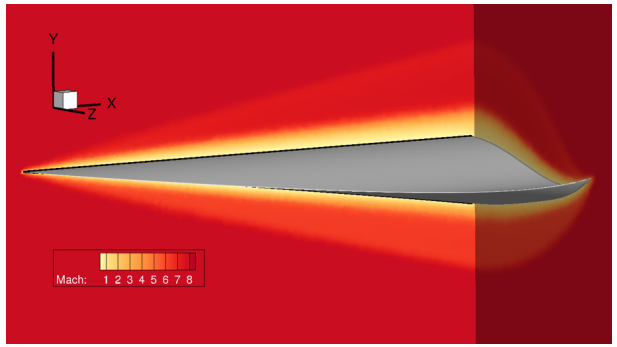
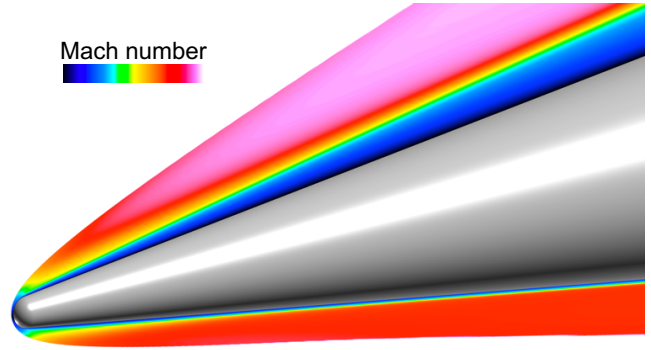
Enabling Execution of a Legacy CFD Mini Application on Accelerators Using OpenMP

June, 2020

I. Nompelis, G. Jost, A. Koniges, C. Daley , D. Eder and C. Stone





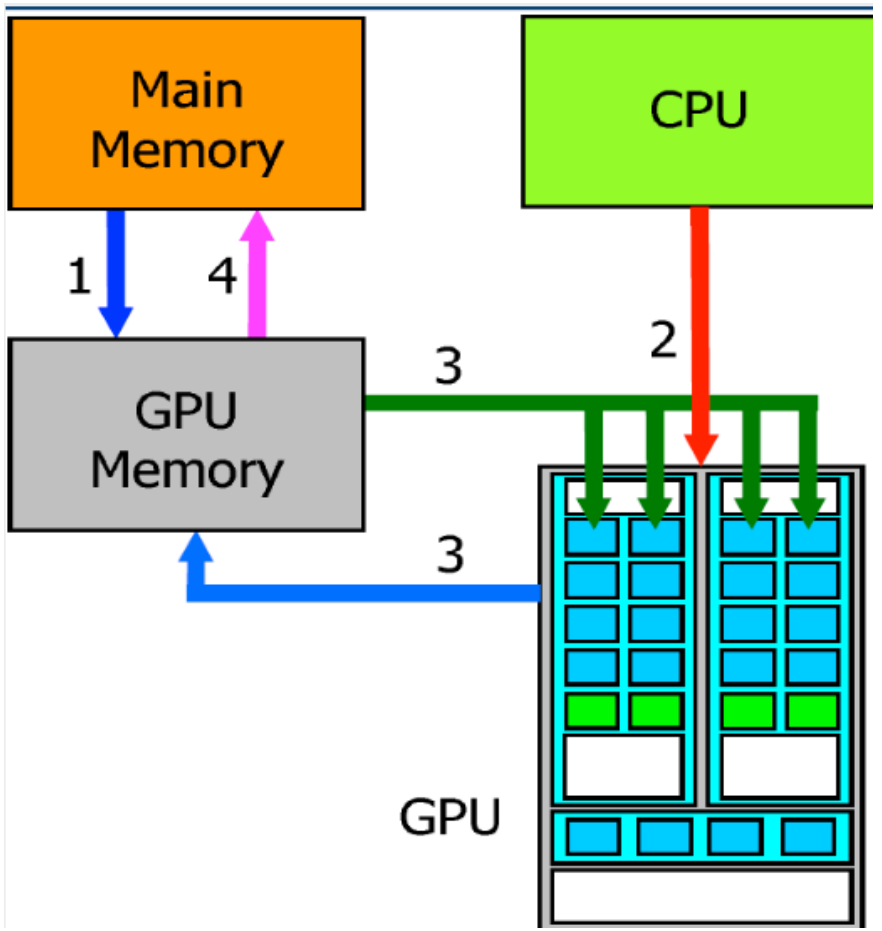
Outline

- Introduction
  - Concepts and Methods of Accelerator Programming
  - OpenMP 4.5 Accelerator programming support
- The NAS Parallel Benchmark SP-MZ
- Background
  - Numerical Method
- Porting Strategy and Optimizations
  - From OpenMP 3.0 to 4.5
  - OpenMP 4.5 Optimizations
- Testing Architectures
- Performance Results
- Conclusions

Device Accelerated Computing

- Device Accelerated Programming
  - Identify and off-load compute kernels
  - Express parallelism within the kernel
  - Manage data transfer between CPU and Device
- Execution flow
  - Data copy from main to device memory (1)
  - CPU initiates kernel for execution on the device (2)
  - Device executes the kernel using device memory (3)
  - Data copy from device to main memory (4)

Example:
Device is a GPU



Background on OpenMP

- Compiler directives, runtime library routines, and environment variables
- OpenMP 5.0 released at SC18
- Accelerator (GPU) programming support since OpenMP 4.0 : Identify kernels for offload, specify parallelism and manage data transfer between host and device
- GPU code builds on existing OpenMP code, makes strides towards portability
- FORTRAN OpenMP supported by many compilers such as gfortran, Cray ftn and IBM xlf. Full 5.0 support in progress
- OpenMP also takes a vendor neutral approach as the standard is written by a consortium of HPC compiler teams, universities and research laboratories

OpenMP Device Programming Support

OpenMP 4.0

target [data]
declare target
target update
simd
declare simd
loop simd
parallel loop simd
teams

distribute [simd]
distribute parallel for [simd]
teams distribute [simd]
teams distribute parallel for [simd]
target teams
target teams distribute [simd]
target teams distribute parallel for [simd]
… and other API Calls …

OpenMP 4.5

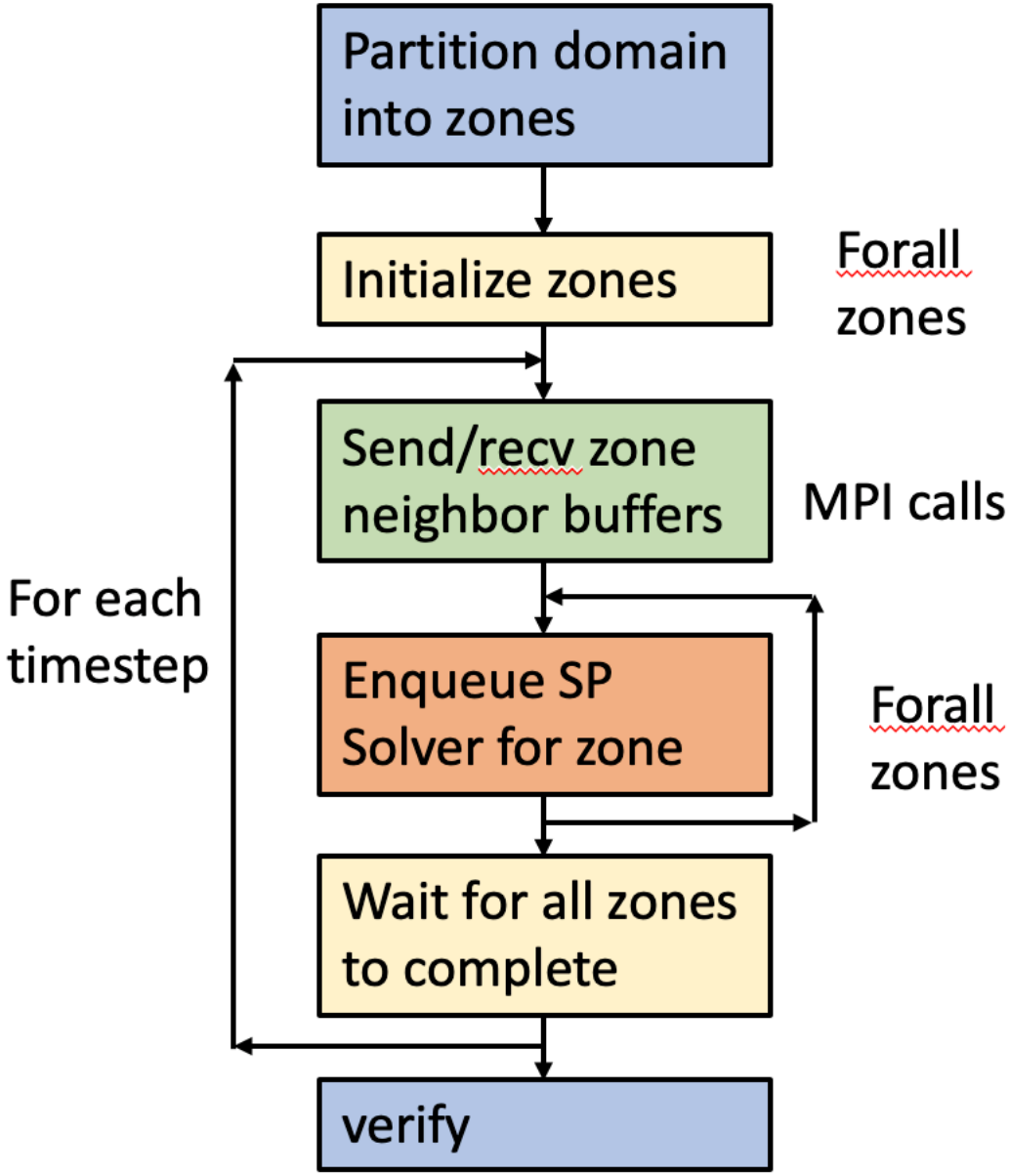
taskloop
taskloop simd
target enter data
target exit data
target simd
… Other API Calls …

OpenMP 5.0

allocate
declare mapper
Memory spaces
parallel loop
teams loop
… Other API Calls …

SP-MZ Benchmark Description

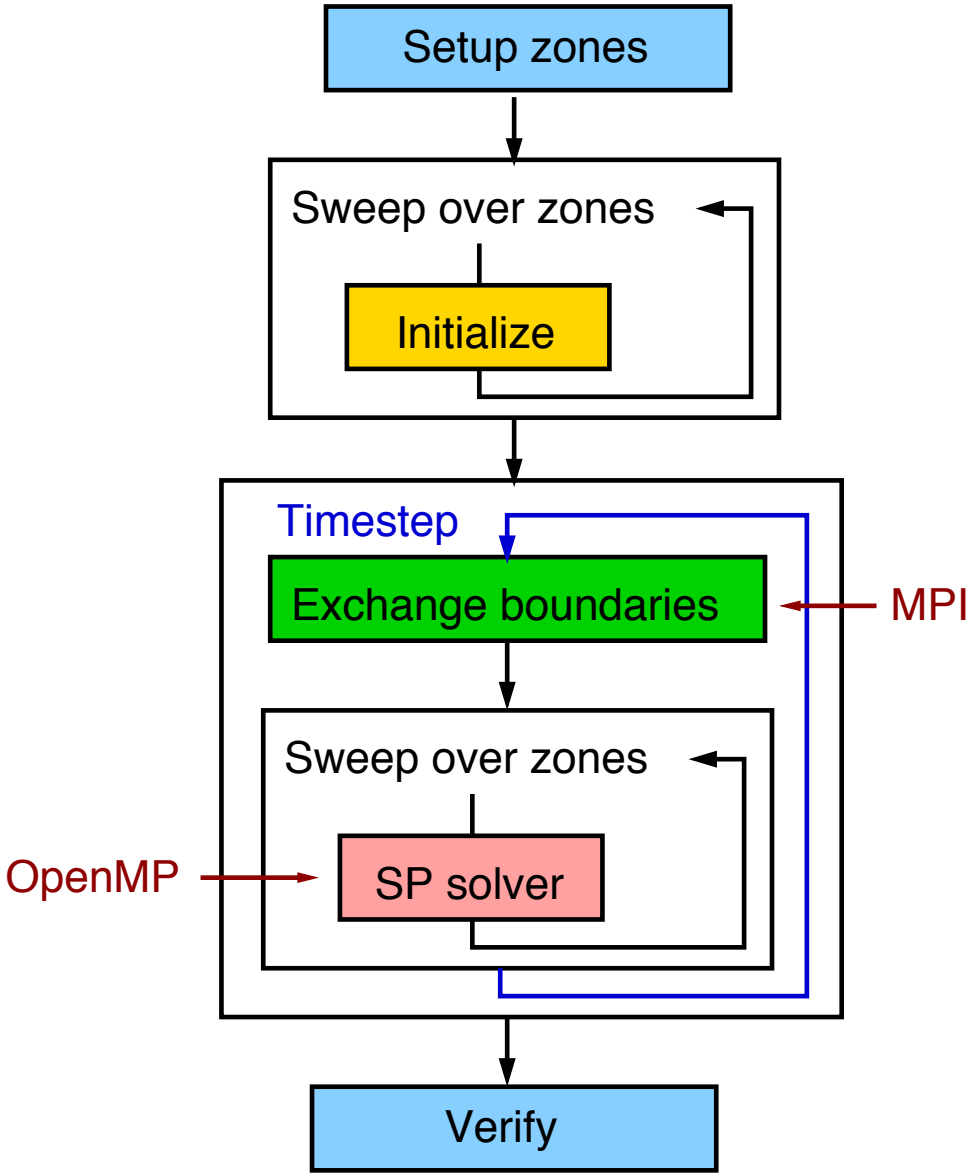
- Mini Application which mimics the performance of CFD applications
- It employs a diagonalized Beam-Warming alternating direction implicit (ADI) algorithm and a Scalar Penta-diagonal solver
- Explicit right-hand-side (RHS) vector with finite difference and the inversion of the **S**calar **P**entadiagonal (SP) matrices along each grid line in all three directional sweeps
- By design there are even-size zones within a problem class
- It is written in FORTRAN with OpenMP 3.0
- Employs distributed memory parallelism via MPI for inter-zone communication and shared multi-core parallelization within the zones
- Open source code with parallel patterns representative of many CFD and Aerospace codes



From OpenMP 3.0 to OpenMP 4.5

- **Goal: Keep Source changes minimal**
- **OpenMP 3.0 Directive Modifications:**
  - Decorate existing OpenMP constructs with off-load extensions
  - Add clauses to improve parallelism on the device
  - Add additional directives for performance improvement, eg mark variables as being available on the device

Execution flow of the SP-MZ Benchmark



CPU version (pink) – GPU version (green)

```
        implicit none

        integer nx, nxmax, ny, nz
-       double precision rhs(5,0:nxmax-1,0:ny-1,0:nz-1),
-     $                  u  (5,0:nxmax-1,0:ny-1,0:nz-1)
+       double precision rhs(0:nxmax-1,0:ny-1,0:nz-1,5),
+     $                  u  (0:nxmax-1,0:ny-1,0:nz-1,5)

        integer i,j,k,m

-!$OMP PARALLEL DO DEFAULT(SHARED) PRIVATE(m,i,j,k)
-!$OMP&  SCHEDULE(STATIC) COLLAPSE(2)
+!$OMP TARGET TEAMS DISTRIBUTE
+!$OMP&    PRIVATE( m,i,j,k )
        do k = 1, nz-2
           do j = 1, ny-2
+!$OMP        PARALLEL DO PRIVATE(i,m)
             do i = 1, nx-2
                do m = 1, 5
-                  u(m,i,j,k) = u(m,i,j,k) + rhs(m,i,j,k)
+                  u(i,j,k,m) = u(i,j,k,m) + rhs(i,j,k,m)
               end do
             end do
+!$OMP        END PARALLEL DO
          end do
        end do
-!$OMP END PARALLEL DO
+!$OMP END TARGET TEAMS DISTRIBUTE

        return
```

From OpenMP 3.0 to 4.5 Examples

```
!$OMP PARALLEL DO DEFAULT(SHARED)
!$OMP&  PRIVATE(fac2,m,fac1,i2,i1,ru1,i,j,k)
!$OMP&  SCHEDULE(STATIC) COLLAPSE(2)
 do k = 1, nz-2
    do j = 1, ny-2
       ……
       call lhsinit(lhs, lhsp, lhsm, nx-1)
       ....   !--- operations localized at "i"
       do i = 0, nx-3 !--- Thomas alg. forward elim.
          i1 = i + 1
          i2 = i + 2
          fac1      = 1.d0/lhs(3,i)
          lhs(4,i)  = fac1*lhs(4,i)
          lhs(5,i)  = fac1*lhs(5,i)
          do m = 1, 3
             rhs(m,i,j,k) = fac1*rhs(m,i,j,k)
          end do
          lhs(3,i1) = lhs(3,i1) -
                      lhs(2,i1)*lhs(4,i)
          lhs(4,i1) = lhs(4,i1) -
                      lhs(2,i1)*lhs(5,i)
          do m = 1, 3
             rhs(m,i1,j,k) = rhs(m,i1,j,k) -
                      lhs(2,i1)*rhs(m,i,j,k)
          end do
          ....
          lhsm(4,i1) = lhsm(4,i1) -
                       lhsm(2,i1)*lhsm(5,i)
          ....
       end do
       ....
    end do
 end do
```

```
!$OMP  TARGET TEAMS DISTRIBUTE
!$OMP&  PARALLEL DO SIMD COLLAPSE(2)
!$OMP&  NOWAIT DEPEND( inout: rhs )
!$OMP&    MAP( ALLOC: lhs4, lhsp4, lhsm4, rhst )
!$OMP&    PRIVATE(fac2,m,fac1,i2,i1,ru1,i)
!$OMP&    PRIVATE( cv_im1, cv_ip1 )
!$OMP&    PRIVATE( rhon_ip1, rhon_im1, rhon_i )
 do k = 1, nz-2
    do j = 1, ny-2

       lhs4 (j,1:5,   0,k) = 0.0d0
       lhs4 (j,1:5,nx-1,k) = 0.0d0
       ....   !--- operations localized at "i"

       do i = 0, nx-3    !--- Thomas alg. forward elim.
          i1 = i  + 1
          i2 = i  + 2
          fac1          = 1.d0/lhs4(j,3,i,k)
          lhs4(j,4,i,k) = fac1*lhs4(j,4,i,k)
          lhs4(j,5,i,k) = fac1*lhs4(j,5,i,k)
          do m = 1, 3
             rhst(j,m,i,k) = fac1*rhst(j,m,i,k)
          end do
          lhs4(j,3,i1,k) = lhs4(j,3,i1,k) -
                           lhs4(j,2,i1,k)*lhs4(j,4,i,k)
          ....
       end do
       ....
    end do
 end do
```

Source Code Modifications to Improve Device Parallelism and Memory Access

- **Optimizations**
  - In-lining of small routines:
    - Improve compiler dependence analysis
  - Use of the **“omp declare”** construct
  - Re-organize some data structures to allow stride 1 memory access
  - Avoid using arrays private to a thread and rather use shared arrays with higher dimensionality
  - Use pre-allocated scratch space rather than dynamic allocation
  - Exploit limited implicit zone-level parallelism by enabling asynchronous kernel execution via OpenMP tasking

Cray system at Berkeley (Cori-GPU)

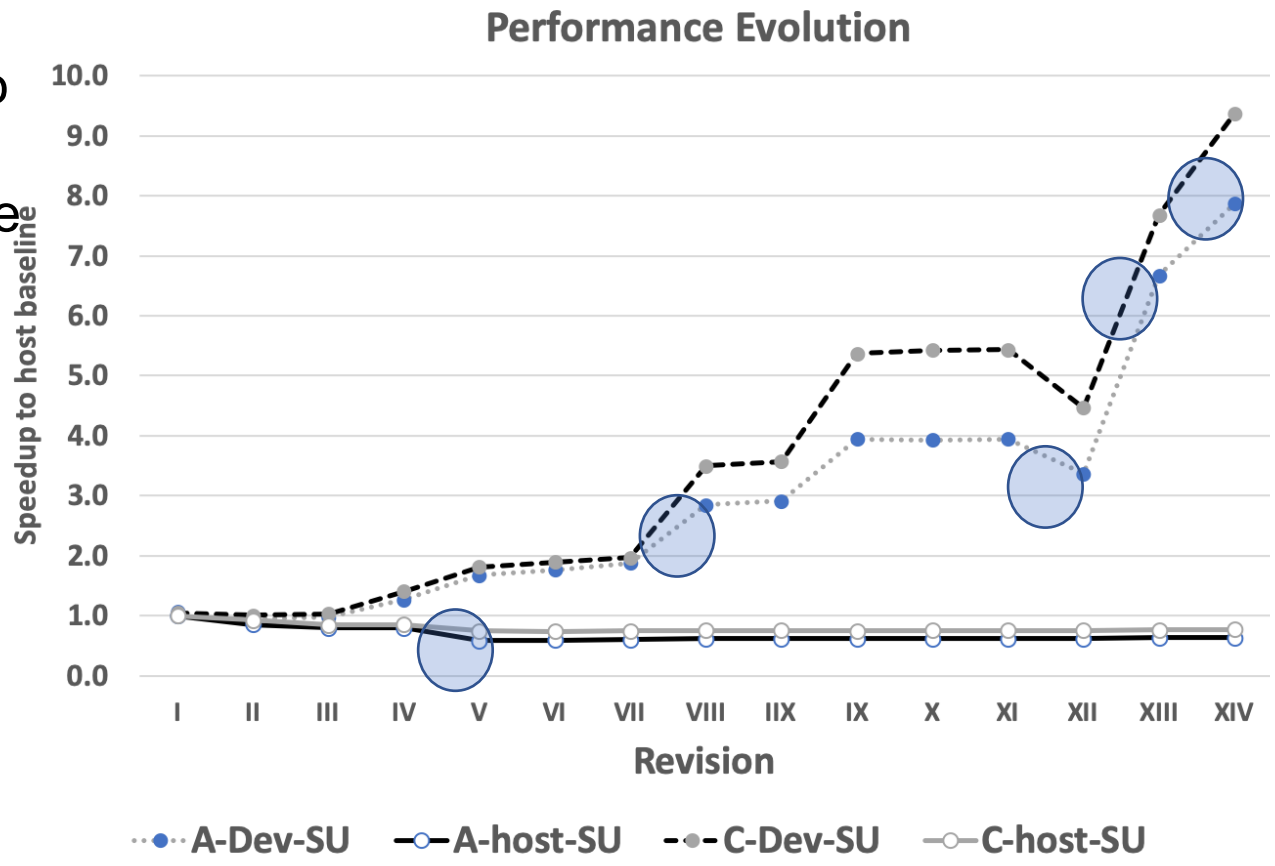
- Each node contains 8 NVIDIA V100 GPUs connected to each other in a 'hybrid cube-mesh' topology.
- Each GPU is connected directly to 4 others with NVLINK.
- All GPUs are connected to the Skylake CPUs and the Infiniband network interface cards (NICs) via PCIe 3.0.

IBM system at Oak Ridge (Ascent/Summit)

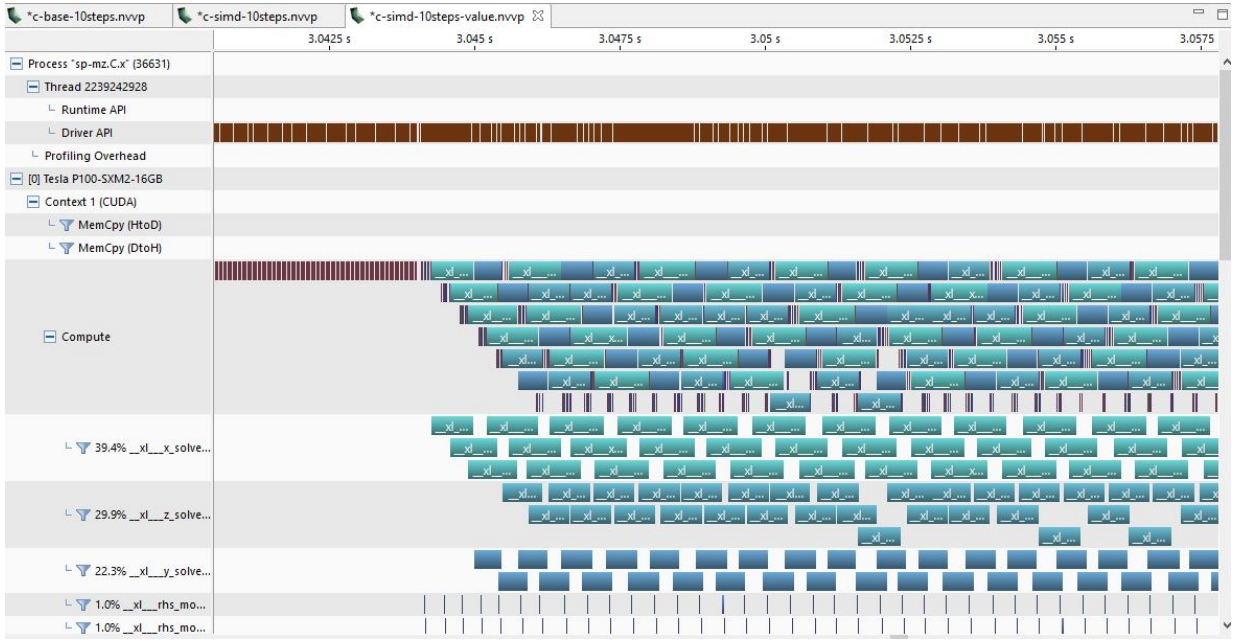
- Each node contains two IBM POWER9 processors and six NVIDIA V100 GPUs.
- Each POWER9 processor is connected via dual NVLINK bricks, each capable of a 25GB/s transfer rate in each direction.
- The POWER9 processor is built around IBM’s SIMD Multi-Core (SMC).
- The processor provides 22 SMCs with separate 32kB L1 data and instruction caches.
- Pairs of SMCs share a 512kB L2 cache and a 10MB L3 cache.

Performance Evolution on the Ascent (IBM) System

- Reduced or eliminated unnecessary host-to-device transfers, eg:
  - Eliminate unnecessary **"firstprivate"** clauses
  - Change argument passing from by-reference to by-value to allow the **IBM OpenMP v4.5 compiler to avoid unnecessary** host-to-device transfers; **No impact on Cray systems**
- Enabled asynchronous execution of some of the kernels
- Added CUDA-aware MPI option to bypass host transfers during MPI communication.
- Added SIMD to the PARALLEL DO ~ directive to improve performance **on the CRAY; Note: Not portable to IBM!**
- Change parallelism from within each zone to all zones at once to allow for partial kernel overlap



Tools such as nvprof used for kernel execution timelines



Compilation using the Cray compiler

```
ftn -openmp
```

Compilation using the IBM compiler

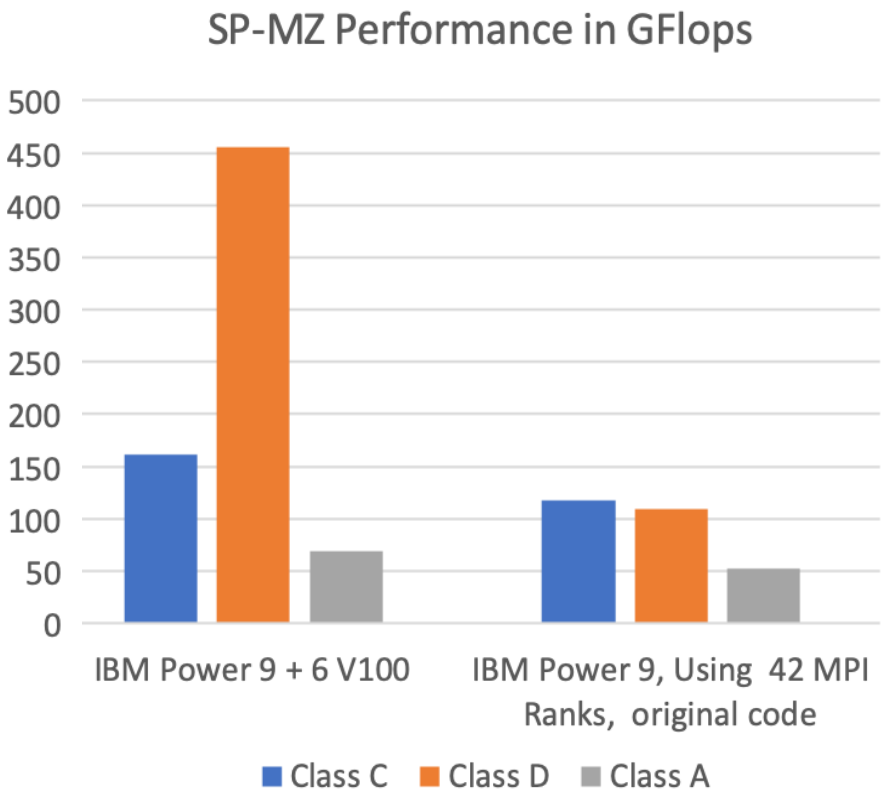
```
xlf -qsmp -qoffload
```

- **IBM compiler introduced unnecessary memory copies of constants during offloading**

- **Cray compiler did not work with MPI implementation on Cori-GPU system**

Conclusions

- OpenMP target off-load is an excellent programming API for porting legacy FORTRAN codes to the GPU
- Changes to increase GPU performance sometimes results in slightly worse CPU performance, more comparisons required
- Exploring newly developed larger size classes more appropriate for real applications and modern architectures that should perform well on GPUs
- OpenMP 5.0 features allow for additional optimization on the GPUs. Compilers and tools are still immature for FORTRAN



Backup Material

Code transformations of our initial implementation

- We manually in-lined routines called within OpenMP target regions.
- We transposed some of the arrays to allow for stride one memory access to enables simultaneous loads and stores.
- We used the OpenMP COLLAPSE clause to collapse as many loops as possible.
- We found that OpenMP PRIVATE arrays per thread gave poor performance because of the large size of the arrays.
- Therefore, we made the array shared. This was accomplished by adding two extra dimensions.
- To exploit some of the available zone-level parallelism within the code, we enabled asynchronous kernel execution using deferred OpenMP target tasks with dependencies.
- This is why the directive in the code listing contains the NOWAIT and DEPEND clauses.

Optimizations include:

- Changes to the memory layout of the arrays
- Addition of extra array dimensions to eliminate need for threadprivate data
- Reordering of array dimensions
- Loop collapsing
- Replacing unnecessary OpenMP firstprivate clauses with private, in order to avoid unnecessary HtD data transfers
- Using OpenMP nowait to enable asynchronous execution to allow for overlap of kernel execution

Further Optimizations

- We eliminated some small host-to-device data transfers by using the DECLARE TARGET construct
- Instead of a kernel for each rind copy in/out in “exch-bc” for each zone, we created a target region over the zone loop so all the copy in/out operations run within the same kernel
- We pre-allocated a large temporary array for all the lhs structures on all zones and passed them to the x, y and z solver routines
- We merged asynchronous kernels with communication optimizations
- We also combined the PARALLEL DO directives with TARGET TEAMS DISTRIBUTE since this enables the compiler to generate simpler code where all GPU threads execute the same computation on different data.