Case Studies: Memory Behavior of Multithreaded Multimedia and AI Applications

Lu Peng*, Justin Song, Steven Ge, Yen-Kuang Chen,
Victor Lee, Jih-Kwon Peir*, and Bob Liang

* *Computer Information Science & Engineering*
University of Florida
{lpeng, peir}@cise.ufl.edu

Architecture Research Lab
Intel Corporation
{justin.j.song, steven.ge, yen-kuang.chen,
victor.w.lee, bob.liang}@intel.com

Abstract

Memory performance becomes a dominant factor for today's microprocessor applications. In this paper, we study memory reference behavior of emerging multimedia and AI applications. We compare memory performance for sequential and multithreaded versions of the applications on multithreaded processors. The methodology we used including workload selection and parallelization, benchmarking and measurement, memory trace collection and verification, and trace-driven memory performance simulations. The results from the case studies show that opposite reference behavior, either constructive or disruptive, could be a result for different programs. Care must be taken to make sure the disruptive memory references will not outweigh the benefit of parallelization.

1. Introduction

Multimedia and Artificial Intelligent (AI) applications have become increasingly popular in microprocessor applications. Today, almost all the commercial processors, from ARM to Pentium®, include some type of media enhancement in their ISA and hardware [1][2]. While there has been much work put into studying memory performance for general integer and large-scale scientific applications due to the widening processor-memory performance gap, this paper focuses on memory behavior studies for multimedia and AI applications.

The memory reference behavior study of the emerging applications is complicated by the fact that general-purpose processors have begun to support multithreading to improve throughput and hardware utilization. The new Hyper-Threading Technology brings the multithreading idea into Intel architecture to make a single physical processor operated as two logical processors [3]. Active threads on each processor have their own local states, such as Program Counter (PC), register file, and completion logic, while sharing other expensive components, such as functional units and caches. On a multithreaded processor, multiple active threads can be homogenous (from the same application), or heterogeneous (from different independent applications). In this paper, we investigate memory-sharing behavior from homogenous threads of emerging multimedia and AI workloads.

Two applications, *AEC* (Acoustic Echo Cancellation) and *SL* (Structure Learning), were examined. AEC is widely used in telecommunication and acoustic processing systems to effectively eliminate echo signals [4]. SL is widely used in bioinformatics for modeling of gene-expression data [5]. Both applications were parallelized using OpenMP [6] and were run on Pentium® 4 processors with/without Hyper-Threading Technology [7].

When multiple threads are running in parallel on a multithreaded processor with shared caches, their memory behavior is influenced by two essential factors. First, the memory reference sequence among multiple threads can be either *constructive* or *disruptive*. The memory behavior is constructive when multiple threads share the data that was brought into the shared cache by one or another [8]. In other words, memory reference locality exhibits in the original single thread may be transformed and/or further enhanced among multiple threads. The memory behavior is disruptive when multiple threads have rather distinctive working sets and compete with the limited memory hierarchy resources. Second, the memory usage or footprint may increase among multiple threads based on the fact that certain data structures or local variables may be replicated to improve parallelism. The increase of memory footprint

alters the reference behavior and demands larger caches to hold the working set.

There have been many studies targeting memory performance for commercial workload [9,10] as well as media [11] and Java applications [12,13]. In [11], hardware trace collection and trace-driven simulation is used to study the memory behaviors of a group of multimedia applications running on different Operating Systems including Linux, Window NT and Tru-Unix. Techniques of using compiler-based, dynamic profile-based, or user annotation-based methods to improve cache performance were discussed [14,15,16,17,18]. Instead of inventing new hardware or software techniques to improve memory reference behavior, this paper demonstrates a methodology for memory behavior analysis and shows initial simulation results of the latest multimedia and AI applications. The detailed analysis methodology will be described in the next section. Section 3 compares the memory reference behavior based on data reuse distances of the multithreaded AEC and SL. In comparison with the original sequential program, we found that the reuse distance of AEC increases with the number of parallel threads. On the contrary, the reuse distance of SL decreases with the degree of parallelization. The reasoning for such opposite memory behaviors will be discussed. Finally, Section 4 concludes this study.

2. Methodology for Memory Behavior Studies

Execution-driven, whole-system simulation method has become increasingly popular for studying memory reference behavior of multithreaded applications. In this paper, we use Simics [19], a whole system simulation tool, which is capable of running unmodified commercial operating system and applications. We first select interesting, emerging media and AI applications as the workload for our studies. The selected applications are parallelized using OpenMP and run on Pentium® 4 systems for performance measurement. After demonstrating performance advantages, the parallelized applications are ported and run on the Simics simulation environment.

A precise, cycle-based memory model can be built and integrated with Simics virtual machine to drive the simulation. However, in order to obtain fast performance approximations over a large design space, we decided to apply trace-driven simulations based on the trace collected from the Simics simulation environment. The generic cache model in the Simics tool set was modified for trace collection. The characteristics of the collected trace is compared and verified with the measurement results. Detailed trace-driven simulations are followed based on various cache sizes, topologies and degrees of parallelization. The above steps are described below.

Workload Selection and Parallelization: We selected two emerging applications, AEC (Acoustic Echo Cancellation) and SL (Structure Learning), in this study. First, beside video communication, audio is also an important component in an interactive communication; for example, audio glitch is often more annoying than video glitch. While many studies have been working on video related workload, limited studies have been done on audio communication. To make sure that future processors deliver good acoustic quality, we select AEC [4] as the first workload. Second, beside multimedia communication, modern processors start to play an important role in the bioinformatics studies, e.g., estimating certain expressions by transcription (amount of mRNA made from the gene). Often the best clue to a disease or measurement of successful treatment is the degree to which genes are expressed. Such data also provides insights into regulatory networks among genes, i.e., one gene may code for a protein that affects another's expression rate. We select SL [5], which models gene-expression profile data, as the second workload.

The selected applications are parallelized with OpenMP and compiled with Intel compiler 7.1 release [20]. In this environment, the OpenMP master thread automatically creates and maintains a team of worker threads; each executing specific tasks assigned by the master thread. All worker threads will not exit until the end of process, thus forming a pool of active threads. Basically, the main parallelization for both workloads is done on *for-loops*. For-loop parallelization can be specified using the OpenMP specification. Task queue based parallelization is also applied, which allows *do-while* style parallelization. The first parallel region hitting thread enqueues parameters needed by worker threads, and other threads fetch parameters in stack and begin execution. With the task queue mode, dynamic execution of threads does not have to be identical.

Benchmarking and Measurement: The parallelized applications first run on Pentium® 4 systems. To avoid excessive overheads associated with thread scheduling, we only run the parallelized programs with the number of workers less than or equal to the capacity of the physical system. For example, in our studies on a Quad 2.8GHz Pentium® 4 Xeon™ MP machine with 2GB DDR SDRAM, a maximum of 4 worker threads are generated. We use VTune™ performance analyzer [21] to collect statistics including execution clock ticks, number of instructions, number of memory references, cache hits/misses, etc. From these measurement results, information, such as speedups and the total memory

references among sequential and parallel executions of the applications can be calculated. Furthermore, we use the Windows-based Perfmon [22] to collect page-fault information for calculating the memory usage. Since physical memory is much larger than the maximal memory requirement of both applications, once a page comes into working set, it will not be swapped out. In other words, this counting should provide a reasonable estimation for the pages needed by the applications. There are two important examinations:

(1) The total memory usage and the total memory references for the single and multiple threaded versions of each application are compared. These measurement results are then verified against what presents in the sequential and parallel source code.
(2) The execution time and speedup of the single and multiple threaded versions of the application are also compared against the potential speedup exhibited in the source code.

Various problem sizes are tested to understand parallelization overheads and memory hierarchy limitations associated with the program. Proper problem sizes are selected for memory behavior studies with the following considerations. First, the problem size must be reasonable large to be practical for real applications. Large problem sizes present significant demand on memory hierarchy for our studies. Large problem sizes can also amortize overheads paid to initialize and synchronize multiple threads. Second, the problem size should not be too big to create distortions due to cache limitations on both the measurement and the simulation environments. In addition, large problem sizes may increase the simulation time and make it difficult to handle for trace-driven simulations.

Memory Trace Collection: Trace-driven simulation with sequential and parallel versions of the source code is applied to study memory behavior. Memory traces are collected under Simics. Based on the VTune™ measurement, proper problem sizes are selected. Both sequential and parallel versions of the program are running on the Simics environment for trace collections. To mimic hardware-collected traces, a memory hierarchy model with a generic cache is integrated to enable execution-driven simulations. The memory model is invoked whenever a memory instruction is executed. A search through the generic cache is performed. Upon a miss, the thread will be delayed a specific cycles assigned a priori that matches the real hardware delay on the experimental system to provide more accurate reference sequence among multiple threads.

The trace-driven simulation has two advantages over the execution-driven simulation. First, it is possible to post-processing traces for ease of study of block reuse distances, optimal replacement policy, etc. Second, it is much faster to simulate traces off-line comparing with on-the-fly, execution-driven simulation on Simics. This fast simulation speed allows us to exploit bigger design space and to quickly grasp the general behavior; and thus permits more accurate simulations to limited interesting design points. The collected memory traces are verified against the VTune™ and Perfmon measurement results. Two important parameters, the total number of memory references and the memory usage are compared. Due to the diversity between the measurement and the simulated environments, we pay more attention to the relative behavior among sequential and parallel versions of the applications. A closely match indicates that the memory traces are representative to the real hardware traces.

Trace-driven simulations: To compare memory reference behavior of single and multiple threads, the traces from various parallelization configurations are used for off-line cache simulations. In this step, a functional cache model is developed. Several parameters, such as block reuse distances, memory footprint size, cache hit/miss ratios, etc. are collected and compared. The block reuse distance between consecutive references to the same block provides the key insight in comparison of memory reference behavior. The hit/miss ratios of various cache sizes and set-associativities with different degrees of parallelization summarize the performance comparison results.

3. Performance Evaluation

Memory performance comparisons between single- and multi-threaded versions of AEC and SL are given in this section. We will first show the parallelization and measurement results that will be followed by the reuse distances and cache hit/miss results. It is interesting to observe that the memory reuse distance for the SL goes down with the degree of parallelization, while the distance for the AEC goes up with the degree of parallelization.

3.1. Parallelization and Measurement

Both AEC and SL are computational-intensive and demand large memory space. The hard real-time requirement of AEC and the stringent search requirement of SL make parallelization of these applications a viable approach on current Pentium® 4 systems to achieve faster execution time. AEC employs the algorithm of frequency-domain RLS (Recursive Least Square) [23]. The AEC's computational complexity is $O(kn\log n)$, where k is the number of channels and n is the block size. We select a large block size (1024 sample points per sample unit) and multiple channels (12 microphone

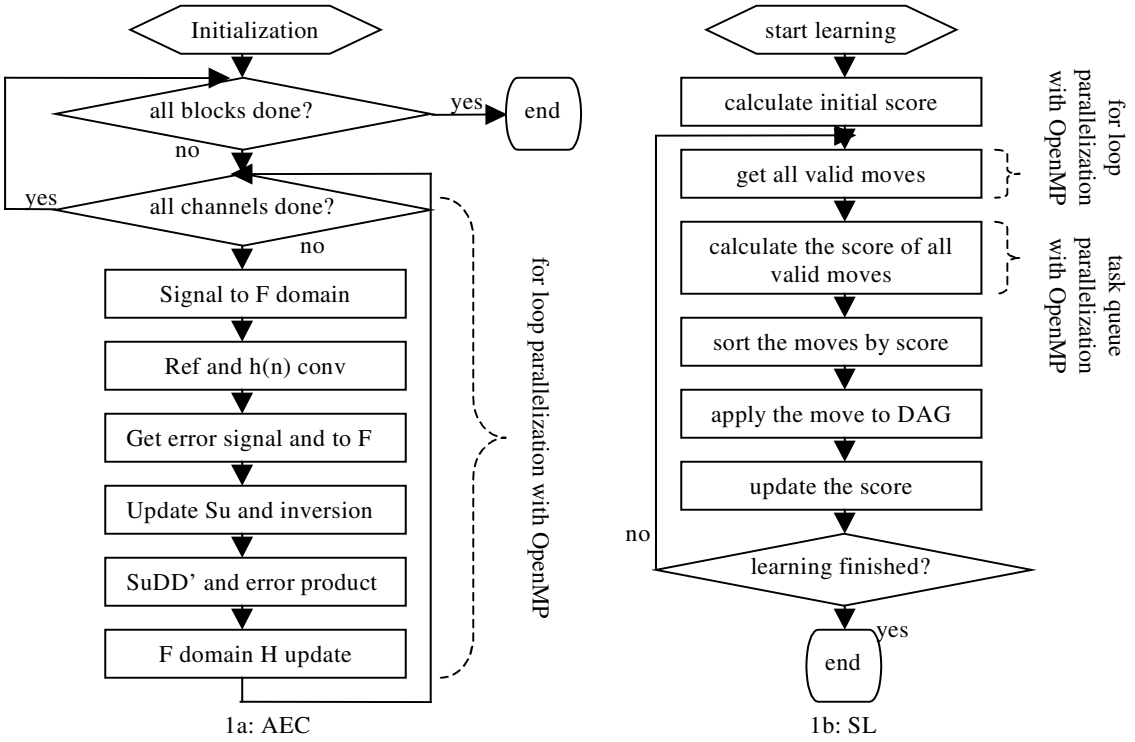

Figure 1. AEC and SL Flowchart and Parallelization Regions

arrays and 2 reference channels) to maintain high echo cancellation quality. With the audio sampling frequency 192KHz that is necessary for digital audio editing, AEC must finish processing of each block within 5.3ms for maintaining the hard real-time requirement. Our experiment shows that with 1K block size, 12 input and 2 reference channels, and with the constraint that CPU utilization ratio must be no more than 10% (because AEC processing is often used as a preprocessing component for other applications or in the driver), the hard real-time requirement cannot be met on a 3.0GHz Pentium® 4 CPU with dual-channel DDR400 memory. SL uses greedy hill-climbing algorithm to search state space of a Bayes Network. The SL's computational complexity is $O(kn^2)$, where k is the number of iteration samples and n is the number of nodes in a Bayes Network. Our experiments use only 37 nodes in a Bayes Network with 3000 samples. The running time on a 3.0GHz Pentium® 4 CPU with dual channel DDR400 platform is 3.4 seconds. For practical problem solving, the number of nodes and samples can be increased tremendously.

Figure 1 shows the flowchart of AEC and SL processing. We highlight regions where parallelization is adopted. These regions are identified as execution hotspots with Intel VTune™, and are worth efforts of parallelization. The main parallelization for both workloads is done on *for-loops*. Every channel is processed as an iteration of the *for-loops*. For SL, besides *for-loops* parallelization, a task-queue based parallelization is also applied for the region of calculating the score for all valid moves. The first parallel region hitting thread enqueues parameters needed by all worker threads, and then other worker threads fetch parameters in stack and begin execution.

The speedups, the total memory references, and the total memory usages for AEC and SL with the chosen parameters are plotted in Figure 2. The sequential programs as well as the parallel versions with 1 to 4 workers are executed on a quad Pentium® 4 Xeon™ MP system for collecting the statistics. We can make several observations.

1. Both workloads show performance improvement up to 4 workers. For SL, part of data is shared by all the threads. Simultaneous accesses to the shard data incur large amount of inter-processor traffic (e.g. MESI-coherence invalidations). For Quad Pentium® 4 Xeon™ MP system, the 4 processors are sharing one front-side bus (FSB), so the large MESI coherence traffic saturates FSB, and thereby limiting its speedup. For AEC, each channel processing is

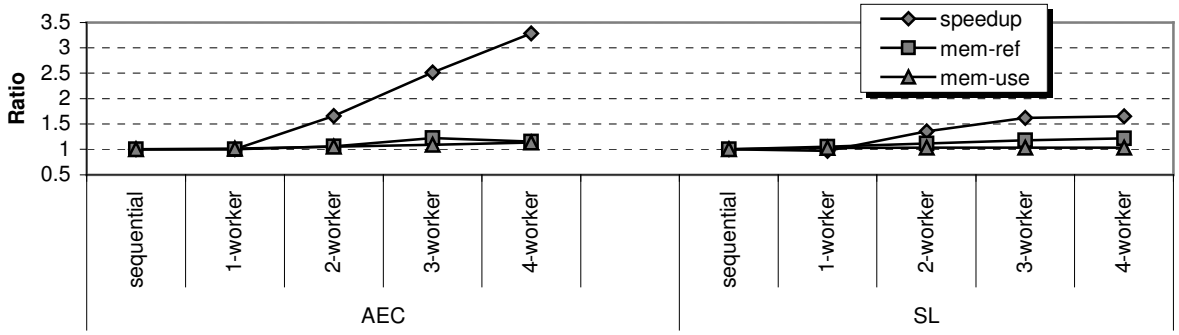

Figure 2. Measurement Results of Sequential and Parallel AEC and SL

executed in parallel with little cache interference. The access of the input channel data is initiated at the program initiation phase and memory bandwidth requirement is not a problem. This is the main reason that AEC displays much better speedup than SL. Figure 3 shows the FSB activity of SL and AEC on Quad Pentium® 4 Xeon™ MP system without using the Hyper-Threading Technology. Note that cache coherence traffic can be eliminated if the parallelized SL is running on a single multithreaded processor with shared caches. Instead of 1.3x speedup of 2-worker SL on quad Pentium® 4 without Hyper-Threading Technology, our experiment with 2-worker SL on a single Pentium® 4 system with Hyper-Threading Technology has shown 1.6x speedup. This is because SL's shared data are maintained as one copy in Hyper-Threading's shared cache and updates the shared data do not generate extra front-side bus traffic.

2. The parallel versions of both programs show

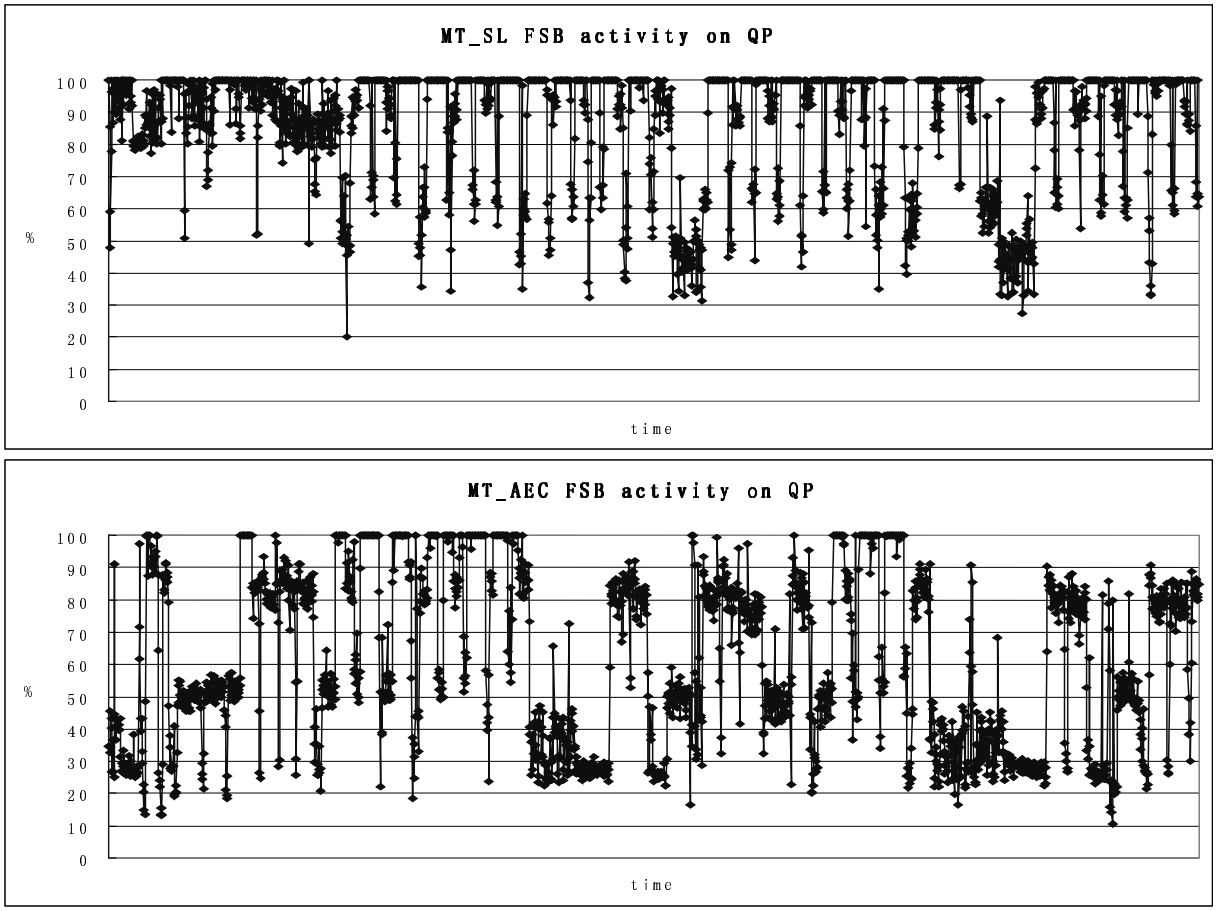

Figure 3. MT_SL and MT_AEC's FSB behavior

moderate overheads in terms of the total number of memory references. This is because of extra data structures such as the thread queues and OpenMP aware thread control blocks. Besides, worker threads need to replicate certain data structures, operate on the data, and copy them back and forth from/to the shared data area.

3. The total memory usage increases slightly with parallel versions. The memory usage goes up with the number of worker thread because worker threads have their own replicated private data set. OpenMP clauses, e.g. "reduction" and "captureprivate" also require worker threads to operate on temporary variables from the OpenMP stack.

3.2. Memory Trace Collection and Verification

Memory reference traces collected from execution-driven simulations of sequential and parallel versions of AEC and SL on Simics are used to drive off-line cache models. The Simics virtual machine is configured to the same as that of the Pentium® 4 used for measurement. The applications running on both environments are carefully tuned to start and end at identical points. A generic L_1/L_2 cache model is integrated with Simics simulations for approximating the correct sequence of memory references. We simulate a direct-mapped, I/D split, 8KB L_1 with 2-cycle access, and a combined, 8-way, 512KB L_2 with 7-cycle access. L_2 misses are charged for 200 cycles. L_1 miss penalty is processor-design specific, and we measured L_2 miss penalty with Vtune. Note that in both environments, only the user threads are counted. The system threads represent less than 5% of the overall memory references.

In Figures 4 and 5, the simulated results show that both the total memory references and the memory usage increase slightly with the degree of parallelization similar to that from the measurement results.

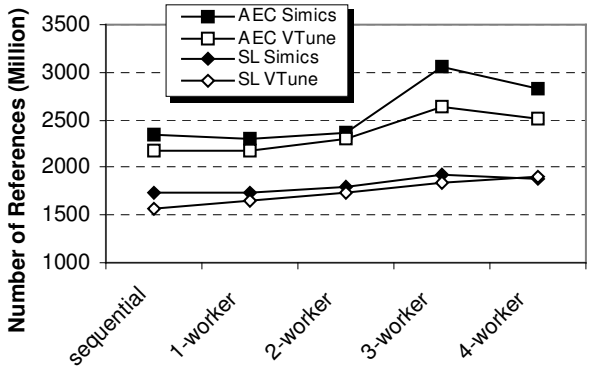

Figure 4. Number of Memory References Comparison Between VTune™ and Simics

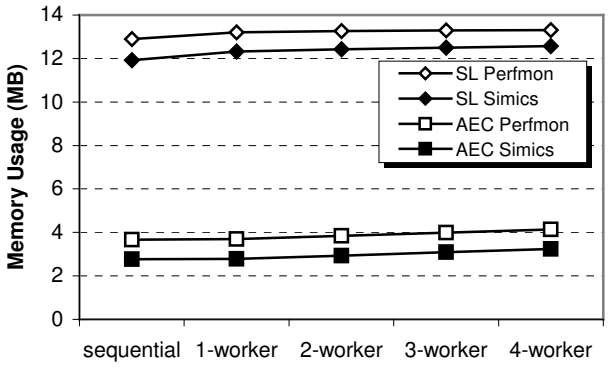

Figure 5. Memory Usage Comparison Between Perfmon and Simics

(1) For memory references, VTune™ counts the total committed loads and stores, which is slightly less than the memory references from the Simics reference traces. Through detailed analysis, we found that Simics may increase the total references by repeating a cache miss at a second time. As a result, the collected trace (unfiltered) could exhibit a slightly better cache behavior.

(2) For memory usages, the hardware counters in VTune™ cannot record the memory footprint for each process. For comparison purposes, we use another measurement tool, Perfmon on Windows that counts the number of page faults for each process. We use the measured total page faults to estimate the total memory usage. As shown in Figure 4, the Perfmon results of memory usages are consistently larger than that from the trace results (i.e., from virtual address traces simulated with 4KB page size). There are two reasons for this discrepancy. First, Perfmon measures the total page faults including instruction references while the collected traces only have data references. Second, each virtual page could potentially cause more than one page fault. Therefore, we conclude that the collected traces should reasonably reflect the real hardware behavior.

3.3 Memory Behavior Studies

The collected memory traces are post-processed to insert the reuse distance information to each memory request. Table 1 summarizes the distribution of reuse distances for sequential and parallel versions of AEC and SL. Note that we are focusing on the impact of memory references on multithreaded processors where multiple threads are likely to share processor caches. Therefore, the reuse distance is measured among all the threads. Note also that the reuse distance reflects the impact of both constructive/disruptive memory reference behavior

Table 1. The Reuse Distance Distribution of AEC and SL

AEC	Sequential	1-worker	2-worker	3-worker	4-worker
0 ~ 999	0.8506	0.8498	0.8453	0.8417	0.7876
1000 ~ 9999	0.0824	0.0828	0.0845	0.0848	0.1320
10000 ~ 19999	0.0161	0.0162	0.0123	0.0147	0.0180
> 20000	0.0510	0.0512	0.0580	0.0588	0.0624
SL	Sequential	1-worker	2-worker	3-worker	4-worker
0 ~ 999	0.9289	0.9295	0.9359	0.9354	0.9342
1000 ~ 9999	0.0027	0.0054	0.0096	0.0125	0.0142
10000 ~ 19999	0.0005	0.0005	0.0010	0.0011	0.0016
> 20000	0.0678	0.0645	0.0535	0.0510	0.0500

and changes of memory usage among sequential and various parallel versions of the program.

The results show opposite memory reference behavior of AEC and SL with respect to the parallelization. For AEC, the short reuse distance (<1000, that can be fitted in a 64KB cache) decreases with the number of workers while the long reuse distance (>20000) goes up with the number of workers. SL, on the other hand, has more shorter reuse distances (<20000) with higher degrees of parallelization. Careful examining the source codes, we discover the key difference between the two workloads. In AEC, each thread's working set is more independent due to replications of a few data structures. Since a significant portion of the memory references targets private data, larger reuse distances are observed when mixing references among multiple threads. In SL, a majority of memory accesses from each thread target the shared data. By interlaced executions of threads in the thread queue causes data to be accessed more closely than that in the single-thread SL.

The hit ratios of various shared cache sizes of the sequential and parallel versions of AEC and SL are plotted in Figures 6 and 7 respectively. We can clearly see that for AEC, the 4-worker version has the lowest hit ratios. The hit ratio increases with decreasing degree of parallelization. The low hit ratio may increase memory

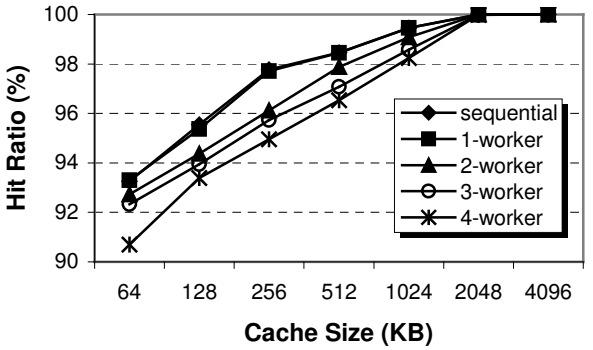

Figure 6. Hit Ratios of Single and Multiple Threaded AEC

stalls significantly and outweigh the benefit of multithreading. Recall that the main reason for this memory performance degradation is due to accessing independent data in each thread. These largely independent working sets compete with limited cache space. Misses are almost gone with a 1MB cache, in which the entire working set can be held. The original sequential program and the parallelized program with a single worker have almost identical memory behavior indicating a similar memory behavior due to the extra code from OpenMP parallelization.

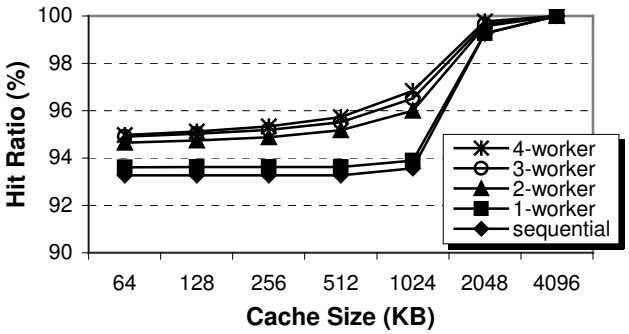

Figure 7. Hit Ratios of Single and Multiple Threaded SL

SL shows considerable better memory behavior with higher degrees of parallelization of 2 or more workers. This is due to the fact that data sharing among multiple threads makes it possible to prefetch shared data for one another. When the cache is smaller than 1MB, the hit ratios are almost the same for all configurations. Careful analysis of the SL source code discloses that SL operates a data block of 64KB for a period before switching to another block. The next data block is hard to predict. Misses usually occur between block switching unless cache size is big enough (2MB) to hold the entire working set. The small gap between the original sequential program and the parallelized program with a single worker indicates a slightly better reference locality due to the extra code from OpenMP parallelization.

4. Conclusion

As multithreading technology becomes popular in today's microprocessors, this paper studies memory behaviors of multithreaded multimedia and AI applications. A methodology including workload selection and parallelization, benchmarking applications on Pentium® 4 systems for performance measurement using VTune™ and Perfmon, memory reference traces collection and verification using Simics, and trace-driven simulation and analysis, has described. The performance evaluation indicates a constructive and a disruptive memory behavior can both be a result from parallelization of the selected applications. Care must be taken to avoid disruptive memory reference behavior. Our reuse-distance analysis not only can help understanding constructiveness/destructiveness behaviors of the multithreaded workloads, but can also feedback to the programmer for improving program parallelization, and/or to the designer for enhancing future memory hierarchy designs.

5. Acknowledgement

This work is supported in part by an NSF grant EIA-0073473 and by research donations from Architecture Research Lab and China Research Center of Intel Corp. Anonymous referees provide helpful comments.

6. Reference

1. Intel Corp., Intel® Architecture MMX™ Technology Developer's Manual, IL: Intel Corporation, Order Number 243006, 1996.
2. Intel Corp., Intel® Architecture MMX™ Technology Programmer's Reference Manual, IL: Intel Corporation, Order Number 243007.
3. D. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "HyperThreading Technology Microarchitecture and Performance," *Intel Technology Journal,* Q1 2002.
4. J. Song, J. Li, and Y.-K. Chen, "Quality-Delay-and-Computation Trade-Off Analysis of Acoustic Echo Cancellation On General-Purpose CPU," *Proc. of Int'l Conf. on Acoustics, Speech, and Signal Processing,* vol. 2, pp. 593-596, Apr. 2003.
5. D. Heckerman, "A tutorial on learning with Bayesian networks", *Microsoft Research tech. report,* MSR-TR-95-06, 1996.
6. OpenMP forum, http://www.openmp.org
7. X. Tian, Y.-K. Chen, M. Girkar, S. Ge, R. Lienhart, and S. Shah, "Exploring the Use of HyperThreading Technology for Multimedia Applications with Intel OpenMP Compiler," *Proc. of Int'l Parallel and Distributed Processing Symp.,* Apr. 2003.
8. Y.-K. Chen, R. Lienhart, E. Debes, M. Holliman, and M. Yeung, "The Impact of SMT/SMP Designs on Multimedia Software Engineering---A Workload Analysis Study," *Proc. of Int'l Symp. on Multimedia Software Engineering,* Dec. 2002.
9. L.A. Barroso, K. Gharachorloo, and E. Bugnion. Memory System Characterization of Commercial Workloads. *Proc of. ISCA 25*, pages 3–14, June 1998.
10. R. Hankins, T. Diep, M. Annavaram, B. Hirano, H. Eri, H. Nueckel, J. P. Shen, Scaling and Charact rizing Database Workloads: Bridging the Gap between Research and Practice, *Proc. of MICRO 36,* pages 151-164, December, 2003.
11. S. Sohoni, R. Min, Z. Xu, Y. Hu, A study of memory system performance of multimedia applications, *Proc. of SIGMETRICS 2001,* pages 206-215, June, 2001.
12. M. Karlsson, K. E. Moore, E. Hagersten, and D. A.Wood. Memory System Behavior of Java-Based Middleware, *Proc. of HPCA 9*, pages 217-228, February 2003.
13. Y. Luo and L. K. John. Workload Characterization of Multithreaded Java Servers. *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software*, 2001.
14. S. G. Abraham and D. E. Hudak. "Compile-time partitioning of iterative parallel loops to reduce cache coherency traffic". volume 2, pages 318–328, Jul. 1991.
15. K. Kennedy and K. S. McKinley. "Optimizing for parallelism and data locality. *Proc. of ACM International Conference on Supercomputing*, Jul. 1992.
16. G. Narlikar. "Scheduling threads for low space requirement and good locality". *Proc. of ACM Symposium on Parallel Algorithms and Architectures (SPAA),* Jun. 1999.
17. M. Steckermeier and F. Bellosa. „Using locality information in user level scheduling". *Technical Report TR-95-14,* University Erlangen-Nurnberg, 1995.
18. B. Weissman. "Performance counters and state sharing annotations: a unified approach to thread locality*". Proc. of ASPLOS-VIII,* Oct. 1998.
19. P.S. Magnusson, etc. "Simics: A Full System Simulation Platform*", IEEE Computer,* Feb. 2002, pp50-58.
20. Intel C/C++ Compiler for Windows/Intel platforms, http://www.intel.com/software/products
21. http://www.intel.com/software/products/vtune/techtopic/advantage_vtune.pdf
22. Perfmon,http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/winxppro/proddocs/NT_Command_Perfmon.asp
23. M. Sondhi and D. R. Morgan, "Acoustic Echo Cancellation for Stereophonic Teleconferencing", *Proc.of IEEE ASSP Workshop Appls. Signal Processing Audio Acoustics,* 1991.