CONTROL FLOW EXPLORATION AND OPTIMIZATION OF SUPERSCALAR PROCESSOR

Priya P. Ravale-Nerkar*

Dr. (Mrs.) Sulabha S. Apte**

Abstract— Design of a microprocessor involves consideration of an optimal microarchitecture for a given objective function and a given set of constraints. Superscalar processing is the latest in along series of innovations aimed at producing ever-faster microprocessors. By exploiting instruction-level parallelism, superscalar processors [1] are capable of executing more than one instruction in a clock cycle. The architectural design of super scalar processor involves a lot of trade off issues when selecting parameter values for instruction level parallelism. The use of critical quantitative analysis based upon the Simple Scalar simulations is done to select optimal parameter values for the processor aimed at performance improvement. This paper aims at finding optimal values for the branch prediction for super scalar processor and determines which processor parameters have the greatest impact on the performance of the system.

Index Terms — Superscalar, Control Hazards, Branch Prediction, Benchmarks, Microprocessor Optimization, Simple Scalar, Instruction Level Parallelism.

* Rearch Scholar, W.I.T., Solapur, Solapur University, Solapur

** W.I.T., Solapur, Solapur University, Solapur

1 INTRODUCTION

There would hardly be any application today, which does not require computer Research to design new architectures. Designing a new microprocessor is a complex process. These design issues typically concern performance, cycle time, power consumption, chip area, reliability, security, verifiability, etc. The task for a designer is to optimize the microarchitecture such that a given objective function is optimized. This paper covers the Control dependence problem in Superscalar Architectures and the resolution method area in detail.

Superscalar architecture is one of the most popularly used architectures in recent high-end processors. It uses various techniques to achieve high system speed like, different branch prediction techniques, software based data dependence etc. Actual implementation of design is a very tedious task. Equally difficult is measuring performance of such a system. Since various factors like branch misprediction, cache miss, data hazards interact with each other in a complex manner. Generally, in any research work, a novel idea is tested for only one of the factors and at the same time it should also be tested to see the performance related to other factors like miss ratio of cache, instruction issue rate, instructions committed per cycle (IPC), instructions executed per cycle etc. In the present investigations, performance is tried out on various workloads in different areas. Every parameter, in the areas branch prediction policies is varied and result of its effect on all the performance factors is considered. The main performance metric is IPC. Finally, we suggest a superscalar processor system, which will give optimum performance.

2 Simulation Tool

Any system can be designed by one of the following three ways:

1. Building the actual prototype and testing
2. Building the analytical model and testing
3. Simulating the model on computer and testing it on Computer.

Today's processor architectures are so complex that for all new developments, the third method is used. There are a few simulators available for development of new architectures. Simulation tools used in computer architectures arere broadly classified in to two categories:

1. Trace driven simulators
2. Execution driven simulators

Trace driven simulators execute an instruction trace. This trace is captured during a previous execution. An instruction trace consists of a sequence of instruction records [64]. Each record contains: Instruction's virtual address, Instruction word, Effective address (if required), Some flags etc. On the other hand Execution driven simulators accept a program as an input and it runs the program, at the same time generating a trace on the fly.

3 SIMPLESCALAR BACKGROUND

SimpleScalar is a freeware tool available for simulating processor architectures and measuring their performance. It is an execution driven simulator [2] with a set of tools that model a virtual computer system with CPU, Cache and Memory Hierarchy. Using the Simple Scalar tools [1], user can build modeling applications that simulate real programs running on a range of modern processors and systems. The tool set includes sample simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and state-of-the-art branch prediction. In this paper branch prediction policies and its related parameters are considered for evaluating the performance of a superscalar processor.

4 EXPERIMENTAL SETUP

To determine the effect of various processor parameters and their interactions, we used sim-outorder from the SimpleScalar tool suite (version 3.0) [3]. This simulator has several processor parameters that can be easily changed. The basic experimental setup used for simulation is shown in below Fig. 1.

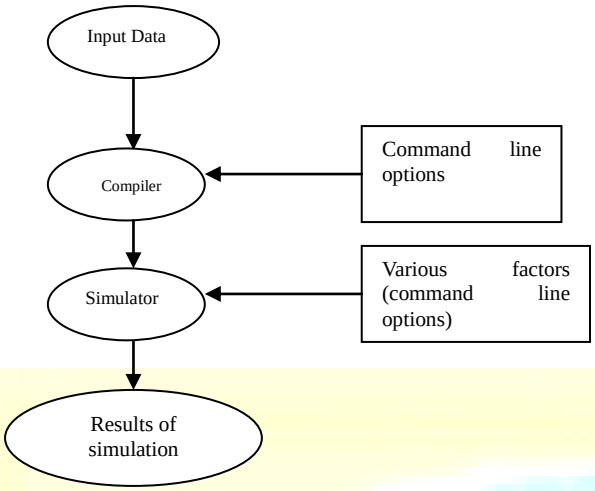

Fig.1. Experimental Setup

5 FACTORS AVAILABLE FOR BRANCHES UNDER CONTROL DEPENDENCES

Control dependence problem arises due to branch instructions. There are many resolution methods on control hazard problem and intensive research is going on, on this problem, even today. A misprediction causes heavy penalty in the form of pipeline flush or pipeline stall, so many cycles are wasted. All resolution methods include hardware for prediction of branch in fetch stage. These methods are commonly known as branch prediction methods. There are various methods for predicting the outcome of the branch [8].

The methods used can be broadly classified into two categories:

1. Static 2. Dynamic

Static methods are dealt with by the compiler, so that when compiler is converting a program in to machine language, it checks the opcode and depending on some preset criteria, decides whether a branch will be taken or not. e.g. in programs, many loop branches have negative direction (‘for’ loop in C) and most of the times they are taken (except for the last one). Therefore all the negative branches can be predicted as ‘taken’. On the other hand in dynamic branch prediction methods, the prediction depends on the past history and the current execution of the program. The information becomes actually available during the execution of the program and depending upon the past history changes dynamically. In Simple Scalar, five resolution methods are facilitated. Out of these five methods, two are static and three are dynamic. The methods are given in table of Table 1.

Sr.No.	Name of Scheme	Type
1	Taken	Static
2	Nottaken	Static
3	bimod	Dynamic
4	2 lev	Dynamic
5	comb	Dynamic

TABLE 1 BRANCH PREDICTION SCHEMES IN SIMPLESCALAR

Two static methods are taken and nottaken, that consider a conditional branch instruction to be 'always taken' and 'always

not taken' respectively. The dynamic predictors 'bimod' is a bimodal branch predictor, which is a single level '2 bit counter (2bc)' stored in Branch Target Buffer (BTB) for every branch. The predictor described in the Fig. 2 is a '2lev'-dynamic 2 level branch predictor. At the first level, it maintains an 'n' bit Branch History Shift Register (BHSR), which maintains past history of past 'n' occurrences of branches, so it is a global branch history register. The second level is an array of two bit counters with the help of which the outcome of the current branch is predicted.

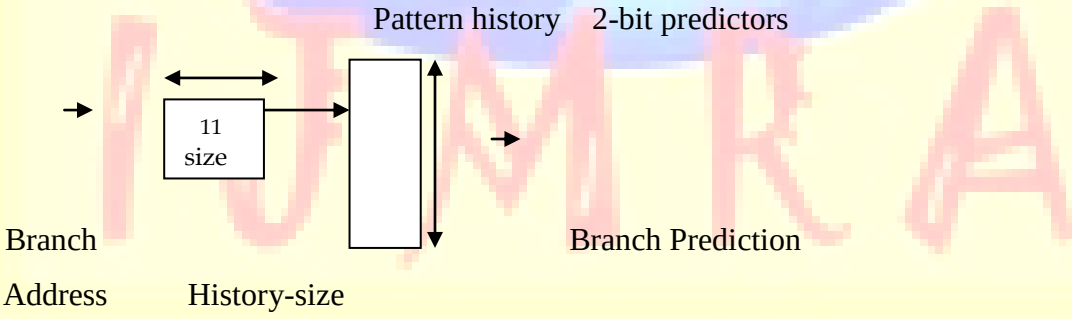

Fig.2. '2-lev' Predictor in Simple Scalar

A 'comb' is a combination of bimodal and 2level predictors. This in fact uses three predictors in parallel. Bimodal, gshare (a type of 2level branch predictor) and a bimodal like predictor. This is shown in a Fig. 3.

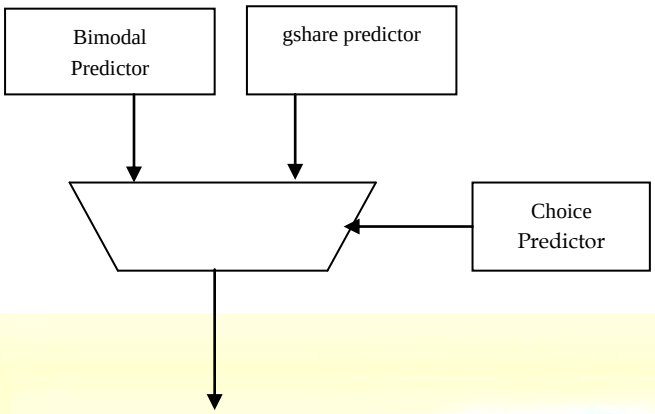

Fig.3. 'comb' Predictor in Simple Scalar

In the initial stages of research, any 'C' program was considered as a workload and it was tested on all simulators. The results were tabulated and analyzed. Such hundreds of thousands of programs were tried and analyzed. Slowly, a clear analysis picture emerged, from which we could take the decision as to which workloads affect the performance and which parameter gets affected. From this analysis, the areas were decided to consider for further experimenting, so that a final decision of optimum performance of the system can be taken.

Areas of benchmarks considered as workloads are as follows:

1. Data structures
2. Operating systems
3. Numerical programs:– including programs like, Series, Pyramid, Pi etc

7 RESULTS OF BRANCH PREDICTION POLICIES WITH DEFAULT PARAMETERS

As stated earlier, two static predictors 'taken', 'nottaken' and three dynamic predictors 'bimod', '2 lev' & 'comb' are available in branch prediction policies. Also, three areas of benchmarks were considered in the research: Small, medium and large. Also,it was observed that both static prediction schemes perform very poor and as for the dynamic schemes, the results are really good for the default prediction policies[7]. Therefore in further discussions only three default dynamic prediction schemes with large workload area are taken into account.

7.1 Performance by varying the default predictors with IPC

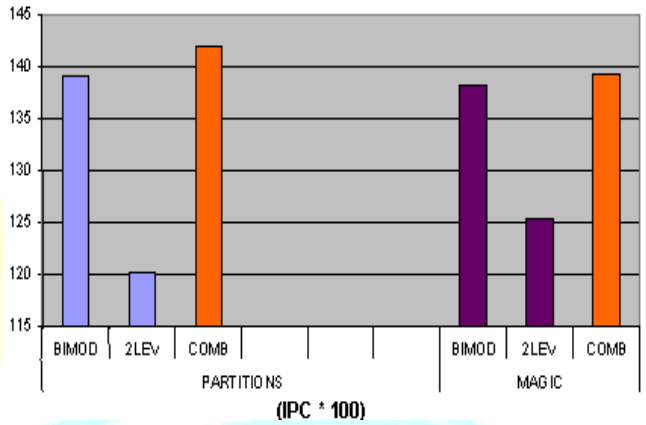

Fig.4. IPC for Default Branch Prediction Schemes with LARGE Work loads

Fig. 4 displays results of large workloads for dynamic schemes in default mode. It is clear from the result that, 'comb' has comparatively increased IPC. The default parameters are further varied to judge the change in performance with IPC.

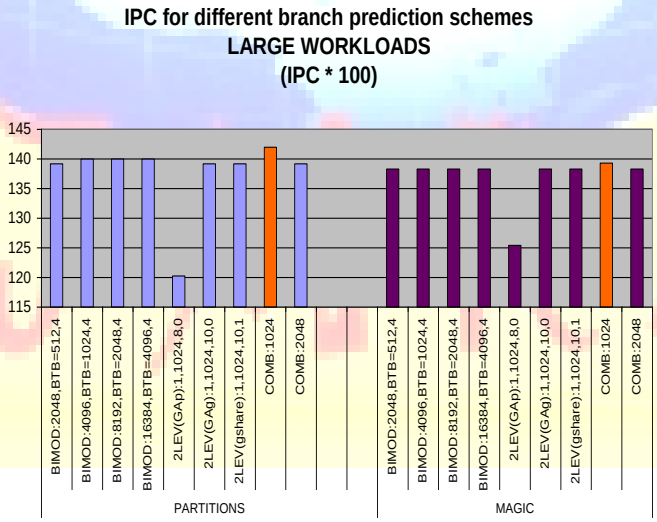

Fig.5. IPC for LARGE Workloads

In the case, shown in Fig. 5, it is observed that, 'comb' has given the maximum IPC, but excepting '2level' (default), all others are very close by. The best choice is still 'comb'. Before selecting the parameters for optimum performance, the combination of 'comb' with others are tried out. That is taken up in the next section.

7.2 Performance of combining 'comb' with other factors

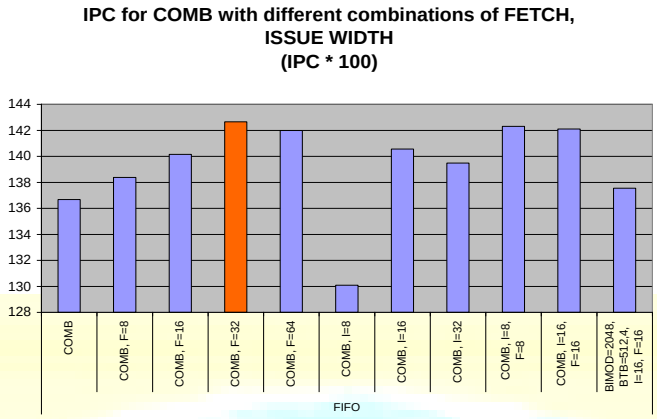

Fig.6. IPC for Combination of Prediction Policies with Fetch and Issue Widths

As shown in fig. 6, increasing fetch width to 32 in combination with default 'comb' gives highest IPC. Further increasing fetch width does not improve the performance, rather it starts going in the negative direction. As for the issue width attached with 'comb', IPC enhancement is observed up to issue width 16. Further it starts declining. If both fetch and issue are combined with 'comb', then the result is different. For both widths with values of 8 and 16, performance boosts up substantially. It is almost at par with combination of fetch=32 and 'comb'.

Before drawing any conclusions, the performance for large benchmarks, prediction policy with other factors are also taken into consideration.

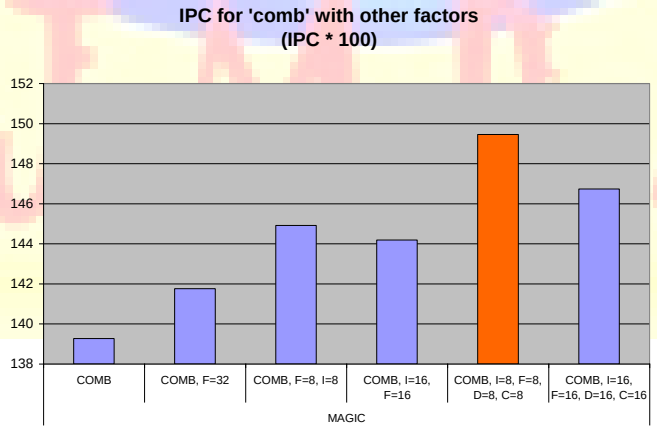

Fig.7. IPC for Combination of Prediction Policies with Fetch, Issue , Commit Widths

As shown in Fig. 7, Maximum IPC is resulted at 'comb' with fetch, issue, decode and commit widths at 8. It gives a sudden decline at all the widths at 16. When in any program, more hard-ware is facilitated; it gives good result only if it can extract the parallelism. It is not always poss-

ible as, it needs software help by compiler. Without that, increased hardware will go waste. The same thing is happening here. The best combination for IPC is then 'comb' with fetch, issue, decode and commit widths at 8. Thus the various combinational results of IPC are considered to decide a combination for optimum result.

8 CONCLUSION

For all the above variations, IPC is also noted down. It is always necessary to consider IPC along with the metric for that particular area. Actually, for superscalar architectures, getting better IPC is more important for improving overall performance of the system.

In the results presented in fig. 4 & 5'comb' has comparatively increased IPC. To decide on other parameters along with 'comb' results of fig 6 are alsotaken into considerations, and it is clearly observed that IPC enhances for widths 8 & 16. Finally results of Fig. 7, shows, maximum IPC is obtained for 'comb' combined with all fetch, issue, decode and commit widths to be 8. Finally, the combination suggested for optimum results is tabulated in table ahead.

Sr. No.	Parameter	Value suggested for optimum result
1.	Prediction Policy	Combinational
2.	Fetch Width	8
3.	Issue Width	8
4.	Decode Width	8
5.	Commit Width	8

TABLE 2 SHOWING PARAMETERS SUGGESTED FOR OPTIMUM RESULTS

REFERENCES

[1] http://www.simplescalar.com

[2] Doug Burger and Todd M. Austin, The SimpleScalar Tool Set.

[3] J. Hennessy and D. Patterson, \Computer architecture: A quantitative approach, fourth edition"

[4] D. C. Burger and T. M. Austin, The SimpleScalarTool Set, Computer Architecture News, 1997.

[5] Doug Burger, James R. Goodman, Billion -Transistor Architecture IEEE 1997.

[6] James E. Smith, Microarchitecture of Superscalar Processors IEEE 1995.

[7] Priya P. Ravale and Sulabha S. Apte, Design of a Branch Prediction Unit of a Microprocessor Based on Superscalar Architecture using VLSI, IEEE2010, V3-355.

[8] Lee and Smith, Branch prediction strategies and branch target buffer design, IEEE-84.

[9] Manoj Ransing and Vishal Mamania, Branch Prediction and Case Studies.

[10] Collins, Wang, et.al. Speculative precomputation: Long range prefetching of delinquent loads, IEEE-2001.

[11] Zilles and Sohi, Execution based prediction using speculative slices, ISCA-2001.

[12] Kevin Skadron, Douglas W. Clark, et.al., Branch Prediction, Instruction-Window Size, and Cache Size : Performance Trade-offs and Simulation Techniques, IEEE 1999.

[13] Krishna Kavi, Georgi, et.al. Scheduled data flow : Execution paradigm, architecture and performance evaluation, IEEE-2001.