The SUIF Compiler System

SUIF = Stanford University Intermediate Format

Research compiler, especially for parallel machines

- slow, not production quality
- modular design supports experimentation
- intermediate representation and passes designed to support parallelization analysis

A LARGE software system

- 100K-200K lines of code
- 6+ years of development
- 6 PhDs completed, 8+ ongoing

Other parallelizing compiler infrastructures: Rice, Univ. of Illinois, IBM Yorktown Heights

SUIF Compiler Organization

Optimizations implemented as separate passes

- each pass is a separate program
- passes read and write (binary) SUIF files
- allows passes to be arbitrarily rearranged

Passing information between passes

- library contains routines common to different passes
- annotations may be attached to most parts of program (instructions, symbols)
- passes communicate by
  - modifying code
  - reading and writing annotations

Utilities

- printsuif generates text form
- s2c generates C form
- fortback generates Fortran form

Completely implemented in C++

Simple-SUIF Compiler

Subset of SUIF libraries, simplified intermediate representation

- C implementation
- program representation designed for scalar optimizations
- simpler types

Fully functioning compiler

- can run your optimization pass on any C program
- simple interface

```
ssfe foo                ; input in foo.f/foo.c
                        ; output in foo.tmp

printsimple foo.tmp     ; print ascii for
                        ; foo.tmp to stdout

<YOURPASS> foo.tmp foo.spx
                        ; result in foo.spx

ssbe foo                ; generate foo.out.c
                        ; and executable "foo"
```

Types and Memory

Types

- Types are used in
  - instructions (type of the result)
  - variables
- Simple-SUIF makes two of the properties easily accessible.
  - Size: (number of bytes of storage required)
  - Base type:
    * signed, unsigned, float, address
    * record (structures, arrays, unions)
    * void (no value present)
- In general, types can be complex user-defined types

Memory

- addressed by location
- remember, compiler works on any C program (including ones with arbitrary pointers)
- must be very conservative when performing optimizations on instructions involving memory accesses (without pointer analysis)

Infinite Register Space

Pseudo registers: can only be accessed by register number

- Motivation: keep variables separate whenever possible to retain opportunities for optimization

```
int f ()
{   int b;
    b = 1;
    *ptr = 2;
       = b;
}
```

- Candidates for pseudo registers?

Temporary registers

- Register with only one assignment and one use, all within a basic block
- Sometimes created by 3-address intermediate format to hold partial computation
- Avoids constraining optimization too early
- Distinction from pseudo registers?

Writing a Compiler Pass

Provided for you

- Simple-SUIF data structures and library function interfaces
- `main.cc` (written in C++)

  For each function or procedure in ⟨Cfile⟩.tmp

  - construct the Simple-SUIF data structures for the procedure
  - calls the procedure `do_procedure(pointer-to-instructions, proc_name)`

You supply the `do_procedure` function