CTest & CDash add support for new dynamic analysis tools

Zack Galbreath and Bill Hoffman on October 28, 2014 Tags: CMake , Software Process

CTest and CDash now support the new suite of "sanitizer" dynamic analysis tools that are available for gcc and clang! Support for these tools will be included in the upcoming release of CMake 3.1. They can be found in the first release candidate CMake 3.1-rc1, or you can checkout CMake master from git and build it from source.

These new features were added as part of the Google Project Tango effort to create mobile 3D scanning devices. The sanitizer tools allow for low overhead run time error checking similar to what you can find in valgrind. However, instead of running a machine code level simulator to detect the errors, the compiler based sanitizers add extra error checks that are compiled into the code. This provides better performance. There is one price that is paid for that performance gain. Once the first error is detected, the program halts.

This blog will provide examples of how to use CTest and CDash to run sanitized test programs and collect and display the results. In order to make it easier for you to replicate the examples below, all of the sample code is available as an attachment to this blog post.

Use of the sanitizer tools requires building your code with extra compile flags that inject the error checking into the final executable. There are several ways to do that. You can set CMAKE_C_FLAGS and CMAKE_CXX_FLAGS on the command line of cmake or by preloading the cache with values in a ctest script.

ThreadSanitizer

ThreadSanitizer (TSan) allows us to detect data races in our programs.

Example code

Here's a simple example of a program with a data race:

```
#include <pthread.h>
#include <stdio.h>

int Global;

void *Thread1(void *x) {
  Global++;
  return NULL;
}

void *Thread2(void *x) {
```

SEPTEMBER, 2017
SORT OPTIONS

10 - 14 SEP MICCAI 2017
THE 20TH INTERNATIONAL CONFERENCE ON MEDICAL IMAGE COMPUTING AND COMPUTER ASSISTED INTERVENTION

12 - 14 SEP BROCON '17

13 SEP JOURNÉE INDUSTRIELLE 2017

14 SEP POINT-OF-CARE ULTRASOUND: ALGORITHMS, HARDWARE, AND APPLICATIONS
A MICCAI 2017 WORKSHOP

17 - 20 SEP 2017 IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING
ICIP 2017

21 - 22 SEP BIG DATA & PREDICTIVE ANALYTICS FOR INTELLIGENCE AND DEFENSE

23 - 24 SEP ALL THINGS OPEN 2017
A CONFERENCE EXPLORING OPEN SOURCE, OPEN TECH, AND THE OPEN WEB IN THE ENTERPRISE

Filter by Platform
3D Slicer, CDash, CMake, CMB, ITK, KWIVER, Open Chemistry, ParaView, Resonant, Tomviz, VTK, see all...

Filter by Topic
Kitware News, Data & Analytics, Computer Vision, Medical Imaging, Scientific Visualization, Software Process

Recent Comments
Ken Martin on Taking ParaView into Virtual Reality
Matthew Russell on Color legend improvements coming to ParaView 5.4
Cory Quammen on Color legend improvements coming to ParaView 5.4
Matthew Russell on Color legend improvements coming to

```
  Global--;
  return NULL;
}

int main() {
  pthread_t t[2];
  pthread_create(&t[0], NULL, Thread1, NULL);
  pthread_create(&t[1], NULL, Thread2, NULL);
  pthread_join(t[0], NULL);
  pthread_join(t[1], NULL);
}
```

Building with TSan

All we need to do to test this program with ThreadSanitizer is add some flags to the C and CXX flags used to build the project. The flags required are:

```
-g -O1 -fsanitize=thread -fno-omit-frame-pointer -fPIC
```

Example Output

After we build this executable and run it, we see the following output from TSan:

```
==================
WARNING: ThreadSanitizer: data race (pid=5618)
  Write of size 4 at 0x7f47f6025c3c by thread T2:
    #0 Thread2(void*) simple_race.cc:12
(simple_race+0x0000000a8883)

  Previous write of size 4 at 0x7f47f6025c3c by thread T1:
    #0 Thread1(void*) simple_race.cc:7
(simple_race+0x0000000a8843)

  Location is global 'Global' of size 4 at 0x7f47f6025c3c
(simple_race+0x0000016f0c3c)

  Thread T2 (tid=5622, running) created by main thread at:
    #0 pthread_create <null>:0
(simple_race+0x00000004de4b)
    #1 main simple_race.cc:19 (simple_race+0x0000000a88e1)

  Thread T1 (tid=5621, finished) created by main thread
at:
    #0 pthread_create <null>:0
(simple_race+0x00000004de4b)
    #1 main simple_race.cc:18 (simple_race+0x0000000a88ca)
```

ParaView 5.4

John on VTK Textbook and User's Guide now available for download

Archive by Month

Select Month

Archive by Author (alphabetical)

Select Author

Archive by Author (post count)

Select Author

Subscribe to the Kitware Blog

Email Address

Subscribe

Site Tools

Log in

Entries RSS

Comments RSS

WordPress.org

```
SUMMARY: ThreadSanitizer: data race simple_race.cc:12
Thread2(void*)
==================
ThreadSanitizer: reported 1 warnings
```

Submitting to CDash

Here's a CTest script that you can use to run this example and submit the results to CDash's public dashboard.

```
set(CTEST_PROJECT_NAME "tsan_example")
set(CTEST_SITE "localhost")
set(CTEST_BUILD_NAME "ThreadSanitizer")

set(CTEST_SOURCE_DIRECTORY "${CTEST_SCRIPT_DIRECTORY}")
set(CTEST_BINARY_DIRECTORY
"${CTEST_SCRIPT_DIRECTORY}/bin")

set(CTEST_CMAKE_GENERATOR "Unix Makefiles")
set(CTEST_MEMORYCHECK_TYPE "ThreadSanitizer")
ctest_start(Experimental)
file(WRITE "${CTEST_SCRIPT_DIRECTORY}/bin/CMakeCache.txt"
"
CMAKE_CXX_FLAGS=-g -O1 -fsanitize=thread -fno-omit-frame-
pointer -fPIC
")
ctest_configure()
ctest_build()
ctest_test()
ctest_memcheck()

set(CTEST_DROP_METHOD "http")
set(CTEST_DROP_SITE "open.cdash.org")
set(CTEST_DROP_LOCATION
"/submit.php?project=PublicDashboard")
ctest_submit()
```

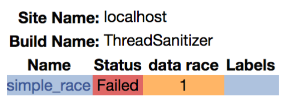
Other than the compile flags mentioned earlier, of particular note here is the line

```
set(CTEST_MEMORYCHECK_TYPE "ThreadSanitizer")
```

This is what tells CTest how to intrepret the output of the dynamic analysis (memcheck) step.

After you've submitted your results to CDash, you should see a new row in the Dynamic Analysis section of the page. Here's an example of what this will look like:

When you click on the "1" (under Defect Count) you'll be taken to a page displaying the following information:



Site Name: localhost
Build Name: ThreadSanitizer

Name	Status	data race	Labels
simple_race	Failed	1	

From here, if you click on the name of the test (simple_race) you will see the output of the sanitizer tool.

AddressSanitizer

AddressSanitizer (asan) allows us to detect buffer overflows and cases where memory is read after being freed.

Example code

Here's an example C program that attempts to read some memory after freeing it:

```
#include <stdlib.h>
int main() {
  char *x = (char*)malloc(10 * sizeof(char*));
  free(x);
  return x[5];
}
```

Building with ASan

Similarly to the TSan example, we enable ASan dynamic analysis by modifying the flags that we use to compile our program:

```
-DCMAKE_C_FLAGS="-g -O1 -fsanitize=address -fno-omit-
frame-pointer"
```

Example output

Here is an example of the output you should expect when ASan detects a defect in your program:

```
==========================================================
=======
==6140==ERROR: AddressSanitizer: heap-use-after-free on
address 0x60700000dfb5 at pc 0x4820cb bp 0x7fffa1ac1130 sp
0x7fffa1ac1128
READ of size 1 at 0x60700000dfb5 thread T0
    #0 0x4820ca in main use-after-free.c:5
    #1 0x7fb5d55b776c in __libc_start_main /build/buildd
/eglibc-2.15/csu/libc-start.c:226
    #2 0x481f9c in _start (bin/use-after-free+0x481f9c)

0x60700000dfb5 is located 5 bytes inside of 80-byte region
[0x60700000dfb0,0x60700000e000)
freed by thread T0 here:
    #0 0x46bd39 in free (bin/use-after-free+0x46bd39)
    #1 0x48209a in main use-after-free.c:4
    #2 0x7fb5d55b776c in __libc_start_main /build/buildd
/eglibc-2.15/csu/libc-start.c:226

previously allocated by thread T0 here:
    #0 0x46beb9 in __interceptor_malloc (bin/use-after-
free+0x46beb9)
    #1 0x48208f in main use-after-free.c:3
    #2 0x7fb5d55b776c in __libc_start_main /build/buildd
/eglibc-2.15/csu/libc-start.c:226

SUMMARY: AddressSanitizer: heap-use-after-free use-after-
free.c:5 main
```

Submitting to CDash

To submit these results to CDash, you can use a script very similar to the one above in the TSan example. The only important difference is that instead of:

```
set(CTEST_MEMORYCHECK_TYPE "ThreadSanitizer")
```

you should specify:

```
set(CTEST_MEMORYCHECK_TYPE "AddressSanitizer")
```

and the cache should will be:

```
file(WRITE "${CTEST_SCRIPT_DIRECTORY}/bin/CMakeCache.txt"
"
CMAKE_C_FLAGS=-g -O1 -fsanitize=address -fno-omit-frame-
pointer
")
```

Here are a couple views of these results from CDash:

MemorySanitizer

MemorySanitizer (msan) allows us to detect reads of uninitialized memory.

Example code

In this example, our program reads from memory before it has been initialized:

```
#include <stdio.h>

int main(int argc, char** argv) {
  int* a = new int[10];
  a[5] = 0;
  if (a[argc])
    printf("xx\n");
  return 0;
}
```

Building with ASan

As before, we simply need to modify the compile flags for the executable that we wish to test. In this case, we specify -fsanitize=memory instead of =thread or =address.

```
-DCMAKE_CXX_FLAGS="-g -O1 -fsanitize=memory -fno-omit-
frame-pointer"
```

Example output

Here is the output you should expect if you run the example above through MSan:

```
==6805== WARNING: MemorySanitizer: use-of-uninitialized-
value
    #0 0x7ff16024fcb6 in main umr.cc:6
    #1 0x7ff15edd276c in __libc_start_main /build/buildd
/eglibc-2.15/csu/libc-start.c:226
    #2 0x7ff16024fadc in _start (bin/umr+0x7badc)

SUMMARY: MemorySanitizer: use-of-uninitialized-value
umr.cc:6 main
```

Submitting to CDash

You can follow the pattern outlined above for TSan. The important change is to specify

```
set(CTEST_MEMORYCHECK_TYPE "MemorySanitizer")
```

instead of:

```
set(CTEST_MEMORYCHECK_TYPE "ThreadSanitizer")
```

and the cache should have:

```
file(WRITE "${CTEST_SCRIPT_DIRECTORY}/bin/CMakeCache.txt"
"
CMAKE_CXX_FLAGS=-g -O1 -fsanitize=memory -fno-omit-frame-
pointer
")
```

Finally, here's what these results look like on CDash:

That's all there is to it! I hope that you found these examples useful. With the use of these new tools, you can be more confident that there aren't any nasty memory defects lurking in your codebase. If you have any questions or comments about these new features of CTest & CDash, please contact us on the CMake users mailing list.

Post Views: 5

Share this:

Related Posts

CMake 3.9.2 available for download

CMake 3.9.1 available for download

CMake and the Default Build Type

7 Responses to *CTest & CDash add support for new dynamic analysis tools*

Kannengieser *says:*
June 20, 2016 at 4:05 pm

It does not work!! It just do not compile any thing. Any real example on github????

Reply

Bill Hoffman *says:*
June 20, 2016 at 4:20 pm

What does not work? Here is the nightly script that runs asan on CMake: https://open.cdash.org/viewNotes.php?buildid=4419935. All of this is now in official CMake releases, so you don't need to build CMake.

Reply

Xavier *says:*
November 28, 2016 at 5:56 pm

In the CTest script, you call ctest_test() and ctest_memcheck(). Does it mean that the tests are run twice?

Reply

Bill Hoffman *says:*
November 28, 2016 at 8:57 pm

Yes, it will call it once with memcheck and one without. Although now that you mention it this does not make as much sense with compiled in memory checking. If you leave out the ctest_test the test column will be empty on CDash, but it should work fine.

Reply

Xavier *says:*
November 28, 2016 at 9:10 pm

Yes, that was my point.

So, there is no way to run the tests only once and populate both the test columns and the dynamics analysis group on CDash?

Reply

Bill Hoffman *says:*
November 28, 2016 at 9:19 pm

Unfortunately No, they are treated differently by ctest.

Reply

Xavier *says:*
November 28, 2016 at 9:53 pm

Thanks for the clarification!

Reply

Questions or comments are always welcome!

Enter your comment here...