Ensuring the Fairness of program's performance on CMP

Qilong Wang, Jun Gao, Guangsong Hou, Hongkui Li[a], Ke Xu

State Grid Shandong Heze Electric Power Company, 274000 Heze, Shandong Province, China

Abstract. On CMP, programs commonly interference with each other. But for programs co-running on CMP, serious performance dropdown is not expected. With the growth of the program's size, program's contention for limited shared resources in the system will become increasingly intense, and the resulting impact on system performance will be even more pronounced. How to allocate and share limited shared resources among programs is a very important problem. In this paper we propose a new method to ensure the fairness of program's performance on CMP. The share resource CPU and memory are included in our study. Meanwhile, the Linux resource management tool Cgroups is used to realize our idea. By the resource we profiled and the tool support, we realize a reasonable resource dividing method to ensure the fairness of program's performance. The reasonable resource dividing method include three Engine. The experiment result show that making use of the fairness resource dividing method proposed by our paper, compared with operating system scheduling, program's performance difference can be largely reduced.

1 Introduction

With the development of integrated circuit technology, chip multi-processor (CMP) has become the most common processor in the current cluster and desktop computer, and it is also the current development direction [1, 2].

On CMP, programs commonly interference with each other [3]. Some times the effect of interference is little, which do little effect to program's performance. Some time the interference takes big effect, which cause serious drop down of performance [4,5]. Further more, for programs co-running on CMP, serious performance dropdown is not expected.

On the CMP, cores typically share the last level cache, DRAM controllers, memory bus bandwidth, etc. With the growth of the program's size, program's contention for limited shared resources in the system will become increasingly intense, and the resulting impact on system performance will be even more pronounced. How to allocate and share limited shared resources among programs is a very important problem. In order to alleviate this problem, in addition to increasing the availability of shared resources, a scheduling algorithm that can share the shared resources among programs in a fair and efficient manner is also critical. Among all kinds of shared resources, CPU and memory have the greatest impact on program's performance.

If a scheduling policy exists to allocate the shared resources fairly and efficiently among multiple parallel threads, it can mitigate the negative impact of contention for shared resources, thereby improving system performance. Therefore, the allocation and scheduling of shared resources in CMP systems has become a hot issue to be studied.

In this paper we propose a new method to ensure the fairness of program's performance on CMP. The share resource CPU and memory are included in our study. Meanwhile, the Linux resource management tool Cgroups is used to realize our idea. By the resource we profiled and the tool support, we realize a reasonable resource dividing method to ensure the fairness of program's performance. The reasonable resource dividing method include three Engine.

The first is the profiling engine, in the profiling engine, the resource usage information of every programs are profiled. The resource profiled are CPU and memory. We top program's resource usage of CPU and memory and transfer the information to the planning engine.

The second is the planning engine, the function of the planning engine is to decide how to divide CPU and memory resource to all the programs, with the aim of making program's performance almost the same. The input of the planning engine is the information of CPU and memory resource usage passed from the profiling engine, and the output of the planning engine is is the size of CPU and memory resource should be applied for all the programs.

The third is the implement engine, we use Cgroups to realize the implement engine. The input of the implement engine is the output of the planning engine. The data of the size of CPU and memory are used to set the size of Cgroups control groups. After that, programs can be running in it's own execution environment.

The experiment result show that making use of the fairness resource dividing method proposed by our paper,

[a] Corresponding author: dkzxzdh@163.com

compared with operating system scheduling, program's performance difference can be largely reduced.

2 Related work

There are some related work on the fairness of program's performance on CMP. Mutlu et al. [6] propose a shared DRAM controller, with the aim of both improve program's performance and the QoS of programs. Zhou et al. [7] introduces a model to analyze the perfor- mance impact of cache sharing, and proposes a mechanism of cache sharing management to provide performance fairness for concurrently executing applications.

3 Fairness resource dividing method

The fairness resource dividing method consist of three engines: the profiling engine is use to profiling program's basic information on CPU and memory, the planning engine is used to do the resource assignment plan, and the implement engine is used to divide CPU and memory resource to programs. Figure 1 shows the flow chart.

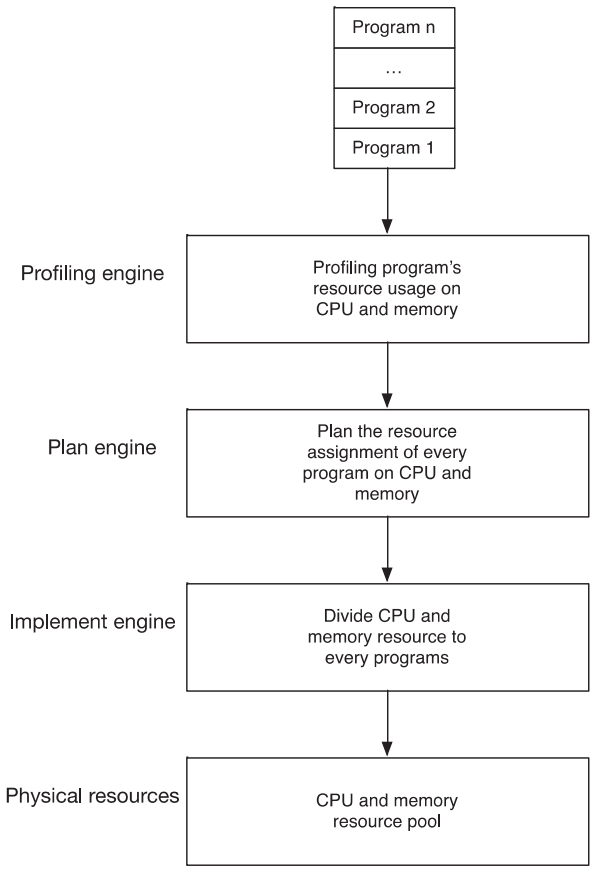

Figure 1. The flow chart of the fairness resource dividing method

3.1 The profiling engine

In the profiling engine, the resource usage information of every programs are profiled. The resource profiled are CPU and memory. The general performance profiling tools can be used to accomplish the task. In our paper we use Linux tool Top. We top program's resource usage of CPU and memory and transfer the information to the planning engine. This procedure can be accomplished online. Then the profiling result is the input of the planning engine.

3.2 The planning engine

The function of the planning engine is to decide how to divide CPU and memory resource to all the programs, with the aim of making program's performance almost the same. This part is very important for our fairness resource dividing method, which directly resulting in program's performance.

The first step is to decide the number of programs can be running on one server, so we calculate the number of programs based on the pseudo-code in Figure 2.

Pseudo-code of caculating the number of program running on one server

```
1: cpu_sum=cpu[p1]
2: mem_sum=mem[p1]
3: for i = p1 to pn do
4:    cpu_sum=cpu_sum+cpu[i+1];
5:    mem_sum=mem_sum+mem[i+1];
6:    if cpu_sum >cpu_usage
7:    {
8:    num_pro=i
9:    break;
10:   }
11:   if mem_sum >mem_usage
12:   {
13:   num_pro=i
14:   break;
15:   }
16: end for
```

Figure 2. Pseudo-code of calculating the number of program running on one server

In the pseudo-code, we sum the program's usage of CPU from p1 until the cpu_sum more than the CPU size of serve, meanwhile, we sum the program's usage of memory from p1 until the mem_sum more than the memory size of server. It is means once one of the CPU or memory resource is bigger than the size of CPU or memory, we will not add on new program. Then we can make sure the number of the program running on one server, which is pi. Then the programs which can not scheduled on this server will be scheduled onto the next server.

The input of the planning engine is the information of CPU and memory resource usage passed from the profiling engine, and the output of the planning engine is is the size of CPU and memory resource should be applied for all the programs. The output of the planning engine can be calculated obey the formula below.

Assume that there are n programs running on one server. There are p1, p2, …, pn respectively. The CPU resource usage of these program when running alone are Cp1, Cp2, …, Cpn, respectively, and the memory resource usage of these program when running alone are Mp1, Mp2, …, Mpn, respectively.

Then the CPU resource calculate method should obey formula (1). And the memory resource calculate method should obey formula (2).

$$C_{pi-schedule} = \frac{C_{pi}}{C_{p1} + C_{p2} + ... + C_{pn}} \times C_{server} \quad (1)$$

$$M_{pi-schedule} = \frac{M_{pi}}{M_{p1} + M_{p2} + ... + M_{pn}} \times M_{server} \quad (2)$$

In formula (1), pi is the number of the program, C_{pi} is the resource usage of program pi when it running alone on a server. C_{server} is the total CPU resource of the server. $C_{pi-schedule}$ is the size of the CPU resource when program pi is scheduled along with other programs.

In formula (2), M_{pi} is the memory resource usage of program pi when it running alone on a server. M_{server} is the total memory resource of the server. $M_{pi-schedule}$ is the size of the memory resource when program pi is scheduled along with other programs.

By the algorithm above, the size of CPU and memory resource divided for the programs can be calculate properly.

3.3 The implement engine

In the implement engine, Linux resource management tool Cgroups play a very important role, Cgroups is used to realize the implement engine. The input of the implement engine is the output of the planning engine. The data of the size of CPU and memory are used to set the size of Cgroups control groups, then the program's execution environment can be created. After that, programs can be running in it's own execution environment.

By making use of this method, all programs can be running under proper resource, ensuring the fairness of every programs.

4 Experiment

Our experiments are executed on a NUMA server, which with two socket and 4 core on each socket. The detail server information is in TABLE 1. We select programs from SPEC CPU2006 and NAS Parallel Benchmarks (NPB). The SPEC CPU2006 benchmark is SPEC's industry-standardized, CPU-intensive benchmark suite, stressing a system's processor, memory subsystem and compiler. The NPB are a set of benchmarks targeting performance evaluation of highly parallel supercomputers. The programs as workloads are in TABLE 2.

We compare the program's performance under fairness resource dividing method with program's performance when running alone.

Table 1. Server configuration

CPU	Intel Xeon E5620
core	4 cores@2.13G
L1 caches	32K
L2 caches	256K
L3 caches	4M
Threads per core	1 thread
Sockets	2
Memory	8GB, DDR3

Table 2. Workloads

Workloads	program
Workload 1	410, cg.C.8
Workload 2	445, 450, 453, ft.C.4
Workload 3	445, mg.C.8

We use three workloads for our experiment, Figure 3 shows the program's performance with different strategy, and the baseline is the program's performance when program running alone on a server. The programs in the first workload is 410, cg.C.8, we decide the number of program based on the idea in Figure 2. It is shows by the OS scheduling strategy, the performance difference of two programs is 8%, but by our fairness resource dividing method, ther performance difference of two programs is only 5%. It is because compared with OS scheduling, our scheduling can realize more fairness. By making use of Cgroups, the CPU and memory resource can be divided to all programs fairnessly. However, by OS scheduling, when programs running together on one server, the will competing the CPU and memory resource. Then the resource size can not be used very fairness by all programs. It is because different programs have different competitive power. Programs with a bigger competitive power will take over more resource, then the program's performance will be better. Programs with a smaller competitive power will take over little resource, then the program's performance will be poor. The competitive power is very complex, it is related the program's resource access pattern, coding method etc.

Figure 4 shows the program's performance with different strategy for workload 2, and the baseline is the program's performance when program running alone on a server. The programs in the second workload is 445, 450, 453, ft.C.8. It is shows by the OS scheduling strategy, the biggest performance difference of programs is 53%, but by our fairness resource dividing method, the biggest performance difference of programs is only 9%.

Figure 5 shows the program's performance with different strategy for workload 3, and the baseline is the program's performance when program running alone on a server. The programs in the second workload is 445, mg.C.8. It is shows by the OS scheduling strategy, the biggest performance difference of programs is 8%, but by our fairness resource dividing method, the biggest performance difference of programs is only 2%.

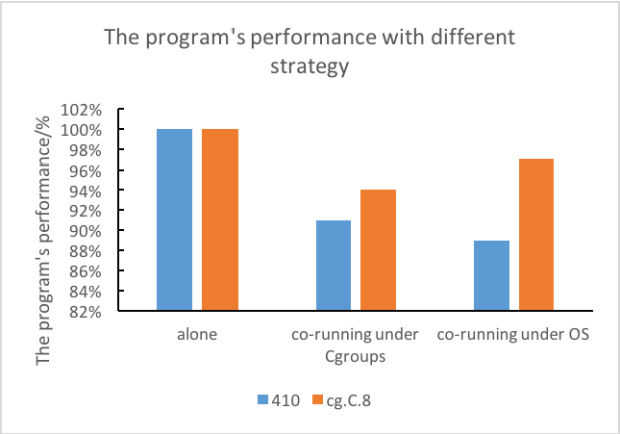

Figure 3. Program's performance with different strategy (workload 1)

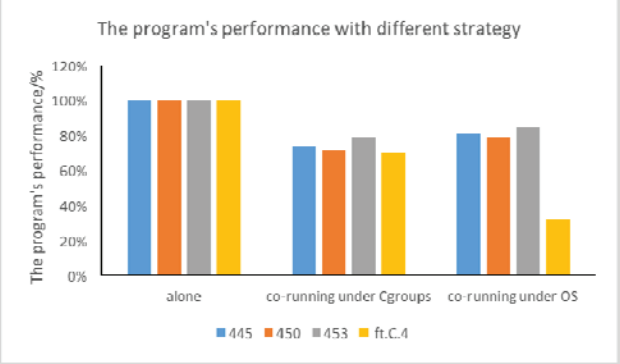

Figure 4. Program's performance with different strategy (workload 2)

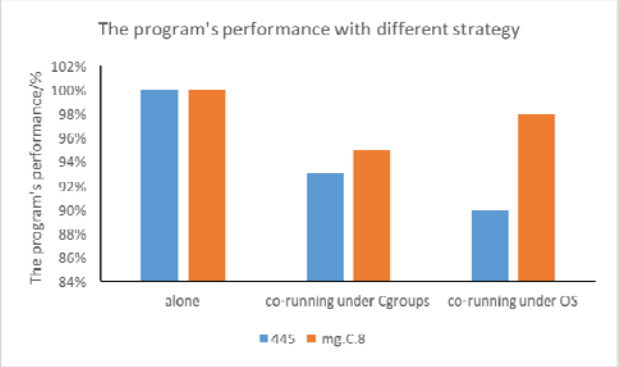

Figure 5. Program's performance with different strategy (workload 3)

5 Conclusion

In this paper we realize a reasonable resource dividing method to ensure the fairness of program's performance on CMP. The share resource CPU and memory are included in our study. Meanwhile, the Linux resource management tool Cgroups is used to realize our idea. The reasonable resource dividing method include three Engine.

The first is the profiling engine, in the profiling engine, the resource usage information of every programs are profiled. The resource profiled are CPU and memory. We top program's resource usage of CPU and memory and transfer the information to the planning engine.

The second is the planning engine, the function of the planning engine is to decide how to divide CPU and memory resource to all the programs, with the aim of making program's performance almost the same. The input of the planning engine is the information of CPU and memory resource usage passed from the profiling engine, and the output of the planning engine is is the size of CPU and memory resource should be applied for all the programs.

The third is the implement engine, we use Cgroups to realize the implement engine. The input of the implement engine is the output of the planning engine. The data of the size of CPU and memory are used to set the size of Cgroups control groups. After that, programs can be running in it's own execution environment.

The experiment result show that making use of the fairness resource dividing method proposed by our paper, compared with operating system scheduling, program's performance difference can be largely reduced.

References

1. C. Dhruba, G. Fei, K. Seongbeom, et al. Predicting inter-thread cache contention on a chip multi-processor architecture[C]. 11th International Symposium on High-Performance Computer Architecture, 2005: 340-351.
2. K. Seongbeom, C. Dhruba, S. Yan. Fair cache sharing and partitioning in a chip multiprocessor architecture[C]. Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, 2004: 111-122.
3. Y. Jiang, X. Shen, C. Jie, et al. Analysis and approximation of optimal co-scheduling on chip multiprocessors[C]. 2008 International Conference on Parallel Architectures and Compilation Techniques, 2008: 220-229.
4. K. Rob, B. Paul, H. Barbara, et al. Using OS observations to improve performance in multicore systems[J]. IEEE MICRO, 2008(28):0272-1732.
5. L. Wang, R. Wang, C. Fu, et. al. Interference-aware Program Scheduling for Multicore Processors[C]. ICA3PP 2013, 201312: 436-445.
6. O. Mutlu, T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems[J]. ACM SIGARCH Computer Architecture News, 2008(36): 63-74
7. X. Zhou, W. Chen, and W. Zheng. Cache sharing manage- ment for performance fairness in chip multiprocessors. In PACT, pages 384–393, Sept. 2009.