

In this assignment you will investigate the C-Breeze compiler by applying some transformation phases to C benchmarks and analyzing the change in execution time, by visualizing the control-flow graph generated by C-Breeze, and by writing a simple analysis. I recommend working in groups of size two.

1 Motivation

Doing research in the compilers area usually involves prototyping your algorithms and showing their effect on real codes. Instead of building a compiler infrastructure by scratch, most researchers use and contribute to an existing compiler infrastructure. In this class we will be investigating the C-Breeze compiler infrastructure.

Testing the results of previous optimizations and designing and implementing new optimizations are common activities that occur in compilers research. These activities require a certain set of skills: finding benchmarks, timing program execution, visualization, designing test cases, and implementing new analyses and transformations.

The goal of this assignment is to introduce you to many of these skills by working with real C benchmarks that you will find on the internet. You will also be designing some simple test cases yourself, learning how to time program execution, learning how to use the graphviz package for graph visualization, and learning how to write a simple analysis in C-Breeze.

2 Getting Started

1. Get a copy of the initial project2 files

```
% cp ~cs553/project2/Project2.tar .  
% tar xf Project2.tar
```

2. Build project2

```
% make
```

Building project 2 results in an executable called cbz. Here are some examples of cbz usage:

```
% ./cbz -ast test1.c  
% ./cbz -dismantle test1.c  
% ./cbz -cfg -c-code test1.c
```

The above commands generate a file called test1.p.c. Look at the file to see what the compiler is doing.

3 The Assignment

3.1 Gather Ye Benchmarks While Ye May

Find a small ($> 10^2$ LOC), medium ($> 10^3$ LOC), and large ($> 10^4$ LOC) C benchmark from the net with a small and large input set for each. WARNING: This probably won't be easy!

- There should have a separate directory for each benchmark under your project2 directory.
- The project2/Makefile should have a rule for "make testbench" that runs each benchmark through cbz with the command `./cbz -dismantle benchmark.cc`.
- The project2/Makefile should have a rule "make runbench" that runs each benchmark on its small and large input set.

Make sure to keep track of which benchmarks you are unable to compile with cbz. See the report requirements.

3.2 Timing the Results of C-Breeze Passes

Use the unix "time" command to the benchmark runs before and after applying constant propagation and copy propagation separately. For example, assuming the big benchmark executable was in the subdirectory big and was called bigexec:

```
% time big/bigexec smallInput
// recompile big using constprop
% ./cbz -cfg -constprop -c-code big/*.c
% time big/bigexec-cp smallInput
```

The commands above are not exactly what you need to do. You will have to organize things yourself so that the following invocations of make work:

- The project2/Makefile should have a rule for "make timebench" that runs runs and times each benchmark on its small and large input set.
- The project2/Makefile should have a rule "make timeconstpropbench" that runs each benchmark on its small and large input set after it has had constant propagation performed.
- The project2/Makefile should have a rule "make timecopypropbench" that runs each benchmark on its small and large input set after it has had constant propagation performed.

3.3 Writing a Walker

Write a program analysis that counts the number of addition, subtraction, multiplication, and division operations that occur within a program. Start by understanding the CountAssign example in the online documentation. (The code for this is already in the starting Project2 directory).

3.4 Visualizing Control-Flow Graphs

Visualization of graphs that result from program analysis can be immensely useful when it comes to debugging analysis implementations. `dot` (<http://www.graphviz.org/>) is a very popular and free graph visualization package, which already happens to be installed on the CS linux machines (yeah Wayne!). For this part of the assignment you will be generating a text dot file that describes the CFG created by C-Breeze and then using `dot` to visualize the graph by generating a postscript file.

To generate the dot file, write a walker for C-Breeze. The dot user guide at <http://www.graphviz.org/Documentation/dotguide.pdf> describes the various components of a dot file. The CFG dot files should have a node for each basic block, edges between nodes to represent the control flow, and the statements for the basic block should be the label for the corresponding node.

Here is the command you use to create the postscript file:

```
dot -Tps test.dot > test-dot.ps
```

You will need to use something like `ps2pdf` if you are using `pdflatex` to generate your report.

4 Hints for doing the assignment

For generating dot files, use `print_walker.h` and `print_walker.cc` as example of how you might generate a text description for a control-flow graph. The key methods will be `at_basicblock` in your Walker and `basicblocknode::preds()`. You will have to maintain a mapping between basic block nodes and unique integer ids.

Not all parts of the assignment depend upon one another. Therefore, you can skip around.

As always **you should start this assignment early**. I will be available during regular office hours and by email. I can look at your code and help point you in the right direction, but the amount of help I can give may be inversely proportional to the amount of time until the due date. By no means should you spend several hours trying to figure out a weird bug; consult me for help. When e-mailing me about the project, send a copy of the relevant section of your code (not as an attachment; send it as text appended to your message). Give a good description of what is going wrong, which should include information about the stack in a debugger.

5 Your Report

The report is an essential part of your completed assignment. Use it to describe your solution, assumptions, difficulties, insights, and results. Organize and present your document as if it were the only basis for your assignment's grade. The format of your writeup is up to you, but it should minimally answer the following questions:

- Where did you look for the benchmarks? Which benchmarks was `cbz` unable to compile? Present the following information in a table: source for benchmarks, phrase describing the benchmark, source lines of code (SLOC does not include comment lines, tools to calculate this are available), whether C-Breeze can compile it or not.
- Did the time command provide enough precision to time all of your benchmarks? If not what other timing options might you use?

- Present the before and after constant propagation and copy propagation timing results. Did they improve the performance of your benchmarks? Why or why not?
- Write three small C programs that contain different control constructs (eg. break, continue, while, for, if, etc.). Include the visualization of the CFG in your report. Did visualizing the CFG for some sample programs help you answer the previous question? How?
- What problems did you encounter while doing project 2? If you knew someone who was just about to start work on this assignment, what advice would you give them?
- What, if any, outside sources did you use (e.g., articles, books, other students)? This is particularly important. It's OK to look at books and articles and speak with your professor and fellow students (although sharing code and working together is strictly forbidden), but as with any scientific document, you should always cite your references and collaborations. You can either cite collaborations in footnotes or in a separate Acknowledgment section.

The results will cause this writeup to be longer than the one for project 1.

6 What to turn in

You will turn in your work using WebCT. Put your paper in your copy of the `Project2` directory, tar up a clean version of `Project2` (ie. do a `make clean`), and submit the tar ball.

Do not submit your executable or object files; these can be very large and might overflow the disk if many students were to submit them. Your writeup should be in one of the following formats: ASCII text, \LaTeX , or PDF.

Also, turn in a hard copy of your report at the **beginning** of class on the due date.

7 Due date

This assignment is due Friday October 21st, **at 2:00pm**. Late assignments will be penalized 10% per day.