Project 2B

CS111

02/15/19

What's new ?

- lab2_list.c -> --lists option
  - --takes number of lists as a parameter
  - Initialize several shared lists
  - Decreases contention !
- profile.out -> execution profiling report
  - How much time was spent in spin-locks and mutexes?
  - Add 8th column of data to your .csv
- New graphs
  - Throughput
- gperftools -> multi-threaded application friendly
  - Heap checker
  - Heap profiler
  - Cpu-profiler -> what we'll use

gperftools - Install

- Releases found : https://github.com/gperftools/gperftools/releases
- Unpack and cd to directory
- ./configure --prefix=<path>
- make
- make install
- (make clean)

Rmk: You do not need root permission to do this

Rmk: You also need to install gv to access graphical output, this is optional

gperftools - Use

The pprof command will allow you to analyse the profiling result.

To turn on CPU profiling, you have two options:

1. Define environment variable CPUPROFILE to the filename to dump the profile to (in your makefile)

   Rmk: delete that file before each run

2. Bracket the code you want profiled using ProfilerStart() and ProfilerStop(). ProfilerStart() takes the profile filename as argument

Updating your makefile : profile option

1. Link the library:

LD_PRELOAD = <path_to_lib> (should be in /usr/lib(64))

2. Link file to record dump

CPUPROFILE = ./raw.gperf

3. pprof with --text

google-pprof --text <executable> <dump file> > $@

4. pprof with --list

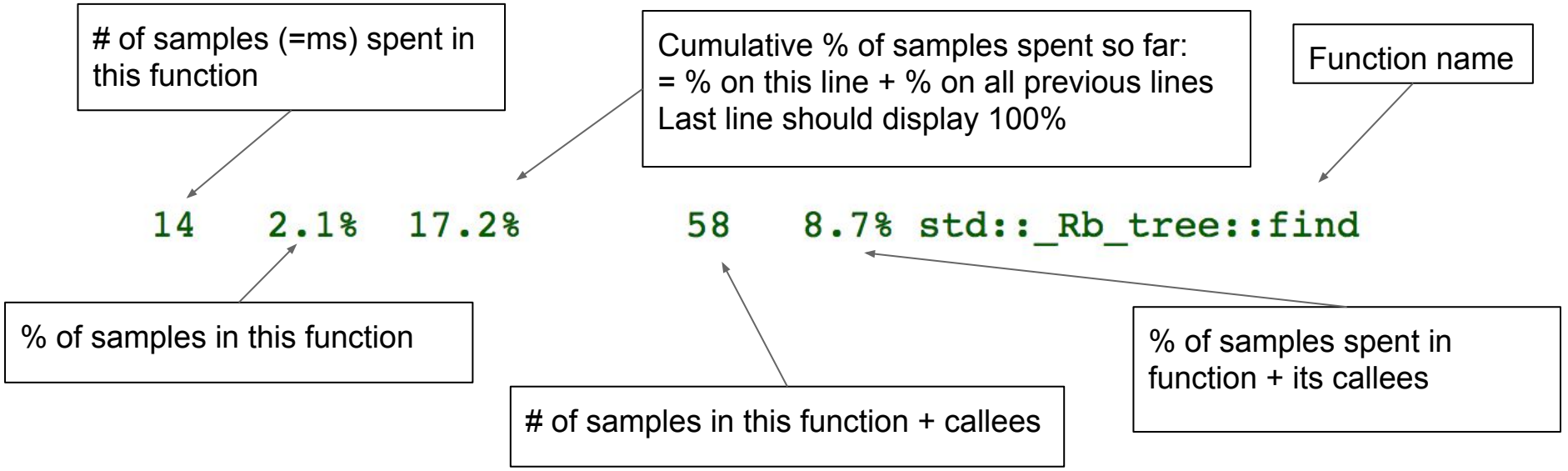
google-pprof --list=<function> <exec> <dump> >> $@

($@ is profile.out in this case)

Rmk: You also have to compile with the -lprofile flag

Analyzing the output

- For more information
- You **need** to use the text output, but if you first want to use the graphical output to make things clearer you can
- Text output will produce lines that look like:



Timing Mutex Waits

- Not spinning -> can’t use profiling
- Instead : we will compute wait-for-lock time
  - Measure time before and after getting the lock
  - Add up all wait time for all threads
  - Divide by number of operations
  - Output Statistics for the run

- If you are not locking -> should get 0
- For example:
  - Allocate an array of timers (# threads)
  - clock_gettime, pthread_mutex_lock([sublist]), clock_gettime
  - add time to timer[threadnum]

Addressing the Underlying Problem

- Degradation is a result of increased contention

- To decrease contention : split the list
  - Add a lists option to specify the number of chunks
  - Select which list to insert node in
  - Get list length
  - Delete inserted nodes
  - exit
- To select which list : hash function
  - Maps threads*iterations -> #lists (key_value % lists)