Understanding and Improving Register Assignment[*]

Cindy Norris and James B. Fenwick, Jr.

Department of Computer Science
Appalachian State University
Boone, North Carolina, USA 28608
(828) 262-2359
{can,jbf}@cs.appstate.edu

Abstract. Register allocation can decrease instruction-level parallelism by prohibiting the scheduler from reordering instructions. The impact of register assignment strategies on a subsequent scheduling phase is explored. A new register assignment strategy and experimental results are presented.

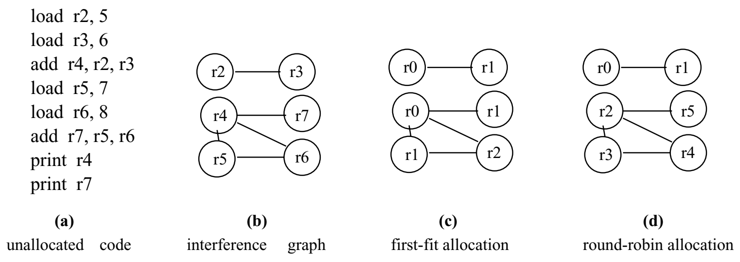
1 Introduction

Register assignment is the phase of the register allocator that decides what values to put in each register. First-Fit register assignment [5] chooses the first available register in a sequential ordering of the registers. Round-Robin assignment [5] begins searching for an available register at the point where the last successful search ended. Scheduling increases run-time performance by rearranging the code to overlap the execution of independent instructions. A register assignment strategy cooperates better with a subsequent scheduling phase than another strategy if it (1) introduces less false dependences, or (2) introduces false dependences that don't prevent the scheduler from uncovering sufficient fine-grain parallelism.

This paper examines the impact of register assignment strategies within a *global* register allocator on a subsequent local instruction scheduling phase. The findings presented in section 3 indicate that register assignment of a global allocator *does* impact scheduling, particularly for functional units with higher latencies. Our Improved First-Fit strategy was found to cooperate better than First-Fit and better than the Round-Robin strategy when register pressure is high. The experimental study incorporates a postpass scheduler. However, in the absence of this scheduling phase, the Improved First-Fit strategy should still prove to be effective by reducing the number of hardware stalls caused by the false dependences added by register assignment.

As an illustration of this impact, figure 1(a) contains an unallocated section of code. References to *r2*, *r3*, *r4*, *r5*, *r6*, *r7* are references to live ranges. It is the job of the register allocator to rewrite this code changing the references to

[*] This work was partially supported by NSF under grant CCR-9625219.



```
load  r2, 5
load  r3, 6
add  r4, r2, r3
load  r5, 7
load  r6, 8
add  r7, r5, r6
print  r4
print  r7
```

Fig. 1. Register Assignment Strategies

live ranges to references to physical registers. Graph coloring register allocators [4, 3] represent the register allocation problem as a graph coloring problem in which nodes in a graph represent live ranges and edges are added between two nodes (which are then called *neighbors*) if the corresponding live ranges overlap and must occupy different registers. The allocation is determined by coloring the nodes of the graph, called an *interference graph*, with K colors where K is the number of physical registers. Two nodes can not be colored the same color if they are connected by an edge in the interference graph. Figure 1(b) contains the interference graph corresponding to the code in figure 1(a).

Unlike local register allocation [7] in which the live ranges are assigned physical registers in order of proximity, nodes in an interference graph are assigned physical registers in a priority order based partially upon the number of uses and definitions in the code. In this example, each live range contains one definition and one use making the priority of all nodes the same, and thus they can be arbitrarily assigned physical registers in the order *r2*, *r3*, etc. Figures 1(c) and (d) demonstrate the register assignment decided by the First-Fit and Round-Robin strategies, respectively. First-Fit register assignment chooses the first available register in a sequential ordering of the registers. Round-Robin assignment begins searching for an available register at the point where the last successful search ended. This explains why *r4* in the interference graph is mapped to *r0* by First-Fit and *r2* by Round-Robin as shown in figure 1. Register assignment, in particular First-Fit register assignment, adds dependences known as *false dependences.* They are not present because of references to a single value, but because of accesses to a single register. False dependences can prevent a scheduler from reordering instructions in order for them to be executed in parallel.

When the register allocator is unable to assign a physical register to a live range, the allocator *spills* the live range to memory. Intuitively, it is expected that the two assignment strategies result in different amounts of spill code. Since First-Fit chooses the first free register available, it uses registers more sparingly. Assigning two live ranges the same physical register may save a physical register to be assigned later and prevent a spill.

2 Improved First-Fit Register Assignment

The Round-Robin strategy adds fewer false data dependences during register assignment than the First-Fit strategy. However, the false dependences that are not added are avoided only accidentally, if at all. In addition, Round-Robin may cause more spilling. This section discusses a register assignment strategy that *deliberately* attempts to avoid adding the false data dependences of First-Fit, avoid spill code, and *remain as simple* as the other register assignment strategies. More complicated strategies exist [8, 9, 1], however our goal is to improve register allocation without impacting the time required to do the allocation.

Our Improved First-Fit (IFF) register assignment uses an awareness of the sequential nature of register usage to make register assignment decisions. If two definitions are *near* each other in the sequential ordering of the statements, the IFF assignment strategy tries to avoid assigning those definitions to the same physical register. The IFF register assignment strategy takes as input the size of the *nearness window* which identifies how many definitions near a particular defining statement are to be examined. When assigning a physical register to a definition *d*, the definitions near to *d* are examined and if a physical register has been assigned to one of those near definitions, then that physical register is not assigned to *d*. For example in the code below, given a nearness window of size 3, the IFF strategy attempts to avoid assigning to *r13* the same physical registers assigned to *r14*, *r15*, and *r16*.

```
ldi  r13 4
mul  r14 r13 r9
add  r15 r5 r14
ldi  r16 4
mul  r17 r16 r9
```

Why choose to examine nearby statements when making an assignment decision? First, techniques such as loop unrolling and global instruction scheduling precede register allocation and increase the number of statements within a basic block in order to increase the instruction-level parallelism. After increasing the available parallelism within a basic block, the IFF strategy avoids adding the false data dependences that prevent the simultaneous execution of sequential statements. Second, examining nearby statements when making the assignment decision makes the IFF strategy easy to implement and very efficient.

3 Experimental Study

An experimental study was performed to evaluate the performance of the three register assignment strategies. A C program is processed by the SUIF compiler system [6] to convert the program into an intermediate code format. This intermediate code is then translated into Iloc, which is a low-level intermediate code designed at Rice University for the development of optimizing compilers [2]. Our implementation of the optimistic allocator [3] performs register allocation on the Iloc code using any of the three register assignment strategies discussed previously. Allocated code is then fed to a local instruction scheduler, and the scheduled code is converted back to C, inserting instructions to simulate a fine-grain parallel machine.

Two different fine-grain parallel architectures were simulated: a pipelined machine with low latencies and a pipelined machine with high latencies. Twenty-five programs taken from the Livermore loops, SPEC, and Stanford benchmark suites were used as input to the experimental study of the three different register assignment strategies on each of the two architecture classes. Simulations were performed using three different register set sizes (8, 16, and 32). The register set sizes are not typical of current architectures, but were chosen to impact the register pressure of the benchmark programs. In other words, the register set of size 8 causes a great deal of register pressure for the selected benchmark programs while 32 registers generated little register pressure.

The performance of the assignment strategy was measured by calculating $totalcycles(FirstFit)/totalcycles(RoundRobin)$ and $totalcycles(FirstFit)/totalcycles(ImprovedFirstFit)$ where $totalcycles$ is the number of cycles required to execute the Iloc code as determined by the simulator. Due to page length restrictions, a table showing the performance speedups was omitted, but it may be viewed at `http://www.cs.appstate.edu/~can/research/papers.html`.

The study indicates that assigning registers with a Round-Robin strategy generally produces code that executes faster than code assigned registers using First-Fit. Two items in particular are noted. First, Round-Robin results in significantly better code for high-latency pipelined architectures with a register set large enough to keep spilling at a minimum. Second, for low-latency machines when register pressure is high, First-Fit is a better strategy. The reason First-Fit performs better under high register pressure is due to the increased amount of spill code inserted by Round-Robin. The simulations show that for 8 registers, Round-Robin resulted in 5% more spills than First-Fit.

In general, the Improved First-Fit strategy performs as well as or better than the Round-Robin strategy. Of particular note is that Improved First-Fit is significantly better than Round-Robin for high-latency machines under a high register pressure. In the case of low register pressure, the Round-Robin strategy has the potential to add fewer false dependences since it reuses a register only after all other registers have been used once. For the Improved First-Fit strategy, reuse of registers occurs more quickly when the nearness window is small. Even a window size of 7 can cause more false dependences than Round-Robin.

The Improved First-Fit strategy performs better as the size of the nearness window increases. For example on the high-latency machine with 32 registers, the average speedup is 1.20 when the nearness window size is 3 and 1.24 when the nearness window size is increased to 7. However, increases in the nearness window size have less impact on smaller register sets. With less registers overall, the algorithm is less successful in finding a register not used in the nearness window; thus, reverting to First-Fit.

It was expected that the Round-Robin strategy would result in the most spill code being inserted of the three assignment strategies, with First-Fit causing the least amount and Improved First-Fit falling in between. The experimental study supports this hypothesis. Using 8 registers 670, 684, and 703 spills occurred among the 25 benchmark programs for the First-Fit, Improved First-Fit with nearness window of 3, and Round-Robin assignment strategies, respectively. (The larger sized register sets cause dramatically fewer spills as more free registers were available.) As the nearness window size increased, the amount of spill code inserted by Improved First-Fit decreased on average although the results for individual benchmarks varied. This decrease occurs because the IFF strategy reverts more often to First-Fit when it has a larger set of instructions to examine when looking for an unused register.

4 Summary

The goal of this research was to understand better the effect of register assignment strategies within a global register allocator on instruction-level parallelism. A new register assignment strategy, Improved First-Fit, was presented that deliberately avoids the false dependences created by First-Fit, and inserts less spill code than Round-Robin. This new strategy is a simple, effective technique requiring only minor modifications to existing global register allocation algorithms.

References

[1] David A. Berson, Rajiv Gupta, and Mary Lou Soffa. Resource spackling: A framework for integrating register allocation in local and global schedulers. In *PACT '94: International Conference on Parallel Architectures and Compilation Techniques*, Montreal, Canada, August 1994.

[2] Preston Briggs, Keith D. Cooper, and Linda Torczon. R^n Programming Environment Newsletter #44. Dept. of Computer Science, Rice University, Sept. 1987.

[3] Preston Briggs, Keith D. Cooper, and Linda Torczon. Rematerialization. In *Proceedings of the SIGPLAN '92 Conference on Programming Language Design and Implementation*, 1992.

[4] G. J. Chaitin. Register allocation and spilling via graph coloring. In *SIGPLAN Symposium on Compiler Construction*, Boston, June 1982.

[5] James R. Goodman and Wei-Chung Hsu. Code scheduling and register allocation in large basic blocks. In *Supercomputing '88 Proceedings*, pages 442–452, Nov. 1988.

[6] Stanford SUIF Compiler Group. *The SUIF Parallelizing Compiler Guide.* Stanford University, 1994. Version 1.0.

[7] Wei Chung Hsu, Charles N. Fischer, and James R. Goodman. On the minimization of loads/stores in local register allocation. *IEEE Transactions on Software Engineering*, 15(10):1252–1260, 1989.

[8] Cindy Norris and Lori L. Pollock. A scheduler-sensitive global register allocator. In *Supercomputing '93 Proceedings*, Portland, OR, November 1993.

[9] S. S. Pinter. Register allocation with instruction scheduling: a new approach. In *Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation*, June 1993.