

# Performance Analysis of Embedded Applications on a Pentium-4 Based Machine

Pradeep S. Nair, Dhireesha Kudithipudi, Eugene B. John and Fred W. Hudson  
{pnair@lonestar.utsa.edu; dhireesha.kudithipudi@utsa.edu; eugene.john@utsa.edu;  
fred.hudson@utsa.edu}

Department of Electrical and Computer Engineering  
University of Texas at San Antonio  
San Antonio, TX 78249, USA

**Abstract** – *As embedded applications find widespread use in the PC domain, there arises a need to study their performance on such systems. Although hardware characterization and performance of embedded systems has been widely researched and continues to be so, the performance of general-purpose personal computing systems while running embedded applications has not been widely reported hitherto. This paper makes an attempt to examine the performance of a subset of widely used embedded applications on a general purpose Intel Pentium 4 based personal computer. In general, these benchmarks returned high CPI values. Some of them also experienced high rates of trace cache and L1 cache miss rates. However, the L2 hit ratio and the branch prediction rates were fairly high thus implying smaller and repetitive working sets.*

**Keywords:** Embedded, Performance Analysis.

## 1. Introduction

The last two decades have witnessed a steady increase in the demand for embedded applications such as streaming video, audio, palmtop and missile control applications etc. This further resulted in the development of complex embedded systems, many of which include complicated I/O ports, displays, keyboard etc. One of the main reasons for this increased demand is the widespread growth of worldwide web (WAN's, LAN's), which brought with it numerous opportunities and challenges. Technological advances in integration technology have also made these developments possible.

The embedded application subsets that emerged in recent times are used not only in traditional embedded processors but also in other domains such as desktop and laptop computers (general purpose processors). Notable among such applications are steaming audio, video and other internet-based applications. In general purpose processors, embedded applications like Voice over Internet Protocol (VOIP) are now widely being

used for communication between cell phones, landlines and other communication systems. Streaming audio and video web casts are also becoming popular by the day. These applications often use personal computers either as a routing point or as a final port. Therefore, the performance of these applications is strongly dependent on the processing and computational capability of personal computers. Thus, it becomes imperative to measure the performance of embedded applications when they are executed on the general-purpose processors (PC). Analyzing the performance of embedded applications on existing personal computing systems forms an important aspect in the run-up to future PC designs. Such designs will be capable of yielding good performance not only with respect to embedded benchmarks but also other categories of applications. Most of the previous research in this area has concentrated on hardware characterization and simulator-based performance evaluation. A performance analysis-based study of embedded benchmarks on general-purpose processors or PC's has so far not received much attention from researchers. Performance analysis deals with the performance of existing systems i.e. real systems that have been implemented in hardware. The data obtained in such cases tends to be more accurate and believable when compared to performance evaluation and characterization studies using simulators. In this study, we chose to analyze the performance of a Pentium-4 based PC on embedded benchmarks. The benchmarks are chosen from the MiBench suite, a freely available representative set of benchmarks from the University of Michigan at Ann Arbor [1].

The main objective of this study is to analyze the performance of embedded applications when executed on desktop PC's; with an aim to find out if the organization of such a machine is well suited for embedded benchmarks. The remainder of this paper is organized as follows: section 2 discusses previous research related to this paper. Section 3 contains

information pertaining to the benchmarks used. The tools and methodology adopted in this study are described in section 4. The analysis results are presented in section 5. Finally, section 6 concludes the paper

## 2. Previous Research

This section describes previous research in the area of performance evaluation and characterization of embedded systems and applications.

In their paper, Russell and Jacome [2] proposed a performance evaluation technique for component-based embedded systems in which the issues of defining an *evaluation scenario* and performance evaluation based on such scenarios were addressed. An evaluation scenario refers to the relevant subsets of the system components, which consists of both hardware and software. This paper emphasized on performance evaluation at a lower level of detail consisting of instructions and operations.

Gupte et al. considered the design challenges faced in high performance embedded DSP design in their paper [3]. Improving performance by faster address decoding and faster memory access was one of the main objectives of this paper. They also proposed gate-level performance enhancing techniques for such systems.

Li and Malik proposed a solution to the problem of determining the best and worst times bounding the running time of a program on an embedded system [4]. The main motivation for this work was that the knowledge of the best and worst bounds would be very helpful in the design of real-time systems, which have to meet response deadlines. It could also be used in the development of real-time scheduling algorithms.

Grunewald et al. presented a simulator-based performance evaluation method to optimize embedded applications [5]. The optimization is achieved through the characterization of execution time, chip area and energy dissipation of an embedded system.

Corsaro and Schmidt focused on the quality of Real Time Specification for Java (RTSJ) implementations [6]. In this study, various performance metrics were measured on a Pentium III machine for the RTJPerf suite of Java programs. The main objective of this paper was to evaluate the performance of RTSJ features that are crucial to the development of many embedded applications.

Lee et al. proposed a suite of multimedia and communication applications called *Mediabench* [7]. They aimed to provide a new set of real programs accurately representing the then emerging class of media and communication applications. They also used Mediabench to drive a synthesis experiment using a

processor core based on the IBM 40x PowerPC cores and then repeated the same with SPECint.

Milenkovic et al. [8] evaluated the performance of memory hierarchy in embedded systems using the MiBench benchmark suite. In this paper, an architectural simulator toolset, called SimpleScalar [9], based on the ARM (Advanced RISC Machines) instruction set was used to evaluate the effects of various cache design parameters such as block size, replacement policy, cache size and organization the performance of embedded applications.

Bhargava et al. evaluated the performance of x86 Multimedia extension (MMX) instruction set on a set of benchmarks which included kernels and applications [10]. In this paper, the speedup obtained by choosing MMX versions of the considered benchmarks over the non-MMX version was presented. A detailed profile of the benchmarks was also obtained using the VTune performance analyzer [11].

Kohalmi and Newport reported a new approach to embedded computer performance measurement by combining the performance aspects of hardware, operating systems, and compilation and runtime systems. [12]. Subsequently, in [13], a novel approach for measuring throughput and I/O was implemented and utilized.

Dalal and Ravikumar studied software optimization techniques that could be used with compilers in order to reduce power dissipation in CPU, address and data buses and memory units [14]. They used an emulator based on an ARM processor to estimate the effects of software optimizations on power consumption and performance of embedded systems.

Kalavade and Moghe addressed the issue of verification of embedded systems which ran applications with run-time skew between them [15]. They formulated a problem aimed at determining this skew and presented a methodology for verification of such systems.

Rapaka and Marculescu proposed a novel technique to enhance the simulation speed of embedded and general purpose applications at the architectural level, without sacrificing the accuracy of performance and power consumption estimates [16]. The proposed technique involved identifying sections of code with high temporal locality. This information was then used with hybrid simulators. These hybrid simulators are capable of switching from detailed simulation mode to fast simulation mode upon encountering sections of code that are predictable in nature. This technique can thus help reduce simulation time.

Lee et al. examined the performance of desktop applications on an x86 machine with a Windows NT operating system [17]. One of the main objectives of this paper was to identify the characteristics of typical desktop applications and compare them to those of the

SPEC95 suite. The impact of the execution environment provided by the operating system on program execution was also studied.

Petko et al. characterized the cache subsystem for embedded applications with special emphasis on video computation workloads [18]. The effects of cache size, block size, and set associativity on performance were investigated using the SimpleScalar toolset and it was concluded that these workloads utilize caches pretty well.

In another related paper by Kudithipudi and John [19], the effects of varying gate lengths, level 1 cache sizes and level 2 cache sizes on leakage power consumption were investigated by simulating an ALPHA 21264 processor running embedded applications. The MiBench suite of embedded benchmarks was chosen in this study.

Most of the related research preceding this paper concentrated on the following broad categories: 1. proposing novel techniques which directly or indirectly improve the performance or power consumption of embedded systems. 2. Techniques to improve simulation speed for embedded systems. 3. Simulation based studies aimed at characterizing the micro-architecture of such systems and 4. Studies aimed at understanding the program structure and characteristics of embedded benchmarks. In contrast to this, there have been very few studies concerning how efficiently real desktop computers perform while running embedded applications as opposed to systems that were simulated. While the flexibility offered by simulators is very helpful and important for characterization and estimation, the overhead introduced by them usually tends to reduce the accuracy of the presented data. Analyzing the performance of desktop systems accurately for embedded benchmarks is the main motivation for this work. We also hope that this paper will be a precursor to similar, subsequent studies that may be performed on other real, existing systems.

### 3. Benchmarks

The benchmarks used in this paper were chosen from the MiBench suite. MiBench is a freely available suite of embedded benchmarks and contains most commonly used applications from different areas within the embedded domain. These areas include automotive and industrial control, office, network, security and telecommunication. The benchmarks chosen in this study are described below:

*basicmath*: This program performs simple math calculations such as square root and angle conversion. The input is a set of constants.

*bitcount*: This program tests the bit manipulation capabilities of a processor. It achieves this by counting

the number of bits in an array of integers, using five different methods. The input data is an array of integers with equal number of 0's and 1's.

*susan*: *Susan* is an image recognition package. It was developed for recognizing corners and edges in Magnetic Resonance Images (MRI) of the brain. It is typical of a real world program that would be employed for a vision based quality assurance application. It can smooth an image and has adjustments for threshold, brightness, and spatial control. The small input data is a black and white image.

*jpeg encode/decode*: JPEG is a standard, lossy compression image format. It is included in MiBench because it is a representative algorithm for image compression and is commonly used to view images embedded in documents. The input data is large and small color images.

*tiff2bw*: *Tiff2bw* converts a color TIFF image to black and white image.

*FFT/IFFT*: This benchmark performs a Fast Fourier Transform (FFT) and its inverse transform on an array of data. Fourier transforms are used in digital signal processing to find the frequencies contained in a given input signal. The input data is a polynomial function with pseudorandom amplitude and frequency sinusoidal components.

*CRC32*: This benchmark performs a 32-bit Cyclic Redundancy Check (CRC) on a file. CRC checks are often used to detect errors in data transmission. The data input is the sound files from the ADPCM benchmark.

*dijkstra*: The Dijkstra benchmark constructs a large graph in an adjacency matrix representation and then calculates the shortest path between every pair of nodes using repeated applications of Dijkstra's algorithm. Dijkstra's algorithm is a well known solution to the shortest path problem and completes in  $O(n^2)$  time.

*rijndael encrypt/decrypt*: *Rijndael* was selected as the National Institute of Standards and Technologies Advanced Encryption Standard (AES). It is a block cipher with the option of 128-, 192-, and 256-bit keys and blocks.

### 4. Experimental setup

The experimental setup used for our analysis is described in this section. The characteristics of the machine used are tabulated in table 1.

<b>Processor</b>	<b>Intel Pentium 4</b>
<b>Clock Frequency</b>	1.8 GHz.
<b>Operating</b>	Windows 2000

<b>System</b>	
<b>FSB</b>	400 MHz.
<b>L1 I-Cache</b>	12K micro-ops, 8-way set associative trace cache
<b>L1 D-Cache</b>	8 KB; 4-way set associative; 64 B line size; 2/9 clocks latency (integer/floating point); Write-through
<b>L-2 Cache</b>	512KB; 8-way set associative; 64 B line size; 7 clocks latency; Write-back
<b>Main Memory Size</b>	256 MB

Table.1. Hardware configuration of the machine used in this study.

This processor was chosen as a representative one, because it is one of the most commonly used. To perform the analysis, the first step was to build and compile the MiBench benchmarks using Microsoft C++ compiler (version 6.0), without setting any compiler optimization flags. Each of the benchmarks was then fed with small input data sets to run on the Windows 2000/Pentium-4 based machine. The performance data was collected using the Intel VTune analyzer.

The VTune analyzer provides time and event based system-wide sampling with the most accurate representation of actual software performance with minuscule impact on the program being executed (for Intel Processors). We used VTune 7.2, which automatically recognizes the Intel processor and interactively provides a framework with the corresponding architecture's metrics. We chose to monitor those events, which reflect micro architectural behavior, such as instruction mix, cache, branch-prediction events in the event-based sampling mode.

Sampling is a statistical technique in which samples of the events of interest are taken at regular intervals known as sampling intervals. In event-based sampling, samples are collected after a specified number of occurrences of an event by interrupting the processor. The execution context of the program being executed is stored upon an interrupt. Based on the number and type of events, the VTune sampling collectors may run the collector several times. In other words, many runs may be required to collect data relating to different events. The main advantage of sampling is that it does not affect the binary files of the program and can hence be considered non-intrusive. The only minor drawback of this technique is that it extrapolates the information provided by the samples to obtain information on the overall execution.

## 5. Performance Results

The number of retired instructions for each of the simulated embedded applications is shown in Fig 1. A retired instruction is an instruction whose execution is reflected in the machine state as opposed to an instruction that is just executed but whose execution is not reflected in the updated machine state. The number of retired instructions is in the order of millions of instructions and varies from a few tens of millions of instructions to a few hundred millions of IA-32 instructions. This disparity can be attributed to the widely different nature of these benchmarks. For example, the basicmath benchmark involves a large number of static computations to evaluate useful mathematical functions such as the square root, cube root etc. A detailed discussion on the program structures of all the benchmarks considered here can be found in the paper on MiBench [1].

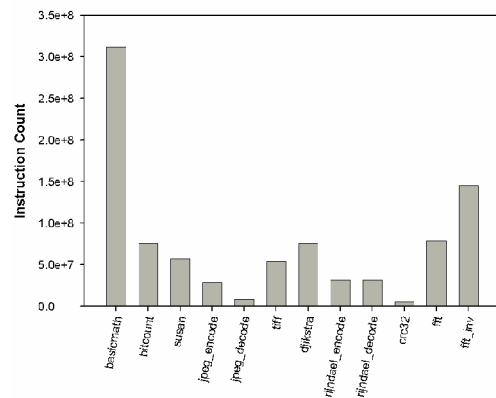


Fig 1. Number of retired instructions for various embedded benchmarks

The average number of clock cycles taken to execute one instruction, or CPI, is one of the primary parameters that indicates performance. The CPI values of the various applications were recorded as shown in fig. 2.

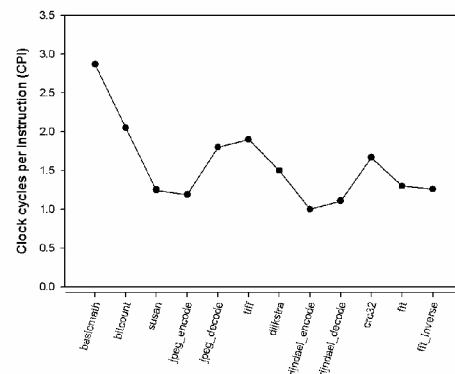


Fig. 2 CPI values for various embedded benchmarks

It was observed that none of the applications yielded a CPI of less than 1, or an IPC count of greater than 1, which is expected of a superscalar processor. CPI values spanned a wide range of low values with *rijndael\_encode* yielding the best value of 1 among all the applications. The worst CPI value was recorded for the *basicmath* program, which gave a CPI value of 2.87. One reason for this high CPI value could be that this benchmark, while being computation intensive, also has a significant number of memory accesses associated with it.

In order to ascertain the cause of the high CPI values, parameters reflecting the processor hardware efficiency were recorded. First among these is the Trace cache Delivery Rate. The Trace cache is a special type of first level cache used in the Pentium 4 microarchitecture [4]. Most instructions are fetched from the trace cache and only when there is a trace cache miss is the Level 2 (L2) cache accessed for fetching instructions, which then have to be decoded. The trace cache is capable of providing up to three micro-ops to the out-of-order execution core and has a capacity to hold up to 12K micro-ops. The Trace Cache delivery rate is shown in figure 3 below.

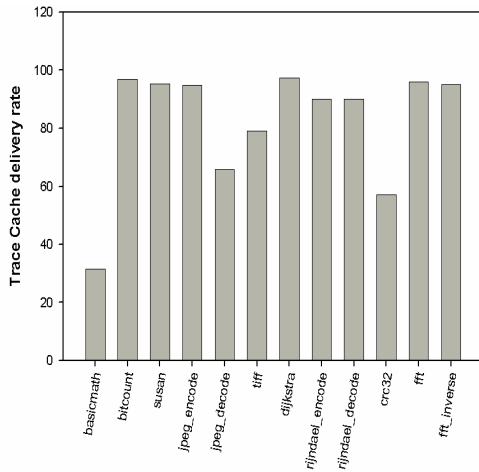


Fig. 3 Trace Cache delivery rates obtained for various embedded benchmarks

It was observed that the trace cache delivery rate was very low for *basicmath* and *crc32*. The CPI values for these benchmarks may be improved to some extent by improving the trace cache delivery rates. However, it cannot be concluded that low trace cache delivery rate is the sole reason for high CPI values, as is evident from the high CPI value for the *bitcount* benchmark.

The next step in the analysis consisted of studying the L1 and L2 cache statistics for these benchmarks. The L1 cache hit load rate also varied considerably among the benchmarks and it was observed that the increase in CPI values reflected the decrease in the L1 cache hit rates, except in case of the *bitcount*

benchmark. This suggests that increasing L1 cache hit rates could help achieve better performance for these benchmarks. Memory intensive applications, such as *basicmath*, *tiff* and *crc* etc., generally yielded high CPI values and it this is due to the fact that these benchmarks either experienced high number of trace cache or L1 cache misses. To improve performance, further characterization studies are necessary to find out ways to reduce the miss rates. For example, it needs further analysis to find out which parameter of the cache organization is responsible for the low L1 hit rates. Another important factor that could be detrimental for performance is that of accesses which need data or instructions across cache block boundaries.

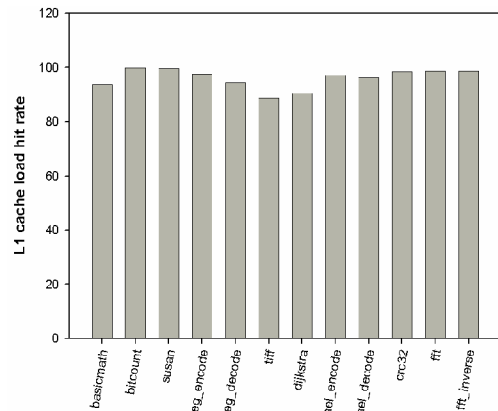


Fig. 4 L1 cache statistics for MIBench embedded applications

As far as L2 cache statistics are concerned, most benchmarks showed good L2 cache load hit rate. This suggests that embedded applications generally have small working sets and can benefit greatly by larger L2 caches.

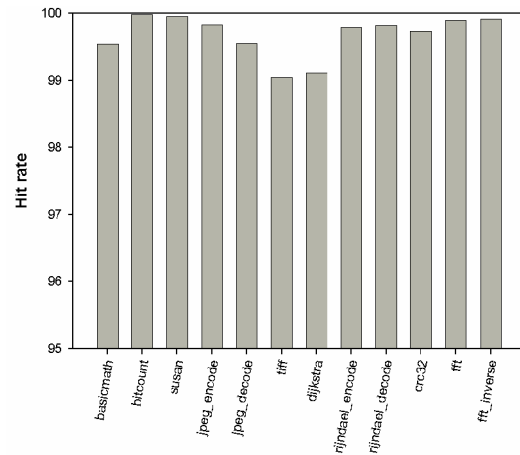


Fig. 5 Level 2 cache load hit rates

Another very important parameter that can greatly hinder performance is the branch prediction rates. This

is because of the extra delay in resolving the branch target addresses after the branch instruction has already been decoded. This delay is introduced because the branch instruction has to wait until the execute stage of a pipeline to know where it has to branch. The branch prediction rates for the various embedded applications are shown in fig. 6. The branch prediction rates were fairly high and this trend suggests that these benchmarks involve a large number of repetitive operations.

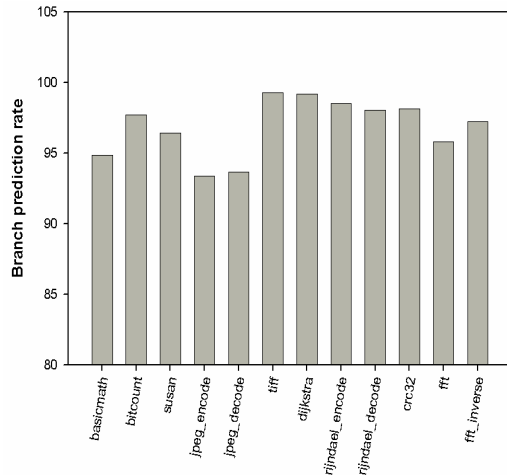


Fig. 6 Branch prediction rates for the embedded benchmarks

## 6. Conclusions and future work

In this paper, the performance of a representative subset of embedded applications was analyzed on a real machine based on a Pentium-4 system and Windows 2000 operating system. These benchmarks form a wide range of applications and generally have small active working sets. Achieving better cache hit rates and using larger caches may significantly improve their performance on desktop systems. However, while executing these applications, the overall performance of the desktop PC considered in this study was found to be rather poor. This suggests that the microarchitecture of current desktop system needs to be modified accordingly in order to achieve better performance. It also means that the effects of such changes on performance while running other class of applications needs to be investigated further to ensure that the performance, in general, does not get affected drastically.

## 7. References

- [1] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, Richard B. Brown, "MiBench: A free, commercially representative embedded Benchmark suite" IEEE International Workshop on Workload Characterization, Dec 2001. Page(s):3 – 14
- [2] Russell, J.T.; Jacome, M.F., "Architecture-level performance evaluation of component-based embedded systems" Proceedings, Design Automation Conference, June 2003. Page(s): 396 – 401
- [3] Gupte, A.; Mehendale, M.; Ramamritham, R.; Nair, D., "Performance considerations in embedded DSP based system-on-a-chip designs", Fourteenth International Conference on VLSI Design, Jan 2001, pp. 36-41
- [4] Li, Y.-T.S.; Malik, S., "Performance Analysis of Embedded Software Using Implicit Path Enumeration", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Volume 16, Issue 12, Dec. 1997 Page(s):1477 – 1487
- [5] Grunewald, M.; Niemann, J.-C.; Ruckert, U., "A performance evaluation method for optimizing embedded applications", The 3rd IEEE International Workshop on system-on-Chip for Real-Time Applications, July 2003. pp 10-15
- [6] Corsaro, A.; Schmidt, D.C., "Evaluating real-time Java features and performance for real-time embedded systems," Proceedings, Eighth IEEE Real-Time and Embedded Technology and Applications Symposium, Sept 2002. pp 90-100
- [7] Chunho Lee; Potkonjak, M.; Mangione-Smith, W.H., "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems", Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, Dec 1997. pp 330-335
- [8] Milenkovic, A.; Milenkovic, M.; Barnes, N., "A performance evaluation of memory hierarchy in embedded systems, Proceedings of the 35th Southeastern Symposium on System Theory, march 2003. pp 427-431
- [9] D.C. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0", Technical Report CS-TR-97-1342, University of Wisconsin-Madison, June 1997.

- [10] Bhargava, R.; John, L.K.; Evans, B.L.; Radhakrishnan, R., "Evaluating MMX technology using DSP and multimedia applications", MICRO-31, 31st Annual ACM/IEEE International Symposium on Microarchitecture, Dec. 1998. pp 37-46
- [11] "Intel VTune Performance Analyzers". Available at: <http://www.intel.com/cd/software/products/asmona/eng/vtune/index.htm>
- [12] Kohalmi, D.; Newport, J.; Paul, D.; Roark, C.; Struble, D.G., "A new development in embedded computer performance measurement", Proceedings, IEEE/AIAA 10<sup>th</sup> Digital Avionics Systems Conference, Oct. 1991. pp 232-236
- [13] Kohalmi, D.; Newport, J.; Paul, D.; Roark, C.; Struble, D.G., "A new development in embedded computer performance measurement", Aerospace and Electronics Conference, May 1991. pp 633-639
- [14] Dalal, V.; Ravikumar, C.P., "Software power optimizations in an embedded system", Fourteenth International Conference on VLSI Design, Jan. 2001. pp 254 -259.
- [15] Kalavade, A.; Moghe, P., "On the performance verification of embedded systems with concurrent dynamic applications", Conference Record of the Thirtieth Asilomar Conference on Signals, Systems and Computers, Nov. 1996. pp 1349 - 1353 vol.2
- [16] Venkata Syam P. Rapaka, Diana Marculescu, "Pre-Characterization Free, Efficient Power/Performance Analysis of Embedded and General Purpose Software Applications", Design, Automation and Test in Europe Conference and Exhibition, 2003. pp 504-509
- [17] Lee, D.C.; Crowley, P.J.; Baer, J.-L.; Anderson, T.E.; Bershad, B.N., "Execution characteristics of desktop applications on Windows NT", Proceedings, The 25th Annual International Symposium on Computer Architecture, July 1998. pp 27-38
- [18] Petko, S.; Kudithipudi D.; John, E., "Cache performance of video computation workloads", Third International Workshop on Digital and Computational Video, Nov. 2002. pp 169-175
- [19] Kudithipudi, D.; Petko, S.; John, E., "Cache leakage power in embedded applications", The 2004 Annual Midwest Symposium on Circuits and Systems, July 2004. pp II-517–II-520. vol. 2
- [20] Hinton, G.; Sager, D.; Upton, M; Boggs, D.; Carmean, D; Kyker, A; Roussel, P., "The Microarchitecture of the Pentium 4Processor" in Intel Technology Journal Q1, 2001