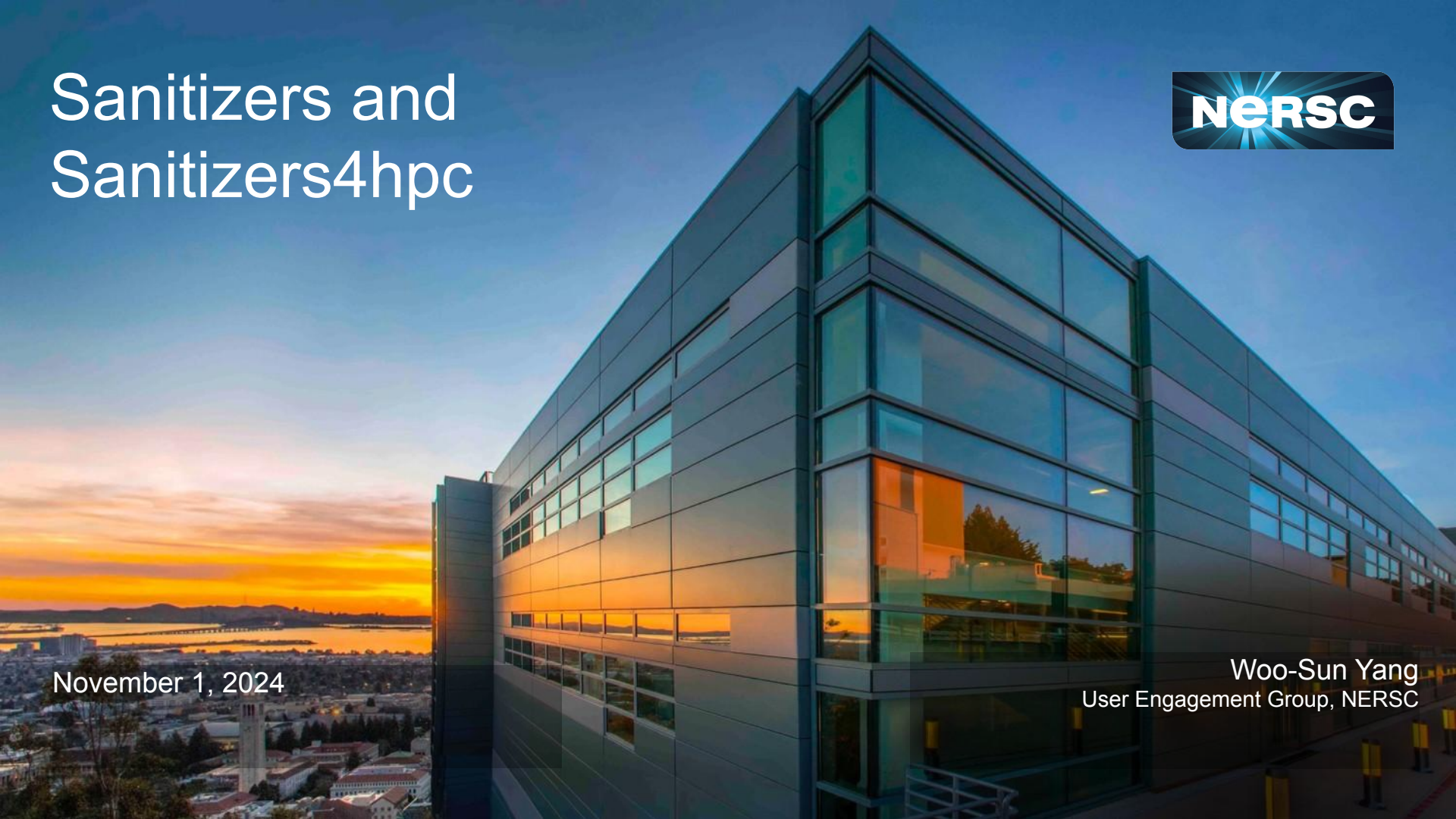
Sanitizers and Sanitizers4hpc



November 1, 2024

Woo-Sun Yang
User Engagement Group, NERSC

Introduction

- LLVM Sanitizers are a suite of debugging tools for detecting various kinds of bugs
  - AddressSanitizer, LeakSanitizer, ThreadSanitizer, MemorySanitizer, ...
  - C and C++ codes
- Documentations
  - https://docs.nersc.gov/tools/debug/sanitizers/
  - https://clang.llvm.org/docs/index.html
  - https://github.com/google/sanitizers

Supported Compilers

- Sanitizers can be used with more than just LLVM compilers
  - Compatible with all compilers on Perlmutter, except the Nvidia compiler
  - Accept many LLVM sanitizer compiler flags
- For MPI codes, use Cray compiler wrappers (cc and CC)
- For non-MPI codes, use the following base compilers

GNU	Cray	Intel	AOCC	LLVM
gcc/g++	craycc/craycxx	icx/icpx	clang/clang++	clang/clang++

  - Intel's icc and icpc don't work for Sanitizers

Build for Sanitizers

- Build an executable instrumented for the desired Sanitizer

```
<compiler_command> -sanitizer=<sanitizer_name> \
  <other compile or sanitizer options> ...
```

- Typically, executables run several times slower and use several times more memory
- Make sure to rebuild your code without the sanitizer flags after debugging

AddressSanitizer (ASAN): Detecting memory errors

- Memory errors detected include:
  - **Use after free**: dangling pointer dereference
  - **Out of bounds array accesses** to heap, stack and globals
  - **Use after return**: use of a stack object after returning from the function where the object is defined
  - **Use after scope**: use of a stack object outside the scope
  - **Initialization order bug**: non-deterministic outcome due to unspecified order in which constructors for global objects in different source files run
  - **Double-free**, **invalid free**
  - **Memory leaks**

AddressSanitizer (ASAN): Shadow bytes

- ASAN uses an extra memory block (“shadow bytes”) for tracking state of an allocated memory block
- One shadow byte represents 8 application bytes
  - Mapping for x86_64: Shadow = (Mem >> 3) + 0x7fff8000
- Shadow bytes are marked as:
  - **`00`**: 8 bytes are all addressable
  - **`01`**, **`02`**, …, **`07`**: partially addressable (out of the 8 bytes)
  - **`fa`**: heap left redzone - not to be disturbed
  - **`fd`**: freed heap region
  - …
- ASAN uses these for detecting illegal memory accesses
  - Example in next slides

AddressSanitizer (ASAN), illegal memory access example 1

- The example code makes an out-of-bound array reference - `array[10]` when there are 10 elements

```
$ cat -n illegalmemoryaccess.cpp
...
 4  int *array = new int[10];
 5
 6   for (int i=0; i < 11; ++i)
 7       array[i] = i + 1;
...
```

- Compile with `-fsanitize=address`

```
$ g++ -O0 -g -fsanitize=address
illegalmemoryaccess.cpp
```

AddressSanitizer (ASAN), illegal memory access example 2

Error detected:
- Heap-buffer-overflow for attempting to write 4 bytes
- Line 7 of `illegalmemoryaccess.cpp`

Memory block in question
- 40 bytes allocated at line 4

Shadow byte analysis
- The block mapped to [0xc087fff8002,0xc087fff8007)
- Memory block of 5 * 8 bytes = 40 bytes
- `[fa]`: An attempt was made at line 7 to write to the next shadow byte, marked here as ‘`fa`’

```
$ ./a.out
=================================================================
==2267569==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x604000000038 at pc
0x0000004009df bp 0x7ffe9e373680 sp 0x7ffe9e373678
WRITE of size 4 at 0x604000000038 thread T0
    #0 0x4009de in main /pscratch/sd/e/elvis/addresssanitizer/illegalmemoryaccess.cpp:7
    #1 0x7fbf17c3c24c in __libc_start_main (/lib64/libc.so.6+0x3524c)
    #2 0x4008b9 in _start ../sysdeps/x86_64/start.S:120

0x604000000038 is located 0 bytes to the right of 40-byte region [0x604000000010,0x604000000038)
allocated by thread T0 here:
    #0 0x7fbf188bba88 in operator new[](unsigned long) (/usr/lib64/libasan.so.8+0xbba88)
    #1 0x40097e in main /pscratch/sd/e/elvis/addresssanitizer/illegalmemoryaccess.cpp:4
    #2 0x7fbf17c3c24c in __libc_start_main (/lib64/libc.so.6+0x3524c)

SUMMARY: AddressSanitizer: heap-buffer-overflow
/pscratch/sd/e/elvis/addresssanitizer/illegalmemoryaccess.cpp:7 in main
Shadow bytes around the buggy address:
  0x0c087fff7fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c087fff7ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c087fff8000: fa fa 00 00 00 00 00[fa]fa fa fa fa fa fa fa fa
  0x0c087fff8010: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8020: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8030: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8040: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
  0x0c087fff8050: fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa fa
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  ...
  Right alloca redzone:    cb
==2267569==ABORTING
```

LeakSanitizer (LSAN): Detecting memory leaks

- In the code pointer p is set to NULL without freeing the previously allocated block

```
$ cat -n memory-leak.c
 1  #include <stdlib.h>
 2  void *p;
 3  int main() {
 4    p = malloc(7);
 5    p = 0; // The memory is leaked here.
...
```

- Compile

```
$ clang -O0 -g -fsanitize=leak
memory-leak.c
```

LeakSanitizer: Detecting memory leaks (cont'd)

- LSAN detects a memory leak of 7 bytes which were allocated at line 4

```
$ ./a.out
=================================================================
==2335900==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 7 byte(s) in 1 object(s) allocated from:
    #0 0x55966653a842 in malloc
/.../nersc/nersc-user-env/prgenv/llvm_src_17.0.6/compiler-rt/lib/lsan/lsan_interceptors.cpp:75:3
    #1 0x559666565898 in main /pscratch/sd/e/elvis/addresssanitizer/memory-leak.c:4:7
    #2 0x7efe8f83e24c in __libc_start_main (/lib64/libc.so.6+0x3524c) (BuildId:
ddc393ac74ed8f90d4fdfff796432fbafd281e1b)

SUMMARY: LeakSanitizer: 7 byte(s) leaked in 1 allocation(s)
```

LeakSanitizer: Detecting memory leaks (cont'd)

- Can be combined with AddressSanitizer to get both memory error and leak detection, too:

```
$ clang -fsanitizer=address -g memory-leak.c
$ ASAN_OPTIONS=detect_leaks=1 ./a.out
=================================================================
==2339511==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 7 byte(s) in 1 object(s) allocated from:
    #0 0x56040740afde in malloc
/.../nersc/nersc-user-env/prgenv/llvm_src_17.0.6/compiler-rt/lib/asan/asan_malloc_linux.cp
p:69:3
    #1 0x560407447a68 in main /pscratch/sd/e/elvis/addresssanitizer/memory-leak.c:4:7
    #2 0x7fdab443e24c in __libc_start_main (/lib64/libc.so.6+0x3524c) (BuildId:
ddc393ac74ed8f90d4fdfff796432fbafd281e1b)

SUMMARY: AddressSanitizer: 7 byte(s) leaked in 1 allocation(s)
```

MemorySanitizer (MSAN): Detecting uninitialized variables

- Warns if a uninitialized variable is read
- Compile
  - Use `-fsanitize=memory`
  - For better performance, add -O1 or higher
- The GNU compilers don't support MSAN
- With `PrgEnv-cray` and `PrgEnv-intel`, the source line info in error messages is not displayed
  - Workaround:

    ```
    export MSAN_OPTIONS="allow_addr2line=true"
    ```

  - Fixed in CCE/18.0.0 for `PrgEnv-cray`; `PrgEnv-intel` problem won't be fixed due to limited support

MemorySanitizer (MSAN): example

- In the example code, a[1] is uninitialized but it is used for a comparison

```
$ cat -n umr.cc
...
 3  int main(int argc, char** argv) {
 4    int* a = new int[10];
 5    a[5] = 0;
 6    if (a[argc])
 7      printf("xx\n");
...
```

```
$ CC -fsanitize=memory -g -O1 umr.cc

$ ./a.out
==578284==WARNING: MemorySanitizer:
use-of-uninitialized-value
    #0 0x2cf202 in main
/pscratch/sd/e/elvis/sanitizers/umr.cc:6:7
    #1 0x7fc4fa63e24c in __libc_start_main
(/lib64/libc.so.6+0x3524c)
    #2 0x24e4b9 in _start
/home/abuild/rpmbuild/BUILD/glibc-2.31/csu/../sysdeps/x8
6_64/start.S:120

SUMMARY: MemorySanitizer: use-of-uninitialized-value
/pscratch/sd/e/elvis/sanitizers/umr.cc:6:7 in main
Exiting
```

ThreadSanitizer (TSAN): Detecting thread data races

- Data race: When multiple threads are allowed to access the same memory location simultaneously and one of them is to modify the value, the order of operations among threads (a data race condition) can affect the numerical result
- TSAN detects data races among threads
- Compile
  - With `-fsanitize=thread`
  - For better performance, add `-O1` or higher
- May have to run a few times because of a race condition

ThreadSanitizer (TSAN): example

- The example code has a race condition as it doesn't include the `reduction` clause

```
$ cat -n buggyreduction_omp.c
...
 4   int sum = 0;
 5   #pragma omp parallel for shared(sum)
 6   for (int i=0; i<1000; i++)
 7     sum += i;
...
```

```
$ cc -fsanitize=thread -g -O1 -fopenmp
buggyreduction_omp.c
```

ThreadSanitizer (TSAN): example (cont'd)

sum read by Thread T1

sum written by "main thread"

About Thread T1

```
$ export OMP_NUM_THREADS=8
$ ./a.out
==================
WARNING: ThreadSanitizer: data race (pid=2240264)
  Read of size 4 at 0x7ffdf6e678bc by thread T1:
    #0 main._omp_fn.0 /pscratch/sd/e/elvis/sanitizers/buggyreduction_omp.c:6
(a.out+0x400895)
    #1 <null> <null> (libgomp.so.1+0x1dd4d)

  Previous write of size 4 at 0x7ffdf6e678bc by main thread:
    #0 main._omp_fn.0 /pscratch/sd/e/elvis/sanitizers/buggyreduction_omp.c:7
(a.out+0x4008aa)
    #1 GOMP_parallel <null> (libgomp.so.1+0x14e95)

  Location is stack of main thread.

  Location is global '<null>' at 0x000000000000 ([stack]+0x1e8bc)

  Thread T1 (tid=2240266, running) created by main thread at:
    #0 pthread_create <null> (libtsan.so.2+0x61be6)
    #1 <null> <null> (libgomp.so.1+0x1e38f)

SUMMARY: ThreadSanitizer: data race
/pscratch/sd/e/elvis/sanitizers/buggyreduction_omp.c:6 in main._omp_fn.0
==================
sum = 335625
ThreadSanitizer: reported 1 warnings
```

Sanitizers4hpc

- HPE tool for a MPI code that merges Sanitizer output with the same stack traces
- Present Sanitizer result by group of MPI ranks sharing the same stack traces
- Currently it supports
  - AddressSanitizer
  - LeakSanitizer
  - ThreadSanitizer
  - Nvidia Compute Sanitizer's (formerly CUDA Memcheck) Memcheck
- Aggregations not perfect for certain Sanitizers (TSAN, Compute Sanitizer, …) - fixed in CPE/24.07

Sanitizers4hpc - How to run

- Run an app with the tool

```
$ module load sanitizers4hpc
$ sanitizers4hpc <sanitizers4hpc options> -- ./a.out <app options>
```

- Some options
  - **`-l <launch_args>`**: e.g., `-l "-n 4 -c 64 --cpu-bind=cores"`
  - **`-f`**: Bypass Clang Sanitizers check
  - **`-n`**: Bypass MPI check
  - **`-a <ASAN_options>`**
  - **`-o <LSAN_options>`**
  - **`-t <TSAN_options>`**

Sanitizers4hpc for CPU code example

- The example code is an embarrassingly-parallel MPI version of buggyreduction_omp.c
- The code has a race condition for OpenMP threads

```
$ cat -n buggyreduction_mpiomp.c
...
  7    MPI_Init(&argc, &argv);
  8    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  9
 10    int sum = 0;
 11    #pragma omp parallel for shared(sum)
 12    for (int i=0; i<1000; i++)
 13      sum += i;
...
```

```
$ cc -fsanitize=thread -g -O1 -fopenmp
buggyreduction_mpiomp.c
```

Sanitizers4hpc for CPU code example (cont'd)

- The example code is an (embarrassingly-parallel) MPI version of buggyreduction_omp.c
- The code has a race condition
- Aggregation of output over MPI ranks needs improvement - fixed in version 1.1.3 in CPE/24.11

```
$ salloc -C cpu -N 1 -q debug -t 10 ...
...
$ module load saniters4hpc
$ export OMP_NUM_THREADS=2
$ sanitizers4hpc -l "-n 2" -- ./a.out
0: sum = 499500
1: sum = 499500
RANKS: <1>
ThreadSanitizer: data race
  Read of size 4 at 0x7fff57a3313c by thread T1:
    #0  main._omp_fn.0
/pscratch/sd/e/elvis/sanitizers/buggyreduction_mpiomp.c:12
(a.out+0x400a55)
    #1    (libgomp.so.1+0x1dd4d)
...
RANKS: <0>
ThreadSanitizer: data race
  Read of size 4 at 0x7ffddd5bccfc by thread T1:
    #0  main._omp_fn.0
/pscratch/sd/e/elvis/sanitizers/buggyreduction_mpiomp.c:12
(a.out+0x400a55)
    #1    (libgomp.so.1+0x1dd4d)
...
```

Sanitizers4hpc for GPU code example

- The example code is an embarrassingly-parallel MPI version of `memcheck_demo.cu`, which contains many memory errors (unalignment, out-of-bounds references, a memory leak, …)
- To use Nvidia's Compute Sanitizer underneath

```
$ cat memcheck_demo.cu
...
void launch_memcheck_demo() {
    int *devMem = nullptr;

    std::cout << "Mallocing memory" << std::endl;
    cudaMalloc((void**)&devMem, 1024);

    run_unaligned();
    run_out_of_bounds();
    ...
}
$ cat main.cc
...
    MPI_Init (&argc, &argv);
    ...
    launch_memcheck_demo();
...
$ nvcc -Xcompiler -rdynamic -G -c memcheck_demo.cu
$ CC main.o memcheck_demo.o
```

Sanitizers4hpc for GPU code example (cont'd)

- `-f` flag: to bypass the requirement that the executable is instrumented for a LLVM Sanitizer

- Aggregation of output needs improvement - fixed in CPE/24.07

```
$ salloc -C gpu -N 1 --gpus-per-node=4 -q debug -t 30 ...
...
$ module load sanitizers4hpc
$ sanitizers4hpc -l "-n 4 -c 32 --cpu-bind=cores --gpus-per-task=1
--gpu-bind=none" \
    -m ${CUDA_HOME}/compute-sanitizer/compute-sanitizer -f --
./a.out
RANKS: <2,3>
...
Saved host backtrace up to driver entry point at error
    #0 0x2eae6f in /usr/local/cuda-12.2/compat/libcuda.so.1
    #1 0xd8f0 in
/home/jenkins/src/gtlt/cuda/gtlt_cuda_query.c:325:gtlt_cuda_pointer
_type /opt/cray/pe/lib64/libmpi_gtl_cuda.so.0
...
RANKS: <0-1>
...
Saved host backtrace up to driver entry point at error
    #0 0x2eae6f in /usr/local/cuda-12.2/compat/libcuda.so.1
    #1 0xd8f0 in
/home/jenkins/src/gtlt/cuda/gtlt_cuda_query.c:325:gtlt_cuda_pointer
_type /opt/cray/pe/lib64/libmpi_gtl_cuda.so.0
...
```

Hands-on

- Exercise materials at https://github.com/NERSC/debugging

```
$ git clone https://github.com/NERSC/debugging
$ cd debugging/Sanitizers
```

- Follow the instructions in README.md in each directory
  - `AddressSanitizer/illegalmemoryaccess.cpp`
  - `LeakSanitizer/memory-leak.c`
  - `MemorySanitizer/umr.cc`
  - `ThreadSanitizer/buggyreduction_omp.c`
  - `Sanitizers4hpc/CPU/buggyreduction_mpiomp.c`
  - `Sanitizers4hpc/GPU/{main.cc,memcheck_demo.cu}`

Thank You!