# Porting ATLAS FastCaloSim to GPUs with OpenMP Target Offloading

Mohammad Atif[1], Zhihua Dong[1], Charles Leggett[2], Meifeng Lin[1], Tianle Wang[1]

[1]Brookhaven National Laboratory, Upton, NY 11973, USA
[2]Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

High Energy Physics - Center for Computational Excellence

https://www.anl.gov/hep-cce

## Abstract

OpenMP is a directive based shared-memory parallel programming model traditionally used for multicore CPUs. In its recent versions, OpenMP was extended to enable GPU computing via its "target offloading" model. The architecture agnostic compiler directives can in principle offload to multiple types of GPUs and FPGAs, and its compiler support is under active development.

We investigate the performance of OpenMP's GPU offloading capability by porting the ATLAS FastCaloSim code. FastCaloSim is a relatively self-contained parametrized calorimeter simulation, and is used as a testbed for our investigations of different portable programming models. We find the OpenMP GPU offloading easy to implement and that it does not require major changes to the C++ code. However, the performance varies from compiler to compiler and the specialized operations (e.g. atomic) are currently less performant than CUDA. We compare the performance with the existing CUDA port across hardware (NVIDIA, AMD) and compilers (LLVM Clang, AMD Clang, gcc, nvc++).

## FastCaloSim Overview

- Random Numbers
  - Generate on GPU (cuRAND/ rocRAND)
  - Generate on CPU, copy to GPU
- Load Geometry
- Simulate Hits: 3 parallelizable kernels with thread local flops
  - reset (*for* loop over $\sim 187,000$ cells)
  - simulate (*for* loop over $\sim 5,000 - 6,000$ hits)
  - reduce (*for* loop over $\sim 187,000$ cells)
- Copy energy, hit counts from device to host

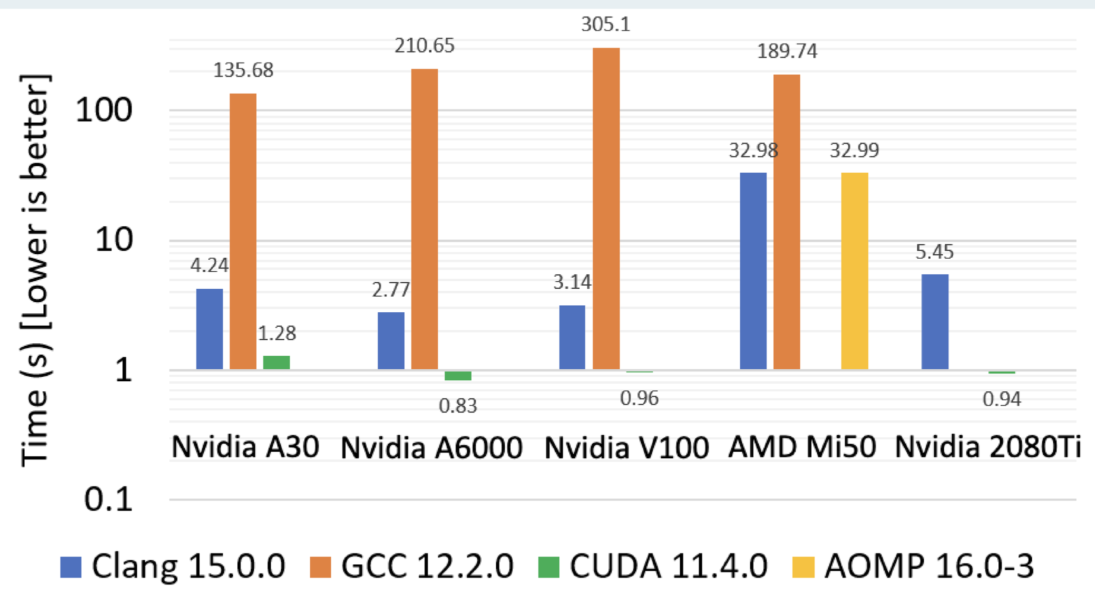


Figure 1: Runtimes of kernels and data copy for 65 GeV using various hardware and compilers.

## CUDA vs OpenMP Target Offloading 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)
                   num_teams(GRID_SIZE)
for ( ; ; ) {
...
      #pragma omp atomic
...
}
```

## Lessons Learned

- Important to tune number of threads per team (block size), default values did not yield the best performance
- Use flags `-fopenmp-cuda-mode`, `-foffload-lto`, `-fopenmp-assume-no-thread-state`, `-fopenmp-assume-no-nested-parallelism` whenever possible
- Use `OMP_TARGET_OFFLOAD=mandatory`
- LLVM Clang's environment variables `LIBOMPTARGET_INFO` can help with debugging
- LLVM Clang's optimization remarks such as `Rpass=openmp-opt`, `-Rpass-analysis=openmp-opt`, `-Rpass-missed=openmp-opt` offer insights to gain performance
- Nsight Systems creates a larger overhead for profiling OpenMP target offloads

## Experiences

- Easy to implement, does not require major changes to the C++ code
- Performance varies from compiler to compiler
- Specialized operations (e.g. atomic) less performant than CUDA
- Does not support GPU scan, memset operations
- Under active development
- Architecture agnostic compiler directives can offload to multiple GPUs, FPGAs

## Performance compared to CUDA

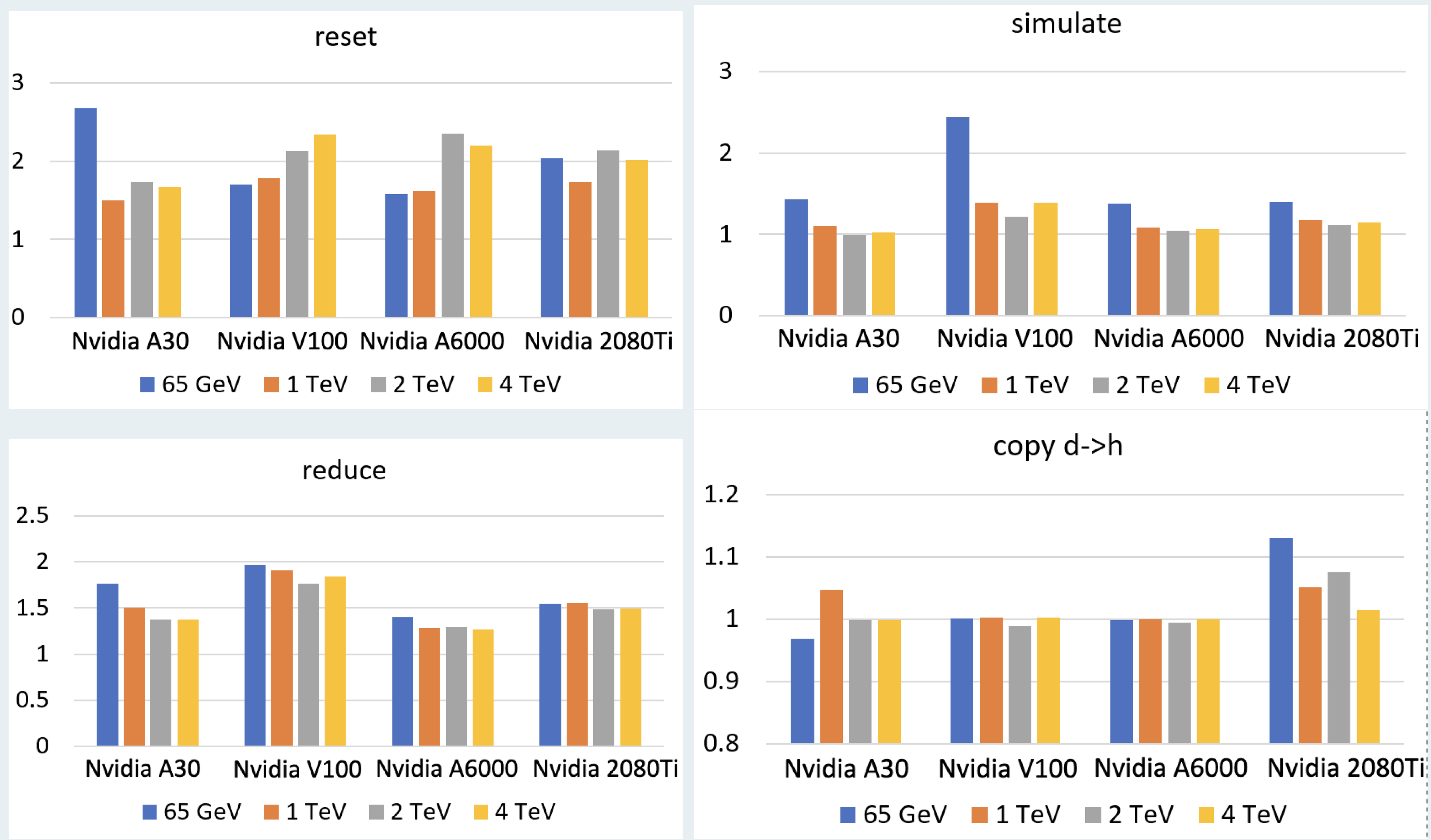


Figure 2: Slowdown of various kernels for group simulations using LLVM Clang 15.0.0 for different hardware.

## Status of Compilers and GPUs for OpenMP Offloading

| | Nvidia | | AMD | |
|---|---|---|---|---|
| | **Standalone saxpy** | **FastCaloSim** | **Standalone saxpy** | **FastCaloSim** |
| **Clang-15.0.0** | Working | Working | Working | Working |
| **NVC++-22.9** | Working | Runtime error | | |
| **GCC-12.2** | Working | Working | Working | Working |
| **AOMP-16.0-3** | | | Working | Working |

## Future Work

- Profiling tools for AMD GPUs (Rocprofiler, omnitrace)
- Portable Random Number Generator (Do check out Tianle Wang's poster)
- Investigate performance of Clang-16.0.0 and Intel compiler+GPUs

## References

[1] ATLAS Collaboration. The simulation principle and performance of the ATLAS fast calorimeter simulation FastCaloSim. No. ATL-PHYS-PUB-2010-013. ATL-COM-PHYS-2010-838 (2010).

[2] Dong Z, Gray H, Leggett C, Lin M, Pascuzzi VR and Yu K. Porting HEP Parameterized Calorimeter Simulation Code to GPUs. Front. Big Data 4:665783. doi: 10.3389/fdata.2021.665783 (2021).

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.

CHEP-2023,
5/8/2023-5/12/2023