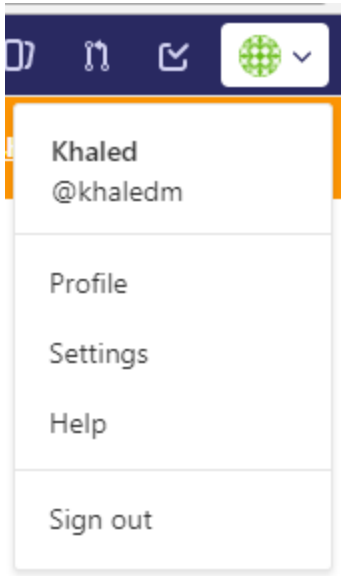
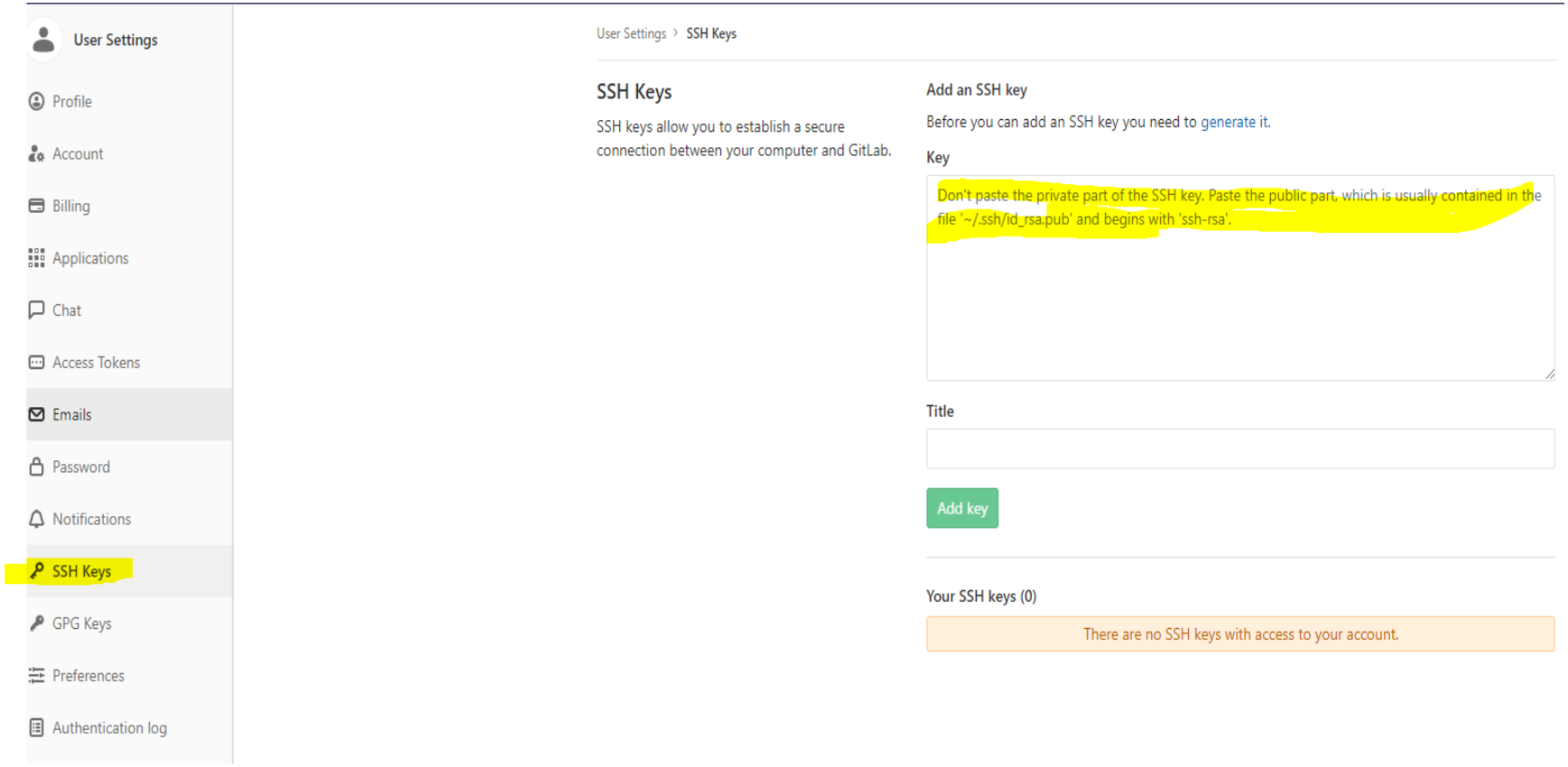
LLVM Setup

1. Create a gitlab.com account
2. Accept the invitation to join csus_llvm as a developer.
3. Assuming you already have a linux system up and running, create ssh keys: https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

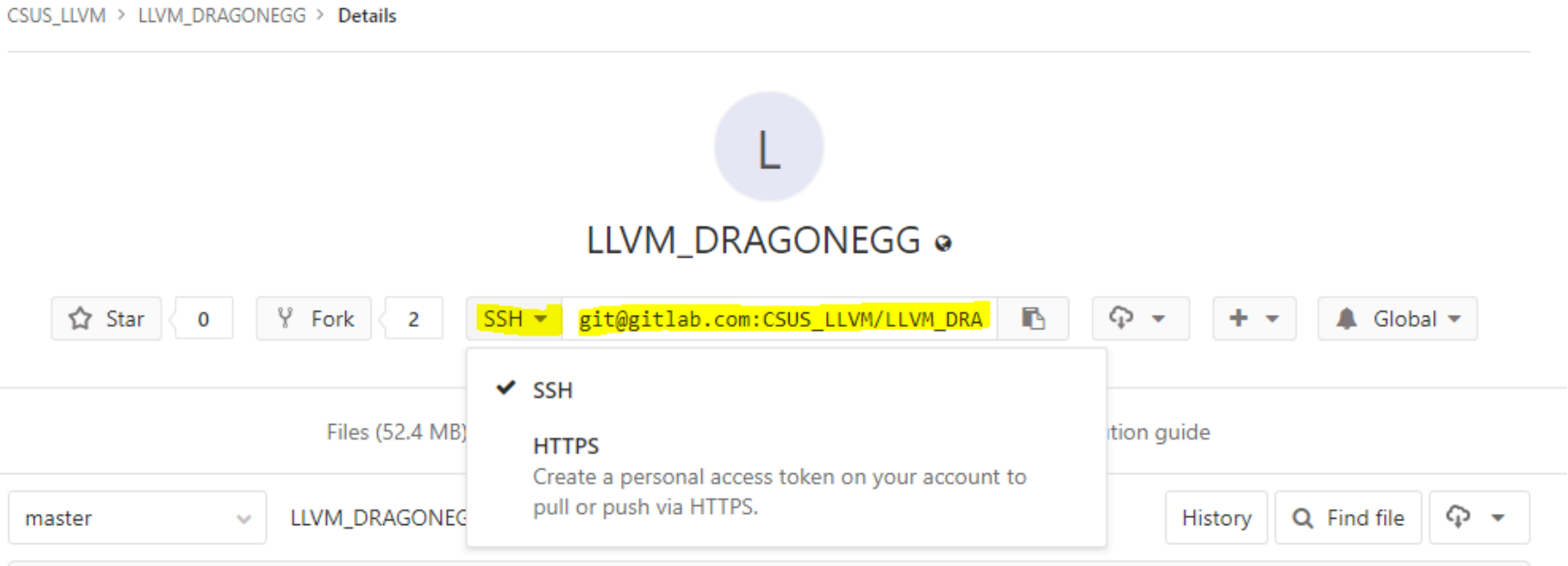
4. Save your ssh keys to gitlab.com
    a. Public key is likely located at ~/.ssh/id_rsa.pub, copy the contents of the file by:
        i. `cat ~/.ssh/id_rsa.pub`
        ii. Highlighting the entire key and copying to clipboard with `ctrl+shift+c`
    b. Sign into your account, and choose settings



c. Save your ssh key by selecting



5. Clone the project to your machine
    a. Locate the project in gitlab.com



    b. khaledm@csus-llvm:~$ git clone git@gitlab.com:CSUS_LLVM/LLVM_DRAGONEGG.git llvm_dragonegg

6. Download dragonegg from https://drive.google.com/drive/folders/0B0PcXgBFyHNqcXkyb0pCU3QxY2M
7. Copy it into llvm_dragonegg/Generic directory
8. Extract it using “tar zxvf dragonegg.tar.gz” (step 6-8 can be improved by copying dragonegg.tar.gz to the repository, and changing the CompilerLLVMandDRAGONEGG.sh to do extraction)
9. Building llvm by moving to the llvm_dragonegg directory and running:
    a. `./CompileLLVMandDRAGONEGG.sh`
    b. Choose to install the dependent tools/libs for the first time. (an improvement can be to automatically detect if a dependency is missing)
    c. Choose to build the release version.
    d. Wait. Then wait some more.

10. Testing the fresh compiler

```
khaledm@csus-llvm:~/llvm_dragonegg$ ls -l `which clang`
lrwxrwxrwx 1 root root 9 Feb 10 13:38 /usr/local/bin/clang -> clang-4.0
khaledm@csus-llvm:~/llvm_dragonegg$ mkdir mytests
khaledm@csus-llvm:~/llvm_dragonegg$ cd mytests/
khaledm@csus-llvm:~/llvm_dragonegg/mytests$ cat > test.cpp
#include <iostream>
int main() {
      std::cout << "Hello world" << std::endl;
      return 0;
}
Hit "Ctrl+D"
khaledm@csus-llvm:~/llvm_dragonegg/mytests$ clang++ test.cpp -o test
khaledm@csus-llvm:~/llvm_dragonegg/mytests$ ./test
Hello world
```

11. Compiling with our scheduler:
    a. Running `ClangRunOptSched.sh` will produce the command that will invoke our scheduler when you compile a test program:
        i. Open ClangRunOptSched.sh in any text editor and read the information about the script. Essentially it uses the options you select to produce a command that you can then use to compile a program using our scheduler.
        ii. Before you use it, however, you will need to ensure the script has the correct paths, particularly for `BASEDIR`. If your `BASEDIR` path is correct you should be good to go. If you followed the instructions above, you can use `BASEDIR=$(pwd)"/Generic"`
            1. Note: If you were able to compile your test program such as above by just invoking `clang++` (i.e. clang is already in your global path) then you should change line 51 to just `COMMAND="$cc $args $@"`
        iii. Run `./ClangRunOptSched -g++ -p test.cpp` to generate the command that you need to use to compile your program. Copy this.

b. Move to the directory with your test program and paste the code generated from the above step. When you execute you should get something like the following:

```
clang++ -O3 -mllvm -misched=optsched -mllvm -optsched-cfg=**/path/to/OptSchedCfg/** test.cpp

INFO: Machine model: x86-64 (Time = 114 ms)
INFO: ********** Opt Scheduling ********** (Time = 114 ms)
INFO: Building opt_sched DAG out of llvm DAG (Time = 114 ms)
```

.. and many more lines

CPU2006 Setup

1. Download CPU2006 from the google drive.
2. Unzip it in any location
    a. Use the command `tar xvzf CPU2006.tar.gz -C *path to target directory*`
3. From the root of CPU2006, run `source shrc`
4. Run `./install.sh`
5. Open Intel_llvm_3.9.cfg in a text editor. This file is located in the base LLVM_DRAGONEGG directory.
    a. Verify the paths between lines 30-40 match the paths of your machine. Save.
6. From the base of the CPU2006 directory run `cp *path to LLVM_DRAGONEGG*/Intel_llvm_3.9.cfg ./config/`
    a. I chose to use `cp` here to just keep a master copy of the Intel config file, you could use `mv` as well.
7. Still in the base of the CPU2006 directory run:
   `mv *path to LLVM_DRAGONEGG*/python_scripts/runspec-wrapper-v11_spillCost.py .` (the period is important)
8. Open the runspec file in a text editor and ensure line 54 is uncommented and lines 53 and 55 are commented:

```
51  # Regular expressions.
52  SETTING_REGEX = re.compile(r'\bUSE_OPT_SCHED\b.*')
53  #SPILLS_REGEX = re.compile(r'Function: (.*?)\nEND FAST RA: Number of spills: (\d+)\n')
54  SPILLS_REGEX = re.compile(r'Function: (.*?)\nGREEDY RA: Number of spilled live ranges: (\d+)')
55  #SPILLS_REGEX = re.compile(r'Function: (.*?)\nTotal Simulated Spills: (\d+)')
```

9. You also should clear out the logs that traveled along with the CPU2006 file.
    a. Change to the results directory
    b. Run `rm -rf *`
10. Final task is to open the runspec file in a text editor. This can be found from the base CPU2006 directory in the bin directory. Change the first two lines to the proper path for your setup. You basically just need to ensure that the path up until the CPU2006 directory:

```
1  #!/home/joe/csc199/CPU2006/bin/specperl
2  #!/home/joe/csc199/CPU2006/bin/specperl -d
3  #!/usr/bin/perl
4  #
```

11. You're now ready to run some benchmarks. Change to the base CPU2006 directory.
    a. Run `./runspec-wrapper-v11_spillCost.py -o data/ -b lbm` to test your setup.

b. Change to the new data directory to view your results.

12. You can view some options for the runspec wrapper with:

```
./runspec-wrapper-v11_spillCost.py --help
```