Assignment #1: Introduction to SUIF

Due: Before class, January 29th

Introduction

In this assignment, you will compile a simple C program using SUIF, and debug and complete a simple peephole optimization pass.

The main purpose of the assignment is to introduce you to the SUIF compiler, rather than to test your knowledge of compilation techniques. Your success in the later homeworks and possibly your project depends upon learning to use SUIF. As you will see, we request that you submit only a small amount of material. In fact, in this assignment, we will provide you with a pointer to the solution binary (part 2 only). You can try it, and understand how it behaves in case you follow down the wrong path.

We encourage you to attempt to solve the problems below before looking at the solution. In any case, you should try our solution binary after you've grappled with the assignment as it will take time for you to become comfortable with the system.

Part 1

First, write a little C program that sorts its command line arguments (which should be integers) and prints the sorted list to standard out. It doesn't matter what kind of sort you use; keep it simple. Test this code using the GNU C compiler (`gcc`). Next look at the note on using the Machine SUIF compiler and find the directions (near the bottom) for setting up your shell environment. Run `setup-suif-localtree` and add the `setup-suif-env` script to your shell initialization file, as described, to define the environment variables needed for use of SUIF. Read the Machine SUIF Overview document to find out how to compile and run your program using SUIF. That document assumes that the reader knows how to run the base SUIF front end to prepare an input file written in C for processing by the Machine SUIF back end. But all you need to know is that (on the alphas) the `c2sby1` command invokes the necessary passes to transform a `.c` file into a `.suif` file, which is a SUIF intermediate file. E.g.,

```
c2sby1 hello.c
```

produces `hello.suif`. The next step in compilation is to use `do_lower`, which is discussed in the overview document. E.g.

```
do_lower hello.suif hello.lsf
```

produces another intermediate file that is the direct input to the first MachSUIF pass. Then you apply other MachSUIF passes until you have an assembly language (`.s`) file. (Part of your job in this assignment is to figure out which passes to run and in what order.) This you can assemble into an executable, e.g.

```
gcc -g -o hello hello.s
```

You will hand in your `hello.c` and the `hello.s` file produced by Machine SUIF. Also, please hand in a text file named `passes.txt` containing the ordered list of passes used to obtain the `hello.s` from `hello.c` and a short explanation of what each pass does and why you ordered them in such way.

Part 2

We have implemented a peephole optimization pass that removes unnecessary move instructions from a Machine-SUIF intermediate file. The code for this pass is in the directory:

```
/afs/cs.cmu.edu/academic/class/15745-s03/assignments/1/
```

Copy the whole directory to your own `$LOCAL_BASE` directory. E.g., run the commands:

```
mkdir -p ~/localnci/assignments/
cp -a /afs/cs.cmu.edu/academic/class/15745-s03/public/assignments/1 ~/localnci/assignments/
```

Now you can `cd` into the new directory and use `gmake` to compile the pass. (Remember to customize your shell environment as described above.) The peep pass comes with one test input called `input.suifcfg` which is already in the required format. You can use the program `do_print` to generate an ASCII version of this SUIF binary file that looks almost like a valid Digital Alpha assembly language file:

```
do_print input.suifcfg input.suif.txt
```

Once you have compiled `peep.cpp` and the other stuff in homework 1, you invoke the optimization pass as follows:

```
do_peep -debug 1 input.suifcfg output.suifcfg
```

The file `output.suifcfg` is the optimized output file; you can view this file with `do_print`. The switch `-debug 1` prints out statistics summarizing how many moves were eliminated.

What to do: We have introduced a few bugs into `peep.cpp`, and we have removed some code from one of the three useless-move patterns recognized (see `peep.cpp` for more details). By fixing the bugs and inserting a small bit of code, you should be able to have peep produce the debugging output found in the file `debug.good`. Please note that the bugs are bugs in the data-flow analysis or in the use of the SUIF and MachSUIF library functions. The bugs are not compilation bugs.

What will be turned in for part2: A fixed version of `peep.cpp`.

Where is the solution binary: The solution binary is located in the Machine SUIF tree used for the class. You should be able to use it if your environment is set correctly. One piece of advice, as Machine SUIF passes are organized as a driver executable and a shared library, please make sure you are using the correct version of both files (`libpeep.so` and `do_peep`) when trying out the solution.

Hand In

To hand in your solution create a tar ball (e.g. `tar cvf handin.tar ~/ncihome/1/tohandin/*`) with all the required files and copy it to:

```
/afs/cs.cmu.edu/academic/class/15745-s03/public/handin/<user_name>/<assignment #>
```

you can copy as many tar balls as you wish. Only the last version will be graded.

Good luck!

Acknowledgments

Thanks to Mike Smith and Glenn Holloway who originally created this assignment.