Report 3

Xing Pan (xpan)

Project introduction:

In modern CPU, there are many cores and several memory controllers in one processor. For example, in the 2-way SMPs with AMD Opteron 6128, there are 16 cores and 4 memory controllers in 1 node. Each core access different memory controllers will get the different latency. So it is not an easy work to predict the worst case execution time when we run the program on the multicore system.

The goal of this project is to find a new virtual and physical address mapping algorithm help us to predict the memory access latency exactly in the multicore system.

Tasks accomplished:

Step 1: verify there is the memory latency existed.

To verify the memory access latency existed, I did the brute-force test program on the 2-way SMPs with AMD Opteron 6128. Use ”go back and forth” to avoid the hardware prefetch and ”calloc a large address space” to avoid the cache hit. Utilize the /proc/pid/pagemap and /proc/pid/maps file to map the virtual address to physical address. Get the following data to verify that there exist the latency which caused by accessing multiple memory controllers.

Virtual address : 0x2B82002EB010

Physical address: 0x1DAF32010,

Node: 0, channel: 1, rank: 3, bank: 2, row: 6959, column: 512

latency: 8111us *

Virtual address : 0x2B82FE2ED010

Physical address: 0x2055BE010,

Node: 0,channel:0, rank: 3, bank: 3, row: 15829, column: 2560

latency: 12466us

Virtual address : 0x2B8535AEF010

Physical address: 0x611485010,

Node:2,channel: 1, rank: 2, bank: 0, row: 16020, column: 2304

```
    latency: 17152us
Virtual address : 0x2b85f8af0010
    Physical address: 0x556E74010,
    Node: 2, channel: 1, rank: 3, bank: 6, row: 4846, column: 2048
    latency: 17006us
Virtual address : 0x2b86f7af1010
    Physical address: 0x80ED03010,
    Node: 3, channel: 1, rank: 2, bank: 0, row: 7789, column: 768
    latency: 16957us
```

The data clearly shows that the latency from accessing different memory controller. For example, if the core access node 0 (memory controller id 0) will get the least latency. However, if it accesses the node 2 or node 3, there will be more latency.

Step 2: Configure out the AMD Opteron 6128 architecture

In the experiment, I found that the physical address is grouped by the NodeID.

For AMD Opteron 6128, there are 4 nodes (memory controllers) in one CPU. Each node (memory controller) has different physical address range.

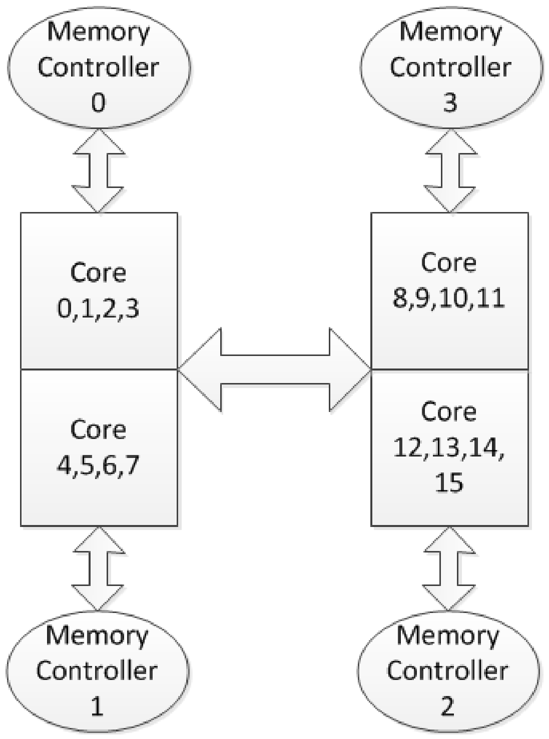
Node 0:0x0~~0x227FFFFFF

Node 1:0x228000000~~0x427FFFFFF

Node 2:0x428000000~~0x627FFFFFF

Node 3: 0x628000000~~0x827FFFFFF

There are 16 cores in one CPU. When one core in the CPU accesses the main memory, it will access different memory controller and get the different access latency. The following graph is the architecture of AMD Opteron 6128.



There are some observations:

1. When a program run on one core, the nearest memory controller is preferential.

I did another experiment about this observation. The program just calloc 8GB which is no more than the physical space in one memory controller. The result shows that the program just use the nearest memory controller.

2. It takes least latency when Core 0,1,2,3 access memory controller 0. It takes the longest latency when they access memory controller 2 and 3. However, it takes medium latency when they access memory controller 1. Same situation for the other cores. For example, for core 0

memory controller 0: 9000us *

memory controller 1: 12000us

memory controller 2,3: 17000us

Step 3 test the contend and latency between multiple threads in multicore system

Use a latency test program. In this program, it does the initialization (calloc a memory space) at this main thread and creates several threads. Each thread will bind to different core and access a same memory controller respectively.

Then I measure the accessing latency for each thread and compare the latency with the latency if there is only one thread access the memory controller (same large memory space).

The following is the data:

One thread assigned on the core 8 and access the memory controller 3 :

latency is 9400us *

Two threads, one of them assigned on the core 8 and the other one assigned to core 9. Both of them access the memory controller 3 :

the maximum latency is 12700us, the minimum latency is 10900us

Four threads, assign them to core 8, core 9, core 10 and core 11 respectively. All of them access the memory controller 3 :

the maximum latency is 19500us, the minimum latency is 12200us

Four threads, assign them to core 0, core 4, core 8 and core 12 respectively. They access the memory controller 0, 1, 2, 3 correspondingly:

latency is 9300us

There are some observations:

1. The latency is much longer than the latency if there is only one thread access one memory controller alone. (about 9000us).

2. The more threads access a same memory controller, the worse latency they get. For example, the latency when I use 4 threads is much longer than only 2 threads.

3. If we use multi-threads to access the same memory controller, there definitely is a contend. Besides, the latency is variable and irregular fluctuated. For real time system, It is very hard to predict it.

4. When a program run, it is very hard to know which memory space it uses. Although the operation system has the “first tough” mechanism, the program still use multiple memory controllers instead of single memory controller potentially.

5. Due to the program does the initialization (calloc the memory space) in the main thread, I didn’t find there is a contention on the channel, rank, bank.

Next step:

1. Calloc the memory space in each thread. Test and try to find where the contention comes from.

2. Try to design a coloring algorithm to solve the contention. We can try to color the banks in each memory controller and assign the colors to each thread. Use this coloring algorithm to make multicore system predictable for real time system.

3. Measure the latency and performance for the new algorithm.

Reference

[1] TILEPROCESSOR ARCHITECTURE OVERVIEW FOR THE TILEPROSERIES

[2] Christopher Zimmer and Frank Mueller. Low Contention Mapping of Real-Time Tasks onto a TilePro 64 Core Processor.

[3] Balasubramanya Bhat and Frank Mueller. Making DRAM Refresh Predictable

[4] Zheng Pei Wu, Yogen Krish, and Rodolfo Pellizzoni. Worst Case Analysis of DRAM Latency in Multi-Requestor Systems

[5] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo and Lui Sha. Memory Access Control in Multiprocessor for Real-time Systems with Mixed Criticality

[6] Wei Wang, Tanima Dey, Jack W. Davidson, and Mary Lou Soffa. DraMon: Predicting Memory Bandwidth Usage of Multi-threaded Programs with High Accuracy and Low Overhead

[7] Noriaki Suzuki, Hyoseung Kim, Dionisio de Niz. Coordinated Bank and Cache Coloring for Temporal Protection of Memory Accesses

[8] Hyoseung Kim, Dionisio de Niz, Bjorn Andersson. Bounding Memory Interference Delay in COTS-based Multi-Core Systems.

[9] Heechul Yun, Renato Mancuso, Zheng-Pei Wu, Rodolfo Pellizzoni . PALLOC: DRAM Bank-Aware Memory Allocator for Performance Isolation on Multicore Platforms,

* The latency data in this document is the totally time when CPU access physical memory 524288 times.