

Mixed-Language Programming with Fortran and Data Parallel C++

Christoph Bauinger (Technical Consulting Engineer Intern)

James Tullos (Technical Consulting Engineer)



Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice Revision #20110804. <https://software.intel.com/en-us/articles/optimization-notice>

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors.

Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. See backup for configuration details. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

Goals

- Offload computationally intensive tasks in existing Fortran code using DPC++
- Use as much of the existing Fortran code as possible and use DPC++ only for offload
- Minimize changes to the code
- Establish BKM, patterns and designs useful for DPC++ code migration
- Quantify and compare performance (DPC++, OMP)

Code Analysis – The Driver

```
100  call write_state_stats("Input State", log_file_unit)
101
102  ! Get the start time
103  call system_clock(count_start, count_rate)
104
105  #ifdef ENABLE_GPTL
106  if (do_profile == 1) then
107    ret = gptlstart('kernel')
108  endif
109  #endif
110
111  ! Run the kernel
112  !$OMP parallel do schedule(runtime)
113  do k=1, npz
114    call c_sw(sw_corner, se_corner, nw_corner, ne_corner, &
115             rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, &
116             sina_u, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc, &
117             cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, &
118             delpc(isd,j,k), delp(isd,j,k), ptc(isd,j,k), &
119             pt(isd,j,k), u(isd,j,k), v(isd,j,k), &
120             w(isd,j,k), uc(isd,j,k), vc(isd,j,k), &
121             ua(isd,j,k), va(isd,j,k), wc(isd,j,k), &
122             ut(isd,j,k), vt(isd,j,k), divg_d(isd,j,k), dt2)
123  enddo
124
125  #ifdef ENABLE_GPTL
126  if (do_profile == 1) then
127    ret = gptlstop('kernel')
128  endif
129  #endif
130
131  ! Get the stop time
132  call system_clock(count_end, count_rate)
133
134  ! Write the output state statistics to the log file
135  call write_state_stats("Output State", log_file_unit)
```

Bottleneck → Offload

Driver – in Fortran

```
82
83  !-----
84  ! c_sw
85  !-----
86  ! Top-level routine for c_sw kernel
87  !-----
88  subroutine c_sw(sw_corner, se_corner, nw_corner, ne_corner, &
89                rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, &
90                sina_u, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc, &
91                cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, delpc, &
92                delp, ptc, pt, u, v, w, uc, vc, ua, va, wc, ut, &
93                vt, divg_d, dt2)
94
95  logical, intent(in) :: sw_corner, se_corner, ne_corner, nw_corner
96  real,    intent(in), dimension(isd:ied, jsd:jed) :: rarea
97  real,    intent(in), dimension(isd:ied+1, jsd:jed+1) :: rarea_c
98  real,    intent(in), dimension(isd:ied, jsd:jed) :: sin_sg, cos_sg
99  real,    intent(in), dimension(isd:ied, jsd:jed+1) :: sina_v, cosa_v
100 real,    intent(in), dimension(isd:ied+1, jsd:jed) :: sina_u, cosa_u
101 real,    intent(in), dimension(isd:ied+1, jsd:jed+1) :: fC
102 real,    intent(in), dimension(isd:ied+1, jsd:jed) :: rdx, dy, dxc
103 real,    intent(in), dimension(isd:ied, jsd:jed+1) :: rdy, dx, dyc
104 real,    intent(in), dimension(isd:ied, jsd:jed) :: cosa_s
105 real,    intent(in), dimension(isd:ied+1, jsd:jed) :: rsin_u
106 real,    intent(in), dimension(isd:ied, jsd:jed+1) :: rsin_v
107 real,    intent(in), dimension(isd:ied, jsd:jed) :: rsin2, dxa, dya
108 real,    intent(inout), dimension(isd:ied, jsd:jed+1) :: u, vc
109 real,    intent(inout), dimension(isd:ied+1, jsd:jed) :: v, uc
110 real,    intent(inout), dimension(isd:ied, jsd:jed) :: delp, pt, ua
111 real,    intent(inout), dimension(isd:ied, jsd:jed) :: va, ut, vt, w
112 real,    intent(inout), dimension(isd:ied, jsd:jed) :: delpc, ptc, wc
113 real,    intent(inout), dimension(isd:ied+1, jsd:jed+1) :: divg_d
114 real,    intent(in) :: dt2
115
116  ! Local:
117  real, dimension(is-1:ie+1, js-1:je+1) :: vort, ke
118  real, dimension(is-1:ie+2, js-1:je+1) :: fx, fx1, fx2
119  real, dimension(is-1:ie+1, js-1:je+2) :: fy, fy1, fy2
120  real :: dt4
121  integer :: i, j
122  integer :: iep1, jep1
123  integer :: ret
```

Kernel Parameters: 2- and 3-dimensional arrays

Code Analysis – Kernel c_sw

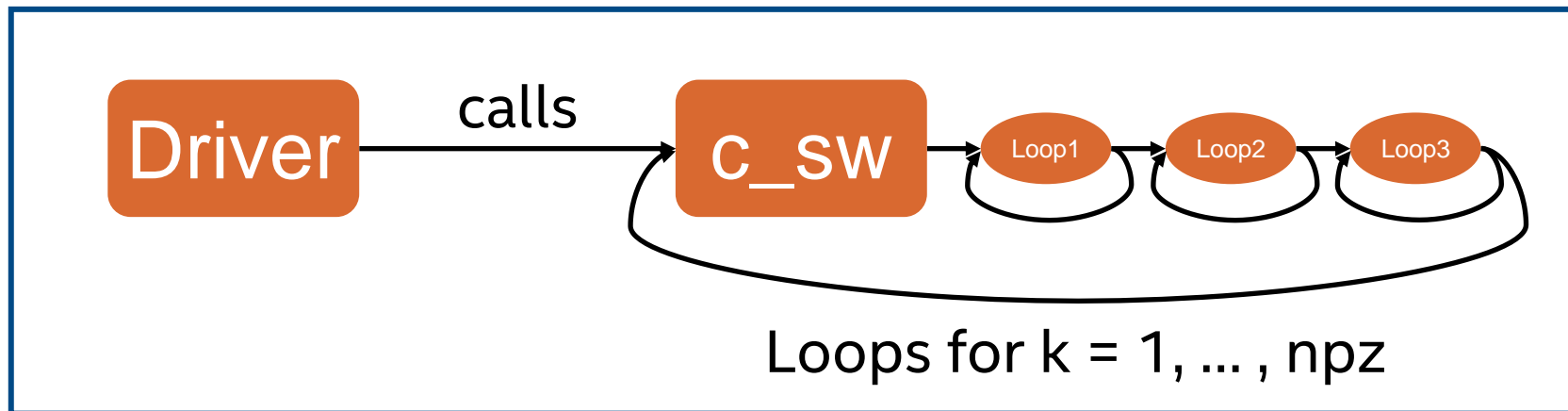
- Kernel consists of function calls and nested loops
- ~800 lines of c_sw kernel Fortran code

```
130
131     ...iep1 = ie + 1
132     ...jep1 = je + 1
133
134     ...call d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, ... &
135     ...sin_sg, cosa_u, cosa_v, cosa_s, rsin_u, rsin_v, rsin2, &
136     ...dxa, dya, u, v, ua, va, uc, vc, ut, vt)
137
138     ...if ( nord > 0 ) then
139     ...call divergence_corner(sw_corner, se_corner, ne_corner, nw_corner, &
140     ...rarea_c, sin_sg, cos_sg, ... &
141     ...dxc, dyc, u, v, ua, va, divg_d)
142     ...endif
143
144     ...do j = js-1, jep1
145     ...do i = is-1, iep1+1
146     ...if (ut(i, j) > 0.) then
147     ...    ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i-1, j, 3)
148     ...else
149     ...    ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i, j, 1)
150     ...end if
151     ...enddo
152     ...enddo
153     ...do j = js-1, je+2
154     ...do i = is-1, iep1
155     ...if (vt(i, j) > 0.) then
156     ...    vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j-1, 4)
157     ...else
158     ...    vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j, 2)
159     ...end if
160     ...enddo
161     ...enddo
162
```

Piece of the kernel code

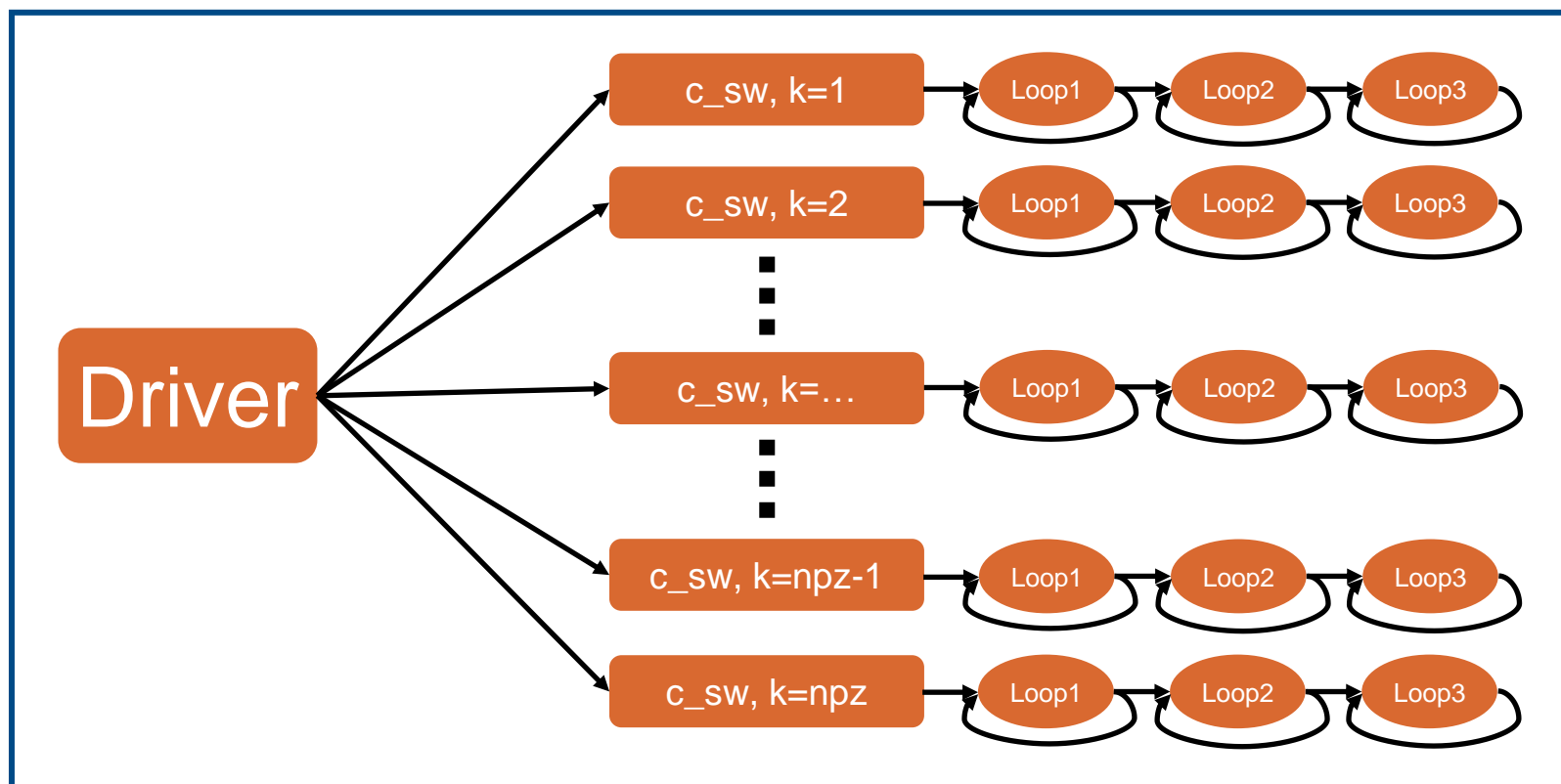
Code Flow Chart

Serial

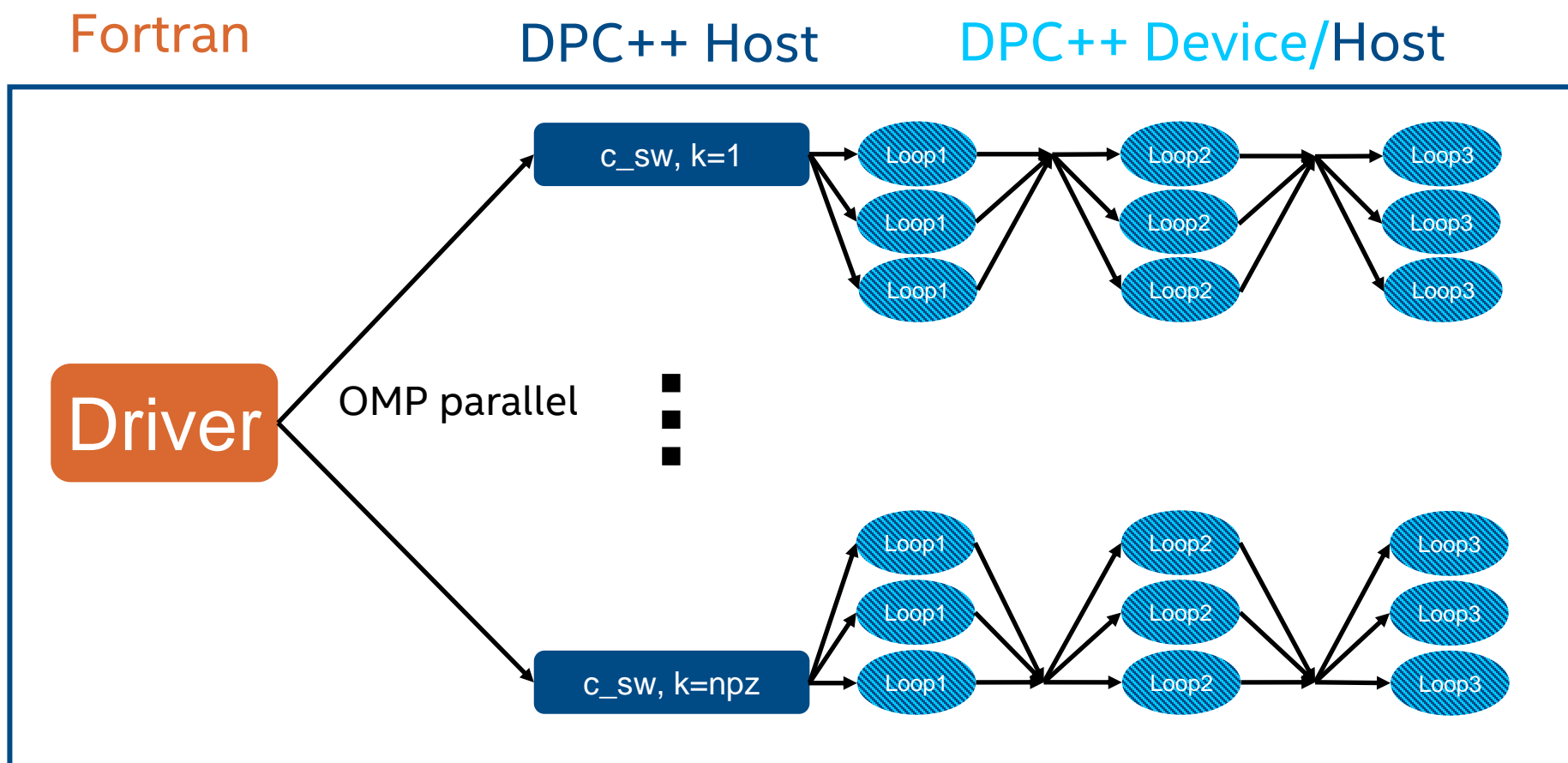


Code Flow Chart

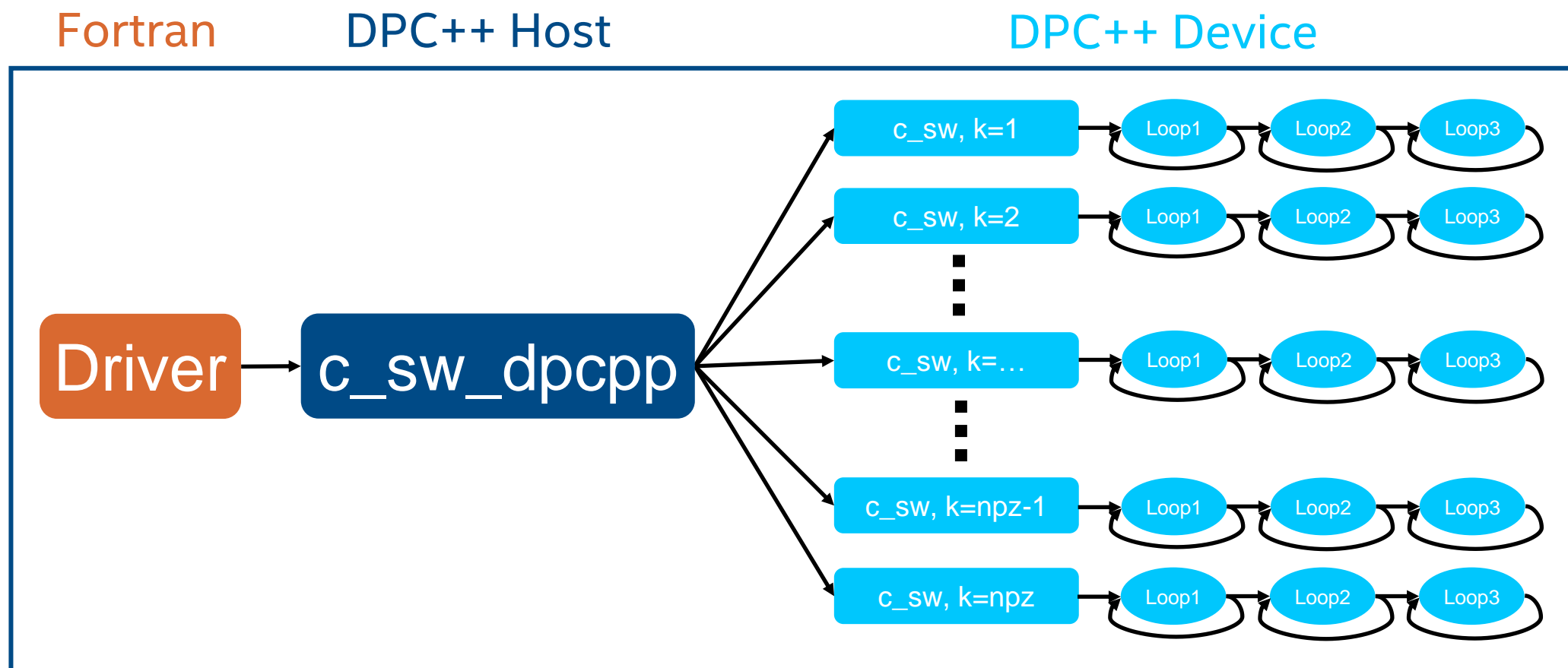
OMP parallel



Possible Offload - Approach 'Loop'



Possible Offload - Approach 'Outer'



Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'

```
82
83  !-----
84  ! c_sw
85  !
86  ! Top-level routine for c_sw kernel
87  !-----
88  subroutine c_sw(sw_corner, se_corner, nw_corner, ne_corner, &
89  ..... rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, &
90  ..... sina_u, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc, &
91  ..... cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, delpc, &
92  ..... delp, ptc, pt, u, v, w, uc, vc, ua, va, wc, ut, &
93  ..... vt, divg_d, dt2)
94
95  ..... logical, intent(in) :: sw_corner, se_corner, ne_corner, nw_corner
96  ..... real, intent(in), dimension(isd:ied, jsd:jed) :: rarea
97  ..... real, intent(in), dimension(isd:ied+1, jsd:jed+1) :: rarea_c
98  ..... real, intent(in), dimension(isd:ied, jsd:jed, 9) :: sin_sg, cos_sg
99  ..... real, intent(in), dimension(isd:ied, jsd:jed+1) :: sina_v, cosa_v
100 ..... real, intent(in), dimension(isd:ied+1, jsd:jed) :: sina_u, cosa_u
101 ..... real, intent(in), dimension(isd:ied+1, jsd:jed+1) :: fC
102 ..... real, intent(in), dimension(isd:ied+1, jsd:jed) :: rdx, dy, dxc
103 ..... real, intent(in), dimension(isd:ied, jsd:jed+1) :: rdy, dx, dyc
104 ..... real, intent(in), dimension(isd:ied, jsd:jed) :: cosa_s
105 ..... real, intent(in), dimension(isd:ied+1, jsd:jed) :: rsin_u
106 ..... real, intent(in), dimension(isd:ied, jsd:jed+1) :: rsin_v
107 ..... real, intent(in), dimension(isd:ied, jsd:jed) :: rsin2, dxa, dya
108 ..... real, intent(inout), dimension(isd:ied, jsd:jed+1) :: u, vc
109 ..... real, intent(inout), dimension(isd:ied+1, jsd:jed) :: v, uc
110 ..... real, intent(inout), dimension(isd:ied, jsd:jed) :: delp, pt, ua
111 ..... real, intent(inout), dimension(isd:ied, jsd:jed) :: va, ut, vt, w
112 ..... real, intent(out), dimension(isd:ied, jsd:jed) :: delpc, ptc, wc
113 ..... real, intent(out), dimension(isd:ied+1, jsd:jed+1) :: divg_d
114 ..... real, intent(in) :: dt2
115
116  ! Local:
117 ..... real, dimension(is-1:ie+1, js-1:je+1) :: vort, ke
118 ..... real, dimension(is-1:ie+2, js-1:je+1) :: fx, fx1, fx2
119 ..... real, dimension(is-1:ie+1, js-1:je+2) :: fy, fy1, fy2
120 ..... real :: dt4
121 ..... integer :: i, j
122 ..... integer :: iep1, jep1
123 ..... integer :: ret
```

Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'

c_sw code

```
130
131     iep1 = ie + 1
132     jep1 = je + 1
133
134     call d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, &
135                   sin_sg, cosa_u, cosa_v, cosa_s, rsin_u, rsin_v, rsin2, &
136                   dxa, dya, u, v, ua, va, uc, vc, ut, vt)
137
138     if ( nord > 0 ) then
139         call divergence_corner(sw_corner, se_corner, ne_corner, nw_corner, &
140                           rarea_c, sin_sg, cos_sg, &
141                           dxc, dyc, u, v, ua, va, divg_d)
142     endif
143
144     do j = js-1, jep1
145         do i = is-1, iep1+1
146             if (ut(i, j) > 0.) then
147                 ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i-1, j, 3)
148             else
149                 ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i, j, 1)
150             end if
151         enddo
152     enddo
153     do j = js-1, je+2
154         do i = is-1, iep1
155             if (vt(i, j) > 0.) then
156                 vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j-1, 4)
157             else
158                 vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j, 2)
159             end if
160         enddo
161     enddo
162
```

Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'

```
130
131     iep1 = ie + 1
132     jep1 = je + 1
133
134     call d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, &
135                   sin_sg, cosa_u, cosa_v, cosa_s, rsin_u, rsin_v, rsin2, &
136                   dxa, dya, u, v, ua, va, uc, vc, ut, vt)
137
138     if ( nord > 0 ) then
139         call divergence_corner(sw_corner, se_corner, ne_corner, nw_corner, &
140                           rarea_c, sin_sg, cos_sg, &
141                           dxc, dyc, u, v, ua, va, divg_d)
142     endif
143
144     do j = js-1, jep1
145         do i = is-1, iep1+1
146             if (ut(i,j) > 0.) then
147                 ut(i,j) = dt2 * ut(i,j) * dy(i,j) * sin_sg(i-1,j,3)
148             else
149                 ut(i,j) = dt2 * ut(i,j) * dy(i,j) * sin_sg(i,j,1)
150             end if
151         enddo
152     enddo
153     do j = js-1, je+2
154         do i = is-1, iep1
155             if (vt(i,j) > 0.) then
156                 vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j-1,4)
157             else
158                 vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j,2)
159             end if
160         enddo
161     enddo
162
```

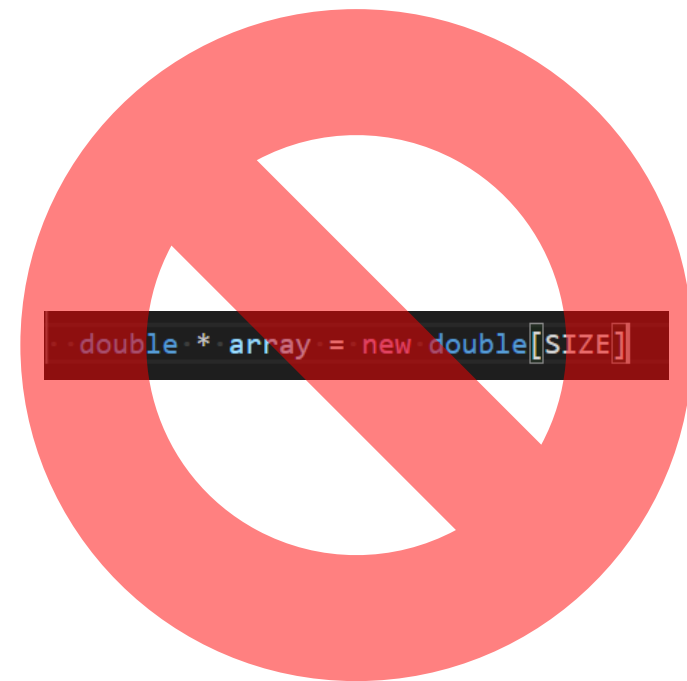
Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'

```
130
131     iep1 = ie + 1
132     jep1 = je + 1
133
134     call d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, &
135                   sin_sg, cosa_u, cosa_v, cosa_s, rsin_u, rsin_v, rsin2, &
136                   dxa, dya, u, v, ua, va, uc, vc, ut, vt)
137
138     if ( nord > 0 ) then
139         call divergence_corner(sw_corner, se_corner, ne_corner, nw_corner, &
140                           rarea_c, sin_sg, cos_sg, &
141                           dxc, dyc, u, v, ua, va, divg_d)
142     endif
143
144     do j = js-1, jep1
145         do i = is-1, iep1+1
146             if (ut(i,j) > 0.) then
147                 ut(i,j) = dt2 * ut(i,j) * dy(i,j) * sin_sg(i-1,j,3)
148             else
149                 ut(i,j) = dt2 * ut(i,j) * dy(i,j) * sin_sg(i,j,1)
150             end if
151         enddo
152     enddo
153     do j = js-1, je+2
154         do i = is-1, iep1
155             if (vt(i,j) > 0.) then
156                 vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j-1,4)
157             else
158                 vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j,2)
159             end if
160         enddo
161     enddo
162
```

Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'



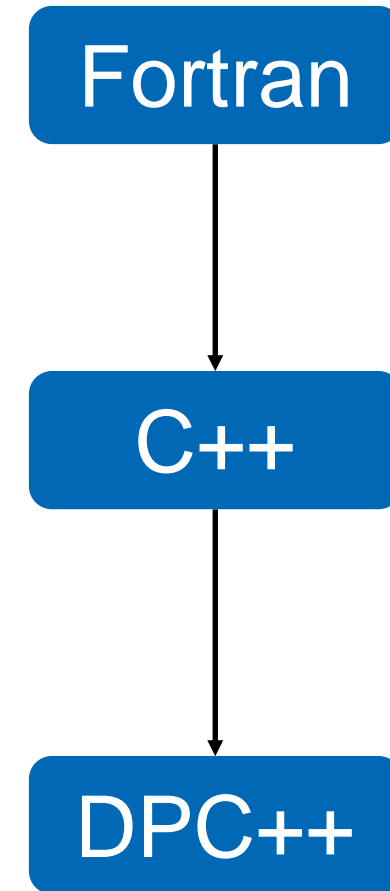
Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'

```
100  call write_state_stats("Input State", log_file_unit)
101
102  ! Get the start time
103  call system_clock(count_start, count_rate)
104
105  #ifdef ENABLE_GPTL
106  if (do_profile == 1) then
107    ret = gptlstart('kernel')
108  end if
109  #endif
110
111  ! Run the kernel
112  !$OMP parallel do schedule(runtime)
113  do k=1, npz
114    call c_sw(sw_corner, se_corner, nw_corner, ne_corner, &
115            & rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, &
116            & sina_u, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc, &
117            & cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, &
118            & delpc(isd,jsd,k), delp(isd,jsd,k), ptc(isd,jsd,k), &
119            & pt(isd,jsd,k), u(isd,jsd,k), v(isd,jsd,k), &
120            & w(isd,jsd,k), uc(isd,jsd,k), vc(isd,jsd,k), &
121            & ua(isd,jsd,k), va(isd,jsd,k), wc(isd,jsd,k), &
122            & ut(isd,jsd,k), vt(isd,jsd,k), divg_d(isd,jsd,k), dt2)
123  enddo
124
125  #ifdef ENABLE_GPTL
126  if (do_profile == 1) then
127    ret = gptlstop('kernel')
128  end if
129  #endif
130
131  ! Get the stop time
132  call system_clock(count_end, count_rate)
133
134  ! Write the output state statistics to the log file
135  call write_state_stats("Output State", log_file_unit)
```

Design Challenges

- Non-zero start of Fortran arrays
- Non-zero start of loops
- Fortran column-major vs C++ row-major
- Access operator in Fortran differs from C++
- No memory allocations in device code
- Many arrays as parameters
- Solution: Two step migration to C++ and then DPC++, use Approach 'Outer'



Challenges Fortran - C++

- Main Issue: Array Indexing
 - Unit stride is inner index in Fortran vs outer in C++
 - Non-zero indexing in Fortran possible (and used in code)
- Solution: Class OffsetArray (1D, 2D, 3D) wrapping dynamic array emulating Fortran style arrays (see Figure)

```
1  template <class T>
2  class OffsetArray
3  {
4      ....const int istart;
5      ....const int iend;
6      ....T* const values;
7
8      public:
9
10     OffsetArray(int istart_in, int iend_in) :
11         ....istart(std::min(istart_in, iend_in)),
12         ....iend(std::max(istart_in, iend_in)),
13         ....values(new T[iend-istart+1])
14     {
15     }
16
17     ~OffsetArray() {
18         ....delete [] values;
19     }
20
21     T&operator()(int i) {
22         ....const int index=i-istart;
23         ....return values[index];
24     }
25
26 };
27
28
29
30  /////USAGE/////
31  int main()
32  {
33      ....OffsetArray<double> u(5, 10);
34      ....u(5) = 1.0;
35  }
36
```

Fortran - C++ Code Comparison

Fortran

```
133
134 ...call d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, .....&
135 .....sin_sg, cosa_u, cosa_v, cosa_s, rsin_u, rsin_v, rsin2, &
136 .....dxa, dya, u, v, ua, va, uc, vc, ut, vt)
137
138 ...if (nord > 0.) then
139 ...call divergence_corner(sw_corner, se_corner, ne_corner, nw_corner, &
140 .....rarea_c, sin_sg, cos_sg, .....&
141 .....dxc, dyc, u, v, ua, va, divg_d)
142 ...endif
143
144 ...do j = js-1, jep1
145 ...do i = is-1, iep1+1
146 ...if (ut(i, j) > 0.) then
147 ...ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i-1, j, 3)
148 ...else
149 ...ut(i, j) = dt2 * ut(i, j) * dy(i, j) * sin_sg(i, j, 1)
150 ...end if
151 ...enddo
152 ...enddo
153 ...do j = js-1, je+2
154 ...do i = is-1, iep1
155 ...if (vt(i, j) > 0.) then
156 ...vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j-1, 4)
157 ...else
158 ...vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j, 2)
159 ...end if
160 ...enddo
161 ...enddo
```

C++

```
1049 ...d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, sin_sg, cosa_u,
1050 ...cosa_v, cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, u, v, ua, va, uc, vc,
1051 ...ut, vt);
1052
1053 ...if (nord > 0)
1054 ...{
1055 ...divergence_corner(sw_corner, se_corner, ne_corner, nw_corner,
1056 .....rarea_c, sin_sg, cos_sg, dxc, dyc, u, v, ua, va, divg_d);
1057 ...}
1058
1059 ...for (j=js-1; j<=jep1; j++)
1060 ...{
1061 ...for (i=is-1; i<=iep1+1; i++)
1062 ...{
1063 ...if (ut(i,j)>0.0)
1064 ...ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i-1,j,3);
1065 ...else
1066 ...ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i,j,1);
1067 ...}
1068 ...}
1069
1070 ...for (j=js-1; j<=je+2; j++)
1071 ...{
1072 ...for (i=is-1; i<=iep1; i++)
1073 ...{
1074 ...if (vt(i,j)>0.0)
1075 ...vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j-1, 4);
1076 ...else
1077 ...vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j, 2);
1078 ...}
1079 ...}
```

Challenges C++ - DPC++

■ Main Issues:

- OffsetArray class cannot be used in device code (as it is)
- DPC++ parallel_for indices start at 0 ('loop indexing starting at 0')

■ Solutions:

- Naively: Simple offsets based on loop and array start index
- Better: Adjusting the OffsetArray class and use Approach 'Outer'

C++ - DPC++ Code Comparison: Naively

C++

```
1049 d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, sin_sg, cosa_u,
1050 cosa_v, cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, u, v, ua, va, uc, vc,
1051 ut, vt);
1052
1053 if (nord > 0)
1054 {
1055     divergence_corner(sw_corner, se_corner, ne_corner, nw_corner,
1056 rarea_c, sin_sg, cos_sg, dxc, dyc, u, v, ua, va, divg_d);
1057 }
1058
1059 for (j=js-1; j<=jep1; j++)
1060 {
1061     for (i=is-1; i<=iep1+1; i++)
1062     {
1063         if (ut(i,j)>0.0)
1064             ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i-1,j,3);
1065         else
1066             ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i,j,1);
1067     }
1068 }
1069
1070 for (j=js-1; j<=je+2; j++)
1071 {
1072     for (i=is-1; i<=iep1; i++)
1073     {
1074         if (vt(i,j)>0.0)
1075             vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j-1, 4);
1076         else
1077             vt(i, j) = dt2 * vt(i, j) * dx(i, j) * sin_sg(i, j, 2);
1078     }
1079 }
```

DPC++

```
169 d2a2c_vect();
170
171 if (nord > 0)
172     divergence_corner();
173
174 Q.submit([&](handler &h) {
175     accessor abut{but, h, read_write};
176     accessor abdy{bdy, h, read_only};
177     accessor absin_sg{bsin_sg, h, read_only};
178     const int i_off = -isd + is-1;
179     const int j_off = -jsd + js-1;
180     const int k_off = -1;
181     h.parallel_for(range<2>{static_cast<size_t>(jep1-(js-1)+1), static_cast<size_t>(iep1+1-(is-1)+1)}, [=](id<2> idx) {
182         const int i = idx[1] + i_off;
183         const int j = idx[0] + j_off;
184         if (abut[j][i]>0.0)
185             abut[j][i]=ldt2*abut[j][i]*abdy[j][i]*absin_sg[3+k_off][j][i-1];
186         else
187             abut[j][i]=ldt2*abut[j][i]*abdy[j][i]*absin_sg[1+k_off][j][i];
188     });
189
190 Q.submit([&](handler &h) {
191     accessor abvt{bvt, h, read_write};
192     accessor abdx{bdx, h, read_only};
193     accessor absin_sg{bsin_sg, h, read_only};
194     const int i_off = -isd + is-1;
195     const int j_off = -jsd + js-1;
196     const int k_off = -1;
197     h.parallel_for(range<2>{static_cast<size_t>(je+2-(js-1)+1), static_cast<size_t>(iep1-(is-1)+1)}, [=](id<2> idx) {
198         const int i = idx[1] + i_off;
199         const int j = idx[0] + j_off;
200         if (abvt[j][i]>0.0)
201             abvt[j][i]=ldt2*abvt[j][i]*abdx[j][i]*absin_sg[4+k_off][j-1][i];
202         else
203             abvt[j][i]=ldt2*abvt[j][i]*abdx[j][i]*absin_sg[2+k_off][j][i];
204     });
205 }
```

} DPC++ Accessors
} Offsets due to array and loop
For loop range

C++ - DPC++ Code Comparison: Naively

- Tedious migration
- Hard-to-read code
- Need to rewrite everything carefully
- Debugging is a horror
- Focus on Approach 'Outer'

C++

```
1049 d2a2c_vect(sw_corner, se_corner, ne_corner, nw_corner, sin_sg, cosa_u,
1050 cosa_v, cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, u, v, ua, va, uc, vc,
1051 ut, vt);
1052
1053 if (nord > 0)
1054 {
1055     divergence_corner(sw_corner, se_corner, ne_corner, nw_corner,
1056                       rarea_c, sin_sg, cos_sg, dxc, dyc, u, v, ua, va, divg_d);
1057 }
1058
1059 for (j=js-1; j<=jep1; j++)
1060 {
1061     for (i=is-1; i<=iep1; i++)
1062     {
1063         if (ut(i,j)>0.0)
1064             ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i-1,j,3);
1065         else
1066             ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i,j,1);
1067     }
1068 }
1069
1070 for (j=js-1; j<=jep2; j++)
1071 {
1072     for (i=is-1; i<=iep1; i++)
1073     {
1074         if (vt(i,j)>0.0)
1075             vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j-1,4);
1076         else
1077             vt(i,j) = dt2 * vt(i,j) * dx(i,j) * sin_sg(i,j,2);
1078     }
1079 }
```

DPC++

```
1049 d2a2c_vect();
1050
1051 if (nord > 0)
1052     divergence_corner();
1053
1054 Q_submit([&](handler h) {
1055     accessor abut(but, h, read_write);
1056     accessor abdy(bdy, h, read_only);
1057     accessor absin_sg(bsin_sg, h, read_only);
1058     const int i_off = -isd + is-1;
1059     const int j_off = -isd + js-1;
1060     const int k_off = -1;
1061     h.parallel_for(range2d(static_cast<int>(jep1-(is-1)+1), static_cast<int>(jep1-(is-1)+1)), [=](id2 id) {
1062         const int i = id[0] + i_off;
1063         const int j = id[0] + j_off;
1064         if (abut[i][j]>0.0)
1065             abut[j][i]=dt2*abut[j][i]*abdy[j][i]*absin_sg[i+k_off][j][1];
1066         else
1067             abut[j][i]=dt2*abut[j][i]*abdy[j][i]*absin_sg[i+k_off][j][1];
1068     });
1069 }
1070
1071 Q_submit([&](handler h) {
1072     accessor abvt(bvt, h, read_write);
1073     accessor abdy(bdy, h, read_only);
1074     accessor absin_sg(bsin_sg, h, read_only);
1075     const int i_off = -isd + is-1;
1076     const int j_off = -isd + js-1;
1077     const int k_off = -1;
1078     h.parallel_for(range2d(static_cast<int>(jep2-(js-1)+1), static_cast<int>(jep1-(is-1)+1)), [=](id2 id) {
1079         const int i = id[0] + i_off;
1080         const int j = id[0] + j_off;
1081         if (abvt[i][j]>0.0)
1082             abvt[j][i]=dt2*abvt[j][i]*abdy[j][i]*absin_sg[i+k_off][j][1];
1083         else
1084             abvt[j][i]=dt2*abvt[j][i]*abdy[j][i]*absin_sg[i+k_off][j][1];
1085     });
1086 }
```

DPC++ Accessors
Offsets due to array and loop
For loop range

Adjusting OffsetArray

- Challenges when implicitly copying objects from host to device
 - Trivial destructor required
 - Const access operator necessary (copy to device is by const value)
 - No memory allocations in device code possible; Host needs to handle memory

```
19 template <class T>
20 class OffsetArray
21 {
22     ... const int istart;
23     ... const int iend;
24     ... T* const values;
25
26     public:
27     ... OffsetArray() = delete;
28
29     ... OffsetArray(int istart_in, int iend_in, T* values_in) :
30         ... istart(istart_in),
31         ... iend(iend_in),
32         ... values(values_in)
33     {
34     }
35
36     ... T&operator()(int i) const
37     {
38         ... return values[i-istart];
39     }
40 };
41
```

Adjusting OffsetArray

- Challenges when implicitly copying objects from host to device
 - Trivial destructor required
 - Const access operator necessary (copy to device is by const value)
 - No memory allocations in device code possible; Host needs to handle memory

```
19 template <class T>
20 class OffsetArray
21 {
22     ... const int istart;
23     ... const int iend;
24     ... T* const values;
25
26     public:
27     ... OffsetArray() = delete;
28
29     ... OffsetArray(int istart_in, int iend_in, T* values_in) :
30         ... istart(istart_in),
31         ... iend(iend_in),
32         ... values(values_in)
33     {
34     }
35
36     ... T&operator()(int i) const
37     {
38         ... return values[i-istart];
39     }
40 };
41
```

Adjusting the OffsetArray Class for Approach 2

- Challenges when implicitly copying objects from host to device
 - Trivial destructor required
 - Const access operator necessary (copy to device is by const value)
 - No memory allocations in device code possible; Host needs to handle memory

```
19  template <class T>
20  class OffsetArray
21  {
22  ..  const int istart;
23  ..  const int iend;
24  ..  T* const values;
25
26  public:
27  ..  OffsetArray() = delete;
28
29  ..  OffsetArray(int istart_in, int iend_in, T* values_in) :
30  ..  ..  istart(istart_in),
31  ..  ..  iend(iend_in),
32  ..  ..  values(values_in)
33  ..  {
34  ..  }
35
36  ..  T&operator()(int i) const
37  ..  {
38  ..  ..  return values[i-istart];
39  ..  }
40  };
41
```


Usage of the OffsetArray

- Allocate device memory explicitly
- Construct OffsetArray with it
- Copy memory back to host after device code execution
- Free device memory explicitly after device code execution
- Minimize allocations and copies between host and device
- OffsetArray can be used on host and device

```
19  template <class T>
20  class OffsetArray
21  {
22      ... const int istart;
23      ... const int iend;
24      ... T* const values;
25
26  public:
27      ... OffsetArray() = delete;
28
29      ... OffsetArray(int istart_in, int iend_in, T* values_in) :
30          ... istart(istart_in),
31          ... iend(iend_in),
32          ... values(values_in)
33          {
34          }
35
36      ... T&operator()(int i) const
37      {
38          ... return values[i-istart];
39      }
40  };
41
```

Variations of the OffsetArray

- Can use accessors to buffers instead of USM ($T * \text{const}$)
- Shared memory (malloc_shared) is also a possibility → can be used on host and device with implicit copying → would work well for Approach 'Loop'
- A possible extension would be a factory pattern for the generation of the OffsetArray

```
19 template <class T>
20 class OffsetArray
21 {
22     const int istart;
23     const int iend;
24     T* const values;
25
26 public:
27     OffsetArray() = delete;
28
29     OffsetArray(int istart_in, int iend_in, T* values_in) :
30         istart(istart_in),
31         iend(iend_in),
32         values(values_in)
33     {
34     }
35
36     T&operator()(int i) const
37     {
38         return values[i-istart];
39     }
40 };
41
```

Putting Everything Together

Fortran Driver

```
120  ..! Run the kernel
121  ..call c_sw_dpcpp(isd, ied, jsd, jed, is, ie, js, je, nord, npx, npy, &
122  ..npz, do_profile, dt2, .....&
123  ..sw_corner, se_corner, nw_corner, ne_corner, .....&
124  ..rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, .....&
125  ..sina_u, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc, .....&
126  ..cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, .....&
127  ..delpc, delp, ptc, pt, u, v, w, uc, vc, ua, va, wc, .....&
128  ..ut, vt, divg_d)
129
130  ..! !$OMP parallel do schedule(runtime)
131  ..! do k=1, npz
132  ..! ..call c_sw(sw_corner, se_corner, nw_corner, ne_corner, .....&
133  ..! ..delpc(isd, jsd, k), delp(isd, jsd, k), ptc(isd, jsd, k), .....&
134  ..! ..pt(isd, jsd, k), u(isd, jsd, k), v(isd, jsd, k), .....&
135  ..! ..w(isd, jsd, k), uc(isd, jsd, k), vc(isd, jsd, k), .....&
136  ..! ..ua(isd, jsd, k), va(isd, jsd, k), wc(isd, jsd, k), .....&
137  ..! ..ut(isd, jsd, k), vt(isd, jsd, k), divg_d(isd, jsd, k), dt2)
138  ..! enddo
139
```

DPC++ Device Code = C++ Code

```
119  ..d2a2c_vect(u, v, ua, va, uc, vc, ut, vt, utmp, vtmp);
120
121
122  ..if (nord > 0)
123  ..{
124  ..  ..divergence_corner(u, v, ua, va, divg_d, uf, vf);
125  ..}
126
127  ..for (j=js-1; j<=jep1; j++)
128  ..{
129  ..  ..for (i=is-1; i<=iep1+1; i++)
130  ..  ..{
131  ..    ..if (ut(i,j)>0.0)
132  ..    ..  ..ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i-1,j,3);
133  ..    ..else
134  ..    ..  ..ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i,j,1);
135  ..  ..}
136  ..}
137
138
139  ..for (j=js-1; j<=je+2; j++)
140  ..{
141  ..  ..for (i=is-1; i<=iep1; i++)
142  ..  ..{
143  ..    ..if (vt(i,j)>0.0)
144  ..    ..  ..vt(i,j) = dt2 * vt(i,j) * dx(i, j) * sin_sg(i, j-1, 4);
145  ..    ..else
146  ..    ..  ..vt(i,j) = dt2 * vt(i,j) * dx(i, j) * sin_sg(i, j, 2);
147  ..  ..}
148  ..}
149
```

New DPC++ c_sw_dpcpp

Fortran Driver

```
112
113 #ifdef ENABLE_GPTL
114 if (do_profile == 1) then
115   ret = gptlstart('kernel')
116   end if
117 #endif
118
119 ! Run the kernel
120 call c_sw_dpcpp(isd, ied, jsd, jed, is, ie, js, je, nord, npz, npx, npy, &
121   npz, do_profile, dt2,
122   sw_corner, se_corner, nw_corner, ne_corner, rarea, rarea_c, sin_sg,
123   cos_sg, sina_u, fC, rdx, rdy, dx, dy, dxc, dyc,
124   cosa_u, cosa_v, cosa_w, cosa_x, cosa_y, cosa_z,
125   cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, dxb, dyb,
126   delpc, delp, ptc, pt, u, v, w, uc, vc, ua, va, wc,
127   ut, vt, divg_d)
128
129 ! $OMP parallel do schedule(runtime)
130 do k=1, npz
131   call c_sw_dpcpp(sw_corner, se_corner, nw_corner, ne_corner,
132     delpc(isd, jsd, k), delp(isd, jsd, k), ptc(isd, jsd, k),
133     pt(isd, jsd, k), u(isd, jsd, k), v(isd, jsd, k), w(isd, jsd, k),
134     uc(isd, jsd, k), vc(isd, jsd, k), ua(isd, jsd, k), va(isd, jsd, k),
135     wc(isd, jsd, k), ut(isd, jsd, k), vt(isd, jsd, k), divg_d(isd, jsd, k), dt2)
136   enddo
137
138
139
```

```
168 Q.submit([&](handler &h) {
169   CSW Foo(isd, ied, jsd, jed, is, ie, js, je, nord, npx, npy, npz, do_profile,
170     dt2, sw_corner, se_corner, nw_corner, ne_corner, rarea, rarea_c, sin_sg,
171     cos_sg, sina_u, cosa_v, cosa_u, fC, rdx, rdy, dx, dy, dxc, dyc,
172     cosa_s, rsin_u, rsin_v, rsin2, dxa, dya);
173
174   h.parallel_for(npz, [=](auto k) {
175     Offset2DArray<double, true> delpc(isd, ied, jsd, jed, pdelpc+k*isize*jsize);
176     Offset2DArray<double, true> delp(isd, ied, jsd, jed, pdelp+k*isize*jsize);
177     Offset2DArray<double, true> ptc(isd, ied, jsd, jed, pptc+k*isize*jsize);
178     Offset2DArray<double, true> pt(isd, ied, jsd, jed, ppt+k*isize*jsize);
179     Offset2DArray<double, true> u(isd, ied, jsd, jed+1, pu+k*isize*jsize1);
180     Offset2DArray<double, true> v(isd, ied+1, jsd, jed, pv+k*isize1*jsize);
181     Offset2DArray<double, true> w(isd, ied, jsd, jed, pw+k*isize*jsize);
182     Offset2DArray<double, true> uc(isd, ied+1, jsd, jed, puc+k*isize1*jsize);
183     Offset2DArray<double, true> vc(isd, ied, jsd, jed+1, pvc+k*isize*jsize1);
184     Offset2DArray<double, true> ua(isd, ied, jsd, jed, pua+k*isize*jsize);
185     Offset2DArray<double, true> va(isd, ied, jsd, jed, pva+k*isize*jsize);
186     Offset2DArray<double, true> wc(isd, ied, jsd, jed, pwc+k*isize*jsize);
187     Offset2DArray<double, true> ut(isd, ied, jsd, jed, put+k*isize*jsize);
188     Offset2DArray<double, true> vt(isd, ied, jsd, jed, pvt+k*isize*jsize);
189     Offset2DArray<double, true> divg_d(isd, ied+1, jsd, jed+1, pdivg_d+k*isize1*jsize1);
190
191     Offset2DArray<double, true> vort(is-1, ie+1, js-1, je+1, pvort+k*bufisize*bufjsize);
192     Offset2DArray<double, true> ke(is-1, ie+1, js-1, je+1, pke+k*bufisize*bufjsize);
193     Offset2DArray<double, true> fx(is-1, ie+2, js-1, je+1, pfx+k*bufisize1*bufjsize);
194     Offset2DArray<double, true> fx1(is-1, ie+2, js-1, je+1, pfx1+k*bufisize1*bufjsize);
195     Offset2DArray<double, true> fx2(is-1, ie+2, js-1, je+1, pfx2+k*bufisize1*bufjsize);
196     Offset2DArray<double, true> fy(is-1, ie+1, js-1, je+2, pfy+k*bufisize*bufjsize1);
197     Offset2DArray<double, true> fy1(is-1, ie+1, js-1, je+2, pfy1+k*bufisize*bufjsize1);
198     Offset2DArray<double, true> fy2(is-1, ie+1, js-1, je+2, pfy2+k*bufisize*bufjsize1);
199     Offset2DArray<double, true> utmp(isd, ied, jsd, jed, putmp+k*isize*jsize);
200     Offset2DArray<double, true> vtmp(isd, ied, jsd, jed, pvtmp+k*isize*jsize);
201     Offset2DArray<double, true> uf(is-2, ie+2, js-1, je+2, puf+k*(bufisize+2)*bufjsize1);
202     Offset2DArray<double, true> vf(is-1, ie+2, js-2, je+2, pvf+k*bufisize1*(bufjsize+2));
203
204     Foo.c_sw(delpc, delp, ptc, pt, u, v, w, uc, vc, ua, va, wc, ut, vt, divg_d,
205       vort, ke, fx, fx1, fx2, fy, fy1, fy2, utmp, vtmp, uf, vf);
206   });
207 }).wait();
208
```

DPC++ Device Code

```
119 d2a2c_vect(u, v, ua, va, uc, vc, ut, vt, utmp, vtmp);
120
121
122 if (nord > 0)
123 {
124   divergence_corner(u, v, ua, va, divg_d, uf, vf);
125 }
126
127 for (j=js-1; j<=jep1; j++)
128 {
129   for (i=is-1; i<=iep1+1; i++)
130   {
131     if (vt(i,j)>0.0)
132       ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i-1,j,3);
133     else
134       ut(i,j)=dt2*ut(i,j)*dy(i,j)*sin_sg(i,j,1);
135   }
136 }
137
138
139 for (j=js-1; j<=jep2; j++)
140 {
141   for (i=is-1; i<=iep1; i++)
142   {
143     if (vt(i,j)>0.0)
144       vt(i,j) = dt2 * vt(i,j) * dx(i, j) * sin_sg(i, j-1, 4);
145     else
146       vt(i,j) = dt2 * vt(i,j) * dx(i, j) * sin_sg(i, j, 2);
147   }
148 }
149
```

Compilation

- export FC=ifx
- export CXX=dpcpp
- Link for_main.o
- Use linker flag -lifcore
- Use dpcpp for linking

```
5 cmake_minimum_required( VERSION 3.10 )
6
7 project( c_sw LANGUAGES Fortran C CXX )
8
9 list( APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake )
10 set( CMAKE_DIRECTORY_LABELS ${PROJECT_NAME} )
11 set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore" )
12
13 enable_testing()
14
15 include(${PROJECT_NAME}_compiler_flags)
16
17 find_package( OpenMP COMPONENTS C Fortran )
18 find_package( NetCDF REQUIRED COMPONENTS C Fortran )
19
20 add_subdirectory(src)
21 add_subdirectory(test)
22
23 set_target_properties( ${PROJECT_NAME} PROPERTIES LINKER_LANGUAGE CXX )
24
```

CMakeLists.txt

```
12 SET(OBJS
13   /global/panfs01/admin/opt/intel/oneAPI/2021.1.2/compiler/2021.1.2/linux/compiler/lib/intel64_lin/for_main.o
14 )
15
16 add_executable( ${PROJECT_NAME} ${OBJS} ${c_sw_src_files} )
```

src/CMakeLists.txt

Compilation

- export FC=ifx
- export CXX=dpcpp
- Link for_main.o
- Use linker flag -lifcore
- Use dpcpp for linking

```
5 cmake_minimum_required( VERSION 3.10 )
6
7 project( c_sw LANGUAGES Fortran C CXX )
8
9 list( APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake )
10 set( CMAKE_DIRECTORY_LABELS ${PROJECT_NAME} )
11 set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore" )
12
13 enable_testing()
14
15 include(${PROJECT_NAME}_compiler_flags)
16
17 find_package( OpenMP COMPONENTS C Fortran )
18 find_package( NetCDF REQUIRED COMPONENTS C Fortran )
19
20 add_subdirectory(src)
21 add_subdirectory(test)
22
23 set_target_properties( ${PROJECT_NAME} PROPERTIES LINKER_LANGUAGE CXX )
24
```

CMakeLists.txt

```
12 SET(OBJS
13   /global/panfs01/admin/opt/intel/oneAPI/2021.1.2/compiler/2021.1.2/linux/compiler/lib/intel64_lin/for_main.o
14 )
15
16 add_executable( ${PROJECT_NAME} ${OBJS} ${c_sw_src_files} )
```

src/CMakeLists.txt

Compilation

- export FC=ifx
- export CXX=dpcpp
- Link for_main.o
- Use linker flag -lifcore
- Use dpcpp for linking

```
5 cmake_minimum_required( VERSION 3.10 )
6
7 project( c_sw LANGUAGES Fortran C CXX )
8
9 list( APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake )
10 set( CMAKE_DIRECTORY_LABELS ${PROJECT_NAME} )
11 set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore" )
12
13 enable_testing()
14
15 include(${PROJECT_NAME}_compiler_flags)
16
17 find_package( OpenMP COMPONENTS C Fortran )
18 find_package( NetCDF REQUIRED COMPONENTS C Fortran )
19
20 add_subdirectory(src)
21 add_subdirectory(test)
22
23 set_target_properties( ${PROJECT_NAME} PROPERTIES LINKER_LANGUAGE CXX )
24
```

CMakeLists.txt

```
12 SET(OBJS
13 /global/panfs01/admin/opt/intel/oneAPI/2021.1.2/compiler/2021.1.2/linux/compiler/lib/intel64_lin/for_main.o
14 )
15
16 add_executable( ${PROJECT_NAME} ${OBJS} ${c_sw_src_files} )
```

src/CMakeLists.txt

Compilation

- export FC=ifx
- export CXX=dpcpp
- Link for_main.o
- Use linker flag -lifcore
- Use dpcpp for linking

```
5 cmake_minimum_required( VERSION 3.10 )
6
7 project( c_sw LANGUAGES Fortran C CXX )
8
9 list( APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake )
10 set( CMAKE_DIRECTORY_LABELS ${PROJECT_NAME} )
11 set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore" )
12
13 enable_testing()
14
15 include(${PROJECT_NAME}_compiler_flags)
16
17 find_package( OpenMP COMPONENTS C Fortran )
18 find_package( NetCDF REQUIRED COMPONENTS C Fortran )
19
20 add_subdirectory(src)
21 add_subdirectory(test)
22
23 set_target_properties( ${PROJECT_NAME} PROPERTIES LINKER_LANGUAGE CXX )
24
```

CMakeLists.txt

```
12 SET(OBJS
13   /global/panfs01/admin/opt/intel/oneAPI/2021.1.2/compiler/2021.1.2/linux/compiler/lib/intel64_lin/for_main.o
14 )
15
16 add_executable( ${PROJECT_NAME} ${OBJS} ${c_sw_src_files} )
```

src/CMakeLists.txt

OMP Offload – Approach ‘Loop’

Fortran OMP Offload

```
706 ... !$omp target parallel do private(i,j) map(to:js,je,is,ie,npj) &
707 ... !$omp map(to: va,dya,sin_sg,vtmp,u,cosa_v,rsin_v) &
708 ... !$omp map(tofrom: vt,vc)
709 ... do j = js-1, je+2
710 ...   if ( j == 1 ) then
711 ...     do i = is-1, ie+1
712 ...       vt(i,j) = edge_interpolate4(va(i,-1:2), dya(i,-1:2))
713 ...       if ( vt(i,j) < 0. ) then
714 ...         vc(i,j) = vt(i,j) * sin_sg(i,j-1,4)
715 ...       else
716 ...         vc(i,j) = vt(i,j) * sin_sg(i,j,2)
717 ...       end if
718 ...     enddo
719 ...   elseif ( j == 0 .or. j == (npj-1) ) then
720 ...     do i = is-1, ie+1
721 ...       vc(i,j) = c1 * vtmp(i,j-2) + c2 * vtmp(i,j-1) + c3 * vtmp(i,j)
722 ...       vt(i,j) = (vc(i,j) - u(i,j) * cosa_v(i,j)) * rsin_v(i,j)
723 ...     enddo
724 ...   elseif ( j == 2 .or. j == (npj+1) ) then
725 ...     do i = is-1, ie+1
726 ...       vc(i,j) = c1 * vtmp(i,j+1) + c2 * vtmp(i,j) + c3 * vtmp(i,j-1)
727 ...       vt(i,j) = (vc(i,j) - u(i,j) * cosa_v(i,j)) * rsin_v(i,j)
728 ...     enddo
```

DPC++ Offload

```
404 ... Q.parallel_for(static_cast<size_t>((je+2)-(js-1)+1),
405 ... [=](id<1> idx) {
406 ...   const int j = idx[0] + js-1;
407 ...   int i;
408 ...
409 ...   if ( j == 1 )
410 ...   {
411 ...     for (i = is-1; i <= ie+1; i++)
412 ...     {
413 ...       //vt(i,j) = edge_interpolate4(va(i,-1:2), dya(i,-1:2))
414 ...       double va_interp[4];
415 ...       double dya_interp[4];
416 ...       for (int k=0; k<=3; k++)
417 ...       {
418 ...         va_interp[k] = va(i,k-1);
419 ...         dya_interp[k] = dya(i,k-1);
420 ...       }
421 ...
422 ...       vt(i,j) = edge_interpolate4(va_interp, dya_interp);
423 ...       if ( vt(i,j) < 0. )
424 ...         vc(i,j) = vt(i,j) * sin_sg(i,j-1,4);
425 ...       else
426 ...         vc(i,j) = vt(i,j) * sin_sg(i,j,2);
427 ...     }
428 ...   }
```

OMP Offload – Approach ‘Outer’

Fortran OMP Offload

```
128  ..! Run the kernel
129  ..!$omp target map(to:sw_corner,se_corner,ne_corner,nw_corner,rarea) &
130  ..!$omp map(to:rarea_c,sin_sg,cos_sg,sina_v,cosa_v,sina_u,cosa_u,fC) &
131  ..!$omp map(to:rdxc,dy,dxc,rdyc,dx,dyc,cosa_s,rsin_u,rsin_v,rsin2) &
132  ..!$omp map(to:dxa,dya,dt2) map(from:delpc,ptc,wc,divg_d) &
133  ..!$omp map(tofrom:u,vc,v,uc,delp,pt,ua,va,ut,vt,w)
134  ..!$omp parallel do
135  ..do k=1,npz
136  ..call c_sw(sw_corner, se_corner, nw_corner, ne_corner, ..... &
137  ..... rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, ..... &
138  ..... sina_u, cosa_u, fC, rdxc, rdyc, dx, dy, dxc, dyc, ..... &
139  ..... cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, ..... &
140  ..... delpc(isd,jsd,k), delp(isd,jsd,k), ptc(isd,jsd,k), ..... &
141  ..... pt(isd,jsd,k), u(isd,jsd,k), v(isd,jsd,k), ..... &
142  ..... w(isd,jsd,k), uc(isd,jsd,k), vc(isd,jsd,k), ..... &
143  ..... ua(isd,jsd,k), va(isd,jsd,k), wc(isd,jsd,k), ..... &
144  ..... ut(isd,jsd,k), vt(isd,jsd,k), divg_d(isd,jsd,k), dt2)
145  ..enddo
146  ..!$omp end target
147
148  ..
```

DPC++ Offload

```
120  ..! Run the kernel
121  ..call c_sw_dpcpp(isd, ied, jsd, jed, is, ie, js, je, nord, npx, npy, ..&
122  ..... npz, do_profile, dt2, ..... &
123  ..... sw_corner, se_corner, nw_corner, ne_corner, ..... &
124  ..... rarea, rarea_c, sin_sg, cos_sg, sina_v, cosa_v, ..... &
125  ..... sina_u, cosa_u, fC, rdxc, rdyc, dx, dy, dxc, dyc, ..... &
126  ..... cosa_s, rsin_u, rsin_v, rsin2, dxa, dya, ..... &
127  ..... delpc, delp, ptc, pt, u, v, w, uc, vc, ua, va, wc, ..... &
128  ..... ut, vt, divg_d)
129
130  ..!$OMP parallel do schedule(runtime)
131  ..!do k=1,npz
132  ..!call c_sw(sw_corner, se_corner, nw_corner, ne_corner, .....&
133  ..... delpc(isd,jsd,k), delp(isd,jsd,k), ptc(isd,jsd,k), .....&
134  ..... pt(isd,jsd,k), u(isd,jsd,k), v(isd,jsd,k), .....&
135  ..... w(isd,jsd,k), uc(isd,jsd,k), vc(isd,jsd,k), .....&
136  ..... ua(isd,jsd,k), va(isd,jsd,k), wc(isd,jsd,k), .....&
137  ..... ut(isd,jsd,k), vt(isd,jsd,k), divg_d(isd,jsd,k), dt2)
138  ..!enddo
139
```

Performance Comparison

Testcase	No Offload (+OMP)		Approach 'Outer'			Approach 'Loop'		
	Fortran	C++	OMP	DPC++ GPU	DPC++ CPU	OMP	DPC++ GPU	DPC++ CPU
Default	0.003 s	0.005 s	7.95 s	1.094 s	2.378 s	4.32 s	3.175 s	0.225 s

- Test performed on Intel® DevCloud
- Running on a small test
- Timings exclude memory movement and show pure kernel time
- DPC++ is much slower than Fortran. Why? → 'just in time' compilation

Testing Date: Performance results are based on testing by Intel as of June 17, 2021 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® E-2176G Processor with Intel® UHD Graphics P630 on Intel® DevCloud. Ubuntu 20.04.2 LTS, Intel® oneAPI Base Toolkit 2021.2, Intel® oneAPI HPC Toolkit 2021.2. SENA c_sw kernel at https://github.com/NOAA-GSL/SENA-c_sw.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

Ahead of time compilation

- Device kernel code is compiled 'just in time' (JIT) at runtime
- A lot of device code → Takes a lot of time
- If hardware is known a priori: Pre-compile (ahead of time, AOT) device code
- Negligible for kernels that are called multiple times, or longer running kernels

CMakeLists.txt

```
5 cmake_minimum_required( VERSION 3.10 )
6
7 project( c_sw LANGUAGES Fortran C CXX )
8
9 list( APPEND CMAKE_MODULE_PATH ${CMAKE_CURRENT_SOURCE_DIR}/cmake )
10 set( CMAKE_DIRECTORY_LABELS ${PROJECT_NAME} )
11 #set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore" )
12 set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice \
13     -Xs -march=sse4.2" )
14 #set( CMAKE_EXE_LINKER_FLAGS "-g -lifcore -fsycl-targets=spir64_gen-unknown-unknown-sycldevice \
15     -Xs -device=gen9" )
16
17
18 enable_testing()
19
20 include(${PROJECT_NAME}_compiler_flags)
21
22 find_package( OpenMP COMPONENTS C Fortran )
23 find_package( NetCDF REQUIRED COMPONENTS C Fortran )
24
25 add_subdirectory(src)
26 add_subdirectory(test)
27
28 set_target_properties( ${PROJECT_NAME} PROPERTIES LINKER_LANGUAGE CXX )
```

```
16 add_executable( ${PROJECT_NAME} ${OBJS} ${c_sw_src_files} )
17
18 #target_compile_options( ${PROJECT_NAME} PUBLIC $<$<COMPILE_LANGUAGE:CXX>:-v \
19     -fsycl-targets=spir64_gen-unknown-unknown-sycldevice> )
20 target_compile_options( ${PROJECT_NAME} PUBLIC $<$<COMPILE_LANGUAGE:CXX>:-v \
21     -fsycl-targets=spir64_x86_64-unknown-unknown-sycldevice> )
```

src/CMakeLists.txt

Performance Comparison

Testcase	No Offload (+OMP)		Approach 'Outer'			Approach 'Loop'		
	Fortran	C++	OMP*	DPC++ GPU	DPC++ CPU	OMP*	DPC++ GPU	DPC++ CPU
Default	0.003 s	0.005 s	7.95 s	1.094 s	2.378 s	4.32 s	3.175 s	0.225 s
AOT	0.003 s	0.005 s	7.95 s	0.094 s	0.042 s	4.32 s	3.049 s	0.079 s

- Gains from AOT more prominent if there is more device code
- DPC++ outperforms OMP
- Offload still slower due to low GPU usage and the device used
- *Known issue with AOT, OMP offload and Gen9 graphics

Testing Date: Performance results are based on testing by Intel as of June 17, 2021 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® E-2176G Processor with Intel® UHD Graphics P630 on Intel® DevCloud. Ubuntu 20.04.2 LTS, Intel® oneAPI Base Toolkit 2021.2, Intel® oneAPI HPC Toolkit 2021.2. SENA c_sw kernel at https://github.com/NOAA-GSL/SENA-c_sw.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

Performance Comparison

Testcase	No Offload (+OMP)		Approach 'Outer'			Approach 'Loop'		
	Fortran	C++	OMP	DPC++ GPU	DPC++ CPU	OMP	DPC++ GPU	DPC++ CPU
Default	0.003 s	0.005 s	7.95 s	1.094 s	2.378 s	4.32 s	3.175 s	0.225 s
AOT	0.003 s	0.005 s	7.95 s	0.094 s	0.042 s	4.32 s	3.049 s	0.079 s
Large Case	4.239 s	3.95 s	11.66 s	16.33 s	3.94 s	60 s	7.73 s	9.29 s

- 'Large Case' test is ~330 times the size of 'Default' test
- AOT is also used in these tests (except OMP)

Testing Date: Performance results are based on testing by Intel as of June 17, 2021 and may not reflect all publicly available security updates.

Configuration Details and Workload Setup: Intel® Xeon® E-2176G Processor with Intel® UHD Graphics P630 on Intel® DevCloud. Ubuntu 20.04.2 LTS, Intel® oneAPI Base Toolkit 2021.2, Intel® oneAPI HPC Toolkit 2021.2. SENA c_sw kernel at https://github.com/NOAA-GSL/SENA-c_sw.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

Performance varies by use, configuration, and other factors. Learn more at www.intel.com/PerformanceIndex. Your costs and results may vary.

Summary

- Fortran – DPC++ migration possible and feasible
- Still requires Fortran – C++ migration for kernel code
- Leveraging C++ in kernel code and implicit data movement between host and device allows neat code and easy migration
- Performance in such a first approach might not be ideal
- One of the core strengths of DPC++ is C++

Acknowledgement

The NOAA HPCC Software Engineering for Novel Architectures (SENA) program and the participating organizations: the Geophysical Fluid Dynamics Laboratory, the Global Systems Laboratory and the Environmental Modeling Center.

Resources

- NOAA c_sw kernel - https://github.com/NOAA-GSL/SENA-c_sw
- Intel® oneAPI – <https://software.intel.com/oneapi>
- Intel® DevCloud for oneAPI – <https://devcloud.intel.com/oneapi>

