CS 6241 : Advanced Compiler Optimizations
Project
Due Date: April 13th 2014, Sunday midnight

Goal : Redundant array bounds check removal

Description: Our goal in this framework is to first make C code safe by inserting array bounds checking for each array reference occurring in the code. Array bounds involves ensuring that the loop index is within the bounds of array declarations for static (case such as int a[100]) or dynamic arrays – (int a[n] where n is compile time unknown but run time constant). At each array reference, checks that compute the array index expression are inserted and checked with the corresponding declared bound – for example consider the reference, a[2i+5] – the expression 2i+5 will be checked with the lower and upper bound of the declared array before accessing the array element and if violated a runtime exception will be thrown.

(1) Baseline: First, implement the array bounds check in the compiler and measure the code size increase and performance (speed) degradation for a large array intensive benchmarks supplied with LLVM.
(2) Now, our goal is to optimize the bounds checking. A bound check is redundant at a given program point if it is already covered by the preceding checks and its removal does not lead to a removal of exception that would have occurred if it were in place. Range check is one of the most important techniques that is used for determining the coverage of a check; enclosed are a couple of papers (authored by Rajiv Gupta) that define bounds checking redundancy and develop dataflow techniques for the same. Please implement the techniques developed in the papers.
(3) In this part, you will improve the bounds checking technique defined in (2). First you can improve the effectiveness of removal by undertaking CFG restructuring (see enclosed paper CGO 2008 authored by Thakur and Govindarajan). Then you can try to use more complex analysis such as using path conditions in conjunction with aggressive induction variable analysis or by using Global Value Numbering to find more cases of induction variable equivalences to yield more aggressive coverage of checks. You could extend the framework to use PRE style coverage of the check (note that straightforward use of PRE is not expected as a solution here – rather techniques to most efficiently cover a check using the concept of partial redundancy are expected). Thus, in this step you will propose and augment the baseline results of (2) by developing your own analysis which will pick up more cases of redundancy (2). You will demonstrate them on your own micro-benchmarks to prove the point and also show results on the large benchmarks. Suggestion: Think of how you can improve the techniques of the paper OR work with kernels and test cases to uncover the limitation of (2) and then augment it.

Deliverable: You will deliver modified LLVM with a phase built to insert and optimize the bounds checking. You will measure the performance of the same and separately compare the improvements : report baseline performance, improvement due to (2) only and improvement due to (2) and (3). Use the large array intensive benchmark to report the findings delivering a report that documents your techniques developed in (2) and (3), the cases found by (2) and not found by it and cases found by (3) and that are finally left out. The report should be limited to be about 5-6 pages.

Demo: Each team will demo the project setting up a 20 minute slot with the TA.