Accelerating the Performance of Modal Aerosol Module (MAM) of E3SM Using OpenACC

Hongzhang Shan[1)], Zhengji Zhao[2)], and Marcus Wagner[3)]

[1)] Lawrence Berkeley National Laboratory,
[2)] National Energy Scientific Computing Center
[3)] Cray, Inc

WACCPD 19, Denver CO

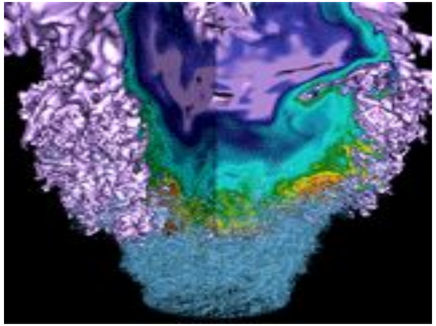
Outline

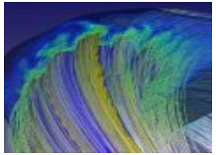
- Introduction
- MAM Algorithms and Kernels
- Offloading Four Representative Kernels in MAM to GPU Using OpenACC directives
- MAM Performance on Summit
- Summary and Conclusion

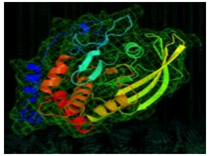
Introduction

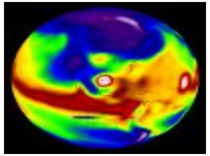
- Energy Exascale Earth System Model (E3SM) is a state-of-the-art earth system simulation code
  - It has a large code base with over a million lines of Fortran code
  - Production code are currently optimized for advanced CPU systems
- Making effective use of GPUs, however, remains a challenge
- In this work, using the modal aerosol module (MAM) of E3SM as a driving example, we investigated how to effectively offload computational tasks to GPUs
- We chose to work with OpenACC directives

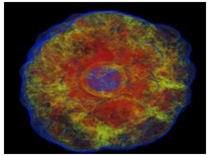
MAM Algorithms and Kernels

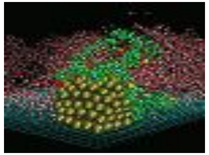










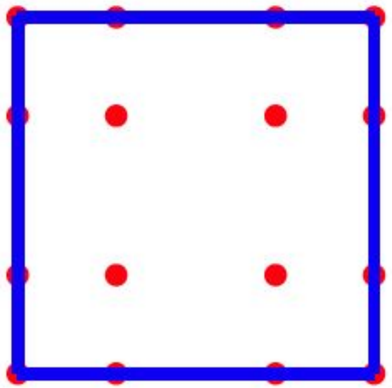


Modal Aerosol Module (MAM) of E3SM

- E3SM was developed to reliably project decade-to-century scale changes that could critically impact the U.S. energy sector. It combined the atmosphere, ocean, land, river, ice, and other components.
- The computation of the atmosphere component is based upon the Spectral Element (SE) numerical discretization of underlying PDEs for stratified, hydrostatic fluid dynamics on rotating spheres.
- MAM is a submodule from the atmosphere component that plays an important role in the climate system by influencing the Earth's radiation budgets and modifying cloud properties. It predicts the mass and mixing ratios of cloud liquid and cloud ice, diagnoses the mass and mixing ratios of rain and snow, and handles complicated conversions between cloud hydrometeors.

MAM in E3SM

a)



b)

- E3SM models the Earth with a cubed-sphere grid (6 faces) as shown in Fig. a).
- The resolution of the meshes is defined as the number of spectral elements `ne` along the edge of each cube face. ($6ne^2$ elements total in the mesh)
- Each element contains a `np*np` tensor product of Gauss-Lobatto-Legendre (GLL) points depicted in Figure b), the number of unique points (physics columns)
- There exists another dimension, namely the vertical direction (except the sphere faces).
- Computations between physical columns are independent

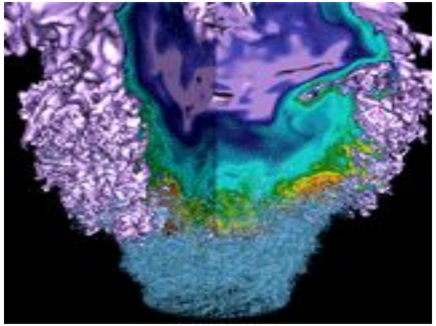
MAM in E3SM

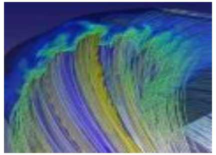
Loop structure for computations

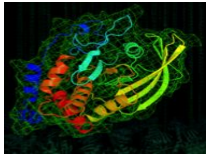
```
do j = 1, nchunks               !number of chunks
  do k = 1, nlev                !vertical levels, may have data dependency
    do i = 1, ncols(j)          !number of columns in chunk j
                                !sum(ncols(j) = total physical columns
      computation_kernels()     !many different kernels
    enddo
  enddo
enddo
```

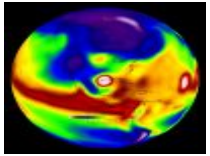
- In the parallel implementation physical columns are distributed among the processes based on a set of load balancing strategies.
- To get better caching effects, all the columns assigned to a process will be grouped in a data structure called a chunk.
- In each chunk, a maximum number of columns `PCOL` is specified at compilation time.

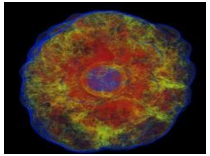
Experiment Configuration

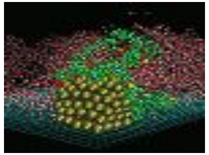




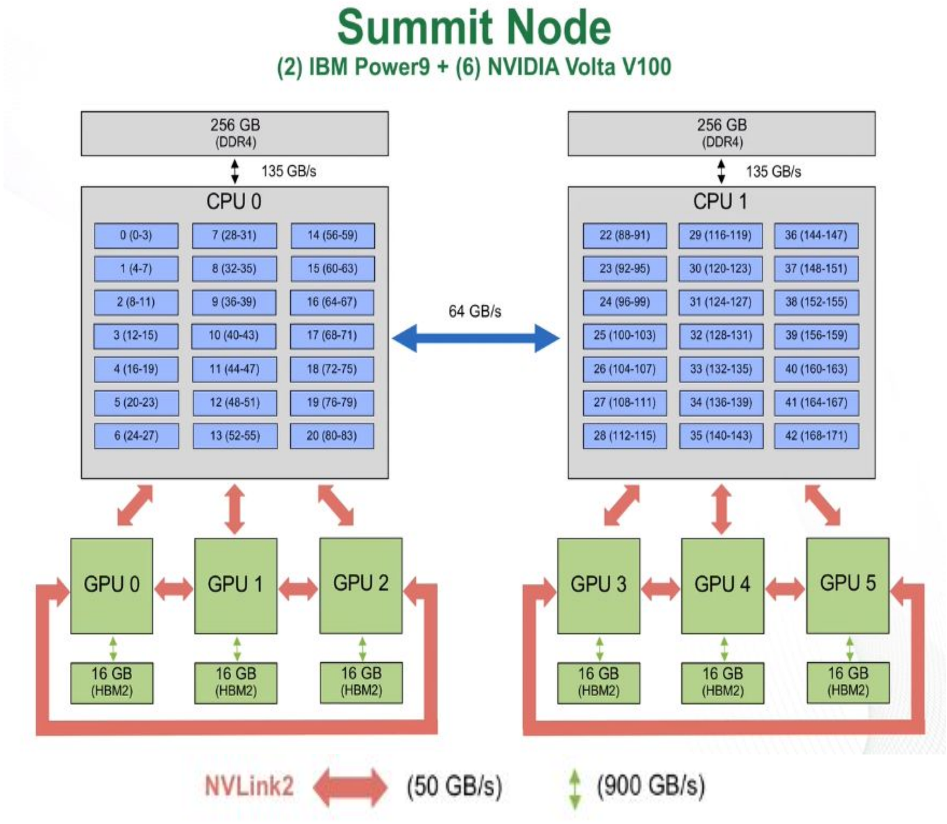








Experiment System – Summit at ORNL

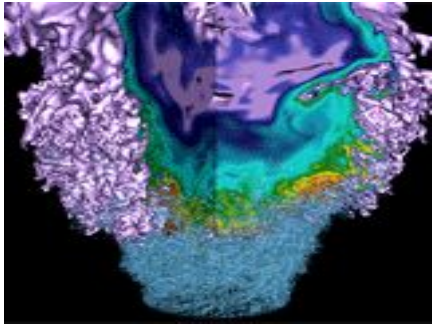


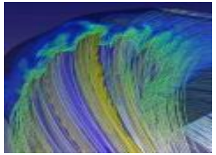
- Theoretical peak ~200 PF (dp),
- Each Summit node has two IBM Power9 processors with six Nvidia V100 GPUs.
- Power9 CPUs are connected with GPUs through dual NVLINK
- 512 GB of DDR4 memory for Power9 CPUS and 96 GB of HBM for GPUs.
- Each Nvidia V100 has 80 SMs, 16 GB of HBM, and a 6 MB L2 cache.
- Each SM contains 64 FP32, 64 INT32, 32 FP64 cores; partitioned to four processing blocks, each with a warp scheduler.

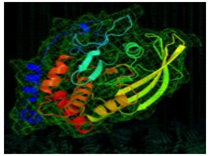
Experiment Setup

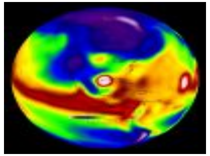
- E3SM: https://github.com/E3SM-Project/E3SM.git, branch shz0116/cam/cam_openacc
- We used the PGI compiler version 19.4, Spectrum MPI version 10.3.0.1, and CUDA 10.1.168
- Other libraries used in the E3SM code included NETCDF 4.6.1, NETCDF-FORTRAN 4.4.4, ESSL 6.1.0 , Parallel NETCDF 1.8.1, and HDF5 1.10.3
- The data set for E3SM is SMS_PS_Ld5.ne16_ne16.FC5AV1C-L, which stresses the atmosphere physics. Here, ne16 ne16 defines the cubed sphere grid resolution

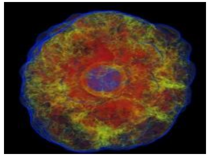
Offloading Four Representative Kernels in MAM to GPU Using OpenACC directives

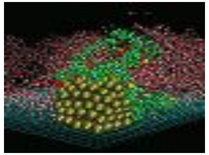












U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Offloading MAM Kernels to GPUs

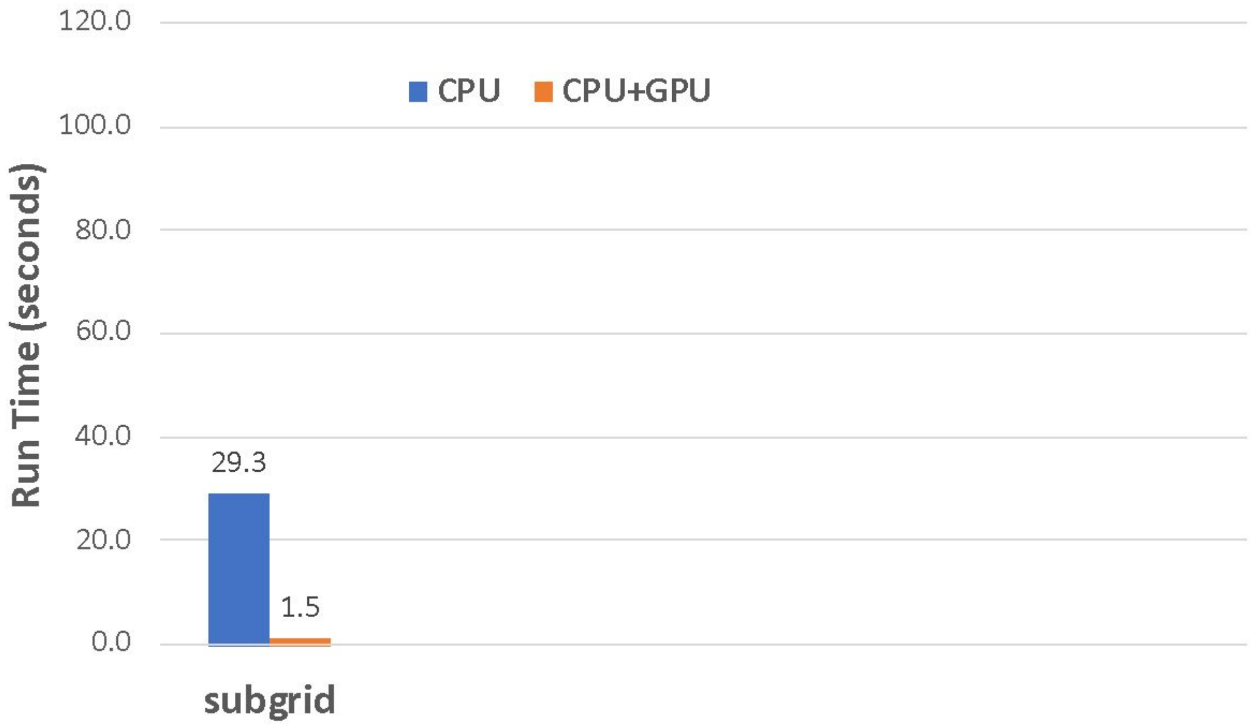
- Data transfer was not trivial in MAM
  - MAM has a large code base with tens of thousands of lines of source code
  - MAM does checkpointing with various I/O operations scattered all over the code
  - an excessive number of temporary subroutines or function variables need to be promoted and explicitly allocated on the GPU memory as well
- MAM has a flat profile, its run time is distributed across many functions, meaning we could not focus on just a couple of loops
- The programming effort needed to optimize different kernels also varies significantly by kernel. Some required a significant code refactoring

Kernel: subgrid_mean_updraft

```
!$acc parallel loop collapse(2) copyin(wsig,w0) copyout(ww) private(zz,wa)
do k = 1, pver
  do i = 1, ncol
    sigma  = max(0.001_r8, wsig(i,k))
    wlarge = w0(i,k)
    xx = 6._r8 * sigma / nbin
    do ibin = 1, nbin              !constant nbin=50
      yy = wlarge - 3._r8*sigma + 0.5*xx
      yy = yy + (ibin-1)*xx
      zz(ibin) = yy * exp(-1.*(yy-wlarge)**2/(2*sigma**2))/(sigma*sqrt(2*pi))*xx
      if (zz(ibin) .gt. 0._r8) then
        wa(ibin) = zz(ibin)
      else
        wa(ibin) = 0._r8
      endif
    end do
    sum_wa = sum( wa(:))
    if (sum_wa .gt. 0._r8) then
      ww(i,k) = sum_wa
    else
      ww(i,k) = 0.001_r8
    end if
  enddo
enddo
```

- calculates the mean updraft velocity

Kernel Performance on a Summit Node



Kernel: hetfrz_classnuc_cam_calc

```
1 !$acc declare create(ncnst, nmodes, ...)!create module variables on device
2 !$acc update device(ncnst, nmodes, ...)
3 ...
4 !$acc enter data create(total_aer_num, ...)
5 ...
6 !$acc parallel loop collapse(2) private(fn), copyin(t,pmid) &
7 !$acc& copyout(frzbccnt,...) default(present)
8   do k = top_lev, pver
9     do i = 1, ncol
10      if (t(i,k) .gt. 235.15_r8 .and. t(i,k) .lt. 269.15_r8) then
```

Lines 11-34 in next slide

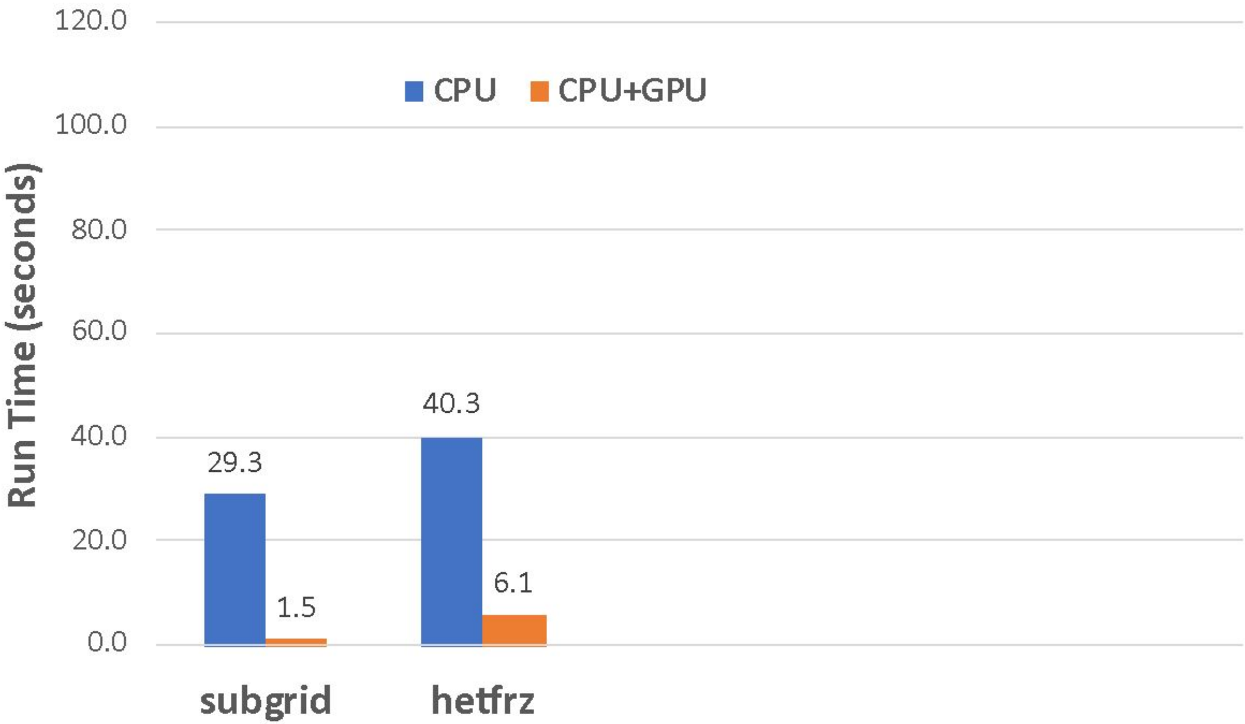
```
35      end if
36    end do
37  end do
38 ...
39 !$acc exit data delete(total_aer_num, ...)
```

- calculates the heterogeneous freezing rates from classical nucleation theory

Kernel: hetfrz_classnuc_cam_calc

```
qcic = min(qc(i,k)/lcldm(i,k), 5.e-3_r8)
ncic = max(nc(i,k)/lcldm(i,k), 0._r8)
con1 = 1._r8/(1.333_r8*pi)**0.333_r8
r3lx = con1*(rho(i,k)*qcic/(rhoh2o*max(ncic*rho(i,k), 1.0e6_r8)))**0.333_r8
r3lx = max(4.e-6_r8, r3lx)
supersatice = svp_water(t(i,k))/svp_ice(t(i,k))
                !svp_water and svp_ice are two subroutines
fn(1) = factnum(i,k,mode_accum_idx)
if (nmodes == MAM3_nmodes .or. nmodes == MAM4_nmodes) then
    fn(2) = factnum(i,k,mode_accum_idx)
    fn(3) = factnum(i,k,mode_coarse_idx)
else if (nmodes == MAM7_nmodes) then
    fn(2) = factnum(i,k,mode_finedust_idx)
    fn(3) = factnum(i,k,mode_coardust_idx)
end if
call hetfrz_classnuc_calc( &
   deltatin,  t(i,k),  pmid(i,k),  supersatice,   &
   fn,  r3lx,  ncic*rho(i,k)*1.0e-6_r8,  frzbcimm(i,k),  frzduimm(i,k),   &
   frzbccnt(i,k),  frzducnt(i,k),  frzbcdep(i,k),  frzdudep(i,k),  hetraer(:,i,k), &
   awcam(:,i,k), awfacm(:,i,k), dstcoat(:,i,k), total_aer_num(:,i,k),  &
   coated_aer_num(:,i,k), uncoated_aer_num(:,i,k), &
   total_interstitial_aer_num(:,i,k), &
   total_cloudborne_aer_num(:,i,k), errstring)
               !hetfrz_classnuc_calc is a sequential routine with hundreds of lines}
```

Kernel Performance on a Summit Node



Kernel: ccncalc

```
do k=top_lev,pver
  do i=1,ncol
    a(i)=surften_coef/tair(i,k)
    smcoef(i)=smcoefcoef*a(i)*sqrt(a(i))
  enddo
  do m=1,ntot_amode
    phase=3
    call loadaer(state, pbuf, 1, ncol, k, &
         m, cs, phase, naerosol, vaerosol, hygro)
         !get data from pbuf to naerosol, vaerosol, and hygro
    where(naerosol(:ncol) .gt. 1.e-3_r8)
      amcube(:ncol)=amcubecoef(m)*vaerosol(:ncol)/naerosol(:ncol)
      sm(:ncol)=smcoef(:ncol)/sqrt(hygro(:ncol)*amcube(:ncol))
    elsewhere
      sm(:ncol)=1._r8
    endwhere
    do l=1,psat
      do i=1,ncol
        arg(i)=argfactor(m)*log(sm(i)/super(l))
        ccn(i,k,l)=ccn(i,k,l)+naerosol(i)*0.5_r8*(1._r8-erf(arg(i)))
      enddo
    enddo
   enddo
enddo
```

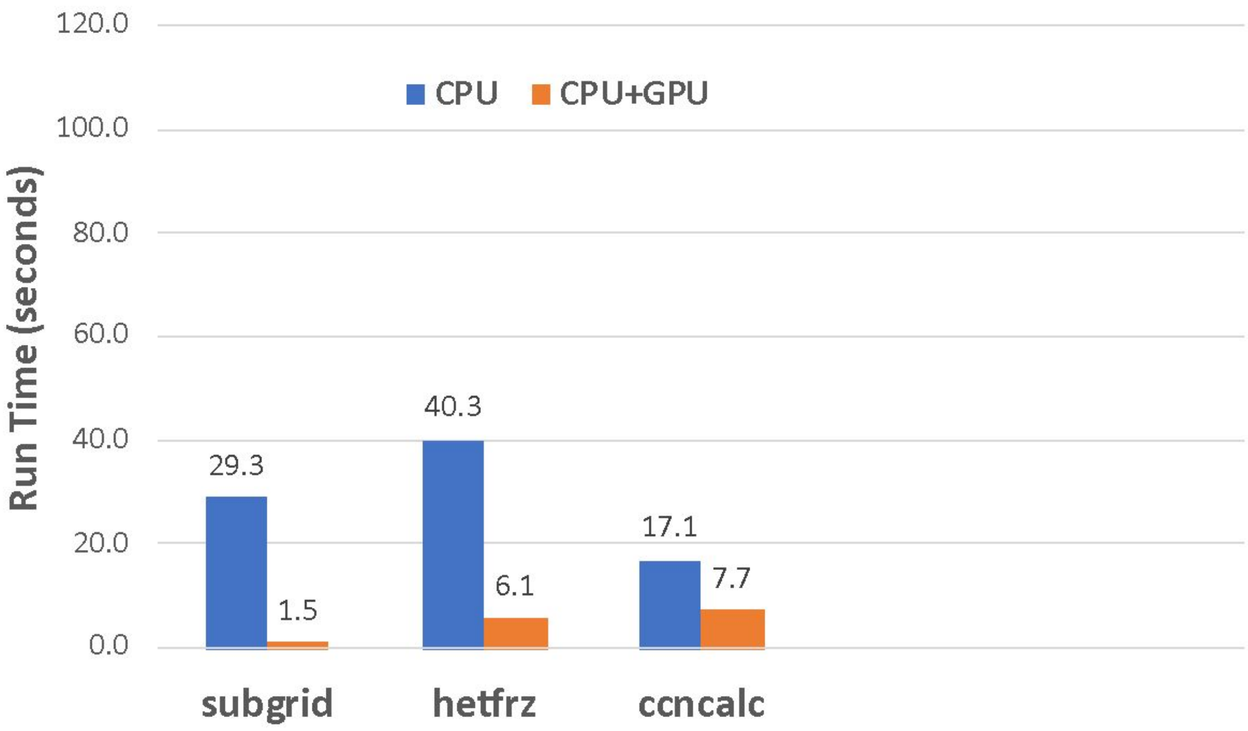
- calculate the number of concentrations of aerosols activated when cloud condensation nuclei are at supersaturation

GPU version

```
do k=top_lev,pver
do m = 1, ntot_amode
   call loadaer(state, pbuf, 1, ncol, k, &
        m, cs, phase, naerosol(:,m,k), vaerosol(:,m,k), hygro(:,m,k))
        !define naerosol, vaeroosol, hygro as 3D instead of 1D
enddo
enddo
```

```
$acc data copy(ccn) copyin(vaerosol, naerosol, hydro) &
$acc& copyin(super,amcubecoef,argfactor,tair,smccoefcoef,surften_coef)
$acc parallel loop private(a,smcoef,arg,sm,amcube,m,i,l) default(present)
do k=top_lev,pver
  do i=1,ncol
    a(i)=surften_coef/tair(i,k)
    smcoef(i)=smcoefcoef*a(i)*sqrt(a(i))
  enddo
  do m=1,ntot_amode
    phase=3
    where(naerosol(:ncol) .gt. 1.e-3_r8)
      amcube(:ncol)=amcubecoef(m)*vaerosol(:ncol.m.k)/naerosol(:ncol,m,k)
      sm(:ncol)=smcoef(:ncol)/sqrt(hygro(:ncol,m,k)*amcube(:ncol))
    elsewhere
      sm(:ncol)=1._r8
    endwhere
    do l=1,psat
      do i=1,ncol
        arg(i)=argfactor(m)*log(sm(i)/super(l))
        ccn(i,k,l)=ccn(i,k,l)+naerosol(i,m,k)*0.5_r8*(1._r8-erf(arg(i)))
      enddo
    enddo
  enddo
enddo
...
$acc end data
```

Kernel Performance on a Summit Node



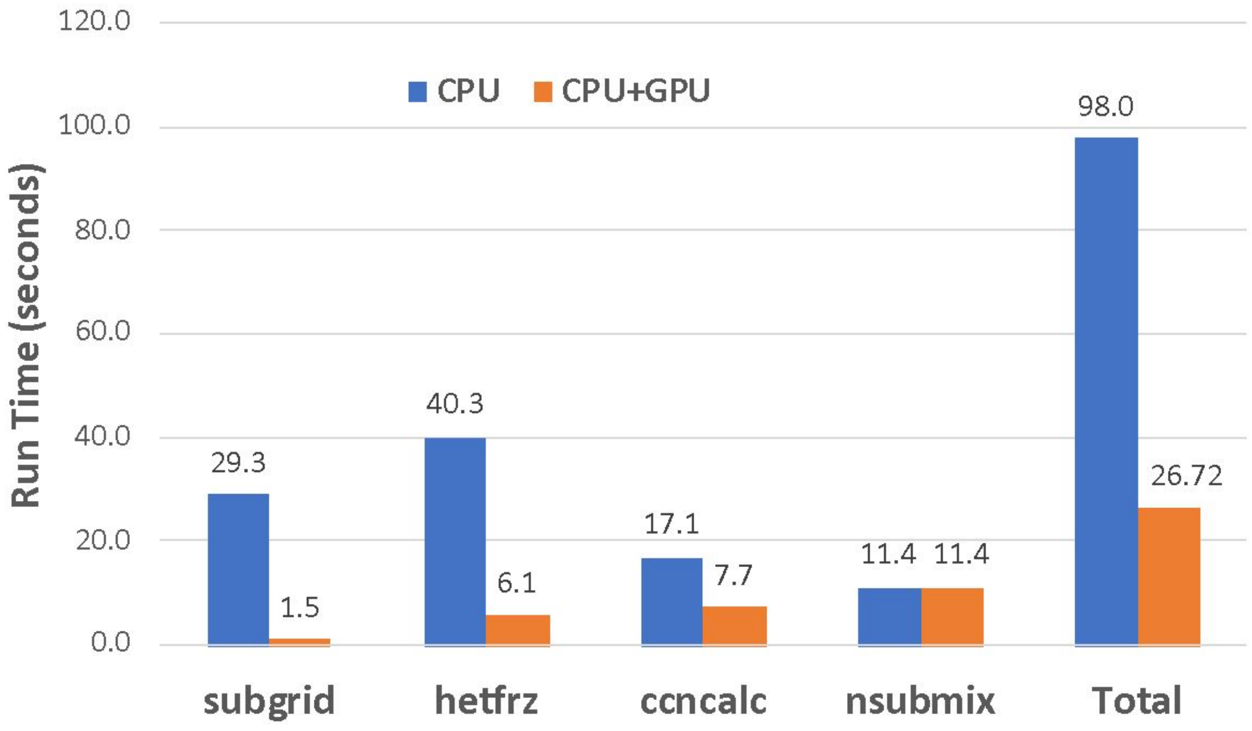
Kernel: nsubmix

```
1  do i = 1, ncol
2  ... !more than 400 lines of code
3    do n = 1, nsubmix
4      qncld(:) = qcld(:)
5      nnew <--> nsav !nnew = 1, nnsav=0
6      srcn(:) = 0
7      do m = 1, ntot_amode
8        mm = mam_idx(m,0)
9        srcn(top_lev:pver-1) = srcn(top_lev:pver-1) +  &
10           nact(top_lev:pver-1,m)*raercol(top_lev+1:pver,mm,nsav)
11       tmpa = raercol(pver,mm,nsav)*nact(pver,m) + &
12           raercol_cw(pver, mm, nsav) * (...)
13       srcn(pver) = srcn(pver) + max(0.0_r8, tmpa)
14     enddo
15     call explmix(qcld, srcn, ..., qncld) !compute qcld from qncld
16
17     do m = 1, ntot_amode
18       mm = mam_idx(m,0)
19       source(top_lev:pver-1) = &
20             nact(top_lev:pver-1,m)*(raercol(top_lev+1:pver,mm,nsav))
21       tmpa = ... !same as line 9
22       source(pver) = max(0.0_r8, tmpa)
23       call explmix(raercol_cw(:, mm, nnew), source, ..., raercol_cw(:, mm, nsav), ...)
24                                    !compute raercol_cw(,,nnew) from raercol_cw(,,nsav)
25       call explmix(raercol(:, mm, nnew), source, ..., raercol(:, mm, nsav), &
26         raercol_cw(:, mm, nsav))
27                 !compute raercol(,,nnew) from raercol(,,nsav) and raercol_cw(,,nsav)
28       do l = 1, nspec_amode(m)
29         mm = mam_idx(m, l)
30         source(top_lev:pver-1) = !same as line 17 except using mact instead nact
31         tmpa = !same as line 19 except using mact instead nact variable
32         source(pver) = max(0.0_r8, tmpa)
33         call explmix  !same as line 21
34         call explmix  !same as line 23
35       enddo
36     enddo
37   enddo
38 ...
```

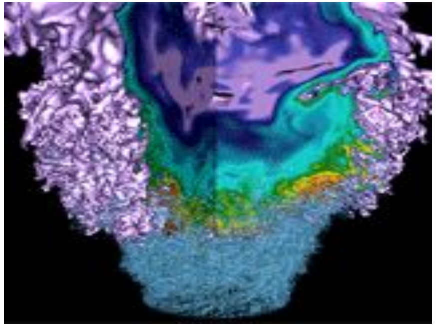
Kernel: restructured nsubmix

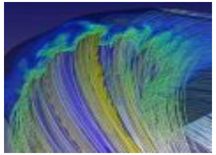
```
do mm = 1, ncnst_tot
  do k = top_lev, pver
    m   = mam_idx_1d(1, mm)
    l   = mam_idx_1d(2, mm)
    kp1 = min(k+1,pver)
    km1 = max(k-1,top_lev)
    if (l == 0) then
      tmpa = nact(k,m)*raercol(kp1,mm,nsav)
      if (k == pver) then
        tmpa = tmpa + raercol_cw(pver,mm,nsav)*(nact(pver,m) - taumix)
        tmpa = max(0.0_r8, tmpa)
      endif
    else
      tmpa = mact(k,m)*raercol(kp1,mm,nsav)
      if (k == pver) then
        tmpa = tmpa + raercol_cw(pver,mm,nsav)*(mact(pver,m) - taumix)
        tmpa = max(0.0_r8, tmpa)
      endif
    endif
    call explmix(raercol_cw(k, mm, nnew), source, ...)
    call explmix(raercol(k, mm, nnew), source, ...)
  end do
end do
```

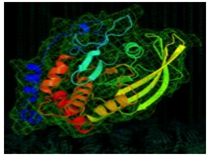
Kernel Performance on a Summit Node

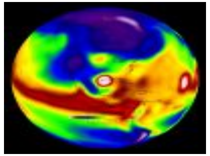


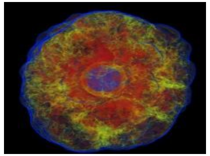
MAM Kernel Performance on Summit

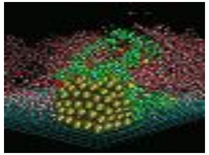












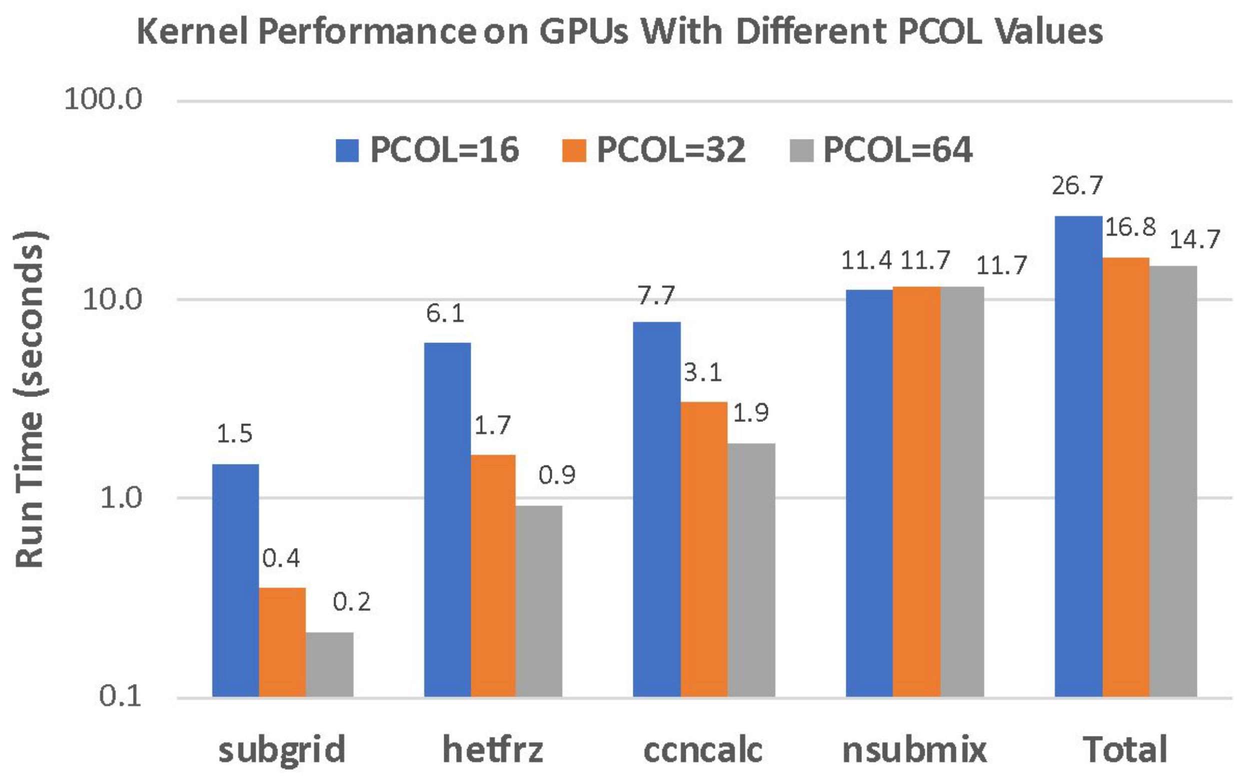
Kermel Performance Analysis

- MAM kernels are light, the average run time is within milliseconds.
- Secondly, the parallelism mainly comes from the vertical level `(pver=72)` and the number of columns in a block (pcol). Considering that the Nvidia GPUs use a warp size of 32 as the scheduling unit, neither `pver(= 72)` nor `ncol(<= 16)` is a perfect fit, resulting in thread resource waste.
- To improve the performance from an application perspective, the size of `pcol` and `pver` can therefore be aligned to multiples of warp size.

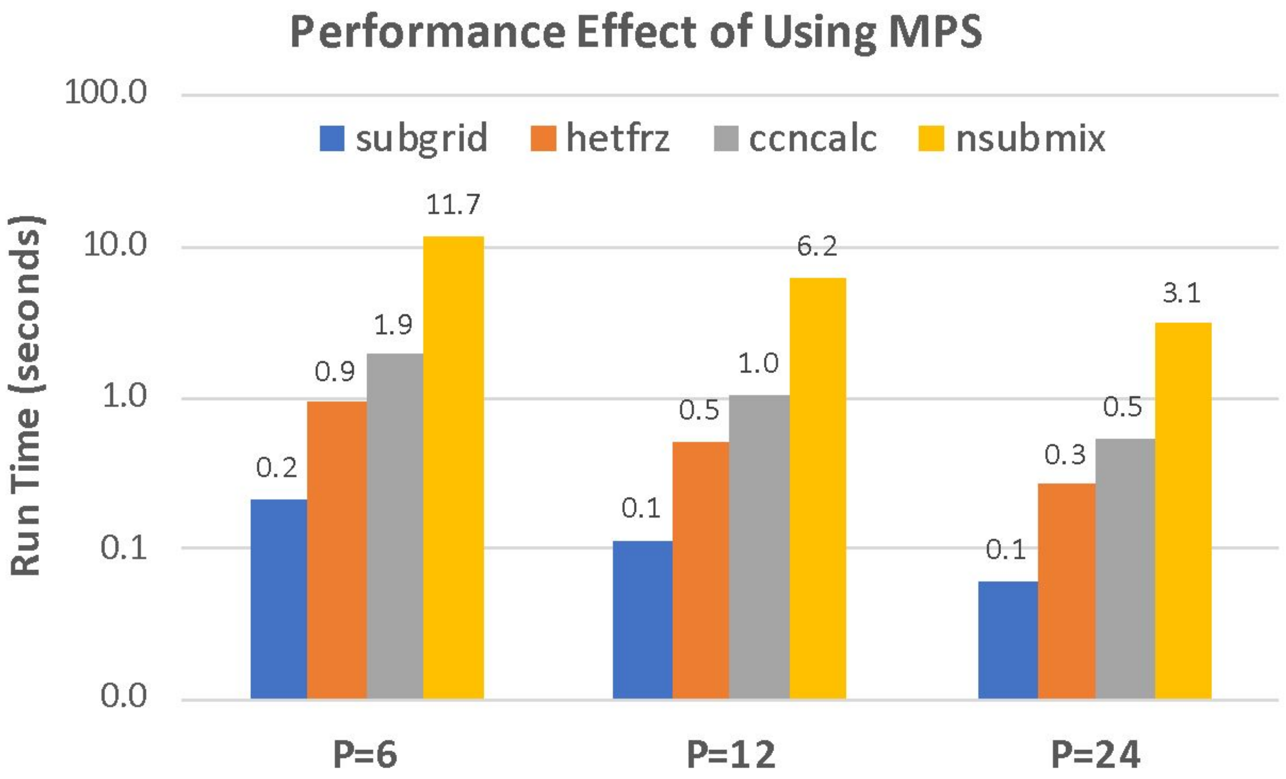
Kernel Average runtime on GPUs and CPUS

	CPU Total Time (s)	GPU Total Time	Called Times	Avg on CPU (ms)	Avg on GPU (ms)
subgrid	29.3	1.5	5520	5.3	0.3
hetfrz	40.3	6.1	5520	7.3	1.1
ccncalc	17.1	2.8 [3]	5520	3.1	1.4
nsubmix	11.4	11.4	88740	0.1	

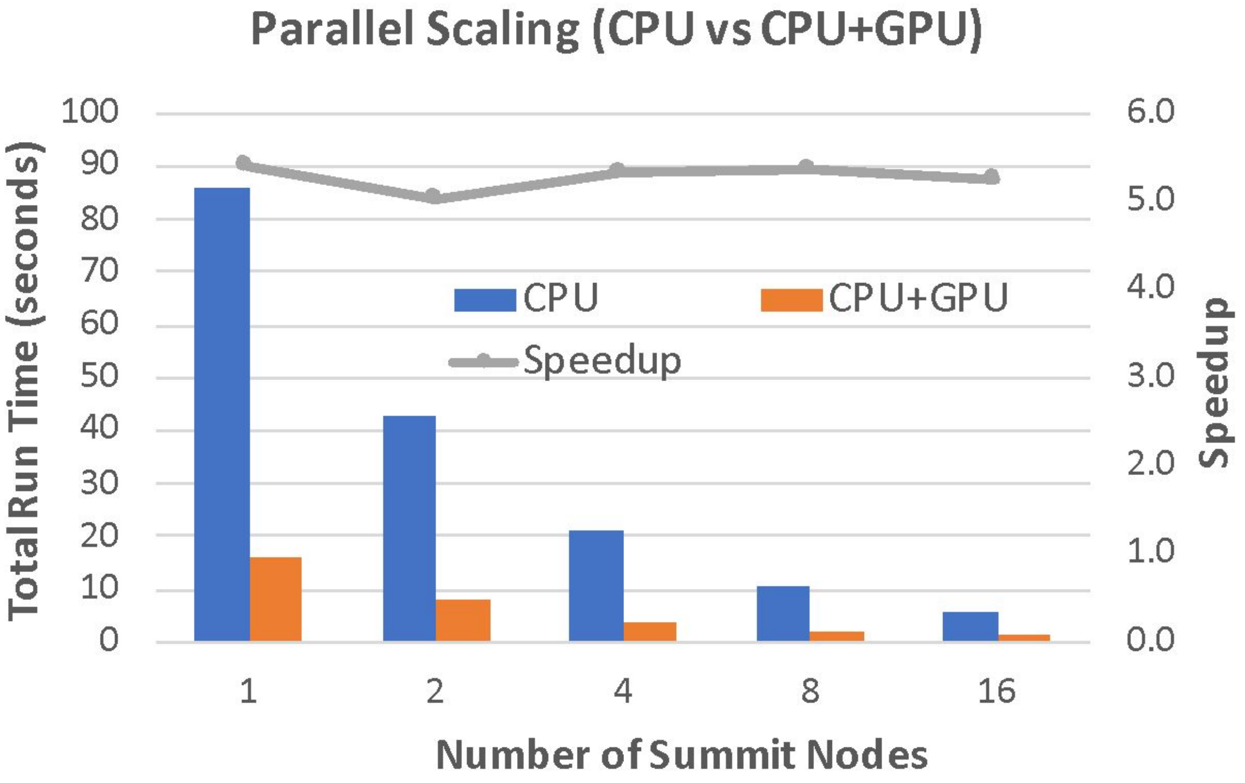
Kernel Performance vs PCOL Values



Kernel Performance vs MPS



Both CPU and CPU+GPU Versions Scale Well across Multiple Nodes



Summary and Conclusion

- We investigated if GPUs can be used to accelerate the performance of MAM, a module of E3SM on Summit using OpenACC.
- We have achieved over a 5X performance speedup by offloading some of the kernels to Nvidia Volta GPUs.
- The results revealed that under the current E3SM configuration for product runs, some parameter settings do not suit offloading, such as the number of columns per block and the number of vertical levels.
    - These settings not only severely limit the degree of parallelism but also fail to make effective use of GPU thread resources, becoming a performance bottleneck.

Summary and Conclusion (cont.)

- Run time scatters across many kernels, and each computational kernel is relatively light, with average run time in milliseconds or less per call. Such light computational kernels particularly require OpenACC implementation to further reduce overhead from kernel launching, data transfer, etc.
- Performance was primarily limited by the kernel `nsubmix`, which did not benefit from GPU offloading.
- We are looking into overlapping computations and data transfer using async OpenACC directives and possibly merging kernel computations to improve MAM's performance.

Acknowledgement

- All authors from Lawrence Berkeley National Laboratory were supported by the Office of Advanced Scientific Computing Research in the Department of Energy Office of Science under contract number DE-AC02-05CH11231.
- This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.


Thank You!
NERSC