

SSA-Form-Based Register Allocation for the Java HotSpot™ Server Compiler

Master's Thesis Proposal

Register allocation, i.e., the task of assigning processor registers to local variables and temporary values, is one of the most important compiler optimizations. A vast amount of research has led to algorithms ranging from simple and fast heuristics to optimal algorithms with exponential time complexity. Because the problem is known to be NP-complete [2], algorithms must balance the time necessary for allocation against the resulting code quality. Two common algorithms in modern compilers are *graph coloring* (see for example [1, 2]), which is suitable when compilation time is not a major concern, and *linear scan* [6, 8, 10], which is faster and therefore frequently used for just-in-time compilers where compilation time adds to run time.

Static single assignment (SSA) form [3] is a type of intermediate representation that simplifies many compiler optimizations. All variables have only a single point of definition. At control flow joins, *phi functions* are used to merge different variables of the predecessor blocks. Because processors cannot execute phi functions, it is necessary to replace them with move instructions during code generation (*SSA form deconstruction*).

Traditionally, SSA form deconstruction was performed before register allocation. Only recently has it been observed that register allocation on SSA form has several advantages due to additional guarantees on variable lifetime. Lifetime information is essential for register allocation because two variables that interfere, i.e., that are live at the same time, must not have the same register assigned. The interference graph of a program in SSA form is *chordal* (every cycle with four or more edges has an edge connecting two vertices of the cycle, leading to a triangulated structure).

Many graph algorithms are simpler on chordal graphs, e.g., graph coloring can be performed in polynomial time. These properties were used to simplify register allocators based on graph coloring [4]. When the maximum register pressure is below or equal to the number of available registers, allocation is guaranteed to succeed. This allows to split the algorithms for spilling and register assignment. Traditionally, spilling and register assignment were interleaved, i.e., a variable was spilled when the graph turned out to be not colorable. This led to a time-consuming repeated execution of the graph coloring algorithm.

Recent research at the University of California, Irvine, also applied the benefits of SSA Form on linear scan register allocation [9]. The *lifetime intervals*, which are the basic data structure of the algorithm, are easier to construct and have a simpler structure. Additionally, infrastructure already present in the linear scan algorithm can be used to perform SSA form deconstruction after register allocation, thus making a separate SSA form deconstruction algorithm unnecessary.

When comparing graph coloring and linear scan register allocation on SSA form, it turns out the two algorithms are not so different than they look at the first glance. The simplifications allowed by SSA form eliminate parts that were previously handled differently. Therefore, the next research step is to analyze the cases where the two algorithms are equivalent (e.g., register allocation without spilling is very similar), and find cases where the algorithms still differ (e.g., selecting variables to be spilled). Finally, both algorithms should be merged to one algorithm that covers graph coloring and linear scan as special cases, but also allows a gradual “sliding” between them, i.e., it should be configurable to what extent graph coloring and linear scan are used.

Implementation

The new algorithm should be implemented for the Java HotSpot™ server compiler [5], one of the two just-in-time compilers of the Java HotSpot™ virtual machine. The source code is available as open source from the

OpenJDK project [7]. Currently, the server compiler uses an old graph coloring algorithm that is not based on SSA form, although the intermediate representation is completely in SSA form. The current algorithm is slow (it requires about 45% of the already long compilation time) and known to produce sub-optimal code in several cases.

The goal is a completely new register allocator that is significantly faster than the current one, and that generates equally good or even better code. However, the main focus is on compilation speed. Test environment is the Intel x86 architecture both in 32-bit mode (as an example for an architecture with a low number of registers) and 64-bit mode (as an example for an architecture with a high number of registers).

First, the student should get familiar with the topic of register allocation and the source code base of the Java HotSpot™ VM. Expertise on both topics is available from the other members of the research group. The next step, the student should implement a minimal SSA-form-based register allocator without any optimizations. After this first phase, the evaluation of the generated code will show potential for optimizations and tuning. This includes the heuristic about which variables should be spilled, the places where spill stores and loads are inserted, The final goal is then an optimized but still simple register allocator.

References

- [1] P. Briggs, K. D. Cooper, and L. Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems*, 16(3):428–455, 1994.
- [2] G. J. Chaitin, M. A. Auslander, A. K. Chandra, J. Cocke, M. E. Hopkins, and P. W. Markstein. Register allocation via coloring. *Computer Languages*, 6:47–57, 1981.
- [3] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451–490, 1991.
- [4] S. Hack, D. Grund, and G. Goos. Register allocation for programs in SSA-form. In *Proceedings of the International Conference on Compiler Construction*, pages 247–262. LNCS 3923, Springer Verlag, 2006.
- [5] M. Paleczny, C. Vick, and C. Click. The Java HotSpot™ server compiler. In *Proceedings of the Java Virtual Machine Research and Technology Symposium*, pages 1–12. USENIX, 2001.
- [6] M. Poletto and V. Sarkar. Linear scan register allocation. *ACM Transactions on Programming Languages and Systems*, 21(5):895–913, 1999.
- [7] Sun Microsystems, Inc. *OpenJDK*, 2009. <http://openjdk.java.net/>.
- [8] O. Traub, G. Holloway, and M. D. Smith. Quality and speed in linear-scan register allocation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 142–151. ACM Press, 1998.
- [9] C. Wimmer and M. Franz. Linear scan register allocation on ssa form. In *Proceedings of the International Symposium on Code Generation and Optimization*. ACM Press, 2010.
- [10] C. Wimmer and H. Mössenböck. Optimized interval splitting in a linear scan register allocator. In *Proceedings of the ACM/USENIX International Conference on Virtual Execution Environments*, pages 132–141. ACM Press, 2005.