Modernising register allocation in SBCL

Author: Alexandra Barchunova
Date: 2013-05-02 13:04:59 CEST

1 Project Aim

This project aims to improve the compiler back end by enhancing the register allocation procedure. To realize this goal, we propose to implement
+ a new heuristic optimization based on coloring the global interference graph,
+ live ranges and live range splitting,
+ spill code insertion.

The task of the register allocator is to assign an unconstrained number of temporary names (TNs) in the intermediate representation (IR) to a finite number of registers. A naive approach yields too many memory operations resulting in a reduced execution speed. The currently implemented SBCL register allocator performs only graph coloring, essentially treating spills by coloring with stack slots. In order to improve the allocator, we propose to integrate a new heuristic graph coloring method inspired by the concepts introduced by Briggs et al. (1994) in the SBCL compiler. According to Briggs' results of the *optimistic coloring*, the proposed enhancements can be expected to increase performance of produced code by up to 15\%.

2 Project Plan

The steps to take are described in the following subsections.

2.1 Find suboptimal examples

In order to improve register allocation in SBCL, as a first step we search and analyze examples of suboptimal register allocation that are due to suboptimal coloring. Such examples will demonstrate the specific problems of the register allocation, and can serve as benchmark data during the test phase of the extended register allocation method.

Inferring simple heuristics capturing suboptimal register allocation from such examples might enable us to detect problems within larger pieces of machine code automatically. This might give us understanding, how often each type of suboptimal allocation occurs.

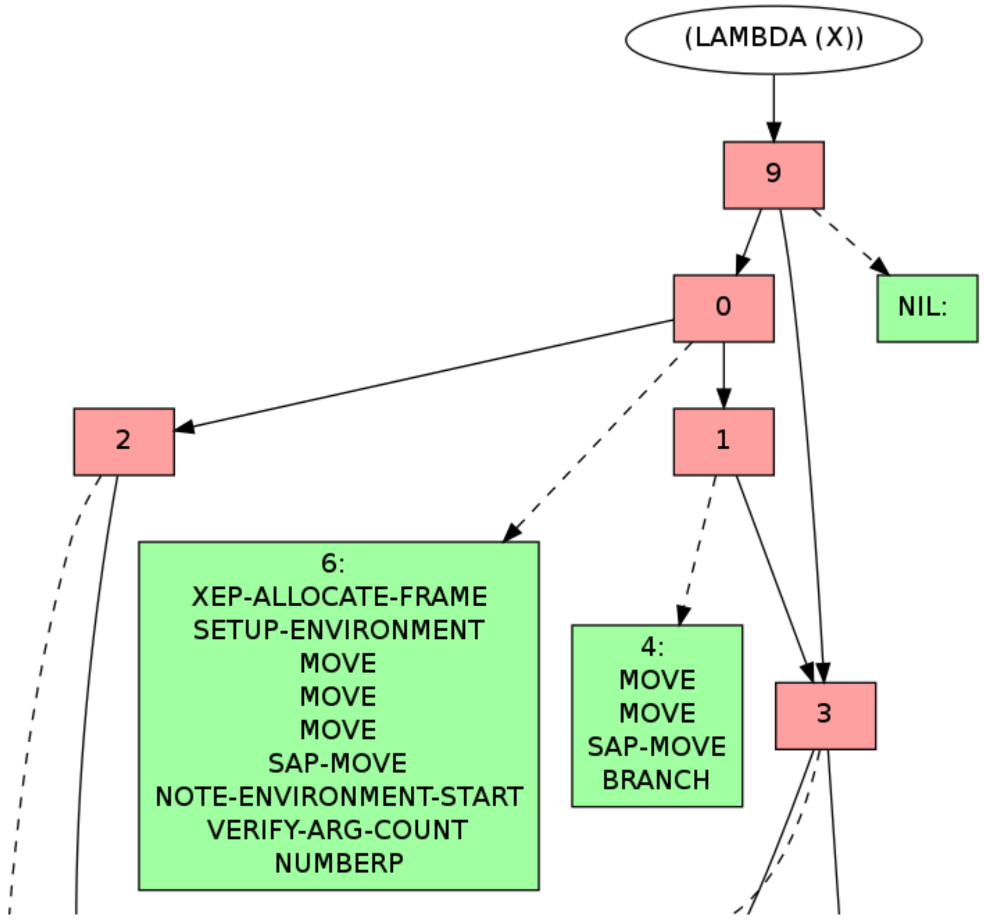
At the same time, we would like to conduct the common analysis of the register allocation quality in which we compare different allocation methods based on benchmark data.

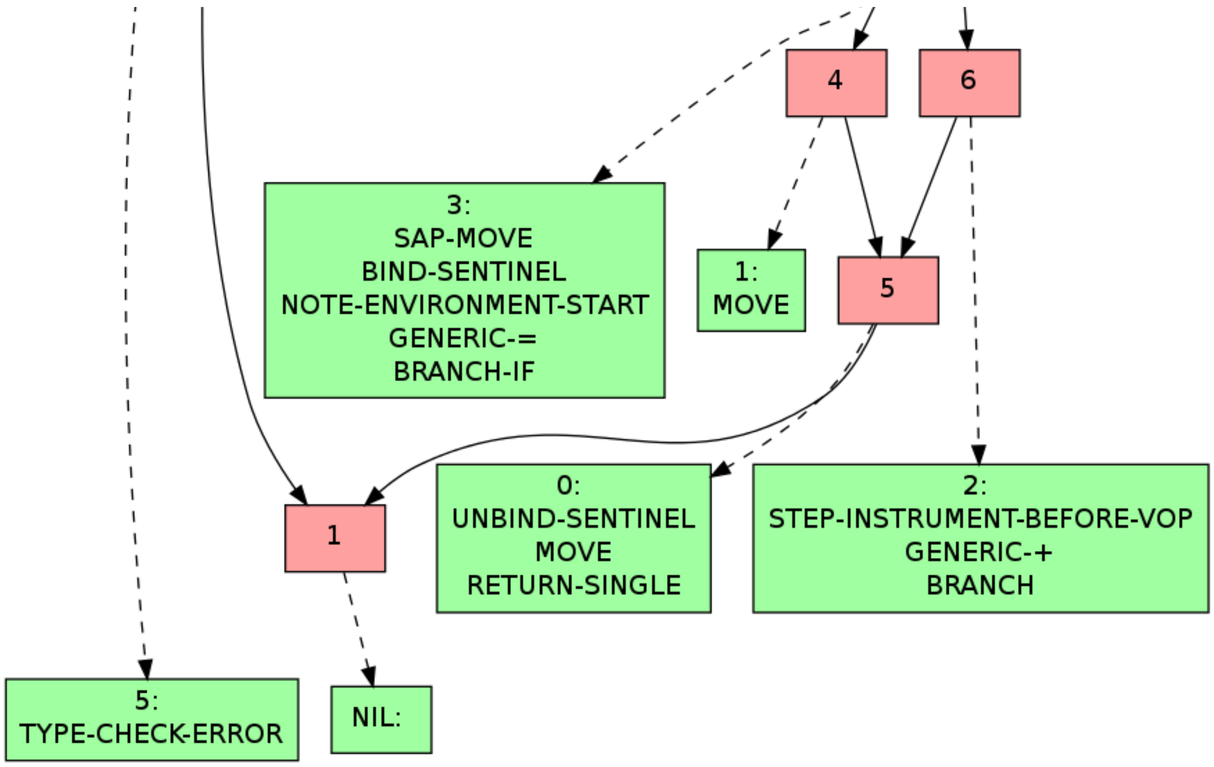
2.2 New simple coloring heuristic

This step will be dedicated to implementation of an alternative graph coloring method based on Briggs's colouring heuristic (without the spill insertion phase). This implementation will be then test-wise integrated in the SBCL compiler.

2.3 Visualization

For convenient debugging, we develop a visualization of the algorithmic steps (coloring etc.) with graphic plug-ins. An example of initial visualization can be found in the following Figure.





```
2.4 Static live range splitting
--------------------------------
In this step we plan to implement static aggressive live range
splitting based on the control flow graph.


2.5 New allocator based on live ranges
---------------------------------------
In this step we rearchitect the allocator to work with live ranges
 rather than with variables.


2.6 Spill code insertion
-------------------------

In this step we extend the allocator by inserting spill code.

In order to improve choice of spills, we can  (optionally) implement
  different criteria that capture the quality of register allocation.
  For this purpose the well known and intuitive heuristics can be
  used:
+ spill temporaries with most conflicts
+ spill temporaries with few definitions and uses
+ avoid spilling in inner loops
+ combination of the above.
```

2.7 New interface for register allocation

In order to accommodate different heuristics, we define an interface that specifies the method of register allocation:

Current interface:

```
  allocate-registers(TNs) -> storage allocation
```

New interface:

```
  allocate-registers(TNs, allocation method) -> storage allocation
```

where the *allocation method* could be defined by coloring, iterative coloring, spills heuristic, etc.. By such accommodation of different allocation strategies, the interface will enable convenient development and comparison of different methods.

2.8 Iterative Graph coloring (optional)

This step implements the iterative heuristic method for register allocation *optimistic coloring* (Briggs et al., 1994). As a test benchmark I propose to use the SBCL code.

2.9 Code and Speed Analysis (optional)

In this step we
+ compare the generated code for different allocation methods
+ compare the speed of compiler for different implementations of register allocation

2.10 Schedule

Period	Plan	Deliverable
14 June - 27 June	- find examples of suboptimal allocation due to graph coloring (Sec.2.1) - start implementing new coloring heuristic in SBCL (Sec.2.2)	(quantitative) analysis of current problems in register allocation (RA)
28 June - 12 July	- finish preliminary implementation and integration of coloring heuristic (Sec.2.2) - visualization of steps (Sec.2.3)	- preliminary new graph coloring heuristic and integration in SBCL - new feature: visualization tool for error analysis
13 July - 26 July	static live range splitting (Sec.2.4)	new feature: live range splitting
27 July - 8 August (mid term)	finish up all preliminary issues: visualization, coloring, range splitting, SBCL-integration	to finish: - preliminary implementation of a new heuristic for RA, - visualization, - preliminary integration
9 August - 22 August	New allocator based on live ranges (Sec.2.5)	new feature: alternative allocator
23 August - 5 Sept.	Insertion of spill code (Sec.2.6)	new feature: simple code spills
5 Sept. - 23 Sept.	Implement and test new interface for register allocation (Sec.2.7)	to finish: - RA based on new heuristic graph coloring; - visualization - live range spilling - live range -based allocator - spill code insertion - new RA interface

```
3 Risks and challenges
=======================
+ more sophisticated allocator could perform worse in practice
+ compilation time could be increased through application of the
  additional procedures
+ it is not clear how to combine the new approach with the present
  heuristic
+ might perform differently on different architectures
```

The above-mentioned open questions have to be clarified based on conducted experiments.

4 Project-relevant background

4.1 degrees

+ diploma in computer science, Christian-Albrechts University , Kiel, Germany
+ PhD student in "Intelligent systems", Bielefeld University, Germany

4.2 studies

+ deep knowledge of Approximative and efficient
  algorithms,   Compilers, Graph theory

4.3 programming experience:

+ Python, Common Lisp, C++, (during PhD)
+ Matlab, Java, scheme, Haskell, sml, bash, 68000 (during studies)

4.4 tools

+ emacs, slime, gnuplot, graphviz

4.5 work experience

+ scheme and sml course-tutor (among other work)

5 Personal Motivation and Suitability

On the one hand, practical experience with assembler programming, lisp programming, and, on the other hand, a solid theoretical background in theoretical computer science and compilers, optimally qualify me for the participation in the GSoC.

As a dedicated user of open source software since childhood, I am highly willing to contribute to an open source project. SBCL has been indespensibe tool for achieving of the most results within the PhD studies.

No commitments for the period 17th of June to 23rd September.

6 Contact with community

During the community bonding period, I have followed the discussion on the SBCL irc-channel and the mailing list. The submitted bug patch can be found on the lauchpad under:

[https://bugs.launchpad.net/sbcl/+bug/789497]