A Microbenchmark Framework for Performance Evaluation of OpenMP Target Offload

Mohammad Atif‡, Tianle Wang, Zhihua Dong, Meifeng Lin
Computational Science Initiative, Brookhaven National Laboratory, NY
‡ fmohammad@bnl.gov

Abstract

We present a framework based on Catch2 to evaluate performance of OpenMP's target offload model via micro-benchmarks. The compilers supporting OpenMP's target offload model for heterogeneous architectures are currently undergoing rapid development. These developments influence performance of various physics applications in different ways. This framework can be employed to track the impact of compiler upgrades and compare their performance with the native programming models. We use the framework to benchmark performance of a few commonly used operations on leadership class supercomputers such as Perlmutter at National Energy Research Scientific Computing (NERSC) Center and Frontier at Oak Ridge Leadership Computing Facility (OLCF). Such a framework will be useful for compiler developers to gain insights into the overall impact of many small changes, as well as for users to decide which compilers and versions are expected to yield best performance for their applications.

Keywords: performance portability, OpenMP target offload, CUDA, HIP, micro-benchmarks, Catch2

OpenMP Target Offload

- Compiler directive-based programming model for shared memory parallelization
- OpenMP 4.0 and above extend support for parallel execution on heterogeneous architectures via the `target offload` model
- An important candidate for portable programming model on heterogeneous architectures
- Architecture agnostic compiler directives can offload to multiple GPU, FPGAs, or use CPU threads
- Easy to use, does not require major changes to `C++` code, interoperable with other programming models, has good support from build systems
- Performance is heavily dependent on compilers and appropriate flags
  - LLVM Clang and GCC are community-developed
  - NVIDIA's nvc++, AMD's amd-clang, AOMP, AFAR, and Intel's icpx are vendor-developed
- We needed a framework to track improvements in the compilers and compare their performance with native programming models.

CUDA vs OpenMP APIs

```
cudaMalloc (**devicePointer, size)
devicePointer = omp_target_alloc (size, deviceID)
```

```
cudaMemcpy (dest, src, count, cudaMemcpyHostToDevice )
omp_target_memcpy (dest, src, count, dest_offset, src_offset,
dst_dev_id, src_dev_id )
```

```
cudaFree (devicePointer)
omp_target_free (devicePointer, deviceID)
```

```
#pragma omp target is_device_ptr ( devicePointer ) map ( )
#pragma omp teams distribute parallel for num_threads
(BLOCK_SIZE)
for ( . . . ; . . . ; . . . ) {
...
#pragma omp atomic
...
}
```

Benchmarking with Catch2

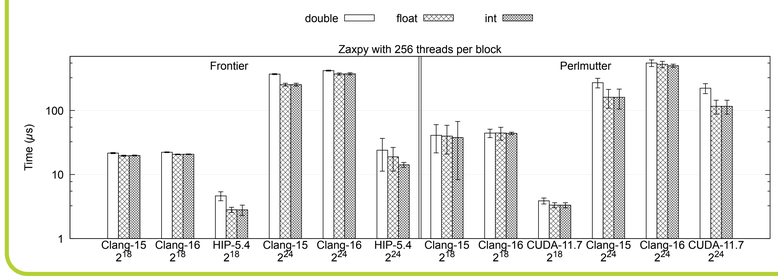
- Identifying and evaluating performance of microbenchmarks has the potential to affect many applications at once
- `Catch2` is primarily a unit-testing framework for `C++` applications which has incorporated benchmarking to its features

```
BENCHMARK ("initialize array") {
return array_init (device_array_ptr, array_size,
block_size, num_blocks);
} ;
```

```
allocate_x_y_z_host_device ();
BENCHMARK_ADVANCED ("zaxpy")
(Catch::Benchmark::Chronometer meter ) {
initialize_x_y_z_copy_host_to_device ();
meter.measure ( [x, y, z, a, array_size,
block_size, num_blocks]
{ return zaxpy (x, y, z, a, array_size, block_size,
num_blocks); } ) ;
copy_z_device_to_host ();
};
```

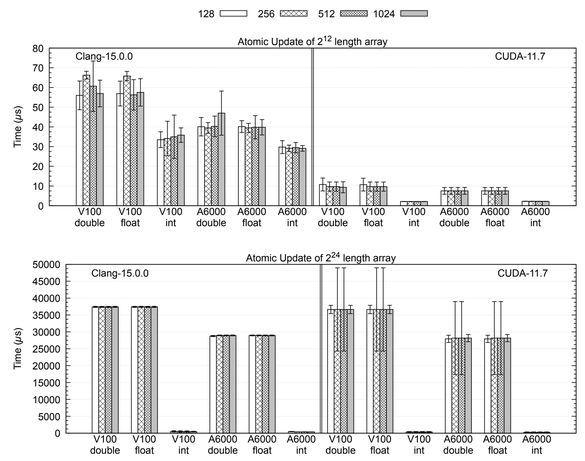
- Macros calculate the number of runs in each sample depending on their execution time.
- The final statistics is calculated with an option for statistical bootstrapping.
- Several useful command line options for samples, resamples, confidence interval, and warmup-time to quantify statistics.

$z = a \times x + y$



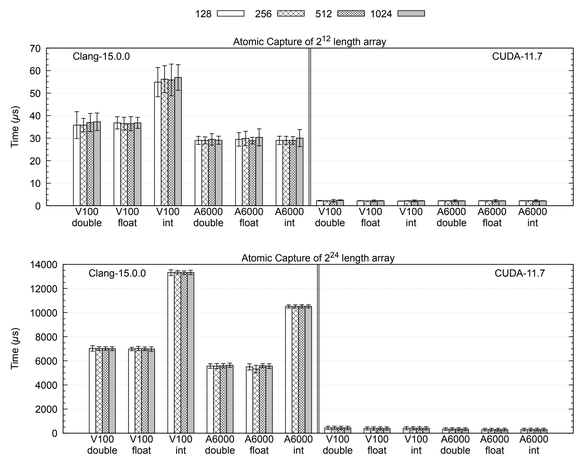
- For array size 2^{24} performance of zaxpy kernel has deteriorated from Clang-15 to Clang-16.
- Slowdown of Clang with respect to HIP is consistent for small and large arrays, whereas with respect to CUDA is milder for large array.

Atomic Update



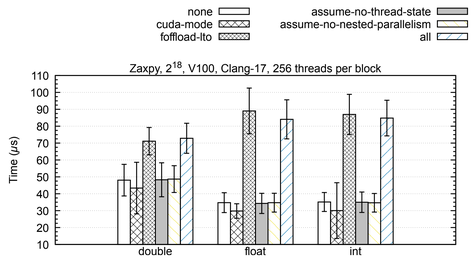
- CUDA is much faster than Clang for smaller array size, however, it becomes comparable for a larger problem.
- The above trend is consistent across different Nvidia GPUs.
- Integer data type is optimized better when array size is increased.

Atomic Capture



- CUDA is much faster than Clang for smaller array as well as the larger problem, shows little variation with data types
- With the exception of small array on A6000, integer data types are the slowest for Clang.

Impact of Compiler Flags



Future Work

- Extend the list of microbenchmarks to more atomic and other commonly used operations
- Compare performance of other vendor and community developed compilers
- Managed memory
- A SYCL port of benchmarks to compare performance on the Intel GPUs.
- Compare overheads with other prortable programming models like Kokkos/RAJA.

Acknowledgement

Work supported by US Department of Energy, Office of Science, Office of High Energy Physics under the High Energy Physics Center for Computational Excellence (HEP-CCE), a collaboration between Argonne National Laboratory, Brookhaven National Laboratory, Fermilab and Lawrence Berkeley National Laboratory.

References

[1] Catch2. https://github.com/catchorg/Catch2. Accessed: 2023-07-24.
[2] OpenMP Benchmarks. https://github.com/BNL-HPC/openmp-benchmarks. Accessed: 2023-08-10.

U.S. DEPARTMENT OF ENERGY

www.bnl.gov

Brookhaven National Laboratory