**Project step 1:**

Follow the instrutions and materials in <https://blog.golang.org/profiling-go-programs> and recreate the experiments for 5 go programs to optimize the program.

The go and g++ we are using:

$ go version

go version go1.8.1 darwin/amd64

$ g++ --version

Configured with: --prefix=/Applications/Xcode.app/Contents/Developer/usr --with-gxx-include-dir=/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/SDKs/MacOSX10.12.sdk/usr/include/c++/4.2.1

Apple LLVM version 8.0.0 (clang-800.0.42.1)

Target: x86\_64-apple-darwin16.5.0

Thread model: posix

InstalledDir: /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin

The blog states that the machine is running with CPU frequency scaling disabled. For MAC, however, I cannot disable CPU throttling or power control under OS X for Turbo Boost capable processors.

Following this link to get g++ from 4.2.1 default to 4.8.x:

http://www.ficksworkshop.com/blog/post/installing-gcc-on-mac

now the gcc become:

Using built-in specs.

COLLECT\_GCC=g++

COLLECT\_LTO\_WRAPPER=/opt/local/libexec/gcc/x86\_64-apple-darwin16/4.8.5/lto-wrapper

Target: x86\_64-apple-darwin16

Thread model: posix

gcc version 4.8.5 (MacPorts gcc48 4.8.5\_1)

The original program from the Gihub (<https://github.com/hundt98847/multi-language-bench)> has changed. The program in this blog is found here:

<https://github.com/rsc/benchgraffiti/tree/master/havlak>

For MAC, we need to install GUN-time first, using

$ brew install gnu-time

Then, we get the result:

$ time ./havlak1cc

# of loops: 76002 (total 3800100)

loop-0, nest: 0, depth: 0

real 0m22.679s

user 0m21.424s

sys 0m1.111s

We can also use the gtime command:

$ gtime ./havlak1cc

# of loops: 76002 (total 3800100)

loop-0, nest: 0, depth: 0

20.26user 1.04system 0:21.39elapsed 99%CPU (0avgtext+0avgdata 682246144maxresident)k

0inputs+0outputs (28major+297518minor)pagefaults 0swaps

The result is slightly different, but it takes around 20 seconds to run the program.

The representation of the term is shown as follow:

* real: Elapsed real (wall clock) time used by the process, in seconds.
* user: Total number of CPU-seconds that the process used directly (in user mode), in seconds.
* sys: Total number of CPU-seconds used by the system on behalf of the process (in kernel mode), in seconds.

According to some stack overflow answer, the correct command for MAC system is gtime

After digging through, we found that the full information can be seen as typing gtime –verbose:

$ gtime --verbose ./havlak1cc

# of loops: 76002 (total 3800100)

loop-0, nest: 0, depth: 0

Command being timed: "./havlak1cc"

User time (seconds): 20.78

System time (seconds): 1.00

Percent of CPU this job got: 99%

Elapsed (wall clock) time (h:mm:ss or m:ss): 0:21.86

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 672448512

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults: 263325

Voluntary context switches: 0

Involuntary context switches: 7881

Swaps: 0

File system inputs: 0

File system outputs: 0

Socket messages sent: 0

Socket messages received: 0

Signals delivered: 0

Page size (bytes): 4096

Exit status: 0

For getting a shorter report, we can simply use command :

$ gtime -f '%Uu %Ss %er %MkB %C' "$@" ./havlak1cc

and we got:

# of loops: 76002 (total 3800100)

loop-0, nest: 0, depth: 0

20.97u 1.02s 22.08r 679755776kB ./havlak1cc

The memory is huge compare to the blog.

Reading Linux “man” blog here:

<https://linux.die.net/man/1/time>

we got :

%M

Maximum resident set size of the process during its lifetime, in Kbytes.

Same for the go program:

$ gtime -f '%Uu %Ss %er %MkB %C' "$@" ./havlak1

# of loops: 76000 (including 1 artificial root node)

26.02u 0.24s 19.83r 1288896512kB ./havlak1