GPU Acceleration of MPAS Physics Schemes Using OpenACC

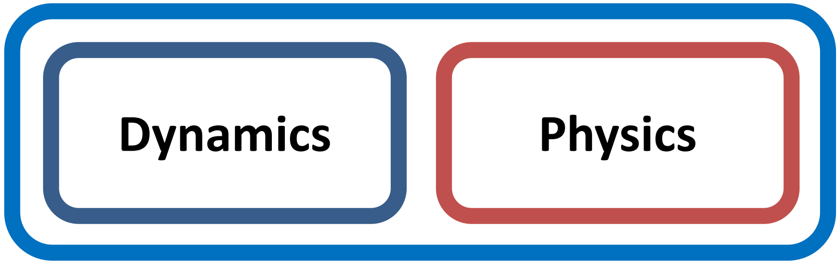
Jae Youp Kim[1,2], Ji-Sun Kang[1], and Minsu Joh[1,2]

[1]Disaster Management HPC Technology Research Center, KISTI, Korea
[2]University of Science and Technology, Korea

Contents

Introduction

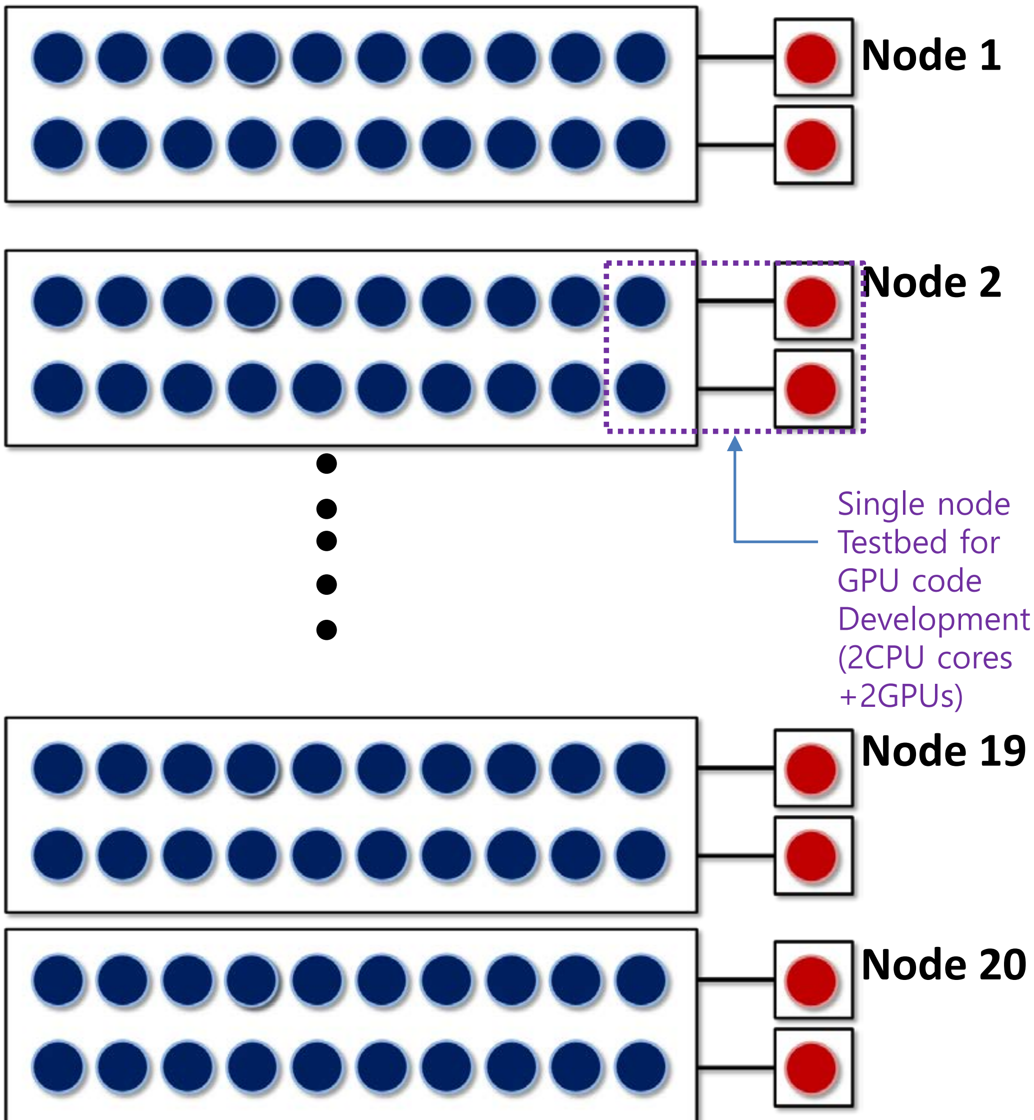
- KISTI has been collaborating on a development of MPAS with NCAR MMM since 2014.
- One of recent collaborative research topics is GPU acceleration of MPAS.
  - For the development of MPAS GPU code, we have also discussed with CISL since Dec. in 2015.

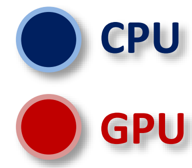


Dynamics: CISL
Physics (WRF): KISTI
Integration of Dynamics and Physics: CISL

- We have made progress in the GPU acceleration of physics schemes of MPAS.

KISTI's GPU systems





System Spec

CPU : Haswell Intel(R) Xeon(R) CPU E5-2660 v3 @ 2.60GHz
# of CPU core : 10 cores, dual-socket
CPU Memory : 125GB

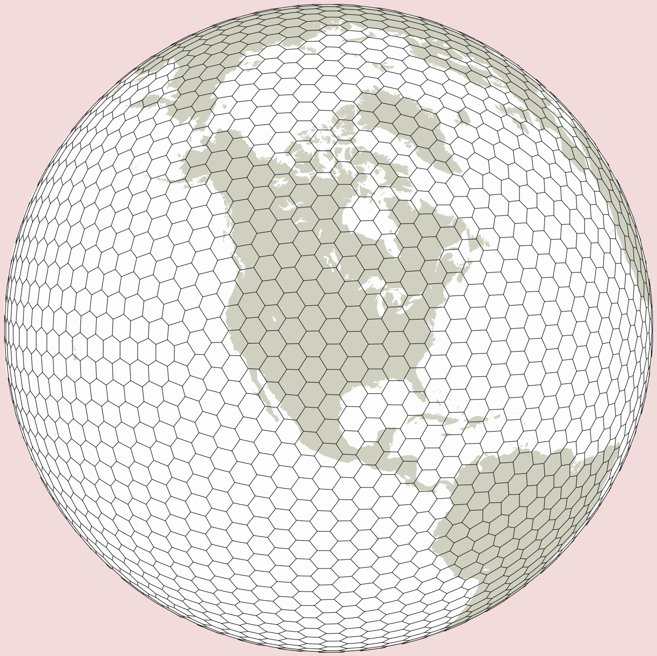
GPU : Tesla K40m
# of GPU core : 2880 CUDA cores, 15 SMs
GPU Memory : 12GB

Total

of CPU cores : 10 * 2 * 20 = 400
# of GPUs : 2 * 20 = 40

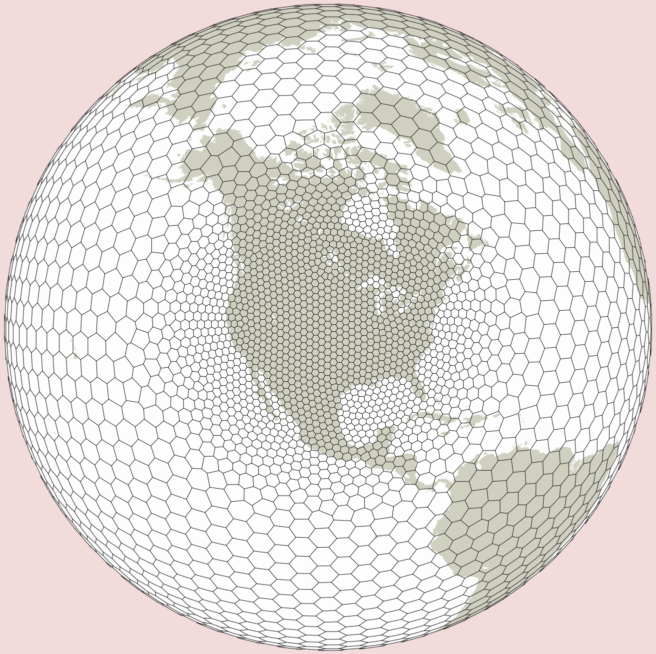
PGI-16.3

MPAS



MPAS
Unstructured Voronoi (hexagonal) grid

- Good scaling on massively parallel computers
- No pole problems

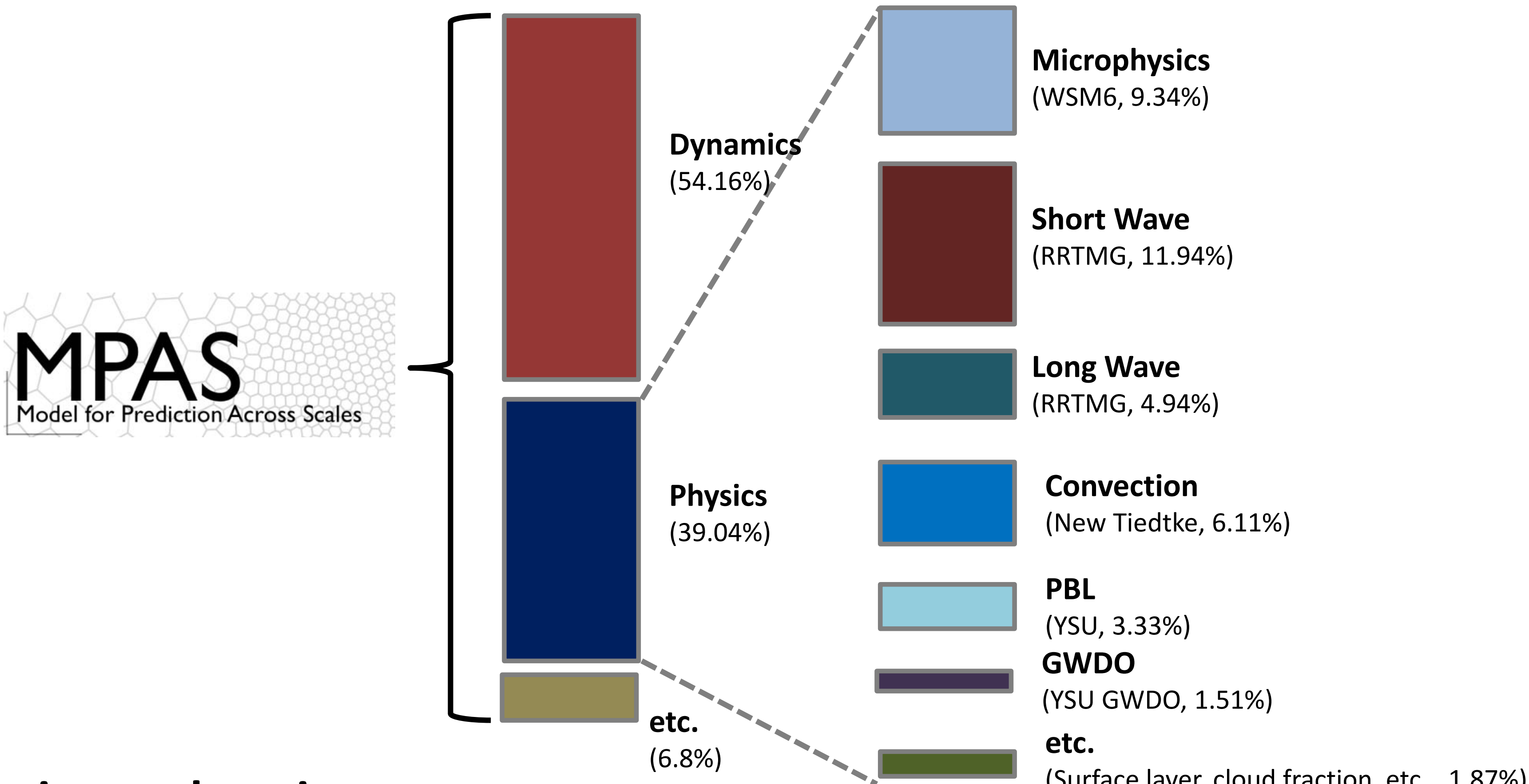


MPAS
Smooth grid refinement on a conformal mesh

- Increased accuracy and flexibility for variable resolution applications
- No abrupt mesh transitions.

Adopted from MPAS tutorial

Profiling computing time of MPAS

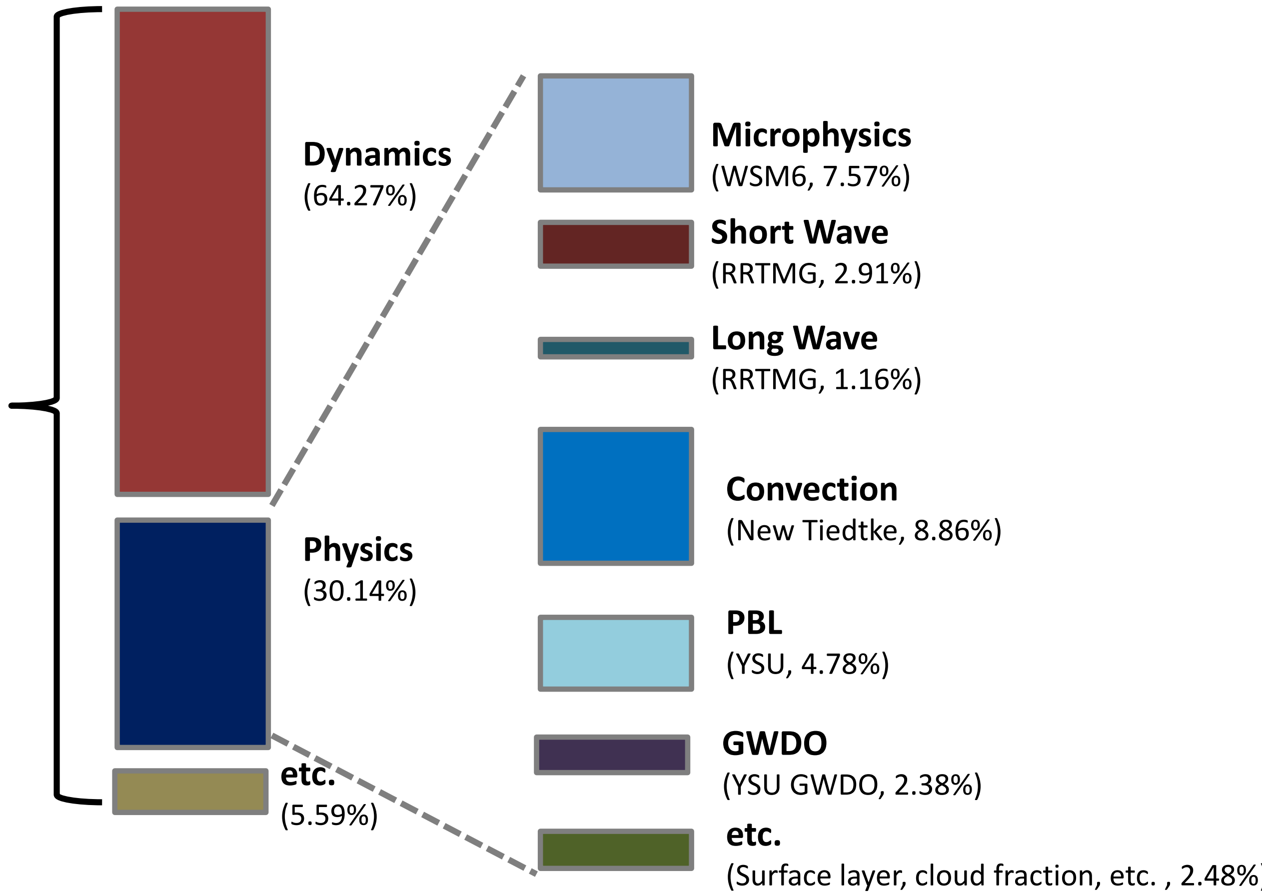


❑ Experimental settings

- **Quasi-uniform 60-km resolution** (163,842 cells)
- Δt=180 sec
- 41 vertical layers
- Δt of radiation scheme=30 min

Profiling computing time of MPAS

MPAS
Model for Prediction Across Scales



Experimental settings

- **60-15 km variable resolution** (535,554 cells)
- Δt=30 sec
- 41 vertical layers
- Δt of radiation scheme=30 min

MPAS physics

Surface Layer: Monin-Obukhov, MYNN

PBL: YSU, MYNN

Land Surface Model: Noah LSM

Gravity Wave Drag: YSU GWDO

Convection: Kain-Fritsch, Tiedtke, **New Tiedtke**, Grell-Freitas

Microphysics: WSM6, Thompson, Kessler

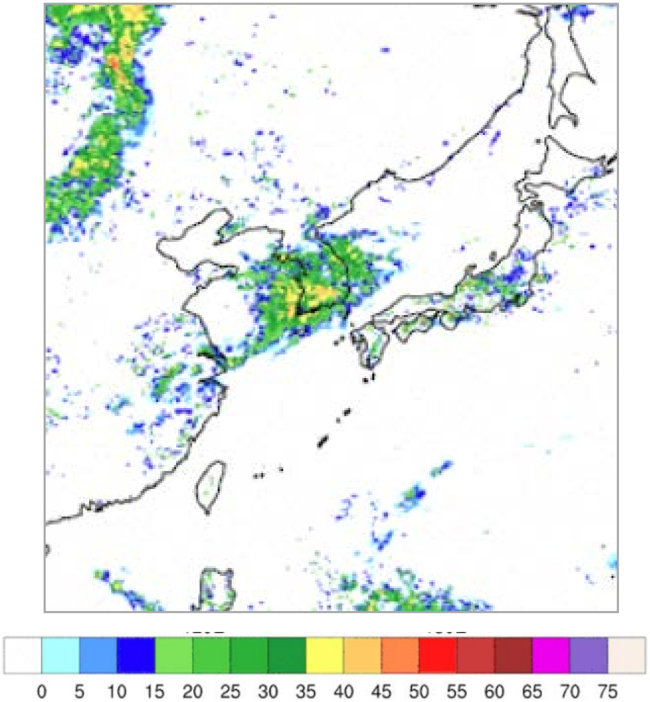
Radiation: RRTMG Short Wave, RRTMG Long Wave, CAM

… **etc.**(cloud fraction….)

RED : Ported on GPU

BLUE : Plan to port on GPU

10cm maximum Radar reflectivity



CUDA & OpenACC

CUDA

```
allocate(qv2d_d(its:ite,kts:kte*ndiff))
 . . . .
qv3d_d = qv3d
 . . . .
blocksize=dim3(128,1,1)
gridsize=dim3(ceiling(real(ite)/real(blocksize%x)),1,1)

call ysu_gpu_1<<<gridsize,blocksize>>>(kzhout_d, kzmout_d, kzqout_d, &
                                       qv2d_d, its, ite, jts, jte, kts, kte)

Kzhout = kzhout_d
 . . . .
```

◀ **allocate memory on GPU**

◀ **Memcpy CPU to GPU**

◀ **Set block and grid size**

▲ **call GPU kernel function**

▲ **Memcpy GPU to CPU**

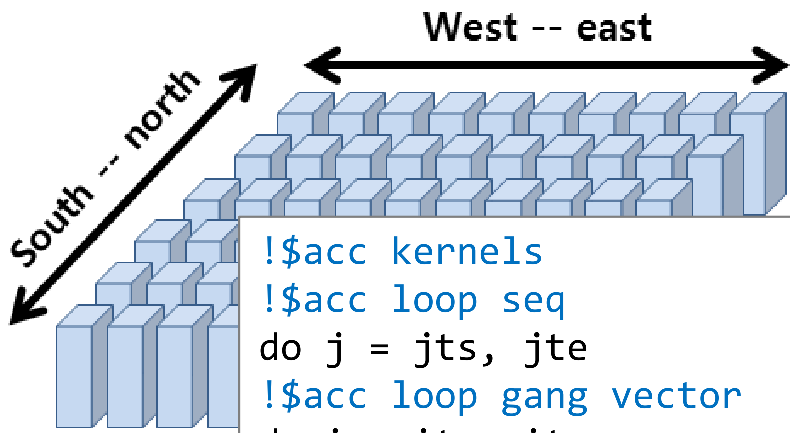
OpenACC

```
!$acc kernels
   do k = kts,kte
   do i = its,ite
      kzhout(i,k,j) = 0.
      kzmout(i,k,j) = 0.
      kzqout(i,k,j) = 0.
   enddo
   enddo
!$acc end kernels
```

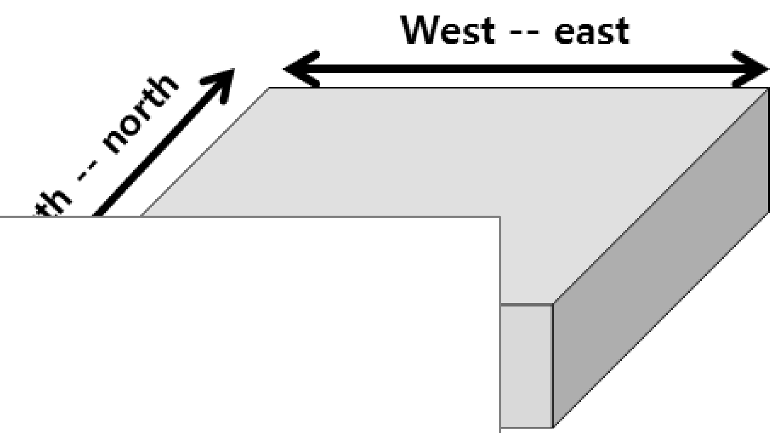
▲ ***OpenACC kernels directives automatically generate allocation function, memcpy function, optimized threads, GPU kernel function.***

Parallelization of MPAS physics on GPU

<CPU>



<GPU>




```
!$acc kernels
!$acc loop seq
do j = jts, jte
!$acc loop gang vector
do i = its, ite
do k = kts, kte
 . . . . . .
   a(k,i,j) = b(k,i,j) + c(k,i,j)
 . . . . . .
end do
end do
end do
!$acc end kernels
```

```
do j = jts,
do i = its,
do k = kts,
 . . . . . .
   a(k,i,j)
 . . . . . .
end do
end do
end do
```

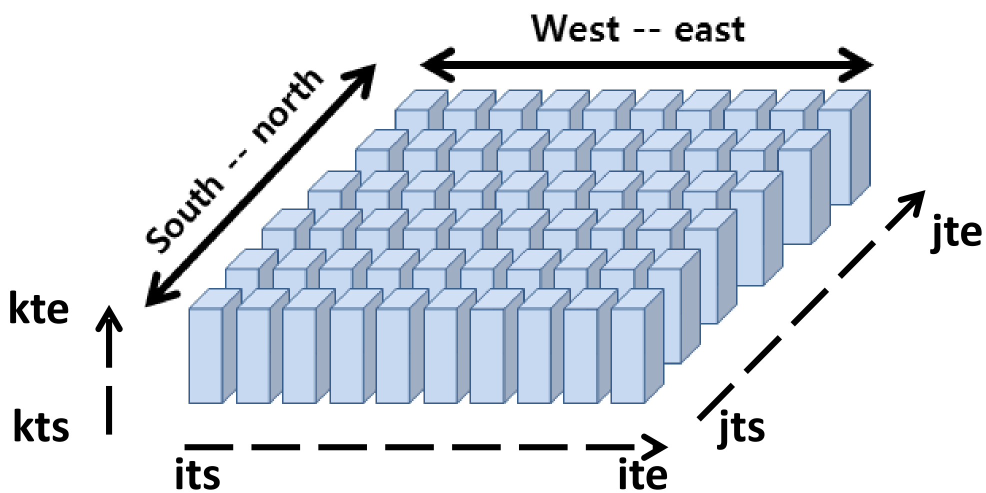
```
x+threadIdx%x
(k,i,j)
end do
end do
```

# of iteration = i*j*k

# of iteration = j*k

# of iteration = k(In MPAS, J loop is 1)

Difference between WRF and MPAS



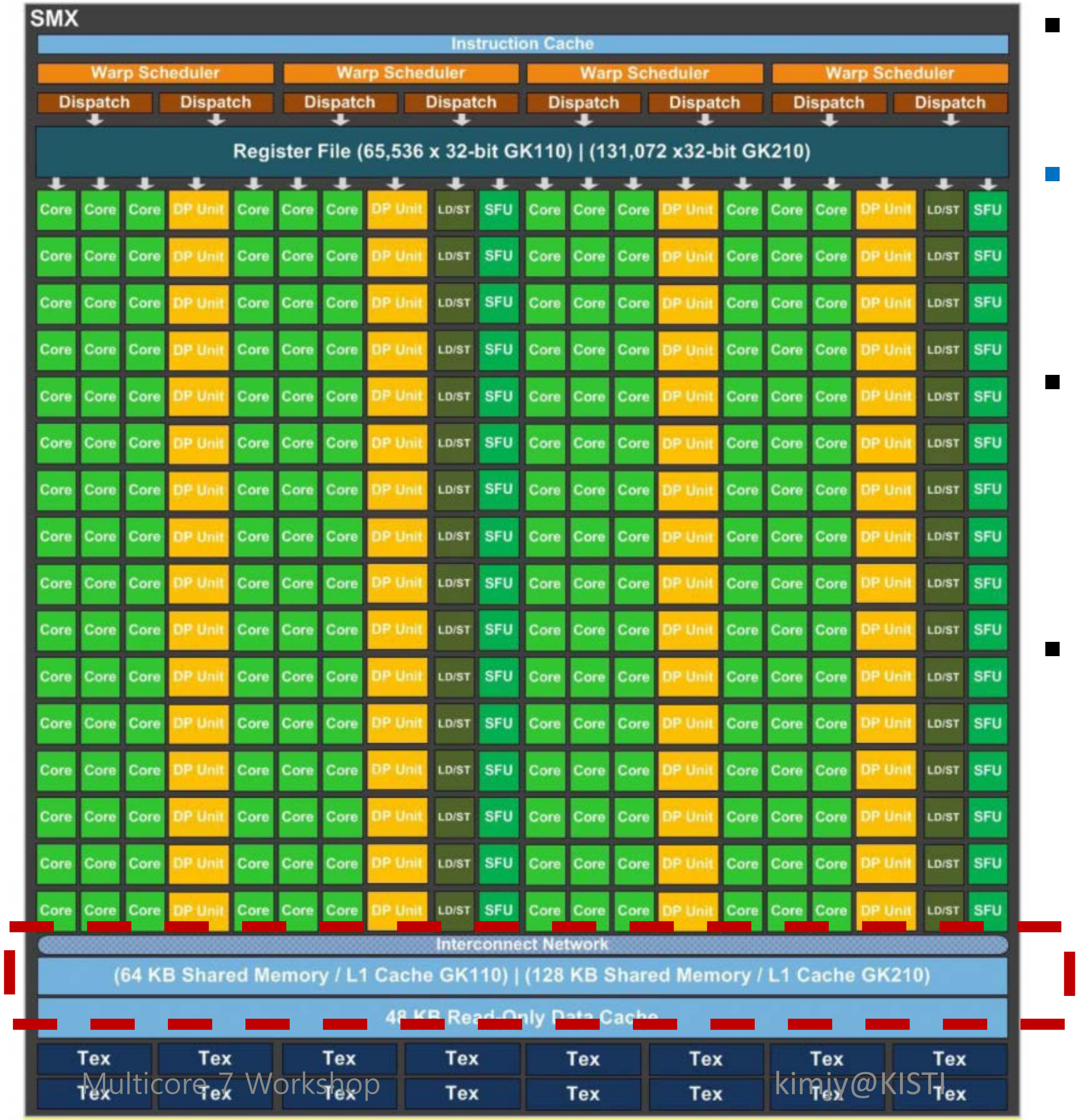
	WRF	MPAS
i loop	west to east	1 to nCells
j loop	South to north	~~1 to 1~~
k loop	1 to nlevels	1 to nlevels

```
DO j=jts,jte
  DO k=kts,kte
  DO i=its,ite
    t(i,k)=th(i,k,j)*pii(i,k,j)
    qci(i,k,1) = qc(i,k,j)
    qci(i,k,2) = qi(i,k,j)
    qrs(i,k,1) = qr(i,k,j)
    qrs(i,k,2) = qs(i,k,j)
    qrs(i,k,3) = qg(i,k,j)
  ENDDO
  ENDDO
. . . . .
  CALL wsm62D(t, q(ims,kms,j), qci, qrs    &
             ,den(ims,kms,j)            &

. . . . .
END DO
```

If we port WRF model on GPU, j loop should be put in subroutines for more efficient GPU parallelization.

However, j loop of MPAS model is always 1, so we did not modify subroutine's loop structure.

On-Chips memory for MPAS physics



From NVIDIA

- GPU has cache memory on their chips.
- **Shared memory and L1 cache memory shared on-chips memory.**
- GPU code developer can adjust how many shared memory allocate on on-chips memory.
- We have **not used shared memory** for parallelization because the number of variables in MPAS physics are too many to estimate when & how much shared memory needs and those variables are not usually reused.

OpenACC routine directives

```fortran
!$acc kernels
do i = its, ite

  . . . .
call slope_rain(qr,den,denfac,tk,&
    tmp,tmp1,tmp2,tmp3,wa,1,1,1,km)
  . . . .

end do
!$acc end kernels
```

```fortran
 subroutine slope_rain(qrs,den,denfac,&
t,rslope,rslopeb,rslope2,rslope3,vt, &
its,ite,kts,kte)
!$acc routine vector
 . . . .
do k = kts, kte
  if(qrs(i,k).le.qcrmin)then
    rslope(i,k) = rslopermax
    rslopeb(i,k) = rsloperbmax
    rslope2(i,k) = rsloper2max
    rslope3(i,k) = rsloper3max
  else
  . . . .
Enddo
  . . . .
 end subroutine
```

- OpenACC directives allow a kernel function to call other kernel functions using routine directives.
- Unfortunately, any functions cannot be called within a GPU kernel in MPAS model which has complex structure.

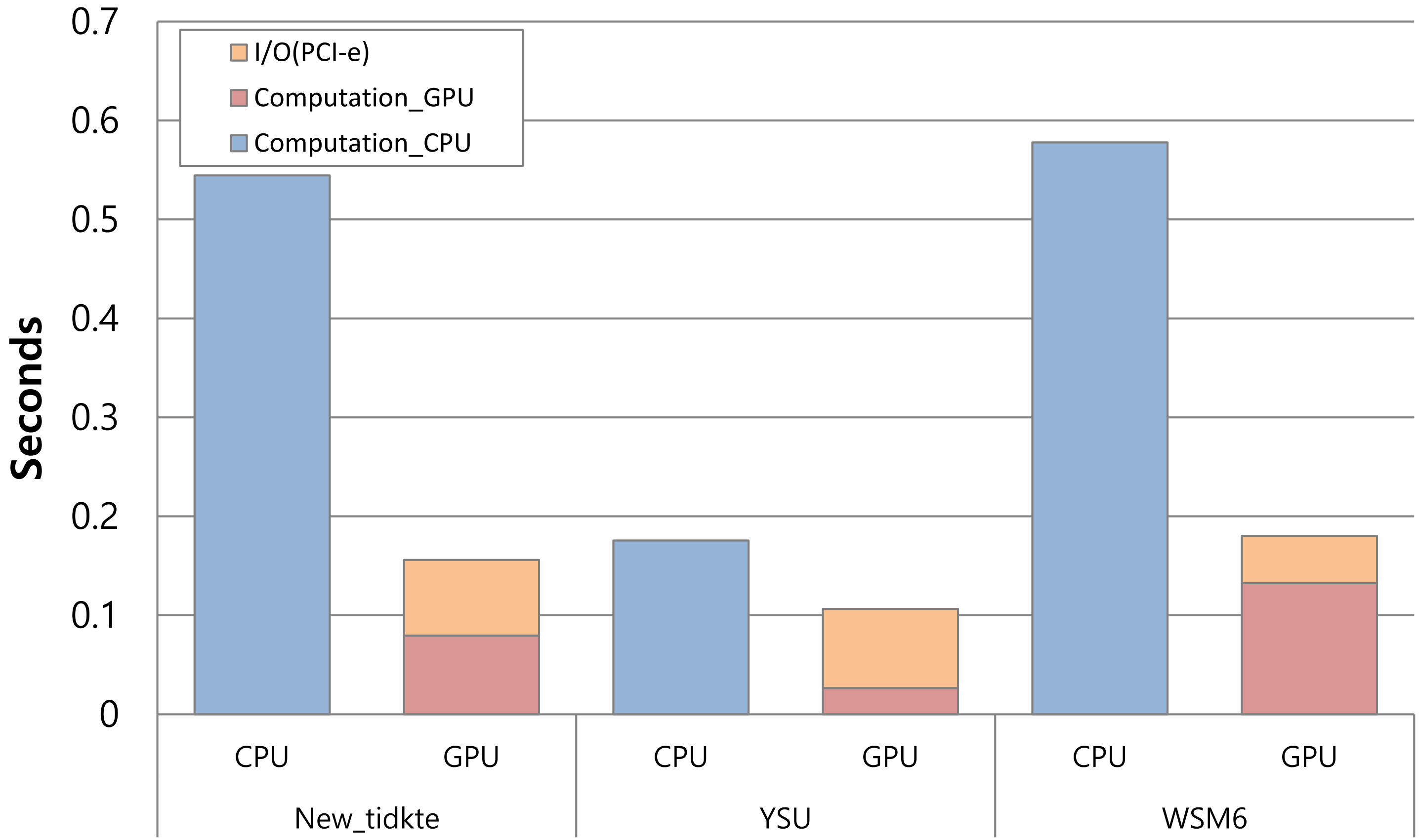
Subroutine inlining

```
!$acc kernels
do i = its, ite

  . . . .
call slope_rain(qr,den,denfac,tk,&
    tmp,tmp1,tmp2,tmp3,wa,1,1,1,km)
  . . . .

end do
!$acc end kernels
```

```
!$acc kernels
do i = its, ite

  . . . .
!call slope_rain(qr,den,denfac,tk,&
!    tmp,tmp1,tmp2,tmp3,wa,1,1,1,km)
!========================================
!    inlining of slope_rain subroutine
!========================================
 do k = 1, km
             if(qr(i,k).le.qcrmin)then
                 !tmp(i,k) = rslopermax
                 tmp1 = rsloperbmax
                 !tmp2 = rsloper2max
                 !tmp3 = rsloper3max
             else
  . . . .
 end do
!========================================
  . . . .
end do
!$acc end kernels
```

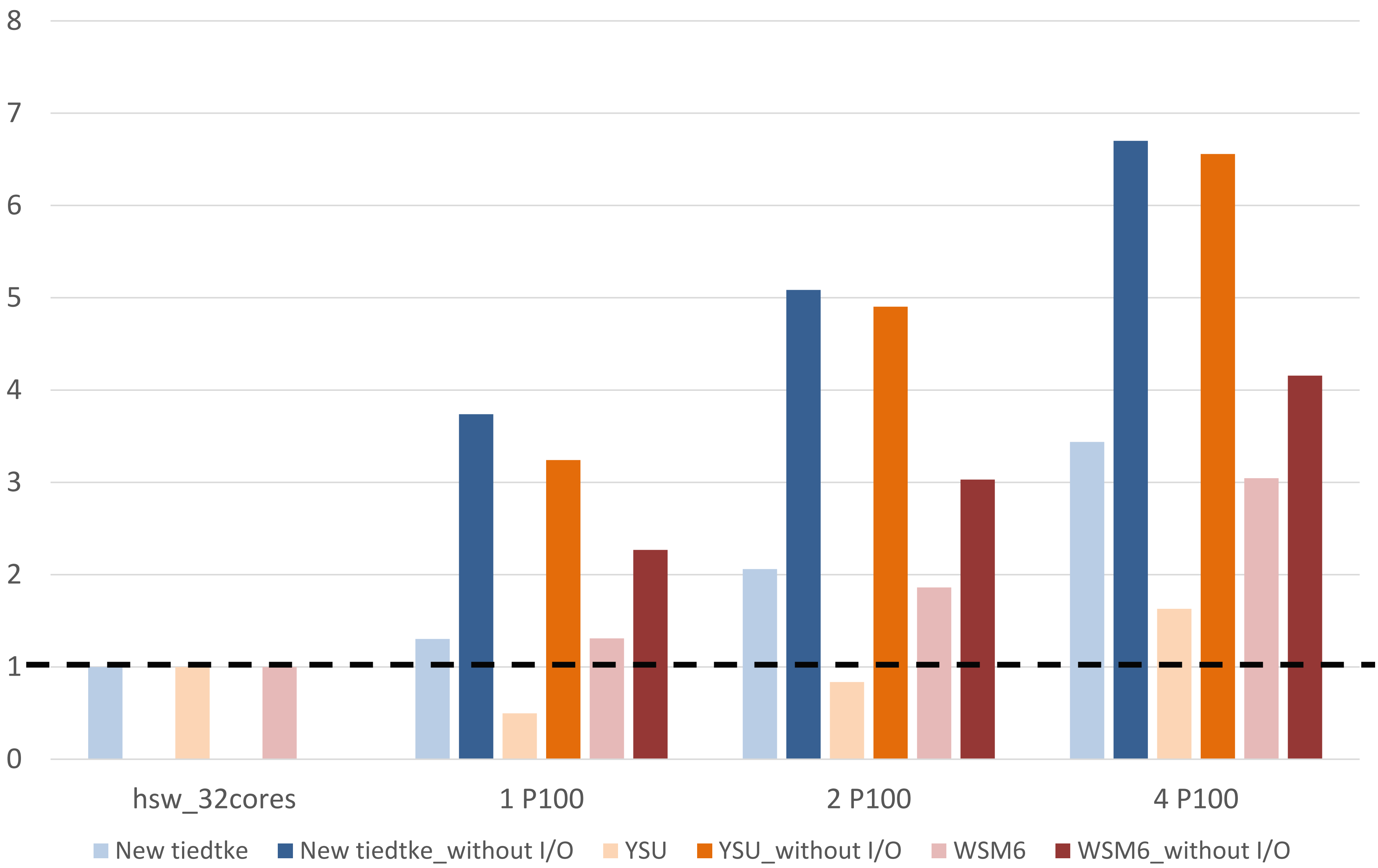
Performance of GPU acceleration - Result

(CPU 32 cores vs. CPU 4 cores + 4 GPUs)

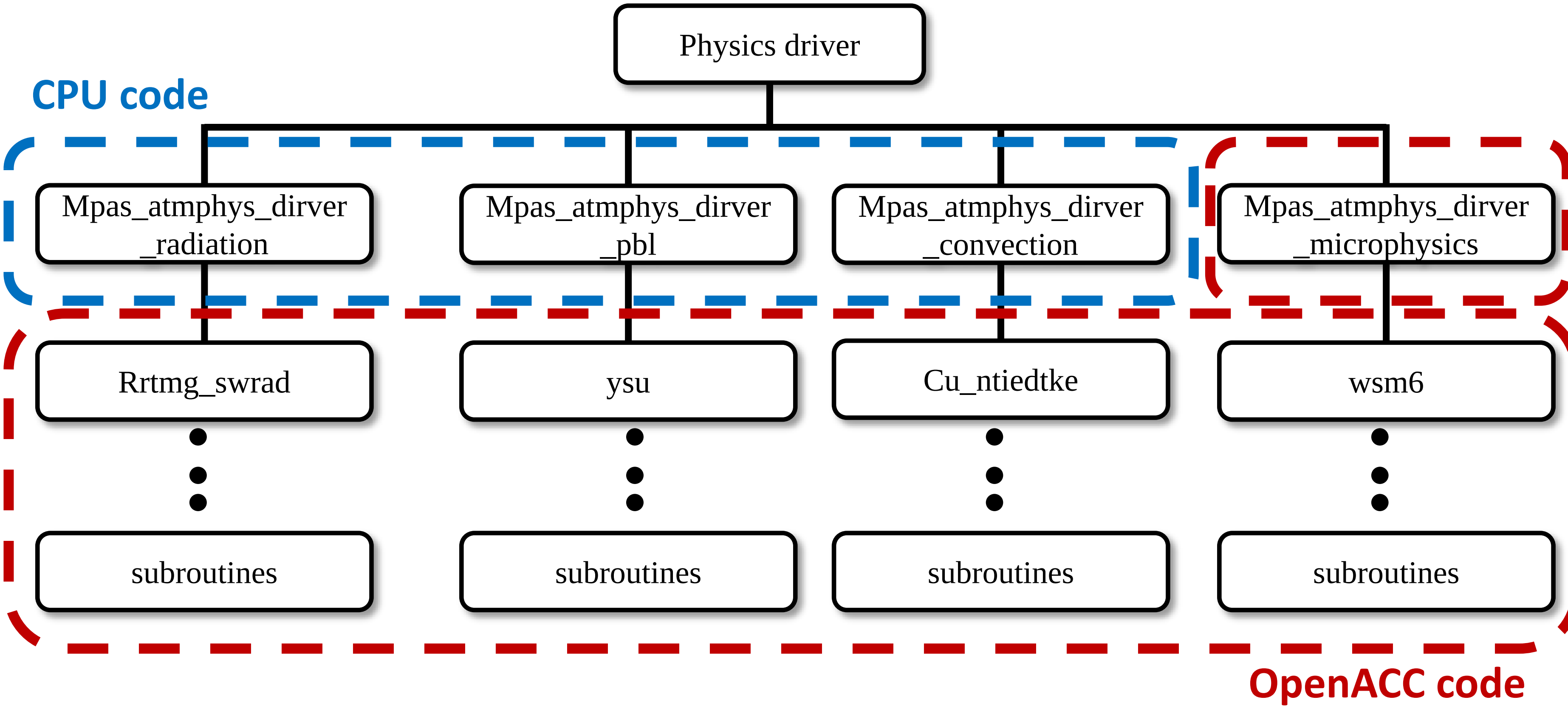


PGI-17.5, 60km resolution(163,842 cells), dt=180s, 1 day forecast
Haswell E5-2698 v3 @ 2.30GHz, dual socket 16-core
NVIDIA Tesla P100

Speed-up factor of MPAS physics



Future work



- MPAS physics schemes are linked on MPAS model through MPAS physics drivers.
- **We will port both of physics drivers and physics schemes on GPU.**

Future work

- We will port other physics schemes which are **RRTMG (Short Wave/Long wave) radiation and YSU GWDO schemes on GPU.**
- **Verification is also very important issue for community to accept our new code (not producing spurious bias in the simulation)**, so we will carefully verify our codes using the verification method as we presented at MultiCore 6 Workshop.

Summary

- **We succeeded in porting WSM6, New Tiedtke, YSU PBL,** and the performance looks very encouraging.
- **Shared memory was not used for GPU parallelization of MPAS physics** due to MPAS physics variables that are not predictable for using shared memory.
- **OpenACC routine directives are not working on MPAS model**, so we have applied subroutine inlining for efficient parallelization.

Thank You!

Please e-mail me If you have question.
kimjy10@kisti.re.kr

Korea Institute of Science and Technology Information
www.kisti.re.kr

UNIVERSITY OF SCIENCE & TECHNOLOGY