

15-745 Project Proposal

Joshua Dunfield
joshuad@cs.cmu.edu

Aleksey Kliger
aleksey@cs.cmu.edu

March 17, 2003

Project web page

<http://www.cs.cmu.edu/~joshuad/compilers/>

Project description

Our goal is to implement tail call optimization in the SUIF framework, and measure its benefit on a mix of real and artificial programs. While this optimization is classic and widely used in compilers for functional languages such as Scheme and Standard ML, it appears to be less common in compilers for imperative languages. Since this can be attributed, at least in part, to the lower incidence of recursion in imperative programs, we will be pleasantly surprised if we find substantial benefits for “real” C programs.

Plan and schedule

We intend to proceed as follows:

- (1) Identify tail calls in SUIF and propagate to MachSUIF
- (2) Eliminate simple tail recursion
- (3) Run simple benchmarks
- (4) Identify and eliminate easy non-simple tail calls (strongly connected components?)
- (5) Analysis/transformation to handle tail calls through function pointers
- (6) Run benchmarks
- (7) Write final report and poster

Milestone

(1), (2), and (3) comprise the milestone.

Schedule

Week 1	03/17–03/23		(1)
Week 2	03/24–03/30	“break” (ICFP hell)	(1)
Week 3	03/31–04/06		(1), (2)
Week 4	04/07–04/13	joshuad @ ETAPS	(2), (3)
Week 5	04/14–04/20	Milestone 04/14	(4), (5)
Week 6	04/21–04/27		(5), (6), (7)
(Week 7)	04/28–04/30	Project Due 04/28; Poster 04/30	

We have scheduled a large amount of time to complete (1) through (3) due to time constraints: both of us are working on submissions to ICFP (deadline 03/29) and one of us is traveling to a conference in early April. We hope to reach our milestone (tasks (1) through (3)) ahead of schedule, as this would allow us to do a more thorough and sophisticated job on the rest of the project (particularly task (5)).

Literature search

Tail-call elimination is a fairly well-understood optimization. Descriptions of it appear in Muchnick [4], and Appel [1]. In functional languages, where recursion is often used to express iteration, tail-call elimination is key to space-efficient compilation. In fact, the Scheme language standard [3] specifies that an implementation must implement full tail call elimination to be considered fully-conforming.

Although GCC had simple tail-recursion elimination as early as 1988 [5], tail call elimination is not a common optimization in compilers for imperative languages. Some prior work [2] has been done on evaluating the effects of tail-recursion elimination in a C compiler. However we are not aware of any prior work evaluating the benefits of tail-call removal in a language such as C.

Resources Needed

We plan to use the SUIF2 and MachineSUIF compiler framework: the analysis of potential tail-call sites will be done in SUIF2, while actual stack manipulation and control-flow rewriting will be done in MachSUIF. We plan on using the ECE Alpha cluster for evaluation and benchmarking. We have access to all of these resources.

Ongoing work

We have not written any code yet, but we've started looking at SUIF2 to learn how to write an analysis pass and to place annotations that can survive the lowering pass to MachineSUIF. We've no other technical constraints, and are ready to proceed with the project.

References

- [1] A. W. Appel. *Modern Compiler Implementation in Java*. Cambridge University Press, 1998.
- [2] Mark W. Bailey and Nathan C. Weston. Performance benefits of tail recursion removal in procedural languages. Technical Report TR-2001-2, Department of Computer Science, Hamilton College, Clinton, NY, June 2001.
- [3] Richard Kelsey, William Clinger, and Jonathan Rees (Editors). Revised⁵ report on the algorithmic language Scheme. *ACM SIGPLAN Notices*, 33(9):26–76, 1998.
- [4] S. Muchnick. *Advanced Compiler Design and Implementation*. Morgan Kaufmann Publishers, 1997.
- [5] Richard M. Stallman. Tail recursion elimination in [G]CC. Internet mailing list post, 25 June 1988. URL: <http://www.geocrawler.com/archives/3/358/1988/6/0/1998441/>.