RC23630 (W0506-054) June 9, 2005
Computer Science

IBM Research Report

A Quantitative Analysis of Content Creation and Content Rendering in Media Applications

Brett Matthews, John-David Wellman, Michael Gschwind
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

A quantitative analysis of content creation and content rendering in media applications

Brett Matthews, John-David Wellman, Michael Gschwind

IBM T.J. Watson Research Center
Yorktown Heights, New York

ABSTRACT

Video games represent a class of media applications for which creating video content is as important as displaying it. In this work, we compare modern, 3D video games running on an Apple Macintosh G5 to the traditional multimedia content rendering applications of DVD playback and the QuickTime media player. We use performance monitor counters to find several metrics of the workload performance, including the IPC, the L1 data cache miss rate and both the AGP and processor memory bandwidth. We find the frame-oriented nature of these applications reflected in many of the metrics used in this study. Video game applications exhibit significantly less idle time between updates to the screen than either the DVD playback or QuickTime applications. The results also show that adding computer controlled characters to a video game, increasing the work per frame in the game application, further reduces the idle time, and also causes a decrease in the L2 cache pressure by spreading the access misses over a longer time period. This illustrates the impact of the content-creation task on the overall execution of the programs, where this content-creation is present in the game workloads and absent in the recorded-media playback applications.

1. INTRODUCTION

Video games have become a leading edge software application in the past several decades. According to a recent study by the Entertainment Software Association [3], U.S. computer and video game software sales topped $7 billion in 2003, and the age of the average video game player was 29 years (and increasing). Video game console systems in particular have evolved from low-end microcontrollers to high-end microprocessors and even multiprocessor systems. In the process, platform performance growth has consistently exceeded Moore's Law (and typical workstation system performance improvement rates) as shown in figure 1.

Historically, this rate of game consol system performance increase has been relatively easy to attain by adopting more sophisticated cores containing more aggressive performance features in each generation of systems. Effectively, the high rate of growth has been partly achieved by starting with a much lower base system performance, and steadily adopting more "workload system" features in each console generation. This, in turn, has become possible because these mainstream workstation features have become much less expensive over time, in terms of component cost, processor performance at desired yield levels, etc.

While this approach has been an adequate response to the increased performance demands of successive video game generations, incremental performance increases have become more difficult and expensive in the most recent generations, both in terms of chip area and power. Not surprisingly, as game console core performance reaches that of traditional workstation cores, the law of diminishing returns on investment, in terms of both area and power, apply increasingly to game console cores. To continue delivering more performance to game applications in the future, console and game designers must focus on obtaining a better understanding of the workload characteristics and how better to exploit them to deliver greater core and system performance.

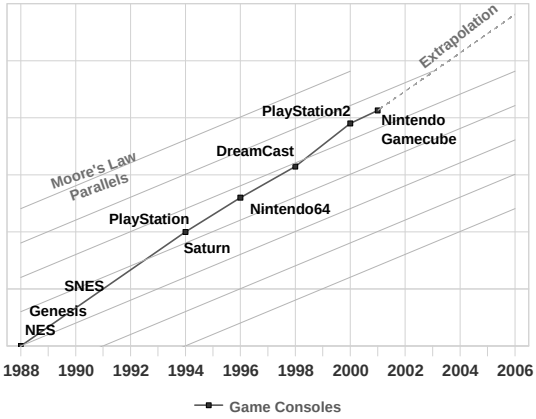


Figure 1: Performance growth of video game console systems (in Mops).

Video games represent a unique and interesting class of applications: they embody the practical intersection of computer graphics, scene databases, and artificial intelligence. Game applications are also developed for a wide variety of

		DVD Player	QuickTime	FP Shooter 1 1600x1200		FP Shooter 2	Parser
				0 Actors	32 Actors		
Run 1	Instr Completed	$6.35 * 10^9$	$.995 * 10^9$	$11.1 * 10^9$	$19.0 * 10^9$	$24.4 * 10^9$	$51.7 * 10^9$
	Execution Cycles	$7.55 * 10^9$	$3.98 * 10^9$	$33.3 * 10^9$	$54.1 * 10^9$	$71.0 * 10^9$	$85.9 * 10^9$
	Instr Per Cycle	0.841	0.250	0.333	0.351	0.343	0.608
Run 2	Instr Completed	$7.92 * 10^9$	$1.03 * 10^9$	$9.02 * 10^9$	$16.4 * 10^9$	$24.9 * 10^9$	$37.3 * 10^9$
	Execution Cycles	$9.48 * 10^9$	$4.07 * 10^9$	$24.9 * 10^9$	$46.7 * 10^9$	$74.1 * 10^9$	$71.0 * 10^9$
	Instr Per Cycle	0.835	0.253	0.370	0.352	0.336	0.526
Run 3	Instr Completed	$7.81 * 10^9$	$1.02 * 10^9$	$9.28 * 10^9$	$17.6 * 10^9$	$17.5 * 10^9$	$37.9 * 10^9$
	Execution Cycles	$9.40 * 10^9$	$4.07 * 10^9$	$25.0 * 10^9$	$49.5 * 10^9$	$58.1 * 10^9$	$17.5 * 10^9$
	Instr Per Cycle	0.830	0.251	0.372	0.356	0.301	0.532

Table 1: Instruction throughput metrics.

systems, including low-power embedded devices (handheld games), sophisticated embedded environments with specialized hardware (console systems) and desktop PCs alike. Video games can be included in the category of media applications, as they produce the same kinds of visual and auditory data streams. The study of video games in this regard is especially interesting because they can at once be considered a streaming media application *and* a multimedia content creation application.

Games are required to continually produce and display real-time video content. They are, however, distinct from traditional streaming multimedia content rendering applications, such as DVD and (other) MPEG players, in that the content to be displayed is generated in real-time, and not taken from pre-recorded input data (e.g., a locally stored MPEG-encoded file, or a streaming feed of encoded frames from the network). It is to be expected that the requirements of content creation, especially in real (or pseudo-real) time, will provide some distinct challenges and thus some unique workload characteristics over traditional media playing applications.

Games are also distinguished from most multimedia content creation applications, in that traditional multimedia content creation is frequently targeted to the generation of files, and thus is not subject to real-time constraints. In this respect, video games are also representative of other future virtual-world applications, such as military training, battlefield simulations, or even virtual-reality interactive environments like virtual shopping malls, which have been discussed as a possible direction for online entertainment.

Throughout this report, we focus extensively on the time-varying behavior of a variety of performance metrics. Previous studies have explored predicting the phase behavior of performance metrics for system optimization [2,9] for SPEC benchmarks. An examination of figures 2 through 7 shows that our results are consistent with the work of Düsterwald *et al.* [2] which finds that for a given application from the SPEC CPU 2000 benchmark suite the waveforms of different performance metrics have characteristically different shapes, but share a quasi-periodic structure. Our results from the analysis of video games and multimedia applications differ from [2], however, in that the phases we find in the time-varying behavior of these performance metrics are shorter (on the order of milliseconds instead of seconds) and correlated directly to the application's frame rate. Effectively, the phases in multimedia applications, including in games, are directly linked to the intra-frame activity, as reflected across the frames.

This work compares video games, as *Real-time Multimedia Content Creation* applications, to traditional streaming media content rendering applications. We make extensive use of the built-in performance counters in the IBM PowerPC 970 processor in a Macintosh G5 desktop PC from Apple. In section 2, we discuss the methodology used in our study, including the hardware, data-gathering methods, and the workloads that were analyzed for this paper. Section 3 presents some of the data we have gathered, and provides an analysis comparing and contrasting the game workloads and the streaming media player applications. Section 4 presents our conclusions to date, and describes our ongoing work in this area.

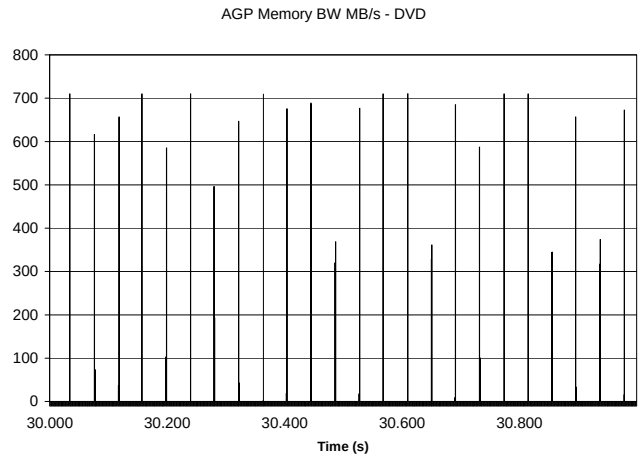
a

2. METHODOLOGY

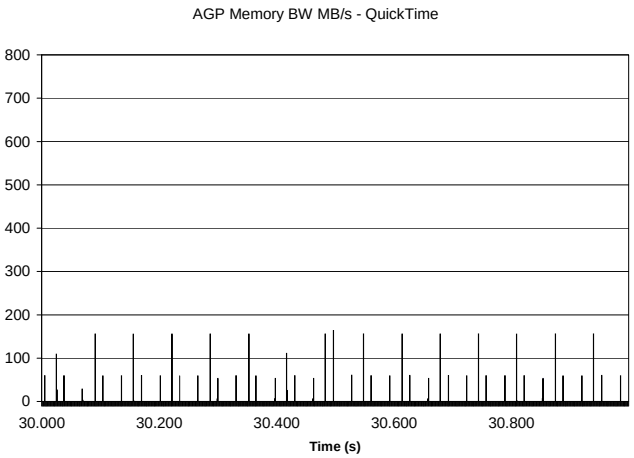
A great deal of work has been done over many years in analyzing computer benchmarks and workloads. Tremendous efforts have been expended both academically and commercially on the analysis of what are deemed to be important benchmarks, including the SPECcpu performance evaluation suite. Companies have also expended a great deal of effort in the analysis and characterization of transaction benchmarks, including the TPC-C and TPC-D benchmarks.

Aware of this historic work and these traditional approaches, one is tempted to apply them in the analysis of game workloads. There are, however, some obvious qualitative difference between traditional benchmarks and the game workloads being analyzed here:

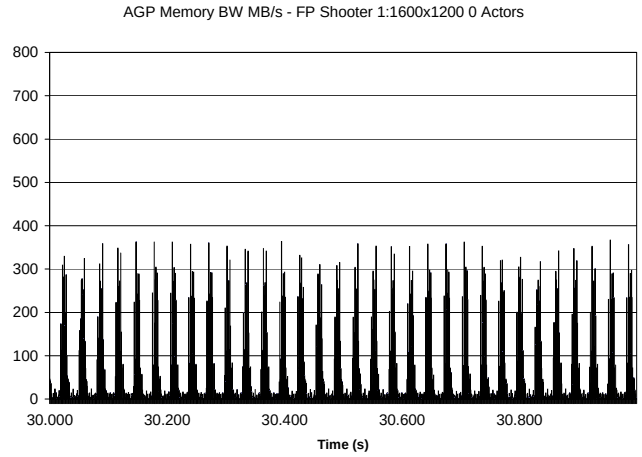
- Most PC-based and all console-based game workloads are commercially packaged application programs for which source code is typically not available.
- Video games typically involve high-bandwidth workloads and a strong dependence on I/O, particularly graphics I/O. Thus, impacting the execution speed of the game application may change the relative ratios of graphics and program memory activity.
- To run across a range of platforms with varying performance, PC-based game workloads change their real-time behavior in response to compute power available.
- Computer games are not easily scripted and have limited repeatability. As such, successive runs may not be able to reproduce the same execution behavior based on timing and type of use input.



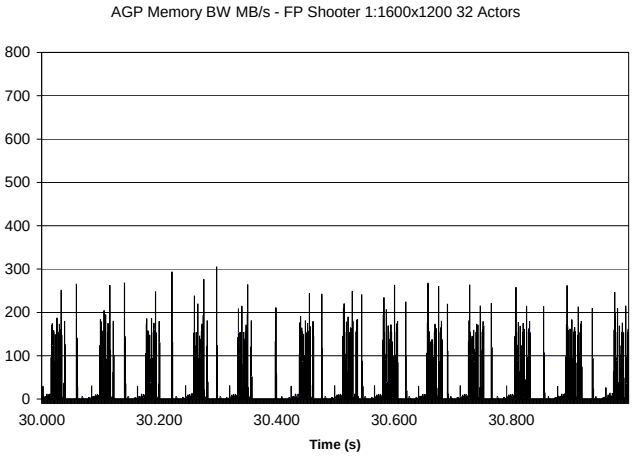
(a) DVD Player



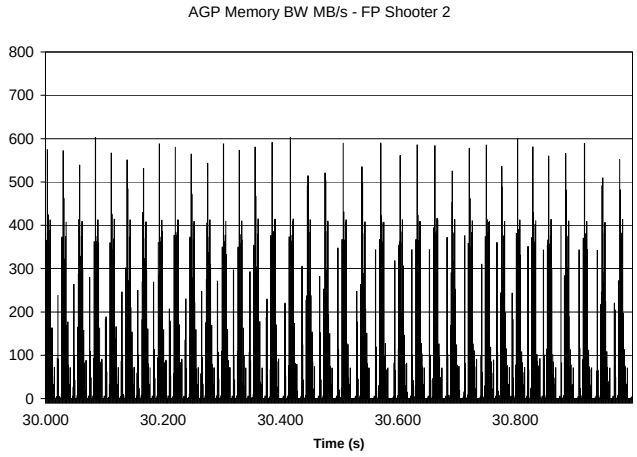
(b) QuickTime



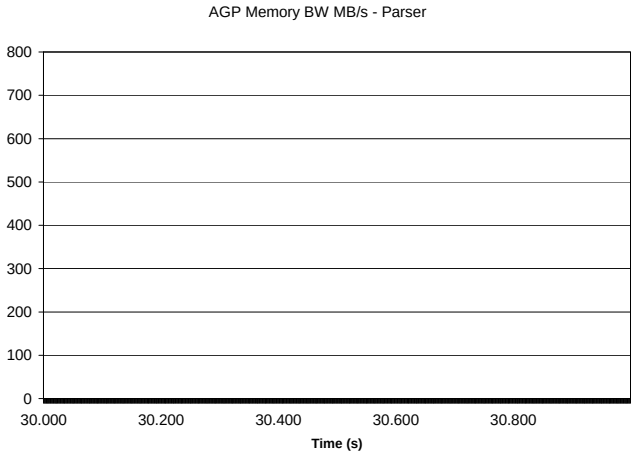
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors



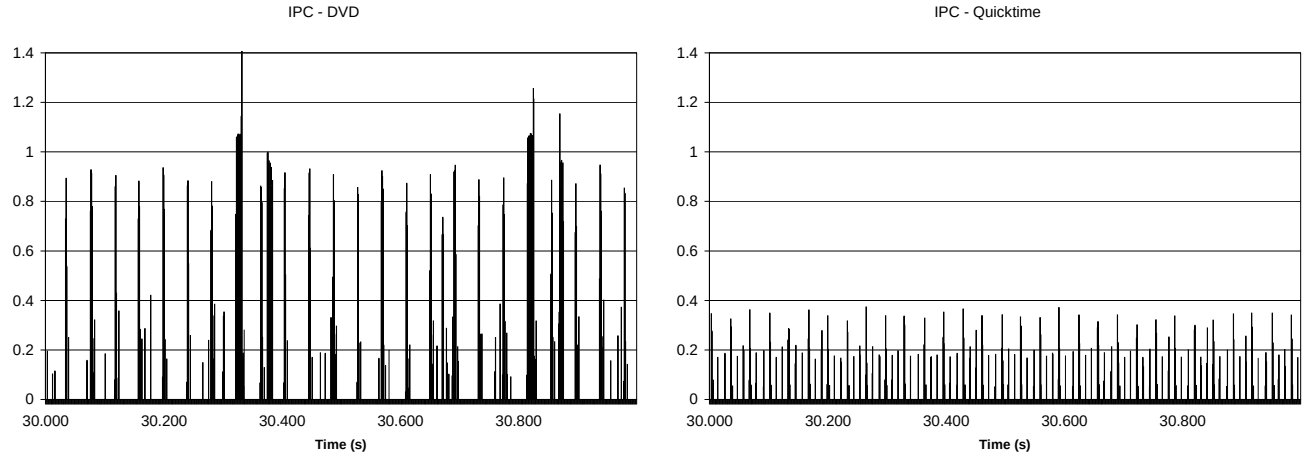
(e) Video Game 2



(f) Parser

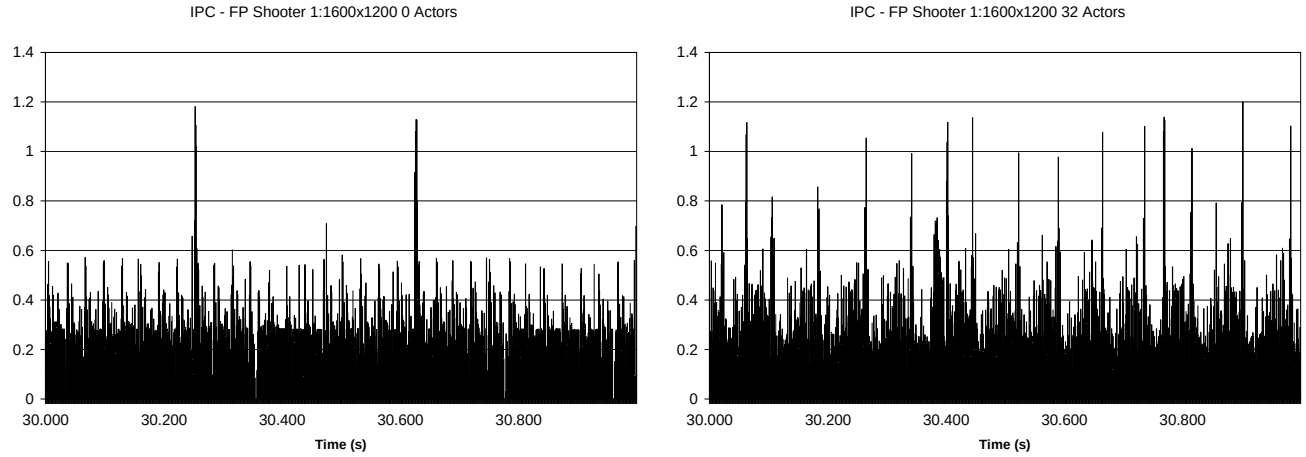
Figure 2: Memory Bandwidth as seen by the Accelerated Graphics Port (AGP) for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.

These characteristics necessitate an analysis methodology which does not require access to source code either for mod-



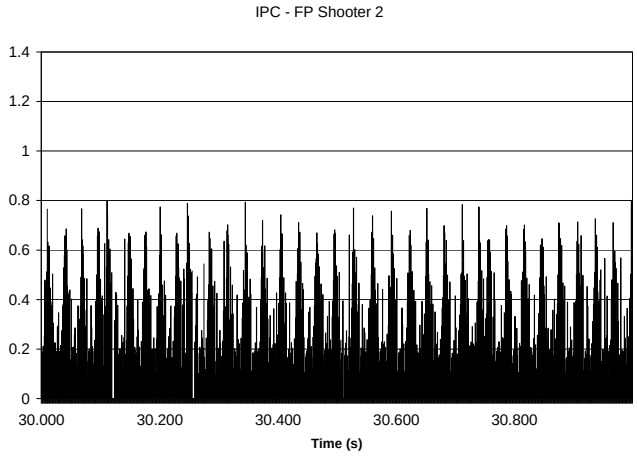
(a) DVD Player

(b) QuickTime

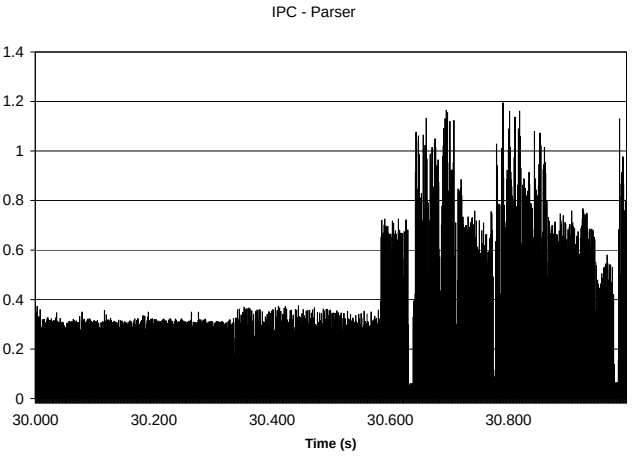


(c) Video Game with 0 Actors

(d) Video Game with 32 Actors



(e) Video Game 2



(f) Parser

Figure 3: IPC for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.

ification or inspection. Further, to address the real-time requirements, the analysis tools must only minimally impact the performance profile of the application.

In the following sections we discuss the methodology and

the tools we used to study and characterize video game and multimedia player workloads. Section 2.1 discusses the desktop PC environment in which our study was performed, drawing important comparisons to video game console systems. Section 2.2 details the performance counters and the tools used to access them. Section 2.3 discusses the application workloads used in the study.

2.1 Environment

There are two major classes of game application environments, general purpose personal computers and video game consoles. While at their core each of these systems is a microprocessor, and in some cases the actual hardware is quite similar, e.g., the first generation Xbox system and a desktop PC, dedicated video game console systems and desktop PCs have some important differences that may significantly impact some aspects of workload behavior.

Games developed for *console systems* are designed for a fixed, application-specific set of hardware; they are typically connected to a TV screen and are usually architected to attempt maintaining a regular frame rate of 30 (or 60) frames per second (fps). While *desktop PC* games can require certain minimum hardware specifications, in order to apply to a large potential marketplace, they must support a relatively wide variety of hardware configurations and abilities. Thus, PC games are often characterized by variable frame rates, where the frame rate is largely determined by the computational needs of the application and the underlying power of the particular hardware platform. The desktop PC games generally provide a series of user-selectable setting controls, governing aspects of the game's graphics (and in some cases even AI and physics models). Most desktop PC games strive to put out the maximum frame rate possible on the given hardware and under the specific video and game settings; if the hardware cannot produce a sufficient frame rate, the user typically must alter these settings to achieve an acceptable frame rate.

Both the desktop PC and console game platforms can also be distinguished by the type of graphics adapters they use. Some platforms opt for highly specialized, high function graphics cards which implement much of the graphics pipeline in hardware (e.g., texture mapping, lighting and shading, rasterization), while other platforms require the general-purpose computing resources to implement the graphics pipeline. Typically, PC-based gaming systems (both Intel x86 and IBM PowerPC based system) use high function graphics cards (such as Nvidia and ATI adapters). In the dedicated game console space, the Microsoft Xbox and the Nintendo Gamecube follow the same high function graphics card paradigm, while Sony's Playstation 2 uses programmable vector units to implement a larger portion of the graphics pipeline in programmable CPUs, driving a lower function graphics adapter. Evidently, these system architecture choices will impact workload characteristics on these platforms, i.e., systems with low function graphics adapters will have a higher fraction of image processing code.

Finally, video game applications developed for PCs are run on top of a general purpose operating system. To date, most console systems have their OS functions tightly integrated into each game, and starting a game involves "rebooting" the system with its own private embedded OS service functions (often in the form of libraries provided by the console vendor). Even where the OS is not implemented as a set of library functions, but as a more dedicated layer (e.g., on the current Xbox system, or Linux game consoles) each game generally includes a copy of the OS code on which it runs (and which is booted when loading the media).

In current generation console systems, the system is dedicated to a single application and all threads are usually under the control of that single (game or other) application. Thus, general-purpose OS function like context switching have no impact on video games developed for console systems, except at the level exploited by programmers of that game, and often then in an application-centric streamlined and performance-focused form. No other system functions run on today's video game console when a game is being played.

2.2 Performance Monitor Counters

To meet the requirements for analyzing game workloads, we have turned to time-interval based sampling and performance monitor counters [6] for program analysis. The time-interval based sampling methodology and results are described in [7]. Here, we report results based on the use of the PowerPC 970's Performance Monitor Counters (PMCs).[1]

Using the processor's built-in PMCs provides a non-intrusive method of extracting data from an application while it runs very close to real time. The remainder of this section describes our methodology and the tools used in this study.

The PowerPC 970 G5 processors support 8 performance monitor counters that can be configured to count from a large variety of events. The Apple Mac G5 U3 Northbridge chip also includes additional counters which can be used to track memory traffic between the various system components. Peaks in AGP traffic corresponding to a screen update, for example, can be used to identify logical frame boundaries as a reference point for inspecting the per-frame behavior of other metrics.

To count more than 8 events for a given workload, the data must be gathered by altering the performance monitor counter controls and taking data either across several different runs or at least across several different sampling intervals. The resulting data must then be combined (and correlated) to provide the larger set of metrics. Given the unscripted, non-repeatable behavior, particularly of the game applications (and especially so at the architectural and microarchitectural levels), this kind of analysis is best used for aggregating statistics across longer time intervals. In table 1, we compare basic metrics across three 50 second runs, to identify variability between runs. While the macroscopic behavior can vary across runs, depending on system state, activity of other processes, and timing of user inputs, it frequently tends to be repeatable. This suggests that large portions of the execution are fairly independent of specific input data, e.g., a game may allocate a fixed time budget to an activity such as game AI, or that the processing effort for different computer-controlled character actions is relatively consistent.

To access the PMCs under Mac OS-X, we have used the Apple CHUD toolset, as well as IBM proprietary PMC analysis tools. We use the core PMCs to obtain data for a number of attributes, including IPC (instructions per cycle), branch prediction accuracy, and cache hierarchy access be-

[1] It should be noted that PMCs are primarily designed for performance tuning and hardware diagnosis, not for workload characterization work. As such, events tend to reflect behavior of a workload on a specific microarchitecture, not in a more abstract microarchitecture-independent form.

havior for the applications in this study. We use the PMCs on the U3 system controller to analyze memory traffic as seen by the processor and the Accelerated Graphics Port.

The Apple CHUD toolset also provides a set of (pseudo) counters that can monitor activities within the operating system. While there are a number of interesting studies that could be pursued using these counters, those studies would be less representative of game workloads in general (especially when considering such dedicated game platforms as consoles), and any such work is outside the scope of this paper.

2.3 Workloads

For our study, we compare video game applications to two traditional streaming media applications: the Quicktime media player and the playback of DVDs. MPEG video encodings typically use 3 types of frame encodings, key frames containing a full image (I-frames), and intermediate P- and B-frames (representing predictive and bi-directionally predictive coded frames, based on motion estimation). DVD video data is typically displayed at a frame rate of 24 or 29.97 fps with a frame size of 720x480 (NTSC) or 25 fps at 720x576 (PAL).

To understand the behavior of game workloads, we have used several commercial game applications for the Apple Macintosh PowerMac G5 under OS X. All game applications use the system OpenGL implementation for graphics rendering. While in previous work we have studied a variety of game applications representing different game playing styles (first person shooter games, war games, racing games, etc.) [7], we concentrate on two first person shooter games in this study. Some applications allow the user to change the number of game-controlled actors ("game bots"), providing a way to study the impact of varying amounts of game AI and other control code.

3. ANALYSIS

Figures 2 through 7 each show a selection of fundamental workload metrics comparing video game applications to traditional streaming multimedia rendering applications (DVD and Quicktime movie). Each of these graphs highlights one second of operation, taken from a 50-second execution of (a) a DVD Player; (b) the QuickTime media player; a First-Person Shooter game application (FPS1), where the computer either controlled (c) 0 or (d) 32 additional non-player characters ("bots"); (e) a second First-Person Shooter video game; and as reference for a more traditional computing workload, (f) a run of the SPECcpu2000 benchmark 197.parser. We elected to show one second of each run (specifically the 31st second) in order to provide, in a single set of graphs, both a reasonably large number of frames (for the game and multimedia workloads) and the ability to distinguish relatively fine-grain behavior.

For the media applications, the behavior shown in figures 2 through 7 for the selected time slice (31st second) is representative of a control set of plots each consisting of other, different 1-second time intervals. The execution behavior of media workloads is periodic, reflecting the frame-oriented nature of their computations. This observation supports the same observation, i.e., that media rendering applications show some periodic behavior reflecting the sequence of different frame types, previously discussed in [5]. For these media workloads, however, no higher-level phase behavior is discernible; all phase behavior appears to be at the frame level.

This periodicity behavior is notably different from some of the traditional desktop computing benchmark workloads, as exemplified by the SPEC CPU suite. For reference, we show the 31st second of 197.parser from the SPECint2000 benchmark suite.[2] As can be seen from figures 12(b) and 15(b), 197.parser has significant phase behavior that cannot be captured within any single one second interval of the run. This illustrates that the behavior of the 197.parser workload over its 31st second of operation in figures 2 through 7 cannot be considered representative of the overall execution behavior for 197.parser. In fact, as can be seen from 3(f), there are several distinct non-periodic phases visible even within the 1 second segment displayed there.

To summarize, the media workloads tend to show periodic phase behaviors in a more regular and localized way. For some applications and application phases, phases of behavior are also observable within a frame, i.e., where each frame's computation will have a similar performance profile, which follows from the sequence of actions taken (computations made) within each frame for frame processing. There is also phase-like behaviors between frames, e.g., for media playback applications, where the media stream consists of different kinds of frames, presumably in a fixed relative ordering, each of which requires slightly different processing (e.g., I-frames compared to P- and B-frames for MPEG players). The media workloads, and in particular the game workloads, do not consistently show longer-term phases of behavior as are often evident in desktop benchmarks.

3.1 AGP Bandwidth

Figure 2 shows the measured memory bandwidth (in MB/s) as seen by the accelerated graphics port (AGP) for each of the applications. In each of the cases, all memory transactions seen by the AGP were reads, indicating that the graphics card does not transmit data back to the processor. This corresponds to our understanding of the current usage of graphics rendering hardware in game workloads, where transfer to graphics adapters is largely (if not exclusively) unidirectional. Note, however, that future game applications may transfer rendered image data back to the processor for further post processing, e.g., for use as computed textures.

On the G5 PowerMac, all graphical applications use the AGP bus to transmit data to the ATI graphics card, and thereby to the display. For all the applications shown here, the peaks in the AGP memory bandwidth correspond directly to the transfer of one frame of output. Graphs of the FFT magnitude spectrum of the 31st second of AGP Memory bandwidth are given in Figure 8. The first significant peak in each of the graphs in figure 8 is its fundamental frequency and is equivalent to the frame rate. This is especially useful for determining the frame rate (or at least an aggregated or average frame rate) for video games where the frame rate varies during execution.

The AGP traffic generated by game workloads and media players are markedly different.[3] Comparing the different

[2]We show a single SPECint2000 benchmark for reference. The present report discusses media workloads, and a broader discussion of SPECcpu benchmark behavior would distract from the focus of this report. [1]

[3]This observation may be specific to the evaluation environ-

workloads, one notes that for the DVD and Quicktime media players, the AGP transfers are quite regular, with very sharply delineated peaks. Further, each transfer tends to involve a single block of data pushed over the AGP bus. This is consistent with the transfer of a full frame of (pixels to be rendered) data, presumably in a single AGP transfer action.

The game applications also show fairly regular peaks, though these peaks occur over a larger number of sample intervals (i.e., the transfers occur over a longer absolute time interval). The total volume of data transferred by the games is notably higher, than for the (single frame of pixels transferred by the) media player applications. This behavior is consistent with the expected utilization of a high-function graphics card by a game workload.

The game (i.e., content creation) workload communicates at a significantly higher abstraction level than the simple bitmap descriptions generally used for transferring the results of media content rendering. Thus, the AGP traffic for the game workload applications – in a high function graphics card environment – corresponds to the collection and processing of the high-level screen description over an extended time period.

This higher-level data consists of a collection of object types which are further processed by the graphics adapter, and currently include textures and polygon information. There is a potential movement in the game industry toward an even higher level of abstraction (e.g., NURBs or other surface representations), but this is not a current mainstream technology, and is not represented by these game workloads.

In comparison, the media players decode a video stream, e.g., in MPEG typically consisting of stored data for I-, P-, and B-frames, into unstructured bitmaps that are more or less directly transferred from the DVD player to the frame buffer.

The volume of data transferred is markedly different for the application types: The FPS1 video game application transfers, on average, approximately 2.6 MB of data across the AGP bus when there are 0 computer-controlled actors. In the case where there are 32 actors, the FPS1 video game transfers on average about 3.1 MB of data. The two media players transfer much less data per frame, with the DVD player sending 700 KB of data, and the QuickTime player sending only 281 KB of data per frame, in each case reflecting the size (resolution) of the videos being rendered. The FPS2 video game averages transfers of about 3.3 MB per frame, which is fairly consistent with the FPS1 with 32 actors results. As to be expected, the 197.parser command line application has no AGP traffic, and is shown here for completeness only.

3.2 Instructions Per Cycle

We analyze IPC for media workloads for phase and periodic behavior at both the multi-second and frame level. Figures 10, 11 and 12 show the distribution of the IPC metric based on sampled performance monitor counters at 1 millisecond intervals for instructions completed, and cycles executed in the application. While figures 10, 11 and 12 give a good reflection of the IPC range, the distribution between the 50000 different sample points is hard to establish from these charts. Figures 13, 14, and 15 show a moving average of the sampled IPC data points using a 200 millisecond sliding window for averaging.

ment based on a high-function graphics card.

Referring to figure 10, the DVD Player shows IPC samples covering a wide spectrum (with a few data points exceeding 1.5 instructions per cycle), compared to the Quicktime format rendering software which has a significantly more narrow CPI distribution band to a maximum of about 0.4 CPI. The average IPC for these applications is about 0.84 instructions per cycle for the DVD player and 0.25 instructions per cycle for Quicktime. Note that the IPC of the Quicktime player also shows the effects of optimizing media software – compared to a previous analysis run with an older version of the same Quicktime rendering software, the new version has a significantly worse CPI, yet only executes about 1/4 of the number of instructions to render the same video clip.[4]

The moving averages shown in figure 13 show a relatively narrow IPC range for both movie decoding applications. The DVD rendering application has narrow IPC variations in the 0.8 to 0.9 instructions per cycle range, whereas Quicktime shows a stable CPI around 0.25 across the entire 50 second run. Both applications represent only a light system load, with most cycles not spent in the actual application.

Figures 11 and 14 show the same metrics for a first person shooter video game without computer controlled characters, and with 32 computer controlled characters. The IPC distribution is generally below 0.5 with some samples periodically exceeding this range. This behavior is significantly more pronounced in figure 14(b) which points at higher IPC – and variability – in control code responsible for computer controlled characters (such as game AI). This behavior is also reflected in the moving averages, where the game application without computer controlled characters oscillates with a very regular pattern in a narrow (0.3 – 0.35 instructions per cycle) range. The overall moving average IPC is higher for the same application when computer controlled characters are present, which also leads to increased variability and less regular IPC variations. While these oscillations are regular and probably influenced by the regular frame rate pattern, these oscillations are at a much lower frequency and do not directly correspond to single frame updates.

The second video game shown in figures 12(a) and 15(a) shows a range of distributions in the 0 to 0.75 instructions per cycle range, with no sample points outside this IPC range. The moving average IPC is similarly limited to a very narrow range throughout the execution. All the game applications show a very high (up to 100%) utilization of the processing capacity of a single PowerPC 970 core.

The 197.parser benchmark shows a markedly different behavior from the media applications discussed in this work, and is typical of a range of SPEC benchmark applications. Regions with high ILP are interrupted with regions that are characterized by low ILP. In 197.parser, high ILP regions also havre high IPC variability covering the entire IPC range from about 0.25 instructions per cycle to about 1.25 instructions per cycle, whereas the DVD player benchmark has a more bi-modal distribution with dense regions below 0.40 instructions per cycle, and within the 0.9 and 1.0 IPC range. Games show a distribution with most sample points located in a single range, although a number of other sample points exist – the number and randomness of these "outliers"

[4]This might for example occur when a memory bound application is optimized to reduce the number of instructions executed. Fewer instructions over which to apportion the memory-dominated latency remain.

increases with the introduction of computer-controlled characters, pointing to more variability in the associated code.

To analyze frame-level application behavior in media workloads, Figure 3 compares instructions per cycle (IPC) for each of the six workloads for the 31st 1 second interval. IPC for the QuickTime media player shows high, discrete peaks, beginning immediately after a frame transfer, followed by idle time after processing each frame. While similar idle periods can be seen in figure 3(a) for the DVD player, several of the IPC peaks in the DVD data are sustained for longer periods. Variations in the duration of IPC peaks for the DVD player correspond to the three types of frames in an MPEG-2 stream: the I-frames require processing over all the pixels of the frame, hence the wider IPC peaks, while the B- and P-frames only require processing on a subset of the frame's pixels. These variations should occur roughly as a one-in-twelve pattern, i.e., one I-frame out of 12 total frames, and this behavior is represented in the graph.

Video games do not exhibit these idle processing periods, as shown for the execution segment captured in figures 3(c), 3(d) and 3(e). Since video games on desktop PCs typically update the screen at a variable frame rate, tracking the maximum achievable frame rate, no such idle periods (which would normally be used to synchronize to an underlying frame rate) can be observed. Note that this is a difference between desktop PC games and video consoles, where the consoles provide a fixed hardware platform and the game workloads are usually architected to match the fixed frame rate of the TV output signal.

The computation load on the processor is significantly different between traditional media content rendering applications, as exemplified by both the DVD and Quicktime format rendering, and the game workloads. This difference is due to the fact that the content is created in realtime in a game, and not rendered from a static (compressed) source. To this effect, games typically must perform a number of processing steps, such as game AI (the operation of computer controlled characters), collision detection, and game physics to create the video content to be displayed on each frame update.

It should be pointed out that graphs of IPC and other metrics plotted against time (including all cache miss rates and branch misprediction rates) that are calculated as ratios of performance counter values do not represent the entire information about workload behavior, because they represent the ratios without regard to the amplitude of a metric. This can be especially misleading in time slices where few cycles are spent in an application (e.g., because the application is idle most of the time). While the rates are "correct" *per se*, they are likely irrelevant for the program performance due to the small amplitude of the phenomenon, which is not reflected in ratios which assign similar importance to the ratios in all time slices. While longer time slices allow correction for such effects, it will also tend to average out some phase behavior. To correct for this and show overall importance of specific effects, we also show summary information as rates and magnitude in tabular form.

Table 1 gives the overall IPC of for each of the applications.

3.3 Branch Prediction

Figure 4 shows the branch misprediction rates for the media rendering workloads, two configurations of FPS1, FPS2 and the 197.parser SPECInt workload. More detailed misprediction rates are shown in table 2 which breaks down the mispredictions by target address misprediction and misprediction of the branch decision (taken or not-taken) outcome. Like many recently produced games, FPS1 is a C++ program and shows a commensurately higher number of indirect function calls, leading to a higher fraction of target address mispredictions. This is not inherent in the application type, as shown by the second game application, FPS2, which has much lower overall branch misprediction rates.

	address	**decision**	**total**
DVD Player	0.25%	7.99%	8.24%
QuickTime	1.86%	3.93%	5.79%
FPS1 w/0	2.62%	5.33%	7.94%
FPS1 w/32	1.92%	5.91%	7.84%
FPS2	0.73%	3.95%	4.68%
Parser	0.01%	6.04%	6.05%

Table 2: Branch misprediction summary

The contemporary game development industry trend appears to be increasingly toward higher-level languages and more general programming methodologies. With the increasing complexity of both the hardware (consoles and PCs) and the games themselves, developers are finding greater need to employ abstraction layers and structured design methods. These trends toward "big-programming" generally imply an increasing emphasis on aspects typical of C++ code, i.e., indirect function calls, middleware layers, etc.

3.4 Cache Memory Hierarchy

Figures 5, 6 and 7 illustrate the performance of these applications with respect to the memory hierarchy. Aggregate values of these and other performance metrics related to the memory hierarchy are also given in table 3.

Figure 5 shows the L1 data cache misses per instruction completed during one second of the run. Referring to figure 5, we note that all applications have a similar range of rates, ranging no higher than 0.2 L1 misses per instruction.

The idle periods of the QuickTime and DVD player applications are again evident, as there are a significant number of intervals where these applications experienced no misses (and no accesses). The video games have a much more regular stream of accesses to the memory system, which is consistent with the work that they must conduct to generate the frame content that is to be displayed. All applications show a strong, repeating, periodic behavior that is aligned with each application's frame rate.

Figure 6 shows the frequency of L2 data cache misses per completed instruction. The DVD player graph, figure 6(a) is truncated, as the tall spike occurring at approximately 30.035 seconds reaches 0.076 misses per instruction.

Comparing the game workloads of figure 6(c) and figure 6(d) to the two multimedia players, there are some notable differences. First, the DVD and QuickTime players exhibit noticeable periods where there are no L2 data misses, while the game workloads provide a much more consistent L2 miss rate, particularly when the computer is controlling the 32 non-player characters in figure 6(d). This is another expression of the idle periods in the media player workloads, but also illustrates that the game workload has phases (per frame) where the L2 is not strenuously exercised (particu-

	DVD Player	QuickTime	FP Shooter 1 1600x1200		FP Shooter 2	Parser
			0 Actors	32 Actors		
L2 Dcache Loads	$1.35 * 10^7$	$1.01 * 10^7$	$17.4 * 10^7$	$26.6 * 10^7$	$32.1 * 10^7$	$68.2 * 10^7$
L2 Dcache Stores	$32.9 * 10^7$	$6.92 * 10^7$	$72.9 * 10^7$	$130. * 10^7$	$150. * 10^7$	$243. * 10^7$
L2 Dcache Load Misses	$1.51 * 10^6$	$2.85 * 10^6$	$33.7 * 10^6$	$52.8 * 10^6$	$90.9 * 10^6$	$84.9 * 10^6$
L2 Dcache Store Misses	$35.8 * 10^5$	$3.88 * 10^5$	$108. * 10^5$	$195. * 10^5$	$403. * 10^5$	$54.5 * 10^5$
L2 Load Misses/1000 Instr	0.24	2.87	3.05	2.78	3.73	1.64
L2 Store Misses/1000 Instr	0.56	0.39	0.97	1.03	1.65	0.11
L2 Data Misses/1000 Instr	0.80	3.26	4.02	3.81	5.38	1.75

Table 3: Memory hierarchy performance metrics.

larly note the periodic zero-misses per instruction regions in figure 6(c) as in relation to the periodic spikes).

Another means of comparing the L2 data cache miss behaviors is by utilizing a moving-average, where the data from a 200-millisecond sliding window is used. Figures 16 through 18 present plots of this moving average of the L2 data cache miss rate per 1000 instructions for the full 50-second sampling run.

Looking at figure 17(a) which illustrates the L2 data cache miss rate for the FPS1 first person shooter video game in which the computer is not controlling non-player characters, there is an interesting semi-periodic behavior, with short periods of higher miss rates followed by lower miss rates, in what looks something like a square-wave pattern. Comparing this plot to figure 14(a) which plots the instructions per cycle metric using the same moving average, one notes that the patterns appear to be well harmonized.

Note, however, when comparing figure 17(b) to figure 17(a) this semi-periodic behavior is no longer apparent. This same difference is apparent in comparing figures 14(a) and (b). In both cases, the primary difference in the workloads is the addition of 32 computer-controlled non-player characters. As has been noted earlier, the addition of computer controlled characters adds significant work per frame, much of which is not as regular as in the 0-actors case. Comparing figures 14(a) and (b) one again sees that the IPC variation for the 32-actors case is much more pronounced than for the 0-actors case. A higher variability in the L2 data cache miss rate would certainly contribute to a higher variability in overall measured IPC, and thus the correlation between figure 14(a) and figure 17(a), and similarly between figures 14(b) and 17(b) is expected and reasonable.

One other notable effect of adding the 32 computer controlled actors is a very large reduction in the frame rate sustained by the game workload – from approximately 30 frames per second to roughly 13 frames per second. At 30 frames per second, each frame requires 33.3 milliseconds, and a 200 millisecond sample window should contain almost exactly six frames. At the lower 13 frames per second rate, however, each frame requires about 77 milliseconds, and a 200 millisecond window would contain slightly less than 3 frames. This might distort the periodicity of the application.

Figure 7 presents the rate of L2 instruction misses, i.e., misses in the (unified) L2 that result from instruction requests. Again, the DVD application has a single data spike, at approximately 30.850, which exceeds the scale (the value of that data point is actually 0.25, which far exceeds all other data in these graphs).

Comparing figure 7 to figure 6 one must observe that the scale of the instruction L2 misses per instruction rate is at least the equal of the data miss rate. In fact, comparing the DVD player graphs of figure 7(a) and figure 6(a) we find that the DVD player suffers many more instruction L2 misses than data L2 misses, often three to five times as many. The Quicktime player, on the other hand, shows much more parity between the instruction and data misses.

The game application is somewhat more variable in its behavior, but generally tends to have a higher rate of data L2 misses than instruction L2 misses per instruction. This is particularly evident in the case where there are no computer-controlled actors in the game, and thus the game is simply generating frame content for display, and shipping that content to the AGP port. When 32 computer-controlled actors are added, the data L2 misses per instruction drops somewhat (comparing figure 6(c) to figure 6(d)) leading one to conclude that the character-control code has better cache behavior, and that this, coupled with the dilation of the frame time (evidenced by the drop in the frame rate from near 30 to nearer 13 frames per second), leads to less pressure on the L2 cache in a fixed time interval.

Referring to figure 6 and figure 7, we note that the L2 data cache miss per instruction rates show very high peaks (up to 60 L2 misses per 1000 completed instructions), but looking at the aggregate data in Table 3 the overall rates are not nearly as alarming as these peaks. Note that the game workloads generally have a higher L2 miss rate per thousand instructions than the media applications (and the SPEC workload).

With larger game worlds, larger code footprints, etc., in future games, it is likely that the overall memory pressure will increase over time. It is not clear, however, whether this memory pressure will increase at a rate faster than the general increase in the L2 cache size for future processors. This may be a concern for future console systems, as console hardware is less frequently updated (refreshed) and attempts a larger leap forward in performance at each new console introduction; missing the L2 balance could have long-lasting repercussions even when the underlying processor and hardware computational power are competitive.

3.5 Computer-Controlled Characters

Another interesting question is how the game applications are affected by the various tasks they are required to conduct, i.e., the content generation compared to the content rendering and display. In the game application runs of figures 2 through 6, there are two different runs of FP Shooter 1. The first run was taken where the game application ran with a single camera (observer) alone in the world; this primarily required the game application to render the

background world. The second run was taken when the computer controlled 32 additional non-player actors ("bots") within the game world. In this case, the game application needs not only to do all the work of the 0-bots case, but also to execute the artificial intelligence (planning) for each bot, and calculate additional physics for both the bots and any other objects that they interact with (e.g., projectiles and explosion particles, etc.).

Comparing the graphs of these runs, we note that the additional processing required for the 32 bots has significantly degraded the overall frame rate (from the approx. 30 frames per second of Figure 2(c) to the approx. 13 fps shown in the Figure 2(d)). Furthermore, the volume of data transferred across the AGP bus has increased significantly (presumably for the rendering of the bots, projectiles, explosion particle effects, etc.) and there is an additional sub-peak per frame in the 32-bots case. Similarly, the IPC graphs of figure 3 show significantly more peaking per frame, and much more irregularity in the overall profile, when the 32 additional bots are included.

Interestingly, the addition of 32 bots to the game has not increased the L1 Dcache misses per instruction, as shown in figure 5. Furthermore, the L2 Dcache misses per instruction of figure 6 indicate lower peaks, and the data of Table 3 show that the L2 misses per instruction actually decreased. This seems counter-intuitive, but is at least partially explained by the dilation of the frame time corresponding to the reduced overall frames-per-second rate. The 32-actor run does have a larger number of total misses, but apparently the actor control code provides a larger proportion of instructions that have good cache behavior, and thus the overall miss rate is lower. For the AI code, this could be understood, at least for the L2, because the data structures used in the computation of the planning could readily fit within an L1 cache (at least per bot) and may be referenced many times.

3.6 Phase Behavior

The Fast discrete Fourier Transform (FFT) is widely used for performing frequency analysis of time-varying signals. Derived from an efficient implementation of the well-known Fourier Series, strong peaks in the graph of the FFT magnitude (or alternatively, the magnitude spectrum) of a waveform both identify periodic behavior and quantify the periods (or inversely, the frequencies) of its strong frequency components.

Figures 8 and 9 show plots of the magnitude spectrum of a 1024-point FFT of the AGP memory bandwidth and the IPC, respectively, for the traditional multimedia applications (DVD player and QuickTime), video games (FPS1 with 0 bots and with 32 bots, and FPS2) and an application from the SPEC CPU 2000 benchmark suite (197.parser). Prominent peaks are discernable in all graphs in figures 8 and 9 that pertain to video game and multimedia applications, but not in the 197.parser. The plot of the magnitude spectrum of AGP memory bandwidth for DVD player in figure 8(a), for example, has clearly visible peaks at 24.4Hz, as well as 48.8Hz, 73.2Hz and all other integer multiples of 24.4Hz shown in the figure.

As discussed in section 2.3, 24Hz is one of the common frame rates used to encode video with the MPEG-2 algorithm and clearly shows up[5] in figure 8(a) as the lowest significant frequency component (hereafter referred as the *fundamental frequency*). The integer multiples of the fundamental frequency are resonances due to sharp cut-offs in the time-varying AGP bandwidth visible in figure 2 (a). Further discussion of resonant frequencies is beyond the scope of this paper and is available in a variety of classic texts on signal processing [8]. We limit the discussion of resonant frequencies in the time-varying behavior of performance metrics to the fundamental frequency (i.e., the first peak in each graph of the magnitude spectrum) in this paper, as we have found that it matches the frame rate for video game and multimedia applications in almost all cases.

Table 4 gives the fundamental frequency of the AGP memory bandwidth and the IPC for each of the six applications taken from an inspection of the FFT magnitude spectrum. The time-varying AGP memory bandwidth and IPC graphs for parser do not exhibit any apparent periodicity (at least not in the 1-second periods for which the analysis was done) and thus no discernable fundamental frequency in the FFT. For DVD player, QuickTime and Game 1 with 32 actors, the fundamental frequencies for the AGP memory bandwidth and the IPC match exactly. The fundamental frequencies for DVD player and QuickTime match each other exactly for AGP memory bandwidth and IPC. They also match their documented frame rates of 30Hz and 15Hz, respectively. The fundamental frequencies for AGP memory bandwidth and IPC are also equal for Game 1 with 32 actors. The frame rates for video games on desktop PCs, however, are not nearly as well documented nor consistent as with traditional multimedia applications. In fact, for video games on desktop PCs, the frame rate varies with time and, with respect to video games, we hereafter refer to the frame rate as the *instantaneous frame rate* since it changes as the application runs. The instantaneous frame rate for video games can only be measured by observation, i.e. several of the video game applications in our suite provide an option for displaying the frame rate while the application runs. Observations of the instantaneous frame rate for video games were consistently close to the fundamental frequency in the AGP memory bandwidth and IPC.

It is interesting to point out the strong influence of the frame rate on phase behavior in multimedia applications because the frame rate is a characteristic of the application's high-level behavior, and must be consistent across variations in software and hardware. Phase behavior in SPEC benchmark applications is likely due to program structure and the underlying hardware and, as such, specific to the implementation. For video games on desktop PCs, the analysis is complicated further: the frame rate is still a characteristic of the application's high-level behavior, but it is affected at a higher level by the resources allocated to the application by the operating system which varies with time.

4. SUMMARY AND CONCLUSIONS

Computer-based consumer entertainment devices have become a major fraction of the CPU market, and an increasingly important segment of the (cost constrained) high-performance CPU market. We have analyzed and compared traditional media applications such as DVD and Quicktime

[5]With a 1024-pt FFT, only 512 equally-spaced, discrete frequencies between 0 and 500Hz (the frequency range shown in the figure) are detectable. This is why a peak at 24.4Hz, and not 24Hz exactly, is detected in the FFT, i.e., it is the nearest to 24Hz in the *discrete* set of frequencies.

	DVD Player	QuickTime	Game 1 (0 Actors)	Game 1 (32 Actors)	Game 2	Parser
AGP	24.4	15.625	27.344	12.69	27.344(????**mistake**????)	N/A
IPC	24.4	15.625	32.226	12.69	31.25	N/A

Table 4: Fundamental frequencies for AGP memory bandwidth and IPC.

rendering, and emerging media applications such as video games.

While many of today's pervasive consumer applications (such as digital audio or video) exhibit lower compute performance requirements, video games and other digital content creation applications will demand significantly higher computer performance.[6]

The phase behavior in both game workloads and more traditional multimedia players is dominated by the frame rate intervals and associated processing. Phase behavior across larger time intervals was not discernable for any of the explored media applications.

It is interesting to note that these measurements were taken on a high-end desktop PC system, with performance specifications well ahead of the current game consoles. Nevertheless, the game workloads were unable to generate a consistent 30 fps frame rate even at lower resolution than high-definition TV standards. Considering that game consoles will likely target the HDTV marketplace in the near future, one could argue that such consoles will require much more computing power (performance) than current consoles, and perhaps even than current desktop PCs. A promising way to supply this increased computing power at acceptable hardware complexity and power cost will be the adoption of a chip multiprocessing (CMP) approach.

Acknowledgments

The authors would like to thank Jack Kouloheris, Erik Altman, Alex Mericas, Ed Tam, Eric Miller, Nathan Slingerland, and Myke Smith for their help over the course of this work.

References

[1] D. Citron. MisSPECulation: partial and misleading use of SPEC CPU2000 in computer architecture conferences. In *Proceedings of the 30th International Symposium on Computer Architecture*, pages 52–61, San Diego, CA, June 2003.

[2] E. Duesterwald, C. Cascaval, and S. Dwarkadas. Characterizing and predicting program behavior and its variability. In *Parallel Architectures and Compilation Techniques (PACT-2003)*, pages 220 – 231, Sept/Oct 2003.

[3] Entertainment Software Association. http://www.theesa.com/pressroom.html.

[4] W. Hassanein, G. Astfalk, and R. Eigenmann. 1D performance analysis and tracing of technical and Java applications on the Itanium2 processor. In *Proceedings of the International Symposium on Systems and Software*, pages 91–100, March 2003.

[5] C. Hughes, P. Kaul, S. Adve, R. Jain, C. Park, and J. Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *Proceedings of the 28th annual international symposium on Computer architecture*, pages 254–265, July 2001.

[6] F. Levine and C. Roth. A programmer's view of performance monitoring in the PowerPC microprocessor. *IBM Journal of Research and Development*, 41(3):345–356, May 1997.

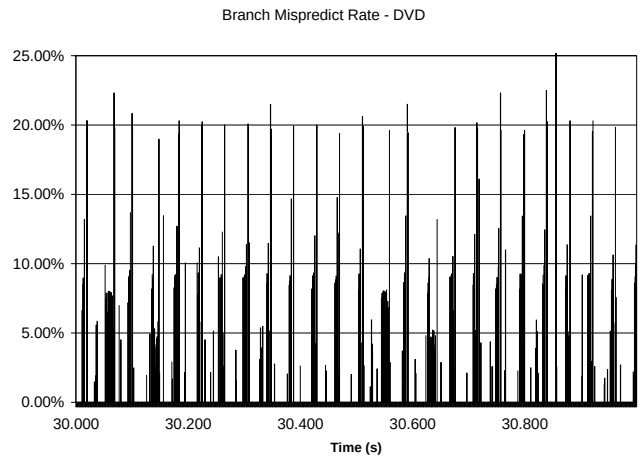
[7] B. Matthews, M. Gschwind, and J.-D. Wellman. Characterizing video game workloads: A sample-based methodology for architectural analysis. Technical report, IBM TJ Watson Research Center, 2004. (in preparation).

[8] A. Oppenheim and A. Willsky. *Signals and Systems, 2nd Edition*. Prentice hall, Upper Saddle River, NJ, 1997.

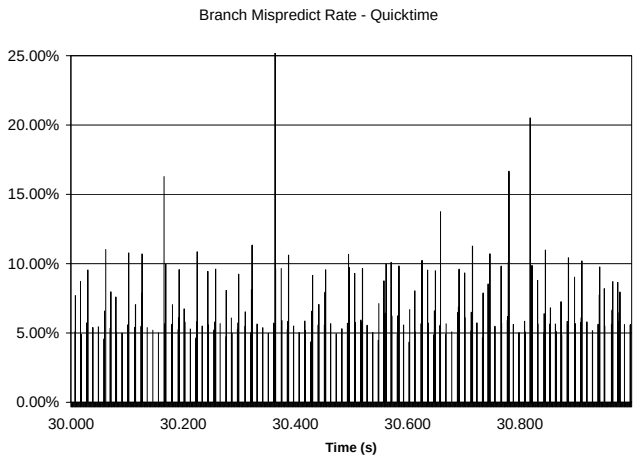
[9] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *Proc. of the 30th Int. Symp. on Computer Architecture*, June 2003.

[10] N. Slingerland and A. J. Smith. Cache performance for multimedia applications. In *Proceedings of the 15th International Conference on Supercomputing*, June 2001.

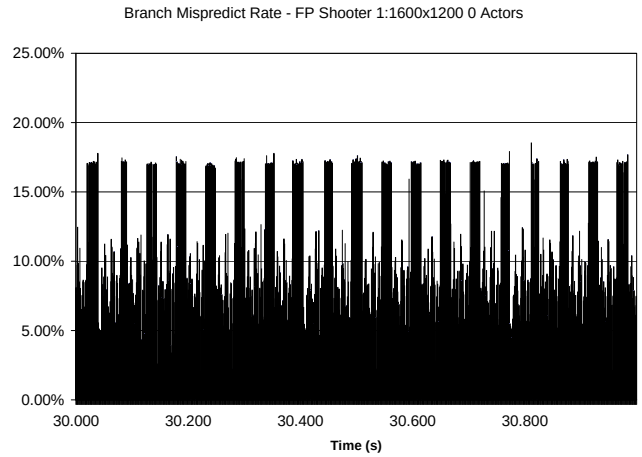
[6] By design, today's video rendering can be accomplished at relatively low instruction counts, whereas content creation requires higher processing power to accomplish multiple steps such as game AI, collision detection, game physics, and so forth. A future, converged home entertainment platform would make larger amounts of processing power available for more aggressive compression, allowing to store more or higher quality content in a given amount of storage, or to deliver it over lower bandwidth links.



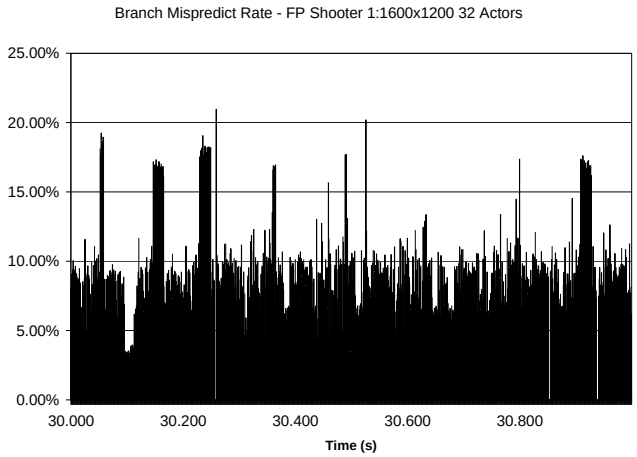
(a) DVD Player



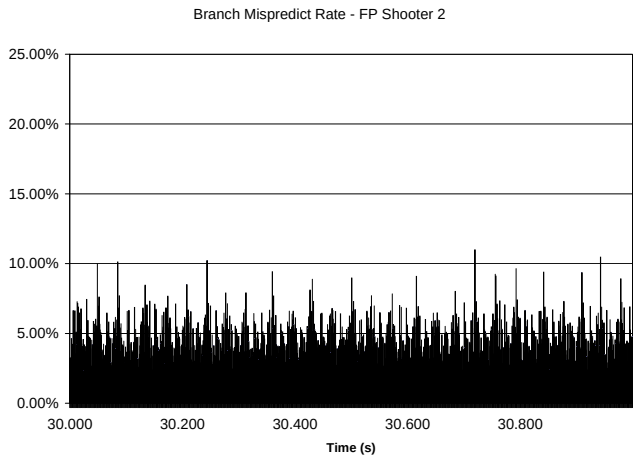
(b) QuickTime



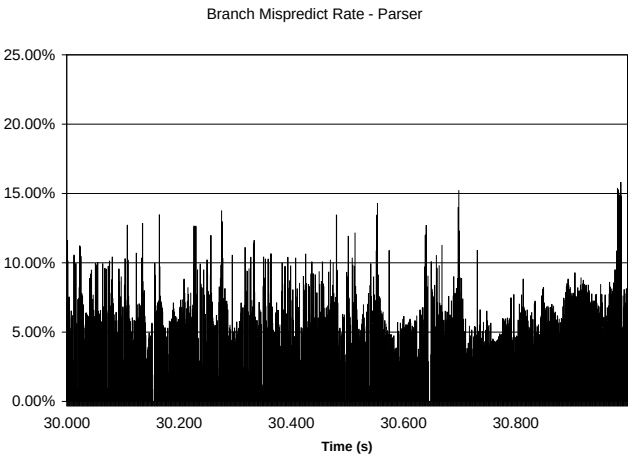
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

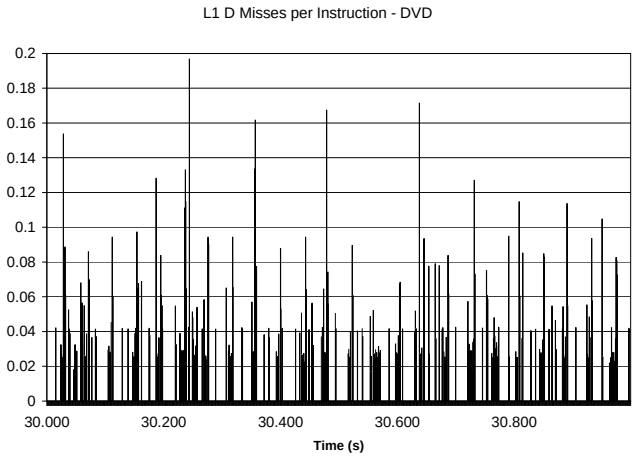


(e) Video Game 2

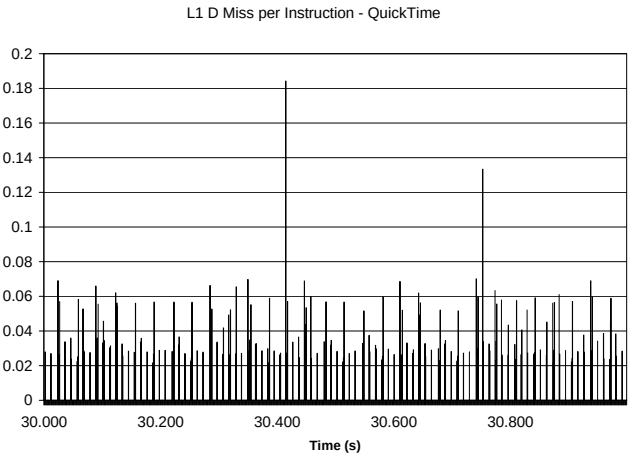


(f) Parser

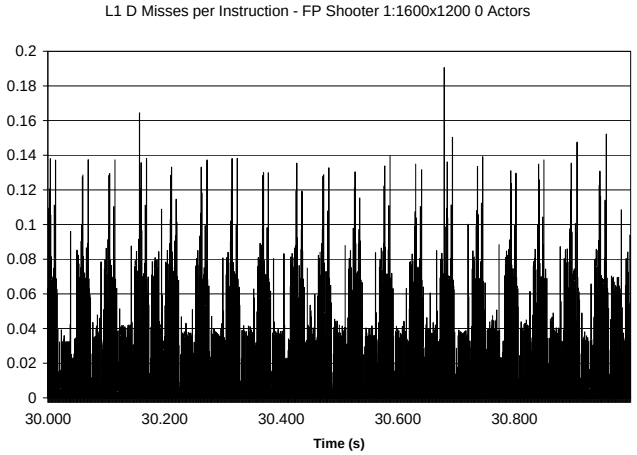
Figure 4: Branch misprediction rate for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.



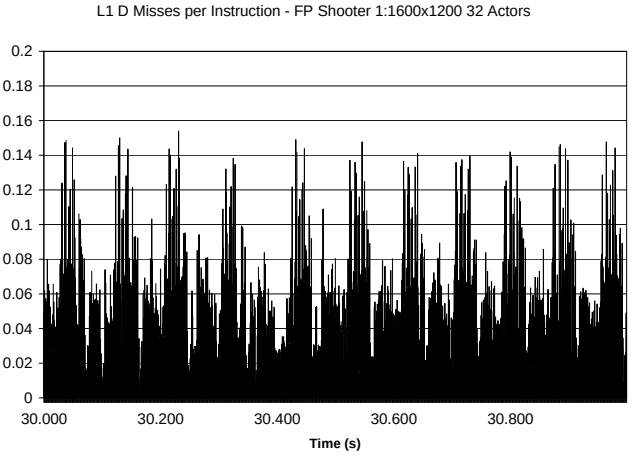
(a) DVD Player



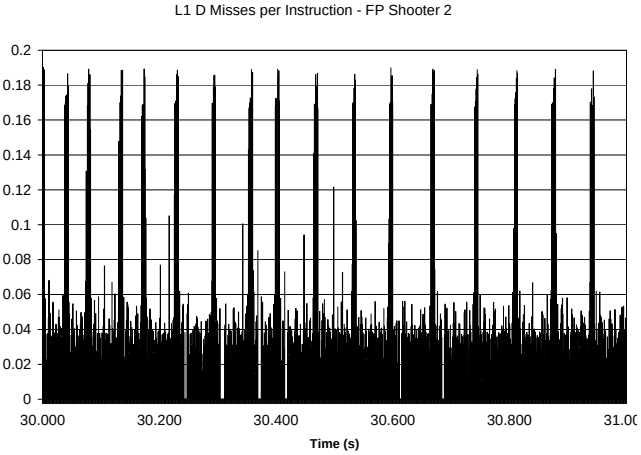
(b) QuickTime



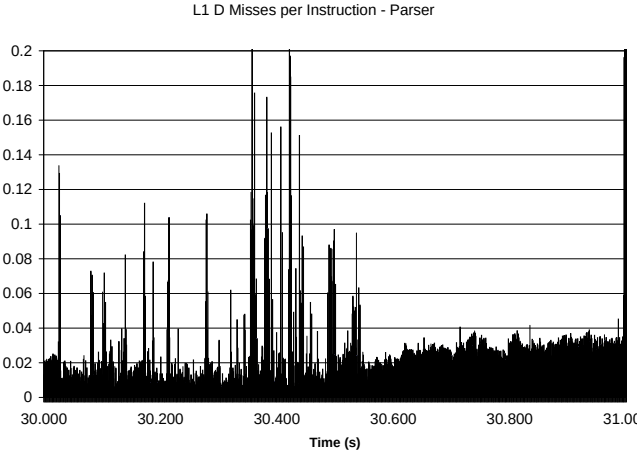
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

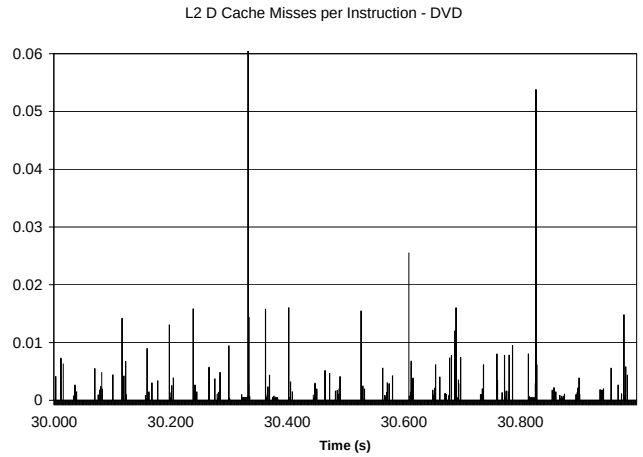


(e) Video Game 2

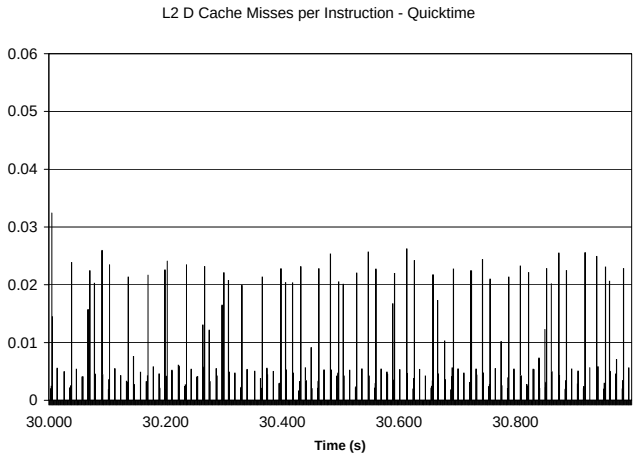


(f) Parser

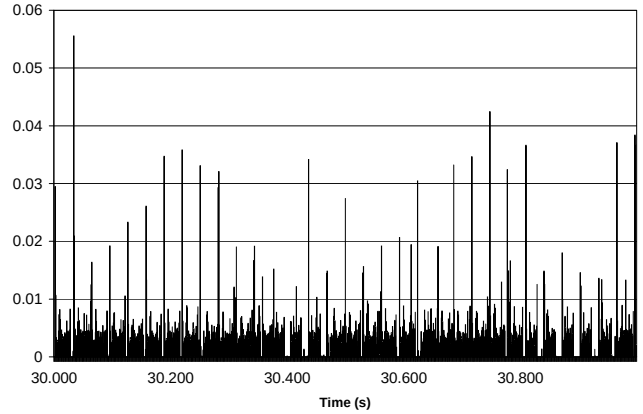
Figure 5: L1 D-Cache Misses per Instruction for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.



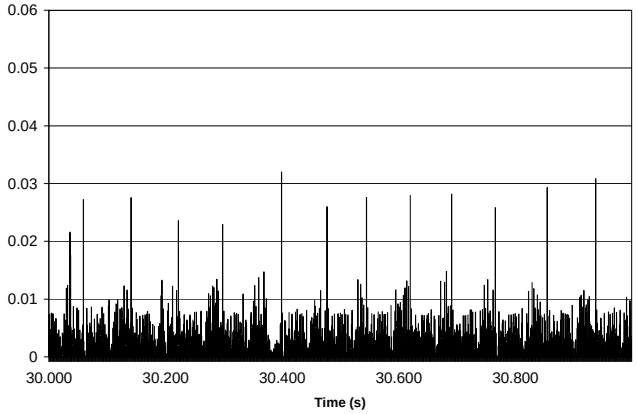
(a) DVD Player



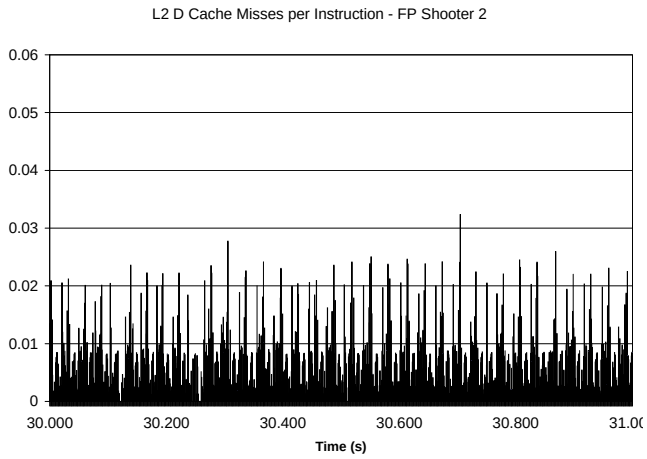
(b) QuickTime



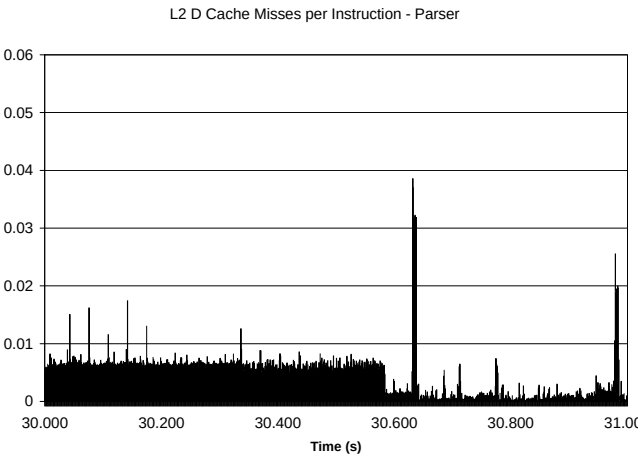
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

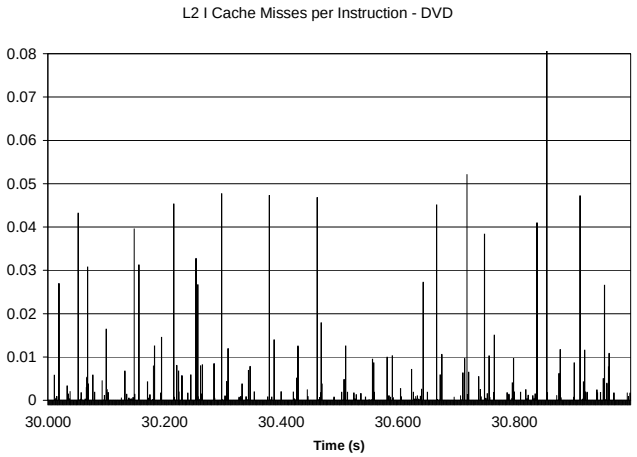


(e) Video Game 2

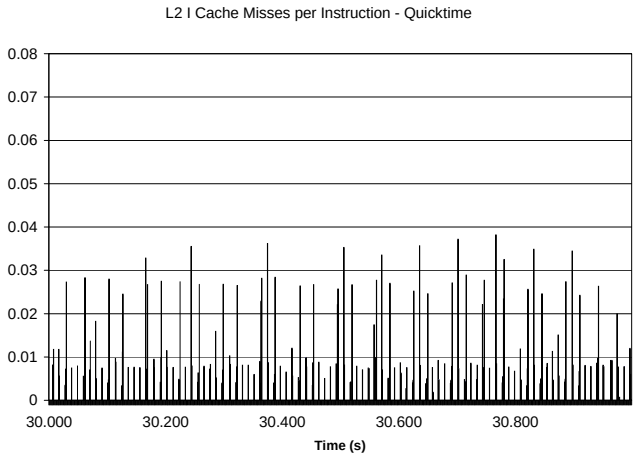


(f) Parser

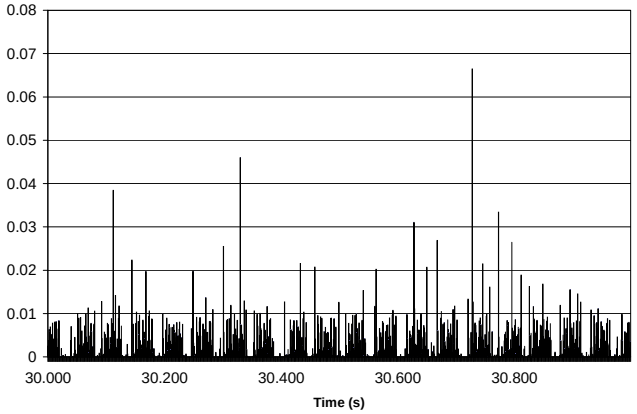
Figure 6: L2 D-Cache Misses per Instruction for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.



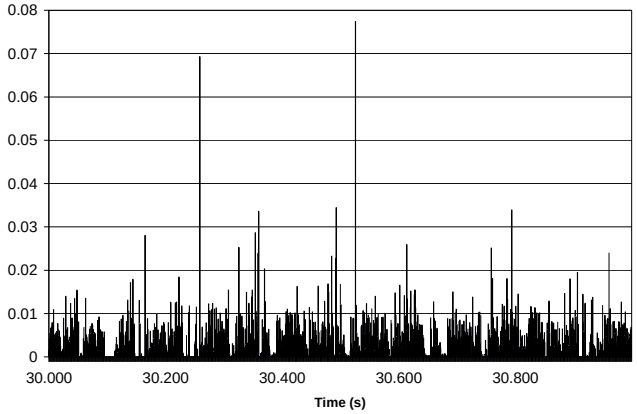
(a) DVD Player



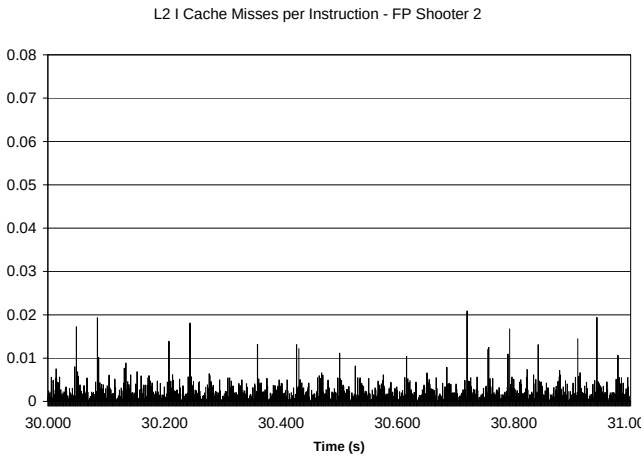
(b) QuickTime



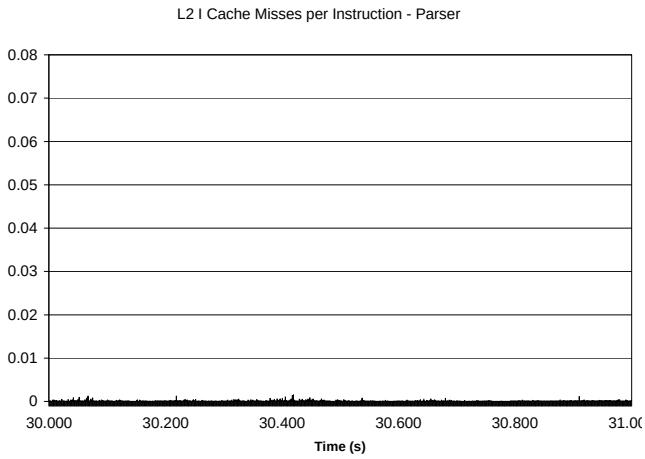
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

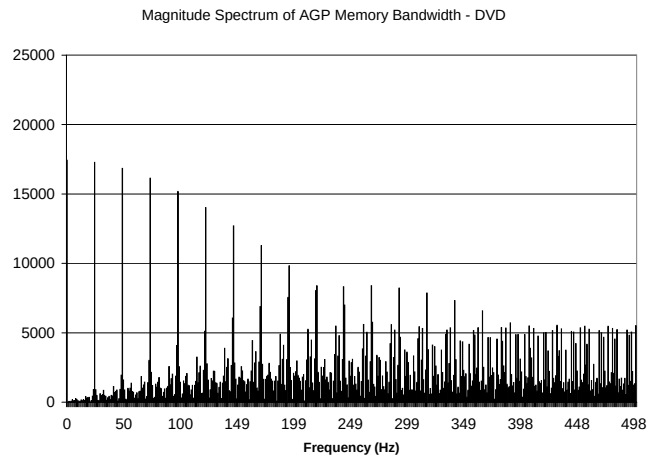


(e) Video Game 2

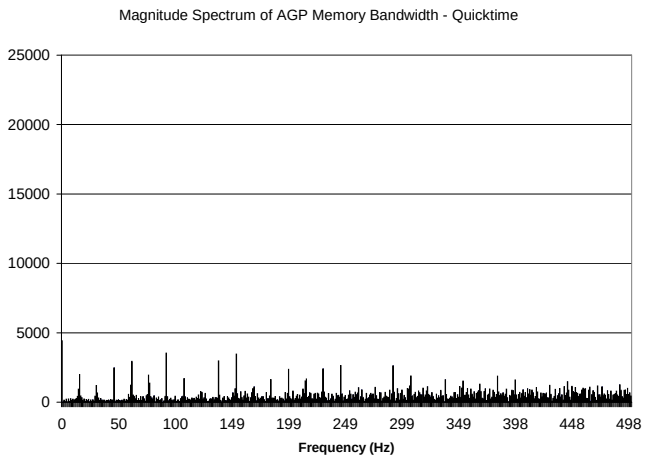


(f) Parser

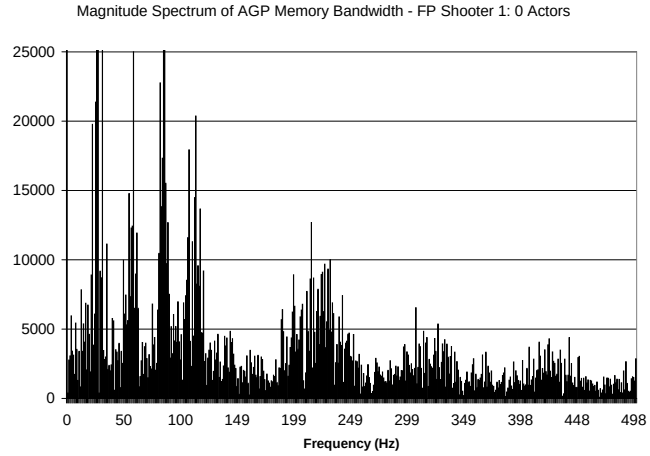
Figure 7: L2 I-Cache Misses per Instruction completed for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.



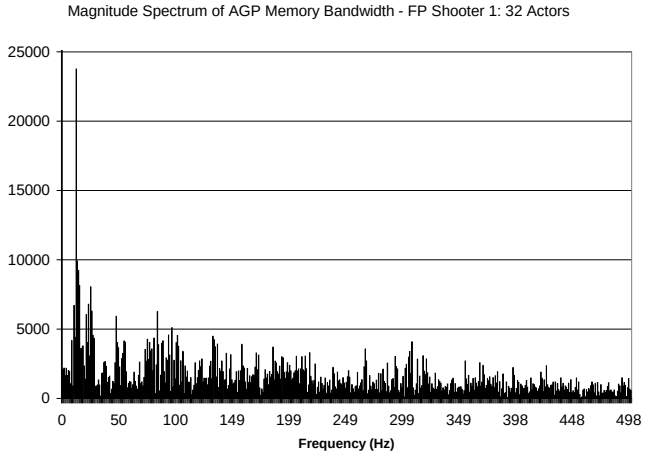
(a) DVD Player



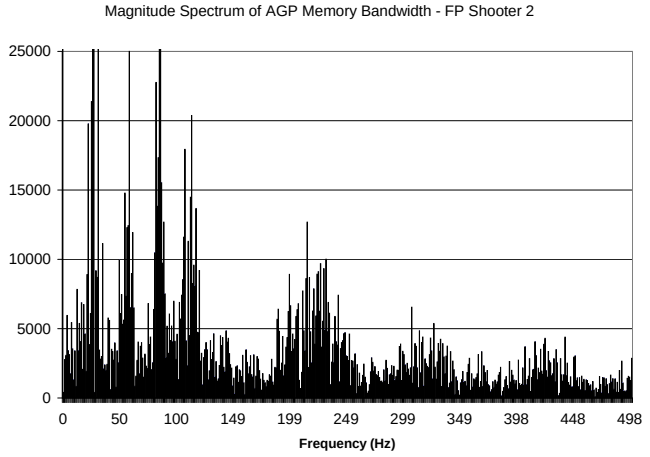
(b) QuickTime



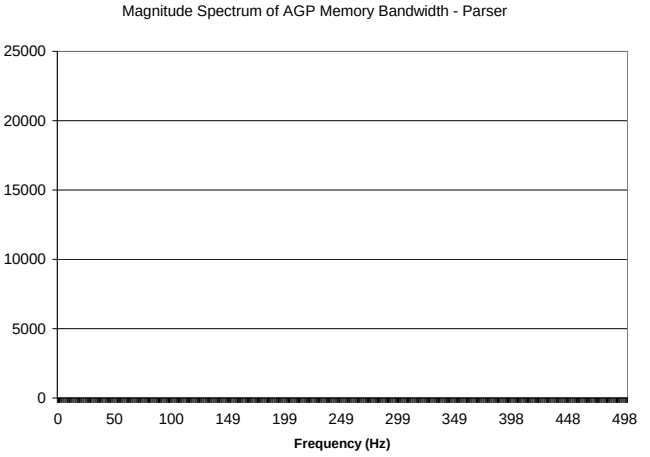
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

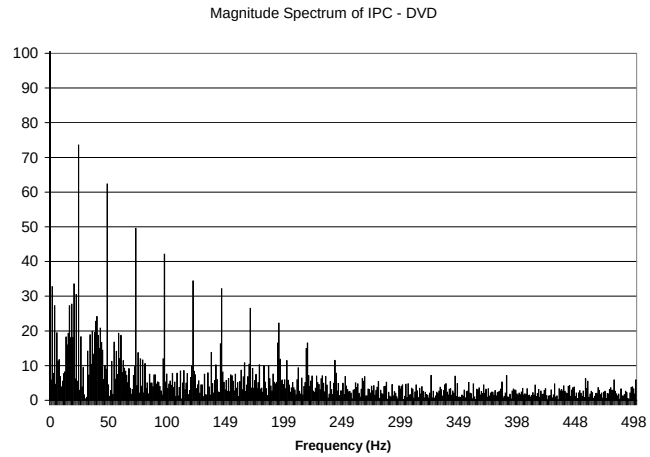


(e) Video Game 2

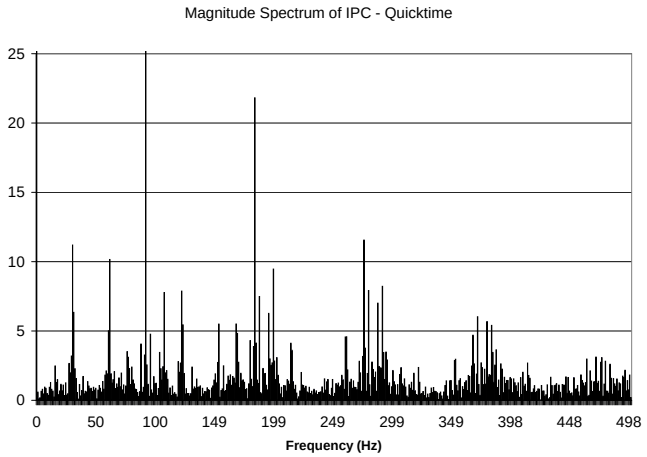


(f) Parser

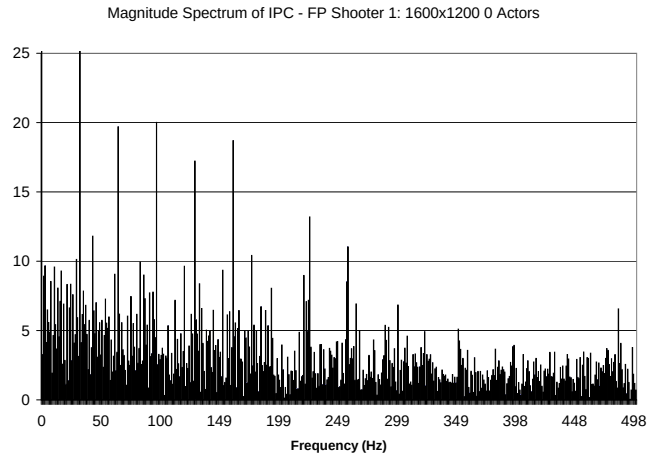
Figure 8: FFT Magnitude Spectrum of the AGP memory BW graphs in figure 2 for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.



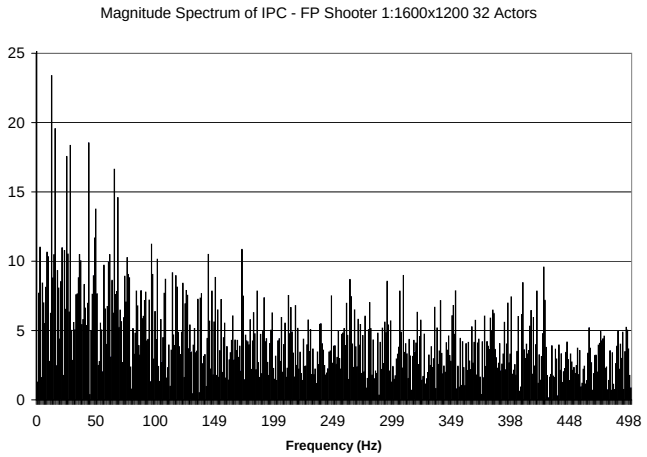
(a) DVD Player



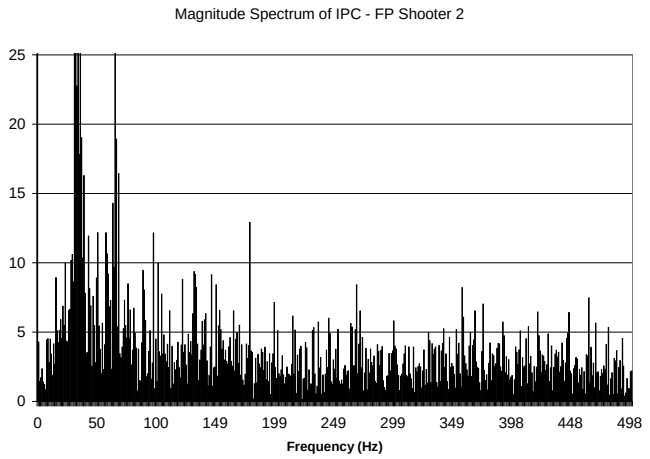
(b) QuickTime



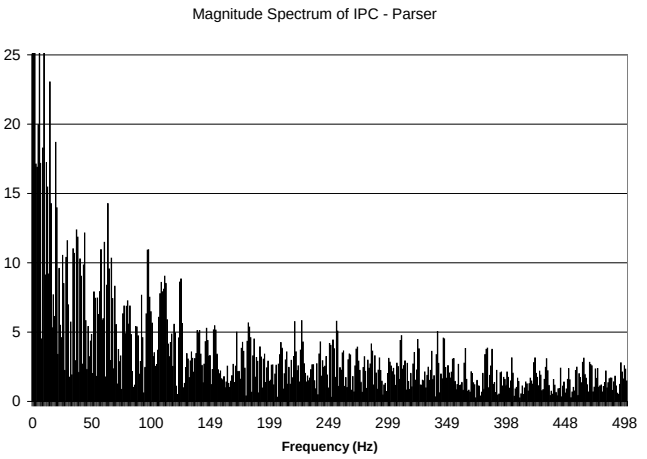
(c) Video Game with 0 Actors



(d) Video Game with 32 Actors

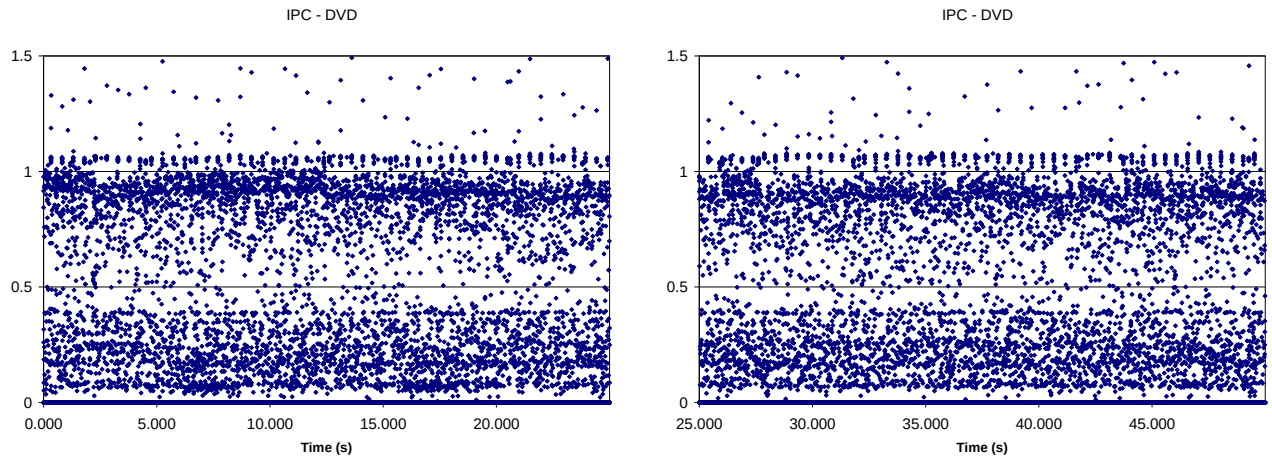


(e) Video Game 2

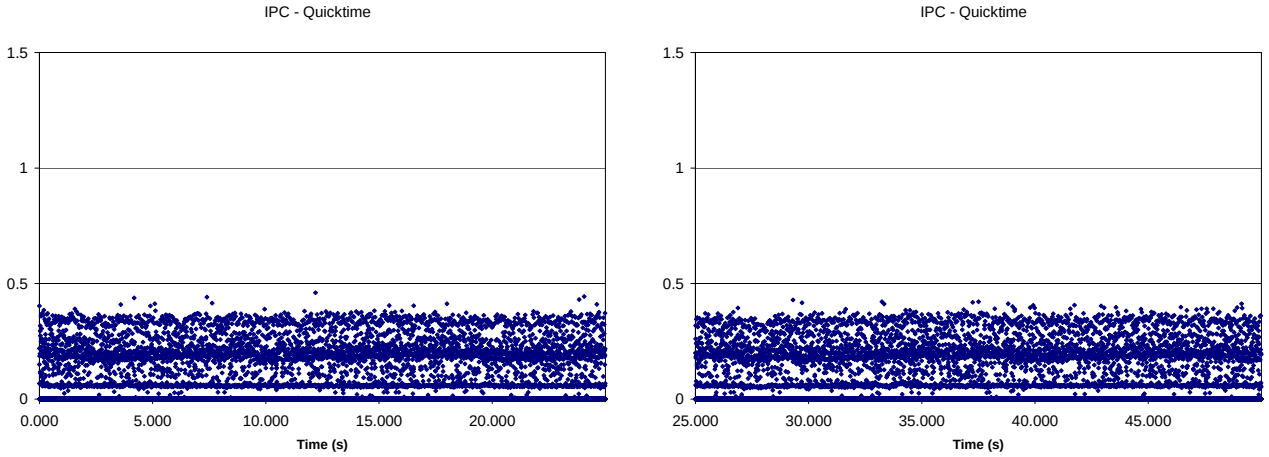


(f) Parser

Figure 9: FFT Magnitude Spectrum of the IPC graphs in figure 3 for (a) DVD Player, (b) QuickTime and a first person shooter video game with (c) 0 on-screen characters and (d) 32 on-screen characters, (e) a second first person shooter video game, and (f) a run of the SPECcpu2000 benchmark 197.parser.

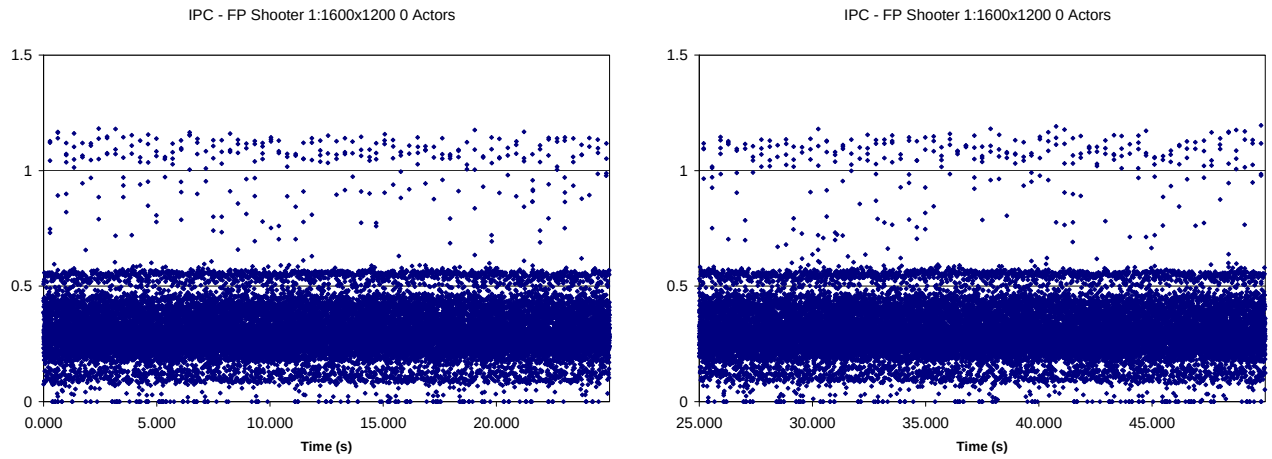


(a) DVD Player

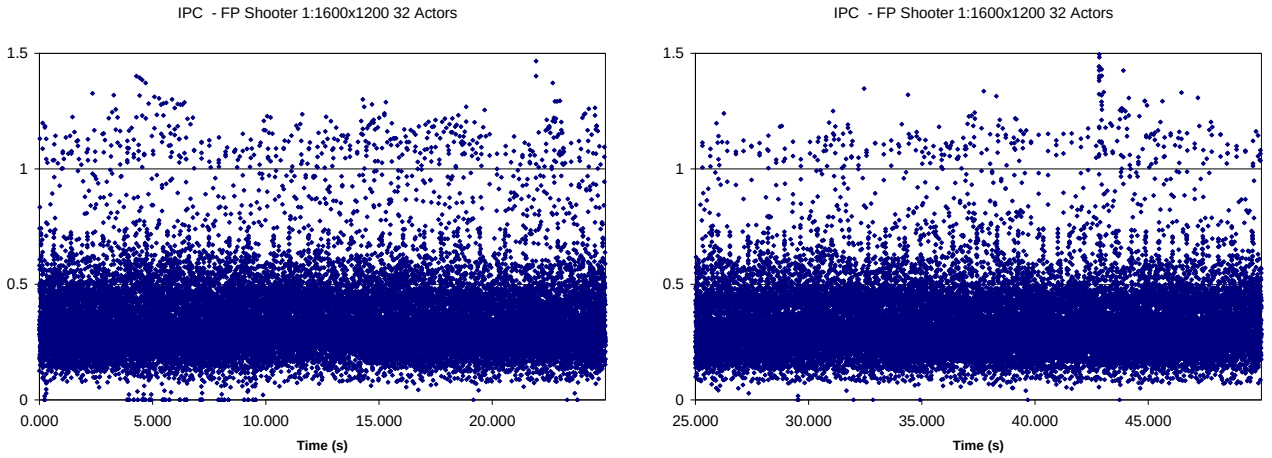


(b) Quicktime

Figure 10: IPC scatter plots for 50 seconds of execution for the (a) DVD Player and (b) Quicktime applications.

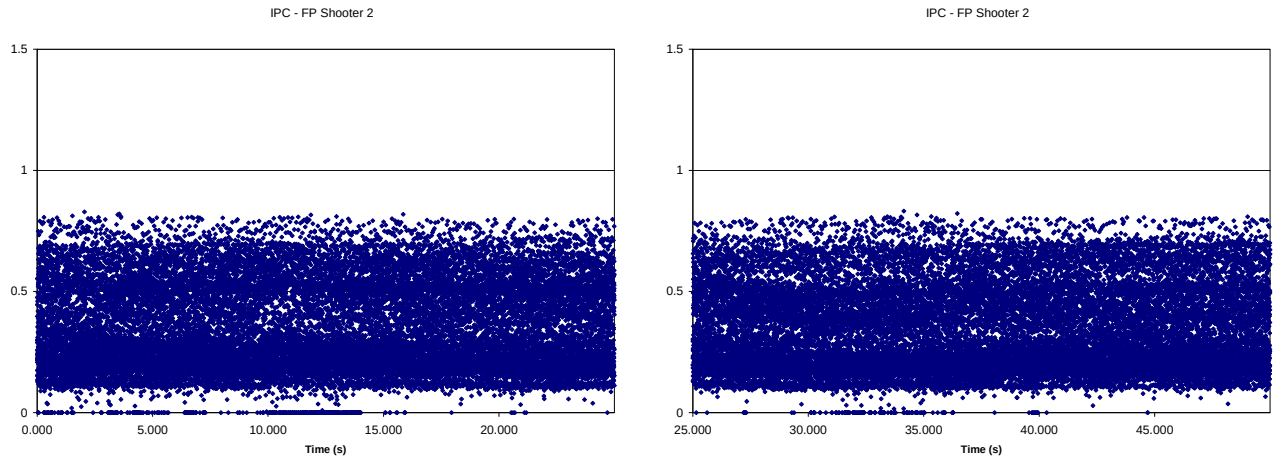


(a) Video Game with 0 Actors

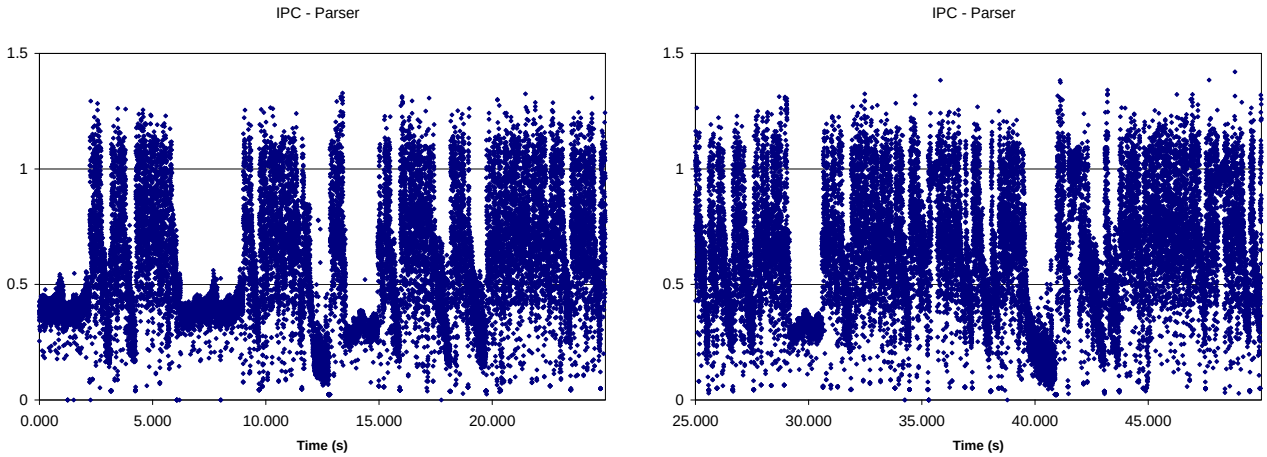


(b) Video Game with 32 Actors

Figure 11: IPC scatter plots for 50 seconds of execution for a video game with (a) 0 computer-controlled characters and (b) 32 computer-controlled characters.

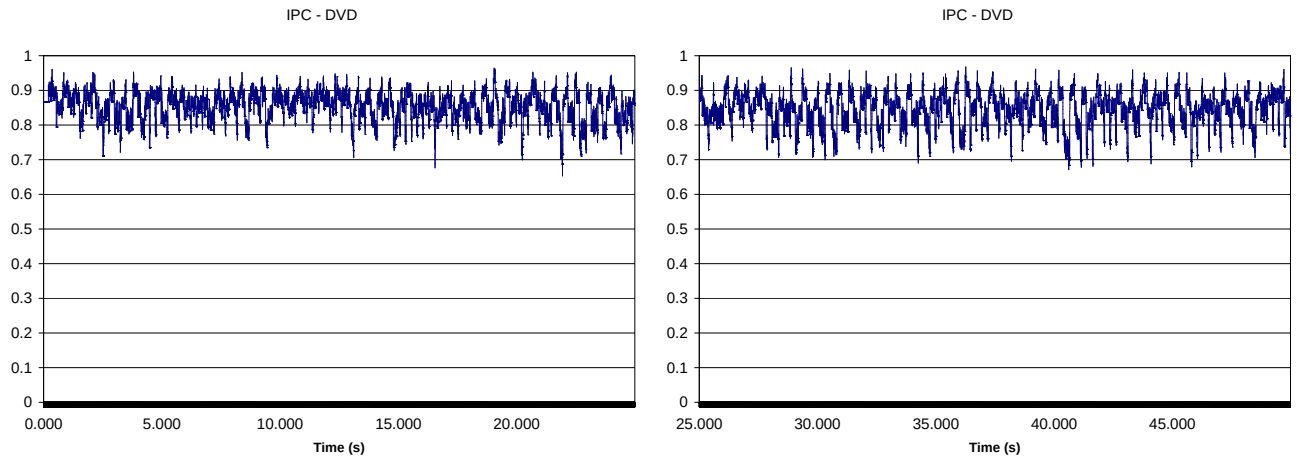


(a) Video Game 2

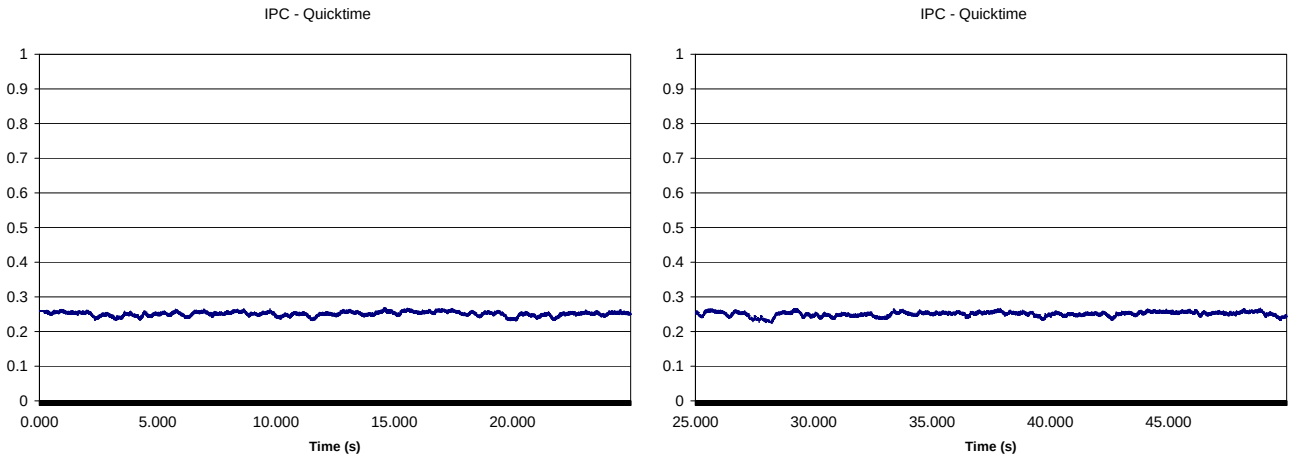


(b) Parser

Figure 12: IPC scatter plots for 50 seconds of execution for (a) Video Game 2 and (b) Parser.

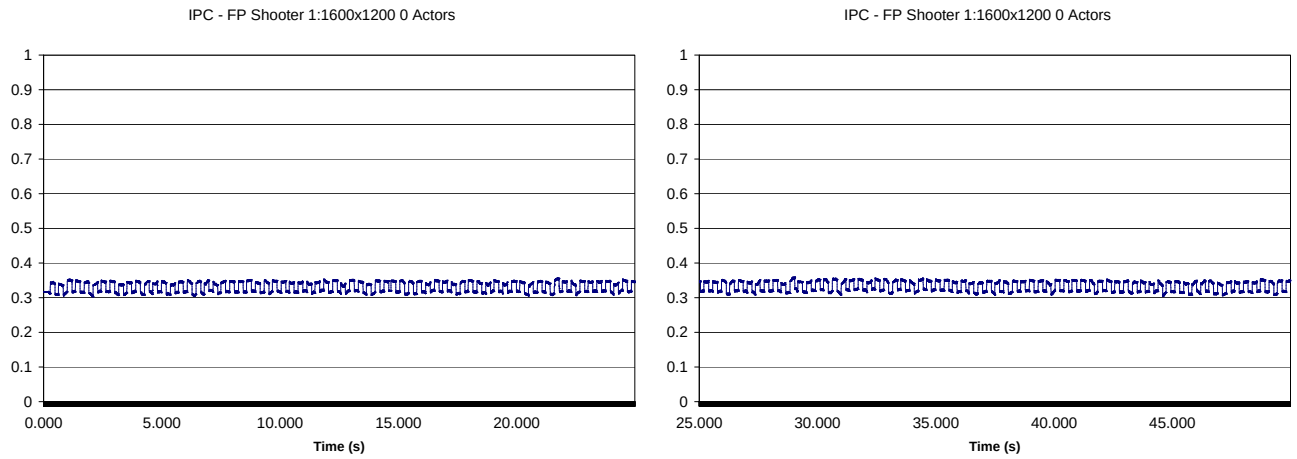


(a) DVD Player

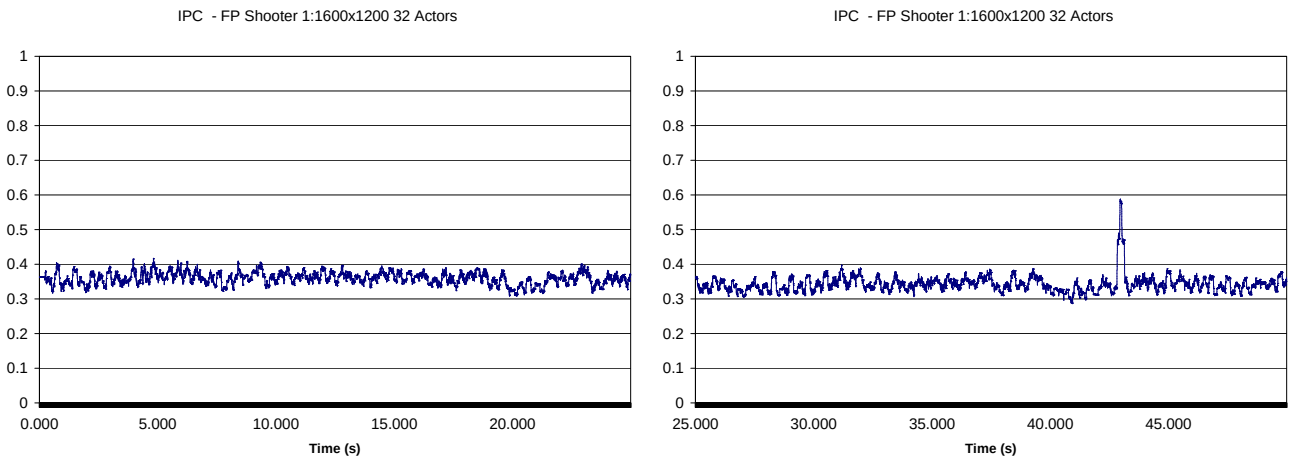


(b) Quicktime

Figure 13: IPC (moving average) plots for 50 seconds of execution for the (a) DVD Player and (b) Quicktime applications.

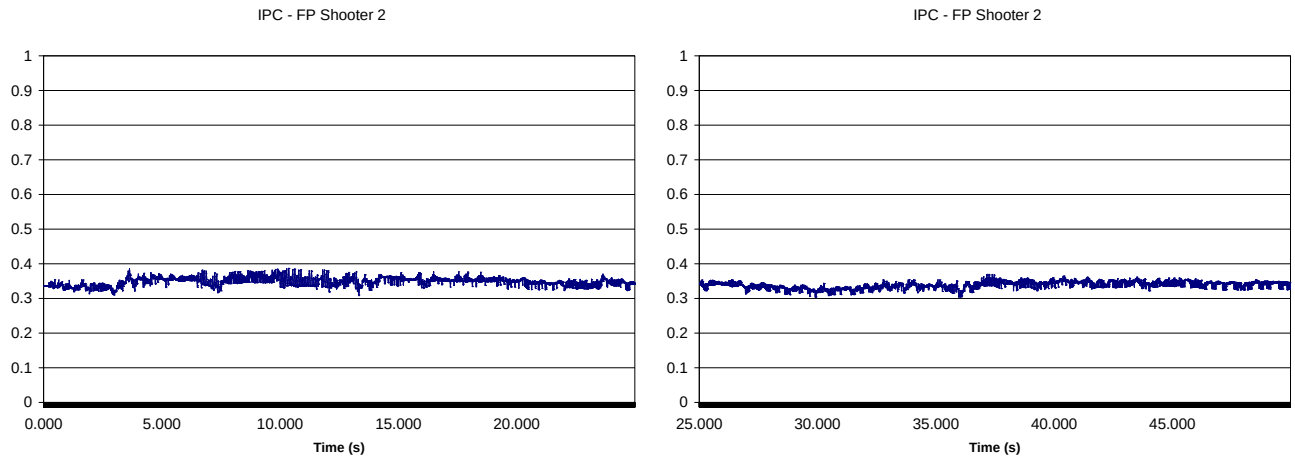


(a) Video Game with 0 Actors

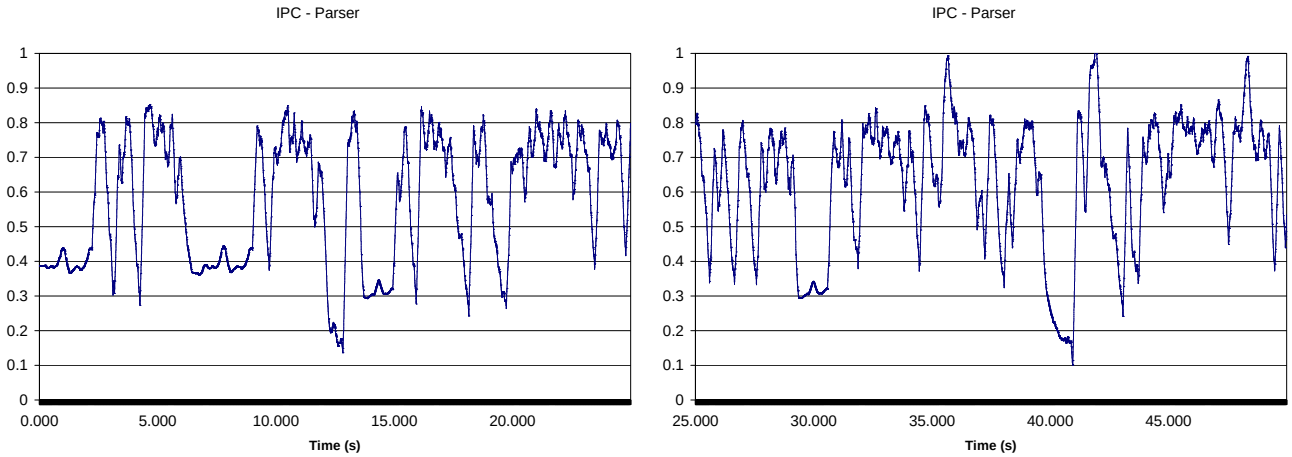


(b) Video Game with 32 Actors

Figure 14: IPC (moving average) plots for 50 seconds of execution for a video game with (a) 0 computer-controlled characters and (b) 32 computer-controlled characters.

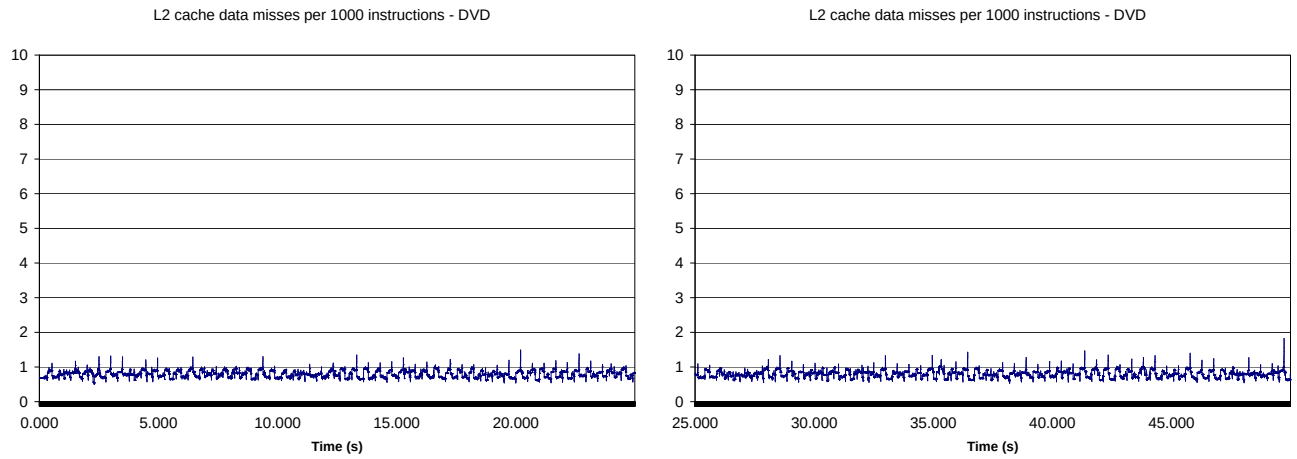


(a) Video Game 2

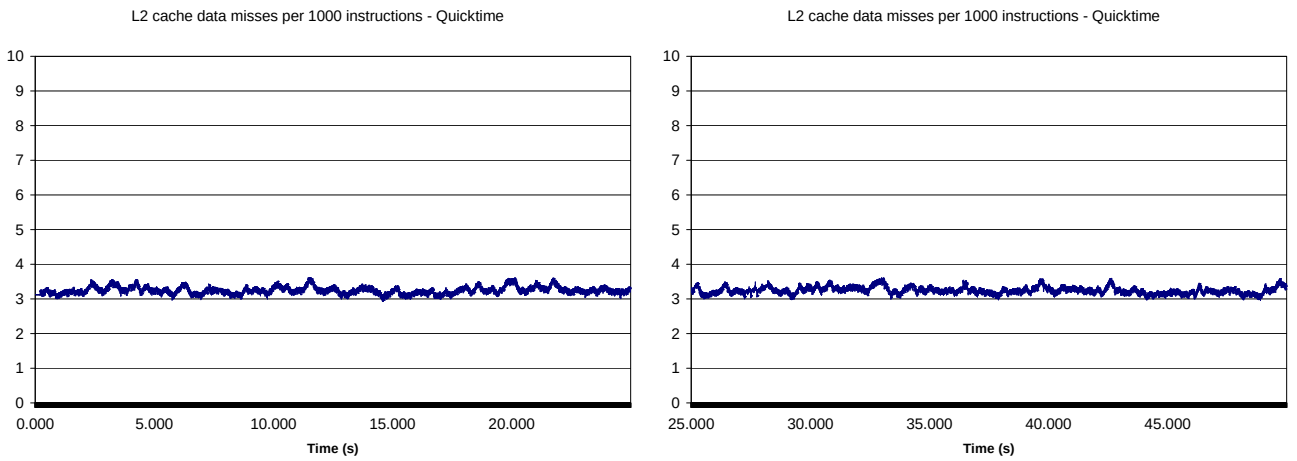


(b) Parser

Figure 15: IPC (moving average) plots for 50 seconds of execution for (a) Video Game 2 and (b) Parser.

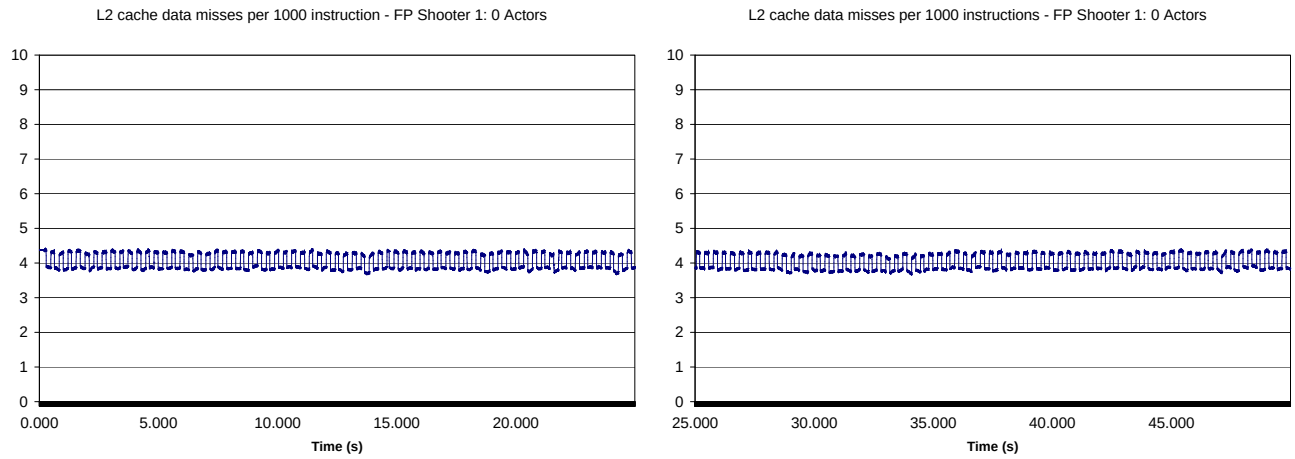


(a) DVD Player

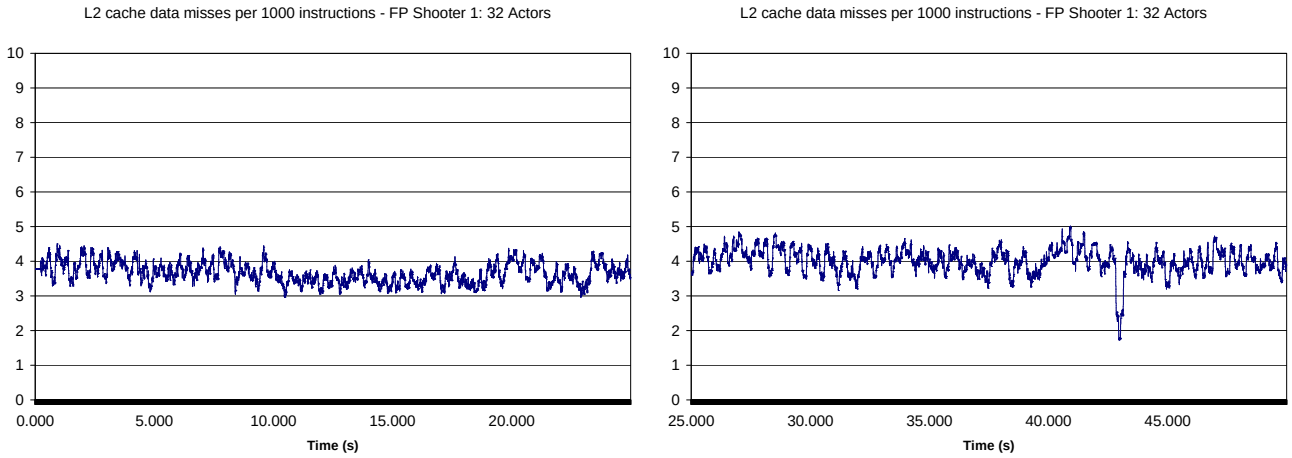


(b) Quicktime

Figure 16: L2 cache data miss (moving average) plots for 50 seconds of execution for the (a) DVD Player and (b) Quicktime applications.

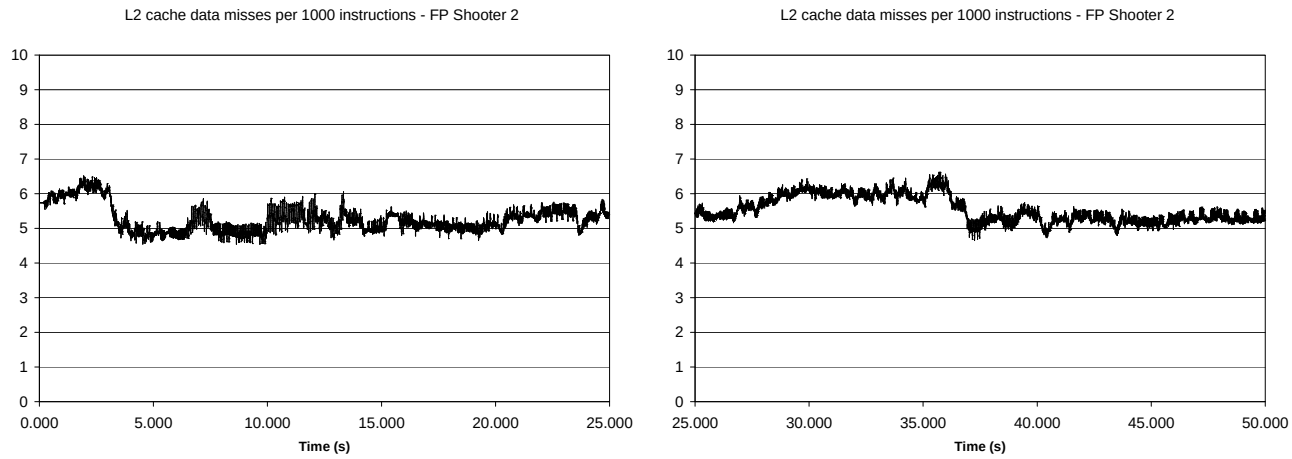


(a) Video Game with 0 Actors

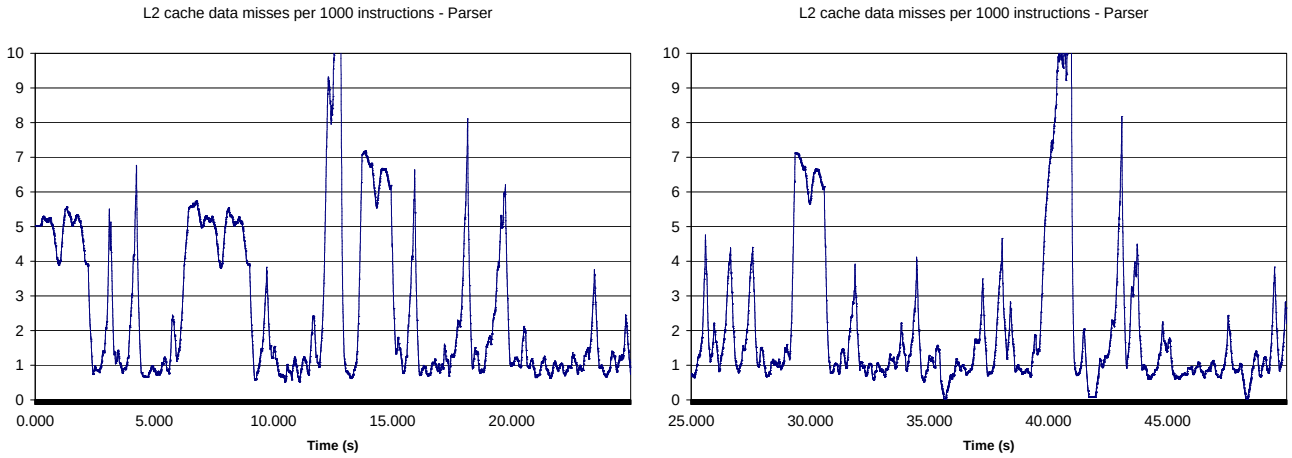


(b) Video Game with 32 Actors

Figure 17: L2 cache data miss (moving average) plots for 50 seconds of execution for a video game with (a) 0 computer-controlled characters and (b) 32 computer-controlled characters.



(a) Video Game 2



(b) Parser

Figure 18: L2 cache data miss (moving average) plots for 50 seconds of execution for (a) Video Game 2 and (b) Parser.