Overcoming Distributed Debugging Challenges in the MPI+OpenMP Programming Model

Lai Wei[2], Ignacio Laguna[1], Dong H. Ahn[1], Matthew P. LeGendre[1], Gregory L. Lee[1]

[1] Lawrence Livermore National Laboratory, [2] Department of Computer Science, Rice University

Summary

Motivation

- Exascale computing will embrace a wide range of programming models.
- Programming models, such as OpenMP, harness the many levels of architectural parallelism (CPUs and devices).
- Debugging tools must help programmers identify errors at the level of the programming model.
- Question: *What are the effective levels for debugging in hybrid programming models such as MPI+OpenMP?*

Contributions

- Our framework reconstructs user-level call stacks for OpenMP threads.
- We share lessons learned from extending STAT – a highly-scalable, lightweight debugging tool for MPI applications – with support for OpenMP threads.
- Our extension to STAT provide an easy-to-understand view of a stack trace to help programmers debug MPI+OpenMP programs at the user code level.

Rebuilding OpenMP Stack Trace in STAT

Challenges

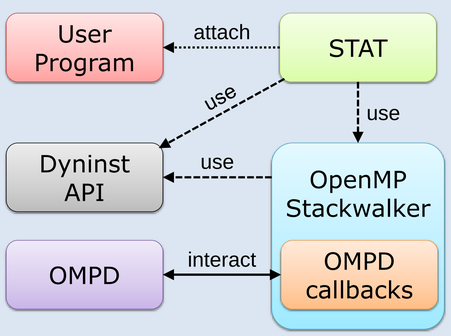
1. Call stacks of OpenMP worker threads provide only partial calling context.
2. Stack frames from routines inside an OpenMP library implementation are confusing to debugger users.

Background on OMPD

OMPD provides debuggers access to OpenMP runtime constructs. Debuggers interact with OMPD to get information, such as a thread's current status and a thread's current parallel region.

Approach

We built an OpenMP stack walker that uses OMPD in conjuction with Dyninst to rebuild call stacks for OpenMP threads. We updated STAT so that it uses the OpenMP stack walker to collect call stacks of threads in MPI+OpenMP programs.



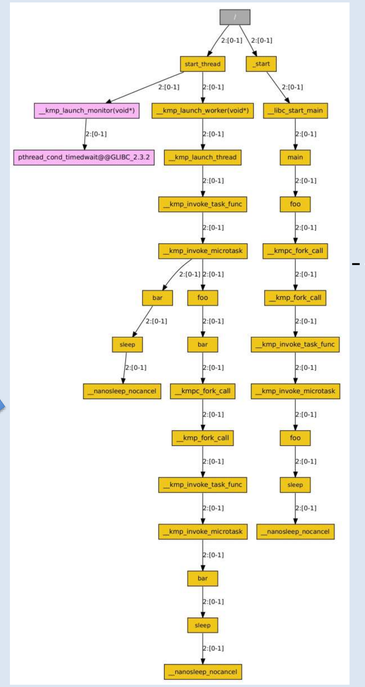
Accomplishments

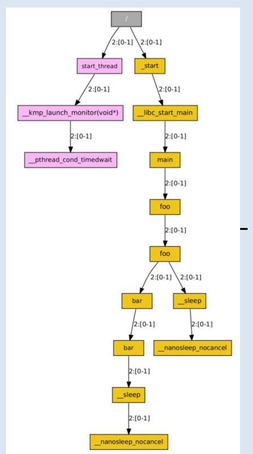
1. Reconstructing full calling context for all OpenMP threads.
2. Eliminating internal frames of OpenMP runtime library.
3. Presenting OpenMP library entry frames upon request.
4. Incorporating these capabilities into STAT to support MPI+OpenMP debugging.

MPI+OpenMP Debuging in STAT

STAT 2D spatial call graph

(2 nodes, 3 OpenMP threads + 1 helper thread per node)





Call stacks of OpenMP threads on node #0

Thread #0
__nanosleep_nocancel
__sleep
foo@example.c:20
__kmp_invoke_microtask
__kmp_invoke_task_func
__kmp_fork_call
__kmpc_fork_call
foo@example.c:17
main@example.c:31
__libc_start_main
_start

Thread #1
__nanosleep_nocancel
__sleep
bar@example.c:11
__kmp_invoke_microtask
__kmp_invoke_task_func
__kmp_fork_call
__kmpc_fork_call
bar@example.c:9
foo@example.c:22
__kmp_invoke_microtask
__kmp_invoke_task_func
__kmp_launch_thread
__kmp_launch_worker
start_thread

Thread #2
__nanosleep_nocancel
__sleep
bar@example.c:11
__kmp_invoke_microtask
__kmp_invoke_task_func
__kmp_launch_thread
__kmp_launch_worker
start_thread

User frames

OpenMP library entry frames (indicates a new parallel construct)

OpenMP library internal frames

Rebuilt call stacks for OpenMP threads on node #0

User frames, mandatory

OpenMP library entry frames, optional

Thread #0
__nanosleep_nocancel
__sleep
foo@example.c:20
foo@example.c:17
main@example.c:31
__libc_start_main
_start

Thread #1 & #2
__nanosleep_nocancel
__sleep
bar@example.c:11
bar@example.c:9
foo@example.c:22
foo@example.c:17
main@example.c:31
__libc_start_main
_start

example.c:

```
 7 void bar()
 8 {
 9   #pragma omp parallel for num_threads(2)
10   for (int i = 0; i < 10; i++)
11     sleep(18);
12 }
13
14 void foo()
15 {
16   omp_set_nested(1);
17   #pragma omp parallel num_threads(2)
18   {
19     if (omp_get_thread_num() == 0)
20       sleep(90);
21     else
22       bar();
23   }
24 }
25
26 int main(int argc, char *argv[])
27 {
28   MPI_Init(&argc, &argv);
29
30   printf("Application: Process %d started.\n", getpid());
31   foo();
32
33   MPI_Finalize();
34
35   return 0;
36 }
```

Conclusions

We leverage OMPD to reconstruct call stacks for OpenMP threads in STAT. This provides better support for debugging MPI+OpenMP programs.

Our results show that we are able to present users with intuitive stack trace views for MPI+OpenMP programs.

Future Work

1. Refine STAT call stack views for MPI+OpenMP programs and evaluate STAT on large applications.
2. Support debugging of OpenMP 4.0 programs, which can include omp target constructs.
3. Present views of OpenMP tasks as well as identify master and worker threads in MPI+OpenMP programs.

References

[1] D. Arnold, D. Ahn, B. Supinski, G. Lee, B. Miller, and M. Schulz, "Stack Trace Analysis for Large Scale Applications," *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, vol., no., pp.1,10, 26-30 March 2007

[2] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. Cownie, R. Dietrich, J. Signore, E. Loh, D. Lorenz, "OMPD: An Application Programming Interface for a Debugger Support Library for OpenMP," *Technical Report, 2014.*

[3] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. Cownie, R. Dietrich, X. Liu, E. Loh, D. Lorenz, "OpenMP Technical Report 2 on the OMPT Interface," *Technical Report, 2014.*

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-675857