Performance Portability of Shallow Water Model with DPC++

Leila Ghaffari and Zephaniah Connell

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

- Increase in the computational capacity of high-performance computing platforms
- GPUs could save energy to get the same amount of work done (higher performance)
- Weather and Climate models are usually computationally expensive and suited for parallelization.
- Execute single source code on different CPU and GPU platforms

Goals

- Port a Weather and Climate mini-app (SWM) to DPC++ with limited modifications
- Optimize the performance of the ported code on different CPU and GPU platforms
A venerable 2D shallow water model benchmark on staggered finite difference equations on a torus

- Fortran version developed at NCAR
- C version used by the UK Met Office as a mini-app
- C++ version developed at NCAR in 2021
- Part of SPEC-FP benchmark suite for many years.
- Written when peak flops and bandwidth were comparable

Data Parallel C++

Libraries:
- oneCCL
- oneDAL
- oneDNN
- oneDPL
- oneMKL
- oneTBB
- oneVPL

Analysis & Debug Tools:
- Intel Advisor
- Intel VTune Profiler
- Intel-enhanced GDB

Optimized Applications

Optimized Middleware & Frameworks

**Programming Model - oneAPI?**

oneAPI Product

Direct Programming

API-Based Programming

Analysis & Debug Tools

CPU

GPU

FPGA

*Libraries: oneCCL, oneDAL, oneDNN, oneDPL, oneMKL, oneTBB, oneVPL

**Tools: Intel Advisor, Intel VTune Profiler, Intel-enhanced GDB
Programming Model - oneAPI?

Optimized Applications

Optimized Middleware & Frameworks

oneAPI Product

Direct Programming
- Data Parallel C++

API-Based Programming
- Libraries*

Analysis & Debug Tools**

Optimized Applications
- CPU
- GPU
- FPGA

*Libraries: oneCCL, oneDAL, oneDNN, oneDPL, oneMKL, oneTBB, oneVPL
**Tools: Intel Advisor, Intel VTune Profiler, Intel-enhanced GDB
Optimized Applications

Optimized Middleware & Frameworks

oneAPI Product

Gen9 (and higher)
- Intel® UHD Graphics P630
- Intel® Iris® Xe MAX Graphics

FPGA Cards and FPGA Custom Platforms
- Intel® Arria® 10 FPGAs
- Intel® Stratix® 10 FPGAs

- Intel® Core™ processor family or higher
- Intel® Xeon® processors family
- Intel® Xeon® Scalable processor family

CPU

GPU

FPGA

*Libraries: oneCCL, oneDAL, oneDNN, oneDPL, oneMKL, oneTBB, oneVPL

**Tools: Intel Advisor, Intel VTune Profiler, Intel-enhanced GDB
**What is Data Parallel C+ (DPC++)?**

**Intel Compilers (GPU):**
- C++: dpcpp, icx
- Fortran: ifort, ifx

**DPC++**
- ISO C++
- SYCL

**Standards-based, Cross-architecture Language**

**Queue**
- Submits command groups to be executed by the SYCL runtime

**Device Selector**
- Queue is submitted to the device through device selectors.
  - gpu_selector
  - cpu_selector
  - default_selector
  - host_selector
  - intel::fpga_selector

**Memory Model**
- **Unified Shared Memory**: pointer-based approach
- **Buffers**: Encapsulate data in a SYCL application
  - **Accessors**: Mechanism to access buffer data

**Kernels**
- Encapsulates methods and data for executing code on the device
<table>
<thead>
<tr>
<th>DPC++</th>
<th>Kokkos</th>
<th>CUDA</th>
<th>HIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
<td>Policy Instance</td>
<td>stream</td>
<td>hipStream_t</td>
</tr>
<tr>
<td>ND_range</td>
<td>league</td>
<td>grid</td>
<td>grid</td>
</tr>
<tr>
<td>work_group</td>
<td>team</td>
<td>block</td>
<td>workgroup</td>
</tr>
<tr>
<td>sub_group</td>
<td>vector</td>
<td>warp</td>
<td>wavefront</td>
</tr>
<tr>
<td>work_item</td>
<td>thread/rank</td>
<td>thread</td>
<td>thread/work item</td>
</tr>
</tbody>
</table>

**More info:** [https://wiki.ucar.edu/x/XgXUGg](https://wiki.ucar.edu/x/XgXUGg)

// Initialize velocities

FIELD_SIZE = (m+2)*(n+2); // Size of each array row

double u[3][FIELD_SIZE]; // 2D array with 3 rows (time levels)

// For-loop which targets a single row of the 3-row array, u

for (int i=1; i<m+1; i++) {
    for (int j=1; j<n+1; j++) {
        int ij = i*(n+2)+j;
        ...
        u[0][ij] = ...;
    }
}
double u[3][DOMAIN_SIZE];
auto R = range<1>{DOMAIN_SIZE};
buffer<double, 1> u0_buf(u[0], R);
q.submit([&](handler &h) {
    auto u0 = u0_buf.get_access(h, write_only);
    ...
    h.parallel_for(R, [=](auto ij) {
        int j = ij%(n+2);
        int i = (int) (ij - j)/(n+2);
        ...
        if (i==0 || j==0 || i == m+1 || j== n+1) {} else {
            u0[ij] = ...;
        }
    });
});
double **u = malloc_shared<double*>(3*DOMAIN_SIZE, q);
for (int i=1; i<m+1; i++)
    u[i] = malloc_shared<double>(DOMAIN_SIZE, q);

auto R = range<1>{DOMAIN_SIZE};
q.parallel_for(R, [=](auto ij) {
    int j = ij%(n+2);
    int i = (int) (ij - j)/(n+2);
    ...
    if (i==0 || j==0 || i == m+1 || j== n+1) {}
    else {
        u[ij] = ...;
    }
});
double **u = malloc_shared<double *>(3*DOMAIN_SIZE, q);
for(int i=1; i<m+1; i++)
    u[i] = malloc_shared<double>(DOMAIN_SIZE, q);
auto R = range<1>{DOMAIN_SIZE};
q.parallel_for(R, [=](auto ij) {
    int j = ij%(n+2);
    int i = (int) (ij - j)/(n+2);
    ...
    if (i==0 || j==0 || i == m+1 || j== n+1) {}
    else {
        u[ij] = ...;
    }
});
double **u = malloc_shared<double *>(3*DOMAIN_SIZE, q);
for(int i=1; i<m+1; i++)
    u[i] = malloc_shared<double>(DOMAIN_SIZE, q);
auto R = range<1>{DOMAIN_SIZE};
q.parallel_for(R, [=](auto ij) {
    int j = ij%(n+2);
    int i = (int) (ij - j)/(n+2);
    ...
    if (i==0 || j==0 || i == m+1 || j== n+1) {}
    else {
        u[ij] = ...;
    } });
Results

Performance - CPU - Serial

Gigaflops

Data Size

5000 10000 50000 100000 500000 5000000

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

Skylake - Single Core
DPC++ (dpcpp -O2)
Results

Performance - GPU

Gigaflops

Data Size

Casper V100
OpenACC (nvhpc/21.3 cuda/10.2 -O2)
Intel® Iris® Xe MAX
DPC++ (dpcpp -O2)
Results

Accuracy compared to Serial C++ - GPU

L_{inf} Norm

Data Size

5000 10000 50000 100000 500000

1.00E-9 1.00E-8 1.00E-7 1.00E-6

Casper V100
OpenACC (nvhpc/21.3 cuda/10.2 -O2)

Intel® Iris® Xe MAX
DPC++ (dpcpp -O2)
## Conclusions

<table>
<thead>
<tr>
<th>Feature</th>
<th>USM</th>
<th>Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Portability</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Easiness</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Syntax</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Compiler</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>CPU Performance</td>
<td>5</td>
<td>?</td>
</tr>
<tr>
<td>GPU Performance</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>Documentation &amp; Support</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Future Work

- More investigation on the buffer model
- More optimization on the USM model
- Change the data structure in the SWM mini-app
- Run the code on Nvidia and AMD GPUs
- Compile and run the ported code on FPGAs
- Add support for OpenMP

Acknowledgements

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

**Research Support:** Richard Loft and Thomas Hauser

**SIParCS Admins and CODE Assistants:**
AJ Lauer, Virginia Do, Jerry Cyccone, Max Cordes Galbraith

Thank you!
Leila.Ghaffari@colorado.edu