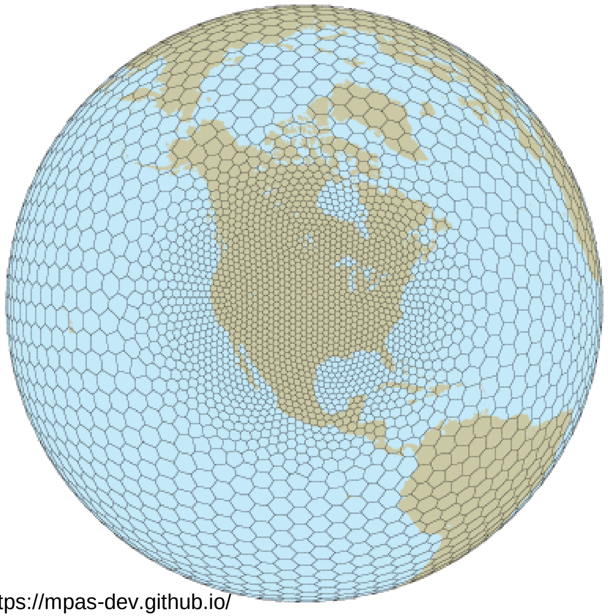
CWA- mesh generation for MPAS model

YING-JHANG WU, TING-AN CHEN, YU-HUNG LO,
Central Weather Administration
(*Technology Development & Numerical Information Divisions*)
Mentors: Leo Chen and Jay Chen

Mesh generation for MPAS model



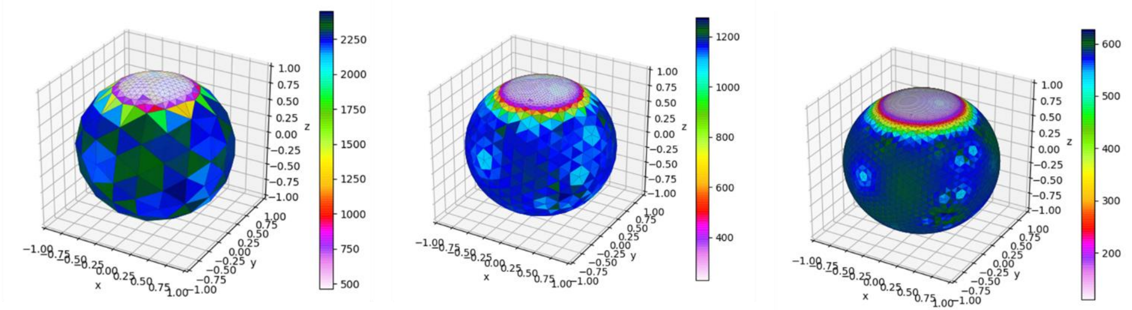
https://mpas-dev.github.io/

$$\rho(\mathbf{x}) = \frac{1-\gamma}{2}\left[\tanh\left(\frac{\beta - \|\mathbf{x}_c - \mathbf{x}\|}{\alpha}\right) + 1\right] + \gamma$$

$$\frac{h_i}{h_j} \approx \left(\frac{\rho(\mathbf{z}_j)}{\rho(\mathbf{z}_i)}\right)^{1/4}$$

Ringler, 2011

- MPAS mesh generation establishes the MPAS model's horizontal resolution
- Fortran Language
- LLOYD'S METHOD



Number of cells	N=255	N=1014	N=4050
Highest res.	480km	240km	120km
Wall time	2min	8min	16min

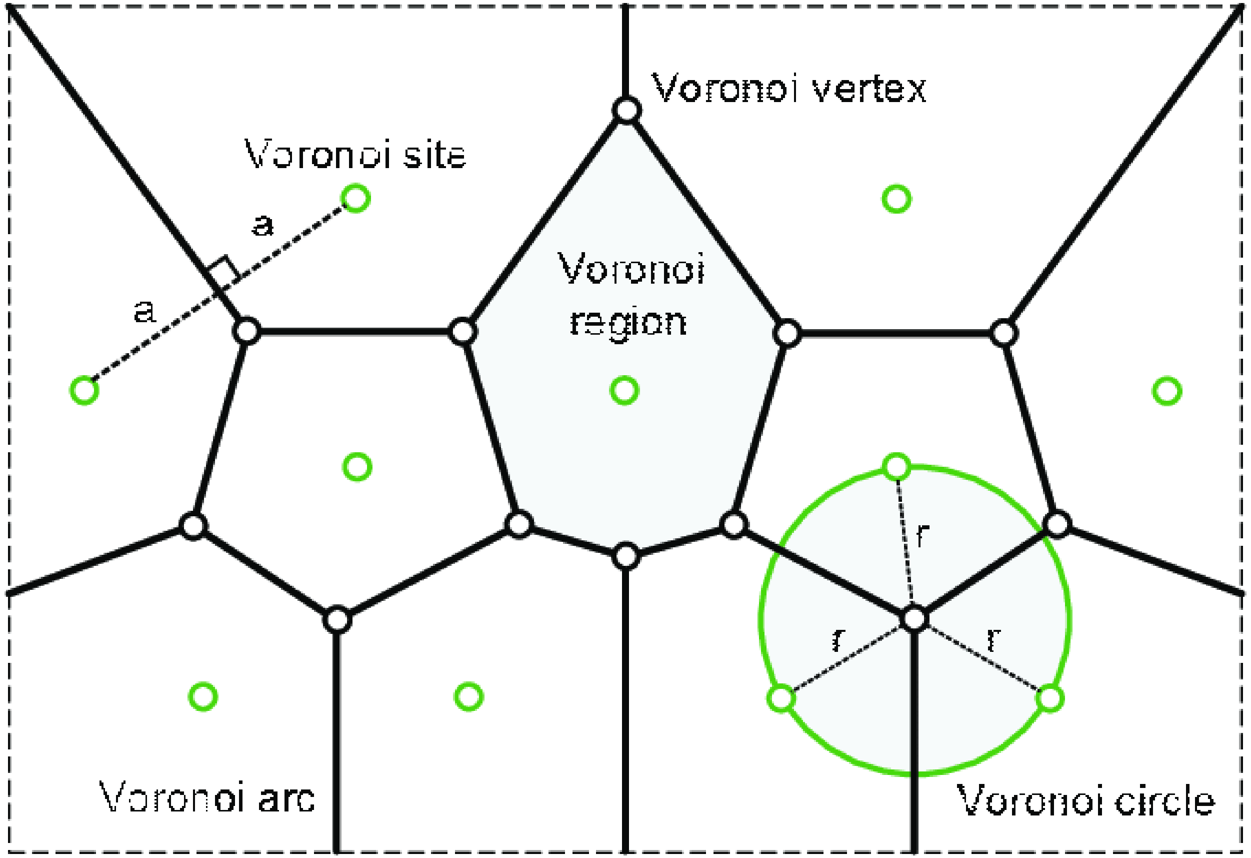
3.75km might takes 12 days or more

- Require several months to complete a single task

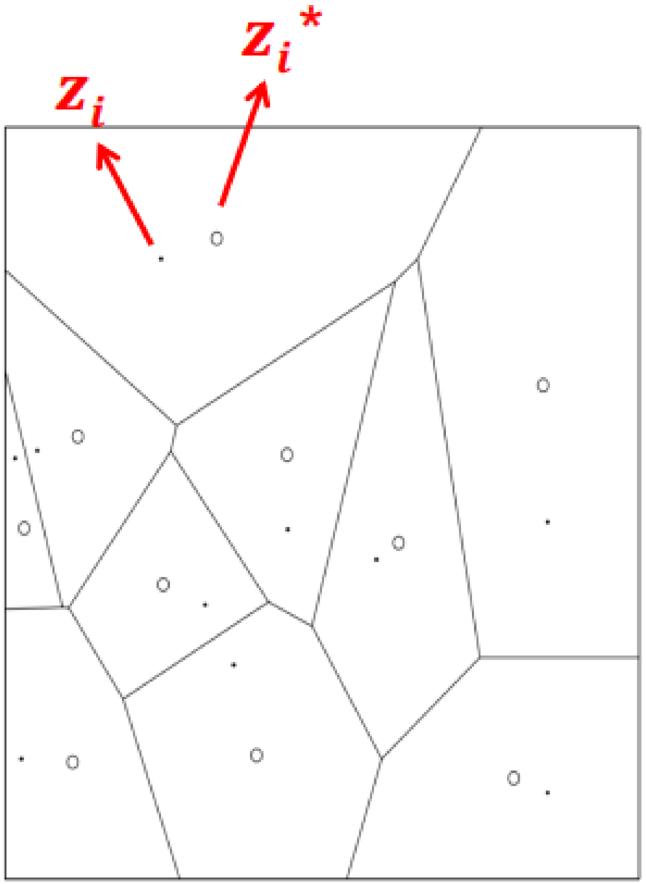
Voronoi Tessellation

1. 首先需要給定任意格點(綠點所示)
2. 找出相鄰三格點的外心(白色點)

各綠點彼此連線為中垂線

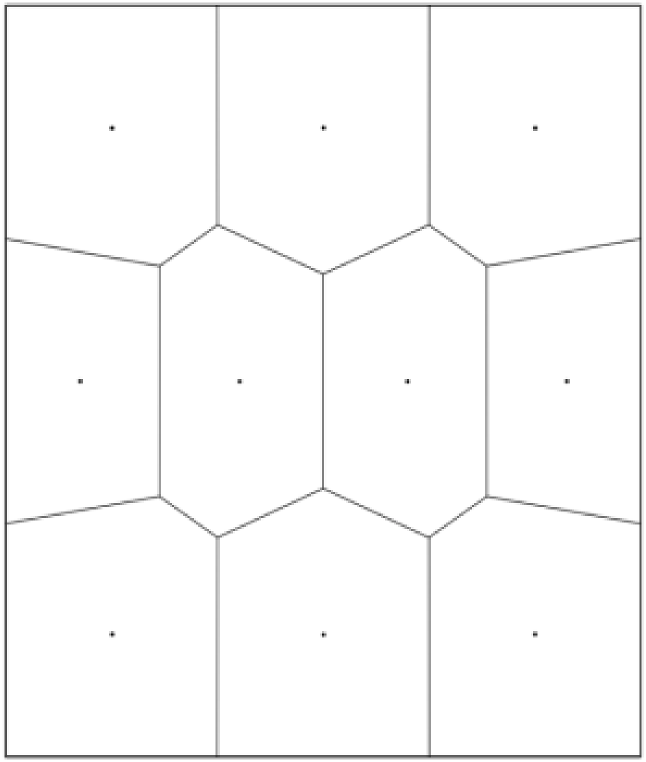


Centroidal Voronoi Tessellation



$\mathbf{z}_i \neq \mathbf{z}_i^*, \ i = 1, \ldots, k$

(Ju et al. 2002)



$\mathbf{z}_i = \mathbf{z}_i^*, \ i = 1, \ldots, k$

1. **若無特別處理，質心(zi*)與參考點zi並不會重合**
2. **我們希望網格各中心連線是垂直平分,且交點恰為質心**

Evolution and Strategy

- What was your goal coming here?
  - Porting on GPU
  - Learn to use openACC

- What was your initial strategy?
  - Modify the source code from openMP to openACC

- How did this strategy change?
  - Under planned (stick to the plan.)

Progress

- Two parts of computation:
  1. Voronoi computation: on CPU
  2. Centroid computation: openMP -> openACC (on GPU)
- Porting part 2 (Centroid computation) on A100
- Compiler: gfortran VS pgfortran VS nvfortran

gfortran

```
Time for Voronoi computation:     0.15 s ,     1000000 Hz
Time for centroid computation:     0.28 s ,     1000000 Hz
       401410         2000    8.8188361151890690E-010    5.1832255029052828E-010
Time for Voronoi computation:     0.15 s ,     1000000 Hz
Time for centroid computation:     0.23 s ,     1000000 Hz
       401410         2100    8.8181093699007062E-010    5.1827546100969774E-010
Time for Voronoi computation:     0.14 s ,     1000000 Hz
Time for centroid computation:     0.25 s ,     1000000 Hz
       401410         2200    8.8172499566554482E-010    5.1822837889265424E-010
```

pgfortran

```
Time for Voronoi computation:     0.15 s ,     1000000 Hz
Time for centroid computation:     0.24 s ,     1000000 Hz
       401410         1100    8.8260357064804557E-010    5.1874652629835801E-010
Time for Voronoi computation:     0.15 s ,     1000000 Hz
Time for centroid computation:     0.27 s ,     1000000 Hz
       401410         1200    8.8253126936933654E-010    5.1869940007437832E-010
Time for Voronoi computation:     0.16 s ,     1000000 Hz
Time for centroid computation:     0.21 s ,     1000000 Hz
       401410         1300    8.8244646233434736E-010    5.1865227956434174E-010
```

nvfortran

```
Time for Voronoi computation:     0.16 s ,     1000000 Hz
Time for centroid computation:     0.03 s ,     1000000 Hz
       401410          400    8.8316747102826590E-010    5.1907652823898715E-010
Time for Voronoi computation:     0.16 s ,     1000000 Hz
Time for centroid computation:     0.03 s ,     1000000 Hz
       401410          500    8.8308888330564438E-010    5.1902937352481088E-010
Time for Voronoi computation:     0.15 s ,     1000000 Hz
Time for centroid computation:     0.03 s ,     1000000 Hz
       401410          600    8.8300602649377994E-010    5.1898221990084138E-010
```

Use NVTX

- use nvtx
- call nvtxStartRange(“…”)

  …..

  nvtxEndRange
- call nvtxStartRange(“...”)

  …..

  nvtxEndRange
- FFLAGS = -acc … -l nvhpcwrapnvtw

  …

  LDFLAGS = -acc … -l nvhpcwrapnvtw

```fortran
+-- 76 lines: module voronoi_utils---------------------------------------------------
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   ! SUBROUTINE COMPUTE_VC
   !
   ! Compute the Voronoi corners of a set of lat/lon locations.
   !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
   subroutine compute_vc(rlat, rlon, n, nrow, ntmx, list, lptr, lend, listc, vclat,
      use nvtx

      implicit none

      integer, intent(in) :: n, nrow, ntmx, nvc
      integer, dimension(nvc), intent(inout) :: list, lptr, listc
      real, dimension(nvc), intent(inout) :: vclat, vclon
+--  9 lines: integer, dimension(n), intent(inout) :: lend-------------------------

      if (nvc < 6*n-12) then
         write(0,*) 'Error: Argument nvc to COMPUTE_VC must be at least 6*n+12'
         return
      end if

      call nvtxStartRange("Trans")
      call trans(n, rlat, rlon, x, y, z)

      !
      ! It is time again to check whether our triangulation is still valid
      !
      call nvtxEndRange
      if (ttl == 0) then

         !
         ! Compute triangulation based on current generating points
         !
         call nvtxStartRange("trmesh")
         call trmesh(n, x, y, z, list, lptr, lend, lnew, near, next, dist, ierr)
         call nvtxEndRange
         if (ierr /= 0) then
            write(0,*) 'Error: TRMESH returned error code ',ierr
         end if
```

```makefile
LDFLAGS = -O3 -fopenmp
endif

ifeq ($(CORE), nv)
FC = nvfortran
CC = gcc
FFLAGS =  -acc -Minfo -fast -Mvect=levels:5  -l nvhpcwrapnvtx
#FFLAGS =  -acc -Minfo -fast
F77FLAGS =
CFLAGS = -O3
CPPFLAGS = -DRKIND=8
PROMOTION = -r8
LDFLAGS = -acc -Minfo  -fast -l nvhpcwrapnvtx
endif
#FC = xlf2003_r
#CC = xlc_r
#FFLAGS = -O4 -qsmp=omp
#F77FLAGS = -qfixed -O4 -qsmp=omp
#CFLAGS = -O4
+-- 11 lines: CPPFLAGS = -DRKIND=8 -Uvector-------------------------------
#LDFLAGS = -Ofast -mp

ifeq ($(CORE), pg)
FC = pgfortran
CC = pgcc
FFLAGS = -Mfree -O3 -mp
F77FLAGS = -O3
CFLAGS = -O3
CPPFLAGS = -DRKIND=8
PROMOTION = -r8
```

Results and Final Profile

- What were you able to accomplish?
  - 2x faster (*256 CPU vs 1GPU numbers)
- What did you learn?
  - Syntax difference & Nsight profiler & nvtx label
  - Achieved new scientific goals?
    - Speed up about 8x (only modify 25 lines)

Process Overview

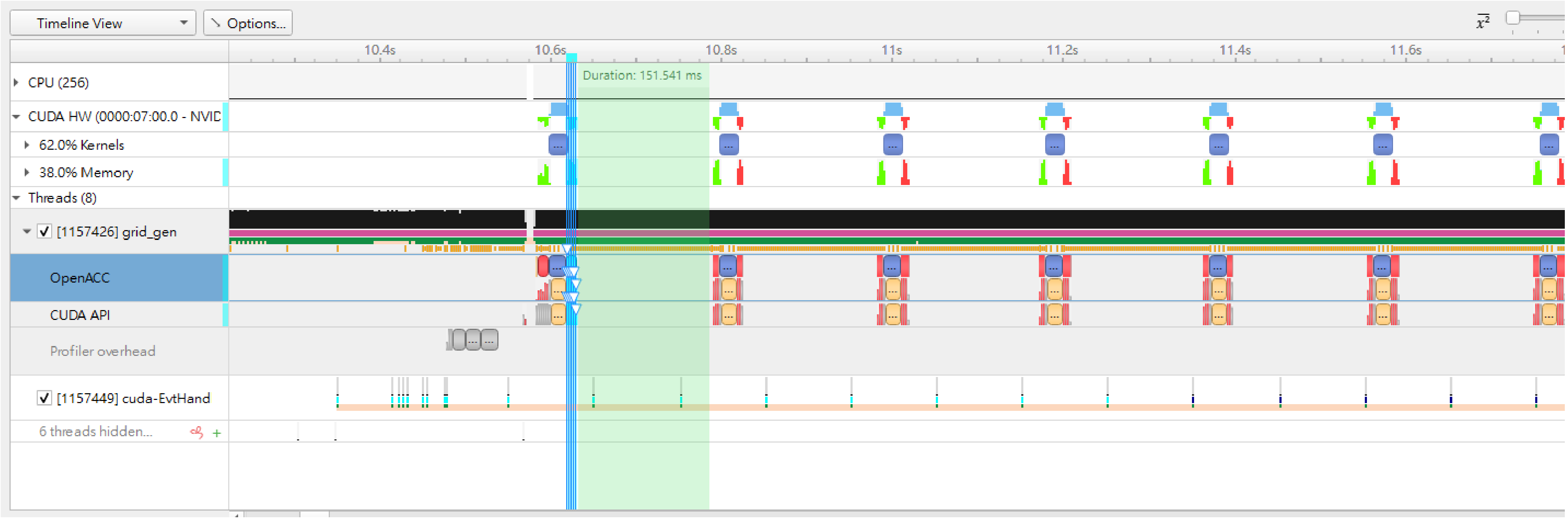
Voronoi Computation: No differences among 3 scenarios

Centroid Computation: nvfortran (on GPU) speeds up, only need 1/8

	gfortran	pgfortran	nvfortran
Voronoi computation	0.15	0.15	0.15
Centroid computation	0.25	0.24	0.03
Total	0.4	0.39	0.18
Ratio	1	0.975	0.45

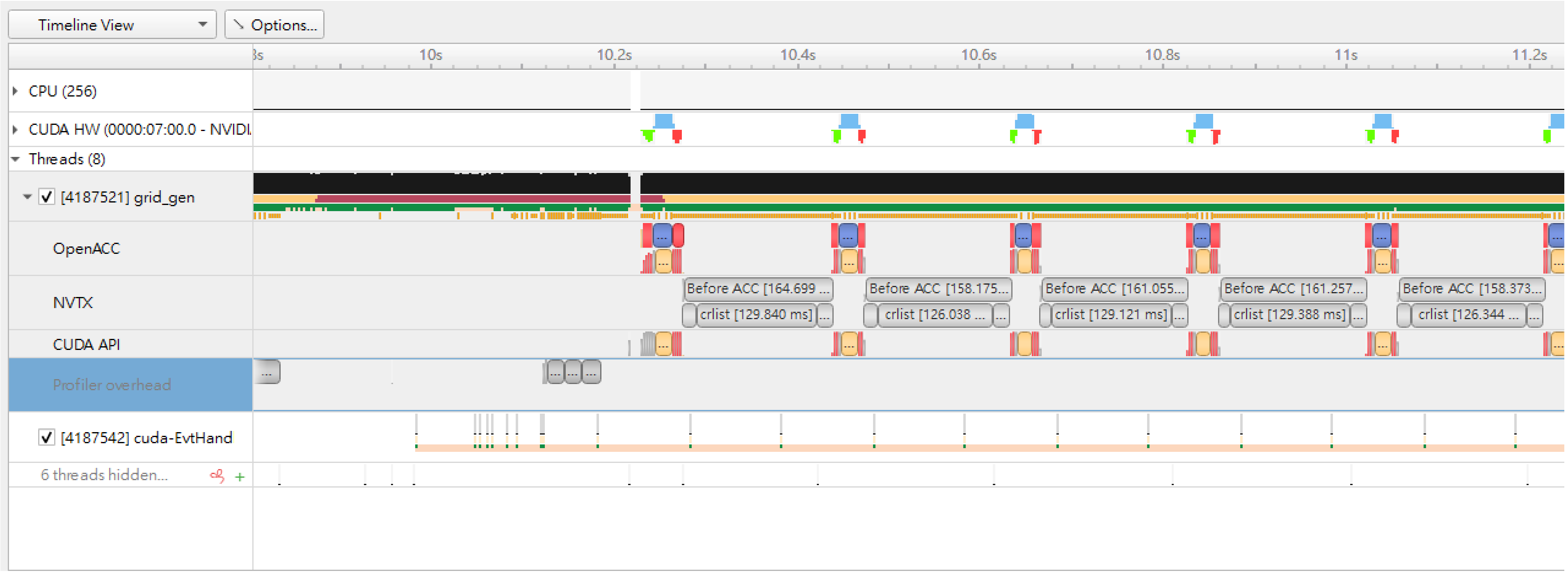
Profiler Output

- Nsight

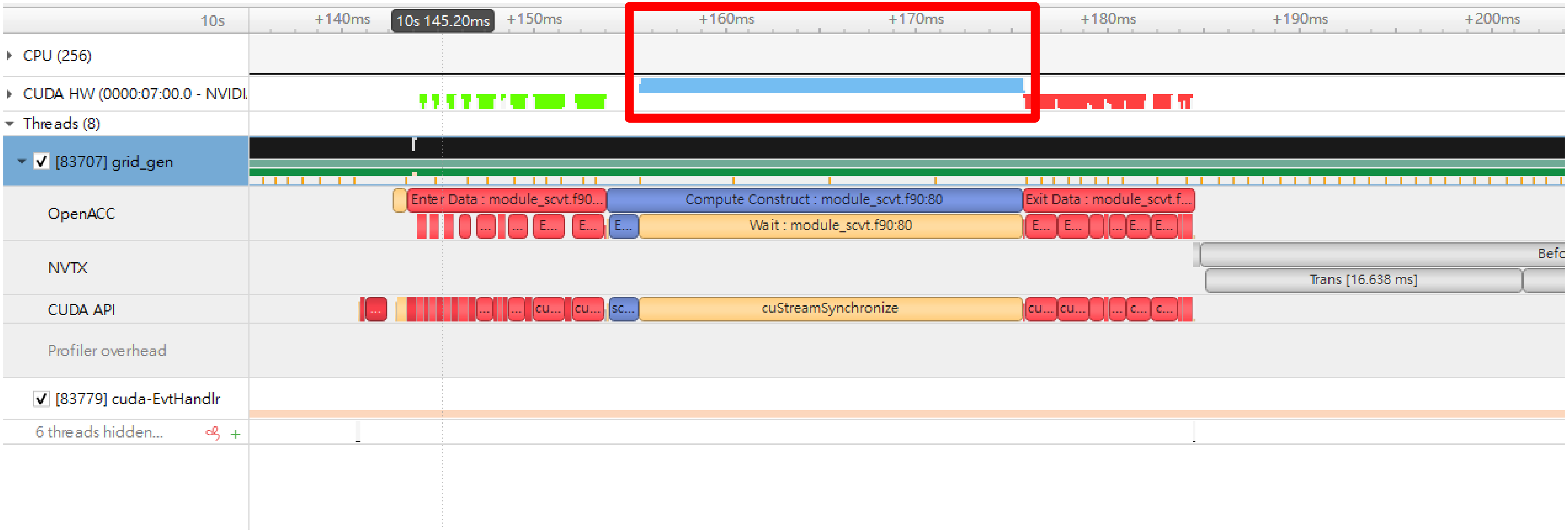


Profiler Output

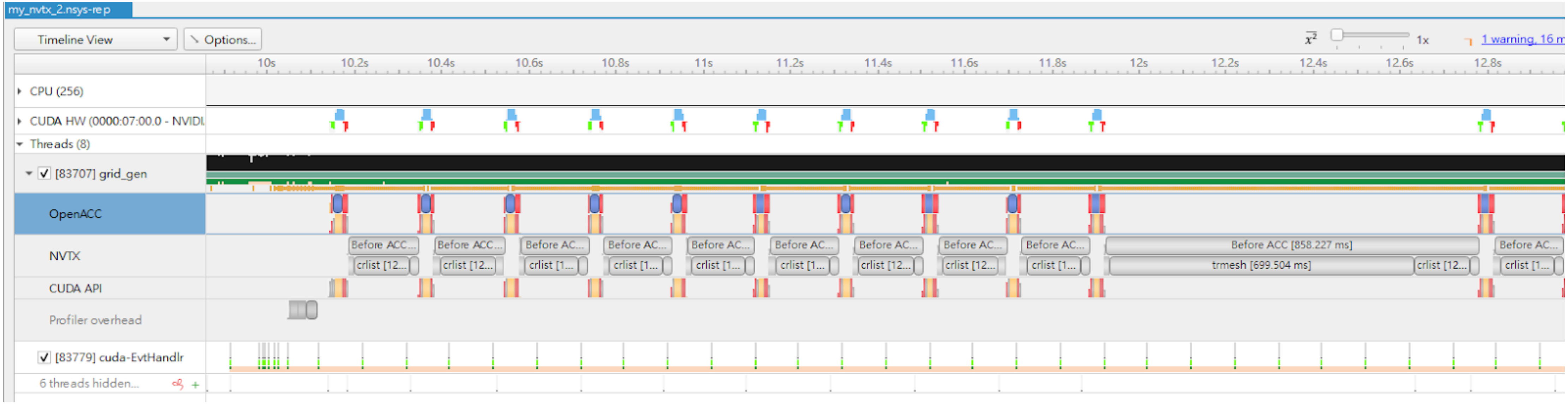
- add nvtx for CPU part



Profiler Output



Process Overview



Problems and Solutions

- Allocation and deallocation of array of type is not supported on device

```fortran
    type (geo_point), allocatable, dimension(:)
    type (geo_point), allocatable, dimension(:,

    glevel = nint(log(real(n)) / log(4.0))   ! E
                                             !

    glevel = (2 ** glevel) + 1
    allocate(line(glevel, glevel))
    allocate(p1p2(glevel))
    allocate(p1p3(glevel))
```

```fortran
    type (geo_point), dimension(5) :: p1p2, p1
    type (geo_point), dimension(5,5) :: line

!   glevel = nint(log(real(n)) / log(4.0))   !
!                                            !
```

Problems and Solutions

- Syntax difference between gfortran(left) and nvfortran(right)

```fortran
do while (k /= lend(i))
    k = lptr(k)
    p2 = p3

    p3%lat = vclat(listc(k)
    p3%lon = vclon(listc(k)
```

```fortran
do while (k /= lend(i))
    k = lptr(k)
    p2%lat = p3%lat
    p2%lon = p3%lon
    p3%lat = vclat(listc(
    p3%lon = vclon(listc(
```

- The 'write' formatting in Fortran is not supported by the nvfortran compiler

Energy Efficiency

INPUTS	
# CPU Cores	256
# GPUs (A100)	1
Application Speedup	8.0x

Node Replacement	128.0x

GPU NODE POWER SAVINGS			
	AMD Dual Rome 7742	8x A100 80GB SXM4	Power Savings
Compute Power (W)	140,800	6,500	134,300
Networking Power (W)	5,944	93	5,851
Total Power (W)	**146,744**	**6,593**	140,151

Node Power efficiency	22.3x

ANNUAL ENERGY SAVINGS PER GPU NODE			
	AMD Dual Rome 7742	8x A100 80GB SXM4	Power Savings
Compute Power (kWh/year)	1,233,408	56,940	1,176,468
Networking Power (kWh/year)	52,069	814	51,255
Total Power (kWh/year)	**1,285,477**	**57,754**	**1,227,723**

$/kWh	$ 0.34
Annual Cost Savings	$ 417,425.82
3-year Cost Savings	$ 1,252,277.46

Metric Tons of CO2	870
Gasoline Cars Driven for 1 year	188
Seedlings Trees grown for 10 years	14,389

(source: Link)

POWER ASSUMPTIONS		
Node Configurations	**Baseline Node** AMD Dual Rome 7742	**Alternative** 8x A100 80GB SXM4
CPU SKU	7742	7742
# CPU	2	2
# CPU Cores	128	128
CPU Power (W)	450	450
GPU SKU	0	A100 80GB SXM4
# GPU	0	8
GPU Power (W)	0	3200
Network Type	IB EDR	IB EDR
# Network Ports	1	2
Network Card Power (W)	30	60
RBoM Power (W)	300	450
Total Compute Node Power (W)	1100	6500
Core Network Power / Node	46	93
Total Power / Node	**1146**	**6593**

ASSUMPTIONS

(1) The workload being input will run 24/7/365 on the node in question

(2) When the workload runs on a fraction of a CPU or GPU server, no other bottlenecks occur to stop it from scaling up to occupy the full server

(3) The calculations use TDP for both CPU and GPU. In reality, neither server will run full time at TDP. The comparison here is "worst case CPU" vs. "worst case GPU"

(4) Annual cost savings are operational for electricity only. Capital, personnel, etc are not included

Wishlist

- What do you wish existed to make your life easier?
  - Automatic Tools (openMP to openACC)
  - Lessons
  - Expand the research probability

Please use 100 words to summarize your team's achievements during this Hackathon

1. Realize the skills on GPUs, including segmentation and parallelization.
2. Adapt existing Fortran code from openMP to OpenACC , and achieve 8x speed-up on Centroid computation.
3. Realize the property of the mesh generation code, and find the bottleneck of acceleration.
4. Acquire proficiency in profiling tools (nsight) for code analysis.

Future work:
Endeavor to apply in research and operational environments in CWA.