

Simulation of Branch Prediction Optimization in Parallel Register Sharing Architecture

Rajendra Kumar and P. K. Singh, *Member IACSIT*

Abstract—In this paper we present simulation of optimization of branch prediction in parallel register sharing architecture to show high degree of ILP. The main idea behind this concept is to use a step beyond the prediction of common branch and permitting the architecture to have the information about the CFG (Control Flow Graph) components of the program to have better branch decision for ILP. The navigation bandwidth of prediction mechanism depends upon the degree of ILP. It can be increased by increasing control flow prediction in procedural languages at compile time. By this the size of initiation is increased that allows the overlapped execution of multiple independent flow of control. The multiple branch instruction can also be allowed. These are intermediate steps to be taken in order to increase the size of dynamic window to achieve a high degree of ILP exploitation.

Index Terms—Basic Block, CFG, CFP, ILP.

I. INTRODUCTION

ILP is the methodology for execution of multiple instructions per cycle. It is now desirable to modern processors for better performance. It has been observed that ILP is greatly forced by branch instructions. Also it has been observed that branch prediction is employed with speculative execution [1]. However, inevitable branch misprediction compromises such a remedy. On the other hand branch prediction exposes high degree of ILP to scheduler by converting control flow into equivalent predicated instructions which are protected by Boolean source operands. The if-conversion [2] has been shown to be promising method for exploitation of ILP in the presence of control flow.

The if-conversion in the prediction is responsible for control dependency between the branches and remaining instructions creating data dependency between the predicate definition and predicated structures of the program. As a result the transformation of control flow becomes optimized traditional data flow and branch scheduling becomes reordering of serial instructions. The degree of ILP can be increased by overlapping multiple program path executions.

Some predicate specific optimization may also be performed as a supplement of traditional sequential computing approaches. The major questions regarding the if-conversion: what to if-convert and when to if-convert explore that the if-conversion should be performed early in

the compilation stage. It has the advantage of classified optimization facilitation on the predicted instructions whereas a delay in if-conversion is scheduled in the time slots for better selection for code efficiency and target processor characteristics. The dynamic branch prediction is fundamentally is restricted to establishing a dynamic window because it can make local decision without any prior knowledge or of global control statement in the code. This short of knowledge creates several problems like (1) branch prediction and (2) its identity. It means the branch must be encountered by parallel register sharing architecture [3].

II. RELATED WORK

The fetch unit has a great role in prediction mechanism [2] in parallel register sharing architecture but Pan, So and Rahmeh (1992) [4], and Yeh Y. Patt (1993) [5] proposed some recent prediction mechanism that do not require the addresses of branches for prediction rather there is requirement of identity of each branch to be known so that the predicted target address can be obtained using either BTB [6] or by decoding branch instructions in parallel register sharing architecture. There are so many commercially available embedded processors that are capable to extend the set of base instructions for a specific application domain. A steady progress has been observed in tools and methodology for automatic instruction set extension for processors that can be configured to exploit ILP. It has been observed that the limited data bandwidth is available in the core processors. This creates a serious performance deadlock. Cong, Han and Zhiru Zhang (2005) [7] represents a very low cost architectural extension and a compilation technique responsible for data bandwidth problem. A novel parallel global register binding is also presented in [7] with the help of hash function algorithm. This leads to a nearly optimal performance speedup of 2% of ideal speedup. A compilation framework [1] is presented that allows a compiler to maximize the benefits of prediction.

Steve Carr (1996) [8] shown how the weakness of traditional heuristics are exploited. Optimal use of loop cash is also explored to release the unnecessary pressure. A technique to enhance the ability of dynamic ILP processors to exploit the parallelism is introduced in [9]. A performance metric is presented in [8] to guide the nested loop optimization. This facilitates ILP with loop as combined optimization.

The impact of ILP processors on the performance of shared memory multiprocessors [10] with and without latency hiding optimizing software prefetching has been

Manuscript received July 10, 2012; revised August 27, 2012.

Rajendra Kumar is with Computer Science and engineering Department, Vidya College of Engineering, Meerut (Uttar Pradesh), India (phone: +91-9412002322, e-mail: rajendra04@gmail.com, website: <http://www.rkronline.in>).

P. K. Singh is with the Computer Science and engineering Department, MMM Engineering College, Gorakhpur (Uttar Pradesh), India (e-mail: topksingh@gmail.com).

represented by Pai, Ranganathan, Shafi and Adve (1999). One of the critical goals in the code optimization for multiprocessor system on single chip architecture [11] is to minimize the number of off chip memory access. A strategy has been represented in [11] to reduce the number of off chip references due to shared data. In contrast to [3], an aggressive register reclamation mechanism targeted to micro-architecture is presented in [12] by Salvador Petit Marti et. al. (2009). Due to normal branch prediction, a prediction can be made while the fetch unit fetches the branch instruction for their execution. Tameesh Suri and Aneesh Aggarwal, (2009) in [13] represent a mechanism to improve the per-core performance while maintaining the scalability. [13] integrates a reconfigurable hardware unit (RHU) in the resource constrained cores for performance improvement.

Static techniques (for example, like trace scheduling [11], [14] predicated execution [15], super block and hyper block scheduling [9], [16], etc.) have been used to promote the impact of control dependencies. Lam Wilson (1992) [17], represents a study that shows the ILP processors which perform branch prediction and speculative execution. But it allows only a single flow of control that can extract a parallelism of only 7.0. The parallelism limit is increased to 13.05 if the ILP processors use the maximal of control dependence information for instruction execution before branches which they are independent.

III. CONTROL FLOW GRAPH CHARACTERISTICS EXTRACTION

The ISB (Instruction Stream Buffer) architecture and the ISB structure are presented in [3] for control flow prediction. The information presented in CFG for a program can be exploited by ISB architecture that presents parallelization of shared register after inspection of control flow graph of a program, it is possible to infer that some of the basic blocks may be executed regardless previous branch outcome. Below is a C language code.

```

for (i = 0; i < input; i++)
{
    a1 = a[0]->ptand[i];
    b1 = b[0]->ptend[i];
if(a1==2)
    a1 = 0;
if(b1==2)
    b1 = 0;
if(a1 != b1)
    {
        if(a1 < b1)
        {
            return -1;
        }
        else
        {
            return 1;
        }
    }
}

```

The Fig. 1 represents a CFG. This shows a number of instructions in each basic block.

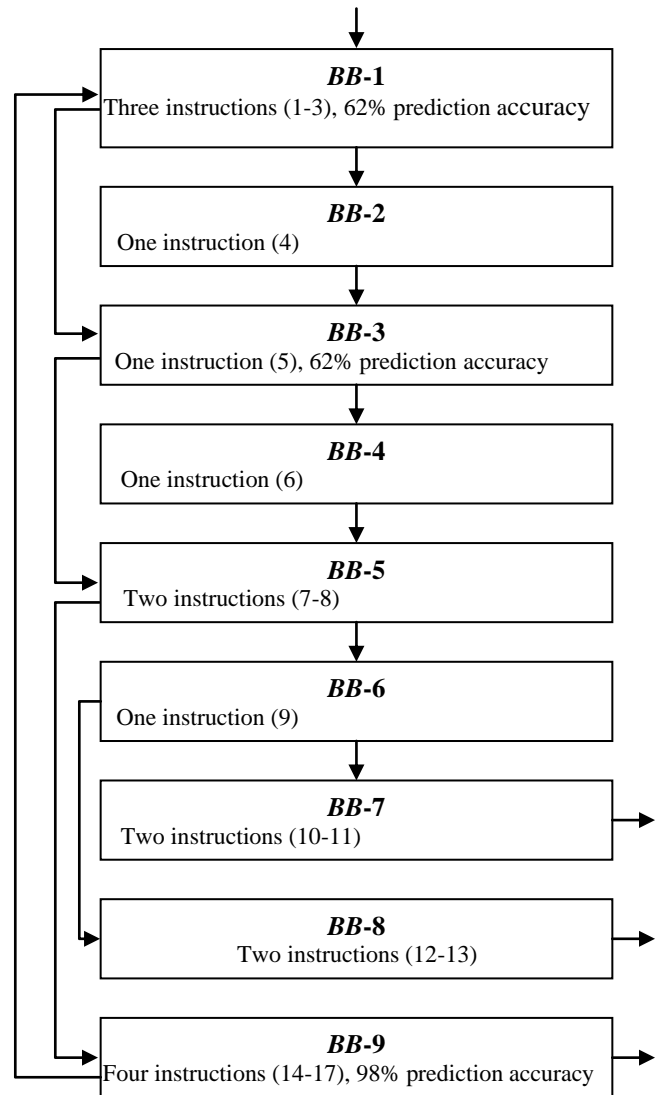


Fig. 1. Control flow graph of above code.

The simulation experiments are performed Trimaran simulator [18] for a MIPS 2000 executable extending from the node BB-1 following multiblock BB-1 to BB-2, BB-1 to BB-3, BB-1 to BB-4, BB-1 to BB-5, and BB-1 to BB-8 with BB-1 to BB-6, BB-1 to BB-7 and BB-1 to BB-9 can not be counted as multiblocks as they have three targets.

A CFG (whose nodes are basic blocks) can be transformed into an equivalent graph whose nodes are multiblocks. The information of multiblock is sent to ISB architecture and informed decisions are navigated through the control free graph. When a multiblock enters then its exit point can be determined easily even though the exact path is unknown.

The execution of multiblocks may overlap each other creating overlapped execution of multiple control flow. The data dependencies between the instructions in multiblocks and parallel register sharing architecture create a platform for a kind of subgraph used in multiblock construction. There are several reasons for restricting the scope of multiblocks. As an instance if the architecture is capable for exploiting inter multiblock parallelism then it could be better to combine the dependent instructions into a single unit (multiblock). Each iteration of data independent loop

can be considered as a multiblock to permit one iteration per cycle initiation. Following code shows loop where iterations are dependent:

```

for(fpnr = xlenv; fpnr; fpnr=cdr(fpnr))
{
for(ep = car(fpnr); ep; ep = cdr(ep))
{
if(sym == car(car(ep)))
return (cdr(car(ep)));
}
}
    
```

As an advantage, an entire loop can be encapsulated to form a multiblock. The code given above is double nested loop. The inner loop is used to traverse a linked list and its execution is dependent of data and control. If we define the entire inner loop to be a single multiblock then there is a possibility of starting several activation of inner loop without waiting for completion of previous one. The flexibility in construction in multiblock is increased by allowing many targets and as a result a larger multiblock is formed. In case, the number of targets are increased the dynamic prediction setup needs additional number of state information and as a result the accuracy of prediction is decreased. Therefore, it allow multiblocks to have maximum two targets that may be compromised. As an exception, when a multiblock has three or more targets then at run time except on or two, all are rarely exercised. The reduced CFG of figure 1 is given by Fig. 2.

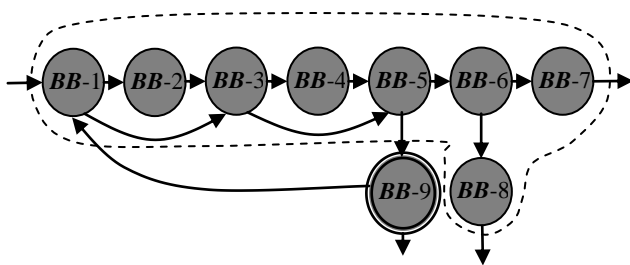


Fig. 2. Reduced CFG of figure 1

Fig. 2 shows a multiblock construction from *BB-1* through *BB-8*. It contains 16 static instructions. An average 7.46 instructions are executed dynamically. The multiblock construction for *BB-9* has 4 instructions.

The first multiblock (*BB-1* to *BB-8*) is called *MB(1-8)* and the second multiblock (only *BB-9*) is *MB(9)*. In this reduced CFG only two predictions are required per iteration of the loop as compare to four predictions in CFG given by above code having double nested loop that an ordinary branch prediction approach would require. Following is the control flow table (CFT) for control flow prediction:

TABLE I: CONTROL FLOW TABLE

Address	Target 1	Target 2	Target 3
MB(1-8)	MB(9)	Return	16
MB(9)	MB(10)	MB(1-8)	4

The control flow prediction buffer (CFPB) is temporary of CFT entries. The CFT entries are appended with sufficient information to help dynamic prediction decision. The CFPB is accessed once for every multiblock activation record to calculate the size and targets of multiblock. The

following table is for CFPB entries of the reduced CFG given by figure 2.

TABLE II: CFPB ENTRIES

Address	State of prediction	Target 1	Target 2	length
MB(1-8)	Taken	MB(9)	Return	16
MB(9)	Taken	MB(10)	MB(1-8)	4

IV. SIMULATION EXPERIMENTS AND ANALYSIS

As discussed earlier, we conducted our experiments on Trimaran Simulator. Trimaran Simulator is an integration of compilation and performance monitoring infrastructure. Trimaran is intended to various compiler techniques like ILP, compiler optimization, retargetable compilation etc., and computer architectures like VLIW, EPIC, etc. We first evaluate the strength of control flow prediction concept on abstract machine that maintains a dynamic window from which ILP is extracted.

For experimental purpose we used *compress*, *gcc*, *SuperMips*, *xlisp*, *yacc* and *tex* coded in C language. The table 3 shows the basic structure for different programs. The programs are evaluated in terms of dynamic instructions, conditional and unconditional branch ratio, static code size, and CFT size.

TABLE III: BASIC STRUCTURE FOR DIFFERENT PROGRAMS

Program Name	Dynamic Instructions (millions)	Conditional Branch Ratio	Un-conditional Branch Ratio	Static code size	Static CFT size
<i>compress</i>	22.68	0.149	0.040	6144	88.5
<i>gcc</i>	1000	0.156	0.042	172032	25653
<i>SuperMips</i>	500	0.111	0.056	14336	1851
<i>tex</i>	214.67	0.143	0.055	60416	9976
<i>xlisp</i>	500	0.157	0.091	21504	3637
<i>yacc</i>	26.37	0.237	0.020	12288	1737

It has been observed that the dynamic window initiates the instructions and the machine executes the instructions. The instructions chosen by the machine at any given time can be from various parts of the dynamic window with different flow of control in the program. The table 4 shows variation in number of branches traversed per cycle without control flow prediction.

TABLE IV: BRANCH TRAVERSAL RESULTS WITHOUT CONTROL FLOW PREDICTION

Program Name	Initiation mean size	Window mean size	Branch prediction accuracy
<i>compress</i>	5.24	64	89.59
<i>gcc</i>	5.02	72	91.12
<i>SuperMips</i>	5.97	320	97.15
<i>Tex</i>	5.02	169	95.87
<i>Xlisp</i>	4.02	143	95.64
<i>Yacc</i>	3.87	103	95.74

The Table V shows variation in number of branches traversed per cycle with control flow prediction. In case of *gcc*, the control flow prediction we observed is 1.47 branches per cycle and in *tex* 1.16 branches per cycle as shown in table 5.

TABLE V: BRANCH TRAVERSAL RESULTS WITH CONTROL FLOW PREDICTION

Program	Initiation mean size	Window mean size	Branch prediction accuracy	Traversed branches per cycle
<i>compress</i>	8.40	86	89.71	1.33
<i>gcc</i>	9.44	105	91.02	1.47
<i>SuperMips</i>	13.24	845	97.72	2.18
<i>tex</i>	6.24	207	96.10	1.16
<i>xlisp</i>	5.11	157	95.34	1.16
<i>yacc</i>	4.96	150	96.51	1.22

The numbers of branches are reduced by control flow prediction. It used traversal of multiple branches in a single prediction. The effect on the accuracy of the branch prediction was not seen uniform across all programs.

V. CONCLUSION

As the prediction decision is over, the instructions from the predicted path are fetched in the next branch as the predicated path is encountered. For any two consecutive arbitrary branches it is sometimes impossible to determine the identity of the next branch to make prediction in the very next cycle when a branch prediction is over.

It was seen that if the branch prediction is not made in each and every cycle then the prediction bandwidth and the number of instructions per cycle are suffered. The prediction mechanism can perform one prediction per cycle as long as the next branch lies inside the block of fetch instruction in the instruction buffer. The number of instruction that can enter into the dynamic window in the cycle is another problem. The best case instruction per cycle is restricted to the number of instruction that can move in to dynamic window. If there is possibility of traversing then only one branch at a time in CFG can be initialized per cycle and average initiation time is restricted by the length of code. As a possible solution of this problem we used multiblocks to traverse multiple branches at a time. This can be achieved by initiating a set of node of control flow graph to execute. The problem of accuracy and the size of dynamic window can be eliminated if some of the branches with low prediction accuracies belong to the if-else structure.

REFERENCES

- [1] E. Quiñones and J. M. Parcerisa, "Improving Branch Prediction and Predicated Execution in Out-of-Order Processors," in *Proc. of 13th International Symposium on High Performance Computer Architecture*, 2007, pp. 75-84.
- [2] D. I. August, W. Mei W. Hwu, and S. A. Mahlke, "The Partial Reverse If-Conversion Framework for Balancing Control Flow and Prediction," *International Journal of Parallel Programming*, vol. 27, issue 5, pp. 381-423, 1999.
- [3] R. Kumar and P. K. Singh, "A Modern Parallel Register Sharing Architecture for Code Compilation," *International Journal of Computer Applications*, vol. 1, no. 16, pp. 108-113, 2010.
- [4] S. T. Pan, K. So, and J. T. Rahmeh, "Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation," in *Proc. of Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pp. 76-84, 1992.
- [5] T. Yeh and Y. Patt, "A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History," in *Proc. of 20th Annual International Symposium on Computer Architecture*, pp. 257-266, May 1993.
- [6] J. K. F. Lee and A. J. Smith, "Branch Prediction Strategies and Branch Target Buffer Design," *IEEE Computer Magazine*, vol. 17, pp. 6-22, 1984.

- [7] J. Cong, G. L. Han, and Z. Zhang, "Architecture & compilation for data bandwidth improvement in configurable embedded processors," in *Proc. of the International Conference on Computer Aided Design IEEE/ACM*, pp. 263-270, 2005.
- [8] S. Carr, "Combining Optimization for Cache and Instruction-Level Parallelism," in *Proc. of the Conference on Parallel Architectures and Compilation Techniques*, pp. 238-247, 1996.
- [9] D. N. Pnevmatikatos, M. Franklin, and G. S. Sohi, "Control flow prediction for dynamic ILP processors," in *Proc. of the 26th annual international symposium on Microarchitecture*, pp. 153 - 163, 1993.
- [10] V. S. Pai, Parthasarathy Ranganathan, Hazim Abdel-Shafi, and Sarita Adve, "The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors," *IEEE Trans. on Computers*, vol. 48, pp. 218 - 226, 1999.
- [11] G. Chen and M. Kandemir, "Compiler-Directed Code Restructuring for Improving Performance of MPSoCs," *IEEE Trans. on Parallel and Distributed Systems*, vol. 19, no. 9, 2008.
- [12] S. Petit Marti et. al., "A Complexity-Effective Out-of-Order Retirement Microarchitecture," *IEEE Trans. on Computers*, vol. 58, no. 12, pp. 1626 - 1639, 2009
- [13] T. Suri and A. Aggarwal, "Improving Scalability and Per-core Performance in Multi-cores through Resource Sharing and Reconfiguration," in *Proc. of the 22nd IEEE International Conference on VLSI Design*, pp. 145 - 150, 2009.
- [14] J. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," *IEEE Trans. on Computers*, vol. C-30, July 1981.
- [15] P. Chang, S. Mahlke, W. Chen, N. Warter, and W. Hwu, "IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors," in *Proc. of the 18th Annual International Symposium on Computer Architecture*, pp. 266 - 275, May 1991.
- [16] S. Mahlke, D. Lin, W. Chen, R. Hank, and R. Bringmann, "Effective Compiler Support for Predicated Execution Using the Hyperblock," in *Proc. of the 25th Annual Workshop on Microprogramming and Microarchitecture*, pp. 45 - 54, 1992.
- [17] L. Wilson, "Limits of control flow on parallelism," in *Proc. of 19th annual International symposium on Computer Architecture*, pp. 46 - 57, 1992.
- [18] Trimaran an integrated compilation and performance monitoring infrastructure developed and maintained by the CCCP Group at the University of Michigan, and the Commit Group at MIT, [Online]. Available: www.trimaran.org



Rajendra Kumar obtained Bachelor of Engineering from BIET Jhansi, Master of Technology from UPTU Lucknow, in Computer Science & Engineering. His area of interest includes Theoretical Computer Science, Human Computer Interaction, Computer Graphics, Biometric Systems, ICT, etc. His current research area is Instruction Level Parallelism.

He is Associate Professor and Head of Computer Science & Engineering department at Vidya College of engineering, Meerut. He is active reviewer of Journal of Computational Biology and Bioinformatics Research, Nairobi. He is author of four text books including Theory of Automata, Languages & Computation from McGraw Hill, and eight distance learning books for MDU Rohtak, CDLU Sirsa, MGU Kerla, etc.

Prof. Kumar is member of IACSIT Singapore, CSTA USA, IAENG Hong Kong, ISTE New Delhi and Amnesty International UK. Prof. Kumar also has been Member of Board of Studies of Uttar Pradesh Technical University, Lucknow.



P. K. Singh graduated from M. M. M. Engineering College, Gorakhpur with a Bachelor of Computer Science degree and M.Tech. from University of Roorkee (now IIT Roorkee) in Computer Science and Technology then obtained a Doctor degree in the area of Parallelizing Compilers. He teaches a number of Computer Science subjects including Compiler Design, Automata Theory, Advanced Computer Architectures, Parallel Computing, Data Structures and Algorithms, Object Oriented Programming C++ and Computer Graphics etc., but mostly he teaches Compiler Design and Parallel Computing.

He is an Associate Professor of Computer Science & Engineering at MMM Engineering College, Gorakhpur. Prof. Singh is author of Computer graphics book. He is member of several societies and bodies of Different organizations and universities.