Performance Profiling Tools in Linux

Quick note: There are free and commercial profiling tools, of varying quality. We talked about that in class. Visual Studio has some built in, relatively poor profiling tools, and there are commercial ones which plugin to VS, but those both out of scope for us, and not hugely relevant anyway.

Today you are going to look at two for Linux.

I’ve based this tutorial on: <http://gernotklingler.com/blog/gprof-valgrind-gperftools-evaluation-tools-application-level-cpu-profiling-linux/> which is a few years old, but the basic concepts remain perfectly valid, along with a couple of other sources (notably <https://www.patricklam.ca/p4p/2014/notes/pdf/L16.pdf>)

Gprof (the gnu profiler) is and was fine for older applications and single threaded problems, but the two newer replacements are Valgrind and gperftools.

# Gperftools

sudo apt-get install graphviz

You also need

sudo apt-get install google-perftools libgoogle-perftools-dev

And

sudo apt install kcachegrind valgrind

You might also need pprof (I have it from making the lab, not sure if it’s required).

In principle Gperftools can run on any binary, so for any already compiled program “a.out” you can run

LD\_PRELOAD=/usr/lib/libprofiler.so CPUPROFILE=a.out.prof a.out

Where a.out.prof is the profile output you want to create. Then you make a call to

google-pprof --callgrind a.out a.out.prof > a.out.callgrind

Which will spit out some results. **Feel free to run on a program you have just to see the output**

That’s not really what you’d want to do though. Really what you want is to profile a program where you have access to the source code.

You can compile a program with gperftools/profiler.h libraries

You can use the profiling library directly as well:

#include <gperftools / profiler. h>

Then, bracket code you want profiled with:

ProfilerStart(nameofprofile.log )

// . . .

ProfilerEnd( )

And then analyse with pprof.

An example of this on a program called ‘cpuload.cpp’

*#!/bin/bash*

*# build the program; For our demo program, we specify -DWITHGPERFTOOLS to enable the gperftools specific #ifdefs*

g++ -std=c++11 -DWITHGPERFTOOLS -lprofiler -g cpuload.cpp -o cpuload

*# run the program; generates the profiling data file (profile.log in our example)*

./cpuload

*# convert profile.log to callgrind compatible format*

pprof --callgrind ./cpuload profile.log > profile.callgrind

*# open profile.callgrind with kcachegrind*

kcachegrind profile.callgrind

You may need the bash script to be more like:  
  
g++ -std=c++11 -DWITHGPERFTOOLS -Wl,--no-as-needed -lprofiler -Wl,--as-needed -g cpuload.cpp -o cpuload

If you’re getting a undefined reference to profilerStart

cpuload.cpp

|  |
| --- |
| #include <vector>  #include <iostream>  #ifdef WITHGPERFTOOLS  #include <gperftools/profiler.h>  #endif  using namespace std;  int foo(vector<int> v) {  int result = 0;  for(auto x: v) {  result += x;  }  return result % 1000;  }  int main() {  #ifdef WITHGPERFTOOLS  ProfilerStart("profile.log");  #endif  vector<int> v;  v.push\_back(1);    int result = 0;  for (int i=0; i<10000; i++) {  result = foo(v);  v.push\_back(result);  }  #ifdef WITHGPERFTOOLS  ProfilerStop();  #endif  cout << result << "\n";  return 1;  } |

Try it first with that sample program**, then give it a whirl on one of your other programs (maybe the boost one, this isn’t likely to play nice with MPI)**

# Valgrind

Installation:

(this should have been included in stuff above but if not: Sudo apt install valgrind)

An example of a program with problems (this deliberately has problems in it, if you don’t compile with -fpermissive it won’t compile at all).

I called this ‘memprobs.cpp’

gcc -fpermissive memprobs.cpp -o memprobs.out

#include <stdlib.h>

void f(void)

{

int\* x = malloc(10 \* sizeof(int));

x[10] = 0; // problem 1: heap block overrun

} // problem 2: memory leak -- x not freed

int main(void)

{

f();

return 0;

}

Run with

valgrind --leak-check=yes ./memprobs.out

(General form)

valgrind --leak-check=yes ./myprog arg1 arg2

Run that on the sample program, give it a shot on some of the other stuff you’ve done and see what you can see. Take some screenshots, submit those (sample program from the lab + 1 other program)