Control Flow Prediction through Multiblock Formation in Parallel Register Sharing Architecture

Rajendra Kumar
Dept. of Computer Science & engineering,
Vidya College of Engineering
Meerut (UP), India
rajendra04@gmail.com

Dr. P K Singh
Dept. of Computer Science & engineering
MMM Engineering College,
Gorakhpur (UP), India
topksingh@gmail.com

***Abstract* - In this paper we introduce control flow prediction (CFP) in parallel register sharing architecture. The main idea behind this concept is to use a step beyond the prediction of common branch and permitting the hardware to have the information about the CFG (Control Flow Graph) components of the program to have better branch decision in navigation. The degree of ILP depends upon the navigation bandwidth of prediction mechanism. It can be increased on increase of control flow prediction. By this the size of initiation is increased that permit the overlapped execution of multiple independent flow of control. It can allow multiple branch instruction that can be resolved simultaneously. These are intermediate steps to be taken in order to increase the size of dynamic window can achieve a high degree of ILP exploitation.**

Keywords: CFP; ISB; ILP; CFG; Basic Block

I. INTRODUCTION

ILP allows the capability of execution of multiple instructions per cycle. It is now essential to the performance of modern processors. ILP is greatly forced by branch instructions. Also it has been observed that branch prediction is employed with speculative execution [2]. 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 has been shown to be promising method for exploitation of ILP in the presence of control flow.

The if-conversion in the predication goes for control dependency between the branches and remaining instructions into 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. ILP can be increased by overlapping multiple program path executions. Some predicate specific optimization may also be enabled as some supplement of traditional approaches. The major questions regarding the if-conversion [2]: what to if-convert and when to if-convert explores that the if-conversion should be performed early in the compilation stage. It has the advantage of classified optimization facilitation on the predicated instructions whereas the delay in if-conversion is schedule in the time leaves 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 makes local decision without any prior knowledge or of global control statement in the program. 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 [1]. Due to normal branch prediction, a prediction can be made while the fetch unit fetches the branch instruction.

II. RELATED WORK

The fetch unit has a great role in prediction mechanism [2] in parallel register sharing architecture but [13, 15] proposes some recent prediction mechanism that do not require the addresses of branch for prediction rather there is requirement of identity of branch to be known so that the predicted target address can be obtained using either BTB [11] or by decoding branch instructions. 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. It has been seen that the limited data bandwidth is available in the core processors. This creates a serious performance deadlock. [8] represents a very low cost architectural extension and a compilation technique that creates data bandwidth problem. A novel parallel global register binding is also presented in [8] with the help of hash function algorithm. This leads to a nearly optimal performance speedup of 2% of ideal speedup. A compilation framework [5] is presented that allows a compiler to maximize the benefits of prediction. A heuristic

[14] is given through experiments on Trimaran simulator [18]. [14] shows 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 [6]. A performance metric is presented in [14] to guide the nested loop optimization. This way the effect of ILP is combined with loop optimization.

The impact of ILP processors on the performance of shared memory multiprocessors with and without latency hiding optimizing software prefetching is represented in [16]. One of the critical goals in the code optimization for multiprocessor system on single chip architecture [17] is to minimize the number of off chip memory access. [17] represents a strategy that can reduce the number of off chip references due to shared data.

Static technique like trace scheduling [4, 7] predicated execution [9] super block and hyper block scheduling [3, 12] have been used to elevate the impact of control dependencies. [10] 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. The parallelism limit is increased to 13 if the ILP processors use the maximal of control dependence information for instruction execution before branches which they are independent.

III. EXPLOITATION OF CONTROL FLOW GRAPG CHARACTERISTICS

The ISB architecture and the ISB structure are presented in [1] for control flow predication. 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;
            }
   }
}
```

Fig. 1 A 'C' language code

The figure 2 represents CFG. This shows a number of instructions in each basic block.

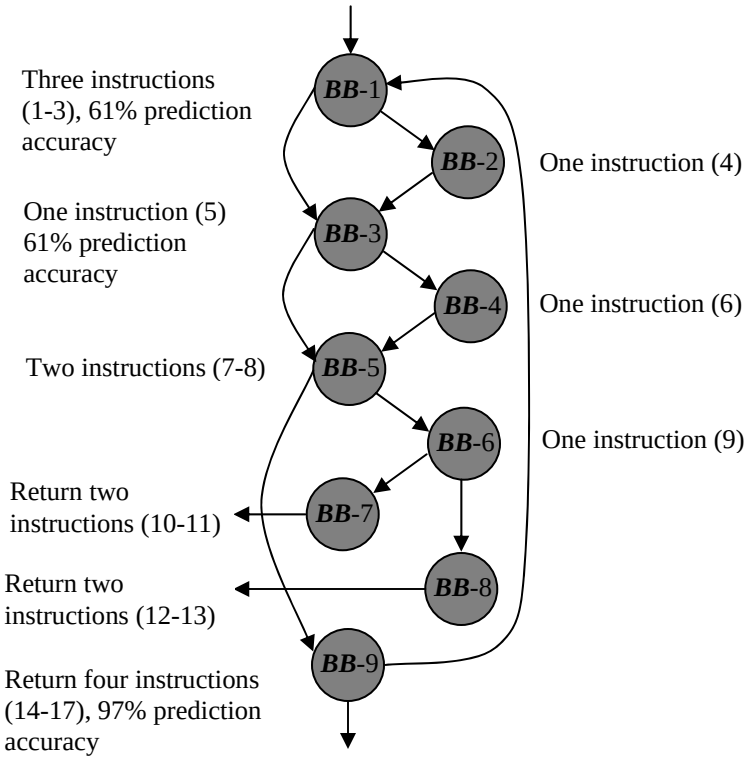

Fig 2. Control Flow Graph of fig 1.

The experiments are performed Trimaran simulator for a MIPS 2000 executable extending from the node BB1 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 can be formed as BB-1 to BB-8 as maximal multiblock. Because they have single target. The multiblock 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 between multiblocks and parallel register sharing architecture creates 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(fpt=xlenv; fpt; fpt=cdr(fpt))
{
for(ep=car(fpt); ep; ep=cdr(ep))
      {
       if(sym== car(car(ep)))
       return (cdr(car(ep)));
      }
}
```

Fig. 3 Iteration dependent loop

As advantage, an entire loop can be encapsulated in a multiblock. The above code 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 predication setup needs additional number of state information and as a result the accuracy of predication is decreased. Therefore, it allow multiblocks to have maximum two targets may be comprised. 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 2 is given by figure 4.

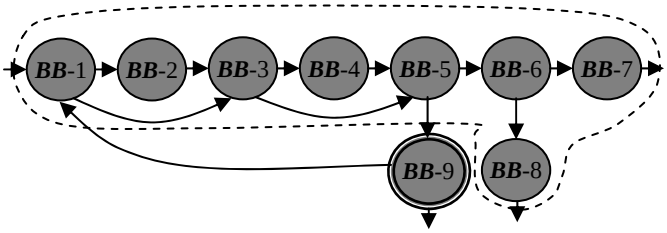

Fig. 4 Reduced CFG of figure 2

Above figure shows a multiblock constructed from BB-1 to BB-8. It contains 16 static instructions. As average 7.46 instructions are executed dynamically.

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 predications are required per iteration of the loop as compare to four predications in CFG of figure 4 that an ordinary branch predication approach would require. Following is the control flow table for control flow predication:

Table 1 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 predication buffer (CFPB) is temporary of CFT entries. The CFT entries are appended with sufficient information to help dynamic predication 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 4.

Table 2 CFPB entries

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

IV. EVALUATION ON ABSTRACT MACHINE

We first evaluate the strength of control flow predication concept on abstract machine that maintains a dynamic window from which ILP is extracted. 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.

For experimental purpose we used *compress*, *gcc*, *xlisp*, *yacc* and *tex* coded in C language. Following table 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 3 Basic structure for different programs

Program Name	Dynamic Instructions (millions)	Conditional branch ratio	Un-conditional branch ratio	Static code size	Static CFT size
gcc	1000	0.156	0.042	172032	25653
compress	22.68	0.149	0.040	6144	88.5
tex	214.67	0.143	0.055	60416	9976
yacc	26.37	0.237	0.020	12288	1737
xlisp	500	0.157	0.091	21504	3637

V. OBSERVATIONS

The table below shows variation in number of branches traversed per cycle with control flow predication. For example, in case of *gcc*, the control flow predication we observed is 1.47 branches per cycle and in *tex* 1.16 branches per cycle.

Table 4 Branch traversal results

	program	Initiation mean size	Window mean size	Branch prediction accuracy	Traversed branches per cycle
Results without control flow predication	*gcc*	5.02	72	91.11	N/A
	compress	5.24	64	89.59	N/A
	tex	5.02	169	95.87	N/A
	yacc	3.87	103	95.84	N/A
	xlisp	4.00	144	95.63	N/A
Results with control flow predication	*gcc*	9.44	105	91.02	1.47
	compress	8.40	86	89.71	1.33
	tex	6.24	207	96.10	1.16
	yacc	4.96	150	96.51	1.22
	xlisp	5.11	1.57	95.34	1.16

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

VI. CONCLUSION

As the prediction decision is over, the instructions from the predicated path are fetched in the next branch in 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. 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 predication 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 in average initiation time is restricted by the length of code. The solution of this problem is that we need a mechanism to traverse multiple branches at a time. This can be done 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] Rajendra Kumar, P K Singh, “A Modern Parallel Register Sharing Architecture for Code Compilation”, IJCA, Volume 1, No. 16, pp. 108-113, 2010

[2] David I., August Wen-mei W. Hwu, Scott A. Mahlke “The Partial Reverse If-Conversion Framework for Balancing Control Flow and Predication”, International Journal of Parallel Programming, Volume 27, No. 5, pp. 381-423, 1999.

[3] P. Chang, S. Mahlke, W. Chen, N. Warter, W. Hwu, “IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors”, Proceeding 18th Annual International Symposium on Computer Architecture, May 1991.

[4] R. Colwell, R. Nix, J. O’Donnell, D. Papworth, and P. Rodman, “A VLIW Architecture for a Trace Scheduling Compiler”, IEEE Transactions on Computers, vol. 37, pp. 967-979, Aug. 1988.

[5] David I. August, Wen-Mei W. Hwu, Scott A. Mahlke, “The Partial Reverse If-Conversion Framework for Balancing Control Flow and Predication”, International Journal of Parallel Programming Volume 27, Issue 5, pp. 381–423, 1999.

[6] Dionisios N. Pnevmatikatos, Manoj Franklin, Gurindar S. Sohi, “Control flow prediction for dynamic ILP processors”, International Symposium on Microarchitectur, Proceedings of the 26th annual international symposium on Microarchitecture, pp. 153 – 163, 1993.

[7] J. Fisher, “Trace Scheduling: A Technique for Global Microcode Compaction”, IEEE Transactions on Computers, vol. C-30, July 1981.

[8] J Cong, Guoling Han, Zhiru Zhang, “Architecture and compilation for data bandwidth improvement in configurable embedded processors”, International Conference on Computer Aided Design Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design, pp. 263-270. 2005

[9] P. Y. T. Hsu and E. S. Davidson, “Highly Concurrent Scalar Processing”, Proceeding 13th Annual International Symposium on Computer Architecture, June 1986.

[10] Lam Wilson, “Limits of control flow on parallelism”, proceedings of 19th annual International symposium on Computer Architecture, pp. 46-57, 1992.

[11] J. K. F. Lee and A. J. Smith, “Branch Prediction Strategies and Branch Target Buffer Design”, IEEE Computer, Volume 17, pp. 6-22, 1984.

[12] S. Mahlke, D. Lin, W. Chen, R. Hank, and R. Bringmann, ‘‘Effective Compiler Support for Predicated Execution Using the Hyperblock”, Proc. of the 25th Annual Workshop on Microprogramming and Microarchitecture, 1992.

[13] S. T. Pan, K. So, and J. T. Rahmeh, “Improving the Accuracy of Dynamic Branch Prediction Using Branch Correlation”, Proceeding Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), 1992.

[14] Steve Carr, “Combining Optimization for Cache and Instruction-Level Parallelism”, Proceedings of the 1996 Conference on Parallel Architectures and Compilation Techniques, 1996

[15] T. Yeh and Y. Patt, “A Comparison of Dynamic Branch Predictors that use Two Levels of Branch History”, Proceeding 20th Annual International Symposium on Computer Architecture, May 1993.

[16] Vijay S. Pai, Parthasarathy Ranganathan, Hazim Abdel-Shafi, Sarita Adve, “The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors”, IEEE Transactions on Computers, Volume 48 , Issue 2, Special issue on cache memory and related problems, pp. 218 – 226, 1999.

[17] Guilin Chen, Mahmut Kandemir, “Compiler-Directed Code Restructuring for Improving Performance of MPSoCs”, IEEE Transactions on Parallel and Distributed Systems, Volume. 19, No. 9, 2008

[18] www.trimaran.org

AUTHORS PROFILE

Rajendra Kumar - He is Assistant Professor and Head of Computer Science & Engineering department at Vidya College of engineering, Meerut. He is author of four text books *Theory of Automata, Languages and Computation* from Tata McGrawHill, *Human Computer Interaction* from Firewall Media, *Information and Communication Technologies* from University Science Press, and *Modeling and Simulation Concept* from University Science Press. He has written distance learning books on *Computer Graphics* for MGU Kerla and MDU Rohtak. His current research area is *Instruction Level Parallelism*.

Dr. P K Singh - He is an Associate Professor of Computer Science & Engineering at MMM Engineering College, Gorakhpur. He graduated from MMM Engineering College, Gorakhpur with a Bachelor of Computer Science degree and M. Tech. from University of 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.