numa_init()

drivers/of/of_numa.c

```
int __init of_numa_init(void)
{
        int r;

        of_numa_parse_cpu_nodes();
        r = of_numa_parse_memory_nodes();
        if (r)
                return r;
        return of_numa_parse_distance_map();
}
```

It reads the Numa ID from the CPU node and the memory node through the device tree, and parses the distance map.

of_numa_parse_cpu_nodes()

drivers/of/of_numa.c

```
/*
 * Even though we connect cpus to numa domains later in SMP
 * init, we need to know the node ids now for all cpus.
*/
```

```
static void __init of_numa_parse_cpu_nodes(void)
{
        u32 nid;
        int r;
        struct device_node *np;

        for_each_of_cpu_node(np) {
                r = of_property_read_u32(np, "numa-node-id", &nid);
                if (r)
                        continue;

                pr_debug("CPU on %u\n", nid);
                if (nid >= MAX_NUMNODES)
                        pr_warn("Node id %u exceeds maximum value\n", nid);
                else
                        node_set(nid, numa_nodes_parsed);
        }
}
```

It traverses the number of CPU nodes and parses them.

- If the "numa-node-id" attribute is greater than or equal to 1, numa_nodes_parsed sets the node in the bitmap.

of_numa_parse_memory_nodes()

drivers/of/of_numa.c

```
static int __init of_numa_parse_memory_nodes(void)
{
        struct device_node *np = NULL;
        struct resource rsrc;
        u32 nid;
        int i, r;

        for_each_node_by_type(np, "memory") {
                r = of_property_read_u32(np, "numa-node-id", &nid);
                if (r == -EINVAL)
                        /*
                         * property doesn't exist if -EINVAL, continue
                         * looking for more memory nodes with
                         * "numa-node-id" property
                         */
                        continue;

                if (nid >= MAX_NUMNODES) {
                        pr_warn("Node id %u exceeds maximum value\n", nid);
                        r = -EINVAL;
                }

                for (i = 0; !r && !of_address_to_resource(np, i, &rsrc); i++)
                        r = numa_add_memblk(nid, rsrc.start, rsrc.end + 1);

                if (!i || r) {
                        of_node_put(np);
                        pr_err("bad property in memory node\n");
                        return r ? : -EINVAL;
                }
        }

        return 0;
}
```

It traverses the number of memory nodes, parses the nodes, and writes the nodes to the memblock.

- If it is a memory node with the "numa-node-id" attribute, write the node information to the memblock in that area.

of_numa_parse_distance_map()

drivers/of/of_numa.c

```
static int __init of_numa_parse_distance_map(void)
{
        int ret = 0;
        struct device_node *np;

        np = of_find_compatible_node(NULL, NULL,
                                     "numa-distance-map-v1");
        if (np)
                ret = of_numa_parse_distance_map_v1(np);

        of_node_put(np);
        return ret;
}
```

It comes from parsing the distance map on nodes with the value of the attribute compatible = "numa-distance-map-v1".

of_numa_parse_distance_map_v1()

drivers/of/of_numa.c

```
static int __init of_numa_parse_distance_map_v1(struct device_node *map)
{
        const __be32 *matrix;
        int entry_count;
        int i;

        pr_info("parsing numa-distance-map-v1\n");

        matrix = of_get_property(map, "distance-matrix", NULL);
        if (!matrix) {
                pr_err("No distance-matrix property in distance-map\n");
                return -EINVAL;
        }

        entry_count = of_property_count_u32_elems(map, "distance-matri
x");
        if (entry_count <= 0) {
                pr_err("Invalid distance-matrix\n");
                return -EINVAL;
        }

        for (i = 0; i + 2 < entry_count; i += 3) {
                u32 nodea, nodeb, distance;

                nodea = of_read_number(matrix, 1);
                matrix++;
                nodeb = of_read_number(matrix, 1);
                matrix++;
                distance = of_read_number(matrix, 1);
                matrix++;

                if ((nodea == nodeb && distance != LOCAL_DISTANCE) ||
                    (nodea != nodeb && distance <= LOCAL_DISTANCE)) {
```

```
                                pr_err("Invalid distance[node%d -> node%d] = %d\n",
                                        nodea, nodeb, distance);
                                return -EINVAL;
                        }

                        numa_set_distance(nodea, nodeb, distance);

                        /* Set default distance of node B->A same as A->B */
                        if (nodeb > nodea)
                                numa_set_distance(nodeb, nodea, distance);
                }

        return 0;
}
```

parse the distance map.

- Read the value of the "distance-matrix" attribute and assign a value to the numa_distance[] array.

numa_set_distance()

arch/arm64/mm/numa.c

```
/**
 * numa_set_distance() - Set inter node NUMA distance from node to node.
 * @from: the 'from' node to set distance
 * @to: the 'to'  node to set distance
 * @distance: NUMA distance
 *
 * Set the distance from node @from to @to to @distance.
 * If distance table doesn't exist, a warning is printed.
 *
 * If @from or @to is higher than the highest known node or lower than zero
 * or @distance doesn't make sense, the call is ignored.
 *
 */
```

```
void __init numa_set_distance(int from, int to, int distance)
{
        if (!numa_distance) {
                pr_warn_once("Warning: distance table not allocated yet\n");
                return;
        }

        if (from >= numa_distance_cnt || to >= numa_distance_cnt ||
                        from < 0 || to < 0) {
                pr_warn_once("Warning: node ids are out of bound, from=%d to=%d distance=%d\n",
                            from, to, distance);
                return;
        }

        if ((u8)distance != distance ||
            (from == to && distance != LOCAL_DISTANCE)) {
                pr_warn_once("Warning: invalid distance parameter, from=%d to=%d distance=%d\n",
                             from, to, distance);
                return;
        }

        numa_distance[from * numa_distance_cnt + to] = distance;
}
```

numa distace.

- numa_distance specifies a distance value in the [from * numa_distance_cnt + to] array.

consultation

- NUMA -1- (ARM64 initialization) | Sentence C – Current post
- NUMA -2- (Fallback Node) (http://jake.dothome.co.kr/numa-fallback-node) | Qc
- NODE Bitmap (API) (http://jake.dothome.co.kr/node-api) | Qc

- NUMA binding description (https://github.com/torvalds/linux/blob/master/Documentation/devicetree/bindings/numa.txt) | kernel.org
- NUMA with Linux (https://lunatine.net/2016/07/14/numa-with-linux/) | Lunatine's Box

❮ bootmem_init() – ARM64 (http://jake.dothome.co.kr/bootmem_init-64/)

ARM64 System Key Registers ❯ (http://jake.dothome.co.kr/registers64/)