



OMPT Support in ROCm™

Past, Present & Future

ROCm OpenMP© GPU Compiler Team

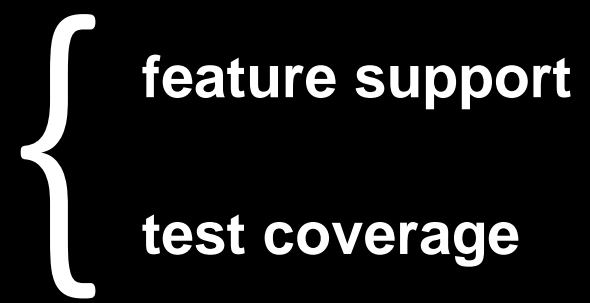
Michael Halkenhäuser, Dhruva Chakrabarti,
Jan-Patrick Lehr, Ron Lieberman

AMD 
together we advance_

„OMPT is important to us“

Our Goals:

Improve OMPT



Ensure a solid foundation for tool developers

Test Case	19.0-3
... Device ...	✓
... Device ...	✓
... Device ...	✓
... Device ...	✓
... Device ...	✓
... Target ...	✓
... Allocator ...	✓
... Allocator ...	✓
... Allocator ...	✓
... Allocator ...	✓
... Allocator ...	✓
... Allocator ...	❖
... Associate ...	✓
... Host ...	✓
... BufferRecord ...	✓
... Callbacks ...	✓
... BufferRecord ...	✓
... Callbacks ...	✓
... BufferRecord ...	✓
... Callbacks ...	✓
... Callbacks ...	✓

Introduction

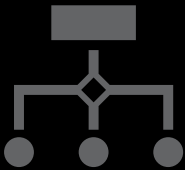
ROCm



AMD ROCm is an open software platform for GPU compute consisting of compilers, libraries, tools, etc.

<https://www.amd.com/en/products/software/rocm.html>

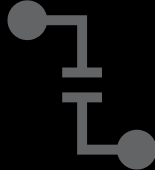
AOMP



AOMP is an open-source Clang/LLVM-based compiler with added support for the OpenMP® API on Radeon™ GPUs that builds on top of ROCm

Releases are more frequent than ROCm
<https://github.com/ROCm/aomp>

LLVM



Since AOMP is Clang/LLVM based, the high-level software architecture of OpenMP target offload support is identical

AOMP is both ahead and behind LLVM mainline trunk (upstream)

“downstream”

“upstream”

OMPT Events 101

```
int main(int argc, char **argv) {  
    int Values[1024] = {0};  
  
    #pragma omp target loop  
    for(int i = 0; i < 1024; ++i) {  
        Values[i] = 1;  
    }  
  
    return 0;  
}
```



CALLBACKS <ul style="list-style-type: none">• synchronous• ordered
TRACE RECORDS <ul style="list-style-type: none">• asynchronous• unordered

Fun fact: Code snippet generates more than 30 device events

OMPT Device Events 101

CALLBACKS

- synchronous
- ordered

TRACE RECORDS

- asynchronous
- unordered

Callback Kernel Submit EMI

```
endpoint=1
req_num_teams=1
...
```

```
rec=0x1257a00
type=10 (Target kernel)
time=8213834914847410
...
```

EMI Callbacks

- Provide event information directly
- Become default in OpenMP 6.0

Trace Records

- Provide info on extended events
 - Begin- & End-time
 - Misc. data structures
 - ...

Tools using OMPT events, e.g.

- score-p, TAU and HPCToolkit

Beginning of Downstream OMPT Support

- **Callbacks implemented in HPCToolkit's LLVM fork (by John Mellor-Crummey)**
- **Callbacks refactored for downstream**
- **Trace Records implemented in downstream**
- **Commitment to open-source software**
→ **Upstream our changes**

Commit

```
[libomptarget] [amdgpu] Foundation for OMPT target callback support
OMPT callback support for target task, target data op, and target submit
(With contributions from Dhruva Chakrabarti <Dhruva.Chakrabarti@amd.com>)

o Connect up libomp, libomptarget and AMDGPU device rtl
o Provide a callback class used to interact with a tool
o Update omp-tool.h.var from LLVM main to ensure it is fresh
o Update libomp sources for two purposes
  1. Adapt to new macros in omp-tools.h
  2. Add API used by libomptarget to extract task_data and target_task_data
  support OMPT EMI calls
o Integrate support for EMI and NONEMI target callbacks
o Target, data operation, and target submit callbacks added

Change-Id: Ia06676e15e7f38587326dddbc58a47ed3c19f0a
```

amd-staging
rocm-6.2.0 ... rocm-5.1.0

jmellorcrummey authored and ronlieb committed on Dec 11, 2021

Commit

Showing 21 changed files with 1,410 additions and 35 deletions.

```
Implementation of OMPT target device tracing
- Device tracing entry points for OMPT data type
  o ompt_set_trace_ompt
  o ompt_start_trace, ompt_flush_trace, ompt_stop_trace
  o ompt_advance_buffer_cursor, ompt_get_record_ompt
- Buffer management for trace records
- Trace record generation at runtime entry points
- Support for flushing a buffer when it is full
- Helper threads for dispatching buffer-completion callbacks

Change-Id: Ia3f243830ed153bf89a199dfc6faeb05fb484d7c
```

amd-staging
rocm-6.2.0 ... rocm-5.2.0

dhruvachak authored and ronlieb committed on Jan 12, 2022

Showing 12 changed files with 1,648 additions and 39 deletions.



Upstreaming of OMPT Device Callback Support

- **Segmentation of downstream-diff**
 - **Result: 8 Phabricator patches**
- **Upstream is a moving target**
 - **Addition of ‘nextgen-plugins’ needed refactoring of patches 4-7**
- **Refactoring**
 - **Templated design desired (by Johannes Doerfert)**
- **Reordering**
 - **“Test first, add callbacks later”**

The screenshot displays a list of eight Phabricator patches related to OpenMP (OMP) device callback support. Each patch entry includes a title, status (Closed), visibility (Public), author, and date.

- [OpenMP] [OMPT] [1/8] Create separate categories for host, device, [no]emi events**
 Authored by [dhruvachak](#) on Apr 8 2022, 6:29 PM.
- [OpenMP] [OMPT] [2/8] Implemented a connector for communication of OMPT callbacks between libraries.**
 Authored by [dhruvachak](#) on Apr 12 2022, 12:00 AM.
- [OpenMP] [OMPT] [3/8] Implemented callback registration in libomptarget**
 Authored by [dhruvachak](#) on Apr 18 2022, 7:24 PM.
- [OpenMP] [OMPT] [amdgpu] [4/8] Implemented callback registration in nextgen plugins**
 Authored by [mhalk](#) on Apr 19 2022, 11:46 PM.
- [OpenMP] [OMPT] [amdgpu] [5/8] Implemented device init/fini/load callbacks**
 Authored by [mhalk](#) on Apr 28 2022, 6:44 PM.
- [OpenMP] [OMPT] [6/8] Added callback support for target data operations, target submit, and target regions.**
 Authored by [mhalk](#) on Jun 8 2022, 5:21 PM.
- [OpenMP] [OMPT] [7/8] Invoke tool-supplied callbacks before and after target launch and data transfer operations.**
 Authored by [mhalk](#) on Jun 8 2022, 6:01 PM.
- [OpenMP] [OMPT] [8/8] Added lit tests for OMPT target callbacks**
 Authored by [dhruvachak](#) on Jun 8 2022, 6:58 PM.

Meanwhile “behind the scenes”

- Adoption of upstream callback support
 - Refactor trace record handling
 - Prepare **downstream** set of OMPT patches

Commit

[OpenMP] [OMPT] [7/8] Invo
 ... target launch and data tran
 Implemented RAII objects, ini
 invoke tool-supplied callback
 implemented.
 Depends on D127365
 Patch from John Mellor-Crumme
 With contributions from:
 Dhruva Chakrabarti <Dhruva.Ch
 Jan-Patrick Lehr <janpatrick.
 Reviewed By: jdoerfert, dhruv
 Differential Revision: <https://reviews.llvm.org/D127367>
 Change-Id: Ic6fcee7059aa4e6237e81e2a702adca6a26bcd6

amd-staging
 rocm-6.2.0

mhalk authored and ronlieb committed on Sep 8, 2023

mhalk committed on Jul 25, 2023

Showing 20 changed files with 1,228 additions and 970 deletions.

Showing 14 changed files with 473 additions and 88 deletions.

- Expansion of OMPT capabilities
 - Implement asynchronous trace records

Commit

✓ [OpenMP][OMPT] Asynchronous Trace Events

Removes the restriction to execute OpenMP target offload runtime calls
 synchronously when executing under OMPT.

The patch introduces a small data environment that is added per
 AsyncInfo object to capture and transport the relevant data from the
 libomptarget-level into the plugin and make it available to the actual
 kernel launch / data copy parts.
 The plugin then invokes callbacks to finalize the already allocated OMPT
 trace records with the respective runtime information about timing, etc.

Change-Id: Id1f0a1905e929d7f739482e6270100b6729d53e3

amd-staging

jplehr authored and ronlieb committed on Jul 9

Showing 8 changed files with 527 additions and 170 deletions.

Downstream OMPT Support History

Test Case	18.0-1	19.0-0	19.0-2	19.0-3
OnDeviceDefaultTeams_Sequenced	✓	✓	✓	✓
OnDeviceSetNumTeams_Sequenced	✓	✓	✓	✓
OnDevice_Sequenced_Grouped	✓	✓	✓	✓
ExplicitDefaultDevice	✓	✓	✓	✓
MultipleHosts	✓	✓	✓	✓
DeclareTargetGlobalAndUpdate	✓	✓	✓	✓
ExplicitAllocatorAPI	✓	✓	✓	✓
ExplicitAllocatorAndCopyAPI	✗	✓	✓	✓
ExplicitAllocatorAndUpdate_NoDataOp	✗	✗	✗	✓
ExplicitAllocatorAndUpdate_DataOp	✗	✗	✗	✓
ExplicitAllocatorAndUpdate_DataOpStack	✗	✗	✗	✓
ExplicitAllocatorAndUpdate_Device	❖	❖	❖	❖
ExplicitAssociateDisassociatePtrAPI	✗	✗	✗	✓
OnHost_One	✓	✓	✓	✓
OnDeviceBufferRecord_parallel_for	✓	✓	✓	✓
OnDeviceCallbacks_parallel_for	✓	✓	✓	✓
OnDeviceBufferRecord_teams_distribute_parallel_for	✓	✓	✓	✓
OnDeviceCallbacks_teams_distribute_parallel_for	✓	✓	✓	✓
OnDeviceBufferRecord_teams_distribute_parallel_for_nowait	✓	✓	✓	✓
OnDeviceCallbacks_teams_distribute_parallel_for_nowait	✓	✓	✓	✓
DeviceLoad	✓	✓	✓	✓

Host

Device

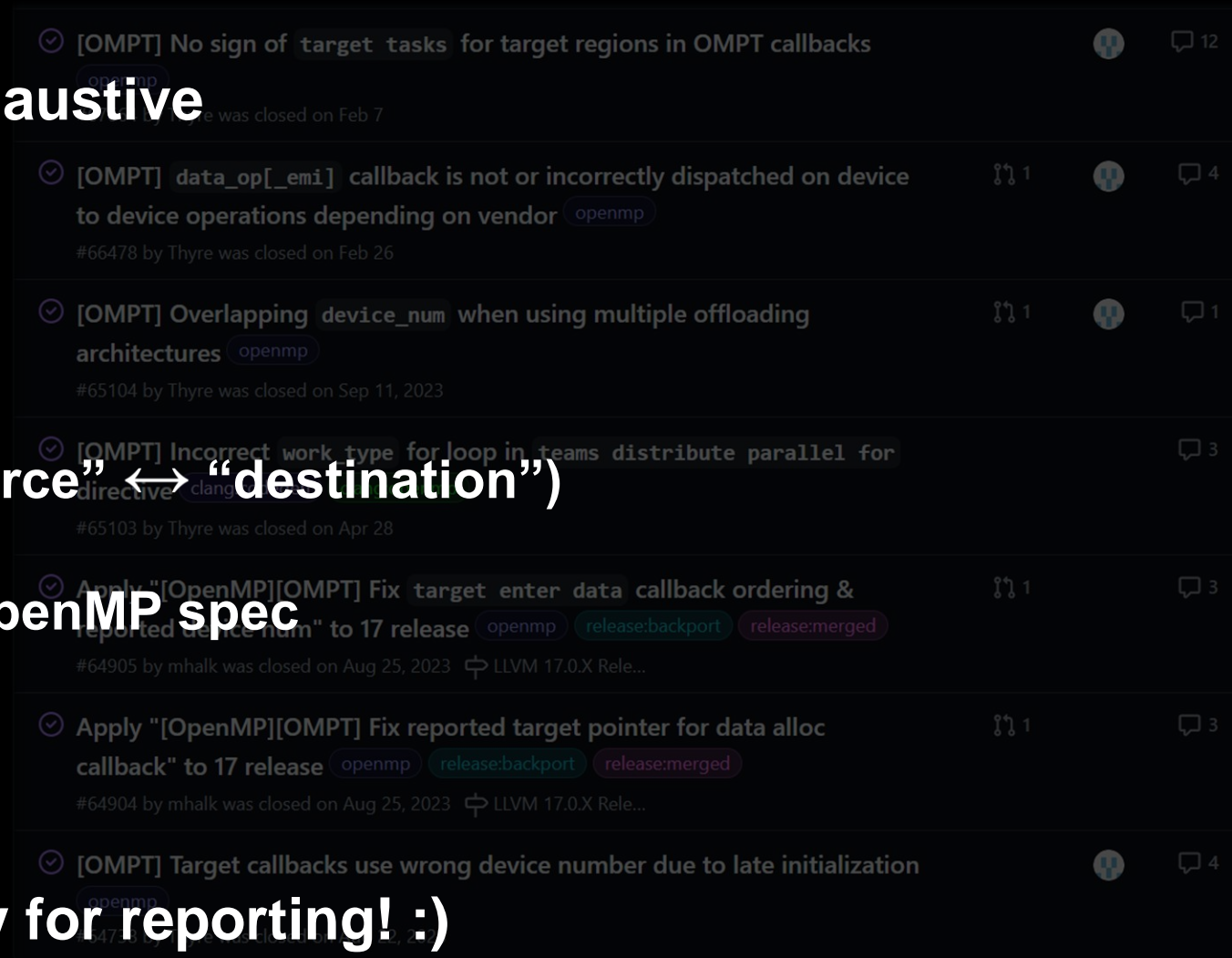
- API

- Callbacks

- Trace Records

Testing & Regressions

- **Test coverage almost never exhaustive**
 - **Regressions happen(ed)**
- **Regression examples**
 - **Missing EMI callback endpoints**
 - **Switched device numbers (“source” ↔ “destination”)**
 - **RAI object lifetimes**
 - **“Wrong” interpretation of the OpenMP spec**
 - **...**
- **Big *thank you* to the community for reporting! :)**



OMPT Testing

• Test scenarios rarely straightforward

• LLVM lit & FileCheck may prove cumbersome

• Key features

• Maintainability

”For each desired change, make the change easy (warning: this may be hard), then make the easy change.”

-- Kent Beck

• Robustness

”A test that fails randomly is worse than no test at all. It teaches you to distrust your tests.”

-- Uncle Bob (Robert C. Martin)

• Speed

”The faster you can run your tests, the more tests you will run.”

-- Kent Beck

```

1 #include <assert.h>
2 #include <omp.h>
3 #include <stdio.h>
4
5 #include "callbacks.h"
6
7 // Map of devices traced
8 DeviceMapPtr_t DeviceMapPtr;
9
10 int N = 100000;
11
12 int a[N];
13 int b[N];
14
15 int i;
16
17 for (i = 0; i < N; i++)
18   a[i] = 0;
19
20 for (i = 0; i < N; i++)
21   for (int j = 0; j < N; j++)
22     a[j] = b[j];
23
24 #pragma omp target teams distribute parallel for
25 {
26   for (int j = 0; j < N; j++)
27     a[j] = b[j];
28 }
29
30 #pragma omp target teams distribute parallel for
31 {
32   for (int j = 0; j < N; j++)
33     a[j] = b[j];
34 }
35
36 for (i = 0; i < N; i++)
37   if (a[i] != b[i]) {
38     rc++;
39     printf("Mismatch at %d\n", i);
40   }
41
42 if (!rc)
43   printf("Success\n");
44
45 return rc;
46
47 }
48

```

```

49 // clang-format off
50
51 // > OMPT device tracing related checks below. <
52
53 // Note: This test will allocate one buffer, big enough to hold all trace
54 // records, hence there will be only one allocation.
55
56 // CHECK-DAG: Allocated {{{[0-9]+}} bytes at [[ADDRX_01:0x[0-f]+]] in buffer request callback
57 // CHECK-DAG: Executing buffer complete callback: {{{[0-9]+}} [[ADDRX_01]] {{{[0-9]+}} [[ADDRX_01]] {{{[0-9]+}}}
58
59 // Note: Split checks for record address and content. That way we do not imply
60 // any order. Records may / will occur interleaved.
61 // CHECK-DAG: rec=[[ADDRX_01]]
62
63 // Note: These addresses will only occur once. They are only captured to
64 // indicate their existence.
65 // CHECK-DAG: rec=[[ADDRX_12:0x[0-f]+]]
66 // CHECK-DAG: rec=[[ADDRX_13:0x[0-f]+]]
67 // CHECK-DAG: rec=[[ADDRX_14:0x[0-f]+]]
68 // CHECK-DAG: rec=[[ADDRX_15:0x[0-f]+]]
69 // CHECK-DAG: rec=[[ADDRX_16:0x[0-f]+]]
70 // CHECK-DAG: rec=[[ADDRX_17:0x[0-f]+]]
71 // CHECK-DAG: rec=[[ADDRX_18:0x[0-f]+]]
72 // CHECK-DAG: rec=[[ADDRX_19:0x[0-f]+]]
73 // CHECK-DAG: rec=[[ADDRX_20:0x[0-f]+]]
74 // CHECK-DAG: rec=[[ADDRX_21:0x[0-f]+]]
75 // CHECK-DAG: rec=[[ADDRX_22:0x[0-f]+]]
76 // CHECK-DAG: rec=[[ADDRX_02:0x[0-f]+]]
77 // CHECK-DAG: rec=[[ADDRX_03:0x[0-f]+]]
78 // CHECK-DAG: rec=[[ADDRX_04:0x[0-f]+]]
79 // CHECK-DAG: rec=[[ADDRX_05:0x[0-f]+]]
80 // CHECK-DAG: rec=[[ADDRX_06:0x[0-f]+]]
81 // CHECK-DAG: rec=[[ADDRX_07:0x[0-f]+]]
82 // CHECK-DAG: rec=[[ADDRX_08:0x[0-f]+]]
83 // CHECK-DAG: rec=[[ADDRX_09:0x[0-f]+]]
84 // CHECK-DAG: rec=[[ADDRX_10:0x[0-f]+]]
85 // CHECK-DAG: rec=[[ADDRX_11:0x[0-f]+]]
86
87 // CHECK-DAG: type=8 (Target task) {{{.+.}} kind=1 endpoint=1
88 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
89 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
90 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
91 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
92 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=1
93 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
94 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
95 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
96 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
97 // CHECK-DAG: type=8 (Target task) {{{.+.}} kind=1 endpoint=2
98 // CHECK-DAG: type=8 (Target task) {{{.+.}} kind=1 endpoint=1
99 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
100 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
101 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
102 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
103 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=0
104 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
105 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
106 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
107 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
108 // CHECK-DAG: type=8 (Target task) {{{.+.}} kind=1 endpoint=2
109
110 // Note: This test will allocate one buffer, big enough to hold all trace
111 // records, hence there will be only one allocation.
112
113 // CHECK-NOT: rec=
114 // CHECK-NOT: host op id=0x0
115

```

ompTest

- **Goal: OMPT unit testing library**
“GoogleTest-like”
- **Own set of advantages**
 - **Unified event creation & verification**
 - **Improved readability & ease-of-use**
 - **Customized equality operators**
- **Limitations due to library-nature**
 - **OpenMP RT init / fini: out of scope**

```

1 #include <assert.h>
2 #include <omp.h>
3 #include <stdio.h>
4
5 #include "callbacks.h"
6
7 // Map of devices to use
8 DeviceMapPtr_t DeviceMapPtr;
9
10 int main() {
11     int N = 100000;
12
13     int a[N];
14     int b[N];
15
16     int i;
17
18     for (i = 0; i < N; i++)
19         a[i] = 0;
20
21     for (i = 0; i < N; i++)
22         b[i] = 1;
23
24     #pragma omp target parallel for
25     {
26         for (int j = 0; j < N; j++)
27             a[j] = b[j];
28     }
29
30 #pragma omp target teams distribute parallel for
31 {
32     for (int j = 0; j < N; j++)
33         a[j] = b[j];
34 }
35
36 int rc = 0;
37 for (i = 0; i < N; i++)
38     if (a[i] != b[i]) {
39         rc++;
40         printf("Wrong value: a[%d]=%d\n", i, a[i]);
41     }
42
43 if (rc)
44     printf("Success\n");
45
46 return rc;
47 }
48
49 // clang-format off
50
51 // > OMPT device tracing related checks below. <
52
53 // Note: This test will allocate one buffer, big enough to hold all trace
54 // records, hence there will be only one allocation.
55
56 // CHECK-DAG: Allocated {{{[0-9]+}}} bytes at {{{ADDRX_01:0x[0-f]+}}} in buffer request callback
57 // CHECK-DAG: Executing buffer complete callback: {{{[0-9]+}}} {{{ADDRX_01}}} {{{[0-9]+}}} {{{[0-9]+}}}
58
59 // Note: Callbacks are executed in record order and content. That way we do not imply
60 // that they occur interleaved.
61 // CHECK-DAG: rec=[ADDRX_01]
62
63 // Note: Callbacks will only occur once. They are only returned to
64 // the user once.
65 // CHECK-DAG: rec=[ADDRX_13:0x[0-f]+}]
66 // CHECK-DAG: rec=[ADDRX_14:0x[0-f]+}]
67 // CHECK-DAG: rec=[ADDRX_15:0x[0-f]+}]
68 // CHECK-DAG: rec=[ADDRX_16:0x[0-f]+}]
69 // CHECK-DAG: rec=[ADDRX_17:0x[0-f]+}]
70 // CHECK-DAG: rec=[ADDRX_18:0x[0-f]+}]
71 // CHECK-DAG: rec=[ADDRX_19:0x[0-f]+}]
72 // CHECK-DAG: rec=[ADDRX_20:0x[0-f]+}]
73 // CHECK-DAG: rec=[ADDRX_21:0x[0-f]+}]
74 // CHECK-DAG: rec=[ADDRX_22:0x[0-f]+}]
75 // CHECK-DAG: rec=[ADDRX_02:0x[0-f]+}]
76 // CHECK-DAG: rec=[ADDRX_03:0x[0-f]+}]
77 // CHECK-DAG: rec=[ADDRX_04:0x[0-f]+}]
78 // CHECK-DAG: rec=[ADDRX_05:0x[0-f]+}]
79 // CHECK-DAG: rec=[ADDRX_06:0x[0-f]+}]
80 // CHECK-DAG: rec=[ADDRX_07:0x[0-f]+}]
81 // CHECK-DAG: rec=[ADDRX_08:0x[0-f]+}]
82 // CHECK-DAG: rec=[ADDRX_09:0x[0-f]+}]
83 // CHECK-DAG: rec=[ADDRX_10:0x[0-f]+}]
84 // CHECK-DAG: rec=[ADDRX_11:0x[0-f]+}]
85 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=1
86 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
87 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
88 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
89 // CHECK-DAG: type=9 (Target kernel) {{{.+.}} optype=2
90 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=1
91 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=1
92 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
93 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
94 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
95 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
96 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
97 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=8
98 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
99 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
100 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
101 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
102 // Note: ADDR_X_01 may not trigger a final callback.
103 // Note: ADDR_X_01 may not be deallocated.
104 // CHECK-NOT: rec=
105 // CHECK-NOT: host_op_id=0x0
106
107 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
108 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
109
110
111
112
113 // CHECK-NOT: rec=
114 // CHECK-NOT: host_op_id=0x0
115

```

// Check for callback
OMPT_ASSERT_SET(TargetSubmit)

// Check for trace record
OMPT_ASSERT_SET(BufferRecord,
/*Type=*/CB_KERNEL)

#pragma omp target loop
for(int i = 0; i < N; ++i) {
Values[i] = 1;
}

ompTest

- **Related papers for SC'24**

- **Correctness Workshop**

<https://correctness-workshop.github.io/2024>

“ompTest – Unit Testing with OMPT”

- **HPC Bugs Fest**

<https://sites.google.com/view/hpc-bugs-fest/home>

“OMPTBench – OpenMP Tool Interface Conformance Testing”

```

1 #include <assert.h>
2 #include <omp.h>
3 #include <stdio.h>
4
5 #include "callbacks.h"
6
7 // Map of devices to use
8 DeviceMapPtr_t DeviceMapPtr;
9
10 int main() {
11     int N = 100000;
12     int a[N];
13     int b[N];
14
15     int i;
16
17     for (i = 0; i < N; i++)
18         a[i] = 0;
19
20     for (i = 0; i < N; i++)
21         b[i] = 1;
22
23     #pragma omp target parallel for
24     {
25         for (int j = 0; j < N; j++)
26             a[j] = b[j];
27     }
28
29     #pragma omp target teams distribute parallel for
30     {
31         for (int j = 0; j < N; j++)
32             a[j] = b[j];
33     }
34
35     int rc = 0;
36     for (i = 0; i < N; i++)
37         if (a[i] != b[i]) {
38             rc++;
39             printf("Wrong value: a[%d]=%d\n", i, a[i]);
40         }
41
42     if (!rc)
43         printf("Success\n");
44
45     return rc;
46 }

```

```

49 // clang-format off
50
51 // > OMPT device tracing related checks below. <
52
53 // Note: This test will allocate one buffer, big enough to hold all trace
54 // records, hence there will be only one allocation.
55
56 // CHECK-DAG: Allocated {{{[0-9]+}} bytes at [[ADDRX_01:0x[0-f]+]] in buffer request callback
57 // CHECK-DAG: Executing buffer complete callback: {{{[0-9]+}} [[ADDRX_01]] {{{[0-9]+}} [[ADDRX_01]] {{{[0-9]+}}
58
59 // Note: Callbacks are executed in parallel, hence the second lines and content. That way we do not imply
60 // that they occur interleaved.
61 // CHECK-DAG: rec=[ADDRX_01]
62
63 // Note: Callbacks will only occur once. They are only returned to
64 // the host once.
65 OMPT_ASSERT_SET(TargetSubmit)
66
67 // CHECK-DAG: rec=[ADDRX_13:0x[0-f]+]]
68 // CHECK-DAG: rec=[ADDRX_14:0x[0-f]+]]
69 // CHECK-DAG: rec=[ADDRX_15:0x[0-f]+]]
70 // CHECK-DAG: rec=[ADDRX_16:0x[0-f]+]]
71 // CHECK-DAG: rec=[ADDRX_17:0x[0-f]+]]
72 // CHECK-DAG: rec=[ADDRX_18:0x[0-f]+]]
73 // CHECK-DAG: rec=[ADDRX_19:0x[0-f]+]]
74 // CHECK-DAG: rec=[ADDRX_20:0x[0-f]+]]
75 // CHECK-DAG: rec=[ADDRX_21:0x[0-f]+]]
76 // CHECK-DAG: rec=[ADDRX_22:0x[0-f]+]]
77 // CHECK-DAG: rec=[ADDRX_02:0x[0-f]+]]
78 // CHECK-DAG: rec=[ADDRX_03:0x[0-f]+]]
79 // CHECK-DAG: rec=[ADDRX_04:0x[0-f]+]]
80 // CHECK-DAG: rec=[ADDRX_05:0x[0-f]+]]
81 // CHECK-DAG: rec=[ADDRX_06:0x[0-f]+]]
82 // CHECK-DAG: rec=[ADDRX_07:0x[0-f]+]]
83 // CHECK-DAG: rec=[ADDRX_08:0x[0-f]+]]
84 // CHECK-DAG: rec=[ADDRX_09:0x[0-f]+]]
85 // CHECK-DAG: rec=[ADDRX_10:0x[0-f]+]]
86 // CHECK-DAG: rec=[ADDRX_11:0x[0-f]+]]
87 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=1
88 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
89 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
90 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
91 // CHECK-DAG: type=9 (Target kernel) {{{.+.}} optype=2
92 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=1
93 // CHECK-DAG: type=9 (Target kernel) {{{.+.}} requested_num_teams=1
94 // CHECK-DAG: type=9 (Target kernel) {{{.+.}} requested_num_teams=1
95 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
96 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
97 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=1
98 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=1
99 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=2
100 // CHECK-DAG: type=10 (Target kernel) {{{.+.}} requested_num_teams=8
101 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
102 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=3
103 // CHECK-DAG: type=9 (Target data op) {{{.+.}} optype=4
104 // CHECK-DAG: type=9 (Target task) {{{.+.}} kind=1 endpoint=2
105
106 // Note: ADDR_X_01 may not trigger a final callback.
107 // Note: ADDR_X_01 may not be deallocated.
108
109 // CHECK-NOT: rec=
110 // CHECK-NOT: host op id=0x0
111
112
113
114
115

```

// Check for callback

OMPT_ASSERT_SET(TargetSubmit)

// Check for trace record

OMPT_ASSERT_SET(BufferRecord, /*Type=*/CB_KERNEL)

```

#pragma omp target loop
for(int i = 0; i < N; ++i) {
    Values[i] = 1;
}

```


Possible Future of OMPT

- Device-side tracing support (upstream)
- “ompTest” library (upstream)
- OMPT “native” data type

```
libomptarget_ompt_set_trace_ompt(/*DeviceId=*/i...
/*Enable=*/1,
/*EventTy=*/0)

OMPT_ASSERT_SET_GROUPED("R1", TargetEM1, /*Kind=*/TARGET,
/*Endpoint=*/BEGIN)

OMPT_ASSERT_SET_GROUPED("R1", BufferRecord, /*Type=*/CB_TARGET,
/*Kind=*/TARGET,
/*Endpoint=*/BEGIN)
OMPT_ASSERT_SET_GROUPED("R1", BufferRecord, /*Type=*/CB_DATAOP,
/*OpType=*/ALLOC, /*Bytes=*/sizeof(a),
/*MinimumTimeDelta=*/10)
OMPT_ASSERT_SET_GROUPED("R1", BufferRecord, /*Type=*/CB_DATAOP,
/*OpType=*/H2D, /*Bytes=*/sizeof(a),
/*Timeframe=*/{10, 100000})
OMPT_ASSERT_SET_GROUPED("R1", BufferRecord, /*Type=*/CB_DATAOP,
/*OpType=*/ALLOC, /*Bytes=*/sizeof(b),
/*MinimumTimeDelta=*/10)
OMPT_ASSERT_SET_GROUPED("R1", BufferRecord, /*Type=*/CB_DATAOP,
/*OpType=*/H2D, /*Bytes=*/sizeof(b),
/*Timeframe=*/{10, 100000})
```

__omp_offloading_10302_2f...

Arguments

args

BeginNs	45599437350144
CompleteNs	45600446427683
DispatchNs	45599437324087
DurationNs	1008975804
EndNs	45600446325948
KernelName	__omp_offloading_10302_2fe0429_main_l58.kd
queue-id	0
queue-index	11
omp-id	1
pid	15184



Disclaimer

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS.' AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

© 2024 Advanced Micro Devices, Inc. All rights reserved.

AMD, the AMD Arrow logo, EPYC, Instinct and combinations thereof are trademarks of Advanced Micro Devices, Inc. PCIe is a registered trademark of PCI-SIG Corporation. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

AMD 