A Coalescing Algorithm for Aliased Registers

Mariza A. S. Bigonha[1], **Fabrice Rastello**[2],
Fernando Magno Quintão Pereira[1], **Roberto S. Bigonha**[1]

[1] Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
Av. Antônio Carlos, 6627 – 31.270-010 – Belo Horizonte – MG – Brazil

{mariza,fpereira,bigonha}@dcc.ufmg.br

[2]Laboratoire de l'Informatique du Paralllisme – École normale supérieure de Lyon
46 alle d'Italie, 69364 – Lyon cedex 07 – France

Fabrice.Rastello@ens-lyon.fr

Abstract. *Register coalescing is a compiler optimization that removes copy instructions such as $a = b$ from a source program by assigning variables a and b to the same register. The vast majority of coalescing algorithms described in the literature assume homogeneous register banks; however, many important computer architectures, such as x86, ARM, SPARC and ST240 contain an irregularity called register aliasing. Two registers alias if assigning a value to one of them changes the contents of the other. Most of the time registers can be divided into subclasses that hierarchically fit into each other. The objective of this research is to design, implement and test new coalescing algorithms that handle hierarchical register aliasing. We expect that an aliasing aware coalescer will be able to remove more copy instructions than an otherwise oblivious algorithm; thus, decreasing the size and increasing the performance of compiled programs.*

1. Problem Description

Register allocation is the task of mapping the variables of a source program into a finite number of registers. If the number of registers is not sufficient, then some of the variables are mapped into memory. A compiler optimization that is performed on top of register allocation is *coalescing*. This optimization consists in mapping variables related by copy instructions to the same register. For instance, we can remove the instruction $a = b$ from the source program provided that a and b are mapped to the same register `r`.

A good coalescing algorithm can improve the execution speed of a program by as much as 12% [12]. Although register coalescing is important, and many coalescing algorithms have been proposed in the literature, these algorithms do not take *register aliasing* into consideration. This project aims at filling this gap. Its objective is to design, implement and test new coalescing algorithms that handle hierarchical register aliasing.

Two registers alias if an assignment to one of them affects the contents of the other. The best known example of aliasing is found in the x86 architecture, which has four general purpose 16-bit registers - `AX, BX, CX` and `DX` - that can also be used as eight 8-bit registers. That is, the x86 architecture combines two 8-bit registers into one 16-bit register. Figure 1 shows the bank of general purpose registers used in the x86. Notice that all the registers contain some sort of aliasing, but only the upper registers are divided into two parts. Another example of aliased registers is the combination of two aligned single

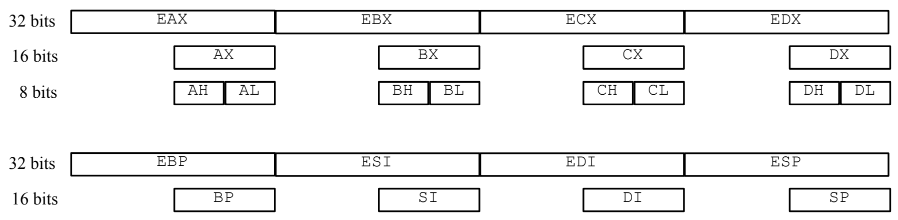

Figure 1. General purpose registers of the x86 architecture

precision floating-point registers to form one double-precision register. As for the x86, register aliasing of most architectures is restricted to hierarchical aliasing: at the lowest level the register bank is composed of atomic registers; the upper level partitions this set of atomic registers into non-intersecting register subsets; to each subset corresponds a register that aliases with its composing registers. Examples of such architectures we are concerned with include the Sun SPARC, the ARM processors, and the ST240. ARM Neon, and ST240 have two levels of aliasing while SPARC V8 has three: single precision floating point registers can be combined into double precision and even in quad precision registers. X86 has a subset of registers with just one level of aliasing - `SP`, `SI`, `DI` and `SP` - plus another subset, formed by registers `AX`, `BX`, `CX` and `DX`, with two levels of aliasing. X86 is thus an example of hybrid hierarchical aliasing.

2. Related Works

The coalescing problem is intrinsically related to the register allocation problem. For instance, Chaitin *et al.* [8] already describe a coalescing strategy in their pioneering work on register allocation via graph coloring. Since its first appearance, Chaitin's work has been the target of a slow, yet never-ending stream of improvements. One quarter century after Chaitin's seminal paper, coalescing has been one of the main forces pushing new variations in graph coloring register allocation. Bouchez *et al.* [3] summarizes some of the best known approaches for performing register coalescing:

- *Aggressive Coalescing* [8, 7]: merges move-related vertices, regardless of the colorability of the interference graph after the merging.
- *Conservative Coalescing* [5]: merges moves if, and only if, the merging does not compromise the colorability of the interference graph.
- *Optimistic Coalescing* [17, 18]: coalesces moves aggressively, and if it compromises the colorability of the graph, then gives up as few moves as possible.
- *Incremental Conservative Coalescing* [11]: removes one particular move instruction, while keeping the colorability of the graph.

Bouchez *et al.* [2, 3] have shown, by means of an ingenious sequence of reductions, that all these different materializations of the register coalescing problem are NP-complete for general interference graphs. Hack *et al.* [12] proposed a new scheme called *Recoloring Coalescing*: color the graph arbitrarily, recolor move-related nodes (to satisfy as much moves as possible) and their interfering neighborhood (to keep the coloring valid).

The finding that programs in Single Static Assignment (SSA) form have chordal interference graphs [2, 6, 13, 20] has given a new boost to research on register coalescing. *Static Single Assignment* (SSA) form is an intermediate representation in which each variable is defined at most once in the program code [9, 22]. Nowadays, there exist several industrial and academic compilers using SSA in their back-end, such as LLVM [15], Sun's HotSpot JVM [25], IBM's Java Jikes RVM [26], LAO [1], and Firm [12]. It is possible to retain the SSA property until the end of the code generation process. Indeed, there exists polynomial time algorithms to discover the chromatic number of chordal graphs [10]; thus, register assignment has polynomial time solution for programs in the SSA representation. However, as a form of live range splitting, the SSA transformation inserts many copies into the program code, what makes register coalescing even more important. Unfortunately, as shown by Bouchez *et al.* [3], most coalescing instances remain NP-complete for chordal graphs. Two new coalescing algorithms in the context of SSA-form based register allocation have been presented in 2008: Hack *et al.* have proposed a recoloring coalescing algorithm [12], and Bouchez *et al.* have proposed a suite of conservative and optimistic algorithms [4].

Aliasing complicates register allocation substantially. For instance, finding the minimal register assignment in face of aliasing is NP-complete, even for programs in SSA-form [16]. Nevertheless, register allocators that handle aliasing have already been described [14, 23, 24]. The algorithm presented by Smith *et al.*, for instance, modifies a graph coloring approach to deal with this phenomenon. However, none of the coalescing algorithms discussed so far takes aliasing into consideration. Recently, Pereira *et al.* [21] have proposed a *Puzzle-Based* register allocator to handle hierarchical aliasing. In this paradigm, the register allocation problem is seen as a collection of puzzles that can be solved in polynomial time. This approach tends to reduce the number of variables sent to memory, at the expenses of increasing the number of copy instructions in the target code; thus, making register coalescing essential.

We can differentiate several coalescing problems. *Global* coalescing consists in minimizing the total amount of move instructions for the entire procedure. This problem is NP-complete [3], regardless of aliasing. *Local* coalescing is the restriction to a basic block. This problem has polynomial time solution in the absence of register aliasing, but Lee et al. have shown that it is NP-complete for architectures that present hierarchically aliased registers. The *biased* coloring problem is the simplest realization of the coalescing problem, consisting in minimizing the number of move instructions inserted between two instructions of a program. Pereira *et al.* have described an optimal algorithm for a particular case of biased coloring - level-1 alias hierarchy with no pre-assignment [19]. We are interested in the global version of the register coalescing problem.

3. The Proposed Approach to the Register Coalescing Problem

We will adapt the graph coloring based coalescing algorithm proposed by Bouchez *et al* [4] to handle computer architectures with hierarchical register aliasing. Our objective is to reduce the number of move instructions in the function being optimized, using, for instance, the expedient of manipulating two small values stored in different halves of the same register with one single instruction. For instance, if a, b, c and d are four single precision floating point values, and a and b are stored in the same double precision register, then we can implement the two copies $c := a$ and $d := b$ with one single register copy. In order to show the advantages of the new approach, we will compare our modified algorithm with Bouchez's original method, which does not take register aliasing into consideration. We will also compare it with the puzzle based register allocator [21], which handles register aliasing, but performs very simple coalescing, based on a biased coloring strategy. The new register coalescing algorithm will be implemented and tested in the back-end of the Low Level Virtual Machine (LLVM) framework [15]. This framework is used, for instance, to JIT compile open-GL applications in Mac OSX 10.5.

4. Expected Results

We expect that our new algorithm will produce target code that is shorter and more efficient than the code produced by traditional coalescers that do not take register aliasing into consideration. Ideally we should be able to reduce the size of the programs produced by the puzzle based allocator [21] by 6-7%. Speed-up improvements will depend on the target architecture. We expect small improvements in the x86 processor; however, we should be able to obtain a 2-4% decrease in execution time on the PowerPC chip. Gains in efficiency should be more noticeable in embedded devises, such as the ARM Neon processor. In addition to the concrete contribution to the research community, this project will have the positive side-effect of creating an environment more favorable to further co-operations between Brazilian and French research institutes.

References

[1] Benoit Boissinot, Sebastian Hack, Daniel Grund, Benoit Dupont de Dinechin, and Fabrice Rastello. Fast liveness checking for SSA-form programs. In *CGO*, pages 35-44 IEEE, 2008.

[2] F. Bouchez. Allocation de registres et vidage en mémoire. Master's thesis, ENS Lyon, 2005.

[3] F. Bouchez, A. Darte, and Fabrice Rastello. On the complexity of register coalescing. In *CGO*, pages 102-104 IEEE, 2007.

[4] Florent Bouchez, Alain Darte, and Fabrice Rastello. Advanced conservative and optimistic register coalescing. In *CASES*, pages 147 – 156. ACM, 2008.

[5] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. TOPLAS, 16(3), pages 428-455, 1994.

[6] P. Brisk, F. Dabiri, J. Macbeth, and M. Sarrafzadeh. Polynomial-time graph coloring register allocation. In *IWLS*, pages 150-155 ACM, 2005.

[7] G. J. Chaitin. Register allocation and spilling via graph coloring. *CC*, 17(6), pages 98–105, 1982.

[8] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *Computer Languages*, 6, pages 47–57, 1981.

[9] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *TOPLAS*, 13(4), pages 451–490, 1991.

[10] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SICOMP*, 1(2), pages 180-187, 1972.

[11] Lal George and Andrew W. Appel. Iterated register coalescing. *TOPLAS*, 18(3), pages 300-324, 1996.

[12] S. Hack and G. Goos. Copy coalescing by graph recoloring. In *PLDI*, pages 227-237 ACM, 2008.

[13] Sebastian Hack, Daniel Grund, and Gerhard Goos. Register allocation for programs in SSA-form. In *CC*, pages 247–262, 2006.

[14] Timothy Kong and Kent D Wilken. Precise register allocation for irregular architectures. In *MICRO*, pages 297–307 ACM, 1998.

[15] Chris Lattner and Vikram S. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *CGO*, pages 75–88 IEEE, 2004.

[16] J. K. Lee, J. Palsberg, and Fernando M. Q. Pereira. Aliased register allocation. In *ICALP*, pages 680-691, 2007.

[17] J. Park and S. Moon. Optimistic register coalescing. In *PACT*, p.196–204 IEEE, 1998.

[18] J. Park and S. Moon. Optimistic register coalescing. *TOPLAS*, 26(4):735-765, 2004.

[19] Fernando M. Q. Pereira. *Register Allocation by Puzzle Solving*. PhD thesis, University of California, Los Angeles, 2008.

[20] Fernando Magno Quintao Pereira and Jens Palsberg. Register allocation via coloring of chordal graphs. In *APLAS*, pages 315–329. Springer, 2005.

[21] Fernando M. Q. Pereira and J. Palsberg. Register allocation by puzzle solving. In *PLDI*, p. 216-226 ACM, 2008.

[22] B. K. Rosen, F. K. Zadeck, and M. N. Wegman. Global value numbers and redundant computations. In *POPL*, pages 12–27 ACM, 1988.

[23] Bernhard Scholz and Erik Eckstein. Register allocation for irregular architectures. In *LCTES/SCOPES*, pages 139–148 ACM, 2002.

[24] Michael D. Smith, Norman Ramsey, and Glenn Holloway. A generalized algorithm for graph-coloring register allocation. In *PLDI*, pages 277–288 ACM, 2004.

[25] JVM Team. The java HotSpot virtual machine. Technical Report Technical White Paper, Sun Microsystems, 2006.

[26] The Jikes Team. Jikes RVM home page, 2007. http://jikesrvm.sourceforge.net/.