gangliao / TIGER

implement a full compiler based on c++ 11 Edit

compiler parser scanner graph-coloring code-generator cplusplus-11 front-end backend Manage topics

317 commits 2 branches 1 release 2 contributors Apache-2.0

Branch: master New pull request Create new file Upload files Find file Clone or download

gangliao committed on GitHub Update README.md Latest commit e273962 18 seconds ago

cmake	add cmake compiler	16 days ago
deprecated	move testcase1 into deprecated	12 hours ago
design_doc	refine doc	8 hours ago
img	add demo	9 hours ago
src	Add more	9 hours ago
testCases2	Fix test5 cfg mode	14 hours ago
.gitignore	Add cmake	16 days ago
.pre-commit-config.yaml	Add clang-format	2 months ago
.travis.yml	load and store vars during blocks	2 days ago
CMakeLists.txt	Add cmake	16 days ago
LICENSE	Initial commit	3 months ago
Phase1_Testing_and_Output.pdf	refine doc	8 hours ago
Phase2_Testing_and_Output.pdf	fix syntax words	8 hours ago
README.md	Update README.md	18 seconds ago
report.pdf	update demo note	8 hours ago

README.md

TIGER - A Tiny Full Compiler

build passing

This tiny compiler includes both front end and back end.

> Front end: Grammar Rules, LL(1) Parse Table, Syntax and Semantic Check and Intermediate Code.

> Back end: IR Optimization (Intra-block CFG optimization), MIPS Register Allocation, Instruction Selection and Code Generation.

You can access https://github.com/gangliao/TIGER to view the documentation!

How to Build

0. development environment

   Currently, this project repository is maintained on github publicly and also been deployed on Travis CI. It supports both Ubuntu and Mac OS X.

1. build:

```
        # cd project dir
        cd Tiger-Compiler
        # build scanner, parser, generator
        mkdir build && cd build
        # cmake building tool
        cmake ..
        make -j4
```

2. run:

   You can parsing test cases named `*.tiger` under `/testCases2` to generate IR code. Default it will utilize CFG optimized technique to generate MIPS asm code.

   ```
           # verbose mode: "-d to implement a verbose mode"
           ./src/parser <filename> -d
   ```

   If you want to use the naive mode to generate asm code, simply issue:

   ```
           ./src/parser <filename> -d -naive
   ```

3. test:

   In testCases2 directory, it includes a test script `test.sh` to execute all test cases and generate the corresponding asm files `*.naive.s` and `*.cfg.s`.

   After souce code is compiled, you can simply issue the commands:

   ```
   cd testCases2
   sh ./test.sh
   ```

Demo

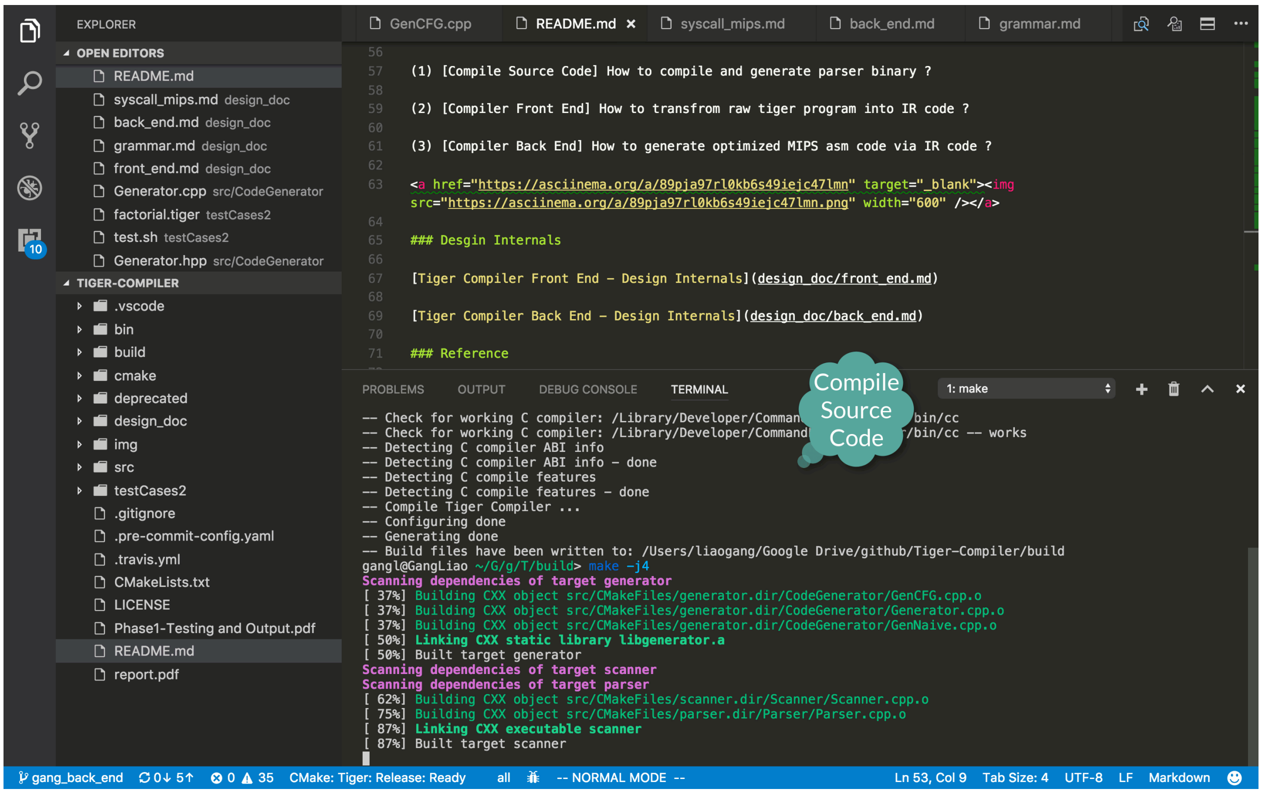
NOTE: This demo is `gif` graph format. If you have problem to view it in the markdown file, you can directly open it which located at `img/demo.gif`.

This Demo shows that

```
(1) [Compile Source Code] How to compile and generate parser binary ?

(2) [Compiler Front End] How to transfrom raw tiger program into IR code ?

(3) [Compiler Back End] How to generate optimized MIPS asm code via IR code ?
```



```
(1) [Compile Source Code] How to compile and generate parser binary ?

(2) [Compiler Front End] How to transfrom raw tiger program into IR code ?

(3) [Compiler Back End] How to generate optimized MIPS asm code via IR code ?

<a href="https://asciinema.org/a/89pja97rl0kb6s49iejc47lmn" target="_blank"><img
src="https://asciinema.org/a/89pja97rl0kb6s49iejc47lmn.png" width="600" /></a>

### Desgin Internals

[Tiger Compiler Front End - Design Internals](design_doc/front_end.md)

[Tiger Compiler Back End - Design Internals](design_doc/back_end.md)

### Reference
```

```
-- Check for working C compiler: /Library/Developer/Comman            bin/cc
-- Check for working C compiler: /Library/Developer/Command           /bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Compile Tiger Compiler ...
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/liaogang/Google Drive/github/Tiger-Compiler/build
gangl@GangLiao ~/G/g/T/build> make -j4
Scanning dependencies of target generator
[ 37%] Building CXX object src/CMakeFiles/generator.dir/CodeGenerator/GenCFG.cpp.o
[ 37%] Building CXX object src/CMakeFiles/generator.dir/CodeGenerator/Generator.cpp.o
[ 37%] Building CXX object src/CMakeFiles/generator.dir/CodeGenerator/GenNaive.cpp.o
[ 50%] Linking CXX static library libgenerator.a
[ 50%] Built target generator
Scanning dependencies of target scanner
Scanning dependencies of target parser
[ 62%] Building CXX object src/CMakeFiles/scanner.dir/Scanner/Scanner.cpp.o
[ 75%] Building CXX object src/CMakeFiles/parser.dir/Parser/Parser.cpp.o
[ 87%] Linking CXX executable scanner
[ 87%] Built target scanner
```

Desgin Internals

Tiger Compiler Front End - Design Internals

Tiger Compiler Back End - Design Internals

Test Cases

We passed all tests cases which provided by TA.

Please check out the details in report Phase2_Testing_and_Output.pdf from current directory, which includes test cases and their quality comparisons for naive and CFG intra-block register allocation.

Accomplishment

- [x] Register allocation code
  - [x] Naive
  - [x] CFG and intra block allocation
  - [ ] EBB and intra-EBB allocation
  - [x] Whole function register allocation
  - [x] Live Range Analysis and Graph Coloring
- [x] Instruction selection and generation code
- [x] Passes tests using generated code executing on simulator.
- [x] Report (desgin Internals, how to build, run, code quality comparisions, etc.)