Branch Prediction X Performance: an analysis on Superscalar Processors

Guilherme D. Pizzol, Maurício L. Pilla, Phillipe O. A. Navaux
Instituto de Informática, Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves 9500 - Campus do Vale - Bloco IV
Bairro Agronomia - Porto Alegre - RS -Brasil
CEP 91501-970 Caixa Postal: 15064
{gpizzol, pilla, navaux}@inf.ufrgs.br

Abstract

Simulation is the most used and efficient method to design new processors. It can reproduce and consider the parameters and variables of a real processor execution. Branch prediction is one of these parameters, being very important in the research and design of newer and better processors. In this way, this paper presents a study on the impact of the branch prediction accuracy in the final performance of superscalar architectures. The results were obtained by simulation, using some of the integer and floating-point benchmarks (ammp, equake, gcc, gzip, mesa, vpr) provided by SPEC2000. *Simprevar*, a variable accuracy branch prediction simulator based on one of the simulators included in the SimpleScalar Tool Set, was used to simulate different prediction accuracies. Our simulations results leads us to conclude that, in some situations, it is better enlarging the hardware than trying to get better accuracy predictors, achieving very similar results.

Keywords **superscalar processors, branch prediction, SPEC 2000, SimpleScalar**

I. INTRODUCTION

Superscalar processors increase the processing performance by the concurrent instruction execution [JOH91]. They explore the instruction's parallelism by making use of multiple stages pipelines. To get maximum performance of the pipeline and, consequently, higher IPC, a high number of instructions must be inserted in the pipeline.

Even if the pipeline fetch stage can deliver a high amount of instructions per cycle, nowadays processors cannot use such number of instructions due to some properties: resource conflicts, data dependence and control dependence [JOH91].

The resource conflicts occur when two or more instructions compete for the same resources [STA96]. Increasing the number of these resources can reduce this problem, but sometimes it is not possible.

Data dependence decreases system performance by stalling the pipeline. One instruction may need the data that is being calculated by another instruction, so it has to wait the writeback of that instruction. There are several methods that try to reduce this problem, but they are out of the scope of this work as well as resource conflicts.

On the other hand, control dependence is one of the problems that affect the performance of superscalar processors, as said before. It limitates the basic blocks' size, decreasing the throughput of instructions to the execution stages. Branch prediction is the most used method to decrease branch penalties, obtaining a higher instructions executed per cycle (IPC).

By simulations, it is possible to notice the impact of branch prediction accuracy in the final performance of SMT (Simultaneous MultiThreaded) and Superscalar architectures. In this way, this paper will show the branch prediction impact in superscalar architectures, using some benchmarks from SPEC 2000 [SPE00] as the base of the simulations.

The rest of this paper is organized as follows. In the next session a briefly presentation about the superscalar architecture will be made. After, in section 3, an analysis of the different branch predictors found in superscalar processors will be made. Section 4 will show the implementation of the variable accuracy branch prediction simulator and, also, the simulation environment. The next sessions, 5 and 6, will show the obtained results of the simulations made with some benchmarks from SPEC2000 [SPE00].

II. THE SUPERSCALAR ARCHITECTURE

Superscalar processors started appearing in the final 80's and were incorporated by high-performance hardware manufacturers as a technological standard. Usually, a superscalar processor has the subsequent characteristics:

- strategies of fetching multiple instructions per cycle, foreseeing conditional branches;
- methods for determination and treatment of data dependency among registers;
- methods for multiple instructions dispatch;
- resources for the parallel execution of multiple instructions, including: multiple functional units and memory hierarchy to allow multiple accesses;
- methods for data communicating through memory by write/read instructions.

Figure 1 illustrates the organization of a typical superscalar processor's pipeline. The main stages of this pipeline are: fetch, decode, dispatch, issue, execution and commit. The branch prediction can be done in the fetch stage, in the decode stage, or combined in this two stages as in PowerPC 620 [DIE95].

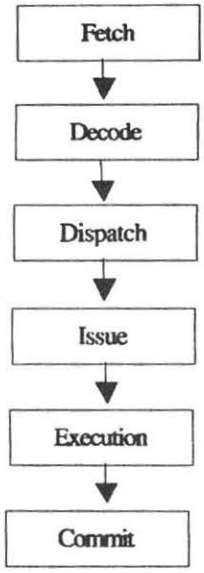

Fig 1. A simple superscalar pipeline

A lot of architectures present another structures and stages, as renaming tables and prefetch mechanisms. Another diversity that exists among them is the instruction set used. While some of them use RISC instruction sets (MIPS, SPARC and PowerPC), others use CISC instruction sets (Intel and AMD). One must have in mind that, over this general organization, there is a pipeline implementation where the specific stages may or may not be aligned with the main phases of the superscalar execution.

In the fetch stage, the instruction fetch and, usually, the branch prediction are performed, accessing the instruction cache (I-Cache) and the branch prediction's tables. The instructions are fetched into the instruction queue. The decodification and issue stages are responsible for the decodification of the instruction in the instruction queue and for their issue to the issue queue associated with the functional units, for later issue and execution. After the instruction's issuing to the issue queue, if they are already ready without non-resolved operands, they are sent to the functional units. During the dispatch, the processor may allocate entries in the reordering queues. As the instructions may be issued to different functional units, and by the way that these instructions may be executed when they are ready, these instructions may be executed out-of-order. So, the processor needs to rearrange these out-of-order executed instructions, making use of reordering queues.

In the functional units, instructions are executed in one or more CPU's cycles, depending on the type of instruction (integer, floating-points, branches, memory), being able to access the data memory. The last pipeline stage, the commitment, is responsible to retire and arrange instructions executed by the previous stage. It also refreshes the data caches and the register bank.

III. Branch Prediction

Due to the high occurrence rate of branch instructions, a lot of techniques were developed to reduce the cost of this kind of instruction. These techniques were developed both in software and in hardware. The techniques developed in software are used during the compilation of an application. Among them, we can mention: branch folding, function in-lining, loop unroll and delayed branch [FER92] [HWA93].

On the other hand, techniques developed in hardware are used during the execution of an application. They are implemented directly in the CPU's core. These techniques are divided in two categories: static and dynamic. In the static techniques the prediction occurs based on the definitions made in a new processor's project time. The dynamic ones dynamically make the predictions based on the information gathered in execution time.

The dynamic prediction is the most used technique in real superscalar processors to reduce conditional branches penalties. There are a lot of prediction mechanisms, but, in general, they all must execute the four subsequent steps [SMI95]: conditional branch recognition, determination of the branch result (taken or not-taken), target calculation and control transfer by fetch redirecting.

Dynamic prediction is done by a table that saves the branch history. This table is called BTB (Branch Target Buffer), being used to predict the target to be followed, based on the previous results. The BTB is organized as a cache, where each entry consists of the address of the branch instruction, a field for prediction and the target address for that branch (Fig. 2). One possible option for prediction is to keep a counter in each BTB entry [SMI95]. When a branch is taken, the counter is incremented (until a highest value), otherwise it is decremented (until a lowest value). In this way, the BTB can keep the dominating result for each branch. Usually, this counter has one or two bits.

.	.	.
.	.	.
Address (LSB)	History Bits	Target Information
.	.	.
.	.	.

Fig. 2 – BTB´s structure

The BTB works like this: the fetch stage compares the instruction's address against the BTB's addresses. If the address is in the BTB, then a prediction is made based on the corresponding history bits. If the prediction says that the branch will be taken, then the address in the destination field is used to access the next instruction. When the branch is resolved, in the execution stage, the BTB can be corrected with the right information about what happened with the branch, in the case of a previously wrong prediction.

After this, Yeh and Patt proposed the idea of collecting dynamically two levels of branch history [YEH91]. The first level keeps the history of the last K branches found. The second level stores what happened with the last J occurrences of a specific standard for the K branches.

The first level is called History Register Table and the second level as Pattern Table. The branch address is mapped to access the first level as in a normal BTB. After mapping the right entry, the history register (Branch History Register) gives the standard bits that will determine which entry will be accessed in the second level. Accessing the second level, the mechanism has the prediction bit that will indicate the path to be followed by the fetch stage to access the next instructions.

After the branch execution, the result (taken or not-taken) is shifted inside the history register of the corresponding entry on the first level, modifying the standard for the branches that will map that entry. The state transition logic evaluates the branch result with the previous prediction and gives the new prediction bit that will be used in the future.

From 1996, new hybrid and multi-hybrid predictors started appearing. Hybrid predictors include many techniques, all of them working in parallel, but only the technique with the better correctness probability gives the prediction's result to the fetch unit. At this moment, it is possible to get prediction accuracy near to 100% for some programs [KES99], but, the percentage of remaining branches is still sufficient to decrease the performance reached by superscalar architectures.

IV. The SimpleScalar Simulators

The SimpleScalar Tool Set [AUS97] is a suite of execution-driven simulators used to evaluate the performance of caches, branch predictors and state-of-art superscalar processors. Eight simulators are included in the suite, from a fast and functional simulator to a detailed timing simulator, the *sim-outorder*.

Sim-prevar is an extended *sim-outorder* that features a new branch prediction mechanism [GON00]. Using this mechanism, the user can introduce any desired rate of branch prediction accuracy comprised between 0 and 1 (representing 0 to 100% of accuracy in the prediction).

When a branch instruction is reached, the mechanism draws a random number also comprised between 0 and 1. If this number is smaller than desired rate, the predictor forces a good prediction. In this case, if the target or the direction was wrong, the predictor corrects to right values, otherwise it keeps the original prediction. On the other hand, if the random number is above the desired rate, the predictor forces a misprediction. In this case, if the target was right, the predictor redirect the fetch to the wrong address, adding the correct penalty, otherwise it keeps the prediction. In this moment, only direction misprediction is simulated, the target is always predicted correctly when a branch is correctly predicted taken. Therefore, target mispredictions are not considered in this version.

The results that will be shown in the next session were obtained using the simulator described above. Six benchmarks from SPEC2000 [SPE00] were used in the simulation, containing floating-point applications (ammp, equake e mesa) and integer applications (gcc, gzip, vpr). We have chosen all benchmarks written in C, to avoid interference of different compilers. Moreover, three benchmarks are well known and used applications (gcc, gzip and mesa).

- gzip: The gzip benchmark is the popular data compression software, but it realizes all compression/decompression in memory;
- gcc: The gcc benchmark is based on the GNU gcc v2.7.2.2, generating code for the 88100 processor from Motorola. The compiler has had its inlining heuristics altered to inline more code, resulting in more timing analyzing its source code inputs, and using more memory;
- vpr: The vpr benchmark is a FPGA (Field-Programmable Gate Array) partitioning and routing software and, algorithmically, belongs to combinational optimization class of programs;
- ammp: The ammp benchmark models large systems of molecules, running molecular dynamics on a protein-inhibitor complex that is embedded in water;
- mesa: The mesa benchmark is a free library like OpenGL that, from a 2D scalar field produces a PNG image;
- equake: The equake benchmark simulates the propagation of elastic waves in large and heterogeneous valleys, like San Fernando Valley, in California.

The benchmarks were compiled using the cross-compiled gcc v2.6.3 provided by the SimpleScalar Tool Set. The optimization level chosen was the greatest supported by the compiler (-O3 -unroll-loops).

Two different configurations were simulated. The first one consists on a configuration with few amount of hardware, representing personal computers used nowadays.

The second one represents a configuration with more hardware, representing servers and future personal computers generations. Table 1 shows these configurations, where IA = integer ALU, FA = floating-point ALU, LS = load/store units, IM = integer Mult/Div and FM = floating-point Mult/Div.

TABLE I
SIMULATION CONFIGURATION

	Superscalar I	Superscalar II
Width	8	16
L1 cache	32K	64K
L2 cache	512K	1024K
FU's	3IA, 3FA, 2LS, 1IM, 1FM	5IA, 5FA, 2LS, 1IM, 1FM
RUU	32	64
LSQ	16	32

V. BRANCH PREDICTION X PERFORMANCE

In figures 3 and 4, it is possible to see the Superscalar I (SS I) and the Superscalar II (SS II) performances, respectively. We can notice that architecture SS II reaches higher performance (IPC) than SS I, as expected.

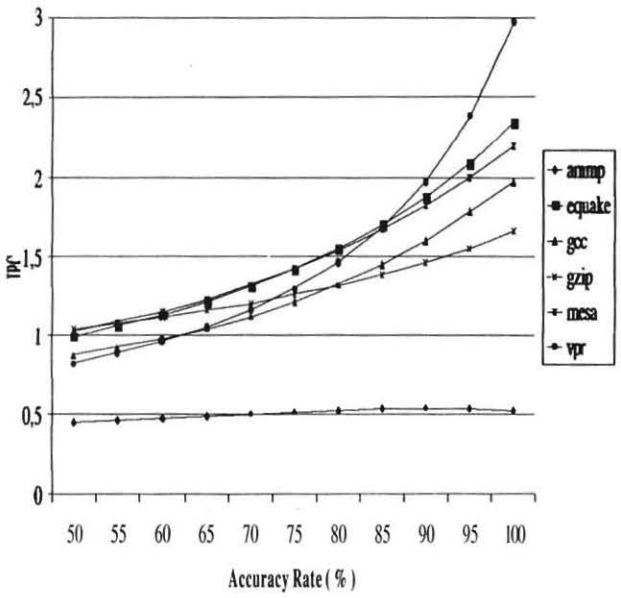

Fig. 3 – Superscalar I – IPC

The ammp's performance does not alter significantly with the increase in the branch prediction accuracy. This happened because the benchmark had a higher miss rate in cache L2 and in data cache. In this way, it can be noticed that after 90% the performance decreases. As we have a higher correct instructions flow, it would be necessary a higher ready data volume, what is not happening because of the higher miss rate in caches.

Another important characteristic is that some benchmarks take more advantage on the accuracy increase than others, as we can see in vpr. This benchmark, with an accuracy of about 50%, has a lower IPC than equake, gzip, gcc and mesa. With about 85-90%, its growth is very big, being the benchmark with the best performance in ways of IPC.

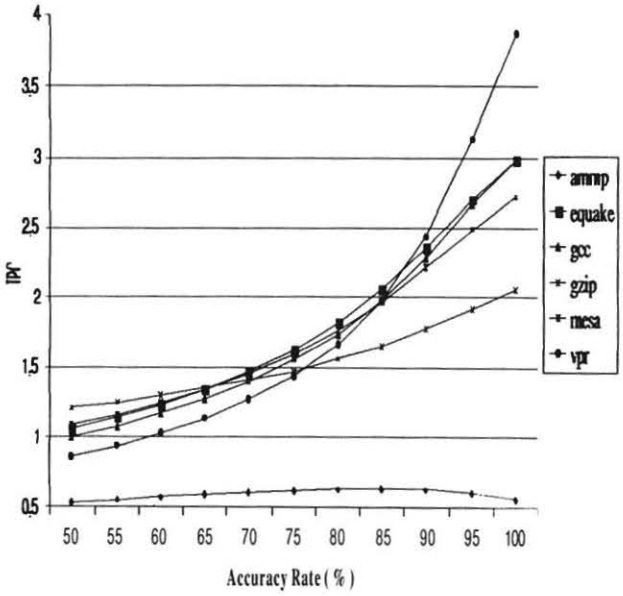

Fig. 4 – Superscalar II – IPC

Analyzing figure 4, it is possible to see that the ammp benchmark presented the same previous behavior, confirming that this benchmark does not take advantage of the accuracy increase. And also, the amount of hardware inserted in SS II does not increase its IPC as it increased the other benchmark's IPCs. Once more, we notice that vpr takes more advantage on the accuracy increase. With an accuracy of about 50%, this benchmark has one of the worst IPC, but when the accuracy reaches about 85% its performance increases exponentially.

Figure 5 shows that the distance between the performances of both architectures increases according to accuracy increase. This happens because when there is more available hardware the instruction level parallelism is better exploited. However, this advantage just is possible when there is an efficient predictor and vice-versa. And besides, with a better predictor, more easily, also, the ILP will be better exploited because fewer instructions will be speculated on the wrong path, increasing the instruction rate executed per cycle.

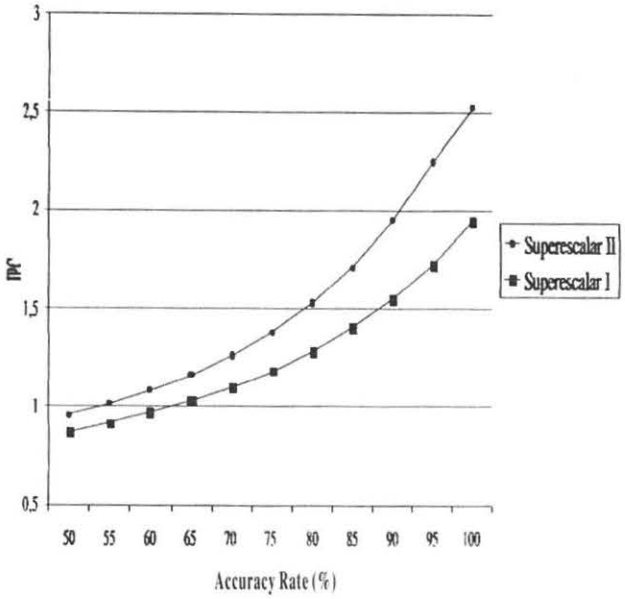

Fig. 5 – Average IPC

In figures 6, 7 and 8 we have the IPC speedup when the accuracy goes from 85% to 100% in the architectures SS I e SS II, respectively. The ammp benchmarks does not appear in these figures due to its instable comportment in these simulations.

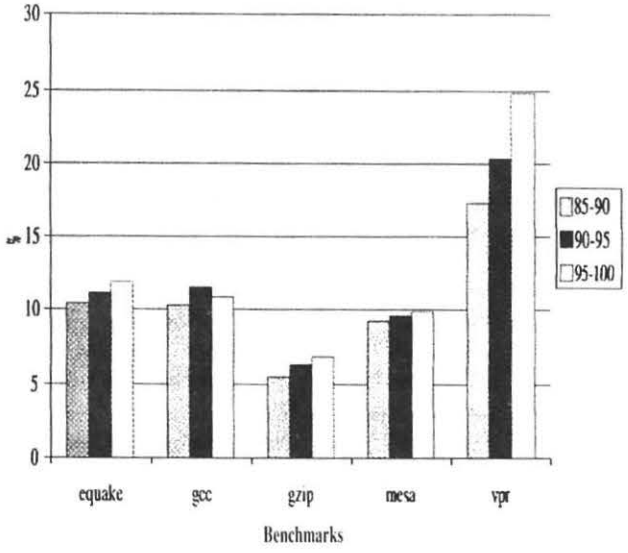

Fig. 6 – Performance Speedup - SS I

These figures reinforce what was exposed before. With the accuracy increase, the IPC also increases. Although, in SS I architecture, the accuracy increase corresponds, in most cases, in a higher IPC increase than in previous bands. On the other hand, it is possible to note that in SS II, when we increase the accuracy from 95 to 100%, the IPC increase is not so big as in another bands (85-90 and 90-95).

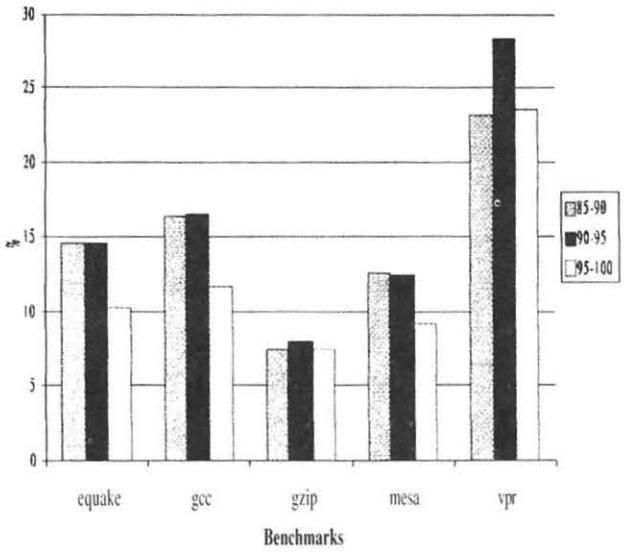

Fig. 7 – Performance Speedup – SS II

This was caused by an unbalanced amount of hardware. In SS II, the fetch and dispatch width and the caches were enlarged, as well as the number of functional units. Therefore, the number of functional units was not enough when a large amount of instructions were inserted in the pipeline. The number of cycles which the RUU was full increased too much due to the lack of functional units, stalling the pipeline, consequently, not allowing the IPC to increase as it were expected.

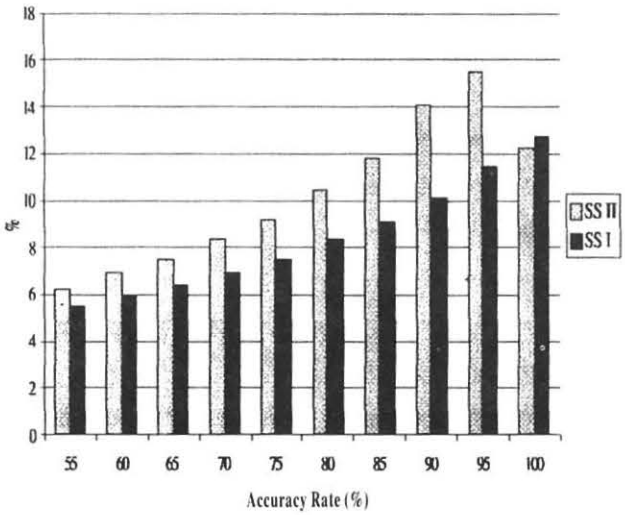

Fig. 8 – Performance Increase – SS I x SS II

VI. Conclusions and Future Work

With *sim-prevar* it is possible to do a lot of studies about the behavior of different architectures towards the branch predictors accuracy. Thus, it is possible to know, for example, if it is a valid effort investing on a predictor to increase its accuracy rate from 90 to 95% or from 95 to 100%. With this study, it was possible to notice that some applications can take more advantage then others with the accuracy increase, only depending of how its source code is formed. And besides, it is easier to exploit the instruction level parallelism when we have more hardware available and higher branch prediction accuracy.

It was possible to note that, not always that the accuracy is increased, we get a performance increase in the same order. Besides, the performance obtained by the SS I architecture with a higher accuracy rate may be reached by the SS II architecture with less accuracy rate, as we could see in figure 5. The performance obtained by SS I with accuracy of about 85% and 100% is similar to the performance obtained by SS II with 75% and 90%, accordingly. This lead us to conclude that it is possible to redirect the efforts in making a better predictor to enlarge the hardware, as, for example, caches, functional units and bandwidth, reaching the same performance.

We must have in mind that to get the maximum performance increase, we need, not only, to invest in the other parts of the hardware, but we must correctly enlarge it to avoid creating an unbalanced architecture and, consequently, stalling the pipeline.

Having a good accuracy and few/unbalanced hardware, we only change the bottleneck's place. It is not worth feeding the pipeline with a higher instruction flow if it cannot èxecute them, even being by the lack of functional units or ready data.

In our simulations, as was said before, were used some of the benchmarks provided by SPEC2000. Despite of the strange behavior of ammp benchmark, the others benchmarks showed that SPEC2000 is more up to date to simulate state-of-art architectures than SPEC95. It is also more exigent in terms of amount of hardware to be simulated than its antecedent.

In the next stage of this research, the source code and assembly code with different levels of optimizations will be analyzed. With this, we wait achieve a better knowledge about SPEC 2000 internals and the differences among the performances of each benchmark.

VII. References

[AUS97] AUSTIN, T. M., **A User's and Hacker's Guide to the Simplescalar Architectural Tool Set,** Intel MicroComputer Research Labs, January 1997.

[BUR97] BURGER, D., AUSTIN, T. M., **The SimpleScalar Tool Set, Version 2.0**, Technical Report #1342, University of Wisconsin – Madison, June, 1997.

[DIE95] DIEP, T.A, NELSON, C., SHEN, J.P., **Performance Evaluation of the PowerPC 620 Microarchitecture**, Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, June, 1995.

[FER92] FERNANDES, E. S. T., SANTOS, A. D., **Arquiteturas Super Escalares: Detecção e Exploração do Paralelismo de Baixo Nível**, VII Escola de Computação, Gramado-RS, Agosto de 1992.

[FER97] FERNÁNDEZ, A., **Un Análisis Cuantitativo del Spec95.** Universitat Politècnica de Catalunya, Barcelona (Technical Report)

[GON01] GONÇALVES, R. A. L.; PILLA, M. L.; PIZZOL, G. DAL; SANTOS, T. G. S.; SANTOS; R. R.; NAVAUX, P. O. A. **Evaluating the Effects of Branch Prediction Accuracy on the Performance of SMT Architectures**. In. EUROMICRO WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING, 9., 2001, Mantova. Proceedings... IEEE Computer Society, 2001. p. 355-362.

[HWA93] HWANG, K. **Advanced Computer Architecture: Parallelism, scalability, programmability.** New York: McGraw-Hill, 1993.

[KES99] KESSLER, R. E. **The Alpha 21264 Microprocessor**. IEEE Micro, v.19, n.2, March/April 1999

[JOH91] JOHNSON, M. **Superscalar Microprocessor Design**. Englewood Cliffs: Prentince Hall, 1991.

[PIZ00] PIZZOL, G. D., PILLA, M. L., NAVAUX, P. O. A. **Variable Accuracy Branch Prediction with Simplescalar Tool Set.** SIM 2000 - UFRGS Microeletronics Seminar. Torres, Julho 2000.

[SMI95] SMITH, J.E, SOHI, G.S., **The Microarchitecture of SuperScalar Processors,** Proceedings of the IEEE, 83(12), pp.1609-1624, December, 1995.

[SOH90] SOHI, G. S. **Instruction Issue Logic for High-Performance, Interruptible, Multiple Funcional Unit, Pipelined Computers.** IEEE Transactions on Computers, 39(3):349-359, March 1990.

[SPE95] **SPEC CPU 95 Technical Manual**. SPEC Steering Committe, 1995

[SPE00] **SPEC CPU 2000 Technical Manual**. SPEC Steering Committe, 2000

[STA96] STALLINGS, W. **Computer Organization and Architecture: Designing for Performance.** Upper Saddle River: Prentice Hall, 1996.

[TAL95] TALCOTT, Adam R. **Reducing the Impact of the Branch Problem in Superpipelined and Superscalar Processors**. Santa Barbara: UCSB, 1995 (Ph.D. Thesis).

[YEH93] YEH, Tse-Yu; PATT, Yale N. **A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History.** In: ANNUAL INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE, 20., 1993. *Proceedings...* New York: ACM, 1993. p. 257-266