Cache Performance for Multimedia Applications

Nathan T. Slingerland
Apple Computer
5 Infinite Loop, MS 305-2AP
Cupertino, CA 95014 USA
nslingerland@apple.com

Alan Jay Smith
Computer Science Devision
University of California
Berkeley, CA 94720 USA
smith@cs.berkeley.edu

ABSTRACT

The caching behavior of multimedia applications has been described as having high instruction reference locality within small loops, very large working sets, and poor data cache performance due to non-locality of data references. Despite this, there is no published research deriving or measuring these qualities. Utilizing the previously developed Berkeley Multimedia Workload, we present the results of execution driven cache simulations with the goal of aiding future media processing architecture design. Our analysis examines the differences between multimedia and traditional applications in cache behavior. We find that multimedia applications actually exhibit lower instruction miss ratios and comparable data miss ratios when contrasted with other widely studied workloads. In addition, we find that longer data cache line sizes than are currently used would benefit multimedia processing.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems - *Design Studies, Performance Attributes*; B.3 [**Hardware**]: Memory Structures; B.3.2 [**Memory Structures**]: Design Styles - *Cache Memories*; I.6 [**Computing Methodologies**]: Simulation and Modeling

General Terms

measurement, performance, design

Keywords

multimedia, cache, CPU caches, simulation, trace driven simulation

1. INTRODUCTION

Multimedia is an amalgamation of various data types such as audio, 2D and 3D graphics, animation, images and video within a computing system or within a user application [4]. Put simply, a *multimedia application* is one which operates on data to be presented visually or aurally. The purpose of this work is to explore the cache behavior of real world multimedia applications. An important motivation is the widespread belief (seemingly without any actual basis in research) that data caches are not useful for multimedia applications because of the streaming nature of the data upon which they operate [9], [13], [21], [22], [25]. The results presented in this paper strongly suggest that contemporary media processing applications perform no worse than traditional integer and floating point workloads.

Further motivating our study is the large role memory latency plays in limiting performance. Consider Table 1, which compares the performance with caching against the same system with all cache levels (L1 and L2) disabled. This was done by setting the appropriate BIOS parameters on our test system at boot time and then measuring the performance on real hardware. From this experiment we can see how highly dependent modern microprocessor performance is on an efficient memory hierarchy. The difference in latency between levels of contemporary memory hierarchies is substantial, explaining the enormous slowdown we observe when the caches are disabled on our test system. Note that the system time (time spent in the operating system) slowdown is considerably less than that of the user time. This corroborates the generally held belief that the memory locality within operating system code is very poor, as it exhibits less of a performance degradation when caching is disabled.

2. RELATED WORK

There have been a limited number of multimedia caching studies. In [34] the data cache behavior of MPEG-2 video decoding is studied with the goal of optimizing playback performance through the cache sensitive handling of the data types used. It was found that although it has been suggested that caches are critically inefficient for video data (several media processor chips dispense with data caches entirely), there was sufficient reuse of values for caching to significantly reduce the raw required memory bandwidth. [17], [10], and [39] study the usefulness of caching the textures used in 3D rendering. A texture cache with a capacity as small as 16 KB has

Funding for this research has been provided by the State of California under the MICRO program, and by Cisco Corporation, Fujitsu Microelectronics, IBM, Intel Corporation, Maxtor Corporation, Microsoft Corporation, Sun Microsystems, Toshiba Corporation and Veritas Software Corporation.

Table 1: **Uncached Performance Slowdown Factor** - ($t_{uncached}/t_{cached}$) when L1 and L2 caches were disabled on a 500 MHz AMD Athlon, 64 Kbyte L1 data cache, 64 Kbyte L1 instruction cache, 512 Kbyte L2 unified cache, 64-byte line size.

Name	User Time Ratio	Sys Time Ratio
ADPCM Encode	25.0	3.6
ADPCM Decode	32.9	11.3
DJVU Encode	56.7	3.6
DJVU Decode	61.0	16.8
Doom	53.0	1.4
Ghostscript	63.7	34.7
GSM Encode	61.3	6.7
GSM Decode	77.8	16.5
JPEG Encode	103.4	1.3
JPEG Decode	103.0	10.5
LAME	80.3	1.4
Mesa Gears	44.2	11.0
Mesa Morph3D	35.9	35.0
Mesa Reflect	77.4	17.5
MPEG-2 Encode DVD	86.3	2.3
MPEG-2 Encode 720P	82.9	1.4
MPEG-2 Encode 1080I	86.5	1.5
MPEG-2 Decode DVD	94.1	9.3
MPEG-2 Decode 720P	95.1	5.9
MPEG-2 Decode 1080I	91.9	10.4
mpg123	83.7	7.5
POVray3	74.5	16.0
Rasta	83.8	7.0
Rsynth	86.5	27.0
Timidity	73.9	20.3
Arithmetic Mean	72.6	11.2
Geometric Mean	68.6	7.1

been found to reduce the required memory bandwidth three to fifteen times over a non-cached design and exhibit miss ratios around 1% [17]. The addition of a larger second level of texture cache (2 MB) to a small first level cache (2 KB) can reduce the memory bandwidth from 475 MB/s to around 92 MB/s [10].

There have been several studies of prefetching for multimedia. [41] examines different hardware data prefetching techniques for MPEG-1 (encoding and decoding) and MPEG-2 (decoding). Three hardware prefetching techniques were considered, with the most successful found to reduce the miss count by 70% to 90%. [35] presents a combined hardware/software solution to prefetching for multimedia. Based on cycle accurate simulation of the Trimedia VLIW processor running a highly optimized video de-interlacing application, it was found that such a prefetching scheme was able to eliminate most data cache misses, with the effectiveness dependent on the timing parameters involved. [11] suggests a two-dimensional prefetching strategy for image data, due to the two separate degrees of spatial locality inherent in image processing (horizontal and vertical). When their 2D prefetching technique was applied to MPEG-2 decoding as well as two imaging applications (convolution and edge tracing), 2D prefetch was found to reduce the miss ratio more than one block look-ahead. Hardware implementation aspects of prefetching are discussed in [37].

3. WORKLOADS

3.1 Berkeley Multimedia Workload

For our study of the cache behavior of multimedia applications, we employ the Berkeley Multimedia Workload, which we develop and characterize in [28]. A description of the component applications and data sets is given in Table 2. The main driving force behind application selection was to strive for completeness in covering as many types of media processing as possible. Open source software was used both for its portability (allowing for cross platform comparisons) as well as the fact that we could directly examine the source code.

The Berkeley workload represents the domains of *3D graphics* (Doom, Mesa, POVray), *document and image rendering* (Ghostscript, DjVu, JPEG), *broadband audio* (ADPCM, LAME, mpg123, Timidity), *speech* (Rsynth, GSM, Rasta) and *video* (MPEG-2). Three MPEG-2 data sets are included to cover Digital Video Disc (DVD) and High Definition Television or HDTV (720P, 1080I) resolutions. The parameters of the DVD, and HDTV data sets are listed in Table 3. "Frames" is the number of frames in the data set.

Table 3. **HDTV Data Set Parameters**

Format	Aspect	Horizontal	Vertical	Frames
DVD	4:3	720	480	16
HDTV 720P	16:9	1280	720	16
HDTV 1080I	16:9	1920	1080	16

3.2 Other Workloads

For comparison purposes, we have included the results of several previous studies of the cache behavior of more traditional workloads.

3.2.1 SPEC92/SPEC95

SPEC CPU benchmarks are taken to be generally representative of traditional workstation applications, with the integer component reflecting system or commercial applications, and the floating point component representing numeric and scientific applications. In [16] Gee analyzed the cache behavior of the SPEC92 benchmark suite running on DECstations with MIPS R2000 or R3000 processors and version 4.1 of the DEC Ultrix operating system. Because the SPEC benchmarks are typically run in a uniprogrammed environment, no cache flushing or other method was used to simulate multiprogramming. Gee also found that for the SPEC92 benchmark suite, system time is insignificant compared to user time, and so operating system memory behavior was unimportant for that study.

SPEC95 is an upgraded version of the SPEC92 benchmark suite. It consists of eight integer intensive and ten floating-point intensive applications, several of which are shared with SPEC92. In general, the applications were designed to have larger code size and greater memory activity than those of SPEC92.

Table 2. **Berkeley Multimedia Workload**

Name	Description	Data Set
ADPCM	IMA ADPCM audio compression	Excerpt from Shchedrin's Carmen Suite, 28 sec., Mono, 16-bits, 44 kHz
DjVu	AT&T IW44 wavelet image compression	491x726 color digital photographic image
Doom	Classic first person shooter video game	25.8 sec. recorded game sequence (774 frames @ 30 fps)
Ghostscript	Postscript document viewing/rendering	First page of Rosenblum and Ousterhout's LFS paper (24.8 KB)
GSM	European GSM 06.10 speech compression	Speech by U.S. Vice President Gore, 24 sec., Mono, 16-bits, 8 kHz
JPEG	DCT based lossy image compression	491x726 color digital photographic image
LAME	MPEG-1 Layer III (MP3) audio encoder	Excerpt from Shchedrin's Carmen Suite, 28 sec., Stereo, 16-bits, 44 kHz
Mesa	OpenGL 3D rendering API clone	Animated gears, morph3d, reflect demos - 30 frames each at 1024x768
MPEG-2	MPEG-2 video encoding	16 frames (1 GOP) at DVD, HDTV 720P, HDTV 1080I resolutions
mpg123	MPEG-1 Layer III (MP3) audio decoder	Excerpt from Shchedrin's Carmen Suite, 28 sec., Stereo, 16-bits, 44 kHz
POVray	Persistance of Vision ray tracer	640x480 Ammonite scene by artist Robert A. Mickelson
Rasta	Speech recognition	2.128 sec. SPHERE audio file: "Laurie?...Yeah...Oh."
Rsynth	Klatt speech synthesizer	181 word excerpt of U.S. Declaration of Independence (90 sec., 1,062 bytes)
Timidity	MIDI music rendering with GUS instruments	X-files theme song, MIDI file (49 sec., 13,894 bytes), Goemon patch kit

3.2.2 Multiprogramming Workload (Mult)

The authors of [5] generated miss ratios for very long address traces (up to 12 billion memory references in length) on the Titan RISC architecture in order to evaluate the performance of a variety of cache designs. Three individual traces were used in addition to another which was a multiprogrammed workload consisting of several jobs. Our comparison includes their miss ratio results for their 7.6 billion reference (68.5% instruction, 30.6% load, 15.4% store) multiprogramming workload (referred to as "Mult" by the authors of [5]).

3.2.3 Design Target Miss Ratios (DTMR)

[31] introduced the concept of *design target miss ratios* (DTMRs), intended to represent typical levels of performance across a wide class of workloads and machines, to be used for hardware design. The DTMRs were synthesized from real (hardware monitor) measurements that existed in the literature and from trace driven simulations using a large number of traces taken from several architectures, and originally coded in several different languages.

3.2.4 VAX 11/780, VAX 8800

Two studies done at Digital Equipment Corporation (DEC) supply miss ratios for a time-shared engineering workload taken with a hardware monitor on VAX 11/780 and VAX 8800 machines [7], [8]. The 11/780 has an 8-KB, write through, unified cache with an 8-byte block size and associativity of two. The 8800 has a 64-KB, write-through, direct mapped, unified cache with a 64-byte block size. On the VAX 11/780 it is possible to disable half of the two-way associative cache through special control bits; a technique which allowed for the measurement of a 4-KB, direct mapped, unified cache configuration as well.

3.2.5 Agarwall Mul3

In [1] an analysis of the effect of operating system references and multiprogramming was presented for a workload of eleven application programs (30 traces in all). The platform used to gather the traces was a VAX 11/780 running either the Ultrix or VMS operating system. All of the traces were gathered through the ATUM scheme of microcode modification, and were roughly 400,000 references long (approximately one half second of execution time). A technique termed *trace sampling* was used to concatenate smaller traces to better simulate the full trace length of a running program. We utilize their three way multiprogrammed workload for comparison.

3.2.6 Amdahl 470

In [30], hardware monitor measurements taken at Amdahl Corporation on Amdahl 470V machines are presented. A standard internal benchmark was run containing supervisor, commercial and scientific code. Supervisor state miss ratios were found to be much higher than problem state miss ratios.

4. METHODOLOGY

In order to measure cache miss ratios, we modified the LibCheetah v2.1 implementation [3] of the trace driven Cheetah cache simulator [36] to operate in an execution driven mode. It was also extended to allow for traces longer than 2^{31} references long. Cheetah simultaneously evaluates many alternative uniprocessor caches, but restricts the design options that can be varied. For each pass through an address trace, all of the caches evaluated must have the same block size, do no prefetching, and use the LRU or MIN replacement algorithms. Other cache simulators were also considered for this study (TychoII [40], Dinero IV [14]), but were found to be considerably slower than Cheetah or otherwise unsuitable for use in execution driven simulation due to dynamic memory allocation issues. DEC's ATOM [12] was used to instrument target applications with the modified Cheetah simulator, allowing for execution driven cache simulation. See [38] and [33] for overviews of trace driven simulation in general, and [27] for a comparison of the performance of a variety of execution and trace driven solutions.

4.1 Trace Length

Many cache studies utilize trace lengths that are a fraction of an application's total run time due the enormous simulation times re-

Table 4: **Berkeley Multimedia Workload Simulation Characteristics** - *Purge interval* is the number of instructions executed in each context interval before flushing the simulated cache. *Data time* (inherent time represented by data set - machine independent), *user time* (time spent processing in user space - machine dependent), and *system time* (time spent processing in system space on behalf of an application - machine dependent) are given in seconds. *Resident Set* is the maximum number of kilobytes in memory active at any one time, as determined by the `getrusage()` system call. All measurements were done on a DEC Alpha DS20 workstation with dual 500 MHz Alpha 21264 processors and 2048 MB of RAM running Compaq Tru64 Unix v5.0A (Rev. 1094). All applications were compiled with GCC v2.8.1 except (*) compiled with DEC C v5.6-075.

Name	**Instruction References**	**Load References**	**Store References**	**Purge Interval**	**Data Time**	**User Time**	**System Time**	**Resident Set (kB)**
ADPCM Enc.	64,020,339	4,302,782	616,116	708,037	27.818	0.102	0.036	1,472
ADPCM Dec.	49,687,192	4,302,782	1,229,491	708,037	27.818	0.054	0.067	1,472
DJVU Enc.	394,242,073	68,204,647	27,458,767	4,754,521	-	0.700	0.033	41,664
DJVU Dec.	328,761,829	59,700,283	31,845,270	4,754,521	-	0.484	0.037	20,992
Doom	1,889,897,116	500,225,773	109,222,846	4,284,671	25.800	2.216	0.939	26,432
Ghostscript*	970,395,449	188,116,952	96,837,718	1,227,194	-	1.190	0.164	32,192
GSM Enc.	375,971,389	55,009,077	14,010,892	297,641	24.341	0.468	0.016	1,024
GSM Dec.	126,489,950	10,711,683	3,812,483	297,641	24.341	0.209	0.014	1,024
JPEG Enc.	177,977,854	41,182,069	14,156,413	3,821,284	-	0.223	0.006	10,880
JPEG Dec.	80,176,365	16,419,065	4,585,079	3,821,284	-	0.093	0.024	10,880
LAME*	7,989,818,554	1,688,230,256	720,826,607	3,358,692	27.818	18.543	0.075	7,104
Mesa Gears*	296,287,705	36,839,087	38,449,257	2,173,610	1.000	0.484	0.039	50,240
Mesa Morph3D*	239,456,087	28,181,931	42,865,365	2,173,610	1.000	0.467	0.050	50,432
Mesa Reflect*	2,752,665,912	431,196,702	221,523,544	2,173,610	1.000	3.672	0.051	59,968
MPEG2 Enc. DVD	17,986,999,069	3,257,725,765	554,222,287	5,339,432	0.533	17.896	0.199	48,128
MPEG2 Enc. 720P	47,606,551,352	8,581,717,942	1,563,082,541	5,339,432	0.533	48.263	0.505	124,032
MPEG2 Enc. 1080I	111,041,463,652	20,148,301,625	3,349,482,784	5,339,432	0.533	113.050	0.521	277,952
MPEG2 Dec. DVD	1,307,000,398	219,595,775	76,688,056	1,055,372	0.533	1.911	0.051	16,512
MPEG2 Dec. 720P	3,992,213,571	673,343,544	243,881,680	1,055,372	0.533	5.796	0.141	41,472
MPEG2 Dec. 1080I	8,038,214,930	1,341,912,185	464,649,094	1,055,372	0.533	12.098	0.182	91,648
mpg123*	574,034,774	166,675,525	45,334,678	1,554,505	27.818	0.735	0.015	3,328
POVray3	6,017,197,975	1,562,189,592	683,690,648	5,928,433	0.033	11.296	0.121	16,000
Rasta*	25,120,492	5,925,648	1,989,604	2,560,537	2.128	0.039	0.014	5,632
Rsynth	402,500,964	102,351,142	39,223,906	594,438	99.680	0.780	0.004	7,808
Timidity	4,588,632,916	1,340,471,112	594,047,710	3,675,086	47.440	2.036	0.104	25,664
Total	217,315,777,907	40,532,832,944	8,943,732,836	-	-	242.805	3.406	-
Arithmetic Mean	8,692,631,116	1,621,313,318	357,749,313	-	-	9.712	0.136	-

quired to account for every instruction and data cache reference. Unfortunately, short trace lengths are problematic because programs exhibit phase behavior; an effect which is easily seen in Figure 1. The graphs depict the number of cache misses per 1,000,000 instructions executed for two sample applications.

In order to be able to simulate the effects of a program's behavior, it is necessary to have a trace which captures all of its behavior. We found that although there are some applications (notably many of the SPEC92/95 benchmarks) that exhibit uniform cache behavior over their entire run times, our multimedia workload applications did not share this property. The result of this is that full applications traces are the only way to completely characterize average cache behavior.

A second difficulty with short trace lengths specific to cache simulations is the *cold start* problem. Cache simulation programs typically start with an empty cache which becomes filled as the simulation progresses. All initial memory accesses will miss the cache (*compulsory* misses), so cold start effects can potentially dominate if traces are too short to mitigate these effects. Traces of a billion or more references may be needed to fully initialize multi-megabyte cache configurations [20]. Our work traces application programs with realistic data sets for full execution runs. The trace lengths for the component Berkeley Multimedia Workload applications are given in Table 4.

Table 4 lists the amount of time the CPU spends either in user space (*user time*) doing actual work for the application, or in system space (*system time*) serving I/O requests and dealing with other overhead on behalf of the application. Both user time and system time are machine dependent, and vary based on the instruction set, clock cycle length and other architectural parameters. *Data time* is machine independent, and is the inherent time length of the data set. For example, 24 frames of a DVD movie might represent one second of data time, even though decoding requires only 0.5 seconds of computation (the sum of system and user time).

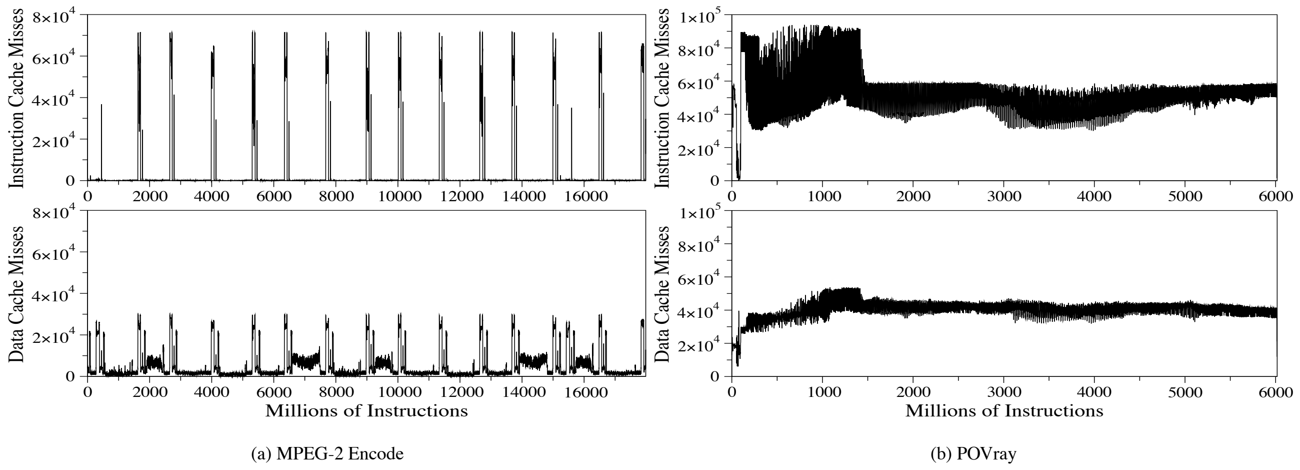


Figure 1: **Example Cache Miss Profiles** - data cache misses per million instructions executed (direct mapped, 8KB cache with 32B line size)

4.2 Operating System Behavior

In general, studies including operating system behavior are rare because of the difficulty involved in obtaining this information. User space is freely manipulated, but tracing system space usually requires that modifications be made to the operating system. Although our traces only include user state references, we can assume that this represents almost all of the memory behavior of the programs under study; less than 1% of our multimedia workload was system time. To some degree this may be an artifact of the nature of the Berkeley Multimedia Workload, which requires system time only for file I/O. In an actual multimedia application where data must be transferred to and from I/O devices such as network, disk, or sound and video controller cards, a larger amount of OS activity could be present.

4.3 Multiprogramming

Despite the fact that the Berkeley Multimedia Workload is dominated by user time computation, it is because of multiprogramming that we cannot entirely ignore operating system behavior. When a context switch occurs, the instructions and data of the newly scheduled process may no longer be in the cache from the last time it was run due to the memory use of programs scheduled in the interim. The number of cycles in this interval (limited by the *quantum*) affects the cache miss ratio. Although a quantum length that depends on clock time or external events remains constant with architectural change (typically 10 to 100 ms), the number of cycles Q in each quantum increases over time for various reasons, including less efficient software and a speedup of the processor relative to the speed of real time events.

4.3.1 Multimedia

Although the level of multiprogramming on a desktop workstation is typically low, multimedia applications are often multi-threaded. For example, in the case of on screen DVD movie play back, there are typically several concurrent threads of execution, each dealing with a particular aspect of MPEG-2 decoding (e.g. audio, video, bitstream parsing/demuxing). Acceptable playback requires that decoding be fast enough to leave time for computing the other components in that unit of time (otherwise video frames may need to be dropped) and to prevent latency effects from disrupting the perceived synchronization between audio and video. These requirement affect scheduling, and are not taken into account in an application which operates in a batch or offline mode.

The effect of multiprogramming can be roughly approximated by periodically flushing (clearing) a simulated cache. The context switch intervals of the actual applications from the Berkeley Multimedia Workload were not measured and used for this because they are primarily file based applications, typically converting between compressed and uncompressed format without presenting the resulting data to the user. So, although the algorithms they employ (and therefore their memory access patterns) should for the most part be similar to their "real world" counterparts, their scheduling behavior is vastly different. In order to correctly simulate the effect of multiprogramming for our multimedia workload, the average context switching interval for commercial (closed source) Microsoft Windows applications was measured on real hardware. The applications were chosen to correspond as closely as possible to those comprising the Berkeley Multimedia Workload, such that, for example, the context interval measured for actual DVD video playback was used in our simulations of MPEG-2 video decoding at DVD resolutions.

Microsoft Windows NT and Windows 2000 both maintain a large amount of performance information for a large number of system objects including context switch count, user time and system time per thread. By dividing the sum of system time and user time by the measured context switch count it was possible to compute the average context switch interval for each type (domain) of multimedia application. Context switch intervals were measured on a 500 MHz AMD Athlon with 256 MB of PC100 DRAM and an MSI MS-6167 motherboard running Windows 2000 Professional v5.00.2195. Both a sound card (Sound Blaster Live Value) and 3D

Application Name	Data Set	Context Interval
3D Flowerbox OpenGL Screen Saver *	1280x1024x32bpp, (1:00)	23,653
RealPlayer v7.0 RealAudio Player	KAMU 64Kbps, stereo, G2 stream, (5:00)	40,396
Real Jukebox v1.0.0.488 MP3 Player	Santana - *Smooth*, 160 Kbps, stereo (4:54)	58,399
MediaPlayer GSM 06.10 *	Speech by Al Gore, 8 kHz Mono, 16-bits (0:24)	297,641
K-Jofol 2000 MP3 Player v1.0	Santana - *Smooth*, 160 Kbps, stereo (4:54)	360,336
3D Pipes OpenGL Screen Saver *	1280x1024x32bpp, (1:00)	567,080
Narrator Text to Speech *	U.S. Declaration of Independence	594,438
MediaPlayer IMA ADPCM *	Santana - *Smooth*, 160 Kbps, stereo (4:54)	708,037
WinDVD v2.0 DVD Player	(5:00) clip from *Amadeus*	921,510
PowerDVD v2.55 DVD Player	(5:00) clip from *Amadeus*	1,189,234
Ghostscript PostScript Previewer	Rosenblum and Ousterhaut's LFS paper (15 pages)	1,227,194
Dragon Naturally Speaking Preferred	U.S. Declaration of Independence	2,560,537
Audio Catalyst v2.1 MP3 Encoder	Santana - *Smooth*, 44 kHz, stereo (4:54)	3,358,692
Audio Compositor MIDI Renderer	X-files theme song, Personal Copy v4.2 Sound Fonts	3,675,086
Irfanview v3.15 Image Viewer	Kodak's Iowa Corn jpeg image (2048x3072x24bpp)	3,821,284
Quake III Arena (Demo)	Internal demo #1, demo #2 (640x480)	4,284,671
DjVushop Document Compression	Scanned cover of March 2000 *IEEE Computer* journal	4,754,521
Avi2Mpg2 MPEG-2 Encoder	160 frames, 720x480 from *Amadeus*	5,339,432
POVray v3.1g Raytracer	Torus (internal demo scene), 800x600 Anti-Aliased	5,928,433
3D Maze OpenGL Screen Saver *	1280x1024x32bpp, (1:00)	5,930,096
	Arithmetic Mean	2,247,389
	Geometric Mean	1,015,426

Table 5: **Average Multimedia Context Switch Intervals** - Measurements are given in 500 MHz AMD Athlon clock cycles. (*) denotes applications packaged with the Windows 2000 operating system.

accelerator card (AGP Nvidia Riva TNT) were installed. Table 5 lists these intervals as measured by the Windows 2000 performance counters, which return results in terms of time (CPU cycles).

In our cache simulations, we simulate normal task switching by flushing the cache every $Q_{context}$ instructions. Because our cache simulation is instruction, rather than cycle based, we require cache purge intervals measured in terms of instructions executed between cache flushes. In order to convert our context switch interval data from cycles to instructions we need to know the corresponding cycles per instruction (CPI) ratio. However, we can not simply treat x86 CISC instructions as being equivalent to the RISC Alpha instructions of our simulation platform, due to the inherently different amounts of work done by each class of instructions. In order to approximate the equivalent number of Alpha RISC-like instructions in each context switch interval, we divide the number of x86 Athlon cycles by the typical number of cycles per micro-op (CPμOp) (the details of our CPI and CPμOp measurements are given in [29]).

Note that in a real system the interval between task switches is variable, not fixed; since we don't have the distribution of inter-interrupt times, we chose to use a constant interval. Alternately, we could have chosen some other distribution, such as exponential, normal or uniform. The simulation quanta (cache flush intervals) applied to each application are listed in Table 4.

4.3.2 SPEC95

SPEC95 was simulated without multiprogramming (cache flushing) for several reasons. First, it is normally run in a uniprogrammed mode in order to extract the highest benchmark performance [16].

CINT95	Context Interval
099.go	21,134,208
124.m88ksim	5,122,455
126.gcc	3,845,678
129.compress	22,719,364
130.li	21,754,551
132.ijpeg	16,093,926
134.perl	16,308,625
147.vortex	13,193,123
CFP95	**Context Interval**
101.tomcatv	10,185,364
102.swim	13,753,700
103.su2cor	9,595,431
104.hydro2d	18,624,108
107.mgrid	17,791,106
110.applu	4,644,660
125.turb3d	22,366,853
141.apsi	11,743,787
145.fpppp	19,004,011
146.wave5	19,015,575
Arithmetic Mean	14,827,585
Geometric Mean	13,158,763

Table 6: **Average SPEC95 Context Switch Intervals** - measurements are given in 500 MHz clock cycles

More importantly, when we measured the actual context switch intervals for SPEC95 on a modern DEC Alpha workstation (DS20 with dual 500 MHz Alpha 21264 processors), the context interval, Q, was measured to be sufficiently large (14.8 million instructions, on average) that multiprogramming has very little effect on miss ratios. Unlike multimedia applications which typically have several tightly cooperating threads, the SPEC applications are single threaded and entirely compute bound.

Many UNIX-type operating systems maintain context switch counts on a per process basis which is accessible through the `getrusage()` system call. The average context switch interval was computed in the same manner as for the Windows multimedia applications. Table 6 lists context switch intervals for SPEC95 measured for Compaq Tru64 Unix v5.0 running on a DEC DS20 workstation (dual 21264 processors, each running at 500 MHz), with 2 GB of RAM, again running in a system with a single active task.

4.4 Simulation Details

The component applications for both the multimedia workload and SPEC95 were compiled for the Alpha AXP architecture running Digital UNIX v4.0E with the default optimization levels in the case of the multimedia workload, and the base optimization level for SPEC95 (the same compiler optimization flags on all applications: `-fast -O5 -non_shared`). The resulting binaries were then instrumented with the Cheetah cache simulator using ATOM and run on 300 MHz DEC Alpha AXP machines with 128 MB of RAM.

All of the applications in the Berkeley Multimedia Workload are written in C with the exception of DjVu which is coded in C++. Data sets were chosen to be on the order of real workloads, with long enough traces (instruction and data) to exercise very large caches, or to at least touch as much address space as the corresponding real applications. The trace lengths and other relevant simulation characteristics are listed in Table 4. Total simulation time for our work, not including false starts, machine down time and other simulation problems, was 24.4 days of CPU time for the multimedia workload, and 147.2 days of CPU time for SPEC95 simulations, for a grand total of 171 days of CPU time. The machine type used for simulation was a DEC AlphaStation 255 workstation with a single 300 MHz Alpha 21064a processor).

5. RESULTS

The two major determinants of cache performance are *access time* (the latency from the beginning of an access until the time the requested data is retrieved) and *miss ratio* (the fraction of cache references which are not found in the cache) [30]. Based on the latencies of a particular cache memory candidate design, in combination with the simulated or measured miss ratio, it is possible to select the design with the highest overall performance (lowest average memory access time) at some level of implementation cost.

Complete tables of the results from all of our simulations are available on the world wide web at *http://www.cs.berkeley.edu/~slingn/research/*, from which the cache performance of any application set of interest can be computed.

As it is necessary to reduce the large volume of our simulation results into a more easily digestible form, we use averaging where necessary to compress results. Because the number of applications representing a particular application domain (audio, speech, document, video, 3D) is arbitrary, we will let each of the five application domains comprise a total of 20% of the averaged workload result, with the component applications of each domain being weighted equally.

5.1 Capacity

Capacity, or total cache size, has the greatest effect on miss ratio, and so it is one of the most important cache design parameters. Capacity, especially for L1 caches which are typically on the same die as the CPU, is limited by physical die size and implementation cost. In addition, the larger the capacity of a cache, the slower it is due to increased loading of critical address and data lines, thus requiring additional buffering [24]. In order to study the effect of cache capacity on miss ratio, caches were simulated ranging in size from 1K to 2M bytes.

5.1.1 Other Workloads

The results of other studies on the effect of cache size on the miss ratio for a variety of other workloads are presented alongside our simulation results for the Berkeley Multimedia Workload. All of the miss ratios presented in Figures 2, 3, and 4 are for caches with a line size of 32 bytes and two-way associativity, which represent common values for these parameters. Because the results shown have been gathered from a motley assortment of studies of disparate ages and architectures, many of which did not analyze configurations precisely identical to ours in terms of line size and associativity, we use adjusted results taken from [16]. These adjustments modify the original results of the studies according to the ratios of miss ratios found in [18] for differences in associativity, and [32] for variations in line size. Extensions to larger cache sizes were made for the DTMR results using the $\sqrt{2}$ rule from [30]. It is important to note that many of the other studies included for comparison purposes also measured or simulated multiprogramming behavior, but because they are based on older machine architectures, their Q (quantum) lengths and therefore their context switch intervals are significantly shorter than those used in our simulations.

The most significant result of Figures 2, 3, and 4 is that far from multimedia applications exhibiting degenerate cache behavior in comparison to more traditional workloads, our results demonstrate that they actually perform better for nearly all cache configurations. We believe that this is attributable to several factors. First, most of the comparison workloads are for timeshared machines on which task switching between users occurred very frequently. Further, the comparison studies are of architectures with much lower clock speeds than modern processors, and so exhibit higher miss ratios due to shorter context switch intervals based on real time periods. Even so, the uniprogrammed SPEC92 and SPEC95 benchmarks still demonstrate higher miss ratios than our multimedia workload. We believe that this is because many multimedia algorithm building blocks (such as the discrete cosine transform and fast Fourier transform) internally reference the same data locations repeatedly. In the case of streaming multimedia applications, data is typically copied into a fixed region of memory (buffer) from the source file

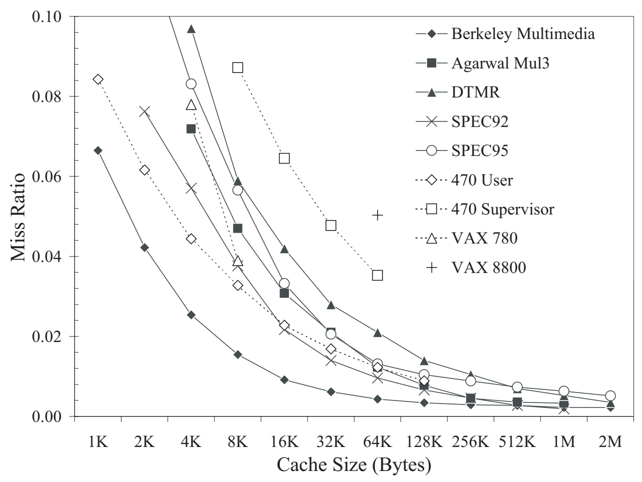


Figure 2. **Unified Cache Miss Ratio** - 32B line size

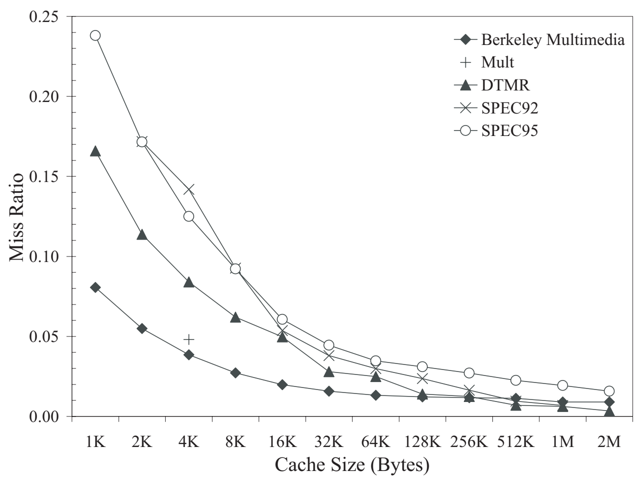


Figure 4. **Data Cache Miss Ratio** - 32B line size

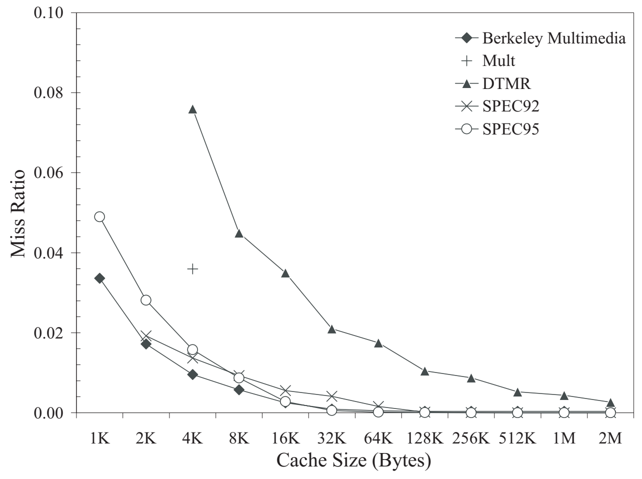


Figure 3. **Instruction Cache Miss Ratio** - 32B line size

of network interface device. Even algorithms which simply traverse enormous arrays of data without re-referencing (such as color space conversion, subsampling) typically do so in linear memory order, and so benefit greatly from the "prefetching" effect of long cache lines. In addition, multimedia data types are typically small (8-bits for video and speech, 16-bits for audio, single precision (32-bit) floating point for 3D geometry calculations). This means that in comparison to the other workloads which utilize full 32-bit integers or 64-bit (double precision) floating point, more multimedia data elements fit in a single cache line, thus improving the relative hit ratio.

5.1.2 Multimedia Domains

When broken down into the five application domains (audio, speech, document, video and 3D graphics), some important trends become apparent (Figures 5, 6, and 7). Instruction cache miss ratios are quite similar across the various application domains, with a 16 KB or 32 KB cache being sufficient. This supports the idea that multimedia applications are dominated by small kernel loops, rather than large code sizes. Data cache miss ratios show significant variation between domains. Speech, video, and audio domains exhibit similar (low miss ratio) cache performance, while the document and 3D applications have higher miss ratios. This is attributable to the non-linear way in which data sets are traversed during processing for these applications.

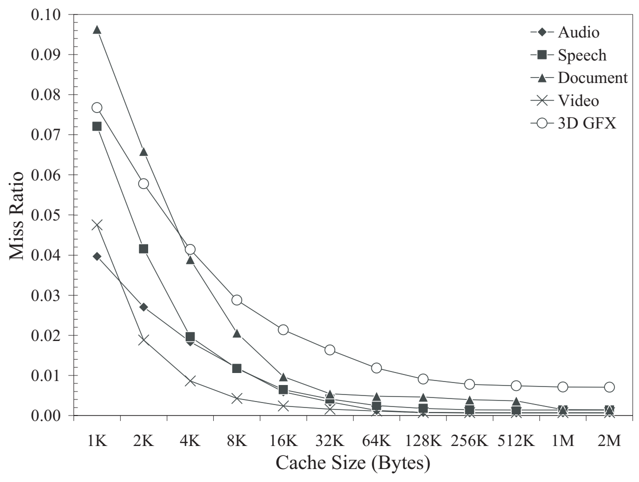


Figure 5. **Multimedia Domains: Unified Cache** - 32B line size

5.1.3 SIMD Effects

The motivation behind the SIMD within a register approach taken by multimedia extensions such as Intel's MMX or Motorola's AltiVec is the fact that on general purpose microprocessors, data paths are typically 32 or 64-bits wide, while multimedia applications typically deal with narrower width data. By packing multiple narrow

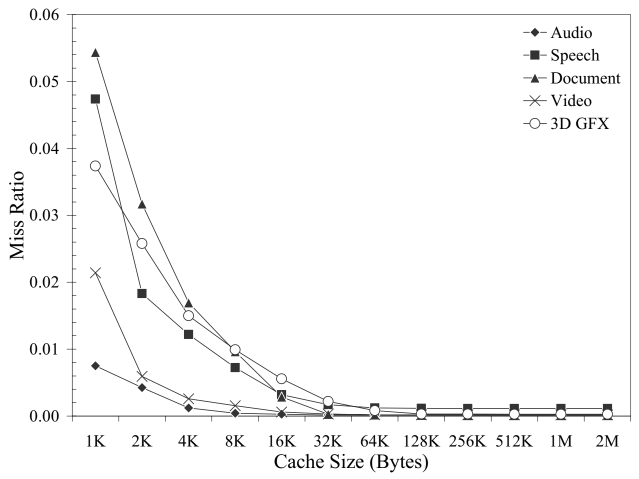


Figure 6: **Multimedia Domains: Instruction Cache** - 32B line size

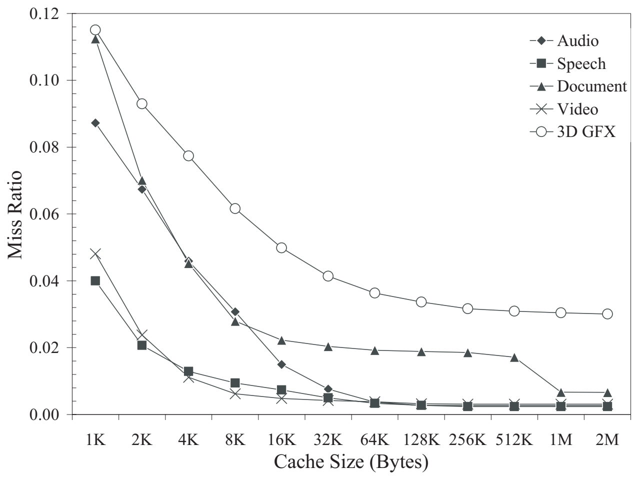


Figure 7. **Multimedia Domains: Data Cache** - 32B line size

operations into the wider native processor data path, it is possible to improve performance.

Although it might be expected that current scalar compilers would place multiple short values into a register and then extract them with register to register operations in order to minimize memory access overhead, we found that this was not the case for the two compilers available on our DEC Alpha test platform. Instead, multiple independent short loads are issued. Because of this, the use of SIMD instruction set extensions for multimedia will result in higher cache miss ratios, although the total number of memory references would decrease, due to the folding of several scalar load operations into a single parallel operation for sub-word data types which are adjacent in memory. Note that programs employing multimedia (SIMD) instruction sets are likely to be hand-coded, as no currently available commercial compilers are able to generate SIMD instructions automatically; this will also affect their memory behavior.

5.2 Line Size

The block or line size of a cache memory is another cache design parameter that strongly affects cache performance [32]. Generally, increasing the line size decreases the miss ratio, since each fetch from memory retrieves more data, thus fewer accesses outside the cache are required. When the line size is made too large, *memory pollution* can adversely affect cache performance, causing material to be loaded that is either never referenced or evicting information that would have been referenced before being replaced. Large lines also decrease the likelihood of "line crossers" - multibyte memory accesses across the boundary between two cache lines, such as occur with many CISC architectures. This type of unaligned access incurs a performance penalty since it usually requires two cache accesses; string operations can induce multiple cache data misses Additionally, small line sizes require a greater number of bits be dedicated to tag space than for larger lines, although a *sector* or *sub-block* cache is one way to avoid this problem. (See [26] for an investigation into sub-sector cache design issues.)

In addition to affecting the performance metric of miss ratio, large line sizes can have long transfer times and create excessively high levels of memory traffic [32]. It is possible to model the time to fetch a cache line, t_{line}, assuming no prefetching and that all loads load a full cache line:

$$t_{line} = t_{latency} + \frac{\left(\frac{L}{d}\right)}{r_{xfer}} \tag{1}$$

where,

L - line size (bytes)

d - data path width to memory (bytes)

$t_{latency}$ - delay for any memory transaction, consisting primarily of memory latency and address transmission time (seconds)

r_{xfer} - bus transfer rate or bandwidth (bytes per second)

For every cache capacity there is an optimal line size that minimizes the average memory reference delay. In order to select an optimal line size, it is necessary to minimize $t_{line} \cdot m(L)$, where $m(L)$ is the miss ratio as a function of line size. To investigate the effect of line size choice on miss ratio, instruction and data caches were simulated with line sizes ranging from 16 bytes to 256 bytes and total capacities ranging from 1 KB to 2 MB. For the sake of example, we use the parameters measured for the memory hierarchy on a 500 MHz AMD Athlon system, listed in Table 7 (the methodology used to obtain these parameters is detailed in [29]). Because we are only considering one level caches in this work, we use the measured L2 parameters for the memory miss latency and bandwidth.

In the case of the largest caches simulated (1M and 2M capacity), the largest line size of 256 bytes produced minimal average delay for instruction caches. Table 8 summarizes the mean memory reference delay for the multimedia workload for SPEC92 and SPEC95, in addition to the Berkeley Multimedia Workload. The best values are highlighted in bold text. Some of the instruction cache results exhibit anomalies for extremely small miss ratios due to the limited precision of our results in those instances (only a few misses for many millions of instruction references).

Table 7: **Memory Latency and Bandwidth** - where $t_{latency}$ is the time delay for any memory transaction, consisting primarily of memory latency and address transmission time and r_{xfer} is the bus transfer rate or bandwidth in bytes transferred per unit time. (*)Microstar - Microstar 6167 motherboard utilizing AMD's AMD-750 chipset, Mandrake Linux v7.0, 256 MB RAM (**)BX - unknown motherboard employing the Intel 440BX chipset, RedHat Linux v6.0, 128 MB RAM

System	**L1** $t_{latency}$	**L1** r_{xfer}	**L2** $t_{latency}$	**L2** r_{xfer}	**Mem**$t_{latency}$	**Mem**r_{xfer}
Microstar* AMD Athlon (500 MHz)	4.0 ns	2657.18 MB/s	109.7 ns	1182.90 MB/s	242.5 ns	305.76 MB/s
DEC DS10 Alpha 21264 (466 MHz)	4.3 ns	1939.14 MB/s	30.4 ns	825.27 MB/s	197.2 ns	336.92 MB/s
BX** Intel Pentium III (450 MHz)	4.4 ns	1695.97 MB/s	46.6 ns	806.94 MB/s	149.8 ns	308.33 MB/s
HP N-Class PA-8500 (3 x 450 MHz)	4.6 ns	2190.42 MB/s	-	-	293.3 ns	338.50 MB/s

Our results indicate that for the Berkeley Multimedia Workload (as well as SPEC95), instruction cache line sizes should be as large as possible, due to the extremely low miss ratios exhibited for even moderate capacities. Instructions are likely to be accessed sequentially, so the fetching of large line sizes pays off. Data caches, on the other hand, have clearly optimal line sizes, depending on the total cache capacity. In the selection of an optimal line size, it should be kept in mind that large line sizes can be problematic in multiprocessor systems where system bus bandwidth must be shared. Very long line sizes may also cause real-time problems, as when I/O operations cause buffer overruns due to an inability to get on the memory bus. With many desktop computer manufacturers already offering 2 and even 4-way multiprocessor support, this may have a limiting effect on the usefulness of long cache lines.

5.3 Associativity

Determining optimal associativity is important because changing associativity has a significant impact on cache performance (latency) and cost. Increasing set associativity may require additional multiplexing in the data path as well as increasing the complexity of timing and control [24]. [18] develops a rule of thumb for how associativity affects miss ratio: reducing associativity from eight-way to four-way, from four-way to two-way, and from two-way to direct mapped was found to cause relative miss ratio increases of approximately 5, 10, and 30 percent, respectively. In order to see how associativity affects miss ratios for our multimedia workload, *miss ratio spreads* were calculated for unified, data and instruction caches for our suite of multimedia applications. Miss ratio spread computes the benefit of increasing associativity, and is defined in [18]:

$$Miss\,Ratio\,Spread = \frac{m(A=n) - m(A=2n)}{m(A=2n)} \qquad (2)$$

Where $m(A=n)$ is the miss ratio for n-way set associativity, A. As in [18], a block size of 32 bytes was chosen, with all simulated caches utilizing LRU replacement. The miss ratio spreads of the Berkeley Multimedia Workload as well as SPEC92 and SPEC95 are shown in Figure 8. Please note that in order to preserve visual detail across the wide range of workload behaviors observed, the subfigures use different vertical scales. Unlike the original [18] study, our curves are not smoothed or averaged.

From the miss ratio spread results in Figure 8, we can see that instruction caches for multimedia applications (and generally for SPEC92 and SPEC95) benefit from 2- or 4-way associativity for moderate size caches (16 KB to 256 KB). For the multimedia workload, most of the benefit from associativity seems to be obtained with two-way set associativity; additional associativity does not to improve performance significantly, except for small cache sizes. Increasing associativity can also be a useful way to increase overall cache capacity when limited by virtual memory constraints (a limited number of page offset bits to index the cache). This was the approach taken both by the designers of Motorola's G4 processor (which includes 8-way associative L1 caches) as well as the IBM 3033 which has a 16-way associative 64k cache.

6. MULTIMEDIA TRENDS

The final determination we would like to make is what cache designers should plan for to support future multimedia applications. This can be thought of in terms of the potential for data set expansion within each multimedia application domain. We expect that audio and speech application data sets will not change significantly in size, as current data sets are already at the limit of human audio fidelity. Document processing should also not change as current documents are sufficient for either printing or previewing at laser printer resolutions.

Video resolutions are not yet close to the limits of the human eye. This can be seen in the high resolution digital formats currently in the pipeline for consumer level products: DVD (720x480), HDTV 720P (1280x720), and HDTV 1080I (1920x1080). In order to determine if the working set size of video applications is increasing, and therefore larger cache capacities are necessary to support these new resolutions, we compared the effect of cache capacity on miss ratios for them in Figures 9 and 10 utilizing the ratio of miss ratios for increasing resolution. Our results were obtained by running the same MPEG-2 decoding and encoding applications with data sets at DVD, HDTV 720P and HDTV 1080I resolutions. As an example of how to interpret the figures, DVD$\Rightarrow$720P refers to the ratio of miss ratios of 720P/DVD resolutions. This metric shows the relative change in miss ratio for the higher resolution compared to the preceding lower resolution.

From Figure 9, we can see that instruction miss ratios are hardly affected by changes in resolution and although there are some minor fluctuations, the ratios are generally quite close to 1.0. Data miss ratios (Figure 10) show a stronger influence for small caches (capacities less than 32K), but level off for larger caches. The type of data locality being exploited by data caches for digital video is presumably at the block or macroblock level (which are the same size in all formats) rather than the frame level since caches are equally effective on all resolutions above a minimum working set size.

Table 8. **Average Delay per Memory Reference (ns)**

Multimedia					
Instruction Cache					
	Block Size (bytes)				
Size	16	32	64	128	256
1K	6.22630	3.79536	2.68253	1.96086	**1.74648**
2K	3.13298	1.94226	1.40646	1.01556	**0.93899**
4K	1.67616	1.08187	0.81106	0.60246	**0.57495**
8K	0.95800	0.64909	0.46229	0.35912	**0.33620**
16K	0.43464	0.28182	0.19453	0.15618	**0.15525**
32K	0.16759	0.10355	0.07412	0.05721	**0.04810**
64K	0.09868	0.05709	0.03657	0.02492	**0.01902**
128K	0.07714	0.04281	0.02534	0.01516	**0.01016**
256K	0.07514	0.04126	0.02407	0.01393	**0.00897**
512K	0.07496	0.04110	0.02392	0.01379	**0.00883**
1M	0.07496	0.04110	0.02392	0.01379	**0.00882**
2M	0.07496	0.04110	0.02392	0.01379	**0.00882**
Data Cache					
Size	16	32	64	128	256
1K	10.70698	**9.09775**	10.17477	14.66781	24.87838
2K	8.04009	**6.20969**	6.31052	8.56975	14.26147
4K	6.15572	4.35004	**3.83097**	4.40862	6.87190
8K	4.64852	3.06934	**2.40616**	2.44560	3.27708
16K	3.48517	2.24199	1.61893	**1.51314**	1.86139
32K	2.81276	1.77501	1.24335	**1.05964**	1.14475
64K	2.44197	1.49988	1.03259	0.82706	**0.78846**
128K	2.30867	1.38347	0.91999	0.71591	**0.65241**
256K	2.23803	1.31936	0.85758	0.64236	**0.57225**
512K	2.18862	1.27134	0.80799	0.58706	**0.50072**
1M	1.94165	1.02288	0.55434	0.31764	**0.19900**
2M	1.93021	1.01452	0.54805	0.31211	**0.19121**
Unified Cache					
Size	16	32	64	128	256
1K	10.28672	7.50803	**6.78065**	7.94751	11.89479
2K	6.56454	4.76822	**4.25393**	4.83131	6.63532
4K	3.94739	2.86512	**2.46797**	2.81398	3.51678
8K	2.47522	1.74480	1.39840	**1.38439**	1.70539
16K	1.54548	1.03360	0.78006	**0.77335**	0.94230
32K	1.06232	0.69658	0.49541	**0.44901**	0.48632
64K	0.75353	0.48619	0.34184	0.28606	**0.27540**
128K	0.61073	0.38293	0.26271	0.21472	**0.19302**
256K	0.55575	0.32711	0.21159	0.15799	**0.13967**
512K	0.53903	0.31085	0.19540	0.13926	**0.11636**
1M	0.48135	0.25490	0.13870	0.07990	**0.05011**
2M	0.47826	0.25256	0.13686	0.07821	**0.04800**

SPEC92					
Instruction Cache					
	Block Size (bytes)				
Size	16	32	64	128	256
1K					
2K	3.71840	2.17213	1.33401	0.99493	**0.77374**
4K	2.70159	1.54599	0.94898	0.63635	**0.47634**
8K	1.90405	1.05308	0.62611	0.40832	**0.29344**
16K	1.15215	0.62794	0.36595	0.23302	**0.16946**
32K	0.88056	0.46364	0.25446	0.14906	**0.09699**
64K	0.33792	0.17852	0.09883	0.05834	**0.03805**
128K	0.02737	0.01647	0.01171	0.00807	**0.00689**
256K	0.00734	0.00519	0.00359	0.00318	**0.00216**
512K	0.00178	0.00124	0.00069	**0.00012**	0.00014
1M	0.00122	0.00067	0.00012	**0.00012**	0.00014
2M					
Data Cache					
Size	16	32	64	128	256
1K					
2K	20.96124	**19.39993**	19.84360	23.28471	30.41951
4K	17.25565	**16.02179**	16.25980	18.30165	23.07412
8K	11.54032	**10.47322**	10.94480	12.22563	15.48550
16K	8.27219	6.05842	**5.26856**	5.28466	6.37997
32K	6.55353	4.29003	3.35500	**3.00983**	3.30687
64K	5.35748	3.37025	2.49839	**2.12650**	2.21407
128K	4.33509	2.67295	1.90952	**1.53742**	1.55691
256K	3.07950	1.84523	1.23878	0.92322	**0.84286**
512K	1.95191	1.08356	0.65050	0.43468	**0.34085**
1M	1.47494	0.77360	0.42169	0.24833	**0.16336**
2M					
Unified Cache					
Size	16	32	64	128	256
1K					
2K	10.24520	8.61222	**8.09564**	9.04907	11.75792
4K	7.78352	6.44349	**6.00378**	6.40153	7.89891
8K	5.28845	4.25106	**3.97639**	4.21803	5.13135
16K	3.50912	2.45442	2.00881	**1.95240**	2.27774
32K	2.52124	1.57757	1.17196	**1.02740**	1.11110
64K	1.77878	1.08804	0.77880	**0.65592**	0.67819
128K	1.21944	0.74588	0.52501	**0.42360**	0.42752
256K	0.86048	0.51497	0.34442	0.25689	**0.23311**
512K	0.54715	0.30834	0.18589	0.12634	**0.10034**
1M	0.40357	0.21293	0.11622	0.06996	**0.04615**
2M					

SPEC95					
Instruction Cache					
	Block Size (bytes)				
Size	16	32	64	128	256
1K	14.27069	7.91963	4.97274	4.35598	**2.87418**
2K	9.98204	5.53019	3.48003	2.85900	**1.89096**
4K	5.59066	3.17658	2.10667	1.75073	**1.23805**
8K	3.20456	1.77827	1.18120	0.98336	**0.72259**
16K	1.81675	0.98199	0.64224	0.51674	**0.38979**
32K	0.58192	0.31805	0.22947	0.22521	**0.16785**
64K	0.14942	0.06065	**0.04621**	0.08140	0.05070
128K	0.09541	0.01656	0.01220	0.03635	**0.00956**
256K	0.07924	0.00570	0.00406	0.02055	**0.00286**
512K	0.07324	0.00188	0.00137	0.01320	**0.00096**
1M	0.07106	0.00045	0.00035	0.00874	**0.00030**
2M	0.07071	0.00019	0.00016	0.00561	**0.00014**
Data Cache					
Size	16	32	64	128	256
1K	28.39914	**26.87923**	29.13518	35.47889	49.55017
2K	21.39439	**19.36667**	20.11207	24.15869	33.54895
4K	16.71953	14.11966	**14.10880**	16.75037	22.38782
8K	13.18879	10.41741	**10.04470**	11.51744	15.18622
16K	10.39420	6.84679	**6.48190**	7.54309	10.22041
32K	8.88017	5.03129	**3.99639**	4.41458	6.26585
64K	7.80708	3.92517	**2.53108**	2.49027	3.74590
128K	7.21604	3.49942	2.14027	1.49853	**1.43653**
256K	6.57932	3.05452	1.78869	1.17317	**0.90454**
512K	5.87008	2.53829	1.39700	0.82849	**0.55867**
1M	5.29881	2.18049	1.14680	0.62642	**0.36857**
2M	4.57430	1.78668	0.93162	0.49926	**0.28320**
Unified Cache					
Size	16	32	64	128	256
1K	23.79340	18.25885	**16.44461**	17.68433	23.34513
2K	17.96932	13.42188	**11.77179**	12.32043	15.71593
4K	12.51167	9.38807	**8.12927**	8.38950	10.45317
8K	8.50513	6.38475	**5.44560**	5.51777	6.80582
16K	5.35151	3.75193	**3.22112**	3.39591	4.25854
32K	3.56756	2.31962	**1.82203**	1.89118	2.48627
64K	2.53677	1.48147	0.99837	**0.97665**	1.39368
128K	2.08627	1.17987	0.73081	0.52139	**0.51386**
256K	1.81360	0.99893	0.58455	0.38362	**0.29702**
512K	1.56373	0.82786	0.45573	0.27048	**0.18301**
1M	1.37100	0.70968	0.37403	0.20508	**0.12160**
2M	1.13350	0.58111	0.30338	0.16290	**0.09276**

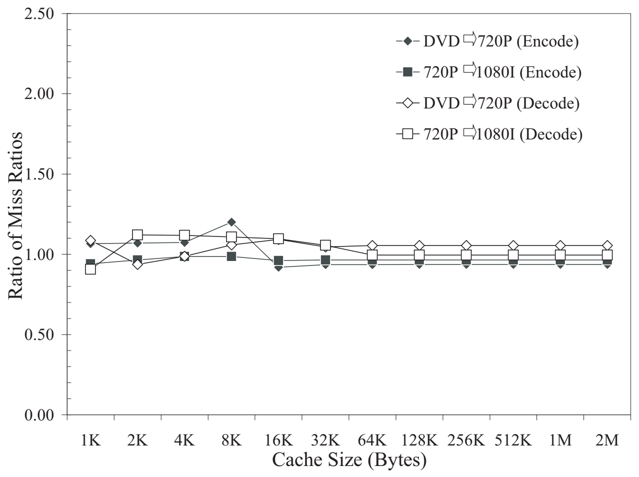


Figure 9: **Instruction Cache Trend** - ratio of miss ratios for increasing resolution

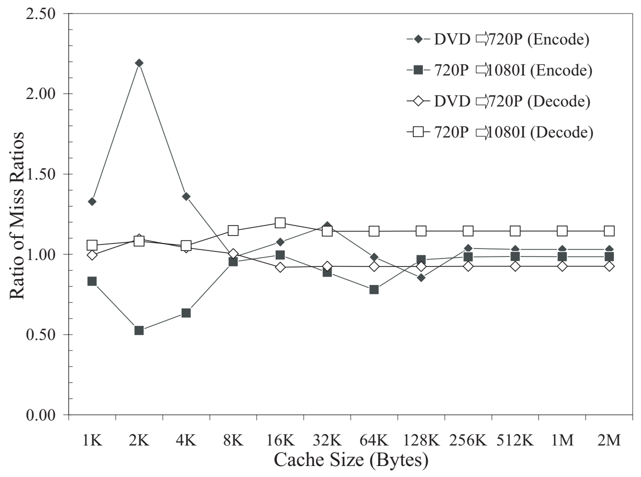


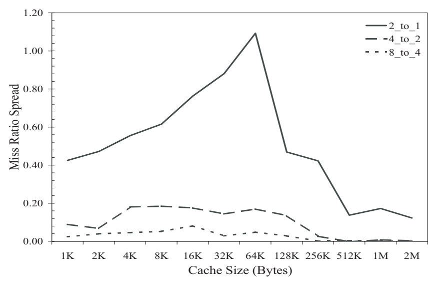
Figure 10: **Data Cache Trend** - ratio of miss ratios for increasing resolution

Previous research ([17], [10], [39]) has found that even a small texture cache located on a 3D accelerator board reduces the required bandwidth to main memory significantly. Past architectural trends suggest that all 3D rendering functionality will eventually be folded into the main processor, at such time as there is adequate silicon (and perhaps pins) to devote to it. We found that 3D applications exhibited the poorest locality of the multimedia domains. Moving 3D functionality entirely onto the CPU (and therefore sharing the cache with other applications) may require the reorganization of program structures to render vertices in an order amenable to LRU caching ([17] examines several approaches for doing this) or larger caches to hold the substantial working sets of such applications. Texture size is dependent more upon the quality of rendered output rather than on display resolution, and is therefore subject to great pressure for growth [19].

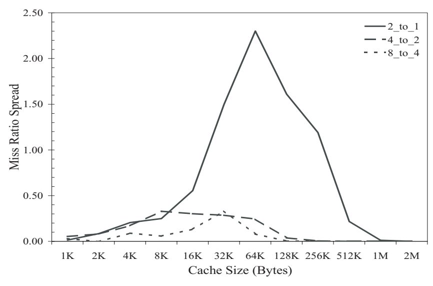
7. SUMMARY

7.1 Cache Design Parameters

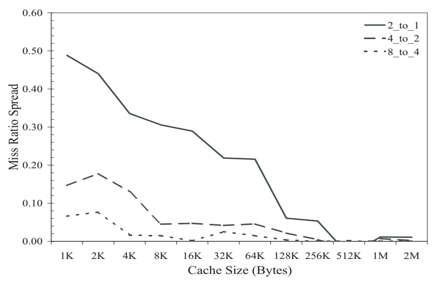
In this paper we have provided a thorough analysis of three important cache parameters in order to support multimedia applications: cache capacity, line size and set associativity. Using execu-



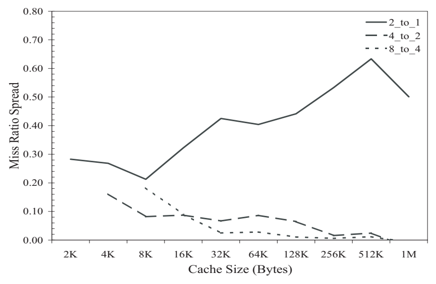
(a) Multimedia - Unified Cache



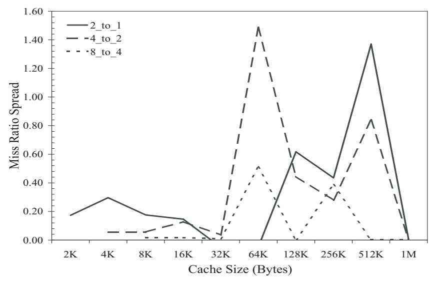
(b) Multimedia - Instruction Cache



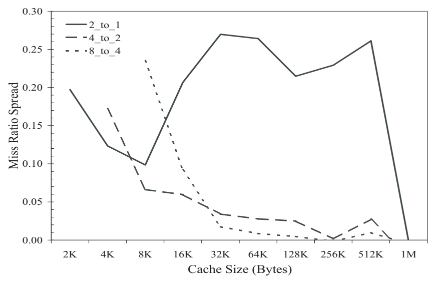
(c) Multimedia - Data Cache



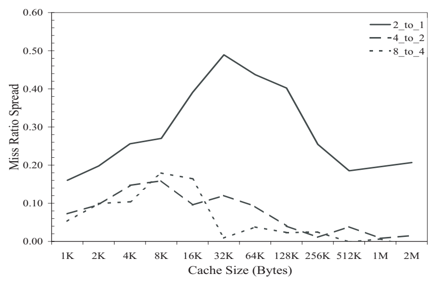
(d) SPEC92 - Unified Cache



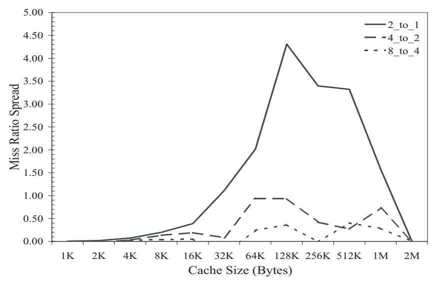
(e) SPEC92 - Instruction Cache



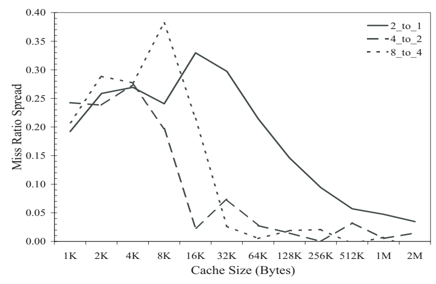
(f) SPEC92 - Data Cache



(g) SPEC95 - Unified Cache



(h) SPEC95 - Instruction Cache



(i) SPEC95 - Data Cache

Figure 8: **Berkeley Multimedia, SPEC92 and SPEC95 Miss Ratio Spreads** - Each line, labeled N to M, indicates the fraction increase in miss ratio when reducing associativity from N-way to M-way. To preserve detail across the wide range of workload behaviors observed, different vertical scales are used in each graph.

tion driven simulation, a large design space was simulated incorporating multiprogramming effects. As can be seen from Table 9, currently available processors are very similar in their cache design choices and based on our derived design parameters, are for the most part well suited for multimedia .

Capacity A moderate instruction cache capacity of 16 KB or 32 KB was found to be sufficient for all of the applications in our multimedia workload. Despite the widespread misconception that multimedia applications exhibit poor data cache performance, the Berkeley Multimedia Workload was found to exhibit quite low miss ratios. Optimal data cache size depends on the type of multimedia applications that are of interest. For the most common audio, speech and video multimedia applications, a data cache of 32 KB in capacity is large enough to exhibit low (<1%) miss ratios. Document and 3D processing exhibit less locality, and in fact even the largest cache sizes simulated (2 MB) still suffered significant misses for 3D graphics. As mentioned, this is due in large part to the fact that 3D graphics primitives (vertices) are processed in object order rather than memory order, leading to poor memory referencing behavior.

Table 9. **Current L1 Cache Configurations [6][23]**

	$I Size (KB)	**$I Assoc**	**$I Line Size (B)**	**$D Size (KB)**	**$D Assoc**	**$D Line Size (B)**
AMD Athlon	64	2	64	64	2	64
DEC 21264A	64	2	16	64	2	64
HP PA-8500	512	4	32/64	1024	4	32/64
Intel Pentium III	16	4	32	16	4	32
MIPS R12000	32	2	32	32	2	32
Motorola 7400 (G4)	32	8	32	32	8	32
Sun UltraSPARC IIi	16	2	32	16	1	32

Line Size We found benefit in instruction cache block sizes as large as the largest in our study (256 bytes) for the memory technology examined at any cache capacity. Data cache block size selection is more dependent on the capacity of the cache. It is important to note that our block size choices considered only average memory access time, and did not consider issues such as total memory traffic or bus busy periods, which are important considerations for multiprocessor machines.

Associativity Based on the results of the miss ratio spread analysis, instruction caches can optimally benefit from 2- or 4-way associativity for most moderate cache sizes (16 KB to 256 KB). Data cache benefits from varying degrees of associativity are more difficult to generalize and appear to be highly dependent on the specific workload, but in general, 2-4 way associativity is also a good choice.

7.2 Conclusion

We have presented a large quantity of simulation and measurements which strongly suggests that multimedia applications exhibit lower instruction miss ratios and comparable data miss ratios when contrasted with other other widely studied, more traditional workloads. Our research indicates, and is supported by the results in [15] on caches for vector architectures, that significant thought and effort must be put into an algorithm for it to exhibit truly degenerate cache behavior. Even though many multimedia algorithms operate on large streams of data which, when considered overall, do flush the cache, at the lowest levels a multimedia algorithm is like any other. Intermediate and constant values are reused, registers are spilled and reloaded, etc. Caches are beneficial on a smaller scale within each algorithmic step, where data (or at least each line of data) is referenced multiple times.

References

[1] Anant Agarwal, John Hennessy, Mark Horowitz, "Cache Performance of Operating System and Multiprogramming Workloads," *ACM Trans. on Computer Systems*, Vol. 6, No. 4, November 1988, pp. 393-431

[2] Advanced Micro Devices, Inc., “AMD Athlon Processor x86 Code Optimization Guide,” Publication #22007G/0, April 2000, *http://www.amd.com/products/cpg/athlon/techdocs/pdf/22007.pdf*, retrieved April 24, 2000

[3] Todd M. Austin, Doug Burger, Manoj Franklin, Scott Breach, Kevin Skadron, "The SimpleScalar Architectural Research Tool Set," *http://www.cs.wisc.edu/~mscalar/simplescalar.html*, retrieved April 24, 2000

[4] Vasudev Bhaskaran, Konstantinos Konstantinides, and Balas Natarajan, "Multimedia architectures: from desktop systems to portable appliances," *Proc. Multimedia Hardware Architectures*, San Jose, California, February 2-14, 1997, *SPIE Vol. 3021*, pp. 14-25

[5] Anita Borg, R. E. Kessler, David W. Wall, “Generation and Analysis of Very Long Address Traces,” *Proc. 17th Int’l Symp. on Computer Architecture*, Seattle, Washington, May 28-31, 1990, pp. 270-279

[6] Tom Burd, “CPU Info Center: General Processor Info,” *http://bwrc.eecs.berkeley.edu/CIC/summary/local/summary.pdf*, retrieved April 24, 2000

[7] D. W. Clark, "Cache Performance in the VAX-11/780," *ACM Trans. on Computing Systems*, Vol. 1, No. 1, February 1983, pp. 24-37

[8] D. W. Clark, P. J. Bannon, J. B. Keller, "Measuring VAX 8800 Performance with a Histogram Hardware Monitor," *Proc. 15th Int’l Symp. on Computer Architecture*, Honolulu, Hawii, May 30-June 2, 1988, pp. 176-185

[9] Thomas M. Conte, Pradeep K. Dubey, Matthew D. Jennings, Ruby B. Lee, Alex Peleg, Salliah Rathnam, Mike Schlansker, Peter Song, Andrew Wolfe, "Challenges to Combining General-Purpose and Multimedia Processors," *IEEE Computer*, Vol. 30, No. 12, December 1997, pp. 33-37

[10] Michael Cox, Narendra Bhandari, Michael Shantz, "Multi-Level Texture Caching for 3D Graphics Hardware," *Proc. 25th Int’l Symp. on Computer Architecture*, Barcelona, Spain, June 27-July 1, 1998, pp. 86-97

[11] Rita Cucchiara, Massimo Piccardi, Andrea Prati, “Exploiting Cache in Multimedia,” *Proc. of IEEE Multimedia Systems ’99* Vol. 1, Florennce, Italy, July 7-11 1999, pp. 345-350

[12] Digital Equipment Corporation, *ATOM Reference Manual*, *http://www.partner.digital.com/www-swdev/files/DECOSF1/Docs/Other/ATOM/ref.ps* retrieved April 24, 2000

[13] Keith Diefendorff, Pradeep K. Dubey, "How Multimedia Workloads Will Change Processor Design," *IEEE Computer*, Vol. 30, No. 9, September 1997, pp. 43-45

[14] Jan Edler, Mark D. Hill, “Dinero IV Trace-Driven Uniprocessor Cache Simulator,” *http://www.neci.nj.nec.com/homepages/edler/d4/*, retrieved April 24, 2000

[15] J. Gee, A. J. Smith, “The Performance Impact of Vector Processor Caches,” *Proc. 25th Hawii Int’l Conf. on System Sciences*, Vol. 1, January 1992, pp. 437-449

[16] Jeffrey D. Gee, Mark D. Hill, Dionisios N. Pnevmatikatos, Alan Jay Smith, "Cache Performance of the SPEC92 Benchmark Suite," *IEEE Micro,* Vol. 13, No. 4, August 1993, pp. 17-27

[17] Ziyad S. Hakura, Annop Gupta, "The Design and Analysis of a Cache Architecture for Texture Mapping," *Proc. 24th Int'l Symp. on Computer Architecture*, Denver, Colorado, June 2-4, 1997, pp. 108-120

[18] Mark D. Hill, Alan Jay Smith, "Evaluating Associativity in CPU Caches," *IEEE Trans. on Computers*, Vol. 38, No. 12, December 1989, pp. 1612-1630

[19] Intel Corporation, "Accelerated Graphics Port Interface Specification v2.0," May 4, 1998, *http://developer.intel.com/technology/agp/*, retrieved April 24, 2000

[20] Richard Eugene Kessler, "Analysis of Multi-Megabyte Secondary CPU Cache Memories," *University of Wisconsin-Madison Computer Sciences Technical Report #1032*, July 1991

[21] Ichiro Kuroda, Takao Nishitani, "Multimedia Processors," *Proc. IEEE*, Vol. 86, No. 6, June 1998, pp. 1203-1221

[22] Ruby B. Lee, Michael D. Smith, "Media Processing: A New Design Target," *IEEE Micro*, Vol. 16, No. 4, August 1996, pp. 6-9

[23] Kim Noer, "Heat Dissipation Per Square Millimeter Die Size Specifications," *http://home.worldonline.dk/~noer/*, retrieved April 24, 2000

[24] Steven Przybylski, Mark Horowitz, John Hennessy, "Performance Tradeoffs in Cache Design," *Proc. 15th Annual Int'l Symp. on Computer Architecture,* Honolulu Hawaii, May 30-June 2, 1988, pp. 290-298

[25] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo Lopez-Lagunas, Peter R. Mattson, John D. Owens, "A Bandwidth-Efficient Architecture for Media Processing," *Proc. 31st Int'l Symp. on Microarchitecture*, Dallas, Texas, November 30-December 2, 1998, pp. 3-13

[26] Jeffrey B. Rothman, Alan Jay Smith, "The Pool of Subsectors Cache Design," *Proc. Int'l Conference on Supercomputing*, Rhodes, Greece, June 20-25 1999, pp. 31-42

[27] Jeffrey B. Rothman, Alan Jay Smith, "Multiprocessor Memory Reference Generation Using Cerberus," *Proc. 7th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, College Park, Maryland, October 24-28, 1999, pp. 278-287

[28] Nathan T. Slingerland, Alan Jay Smith, "Design and Characterization of the Berkeley Multimedia Workload," *University of California at Berkeley Technical Report CSD-00-1122*, December 2000

[29] Nathan T. Slingerland, Alan Jay Smith, "Cache Performance for Multimedia Applications," *University of California at Berkeley Technical Report CSD-00-1123*, December 2000

[30] Alan Jay Smith, "Cache Memories," *ACM Computing Surveys*, Vol. 14, No. 3, September 1982, pp. 473-530

[31] Alan Jay Smith, "Cache Evaluation and the Impact of Work-load Choice," *Proc. 12th Int'l Symp. on Computer Architecture*, Boston, Massachusetts, June 17-19, 1985, pp. 64-73

[32] Alan Jay Smith, "Line (Block) Size Choice for CPU Cache Memories," *IEEE Trans. on Computers*, Vol. C-36, No. 9, September 1987, pp. 1063-1075

[33] Alan Jay Smith, "Trace Driven Simulation in Research on Computer Architecture and Operating Systems," *Proc. Conference on New Directions in Simulation for Manufacturing and Communications*, Tokyo, Japan, August 1994, pp. 43-49

[34] Peter Soderquist, Miriam Leeser, "Optimizing the Data Cache Performance of a Software MPEG-2 Video Decoder," *Proc. ACM Multimedia '97*, Seattle, Washington, November 9-13, 1997, pp. 291-301

[35] Pieter Struik, Pieter van der Wolf, Andy D. Pimentel, "A Combined Hardware/Software Solution for Stream Prefetching in Multimedia Applications," *Proc. 10th Annual Symp. on Electronic Imaging*, San Jose, California, January 1998, pp. 120-130

[36] Rabin A. Sugumar, Santosh G. Abraham, "Efficient Simulation of Caches under Optimal Replacement with Applications to Miss Characterization," *Proc. 1993 ACM Sigmetrics Conference on Measurements and Modeling of Computer Systems*, May 1993, pp. 24-35

[37] J. Tse, A. J. Smith, "CPU cache prefetching: Timing evaluation of hardware implementations," *IEEE Trans. on Computers*, Vol. 47, No.5, May 1998. pp. 509-26

[38] Richard Uhlig, Trevor Mudge, "Trace-Driven Memory Simulation: A Survey", *ACM Computing Surveys*, Vol. 29, No. 2, June 1997, pp. 128-170

[39] Alexis Vartanian, Jean-Luc Bechennec, Natalie Drach-Temam, "Evaluation of High Performance Multicache Parallel Texture Mapping," *Proc. 1998 Int'l Conference on Supercomputing*, Melbourne, Australia, July 13-17, 1998, pp. 289-296

[40] Glenn Ammons, Tom Ball, Mark Hill, Babak Falsafi, Steve Huss-Lederman, James Larus, Alvin Lebeck, Mike Litzkow, Shubhendu Mukherjee, Steven Reinhardt, Madhusudhan Talluri, and David Wood, "WARTS: Wisconsin Architectural Research Tool Set," *http://www.cs.wisc.edu/~larus/warts.html*, retrieved April 24, 2000

[41] Daniel F. Zucker, Michael J. Flynn, and Ruby B. Lee, "A Comparison of Hardware Prefetching Techniques for Multimedia Benchmarks," *Proc. 3rd IEEE Int'l Conference on Multimedia Computing and Systems,* Hiroshima, Japan, June 17-23, 1996, pp. 236-244