A Power-Aware Multi-Level Cache Organization Effective for Multi-Core Embedded Systems

Abu Asaduzzaman
Wichita State University/EECS Department, Wichita, Kansas, USA
Email: abuasaduzzaman@ieee.org

***Abstract*— Recent system design trends suggest multicore architecture for all computing platforms including distributed and embedded systems running real-time applications. Multilevel caches in a multicore system pose serious challenges as cache requires huge amount of energy to be operated and cache increases unpredictability due to its dynamic behavior. Bandwidth and synchronization problems are also critical design factors for distributed and embedded systems. In this work, we propose a "miss table" based cache memory organization which is very effective for real-time distributed and embedded systems. Cache-level miss table holds information about the memory blocks that cause most level-1 cache (CL1) misses under normal execution. Proposed cache organization also includes private victim caches (VCs) to hold level-1 victim blocks and shared level-2 cache (CL2) to help synchronization. Proposed cache organization improves CL1 cache hits that decrease memory latency and total power consumption and improve predictability and bandwidth. We simulate an 4-core system with two-level caches using MPEG4, H.264/AVC, FFT, MI, and DFT workload. Experimental results show that the proposed miss table based cache organization helps reduce average memory latency and total power consumption by 31% and 38%, respectively, when compared with cache organization without miss table and victim caches.**

***Index Terms*—cache organization, distributed systems, embedded systems, miss table, performance/power ratio, real-time applications**

I. INTRODUCTION

Most chip design vendors are using multicore architecture in their products and multicore processors are being used for all computing platforms including servers, workstations/PCs, and embedded systems. High performance/power ratio, predictability, synchronization, and bandwidth are important design factors for all modern systems, especially for distributed and embedded systems running real-time applications. In a distributed system, multiple sovereign computers communicate through a computer network to interact with each other in order to solve a common problem [1][2][3]. Embedded systems, in terms of complexity, range from simple (with a single microcontroller chip) to very complex (with multiple microcontrollers, peripherals, and networks mounted inside a large framework) [4][5][6][7][8][9]. Like desktop computing, the high performance/power ratio requirement for distributed and embedded systems is also increasing. Distributed and embedded systems are having multicore architectures for higher processing speed. Execution time predictability and total power consumption are critical issues for mobile and/or battery operated (multicore) systems supporting real-time applications.

In order to satisfy the needs for increased processing speed, there are significant changes in the direction of designing and developing processors. Most chip-vendors including Intel, AMD, IBM, and Sun are deploying multicore (instead of single-core) processors to their product lines [10][11][12][13][14][15]. In a multicore processor or chip-level multiprocessor (CMP), two or more independent cores are combined into a die. Normally, each core has its private CL1 – CL1 may be split into instruction cache (I1) and data cache (D1) to improve performance. In addition to CL1, a multicore processor usually has unified shared CL2 or distributed CL2s. Intel's Advanced Smart Cache is optimized for multicore processors to improve performance by sharing CL2 among the cores. AMD's multicore processor has distributed and dedicated CL2s (and shared CL3 in Opteron quad-core). Cache memory is first introduced by IBM in 1960s to bridge the speed gap between the CPU and the main memory. Almost immediately after that all gigantic chip-vendors introduce cache to their processors [16][17][18]. Today, processors are having multiple processing cores and most processors have on-chip CL1 and off-chip CL2 [19][20][21][22]. Even though cache improves overall system performance, a system with cache memories consumes more total power than the system without caches [23][24][25][26][27][28][29][30][31]. Excessive power consumption and execution time unpredictability may defeat the performance gain of distributed and embedded systems, especially when the system is mobile and/or battery operated and runs real-time applications [32][33][34][35][36][37][38][39]. Multicore parallel and/or distributed systems are very suitable for high-performance systems, because concurrent execution of tasks is possible there. In many respects including power consumption and heat dissipation, single-core architecture is inadequate for achieving the required level of performance and/or the required level of reliability. Multicore architecture

Manuscript received May 23, 2011; revised Month Day, 2011; accepted Month Day, 2011.
Corresponding author(s): Abu Asaduzzaman; Wichita State University, Wichita, KS; E-mail abuasaduzzaman@ieee.org

doi:10.4304/jcp.8.1.49-60

consumes less amount of power as it runs at a lower frequency. However, significant amount of power is required to operate the caches as cache is power-hungry.

It is established that cache parameters (cache size, line size, associativity level, etc.) have significant impact on overall system performance. Studies show that victim buffer/cache and stream buffering improves performance by improving cache hits [40][41][42][43]. However victim cache requires additional power to be operated. Studies also show that cache locking improves performance/power ratio (up to 25% cache locking) and execution time predictability in single-core systems by holding all or some important blocks inside the cache [24][44][45][46][47][48][49][50]. Billions of transistors in a single chip are now possible and the design trend of multicore in distributed and embedded systems is expected to grow in the future. To the best of our understanding, current cache organizations are not adequate for multicore system. As an example, cache makes the unpredictability in multicore even worse. Cache locking techniques are developed to improve predictability. Most of the existing cache locking techniques are developed for single-core systems. A few articles those mention cache locking in multicore do not elaborate the strategy well [13][14]. Our goal is to provide a single (and simple) cache architecture solution to reduce the average memory latency and total power consumption and increase the predictability, bandwidth, and synchronization for multicore distributed and embedded systems running real-time applications. In this work, we propose a "miss table" based cache architecture with private victim caches and shared CL2, which is very effective for distributed and embedded systems. Information about memory blocks with higher level-1 cache misses are stored in the miss table and level-1 victim blocks are stored in victim caches.

The rest of the paper is organized as follows. In Section 2, relevant surveyed articles are discussed. Proposed miss table based cache organization for real-time distributed and embedded systems is presented in Section 3. In Section 4, the proposed cache organization is evaluated by presenting simulation details and some important simulation results. Finally, this work is concluded in Section 5.

II. SURVEY

Improving the performance of distributed and embedded systems without any negative impact on power consumption is very challenging. Performance/power ratio improvement (in multicore) by optimizing cache memory subsystem has regained attention in the recent years. Several cache optimization techniques have been proposed to improve the performance of distributed and embedded systems. Cache locking is also used to improve performance/power ratio. However, currently available cache locking mechanism is not suitable for multicore architecture. In this section, we present some popular single-core cache memory hierarchies, followed by a number of existing single-core cache locking techniques, and cache organization used in contemporary popular multicore processors.

A. Single-Core Cache Memory Hierarchies

Cache memory has a very rich history in the evolution of modern computing [18]. Cache memory is first seen in the IBM System/360 Model 85 in late 1960s. In 1989, Intel 468DX microprocessor introduced on-chip 8 KB CL1 cache for the first time. In early 1990s, off-chip CL2 cache appeared with 486DX4 and Pentium microprocessor chips. Today's microprocessors usually have 128 KB or more of CL1, and 512 KB or more of CL2, and optional 2 MB or more CL3. Some CL1 cache is split into I1 and D1 in order to improve performance [11][12]. Intel Pentium 4 processor, one of the most popular single-core processors that use inclusive cache architecture, is discussed in [21].

The schematic diagram of a typical inclusive cache memory subsystem is shown in Figure 1. In inclusive cache architecture, CL2 contains each and every blocks that CL1 (i.e., I1 and D1) may contain. In case of a CL1 miss followed by a CL2 miss, the block is first brought into CL2 from main memory, then into CL1 from CL2. Intel Pentium 4 Willamette has on-die 256 KB inclusive CL2; with 8 KB level-1 trace/instruction cache (I1) and 8

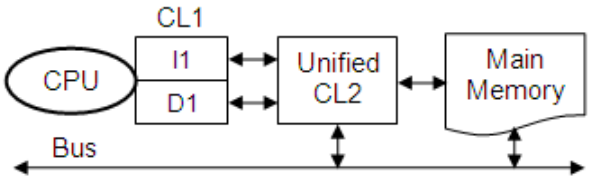

Figure 1. Schematic diagram of inclusive cache architecture.

KB level-1 data cache (D1).

The schematic diagram of a memory system with a victim cache between CL1 (I1, D1) and CL2 is presented in Figure 2. Victim cache reduces average memory latency. Usually the effective cache size of this architecture is more than CL2 and it provides performance gain [43][51]. The victim cache hierarchy is suitable for systems with limited cache-memory area (like embedded systems) and applications that perform a large amount of memory accesses (like multimedia) [52].

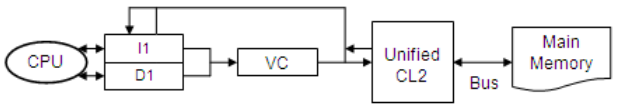

Figure 2. Cache architecture with victim cache.

B. Single-Core Cache Locking

Some cache locking approaches in real-time systems are presented in [47][48][49]. According to these approaches, cache contents are statically locked so as to make memory access time and cache-related preemption delay predictable. Cache locking may cause more power consumption due to the extra logic to implement it. However, these approaches can be used to improve

performance/power ratio if the right amount of correct memory blocks are locked.

In [29], 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 is more predictable than a system without cache. In [26], various algorithms to select a set of instructions to be locked in cache are compared. The algorithms mentioned in [25][26][29][44] show performance improvement and estimate a tight upper bound of the response time of tasks. The techniques mentioned above are used mainly to evaluate predictability in a single-core system. These cache locking techniques are inadequate to evaluate the performance/power ratio analysis – a crucial design factor for distributed and embedded systems.

An algorithm for off-line selection of the contents of two on-chip memories – locked caches and scratchpad memories is proposed in [49]. Experimental results show that the algorithm generates good ratio of on-chip memory accesses on the worst-case execution path. The major problem with this algorithm is that the worst-case performance with locked caches may degrade with large cache lines due to cache pollution.

In [45], cache locking in Intel Pentium-like single-core architecture is simulated running FFT, MI, and DFT workload. Simulated architecture has one processing core and two levels of cache memory hierarchy. Experimental results show that cache locking improves both the performance and predictability up to a limit (approximately 25%) of locked cache size. After that limit, predictability can be further improved by sacrificing performance. No analysis on power consumption is done in this work.

C. Multicore Cache Organizations

Most manufacturers are adopting multicore processors to acquire high processing speed for the future computing systems. Various contemporary multicore processors are classified in Figure 3. It is noted that, most popular multicore processors from Intel, ADM, and IBM have multilevel caches [11][12][20].

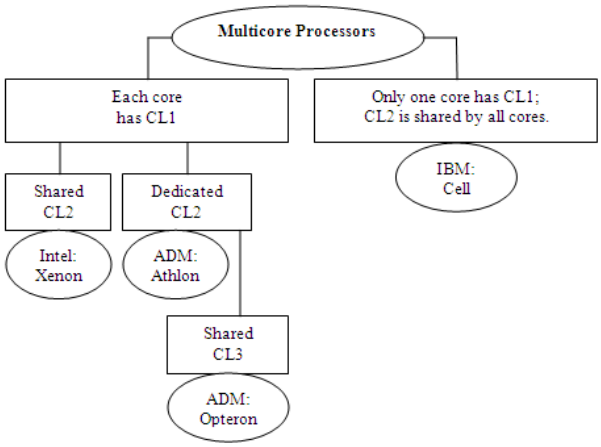

Figure 3. Classification of contemporary multicore processors.

AMD quad-core (Opteron Deerhound) has 256 KB I1, 256 KB D1, 2 MB dedicated CL2s, and 4 MB shared CL3 [11]. Figure 4 depicts the schematic diagram of Intel quad-core (Xeon DP) processor architecture that has 128 KB I1, 128 KB D1, and 8 MB shared CL2 [12].

Sony, Toshiba, and IBM (STI) have designed Cell

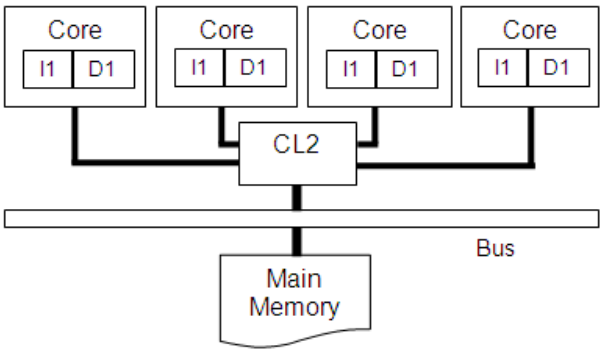

Figure 4. Schematic diagram of Intel quad-core architecture.

multicore processor, primarily to boost up the processing speed demanded by the electronic games [13][14][15]. This architecture has a Primary Processing Entity (PPE) and some helper units called Synergistic Processing Elements (SPE's) (see Figure 5). The PPE contains a 32 KB I1 and a 32 KB D1 caches. A 512 KB CL2 is shared by the PPE and SPEs. Each SPE may have 256 KB SRAM, 4 x 128 bit ALU (Arithmetic Logical Unit), and 128 of 128-bit registers. The Element Interconnect Bus (EIB) is the communication bus internal to the Cell processor.

The advances in semiconductor technologies facilitate

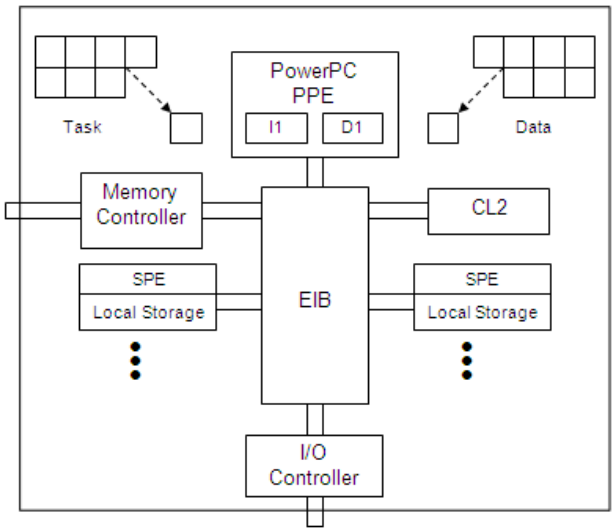

Figure 5. Diagram of IBM Cell-like multicore architecture.

to support increasing number of processing cores. As a result, homogenous or heterogeneous multicore platforms with more and more processing cores are expected to become the leading craze in the future. Private CL1 and CL2 and shared CL3 are being used to improve system bandwidth [53][57]. Prepushing and software controlled eviction are being used to improve communications support for multithreaded applications [54]. Cache-based memory copy hardware accelerator in a multicore system

is proposed to improve cache memory performance [55]. Contemporary multicore architectures indicate that the number and/or level of caches increase with the increase in processing cores. Therefore, caches in distributed and embedded multicore systems should be used very wisely so that high performance/power ratio is achieved [56][58][59].

III. Miss Table Based Cache Organization

In this work, we propose a miss table based cache architecture with private victim caches and shared CL2 for multicore systems for improved performance. In this organization, a miss table is introduced at the cache level to store the cache miss information of the code currently being executed. This architecture makes better use of memory blocks by using the miss table information. In addition to improving the performance/power ratio, this architecture improves the predictability and bandwidth. Therefore, this cache organization makes it suitable for high-performance low-power multicore real-time distributed and embedded systems. Schematic diagram of proposed miss table based cache organization with victim caches for multicore systems is shown in Figure 6. It in an Intel Xeon-like quad-core architecture.

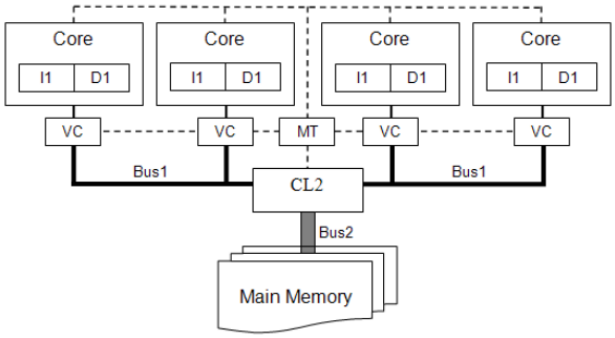

Figure 6. Miss table based cache architecture with victim cache.

In the following subsections, we discuss the implementation of the miss table with victim caches and CL2 and the methodology of workload characterization for multicore systems.

A. Miss Table

We introduce a miss table (MT) inside the CPU such that it can be accessed from CL1s, victim caches, and CL2. MT can be a small cache or a set of registers to hold information about all (or some important) blocks that cause the most cache misses. Block addresses are sorted in descending order of the number of misses. For each application/function, after post-processing the tree-graph generated by Heptane (Hades Embedded Processor Timing ANalyzEr) [60], block address information is prepared for the MT. We use popular Heptane package to analyze each application/function, because it will be assigned to a core in the multicore system. When an application/function is assigned to a core, MT is populated by the related block addresses. MT information is used to select a cache block to be locked or a victim block to be replaced. For both entire and way cache locking, MT should store information about 'N' cache blocks (at most) when the cache can be divided into 'N' blocks. Each core has its private CL1 (I1 and D1). The CPU has one shared CL2. VCs hold level-1 victim blocks and additional memory blocks when stream buffering is used. A modified replacement policy is used. Using the MT, this policy selects a block that has the minimum number of misses and that is not locked (at CL2). In case of a tie in the number of misses, a block is selected randomly. If a block's information is not in the MT, it should be selected first. The schematic diagram of the data flow inside a core is shown in Figure 7.

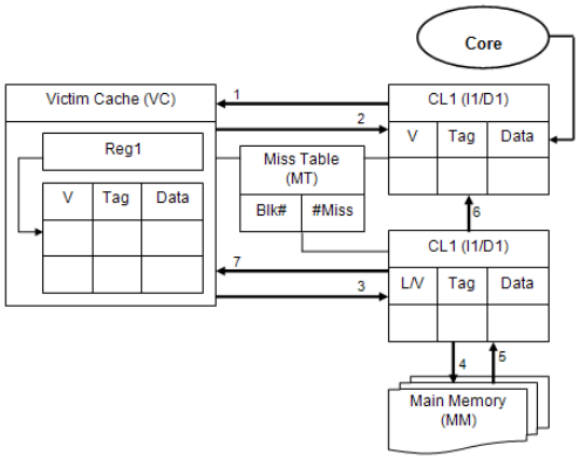

Figure 7. Schematic diagram of a multicore processor (partial) showing MT, VC, CL2, and one core.

When a core starts execution, if the requested memory block is not found in CL1, VC is checked (if VC is enabled). A victim block in CL1 is selected using the miss table. Finally, if the requested memory block is not found in CL2, it is fetched from the main memory. When a new block comes to CL2, it is checked using the MT information if the block should be locked. 'L' bit is set to indicate a block is locked.

Victim cache control logic for the proposed cache architecture is presented in Figure 8. The victim cache

```
:: Victim Cache Control Logic
START:
If (CL1_Request) Then
  If VC_Disable) Then
    Done // Perform normal as if VC does not exist
  Else // VC_Enabled
    Reg1 ← CL1 cache line victim block (VB)
    If (VC_Hit) Then
      CL1 ← VC cache line (Hit)
      VC ← Reg1
    Else // VC_Miss
      If (VC_Full and Dirty_Block) Then
        CL2 ← VC cache line(s) with minimum misses
      End If
      If CL2 miss, Issue Pre-fetch or Stream_Buffering
      CL1 ← Reference block from CL2 or MM
      IF (Stream_Buffering) Then
        VC ← Additional blocks from CL2 or MM
      End If
    End If // VC_Hit / VC_Miss
  End If // VC_ Disable / VC_Enabled
End If // CL1_Request
END
```

Figure 8. Control logic of the proposed victim cache.

can be enabled or disabled as needed. For stream buffering, victim cache must be enabled.

B. Workload

Currently available workload and workload characterization methodology are not useful to run our simulation programs. So, we develop a workload characterization methodology that can be used to simulate both single-core and multicore systems. Our workload characterization methodology has three important phases – code division, code estimation, and block selection.

Code Division: In phase-I, we analyze the application(s) and divide the code into smaller segments as needed. Small applications can be mapped directly among the cores. In order to support large applications, the code is divided into smaller (end-to-end) functions in such a way that a function can be assigned to a core. Code division is crucial for achieving better performance as it helps load balancing.

Code Estimation: In phase II, we estimate important operations for each application (in case of small applications) or function (in case of large applications).

TABLE I.
CODE ESTIMATION FOR FFT

Type of Operation	Number of Operations (%)
Integer	18
Floating-point	71
Load/Store	9
Branch	2

Code estimation can be done manually for smaller code segments. Results (types and numbers of operations) of code estimation for FFT code are shown in Table I.

Block Selection: Finally in phase III, we select the blocks that cause more cache misses under normal execution. Major steps in this process are explained with an example (see Table II). A tree-graph for each code segment is created using Heptane simulation package. The nodes of the tree-graph represent the structure of programs in the high-level language and the leaves represent the basic blocks. From the tree graph, we collect the number of instructions and cache miss information for each node. From off-line analysis we determine which code section of the source file causes more misses. By post-processing the information collected from the tree-graph, we obtain the block address that can be locked. The number of instructions and cache misses for sequence nodes are negligible. The number of instructions and cache misses for loop nodes are excluded in this work because we consider all code nodes (where a loop node represents a set of code nodes). Table II shows the block addresses and total misses (sorted in descending order of the number of misses) obtained for FFT code. For locking, blocks are selected depending on the cache size, line size, and locked cache size. For an example: if cache size is 2 KB and line size is 128 B, then number of blocks is 2*1024/128 = 16. Now, 25% cache locking means first 4 (25% of 16 is 4) blocks should be selected for locking. A small routine is required to be executed at the system start-up to load the content of the cache with the selected block address values and lock the cache for the whole execution period.

TABLE II.
SORTED BLOCK ADDRESS FOR FFT

Number	Block Address	Total Cache Misses	Block in MT?
1	0	35	Yes
2	80	33	Yes
3	300	32	Yes
4	100	28	Yes
5	400	17	Yes
6	320	15	Yes
7	380	14	Yes
8	180	12	Yes
9	280	11	Yes
10	360	11	Yes
11	420	10	Yes
12	200	9	Yes
13	140	4	Yes
14	480	4	Yes
15	160	3	Yes
16	220	3	Yes
17	440	3	No
18	120	1	No
19	260	1	No

From Table II, total cache misses is 246. By locking the first four cache blocks (approximately 25%), 124 cache misses can be avoided (i.e., cache miss is reduced by 50%).

IV. EVALUATION

We use simulation technique to evaluate the proposed miss table based cache organization with victim caches and CL2 for multicore distributed and embedded systems. We develop simulation platform, generate workload by characterizing the applications, and model the multicore architecture used in distributed and embedded systems. In the following subsections, we first briefly discuss simulation details. Then, we present some important simulation results.

A. Simulation Details

We develop a multicore cache simulation program using VisualSim [61]. We generate workload for MPEG4, H.264/AVC, FFT, MI, and DFT applications using Heptane to run the VisualSim simulation program.

We obtain results for various cache parameters and locked CL2 size. In this subsection, we briefly discuss the assumptions, simulation platform, workloads, and parameters used in this simulation.

Assumptions: Important assumptions for modeling the selected architecture and for running the simulation program include,

- There is only one miss table, which is accessible from CL1s, VCs, and CL2.
- Each core has its own private VC. VCs can be functionally enabled and disabled.
- Only level-2 cache locking is implemented (using MT) to reduce cache inconsistency.
- Modified cache replacement strategy is considered for CL2. Locked blocks are excluded by the modified replacement policy. Random cache replacement strategy is used for CL1.
- For both CL1 and CL2, write-back memory update policy is used.
- The delay introduced by the bus that connects CL2 and main memory is 12 times longer than the delay introduced by the bus that connects CL1 and CL2.

Simulation Platforms: In this work, we develop a simulation platform using two popular simulation tools – Heptane [60] and VisualSim [61]. We install/configure Heptane in Linux Fedora operating system and VisualSim in Windows XP operating system in a Dell PowerEdge 1600SC PC. Heptane simulates a processing core, takes C code as the input application, and generates tree-graph that shows the blocks that cause misses. After post-processing the tree-graph, block addresses are selected for the miss table. Input and output parameters for Heptane are shown in Table III.

VisualSim provides a GUI interface to model and simulate embedded multicore systems using applications' workload. We use VisualSim to simulate multicore cache architecture with MT, VCs, and CL2. Input and output parameters for VisualSim are shown in Table IV.

TABLE III.
INPUT / OUTPUT FOR HEPTANE

Input	Output
Application: C code XML file (Miss Blocks)	Block Address Number of misses Number of instructions

Workload: In this work, we use Moving Picture Experts Group's MPEG4, Advanced Video Coding – widely known as H.264/AVC, Fast Fourier Transform (FFT), Matrix Inversion (MI), and Discrete Fourier Transform (DFT) applications to run the simulation program. Some important characteristics of these applications are shown in Table V.

Important Input / Output Parameters: Important input parameters and their respective values used in this work are shown in Table VI. We keep I1 cache size equal to D1 cache size. Also, we keep the CL1 (I1 and D1) line size equal to the CL2 line size and the CL1 associativity level equal to the CL2 associativity level. In this work, we simulate an 4-core system and keep CL2 cache size fixed at 256 KB.

TABLE IV.
INPUT / OUTPUT FOR VISUALSIM

Input	Output
Number of integers Number of floats Number of loads/stores Number of branches Cache miss rate Miss table (MT)	Average memory latency Total power consumption

TABLE V.
CHARACTERISTICS OF THE APPLICATIONS

Applications	Code Size (KB)	Number of Instructions
MPEG4 Decoder	233.20	25,207,118
H.264/AVC Decoder	199.14	19,810,037
FFT	2.34	365,184
MI	1.47	227,518
DFT	1.16	171,307

TABLE VI.
IMPORTANT INPUT PARAMETERS

Input Parameters	Value
I1/D1 cache size (KB)	2, 4, 8, 16, or 32
CL1/CL2 line size (Byte)	16, 32, 64, 128, or 256
CL1/CL2 associativity level	1-, 2-, 4-, 8-, or 16-way
CL2 cache size (KB)	256 (fixed)
Number of cores	4 (fixed)

Output parameters in this work are the average memory latency per task and total power consumption. We define delay as the time between the start of execution of a task and its end. We use an activity based power analysis method. Power consumption by each component is considered. CL1 consumes about 43% processor power [62]. Table VII shows how power consumption is distributed among processor, CL2, and main memory. In this method, a system component is considered to be one of the three states – active (component consumes adequate amount of energy to be turned on and active), ideal (component consumes minimum amount of energy just to be turned on), or sleep (component is turned off and consumes no energy). For a system with X components and Y tasks, the total power consumption can be expressed as shown below,

$$Pt\ (total) = \sum_{i=1}^{X} \sum_{j=1}^{Y} (Pij\ (active) + Pij\ (ideal) \qquad \text{Equation (1)}$$

TABLE VII.
POWER CONSUMPTION BY PROCESSOR, CL2, AND MEMORY

Parameter	Power (%)	Unit/Usage
I1	27	6
D1	16	3
CPU	36	7
Others	21	4
Processor Total	*100*	*20*
CL2	-	CL2/(I1+D1)*(5+3)
Main Memory (MM)	-	MM/(I1+D1)*(5+3)

B. Simulation Results

We present some important simulation results in the following subsections. We model a system with 4 cores and run the simulation program using MPEG4, H.264/AVC, FFT, MI, and DFT workloads.

Impact of I1 Cache Size: The average memory latency per task versus I1 cache size for no locking and 25% level-2 cache locking is shown in Figure 9. Experimental results show that average memory latency per task for MPEG4 and H.264/AVC decreases when we move from no locking to 25% locking and/or from smaller I1 to larger I1; the decrement is significant for smaller I1. However, average memory latency per task for FFT, MI, and DFT remains the same when we move from no locking to 25% locking and/or from 4KB I1 to larger I1. This is because FFT, MI, and DFT code entirely fit into 4KB or larger I1. But MPEG4 and H.264/AVC applications are bigger (than those of FFT, MI, or DFT) and do not entirely fit into I1. Results also show that average memory latency per task due to MPEG4 is always greater than those of others. This is because MPEG4 has more stressful workload than the others do. Similar behavior is observed for the total power consumption versus I1 cache size for no locking and 25% level-2 cache locking (see Figure 10). The total power consumption decreases when we move from no locking to 25% CL2 locking and/or from 4KB I1 to larger I1 for MPEG4 and H.264/AVC; the decrement is significant for smaller I1. However, total power consumption for FFT remains almost the same. Total power consumption due to MPEG4 is always greater than those of others.

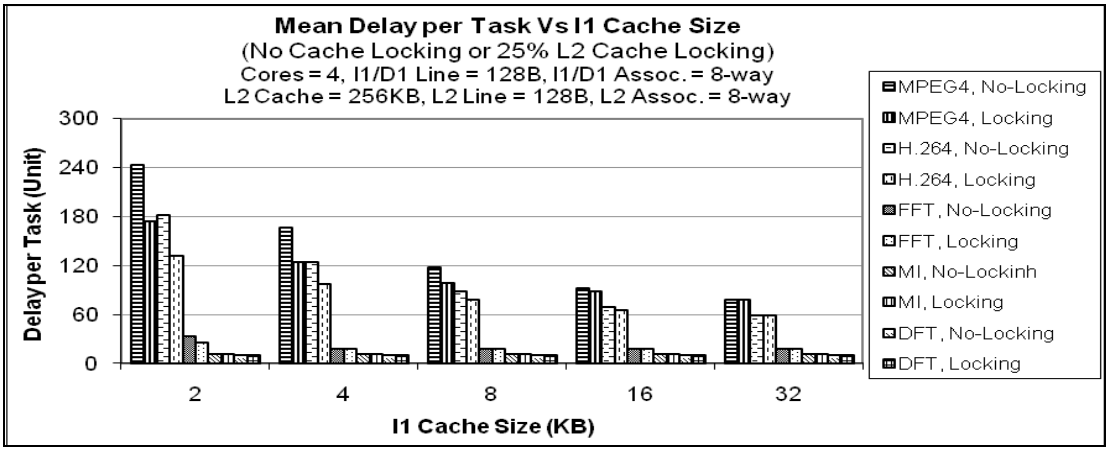

Figure 9. Average memory latency per task versus I1 cache size.

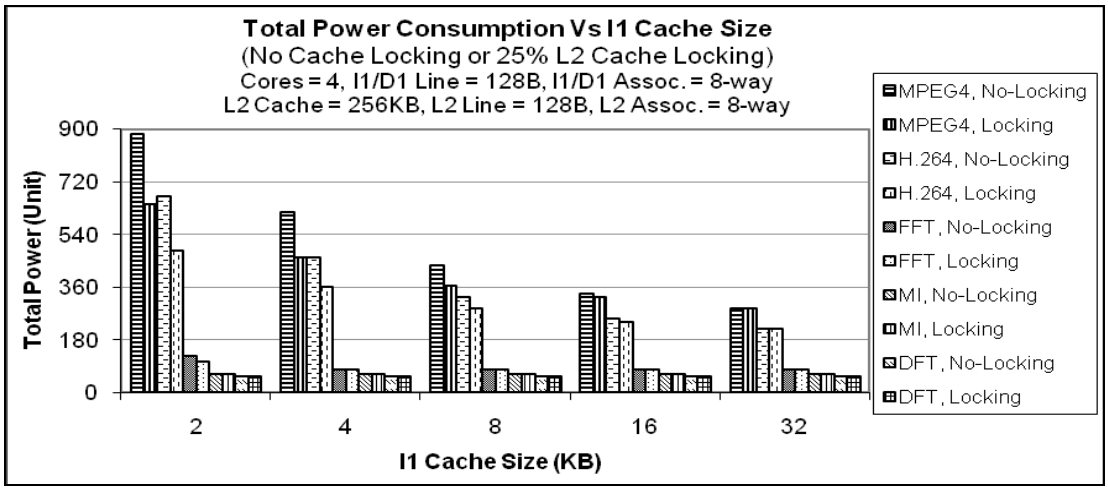

Figure 10. Total power consumption versus I1 cache size.

The results due to MPEG4 decoder and H.264/AVC decoder workloads are very similar; so in the rest of the discussion, the MPEG4 results will also represent the omitted H.264/AVC results. For similar reason, the FFT results will also represent the omitted MI and DFT results.

Impact of I1 Line Size: The average delay per task versus I1 line size for no locking and 25% locking using a miss table is shown in Figure 11. We notice that the mean delay per task goes down for MPEG4, regardless of the line size, with increasing line size leveling off at a line size of 128B. However, the average delay per task for FFT remains the same. This is because FFT code fits entirely in 4KB I1, so changing line size and/or using a miss table do not impact on average memory latency.

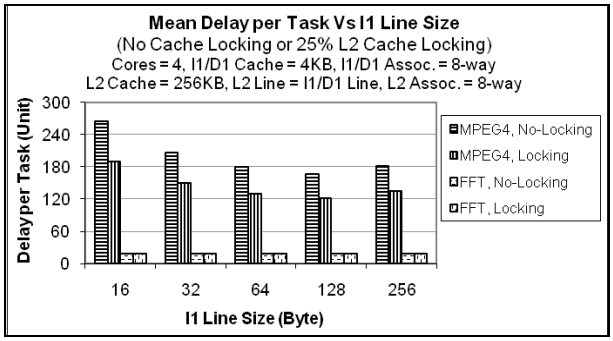

Figure 11. Average memory latency per task versus I1 line size.

Similarly, we notice that 25% level-2 cache locking using a miss table helps decrease total power consumption regardless of I1 line size [see Figure 12]. It is also noted that total power consumption decreases with increasing I1 line size leveling off at a line size of 128B. Again, total power consumption for FFT remains the same because FFT code fits entirely in 4KB I1 and changing line size and/or applying cache locking do not impact on total power consumption.

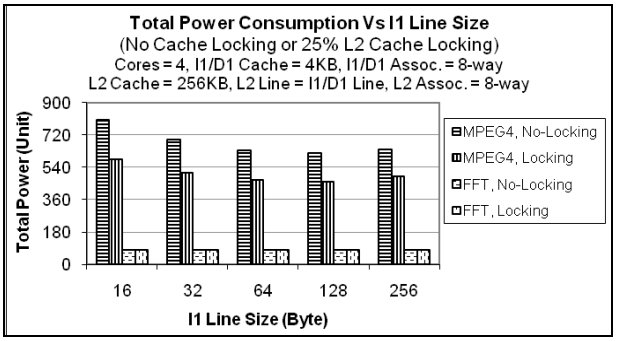

Figure 12. Total power consumption versus I1 line size.

Impact of I1 Associativity Level: The impact of no cache locking and 25% L2 cache locking on average memory latency by varying I1 associativity level is shown in Figure 13. Experimental results show that for any I1 associativity level, average memory latency per task for MPEG4 decreases when we move from no locking to 25% locking or I1 associativity level is increased. The decrease is significant for smaller associativity levels. For 4KB I1, average memory latency per task for FFT remains the same at any associativity level.

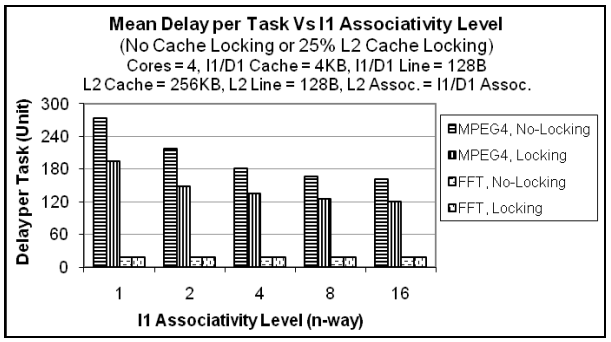

Figure 13. Mean delay per task versus I1 associativity level.

Figure 14 illustrates the impact of no cache locking and 25% level-2 cache locking on total power consumption by varying I1 associativity level. Experimental results show that for any I1 associativity level, total power consumption for MPEG4 decreases when we move from no locking to 25% locking and I1 associativity level is increased. Again, total power consumption for FFT remains the same for 4KB I1.

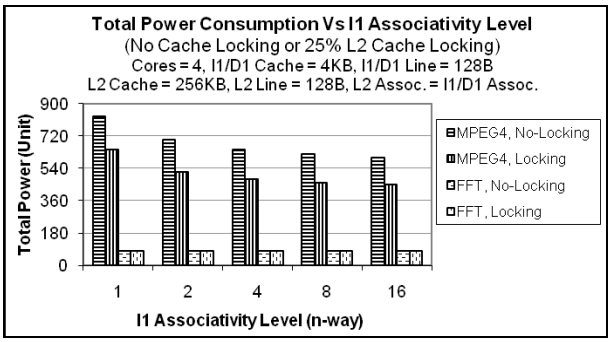

Figure 14. Total power consumption versus I1 associativity level

Impact of Miss Table: Using a miss table has significant impact on average memory latency per task and total power consumption. Experimental results show that using a miss table with cache locking decreases average memory latency per task for MPEG4 application. From Figure 15, it is also noted that beyond 25% cache locking (for MPEG4), the average memory latency per task increases with the increase in the number of locked blocks. Simulation results also show that when smaller applications like FFT entirely fits in I1, there is no positive impact of the miss table and cache locking on average memory latency per task. Using a miss table with cache locking has positive impact on total power consumptions for some applications. From experimental results we notice that total power consumption starts decreasing with the increase in the number of the locked blocks (up to 25% cache locking) for MPEG4 application (see Figure 16). We also notice that total power consumption decreases when the miss table is used with cache locking (up to 25% cache locking for MPEG4). On the other hand, for smaller applications like FFT that entirely fits in 4KB I1, there is no positive impact of cache locking on total power consumption.

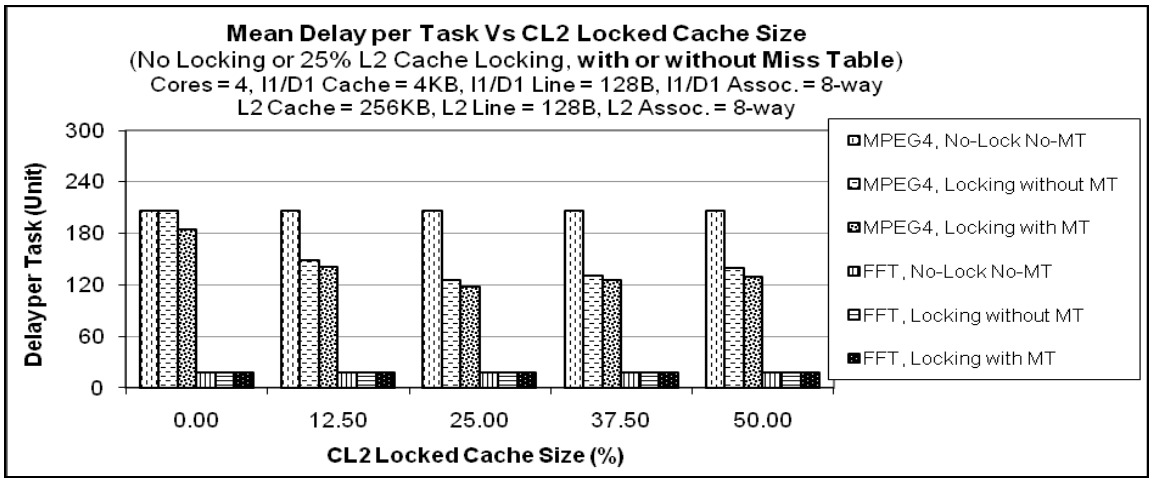

Figure 15. Average memory latency per task versus level-2 locked cache size (with and without a miss table).

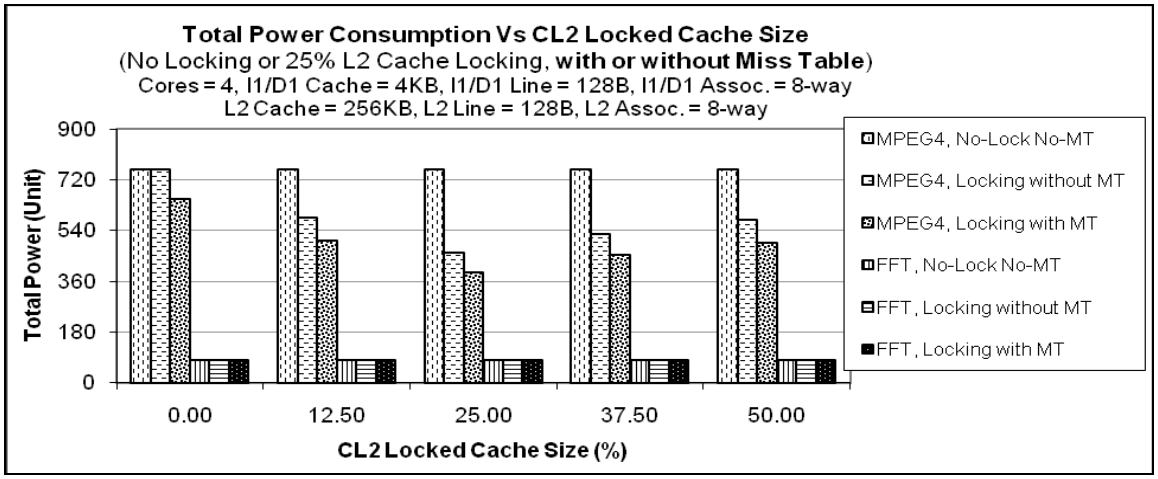

Figure 16. Total power consumption versus level-2 locked cache size (with and without a miss table).

Impact of Victim Caches: Finally, we present the impact of using victim caches with a miss table on average memory latency per task and total power consumption. Experimental results reveal that for MPEG4 application, average memory latency per task decreases when victim caches are used with cache locking (see Figure 17). Beyond 25% cache locking (for MPEG4), the average memory latency per task increases with the increase in the number of locked blocks. Simulation results also show that when applications entirely fits in I1 (as FFT fits in 4KB I1), there is no positive impact of using victim caches on average memory latency per task.

Experimental results also reveal that total power consumption starts decreasing with the increase in the amount of the locked blocks for MPEG4 application as shown in Figure 18. We also notice that total power consumption decreases when victim caches are used with

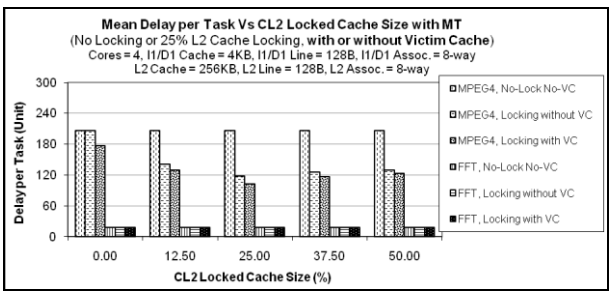

Figure 17. Average memory latency per task versus level-2 locked cache size (with and without victim caches).

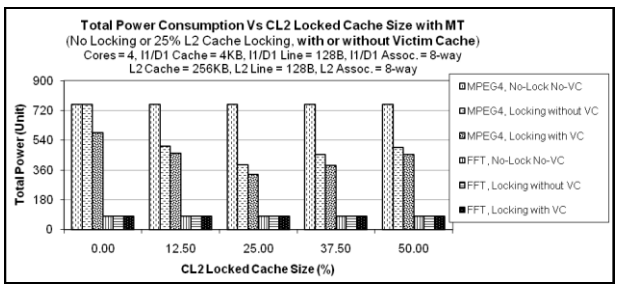

Figure 18. Average memory latency per task versus level-2 locked cache size (with and without victim caches).

a miss table based cache locking. Beyond 25% cache locking (for MPEG4), total power consumption increases with the increase in the amount of locked blocks. Again, when applications entirely fits in I1 (as FFT fits in 4KB I1), there is no positive impact of using victim caches on total power consumption.

In summary, we present the maximum increment (+) or decrement (-) in average memory latency per task and total power consumption from their initial values (the values of the respective parameters are mentioned in Figures 9 – 18). As shown in Table VIII, using a miss table help achieve 23% reduction in average memory latency per task and 31% reduction in total power consumption for H.264/AVC. For MPEG4, the reductions are less than H.264/AVC but significant (21% reduction in average memory latency per task and 28% reduction in total power consumption). So, performance/power ratio for MPEG4 and H.264/AVC is improved significantly by using a miss table and victim caches. It is noted that in this experiment, for I1 size 4KB or larger, using a miss table and victim caches with cache locking has no positive impact of performance/power ratio for FFT.

For MPEG4 and H.264/AVC, average memory latency per task and total power consumption can be reduced even further using cache locking as shown in Table VII.

TABLE VIII.
IMPACT OF MISS TABLE

		Changes (%)	
		Latency	Power
Miss Table (No Locking)	MPEG4	(-) 21	(-) 28
	H.264	(-) 23	(-) 31
	FFT	0	0
Locking using miss table and victim caches	MPEG4	(-) 29	(-) 36
	H.264	(-) 31	(-) 38
	FFT	0	0

V. CONCLUSION

Multicore architectures provide a viable platform for distributed and embedded systems to implement highly computation intensive and real-time applications. However, multilevel cache organization in multicore architecture may decrease performance/power ratio and increase execution time predictability due to cache's adaptive and dynamic behavior. Also, high-performance processing cores (with multiple caches) consume significant amount of energy and dissipate tremendous amount of heat which are critical for embedded systems. Bandwidth and synchronization are critical design factors for distributed systems. Therefore, running complex applications on distributed and embedded multicore systems with cache memories encounters serious challenges. Recently published articles show that various cache optimization techniques are proposed to improve cache performance in multicore architecture. In this work, we introduce a "miss table" based cache memory organization that also includes private victim caches and shared CL2. Miss table helps improve cache locking and cache replacement performance by providing the cache miss information (if not locked) for memory blocks. This approach reduces cache misses by locking the cache blocks with maximum number of cache misses and replacing the cache blocks with minimum number of cache misses. Victim caches help improve cache hits and bandwidth by holding some level-1 victim blocks. Shared CL2 helps synchronization among cores. In this scheme, a better use of the cache is possible because the expected cache miss information is known (via the miss table) before running a code segment.

We model a multicore architecture with 4 processing cores and 2 levels of caches and run the simulation program using workload from MPEG4, H.264/AVC, FFT, MI, and DFT applications. We find the addition of the miss table to a multilevel cache organization is very promising. Experimental results show that for MPEG4 workload a reduction of 21% in average memory latency per task and a reduction of 28% in total power consumption are achieved by introducing the miss table (without any cache locking). Results (from Table II) also show that this method may improve predictability by reducing cache misses by 50% or more when one-fourth cache is locked. It is also noticed that miss table and victim caches with cache locking improve performance/power ratio for MPEG4 (reduce average memory latency per task by 29% and total power consumption by 36%). According to the simulation results, the presence of a miss table has more impact on H.264.AVC results than on MPEG4 results; this is because H.264.AVC workload is less stressful when compared with MPEG4 workload. Finally, it is noted that miss table, victim caches, and/or cache locking has no positive impact on performance/power ratio for FFT (also for MI and DFT) as FFT code is very small.

It is important to keep the maximum number of cores in active state to achieve the maximum performance. However, the more cores are active, the more power should be consumed. A balance must be maintained in the core allocation strategy for multicore systems between the performance and power consumption trade-off. We plan to investigate power-aware core allocation strategy for multicore architecture in our next endeavor.

REFERENCES

[1] S. Ghosh, "Distributed Systems – An Algorithmic Approach", Chapman & Hall/CRC, ISBN 978-1-58488-564-1, 2007.
[2] G.R. Andrews, "Foundations of Multithreaded, Parallel, and Distributed Programming", Addison–Wesley, ISBN 0-201-35752-6, 2000.
[3] S. Dolev, "Self-Stabilization", MIT Press, ISBN 0-262-04178-2, 2000.
[4] P.J. Koopman, "Embedded System Design Issues (the Rest of the Story)", Proceedings of the International Conference on Computer Design (ICCD'96), 1996.
[5] S. Mohanty, and V.K. Prasanna, "Design of High-Performance Embedded System using Model Integrated

Computing", 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'04), 2004.
[6] L. Benini and G.D. Micheli, "Networks on Chip: A New Paradigm for Systems on Chip Design", IEEE, 2002.
[7] S. Mohanty, V.K. Prasanna, S. Neema, and J. Davis, "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation", LCTES'02–SCOPES'02, Germany, 2002.
[8] "Improving Multicore Computing Systems", NTNU Computer Architecture Research Group. DOI=http://www.idi.ntnu.no/~dam/ncar/, 2009.
[9] F.J. Villa, M.E. Acacio, and J.M. Garcia, "On the Evaluation of Dense Chip-Multiprocessor Architectures", IEEE, 2006.
[10] R.M. Ramanathan, "Intel Multi-Core Processors: Making the Move to Quad-Core and Beyond", White Paper, 2006.
[11] V. Romanchenko, "Evaluation of the multi-core processor architecture Intel core: Conroe, Kentsfield…", Digital-Daily.com, 2006.
[12] V. Romanchenko, "Quad-Core Opteron: architecture and roadmaps", Digital-Daily.com, 2006.
[13] J. Stokes, "Introducing the IBM/Sony/Toshiba Cell Processor – Part II: The Cell Architecture", DOI=http://arstechnica.com/articles/paedia/cpu/cell-2.ars, 2005.
[14] J. Stokes, "Xenon's L2 vs. Cell's local storage, and some notes on IBM/Nintendo's Gekko", DOI=http://arstechnica.com/articles/paedia/cpu/xbox360-1.ars/6, 2005.
[15] A. Vance, "Cell processor goes commando", Mountain View, DOI=http://www.theregister.co.uk/2006/01/22/cell_mecury_army/, 2006.
[16] K. Kavi, "Cache Memories", White Paper, University of Alabama in Huntsville, 2007.
[17] L. Schoeb and G. Darnell, "Large Processor L2 Cache Sizes in Dell PowerEdge Servers", White Paper, Dell, 1999.
[18] "Cache', Smart Computing Encyclopedia, DOI=http://www.smartcomputing.com/editorial/dictionary/detail.asp?guid=&searchtype=&DicID=16600&RefType=Encyclopedia, 2011.
[19] R. Cook, J. Linn, C. Linn, and T. Walker, "Cache memories: A Tutorial and Survey of Current Research Directions", ACM, pp.99–110, 1982.
[20] D.K. Every, "IBM's Cell Processor: The next generation of computing?", Shareware Press DOI=http://www.mymac.com/fileupload/CellProcessor.pdf, 2005.
[21] G. Torres, "Inside Pentium 4 Architecture", DOI=http://www.hardwaresecrets.com/printpage/235/1, 2005.
[22] "Multi-core (computing)", Wikipedia, DOI=http://en.wikipedia.org/wiki/Xeon; http://en.wikipedia.org/wiki/Athlon, 2011.
[23] D. Lenoski, J. Laudon, M.S. Lam, et al., "The Stanford Dash Multiprocessor", IEEE, 1992.
[24] I. Mahgoub, A. Asaduzzaman, and M. Yousif, "Evaluation of memory latency in cluster-based cache-coherent multiprocessor systems with different interconnection topologies", Computers & Electrical Engineering, Vol. 26, Issue 3-4, pp.207-220, 2000.
[25] 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 International Conference in Coimbra, Portugal, 2005.
[26] E. Tamura, F. Rodriguez, 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.
[27] L. Thiele and R. Wilhelm, "Design for Timing Predictability", Real-Time Systems, Vol. 28, Issue 2-3, pp.157-177, 2004.
[28] "Improving cache performance", UMBC, DOI=www.csee.umbc.edu/help/architecture/611-5b.ps, 2009.
[29] X. Vera and B. Lisper, "Data Cache Locking for Higher Program Predictability", SIGMETRICS'03, CA, 2003.
[30] R. Wilhelm and L. Thiele, "Timing Predictability — a Must for Avionics Systems", White Paper, 2006.
[31] R. Wilhelm, J. Engblom, et al., "The Determination of Worst-Case Execution Times", ARTIST, 2003.
[32] T. Bjerregaard, and S. Mahadevan, "A survey of research and practices of Network-on-chip", Source ACM Computing Surveys (CSUR), Vol. 38, Issue 1, 2006.
[33] P. Gepner and M.F. Kowalik, "Multi-Core Processors: New Way to Achieve High System Performance", Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), pp.9-13, 2006.
[34] A. Hajare, "Performance Modeling of a Multiprocessor Bus Architecture", IEEE, 1991.
[35] A. Jerraya, H. Tenhunen, and W. Wolf, "Multiprocessor Systems-on-Chips", IEEE Computer Society, pp.36–40, 2005.
[36] M. Loghi, M. Poncino, and L. Benini, "Cache coherence tradeoffs in shared-memory MPSOCs", ACM Transaction on Embedded Computing Systems, Vol. 5, No. 2, pp.383-407, 2006.
[37] N. Pazos, W. Brunnbauer, J. Foag, and T. Wild, "System level performance estimation of multi-processing, multi-threading SoC architectures for networking applications", SystemC: methodologies and applications book contents, pp.157-190, 2003.
[38] P.F. Sweeney, M. Hauswirth, et al., "Understanding Performance of MultiCore Systems using Trace-based Visualization", STMCS'06, Manhattan, NY, 2006.
[39] W. Wolf, "The Future of Multiprocessor Systems-on-Chips", DAC'04, San Diego, pp.681–685, 2004.
[40] G. Albera and R.I. Bahar, "Power/Performance Advantages of Victim Buffer in High-Performance Processors", IEEE Volta International Workshop on Low Power Design. Como, Italy, 1999.
[41] C. Zhang and F. Vahid, "Using a Victim Buffer in an Application-Specific Memory Hierarchy", in the Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE-2004), 2004.
[42] C. Harrison, "Programming the cache on the PowerPC 750GX/FX - Use cache management instructions to improve performance", IBM Microcontroller Applications Group, DOI=http://www-128.ibm.com/developerworks/library/pa-ppccache.html, 2005.
[43] S. Fang, K. Lin, J. Tsai, and Q. Yu, "Final Project. Super-Scalar. Stream Buffer. Victim Cache.", UCB, www.cs.berkeley.edu/~kubitron/courses/cs152-S01/projects/submit/project7_report2.doc, 2006.
[44] A. Arnaud and I. Puaut, "Dynamic Instruction Cache Locking in Hard Real-Time Systems", IEEE, 2005.

[45] A. Asaduzzaman, N. Limbachiya, I. Mahgoub, and F. Sibai, "Evaluation of I-Cache Locking Technique for Real-Time Embedded Systems", IEEE-IIT'07, UAE, 2007.
[46] A.M. Campoy, E. Tamura, S. Saez, F. Rodriguez, and J.V. Busquets-Mataix, "On Using Locking Caches in Embedded Real-Time Systems", In ICESS-05, LNCS 3820, pp.150-159, 2005.
[47] I. Puaut, "Cache Analysis Vs Static Cache Locking for Schedulability Analysis in Multitasking Real-Time Systems", DOI=http://citeseer.ist.psu.edu/534615.html, 2006.
[48] I. Puaut, D. Decotigny, "Low-Complexity Algorithms for Static Cache Locking in Multitasking Hard Real-Time Systems", IEEE, 2002.
[49] I. Puaut, 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.
[50] T. Tarui, T. Nakagawa, N. Ido, M. Asaie, and M. Sugie, "Evaluation of the lock mechanism in a snooping cache", ACM Proceedings of the 6th international conference on Supercomputing, 1992.
[51] N.P. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers", Western Research Laboratory (WRL), Digital Equipment Corporation, 1990.
[52] Y. Zheng, B.T. Davis, and M. Jordan, "Performance Evaluation of Exclusive Cache Hierarchies", IEEE-2004, pp.89-96, 2004.
[53] X. Jiang, et al., "CHOP: integrating DRAM caches for CMP server platforms", IEEE Micro v. 31 no. 1, pp. 99-108, January/February 2011.
[54] S. Varoglu and S. Jenks, "Architectural support for thread communications in multi-core processors", Parallel Computing 37.1, pp. 26(16), Jan 2011.
[55] F. Duarte, et al., "Cache-Based Memory Copy Hardware Accelerator for Multicore Systems", IEEE Transactions on Computers v. 59 no. 11, pp. 1494-507, November 2010.
[56] P. Conway, et al., "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor", IEEE Micro v. 30 no. 2, pp. 16-29, March/April 2010.
[57] Y. Mori, K. Kise, "The cache-core architecture to enhance the memory performance on multi-core processors", International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2009), pp. 445-450, Hiroshima, Japan, 2009.
[58] W. Wang, P. Mishra, "Dynamic Reconfiguration of Two-Level Cache Hierarchy in Real-Time Embedded Systems", Journal of Low Power Electronics, Volume 7, Number 1, pp. 17-28(12), February 2011.
[59] J. Gu, H. Guoa, and P. Lia, "An on-chip instruction cache design with one-bit tag for low-power embedded systems", Microprocessors and Microsystems, Volume 35, Issue 4, pp. 382-391, June 2011.
[60] "Heptane – A tree-based WCET analysis tool", DOI=http://ralyx.inria.fr/2004/Raweb/aces/uid43.html, 2012.
[61] "VisualSim – A system-level simulator from Mirabilis Design, Inc.", DOI= http://www.mirabilisdesign.com/, 2012.
[62] W. Tang, R. Gupta, and A. Nicolau, "Power Savings in Embedded Processors through Decode Filter Cache", Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE'02), pp.1-6, 2002.

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

Currently, he is working as an Assistant Professor in the department of electrical engineering and computer science at Wichita State University in Wichita, Kansas. Previously (2003-2010), he worked as research associate, teaching instructor, and specialist computer applications at FAU. He carried out a very significant role in Motorola-FAU OPP project funded by Motorola. He taught various hardware/software/database courses in the CEECS Department at FAU. He has published several journal and conference papers and book chapter out of his research work. His research interests include computer architecture, real-time distributed/embedded systems, parallel computing, and performance evaluation.

Mr. Asaduzzaman is a member of the IEEE, the honor society of PKP, TBP, UPE, Golden Key, and Who's Who (in 1997, in 2010-2011, and in 2011-2012). He obtained the first place (in oral presentation) at FAU Graduate Research Symposium in 2005. He received several prestigious awards including FAU Wise Owl Awards (one per college) in 2009 and Student Affairs Fellowship Award in 2006. He served as Session Chair at IASTED PDCS-2008 and IMETI CCCT-2009, both in Orlando, Florida. He served as reviewer of NSF TUES (2011) and GRFP (2012) programs. He is currently serving as a TPC member of IEEE IPCCC 2012 and as an IPC member of IEEE ICCIT 2012 conferences.