OpenMP

SC'21 Booth Talk Series

SC21

St. Louis, MO | science & beyond.

OvO: Systematically Testing a Subset of OpenMP Offload

Thomas Applencourt,
Argonne National Laboratory

Table of contents

Introduction

Goals of This Presentation

- OvO: Functional Testing of a subset of OpenMP Offload
- Highlight the systematic testing approach used
- Encourage you to use this tool to identify bugs in your compiler
- Possibly learn some OpenMP Offload subtlety

What I will not show in this talk:

- Not enough time to discuss the pass rate of various compilers. Don't want this talk to be a quick "Name and Shame". Just try by yourself :)

What is OvO?

- An OpenMP Offload Mathematical and Hierarchical Parallelism Functional Test Suite (*https://github.com/TApplencourt/OvO*)
- C++ and FOR$_{\text{mula}}$TRAN$_{\text{slation}}$)[1]
- Scripts to compile, run, and check correctness
- Found numerous[2] bugs in all compilers tested[3]

Due time constrain, I will only talk about Hierarchical Parallelism today

[1] More precisely C++11 and F90 standards. Some math function are defined in C++17 and C++20 standards
[2] ≥ 1
[3] GNU, LLVM, OneAPI, HPE Cray Programming Environment, NVIDIA HPC Software Development Kit

Why OvO?

1. OpenMP specification is extensive
2. Compilers only support a subset of it

For application developers:

- Check if a required feature is supported by a majority of compilers

For compiler developers:

- Check if a required feature is supported by your compiler

Comparison To Similar Projects

- DOE ECP SOLLVE V&V (*https://github.com/SOLLVE/sollve_vv*)[4]
- IBM OpenmpTest (*https://github.com/clang-ykt/omptests*)

1. OvO focuses on subsets of the OpenMP 5.1 specification: hierarchical parallelism and math functions for OpenMP Offload
2. Tests coverage is exhaustive

[4] See really nice presentation in Last year Bof
https://www.openmp.org/wp-content/uploads/SC2020_SOLLVE_VV_Pophale.pdf

Hierarchical Parallelism

What is Hierarchical Parallelism?

- OpenMP supports nesting of regions[5]
- Used extensively in offload regions

```
#pragma omp target map(to: pS[0:size]) map(from: pD[0:N0*N1*N2])
#pragma omp teams distribute
for (int i0 = 0 ; i0 < N0 ; i0++ )
{
  #pragma omp parallel for
  for (int i1 = 0 ; i1 < N1 ; i1++ )
  {
    #pragma omp simd
    for (int i2 = 0 ; i2 < N2 ; i2++ )
    {
      const int idx = i2+N2*(i1+N1*(i0));
      pD[idx] = pS[idx];
    }
  }
}
```

[5] 2.22 Nesting of Regions:
https://www.openmp.org/spec-html/5.1/openmpse30.html#x155-1880002.22

There Are Only Six Directives-So Test Them All!

1. Idea comes from "There are Only Four Billion Floats–So Test Them All!" from Bruce Dawson [6]
2. Only 6 OpenMP Offload directives `target`,`teams`,`distribute`,`parallel`,`for`,`simd`, so test them all!

[6] *https://randomascii.wordpress.com/2014/01/27/theres-only-four-billion-floatsso-test-them-all/*

How Many tests?

- Six directives `target`, `teams`, `distribute`, `parallel`, `for`, `simd`
- In maximum 76 Combinations are conforming[7]

Conforming:

```
1 !$OMP TARGET TEAMS DISTRIBUTE
2 !$OMP PARALLEL
3 !$OMP SIMD
```

Non Conforming:

```
1 !$OMP TARGET TEAMS
2 !$OMP DISTRIBUTE PARALLEL DO SIMD
```

Some OpenMP directives have restrictions on how they can be mapped[8], OvO tests-generator takes care of it.

[7] For more information on the algorithm see *https://github.com/TApplencourt/OvO/blob/dc65a087f5abb3ff2164d41f425933982d15eff7/src/gtest.py#L97*

[8] For example atomic cannot be nested in a teams region

Scope and Design Choices

1. Three categories of tests: memory copy, reduction and atomic
2. Multiple datatype: float, `double<complex>`, ...
3. For both C++ and Fortran

Tests should be:

1. Self-contained
2. Return 0 on success and $\neq 0$ on failure[9]
3. Independent of the number of teams/threads

[9] Test which produce the wrong value will return 112, tests which hang will return 124

Example ”OpenMP Offload 101 Memcopy”

```
1  PROGRAM target_teams_distribute_parallel_do_simd
2    implicit none
3    INTEGER :: N0 = 32768
4    INTEGER :: i0
5    INTEGER :: idx, S
6    INTEGER :: errno = 0
7    REAL, ALLOCATABLE :: src(:)
8    REAL, ALLOCATABLE :: dst(:)
9    S = N0
10   ALLOCATE(dst(S), src(S) )
11   CALL RANDOM_NUMBER(src)
12   !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD map(to: src) map(from: dst) PRIVATE(IDX)
13   DO i0 = 1, N0
14     idx = i0-1+1
15     dst(idx) = src(idx)
16   END DO
17   IF (ANY(ABS(dst - src) > EPSILON(src))) THEN
18     WRITE(*,*) 'Wrong value', MAXVAL(ABS(DST-SRC)), 'max difference'
19     errno = 112
20   ENDIF
21   DEALLOCATE(src, dst)
22   IF (errno .EQ. 112) STOP 112
23 END PROGRAM target_teams_distribute_parallel_do_simd
```

Side-way OpenMP subtlety 1: Scalar Attribute

```
1  !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD map(to: src) map(from: dst)
2  DO i0 = 1, N0
3    idx = i0
4    dst(idx) = src(idx)
5  END DO
```

idx is firsprivate.

```
1  !$OMP TARGET map(to: src) map(from: dst)
2  !$OMP TEAMS DISTRIBUTE PARALLEL DO SIMD
3  DO i0 = 1, N0
4    idx = i0
5    dst(idx) = src(idx)
6  END DO
```

But here, in line 4 *idx* is considered as a shared variable.
You probably want to add **PRIVATE**(*idx*) in the *DISTRIBUTE* construct line to avoid race condition.

Testing for Atomic and Reduction: Cuda Style

1. Tests are designed so that the theoretical results of the reduction are known *a priori*.
2. Result is independent of the number of threads and teams.

```
float counter_parallel{};
#pragma omp target parallel reduction(+: counter_parallel)
{
  counter_parallel = counter_parallel + 1. / omp_get_num_threads() };
}
//assert(counter_parallel==1)
```

```
float counter_parallel{};
#pragma omp target parallel map(tofrom: counter_parallel)
{
  #pragma omp atomic update
  counter_parallel = counter_parallel + 1. / omp_get_num_threads() };
}
//assert(counter_parallel==1)
```

Floating Point Arithmetic: Problem

```
1  float counter_parallel{};
2  #pragma omp target parallel reduction(+: counter_parallel)
3  {
4    counter_parallel = counter_parallel + 1. / omp_get_num_threads() };
5  }
```

1. For mathematician this function always return 1.
2. In ”the real world”, due to Floating Point Arithmetic error will accumulate[10]

[10] In this example, the error is even unbounded

Side-way on Floating Point Arithmetic: Solution

Solution[11]:

1. Specify the maximum number of threads and teams[12]
2. Compute the maximum theoretical deviation
3. Compare results within this tolerance

```
bool almost_equal(float x, float gold, float rel_tol) {
  return std::abs(x-gold) <= rel_tol * std::max(std::abs(x), std::abs(gold));
}
void test_target_simd() {
  float counter_parallel{};
  #pragma omp parallel num_threads(32768) reduction(+: counter_parallel)
  {
    counter_parallel = counter_parallel + 1.f/ omp_get_num_threads();
  }
  assert(almost_equal(counter_N0, 1., 0.01));
}
```

[11] Thanks a lot to Pablo Oliveira for his expertise

https://github.com/TApplencourt/OvO/blob/master/documentation/FP_error.md

[12] In OpenMP, as far as I know, it's not possible to set a number of threads, hence set maximum born

Side-way OpenMP subtlety 2: Mapping and Reduction

```
1  #pragma omp target map(tofrom: counter_teams)
2  #pragma omp teams reduction(+: counter_teams)
3  {
4    counter_teams = counter_teams + 1.f/ omp_get_num_teams();
5  }
```

Need the `map:tofrom`. Scalar are by default passed as `map:to`.

```
1  #pragma omp target teams reduction(+: counter_teams)
```

When used in a target-combined construct, will be `map:tofrom` by default[13]

[13] See Data-Mapping Attribute *https://www.openmp.org/spec-html/5.1/openmpsu119.html#x154-1830002.21.7*

Advanced Tests: Host Threaded + Atomic

OvO is easily extended to generate more intricate tests[14]

```
1  const float expected_value { N0*N1*N2 };
2  float counter_N0{};
3  #pragma omp parallel for
4  for (int i0 = 0 ; i0 < N0 ; i0++ ) {
5    #pragma omp target teams map(tofrom: counter_N0)
6    {
7      #pragma omp parallel for
8      for (int i1 = 0 ; i1 < N1 ; i1++ ) {
9        #pragma omp simd
10       for (int i2 = 0 ; i2 < N2 ; i2++ ) {
11         #pragma omp atomic update
12         counter_N0 = counter_N0 + float{ 1. } / omp_get_num_teams();
13       }
14     }
15   }
16 }
```

[14] Reference counting is "performed as a single atomic operation". See *https://www.openmp.org/spec-html/5.1/openmpsu119.html#x154-1830002.21.7*

Summary Hierarchical Parallelism

- Test all possible combination of offload directives
- Clang has been reactive to fix all the bugs, clang 14.0.0 63c566b1f has 100% pass rate for basic tests.
  - A lot of bugs are still lurking in the wild. SIMD has generally been hard for compilers

Conclusion

Conclusion

```
$ git clone git@github.com:TApplencourt/OvO.git ; cd OvO
# Set your env and run
$ CXX="foo" CXXFLAGS="-fopenmp" FC="bar" FFLAGS="-fopenmp" ./ovo.sh run
[...]
# Display a nice reports
$ ./ovo.sh report
>> Overall result test_result/1957-04-01_19-02_CDC6600.lanl.gov
pass rate(%)  test(#)  success(#)  comp error(#)  run error(#)  wrong value(#)  timeout(#)
------------  ------   ----------  -------------  ------------  --------------  ----------
         68%     910          618            152            46              82          12
```

1. Testing systematically all combinations of nested offload directives and Math functions was feasible.
2. Bugs were discovered (and fixed) in all compilers tested
3. OpenMP specification has some subtleties![15]
4. Please use OvO and report important-to-you bugs to your vendor!

[15] Be careful or enjoy them depending of your mentality

OpenMP

SC'21 Booth Talk Series

openmp.org OpenMP API specs, forum, reference guides, and more

link.openmp.org/sc21 Videos and PDFs of OpenMP SC'21 presentations

Backup: Mathematical Functions

Design principle

1. Test all math functions for one arbitrary input.
2. Test if the GPU function math gives the same result as the CPU version.

OpenMP doesn't specify any accuracy requirements. Hence took 4 ulp arbitrary, but it's configurable[16]

[16] Corresponding to low precision accuracy of MKL function *https://software.intel.com/content/www/us/en/develop/documentation/onemkl-vmperfdata/top.html*

Code Example

```
1  bool almost_equal(float x, float y, int ulp) {
2       return std::fabs(x-y) <= std::numeric_limits<float>::epsilon() * std::fabs(x+y) * ulp ||
3              std::fabs(x-y) < std::numeric_limits<float>::min();
4  }
5  void test_ceil(){
6     const char* usr_precision = getenv("OVO_TOL_ULP");
7     const int precision = usr_precision ? atoi(usr_precision) : 4;
8     float in0 { 0.42 };
9     float out1_host {};
10    float out1_device {};
11    out1_host = ceil(in0);
12
13    #pragma omp target map(tofrom: out1_device )
14    {
15     out1_device = ceil(in0);
16    }
17    if ( !almost_equal(out1_host,out1_device, precision) ) {
18         std::cerr << std::setprecision (std::numeric_limits<float>::max_digits10 )
19         << "Host: " << out1_host << " GPU: " << out1_device << std::endl;
20         std::exit(112);
21     }
22 }
```

Math Conclusion

- Not a thorough testing, more a ”at least it can offload and give correct results sometimes”
- Compilers have various support for math functions
  - ”is_nan” was/is hard for a lot of compilers.
  - Math ICE provided me with one of my favorite error message ”compilation completed with severe errors”