Investigating the Effectiveness of a Third Level Cache

by

S. Ghai
IBM Corporation
Austin, TX 78758
sanj@austin.ibm.com

J. Joyner
IBM Corporation
Austin, TX 78758
joyner@austin.ibm.com

L. John
ECE Department
The University of Texas at Austin
Austin, TX 78712
ljohn@ece.utexas.edu

Abstract

Even as trends in processor development yield wider superscalar processors, multiple processors on a single die, or integrated processor/memory configurations, faster access to system memory is required to meet the performance needs of these advanced processors. While investigating the performance of the PentiumPro processor [12], Bhandarkar and Ding observe that the single factor that showed the strongest correlation to CPI (Cycles Per Instruction) was L2 misses indicating that L2 miss latency could not be overlapped by all the latency hiding and ILP techniques employed. It is unknown whether the incorporation of additional levels of cache is a solution to this problem. In this paper, we investigate the effectiveness of an L3 cache in uniprocessor, dual-processor and quad-processor configurations. Simulation studies with SPECint and SPECfp benchmarks demonstrate that a 4MB L3 with 8-way set associativity is capable of running in a shared mode among 4 processors with over a 96% hit rate, decreasing the average memory access time (AMAT) by 6.5% and reducing the traffic by 95% from a system without an L3. Thus L3 yields dramatic improvements in reducing bus traffic and reasonable improvement in AMAT. Considering all the system configurations considered, AMAT decreases by approximately 10% for the SPEC integer benchmarks and 3.5% for the floating point benchmarks.

1.0 Introduction

A sophisticated memory hierarchy is required in order to meet the ever-increasing data and instruction demands from wide and aggressive superscalar processors. While investigating the performance of the PentiumPro processor [12], Bhandarkar and Ding observe that the single factor that showed the strongest correlation to CPI (Cycles Per Instruction) was L2 misses indicating that L2 miss latency could not be overlapped by all the latency hiding and ILP techniques employed. Current trends seem to indicate that system demands will focus on scaling processor resources by either adding more individual processors or incorporating multiple processors in a single package. Especially in the case of networked computing environments where clients run applications off of a small group of servers, multiprocessor (MP) systems coupled with a sophisticated memory hierarchy to feed the processors efficiently with enough information are required. In this paper, we investigate whether a third-level (L3) cache is beneficial for a server class machine. Figure 1 shows how an L3 may be incorporated into a typical MP system. An L3 cache can potentially improve the performance of a system as described below.

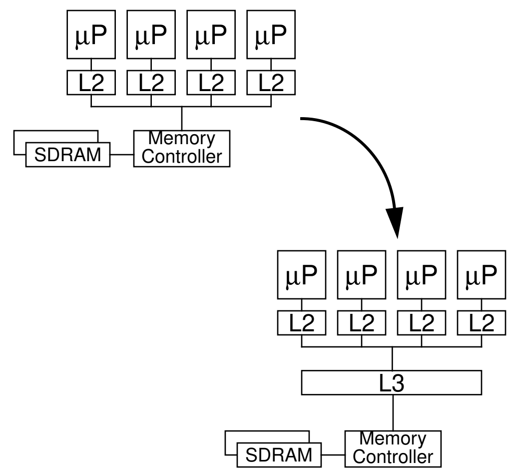


Figure 1. Typical incorporation of an L3

In a typical system without an L3, all misses in the L2 cache would be forced to access a slower system memory. Even with a complex memory controller, the processor must wait on the order of tens of cycles for the data to eventually be retrieved into the upper-level caches. However, with an L3 configuration as given in Figure 1, this unacceptable delay may be somewhat averted. After the cold start, or compulsory, misses have filled the L3 with the initial data required by the processors above it, most accesses that miss the processors' L2s may be serviced by the L3. Current technologies in static RAM can support such an L3 at speeds of up to 166MHz or more [7] and thus provide the L2 cache with the needed information much faster than standard

SDRAM in system memory. Thus, instead of these misses incurring the delay of a slow main memory, the misses may undergo a much smaller delay should the data be present in the L3.

Fig.2 illustrates three systems, the first of which has no L3 (Figure 2(a)) and is similar in configuration to an IBM RS/6000 F50 Enterprise Server [5]. However, if an L3 were to be incorporated as an "intelligent" switch as shown in Figure 2(b), the speed of the bus and the performance of the overall system may be improved. The grouping of 4 processors and a single L3 as a node may be duplicated up to 4 times in a system as illustrated in Figure 2(c). This modularization of the system allows for easy expandability to meet varying customer needs and permits the expansion from a simple 4-way MP to a 16-way MP system while having the added benefit of running the primary system bus at the same speed as the original configuration in Figure 2(a).

For a multiple L3 system as illustrated in Figure 2(c), the L3 provides the point of memory coherency that controls what information its respective L2s and L1s have. For example, an inclusion policy would demand that any data held in a higher-level cache (e.g. L1) must be present in the lower-level cache (e.g. L2), though not necessarily in the same dirty/clean/shared state [4]. In a typical MP system, all processors' L2s must snoop the system bus for traffic, such as flushes or invalidates, which might affect the state of data in their cache. However, with multiple L3s in a system, the L3s act as buffers for their respective L2s by filtering the traffic present on the system bus [1]. Rather than all of the processors snooping the system bus for cache-related operations, only the L3s need sample the bus. Due to the data inclusion policy, the L3 knows which system bus cacheable operations affect higher-level caches in its node and which operations it may filter out and ignore. Thus, the L3 acts as a snoop filter and passes on only the relevant cacheable operations to its L2s while "hiding" other irrelevant operations. This feature decreases the snoop operation traffic on the processor bus side of the L3 and allows for more processor bus bandwidth to be used for L2/L3 data transactions.

In this paper, we perform a quantitative characterization of the effectiveness of an L3 cache in the memory hierarchy of uniprocessors and small scale multiprocessor configurations. Trace driven simulation is used as the performance evaluation technique. In section 2, an overview of the cache simulation model and model stimulus is presented. Section 3 contains the results and analysis. Section 4 provides summary and concluding remarks.

2.0 L3 Cache Simulator Model

We developed a C++ trace-driven cache simulator aimed at gaining some insight into the effectiveness of including a shared L3 cache in a memory subsystem. This section provides an overview of the simulator model, details how the model operates, lists the traces we used in testing the simulator, and specifies the scenarios that we ran in gathering performance data for the L3 cache.

2.1 Model Overview

Our trace-driven cache simulation model, shown in Figure 3, is a C++ program that reads PDATS trace files [6] and simulates a memory hierarchy which includes dedicated L1 and L2 caches and a shared L3 cache. The caches, illustrated in Figure 3, are C++ standalone objects designed by Bryan Hunt from New Mexico State University and comprise the basic building blocks for our cache simulator [3,4]. The cache objects are programmable in that they permit the user to vary parameters, such as the block size of a cache, the number of blocks contained in a cache, and the cache's associativity. The use of a shared L3 cache required us to consider an arbitration scheme for accessing the L3. For each simulation cycle, we service each processor's trace reference in a round-robin manner. At the end of each simulation cycle, we update who would have their reference handled first in the next cycle, thus indicating where the next round-robin circle would begin.

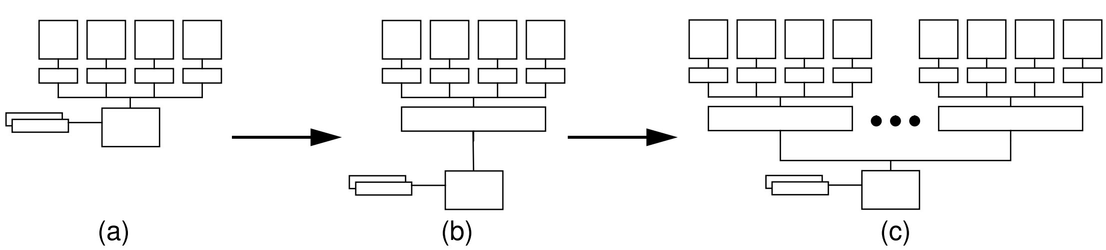


Figure 2. (a) Typical, (b) single L3, and (c) multiple L3 configurations

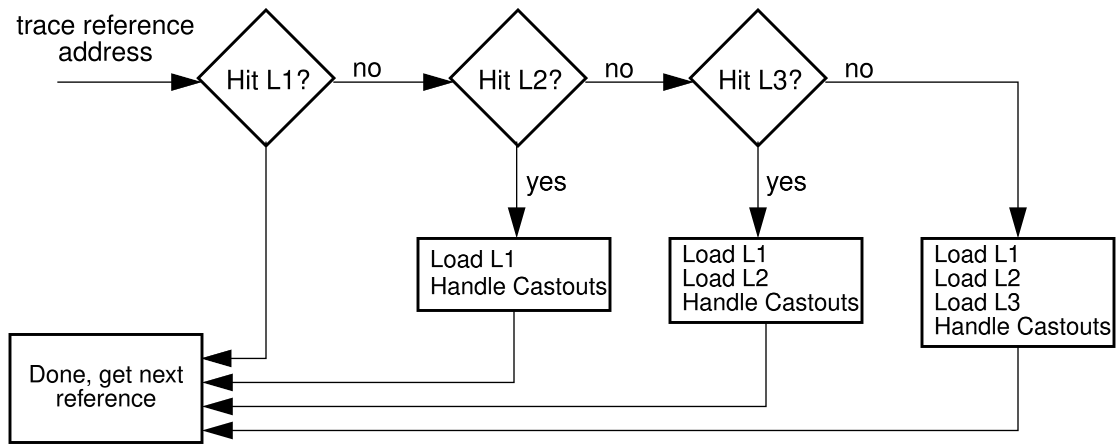


Figure 4. Dataflow representation of L3 cache simulator

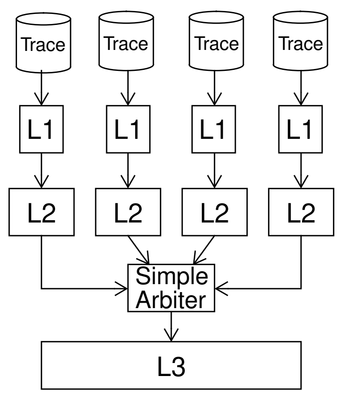


Figure 3. L3 cache simulation model

Figure 4 depicts the general dataflow of how our cache simulator works. Within a simulation cycle, the model reads in a reference from the corresponding processor's program trace and determines what actions need to take place to satisfy the reference. There are four cases that we account for depending upon whether the reference address:

- hits in the L1 cache
- misses the L1 cache, but hits in the L2 cache
- misses the L1 and L2 caches, but hits in the L3 cache
- misses the L1, L2, and L3 caches causing data for the reference to be retrieved from main memory.

For use in this project, we obtained several traces from the New Mexico State University web site [9]. These are large, 1-billion reference traces for eight of the SPEC92 benchmark programs. We selected four SPECint92 programs (compress, eqntott, espresso, sc) and four SPECfp92 programs (ear, nasa7, spice2g6, swm256) with which to test our simulator. The simulation model reads in "gzip"ped PDATS formatted traces that were converted from a dinero format by an executable provided from the NMSU web site [9]. The lack of shared data and true multiprocessor benchmarks emulates a typical server configuration where independent programs are applied to different processors in the system.In testing our cache simulator, we made a conscious effort to fix the parameters of the L1 and L2 caches, while varying those of the L3 cache. By altering the L3 configuration, we hoped to determine the optimal solution for our environment. The L1 cache we used was a unified 16KB 4-way set associative cache with 32B blocks, while the L2 cache was configured as a unified 256KB 4-way set associative cache with 64B blocks. For the unified L3 cache, we held the block size constant at 128B, but varied the cache size (1MB/2MB/4MB) and the associativity (4-way/8-way set associative). Finally, we varied the number of processors in our system among a uniprocessor case, a 2-way MP arrangement, and a 4-way configuration.

For each uniprocessor configuration, we ran four integer and four floating point programs. For the 2-/4-way MP configurations, we ran 4/1 integer, 4/1 floating point, and 4/4 runs where we mixed integer and floating point programs together, respectively. These precise scenarios are detailed in Table 1. We performed these simulation runs on an IBM RS/6000 F50 Enterprise Server that executed 1 million references per second, which expedited the gathering of data for this project.

3.0 Results

This section provides details concerning the miss rate formulas that we used to analyze the results, and average memory access time (AMAT) and bus traffic of systems with and without L3 caches. All of the detailed statistics used to create the graphs are contained in a detailed version of the paper available at http://www.ece.utexas.edu/~ljohn/l3.ps.

3.1 Miss Rate Trends

In analyzing the data gathered in our simulation, we chose to evaluate the miss rate of the L3 for various L3 cache configurations. Equation 1 illustrates how we calculated the raw miss rates of our L3 caches. Compulsory misses, capacity misses, and conflict misses [2] are considered and summed together.

$$\text{miss rate} = \frac{\text{\# of compulsory/capacity/conflict misses}}{\text{\# of L3 references}} \quad (1)$$

In looking at the L3 cache miss rates in Figure 5, there are two trends to be noted. First, as the size and associativity of the L3 cache are increased, the miss rate improves across all of the program trace types (integer, floating point, or a mixture of integer and floating point programs). Secondly, for the 4MB 8-way set associative L3 configuration, the L3 miss rates decrease as the number of processors in the system increases from a uniprocessor scenario to a 2-way and 4-way MP configuration. In all of the scenarios, the configuration which performs the best is a 4MB 8-way set associative L3 cache that achieves miss rates as low as 1% for some integer programs.

Table 1: Scenarios simulated

# of Processors	Benchmark Type	Benchmarks Run
1	int	008.espresso
		023.eqntott
		026.compress
		072.sc
	fp	013.spice2g6
		056.ear
		078.swm256
		093.nasa7
2	int	008.espresso, 023.eqntott
		026.compress, 072.sc
		008.espresso, 026.compress
		023.eqntott, 072.sc
	fp	013.spice2g6, 056.ear
		078.swm256, 093.nasa7
		013.spice2g6, 078.swm256
		056.ear, 093.nasa7
	mix	013.spice2g6, 072.sc
		078.swm256, 023.eqntott
		056.ear, 026.compress
		093.nasa7, 008.espresso
4	int	008.espresso, 023.eqntott, 026.compress, 072.sc
	fp	013.spice2g6, 056.ear, 078.swm256, 093.nasa7
	mix	013.spice2g6, 023.eqntott, 078.swm256, 008.espresso
		056.ear, 072.sc, 093.nasa7, 026.compress
		013.spice2g6, 072.sc.pdt, 093.nasa7, 008.espresso
		078.swm256, 023.eqntott, 056.ear, 026.compress

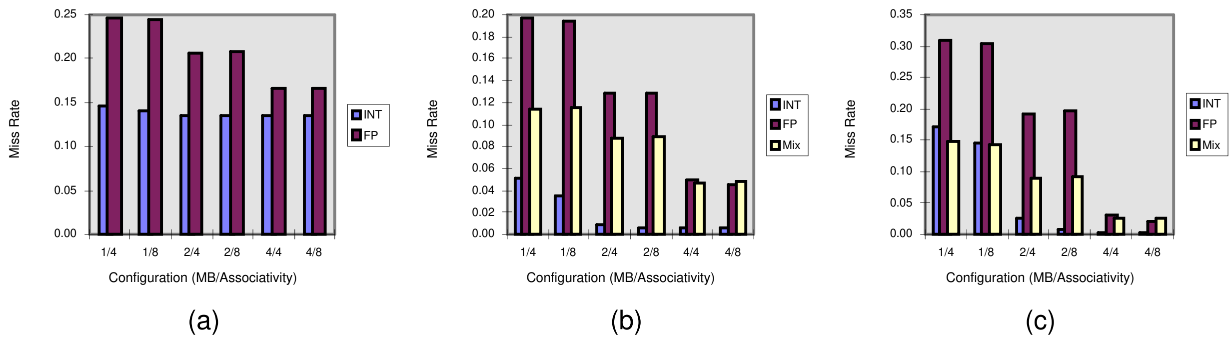


Figure 5. Miss rates for (a) uniprocessor, (b) 2-way MP, and (c) 4-way MP configurations

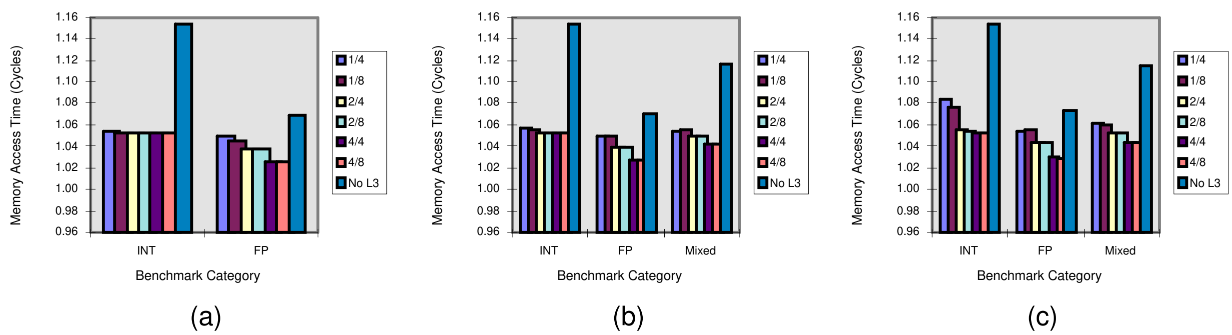


Figure 6. Average memory access times for (a) uniprocessor, (b) 2-way MP, and (c) 4-way MP configurations

3.2 Average Memory Access Time With and Without L3

Given the miss rates, we can estimate the performance of a system in terms of the average memory access time (AMAT). In comparing the AMAT of a system utilizing an L3 to a system without an L3, we may better understand whether or not an L3 would prove beneficial or detrimental to the memory hierarchy. To determine the AMAT, we chose reasonable estimates for the cycle cost of L1, L2, and L3 hits and misses as well as the additional cycles required to perform the write-backs from higher-level caches to lower-level caches in the memory subsystem. These parameters are detailed in Table 2.

Table 2: Estimated cycle delays

Access	Symbol	Cycles
L1 hit	L1HT	1
L2 hit	L2HT	4
L3 hit	L3HT	16
Memory read	MEM	96

We calculated the AMAT for integer, floating point, integer/floating point mixed benchmarks in each of the L3 configurations and the system without an L3. These configurations are further divided into uniprocessor, 2-way MP and 4-way MP models. As these results in Figure 6 indicate, the AMAT decreases as the size of the L3 and its associativity are increased (the legend in the figure reflects the configuration of the L3 -- MB/associativity). Furthermore, in all cases, the 4MB 8-way set associative L3 has a lower AMAT than a system without an L3. Averaged across all of the benchmark types, this L3 configuration has an AMAT that is greater than 6.5% better than a system without an L3.

3.3 Traffic Ratio of a System With an L3

Given the number of memory accesses in a system, we may determine the traffic ratio of the system with an L3 compared to the system without the L3. Equation 2 indicates this traffic ratio relationship as the amount of traffic (i.e. number of bytes transferred) in a L3-equipped system to the traffic in a non-L3 system.

$$\text{traffic ratio} = \frac{(\text{\# of L3 to memory xfers})(\text{L3 block size})}{(\text{\# of L2 to memory xfers})(\text{L2 block size})} \quad (2)$$

The number of memory transfers reflects both the number of misses to the respective cache that require a memory access and the number of write-backs that the cache has to perform to system memory.

Using Equation 2, we may calculate the traffic ratio for integer, floating point, integer/floating point mixed benchmarks in each of the L3 configurations. As with the AMAT, these configurations are further divided into uniprocessor, 2-way MP and 4-way MP models. When an L3 is included in a system, the amount of traffic on the system bus to main memory decreases substantially over a similar system without an L3. As the results in Figure 7 indicate, the traffic ratio decreases as the size of the L3 and its associativity are increased (the legend in the figure reflects the configuration of the L3 -- MB associativity). However, in the case of a 4-way MP system, the 1MB L3 proves to be detrimental to the traffic ratio for the floating point benchmark test. This result is not surprising given that the floating point benchmarks tend to cause more castouts which require more data to be written back to memory. Given a small 1MB L3, such castouts would be even more prevalent due to the higher probability of a capacity or conflict miss with 4 different processors competing for a smaller shared cache resource. Nevertheless, for all other L3 configurations with varying numbers of processors, the amount of traffic to system memory is reduced by a minimum of approximately 25%. Moreover, for the best L3 configuration (4MB/8-way set associative) running in a 4-way MP system, the traffic on the system bus is reduced by an average of 95% over a system without an L3.

4.0 Conclusions

In this paper, we presented simulation studies quantifying the performance of a shared L3 cache in the memory hierarchy of a multiple processor system. It is seen that an L3 used in an MP-configured system proves to be useful in decreasing the overall memory access time and system bus traffic over a system without an L3. For the specific scenarios and configurations we ran, the results reflect that most of the L2 miss accesses that normally would have been issued to system memory are actually processed and handled by the much faster L3. The average memory access time (AMAT) decreases by approximately 10% for the SPEC integer benchmarks and 3.5% for the floating point benchmarks. The most significant observation has been the dramatic reduction in bus traffic, which reduces up to 95% in some cases.We believe that our study will provide valuable information to computer architects and designers considering the feasibility of adding further levels to a cache hierarchy.

5.0 Acknowledgments

We would like to thank Bryan Hunt from New Mexico State University and IBM Austin for his C++ cache and trace objects, consultations on the inclusion policy of the L3, access to the 8 1-billion reference SPEC92 traces from NMSU, and CPU time on IBM RS/6000 F50. We would also like to thank Baba Arimilli and Sudhir Dhawan of IBM Austin for consultations regarding the practical application for an L3 in a server environment and the impetus for this project.

6.0 References

[1] Arimilli, Baba. Private Communication at IBM Austin, April 30, 1997.

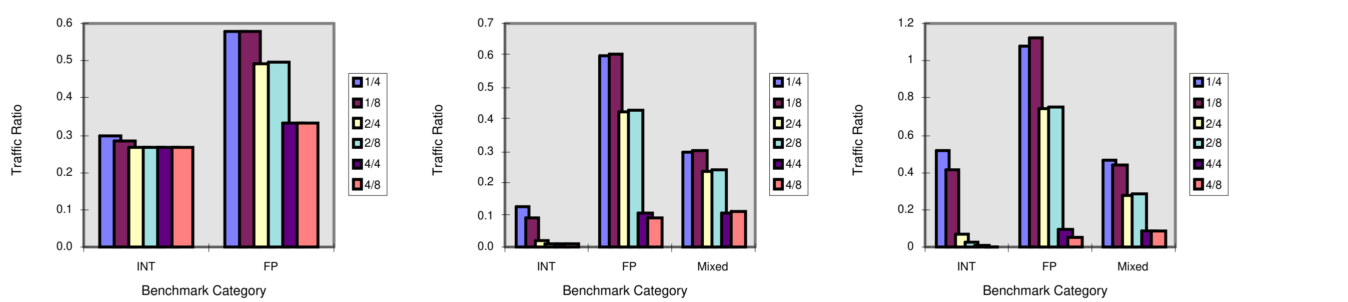
[2] Hennessy, John L. and David A. Patterson. Computer Architecture: A Quantitative Approach, Second Edition. San Francisco: Morgan Kaufman Publishers, Inc., 1996.

[3] Hunt, Bryan. "ACS." *http://atanasoff.nmsu.edu/~acme/acs.html*.

[4] Hunt, Bryan. Private Communication at IBM Austin, March and April 1997.

[5] IBM RS/6000. "RS/6000 Model F50." IBM RS/6000 Division, *http://www.rs6000.ibm.com/hardware/enterprise/f50.html*, April 1997.

[6] Johnson, Eric J. and Jiheng Ha. "PDATS: Lossless Address Trace Compression For Reducing File Size And Access Time." Proceedings, 1994 IEEE



(a) (b) (c)

Figure 7. Traffic ratios for (a) uniprocessor, (b) 2-way MP, and (c) 4-way MP configurations

International Phoenix Conference on Computers and Communications, 1994, pp. 213-219.

[7] Motorola. “256K x 18 Bit Pipelined BurstRAM(TM) Synchronous Fast Static RAM.” Motorola Semiconductor Technical Data, *http://www.mot-sps.com/fsram/productupdate/synch.html*, Revision 1, 8 November 1996.

[8] Olukotun, Kunle et. al. “The Case for a Single-Chip Multiprocessor.” ASPLOS VII, 1996, pp. 2-11.

[9] Parallel Architecture Research Lab. *http://tracebase.nmsu.edu*.

[10] PowerPC(TM) 604 RISC Microprocessor User’s Manual. Motorola Inc., 1994.

[11] Saulsbury, Ashley et. al. “Missing the Memory Wall: The Case for Processor Memory Integration.” ISCA, 1996, pp. 90-101

[12] D. Bhandarkar and J. Ding, “Performance Characterization of the PentiumPro Processor”, Proceedings of the 3rd High Performance Computer Architecture Symposium, 1997, pp. 288-297.

[13] C. E. Kozyrakis et. al, “Scalable Processors in the Billion Transistor Era: IRAM”, IEEE Computer magazine, September 1997, pp. 75-78.

[14] L. Hammond, B. A. Nayfeh and K. Olukuton, “A Single-Chip Multiprocessor”, IEEE Computer magazine, September 1997, pp. 79-85