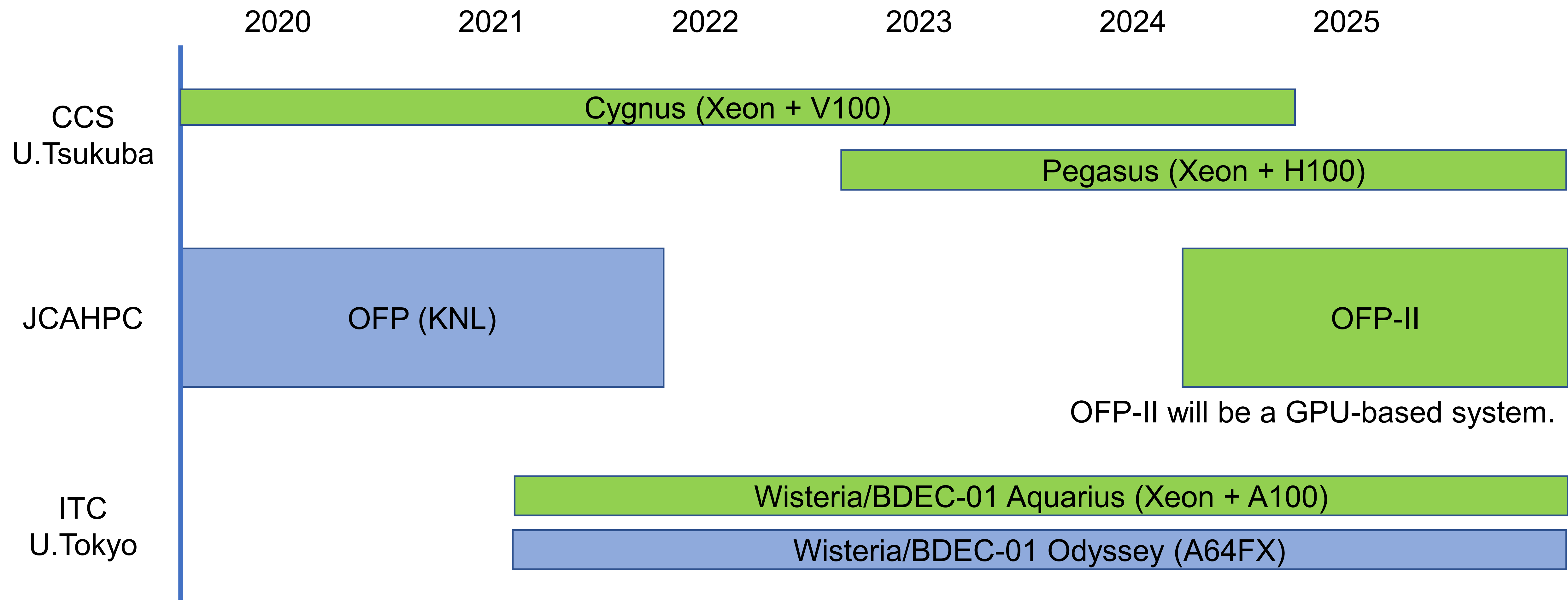
Easy software porting to GPU systems : experience with OpenSWPC

Akira Nukada, CCS, University of Tsukuba

System Roadmap



Activities with U.Tokyo & NVIDIA

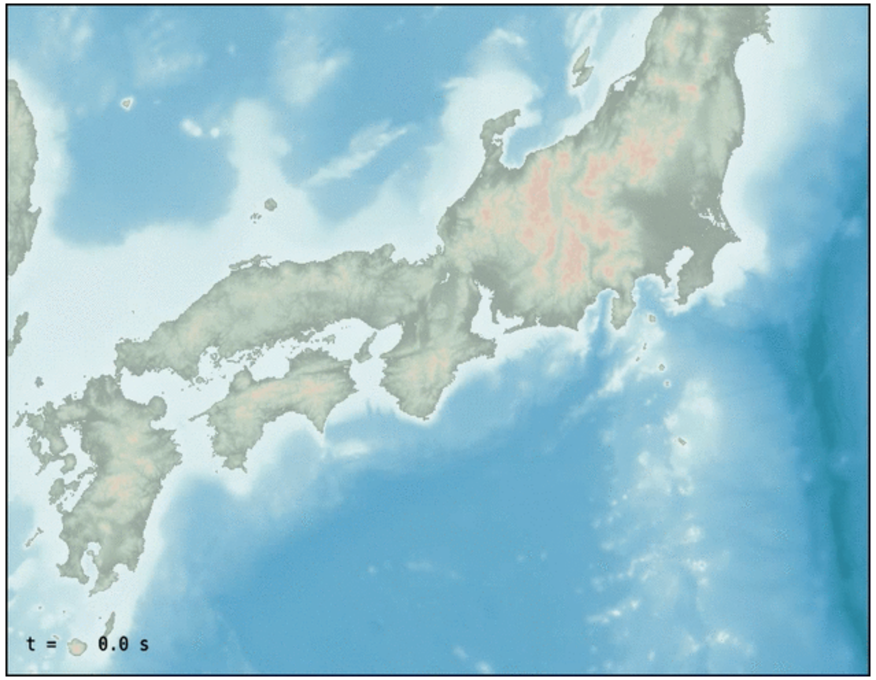
- GPU seminars
- GPU minicamps
- Consultation service for GPU programming
- GPU code porting service

Target application

OpenSWPC (Seismic Wave Propagation Code)
developed by Prof. Furumura's team at U.Tokyo.

The original CPU code is parallelized with
MPI + OpenMP, and vectorized for SIMD.

3-D stencil-based computation,
which updates velocity and stress.



GPU programming

Many HPC applications are still implemented in Fortran.
Cost and Performance depends on programming method.

	Performance	Difficulty
CUDA	Best, depending on skill	Hard. Need to rewrite kernel code.
OpenACC	Near best, depending on calculation	Easy. May mistake.
OpenMP	Near best, depending on calculation	Easy. May mistake.
stdpar do concurrent	Near best, depending on calculation	Easy. No chance to mistake.

Offloading loops for GPU

STDPAR (do concurrent)

```
do concurrent (i=1:n)
   v(i)=v(i)+w(i)
end do
```

OpenMP target

```
!$omp target loop
do i=1,n
   v(i)=v(i)+w(i)
end do
```

OpenACC

```
!$acc parallel loop
do i=1,n
   v(i)=v(i)+w(i)
end do
```

“!$omp target teams distribute” is not recommended.
It specifies # of thread blocks to # of SMs of GPU, and # of threads per block to 128.

Subroutine calls in offloaded loops

- do concurrent

  Fortran 2008 supports calls of pure functions.

  nvfortran doesn’t support any calls.
- OpenMP target

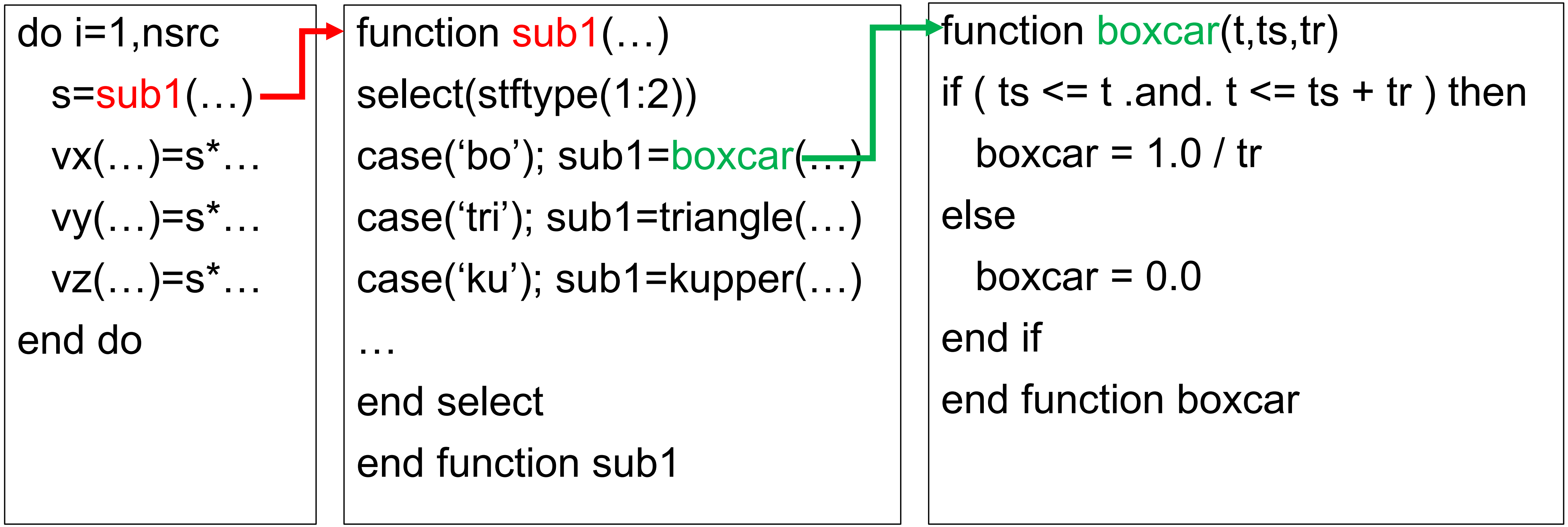
  Supports subroutine calls using target declare directives.

  Not works with nvfortran… compiler internal errors.
- OpenACC

  Supports subroutine calls using routine directives.

  Some cases degrade memory access performance.

Subroutine call (1)



```
do i=1,nsrc
   s=sub1(…)
   vx(…)=s*…
   vy(…)=s*…
   vz(…)=s*…
end do
```

```
function sub1(…)
select(stftype(1:2))
case('bo'); sub1=boxcar(…)
case('tri'); sub1=triangle(…)
case('ku'); sub1=kupper(…)
…
end select
end function sub1
```

```
function boxcar(t,ts,tr)
if ( ts <= t .and. t <= ts + tr ) then
   boxcar = 1.0 / tr
else
   boxcar = 0.0
end if
end function boxcar
```

This part updates velocity or stress of sources.
2-level nested calls of simple functions. Each function will be executed by one thread.

Subroutine call (2)

```
 do jj = js0, js1
    do kk= ks0, ks1
       k = kk * kdec - kdec/2
       j = jj * jdec - jdec/2
       call divrot(k,i,j,div,rot)
       buf(jj,kk,1,1)=div*UC*M0*1e-3
       buf(jj,kk,2,1)=rot(1)*UC*M0*1e-3
       buf(jj,kk,3,1)=rot(2)*UC*M0*1e-3
       buf(jj,kk,4,1)=rot(3)*UC*M0*1e-3
    end do
end do
```

```
subroutine divrot(k,i,j,div,rot)
dxVx=(Vx(k  ,i    ,j  )-Vx(k  ,i-1,j  ))*r20x
dxVy=(Vy(k  ,i+1,j  )-Vy(k  ,i    ,j  ))*r20x
dxVz=(Vz(k  ,i+1,j  )-Vz(k  ,i    ,j  ))*r20x


….
end subroutine divrot
```

module variables are accessed in the subroutine.

Subroutine call (2) : Solutions

- OpenACC

  It works by adding “!$acc declare create()” to variables.

  This also changes how the other kernels access the variables
- OpenMP

  It should work with “!$omp declare target()”, but doen’t work.

  Now we need inlining of the subroutine.
- do concurrent

  We need inlining of the subroutine.

Fortunately, this inlining is easy using preprocessor macros.

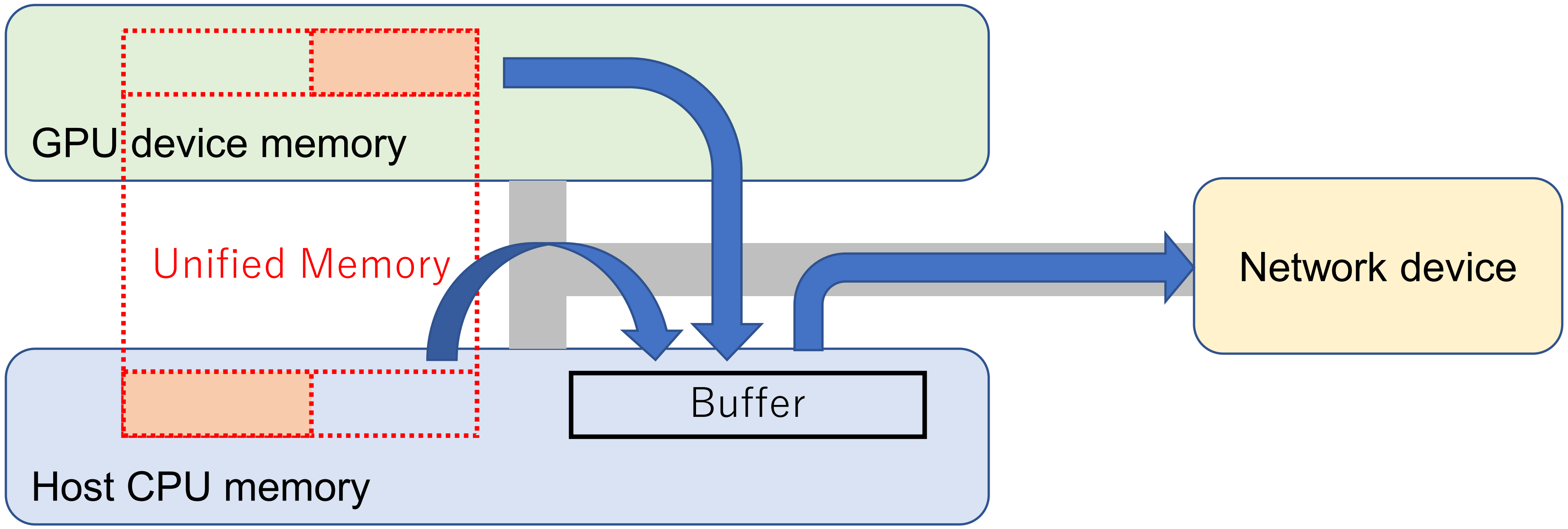
Variables used in kernel

	do concurrent
Scalars real :: x, y, z integer :: i, j, k	Stored in host memory Passed as parameters
Fixed-length array real :: c(4)	Stored in host memory Copied before/after the kernel
Allocatable real, allocatable :: vx(:)	Stored in unified memory No copy required. Addresses are passed as parameters

MPI data transfers

MPI with GPUDirect RDMA

- works well for host and device memory (OpenACC/OpenMP)
- works but too slow for unified memory (do concurrent)



Workaround in do concurrent

Explicitly allocate device memory (or pinned host memory).

Add device attribute（CUDA Fortran）

```
real,device,allocatable :: sbuf(:)

!$acc data deviceptr(sbuf)
do concurrent (k=kbeg:kend)
   sbuf(k) = vx(k)
end do
!$acc end data

call MPI_Isend(sbuf,…)
```

Specify sbuf is device-side address（OpenACC）

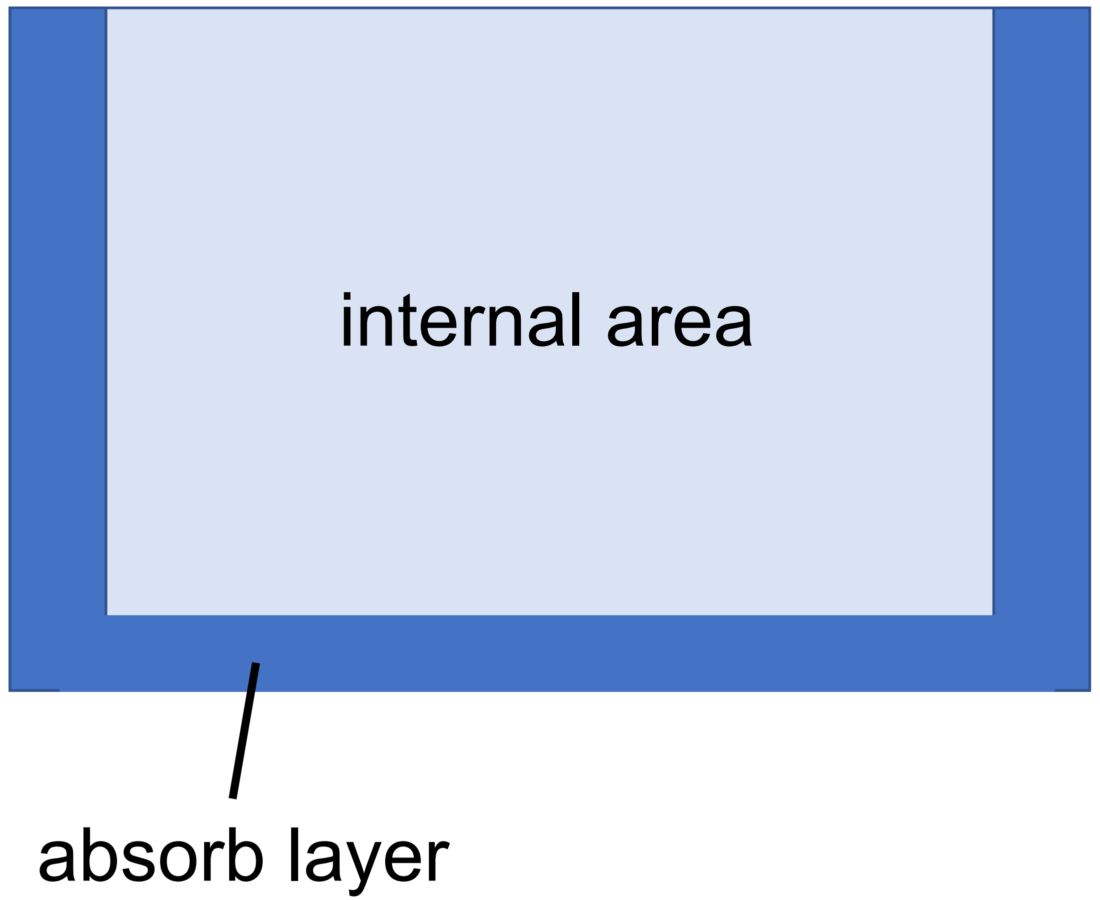
Packing data into send buffer

MPI function handles device address

3-D Stencil

Non-uniform

- different kernel for internal area and absorb layer
- different computation around free surface and sea floor (internal area)



Kernel for absorb layer

# of iteration of inner loop depends on outer loop

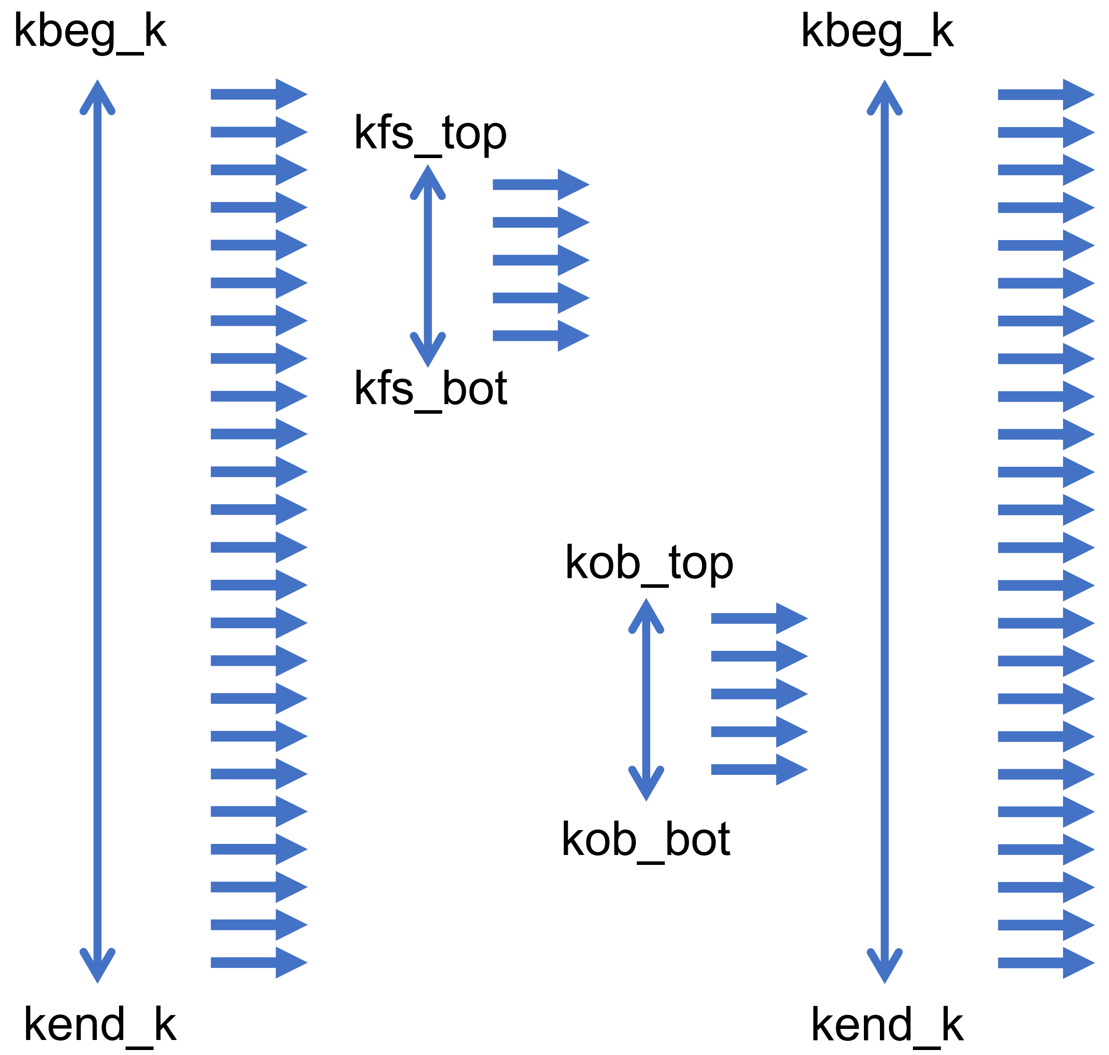
```
do j=jbeg, jend
    do k=kbeg_a(1,j), kend
        Vx(k,0,j) = 2* Vx(k,1,j)-Vx(k,2,j)
    end do
end do
```

Use if statement and fixed iteration count

```
do concurrent(j=jbeg: jend, k=kbeg: kend)
    if (k >= kbeg_a(1,j)) then
        Vx(k,0,j) = 2* Vx(k,1,j)-Vx(k,2,j)
    end if
end do
```

Kernel for internal area

```
do i=ibeg, iend
  do j=jbeg, jend
    do k=kbeg_k, kend_k
      d3Sx3(k) = …
    end do
    do k=kfs_top(i,j),kfs_bot(i,j)
      d3Sx3(k) = …
    end do
    do k=kob_top(i,j), kob_bot(i,j)
      d3Sx3(k) = …
    end do
    do k=kbeg_k, kend_k
      Vx(k,i,j) = …  d3Sx3(k) …
    end do
  end do
end do
```



Kernel for internal area

```
do i=ibeg, iend
  do j=jbeg, jend
    do k=kbeg_k, kend_k
      d3Sx3(k) = …
    end do
    do k=kfs_top(i,j),kfs_bot(i,j)
      d3Sx3(k) = …
    end do
    do k=kob_top(i,j), kob_bot(i,j)
      d3Sx3(k) = …
    end do
    do k=kbeg_k, kend_k
      Vx(k,i,j) = …  d3Sx3(k) …
    end do
  end do
end do
```

```
do concurrent (i=ibeg:iend, j=jbeg:jend,
                k=kbeg_k:kend_k) local(d3Sx3)
    d3Sx3 = …
    if (k>=kfs_top(i,j) .and. k<=kfs_bot(i,j)) then
       d3Sx3 = …
    end if
    if (k>=kob_top(i,j) .and. k<=kob_bot(i,j)) then
       d3Sx3 = …
    end if
    Vx(k,i,j) = …  d3Sx3 …
end do
```

Computer System : Pegasus

Compute node (120 nodes in total)

CPU	Intel Xeon Platinum 8468 (Sapphire Rapids)
Memory	DDR5-4800 128GB + Optane DC 2TB
GPU	NVIDIA H100 PCI-E 80GB
Network	NVIDIA Quantum-2 InfiniBand (NDR200)

OS	Ubuntu 20.04.06 LTS
SDK	NVIDIA HPC SDK 22.11 (nvfortran)
MPI	OpenMPI 4.1.5

STD (do concurrent)	-stdpar=gpu –gpu=cc90 –cuda –acc=gpu -mp
OpenMP	-mp=gpu –gpu=cc90
OpenACC	-acc=gpu –gpu=cc90 -mp

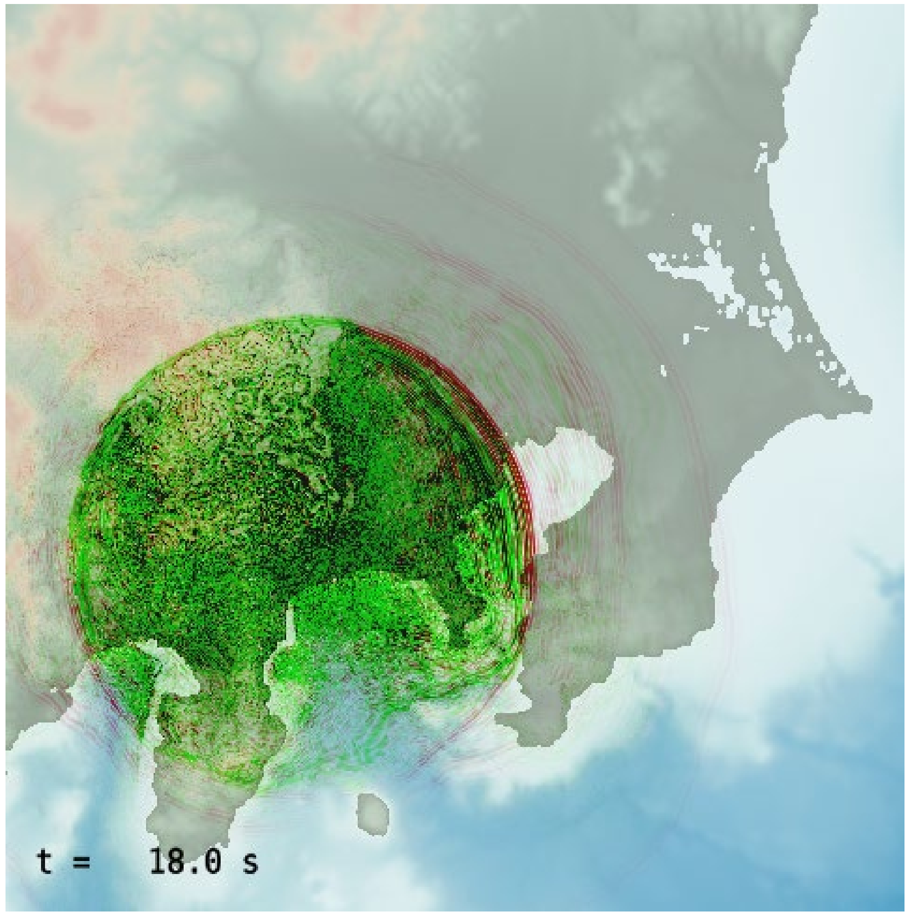
Input Data : Odawara earthquake

Grid: 1,024x1,024x400(0.25km)

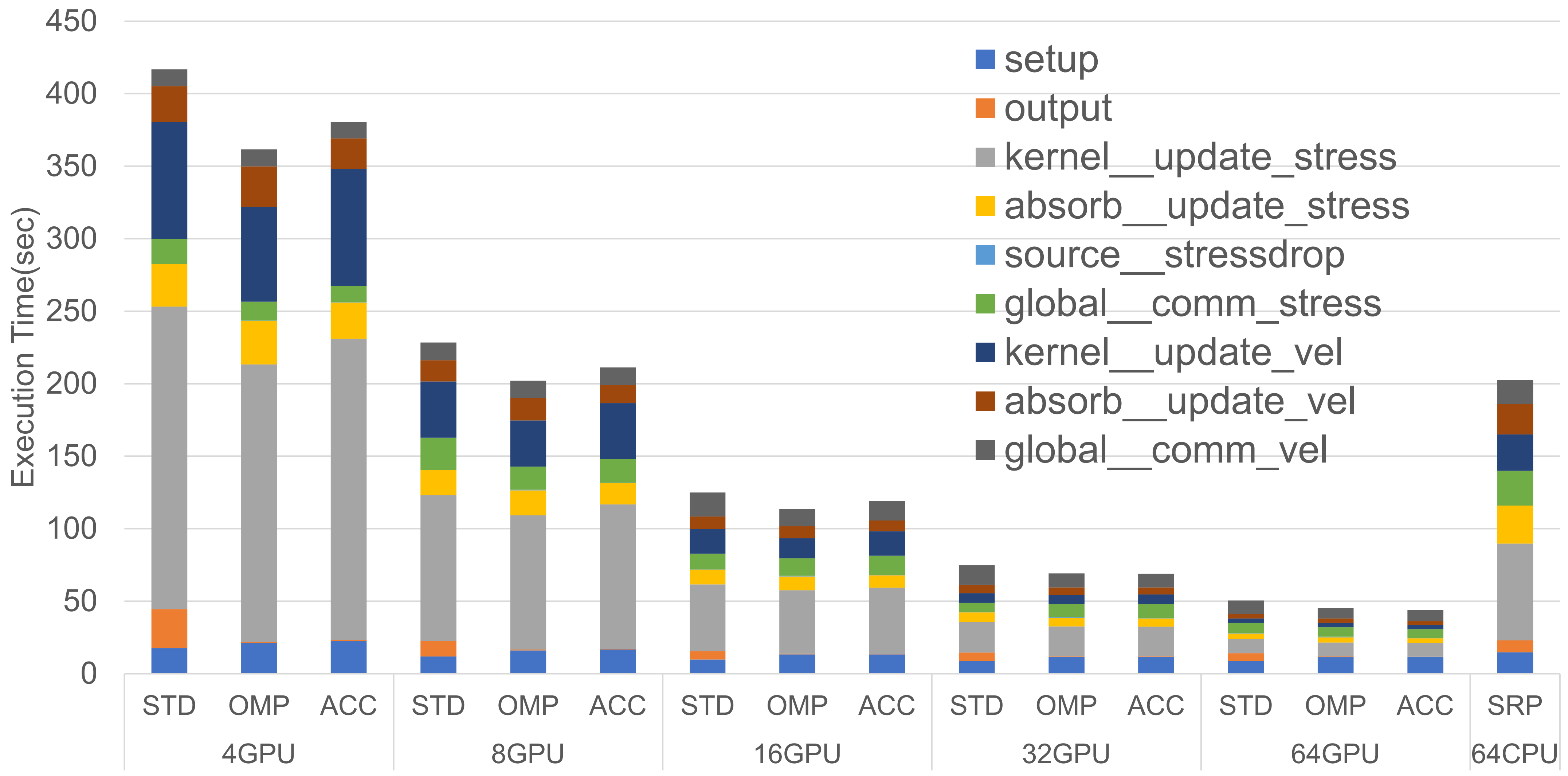
Steps: 8,000

Snapshot: every 240 steps

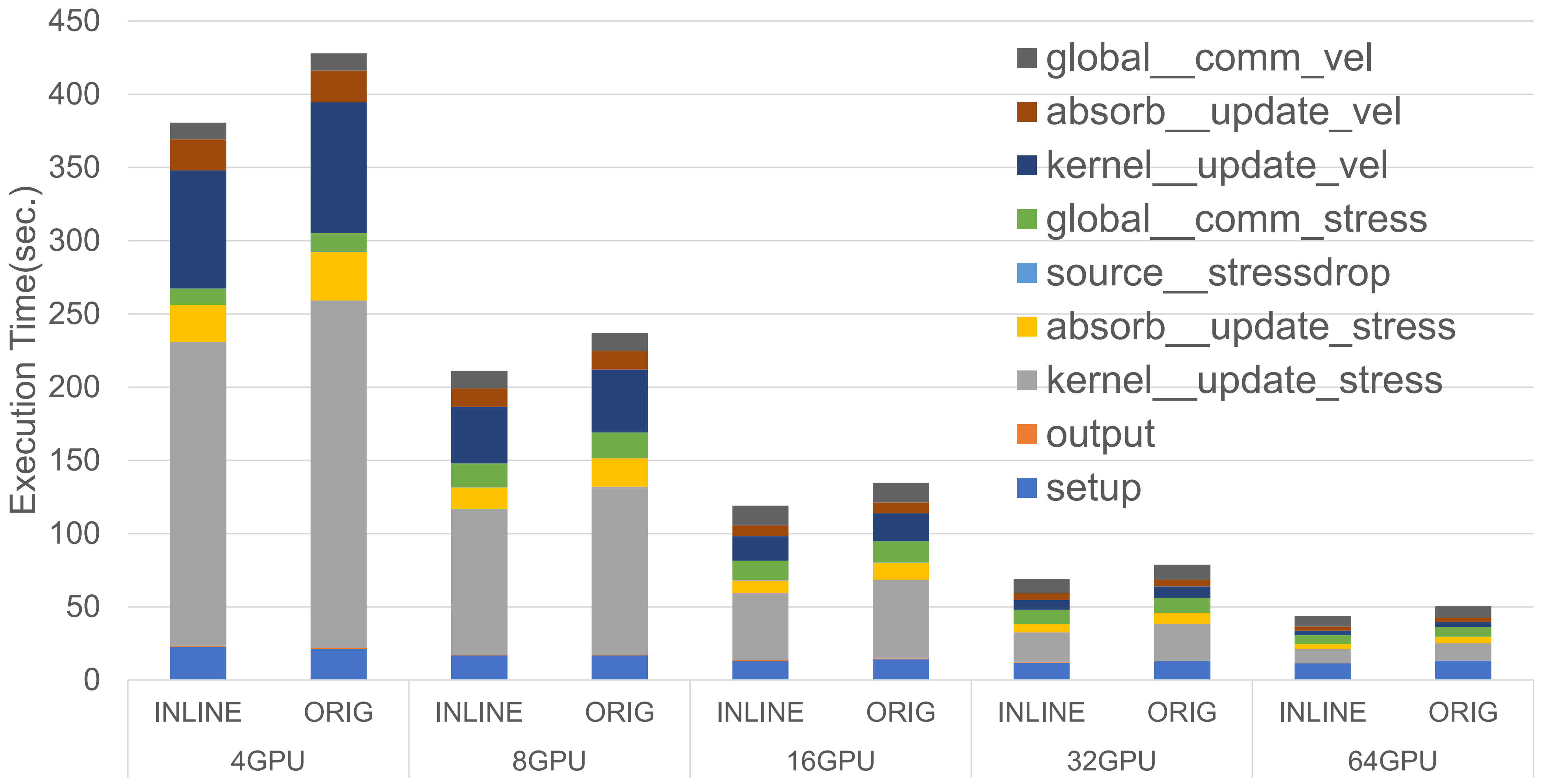
Absorb layer: 20 grids thickness



Performance results



OpenACC (inline vs. routine directives)



Summary

- Possible to port OpenSWPC to GPU using StdPar/OMP/ACC
- Performance and cost depends on the method
  - Some require inlining of subroutine calls

- Performance and compatibility depends on compiler implementation
  - Future version will have higher compatibility and less issues