EARLY ESTIMATION OF CACHE PROPERTIES FOR MULTICORE EMBEDDED PROCESSORS

[1]KISHORE K. CHIDELLA, [2]MUHAMMAD F. MRIDHA, [3]ABU ASADUZZAMAN

[1,3]Wichita State University, [2]University of Asia Pacific,
E-mail: [1]kkchidella@wichita.edu, [2]firoz@uap-bd.edu, [3]Abu.Asaduzzaman@wichita.edu,

Abstract—The state-of-the-art embedded systems are expected to have multicore processors as multicore architecture provides high performance to power ratio. Although cache improves the overall performance, designing multicore embedded processors with multilevel caches is a great challenge. Caches make thermal constraint crucial; parallel thread execution difficult; and timing unpredictability even worse. An effective early estimation technique can be very valuable to design complex systems like multicore embedded systems. In this paper, we propose a simulation methodology to determine the impact of cache on performance, power consumption, and predictability to facilitate the design of future embedded multicore systems. Effective cache parameters (such as cache size), organizations (such as shared CL2), and techniques (such as cache locking) for target applications can be predetermined using this method. We model a quad-core embedded system with two levels of caches (where CL2 is shared). By varying total CL2 cache size and locked CL2 cache size, we run the simulation program using popular FFT, GIF, JPEG, MPEG-3, and MPEG-4 workloads. Simulation results indicate that the optimal value of CL2 is 128 KB in this experiment for the selected applications when average memory access time (i.e., delay) per task and total power consumption are concerned. Experimental results also indicate that up to 25% CL2 cache locking is helpful for the simulated system. It is observed that both mean delay per task and total power consumption decrease when cache size is increased and/or 25% cache locking is applied; however, the impact of shared CL2 cache on power consumption is more significant than that on mean delay.

Index Terms—Average Memory Access Time (Delay), Cache Memory, Embedded Systems, Multicore Processors, Power Consumption.

I. INTRODUCTION

To move from current state-of-the-art embedded systems to the advanced level, embedded systems design methodology must need supports from early estimation techniques. Due to requirements for more processing speed and advancement in semiconductor technology, future embedded systems should have multicore processor. Currently available single-core based modeling and simulation techniques are not adequate to design modern multicore embedded systems [1-4]. According to multicore architecture design techniques, four cores running at one fourth of the frequency can approach the performance of a single-core running at full frequency, while the quad-core power consumption is less.

As the number of cores in the processors is increasing, software applications are having more and more threads to take advantage of the available cores [5-8]. Multicore architecture in embedded systems has proven the potential of supporting multithreaded parallel processing. Multicore processors are frequently deployed with multilevel cache memories [9]. Parallel thread execution to achieve the best performance in such a multicore system is difficult as it relates to cache sharing. Multilevel caches also make the execution time unpredictability worse. Therefore, supporting real-time and/or multithreaded applications on multicore embedded systems with multilevel caches becomes an enormous challenge. In a real-time embedded system, hardware requests must be satisfied in a timely manner regardless of the system load. In a simultaneous multithreading system, multiple threads should be executing (on multiple cores) at the same processor cycle. Real-time and/or multithreaded applications can be efficiently supported on multicore systems by effectively using the cache memory subsystem. Studies show that bigger cache size may improve performance and predictability; however, it needs more power and area, which is not desirable in embedded system design. Recent studies also show that smaller amount of cache locking may improve performance/power ratio and predictability [10]. Cache locking allows preloading memory blocks into the cache and subsequently prevents these blocks from being replaced during runtime. Cache locking at CL1 has some limitations. For example, some processors (like PowerPC 750GX) do not permit entire cache locking at CL1 [11]. Cache locking at shared level (like CL2 or CL3), specifically for multicore architecture, may be a promising alternative. This cache locking strategy is scalable as no additional logic is needed (to the cache locking mechanism) if more cores are added to the system.

This paper is organized as follows. In Section II, some related articles are reviewed. Section III briefly describes cache size and cache locking impacts on performance and power consumption. Proposed cache modeling strategy for embedded multicore systems is presented in Section IV. In Section V, the simulation details are briefly discussed. Some important simulation results are depictedin Section VI. Finally, this work is concluded in Section VII.

II. RELATED WORK

Some relevant articles are discussed in this section.

In [1], the technical challenges associated with the integration of homogeneous and heterogeneous multiple cores in embedded systems is elucidated. However, this book does not provide a viable way to make early estimation on future embedded systems design.

The impact of cache parameters on the power consumption and performance of general purpose multicore systems is investigated in [4]. Results show that cache parameters and the application code size have impact on total power consumption and mean delay per task. This approach is not focused on designing embedded systems and does not cover the cache locking aspect.

Issues related to cache locking at level-1 and level-2 caches are discussed in [11]. Entire (100% of the cache size) level-1 cache locking is not efficient for some applications, especially when the data size to be locked is smaller compared to the cache size.

An algorithm for off-line selection of the contents of two on-chip memories, locked caches and scratchpad memories, are proposed in [12]. Experimental results show that the algorithm generates good ratios of on-chip memory accesses on the worst-case execution path. However, worst-case performance with locked caches may degrade with large cache lines due to cache pollution. In [13], a memory hierarchy is proposed to provide high performance combined with high predictability for complex systems. In [14], various algorithms to select a set of instructions to be locked in cache are compared. The solutions mentioned in [12-14] show that performance improvement can be used to assess a tight upper bound of the response time of tasks. However, these techniques are developed for single-core systems and not suitable for contemporary multicore embedded systems. Also, these techniques are not useful to estimate power consumption, a crucial design factor for embedded systems. Therefore, these techniques are not adequate to analyze the performance, power consumption, and predictability of advanced embedded systems.

III. CACHE SIZE AND CACHE LOCKING

Cache size has direct impact on cache(capacity) misses. Similarly, cache locking has significant impact on cache misses. Future embedded multicore architectures are expected to have multilevel cache memory organizations. Therefore, we briefly describe the multicore architectures before we present the proposed strategy to model and simulate multicore cache subsystem.

A. Contemporary Multicore Cache Organizations

Before year 2005, single-core processors dominated the processor market. AMD produced Athlon 64 X2 (dual-core) CPU in year 2005. Since year 2006, multicore processors are available commercially. Today, there are so many multicore chips available from different chip vendors. As Intel and AMD are the two mainstream chip manufacturers of multicore processors, we choose two of their commonly available multicore processors, Intel Xeon quad-core (with shared CL2) and AMD Opteron quad-core (with shared CL3), to discuss. However, we model only Intel-like quad-core system with shared CL2 cache.

Shared CL2 in Multicore Architecture: There are many different types of multicore processors from Intel. Figure 1 illustrates Intel Xeon DP like processors. Xeon DP is a homogenous quad-core processor. It holds a shared memory model with private CL1s and shared CL2. Each private CL1 is split into I1 (size 128 KB) and D1 (size 128 KB) and shared CL2 is unified (size 8 MB) [15]. We implement our proposed cache locking at CL2 in Intel-like architecture.

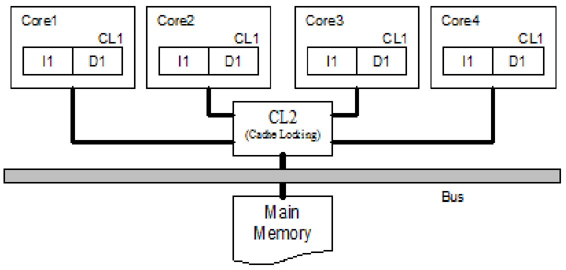

Figure 1: Intel-like quad-core architecture with private CL1 and shared CL2

Shared CL3 in Multicore Architecture: Like Intel, AMD also has many different types of multicore processors. Figure 2 depicts AMD Opteron (quad-core) like processors. Opteron quad-core has a shared memory model with private CL1s, private CL2s, and shared CL3. Each private CL1 is split into I1 (size 256 KB) and D1 (size 256 KB), private CL2 is unified (size 2 MB), and shared CL3 is unified (size up to 4 MB) [16]. We implement our proposed cache locking at CL3 in AMD-like architecture.

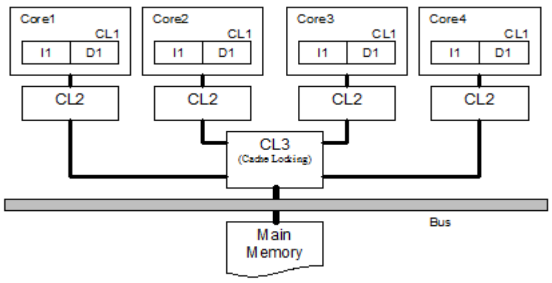

Figure 2: AMD-like quad-core architecture with private CL1 and CL2 and shared CL3

B. Cache Locking

Cache locking is a technique to hold some or all blocks in a cache during the entire execution time. This mechanism prevents some or all of the instruction cache (or data cache) from being overwritten by the cache replacement policy. It is proven to improve execution time predictability and performance/power ratio in single-core and multicore systems. There are two types of cache locking: lock either an entire cache or individual ways within the cache [17]. In way

locking, only a portion of the cache (one or more, not all, ways within the cache) is locked and the remaining unlocked cache performs normally. Unlike entire cache locking (where all entries in a cache are locked and cannot be replaced during the execution time), invalid entries in way locking are accessible and available for data placement. Way cache locking is suitable for most processors and applications. Cache hits are treated in the same manner as hits to an unlocked cache. Cache locking can be implemented at level-1 or at a higher level. In multicore systems, cache locking at shared cache (like CL2 or CL3) is preferable, because it helps reduce cache inconsistency problem and keeps cache locking scheme simple and scalable.
In this work, we concentrate on shared CL2 cache modeling for embedded multicore systems.

IV. PROPOSED CACHE MODELING STRATEGY

In this section, we present our proposed shared cache modeling strategy for multicore embedded systems. According to this methodology, shared CL2 cache size can be changed and/or some blocks (i.e., ways) in shared CL2 cab be locked. Memory blocks are randomly selected for cache locking. The decision about shared cache size and/or locking is done at the master core (at CPU level) before the jobs are actually assigned to the cores. If it is decided to implement cache locking, selected blocks are locked at CL2. There is no cache locking at core level in CL1 (i.e., in I1 or D1). The locked cache remains locked until all the jobs are completed. Figure 3 illustrates proposed cache locking strategy. Depending on the job properties and available cores, jobs are selected for processing. Selected jobs are assigned among the cores and each core is preloaded according to the assigned job. Each core completes its job independently. To calculate average delay per task, the maximum delay is considered for each batch of jobs. To calculate total power, power consumed by each core to complete all tasks is considered. After completion of a batch of jobs, another batch is selected. At that time, caches are re-preloaded according to the newly selected jobs. Mean delay per task is obtained by dividing the total delay by the number of tasks. Total power consumption is obtained by adding all the power consumed by all the tasks of all jobs.

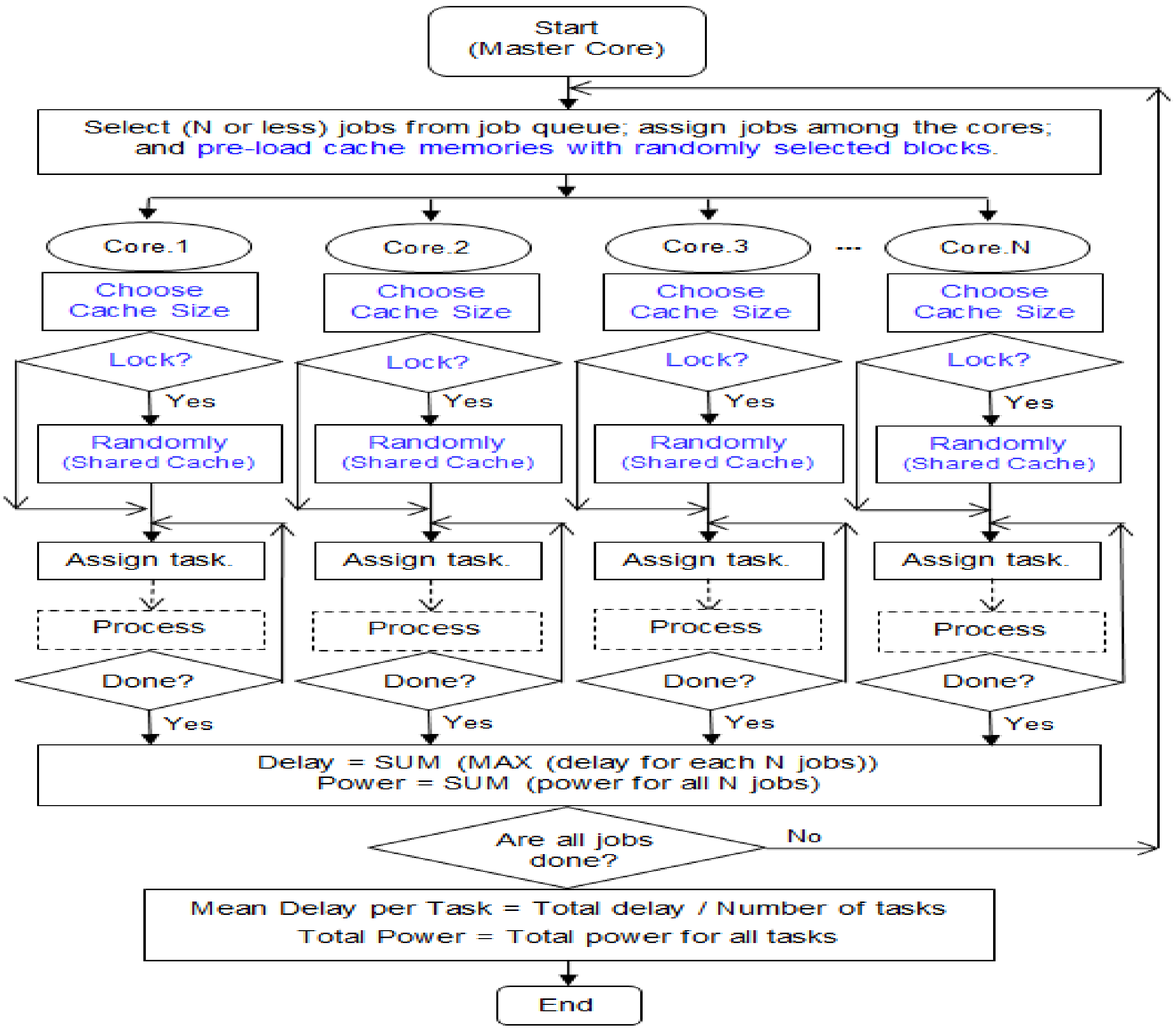
Start
(Master Core)
Select (N or less) jobs from job queue; assign jobs among the cores; and pre-load cache memories with randomly selected blocks.
Core 1
Core 2
Core 3
Core N
Choose Cache Size
Lock?
Yes
Randomly (Shared Cache)
Assign task.
Process
Done?
Yes
Delay = SUM (MAX (delay for each N jobs))
Power = SUM (power for all N jobs)
Are all jobs done?
No
Mean Delay per Task = Total delay / Number of tasks
Total Power = Total power for all tasks
End

Figure 3: Work flow diagram of shared cache locking strategy for N-core architecture

V. SIMULATION DETAILS

In this work, we introduce a simulation methodology to determine the right cache memory organization to facilitate the design of future embedded multicore systems. In this section, we briefly discuss the simulation details including the assumptions, workloads, and input/output parameters.

A. Assumptions

Important assumptions include:

- Cache modeling at shared CL2 (for Intel-like quad-core) is considered.
- The master core (at CPU level) decides the CL2 cache size and whether cache locking should be implemented or not at shared CL2.
- Memory blocks are selected randomly for cache preloading and cache locking.
- All cores are homogenous. CL1 cache parameters used are the same for all cores of a multicore architecture.
- The delay introduced by the bus that connects shared CL2 to main memory is considered 15 times longer than the delay introduced by the bus that connects caches (CL1 to CL2).
- Write-back memory update policy is used.

B. Workloads

We consider workloads from the following popular applications in the work: FFT (Fast Fourier Transform), GIF (Graphics Interchange Format), JPEG (Joint Photographic Experts Group), MPEG (Moving Picture Experts Group)-3, and MPEG-4. Here, FFT is the smallest application (with code size 2.34 KB) and MPEG-4 is the biggest application (with code size 91.83 KB). We use VisualSim tool [18] to develop the modeling platform.

C. Input and Output Parameters

Important input parameters are shown in Table 1.

Table 1: Input parameters

Name	Value
Number of cores	4 (fixed)
I1 / D1 size (KB)	2 / 2 (fixed)
Line size (Byte)	128 (fixed)
Associativity level (n-way)	8 (fixed)
CL2 cache size (KB)	32, 64, 128, 256, or 512
Locked CL2 cache size (%)	0.0, 12.5, 25.0, 37.5, 50.0

Output parameters we obtain from VisualSim are mean delay per task (to represent performance) and total power consumed by the system. Delay is the time between the start of execution of a task and the end. Mean delay is the average delay of all the tasks.

VI. RESULTS ANDDISCUSSION

In this section, we present some important simulation results showing the impact of CL2 cache size and cache locking on performance and power consumption. We simulate a quad-core embedded system using FFT, GIF, JPEG, MPEG-3, and MPEG-4 workloads. First, we discuss the impact of shared CL2 cache size, followed by the impact of shared CL2 cache locking.

A. Shared L2 Cache Size

The capacity misses should decrease with the increase of cache size. We examine the phenomenon: how shared CL2 cache size impacts on mean delay per task and total power consumption (and shared CL2 cache locking is applied). Figures 4 and 5 illustrate the average delay per task and total power consumption, respectively, for various CL2 cache size starting at 32 KB. Results due to JPEG and MPEG-3 are excluded, because JPEG behaves almost like GIF and MPEG-3 behaves almost like MPEG-4. Experimental results show that for CL2 cache size 32 KB to 128 KB, mean delay per task and total power consumption for MPEG-4 decrease significantly when we increase cache size and/or move from no locking to 25% locking. Only for CL2 cache size 32 KB, mean delay per task and total power consumption for GIF decrease when 25% locking is applied. However, CL2 cache size/locking has no positive impact on mean delay per task and total power consumption for FFT. This is because FFT code totally fits in CL2 and the impact of cache locking depends on whether the code fits in the cache or not. Increasing CL2 size beyond 128 KB has no positive impact (consumes more power without reducing the delay).

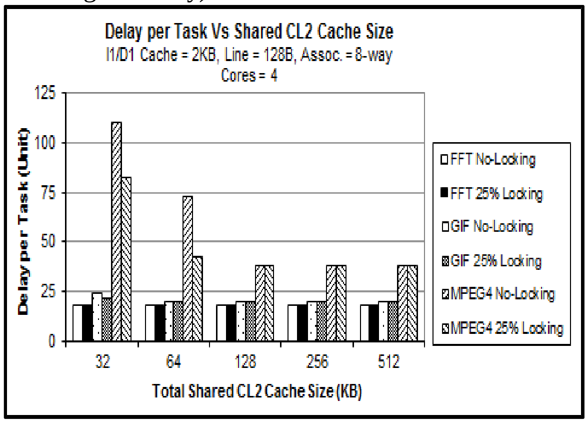

Figure 4: Impact of total shared CL2 cache size on mean delay per task

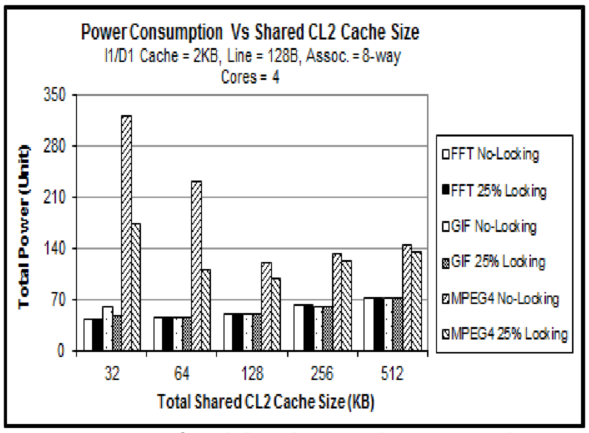

Figure 5: Impact of total shared CL2 cache size on total power consumption

The optimal performance/power (i.e., delay/power) ratio is found for 128 KB shared CL2 for all the workloads (see Figure 6). For CL2 bigger than 128 KB, more power is consumed but the mean delay per task is not reduced.

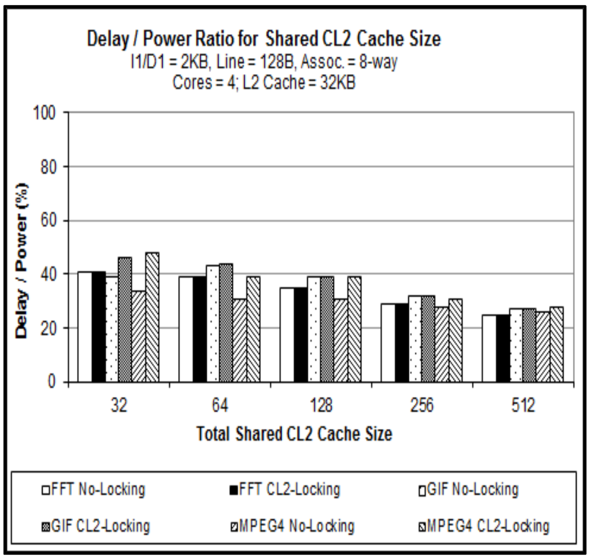

Figure 6: Impact of total shared CL2 cache size on mean delay per task / total power consumption ratio

B. Shared L2 Cache Locking

Figure 7 shows mean delay per task for all five workloads with various locked CL2 cache size (0% to 50% locking). Simulation results show that cache locking at shared CL2 has significant impact on large applications (like MPEG-3 and MPEG-4) than small applications (like FFT). For MPEG-4, mean delay per task decreases sharply as we move from no locking to 25% locked CL2 cache size; mean delay per task start increasing as we move beyond 25% CL2 locking. The impact of shared CL2 cache locking on mean delay for GIF and JPEG is not very significant. For FFT, there is no positive impact of shared CL2 cache locking on mean delay and power consumption. Again, this is because FFT code entirely fits inside CL2 cache but MPEG-4 does not.

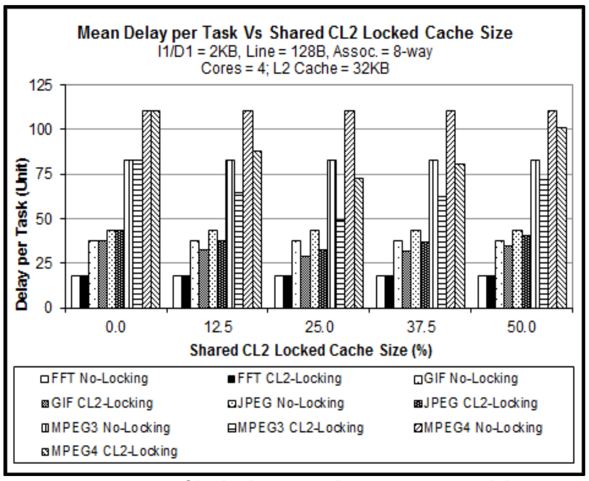

Figure 7: Impact of locked CL2 cache size on mean delay per task

Similarly, cache locking at shared CL2 has significant impact on total power consumption for large applications (like MPEG-4) than small applications (like FFT) as shown in Figure 8. However, cache locking beyond 25% CL2 size is not beneficial for any workloads.

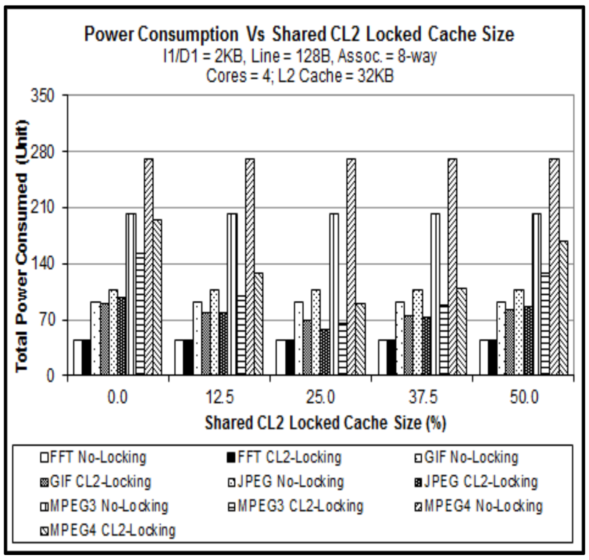

Figure 8: Impact of locked CL2 cache size on total power consumption

According to shared CL2 cache locking results, the optimal performance (delay)/power ratio is obtained for 25% cache locking for all the workloads (see Figure 9).

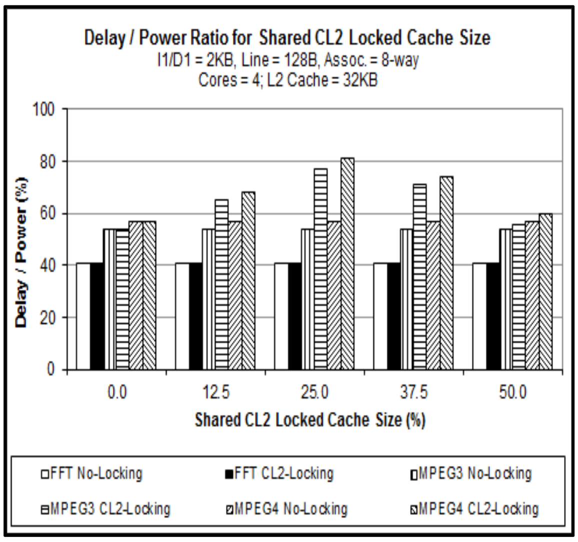

Figure 9: Impact of locked shared CL2 cache size on mean delay per task / total power consumption ratio

CONCLUSIONS

The presence of multilevel caches pose tremendous challenge in designing state-of-the-art embedded multicore systems as caches are power hungry and make thread level parallelism difficult. Therefore, the caches memory organization needs to be selected carefully before building the multicore embedded systems. In this paper, we present a simulation methodology to early estimate the effective cache parameters (like cache size), organizations (like shared CL2), and techniques (like cache locking) for target applications to facilitate the design of future embedded multicore systems. An Intel-like quad-core with shared CL2 is simulated using FFT, GIF, JPEG, MPEG-3, and MPEG-4 workloads. Results (mean delay per task and total power consumption) are obtained by varying CL2 cache size and locked CL2 cache size.

As shared CL2 cache size is normally bigger than CL1, more blocks (that might create cache misses) can be stored and locked in CL2. Consequently, execution time predictability is enhanced. Cache locking at higher level shared cache gives an opportunity to make the system almost entirely predictable, if needed, by locking more blocks. In this experiment, 128 KB shared CL2 cache provides the optimal mean delay per task and total power consumption for the target applications (see Figures 4, 5, and 6). Simulation results also indicate that 25% CL2 cache locking is the best in this experiment. Locking more than 25% shared CL2 may cause additional delay and may need additional power (see Figures 7, 8, and 9). Finally, albeit both mean delay per task and total power consumption decrease when shared CL2 cache size is increased and/or cache locking is applied, it is noted that the impact of shared CL2 on power consumption is more significant than that on delay.

We plan to explore the impact of shared cache at various levels (like shared CL2 and shared CL3) on performance, total power consumption, and predictability for multicore embedded systems in our next endeavor.

REFERENCES

[1] G. Kornaros, “Title Multi-Core Embedded Systems,” CRC Press, 2010.

[2] G. Reinman and N.P. Jouppi, “An Integrated Cache Timing and Power Model,” Western Research Laboratory, California, 2000.

[3] D. Fittes, “Using Multicore Processors in Embedded Systems,”Hitex UK Ltd. Warwick University Science Park, 2009.

[4] A. Asaduzzaman, M. Rani, and F.N. Sibai, “On the Design of Low-Power Cache Memories for Homogeneous Multi-Core Processors,” 22nd International Conference on Microelectronics (ICM'10), pp. 387-390, 2010.

[5] H.V. Caprita and M. Popa, “Design methods of multithreaded architectures for multicore microcontrollers,” 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI-2011), pp. 427-432, 2011.

[6] J. Chen, W. Watson III, and W. Mao, “Multi-Threading on Multi-Core Processors,” Scientific Computing Group, IT Division Jefferson Lab, 2006.

[7] “Multi-core Processor from Wikipedia,” Wikipedia.com, 2011, http://en.wikipedia.org/wiki/Multi-core_processor (accessed on April 3, 2015)

[8] W.C. Ku, S.H. Chou, J.C. Chu, C.L. Liu, T.F. Chen, J.I. Guo, and J.S. Wang, “VisoMT: A Collaborative Multithreading Multicore Processor for Multimedia Applications With a Fast Data Switching Mechanism,” IEEE Transactions on Circuits and Systems for Video Technology, Vol. 19, pp. 1633-1645, 2009.

[9] R.M Ramanathan, “Intel Multi-Core Processors: Making the Move to Quad-Core and Beyond,” Intel White Paper, 2006.

[10] V. Suhendra and T. Mitra, “Exploring Locking & Partitioning for Predictable Shared Caches on Multi-Cores,” DAC'2008, Anaheim, CA, 2008.

[11] C. Harrison, “Programming the cache on the PowerPC 750GX/FX - Use cache management instructions to improve performance. IBM Microcontroller Applications Group,” 2005,http://www-128.ibm.com/developerworks/library/pa-pp ccache.html (accessed on April 3, 2015)

[12] I. Puaut and C. Pais, “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, 2007.

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

[14] E. Tamura, J.V. Busquets-Mataix, J.J.S. Martin, and A.M. Campoy, “A Comparison of Three Genetic Algorithms for Locking-Cache Contents Selection in Real-Time Systems,” Proceedings of the Int’l Conference in Coimbra, Portugal, 2005.

[15] V. Romanchenko, “Evaluation of the multi-core processor architecture Intel core: Conroe, Kentsfield…,” Digital-Daily.com, 2006.

[16] V. Romanchenko, “Quad-Core Opteron: architecture and roadmaps,” Digital-Daily.com, 2006.

[17] “MPC8272 PowerQUICC II – Family Reference Manual,” Freescale.com, 2008,http://www.freescale.com/files/32bit/ doc/ref_manual/MPC8272RM.pdf (accessed on April 3, 2015)

[18] “VisualSim (Mirabilisdesign) – a system-level simulator,” Mirabilisdesign.com, 2015, http://www.mirabilisdesign.com (accessed on April 3, 2015)

★★★