Overcoming Distributed Debugging Challenges in the MPI+OpenMP Programming Model

Lai Wei, Ignacio Laguna, Dong H. Ahn

Matthew P. LeGendre, Gregory L. Lee

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-PRES-676381]

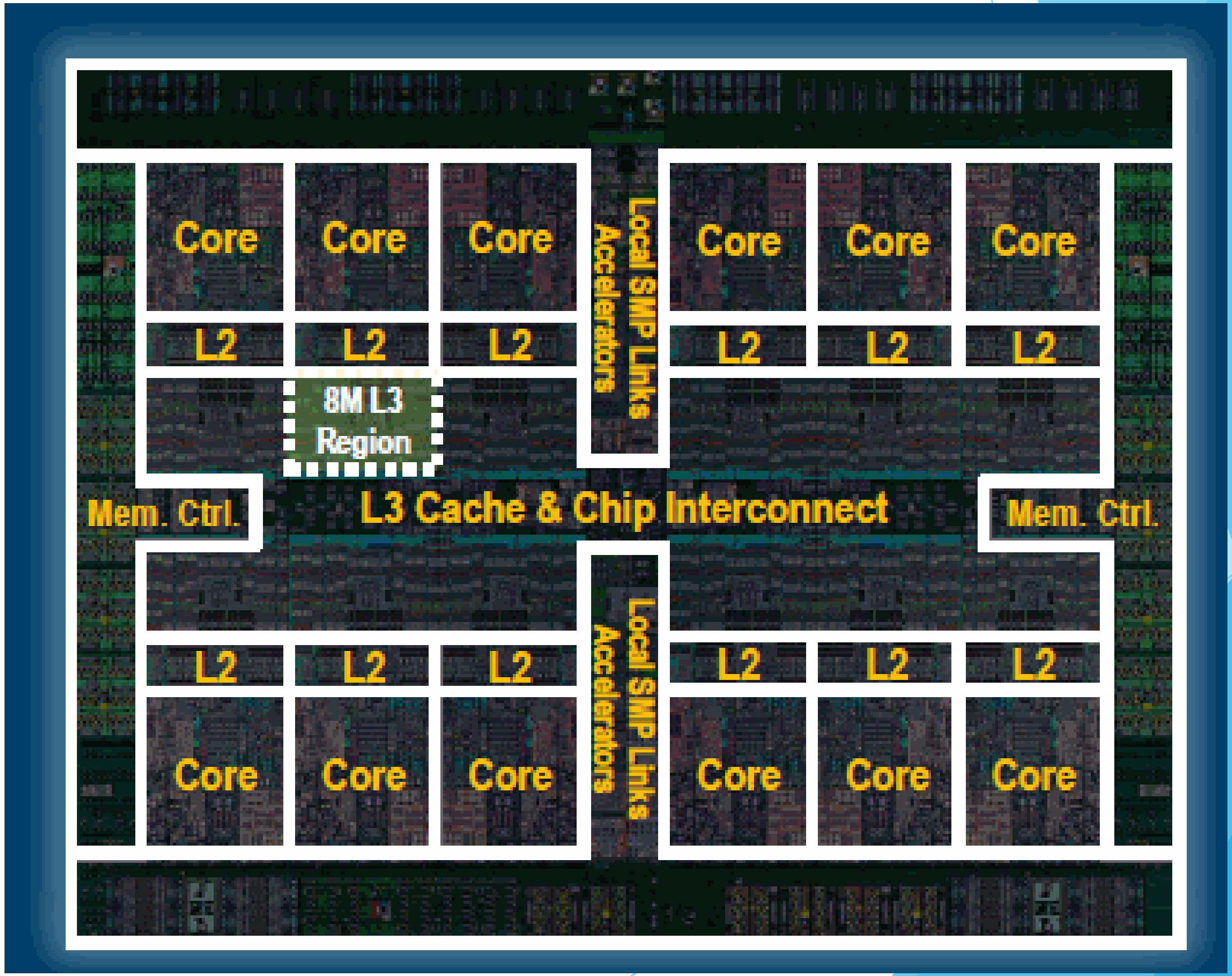
Growing computation power in supercomputers

- Supercomputers
  - Powerful compute nodes
  - Interconnection network & I/O subsystem

- Within each compute node
  - Multiple cores per chip, multiple threads per core
  - Acceleration devices: GPU, Intel MIC

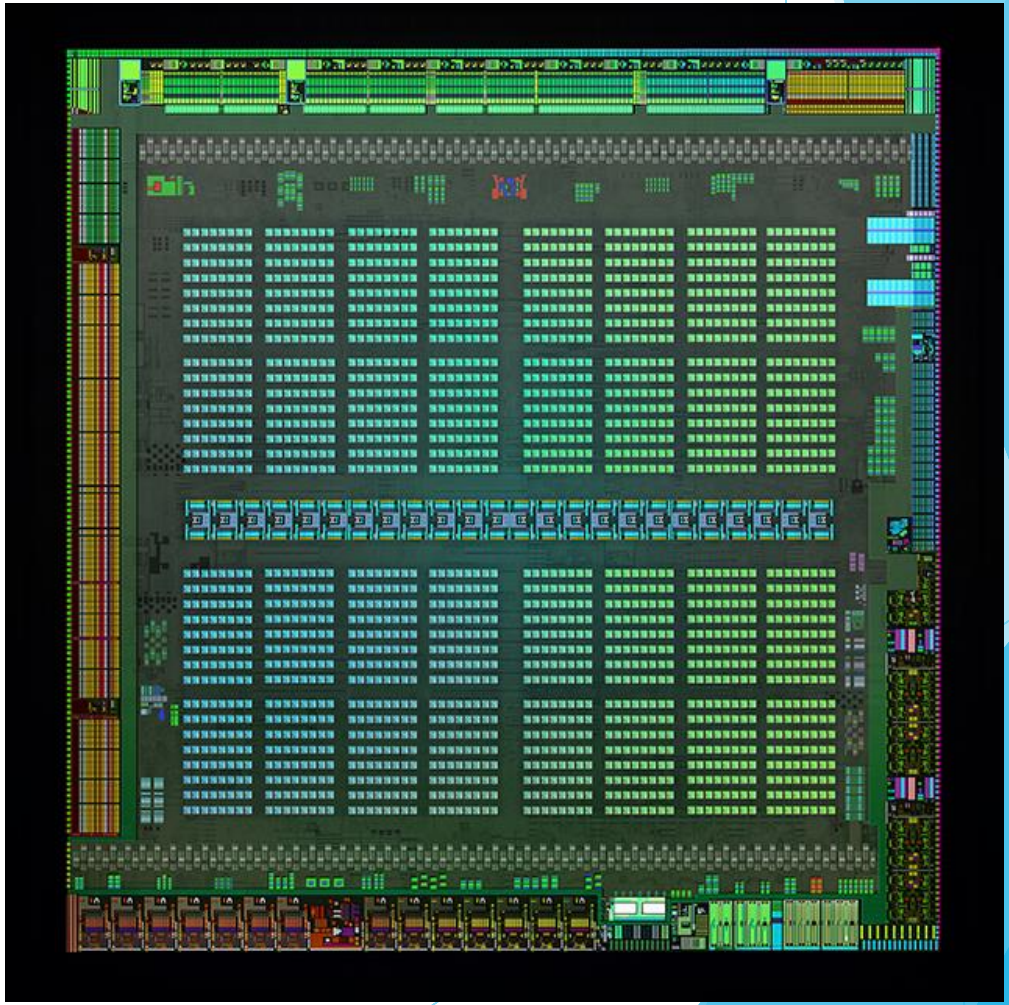
IBM POWER8

- 12 cores per chip
- 8 threads per core



NVIDIA GeForce GTX 980

- 2048 CUDA cores



SIERRA

- Next generation supercomputer coming to LLNL (2017)
  - ~150 petaflops
  - ~5x compared to top
  - IBM POWER9
  - NVIDIA GPU

Harnessing many levels of architectural parallelism

- Using a wide range of parallel programming models
  - Inter-node: MPI
  - Intra-node
    - Between CPUs: OpenMP, Cilk Plus
    - Host - Device: OpenMP 4, CUDA, OpenCL

MPI+OpenMP as a solution

- MPI dominates in message passing programming models
- OpenMP 4 supports both CPUs and devices

- MPI+OpenMP help exploit the SIERRA machine

- However, lack of debugging support

Outline

- OpenMP debugging support
  - Motivation -- what’s at hand
  - Background -- OMPD
  - Approach -- OpenMP stack builder
- MPI+OpenMP debugging support

An example of OpenMP debugging

```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    foo();
29    return 0;
30  }
```

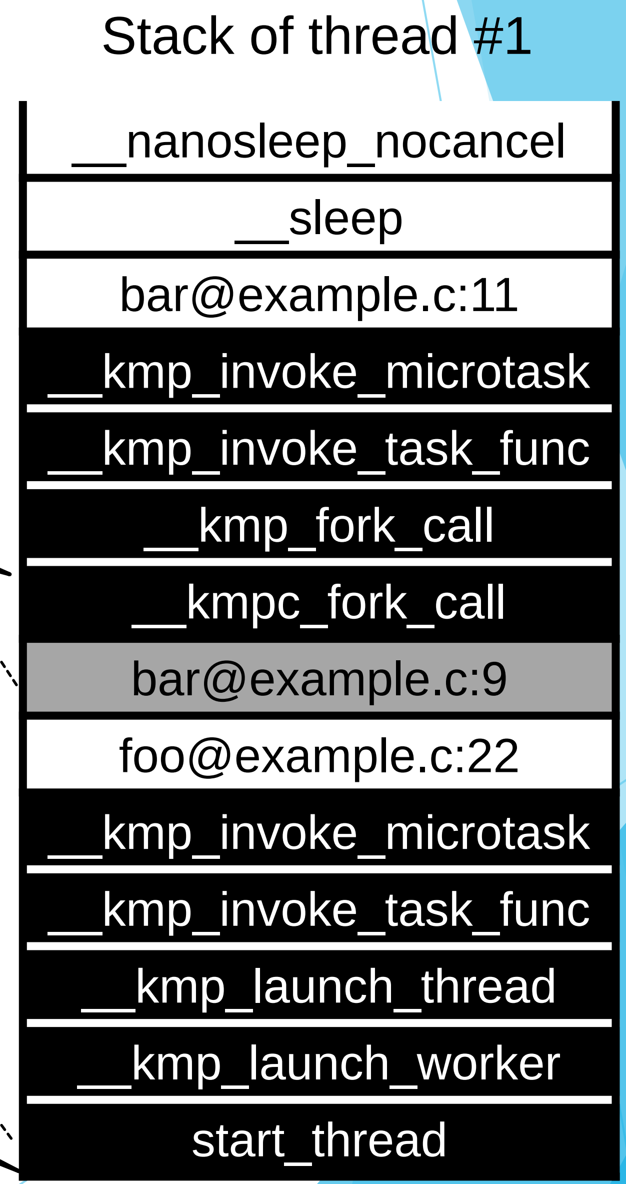
Unnecessary frames from OpenMP runtime

Stack of main thread

__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:28
__libc_start_main
_start

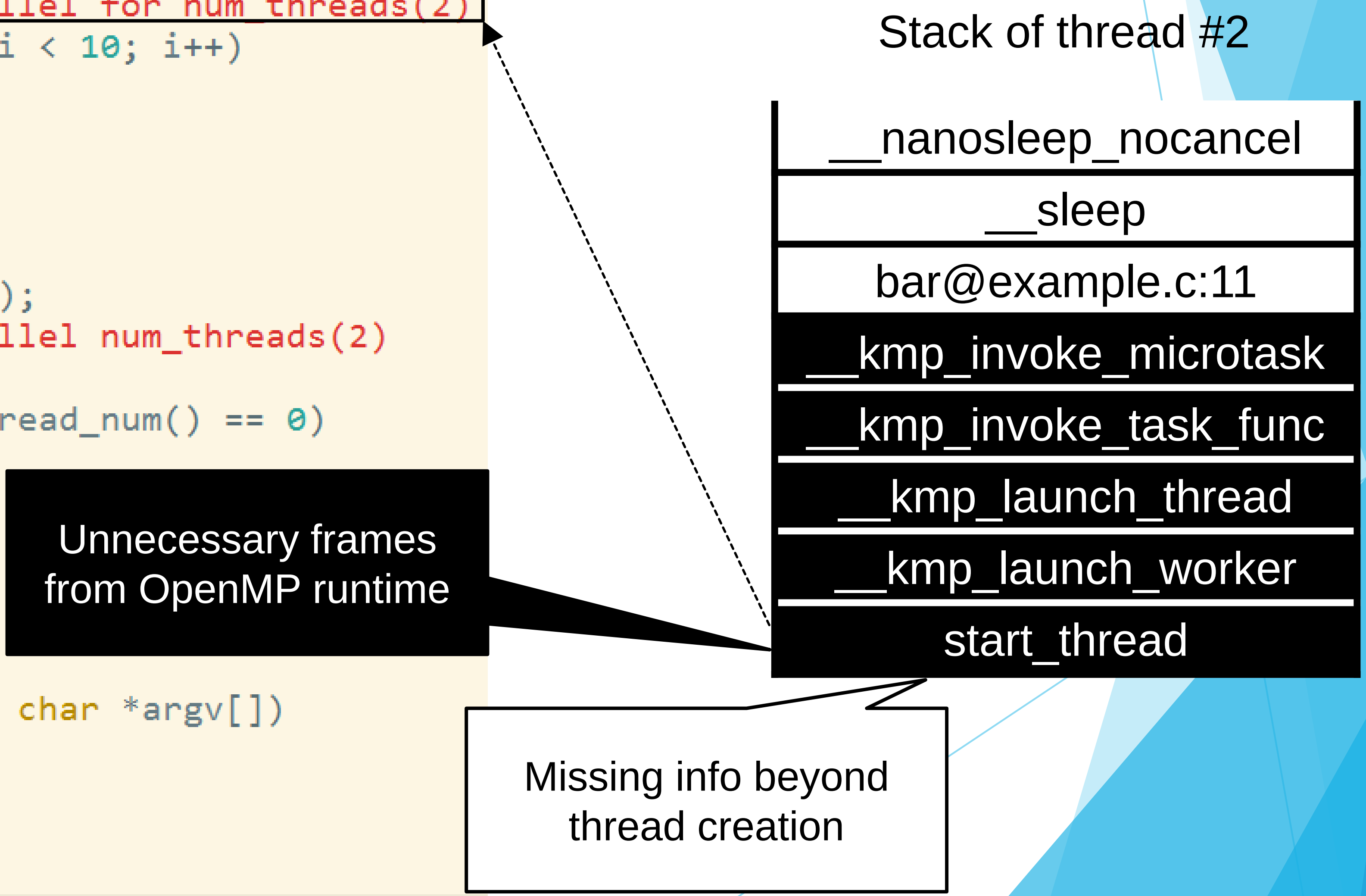
An example of OpenMP debugging

```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    foo();
29    return 0;
30  }
```



An example of OpenMP debugging

```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   foo();
29   return 0;
30 }
```

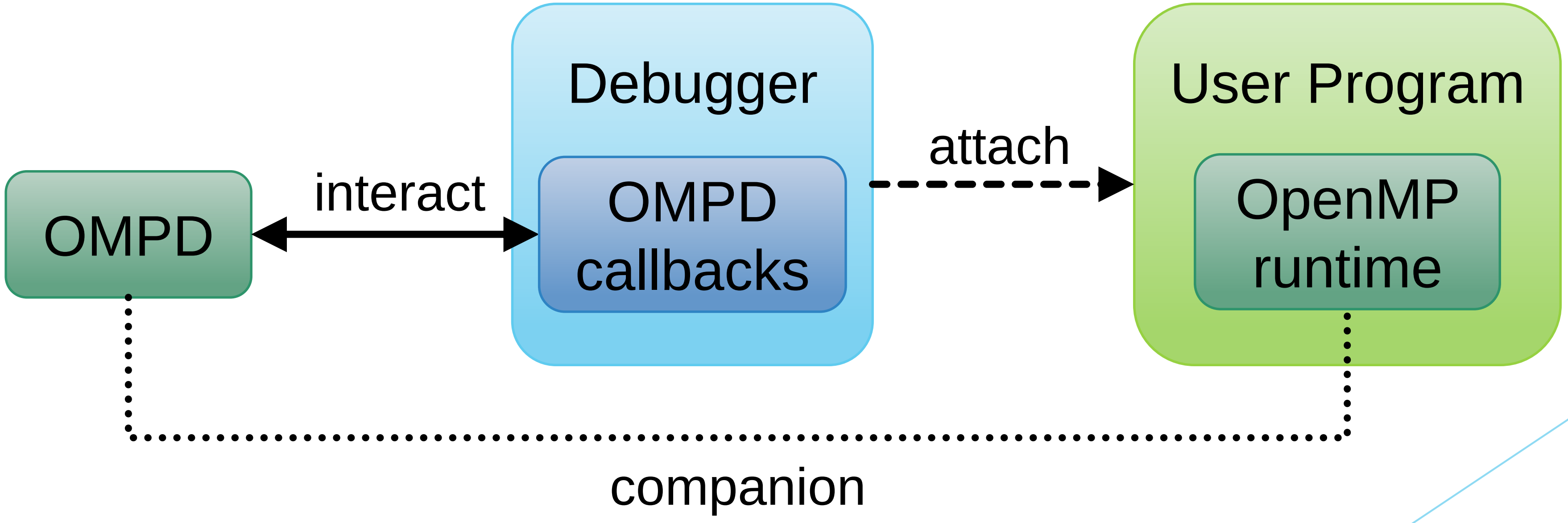


What’s at hand

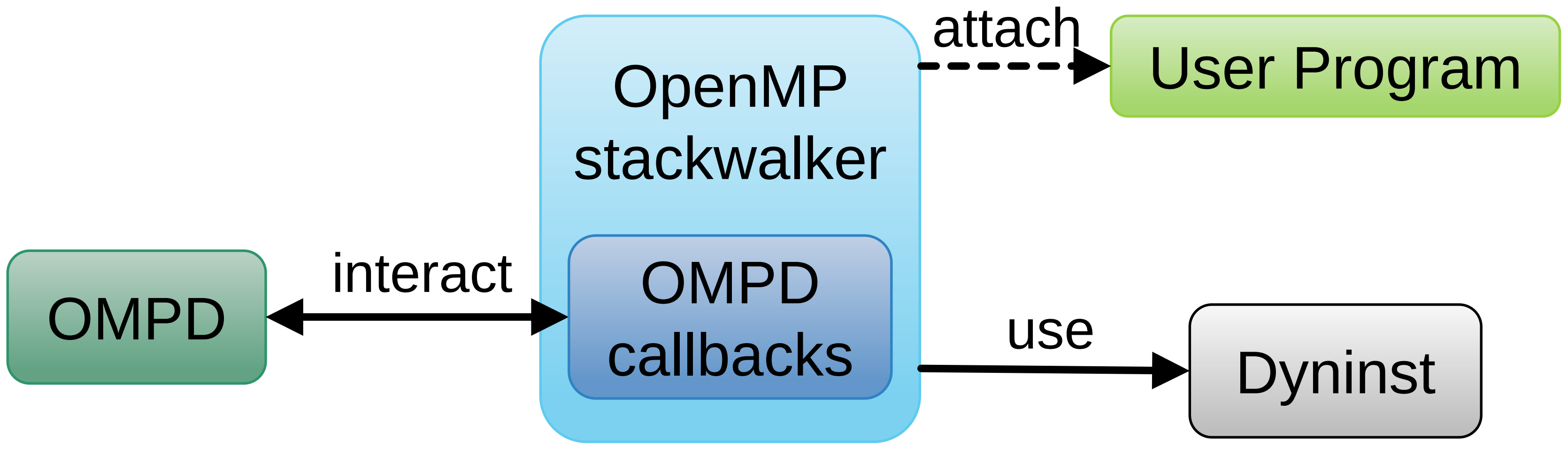
- Debugging OpenMP programs could be painful
  - Debugger users need intuitive stack info to reason about bugs
  - However, raw stacks of OpenMP threads don’t make sense
  - Need a way to reconstruct the stacks

Background -- OMPD

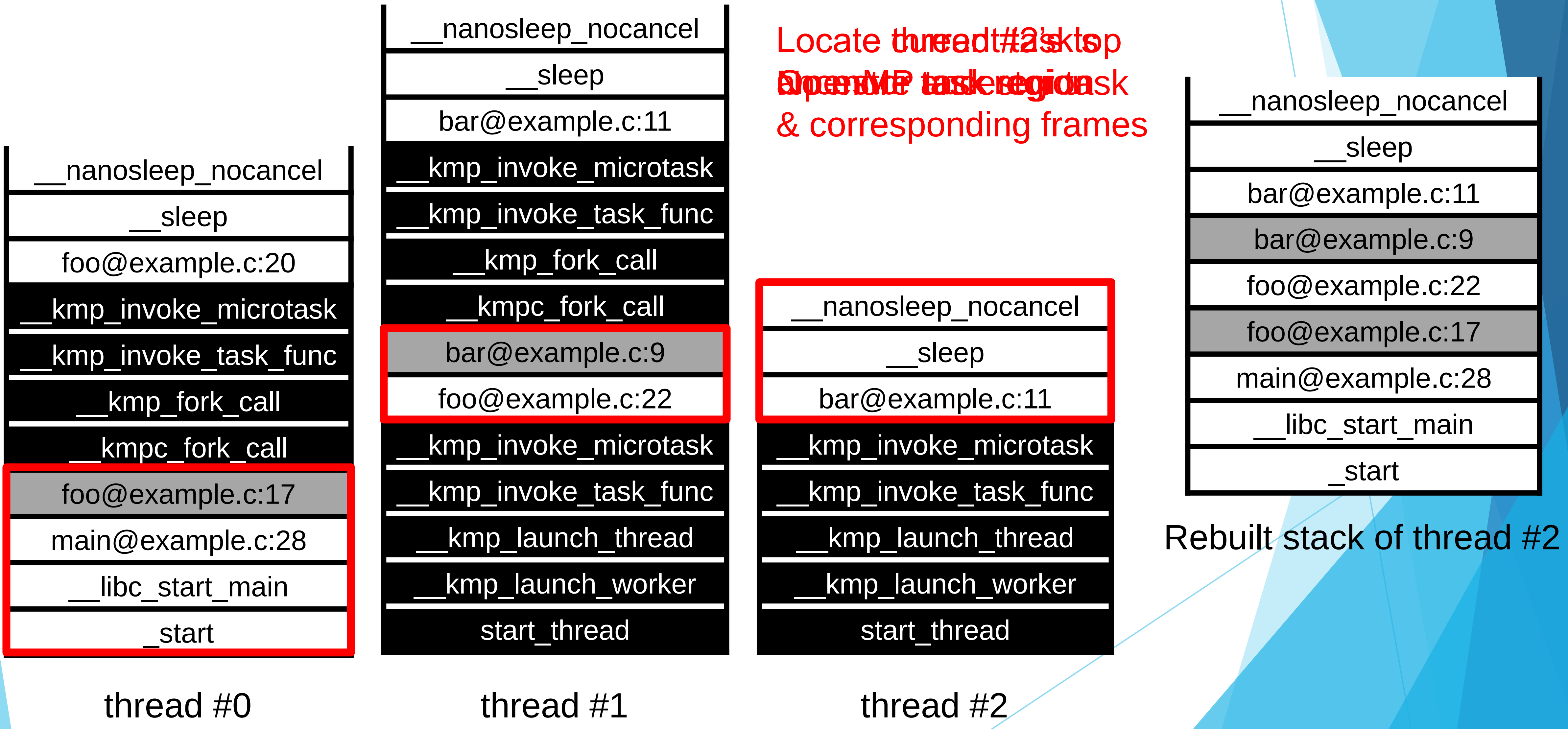
- A shared library companion to an OpenMP runtime system



OpenMP stackwalker



Rebuilding stacks for OpenMP threads



Results of OpenMP stackwalker

```
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    foo();
29    return 0;
30  }
```

Stack of main thread

__nanosleep_nocancel
__sleep
foo@example.c:20
foo@example.c:17
main@example.c:28
__libc_start_main
_start

Results of OpenMP stackwalker

```
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    foo();
29    return 0;
30  }
```

Stack of 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:28
__libc_start_main
_start

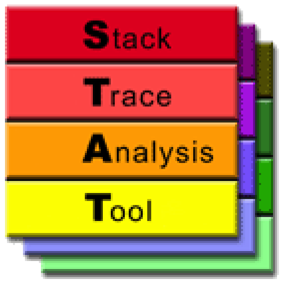
Summary of OpenMP Debugging

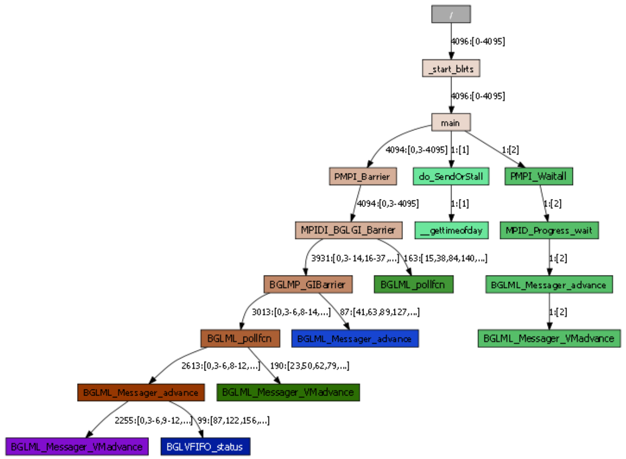
- Debugging OpenMP programs could be painful
  - Stacks of OpenMP threads are not intuitive
- People proposed OMPD to facilitate OpenMP debugging
- Our work: rebuild stacks of OpenMP threads
  - Eliminated unnecessary frames from OpenMP runtime
  - Rebuild the full calling context for OpenMP threads

Outline

- OpenMP debugging support
- MPI+OpenMP debugging support

Stack trace analysis tool (STAT)

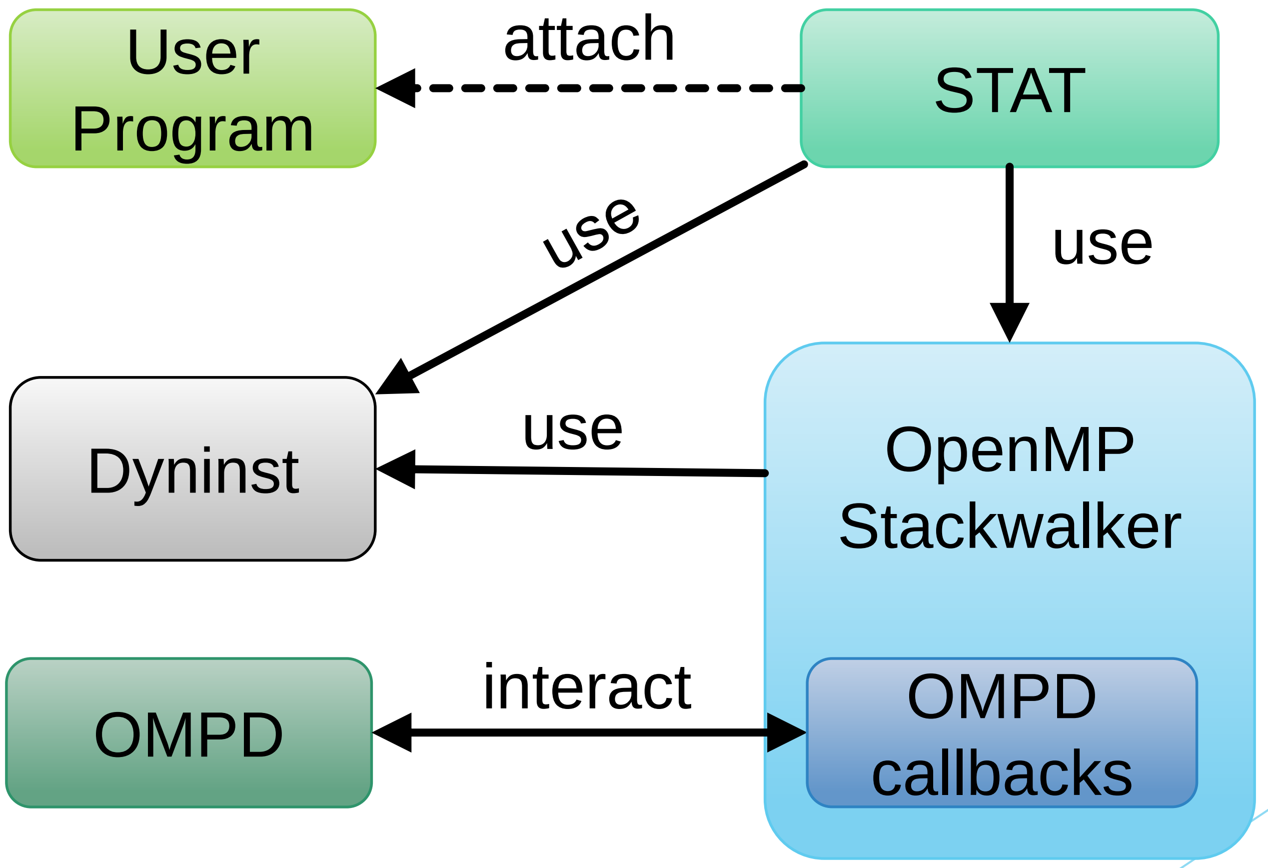




- Developed at LLNL
  - Highly-scalable, lightweight debugging tool for MPI applications
  - Various views to facilitate debugging on MPI

- We try to add OpenMP awareness into STAT so that it supports MPI+OpenMP debugging

STAT for MPI+OpenMP



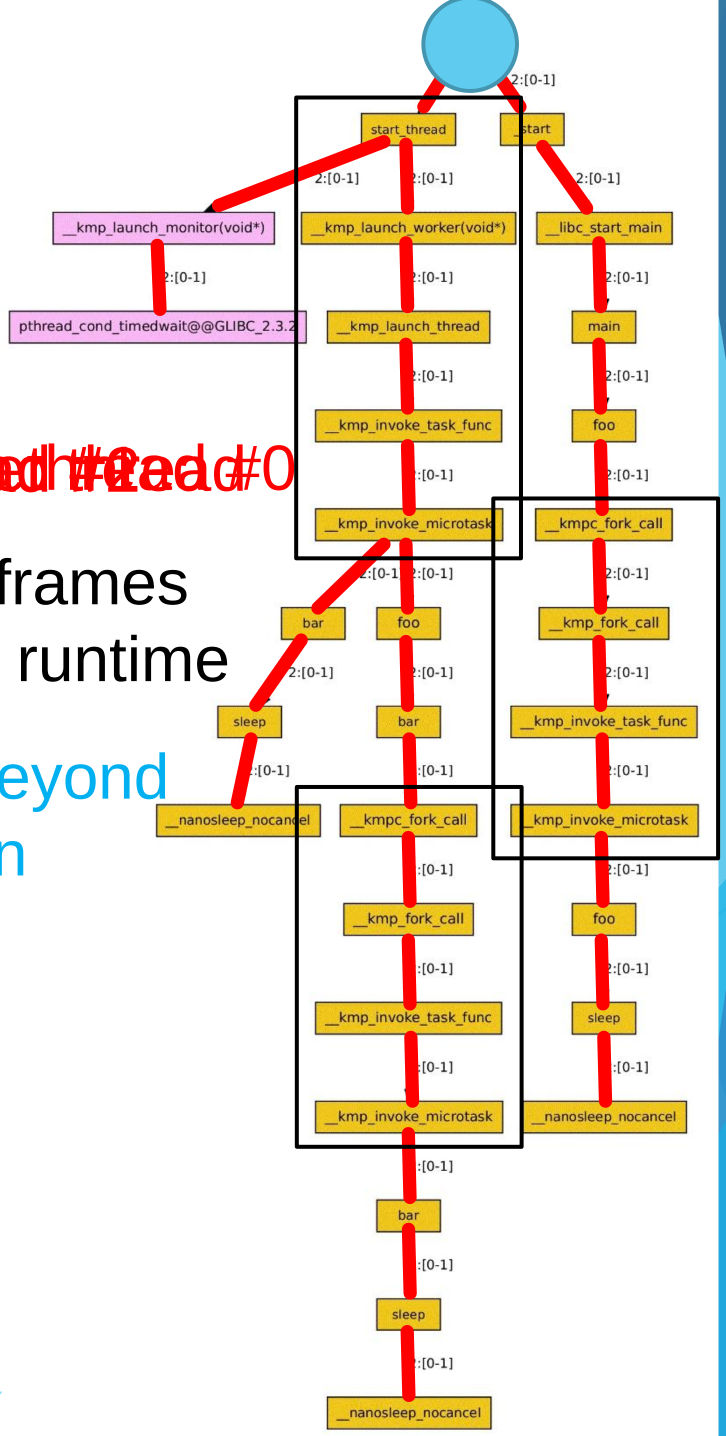
STAT without OpenMP awareness

```
void bar()
{
  #pragma omp parallel for num_threads(2)
  for (int i = 0; i < 10; i++)
    sleep(18);
}

void foo()
{
  omp_set_nested(1);
  #pragma omp parallel num_threads(2)
  {
    if (omp_get_thread_num() == 0)
      sleep(90);
    else
      bar();
  }
}

int main(int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  foo();
  MPI_Finalize();
  return 0;
}
```

OpenMP thread #0

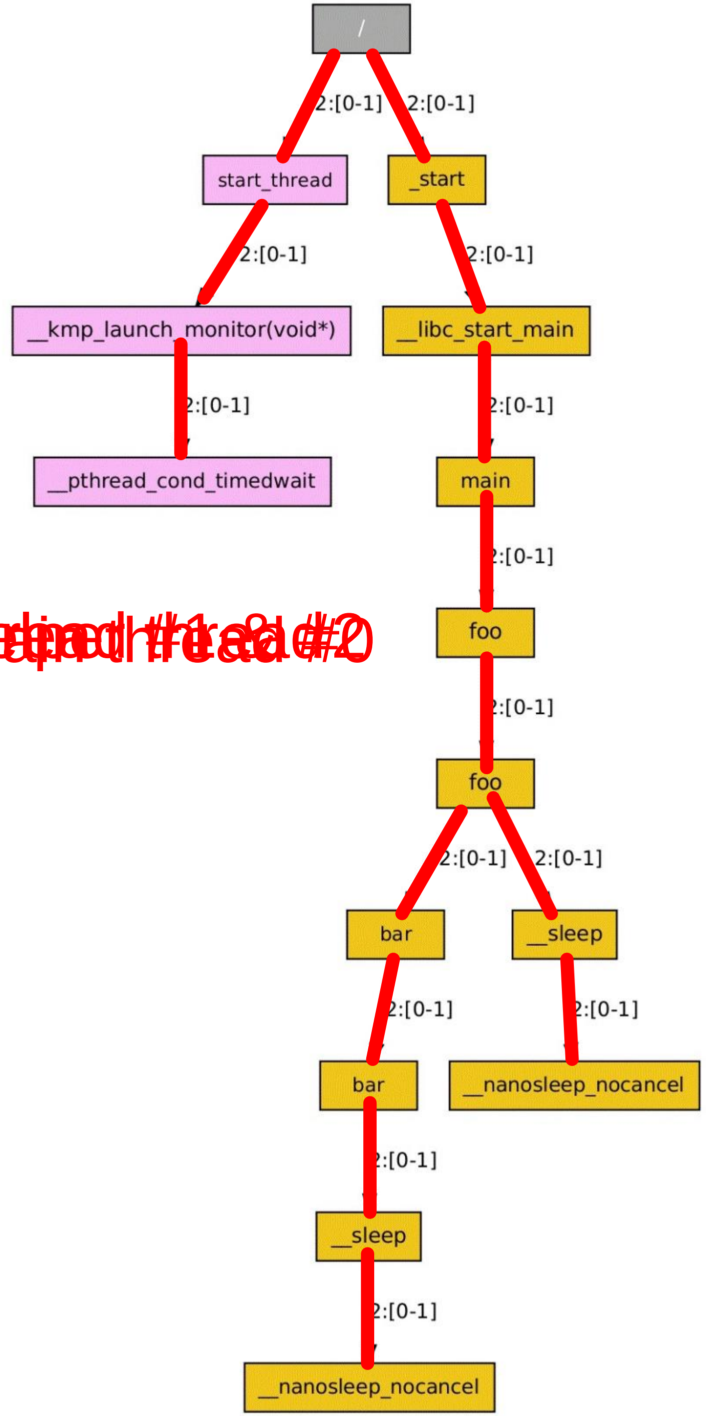
Unnecessary frames from OpenMP runtime

Missing info beyond thread creation



STAT with OpenMP awareness

```c
void bar()
{
  #pragma omp parallel for num_threads(2)
  for (int i = 0; i < 10; i++)
    sleep(18);
}

void foo()
{
  omp_set_nested(1);
  #pragma omp parallel num_threads(2)
  {
    if (omp_get_thread_num() == 0)
      sleep(90);
    else
      bar();
  }
}

int main(int argc, char *argv[])
{
  MPI_Init(&argc, &argv);
  foo();
  MPI_Finalize();
  return 0;
}
```



Conclusion

- Rebuild stacks of OpenMP threads using OMPD
- Provide intuitive stack trace view of MPI+OpenMP programs (prototype)

Future work

- Further improve generated stack trace view
- Evaluate STAT on large MPI+OpenMP applications

- Allowing debugging of OpenMP 4.0 programs (device support)
- Other views including OpenMP task view, parent/child view, etc.

References

- [1] D. Arnold, D. Ahn, B. de Supinski, G. Lee, B. Miller, and M. Schulz. Stack trace analysis for large scale debugging. In Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, pages 1-10, March 2007.
- [2] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. Cownie, R. Dietrich, X. Liu, E. Loh, and D. Lorenz. OpenMP technical report 2 on the OMPT interface. Technical report, 2014.
- [3] A. Eichenberger, J. Mellor-Crummey, M. Schulz, N. Copty, J. Cownie, R. Dietrich, J. Signore, E. Loh, and D. Lorenz. OMPD: An application programming interface for a debugger support library for OpenMP. Technical report, 2014.
- [4] G. Ravipati, A. R. Bernat, N. Rosenblum, B. P. Miller, and J. K. Hollingsworth. Toward the deconstruction of dyninst. Technical report, 2007.