ISSN 2229-5518

Designing Of Cache Partitioning Algorithm For Multicore Based Real Time System

S. Lokhnade, Dr. D.V. Padole

Abstract— Cache partitioning and sharing is important to the successful use of multicore processors. However, maximum studies have quite a few limitations, such as too much simulation time due to inter thread memory interfering, simulation in accurateness.This paper proposes a cache partitioning method for multicore based real time systems. It reduces cache interfering among concurrently executing threads. The cache partitioning algorithm estimates the miss-rate characteristics of each thread at run-time and dynamically partitions the cache among the threads that are executing concurrently and hence improve the performance**.**

Index Terms— cache, performance, benchmark, multiprocessor, fairness, interference, throughput.

1. INTRODUCTION

Many architectures today, have multiple cache partitions, each with potentially different performance characteristics. To meet the strict time to market requirements of systems, compilers require partitioning algorithms for effectively assigning values to the cache partitions. Furthermore, designers need a methodology for improving the performance of a cache hierarchy on an application without relying on time consuming simulation.

Several algorithms are studied and their advantages and drawbacks are used as a guide for the development of cache partitioning algorithm for multicore based real time system that achieves better performance than LRU and shows increasing promise over alternative algorithms as the number of processors increases.

This paper presents algorithms and techniques to effectively meet these needs. First, static cache partitioning algorithm is presented. This is the algorithm for effective partition assignment of cache. It supports a wide variety of memory models including multiple layers of caches partitions. A few ranges of benchmarks are used to demonstrate the effectiveness of the static cache partitioning algorithm. Experiments show optimal or near-optimal results on instruction miss rate and data miss rate on different benchmark application.

This also presents dynamic cache partitioning algorithm for multicore based real time system. This is an algorithm to estimate the effectiveness of the cache partitioning for a given application without requiring time consuming simulations. To show that dynamic cache partitioning generates accurate performance estimates, performance estimates are compared to previous algorithms simulation results. Experiments show performance of this algorithm is better as compared to previous algorithm.

- *S. lokhande is currently pursuing masters degree program in embedded engineering in Nagpur University, INDIA, E-mail: sangeeta173@gmail.com*

Lastly, static cache partitioning algorithm and dynamic cache partitioning algorithm for multicore based real time system are evaluated using a comprehensive SuperEScaler simulator and a wide variety of benchmark application.. This evaluation demonstrates the value and effectiveness of both algorithms in selecting suitable cache hierarchy for embedded systems. This evaluation also demonstrates the need for suitable benchmarks when evaluating partitioning algorithms.

This cache partitioning algorithm for multicore based real time system reduces cache interference among simultaneously executing benchmark application. The cache partitioning algorithm estimates the miss-rate characteristics of each benchmark application at run-time and dynamically partitions the cache among the application that is executing simultaneously and improves overall performance.

2. PREVIOUS WORKS

There are numerous projects that are associated with this project. To make things easier, this section will focus on few projects that are heavily associated with this project**.**

Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture [1]

This summarizes a detailed study of fairness in cache sharing between threads in a chip multiprocessor (CMP) architecture. This paper makes several contributions. First, it proposes and evaluates cache fairness metrics that measure the degree of fairness in cache sharing, and shows that two of them correlate very strongly with the execution time fairness. Secondly, using the metrics, the paper proposes static and dynamic L2 cache partitioning algorithms that optimize fairness. Finally studies the relationship between fairness and throughput in detail. The optimizing fairness usually increases throughput, while maximizing throughput does not necessarily improve fairness.

Dynamic Cache Partitioning for Simultaneous Multithreading Systems [2]

This paper presents a general partitioning scheme that can be applied to set-associative caches at any partition granularity. Further-more, in this scheme threads can have overlapping partitions, which provides more degrees of freedom when partitioning caches with low associativity. Trace-driven simulation results show a relative improvement in the L2 hit-rate of up to 40.5% over those generated by the standard least recently used replacement policy, and IPC improvements of up to 17%.

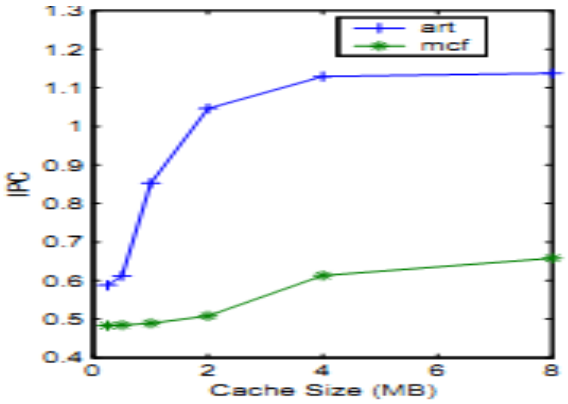


Fig1. IPC of art and mcf under 32-KB 8way L1 caches-IPC as a function of cache size

When executing the threads simultaneously the IPC values are approximated from Figure 2 and the hit-rates are estimated from the trace-driven simulations

A Data Centered Approach for Cache Partitioning in Embedded Real-Time Database System [3] allows different tasks to have a shared locking partition in cache. The hard real-time tasks will have their own partitions and thus they can perform high predictability. At the same time, a shared non-locking partition is reserved for the soft real-time tasks. In this way performance improvements based on the data that are frequently used by many tasks in the system.

The simulation results shown in Figure 2 demonstrate that the data centered cache partitioning scheme can further decrease the average miss rate significantly; for a 16KB data cache, the data centered cache partitioning scheme reduces the miss rate to 76% of that of the LRU policy and it is also superior to the static partitioning scheme as the miss rate is reduced by further 10%.

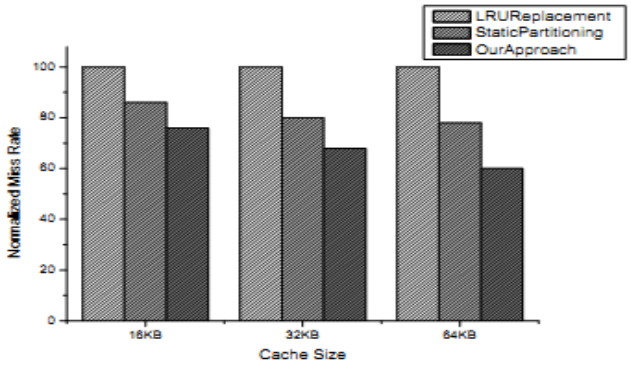
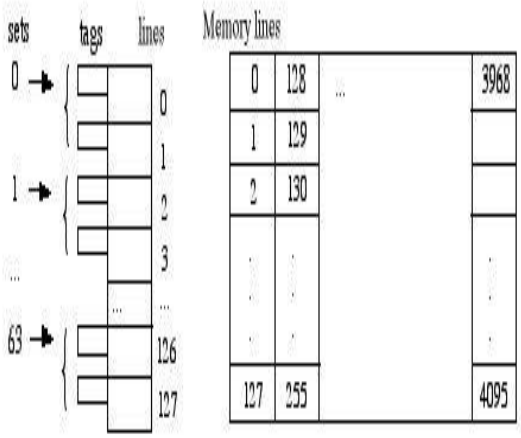


Fig2. Miss rate comparison under different cache size with 32-way associativity

3. PARTITIONING MECHANISMS



.

Fig 3.Set associative partitioned cache

The cache partitioning implementation is done in two steps. The first is develop the algorithm for static cache partitioning algorithm and second modify same algorithm for dynamic cache partitioning.

3.1 STATIC CACHE PARRIRIONING

The logic used for Static cache partitioning algorithm can be explained through Fig 3. Set associative partitioned cache. In this cache is divided in N number of Sets. Each Sets are the partitioned in different ways using N-way set associative mapping .For example some of the cache_line are given to instruction within each set and rest of the cache_lines are given to the data. Using this partitioning, each set in the cache is distributed the same, as determined by a partition register. The partition register is a pointer to the first data way within each set. In addition, minimum and maximum distributions are defined so that, for example, Fig. 3 illustrates a partitioned cache in a particular state such that some of the cache is giv-

en to instruction and remaining is given to data. The following algorithm is used for this mechanism.

ALGORITHM:

- Initialize cache(cache structure)
- Initialize the system with access-no instruction and data miss
- Access cache through access_type and address
- Process miss
- Determine miss rates
- Check for adjustment in favor of instruction cache
- If il1 and dl1 difference > miss rate threshold
    - Invalidate cache line
        - Set.line.valid=invalid
- Set.line.lastused= Repeat above step for Data cache
- Go for next iteration.

3.2 DYNAMIC CACHE PARTITIONING

When a cache miss occurs, the replacement mechanism, least recently used policy selects the entries in the accessed line that have been used. There is no information about entry or the relative occupancy of the cache by each application. Therefore, applications with a high demand, i.e. many accesses to different entries, are allocated more cache space than applications that have a low demand.

However, there is no guarantee that the new entries, brought into the cache because of a miss, will be reused. So by replacing entries owned by other processes, an application could be acquiring space that it does not need, resulting in suboptimal sharing of the cache. Consequently the performance of the system degrades. A representative example of such cases is when one of the competing processes is a streaming application. Previous studies presented the evidence of problems caused by LRU and this proposed cache partitioning as a solution. This evaluates different schemes that attempt to partition the cache dynamically. The algorithm used for implementation of static cache partitioning given above can be used for dynamic cache partitioning by having certain modification like inserting repartition event, comparing with threshold and developing configure file.

This algorithm can be run on simulator with different benchmark application to evaluate the performance.

4. EXPERIMENTAL SETUP

We make use of the SESC: Super E Scalar simulator with different benchmark application to evaluate the performance. Numerous simulations were run on each of the proposed cache, and the configurations that showed to be the most promising are presented below in table 4.1.

Parameter	Value
Cache size	16 KB
Associativity	8-way
Repartitioning period	100,000 cache accesses
Repartitioning threshold	0.02 miss rate difference
Repartitioning step size	16 sets
Replacement strategy	LRU

Table 4.1 Cache configuration

4.1 PERFORMANCE

This section presents the results of a simulation system in order to understand the quantitative effects of our cache partitioning scheme. The simulations concentrate on an 8-way set-associative L1 cache with 32-Byte blocks and vary the size of the cache over a range of 2 KB to 16 KB.

Our scheme is more likely to be useful for an L1 cache, and so that is the focus of our simulations. We believe that this will work on L2 caches as well.

For speed up calculation different benchmarks application are simulated, Lu, gcc, fft, art, mcf etc. from SPEC CPU2000. IPC for L1 cache managed by the LRU policy and one with a L1 cache managed by our cache partitioning algorithm. The absolute improvement in the fig 4 is the IPC of the partitioned case subtracted by the IPC of the standard LRU case. The relative improvement is the improvement relative to the IPC of the standard LRU, and is calculated by dividing the absolute improvement by the IPC of the standard LRU. The fig 4 shows that the partitioning algorithm improves IPC for all cache sizes up with previous algorithm.

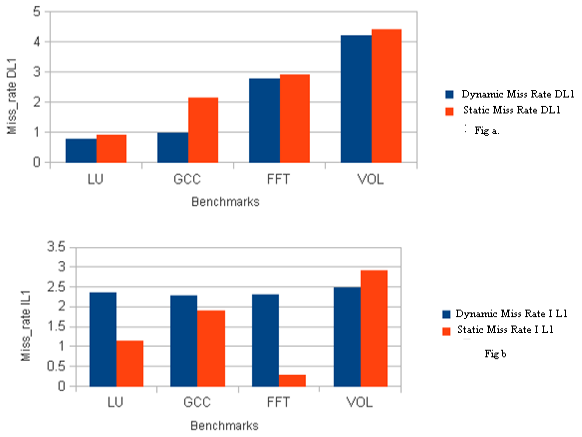


Fig 3. Miss Rate comparison

Fig 3. Shows miss rate, instruction miss rate and data miss rate comparison of static and dynamic cache partitioning on different benchmark application.

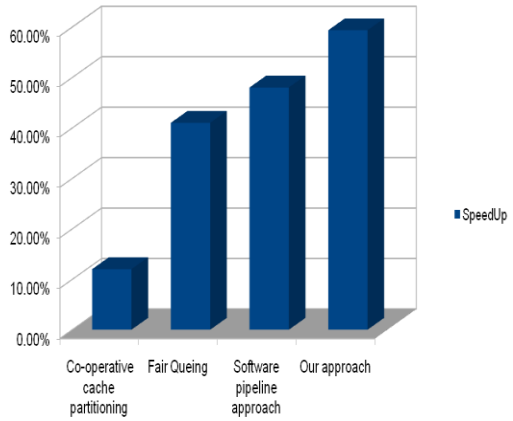


Fig 4. Performance comparison

5. Conclusion

The work presented in this thesis, evaluated cache partitioning as a solution to achieving optimal sharing of the cache amongst the concurrently executing applications and improving the overall system's performance. Initially the effects of Static cache partitioning on the performance of multi-programmed workloads were studied; the miss rate comparison indicated that this replacement policy can result in consistent improvements of instruction miss rate. In contrast to dynamic cache partitioning, statically partitioned caches were found to improve the overall performance of the system. However, these are not practical as they rely on prior knowledge of the characteristics of the applications that are executed.
Therefore several dynamic cache partitioning schemes which attempt to adapt the cache space allocation to the dynamically changing characteristics of the applications was analyzed. This analysis was necessary as the identified advantages and drawbacks were then used to guide the development of an improved cache partitioning scheme.LRU and partitioned cache were evaluated for a different number of cache size sharing a 8KB, 8-way associative L1 cache. Figure 4 presents the overall performance for each case and shows that the gains of cache partitioning over LRU increase with the cache size. This is also evident in Figure 4, where the overall performance compare with the previous algorithms and dynamic cache partitioning for multi-core based real time system.
However, the cache partition scheme is able to allocate cache space to competing processes efficiently by monitoring each running process and adapting dynamically to the characteristics of the multi-programmed workload. Therefore, the utilization of an appropriately partitioned cache can be seen to degrade more slowly than one using the pure, LRU replacement policy. This is important as the current trend is to increase the number of cores integrated on chip, so improving the cache utilization is essential.

References:

[1] K.J. Nesbit et al., "Fair Queuing Memory Systems," Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture, IEEE CS Press, 2006, pp. 208-222.ddd

[2] G. E. Suh, L. Rudolph and S. Devadas, "Dynamic Cache Partitioning for Simultaneous Multithreading Systems", IEEE ISORC 2001, pages 233–240. IEEE Computer Society.

[3] HU WEI, CHEN TIANZHOU, "A Data Centered Approach for Cache Partitioning in Embedded Real-Time Database System, In HPCA-16,2010.

[4] Marco Paolieri1;3, Eduardo Quiˆnones1, Francisco J. Cazorla. A software pipelined approach to multicore execution of timing predictable multi-threaded Hard real time task. In Proceedings of the 14th IEEE ISORC 2011, pages 233–240. IEEE Computer Society

[5] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In HPCA-16, 2010.

[6] Y. Kim et al., "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," Proc. 43rd Ann. IEEE/ACM Int'l Symp. Microarchitecture, IEEE CS Press, 2010.

[7] J. Wolf, M. Gerdes, F. Kluge, S. Uhrig, J. Mische, S. Metzlaff, C. Rochange, H. Casse, P. Sainrat, and T. Ungerer. RTOS Support for Parallel Execution of Hard Real-Time Applications on the MERASA Multi-Core Processor. In Proceedings of the 13th IEEE ISORC 2010,

[8] O. Mutlu and T. Moscibroda, "Parallelism-Aware Batch Scheduling: Enhancing Both Performance and Fairness of Shared DRAM Systems," Proc. 35th Ann. Int'l Symp.Computer Architecture (ISCA 08), IEEE CS Press, 2008, pp. 63-74.