Performance Evaluation of Multicore Cache Locking using Multimedia Applications

Abu Asaduzzaman, Wichita State University, *IEEE Member*

Abstract—**Supporting real-time multimedia applications on multicore systems is a great challenge due to cache's dynamic behavior. Studies show that cache locking may improve execution time predictability and power/performance ratio. However, entire locking at level-1 cache (CL1) may not be efficient if smaller amount of instructions/data compared to the cache size is locked. An alternative choice may be way (i.e., partial) locking. For some processors, way locking is possible only at level-2 cache (CL2). Even though both CL1 cache locking and CL2 cache locking improve predictability, it is difficult to justify the performance and power trade-off between these two cache locking mechanisms. In this work, we assess the impact of CL1 and CL2 cache locking on the performance, power consumption, and predictability of a multicore system using ISO standard H.264/AVC, MPEG4, and MPEG3 multimedia applications and FFT and DFT codes. Simulation results show that both the performance and predictability can be increased and the total power consumption can be decreased by using a cache locking mechanism added to a cache memory hierarchy. Results also show that for the applications used, CL1 cache locking outperforms CL2 cache locking.**

Index Terms—**Cache locking, multicore computer architecture, multimedia applications, performance evaluation**

I. Introduction

MULTICORE computer architecture supporting real-time multimedia applications deals with timing constraints and usually interact with the environment rather than the human operator. Because timeliness and reliability are so important in their behavior, real-time multimedia systems are often distributed among multiple program units (a.k.a., tasks) running simultaneously to perform required functions. Concurrent execution of tasks on a single processor, in many respects including energy and thermal constraints, is inadequate for achieving the required level of performance or required level of reliability. Therefore, the tasks are moved to different interconnected processors, making a real-time system parallel and/or distributed. If the communication time between processing units is negligible with respect to the processing time, then the system is referred as parallel; otherwise it is referred as distributed. Because of high performance and reliability, the popularity and demand of multicore systems (supporting parallel/distributed processing) are increasing in both the desktop and the embedded markets [1]-[3].

Multi-level cache memory hierarchy is a common choice for multicore systems, especially for embedded systems running real-time multimedia applications [4]-[8]. According to this memory hierarchy, CL1s are attached to and privately accessible by each core. A larger CL2 is shared by the cores (e.g., Intel Xeon). Please note that CL2 can be private to the core (like AMD Athlon); but that is beyond the scope of this work. The presence of a shared CL2 offers the flexibility in adjusting the memory allocated per core according to its requirement, as well as the possibility of multiple cores getting fast access to the shared code and/or data. New generation multicore designs have shown that normally two (or more) cores running at (or less than) one half of the frequency can approach the performance of a single core running at full frequency, while the multicore consumes less amount of power. However, the increasing usages of caches potentially increase the execution time unpredictability. Real-time multimedia applications cannot afford to miss deadlines and hence demand timing predictability. Therefore, it becomes a great challenge to support multimedia applications on multicore systems.

Cache locking is introduced in single-core systems to increase the execution time predictability [9]-[19]. Cache locking is the ability to prevent some or all of the cache blocks from being overwritten during runtime. Cache entries can be locked for either an entire cache or for individual ways within the cache. Entire locking (at CL1) is inefficient if the number of instructions or the size of data to be locked is smaller than the cache size. Way locking at the CL1 is not permitted on some processors, but way locking at the CL2 is possible [20]. By locking at CL2, Xeon processor achieves the effect of using local storage by SPEs in Cell processor [21]. Cache locking should be beneficial for multimedia applications. In this work, we model a multicore system with four cores and two levels of caches. Using popular multimedia applications, we assess the effectiveness of cache locking at CL1 and CL2.

This paper is organized as follows. In Section II, some related articles are reviewed. Section III discusses the technique to model and simulate cache locking (at CL1 and CL2). In Section IV, the experimental setup is described. Some important simulation results are presented in Section V. Finally, this work is concluded in Section VI.

Manuscript received on February 15, 2011.

A. Asaduzzaman is with Wichita State University, Wichita, KS 67260-0083 USA (phone: 316-978-5261; fax: 316-978-3984; e-mail: Abu.Asaduzzaman@wichita.edu).

DOI: 10.5176_2010-2283_2.1.137

II. Related Work

Increasing predictability and performance/power ratio of multicore systems has become challenging research area in the recent years. Some published articles, closely related to our work, are discussed in this section.

An invalidation cache lock mechanism is implemented in [13], which utilizes the exclusive state of the snooping cache. Experimental results demonstrate the benefits of the lock mechanism for a few lock contentions and confirm that, in most cases, the lock mechanism works well on the parallel processing machine. However, this mechanism may cause performance degradation in a tightly-coupled multiprocessor system in case of heavy contention.

Various approaches to cope with the predictability problem due to the presence of caches in real-time systems are presented in [10]-[14]. According to these approaches, cache contents are statically locked so as to make memory access time and cache-related preemption delay predictable. However, more study is needed to see the impact of these approaches on performance for larger real benchmarks and the applicability of static cache locking techniques to caches.

Cache locking techniques are introduced by different research groups to improve predictability. In [15], static cache analysis is combined with data cache locking to estimate the worst-case memory performance in a safe, tight, and fast way. Experimental results show that this scheme improves predictability. In [16], a memory hierarchy is proposed to provide high performance combined with high predictability for complex systems. In [17], an algorithm is proposed which partitions the task into a set of regions. Each region owns statically a locked cache content determined offline. A sharp improvement is observed, as compared with a system without any cache. In [18], a methodology to select a set of instructions to be preloaded in the cache using a genetic algorithm is proposed. In [19], various algorithms to select a set of instructions to be locked in cache are compared. These algorithms show better performance and simultaneously estimate a tight upper bound of the response time of tasks. Techniques discussed in [15]-[19] are used mainly to evaluate predictability in a single-core system. These techniques are not capable of power estimation – a crucial design factor for multicore systems. More importantly, these techniques are not adequate to analyze performance, power consumption, and predictability of multicore systems.

An algorithm for off-line selection of the contents of two on-chip memory organizations is proposed in [11]. Experimental results show that the algorithm generates good ratios of on-chip memory accesses on the worst-case execution path for both locked cache and scratchpad memory. However, worst-case performance with locked caches may be degraded with larger cache lines due to cache pollution.

In [9], an efficient memory block selection strategy is presented that can be used to improve the performance of cache memory subsystem. The selected blocks should produce more misses if not locked. Experimental results show that overall cache hit and system performance/power ratio are increased by locking these blocks in the cache.

III. Modeling a Multicore System for Performance Evaluation

Understanding the impact of cache locking on the performance, power consumption, and predictability of multicore systems requires analyzing them separately and observing their interaction with the entire system architecture using the target applications. In the following subsections, we discuss the target architecture, modeling the architecture, simulation of the model, and cache locking techniques.

A. *Target Architecture*

According to the current design trend from Intel, IBM, Sun, and other big chip-vendors, cache memory organization that has two-level caches is very effective for multicore architecture. In this work, we select a popular Intel-like architecture that has four processing cores, CL1s, CL2, and main memory. As shown in Figure 1, each CL1 is split into instruction (I1) and data (D1) caches and CL2 cache is partitioned into two parts. Each part is connected to a group of two cores via a dedicated port. Also, each CL1 is private to its core and CL2 is shared by the cores. We assume the interconnection delay to be negligible making the system more like a parallel system (than a distributed system). We simulate cache locking on CL1 and CL2 on this architecture.

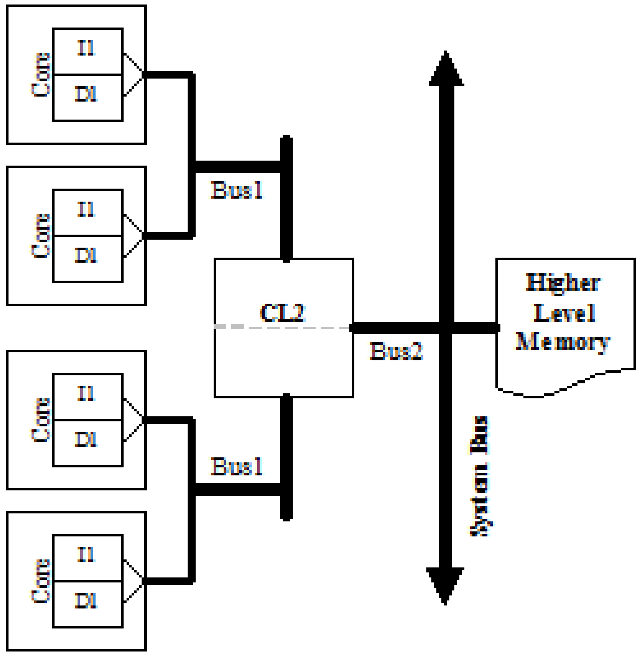

Fig. 1. Simulated multicore architecture

B. *Modeling the Target Architecture*

We abstract the selected architecture by considering only the important components and ignoring any non-relevant minor details. We model the higher level abstraction of the target architecture using VisualSim simulation tool. VisualSim simulator from Mirabilis Design is a graphical simulation tool to build model and execute simulation program [24]. VisualSim contains a complete suite of modeling libraries, simulation engines, report generators, and debugging tools. Model is developed using system components (such as processing core, cache, bus, and memory) and appropriate connections are made between components.

C. *Simulation of the Model*

We develop a simulation platform using the VisualSim model and execute the simulation program using VisualSim simulation cockpit. VisualSim provides simulation cockpit with functionalities to run the simulation program and to collect simulation results. Simulation cockpit can also be used to change the values of the input parameters without modifying the model. The VisualSim results can be stored as text and/or graph files. In this work, we run the simulation program using three representative multimedia applications (H.264/AVC, MPEG4, and MPEG3) and two algorithms (DFT and FFT) by varying cache size, line sizes and associativity levels. We obtain the average delay per task and total power consumed by the system for no locking, CL1 locking, and CL2 locking. Based on the suggestion made in [9], we lock up to 25% of the cache size in order to achieve the maximum performance gain. In the simulation, we randomly select the blocks to be locked.

D. *Cache Locking*

Cache locking is a mechanism that prevents some or all of the instructions or data from being replaced from cache. Cache entries can be locked for either an entire cache or for individual ways within the cache [23].

1) Entire Locking: In entire cache locking, cache hits are treated in the same manner as hits to an unlocked cache. Cache misses are treated as a cache-inhibited access. Invalid cache entries at the time of the locking will remain invalid and inaccessible until the cache is unlocked. Entire cache locking is inefficient if the number of instructions or the size of data to be locked is small compared to the cache size.

2) Way Locking: In way locking, only a portion of the cache is locked by locking ways within the cache. Invalid entries in way locking are accessible and available for data placement – this behavior differs from entire cache locking. Unlocked ways of the cache behave normally. Way cache locking is a potential alternative of entire cache locking. Way locking is more suitable in multicore architecture. In this work, we use way cache locking at CL1 and Cl2.

IV. Experimental Setup

In this work, we model and simulate a multicore system using multimedia applications to investigate the impact of CL1 and CL2 cache locking on the performance, power consumption, and predictability. Discussed below are the assumptions, workloads, and input/out parameters related to the simulation program used in this work.

A. *Assumptions*

Important assumptions for modeling the target architecture and for running the simulation program include the following:

-- Cache locking at CL1 and CL2 is implemented. Both CL1 and CL2 locking are not applied at the same time.

-- For both CL1 and CL2, write-back memory update policy and random cache replacement strategy are used as required.

-- The delay introduced by the bus that connects CL2 and the main memory (Bus2 in Figure 1) is 10 times longer than the delay introduced by the bus that connects CL1s and CL2 (Bus1 in Figure 1).

B. *Workloads*

In this work, we use workloads of three ISO standard popular multimedia applications and two important algorithms to run the simulation program. Multimedia applications are: Advanced Video Coding – widely known as H.264/AVC, Moving Picture Experts Group's MPEG3, and MPEG4 (part-2). Algorithms are: Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT). Table I shows some important characteristics of the applications. The workloads are generated using Heptane simulation package [25].

TABLE I
CHARACTERISTICS OF THE APPLICATIONS

Applications	Code Size (Byte)	Number of Instructions
H.264/AVC	156274	5207118
MPEG3	196305	6258432
MPEG4	182736	5772085
DFT	1165	287209
FFT	2335	365184

C. *Input / Output Parameters*

Important input parameters used in this simulation program include CL2 cache size and CL1 and CL2 line sizes and associativity levels [see Table II].

TABLE II
INPUT PARAMETERS

Parameter	Value
CL2 cache size (KB)	64, 128, 256, 512, or 1024
CL1/CL2 line size (Byte)	16, 32, 64, 128, or 256
CL1/CL2 associativity level	1-, 2-, 4-, 8-, or 16-way

We keep CL1 cache size fixed at I1 = 4KB and D1 = 4KB and change CL1/CL2 line size and associativity level. We obtain mean delay per task and total power consumption by the system as the output parameters. Delay is the time between the start of execution of a task and its end [24]. Mean delay is the average delay for all the tasks.

V. Results and Discussion

In this work, we evaluate the impact of cache locking at CL1 and CL2 on the performance, power consumption, and predictability in a multicore system running H.264/AVC, MPRG4, MPEG3, DFT, and FFT workloads. Cache locking improves the predictability by making the block local and closer to the cores. However, aggressive cache locking may decrease the performance/power ratio due to the reduction of the effective cache size. In both CL1 and CL2 cache locking, we lock 25% of the cache size (as suggested in [9]) and randomly select the blocks to be locked. Some important simulation results are presented in the following subsections.

A. *CL2 Cache Size*

We first investigate the impact of cache locking at CL1 and CL2 on delay (i.e., performance) and power consumption by varying CL2 cache size. The average delay per task for no cache locking and cache locking at CL1 and CL2 is shown in Figure 2. Experimental results show that for any CL2 cache size, mean delay per task for H.264/AVC, MPRG4, and MPRG3 decreases when we move from no locking to cache locking. Mean delay per task increases when we move from CL1 locking to CL2 locking for the used applications. Results also indicate that cache locking has no positive impact on FFT and DFT codes as they entirely fit inside I1 cache.

Figure 3 illustrates the impact of CL1 and CL2 cache locking on total power consumption. Up to CL2 cache size 256KB, total power consumption for H.264/AVC, MPRG4, and MPRG3 decreases when cache locking is applied. However, total power consumption increases for CL2 larger than 256KB. It is noted that CL1 locking has more impact on total power consumption than CL2 locking for the used applications. For FFT and DFT, total power consumption increases as CL2 cache size increases from 64KB.

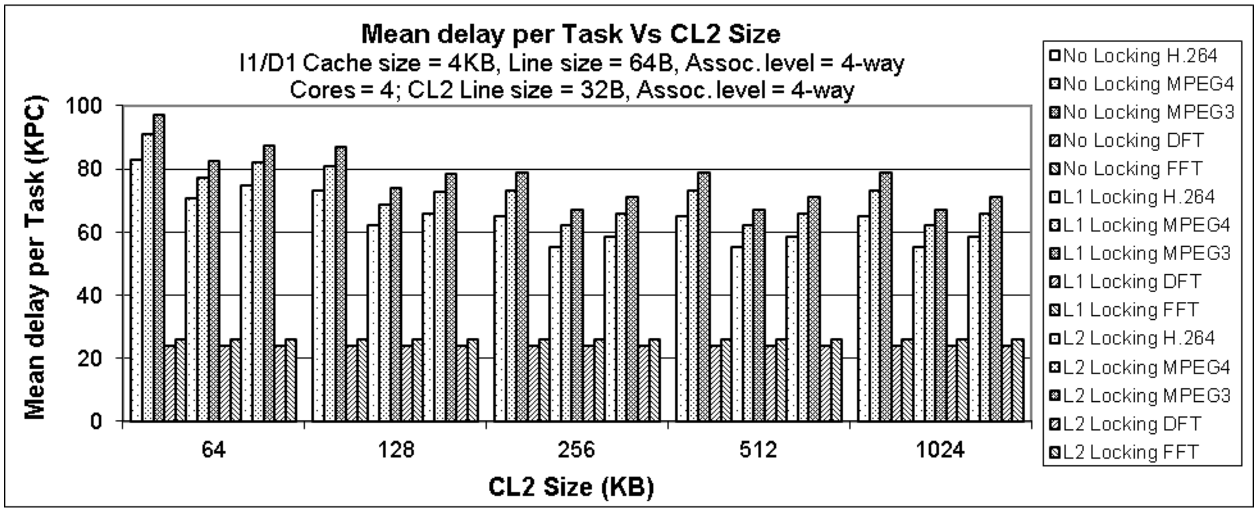

Fig. 2. Mean delay per task Vs. L2 cache size

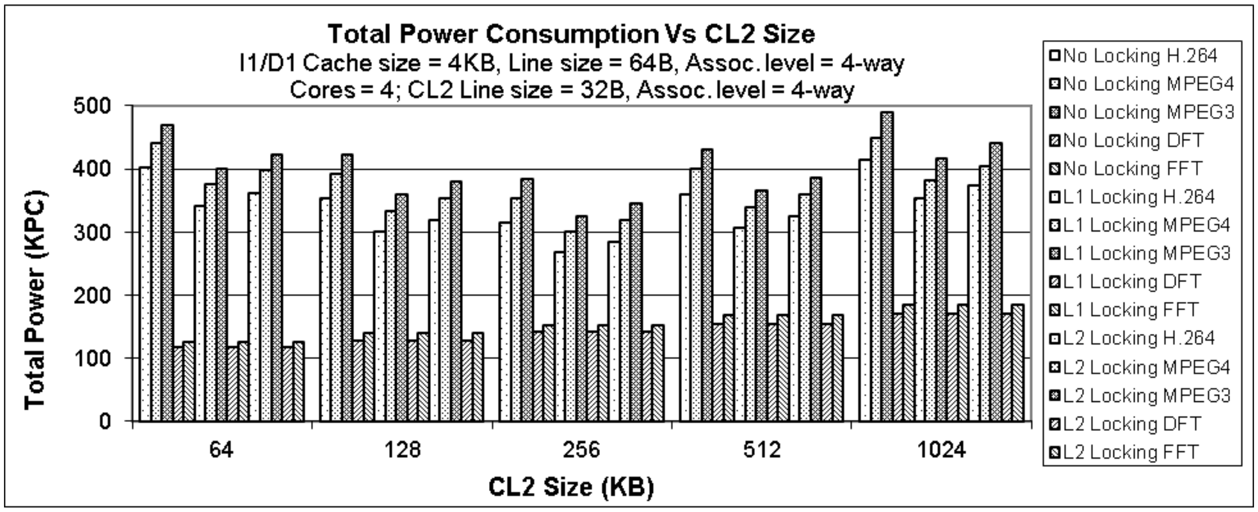

Fig. 3. Total power consumption Vs. L2 cache size

B. *CL1 and CL2 Line Sizes*

We now discuss the impact of CL1 cache locking and CL2 cache locking on delay and power consumption by varying both CL1 and CL2 line size. We discard the results due to FFT and DFT as cache locking has no positive impact on small applications like them. The average delay per task for no locking, CL1 locking, and CL2 locking is shown in Figure 4.

Experimental results also show that for any line size, mean delay per task for H.264/AVC, MPRG4, and MPRG3 decreases when we move from no locking to cache locking. Comparing the impact of CL1 and CL2 cache locking, mean delay per task decreases more for CL1 locking than CL2 locking. It is noted that the mean delay per task goes down with increasing line size leveling off at a line size of 64B. This is because of the cache pollution due to larger cache line.

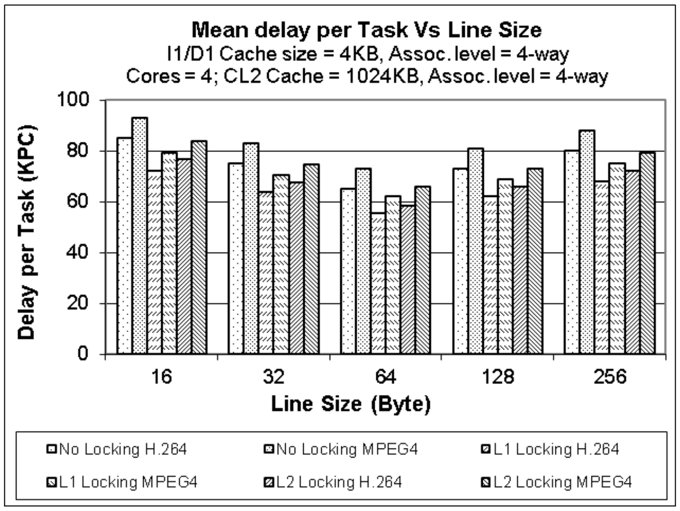

Fig. 4. Mean delay per task Vs. line size

Figure 5 shows the total power consumption for no cache locking, CL1 cache locking, and CL2 cache locking for various line sizes. Simulation results reveal that the total power consumption goes down for all three applications used with the increase in line size for line size between 16B and 64B; after that total power consumption increases. Like mean delay per task, total power consumption decreases when cache locking is applied and CL1 cache locking outperforms CL2 cache locking.

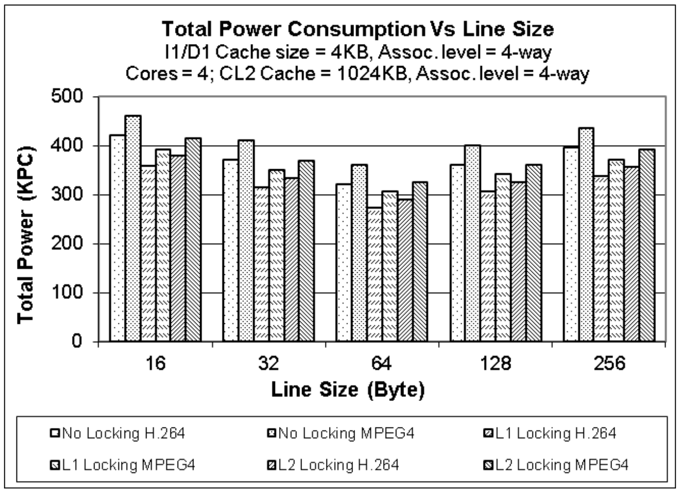

Fig. 5. Total power consumption Vs. line size

C. *CL1 and CL2 Associativity Levels*

Finally, we present the impact of CL1 and CL2 cache locking on delay and power consumption due to different CL1 and CL2 associativity levels. The average delay per task for without and with cache locking is shown in Figure 6. From simulation results, it is noticed that the mean delay per task decreases significantly as associativity level increases up to 4-way. It is also noticed that for H.264/AVC, MPRG4, and MPRG3, mean delay per task decreases when we move from no locking to cache locking. Simulation results also indicate that CL1 cache locking has more impact on the average delay per task than CL2 cache locking.

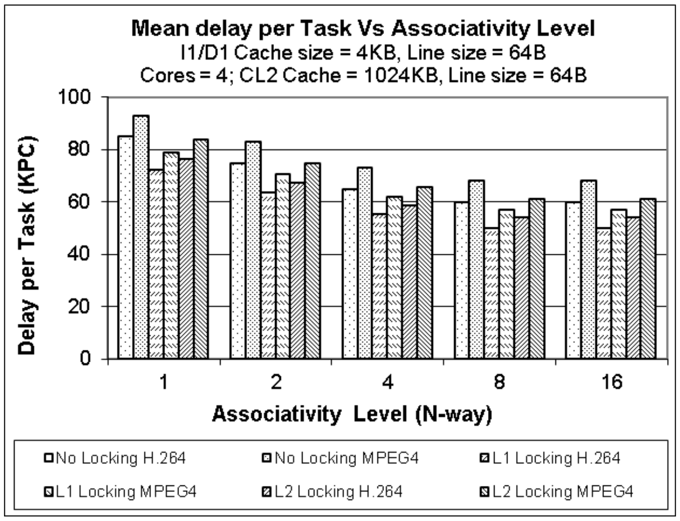

Fig. 6. Mean delay per task Vs. associativity level

Finally, Figure 7 shows the total power consumption for no cache locking and CL1/CL2 cache locking for various associativity levels. Simulation results show that for all three applications used, the total power consumption goes down sharply with the increase in associativity level for associativity level up to 4-way; after that total power consumption remains almost the same. It is also observed that the total power consumption decreases when cache is locked and CL1 cache locking outperforms CL2 cache locking for H.264/AVC, MPRG4, and MPRG3.

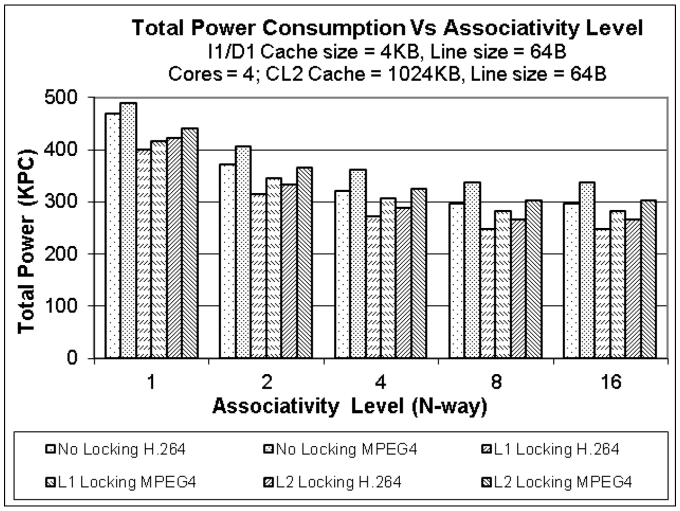

Fig. 7. Total power consumption Vs. associativity level

VI. CONCLUSION

Demands for supporting real-time multimedia applications are growing more than ever. Computing systems are adopting multicore architecture to meet the requirements for high processing speed and low power consumption. However, the presence of multiple caches in multicore architecture makes the execution time predictability even worse and total power consumption even more. Studies show that predictability can be improved using cache locking techniques. Cache locking can be entire (all cache blocks are locked) or way (in a set-associative or fully associative cache, only certain ways are locked). In a single-core system, cache locking can be implemented at CL1 or CL2. Some processors do not allow way locking at CL1, but they allow way locking at CL2. Although both CL1 and CL2 cache locking may improve predictability, it is difficult to justify the performance and power trade-off between CL1 and CL2 cache locking mechanisms.

In this work, we assess the impact of CL1 and CL2 cache locking on the performance, power consumption, and predictability of a multicore system running multimedia applications. We model and simulate a system with four cores and two levels of caches. Three popular ISO standard multimedia applications (H.264/AVC, MPEG4, and MPRG3) and two important algorithms (DFT and FFT) are used to run the simulation program. As cache locking holds the blocks in the cache for the entire execution time, predictability is enhanced by applying cache locking. Experimental results show that the mean delay per task and total power consumption can be decreased by using a cache locking mechanism added to an efficient cache memory organization. It is observed from the simulation results that for large applications like MPEG4, CL1 cache locking performs better than CL2 cache locking. This is because the main loop of these applications (like MPEG4) has a better fit in the CL1 cache. It is also observed that for small algorithms like FFT and DFT, cache locking (at CL1 or CL2) has no positive impact on predictability and performance/power ratio.

We plan to investigate the impact of a multipurpose victim cache without cache locking on performance, power consumption, and predictability of a multicore architecture in our next endeavor.

REFERENCES

[1] J.P. Lozi, G. Thomas, J. Lawall, G. Muller. (2011). Efficient locking for multicore architectures. *RESEARCH REPORT N 7779 Project-Teams REGAL*. Available: http://hal.upmc.fr/docs/00/64/12/52/PDF/rr7779.pdf

[2] V. Suhendra, T. Mitra. (2008). Exploring Locking & Partitioning for Predictable Shared Caches on Multi-Cores. *DAC'2008*. Anaheim, CA.

[3] B. Pfarr, A. Zimmerman, S. Brandt. (2002). Parallel and Distributed Real-Time Systems. *Scientific International Journal for Parallel and Distributed Computing*. Vol. 5, No. 1.

[4] R.M. Ramanathan. (2006). Intel Multi-Core Processors: Making the Move to Quad-Core and Beyond. *White Paper*.

[5] V. Romanchenko. (2006). Quad-Core Opteron: architecture and roadmaps. *Digital-Daily.com*.

[6] V. Romanchenko. (2006). Evaluation of the multi-core processor architecture Intel core: Conroe, Kentsfield…*Digital-Daily.com*.

[7] Multi-core (computing). (2012). *Wikipedia*. Available: http://en.wikipedia.org/wiki/Xeon; http://en.wikipedia.org/wiki/Athlon

[8] D.K. Every. (2005). IBM's Cell Processor: The next generation of computing? *Shareware Press*. Available: http://www.mymac.com/fileupload/CellProcessor.pdf

[9] A. Asaduzzaman. (2011). An Efficient Memory Block Selection Strategy to Improve the Performance of Cache Memory Subsystem. *ICCIT-2011*. Bangladesh.

[10] I. Puaut. (2007). Cache Analysis Vs Static Cache Locking for Schedulability Analysis in Multitasking Real-Time Systems. Available: http://citeseer.ist.psu.edu/534615.html

[11] I. Puaut, C. Pais. (2007). Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. *Design, Automation & Test in Europe Conference & Exhibition (DATE'07)*. pp. 1-6.

[12] A.M. Campoy, E. Tamura, S. Saez, F. Rodriguez, J.V. Busquets Mataix. (2005). On Using Locking Caches in Embedded Real-Time Systems. *ICESS-05, LNCS 3820*. pp. 150-159.

[13] T. Tarui, T. Nakagawa, N. Ido, M. Asaie, M. Sugie. (1992). Evaluation of the lock mechanism in a snooping cache. *ACM Proceedings of the 6th international conference on Supercomputing*.

[14] I. Puaut, D. Decotigny. (2002). Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems. *IEEE*.

[15] X. Vera, B. Lisper. (2003). Data Cache Locking for Higher Program Predictability. *SIGMETRICS'03*. CA.

[16] E. Tamura, F. Rodriguez, J.V. Busquets-Mataix, A.M. Campoy. (2004). High Performance Memory Architectures with Dynamic Locking Cache for Real-Time Systems. *Proceedings of the 16th Euromicro Conference on Real-Time Systems*. pp. 1-4, Italy.

[17] A. Arnaud, I. Puaut. (2005). Dynamic Instruction Cache Locking in Hard Real-Time Systems. *IEEE*.

[18] A.M. Campoy, A.P. Jimenez, A.P. Ivars, J.V. Busquets Mataix. (2001). Using Genetic Algorithms in Content Selection for Locking-Caches. *Proceedings of IASTED International Symposia Applied Informatics*. pp. 271-276, Austria.

[19] E. Tamura, J.V. Busquets-Mataix, J.J.S. Martin, A.M. Campoy. (2005). A Comparison of Three Genetic Algorithms for Locking-Cache Contents Selection in Real-Time Systems. *Proceedings of the International Conference in Coimbra*, Portugal.

[20] C. Harrison. (2005). Programming the cache on the PowerPC 750GX/FX - Use cache management instructions to improve performance. *IBM Microcontroller Applications Group*. Available: http://www-128.ibm.com/developerworks/library/pa-ppccache.html

[21] J. Stokes. (2005). Xenon's L2 vs. Cell's local storage, and some notes on IBM/Nintendo's Gekko. Available: http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars/6

[22] S.S. Mukherjee, S.V. Adve, T. Austin, J. Emer, P.S. Magnusson. (2002). Performance Simulation Tools. *IEEE Computer*.

[23] (2008). MPC8272 PowerQUICC II – Family Reference Manual. Available: http://www.freescale.com/files/32bit/doc/ref_manual/MPC8272RM.pdf

[24] VisualSim Architecture. (2012). VisualSim – A system-level simulator from Mirabilis Design, Inc. Available: http://www.mirabilisdesign.com/

[25] Heptane. (2012). A tree-based WCET analysis tool. Available: http://ralyx.inria.fr/2004/Raweb/aces/uid43.htm

Abu Asaduzzaman (M'96) received the Ph.D. and M.S. degrees, both in computer engineering, from Florida Atlantic University, Florida in 2009 and 1997. He received the B.S. degree in electrical engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 1993.

Currently, Abu is working as an Assistant Professor in EECS department at Wichita State University, Kansas. Previously he worked at FAU. His research interests include computer architecture, embedded systems, parallel computing, and performance evaluation.

Mr. Asaduzzaman is a member of the IEEE, ASEE, and various prestigious honor societies He served as Session Chair and TPC/IPC member of various major conferences and as reviewer of NSF TUES (CS) and GRFP programs.