Treegion Scheduling for Highly Parallel Processors [*]

Sanjeev Banerjia   William A. Havanki   Thomas M. Conte

Department of Electrical and Computer Engineering
North Carolina State University
Raleigh, North Carolina 27695-7911
(919)-515-7983
{sbanerj,wahavank,conte}@eos.ncsu.edu

Abstract

Instruction scheduling is a compile-time technique for extracting parallelism from programs for statically scheduled instruction-level parallel processors. Typically, an instruction scheduler partitions a program into regions and then schedules each region. One style of region represents a program as a set of decision trees or *treegions*. The non-linear nature of the treegion allows scheduling across multiple paths. This paper presents such a technique, termed *treegion scheduling*. The results of experiments comparing treegion scheduling to scheduling for basic blocks and across "simple linear regions" show that treegion scheduling outperforms the other techniques.

1 Introduction

The performance of statically-scheduled, instruction-level parallel (ILP) processors depends on compiler techniques that extract parallelism from programs. In order to extract large amounts of ILP from non-scientific, integer programs, instruction scheduling must be performed across basic blocks [1], [2]. Schedulers typically group together basic blocks which may execute together into *regions* and then schedule each region. Regions are either *linear* (containing a single path of control) or *non-linear* (containing multiple paths of control).

The grouping process (*region formation*) is often done using profile information [2], [3]; if program behavior differs from this information, performance can suffer [4]. Other problems may arise due to *merge points*, instructions to which control can flow from multiple instructions. If an instruction is speculated above a merge point, it must be duplicated along all paths that join at the merge point. Merge points also add complexity to dynamic recompilation techniques [5].

[*] This paper will appear in the *Proceedings of Euro-Par'97*, August 1997, Passau, Germany.

One region that is resistant to unpredictable execution and that does not include merge points is a *treegion*, a tree-shaped subgraph of a program's control flow graph (CFG). This paper describes treegions and how they can be scheduled and is organized as follows. Section 2 defines treegions and introduces treegion scheduling via an example. Section 3 presents experimental results for treegion scheduling and compares the results with scheduling for basic blocks and "simple linear regions". Section 4 describes related work in non-linear regions, and Section 5 concludes with comments on future work and a summary.

2 Treegions

(a) Treegions in a CFG (b) After tail duplication

Figure 1: Figure (a) shows the CFG broken into two treegions A and B. Figure (b) shows how the two treegions can be combined into one treegion A' with tail duplication.

A treegion is a rooted tree subgraph of a CFG. An example of a CFG partitioned into treegions is shown in Figure 1(a). The size and number of treegions in a CFG are determined by the CFG topology, not profile information. However, heuristics using profile information can guide methods to expand treegions; tail duplication on basic blocks 7 and 8 results in the CFG shown in Figure 1(b). Many of the procedures used with superblocks [3] may be applied to treegions.

Treegion formation begins at each entry node of a CFG. Nodes encountered while traversing from each entry node are absorbed into a treegion until merge points are encountered, each of which becomes the root of a new treegion. This process continues until every node is in some treegion.

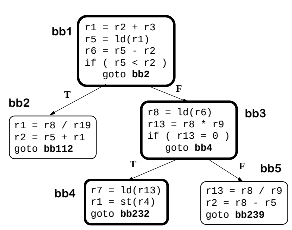


Figure 2: A sample CFG. The emphasized basic blocks are a possible preferred path.

Cycle #	ALU-1	ALU-2
0	r1 = r2 + r3	
1	r5 = ld(r1)	r1 = st(r4)
2	r6 = r5 - r2	blt **bb2**,r5,r2
3	r8 = ld(r6)	
4	r13 = r8 * r9	
5	bne **bb5**,r13,0	r7 = ld(r13)
6	goto **bb232**	
7	**bb2:** r1 = r8 / r19	
8	goto **bb112**	r2 = r5 + r1
9	**bb5:** r13 = r8 / r9	r2 = r8 - r5
10	goto **bb239**	

(a) Successive retirement

Cycle #	ALU-1	ALU-2
0	r1 = r2 + r3	*r1a = r8 / r19*
1	r5 = ld(r1)	r1 = st(r4)
2	r6 = r5 - r2	*r2 = r5 + r1a*
3	blt **bb2**,r5,r2	r8 = ld(r6)
4	r13 = r8 * r9	*r13 = r8 / r9*
5	bne **bb5**,r13,0	r7 = ld(r13)
6	goto **bb232**	
7	**bb2:** goto **bb112**	r1 = r1a
8	**bb5:** goto **bb239**	r2 = r8 - r5

(b) Treegion scheduling

Figure 3: Sample CFG schedules. Underlined instructions are speculated above their control-dependent branches. Italicized instructions have had register renaming performed.

Figure 2 shows a sample CFG. Figure 3(a) shows a schedule formed from the CFG using the successive retirement scheduling algorithm [6] (the example machine is a two-issue processor with universal functional units and unit latency). This schedule retires the exits from the *preferred path*[1] in sequential order and performs speculation only along that path. Program execution along the preferred path { bb1, bb3, bb4 } takes seven cycles (cycles 0–6), assuming there are no cache misses and perfect branch prediction. Program execution along the path { bb1, bb3, bb5 } takes eight cycles (cycles 0–5,9,10).

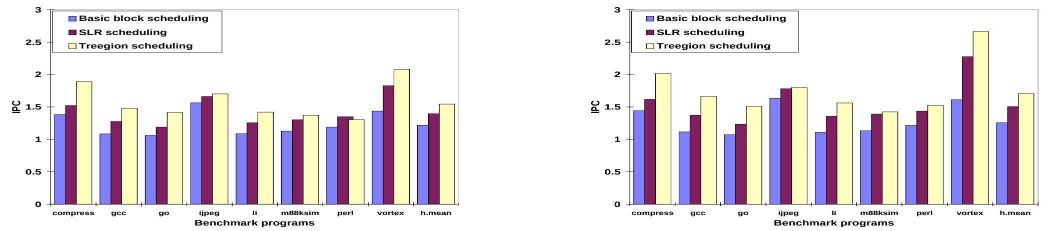
Figure 3(b) is a schedule formed from the CFG using *treegion scheduling*. The priority function used is the number of treegion execution paths through the operation [4]. Unlike successive retirement, operations from other paths ("off-paths") become intermingled into the schedule, so that operations from multiple paths are scheduled to execute together. Compile-time register renaming is used to allow speculation of operations above their control-dependent branches, preserving live-out register values. If the preferred path is executed at run-time, this schedule again takes seven cycles to execute. However, the execution time of the path { bb1, bb3, bb5 } has been reduced from eight to seven cycles.

One strength of treegion scheduling is that by scheduling multiple paths in parallel, a high-performance schedule for a preferred path can be generated without unduly penalizing off-paths. This characteristic hedges against poor performance when the executed path differs from the compile-time preferred path. In this respect, treegion scheduling is similar in spirit to the speculative hedge heuristic [4] of superblock scheduling.

3 Experimental results

Experiments were conducted to gauge the effectiveness of treegion scheduling using the SPECint95 benchmark suite. Classic optimizations and a profiling run using training inputs

[1] The preferred path is the most frequently executed path within a region as indicated by profile information or static heuristics.



(a) EIGHT-CONS model (b) EIGHT-AGGR model

Figure 4: Performance of basic block scheduling, SLR scheduling and treegion scheduling for the two machine models. h.mean denotes harmonic mean.

were applied to the benchmarks before scheduling for treegions, "simple linear regions"[2] (*SLR*s), and basic blocks using the LEGO compiler, a research ILP compiler developed at N.C. State University. Scheduling was performed for two statically-scheduled machine models: an eight-issue processor with universal functional units, EIGHT-AGGR, and one with a mix of four integer/branch, two memory, and two floating-point units, EIGHT-CONS. Instructions are unit latency except loads (2 cycles), floating-point multiply (3 cycles), and floating-point divide (9 cycles). Program performance was measured by using the profile count and schedule height of each region to estimate execution time. The effects of instruction and data caches were ignored. Useful instructions completed per cycle (IPC) was the performance metric used. Instructions added due to renaming were not used in computing IPC.

Figure 4 presents the results. In every case, treegion scheduling yielded higher performance than basic block scheduling, and about the same as or better than SLR scheduling. The treegion schedule performed worse than the SLR schedule for **perl** under EIGHT-CONS because of aggressive speculation, which extends the preferred path schedule by speculating more off-path operations. The IPC improvements are larger with EIGHT-AGGR because the flexibility of the model permitted the treegion scheduler to fill more empty slots in the schedule with off-path operations. This illustrates that treegion scheduling yields the most benefit on highly parallel processors.

4 Related work

Hsu and Davidson's decision tree scheduling (DTS) [7] is the predecessor of the work presented here. DTS schedules along multiple paths within a decision tree, inserting instructions into branch delay slots and using guards to control writeback of speculated instructions. The VLIW project at IBM Research embellished Nicolau's percolation scheduling [8], using them to implement a VLIW compiler [9]. The heart of the IBM VLIW machine is a *tree instruc-*

[2] Simple linear regions are built like superblocks, but without tail duplication.

tion, which has the ability to evaluate multiple branches in one clock cycle. The initial work in VLIW architectures was based on a single-path scheduling algorithm called trace scheduling [2]. The Trace Scheduling-2 algorithm is an extension of the original trace scheduling algorithm that schedules along multiple paths simultaneously [10]. Hyperblock scheduling also schedules multiple paths in parallel [3] by removing branches from the instruction stream entirely through if-conversion.

5 Concluding remarks and acknowledgements

There are issues related to treegions that merit further research. The use of if-conversion and tail duplication could eliminate merge points and allow for the formation of larger treegions. Also, different heuristics for treegion scheduling need to be identified and analyzed.

This paper introduced treegion scheduling, which performs scheduling across the tree subgraphs that compose a CFG. The technique extracts high amounts of ILP by scheduling and speculating operations along multiple paths. The advantages of treegion scheduling were illustrated by comparing treegions to other regions. The latter technique is especially effective for highly parallel processors.

The authors would like to thank Scott Mahlke of the CAR Group at Hewlett-Packard Labs for providing the optimized SPECint95 benchmarks used in this paper, and Kishore Menezes and Sumedh Sathaye for discussions that greatly improved the quality of this paper. The comments from the anonymous referees are also appreciated. This work was supported by IBM, Hewlett-Packard, and the National Science Foundation under grants MIP-9696010, MIP-9625007, and GER-9454175.

References

[1] G. S. Tjaden and M. J. Flynn, "Detection and parallel execution of independent instructions," *IEEE Trans. Comput.*, vol. C-19, pp. 889–895, Oct. 1970.

[2] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 478–490, July 1981.

[3] S. A. Mahlke, *Exploiting instruction level parallelism in the presence of branches.* PhD thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1996.

[4] B. L. Deitrich and W. W. Hwu, "Speculative hedge: regulating compile-time speculation against profile variations," in *Proc. 29th Ann. Int'l Symp. on Microarchitecture* [11].

[5] T. M. Conte and S. W. Sathaye, "Dynamic rescheduling: A technique for object code compatibility in VLIW architectures," in *Proc. 28th Ann. Int'l Symp. on Microarchitecture*, (Ann Arbor, MI), Nov. 1995.

[6] C. Chekuri, R. Johnson, R. Motwani, B. Natarajan, B. Rau, and M. Schlansker, "Profile-driven instruction level parallel scheduling with application to superblocks," in *Proc. 29th Ann. Int'l Symp. on Microarchitecture* [11], pp. 58—67.

[7] P. Y. T. Hsu and E. S. Davidson, "Highly concurrent scalar processing," in *Proc. 13th Ann. Int'l Symp. Computer Architecture*, (Tokyo, Japan), June 1986.

[8] A. Nicolau, "Percolation scheduling: a parallel compilation technique," Technical report TR-678, Department of Computer Science, Cornell University, Ithaca, NY, May 1985.

[9] K. Ebcioğlu, "Some design ideas for a VLIW architecture for sequential-natured software," in *Proceedings of the IFIP Working Group 10.3 Working Conference on Parallel Processing*, (Pisa, Italy), pp. 3–21, North Holland, 1988. (published as *Parallel Processing*, M. Cosnard, et al., (eds).).

[10] J. A. Fisher, "Global code generation for instruction-level parallelism: Trace Scheduling-2," Tech. Rep. HPL-93-43, Hewlett-Packard Laboratories, June 1993.

[11] *Proc. 29th Ann. Int'l Symp. on Microarchitecture*, (Paris, France), Dec. 1996.