SOFTWARE ENGINEERING LAB ASSIGNMENT-2

Gperftools - is a set of tools for performance profiling and memory checking

GROUP NO.:12

510518082 SAYAK CHAKRABORTY

510518014 ANIRUDDHA HAZRA

510517045 SREETAM GANGULY

510518041 ANKITA MARANDI

Introduction to gperftools:

One of the most used profilers is available as a part of the *gperftools* package. It was originally named Google Performance Tools, the code is currently maintained by the community and distributed under the BSD license. *gperftools* provides a statistical CPU profiler, *pprof*, and several tools based around the *tcmalloc* (thread-caching malloc) library. Besides offering an improved memory allocation library for multithreaded environments, *tcmalloc* library supports memory leak detection and dynamic memory allocation profiling.

It is also possible to create call graphs using it. The panorama depicts the amount of time spent in each function/routine and their count.All the functions are represented by boxes whose size depends on the amount of time spent within it. Edges between the functions are labeled with the number of samples between them.

Installation on Linux:

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

Basic CPU Profiling:

Below a simple example has been shown.Here the entire program has been profiled but we can selectively profile only some parts by using #include <gperftools/profiler.h> and surround the sections we want to profile with ProfilerStart("nameOfProfile.prof"); and ProfilerStop();

Compilation:

We have to compile our code using the following flags for linking purposes.

```
gcc -O2 -g mvmult.c -o mvmult -ggdb -lblas -Wl,-no-as-needed
-lprofiler
```

Creation of profiler and looking at the results :

The default sampling frequency is 100 samples per second. However some programs barely run for 1s. So the sampling speed can be further increased to a maximum of 1000 samples per second by using CPUPROFILE_FREQUENCY environment variable.

```
env CPUPROFILE=mvmult.prof ./mvmult 15000
```

```
google-pprof --text mvmult mvmult.prof
```

```
env CPUPROFILE=mvmult1K.prof CPUPROFILE_FREQUENCY=1000
./mvmult 15000
```

```
google-pprof --text mvmult mvmult1K.prof
```

```
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --text mvmult mvmult.prof
Using local file mvmult.
Using local file mvmult.prof.
Total: 128 samples
      69  53.9%  53.9%       69  53.9% init
      58  45.3%  99.2%       58  45.3% ATL_ddot_xp1yp1aXbX
       1   0.8% 100.0%        1   0.8% _int_malloc
       0   0.0% 100.0%        1   0.8% _IO_new_file_overflow
       0   0.0% 100.0%        1   0.8% _IO_new_file_xsputn
       0   0.0% 100.0%        1   0.8% _IO_vfprintf_internal
       0   0.0% 100.0%        1   0.8% __GI__IO_doallocbuf
       0   0.0% 100.0%        1   0.8% __GI__IO_file_doallocate
       0   0.0% 100.0%        1   0.8% __GI___libc_malloc
       0   0.0% 100.0%        1   0.8% ___printf_chk
       0   0.0% 100.0%      128 100.0% __libc_start_main
       0   0.0% 100.0%      128 100.0% _start
       0   0.0% 100.0%      128 100.0% main
       0   0.0% 100.0%       58  45.3% mult
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --text mvmult mvmult1K.prof
Using local file mvmult.
Using local file mvmult1K.prof.
Total: 271 samples
     146  53.9%  53.9%      146  53.9% ATL_ddot_xp1yp1aXbX
     125  46.1% 100.0%      125  46.1% init
       0   0.0% 100.0%      271 100.0% __libc_start_main
       0   0.0% 100.0%      271 100.0% _start
       0   0.0% 100.0%      271 100.0% main
       0   0.0% 100.0%      146  53.9% mult
```

```
PROFILE: interrupts/evictions/bytes = 181/0/328
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --text mvmult mvmult.prof
Using local file mvmult.
Using local file mvmult.prof.
Total: 181 samples
     119  65.7%  65.7%      119  65.7% init
      62  34.3% 100.0%       62  34.3% ATL_ddot_xp1yp1aXbX
       0   0.0% 100.0%      181 100.0% __libc_start_main
       0   0.0% 100.0%      181 100.0% _start
       0   0.0% 100.0%      181 100.0% main
       0   0.0% 100.0%       62  34.3% mult
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ env CPUPROFILE=mvmult1K.prof CPUPROFILE_FREQUENCY=1000 ./mvmult 15000
Size 15000; abs. sum: 7500.000000 (expected: 7500)
PROFILE: interrupts/evictions/bytes = 321/7/824
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --text mvmult mvmult1K.prof
Using local file mvmult.
Using local file mvmult1K.prof.
Total: 321 samples
     167  52.0%  52.0%      167  52.0% init
     154  48.0% 100.0%      154  48.0% ATL_ddot_xp1yp1aXbX
       0   0.0% 100.0%      321 100.0% __libc_start_main
       0   0.0% 100.0%      321 100.0% _start
       0   0.0% 100.0%      321 100.0% main
       0   0.0% 100.0%      154  48.0% mult
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --list=init --lines mvmult mvmult1K.prof
Using local file mvmult.
```

Reporting Granularity:

By default the the output of pprof is shown at function granularity

```
google-pprof --list=init --lines mvmult mvmult1K.prof
```

```
      0   0.0% 100.0%      154  48.0% mult
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --list=init --lines mvmult mvmult1K.prof
Using local file mvmult.
Using local file mvmult1K.prof.
ROUTINE ====================== init in /home/sayak/Desktop/Lab/SE/mvmult.c
   167    167 Total samples (flat / cumulative)
     .      .    1: #include <stdio.h>
     .      .    2: #include <stdlib.h>
     .      .    3: #include <cblas.h>
     .      .    4: #include <time.h>
     .      .    5:
---
     .      .    6: void init(int n, double **m, double **v, double **p, int trans) {
     .      .    7:     *m = calloc(n*n, sizeof(double));
     .      .    8:     *v = calloc(n, sizeof(double));
     .      .    9:     *p = calloc(n, sizeof(double));
     .      .   10:     for (int i = 0; i < n; i++) {
     .      .   11:             (*v)[i] = (i & 1)? -1.0: 1.0;
     .      .   12:             if (trans)
     .      .   13:                     for (int j = 0; j <= i; j++)
     .      .   14:                             (*m)[j*n+i] = 1.0;
     .      .   15:             else
     1      1   16:                     for (int j = 0; j <= i; j++)
   166    166   17:                             (*m)[i*n+j] = 1.0;
     .      .   18:     }
     .      .   19: }
---
     .      .   20:
     .      .   21: void mult(int size, double *m, double *v, double *p, int trans) {
     .      .   22:     int stride = trans? size: 1;
     .      .   23:     for (int i = 0; i < size; i++) {
     .      .   24:             int mi = trans? i: i*size;
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --disasm=ddot_ mvmult mvmult.prof
```

```
google-pprof --disasm=ddot_ mvmult mvmult.prof
```

```
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --disasm=ddot_ mvmult mvmult.prof
Using local file mvmult.
Using local file mvmult.prof.
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                    je     bc9f0 is longer than address length 16
ROUTINE ====================== ATL_ddot_xp1yp1aXbX
    62     62 samples (flat, cumulative) 34.3% of total
: No such file or directory
     .      .      13c450: test   %edi,%edi
     .      .      13c452: jle    13c488 <ATL_ddot_xp1yp1aXbX@@Base+0x38>
     .      .      13c454: lea    -0x1(%rdi),%eax
     .      .      13c457: pxor   %xmm0,%xmm0
     .      .      13c45b: lea    0x8(,%rax,8),%rdx
     .      .      13c463: xor    %eax,%eax
     .      .      13c465: nopl   (%rax)
    21     21      13c468: movsd  (%rsi,%rax,1),%xmm1
     .      .      13c46d: mulsd  (%rcx,%rax,1),%xmm1
    14     14      13c472: add    $0x8,%rax
     .      .      13c476: cmp    %rax,%rdx
     .      .      13c479: addsd  %xmm1,%xmm0
    27     27      13c47d: jne    13c468 <ATL_ddot_xp1yp1aXbX@@Base+0x18>
     .      .      13c47f: repz retq
     .      .      13c481: nopl   0x0(%rax)
     .      .      13c488: pxor   %xmm0,%xmm0
     .      .      13c48c: retq
     .      .      13c48d: nopl   (%rax)
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --evince mvmult.prof
```

Memory leak detection:

```
gcc -O2 mvmult.c -o mvmult -lblas -ltcmalloc
```

```
PPROF_PATH=/usr/bin/google-pprof HEAPCHECK=normal ./mvmult 15000
```

In the following screenshot we can see that a memory leak of 240000 bytes has been detected.

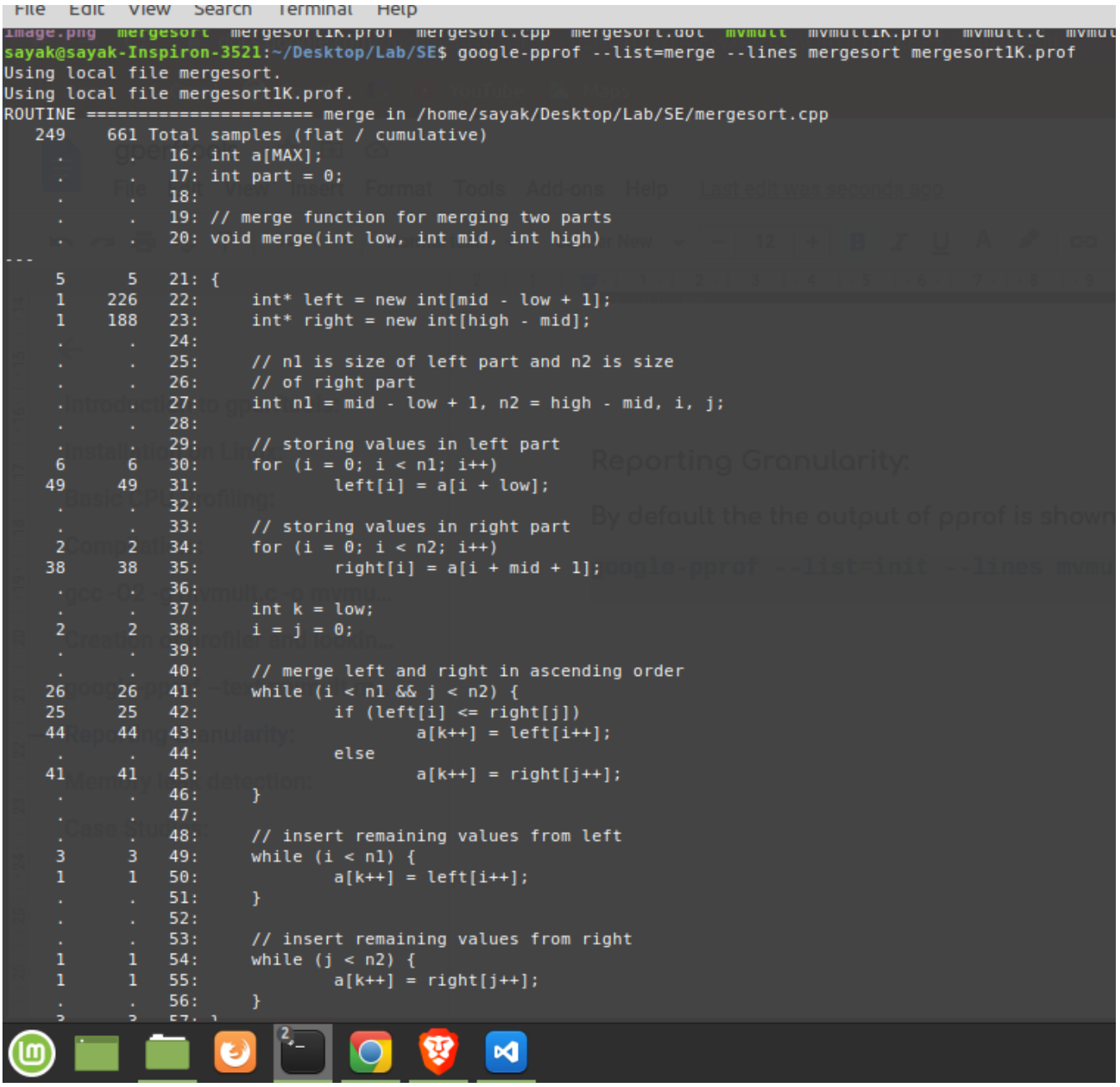
```
http://15000/pprof/symbol doesn't exist
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ PPROF_PATH=/usr/bin/google-pprof HEAPCHECK=normal ./mvmult 15000
WARNING: Perftools heap leak checker is active -- Performance may suffer
tcmalloc: large alloc 1800003584 bytes == 0x55c3903c2000 @  0x7fb595633001 0x55c38de07a11 0x55c38de07821 0x7fb595016b97 0x55c38de078fa
Size 15000; abs. sum: 7500.000000 (expected: 7500)
Have memory regions w/o callers: might report false leaks
Leak check _main_ detected leaks of 240000 bytes in 2 objects
The 2 largest leaks:
Using local file ./mvmult.
Leak of 120000 bytes in 1 objects allocated from:
        @ 55c38de07a32 init
        @ 55c38de07821 main
        @ 7fb595016b97 __libc_start_main
        @ 55c38de078fa _start
Leak of 120000 bytes in 1 objects allocated from:
        @ 55c38de07a22 init
        @ 55c38de07821 main
        @ 7fb595016b97 __libc_start_main
        @ 55c38de078fa _start


If the preceding stack traces are not enough to find the leaks, try running THIS shell command:

pprof ./mvmult "/tmp/mvmult.8692._main_-end.heap" --inuse_objects --lines --heapcheck  --edgefraction=1e-10 --nodefraction=1e-10 --gv

If you are still puzzled about why the leaks are there, try rerunning this program with HEAP_CHECK_TEST_POINTER_ALIGNMENT=1 and/or with HEAP_CHECK_MAX_POINTER_OFFSET=-1
If the leak report occurs in a small fraction of runs, try running with TCMALLOC_MAX_FREE_QUEUE_SIZE of few hundred MB or with TCMALLOC_RECLAIM_MEMORY=false, it might he
lp find leaks more repeata
Exiting with error code (instead of crashing) because of whole-program memory leaks
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$
```

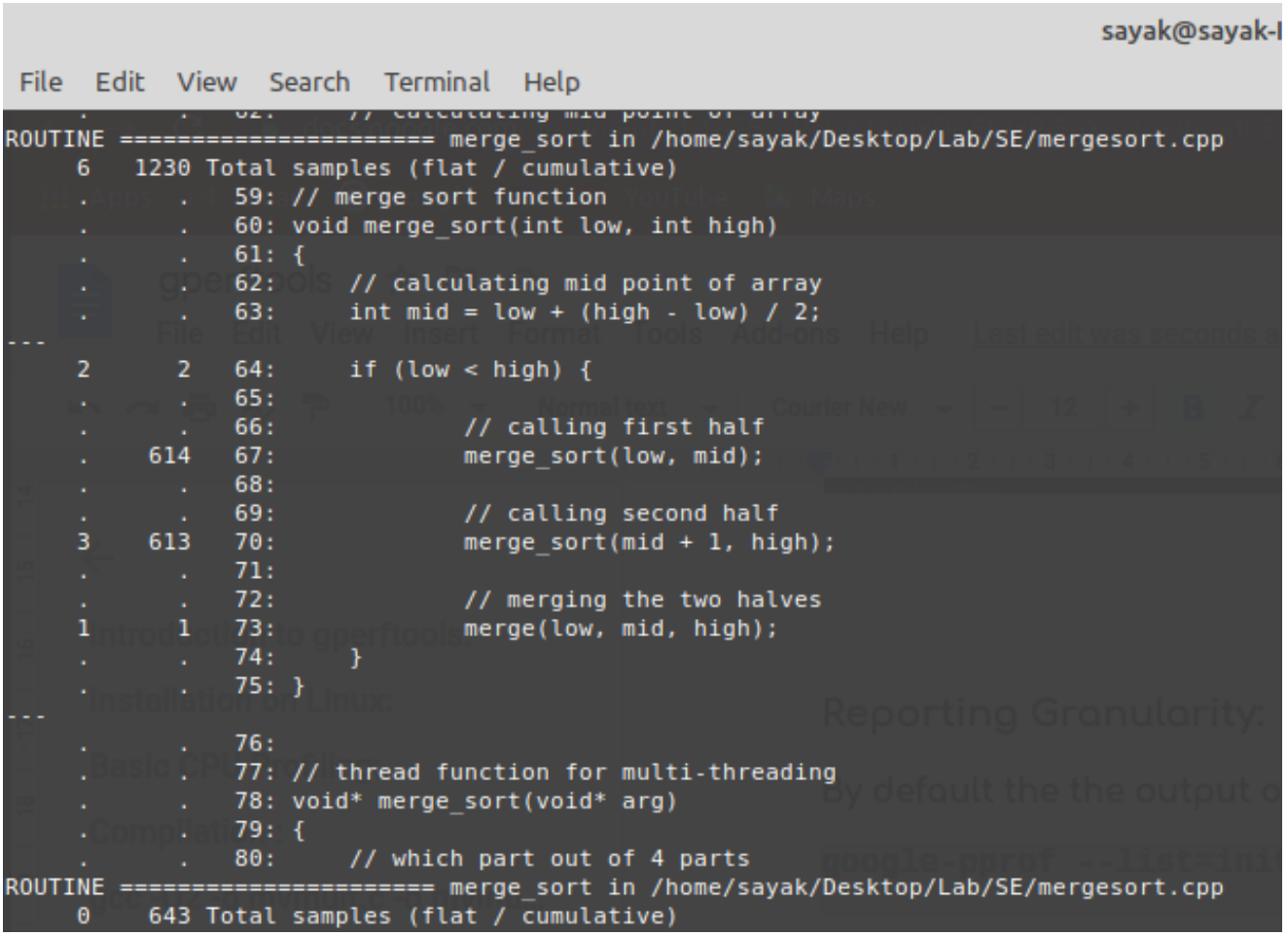
Case Studies:

Multi-threaded merge sort:

The gperftools profiler can profile multi-threaded applications. The run time overhead while profiling is very low and the applications run at “native speed”.

```
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --text mergesort mergesort1K.prof
Using local file mergesort.
Using local file mergesort1K.prof.
Total: 715 samples
     249  34.8%  34.8%      661  92.4% merge
     114  15.9%  50.8%      310  43.4% _int_malloc
     110  15.4%  66.2%      110  15.4% __GI___mprotect
      86  12.0%  78.2%      398  55.7% __GI___libc_malloc
      86  12.0%  90.2%       86  12.0% sysmalloc
      19   2.7%  92.9%       19   2.7% __random_r
      18   2.5%  95.4%       72  10.1% main
      11   1.5%  96.9%       30   4.2% __random
       8   1.1%  98.0%      407  56.9% operator new
       6   0.8%  98.9%      622  87.0% merge_sort@fe0
       3   0.4%  99.3%        3   0.4% std::__once_callable
       2   0.3%  99.6%        2   0.3% alloc_perturb
       2   0.3%  99.9%        2   0.3% operator new[]
       1   0.1% 100.0%        1   0.1% _init
       0   0.0% 100.0%      643  89.9% __GI___clone
       0   0.0% 100.0%       72  10.1% __libc_start_main
       0   0.0% 100.0%       72  10.1% _start
       0   0.0% 100.0%      110  15.4% grow_heap
       0   0.0% 100.0%      643  89.9% merge_sort@1030
       0   0.0% 100.0%       30   4.2% rand
       0   0.0% 100.0%      643  89.9% start_thread
```


File Edit View Search Terminal Help
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$ google-pprof --list=merge --lines mergesort mergesort1K.prof
Using local file mergesort.
Using local file mergesort1K.prof.
ROUTINE ====================== merge in /home/sayak/Desktop/Lab/SE/mergesort.cpp
249 661 Total samples (flat / cumulative)
. . 16: int a[MAX];
. . 17: int part = 0;
. . 18:
. . 19: // merge function for merging two parts
. . 20: void merge(int low, int mid, int high)
---
5 5 21: {
1 226 22: int* left = new int[mid - low + 1];
1 188 23: int* right = new int[high - mid];
. . 24:
. . 25: // n1 is size of left part and n2 is size
. . 26: // of right part
. . 27: int n1 = mid - low + 1, n2 = high - mid, i, j;
. . 28:
. . 29: // storing values in left part
6 6 30: for (i = 0; i < n1; i++)
49 49 31: left[i] = a[i + low];
. . 32:
. . 33: // storing values in right part
2 2 34: for (i = 0; i < n2; i++)
38 38 35: right[i] = a[i + mid + 1];
. . 36:
. . 37: int k = low;
2 2 38: i = j = 0;
. . 39:
. . 40: // merge left and right in ascending order
26 26 41: while (i < n1 && j < n2) {
25 25 42: if (left[i] <= right[j])
44 44 43: a[k++] = left[i++];
. . 44: else
41 41 45: a[k++] = right[j++];
. . 46: }
. . 47:
. . 48: // insert remaining values from left
3 3 49: while (i < n1) {
1 1 50: a[k++] = left[i++];
. . 51: }
. . 52:
. . 53: // insert remaining values from right
1 1 54: while (j < n2) {
1 1 55: a[k++] = right[j++];
. . 56: }


sayak@sayak-
File Edit View Search Terminal Help
ROUTINE ====================== merge_sort in /home/sayak/Desktop/Lab/SE/mergesort.cpp
6 1230 Total samples (flat / cumulative)
. . 59: // merge sort function
. . 60: void merge_sort(int low, int high)
. . 61: {
. . 62: // calculating mid point of array
. . 63: int mid = low + (high - low) / 2;
---
2 2 64: if (low < high) {
. . 65:
. . 66: // calling first half
. 614 67: merge_sort(low, mid);
. . 68:
. . 69: // calling second half
3 613 70: merge_sort(mid + 1, high);
. . 71:
. . 72: // merging the two halves
1 1 73: merge(low, mid, high);
. . 74: }
. . 75: }

. . 76:
. . 77: // thread function for multi-threading
. . 78: void* merge_sort(void* arg)
. . 79: {
. . 80: // which part out of 4 parts
ROUTINE ====================== merge_sort in /home/sayak/Desktop/Lab/SE/mergesort.cpp
0 643 Total samples (flat / cumulative)

```
      .      .   80:     // which part out of 4 parts
ROUTINE ====================== merge_sort in /home/sayak/Desktop/Lab/SE/mergesort.cpp
     0    643 Total samples (flat / cumulative)
      .      .   74:     }
      .      .   75: }
      .      .   76:
      .      .   77: // thread function for multi-threading
      .      .   78: void* merge_sort(void* arg)
---
      .      .   79: {
      .      .   80:     // which part out of 4 parts
      .      .   81:     int thread_part = part++;
      .      .   82:
      .      .   83:     // calculating low and high
      .      .   84:     int low = thread_part * (MAX / 4);
      .      .   85:     int high = (thread_part + 1) * (MAX / 4) - 1;
      .      .   86:
      .      .   87:     // evaluating mid point
      .      .   88:     int mid = low + (high - low) / 2;
      .      .   89:     if (low < high) {
      .    296   90:             merge_sort(low, mid);
      .    334   91:             merge_sort(mid + 1, high);
      .     13   92:             merge(low, mid, high);
      .      .   93:     }
      .      .   94: }
---
      .      .   95:
      .      .   96: // Driver Code
      .      .   97: int main()
      .      .   98: {
      .      .   99:     // generating random values in array
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
Warning: address    bc9e6:      74 08                   je     bc9f0 is longer than address length 16
sayak@sayak-Inspiron-3521:~/Desktop/Lab/SE$
```

Call-graph:

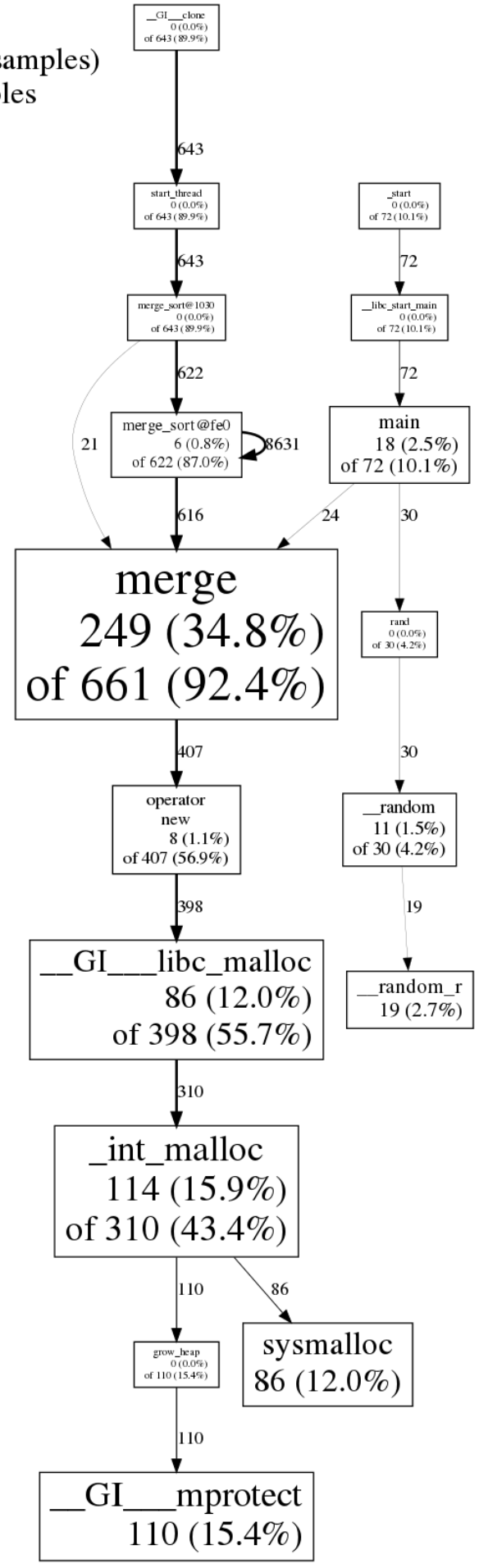
After generating the profile information, the following command can be used to create a call graph in a pdf formatted file.

```
google-pprof --pdf ./mergesort <profile_file> > mergesort.pdf
```

The call graph can be viewed using:

```
gv mergesort.pdf
```

./mergesort
Total samples: 715
Focusing on: 715
Dropped nodes with <= 3 abs(samples)
Dropped edges with <= 0 samples



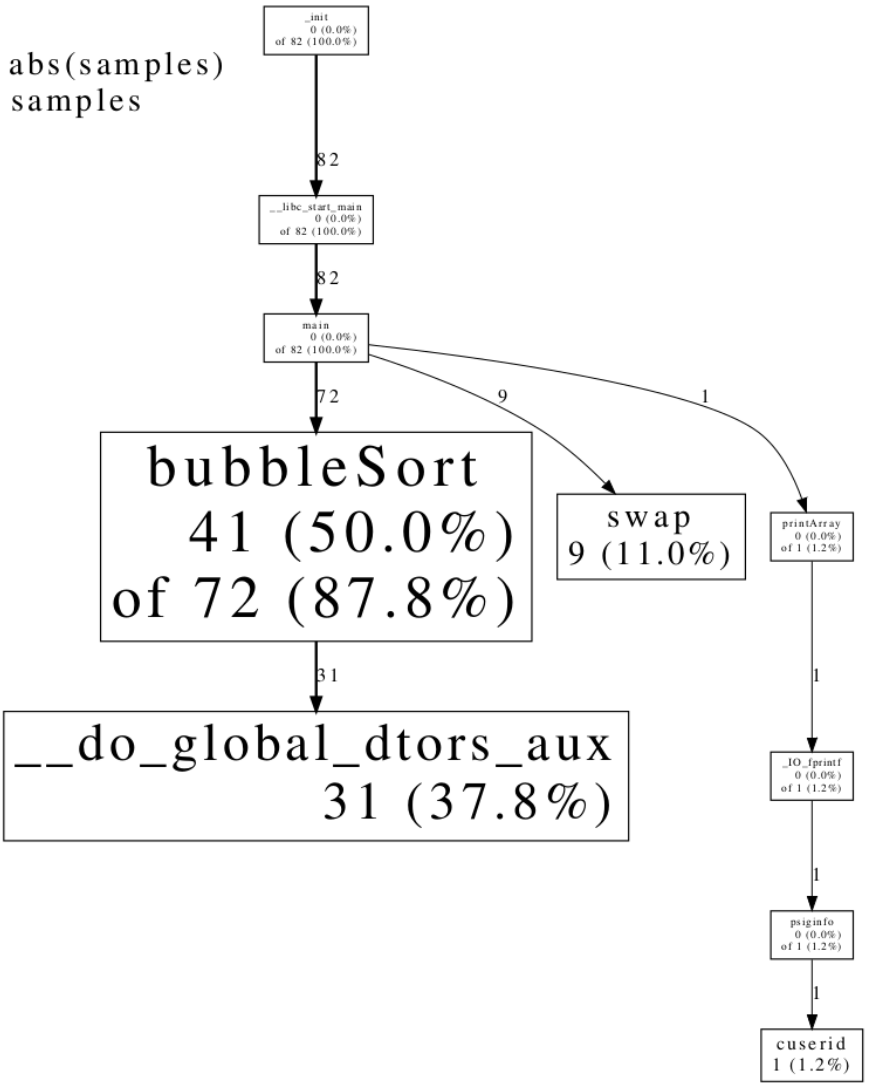
Bubble sort:

Sorting algorithms are some of the best use cases of CPU performance measurement tools. Bubble sort is a sorting algorithm of time complexity O(n^2). Here we have executed bubble sort on randomly generated 100k integers and the results found through google-pprof are as follows -

```
ani@Ani-HP:~/Desktop/SELab$ google-pprof --text ./bubbleperf ani@Ani-HP
Using local file ./bubbleperf.
Using local file ani@Ani-HP.
Total: 82 samples
      41  50.0%  50.0%       72  87.8% bubbleSort
      31  37.8%  87.8%       31  37.8% __do_global_dtors_aux
       9  11.0%  98.8%        9  11.0% swap
       1   1.2% 100.0%        1   1.2% cuserid
       0   0.0% 100.0%        1   1.2% _IO_fprintf
       0   0.0% 100.0%       82 100.0% __libc_start_main
       0   0.0% 100.0%       82 100.0% _init
       0   0.0% 100.0%       82 100.0% main
       0   0.0% 100.0%        1   1.2% printArray
       0   0.0% 100.0%        1   1.2% psiginfo
```

Call-graph:

```
./bubbleperf
Total samples: 82
Focusing on: 82
Dropped nodes with <= 0 abs(samples)
Dropped edges with <= 0 samples
```



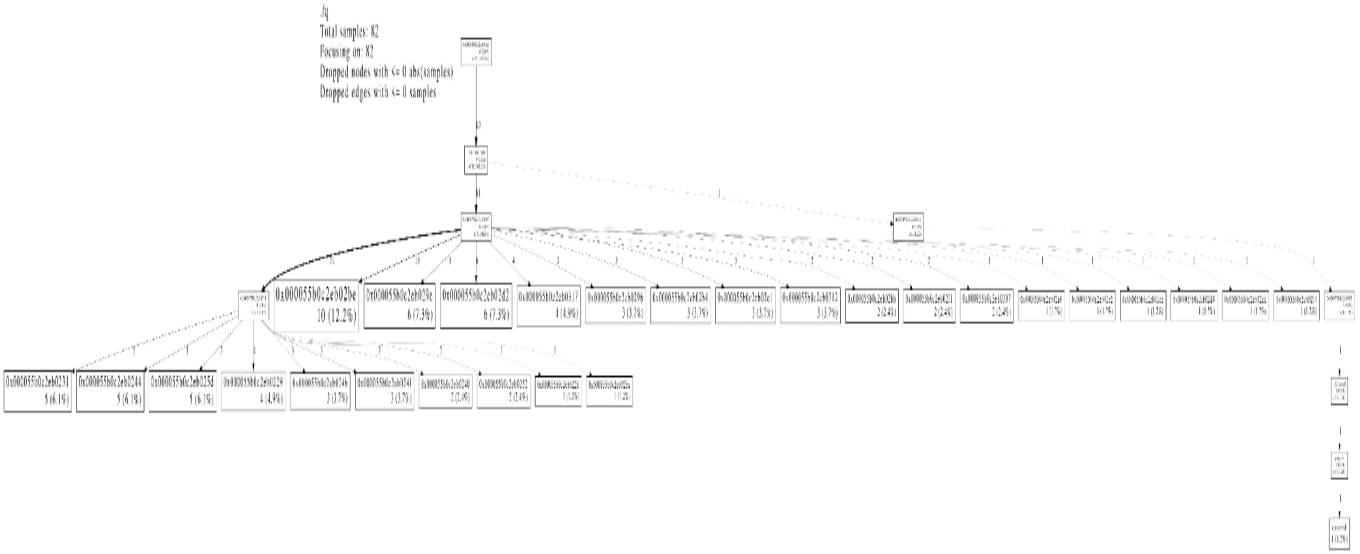
Quick sort:

QuickSort is a sorting algorithm of time complexity Θ(n*log n). So, to get a significant amount of run time and performance analysis we did sorting on randomly generated 1 crore (10 million) integers.

Results are as follows -

```
ani@Ani-HP:~/Desktop/SELab$ g++ -g quicksort.cpp -o q -lprofiler
ani@Ani-HP:~/Desktop/SELab$ ./q
Sorting Started
Sorting Ended
ani@Ani-HP:~/Desktop/SELab$ google-pprof --text ./q ani@Ani-HP
Using local file ./q.
Using local file ani@Ani-HP.
Total: 82 samples
      10  12.2%  12.2%       10  12.2% 0x000055b0c2eb02be
       6   7.3%  19.5%        6   7.3% 0x000055b0c2eb029e
       6   7.3%  26.8%        6   7.3% 0x000055b0c2eb02d2
       5   6.1%  32.9%        5   6.1% 0x000055b0c2eb0231
       5   6.1%  39.0%        5   6.1% 0x000055b0c2eb0244
       5   6.1%  45.1%        5   6.1% 0x000055b0c2eb025d
       4   4.9%  50.0%        4   4.9% 0x000055b0c2eb0229
       4   4.9%  54.9%        4   4.9% 0x000055b0c2eb0317
       3   3.7%  58.5%        3   3.7% 0x000055b0c2eb024b
       3   3.7%  62.2%        3   3.7% 0x000055b0c2eb024f
       3   3.7%  65.9%        3   3.7% 0x000055b0c2eb0296
       3   3.7%  69.5%        3   3.7% 0x000055b0c2eb02b4
       3   3.7%  73.2%        3   3.7% 0x000055b0c2eb02e1
       3   3.7%  76.8%        3   3.7% 0x000055b0c2eb0312
       2   2.4%  79.3%        2   2.4% 0x000055b0c2eb0240
       2   2.4%  81.7%        2   2.4% 0x000055b0c2eb0252
       2   2.4%  84.1%        2   2.4% 0x000055b0c2eb02bb
       2   2.4%  86.6%        2   2.4% 0x000055b0c2eb02f1
       2   2.4%  89.0%        2   2.4% 0x000055b0c2eb0307
       1   1.2%  90.2%        1   1.2% 0x000055b0c2eb022d
       1   1.2%  91.5%        1   1.2% 0x000055b0c2eb025a
       1   1.2%  92.7%        1   1.2% 0x000055b0c2eb02a8
       1   1.2%  93.9%        1   1.2% 0x000055b0c2eb02c2
       1   1.2%  95.1%        1   1.2% 0x000055b0c2eb02ce
       1   1.2%  96.3%        1   1.2% 0x000055b0c2eb02d8
       1   1.2%  97.6%        1   1.2% 0x000055b0c2eb02dd
       1   1.2%  98.8%        1   1.2% 0x000055b0c2eb02f4
       1   1.2% 100.0%        1   1.2% cuserid
       0   0.0% 100.0%       82 100.0% 0x000055b0c2eb016d
       0   0.0% 100.0%       31  37.8% 0x000055b0c2eb0311
       0   0.0% 100.0%        1   1.2% 0x000055b0c2eb0393
       0   0.0% 100.0%       81  98.8% 0x000055b0c2eb0485
       0   0.0% 100.0%        1   1.2% 0x000055b0c2eb04b1
       0   0.0% 100.0%        1   1.2% _IO_fprintf
       0   0.0% 100.0%       82 100.0% __libc_start_main
       0   0.0% 100.0%        1   1.2% psiginfo
```

Call-Graph:



Link to the quicksort call-graph - quicksort.pdf

Caveats:

- If the program exits because of a signal, the generated profile will be incomplete, and may perhaps be completely empty.
- The displayed graph may have disconnected regions because of the edge-dropping heuristics(can be controlled using edgefraction call-graph option)
- If you run the program on one machine, and profile it on another, and the shared libraries are different on the two machines, the profiling output may be confusing: samples that fall within shared libraries may be assigned to arbitrary procedures.
- If your program forks, the children will also be profiled (since they inherit the same CPUPROFILE setting). Each process is profiled separately; to distinguish the child profiles from the parent profile and from each other, all children will have their process-id appended to the CPUPROFILE name. However for multi-threading there is no such issue.

- Due to a hack that has been made to work around a possible gcc bug, the profiles may end up named strangely if the first character of the CPUPROFILE variable has ascii value greater than 127. This should be exceedingly rare, but if there's a need to use such a name, just prepend ./ to the filename: CPUPROFILE=./Ägypten.

Conclusion:

The gperftools CPU profiler has a very little runtime overhead, provides some nice features such as selectively profiling certain areas of interest and has no problem with multi-threaded applications. KCachegrind can be used to analyze the profiling data. Like all sampling based profilers, it suffers from statistical inaccuracy and therefore the results are not as accurate as with Valgrind. However, practically that’s usually not a big problem since we can always increase the sampling frequency if we need more accurate results.