Accelerating the Cloud Scheme Within the Unified Model for CPU-GPU Based High-Performance Computing Systems

Wei Zhang, Min Xu, Mario Morales Hernandez, Matthew Norman, Salil Mahajan and Katherine Evans

2019 MultiCore 9 Workshop, Sep 25th 2019

Thanks to the US Air Force, DOE OLCF, Met Office and EPCC for support and help!
Content

• Overview
  – Introduction of project and motivation
  – CASIM cloud scheme
  – OLCF Summit supercomputer

• CASIM on Summit, from CPU to GPU: current status and future plans
Overview

Forecast Extension to Hydrology – From Rainfall to Flood

Rainfall
- Weather Forecasting Models

Runoff
- NASA
- Land Information System (LIS)

Streamflow
- ERDC
- Streamflow Prediction System (SPT)

Inundation
- ORNL/TTU
- TRITON-GPU

UM Optimization
(Cloud Scheme, Radiation Scheme)
Overview

Air Force Weather and ORNL Collaboration

Met Office Unified Model (UM)

Cloud AeroSol Interacting Microphysics (CASIM)

OLCF Summit
What is cloud microphysics?

Cloud microphysics concerns the mechanisms by which cloud droplets generated from water vapor and the particles in the air, and grow to form raindrops, ice and snow.


- Relative sizes of cloud droplets, raindrops and cloud condensation nuclei (CCN)

  - CCN
    - $r = 0.1$, $n = 10^6$, $v = 0.0001$
  - Large cloud droplet
    - $r = 50$, $n = 10^3$, $v = 27$
  - Typical cloud droplet
    - $r = 10$, $n = 10^6$, $v = 1$
  - Typical raindrop
    - $r = 1000$, $n = 1$, $v = 650$

$r$: radius (um)

$n$: number per liter of air

$v$: fall speed (cm/s)
Why cloud microphysics matters?

- Schematics of some of the warm cloud and precipitation microphysical processes

- The evolution of cloud/rain mass, the number concentration of droplets and particles

- Latent heating/cooling, Temperature
  - condensation, evaporation, deposition, sublimation, freezing, melting

- Affecting surface processes, radiative transfer, cloud-aerosol-precipitation interactions...
Cloud AeroSol Interacting Microphysics - CASIM

• Long-term replacement for UM microphysics and the default microphysics

• User definable
  – number of cloud species (e.g., cloud, rain, ice, snow, graupel)
  – number of moments to describe each species (1: mass, 2: 1 + number, 3: 2 + radar reflectivity)

• Detailed representation of aerosol effects and in-cloud processing of aerosol
  – increase accuracy
  – more intensive calculation
• **CASIM/src**
  - Modern Fortran code
  - 16329 total lines, 116 subroutines

(Run in UM, same COPE case, different microphysics schemes, adopted from Met office technical paper)

Wallclock for KiD_1D Simulations on Summit (no parallelism)
(Same model, same cumulus case, different microphysics schemes)

**HPC + GPU Computing**
• **Objects**
  – Applying new coding to CASIM for GPUs
  – Developing algorithms that will be suited for accelerated machines (Summit now, Frontier, in the future)

• **Compilers**
  – PGI (19.7 on Summit)
  – Cray (will be available when Frontier comes out)
  – CLAW (source-to-source translator, Produces code for the target architectures and directive languages, [https://github.com/claw-project/claw-compiler](https://github.com/claw-project/claw-compiler))

• **Directive**
  – OpenACC

• **Considerations**
  – Portability limitations, CPU-GPU communication
  – Validation & Verification, Robust testing
  – The software stack for these new computing systems
CASIM on Summit

- **Parent model:** The Kinematic Driver Model (KiD, Shipway and Hill, 2011)
  - Kinematic framework to constrain the dynamics and isolate the microphysics
  - Original KiD has no parallelization directives

- **Baseline case:** 2D squall line case
  - $n_x = 320$, $dx = 750$ m, $n_z = 48$, $dz = 250$ m
  - $dt = 1$ s, $t_{\text{total}} = 3600$ s, output saved every 60 s
• Step 1. Access KiD-CASIM 2D-SQUALL Performance on CPU
  – Profiling tool: General Purpose Timing Library (GPTL)
  https://jmrosinski.github.io/GPTL/

<table>
<thead>
<tr>
<th>total</th>
<th>Called</th>
<th>Recurse</th>
<th>Wallclock</th>
<th>min</th>
<th>self.OH</th>
<th>parent.OH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>1187.963</td>
<td>1187.963</td>
<td>1187.963</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>1187.963</td>
<td>1187.963</td>
<td>1187.963</td>
<td>0.000</td>
</tr>
</tbody>
</table>

| mphys_interface_mphys_column_ | 3600 | 1019.095 | 0.784 | 0.071 | 0.000 | 0.000 |
| mphys_casim_mphys_casim_interface_ | 3600 | 1019.088 | 0.784 | 0.071 | 0.000 | 0.000 |

CASIM in KiD: 1019.095/1187.963 = 85.79%

micro_main in CASIM: 987.515/1019.095 = 96.90%
• Step 2: Get CASIM ready for GPU (ongoing)

• General idea:
  – Optimize most time-consuming parts
  – Avoid/minimize data transfer between CPU and GPU

Idealized solution: GPU region sandwiched between two CPU calculation regions

but ....
- Challenge 1: Derived Data Type

1) `ta=tesla:deepcopy (testing)`
2) change to flat array (bit-for-bit on CPU confirmed)

```fortran

type :: process_rate
    real(wp), allocatable :: column_data(:)
end type process_rate
...
type(process_rate), allocatable :: procs(:,:)
real(wp), target, allocatable :: procs_flat,:,::
allocate(procs(ntotalq, nprocs)
allocate(procs_flat(nz, ntotalq, nprocs)
do iprocs=1, nprocs
    do iq=1, ntotalq
        procs(iq, iprocs)%column_data => &
        procs_flat(1:nz, iq, iprocs)
    end do
end do
call micro_common(..., procs_flat, ...)
```
Challenge 2: 
n-loop and k-loop are not parallelable now;

Hotspots locate deep in the call tree
• Former work done in EPCC and UK Met Office:
  – Porting the microphysics model CASIM to GPU and KNL Cray machines (Brown et al., 2016)

  – Parent model: the Met Office NERC Cloud Model (MONC)
  – Compiler: Cray
  – Directive: OpenACC
  – Offloaded the whole CASIM onto GPU on Piz Daint XC50 and XC30

subroutine CASIM()
  !$acc parallel async(ACC_QUEUE)
  !$acc loop collapse(2) gang worker vector
  do i = is, ie
    do j = js, je
      call microphysics_common(i,j, ...)
    end do
  end do
  !$acc end loop
  !$acc end parallel
end subroutine CASIM

subroutine microphysics_common(i,j, ...)
  !$acc routine seq
  ...
end subroutine microphysics_common
Lesson we learned: Much more code refactoring is needed to

- Maximize the number of parallelization in GPU
- Minimize the amount of data transfer between CPU and GPU

From: Accelerating the microphysics model CASIM using OpenACC, Alexandr Nigay, 2016
How to increase the parallelization?

\[
\text{Flux}^k_n = f(\text{Flux}^{k+1}_{n-1})
\]
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ……  n=nsubstep-1  n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3

...}

k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ......  n=nsubstep-1  n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3
...
k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ......  n=nsubstep-1  n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3

......

k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

n=1    n=2    n=3    ......    n=nsubstep-1    n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3

......

k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

\[ n=1 \quad n=2 \quad n=3 \quad \ldots \quad n=n_{\text{substep}}-1 \quad n=n_{\text{substep}} \]

\[ k=\text{nz} \quad k=\text{nz}-1 \quad k=\text{nz}-2 \quad k=\text{nz}-3 \quad \ldots \quad k=3 \quad k=2 \quad k=1 \]
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ......  n=nsubstep-1  n=nsubstep
k=nz  k=nz-1  k=nz-2  k=nz-3  ......  k=3  k=2  k=1
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ......  n=nsubstep-1  n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3
......
k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

n=1  n=2  n=3  ...... n=nsubstep-1  n=nsubstep

k=nz
k=nz-1
k=nz-2
k=nz-3

......

k=3
k=2
k=1
Possible new way for parallelizing n-loop and k-loop

\[
\begin{align*}
n &= 1 \\
    &= 2 \\
    &= 3 \\
    & \cdots \\
    &= n_{\text{substep}} - 1 \\
    &= n_{\text{substep}} \\
\end{align*}
\]

\[
\begin{align*}
k &= n_z \\
    &= n_z - 1 \\
    &= n_z - 2 \\
    & \cdots \\
    &= 1 \\
\end{align*}
\]

limitation: \( n_{\text{substep}} \geq n_z \)
How to reduce the memory traffic?
  - many conditional if branches

```f90
if (nq_l > 0) qfields(:,i_ql)=q1(ks:ke,i,j)
if (nq_r > 0) qfields(:,i_qr)=q2(ks:ke,i,j)
if (nq_l > 1) qfields(:,i_nl)=q3(ks:ke,i,j)
if (nq_r > 1) qfields(:,i_nr)=q4(ks:ke,i,j)
if (nq_r > 2) qfields(:,i_m3r)=q5(ks:ke,i,j)
if (nq_i > 0) qfields(:,i_qi)=q6(ks:ke,i,j)
if (nq_s > 0) qfields(:,i_qs)=q7(ks:ke,i,j)
if (nq_g > 0) qfields(:,i_qg)=q8(ks:ke,i,j)
if (nq_i > 1) qfields(:,i_ni)=q9(ks:ke,i,j)
if (nq_s > 1) qfields(:,i_ns)=q10(ks:ke,i,j)
```

- lookup table for gamma function in sedimentation.F90

| sedimentation_sedr_ | 2.3e+07 | – | 414.461 |
| special_gammalookup_ | 3.8e+09 | – | 172.311 |
| special_set_gammalookup_ | 1 | – | 0.168 |
| special_gammafunc1_ | 1.0e+06 | – | 0.077 |
Future Plan

• Continue to do code refactoring to expose more parallelism
  – Restructure the loops when it’s necessary

• Continue to optimize the data locality
  – Reduce the data transfer between CPU and GPU
  – Reduce the number of system memory accesses

• First do the optimization with KiD-CASIM, then couple accelerated CASIM to UM for global simulation
Thank you