Performance Portability of Shallow Water Model with Kokkos

Zephaniah Connell and Leila Ghaffari

Mentors: Supreeth Suresh, Cena Miller, Jian Sun, and John Dennis

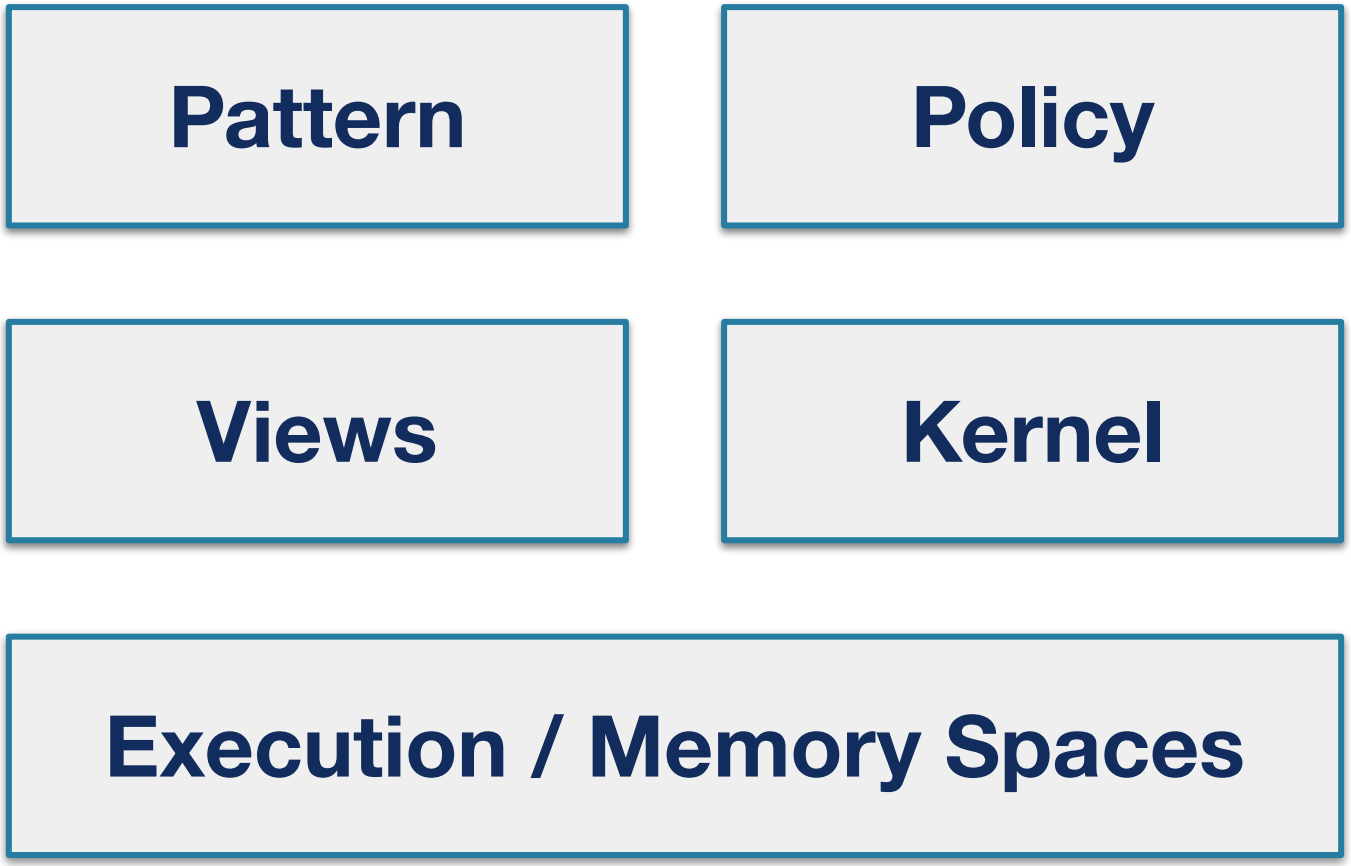
NCAR
UCAR

SIParCS 2021
July 27, 2021

What is Kokkos?

- **C++ library that can be used to write parallel code**
- **Performance Portable** - Architecture Aware
- **Architectures**:
  - Nvidia, AMD, and Intel GPUs
  - x86, Power 8, KNL, and ARM CPUs
- **Compilers**:
  - GNU 5.3.0 or newer
  - Intel 17.0.1 or newer
  - Clang 4.0.0 or newer
  - PGI 18.7 or newer
  - CUDA 9.1 or newer

How does Kokkos work?



Pattern	Parallel structure

Kokkos	parallel_for(), parallel_reduce(), parallel_scan()
CUDA	Explicitly defined in kernel

Kokkos

```
Kokkos::parallel_for("init_stream_func_vals",
  Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0,0},{m+1,n+1}),
  KOKKOS_LAMBDA(const int i, const int j){
    int ii = i*(n+2)+j;
    psi(ii) = a * sin((i + .5) * di) * sin((j + .5) * dj);});
```

Serial

```
int ii;
for (int i=0; i<m+1; i++) {
  for (int j=0; j<n+1; j++) {
    ii = i*(n+2)+j;
    psi[ii] = a * sin((i + .5) * di) * sin((j + .5) * dj);}}
```

Policy	Index space

Kokkos	Integer, MDRangePolicy, TeamPolicy
CUDA	Grid, Block

Kokkos

```
Kokkos::parallel_for("init_stream_func_vals",
  Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0,0},{m+1,n+1}),
  KOKKOS_LAMBDA(const int i, const int j){
    int ii = i*(n+2)+j;
    psi(ii) = a * sin((i + .5) * di) * sin((j + .5) * dj);});
```

Serial

```
int ii;
for (int i=0; i<m+1; i++) {
  for (int j=0; j<n+1; j++) {
    ii = i*(n+2)+j;
    psi[ii] = a * sin((i + .5) * di) * sin((j + .5) * dj);}}
```

Views

Multi-dimensional data class

Kokkos	View, mirrorView, DualView, unmanaged View
CUDA	void **

Kokkos

```
Kokkos::View<double*> psi("psi",DOMAIN_SIZE);
Kokkos::parallel_for("init_stream_func_vals",
  Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0,0},{m+1,n+1}),
  KOKKOS_LAMBDA(const int i, const int j){
    int ii = i*(n+2)+j;
    psi(ii) = a * sin((i + .5) * di) * sin((j + .5) * dj);});
```

Serial

```
double psi[DOMAIN_SIZE];
int ii;
for (int i=0; i<m+1; i++) {
  for (int j=0; j<n+1; j++) {
    ii = i*(n+2)+j;
    psi[ii] = a * sin((i + .5) * di) * sin((j + .5) * dj);}}
```

Kernel	Work performed on each index

Kokkos	Functor, Lambda
CUDA	__global__, __device__

Kokkos

```
Kokkos::parallel_for("init_stream_func_vals",
  Kokkos::MDRangePolicy<Kokkos::Rank<2>>({0,0},{m+1,n+1}),
  KOKKOS_LAMBDA(const int i, const int j){
    int ii = i*(n+2)+j;
    psi(ii) = a * sin((i + .5) * di) * sin((j + .5) * dj);});
```

Serial

```
int ii;
for (int i=0; i<m+1; i++) {
  for (int j=0; j<n+1; j++) {
    ii = i*(n+2)+j;
    psi[ii] = a * sin((i + .5) * di) * sin((j + .5) * dj);}}
```

Execution / Memory Spaces

Memory location, execution hardware, and execution method

	Kokkos	CUDA
Memory Location	HostSpace CudaSpace CudaUVMSpace CudaHostPinnedSpace	cudaHostAlloc(...) cudaMalloc(...) cudaMallocManaged(...) cudaHostAlloc(...)
Execution Hardware/ Method	Serial OpenMP Cuda	__host__ --- __global__, __device__

Debugging and Challenges

Kokkos Installation -

Hello World

Building, Installing, and Compiling for Different Architectures

Functors vs Lambdas -

Functors - More difficult | More feature-rich

Lambdas - Easier | More readable | Limited

Views -

Many Options - View + Mirror View, DualView, Unmanaged View

Element Access - syntax differences

Subviews - Function parameter issues

Complexity

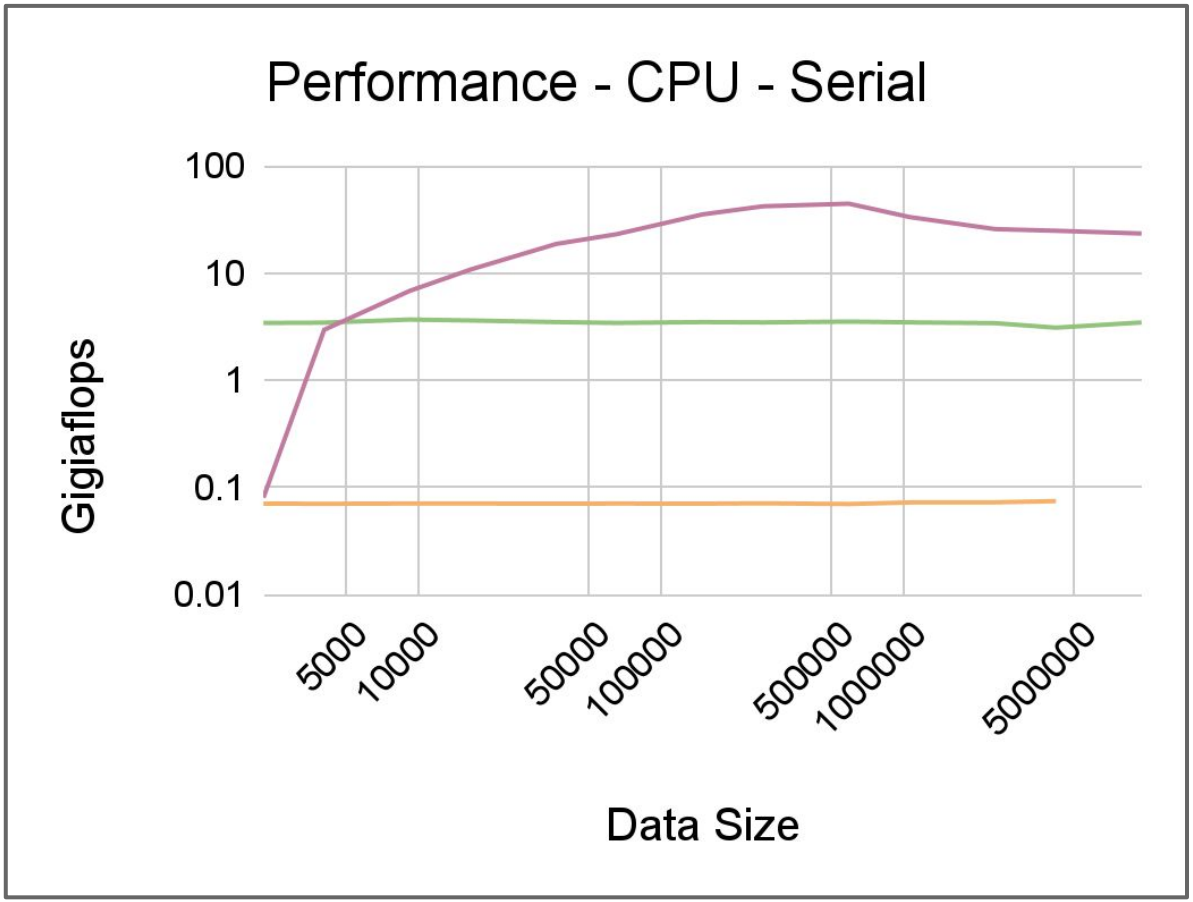
Original Total Lines - 544

	Additional Lines		Modified Lines	
	#	%	#	%
OpenMP	14	2.6%	2	0.4%
OpenACC	28	5.1%	2	0.4%
Kokkos	50	9.2%	114	21.0%
DPC++	90	16.5%	62	11.4%

Performance - CPU - Serial

~50x slower than C++

- **C++** Skylake 1-Core (gnu/8.3.0 -O2)
- **Kokkos** Skylake 1-Core (gnu/8.3.0 -O2)
- **DPC++** Skylake 1-Core (dpcpp -O2)

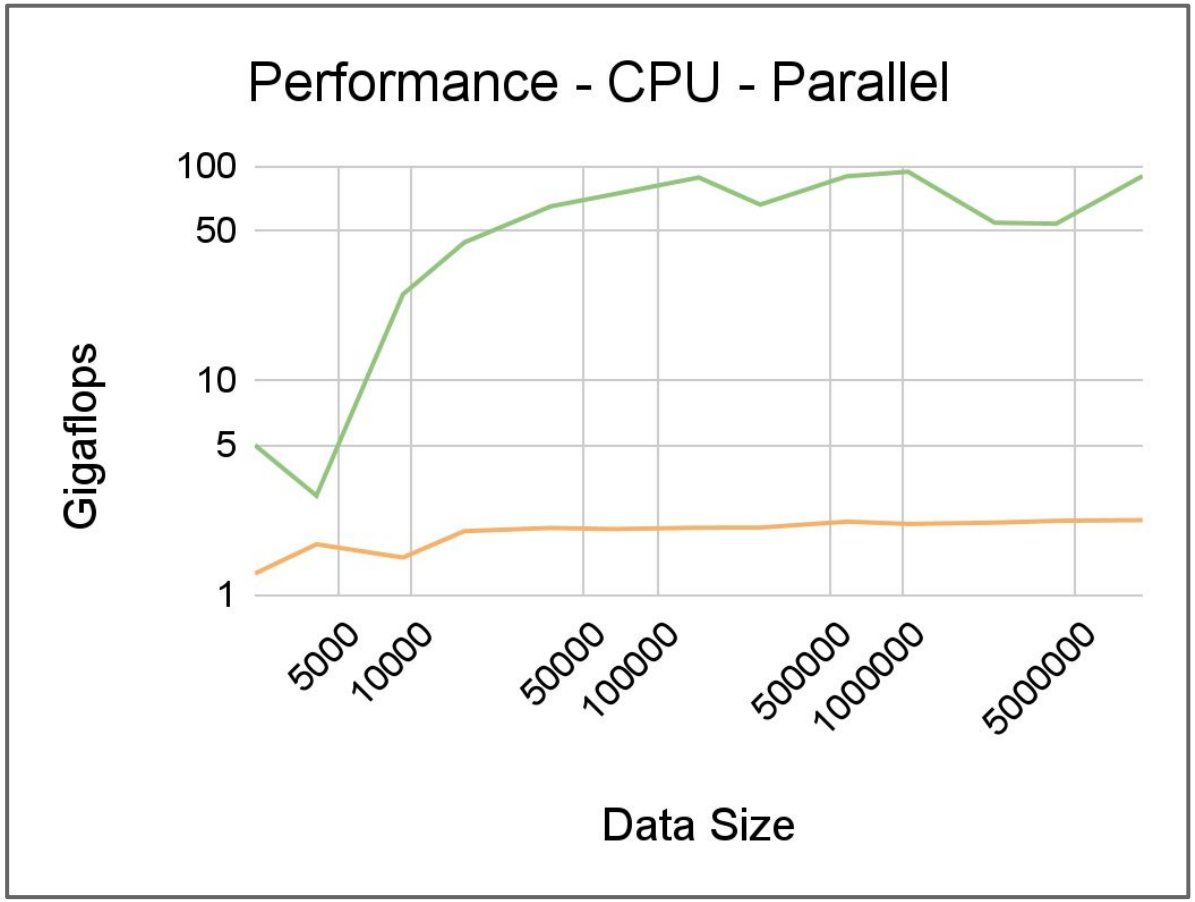


Performance - CPU - Parallel

~44x slower than OpenMP

and

~1.67x slower than C++ Serial



C++ Skylake 36-Core (gnu/8.3.0 -O2)

Kokkos Skylake 36-Core (gnu/8.3.0 -O2)

Performance - GPU - Parallel

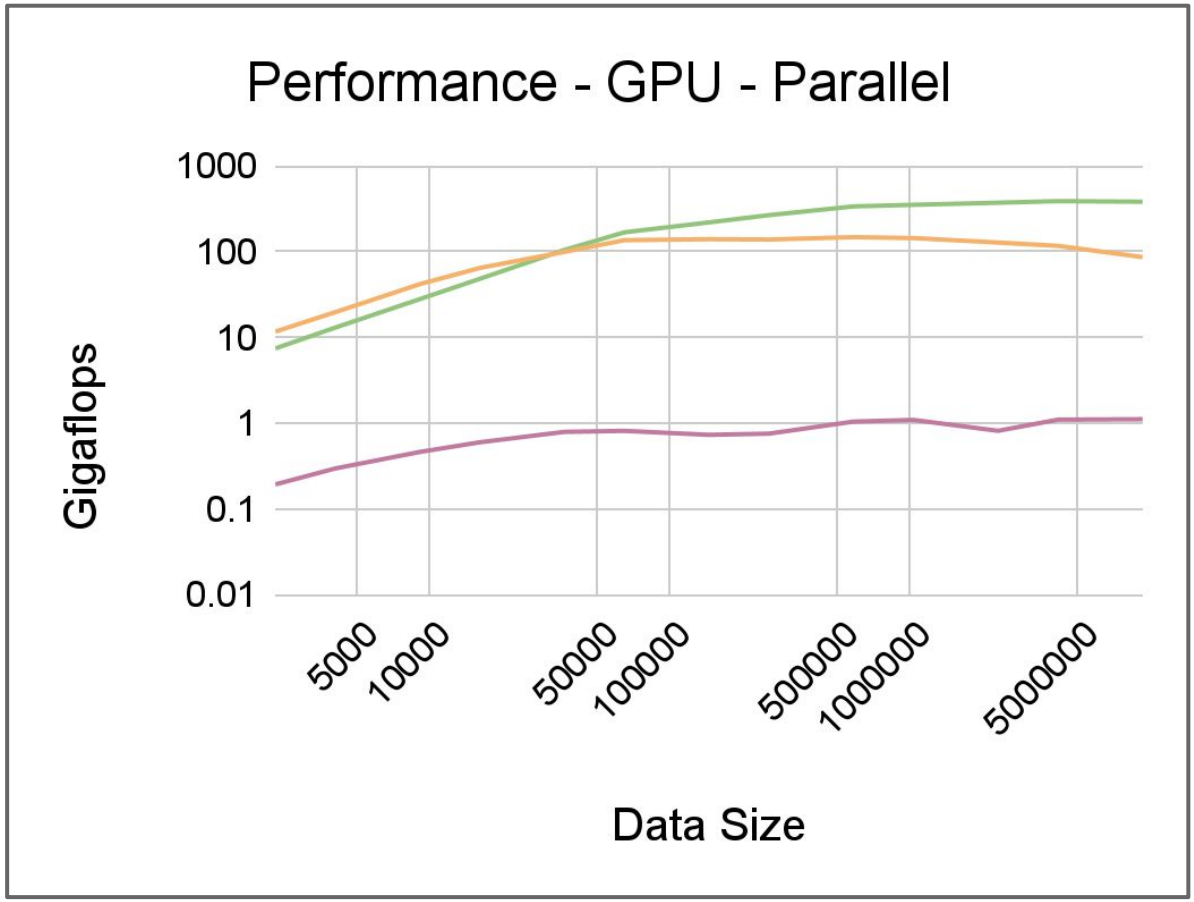
Best performance:

~1.6x faster than OpenACC
and
~43x faster than C++ Serial

Worst performance:

~4.4x slower than OpenACC
and
~3.4x faster than C++ Serial

- **OpenACC** V100 (nvhpc/21.3 cuda/10.1 -O2)
- **Kokkos** V100 (gnu/8.3.0 cuda/10.1 -O2)
- **DPC++** Iris Xe MAX (dpcpp -O2)



Profiling

Kokkos/kokkos-tools - Timing and Memory Usage

Connectors

- caliper
- nvprof
- papi
- systemtap
- timemory
- vtune

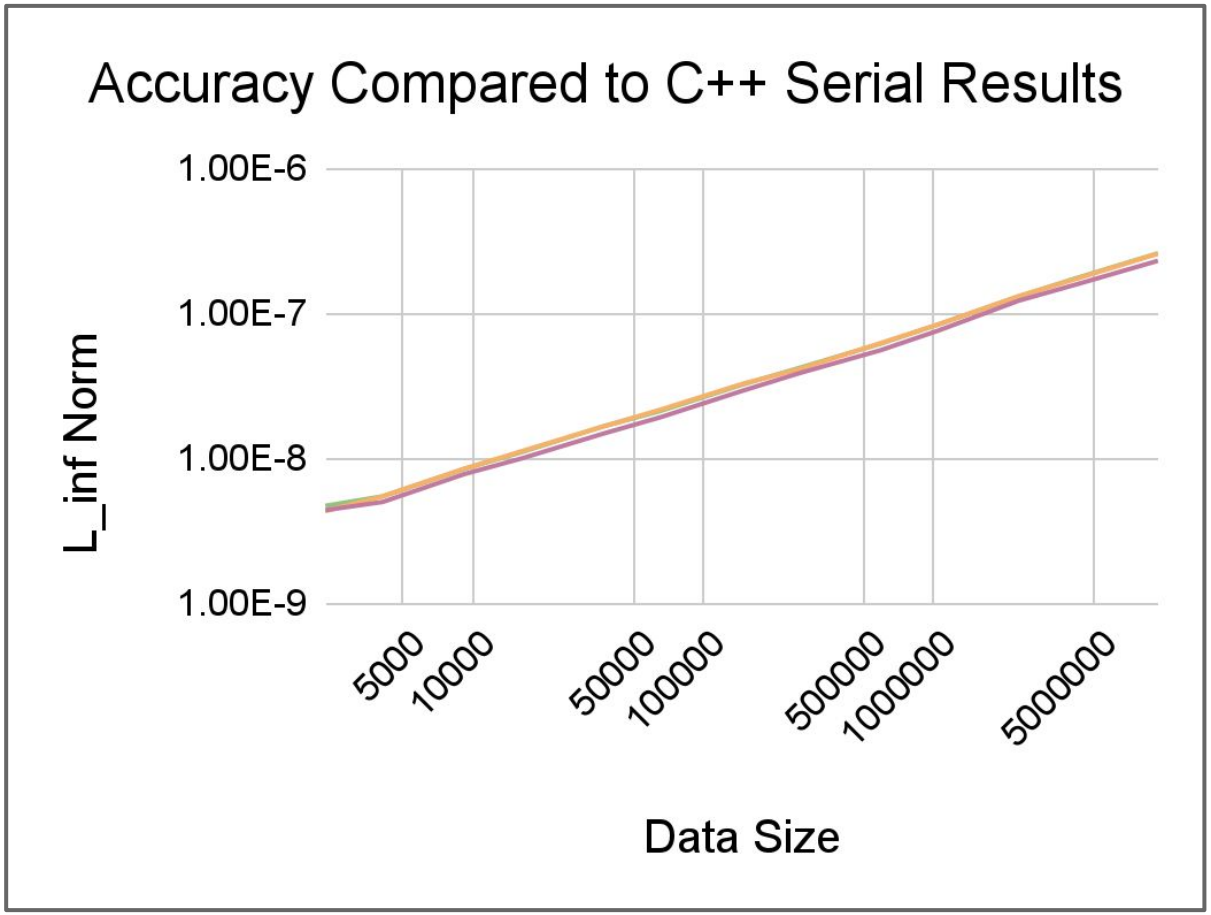
Simple Kernel Timer

- ❑ clone
- ❑ make
- ❑ set KOKKOS_PROFILE_LIBRARY
- ❑ execute

Accuracy

For C++ Serial, OpenMP, Kokkos CPU Serial, and Kokkos CPU Parallel:
L_inf Norm = 0

- **OpenACC** V100 (nvhpc/21.3 cuda/10.1 -O2)
- **Kokkos** V100 (gnu/8.3.0 cuda/10.1 -O2)
- **DPC++** Iris Xe MAX (dpcpp -O2)



Conclusions

Portability	●●●●
Time / Difficulty	●●●●
CPU Performance	●
GPU Performance	●●●
Documentation	●●●
Support	●●●●●

In my opinion, for any project that may benefit from executing code on different GPU architectures, **Kokkos is worthwhile**.

Future Work

- Intel and AMD GPUs
- Deeper profiling exploration
- Unexplored optimization -
    - Explicit memory layouts
    - SWM data refactor
    - Vectorization
    - TeamPolicy
    - Intel and newest compilers
- Multi-node / multiple GPUs

References

Carter Edwards, H., Trott, C. R., & Sunderland, D. (2014). Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, *74*(12), 3202–3216. https://doi.org/10.1016/j.jpdc.2014.07.003

For More Information

DPC++:

https://wiki.ucar.edu/display/~leilag/SIParCS+2021+-+Project+9

Leila.Ghaffari@colorado.edu

Kokkos:

https://wiki.ucar.edu/display/~zconnell/SIParCS+2021+-+Performance+Portability+of+Weather+and+Climate+Modeling+Mini+Apps+-+Kokkos

zephconnell@gmail.com