Lab Report

ECPE 170 – Computer Systems and Networks – Spring 2016

Name: Dominic Lesaca

Lab Topic: Performance Measurement (Lab #: 5)

Question #1:
Create a table that shows the real, user, and system times measured for the bubble and tree sort algorithms

Answer:

Type of Sort	Real Time	User Time	System Time
Bubble Sort	1m 9.866s	1m 9.356s	0m 0.032s
Tree Sort	0.404s	0.196s	0.204s

Question #2:
In the sorting program, what actions take user time?
Answer:
The time it takes to compare the different values in the array.

Question #3:
In the sorting program, what actions take kernel time?
Answer:
Kernel Time is used when setting up virtual memory through using pointers.

Question #4:
Which sorting algorithm is fastest?
Answer:
Tree Sort is fastest

Question #5:
Create a table that shows the total Instruction Read (IR) counts for the bubble and tree sort routines. (In the text output format, IR is the default unit used. In the GUI program, un-check the "% Relative" button to have it display absolute counts of instruction reads).
Answer:

Bubble Sort IRs	Tree Sort IRs
164,783,088,449	473,255,164,473,255,164

Question #6:
Create a table that shows the top 3 most active functions for the bubble and tree sort programs by IR count (excluding main()) - Sort by the "self" column if you're using kcachegrind, so that you see the IR counts for *just* that function, and not include the IR count for functions that it calls.

Answer:

Bubble Sort		Tree Sort	
bubbleSort	164,775,225,641	insert_element'2	61, 385,975
verifySort	1,900,001	inorder'2	3,949,999
initArray	1,200,030	insert_element	3,283,852

Question #7:
Create a table that shows, for the bubble and tree sort programs, the most CPU intensive line that is part of the most CPU intensive function. (i.e. First, take the most active function for a program. Second, find the annotated source code for that function. Third, look inside the whole function code - what is the most active line? If by some chance it happens to be another function, drill down one more level and repeat.)

Answer:

	Line	cost
Bubble Sort	if(array_start[j-1] > array_start[j])	74,999,250,000
Tree Sort	Struct BtreeNode *newNode = (struct BtreeNode*)malloc(sizeof(struct BTreeNode));	369,616,186

After this I reduced the array size to 1000

Question #8:
Show the Valgrind output file for the merge sort with the intentional memory leak. Clearly highlight the line where Valgrind identified where the block was originally allocated.

Answer:

==4558== Memcheck, a memory error detector
==4558== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4558== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4558== Command: ./sorting_program merge
==4558== Parent PID: 3801
==4558==
==4558==
==4558== HEAP SUMMARY:
==4558== in use at exit: 4,000 bytes in 1 blocks
==4558== total heap usage: 2 allocs, 1 frees, 5,024 bytes allocated
==4558==
==4558== 4,000 bytes in 1 blocks are definitely lost in loss record 1 of 1
==4558== at 0x4C2FB55: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==4558== by 0x40085C: main (sorting.c:42)

==4558==
==4558== LEAK SUMMARY:
==4558== definitely lost: 4,000 bytes in 1 blocks
==4558== indirectly lost: 0 bytes in 0 blocks
==4558== possibly lost: 0 bytes in 0 blocks
==4558== still reachable: 0 bytes in 0 blocks
==4558== suppressed: 0 bytes in 0 blocks
==4558==
==4558== For counts of detected and suppressed errors, rerun with: -v
==4558== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

Question #9:
How many bytes were leaked in the buggy program?
Answer:
4000 bytes leaked

Question #10:
Show the Valgrind output file for the merge sort after you fixed the intentional leak.
Answer:
Show the Valgrind output file for the merge sort after you fixed the intentional leak.

Question #11:
Show the Valgrind output file for your tree sort.
Answer:

Question #12:
How many seconds of real, user, and system time were required to complete the amplify program execution with Lenna image (Lenna_org_1024.pgm)? Document the command used to measure this.
Answer:
real 0m0.004s
user 0m0.000s
sys 0m0.000s

Question #13:
Research and answer: why does real time != (user time + system time)?.
Answer:
real time may require waiting for the processor to finish which makes the time longer.

Question #14:
In your report, put the output image for the Lenna image titled output<somenumber>.pgm.

Answer:

Question #15:
Excluding **main()** and everything before it, what are the top three functions run by amplify executable when you **do** count sub-functions in the total? Document the commands used to measure this. Tip: Sorting by the "Incl" (Inclusive) column in kcachegrind should be what you want.

Answer:

commands:

unix> valgrind --tool=callgrind --dump-instr=yes --callgrind-out-file=callgrind.out
./amplify

```
unix>  callgrind_annotate --auto=yes --threshold=100 callgrind.out >
       callgrind_results.txt
unix>  gedit callgrind_results.txt &
```

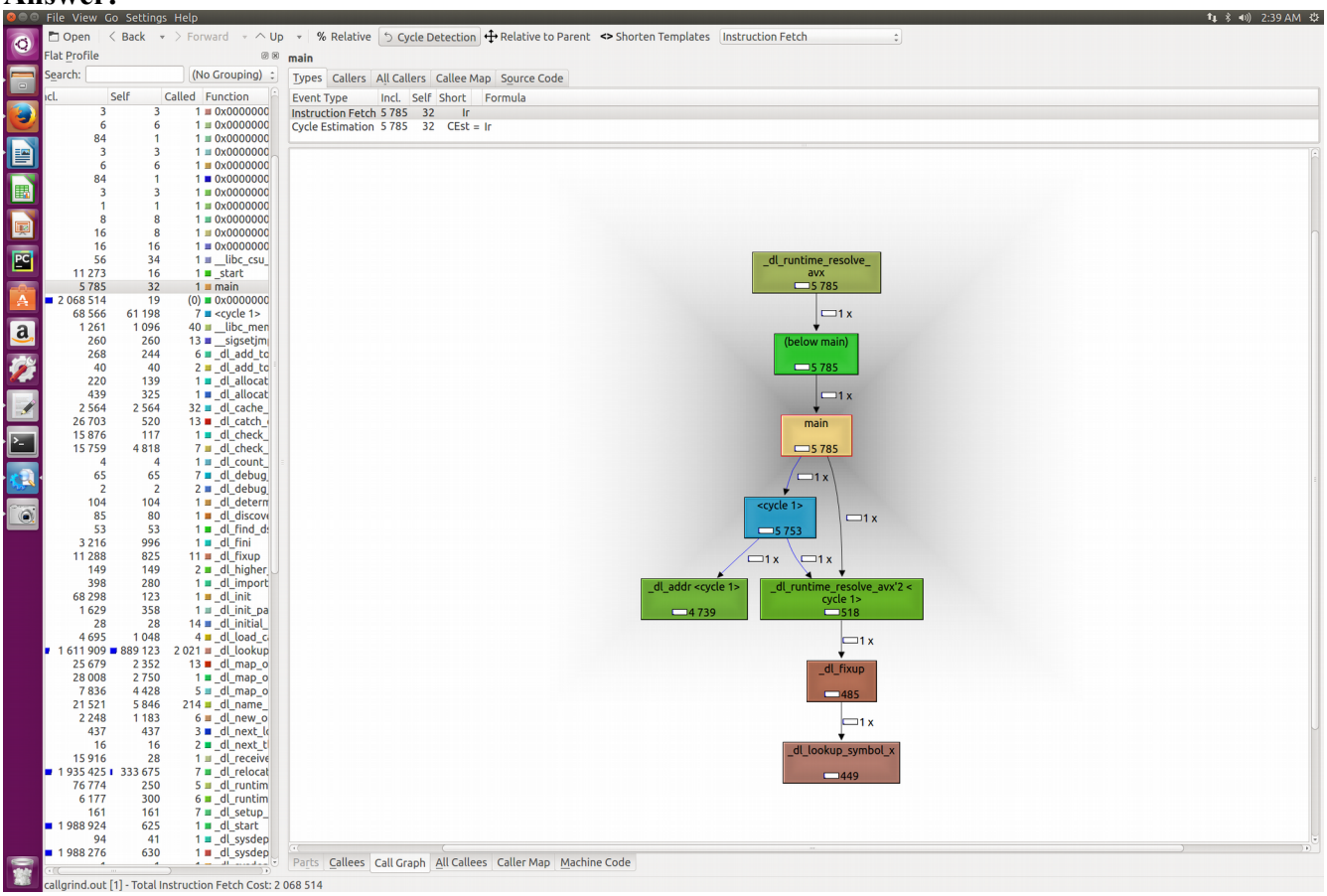
Results:

function	cost
_dl_start	1,988,924
_dl_sysdep_start	1,988,276
dl_main	1,987,586

Question #16:

Include a screen capture that shows the kcachegrind utility displaying the **callgraph** for the amplify executable, starting at main(), and going down for several levels.

Answer:



Question #17:

Which function dominates the execution time? What fraction of the total execution time does this function occupy?

Answer:

_dl_start takes the most time, it takes up 96.15%

Question #19:
Beyond inflicting pain and suffering on newbies, what are 3 advantages of using the command line to control a computer?
Answer:
1) the command line is more precise with its commands as you can do much more with one command line that several clicks in a GUI.
2) The command line is faster as your computer does not have to load the GUI of the computer with every command you give
3) the command line is easier to script for automation

Question #20:
What does one dot (.) mean in a file path? What do two dots (..) mean in a path?
Answer:
(.) refers to the folder you are currently in.
(..) refers to the folder above your current folder