



OpenMP Tool Interfaces in OpenMP 5.0

[Joachim Protze \(protze@itc.rwth-aachen.de\)](mailto:protze@itc.rwth-aachen.de)

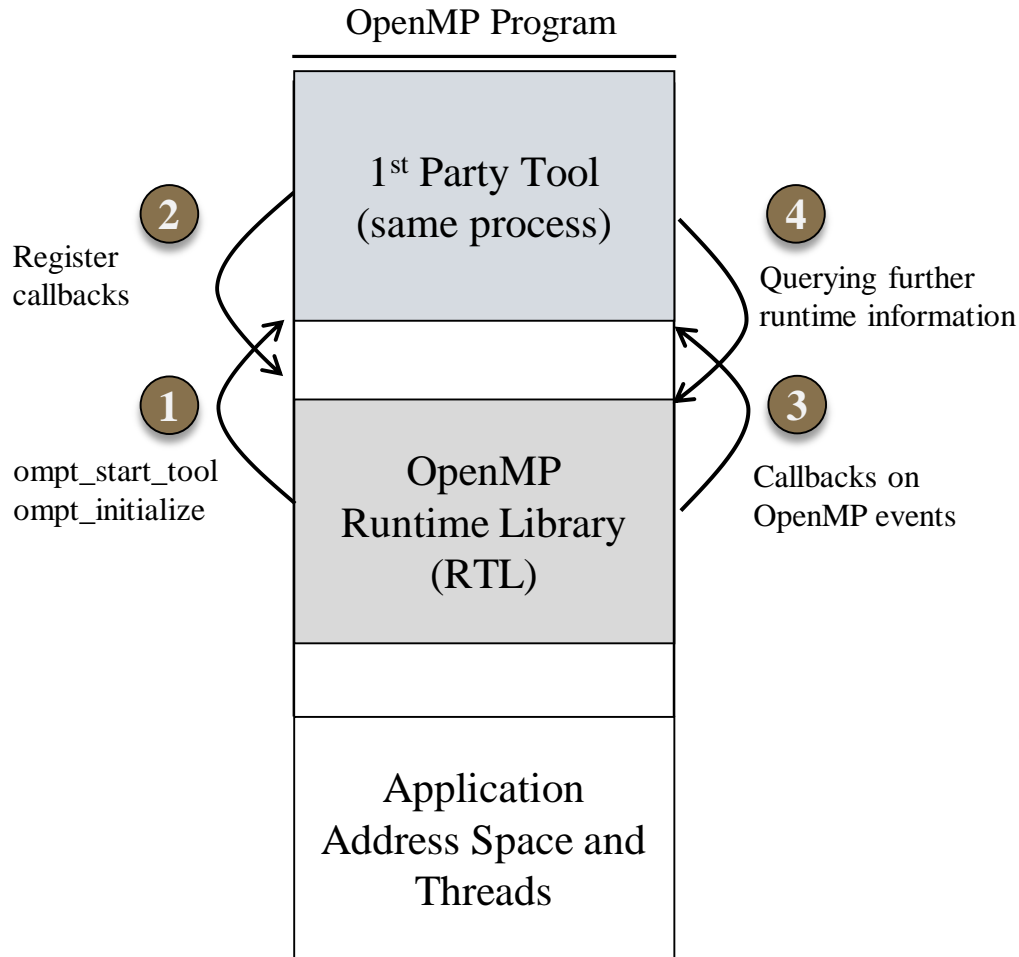
Motivation

- Motivation
 - Tool support essential for program development
 - Users expect tool support, especially in today's complicated systems
 - Question of productivity
 - Users want portable tools across systems/implementations
- Currently not possible in OpenMP due to missing interfaces
 - Tools must be OpenMP implementation specific
 - Only alternative: show underlying system information only

Two OpenMP Tools Interfaces

- OpenMP Tools Interface (OMPT)
 - 1st party interface as integral part of the runtime system
 - Combination of passive state tracking and active callbacks
 - Mechanism to allow “clean” stack traces
- OpenMP Debug Interface (OMPD)
 - 3rd party interface for external debuggers
 - Separate address space, but closely coupled with runtime
 - Follows OMPT concepts and definitions
- Goal: inclusion into OpenMP 5.0 (reached with TR6)

OMPT Architecture



1 Runtime initialization

- OpenMP runtime searches for tool
- Runtime finished initialization
- Runtime allows the tool to initialize

2 Tool initialization

- Tool looks up OMPT functions with look-up
- Tool registers callbacks for OpenMP events

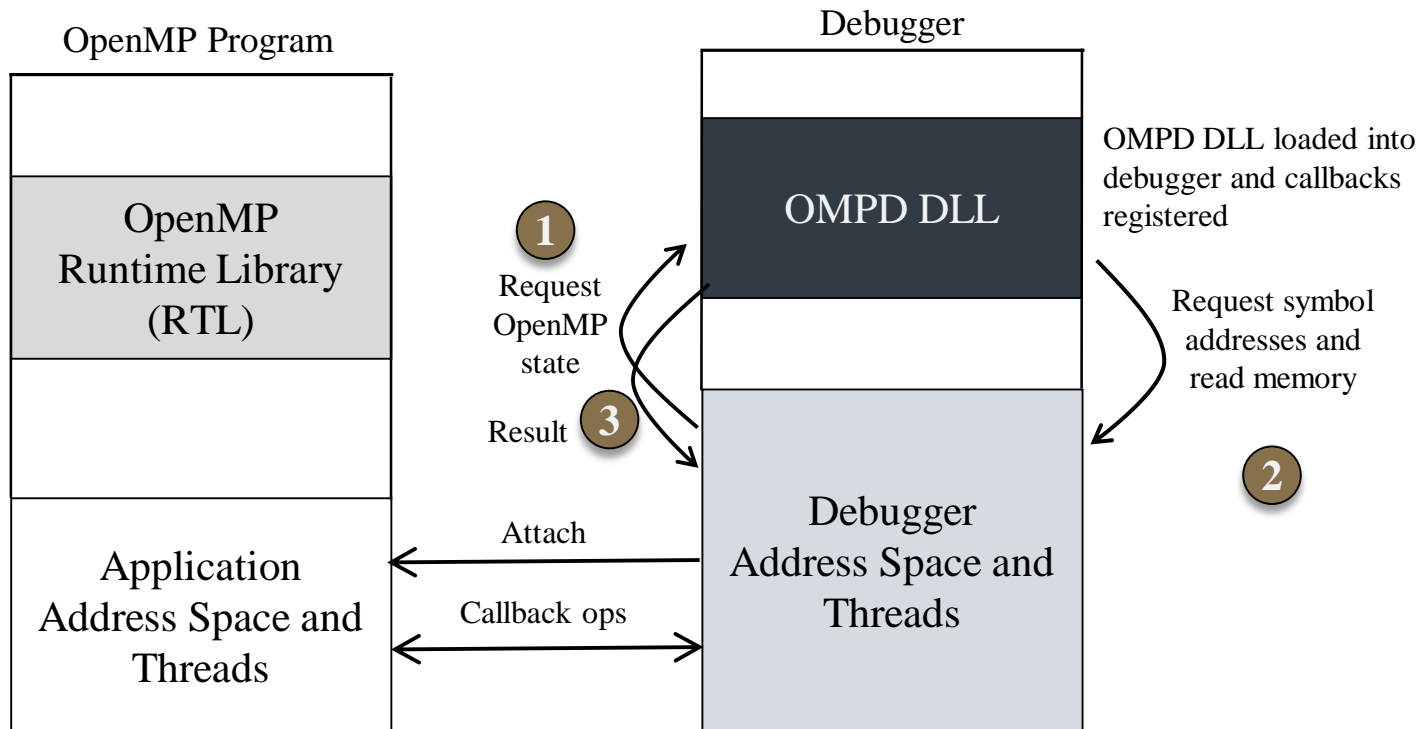
3 Callback functions

- Callbacks on OpenMP events
- E.g., entering/exiting parallel, target, worksharing, or synchronization regions

4 Querying further information

- OMPT functions to
 - investigate thread state
 - identify OpenMP runtime frames

OMPD Architecture



Request types

1

- Handles for threads, parallel regions, tasks
- Parent / child relationships
- State of handles (wait, work, idle)

Callback functions

2

- Lookup symbols in the target process
- Read/write target address spaces
- Support for GPUs (coming soon)

OMPD Use Case: OpenMP Stack Filtering

The image displays two side-by-side screenshots of a debugger's stack trace window, illustrating the process of filtering OpenMP stack frames. Both screenshots show the same process (Process 1 (38822): tx_openmp_parallel_n) and thread (Thread 1 (140508016441152)).

Left Screenshot (Stack Filtering): The stack trace is filtered to show only frames relevant to the OpenMP parallel regions. Three parallel regions are highlighted with red boxes and arrows:

- Region 1:** `breakpoint_h`, `L_h_43_par_region0_2_0`, `__kmp_invoke_microtask`, `__kmp_invoke_task_func`, `__kmp_fork_call`, `__kmpc_fork_call`, `h`. (FP=7fff9b03ba20 to FP=7fff9b03c090)
- Region 2:** `L_g_69_par_region0_2_3`, `__kmp_invoke_microtask`, `__kmp_invoke_task_func`, `__kmp_fork_call`, `__kmpc_fork_call`, `g`. (FP=7fff9b03c170 to FP=7fff9b03c5f0)
- Region 3:** `L_f_105_par_region0_2_5`, `__kmp_invoke_microtask`, `__kmp_invoke_task_func`, `__kmp_fork_call`, `__kmpc_fork_call`, `f`. (FP=7fff9b03c8f0 to FP=7fff9b03cfa0)

Right Screenshot (Full Stack Trace): Shows the complete stack trace, including the filtered frames and the main thread's stack:

- `breakpoint_h`, `L_h_43_par_region0_2_0`, `h`, `L_g_69_par_region0_2_3`, `g`, `L_f_105_par_region0_2_5`, `f`, `main`, `__libc_start_main`, `_start`.

OMPD Use Cases: Linking master/worker threads

The image displays two screenshots of the OpenMP Development Environment (OMPD) interface, showing the debugging of OpenMP threads.

Top Screenshot: Shows Thread 7 (ID: 140326287362432) at a breakpoint in the function `breakpoint_h`. The stack trace shows the following frames:

- `breakpoint_h`, FP=7fa04282b3a0
- `h`, FP=7fa04282b4f0
- `h`, FP=7fa04282bd10
- `L_g_69_par_region0_2_3`, FP=7fa04282b...
- `g` (thread 1.3)

The stack frame for `breakpoint_h` shows the following variables:

- `f_id`: 0x00000001 (1)
- `g_id`: 0x00000001 (1)
- `h_id`: 0x00000000 (0)

Bottom Screenshot: Shows Thread 3 (ID: 140326304155520) at a breakpoint in the function `g`. The stack trace shows the following frames:

- `breakpoint_h`, FP=7fa04382ee20
- `L_h_43_par_region0_2_0`, FP=7fa04382e...
- `h`, FP=7fa04382f490
- `L_g_69_par_region0_2_3`, FP=7fa04382f...
- `g`, FP=7fa04382f9f0
- `L_f_105_par_region0_2_5`, FP=7fa04382...
- `f` (thread 1.I)

The stack frame for `g` shows the following variables:

- `f_id`: 0x00000001 (1)
- `nthreads`: 0x00000004 (4)
- `Local variables:`
- `remainder`: 0x00000001 (1)
- `id (&rax)`: 0x00000001 (1)

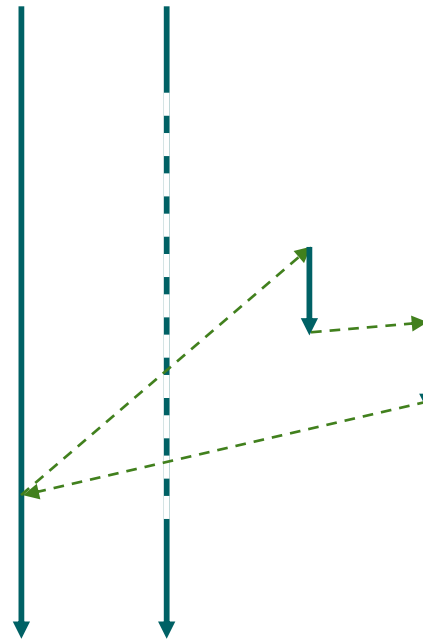
Registers for the frame:

- `&rax`: 0x00000001 (1)
- `&rdx`: 0x00000000 (0)
- `&rsp`: 0x00000000 (0)

OMPD Use Case: Stepping into OpenMP Regions

- OMPD provides information on where task execution starts:
 - This will enable to single step into task execution:
 - Program execution is paused before task scheduling point (task/taskwait/barrier)
 - Click on single-step could jump to start of next executed task

```
#pragma omp parallel
{
  #pragma omp master
  {
    a++;
    #pragma omp task shared(a,b)
    { a++; b++; }
    #pragma omp task shared(b,c)
    { b++; c++; }
    c++;
    #pragma omp taskwait
  }
  sleep(20); // emulate load
}
```



Stack trace views: serial code

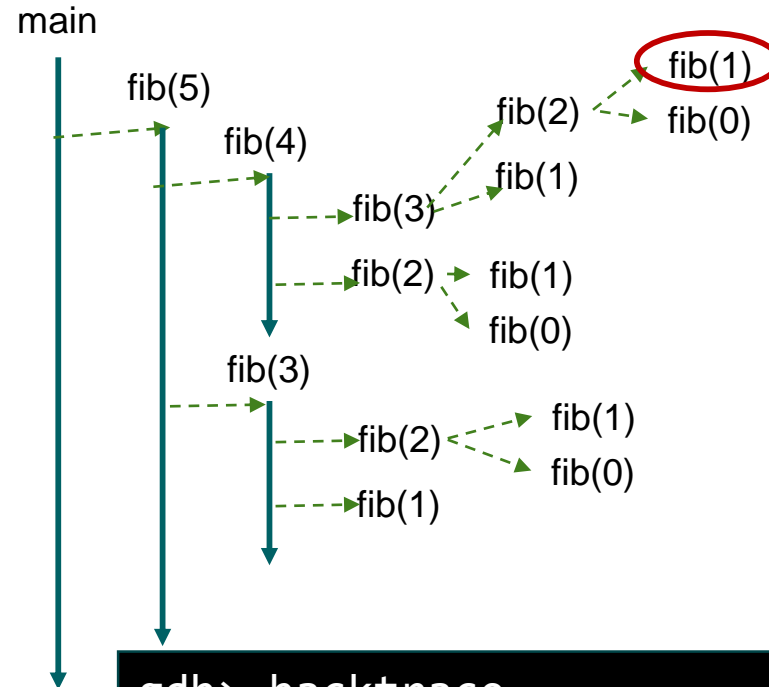
```
int fib(int i){
  if i<=1 return 1;
  int a,b;

  a = fib(i-1);

  b = fib(i-2);

  return a+b;
}
int main(){
  int result;

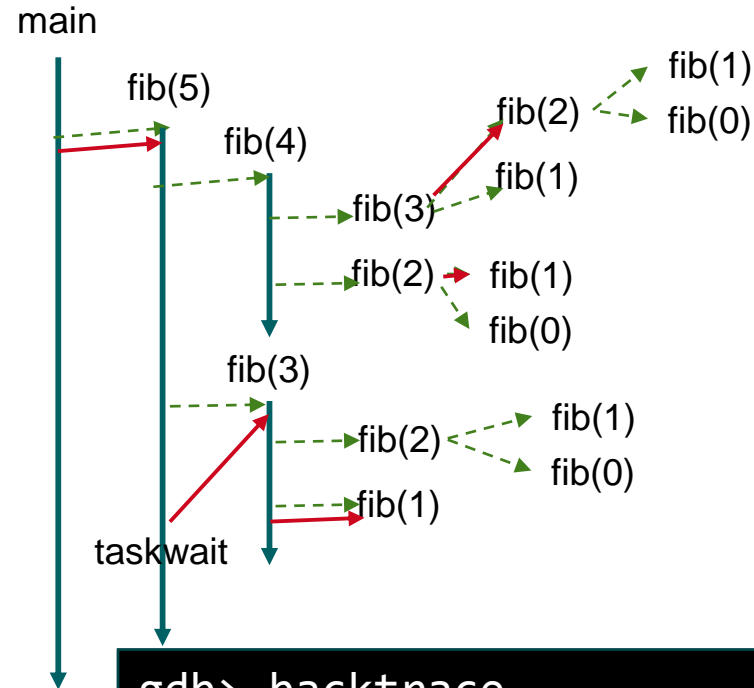
  result = fib(5);
  return result;
}
```



```
gdb> backtrace
#0 fib(1)
#1 fib(2)
#2 fib(3)
#3 fib(4)
#4 fib(5)
#5 main()
```

Stack trace views: OpenMP tasks

```
int fib(int i){
  if i<=1 return 1;
  int a,b;
  #pragma omp task shared(a)
    a = fib(i-1);
  #pragma omp task shared(b)
    b = fib(i-2);
  #pragma omp taskwait
  return a+b;
}
int main(){
  int result;
  #pragma omp parallel sections
    result = fib(5);
  return result;
}
```

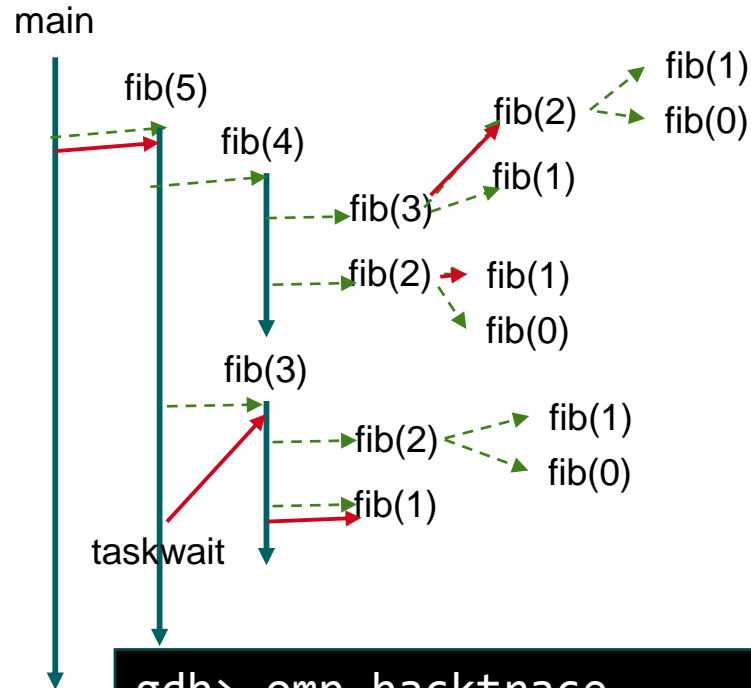


```
gdb> backtrace
#0 fib(2)
#1 fib(5)
#2 main()
```

Stack trace views: OpenMP tasks

```
int fib(int i){
  if i<=1 return 1;
  int a,b;
  #pragma omp task shared(a)
    a = fib(i-1);
  #pragma omp task shared(b)
    b = fib(i-2);
  #pragma omp taskwait
  return a+b;
}

int main(){
  int result;
  #pragma omp parallel sections
    result = fib(5);
  return result;
}
```



```
gdb> omp_backtrace
#0 fib(1)@0
#1 fib(3)@1
#2 fib(4)@1
#3 fib(5)@0
#4 main()@0
```

Implementation Status

- Implementation of OMPT for host available in LLVM/OpenMP master
 - Patches applied just recently
 - Implementation matches OpenMP TR6 specification
- Implementation of OMPT is available in IBM OpenMP runtime
 - compatible with the „TR4“-patch in LLVM/OpenMP
- Implementation of OMPT for target is work in progress
- Implementation of OMPD for host/device is/will be available shortly:
 - <https://github.com/OpenMPToolsInterface/>
 - We have plans to upstream these efforts as well

Thank you for your attention.