CS498P Project
Report

CACHE CONTENTION AWARE SCHEDULING IN CMP - SMT SYSTEMS

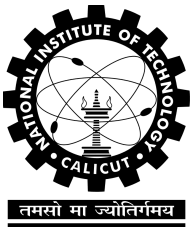
Submitted in partial fulfillment of
the requirements for the award of the degree of

Bachelor of Technology
in
Computer Science and Engineering

Submitted by

Roll No	Names of Students
B110087CS	BLESSON ANDREWS VARGHESE
B110773CS	NIRANJAN G S
B110419CS	RONY JOSHY

Under the guidance of
Mr SAIDALAVI KALADY



Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

Department of Computer Science and Engineering

National Institute of Technology Calicut

Certificate

This is to certify that this is a bonafide record of the project presented by the students whose names are given below during Monsoon 2014 in partial fulfilment of the requirements of the degree of Bachelor of Technology in Computer Science and Engineering.

Roll No	Names of Students
B110087CS	BLESSON ANDREWS VARGHESE
B110773CS	NIRANJAN G S
B110419CS	RONY JOSHY

Mr. Saidalavy Kalady
(Project Guide)

Mrs. Lijiya A.
(Course Coordinator)

Date:

Abstract

Simultaneous multi-threaded multi core processor architecture (CMP-SMT) has significantly improved the efficiency of computing. Completely fair scheduler, the default scheduler in Linux kernel, attempts to ensure a fair share of processor time to all processes and to balance loads at processor cores by distributing threads across the cores. In a multi-core processor architecture with resources shared between different cores, contention for shared resources may cause considerable performance degradation. This is observed when multiple threads competing for same cache lines result in additional cache misses. Our project aims at addressing this issue of cache contention by enabling CFS scheduler to dynamically detect cache contention and act accordingly.

Contents

Chapter 1

INTRODUCTION

As computers evolved the demand for more powerful processors grew at enormous rates. Efforts towards attaining parallelism led to Simultaneous multi-threaded processor architectures . SMT is designed with purpose of extracting maximum instruction level and thread level parallelism. Chip-level Multi processing was brought forth with same goal. Multiple cores were included in a single chip thereby trying to achieve maximum throughput.

A key point to be noted is the sharing of resources between different cores in CMP-SMT architecture. If necessary attention is not given performance may be affected badly. Overheads due to contention for shared resources can outweigh intended benefits. Co-run degradation is an increase in the execution time of an application when it shares resources with a co-runner, relative to running solo. [1]. It has been documented in previous studies [2] [3] that the execution time of a thread can vary greatly depending on which threads run on the other cores of the same chip. Previous works have shown that cache contention is one among the major causes of performance degradation [4].

Cache contention aware scheduler is aimed at providing better throughput and faster response time. Any contention aware scheduler must consist of two parts: a classification scheme for identifying which applications should and should not be scheduled together as well as the scheduling policy that assigns threads to cores given their classification. [1] . Cache miss rate being a clear indicator of co-run degradation due to shared resource contention,

can be used as the basis of a classification scheme aimed at an improved scheduler.

In this work we have come forward with an enhanced scheduling policy aimed at minimizing shared cache contention and ensuring that the better performance does not come at the cost of interactivity. We have used L2 cache miss rate as the coschedule criterion.

Chapter 2

RELATED WORK

The paper titled 'Contention Aware Scheduling on Multicore Systems' [1] identified Last level cache miss rate as one of the most accurate predictors of shared resource contention and used it to design a scheduling algorithm named 'Distributed Intensity'.

The paper titled 'Analysis and approximation of optimal coscheduling on chip multiprocessors' [5] described a methodology which constructs a graph with tasks as the nodes and the sum of co-run degradations of two tasks as the edge between them. The minimum weight perfect matching in this graph was suggested as a perfect scheduling assignment.

The thesis titled 'L2-cache aware coscheduling in CFS for SMT architecture' [6] used L2 cache miss rate as the coschedule criterion. The best task within a fixed window size found using this information is picked as the next one to be scheduled.

Chapter 3

OVERVIEW OF ARCHITECTURE AND SCHEDULER

3.1 CMP - SMT SYSTEMS AND CACHE HIERARCHY

Chip Multi Processing refers to computing with multiple cores integrated into a single chip. Simultaneous Multithreading allows more than one logical processor on a single processor core. With increasing number of processor cores in a single chip, sharing of resources has also increased. Cache is among the shared resources. Different levels of cache may be shared between different cores. In the model which we consider, cache is organized in three levels with L1 and L2 cache being local to each processor core and L3 cache being shared by all the cores.

3.2 COMPLETELY FAIR SCHEDULER

Completely Fair scheduler is the currently used process scheduler in Linux. 80 % of CFS's design can be summed up in a single sentence: CFS basically models an "ideal, precise multi-tasking CPU" on real hardware [8]. This means that if there are n processes ready to run, then each process gets 1/n share of processor. So, at the end of time t, all of the n processes would have run for time t with 1/n of processor power.

An ideal multi-tasking CPU is impractical as no processor can run n processes

simultaneously. So, CFS has the responsibility to ensure that all the runnable processes get a fair share of processor. CFS does this by allowing processes to run on processor by switching between the processes. CFS always tries to ensure that at any given time all the processes have run for nearly same time.

CFS is a priority based scheduler. CFS uses a priority range called NICE value which ranges from -20 to +19. Higher nice value indicates lower priority and vice versa. Weight of a task is calculated at the runtime from the nice value. Time slice for each process is calculated using the formula:

$$\text{slice} = \frac{period \times weight}{Total\ system\ load}$$

where
period is the expected time within which all runnable tasks execute once,
Total system load is sum of weights of all runnable tasks.

Virtual Runtime (vruntime) of a task is its actual runtime normalized to the total number of running tasks[8]. The virtual runtime is used to help us approximate the ideal multi-tasking processor that CFS is modeling. The function _update_curr() calculates execution time of the current process and stores that value in delta_exec. It then passes that runtime to _update_curr(), which weights the time by the number of runnable processes. The current process's vruntime is then incremented by the weighted value[7] .

$$\text{vruntime} \mathrel{+}= \frac{(delta\_exec) \times NICE\_0\_LOAD}{weight}$$

where
delta_ exec is the time for which current task ran,
NICE_0_LOAD is the default weight of a process in linux.

CFS picks the process with minimum vruntime as the next schedulable task. A red black tree (rbtree) is used to manage list of runnable tasks inorder to efficiently find the task with smallest vruntime.

Chapter 4

PROBLEM DEFINITION

"Designing a scheduling policy that enables CFS scheduler to schedule threads based on miss rate factor of co-run threads without compromising interactivity and to show that the enhanced scheduler has improved the performance of overall system"

In this work we are experimenting on completely fair scheduler with goal of achieving maximal system utilization by reducing memory latency linked with cache misses. The idea is to minimize the number of cache misses by using information collected online using performance counters. The new policy must also ensure good interactive performance.

Chapter 5

WORK DONE

5.1 PROPOSED SOLUTION

Our aim is to reduce overall cache contention, that is in all the levels. We assumed that L2 cache miss rate, which we get when two tasks run on the same physical core, is the best indicator of cache contention. We expect that tasks with low miss rates relative to each other cause low contention for L2 cache when scheduled on same core. Also tasks with low relative miss rates result in low L3 cache contention when run simultaneously on multiple cores.

5.1.1 CLASSIFICATION SCHEME

L2 cache miss rate is used as an indicator of cache contention. We define Relative miss rate of two tasks as the rate of L2 cache misses when both the tasks run simultaneously on sibling logical processors. Overall miss rate of a task is defined as the arithmetic mean of relative miss rates with reference to all other tasks.

Following points are considered while determining which applications should and should not be scheduled together :

1) Relative miss rate between corunning tasks should be minimized.

2) Two tasks with high overall miss rates (indicating high cache usage) should not be scheduled simultaneously on same physical core.

The cache miss rate obtained from the performance counters are to be stored in a data structure from which we can efficiently retrieve pairwise miss rates. A triangular matrix using O(n*n) space is a possible candidate.

5.1.2 SCHEDULING POLICY

The scheduling policy is used to assign tasks to cores given their classification. The following points are to be considered when scheduling the next task on a logical processor (Ck).

1) Tasks with high relative cache miss rates with respect to task running on sibling logical processor of Ck should be discouraged.

2) Tasks with high relative cache miss rate with respect to tasks running on other processor cores should be discouraged.

3) Tasks which have high miss rates relative to the current task that is to be preempted should be discouraged.

4) Task which ran immediately before the current task should be encouraged.

5) Tasks which have affinity to the current physical core should be encouraged.

6) Tasks with high overall miss rate should be discouraged if task running on the sibling core also has high overall miss rate.

When a logical core Ck is free, the best task to be scheduled on that core is decided by finding the task with minimum value of discouragement factor.

discouragement factor =

$f1$ × relative miss rate wrt thread running on the sibling logical core

$+f2$ × sum of relative miss rates wrt threads running on other physical cores

$+f3$ × relative miss rate wrt the current task to preempted from Ck

$-lf$ × (1 if this is the task that ran immediately before the current task 0 otherwise)

$-af$ × (1 if this task has affinity with current logical core 0 otherwise)

$+uf$ × (1 if this task and task in sibling logical core have high overall miss rate 0 otherwise)

where,

f1 is the factor of discouragement for contention for L2 cache.

f2 is the factor of discouragement for contention for L3 cache

f3 is factor of discouragement for contention relative to previous task

lf is locality factor,

af is affinity factor

uf is cache use factor

Thus the task with minimum discouragement factor is the one that we get according to our classification scheme. But keeping in mind that maintaining fairness and interactivity is the key motivation behind CFS, we have decided that in every k calls to the scheduler, we go with the task with minimum discouragement factor for k-1 times and with the task returned by default Linux scheduler, ie leftmost node in red black tree, the next time.

Chapter 6

FUTURE WORK

The coefficients for the various terms in the discouragement factor have to be determined experimentally. An efficient data structure used to store miss rate information should be designed. We are also pursuing to minimize space complexity of the design. Kernel module implementing the aforementioned scheme and policy must be written and tested against standards.

Chapter 7

CONCLUSION

Various factors that are expected to have significant role in cache contention were identified.Keeping these in mind, we have suggested a classification scheme and scheduling policy aim that improving the overall efficiency of the process scheduler. We expect the proposed scheduler to perform reasonably well in terms of interactivity.

REFERENCES

[1] BLAGODUROV, S., ZHURAVLEV, S., AND FEDOROVA, A. 2010. *Contention-aware scheduling on multi core systems.* ACM Trans. Comput. Syst. 28, 4, Article 8 (December 2010), 45 pages. DOI: 10.1145/1880018.1880019 , http://doi.acm.org/10.1145/1880018.1880019

[2] CHO, S. AND JIN, L. 2006. *Managing distributed, shared l2 caches through OS-level page allocation.* In Proceedings of the 39th Annual IEEE/ACM International Symposium on Micro-architecture (MICRO39). 455468 , people.cs.pitt.edu/~cho/cs2410/papers/cho-micro06.pdf

[3] TAM , D., AZIMI , R., AND STUMM , M. 2007. *Thread clustering: sharing-aware scheduling on smp- cmp-smt multiprocessors.* In Proceedings of the 2nd ACM European Conference on Computer Systems , dl.acm.org/citation.cfm?id=1273004

[4] FEDOROVA, A., SELTZER , M. I., AND SMITH , M. D. 2007. *Improving performance isolation on chip multiprocessors via an operating system scheduler.* In Proceedings of the 16th International Conference on Parallel Architectures and Compilation Techniques (PACT07). 2538 , http://dash.harvard.edu/bitstream/handle/1/10065537/Improving%20Performance%20Isolation2007%20(1).pdf?sequence=1

[5] J IANG, Y., S HEN, X., C HEN, J., AND TRIPATHI , R. 2008. *Analysis and approximation of optimal co-scheduling on chip multiprocessors.* In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT08). 220229 , people.engr.ncsu.edu/xshen5/Publications/pact08.pdf

[6] Suhas Heeraskar 2014. *L2-cache aware coscheduling in CFS for SMT architecture.* Thesis submitted as part of Mtech programme at National

Institute of Technology, Calicut , people.engr.ncsu.edu/xshen5/Publications/pact08.pdf

[7] ROBERT LOVE *Linux Kernel Development, Third Edition* , book.ilkaddimlar.com/d_pdf_book_proqramlashdirma_23519.do

[8] *CFS documentation* , www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt