Computer Applications: A Novel Architecture to Improve the Performance of Audio Visual Applications

Abu Asaduzzaman[1*]
Kishore K. Chidella[2]
Fadi N. Sibai[3]
[1]*Wichita State University, USA*
[2]*University of Nevada at Las Vegas, USA*
[3]*American International University, Kuwait*
abu.asaduzzaman@wichita.edu

Abstract

To attain the best audio-visual experience, the underlying computing platform should provide high performance in real time. The forthcoming computer systems consist of multicore processors to achieve a high performance-to-power ratio. The algorithms in visual computing are becoming highly complex to meet the requirements. According to the new design paradigm, a simultaneous exploration of multicore architecture and complicated algorithms is beneficial to achieve the design goals for visual systems. However, caches in multicore architecture multiply the timing unpredictability and that creates a serious challenge in running real-time audio/visual applications in multicore systems. In this work, we introduce a novel multicore architecture with miss tables inside level-1 caches to improve performance and decrease power consumption. Miss table holds block address information regarding the application being processed that causes cache misses. Miss table information is used for efficient selection of the blocks to be locked or victim blocks to be replaced. This approach improves predictability by locking important blocks inside the cache during the execution time. At the same time, this approach decreases average delay per task and total power consumption by reducing cache misses when the right cache blocks are locked and/or replaced. We simulate an 8-core architecture that has 2 levels of caches using the Moving Picture Experts Group (MPEG)-4 decoding and Fast Fourier Transform (FFT) workloads. Simulation results show that a reduction of 42% in mean delay per task and a reduction of 40% in total power consumption are achieved by locking 20% of the total level-1 instruction (I1) cache size.

Keywords: Computer applications, multicore architecture, computing performance, cache memory hierarchy, audio-visual applications

1. Introduction

Visual computing uses a visual paradigm rather than a conventional text paradigm and deals with large image files including video sequences (Visual Computing, 2017). The algorithms required for future visual systems are much more complicated than those currently being used. Current single-core processors are not adequate to support future visual computing. According to the newly emerged

Submitted August 14, 2022
Revised September 7, 2022

multicore design, two cores running at one-half of the frequency can approach the performance of a single core running at full frequency, while the dual-core consumes significantly less amount of power. Because of their high performance/power ratio, the multicore processors open new possibilities for system designers in implementing highly complex visual computing algorithms (Suhendra and Mitra, 2008; Chiappetta, 2007). In a multicore architecture, two or more independent cores are combined into a die. In most cases, each processor has its own private level-1 cache memory (CL1). Normally, the CL1 is split into instruction (I1) and data (D1) caches. Also, multicore processors may have one shared level-2 cache (CL2) or multiple distributed and dedicated CL2s (Burt, 2009; Online: Wikipedia, 2022). Cache parameters (such as cache size, line size, and associativity levels) significantly influence system performance (Asaduzzaman et al., 2006). Multicore architectures are more suitable for real-time audio/visual applications because the concurrent execution of tasks on a single processor is inadequate for achieving the required level of performance and reliability. The integration of billions of transistors in a single chip is now possible. As a result, the multicore design trend is expected to grow for the next decade. However, the cache memory introduces timing unpredictability and real-time applications demand timing predictability and cannot afford to miss deadlines. Therefore, it becomes a great challenge to support real-time multimedia applications on multicore systems. Knowing that performance can be sacrificed in real-time systems, a better use of the cache can be very useful to improve predictability and performance and reduce total power consumption.

Cache locking shows promises to improve predictability (Asaduzzaman et al., 2007; Puaut, 2006; Tarui et al., 1992). Cache locking is defined as the ability to put off some or all of the data or instruction cache from being overwritten. Cache entries can be locked either for individual ways within the cache or for the entire cache. In way locking, only a portion of the cache is locked by locking ways within the cache. Unlocked ways of the cache behave normally. Way locking improves predictability and performance; using way locking, the Intel Xeon processor achieves the effect of using local storage by the Synergistic Processing Elements (SPEs) in the International Business Machines (IBM) Cell processor architecture (Thompson, 2012). In entire cache locking, cache hits are treated in the same manner as hits to an unlocked cache. Cache misses are treated as cache-inhibited access. Invalid cache entries at the time of locking will remain invalid and inaccessible until the cache is unlocked. Entire cache locking is inefficient when the data size or instructions to be locked is smaller compared to the cache size.

In currently available single-core way cache locking techniques, the blocks to be locked are selected by offline analysis of the applications. The information collected by offline analysis can be post-processed and re-used in single-core and/or multicore systems to facilitate the efficient selection of blocks to be locked and/or victim blocks to be replaced. In this work, we introduce miss tables inside level-1 caches of multicore processors to hold block address information that causes most or all cache misses. This approach improves predictability and decreases average delay per task and total power consumption by reducing cache misses as the right cache blocks are locked and/or replaced.

Paper Organization: In Section 2, related literature is discussed. The proposed multicore architecture with miss tables in CL1s is introduced in Section 3. In Section 4, the proposed scheme is evaluated by presenting important simulation results. Finally, this work is concluded in Section 5.

2. Literature Survey

Predictability and performance improvement for visual computing systems with cache memories has become an important research topic in the past years. A lot of work has been done to improve predictability in single-core systems by cache locking. In this section, we first briefly discuss cache memory hierarchy, followed by some existing single-core cache-locking techniques. Then, we present some cache memory hierarchies used by contemporary popular multicore processors. In the following section, we introduce our proposed multicore architecture with miss tables in CL1s.

Cache memory has a very rich history in the evolution of modern computing (Lutkevich, 2020). Cache memory is first seen in the IBM System/360 Model 85 in the late 1960s. In 1989, Intel 468DX microprocessor introduced an on-chip 8 KB CL1 cache for the first time. In the early 1990s, off-chip CL2 cache appeared with 486DX4 and Pentium microprocessor chips. Today's microprocessors usually have 128 KB or more of CL1, 512 KB or more of CL2, and an optional 2 MB or more CL3. Some CL1 cache is split into I1 and D1 to improve performance (Inkley et al., 2006). Intel Pentium 4 processor, one of the most popular single-core processors that use inclusive cache architecture, is discussed in (Pentium 4, 2022). A typical inclusive cache is shown in Figure 1. CL2 contains each and every block 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 is a single-core processor that has an on-die 256 KB inclusive level-2 cache; with an 8 KB level-1 trace/instruction cache (I1) and 8 KB level-1 data cache (D1).

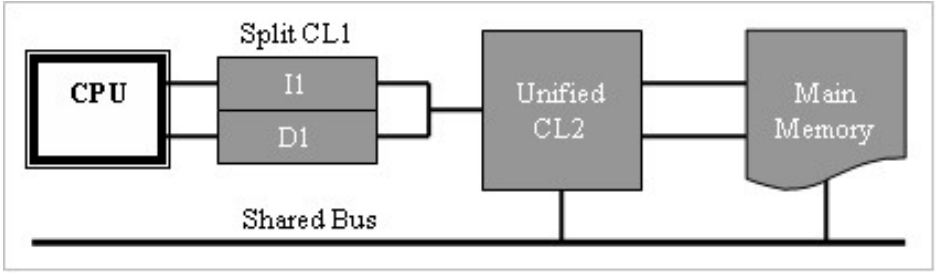

Figure 1: Inclusive cache architecture

In (Vera et al., 2003), a memory hierarchy with a dynamic locking cache is proposed to provide high performance combined with high predictability for complex systems. They conclude in an intuitive way that faster execution times are possible. However, it is very difficult to provide the necessary locking information into the program code since both the hardware and the software might be involved. In (Arnaud et al., 2005), a methodology using a genetic algorithm is proposed that can be used to select a set of instructions to be preloaded and locked in the cache. This scheme may improve predictability by selecting the target set of instructions efficiently. However, the algorithm used in this scheme can be also misleading, because it uses neither the information about task structure nor the problem parameters. In (Tamura et al., 2005), the impact of three different fitness functions is studied. These fitness functions are used in a genetic algorithm that selects the contents of a static locking cache memory in a real-time system. Results indicate that none of the fitness functions perform well in the two basic metrics used. In (Tamura et al., 2004), 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. However, a better analysis that classifies the cache

accesses as misses or hits and locks fewer regions could be beneficial. In (Campoy et al., 2005), an algorithm proposed dynamic cache locking 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 caches. However, this technique is not capable of power estimation – a crucial design factor for embedded systems. Techniques discussed here are developed to evaluate predictability in a single-core system and they are not adequate to analyze performance, power consumption, and predictability of multicore real-time systems.

Most manufacturers are adopting multicore processors to acquire the required high processing power for future visual computing systems. Popular multicore processors from Intel, AMD, and IBM have multilevel caches (Inkley et al., 2006; Chiappetta, 2007; Burt, 2009; Online: Wikipedia, 2022). AMD quad-core Opteron has 256 KB I1, 256 KB D1, 2 MB dedicated CL2, and 2 MB (Santa Rosa) or 4 MB (Deerhound) shared CL3. As shown in Figure 2, the Intel quad-core Xeon DP family has 128 KB I1, 128 KB D1, and 8 MB shared CL2 (Inkley et al., 2006).

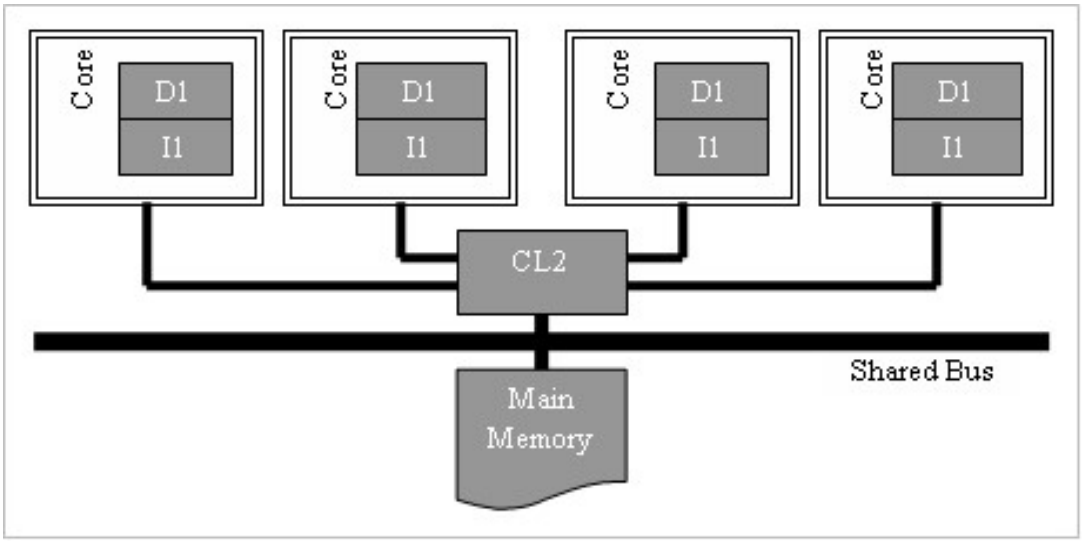

Figure 2: Intel quad-core Xeon DP family architecture

IBM's Cell multicore processor has a Primary Processing Entity (PPE) like IBM dual-threaded PowerPC and 8 Cells. Figure 3 shows a Cell-like multicore architecture (Blachford, 2005; Vance, 2006). Each Cell is also called a Synergistic Processing Element (SPE). The PPE contains a 32 KB I1 and a 32 KB D1 cache. A 512 KB CL2 is shared by the PPE and SPEs. Primarily the PowerPC PPE keeps the processor compatible with lots of applications. Each SPE has 256 KB static random access memory (SRAM) and a 4 x 128-bit ALU (Arithmetic Logical Unit which does the math in a processor), and 128 of 128-bit registers. The Element Interconnect Bus (EIB) is the communication bus internal to the Cell processor which connects the various on-chip system elements: PPE processor, memory controller (MIC), eight SPE coprocessors, and two off-chip I/O interfaces.

Various shared and distributed cache memory organizations and the impact of sharing and privatizing them on performance in homogeneous multicore architectures are studied in (Sibai, 2008).

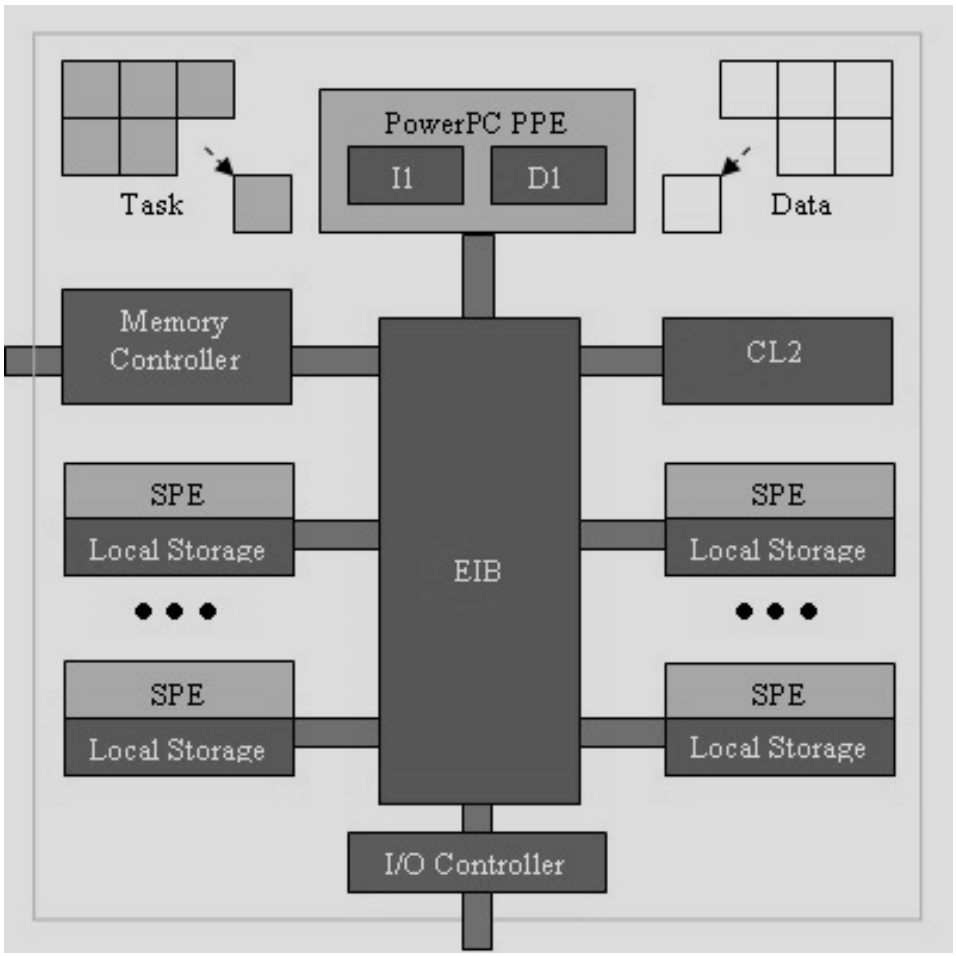

Figure 3: IBM Cell-like multicore architecture

3. Proposed Multicore Architecture with Miss Tables in CL1s

We have discussed the cache memory hierarchy in some popular single-core and multicore processors and some existing cache-locking techniques. In this section, we introduce our proposed multicore architecture with miss tables (MT) in CL1s. The miss table is populated with the block information that causes cache misses by post-processing the results of the worst-case execution time (WCET) analysis of the applications. Using MT, the proposed architecture can take advantage of multicore cache locking, selective pre-loading, and a better cache replacement strategy. In the following subsections, we discuss the architecture, miss table, cache replacement policy, and multicore cache locking algorithm.

3.1 Architecture

Figure 4 illustrates the proposed multicore architecture with 8 cores. Along with MT, each core has one private CL1 which is split into I1 and D1 for improved performance. We use Intel-like shared CL2 in our proposed architecture. CL2 may be partitioned into parts to reduce bus contention so that only a few cores can access each part.

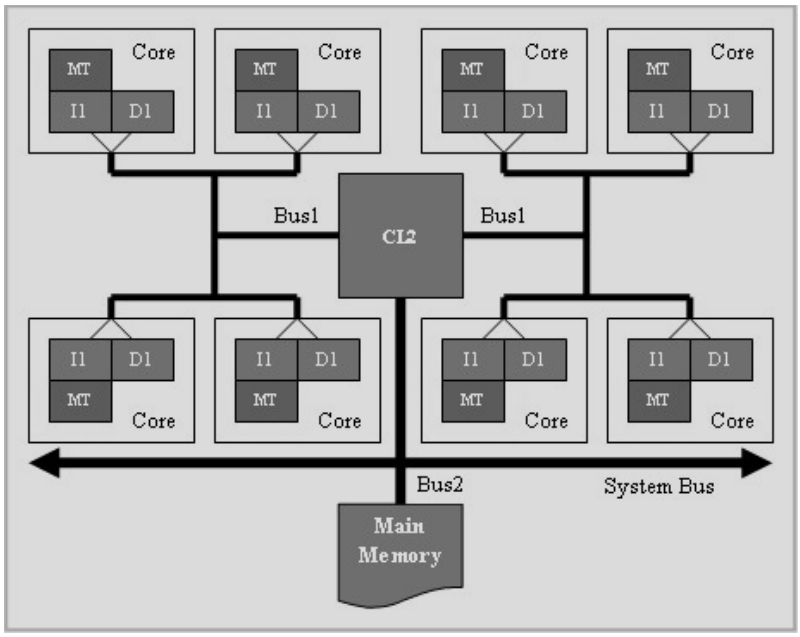

Figure 4: Simulated 8-core architecture with MT

3.2 Miss table

A miss table (MT) is a table that contains the addresses of all or top-most blocks that cause cache misses – block addresses are sorted in descending order of the miss numbers. For each application, after post-processing the tree graph (generated by Heptane), one MT is generated to be used in the simulation program.

3.3 Cache replacement policy

In this work, we consider I1 cache locking. A better cache replacement policy is adopted for I1. Using the MT, this policy always selects unlocked blocks with a minimum number of misses. In case of a tie in the number of misses, a block is selected randomly. However, a random cache replacement policy is used in D1 and CL2.

3.4 Multicore cache locking algorithm

In this subsection, we present the algorithm of the proposed multicore way locking technique. According to this algorithm, incoming tasks (i.e., requests) come to the controller of the multicore device. The controller assigns the incoming task to a free core (if any) and marks the core as busy. If no free core is found, incoming tasks are put into a waiting queue and served when cores become available (i.e., free) on a first come first serve basis. Depending on the assigned task to a core, the respective MT is populated, the pre-selected blocks are loaded, and a portion of the cache (I1 in this work) is locked. Then the core starts processing the task. In case of a cache miss, it selects the victim block (as the cache is already full) using the MT and modified cache replacement policy. The selected victim block is an unlocked block with the minimum number of misses. After the current task is finished, the core is made free (i.e., available) for performing another task.

4. Evaluation

We develop a simulation platform to evaluate our proposed multicore processor architecture. First, we briefly discuss the simulation details. Then, we evaluate our work by presenting some important simulation results.

4.1 Simulation details

We simulate the multicore architecture and cache locking using Moving Picture Experts Group's MPEG4 (part-2) decoding and Fast Fourier Transform (FFT) workloads. The code size for MPEG4 and FFT are 29,937 Bytes and 2,335 Bytes, respectively. We generate a tree graph for each application that shows the blocks that cause misses by single-core WCET analysis. By post-processing the tree graph, we create an MT for each application. We use VisualSim to develop the simulation platform, run the simulation program, and collect the results. In this subsection, we briefly discuss some assumptions, simulation tools, and important input and output parameters.

4.1.1 Assumptions

We make the following important assumptions,

- I1 way cache locking is implemented in this work.
- A modified cache replacement strategy is used for I1 using MT information.
- Write-back memory update policy is used.
- The (memory latency) delay introduced by the bus that connects CL2 and the main memory (Bus2 in Figure 1) is 15 times longer than the delay introduced by the bus that connects CL1 and CL2 (Bus1 in Figure 1).

4.1.2 Simulation tools

VisualSim and Heptane simulation tools are used in this work. VisualSim (a.k.a., VisualSim Architect) from Mirabilis Design is a graphical system-level simulation tool (VisualSim, 2022). We install VisualSim in Windows XP on a Dell PowerEdge 1600SC PC. Using VisualSim, we model the abstracted architecture. Using VisualSim simulation cockpit, we run the simulation program and obtain simulation results. The results are stored as text and/or graph files.

Heptane (Hades Embedded Processor Timing ANalyzEr) is a WCET analysis tool for embedded systems (Heptane, 2022). We configure Heptane in Red Hat Linux 9 in the same Dell PowerEdge 1600SC PC (where we have VisualSim in Windows XP). Once the configuration file is created, Heptane is run by typing “heptane-run.sh”' in a command shell provided that the PATH variable contains the directory where Heptane is installed. The results are placed in the directory as specified in the configuration file and are viewed through a Web browser by opening the file “HTML/index.html”.

4.1.3 Input and output parameters

Important input parameters are shown in Table 1. We simulate an 8-core system with I1, D1, and

CL2. We keep CL2 size fixed at 128 KB and vary the I1/D1 size, CL1/CL2 line size, and associativity level. Output parameters include average delay per task and total power consumption.

Table 1: Important Input Parameters

Parameter	Value
I1 (/D1) cache size (KB)	4, 8, 16, 32, or 64
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)	128 (fixed)
Number of cores	8 (fixed)

4.2 Results

In this work, we propose a multicore architecture with miss tables in CL1s to enhance the predictability and performance of power-aware visual computing systems. We model a system with 8 processing cores and run the simulation program using MPEG4 decoding and FFT algorithms. We obtain results by varying the I1 (/D1) cache size, I1 (D1/CL2) line size, and I1 (D1/CL2) associativity level with and without applying I1 cache locking. Based on our previous work (Asaduzzaman et al., 2006; Asaduzzaman et al., 2013), we apply 20% cache locking in this work for optimal predictability. In the following subsections, we discuss the impact of I1/D1 cache size, CL1/CL2 line size, and CL1/CL2 associativity level on average delay per task and total power consumption.

4.2.1 I1/D1 Cache size

We obtain the average delay per task for various I1/D1 cache sizes for no locking and 20% I1 locking using MPEG4 decoding and FFT workload. As shown in Figure 5, for any I1 cache size used, the mean delay per task for both MPEG4 decoding and FFT workloads decreases when I1 cache is locked partially. Results also show that the average delay per task decreases with the increase in I1 cache size.

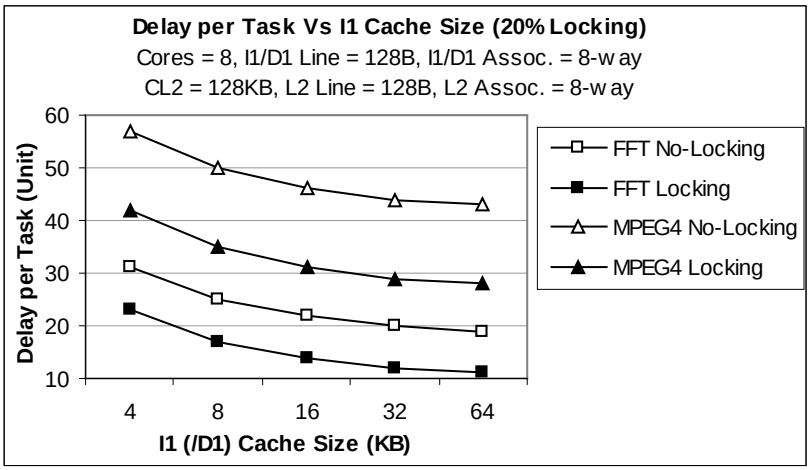

Figure 5: Mean delay per task versus I1 cache size

Experimental results show that regardless of the I1 cache size, total power consumption due to both MPEG4 decoding and FFT decreases when we compare no locking with I1 way locking [Figure 6]. Again, total power consumption decreases with the increase in I1 cache size. The decrease in total power consumption is significant for smaller I1 (4KB to 16KB).

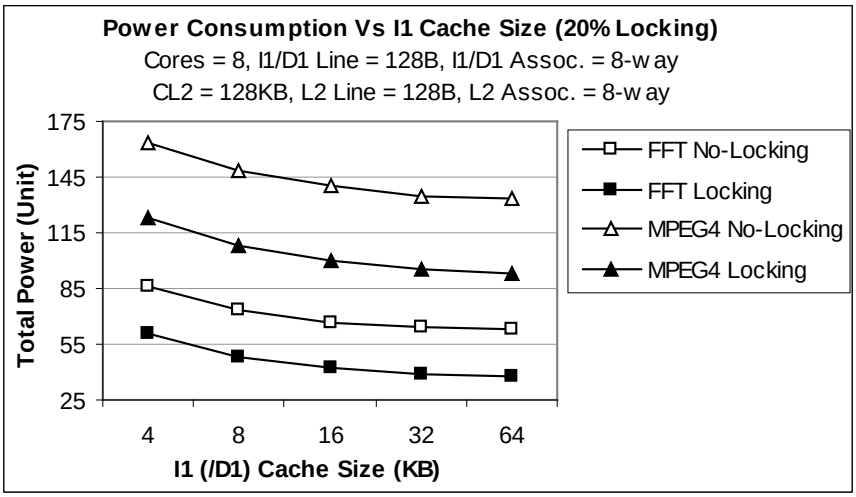

Figure 6: Total power consumption versus I1 cache size

4.2.2 L1/CL2 Line size

We use the same line size for I1, D1, and CL2. Figure 7 shows the average delay per task versus (I1) line size for no locking and 20% I1 way locking. We observe that the average delay per task goes down for MPEG4 and FFT workload, regardless of the line size, with increasing line size leveling off at 128B. For a line size greater than 128B, the average delay per task increases due to cache pollution.

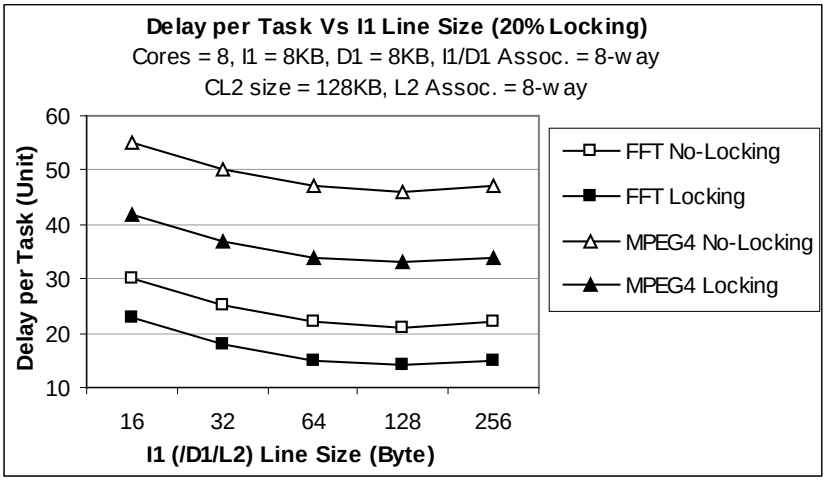

Figure 7: Mean delay per task versus I1 line size

Simulation results also show that total power consumption goes down for MPEG4 and FFT, regardless of the line size, with increasing line size leveling off at a line size of 128B (see Figure 8). It is also observed that for line size greater than 128B, the total power consumption increases.

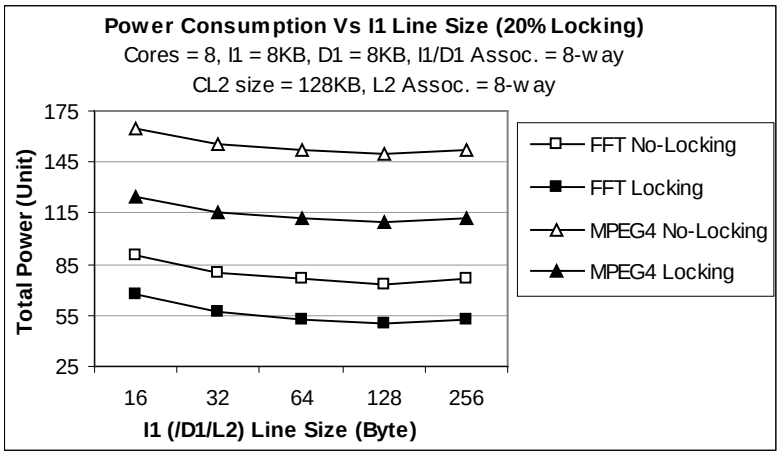

Figure 8: Total power consumption versus I1 line size

4.2.3 CL1/CL2 Associativity level

We use the same associativity level for I1, D1, and CL2. The impact of no locking and 20% I1 way locking on the mean delay and total power consumption by varying (I1) associativity level is shown in Figure 9. Experimental results show that for any I1 associativity level, the mean delay per task for both MPEG4 and FFT decreases when we move from no locking to 20% I1 way locking. Also, the decrease in the mean delay per task is significant for smaller levels of associativity (1-way to 2-way), after 4-way the mean delay per task remains almost the same.

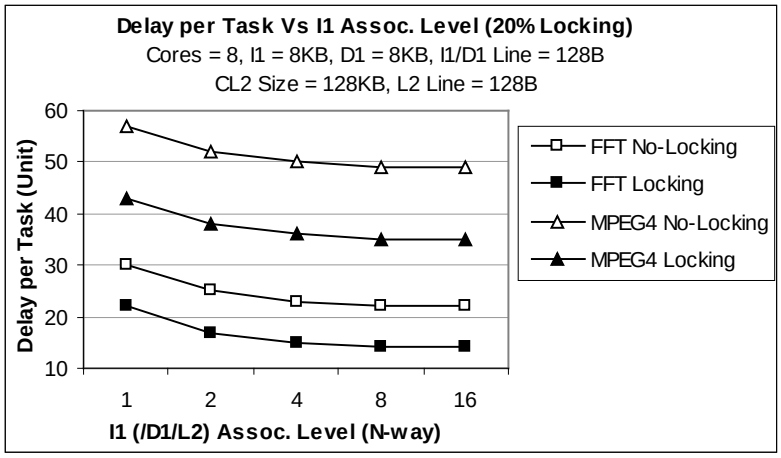

Figure 9: Mean delay per task versus I1 associativity level

We present the total power consumption due to different (I1) associativity levels for no locking and I1 way-locking in Figure 10. Simulation results show that regardless of the (I1) associativity level, the total power consumption due to both MPEG4 and FFT decreases when we compare no locking with 20% I1 way locking. Results also show that total power consumption decreases with the increase in (I1) associativity level. However, the decrease in total power consumption is significant between 1-way and 4-way, after that total power consumption remains almost the same.

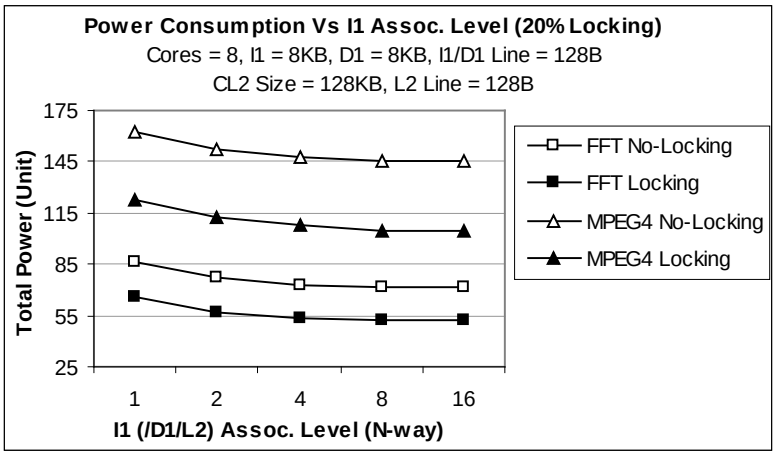

Figure 10: Total power consumption versus I1 associativity level

Finally, we summarize the changes in mean delay and total power consumption. The decrement in value is represented with leading (-) and the increment in value is represented with leading (+). As shown in Table 2, both the mean delay per task and total power consumption decreases when 20% I1 cache locking is applied in a proposed 8-core processor with MTs in CL1s. For I1 = D1 = 64KB, the mean delay per task is decreased by 42% and total power consumption is decreased by 40% for FFT (see Table 2, Figure 5, and Figure 6).

Table 2: Changes Due to 20% I1 Cache Locking

Parameters (Values)	Application Name	Changes (%)	
		Delay	Power
I1 Cache size (64 KB)	MPEG4	(-)35	(-)30
	FFT	(-)42	(-)40
I1 Line size (256 Byte)	MPEG4	(-)28	(-)26
	FFT	(-)32	(-)30
I1 Assoc. level (16-way)	MPEG4	(-)29	(-)27
	FFT	(-)36	(-)28

5. Conclusion

Traditional single-core processors are not capable of supporting future audio-visual computing. Multicore architecture provides a new platform for implementing highly complex audio-visual computing algorithms. The algorithms are becoming very complicated to solve the forthcoming problems efficiently. Complex multicore processors with multilevel caches are being deployed in both desktop and embedded computing to achieve a high performance-to-power ratio. Caches in multicore architecture make the execution time unpredictability worse. Therefore, running real-time applications on multicore visual systems becomes a serious challenge. In this work, we present a multicore processor with miss tables inside level-1 caches to increase predictability and performance-to-power ratio by cache locking. Miss table holds the block address information regarding the current applications being processed that causes cache misses. This approach reduces cache misses by locking/replacing the right cache blocks inside the cache which boost predictability and reduces

average delay per task and total power consumption. We evaluate our proposed architecture by simulating an 8-core processor that has 2 levels of caches using MPEG4 decoding and FFT workload.

We find the proposed multicore processor with miss tables very promising. Experimental results show that a reduction of 35% for MPEG4 decoding and 42% for FFT in mean delay per task and a reduction of 30% for MPEG4 decoding and 40% for FFT in total power consumption are achieved by locking 20% of the total I1 cache size. Using the miss table information, better use of cache is possible to improve predictability and performance to power ratio. We plan to investigate the impact of miss table inside CL2 as an extension of this work in the near future.

6. References

Arnaud, A., Puaut, I. (2005). Dynamic Instruction Cache Locking in Hard Real-Time Systems. Retrieved on Oct. 30, 2022, from https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.332.4514

Asaduzzaman, A., Limbachiya, N., Mahgoub, I., Sibai, F. (2007, August 18-20). *Evaluation of I-Cache Locking Technique for Real-Time Embedded Systems*. In 2007 Innovations in Information Technologies (IIT), Dubai, United Arab Emirates.

Asaduzzaman, A., Mahgoub, I., (2006). Cache Modeling and Optimization for Portable Devices Running MPEG-4 Video Decoder. *Journal of Multimedia Tools and Applications (MTAP)*, 28(1), 239-256.

Asaduzzaman, A., Suryanarayana, V.R., Sibai, F.N. (2013). On level-1 cache locking for high-performance low-power real-time multicore systems. *Journal of Computers & Electrical Engineering*, 39(4), 1333-1345.

Blachford, N. (2005). Cell Architecture Explained Version 2. Retrieved on Oct. 30, 2022, from http://www.blachford.info/computer/Cell/Cell0_v2.html

Burt, J. (2009). IBM Says Cell Processor Technology Will Continue. Retrieved on Oct. 30, 2022, from https://www.eweek.com/networking/ibm-says-cell-processor-technology-will-continue/

Campoy, A.M., Tamura, E., Saez, S., Rodriguez, F., Busquets Mataix, J.V. (2005). *On Using Locking Caches in Embedded Real-Time Systems*. In: Yang, L.T., Zhou, X., Zhao, W., Wu, Z., Zhu, Y., Lin, M. (eds) Embedded Software and Systems. International Conference on Embedded Software and Systems (ICESS) 2005. Lecture Notes in Computer Science, 3820, 150-159. Springer, Berlin, Heidelberg.

Chiappetta, M. (2007). AMD Barcelona Architecture Launch: Native Quad-Core. Retrieved on Oct. 30, 2022, from https://hothardware.com/reviews/amd-barcelona-architecture-launch-native-quadcore

Heptane (2022). Online: inria. Heptane software - A tree-based Worst-Case Execution Time (WCET) analysis tool. Retrieved on Oct. 30, 2022, from https://team.inria.fr/pacap/software/heptane/

Inkley, B. and Tetrick, S. (2006). Intel Multi--core Architecture and Implementations. Retrieved on Oct. 30, 2022, from https://en.wikichip.org/w/images/c/c1/Multicore_MATS002_999_pct.pdf

Lutkevich, B. (2020). *Cache Memory*. TechTarget. Retrieved on Oct. 30, 2022, from https://www.techtarget.com/searchstorage/definition/cache-memory

Multi-core computing: Xeon and Athlon. (2022-July 30). In Wikipedia. http://en.wikipedia.org/wiki/Xeon , https://en.wikipedia.org/wiki/Athlon

Pentium 4. (2022). Online: hardware secrets. Inside Pentium 4 Architecture. Retrieved on Oct. 30, 2022, from https://hardwaresecrets.com/inside-pentium-4-architecture/

Puaut, I. (2006). Cache Analysis Vs Static Cache Locking for Schedulability Analysis in Multitasking RealTime Systems. Retrieved on Oct. 30, 2022, from https://www.cs.york.ac.uk/rts/wcet2002/papers/puaut.pdf

Sibai, F.N. (2008). On the Performance Benefits of Sharing and Privatizing Second and Third Level Cache Memories in Homogeneous Multi-Core Architectures. *Microprocessors and Microsystems*, 32(7), 405-412.

Suhendra, V. and Mitra, T. (2008). Exploring Locking & Partitioning for Predictable Shared Caches on Multi-Cores. In 2008 45th ACM/IEEE Design Automation Conference, Anaheim, CA, USA.

Tamura, E., Busquets-Mataix, J. V., Martín, J.J.S., Campoy, A.M. (2005). A Comparison of Three Genetic Algorithms for Locking-Cache Contents Selection in Real-Time Systems. In: Ribeiro, B., Albrecht, R.F., Dobnikar, A., Pearson, D.W., Steele, N.C. (eds) Adaptive and Natural Computing Algorithms. Springer, Vienna. https://doi.org/10.1007/3-211-27389-1_111

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

Tarui, T., Nakagawa, T., Ido, N., Asaie, M., Sugie, M. (1992). Evaluation of the lock mechanism in a snooping cache. In ACM Proceedings of the 6th international conference on Supercomputing. https://dl.acm.org/doi/pdf/10.1145/143369.143384

Thompson. (2012). Scratchpad memory: Shared L2 vs Cell local stores. Retrieved on Oct. 30, 2022, from https://scratchpaddmemory.blogspot.com/2012/05/shared-l2-vs-cell-local-stores.html

Vance, A. (2006-January, 22). Cell processor goes commando. The Register. Retrieved on Oct. 30, 2022, from http://www.theregister.co.uk/2006/01/22/cell_mecury_army/

Vera, X., Lisper, B., Xue, J. (2003). Data Cache Locking for Higher Program Predictability. *ACM SIGMETRICS Performance Evaluation Review*, 31(1), 272–282. https://doi.org/10.1145/885651.781062

Visual Computing (2017). Online: techopedia. Retrieved on Oct. 30, 2022, from https://www.techopedia.com/definition/16286/visual-computing

VisualSim (2022). Online: mirabilisdesign. VisualSim - A system-level simulator from Mirabilis Design, Inc. Retrieved on Oct. 30, 2022, from https://www.mirabilisdesign.com/visualsim/

7. Appendix

7.1 Appendix A: VisualSim Block Diagram

Figure A shows the VisualSim model of the proposed 8-core processor. In VisualSim, a system to be evaluated is described in three parts – Architecture, Behavior, and Workload. The architecture includes elements like processing core, cache, bus, and main memory. Behavior describes the actions performed on the system. Examples include network traffic shaping. The workload is the transactions that traverse the system such as network traffic.

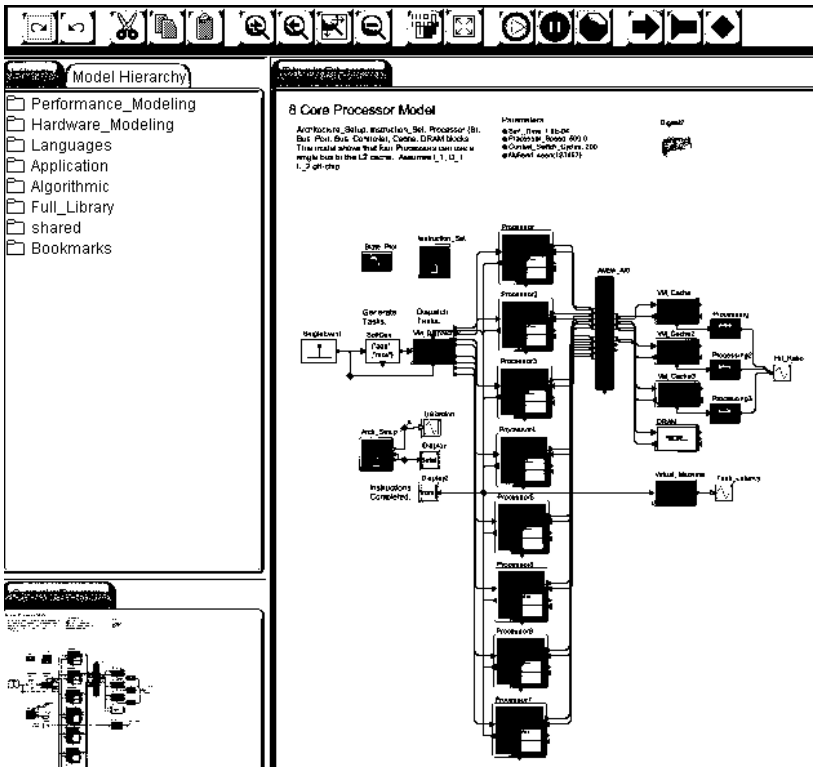

Figure A: 8-core processor model

7.2 Appendix B: VisualSim Simulation Cockpit

Figure B shows the VisualSim Simulation Cockpit. The Simulation Cockpit provides functionalities (left-top in Figure B) to run the model and to collect simulation results (right side in Figure B). Parameters can be changed before running the simulation without modifying the block diagram. The final results can be saved into a file and/or printed for further analysis.

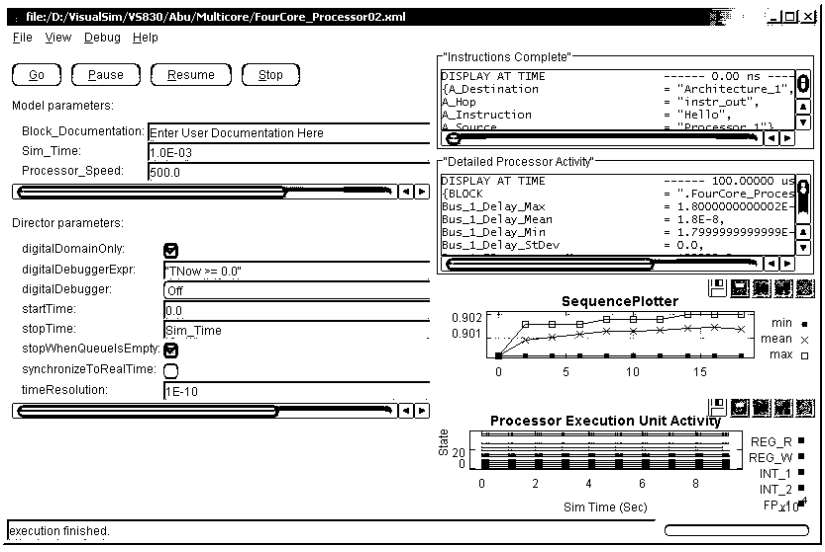

Figure B: VisualSim simulation cockpit for the model in Figure A