NCAR is sponsored by
National Science Foundation

Hands-On Session Using OpenACC in MPAS-A

By: Daniel Howard dhoward@ucar.edu, Consulting Services Group, CISL & NCAR

Date: April 28th 2022

In this notebook, we explore the GPU enabled MPAS-A (Model Prediction Across Scales-Atmosphere) to apply techniques learned from MiniWeather and implementing OpenACC to develop for GPUs.

- Review of exercises from prior OpenACC/MiniWeather sessions Part 1 and Part 2
- MPAS-Atmosphere model overview
- Managing GPU data in large software projects
- Assessing performance of extracted GPU kernels in MPAS-A

Head to the NCAR JupyterHub portal and **start a JupyterHub session on Casper login** (or batch nodes using 1 CPU, no GPUs) and open the notebook in `XXXX`. Be sure to clone (if needed) and update/pull the NCAR GPU_workshop directory.

```
# Use the JupyterHub GitHub GUI on the left panel or the below shell commands
git clone git@github.com:NCAR/GPU_workshop.git
git pull
```

Workshop Etiquette

- Please mute yourself and turn off video during the session.
- Questions may be submitted in the chat and will be answered when appropriate. You may also raise your hand, unmute, and ask questions during Q&A at the end of the presentation.
- By participating, you are agreeing to UCAR's Code of Conduct
- Recordings & other material will be archived & shared publicly.
- Feel free to follow up with the GPU workshop team via Slack or submit support requests to support.ucar.edu
    - Office Hours: Asynchronous support via Slack or schedule a time with an organizer

Notebook Setup

Set the `PROJECT` code to a currently active project, ie `UCIS0004` for the GPU workshop, and `QUEUE` to the appropriate routing queue depending on if during a live workshop session (`gpuworkshop`), during weekday 8am to 5:30pm MT (`gpudev`), or all other times (`casper`). Due to limited shared GPU resources, please use `GPU_TYPE=gp100` during the workshop. Otherwise, set `GPU_TYPE=v100` (required for `gpudev`) for independent work. See Casper queue documentation for more info.

Notebook Setup

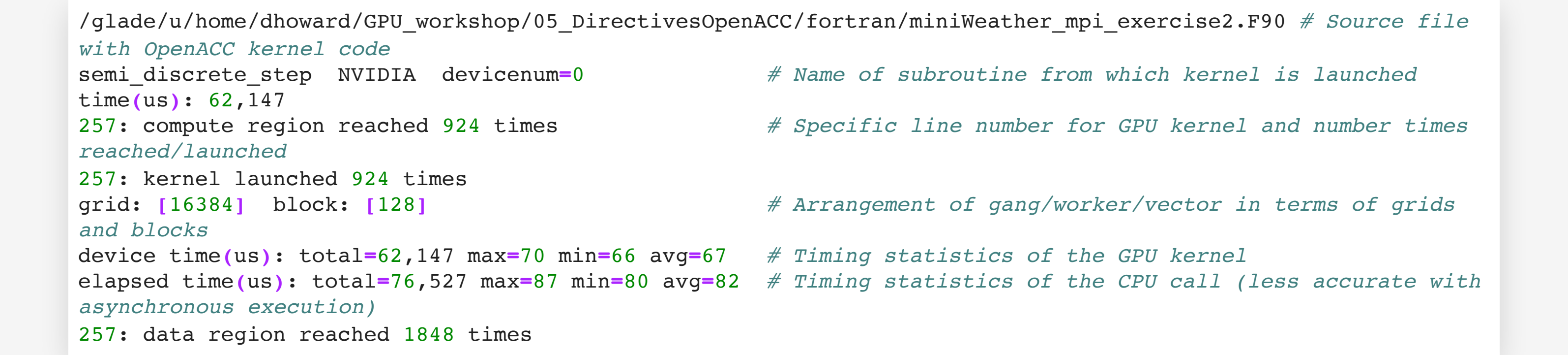
Set the `PROJECT` code to a currently active project, ie `UCIS0004` for the GPU workshop, and `QUEUE` to the appropriate routing queue depending on if during a live workshop session (`gpuworkshop`), during weekday 8am to 5:30pm MT (`gpudev`), or all other times (`casper`). Due to limited shared GPU resources, please use `GPU_TYPE=gp100` during the workshop. Otherwise, set `GPU_TYPE=v100` (required for `gpudev`) for independent work. See Casper queue documentation for more info.

In [ ]:

```
export PROJECT=UCIS0004
export QUEUE=gpudev
export GPU_TYPE=v100
```

Review of MiniWeather Performance Optimization

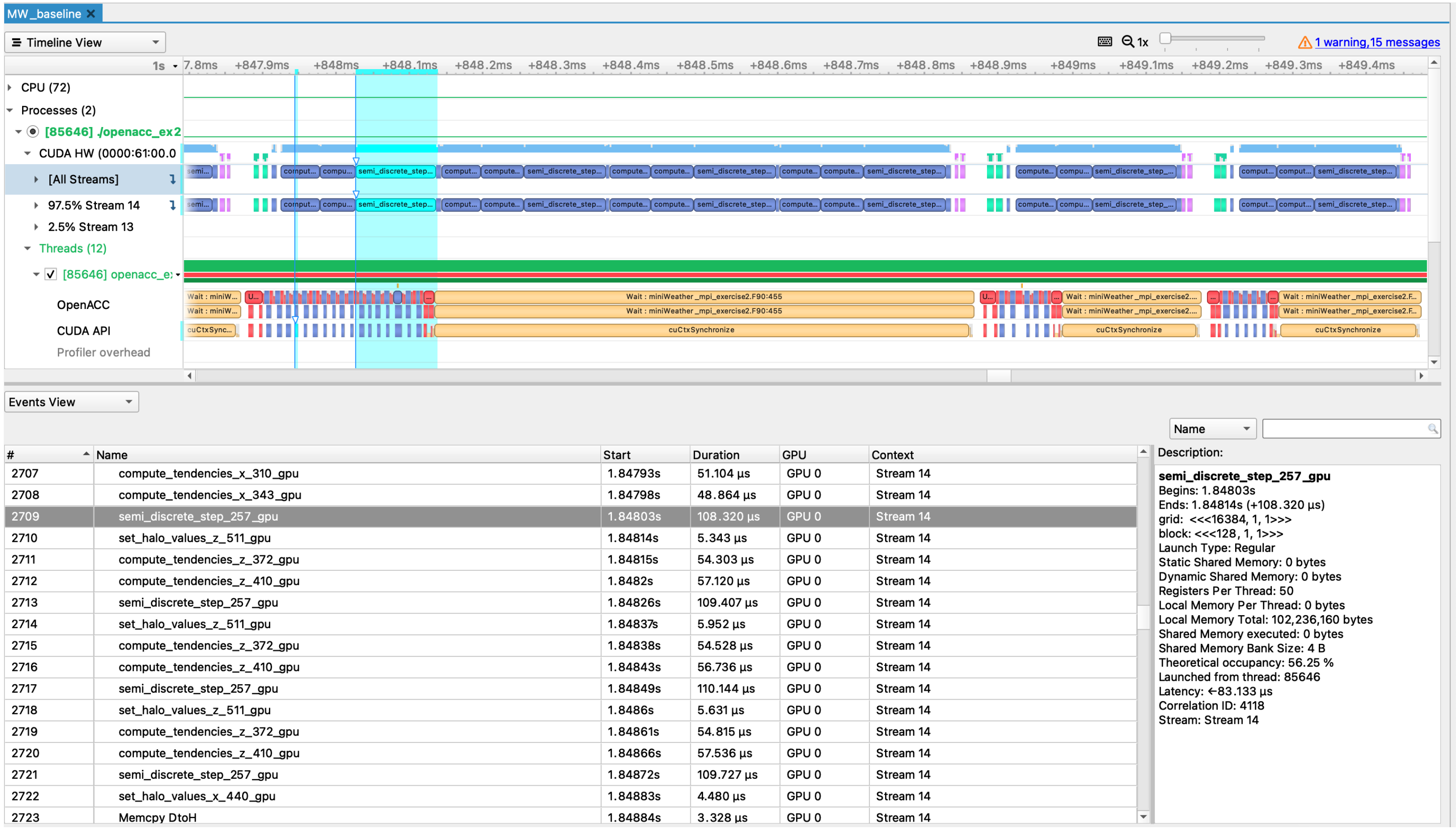
At the end of last session, it was suggested to use `async` and predominantly `collapse` clauses to achieve optimal performance in MiniWeather kernels. Using `NX=1024` and `NZ=512`, the most expensive kernel in terms of compute time was at **Line 231** in the `semi_discrete_step` subroutine, with `NVCOMPILER_ACC_TIME` statistics highlighted below:



```
/glade/u/home/dhoward/GPU_workshop/05_DirectivesOpenACC/fortran/miniWeather_mpi_exercise2.F90 # Source file
with OpenACC kernel code
semi_discrete_step  NVIDIA  devicenum=0              # Name of subroutine from which kernel is launched
time(us): 62,147
257: compute region reached 924 times                # Specific line number for GPU kernel and number times
reached/launched
257: kernel launched 924 times
grid: [16384]  block: [128]                          # Arrangement of gang/worker/vector in terms of grids
and blocks
device time(us): total=62,147 max=70 min=66 avg=67   # Timing statistics of the GPU kernel
elapsed time(us): total=76,527 max=87 min=80 avg=82  # Timing statistics of the CPU call (less accurate with
asynchronous execution)
257: data region reached 1848 times
```

The arrangement of **gang/worker/vector** units is provided by **grid: [NUM_GANGS]** and **block: [VECTOR_LENGTH x NUM_WORKERS]**. The number of workers was 1 in the previous case so is omitted.

Running this version with the NVIDIA NSight Systems Profiler (discussed in later session), we can get a visual representation of the model runtime. You can download and view this profile using the NVIDIA NSight Systems client by downloading (`SHIFT` + `RIGHT-CLICK`) MW_baseline.nsys-rep in this folder.



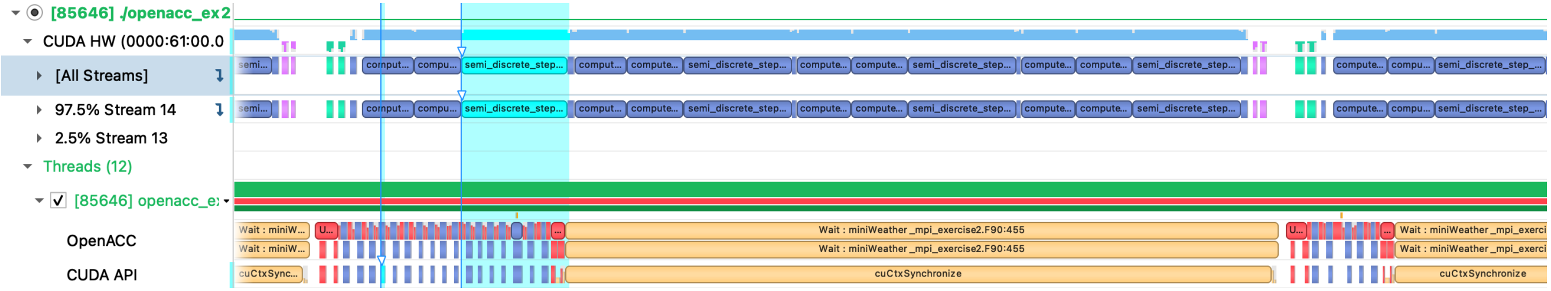
#	Name	Start	Duration	GPU	Context
2707	compute_tendencies_x_310_gpu	1.84793s	51.104 µs	GPU 0	Stream 14
2708	compute_tendencies_x_343_gpu	1.84798s	48.864 µs	GPU 0	Stream 14
2709	semi_discrete_step_257_gpu	1.84803s	108.320 µs	GPU 0	Stream 14
2710	set_halo_values_z_511_gpu	1.84814s	5.343 µs	GPU 0	Stream 14
2711	compute_tendencies_z_372_gpu	1.84815s	54.303 µs	GPU 0	Stream 14
2712	compute_tendencies_z_410_gpu	1.8482s	57.120 µs	GPU 0	Stream 14
2713	semi_discrete_step_257_gpu	1.84826s	109.407 µs	GPU 0	Stream 14
2714	set_halo_values_z_511_gpu	1.84837s	5.952 µs	GPU 0	Stream 14
2715	compute_tendencies_z_372_gpu	1.84838s	54.528 µs	GPU 0	Stream 14
2716	compute_tendencies_z_410_gpu	1.84843s	56.736 µs	GPU 0	Stream 14
2717	semi_discrete_step_257_gpu	1.84849s	110.144 µs	GPU 0	Stream 14
2718	set_halo_values_z_511_gpu	1.8486s	5.631 µs	GPU 0	Stream 14
2719	compute_tendencies_z_372_gpu	1.84861s	54.815 µs	GPU 0	Stream 14
2720	compute_tendencies_z_410_gpu	1.84866s	57.536 µs	GPU 0	Stream 14
2721	semi_discrete_step_257_gpu	1.84872s	109.727 µs	GPU 0	Stream 14
2722	set_halo_values_x_440_gpu	1.84883s	4.480 µs	GPU 0	Stream 14
2723	Memcpy DtoH	1.84884s	3.328 µs	GPU 0	Stream 14

Description:

semi_discrete_step_257_gpu
Begins: 1.84803s
Ends: 1.84814s (+108.320 µs)
grid: <<<16384, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 50
Local Memory Per Thread: 0 bytes
Local Memory Total: 102,236,160 bytes
Shared Memory executed: 0 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 56.25 %
Launched from thread: 85646
Latency: ←83.133 µs
Correlation ID: 4118
Stream: Stream 14

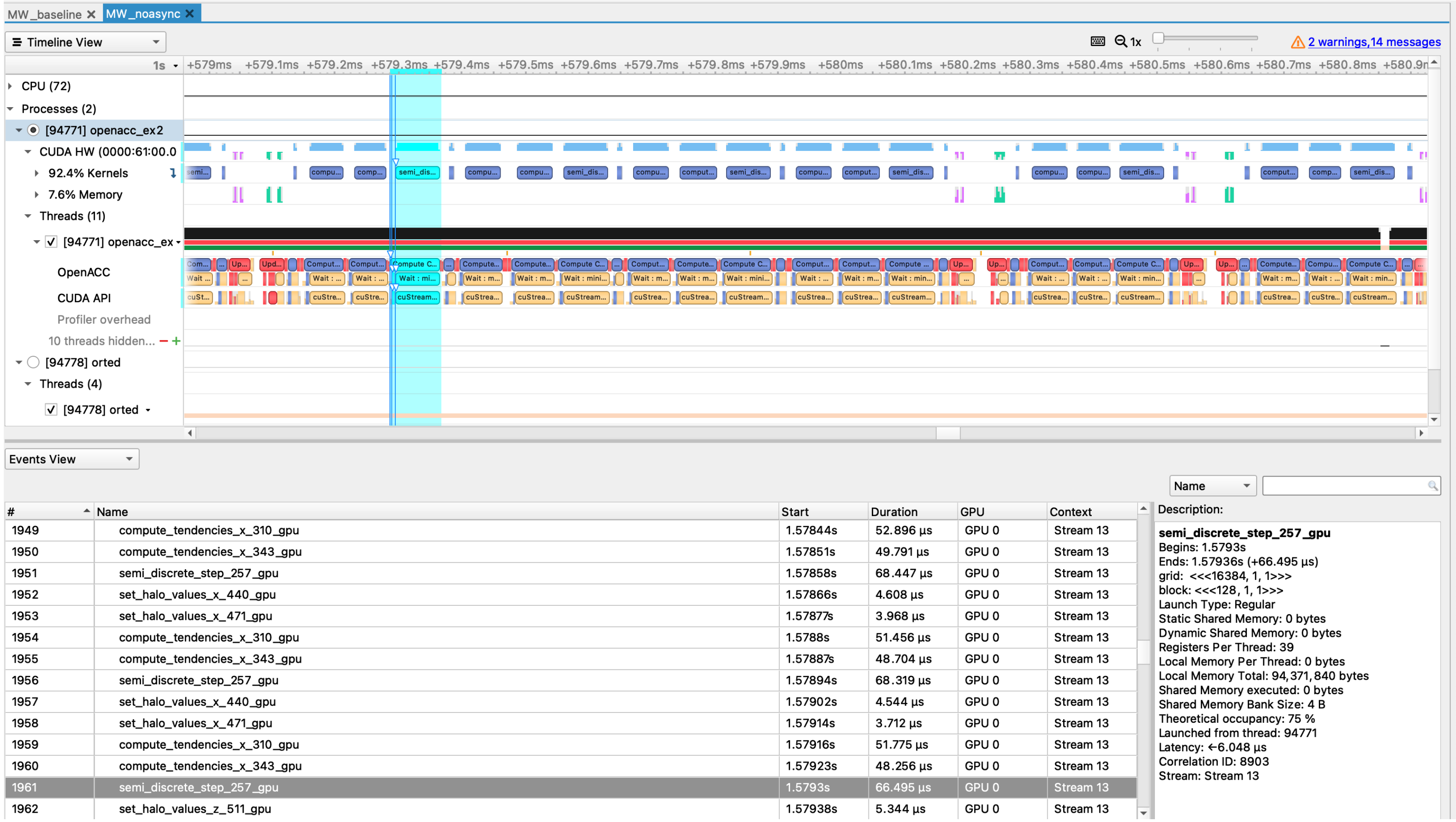
This timeline shows the kernels running on the GPU runtime in the upper **blue** compute kernels, **pink** device to host transfers, and **teal** host to device transfers segments. The lower segments show the CPU runtime in **blue** compute kernel launches, **red** data directives/regions, and beige **wait/synchronize** sections.

The bright blue highlights the most expensive GPU kernel in the `semi_discrete_step` subroutine with the associated launch call from the CPU highlighted earlier in the timeline.



Since we used `async`, the GPU kernels run right after one another without any kernel launch/exit costs.

If we did not use `async`, the profile would look like this (MW_noasync.qdrep) and time would be lost as the CPU waits between every kernel launch/exit.



#	Name	Start	Duration	GPU	Context
1949	compute_tendencies_x_310_gpu	1.57844s	52.896 µs	GPU 0	Stream 13
1950	compute_tendencies_x_343_gpu	1.57851s	49.791 µs	GPU 0	Stream 13
1951	semi_discrete_step_257_gpu	1.57858s	68.447 µs	GPU 0	Stream 13
1952	set_halo_values_x_440_gpu	1.57866s	4.608 µs	GPU 0	Stream 13
1953	set_halo_values_x_471_gpu	1.57877s	3.968 µs	GPU 0	Stream 13
1954	compute_tendencies_x_310_gpu	1.5788s	51.456 µs	GPU 0	Stream 13
1955	compute_tendencies_x_343_gpu	1.57887s	48.704 µs	GPU 0	Stream 13
1956	semi_discrete_step_257_gpu	1.57894s	68.319 µs	GPU 0	Stream 13
1957	set_halo_values_x_440_gpu	1.57902s	4.544 µs	GPU 0	Stream 13
1958	set_halo_values_x_471_gpu	1.57914s	3.712 µs	GPU 0	Stream 13
1959	compute_tendencies_x_310_gpu	1.57916s	51.775 µs	GPU 0	Stream 13
1960	compute_tendencies_x_343_gpu	1.57923s	48.256 µs	GPU 0	Stream 13
1961	semi_discrete_step_257_gpu	1.5793s	66.495 µs	GPU 0	Stream 13
1962	set_halo_values_z_511_gpu	1.57938s	5.344 µs	GPU 0	Stream 13

Description:

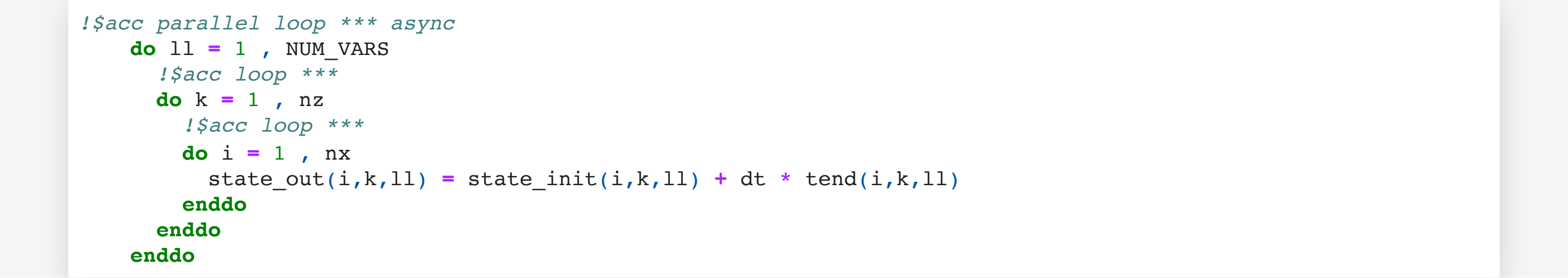
semi_discrete_step_257_gpu
Begins: 1.5793s
Ends: 1.57936s (+66.495 µs)
grid: <<<16384, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 39
Local Memory Per Thread: 0 bytes
Local Memory Total: 94,371,840 bytes
Shared Memory executed: 0 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 75 %
Launched from thread: 94771
Latency: ←6.048 µs
Correlation ID: 8903
Stream: Stream 13

MiniWeather - Testing different kernel launch configurations and clauses

Recall the final exercise of the prior MiniWeather session where we experimented with various launch configurations in the miniWeather_mpi_exercise2.F90 source file for specific kernels.

Were you able to achieve any significant speed-up?

The next panels shows statistical results from some launch configuration experiments using parameters `_NX=1024`, `_NZ=512`, and `_SIM_TIME=10` and different clauses in place of `***` for the `semi_discrete_step` subroutine kernel. Note that `NUM_VARS=4`.



```
!$acc parallel loop *** async
  do ll = 1 , NUM_VARS
   !$acc loop ***
   do k = 1 , nz
    !$acc loop ***
    do i = 1 , nx
     state_out(i,k,ll) = state_init(i,k,ll) + dt * tend(i,k,ll)
    enddo
   enddo
  enddo
```

1. **Using `worker/vector/seq` on each loop respectively, the profiler shows `grid: [1]  block: [32x4]`. Why is this arrangement the least performant?**

MiniWeather Kernel L231, `semi_discrete_step`	**Total Device Time (μs)**
BaseLine (on V100) - `collapse(3)` auto `vector_length(128)`	62,936
clause - `gang/worker/vector` on each loop resepctively	852,859
clause - `worker/vector/seq` (Move NUM_VARS innermost, seq)	2,271,059
clause - `gang/vector/seq` (Move NUM_VARS innermost, seq)	72,584

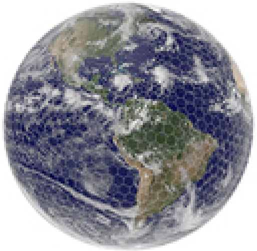
1. **Did you find any better configurations for this or other kernels in MiniWeather? Explain why it performed better.**
2. **Do you trust the compiler to make relatively optimal choices with minimal direction?**

MiniWeather Kernel L231, `semi_discrete_step`	**Total Device Time (μs)**
BaseLine (on V100) - `collapse(3)` auto `vector_length(128)`	62,936
clause - `collapse(3) vector_length(32)`	100,797
clause - `collapse(3) vector_length(64)`	63,010
clause - `collapse(3) vector_length(256)`	62,990
clause - `collapse(3) vector_length(512)`	63,032
clause - `collapse(3) vector_length(1024)`	66,458

1. **For `tile()`, why do you think the `(32,1,NUM_VARS=4)` clause was closest to the most performant?**
2. **Can you infer the condition that causes the `tile()` clause to produce incorrect results?** Hint: What is the max warp size?

MiniWeather Kernel L231, `semi_discrete_step`	**Total Device Time (μs)**
BaseLine (on V100) - `collapse(3)` auto `vector_length(128)`	62,936
clause - `tile(32,32,NUM_VARS)` **INCORRECT**	26,992
clause - `tile(32,32,1)`	73,476
clause - `tile(32,8,NUM_VARS)`	77,124
clause - `tile(32,1,NUM_VARS)`	65,040
clause - `tile(1024,1,1)`	67,393
clause - `tile(128,1,NUM_VARS)`	66,421
clause - `tile(128,2,NUM_VARS)`	74,295
clause - `tile(128,4,NUM_VARS)` **INCORRECT**	35,999
clause - `tile(*,*,*)` -> 32,4,32	150,374

MPAS-Atmosphere Overview



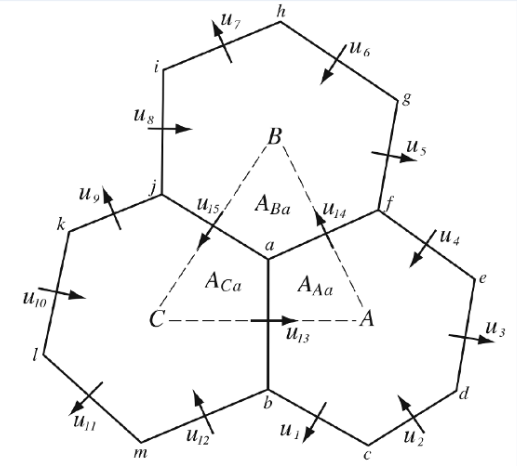
We will now look at a real world production model **MPAS (Model Prediction Across Scales)**, specifically the GPU version of the atmosphere core **MPAS-A** and how this model leveraged OpenACC to refactor towards GPU devices.

So far, **only the v6.x Atmosphere core has been ported to GPUs** and is freely available to review via their website and the stable v6.x or v7.x develop-openacc branches on GitHub. Some work has also been done on the MPAS-Ocean core given this presentation by PhD student Ashwath Venkataraman.

If you'd like a more complete overview of MPAS, how to run the model, and research applications, see the 2021 MPAS Virtual tutorial page or the upcoming 2022 joint WRF/MPAS workshop.

- Fully compressible non-hydrostatic equations written in flux form
- Split-Explicit timestepping via 3rd Order Runge-Kutta, see AMS Paper - Klemp, Skamarock, and Dudhia

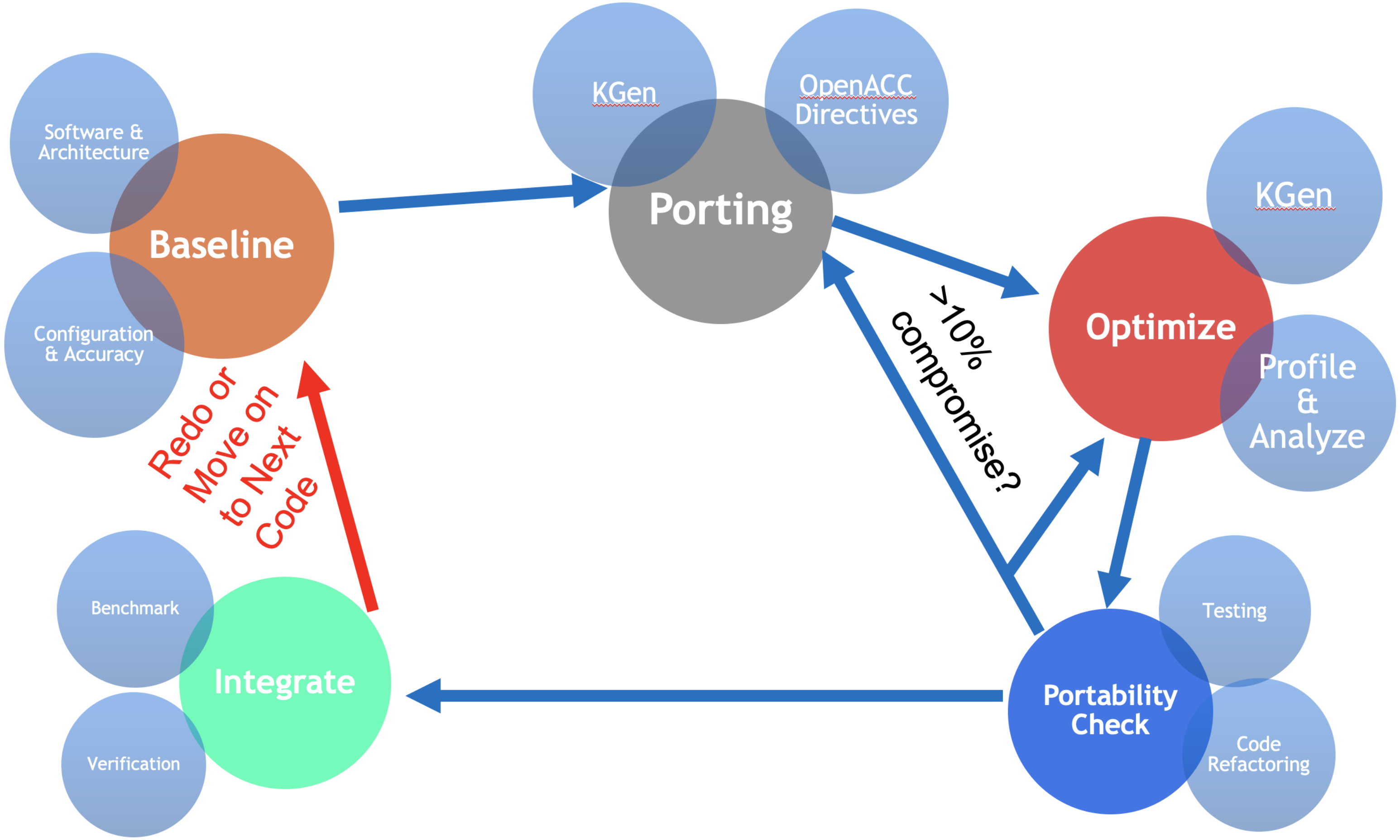
MPAS is based on unstructured centroidal Voronoi (hexagonal) meshes using C-grid staggering and selective grid refinement.



The MPAS-A kernels we will focus on computes coefficients for vertically implicit gravity-wave/acoustic computations needed for each Runge-Kutta timestep. The previously linked paper, specifically section 2 and the appendix, covers this in depth with a broader overview given in the 2021 tutorial Time Integration presentation.

However, understanding the numerical physics at play is not required to port well written code to GPUs.

Development Process of MPAS-A



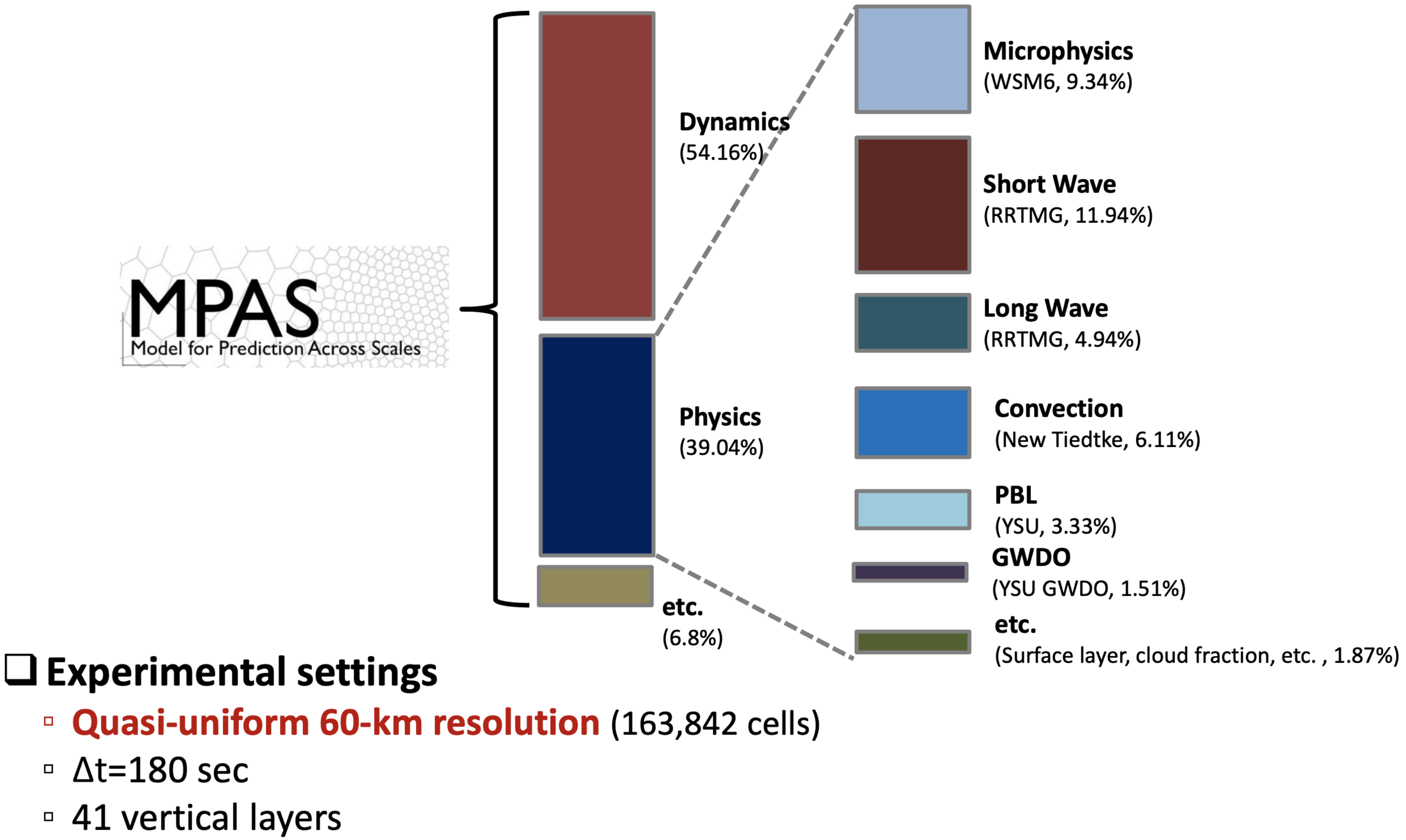
Courtesy of Raghu Raj Kumar, NVIDIA

Identifying an established iterative process for GPU development ahead of work performed significantly eases development cost and increases success outcomes.

1. **Establish a baseline**, ensure working and accurate configuration with target hardware and external software.
2. **Port the code**, using incremental addition of OpenACC, perhaps using tools for kernel extraction like KGen (Fortran only) to allow separation of concerns.
    - See KGen Guide if interested
3. **Optimize computationally expensive kernels** individually via an analysis and profiling iterative process.
4. **Check portability expectations** are met and that code satisfies both CPU and GPU unit tests.
    - Look for and eliminate any **GPU anti-patterns** such as linked lists data structures or global memory variables which may cause excessive data movement.
    - Repeat Steps 2-4 as needed.
5. **Integrate changes into benchmarks and verification suite**, utilizing version control and ideally a continuous integration process.

MPAS-A Performance Baseline

Getting an accurate baseline helps inform where to dedicate development effort. This can be measured using internal timing metrics or your preferred CPU profiler (like TAU, Arm Forge Map, *gprof*, etc), to **identifiy hotspots** in the code.

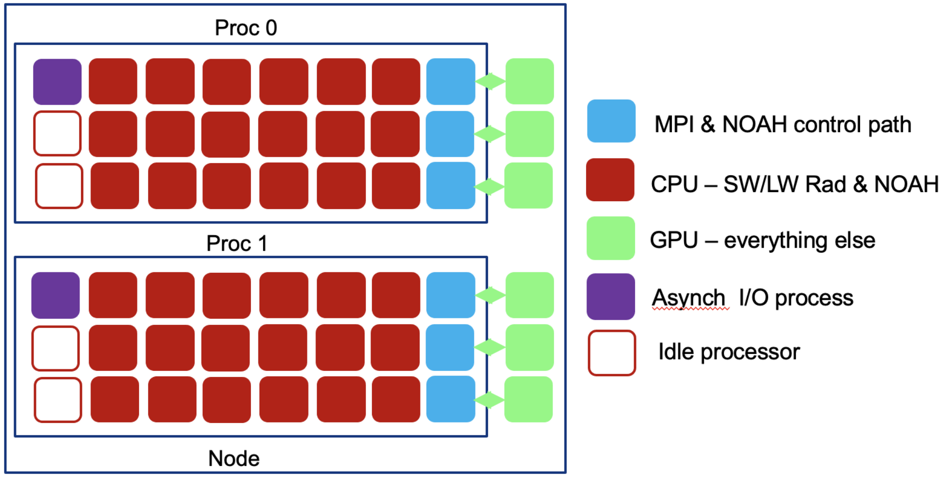


❑ **Experimental settings**

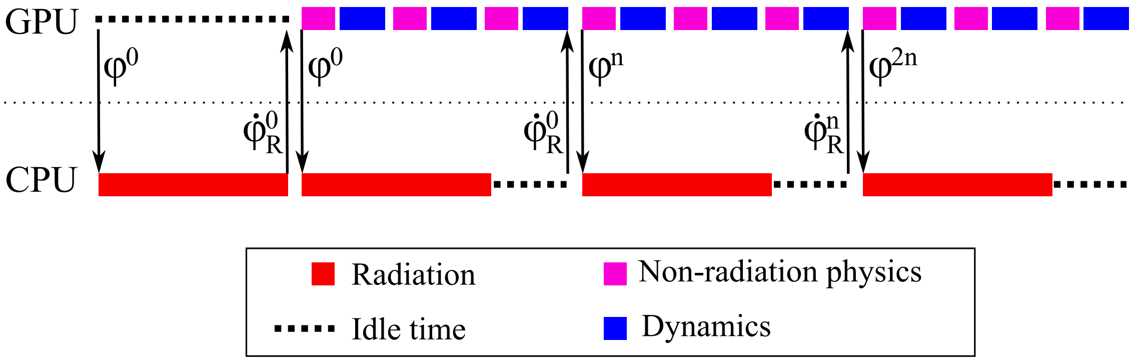
- **Quasi-uniform 60-km resolution** (163,842 cells)
- Δt=180 sec
- 41 vertical layers

From KISTI, Kim, Kang, & Joh GPU Acceleration of MPAS Physics Schemes Using OpenACC

Specific dynamics/physics schemes were prioritized for GPU while some set for CPU.

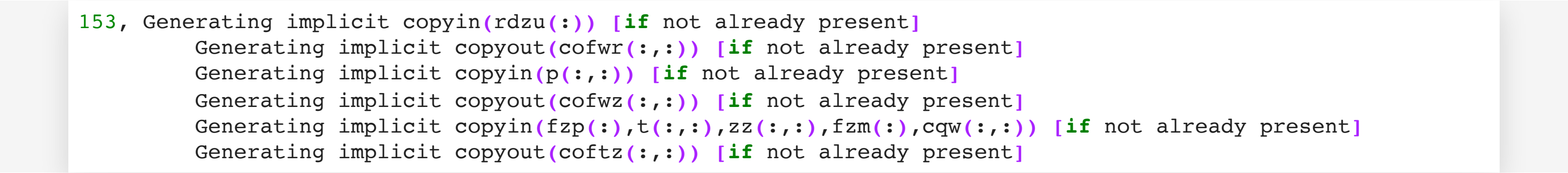


A lagged computation of radiation was established in order to utilize idle CPUs. Requires **manual tuning of load balancing** between number of CPU and GPU tasks.



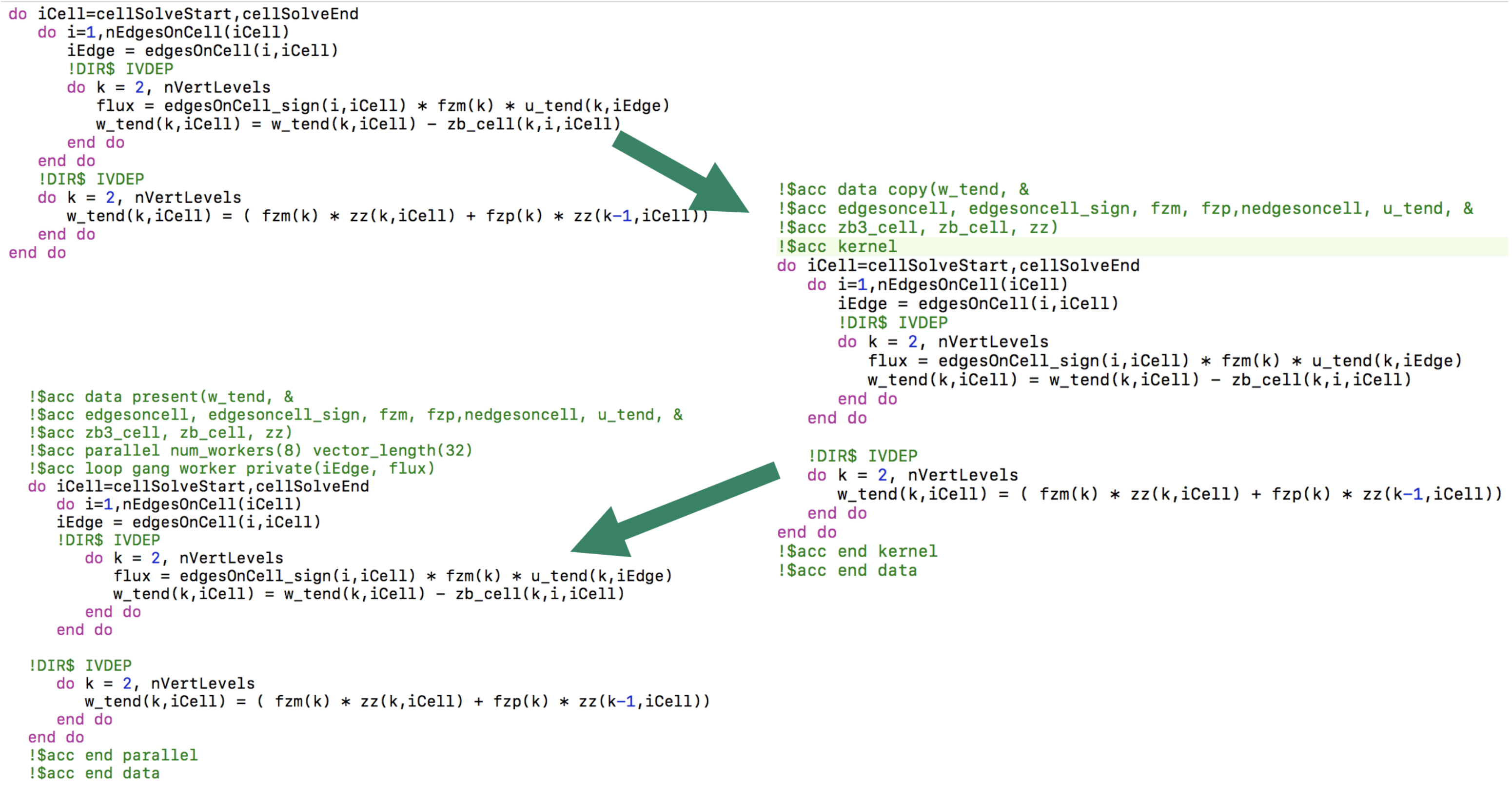
Managing GPU Data in MPAS-A

Recall that using `!$acc kernels ...` and similar directives will generate lists of variables needed to manage data movement for each compute region.



```
153, Generating implicit copyin(rdzu(:)) [if not already present]
          Generating implicit copyout(cofwr(:,:)) [if not already present]
          Generating implicit copyin(p(:,:)) [if not already present]
          Generating implicit copyout(cofwz(:,:)) [if not already present]
          Generating implicit copyin(fzp(:),t(:,:),zz(:,:),fzm(:),cqw(:,:)) [if not already present]
          Generating implicit copyout(coftz(:,:)) [if not already present]
```

These lists can be used and leveraged for your own data directives as GPU development progresses.



```
do iCell=cellSolveStart,cellSolveEnd
   do i=1,nEdgesOnCell(iCell)
      iEdge = edgesOnCell(i,iCell)
      !DIR$ IVDEP
      do k = 2, nVertLevels
         flux = edgesOnCell_sign(i,iCell) * fzm(k) * u_tend(k,iEdge)
         w_tend(k,iCell) = w_tend(k,iCell) - zb_cell(k,i,iCell)
      end do
   end do
   !DIR$ IVDEP
   do k = 2, nVertLevels
      w_tend(k,iCell) = ( fzm(k) * zz(k,iCell) + fzp(k) * zz(k-1,iCell))
   end do
end do
```

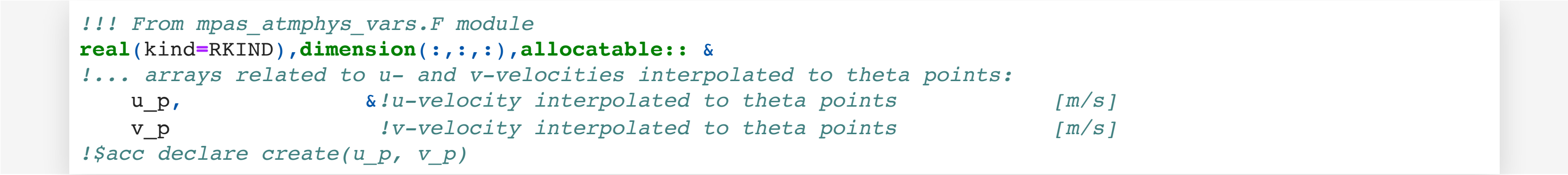
```
!$acc data copy(w_tend, &
!$acc edgesoncell, edgesoncell_sign, fzm, fzp,nedgesoncell, u_tend, &
!$acc zb3_cell, zb_cell, zz)
!$acc kernel
do iCell=cellSolveStart,cellSolveEnd
   do i=1,nEdgesOnCell(iCell)
      iEdge = edgesOnCell(i,iCell)
      !DIR$ IVDEP
      do k = 2, nVertLevels
         flux = edgesOnCell_sign(i,iCell) * fzm(k) * u_tend(k,iEdge)
         w_tend(k,iCell) = w_tend(k,iCell) - zb_cell(k,i,iCell)
      end do
   end do

   !DIR$ IVDEP
   do k = 2, nVertLevels
      w_tend(k,iCell) = ( fzm(k) * zz(k,iCell) + fzp(k) * zz(k-1,iCell))
   end do
end do
!$acc end kernel
!$acc end data
```

```
!$acc data present(w_tend, &
!$acc edgesoncell, edgesoncell_sign, fzm, fzp,nedgesoncell, u_tend, &
!$acc zb3_cell, zb_cell, zz)
!$acc parallel num_workers(8) vector_length(32)
!$acc loop gang worker private(iEdge, flux)
do iCell=cellSolveStart,cellSolveEnd
   do i=1,nEdgesOnCell(iCell)
   iEdge = edgesOnCell(i,iCell)
   !DIR$ IVDEP
      do k = 2, nVertLevels
         flux = edgesOnCell_sign(i,iCell) * fzm(k) * u_tend(k,iEdge)
         w_tend(k,iCell) = w_tend(k,iCell) - zb_cell(k,i,iCell)
      end do
   end do

!DIR$ IVDEP
   do k = 2, nVertLevels
      w_tend(k,iCell) = ( fzm(k) * zz(k,iCell) + fzp(k) * zz(k-1,iCell))
   end do
end do
!$acc end parallel
!$acc end data
```

Given ported kernels, MPAS-A was designed to create CPU and GPU data copies at initialization via `!$acc declare create(...)` and copy data at unstructured data regions via `!$acc enter data copyin(...)` prior to each kernel call. Then, each kernel would only require a `present(...)` clause using the prior variable lists. **Reference counters** would mitigate excessive copies.



```fortran
!!! From mpas_atmphys_vars.F module
real(kind=RKIND),dimension(:,:,:),allocatable:: &
!... arrays related to u- and v-velocities interpolated to theta points:
    u_p,                &!u-velocity interpolated to theta points            [m/s]
    v_p                  !v-velocity interpolated to theta points            [m/s]
!$acc declare create(u_p, v_p)
```

Any lingering excessive data copies could be identified by profilers and fixed while other required copies for CPU algorithms & I/O were managed by `!$acc update` directives.

MPAS-A Kernel Extraction

We will focus on the `atm_compute_vert_imp_coefs_work` subroutine and kernels as extracted by Supreeth Suresh, TDD/ASAP in CISL. This is the link, Line 2641 to the source subroutine in the full model codebase and in this workshop directory is the the extracted set of kernels mpas_atm_compute_vert_imp_coefs_work.F90.

Assuming data locality is resolved, this extracted kernel simply utilizes **randomized input data** as we will be **focusing on optimizing the performance** of the subroutine's kernels. The kernel is run in a repeating loop so we can get a relatively consistent average of measured performance. A validation tool has not been included at this time but is typically highly recommended.

For large codebases, building and/or using an automated tool like NCAR's KGen for Fortran codes or Kernel Tuner from NL eScience Center for CUDA/OpenCL codes will likely speed up the development/optimization process.

EXERCISE: MPAS-A Kernel Optimization

Open the mpas_atm_compute_vert_imp_coefs_work.F90 source file and convert the `!$acc kernels` loops to optimized `!$acc parallel ...` compute constructs. Analyze each set of loops and apply appropriate sets of kernel configuraton clauses to achieve improved performance. Note: `!DIR$ IVDEP` tells compiler to ignore loop dependencies for serial vector SIMD compilations.

You are encouraged to reference the initial attempts at optimization done by the `!$acc kernels` directive output during the compilation process. Data management has already been done for you using `-gpu=managed` and `present(var-list)` / `create(var-list)` clauses.

Record results of your optimization experiments on a chosen kernel and try to determine optimal configurations for that kernel. Compare your achieved performance with the original at Line 2641. Work on other kernels as time allows. Note that most kernels may benefit from similar clause specifications since they operate on similar domain sizes/variables.

In [ ]:

```
module load nvhpc/22.2 &> /dev/null
export _OPENACC=true
make
```

In [ ]:

```
module load nvhpc/22.2 &> /dev/null
export _OPENACC=true
make
```

In [ ]:

```
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=20 -v NVCOMPILER_ACC_TIME=1 -- \
`pwd`/vert_implicit_coefs.exe
```

In [ ]:

```
module load nvhpc/22.2 &> /dev/null
export _OPENACC=true
make
```

In [ ]:

```
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=20 -v NVCOMPILER_ACC_TIME=1 -- \
`pwd`/vert_implicit_coefs.exe
```

MPAS-A Kernels L###	**Device Time (μs)**
BaseLine (on V100) - `!$acc kernels`	XX
clause - `gang/vector`	XX
clause - `tile(##,##)`	XX
clause - `tile(*,*)`	XX
clause - vector_length(XX)	XX
clause - num_workers(XX)	XX
...	XX

Final Points

1. Plan for and **commit to a defined iterative GPU development process** to remove pain points and manage long term goals of your code project
    - **Smaller, validated incremental changes** are easier to debug
2. **Start with descriptive** `!$acc kernels` **then add prescriptive** `!$acc parallel ...` kernels as needed for expensive kernels
    - `!$acc kernels` can still achieve meaningful performance alone
3. Understand that the GPU development process takes time and effort but **specific tools/techniques can drastically speed up development time**.

Suggested Resources

- 2021 MPAS Virtual tutorial
- Computers & Geosciences, GPU acceleration of MPAS microphysics WSM6 using OpenACC directives: Performance and verification by J. Kim, J. Kang, and M. Joh (KISTI)
- OpenACC.org and NVIDIA managed GitHub, presentations, and learning materials GPU Bootcamps
    - Lab sequence on OpenACC
    - Lab sequence on Profiling Tools with MiniWeather
    - Lab sequence on Various GPU Programming Paradigms (CUDA, OpenACC, stdPar, OpenMP)
    - Lab sequence on Multi-GPU Programming
    - Lab sequences on GPU AI with CFD, PINNs, and Climate models

After this session, we will have three weeks until the next workshop. Order of upcoming sessions will also be adjusted to accomodate availability of a NVIDIA engineer to present on Multi-GPU programming. Look out for upcoming announcements.