Directive Based Programming with OpenACC and MiniWeather, Part 2

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

Date: April 14th 2022

In this notebook we explore the mini-app MiniWeather to present techniques and code examples using MiniWeather for using OpenACC to develop for GPUs. Extending from Part 1, we will cover:

1. Detailing of OpenACC API Directives
    - Data Constructs - `!$acc data` & `!$acc end data` plus `!$acc enter/exit data`
    - Routine Directives and Other Clauses - `!$acc routine gang/worker/vector/seq`
    - Async and Wait Directives - `!$acc async()` & `!$acc wait()`
    - OpenACC API Runtime Library Routines
2. Interfacing OpenACC with CUDA
3. Advanced OpenACC Optimization Techniques

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 `05_DirectivesOpenACC/05p2_openACC_miniWeather_Tutorial.ipynb`. 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
```

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
```

Test that MiniWeather builds correctly below. This build uses the already refactored and complete miniWeather_mpi_openacc.F90 source file as well as the CPU only miniWeather_mpi.F90 source file which serves as a basis for later exercises.

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
```

Test that MiniWeather builds correctly below. This build uses the already refactored and complete miniWeather_mpi_openacc.F90 source file as well as the CPU only miniWeather_mpi.F90 source file which serves as a basis for later exercises.

In [ ]:

```
cd fortran/build
source cmake_casper_nvhpc.sh
cd ../..
# After running this, there will be the executables `mpi` and `openacc` in "fortran/build"
```

Lastly, `make` and run the new miniWeather_mpi_exercise2.F90 program to establish a performance baseline.

Lastly, `make` and run the new miniWeather_mpi_exercise2.F90 program to establish a performance baseline.

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

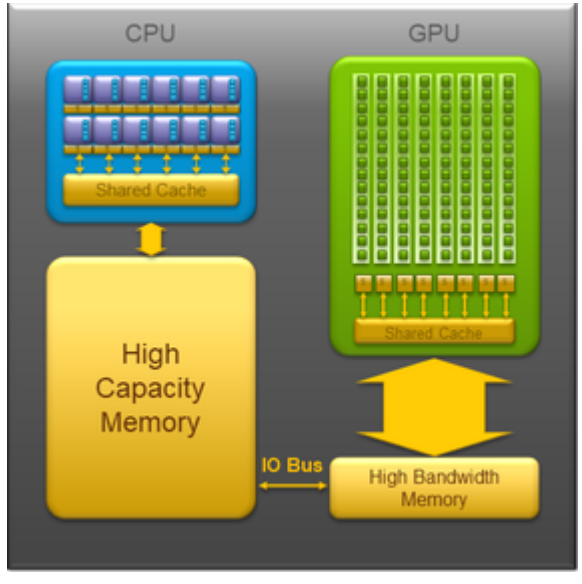
In [ ]:

```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2 1e-13 4.5e-5
cd ../..
```

Data Locaility and Data Movement Bandwidth

Previously, our work with MiniWeather utilized the `-gpu=managed` flag which setup a unified memory environment allowing us to not have to worry about data movement. However, unless the code has been designed to keep data resident on the GPU, **managed memory typically will not allow for optimal performance**.

Recalling the design of heterogeneous GPU accelerated systems, the following diagram shows limits of data movement by the size of the arrows between each memory space.



- CPU to GPU I/O PCIe Bus: **15.75 GB/s** for PCIe Gen3 V100s (31.5 GB/s for PCIe Gen4 capable A100s)
- CPU to High Capacity DDR4-2666 Memory (Casper): **127.8 GB/s** per CPU socket across 6 memory channels, 21.3 GB/s each
- GPU to GPU via NVLink: **300 GB/s** for V100s (600 GB/s for A100s)
- GPU HBM2 (High Bandwidth Memory 2): **900 GB/s** for V100s (1,555 GB/s for the 40GB A100s)

CPU to GPU Data Movement is Costly!

- The bandwidth transfer rate for CPU to GPU data movement is the slowest in present day heterogeneous systems
- Data transfers between the CPU and GPU should be minimized and ideally avoided

Using OpenACC's directives `!$acc data`, `!$acc enter/exit data`, and `!$acc update ...` allow you to manage directly residency of data across the distinct CPU and GPU memory spaces.

Data Directives

Reviewing `-Minfo=accel` output from last session, many data clauses were implicitly specified by the compiler despite using `-gpu=managed`. Data clauses are below:

OpenACC	OpenMP	Description
Data Clauses		Specifies data movement between CPU & GPU in parallel/data regions
`create([zero]:vars)`	`alloc(vars)`	Allocates memory on target device for data object, optionally initializes to **zero** values
`copy(vars)`	`map(tofrom:vars)`	Allocates memory if needed and copies data at region entry/exit
`copyin(vars)`	`map(to:vars)`	Allocates memory if needed and copies data at region entry
`copyout([zero]:vars)`	`map(from:vars)`	Allocates memory if needed and copies data object at region exit, optionally initializes to **zero** values
`present(vars)`	`assert(omp_target_is_present(vars))`	Indicates that a data object is already present on GPU. Previous clauses include an implicit `present()` such that if true, that clause's action will not be performed
`delete(vars)`	`dealloc(vars)`	Deallocates memory on target device for data object
`deviceptr(vars)`	`is_device_ptr(vars)`	Declares device pointers such that data does not need to be moved/allocated
`attach(vars)`	N/A	Increments the attachment counter for a pointer
`detach(vars)`	N/A	Decrements the attachment counter for a pointer
`finalize`	N/A	Sets structured or dynamic reference counter to 0 and forces action of `copyout()`, `detach()`, or `delete()`

Data Directives

Reviewing `-Minfo=accel` output from last session, many data clauses were implicitly specified by the compiler despite using `-gpu=managed`. Data clauses are below:

OpenACC	OpenMP	Description
Data Clauses		Specifies data movement between CPU & GPU in parallel/data regions
`create([zero]:vars)`	`alloc(vars)`	Allocates memory on target device for data object, optionally initializes to **zero** values
`copy(vars)`	`map(tofrom:vars)`	Allocates memory if needed and copies data at region entry/exit
`copyin(vars)`	`map(to:vars)`	Allocates memory if needed and copies data at region entry
`copyout([zero]:vars)`	`map(from:vars)`	Allocates memory if needed and copies data object at region exit, optionally initializes to **zero** values
`present(vars)`	`assert(omp_target_is_present(vars))`	Indicates that a data object is already present on GPU. Previous clauses include an implicit `present()` such that if true, that clause's action will not be performed
`delete(vars)`	`dealloc(vars)`	Deallocates memory on target device for data object
`deviceptr(vars)`	`is_device_ptr(vars)`	Declares device pointers such that data does not need to be moved/allocated
`attach(vars)`	N/A	Increments the attachment counter for a pointer
`detach(vars)`	N/A	Decrements the attachment counter for a pointer
`finalize`	N/A	Sets structured or dynamic reference counter to 0 and forces action of `copyout()`, `detach()`, or `delete()`

Each clause can reference a list of variables, ie `vars` = `var1,var2,...` or subsets of variable arrays. For example, `copy(a,b(10:20))` copies from CPU to GPU at region entry and from GPU to CPU at region exit both the variable `a` (full array by default) and only the slice of `b` indexed from 10 through 20.

Scope of Data Directives and Data Regions

All data clauses can only be used in the scope of a ...

- **Structured Data Region** or localized regions of code
    - **Compute constructs regions** like `!$acc kernels/parallel/serial ...` & `!$acc end kernels/parallel/serial`
    - **Data constructs regions** like `!$acc data ...` & `!$acc end data`
    - **Implicit data regions** of a function, subroutine, or program dependent on where `!$acc declare ...` directive is used
    - Uses structured reference counters
- **Unstructured Data Region** or global program execution
    - In a **data directive** like `!$acc enter data ...` or `!$acc exit data ...`
    - Uses dynamic reference counters

Unstructured Data Regions are preferred particularly when dealing with object oriented codes where the programmer may need to manage data locality across multiple modules, subroutines/functions, and source files.

Specific clauses can only be used in certain contexts. For example:

- `present()` cannot be used with `!$acc enter/exit data ...`
- `delete()` can only be used with `!$acc exit data ...`
- See specifications for each region construct in OpenACC 2.7 Quick Reference Guide for more details

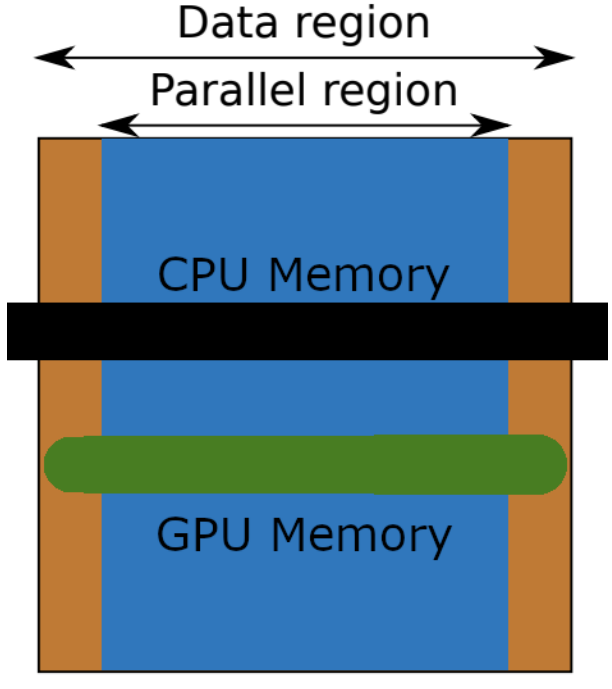
Note: We will not cover `deviceptr()`, `attach()`, and `dettach()` in this notebook. These clauses are typically useful for CUDA libraries interoperaility and CUDA aware MPI with OpenACC.

Visualizing Data Directives with Data and Compute Regions

This code and following visualizations highlight the data locality of variables across **CPU memory space, the black line,** and the **GPU memory space, the green line**. To note, there is **always an implied data region with any compute construct**.

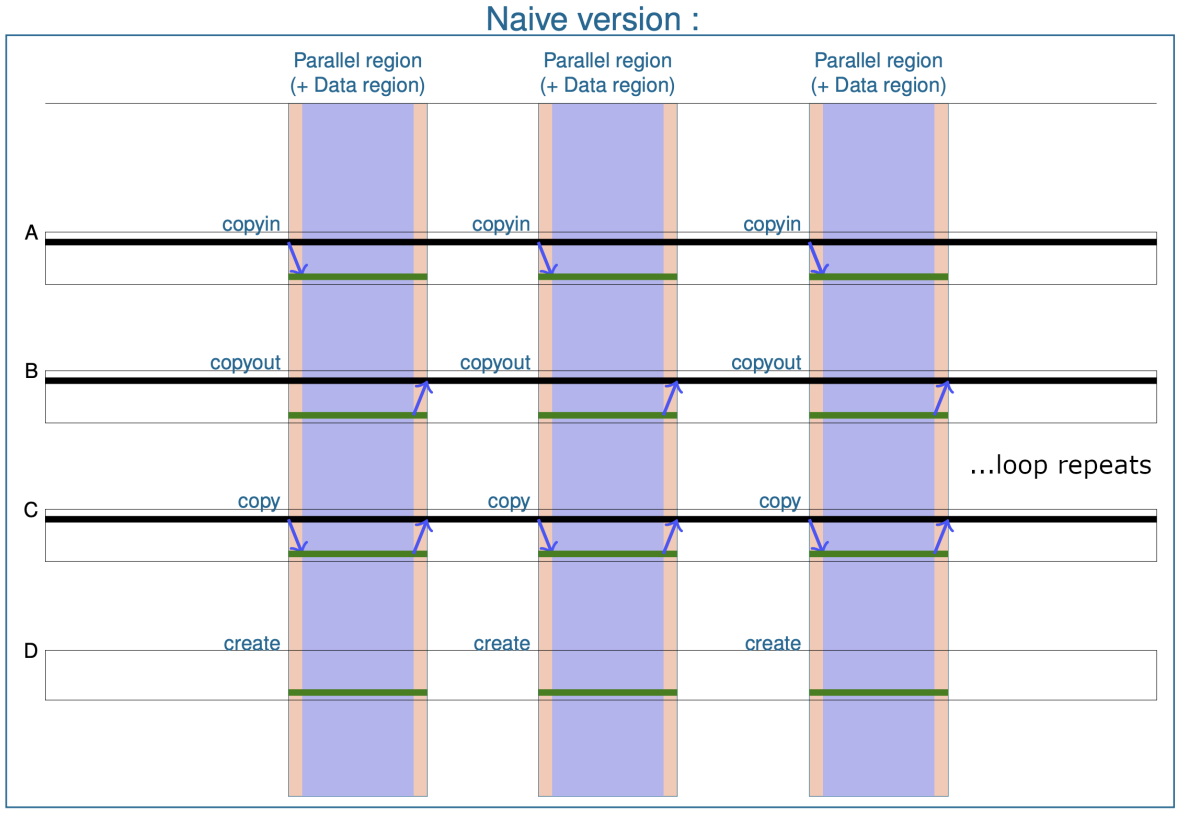
```
!$acc parallel
{!!! data region !!!
    {!** parallel region **!
    !$acc loop
    do i = 1,1000
        flux(i) = ...
    end do
    }!** end parallel region **!
}!!! end data region !!!
!$acc end parallel
```

Implicit Data Region with a Parallel Region



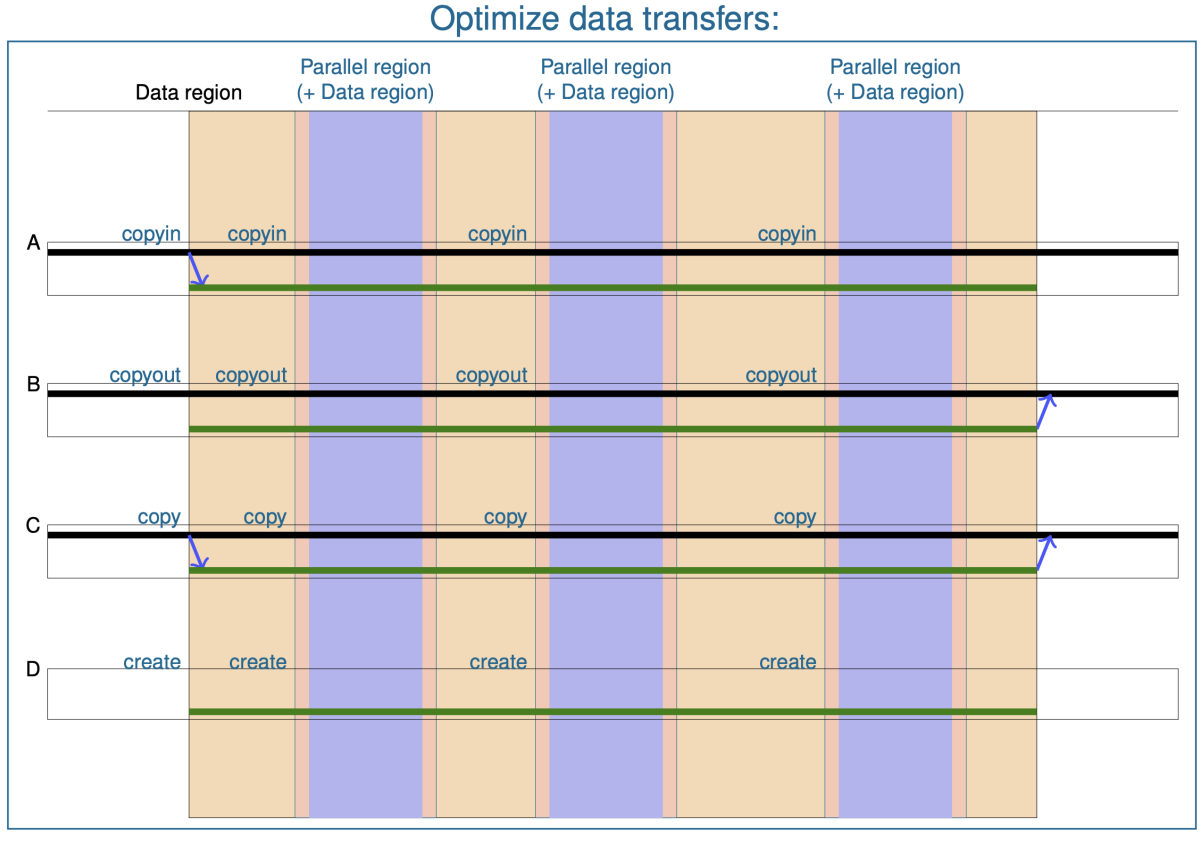
Thanks go to Pierre-François Lavallée and Thibaut Véry from IDRIS, France for this and following images inspiraton. See their PRACE course Introduction to OpenACC and OpenMP GPU presented July 2019.

OpenACC Program without Data Managment



Without specifiying data movement, data transfers/allocations are performed by default at each compute region as determined by provided clauses or compiler (see `-Minfo=accel` output). For a program with many time steps, this could be 1,000s of unecessary transfers!

OpenACC Program with Structured Data Region



By specifying data movement over an encapsulating data region, data actions are done only at region entry and/or exit.

EXERCISE: Specify Variables in Data Regions around Time Step Loop

Let's return to MiniWeather. We will now be using a new exercise source file miniWeather_mpi_exercise2.F90. This file has the `!$acc parallel ...` directives completed from the previous session. Now, we want to add a data region around the main time step loop section of the code near **Line 128** (see `TODO: MANAGE GPU DATA`). This will look like:

```
!$acc data ... ! Structured data region
    !$acc kernels
        ...
    !$acc end kernels
    call f_has_kernels()
!$acc end data

!!!!! OR !!!!!

!$acc enter data ... ! Unstructured data directive
    !$acc kernels
        ...
    !$acc end kernels
    call f_has_kernels()
!$acc exit data ... ! Unstructured data directive
```

Only consider usage of `copy()`, `copyin()`, `copyout()`, and `create()`. Hints are provided (`CTRL` / `CMD` + `F` `GPU_data`) to help define which clauses to use and variables you need to consider. Note: Variables listed in `GPU_data` sections are used across the `reductions()`, `output()`, and `perform_timestep()` subroutines called within the main time step loop.

1. **Which clause do you use to ensure that data is availabile on the GPU and updates the same data on CPU after GPU work is done?**
2. **Which clauses are optimal towards minimizing data movement in MiniWeather?**
3. **If you used an unstructured data region, did this prevent data movement for the structured compute kernels? Why or why not?**

Once you are satisfied with your changes, `make` and run the new executable and see how that has impacted performance.

1. **Which clause do you use to ensure that data is availabile on the GPU and updates the same data on CPU after GPU work is done?**
2. **Which clauses are optimal towards minimizing data movement in MiniWeather?**
3. **If you used an unstructured data region, did this prevent data movement for the structured compute kernels? Why or why not?**

Once you are satisfied with your changes, `make` and run the new executable and see how that has impacted performance.

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

1. **Which clause do you use to ensure that data is availabile on the GPU and updates the same data on CPU after GPU work is done?**
2. **Which clauses are optimal towards minimizing data movement in MiniWeather?**
3. **If you used an unstructured data region, did this prevent data movement for the structured compute kernels? Why or why not?**

Once you are satisfied with your changes, `make` and run the new executable and see how that has impacted performance.

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

In [ ]:

```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2 1e-13 4.5e-5
cd ../..
```

Data Directives - `!$acc update ...` and the `present()` Clause

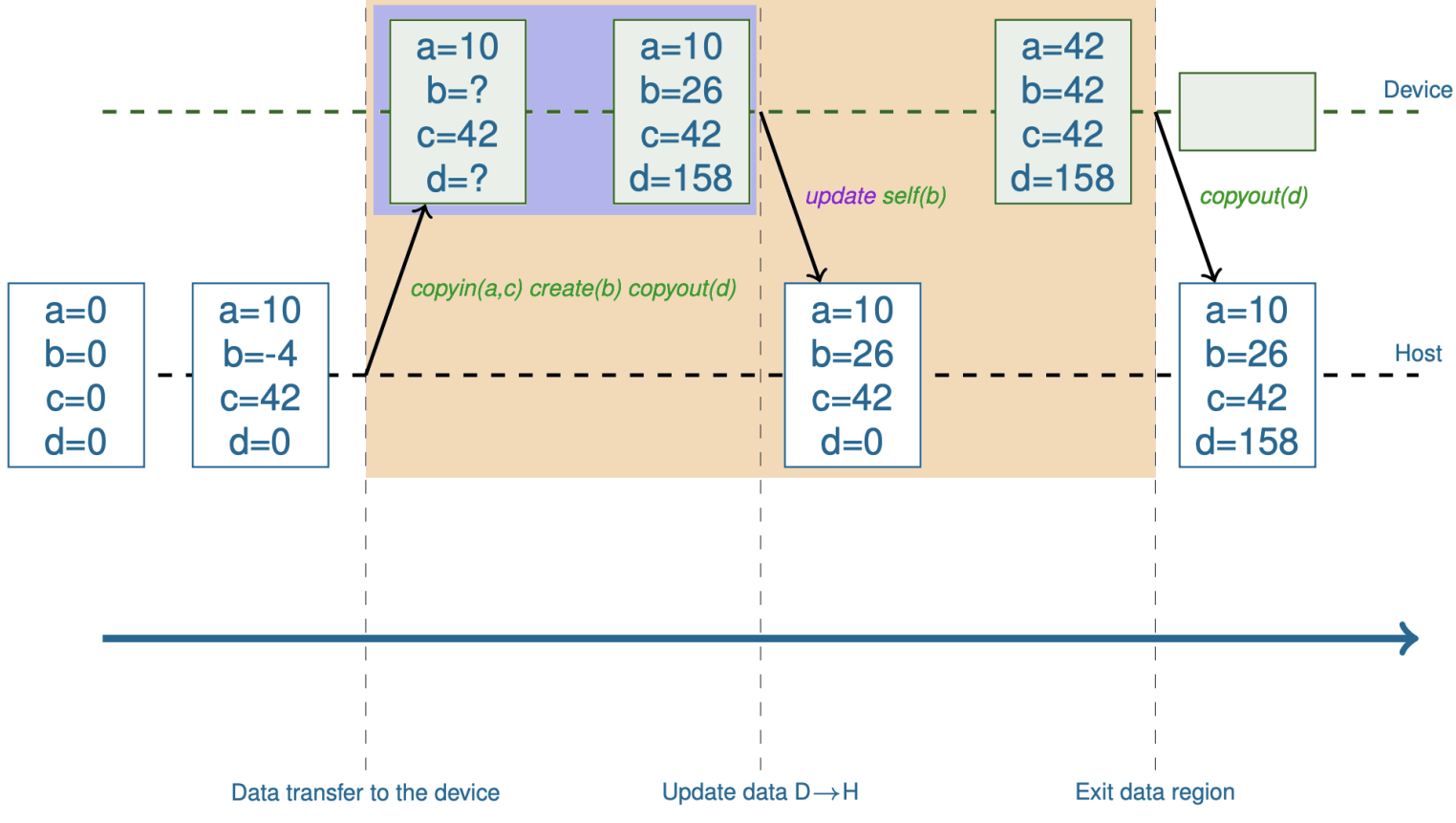
Unfortunately, the previous exercise introduced incorrect results. In some cases, you need to guarantee the movement of data between the host and device in order to ensure correct data is being processed. This uses the `!$acc update ...` directive:

- `!$acc update host(...)` or `!$acc update self(...)` - Copies listed data variables from the GPU device to the host CPU.
- `!$acc update device(...)` - Copies listed data variables from the host CPU to the device GPU.

The `update` directive is useful if you are processing data on the host side after the data has been modified on the device side or vice versa. Note that `!$acc update ...` cannot be added within a compute construct.

The `present()` clause will prevent an implicit data directive from being used on a parallel region. It makes clear to the compiler the availability of data on the device and only checks that data is present on the device. It also can make the code more readable to the programmer. Whether or not the `present()` data is in sync with the data on the CPU depends on recent computations or usage of `!$acc update ...`.

Example Diagram for Data Movement Clauses



Reference Counters and Attachment Counters

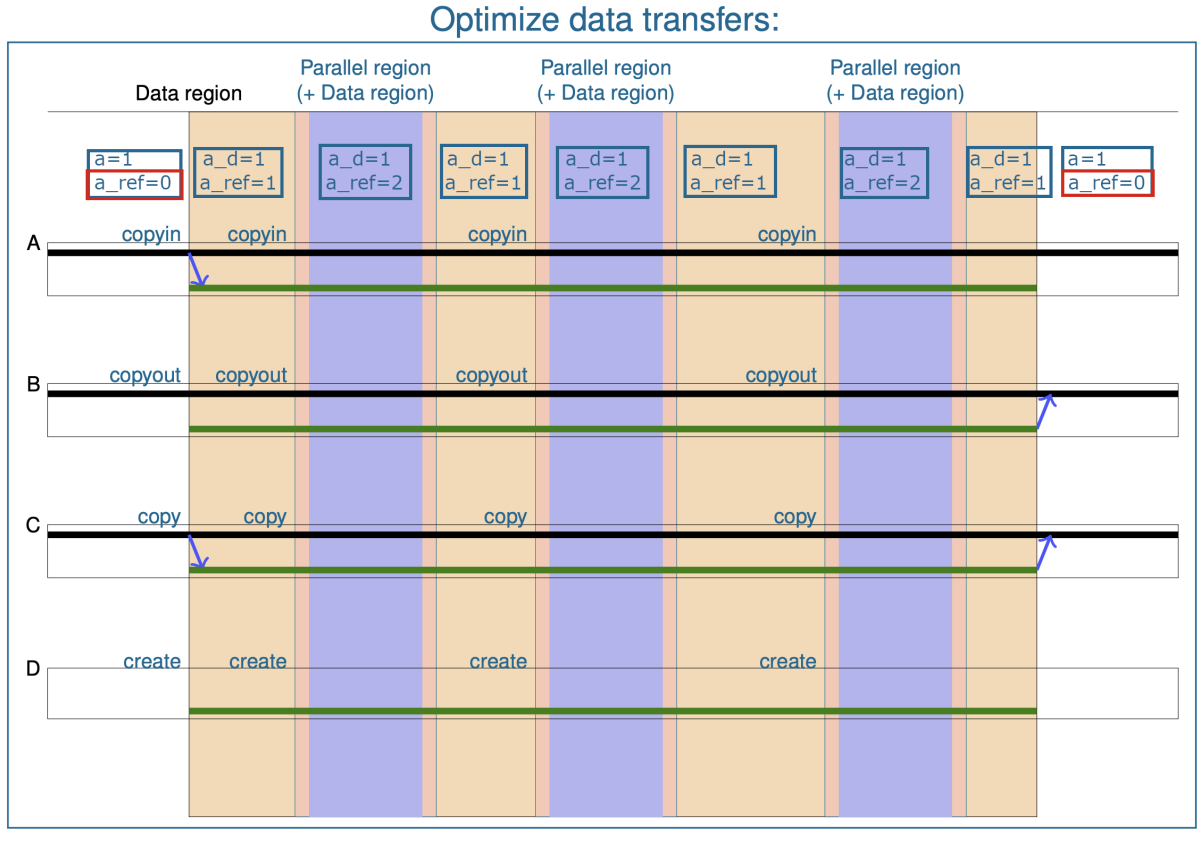
All data clauses also utilize a **reference counter** for device variables or **attachment counter** for device pointers. There are separate counters for **structured** and **unstructured** data regions. Only when a variable's counter increments from 0 or is reduced to 0 does a data action get performed. Example:

1. Enter first data region with `copy(a,b)` clause. `a` & `b` copied from CPU to GPU, **reference counters** increment 0 -> 1
2. Kernel inside data region with `copy(a,b)` clause. No copy action but **reference counters** increment 1 -> 2
3. Kernel exits and no copy is performed. **Reference counters** decrement from 2->1
4. First data region exits, `a` & `b` copied out and deallocated on GPU, **reference counters** decrement 1 -> 0.

Essentially, counters track whether data is already *present* or *not present* on the GPU, ie *present* = *true* if *counter* > 0 and informs data movement actions. This can avoid unnecessarily moving data at each kernel. To ensure a data action is performed, the use of `!$acc update ...` is recommended.

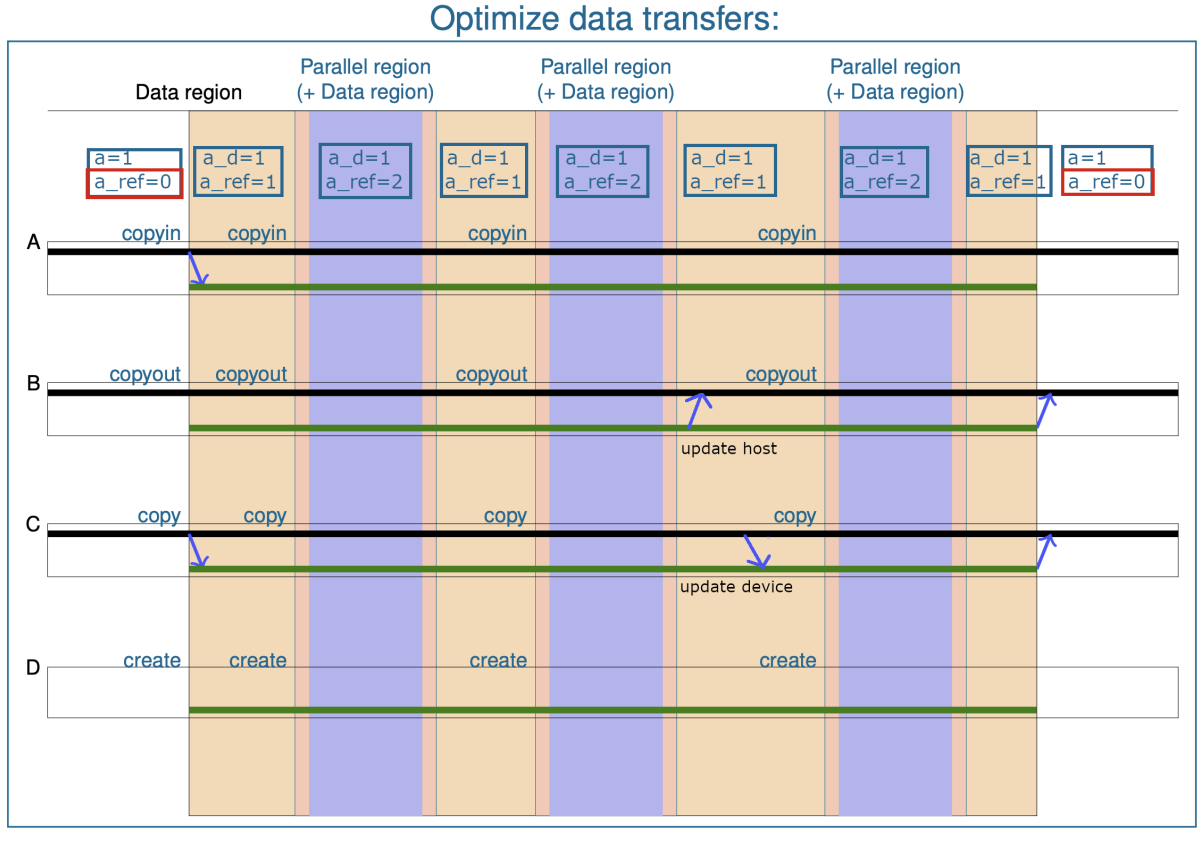
Another option is adding `finalize` for unstructured data regions only, ie `!$acc exit data ...` with `delete()`, `detach()`, or `copyout()`. This forces the **reference counter** to 0.

OpenACC and Reference Counters



Even when compute regions specify data movement clauses, data actions only happen when the counter increments from 0 -> 1 or decrements 1 -> 0. In this example, the variable `a` only has a data action performed on it at at data region entry and exit.

OpenACC and Update Directives



To ensure a data operation takes place, use `!$acc update host()` or `!$acc update device()` outside a parallel compute region. Another option: add `finalize` to clauses `copyout()`, `delete()`, or `detach()` in any structured or unstructured data region. `finalize` will set the reference counter to 0.

EXERCISE: Specify Update Directives for MPI Processes

Again using the exercise source file miniWeather_mpi_exercise2.F90, we want to add a update directives at the following locations to manage the send and receive buffers as well as manage I/O output appropriately. Search for `TODO: UPDATE DATA` or go to:

- MPI Buffers
    - [] Line 448
    - [] Line 460
- Output I/O
    - [] Line 828 - hints provided

1. Look at the buffer variables that need to be passed between MPI tasks. **When do the data objects need to be updated on the host and device when doing an MPI send and MPI receive?**

Once you are satisfied with your changes, `make` and run the new executable and review how that has impacted performance.

Once you are satisfied with your changes, `make` and run the new executable and review how that has impacted performance.

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

Once you are satisfied with your changes, `make` and run the new executable and review how that has impacted performance.

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

In [ ]:

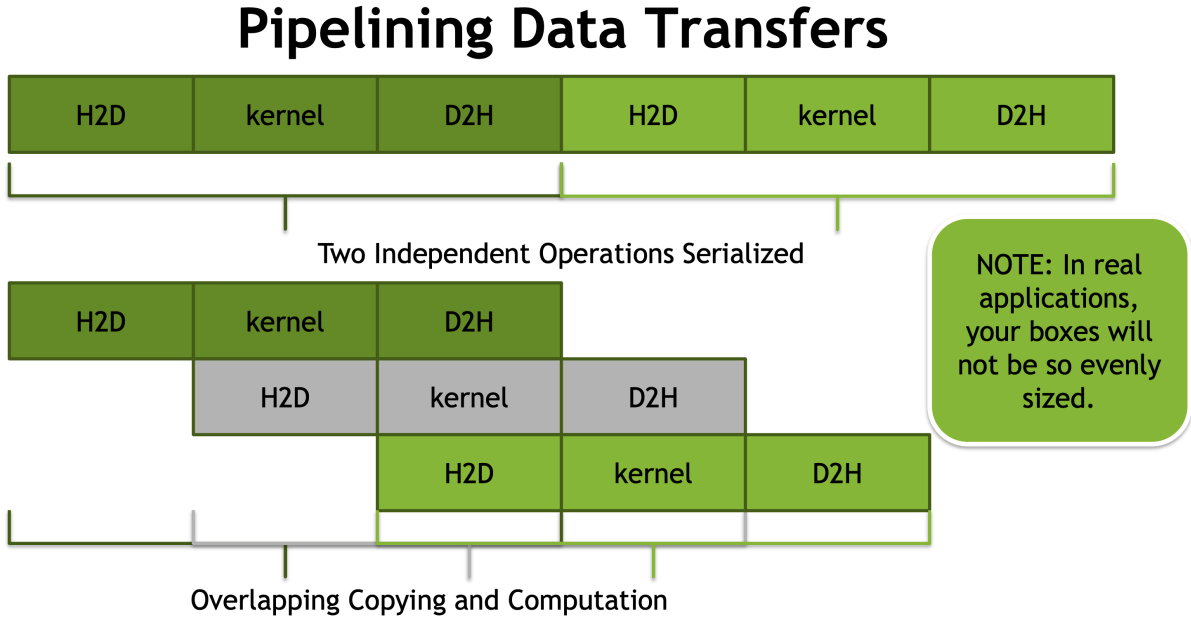
```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2_update 1e-13 4.5e-5
cd ../..
```

Asynchronous Execution with `async()` and `wait()` Clauses

One of the more important advanced OpenACC features is **asynchronous execution**. This allows you to interface with **CUDA Streams** and manage directly the scheduling queues the GPU uses to run kernels on the device in a defined sequence.

- **Synchronous Execution** - Enabled by default for all directives
    - Host must wait for parallel compute regions and data regions/updates to complete
    - All operations and parallel task units must run in a serialized sequence
- **Asynchronous Execution** - Must be specified for every compatible directive
    - Host can perform work while the device GPU is also performing work
    - Data transfers can be scheduled and start processing before the data is needed while other work is performed
    - Multiple parallel compute units can be scheduled on device to run in an uninterrupted pipeline. Can avoid kernel startup/spindown scheduling costs before being forced to wait/synchronize

So far, the CPU and GPU have performed **synchronous execution**. The CPU must wait to synchronize with the completion of compute or data movement work on the GPU. This often creates time gaps between compute kernels on the GPU (say during data movement or kernel initialization) where the hardware is not being utilized.



If multiple asynchronous queues are used and there are available compute resources, it's also possible to overlap compute kernels with each other in addition to data movement kernels. **Serialized execution of parallel kernels is not required.**

Usage of `async()` and `wait()` Clauses

The `async` clause may appear on a `parallel`, `serial`, `kernels`, or `data` construct, or an `enter data`, `exit data`, `update`, or `wait` directive. The `wait` clause can appear on the same (except itself).

- `async(n)` - Launches work asynchronously in queue `n`. If `n` is omitted, the default queue is selected.
- `wait(n,m,...)` - Blocks host (or blocks queue `m` if paired with `async(m)`) until all prior operations in queues `n,m,...` have completed. If an argument is omitted, all queues must complete.

The optional argument `n` must be an integer value. An essentially limitless number of queues/streams are able to be created. However, only 48 CUDA stream contexts (16 pre-Volta) will be allowed to run concurrently.

Complex compute kernel task dependencies can be designed using clever queue scheduling to fully saturate the GPU. However, limited time does not permit us to review this here. Instead, please refer to section 7.1 of the OpenACC Best Practices Programming Guide, May 2021 or more briefly in Steve Abbott's Advanced OpenACC Lecture Slides given at University of Tennessee's Innovation Computing Laboratory.

EXERCISE: Specify `async` Compute Kernels and Place `wait` Barriers in MiniWeather

Again using the exercise source file miniWeather_mpi_exercise2.F90, we want to add `async` and `wait` at appropriate places in the code. Search for `TODO: ASYNC ME` or go to:

- [] Line 251
- [] Line 304
- [] Line 337
- [] Line 366
- [] Line 404
- [] Line 435
- [] Line 449
- [] Line 461
- [] Line 481
- [] Line 503

1. **Is it possible to set any of MiniWeather's compute kernels in thier own queues or do dependencies between kernels prevent this?**
2. How did **asynchronous execution** impact performance? **Review previous incremental changes to MiniWeather and record below the change in CPU time.** (Note: Select cell and press `M` or double click this cell to edit the table)
3. Enabling asynchronous execution only improved performance slightly. **What additional factors are likely contributing to this modest improvement?** This is best answered using a profiler via `nsys profile -t openacc ./openacc_test_ex2` but feel free to speculate.

MiniWeather Edits	CPU Time (s)
BaseLine (on V100)	26.1405
Data regions (d_mass wrong)	XX
Update directives (d_mass fixed)	XX
Async and wait	XX

1. **Is it possible to set any of MiniWeather's compute kernels in thier own queues or do dependencies between kernels prevent this?**
2. How did **asynchronous execution** impact performance? **Review previous incremental changes to MiniWeather and record below the change in CPU time.** (Note: Select cell and press `M` or double click this cell to edit the table)
3. Enabling asynchronous execution only improved performance slightly. **What additional factors are likely contributing to this modest improvement?** This is best answered using a profiler via `nsys profile -t openacc ./openacc_test_ex2` but feel free to speculate.

MiniWeather Edits	CPU Time (s)
BaseLine (on V100)	26.1405
Data regions (d_mass wrong)	XX
Update directives (d_mass fixed)	XX
Async and wait	XX

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

1. **Is it possible to set any of MiniWeather's compute kernels in thier own queues or do dependencies between kernels prevent this?**
2. How did **asynchronous execution** impact performance? **Review previous incremental changes to MiniWeather and record below the change in CPU time.** (Note: Select cell and press `M` or double click this cell to edit the table)
3. Enabling asynchronous execution only improved performance slightly. **What additional factors are likely contributing to this modest improvement?** This is best answered using a profiler via `nsys profile -t openacc ./openacc_test_ex2` but feel free to speculate.

MiniWeather Edits	CPU Time (s)
BaseLine (on V100)	26.1405
Data regions (d_mass wrong)	XX
Update directives (d_mass fixed)	XX
Async and wait	XX

In [ ]:

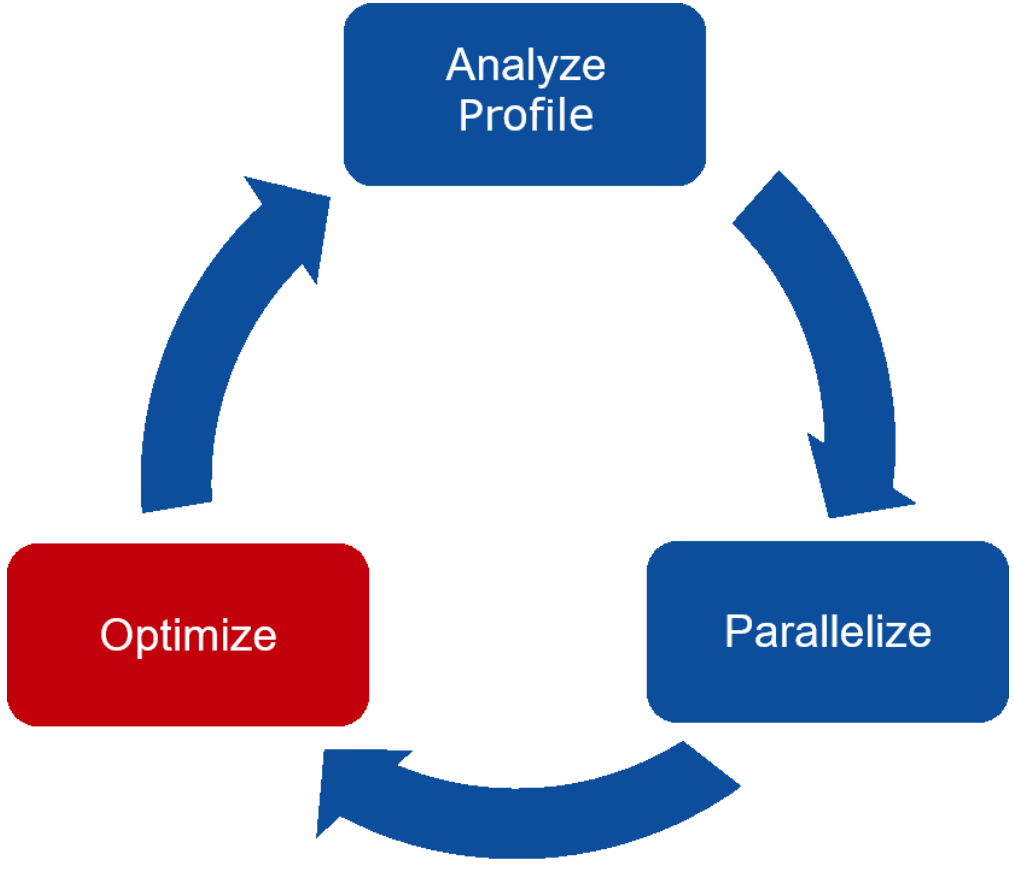
```
make -C fortran/build openacc_test_ex2
```

In [ ]:

```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2 1e-13 4.5e-5
cd ../..
```

Remaining Concepts Will Be Discussed with Time Remaining or Provided for Independent Review

Some great discussions on these advanced concepts are covered in John Urbanic's June 2021 GPU Programming Lectures Advanced OpenACC and Using OpenACC With CUDA Libraries. Nonetheless, most optimizations using OpenACC at this level should include profiling as part of any **development cycle**. We will cover usage of the Nsight Systems and Nsight Compute profilers at later sessions.



Routines and Functions Called from a GPU Kernel - `!$acc routine ...` Directive

Compute regions are able to call external functions from within the kernel. However, the external function needs to be declared to the compiler that it is a GPU kernel.

This is done using the `!$acc routine ...` directive. This directive should be placed in the specification part of the routine it should apply to or you may place it in the calling routine's/module's specification section as `!$acc routine(name)` where `name` is the routine which it applies. Required with any `!$acc routine ...` directive is one of four clauses:

- `seq` - Specifies the routine should run in `seq` mode and is not parallel, should be run sequentially
- `vector` - Specifies the routine should run in `vector` mode and contains a `!$acc loop vector` parallel construct or calls another vector routine
- `worker` - Specifies the routine should run in `worker` mode and contains a `!$acc loop worker` parallel construct or calls another worker routine
- `gang` - Specifies the routine should run in `gang` mode and contains a `!$acc loop gang` parallel construct or calls another gang routine

EXERCISE: Convert Inlined `sample_ellipse_cosine()` to Function Call

In miniWeather_mpi_exercise2.F90, you can try specifying a GPU routine by searching for `TODO: TRY GPU ROUTINE OPTION` or going to **Lines 264 and 769**. As a comment, compilers often have non-optimal and sometimes even broken support for `!$acc routine ...`. Thus, perhaps consider instead inlining required routines into each relevant parallel region.

`make` and run the new executable and evaluate if this changed performance in any way.

1. **Which parallel mode (`seq`, `vector`, `worker`, or `gang`) is required to be set for the function `sample_ellipse_cosine`?**
2. **Do you see any other algorithms/groups of operations, perhaps at a `vector` level or higher, you might write into its own subroutine/function? Try it out in MiniWeather!**

`make` and run the new executable and evaluate if this changed performance in any way.

1. **Which parallel mode (`seq`, `vector`, `worker`, or `gang`) is required to be set for the function `sample_ellipse_cosine`?**
2. **Do you see any other algorithms/groups of operations, perhaps at a `vector` level or higher, you might write into its own subroutine/function? Try it out in MiniWeather!**

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

`make` and run the new executable and evaluate if this changed performance in any way.

1. **Which parallel mode (`seq`, `vector`, `worker`, or `gang`) is required to be set for the function `sample_ellipse_cosine`?**
2. **Do you see any other algorithms/groups of operations, perhaps at a `vector` level or higher, you might write into its own subroutine/function? Try it out in MiniWeather!**

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

In [ ]:

```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2 1e-13 4.5e-5
cd ../..
```

OpenACC API Routines

OpenACC provides access to runtime library routines via `use openacc` module in FORTRAN (or `#include <openacc.h>` header in C/C++). This allows executing many features of OpenACC without directives, as well as a few important features not available through directives. Two important routines are below. These are most relevant for **Multi-GPU Programming** using **CUDA aware MPI** and will be discussed in future sessions.

1. `acc_get_num_devices(acc_device_t dev_type)` - Returns the integer value of the number of devices connected to the host.
    - Use `dev_type` = `acc_device_nvidia` to enumerate NVIDIA devices.
2. `acc_set_device_num(int idev, acc_device_t dev_type)` - Sets for the runtime which device to use, where 0 <= `idev` < `dev_num`.

See page 15 of the OpenACC API Guide, v2.7 or page 89 of OpenACC Full Specification, v3.2 for a complete description of all possible API routines.

OpenACC with CUDA and GPU Accelerated Libraries

There is functionality to use manually developed CUDA kernels or CUDA libraries alongside OpenACC. This is particularly useful if you have decided to develop optimal CUDA code for a particularly expensive kernel or need to interface with a highly performant CUDA Accelerated Library such as cuBLAS or cuFFT. Note however this will likely impact the portability of your code with non-NVIDIA devices.

This use case is effectively explained in John Urbanic's Using OpenACC With CUDA Libraries presentation. Nonetheless, below are the most important concepts to remember:

OpenACC cuTENSOR Example

Here is an example use case for the cuTENSOR library using OpenACC. The complete FORTRAN can be found in NVIDIA's HPC SDK Documentation Using cuTENSOR from OpenACC Host Code. Notably, this code is portable with and without OpenACC using `!@acc`.

```
!@acc use openacc
!@acc use cutensorex
integer, parameter :: ni=1280, nj=1024, nk=960, ntimes=1
real(8) :: a(ni,nk), b(nk,nj), c(ni,nj), d(ni,nj)

call random_number(a)
call random_number(b)
a = dble(int(4.0d0*a - 2.0d0))
b = dble(int(8.0d0*b - 4.0d0))
c = 2.0; d = 0.0

!$acc enter data copyin(a,b,c) create(d)
!@acc istat = cutensorExSetStream(acc_get_cuda_stream(acc_async_sync))
!$acc host_data use_device(a,b,c,d)
do nt = 1, ntimes
  d = c + matmul(a,b)
end do
!$acc end host_data
!$acc update host(d)
```

Advanced Optimizations with `cache`, `tile`, and `gang/worker/vector` Clauses

The compiler typically does a good job choosing close to optimal arrangements of `gang`, `worker`, and `vector` parallel execution. You can review compiler choices in `-Minfo=accel` output, basic profiling via `NVCOMPILER_ACC_TIME=1`, or NSight Systems/Compute profiling. However, further optimization can be explored by manualy specifying optimization parameters. **This process requires significant experimentation with some intuition**. Clauses not previously discussed are detailed below:

- `cache()` - When included inside and at the top of a loop, specifies array elements or subarrays that should be fetched into the highest level of cache, ie the shared memory of the SM assigned to each gang.
- `tile(n,m,...)` - Splits loops into two loops, outer *tile* loops and inner *element* loops. Listed integer arguments specify the size of each tile and correspond to each tightly nested loop. The first `n` corresponds to the innermost loop. If `n` = `*`, the compiler has freedom to choose tile size.
    - A `vector` clause is applied to *element* loops
    - A `gang` clause is applied to *tile* loops
    - A `worker` clause is applied to *element* loops or to *tile* loops if the `vector` clause is also present

Usage of `cache()`

If you have a an array or subarray of data to be accessed frequently within a compute kernel, the `cache()` directive can direct the compiler to keep that portion of memory nearby in a fast L1 shared memory cache for each gang. The size of this cache is limited and dependent on the hardware (V100 - 96kB, A100 - 160kB) but can increase performance if configured correctly. Example below:

```
!$acc parallel loop gang vector
do i = 2, n-1
  !$acc cache(a(i-1:i+1))
  lower = i-1
  upper = i+1
  sum = 0;
  !$acc loop seq
  do j = lower, upper
    sum = sum + a(i);
  end do
end do
!$acc end parallel
```

Feel free to review Lashgar's and Baniasadi's Efficient Implementation of OpenACC cache Directive on NVIDIA GPUs in 2017 IJHPCN for a more in depth discussion from a compiler perspective.

Target Specific Devices when Specifying Parallelism

When tuning performance using the following clauses, OpenACC should target specific hardware or hardware types. To promote portability, consider pairing these clauses with `device_type(d_type)` where `d_type` = `acc_device_nvidia`, `acc_device_radeon`, `acc_device_host`, etc depending on what OpenACC specification is implemented and what the compiler recognizes.

For example, `!$acc loop device_type(acc_device_nvidia) vector_length(1024) device_type(acc_device_radeon) vector_length(128)` will apply distinct clauses for NVIDIA and AMD devices respectively.

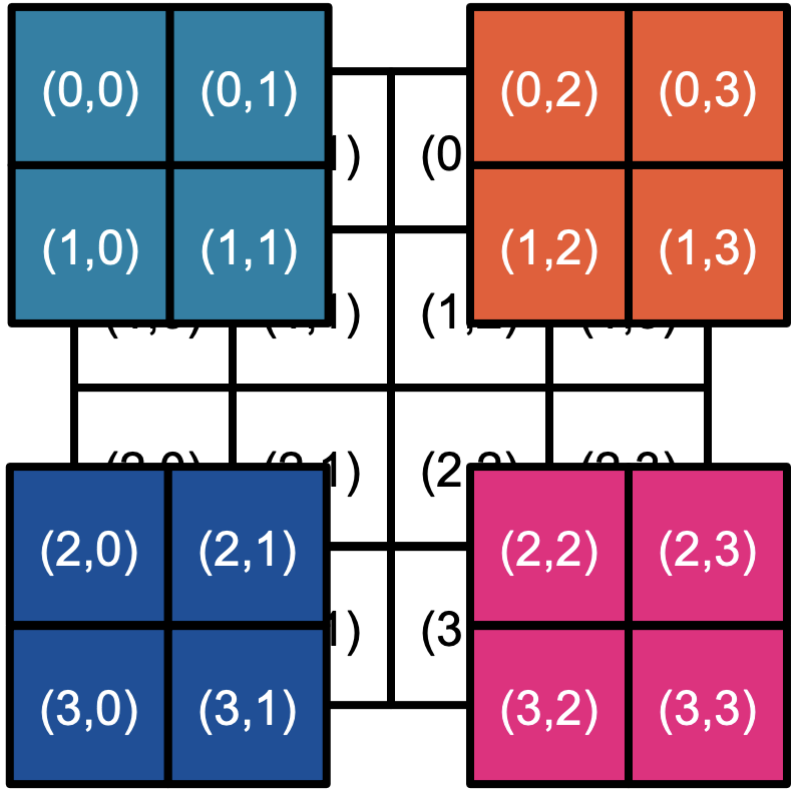
If neither device type is attached, then the compiler chooses its own arrangement of levels of parallelism.

Usage of `tile()`

For multidimensional loops, `tile()` splits loops into blocks to distribute work across a device. This is essentially opposite to `collapse()` which exposes parallelism and unrolls loops into the largest possible units.

Where `collapse()` can often negatively impact data locality, `tile()` should be used to increase data locality or encourage **memory coalescing**, improving performance in some cases. Lastly, tiles can be executed simultaneously across distinct gangs.

tile (2 , 2)




```
!$acc kernels loop tile(2,2)
do x = 1, 4
  do y = 1, 4
    a(x,y) = a(x,y) + 1
  end do
end do
```

Usage of `gang/worker/vector`

Modifying levels of parallelism is more an art than a science, requiring intuition and experimentation to find the optimal arrangement for a given kernel, which also depends on kernel problem size. Sometimes, refactoring the kernel algorithm is recommended instead. Here are some main points:

- `gang` parallism is often best for outer loops and `vector` for inner loops.
- `vector` should be of lengths 32 or multiples of 32 (NVIDIA) since each SIMT warp is length 32.
- The number/length of parallelism level should correlate with expected size of a loop and should not be greater than the number of steps in that loop.
- `num_workers` * `vector_length` is the number of threads in each **CUDA block**, max 1,024 (NVIDIA).
- More often than not, it is best to let the compiler decide the number of gangs.
- Effective tuning can often be done by only adjusting the `vector_length`.
- Usage of a `worker` loop is very rare and often reserved for when `vector_length` is small, ie less than 128.
- **Coallescing memory access** in `vector` loops is likely most important consideration, ie threads access contiguous memory elements in matrix collumns in FORTRAN (rows in C/C++).

Coallesced Memory Example in FORTRAN

```
!$acc parallel
!$acc loop gang
do j = 1, ny
    !$acc loop vector
    do i = 1, nx
        A(i,j) = B(i,j) + C(i,j)
    end do
end do
!$acc end parallel
```

The below example is uncoalesced after switching the `i/j` in the loops (this is opposite in C/C++):

```
!$acc parallel
!$acc loop gang
do i = 1, nx
    !$acc loop vector
    do j = 1, ny
        A(i,j) = B(i,j) + C(i,j)
    end do
end do
!$acc end parallel
```

EXERCISE - Manually Tune Performance of Select MiniWeather Kernels

Try out some of the previous performance tuning directives in miniWeather_mpi_exercise2.F90.

It is recommended to modify user parameters at **Line 57** such that `_NX=1024`, `_NZ=512`, and `_SIM_TIME=10`.

This sets up a sufficiently large problem to fill GPU SMs. Short simulation time promotes rapid profile driven development. You might also try even larger domain sizes but runtime will increase. Focus on just one kernel at a time (choose a random kernel or select the most costly kernel. See from timing output via `NVCOMPILER_ACC_TIME=1` in last few exercises).

`make` and run the new executable and record how performance changes with each modification.

1. **Can you beat the performance of the `collapse()` clause and the automatic choices made by the compiler?**
2. **Share your results in Slack (specify kernel you chose) and see what others were able to achieve.**

MiniWeather Kernel L###	CPU Time (s)
BaseLine (on V100)	XX
clause - gang/vector	XX
clause - tile(#,#,#)	XX
clause - tile(*,*,*)	XX
clause - vector_length(XX)	XX
...	XX

`make` and run the new executable and record how performance changes with each modification.

1. **Can you beat the performance of the `collapse()` clause and the automatic choices made by the compiler?**
2. **Share your results in Slack (specify kernel you chose) and see what others were able to achieve.**

MiniWeather Kernel L###	CPU Time (s)
BaseLine (on V100)	XX
clause - gang/vector	XX
clause - tile(#,#,#)	XX
clause - tile(*,*,*)	XX
clause - vector_length(XX)	XX
...	XX

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

`make` and run the new executable and record how performance changes with each modification.

1. **Can you beat the performance of the `collapse()` clause and the automatic choices made by the compiler?**
2. **Share your results in Slack (specify kernel you chose) and see what others were able to achieve.**

MiniWeather Kernel L###	CPU Time (s)
BaseLine (on V100)	XX
clause - gang/vector	XX
clause - tile(#,#,#)	XX
clause - tile(*,*,*)	XX
clause - vector_length(XX)	XX
...	XX

In [ ]:

```
make -C fortran/build openacc_test_ex2
```

In [ ]:

```
cd fortran/build
qcmd -A $PROJECT -q $QUEUE -l select=1:ncpus=1:ngpus=1 -l gpu_type=$GPU_TYPE -l walltime=60 -v NVCOMPILER_ACC_TIME=1 -- \
$PWD/check_output.sh $PWD/openacc_test_ex2 1e-13 4.5e-5
cd ../..
```

Suggested Resources

- Matt Norman's A Practical Introduction to GPU Refactoring in FORTRAN with Directives for Climate
- May 2021, OpenACC Programming and Best Practices Guide and Github
- OpenACC 2.7 Quick Reference Guide
- Official OpenACC 3.2 Full Standard Specification - Not all updated features are implemented yet by compatible compilers
- If you want to dive deep into lower level control and optimization of GPU performance, check out Oak Ridge National Lab's CUDA Training Series.