# OpenMP on Aurora

Colleen Bertoni, Ye Luo
Argonne Leadership Computing Facility

# Agenda

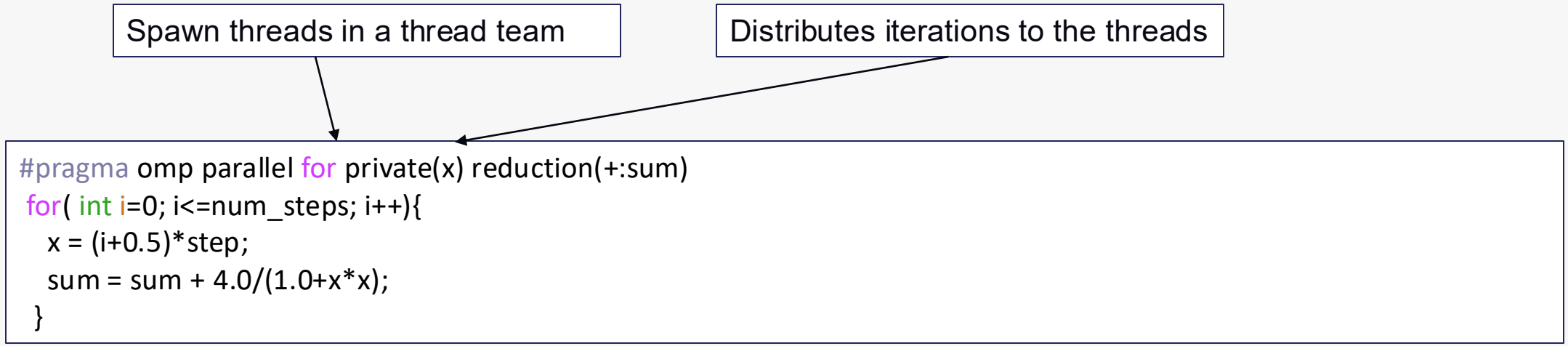
- Using OpenMP on Aurora (~10 min)
  - Why OpenMP?
  - Using GPUs with OpenMP Offload
- Demo of OpenMP (~20 min)
  - OpenMP 101 and basics on Aurora
- Efficient OpenMP code patterns (~20 min)
- Q&A (~10 min)

# Why OpenMP?

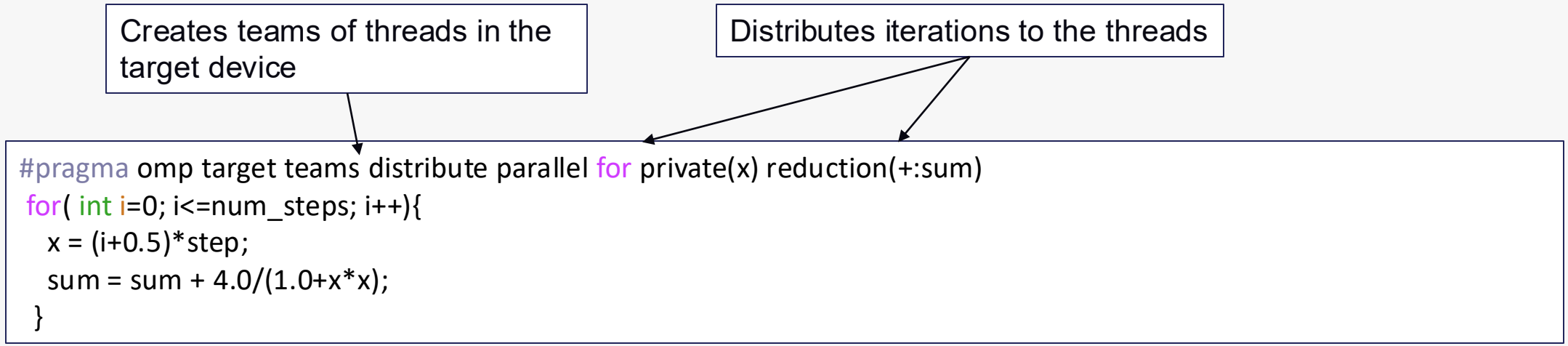
  - Open standard for parallel programming with support across vendors
  - OpenMP runs on CPU threads, GPUs, SIMD units
  - C/C++ and Fortran
  - Supported by Intel, Cray, GNU, LLVM compilers and others
  - OpenMP offload will be additionally supported on Aurora, Frontier, Perlmutter
- Four Important high-level features to express parallelism
  - Fork and join thread parallelism
  - SIMD parallelism (added in 4.0)
  - Device Offload parallelism (added in 4.0)
  - Tasking parallelism (added in 3.0)
- Why instead of a C++ framework?
  - Easy to get started and trivial to parallelize loops
  - The reduction clause simplifies data reduction

# Using GPUs with OpenMP Offload

# CPU OpenMP parallelism

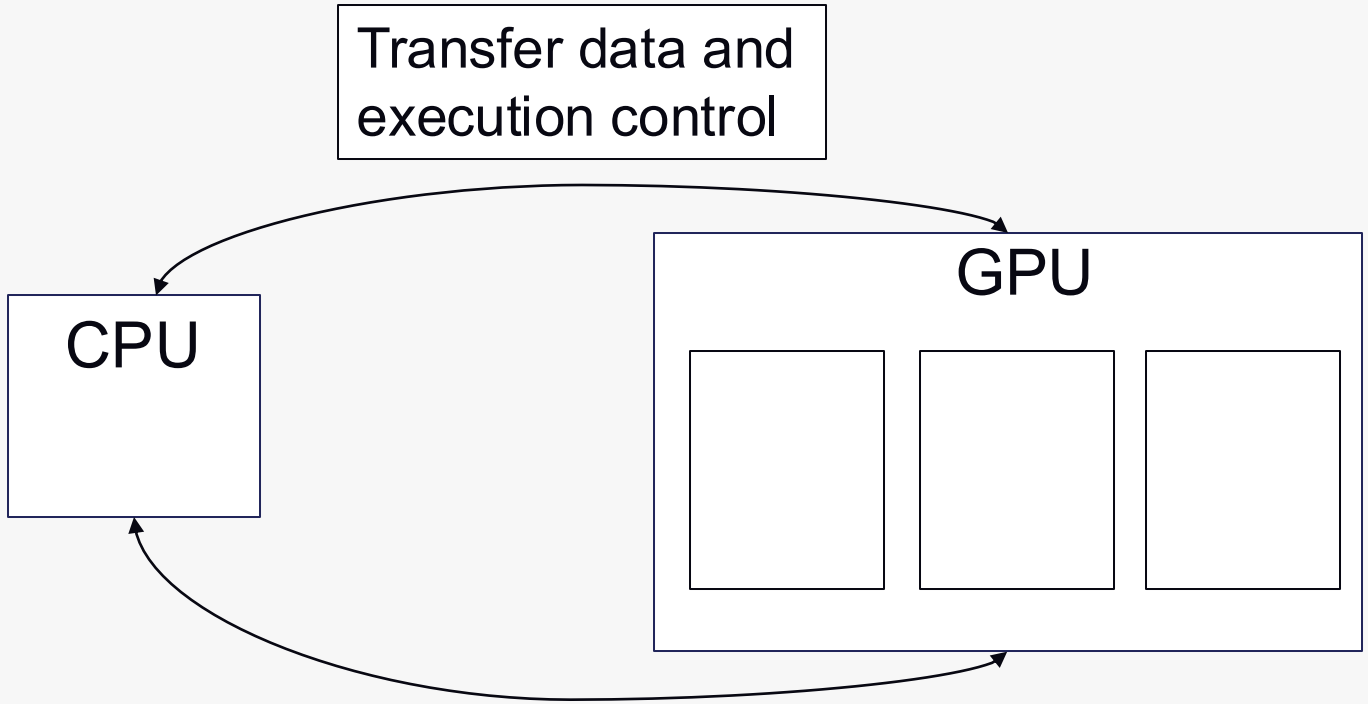


# GPU OpenMP parallelism



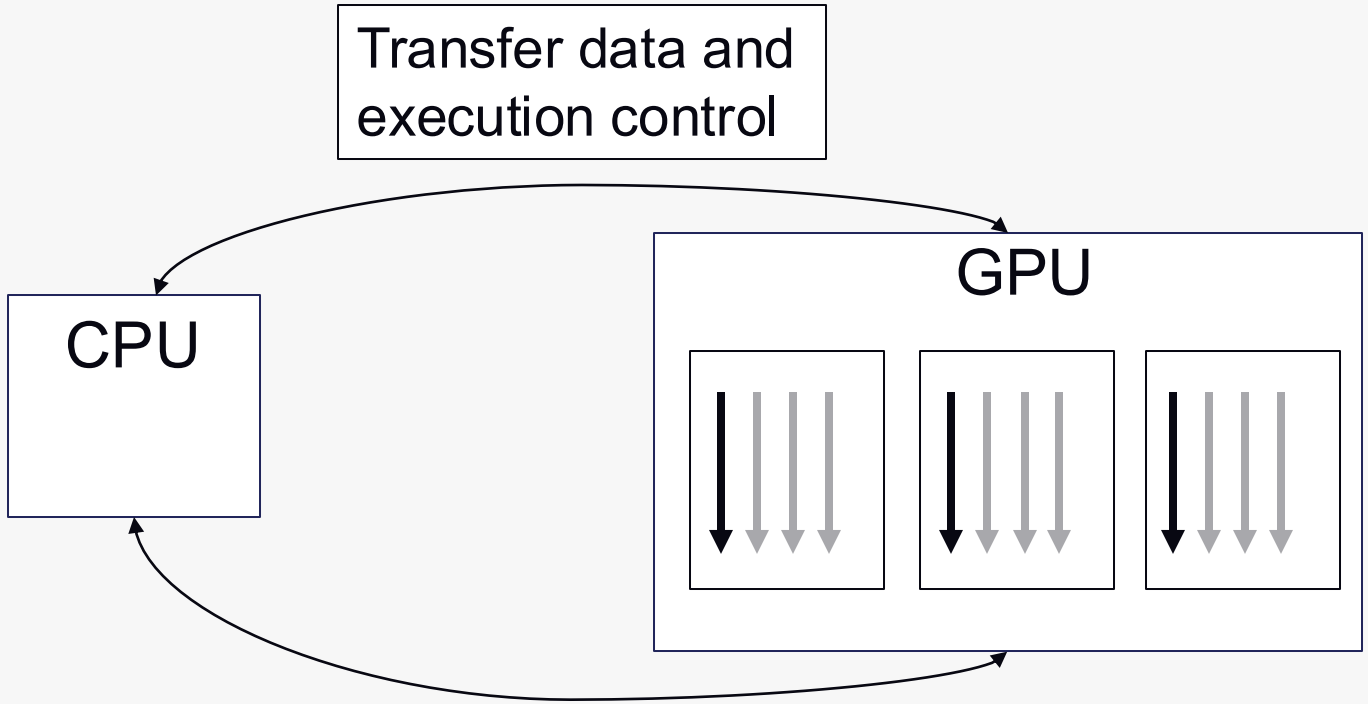
# OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device



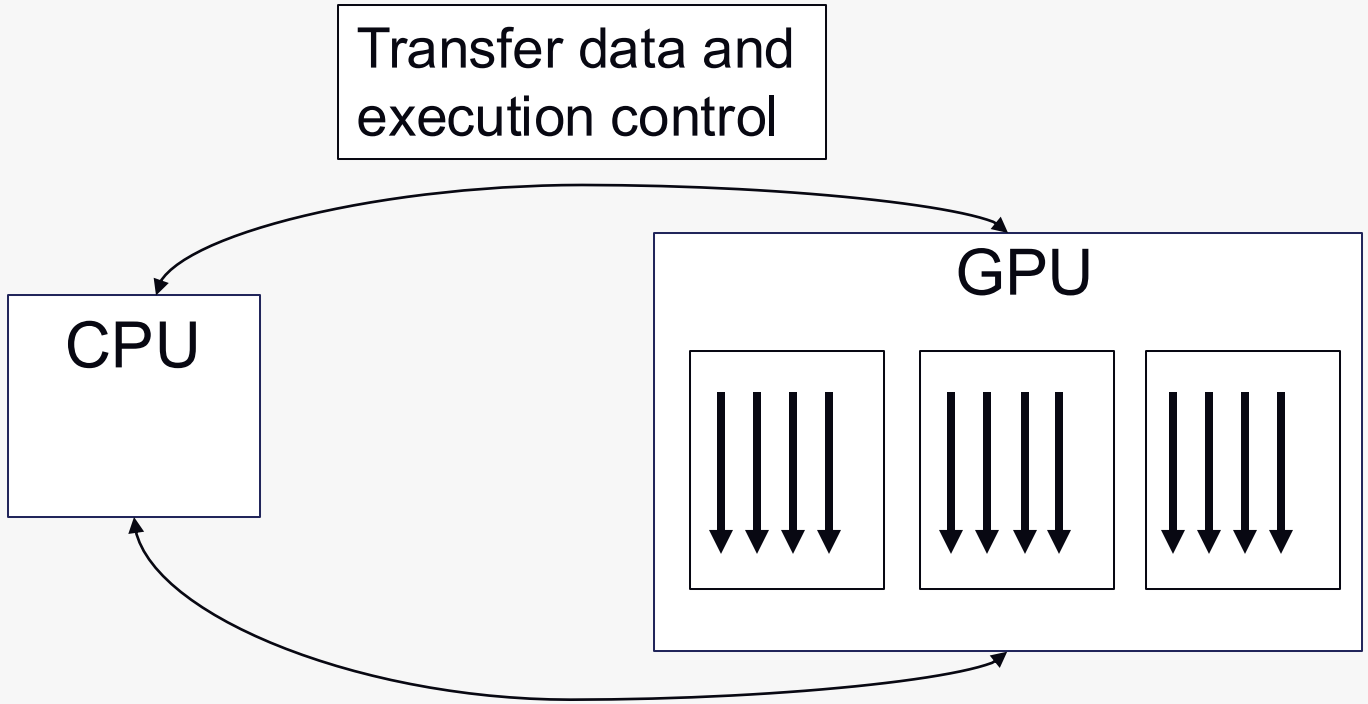
# OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)

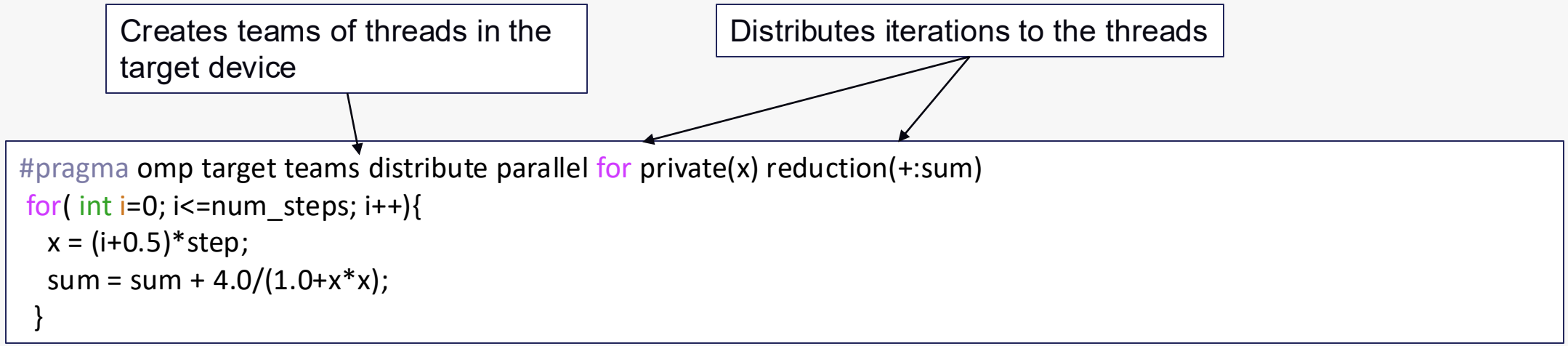


# OpenMP Offload Introduction

- **Target construct**: offloads code and data to the device and runs in serial on the device
- **Teams construct**: creates a league of teams, each with one thread, which run concurrently on SMs (Nvidia terminology)
- **Parallel construct**: creates multiple threads in the teams, each which can run concurrently



# GPU OpenMP parallelism



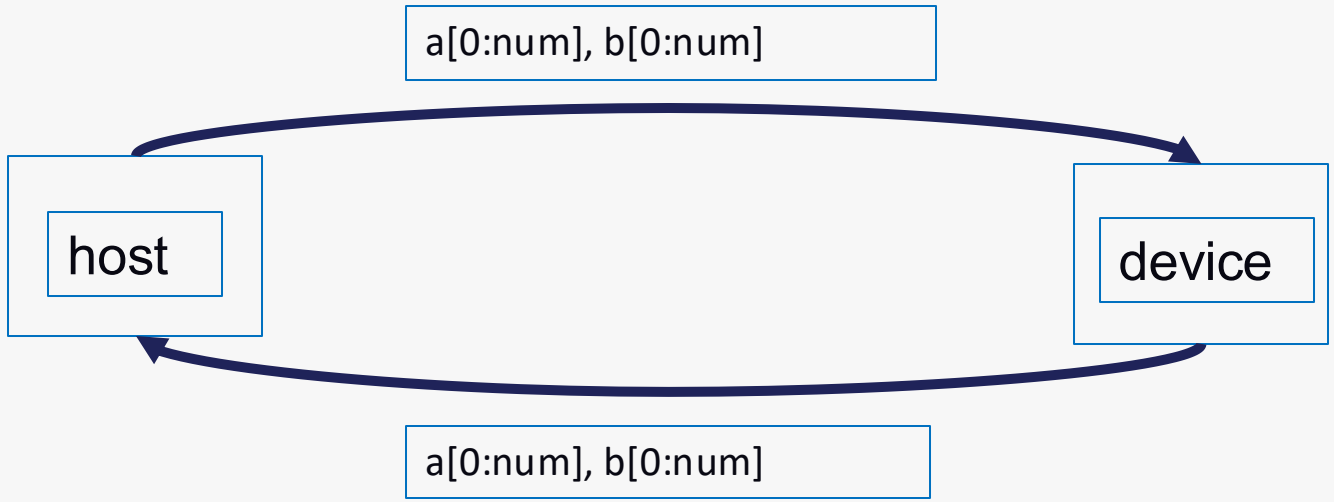
# OpenMP and data transfer

```
…

#pragma omp target teams distribute parallel for map(tofrom:a[0:num], b[0:num])
    for (size_t j=0; j<num; j++) {
      a[j] = a[j]+scalar*b[j];

    }
…

…
```




- Maps *a* and *b* to and from the device.
- These are shared and accessible by all of the threads on the GPU.

# OpenMP offload compilers and flags on Aurora

| Language | MPI Wrapper Compiler (Underlying Compiler) | Flag to Turn on OpenMP Support and Target CPU Threads | Additional Flags to Target GPU Devices |
|---|---|---|---|
| Fortran | mpifort (ifx) | -fiopenmp | -fopenmp-targets=spir64 |
| C | mpicc (icx) | -fiopenmp | -fopenmp-targets=spir64 |
| C++ | mpicxx (icpx) | -fiopenmp | -fopenmp-targets=spir64 |

- Intel OpenMP offload compilers are in the default environment on Aurora
- You can swap “-fopenmp-targets=spir64” for “-fopenmp-targets=spir64_gen -Xs "-device pvc” ” for AOT compilation
- https://docs.alcf.anl.gov/aurora/programming-models/openmp-aurora/

# OpenMP on Aurora: Functionality Benchmarks

- OpenMP vs. Offload: https://github.com/TApplencourt/OvO
- OpenMP Validation and Verification: https://crpl.cis.udel.edu/ompvv/project/
- Some of the tests are for uncommon OpenMP directives, but it gives a general sense that both implementations are generally passing
- (Part of an upcoming paper in IWOMP2025)

| | C/C++ | | Fortran | |
|---|---|---|---|---|
| | **Intel (2025.0) on Aurora** | **Nvidia (23.9) on Polaris** | **Intel (2025.0) on Aurora** | **Nvidia (23.9) on Polaris** |
| OvO | 100% (521/521) | 70% (367/521) | 100% (389/389) | 92% (359/389) |
| OMPVV-4.5 | 96% (141/147) | 89% (131/147) | 95% (99/104) | 93% (97/104) |
| OMPVV-5.0 | 77% (164/213) | 35%(75/213) | 66% (85/128) | 27% (35/128) |
| OMPVV-5.1 | 75% (76/101) | 12%(12/101) | 60% (17/28 ) | 7% (2/28) |
| OMPVV-5.2 | 13% (3/24) | 25%(6/24) | 80% (4/5) | 60% (3/5) |
| OMPVV-6.0 | 22% (5/22) | 4%(1/22) | 0% (0/1) | 0% (0/1) |

# 101 Demo for GPUs

```
$ git clone https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop
$ cd ALCF_Hands_on_HPC_Workshop/programmingModels/OpenMP
$ cd demo
```