Lab Report

ECPE 170 – Computer Systems and Networks – Fall 2021

Name: Dustin Schuette

Lab Topic: Performance measurement (Lab #: 05)

(1) Create a table that shows the real, user, and system times measured for the bubble and tree sort algorithms.

	bubble	tree
real	1.984s	0.021s
user	1.980s	0.016s
system	0.000s	0.000s

(2) In the sorting program, what actions take user time?

user time is the time it takes to run the application so the actual sorting process is done in this time.

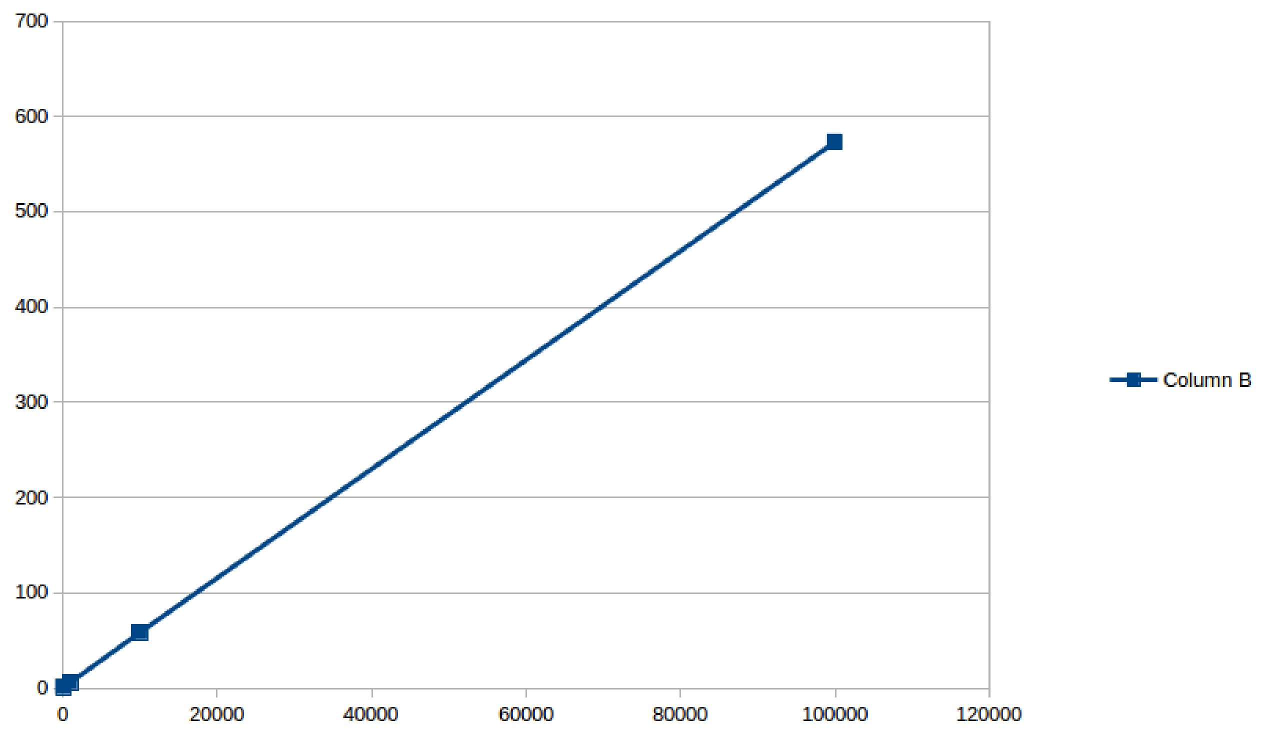
(3) In the sorting program, what actions take kernel time?

In this case none because the LInux kernel didn't have to service our code.

(4) Which sorting algorithm is fastest? (It's so obvious, you don't even need a timer)

Clearly tree sort is faster as it should be.

size	100	1,000	10,000	100,000
Bubble Time	0.001s	0.006s	0.292s	33.874s
Tree Time	0.001s	0.001s	0.005s	0.059
Speedup	1	6	58.4	574.13



Here we can see that the improvement is pretty linear as data sets get larger the speedup grows at a similar rate. This is to say a data set 10 times larger would result in 10 times the speedup. This makes sense as the fundamental method in which both sort functions work do not change as data sets grow and thus as they process more items bubble sort increases it's runtime exponentially while tree sort does not, allowing the speedup to keep growing linearly.

(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).

bubble	tree
10,304,705,196	21,831,756

(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.

	bubble	tree
func1	bubbleSort - 10,302,598,029	insert_element’2 - 12,421,787
func2	random_number - 649,194	_int_malloc - 3,750,105
func3	verifySort - 475,001	inorder’2 - 1,249,972

(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.)

bubble	tree
if (array_start[j-1] > array_start[j]) 4,687,312,500	if(element < ((*node)->element)) 2,024,470

(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.

```
==14849== Memcheck, a memory error detector
==14849== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==14849== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==14849== Command: ./sorting_program merge
==14849== Parent PID: 10496
==14849==
==14849==
==14849== HEAP SUMMARY:
==14849==     in use at exit: 100,000 bytes in 1 blocks
==14849==   total heap usage: 2 allocs, 1 frees, 101,024 bytes allocated
==14849==
==14849== 100,000 bytes in 1 blocks are definitely lost in loss record 1 of 1
==14849==    at 0x4C2FB55: calloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==14849==    by 0x40085C: main (sorting.c:42)
==14849==
==14849== LEAK SUMMARY:
==14849==    definitely lost: 100,000 bytes in 1 blocks
==14849==    indirectly lost: 0 bytes in 0 blocks
==14849==      possibly lost: 0 bytes in 0 blocks
==14849==    still reachable: 0 bytes in 0 blocks
==14849==         suppressed: 0 bytes in 0 blocks
==14849==
==14849== For counts of detected and suppressed errors, rerun with: -v
==14849== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

(9) How many bytes were leaked in the buggy program?
100,000 bytes
(10) Show the Valgrind output file for the merge sort after you fixed the intentional leak.

```
==15114== Memcheck, a memory error detector
==15114== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==15114== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==15114== Command: ./sorting_program merge
==15114== Parent PID: 10496
==15114==
==15114==
==15114== HEAP SUMMARY:
==15114==     in use at exit: 0 bytes in 0 blocks
==15114==   total heap usage: 2 allocs, 2 frees, 101,024 bytes allocated
==15114==
==15114== All heap blocks were freed -- no leaks are possible
==15114==
==15114== For counts of detected and suppressed errors, rerun with: -v
==15114== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

(11) Show the Valgrind output file for your tree sort.

```
==29521== Memcheck, a memory error detector
==29521== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==29521== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==29521== Command: ./sorting_program tree
==29521== Parent PID: 16580
==29521==
==29521==
==29521== HEAP SUMMARY:
==29521==     in use at exit: 0 bytes in 0 blocks
==29521==   total heap usage: 25,001 allocs, 25,001 frees, 601,004 bytes allocated
==29521==
==29521== All heap blocks were freed -- no leaks are possible
==29521==
==29521== For counts of detected and suppressed errors, rerun with: -v
==29521== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

(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.

```
command: unix.> time ./amplify IMAGES/Lenna_org_1024.pgm 11 1.1 2
real	0m0.761s
user	0m0.708s
sys	0m0.052s
```

(13) Research and answer: why does real time != (user time + system time)?
Although in this case they did add up it would not surprise me if they did not as sys time does not count ticks while waiting for disk access which in this case would be needed to access the photo while the real time does.

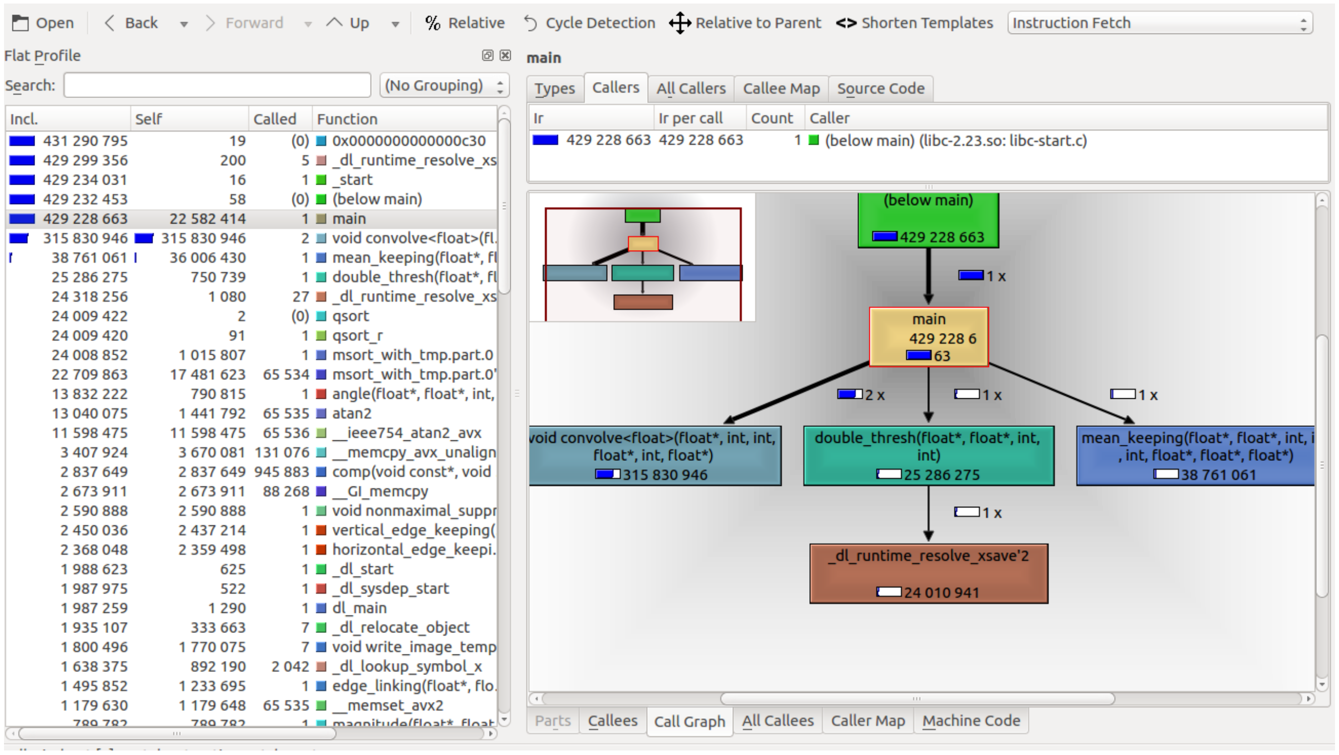
(14) In your report, put the output image for the Lenna image titled output<somenumber>.pgm.
included in a separate pdf, it caused major screen tearing for me.

(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.

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

IMAGES/cameraman.pgm 11 1.1 2
void convolve
mean_keeping
double_thresh

(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.



Incl.	Self	Called	Function
431 290 795	19	(0)	0x0000000000000c30
429 299 356	200	5	_dl_runtime_resolve_xs
429 234 031	16	1	_start
429 232 453	58	(0)	(below main)
429 228 663	22 582 414	1	main
315 830 946	315 830 946	2	void convolve<float>(fl.
38 761 061	36 006 430	1	mean_keeping(float*, fl
25 286 275	750 739	1	double_thresh(float*, fl
24 318 256	1 080	27	_dl_runtime_resolve_xs
24 009 422	2	(0)	qsort
24 009 420	91	1	qsort_r
24 008 852	1 015 807	1	msort_with_tmp.part.0
22 709 863	17 481 623	65 534	msort_with_tmp.part.0'
13 832 222	790 815	1	angle(float*, float*, int,
13 040 075	1 441 792	65 535	atan2
11 598 475	11 598 475	65 536	__ieee754_atan2_avx
3 407 924	3 670 081	131 076	__memcpy_avx_unalign
2 837 649	2 837 649	945 883	comp(void const*, void
2 673 911	2 673 911	88 268	__GI_memcpy
2 590 888	2 590 888	1	void nonmaximal_suppr
2 450 036	2 437 214	1	vertical_edge_keeping(
2 368 048	2 359 498	1	horizontal_edge_keepi.
1 988 623	625	1	_dl_start
1 987 975	522	1	_dl_sysdep_start
1 987 259	1 290	1	dl_main
1 935 107	333 663	7	_dl_relocate_object
1 800 496	1 770 075	7	void write_image_temp
1 638 375	892 190	2 042	_dl_lookup_symbol_x
1 495 852	1 233 695	1	edge_linking(float*, flo.
1 179 630	1 179 648	65 535	__memset_avx2
789 782	789 782	1	magnitude(float*, float

Ir	Ir per call	Count	Caller
429 228 663	429 228 663	1	(below main) (libc-2.23.so: libc-start.c)

(below main)
429 228 663
1 x
main
429 228 6
63
2 x
1 x
1 x
void convolve<float>(float*, int, int, float*, int, float*)
315 830 946
double_thresh(float*, float*, int, int)
25 286 275
mean_keeping(float*, float*, int, i, int, float*, float*)
38 761 061
1 x
_dl_runtime_resolve_xsave'2
24 010 941

(17) Which function dominates the execution time? What fraction of the total execution time does this function occupy?
void convolve, which is responsible for about 65%
(18) Does this program show any signs of memory leaks? Optional: Remove as many memory leaks as possible for 5 extra credits. Document the specific leak(s) you found and the specific code change(s) you made in your lab report.
Yes, valgrind reported 3,146,748 bytes of memory lost
Fixes:
i	free(Gx_mask);

```
    free(Gy_mask);
    free(HRV);
    free(HRH);
    free(Vmap);
    free(Hmap);
    free(org_img);
```

n Main.c:

in Amplify.h:

```
free(patchaxa);
```

```
  free(patch3x3);
```

in Main.h in function double thresh

```
free(supp);
```

in read_image_template :

```
free(supp);
```

that’s all memory leaks
new Valgrind:

```
==12897== Memcheck, a memory error detector
==12897== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==12897== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==12897== Command: ./amplify IMAGES/cameraman.pgm 11 1.1 2
==12897== Parent PID: 16580
==12897==
==12897== Invalid read of size 4
==12897==    at 0x404480: vertical_edge_keeping(float*, float*, float*, int, int, int, float*, float*)
(amplify.h:363)
==12897==    by 0x401590: main (main.c:120)
==12897==  Address 0x5fe5080 is 0 bytes after a block of size 262,144 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x401359: main (main.c:77)
==12897==
==12897== Invalid read of size 4
==12897==    at 0x404872: vertical_edge_keeping(float*, float*, float*, int, int, int, float*, float*)
(amplify.h:366)
==12897==    by 0x401590: main (main.c:120)
==12897==  Address 0x5fe5084 is 4 bytes after a block of size 262,144 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x401359: main (main.c:77)
==12897==
==12897== Invalid read of size 4
==12897==    at 0x405331: horizontal_edge_keeping(float*, float*, float*, int, int, int, float*, float*)
(amplify.h:186)
==12897==    by 0x4015CD: main (main.c:123)
==12897==  Address 0x5fe5080 is 0 bytes after a block of size 262,144 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x401359: main (main.c:77)
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x40172F: main (main.c:138)
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x401738: main (main.c:138)
```

```
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x401CE2: main (main.c:144)
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x401CE7: main (main.c:144)
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x401D12: main (main.c:145)
==12897==
==12897== Conditional jump or move depends on uninitialised value(s)
==12897==    at 0x401D2A: main (main.c:145)
==12897==
==12897== Syscall param write(buf) points to uninitialised byte(s)
==12897==    at 0x57D23C0: __write_nocancel (syscall-template.S:84)
==12897==    by 0x5753C0E: _IO_file_write@@GLIBC_2.2.5 (fileops.c:1263)
==12897==    by 0x5755418: new_do_write (fileops.c:518)
==12897==    by 0x5755418: _IO_do_write@@GLIBC_2.2.5 (fileops.c:494)
==12897==    by 0x575448C: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1331)
==12897==    by 0x57497CA: fwrite (iofwrite.c:39)
==12897==    by 0x406153: write_image (image_template.h:127)
==12897==    by 0x406153: void write_image_template<float>(char*, float*, int, int)
(image_template.h:93)
==12897==    by 0x401B08: main (main.c:216)
==12897==  Address 0x5e8b6cf is 15 bytes inside a block of size 4,096 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x57481E4: _IO_file_doallocate (filedoalloc.c:127)
==12897==    by 0x57565A3: _IO_doallocbuf (genops.c:398)
==12897==    by 0x5755907: _IO_file_overflow@@GLIBC_2.2.5 (fileops.c:820)
==12897==    by 0x575429C: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1331)
==12897==    by 0x5728250: vfprintf (vfprintf.c:1320)
==12897==    by 0x57F1CC8: __fprintf_chk (fprintf_chk.c:35)
==12897==    by 0x406140: fprintf (stdio2.h:98)
==12897==    by 0x406140: write_image (image_template.h:126)
==12897==    by 0x406140: void write_image_template<float>(char*, float*, int, int)
(image_template.h:93)
==12897==    by 0x401B08: main (main.c:216)
==12897==
==12897== Syscall param write(buf) points to uninitialised byte(s)
==12897==    at 0x57D23C0: __write_nocancel (syscall-template.S:84)
==12897==    by 0x5753C0E: _IO_file_write@@GLIBC_2.2.5 (fileops.c:1263)
==12897==    by 0x5754399: new_do_write (fileops.c:518)
==12897==    by 0x5754399: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1342)
==12897==    by 0x57497CA: fwrite (iofwrite.c:39)
==12897==    by 0x406153: write_image (image_template.h:127)
==12897==    by 0x406153: void write_image_template<float>(char*, float*, int, int)
(image_template.h:93)
```

```
==12897==    by 0x401B08: main (main.c:216)
==12897==  Address 0x60e61b1 is 4,081 bytes inside a block of size 262,144 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x405DDF: void write_image_template<float>(char*, float*, int, int)
(image_template.h:88)
==12897==    by 0x401B08: main (main.c:216)
==12897==
==12897== Syscall param write(buf) points to uninitialised byte(s)
==12897==    at 0x57D23C0: __write_nocancel (syscall-template.S:84)
==12897==    by 0x5753C0E: _IO_file_write@@GLIBC_2.2.5 (fileops.c:1263)
==12897==    by 0x5755418: new_do_write (fileops.c:518)
==12897==    by 0x5755418: _IO_do_write@@GLIBC_2.2.5 (fileops.c:494)
==12897==    by 0x57549BF: _IO_file_close_it@@GLIBC_2.2.5 (fileops.c:165)
==12897==    by 0x57483FE: fclose@@GLIBC_2.2.5 (iofclose.c:58)
==12897==    by 0x40615B: write_image (image_template.h:129)
==12897==    by 0x40615B: void write_image_template<float>(char*, float*, int, int)
(image_template.h:93)
==12897==    by 0x401B08: main (main.c:216)
==12897==  Address 0x5e8b6ce is 14 bytes inside a block of size 4,096 alloc'd
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x57481E4: _IO_file_doallocate (filedoalloc.c:127)
==12897==    by 0x57565A3: _IO_doallocbuf (genops.c:398)
==12897==    by 0x5755907: _IO_file_overflow@@GLIBC_2.2.5 (fileops.c:820)
==12897==    by 0x575429C: _IO_file_xsputn@@GLIBC_2.2.5 (fileops.c:1331)
==12897==    by 0x5728250: vfprintf (vfprintf.c:1320)
==12897==    by 0x57F1CC8: __fprintf_chk (fprintf_chk.c:35)
==12897==    by 0x406140: fprintf (stdio2.h:98)
==12897==    by 0x406140: write_image (image_template.h:126)
==12897==    by 0x406140: void write_image_template<float>(char*, float*, int, int)
(image_template.h:93)
==12897==    by 0x401B08: main (main.c:216)
==12897==
==12897==
==12897== HEAP SUMMARY:
==12897==     in use at exit: 72,704 bytes in 1 blocks
==12897==   total heap usage: 81 allocs, 80 frees, 7,525,676 bytes allocated
==12897==
==12897== 72,704 bytes in 1 blocks are still reachable in loss record 1 of 1
==12897==    at 0x4C2DB8F: malloc (in /usr/lib/valgrind/vgpreload_memcheck-amd64-linux.so)
==12897==    by 0x4EC3EFF: ??? (in /usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.21)
==12897==    by 0x40106F9: call_init.part.0 (dl-init.c:72)
==12897==    by 0x401080A: call_init (dl-init.c:30)
==12897==    by 0x401080A: _dl_init (dl-init.c:120)
==12897==    by 0x4000C69: ??? (in /lib/x86_64-linux-gnu/ld-2.23.so)
==12897==    by 0x4: ???
==12897==    by 0xFFF0001A2: ???
==12897==    by 0xFFF0001AC: ???
```

```
==12897==    by 0xFFF0001C1: ???
==12897==    by 0xFFF0001C4: ???
==12897==    by 0xFFF0001C8: ???
==12897==
==12897== LEAK SUMMARY:
==12897==    definitely lost: 0 bytes in 0 blocks
==12897==    indirectly lost: 0 bytes in 0 blocks
==12897==      possibly lost: 0 bytes in 0 blocks
==12897==    still reachable: 72,704 bytes in 1 blocks
==12897==         suppressed: 0 bytes in 0 blocks
==12897==
==12897== For counts of detected and suppressed errors, rerun with: -v
==12897== Use --track-origins=yes to see where uninitialised values come from
==12897== ERROR SUMMARY: 1390511 errors from 12 contexts (suppressed: 0 from 0)
```