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 07_HandsOnMPASA/07_HandsOnMPASA.ipynb. Be sure to clone (if needed) and update/pull the NCAR GPU_workshop directory.

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

Set the PROJECT code to a currently active project, i.e. 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 [ ]:

```bash
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 \( \text{NX}=1024 \) and \( \text{NZ}=512 \), the most expensive kernel in terms of compute time was at Line 231 in the \text{semi_discrete_step} subroutine, with \text{NVCOMPILER_ACC_TIME} statistics highlighted below:

The arrangement of \text{gang/worker/vector} units is provided by \text{grid: [NUM_GANGS]} and \text{block: [VECTOR_LENGTH \times 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.
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.
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.

```fortran
!$acc parallel loop *** async
  do ll = 1 , NUM_VARS
    !$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
enddo
```
1. Using `worker/vector/seq` on each loop respectively, the profiler shows `grid: [1]  block: [32x4]`. Why is this arrangement the least performant?

<table>
<thead>
<tr>
<th>MiniWeather Kernel L231, <code>semi_discrete_step</code></th>
<th>Total Device Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseLine (on V100) - collapse(3) auto vector_length(128)</td>
<td>62,936</td>
</tr>
<tr>
<td>clause - gang/worker/vector on each loop repectively</td>
<td>852,859</td>
</tr>
<tr>
<td>clause - worker/vector/seq (Move NUM_VARS innermost, seq)</td>
<td>2,271,059</td>
</tr>
<tr>
<td>clause - gang/vector/seq (Move NUM_VARS innermost, seq)</td>
<td>72,584</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>MiniWeather Kernel L231, semi_discrete_step</th>
<th>Total Device Time ((\mu s))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (on V100) - collapse(3) auto vector_length(128)</td>
<td>62,936</td>
</tr>
<tr>
<td>clause - collapse(3) vector_length(32)</td>
<td>100,797</td>
</tr>
<tr>
<td>clause - collapse(3) vector_length(64)</td>
<td>63,010</td>
</tr>
<tr>
<td>clause - collapse(3) vector_length(256)</td>
<td>62,990</td>
</tr>
<tr>
<td>clause - collapse(3) vector_length(512)</td>
<td>63,032</td>
</tr>
<tr>
<td>clause - collapse(3) vector_length(1024)</td>
<td>66,458</td>
</tr>
</tbody>
</table>
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?

<table>
<thead>
<tr>
<th>MiniWeather Kernel L231, semi_discrete_step</th>
<th>Total Device Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (on V100) - collapse(3) auto vector_length(128)</td>
<td>62,936</td>
</tr>
<tr>
<td>clause - <code>tile(32,32,NUM_VARS)</code> INCORRECT</td>
<td>26,992</td>
</tr>
<tr>
<td>clause - <code>tile(32,32,1)</code></td>
<td>73,476</td>
</tr>
<tr>
<td>clause - <code>tile(32,8,NUM_VARS)</code></td>
<td>77,124</td>
</tr>
<tr>
<td>clause - <code>tile(32,1,NUM_VARS)</code></td>
<td>65,040</td>
</tr>
<tr>
<td>clause - <code>tile(1024,1,1)</code></td>
<td>67,393</td>
</tr>
<tr>
<td>clause - <code>tile(128,1,NUM_VARS)</code></td>
<td>66,421</td>
</tr>
<tr>
<td>clause - <code>tile(128,2,NUM_VARS)</code></td>
<td>74,295</td>
</tr>
<tr>
<td>clause - <code>tile(128,4,NUM_VARS)</code> INCORRECT</td>
<td>35,999</td>
</tr>
<tr>
<td>clause - <code>tile(*,*,*)-&gt;32,4,32</code></td>
<td>150,374</td>
</tr>
</tbody>
</table>
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

Baseline -> Porting

Porting -> Optimize

Optimize -> Portability Check

Portability Check -> Integrate

Integrate -> Baseline

Redo or Move on to Next Code

>10% compromise?

KGen

OpenACC Directives

KGen

Profile & Analyze

Testing

Code Refactoring

Software & Architecture

Configuration & Accuracy

Benchmark

Verification

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.
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 identify hotspots in the code.

**Experimental settings**
- **Quasi-uniform 60-km resolution** (163,842 cells)
- $\Delta t=180$ 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 \texttt{!$acc kernels ...} and similar directives will generate lists of variables needed to manage data movement for each compute region.

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

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 configuraion 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 [ ]:

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

cqmd -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 [ ]:

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

In [ ]:

```bash
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
```

<table>
<thead>
<tr>
<th>MPAS-A Kernels L###</th>
<th>Device Time (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline (on V100) - !_acc kernels</td>
<td>XX</td>
</tr>
<tr>
<td>clause - gang/vector</td>
<td>XX</td>
</tr>
<tr>
<td>clause - tile(##,##)</td>
<td>XX</td>
</tr>
<tr>
<td>clause - tile(<em>,</em>)</td>
<td>XX</td>
</tr>
<tr>
<td>clause - vector_length(XX)</td>
<td>XX</td>
</tr>
<tr>
<td>clause - num_workers(XX)</td>
<td>XX</td>
</tr>
<tr>
<td>...</td>
<td>XX</td>
</tr>
</tbody>
</table>
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 accommodate availability of a NVIDIA engineer to present on Multi-GPU programming. Look out for upcoming announcements.