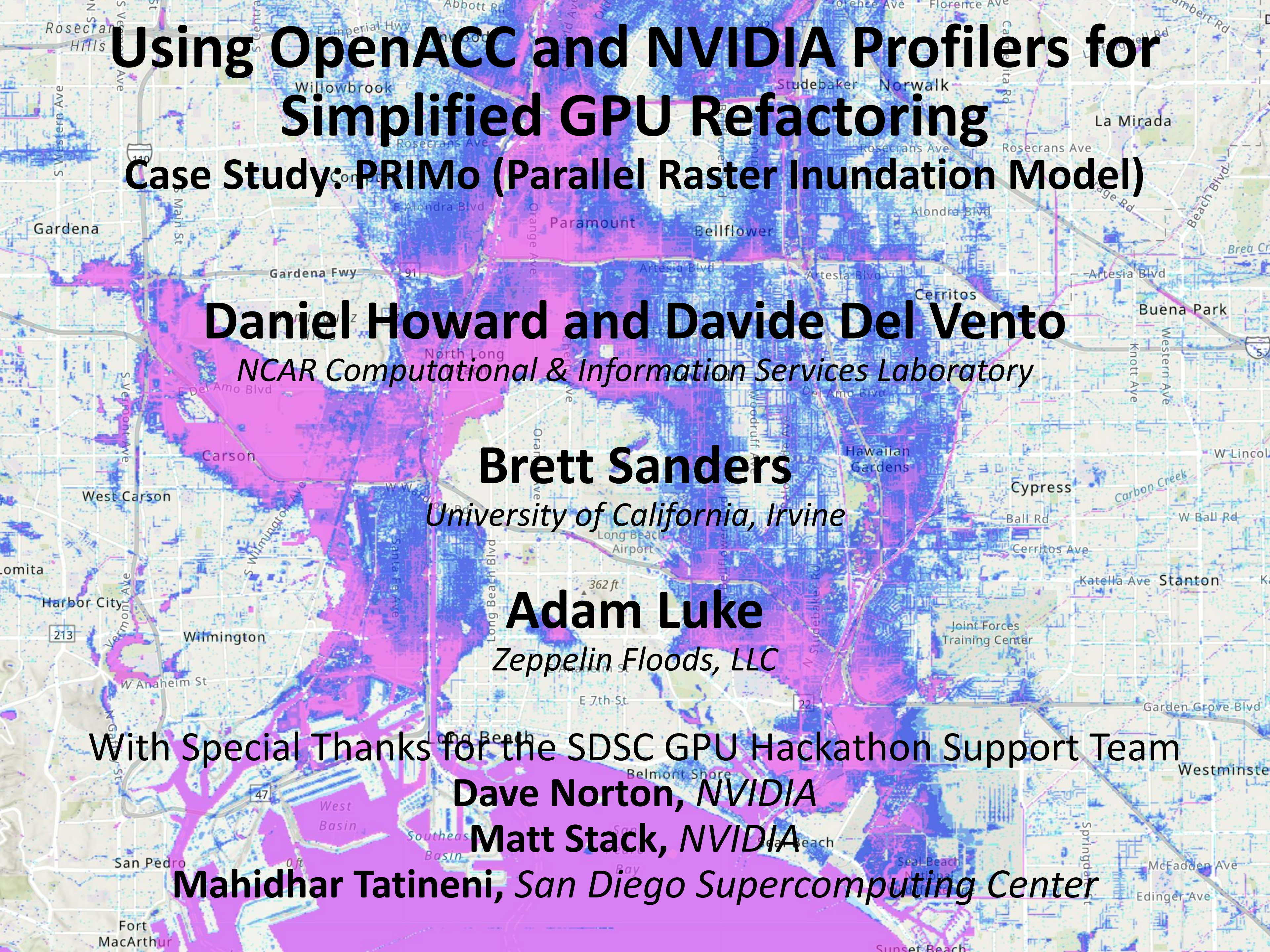


Using OpenACC and NVIDIA Profilers for Simplified GPU Refactoring

Case Study: PRIMo (Parallel Raster Inundation Model)

Daniel Howard and Davide Del Vento
NCAR Computational & Information Services Laboratory

Brett Sanders
University of California, Irvine

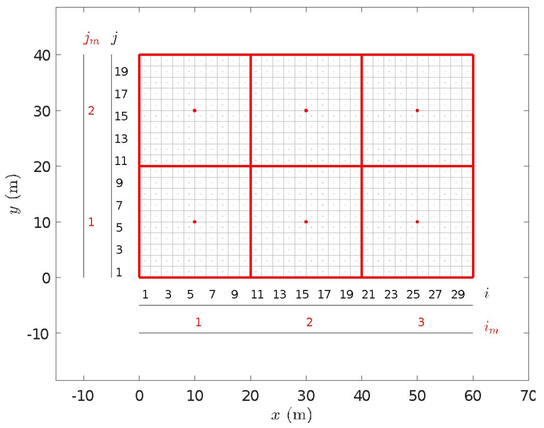
Adam Luke
Zeppelin Floods, LLC

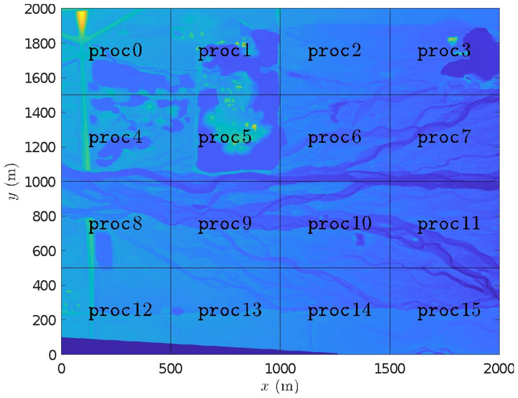
With Special Thanks for the SDSC GPU Hackathon Support Team
Dave Norton, *NVIDIA*
Matt Stack, *NVIDIA*
Mahidhar Tatineni, *San Diego Supercomputing Center*

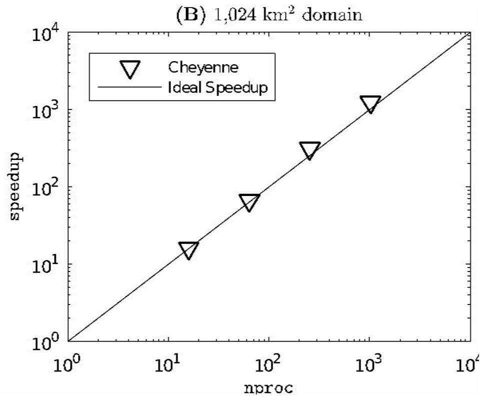
PRIMo App

- Explicit finite-volume scheme
- Solution on a cartesian grid
  - Compute-intensive Riemann solvers
  - Dual grid to exploit detailed topographic data
    - Look-up tables requiring piecewise linear interpolation
- CPU Parallelism
  - Single Process Multiple Data (SPMD) design with
  - domain decomposition (Sanders and Schubert, 2019)
- No libraries
- Fortran language
- Our focus
  - Performance on a single GPU using OpenACC
  - Multi-GPU implementation with MPI & OpenACC

See [2019 publication] of CPU based MPI version of PRIMo







Initial Work Adding OpenACC

Code involves an initial loading of raster data. This section could not be accelerated on GPU due to data movement limitations. *Parallel I/O separate goal*

Primary target for OpenACC regions was the main timestepping loop and associated routines, including Riemann Flux solver routines.

Suggested Approach, initially with serial/single GPU model…

1. Start with selecting and adding `!$acc kernels/parallel` regions
2. Verify correctness of code, can use `-ta=autocompare` at compile
3. Then add `!$acc data` regions to minimize data movement
4. Verify correctness again and iterate.

Parallel regions without data regions forces compiler to be robustly conservative in ensuring correctness, copying all needed variables when entering/exiting parallel regions.

Will initially be slower. **But much harder to isolate bugs if you skip steps.**

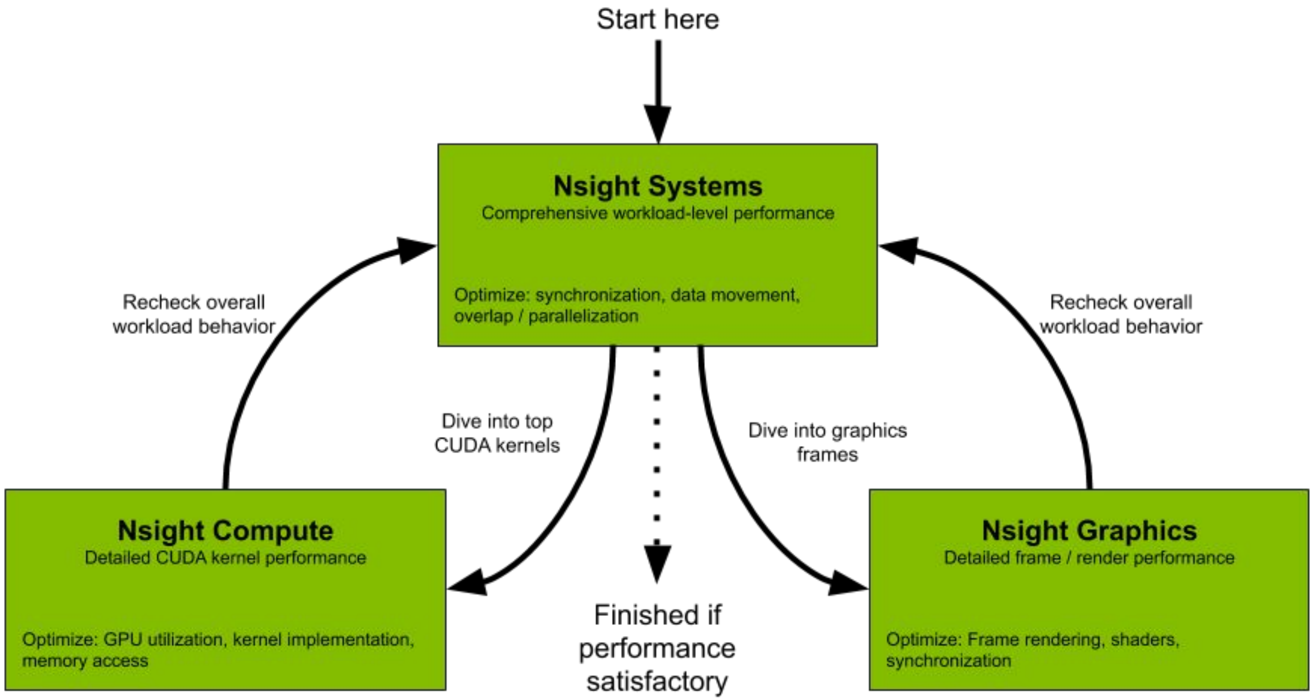
PRIMo Pseudocode

```
!!! Data clauses !!!
!$acc enter data copyin(u,v,eta) create(detax, detay)

!!! Main time loop !!!
do while (t<=tstop .and. n<=ntmax)

    ! Call needed functions. GPU kernels are inside functions
    call fluxes(nxu,nyu,eta,detax,detay)
    ...

    ! Make sure GPU vars are on CPU
    !$acc update host(maxlambda)
    !!! Run any CPU code here !!!
    !$acc update device(maxlambda)

    call advance ! Advance solution to the next time step
    !$acc update host(dt)
    t=t+dt
 enddo

!!! End time loop and data region !!!
!$acc exit data copyout(eta) delete(detax, detay)
```

Data region can be as big as you want!

Profiling with NVIDA NSight

NVIDIA NSight ecosystem has excellent tools to assess performance of a model and isolate where additional refactoring is required. Included with NVHPC SDK



Start here

Nsight Systems
Comprehensive workload-level performance

Optimize: synchronization, data movement, overlap / parallelization

Recheck overall workload behavior

Dive into top CUDA kernels

Dive into graphics frames

Recheck overall workload behavior

Nsight Compute
Detailed CUDA kernel performance

Optimize: GPU utilization, kernel implementation, memory access

Finished if performance satisfactory

Nsight Graphics
Detailed frame / render performance

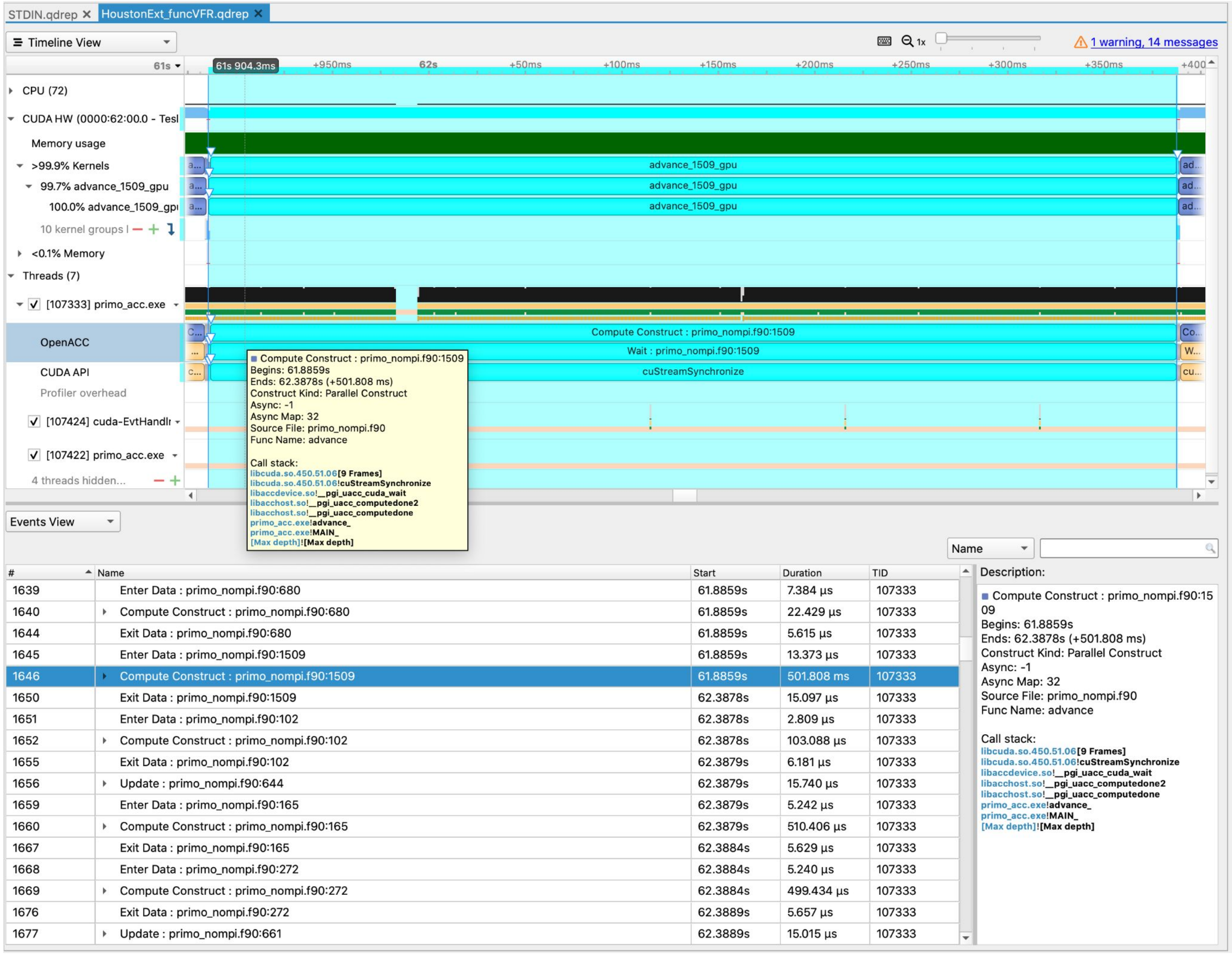
Optimize: Frame rendering, shaders, synchronization

Typical to start with `nsys` then isolate to specific GPU kernel(s) with `ncu`
Simple to run… `nsys profile program.exe`

Good added options include below or use `man nsys/ncu` & see docs…

```
nsys -o $PBS_JOBNAME --trace=openacc,cuda
--cuda-memory-usage=true --stats=true ...
```

PRIMo - Initial Profiling Results



Compute Construct : primo_nompi.f90:1509
Begins: 61.8859s
Ends: 62.3878s (+501.808 ms)
Construct Kind: Parallel Construct
Async: -1
Async Map: 32
Source File: primo_nompi.f90
Func Name: advance

Call stack:
libcuda.so.450.51.06[9 Frames]
libcuda.so.450.51.06!cuStreamSynchronize
libaccdevice.so!__pgi_uacc_cuda_wait
libacchost.so!__pgi_uacc_computedone2
libacchost.so!__pgi_uacc_computedone
primo_acc.exe!advance_
primo_acc.exe!MAIN_
[Max depth]![Max depth]

#	Name	Start	Duration	TID
1639	Enter Data : primo_nompi.f90:680	61.8859s	7.384 µs	107333
1640	Compute Construct : primo_nompi.f90:680	61.8859s	22.429 µs	107333
1644	Exit Data : primo_nompi.f90:680	61.8859s	5.615 µs	107333
1645	Enter Data : primo_nompi.f90:1509	61.8859s	13.373 µs	107333
1646	Compute Construct : primo_nompi.f90:1509	61.8859s	501.808 ms	107333
1650	Exit Data : primo_nompi.f90:1509	62.3878s	15.097 µs	107333
1651	Enter Data : primo_nompi.f90:102	62.3878s	2.809 µs	107333
1652	Compute Construct : primo_nompi.f90:102	62.3878s	103.088 µs	107333
1655	Exit Data : primo_nompi.f90:102	62.3879s	6.181 µs	107333
1656	Update : primo_nompi.f90:644	62.3879s	15.740 µs	107333
1659	Enter Data : primo_nompi.f90:165	62.3879s	5.242 µs	107333
1660	Compute Construct : primo_nompi.f90:165	62.3879s	510.406 µs	107333
1667	Exit Data : primo_nompi.f90:165	62.3884s	5.629 µs	107333
1668	Enter Data : primo_nompi.f90:272	62.3884s	5.240 µs	107333
1669	Compute Construct : primo_nompi.f90:272	62.3884s	499.434 µs	107333
1676	Exit Data : primo_nompi.f90:272	62.3889s	5.657 µs	107333
1677	Update : primo_nompi.f90:661	62.3889s	15.015 µs	107333

Description:

Compute Construct : primo_nompi.f90:1509
Begins: 61.8859s
Ends: 62.3878s (+501.808 ms)
Construct Kind: Parallel Construct
Async: -1
Async Map: 32
Source File: primo_nompi.f90
Func Name: advance

Call stack:
libcuda.so.450.51.06[9 Frames]
libcuda.so.450.51.06!cuStreamSynchronize
libaccdevice.so!__pgi_uacc_cuda_wait
libacchost.so!__pgi_uacc_computedone2
libacchost.so!__pgi_uacc_computedone
primo_acc.exe!advance_
primo_acc.exe!MAIN_
[Max depth]![Max depth]

OpenACC automatically names kernels based on subroutine and line number.

Can further segment profile by adding NVTX ranges and appropriately linking NVTX library at compile time. Need NVTX module for Fortran.

Advance routine initially a bottleneck for the application.

Fix Advance - Subroutines in OpenACC Compute Kernels

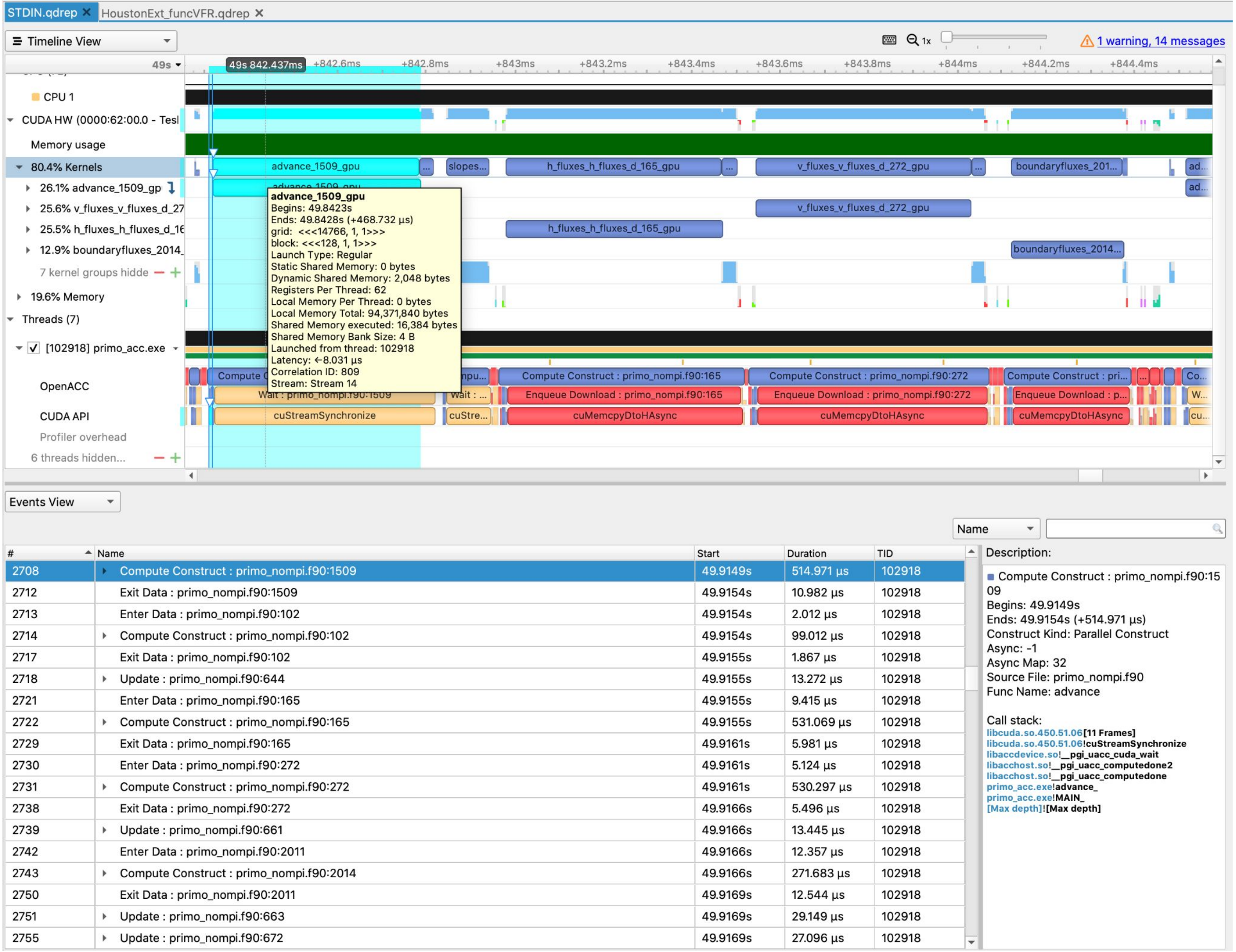
Testing subroutines called inside advance routine’s GPU kernel, we determined that the compiler was casing excess data movement when calling vfr() routine.

To note, it’s possible and usually sufficient to use `!$acc routine` declaration to tell compiler to create GPU code for a function called inside a kernel. But...

```
subroutine advance
    !$acc routine(vfr) seq

    !$acc parallel loop
    do …
        …
        vfr(x,y,z(1:small_range))
    enddo
end subroutine advance
```

Problem arises passing slice or subrange of variable for Fortran function.
New sub data objects are instantiated at each call, slowing the program.

PRIMo – Fix by Inlining Subroutine



advance_1509_gpu
Begins: 49.8423s
Ends: 49.8428s (+468.732 µs)
grid: <<<14766, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 2,048 bytes
Registers Per Thread: 62
Local Memory Per Thread: 0 bytes
Local Memory Total: 94,371,840 bytes
Shared Memory executed: 16,384 bytes
Shared Memory Bank Size: 4 B
Launched from thread: 102918
Latency: ←8.031 µs
Correlation ID: 809
Stream: Stream 14

#	Name	Start	Duration	TID
2708	Compute Construct : primo_nompi.f90:1509	49.9149s	514.971 µs	102918
2712	Exit Data : primo_nompi.f90:1509	49.9154s	10.982 µs	102918
2713	Enter Data : primo_nompi.f90:102	49.9154s	2.012 µs	102918
2714	Compute Construct : primo_nompi.f90:102	49.9154s	99.012 µs	102918
2717	Exit Data : primo_nompi.f90:102	49.9155s	1.867 µs	102918
2718	Update : primo_nompi.f90:644	49.9155s	13.272 µs	102918
2721	Enter Data : primo_nompi.f90:165	49.9155s	9.415 µs	102918
2722	Compute Construct : primo_nompi.f90:165	49.9155s	531.069 µs	102918
2729	Exit Data : primo_nompi.f90:165	49.9161s	5.981 µs	102918
2730	Enter Data : primo_nompi.f90:272	49.9161s	5.124 µs	102918
2731	Compute Construct : primo_nompi.f90:272	49.9161s	530.297 µs	102918
2738	Exit Data : primo_nompi.f90:272	49.9166s	5.496 µs	102918
2739	Update : primo_nompi.f90:661	49.9166s	13.445 µs	102918
2742	Enter Data : primo_nompi.f90:2011	49.9166s	12.357 µs	102918
2743	Compute Construct : primo_nompi.f90:2014	49.9166s	271.683 µs	102918
2750	Exit Data : primo_nompi.f90:2011	49.9169s	12.544 µs	102918
2751	Update : primo_nompi.f90:663	49.9169s	29.149 µs	102918
2755	Update : primo_nompi.f90:672	49.9169s	27.096 µs	102918

Description:

Compute Construct : primo_nompi.f90:1509
Begins: 49.9149s
Ends: 49.9154s (+514.971 µs)
Construct Kind: Parallel Construct
Async: -1
Async Map: 32
Source File: primo_nompi.f90
Func Name: advance

Call stack:
libcuda.so.450.51.06**[11 Frames]**
libcuda.so.450.51.06!**cuStreamSynchronize**
libaccdevice.so!**__pgi_uacc_cuda_wait**
libacchost.so!**__pgi_uacc_computedone2**
libacchost.so!**__pgi_uacc_computedone**
primo_acc.exe!**advance_**
primo_acc.exe!**MAIN_**
[Max depth]!**[Max depth]**

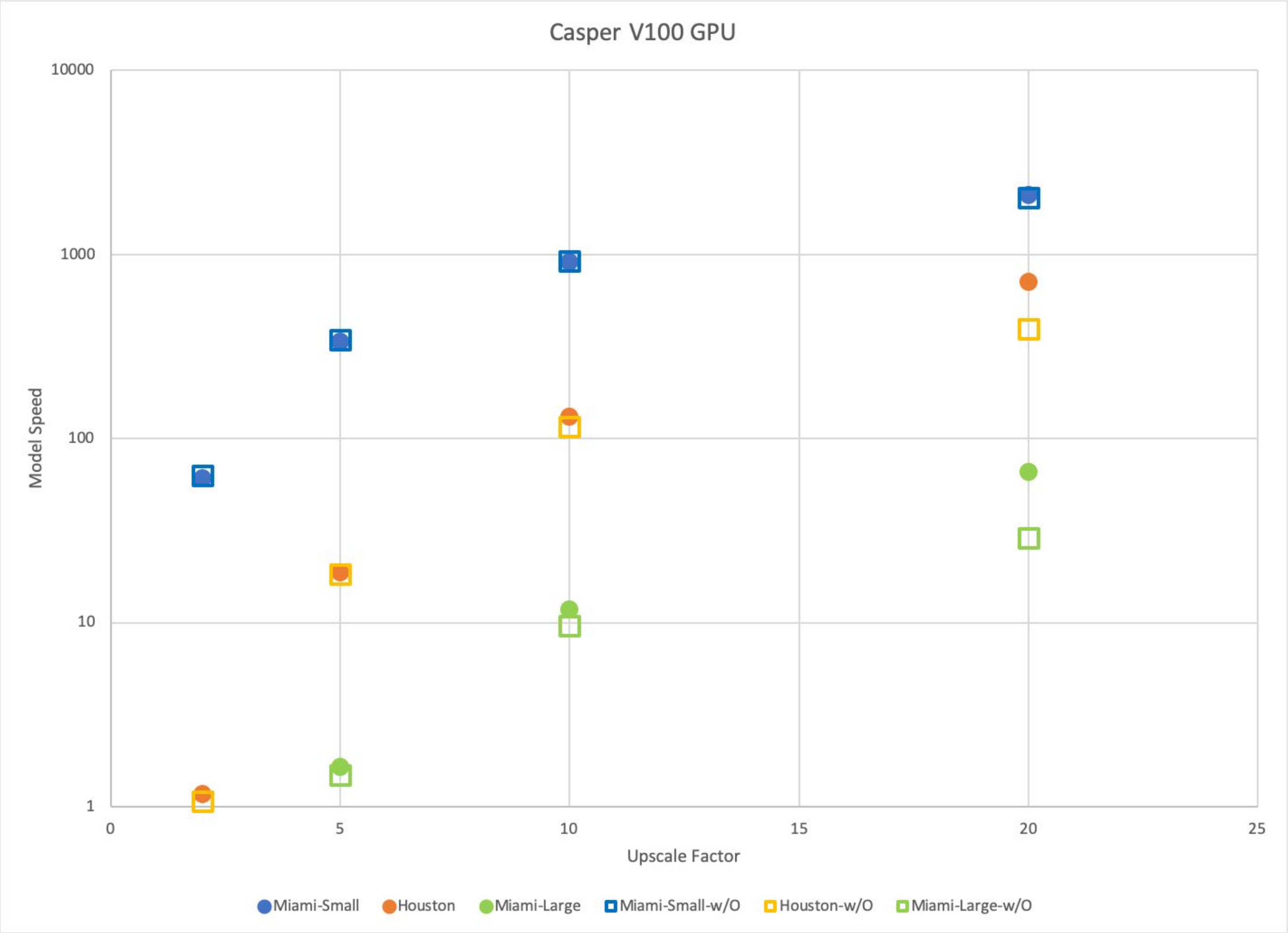
Can actually see all routines now.

The runtime split ratio for each kernel matches closely to a CPU only profile of the model using Arm Forge.

Further optimizations can be pursued by using NVTX ranges to isolate sections of code to similar kernel chunks.

For 27,000 x 7,000 grid test case of Hurricane Harvey for 100s test run, **~56x speedup** **1 V100 GPU vs 1 CPU**
This speedup is for main compute region of model and does not include initialization or I/O.

Preliminary Performance Results

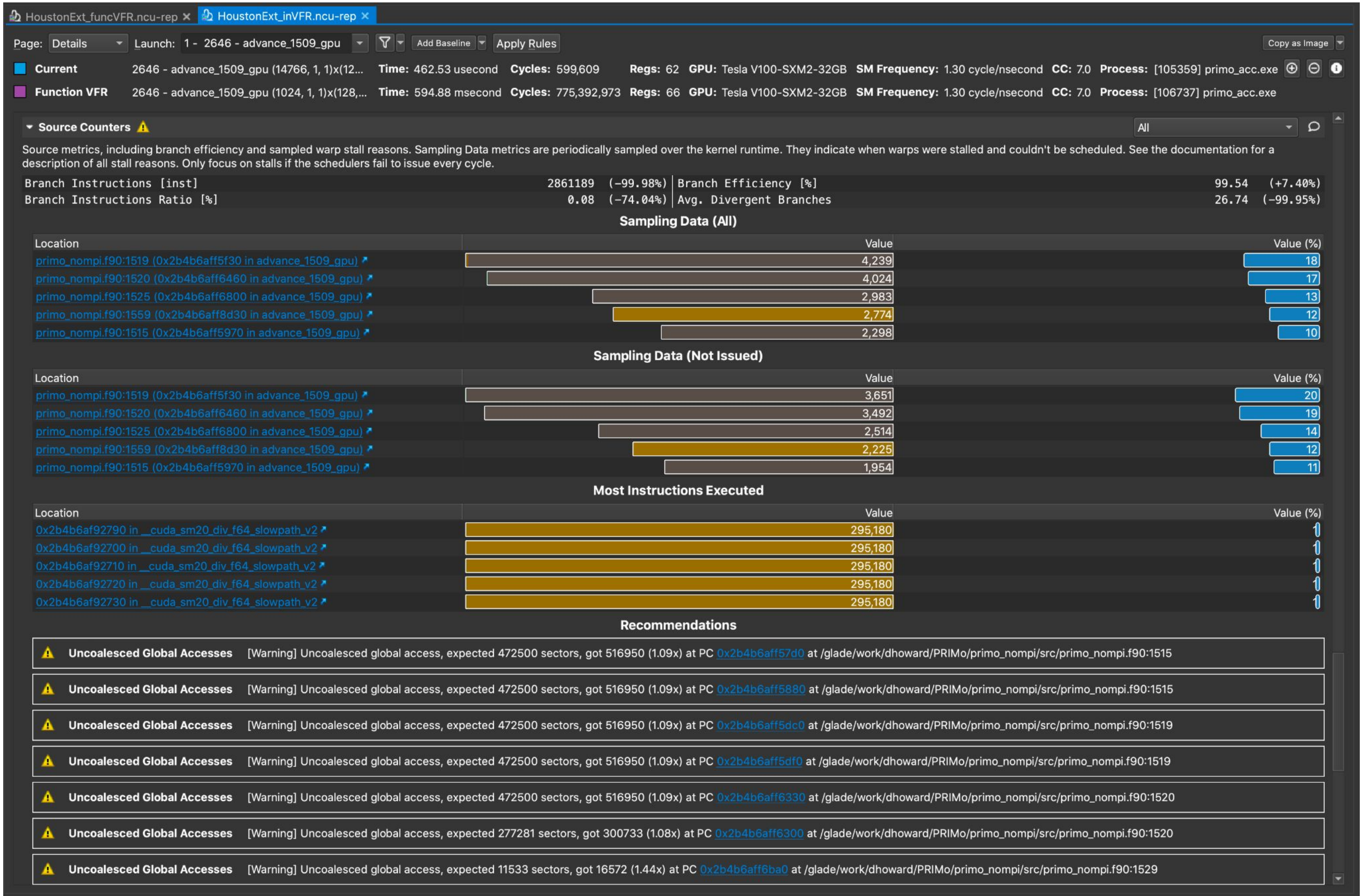


Still need to address initialization I/O speeds and multi-GPU implementation correctness

Analysis of Kernels with `ncu`

Simple to run, see docs… `ncu --set full -f -o HoustonExt_funcVFR --kernel-regex advance_1509_gpu --launch-skip 85 --launch-count 2 ./bin/primo_acc.exe $MODEL`

`advance()` metrics for inlined VFR version compared to function call VFR baseline via Nsight Compute. Subsets of GUI metrics below but can be obtuse. See recommendation flags. Hover cursor in GUI for details.



🟦 **Current**	2646 - advance_1509_gpu (14766, 1, 1)x(12…	**Time:** 462.53 usecond	**Cycles:** 599,609	**Regs:** 62	**GPU:** Tesla V100-SXM2-32GB	**SM Frequency:** 1.30 cycle/nsecond	**CC:** 7.0	**Process:** [105359] primo_acc.exe
🟪 **Function VFR**	2646 - advance_1509_gpu (1024, 1, 1)x(128,…	**Time:** 594.88 msecond	**Cycles:** 775,392,973	**Regs:** 66	**GPU:** Tesla V100-SXM2-32GB	**SM Frequency:** 1.30 cycle/nsecond	**CC:** 7.0	**Process:** [106737] primo_acc.exe

Source Counters ⚠

Source metrics, including branch efficiency and sampled warp stall reasons. Sampling Data metrics are periodically sampled over the kernel runtime. They indicate when warps were stalled and couldn't be scheduled. See the documentation for a description of all stall reasons. Only focus on stalls if the schedulers fail to issue every cycle.

```
Branch Instructions [inst]                 2861189  (-99.98%) Branch Efficiency [%]          99.54  (+7.40%)
Branch Instructions Ratio [%]                 0.08  (-74.04%) Avg. Divergent Branches        26.74 (-99.95%)
```

Sampling Data (All)

Location	Value	Value (%)
primo_nompi.f90:1519 (0x2b4b6aff5f30 in advance_1509_gpu)	4,239	18
primo_nompi.f90:1520 (0x2b4b6aff6460 in advance_1509_gpu)	4,024	17
primo_nompi.f90:1525 (0x2b4b6aff6800 in advance_1509_gpu)	2,983	13
primo_nompi.f90:1559 (0x2b4b6aff8d30 in advance_1509_gpu)	2,774	12
primo_nompi.f90:1515 (0x2b4b6aff5970 in advance_1509_gpu)	2,298	10

Sampling Data (Not Issued)

Location	Value	Value (%)
primo_nompi.f90:1519 (0x2b4b6aff5f30 in advance_1509_gpu)	3,651	20
primo_nompi.f90:1520 (0x2b4b6aff6460 in advance_1509_gpu)	3,492	19
primo_nompi.f90:1525 (0x2b4b6aff6800 in advance_1509_gpu)	2,514	14
primo_nompi.f90:1559 (0x2b4b6aff8d30 in advance_1509_gpu)	2,225	12
primo_nompi.f90:1515 (0x2b4b6aff5970 in advance_1509_gpu)	1,954	11

Most Instructions Executed

Location	Value	Value (%)
0x2b4b6af92790 in __cuda_sm20_div_f64_slowpath_v2	295,180	1
0x2b4b6af92700 in __cuda_sm20_div_f64_slowpath_v2	295,180	1
0x2b4b6af92710 in __cuda_sm20_div_f64_slowpath_v2	295,180	1
0x2b4b6af92720 in __cuda_sm20_div_f64_slowpath_v2	295,180	1
0x2b4b6af92730 in __cuda_sm20_div_f64_slowpath_v2	295,180	1

Recommendations

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 472500 sectors, got 516950 (1.09x) at PC 0x2b4b6aff57d0 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1515

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 472500 sectors, got 516950 (1.09x) at PC 0x2b4b6aff5880 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1515

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 472500 sectors, got 516950 (1.09x) at PC 0x2b4b6aff5dc0 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1519

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 472500 sectors, got 516950 (1.09x) at PC 0x2b4b6aff5df0 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1519

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 472500 sectors, got 516950 (1.09x) at PC 0x2b4b6aff6330 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1520

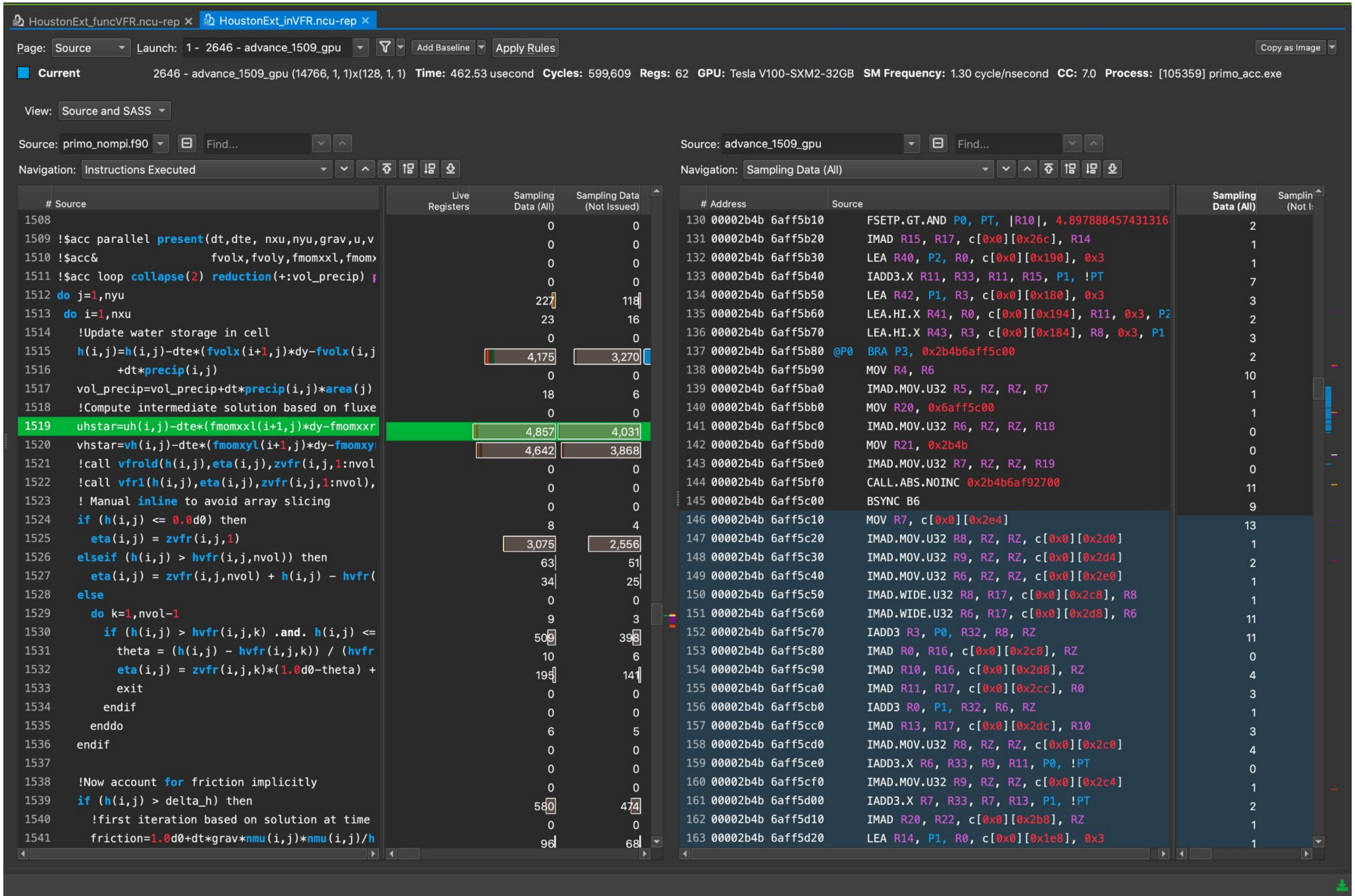
⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 277281 sectors, got 300733 (1.08x) at PC 0x2b4b6aff6300 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1520

⚠ **Uncoalesced Global Accesses** [Warning] Uncoalesced global access, expected 11533 sectors, got 16572 (1.44x) at PC 0x2b4b6aff6ba0 at /glade/work/dhoward/PRIMo/primo_nompi/src/primo_nompi.f90:1529

Source/SASS analysis with `ncu`

Under GUI “Source” page, so long as executable is compiled with line table information, can directly correlate performance time to specific lines in code and underlying assembly code.

This allows you to generate and investigate a heatmap of time spent in code as well as other stats.



Page: Source Launch: 1 - 2646 - advance_1509_gpu

Current 2646 - advance_1509_gpu (14766, 1, 1)x(128, 1, 1) **Time:** 462.53 usecond **Cycles:** 599,609 **Regs:** 62 **GPU:** Tesla V100-SXM2-32GB **SM Frequency:** 1.30 cycle/nsecond **CC:** 7.0 **Process:** [105359] primo_acc.exe

View: Source and SASS

Source: primo_nompi.f90

Navigation: Instructions Executed

```
   # Source                                                     Live       Sampling   Sampling Data
                                                                Registers  Data (All) (Not Issued)
1508                                                                                0            0
1509 !$acc parallel present(dt,dte, nxu,nyu,grav,u,v                                0            0
1510 !$acc&                    fvolx,fvoly,fmomxxl,fmom›                            0            0
1511 !$acc loop collapse(2) reduction(+:vol_precip)                                 0            0
1512 do j=1,nyu                                                                   227          118
1513  do i=1,nxu                                                                   23           16
1514    !Update water storage in cell                                               0            0
1515    h(i,j)=h(i,j)-dte*(fvolx(i+1,j)*dy-fvolx(i,j                                4,175        3,270
1516          +dt*precip(i,j)                                                       0            0
1517    vol_precip=vol_precip+dt*precip(i,j)*area(j)                               18            6
1518    !Compute intermediate solution based on fluxe                               0            0
1519    uhstar=uh(i,j)-dte*(fmomxxl(i+1,j)*dy-fmomxxr                               4,857        4,031
1520    vhstar=vh(i,j)-dte*(fmomxyl(i+1,j)*dy-fmomxy                                4,642        3,868
1521    !call vfrold(h(i,j),eta(i,j),zvfr(i,j,1:nvol                                0            0
1522    !call vfr1(h(i,j),eta(i,j),zvfr(i,j,1:nvol),                                0            0
1523    ! Manual inline to avoid array slicing                                      0            0
1524    if (h(i,j) <= 0.0d0) then                                                   8            4
1525      eta(i,j) = zvfr(i,j,1)                                                    3,075        2,556
1526    elseif (h(i,j) > hvfr(i,j,nvol)) then                                      63           51
1527      eta(i,j) = zvfr(i,j,nvol) + h(i,j) - hvfr(                               34           25
1528    else                                                                        0            0
1529      do k=1,nvol-1                                                             9            3
1530        if (h(i,j) > hvfr(i,j,k) .and. h(i,j) <=                              509          398
1531          theta = (h(i,j) - hvfr(i,j,k)) / (hvfr                               10            6
1532          eta(i,j) = zvfr(i,j,k)*(1.0d0-theta) +                              195          141
1533          exit                                                                  0            0
1534        endif                                                                   0            0
1535      enddo                                                                     6            5
1536    endif                                                                       0            0
1537                                                                                0            0
1538    !Now account for friction implicitly                                        0            0
1539    if (h(i,j) > delta_h) then                                                580          474
1540      !first iteration based on solution at time                                0            0
1541      friction=1.0d0+dt*grav*nmu(i,j)*nmu(i,j)/h                                96           68
```

Source: advance_1509_gpu

Navigation: Sampling Data (All)

```
  # Address           Source                                                  Sampling    Samplin
                                                                              Data (All)  (Not I:
130 00002b4b 6aff5b10      FSETP.GT.AND P0, PT, |R10|, 4.897888457431316            2
131 00002b4b 6aff5b20      IMAD R15, R17, c[0x0][0x26c], R14                         1
132 00002b4b 6aff5b30      LEA R40, P2, R0, c[0x0][0x190], 0x3                       1
133 00002b4b 6aff5b40      IADD3.X R11, R33, R11, R15, P1, !PT                       7
134 00002b4b 6aff5b50      LEA R42, P1, R3, c[0x0][0x180], 0x3                       3
135 00002b4b 6aff5b60      LEA.HI.X R41, R0, c[0x0][0x194], R11, 0x3, P2             2
136 00002b4b 6aff5b70      LEA.HI.X R43, R3, c[0x0][0x184], R8, 0x3, P1              3
137 00002b4b 6aff5b80 @P0  BRA P3, 0x2b4b6aff5c00                                   2
138 00002b4b 6aff5b90      MOV R4, R6                                               10
139 00002b4b 6aff5ba0      IMAD.MOV.U32 R5, RZ, RZ, R7                               1
140 00002b4b 6aff5bb0      MOV R20, 0x6aff5c00                                       1
141 00002b4b 6aff5bc0      IMAD.MOV.U32 R6, RZ, RZ, R18                              0
142 00002b4b 6aff5bd0      MOV R21, 0x2b4b                                           0
143 00002b4b 6aff5be0      IMAD.MOV.U32 R7, RZ, RZ, R19                              0
144 00002b4b 6aff5bf0      CALL.ABS.NOINC 0x2b4b6af92700                            11
145 00002b4b 6aff5c00      BSYNC B6                                                  9
146 00002b4b 6aff5c10      MOV R7, c[0x0][0x2e4]                                    13
147 00002b4b 6aff5c20      IMAD.MOV.U32 R8, RZ, RZ, c[0x0][0x2d0]                    1
148 00002b4b 6aff5c30      IMAD.MOV.U32 R9, RZ, RZ, c[0x0][0x2d4]                    2
149 00002b4b 6aff5c40      IMAD.MOV.U32 R6, RZ, RZ, c[0x0][0x2e0]                    1
150 00002b4b 6aff5c50      IMAD.WIDE.U32 R8, R17, c[0x0][0x2c8], R8                  1
151 00002b4b 6aff5c60      IMAD.WIDE.U32 R6, R17, c[0x0][0x2d8], R6                 11
152 00002b4b 6aff5c70      IADD3 R3, P0, R32, R8, RZ                                11
153 00002b4b 6aff5c80      IMAD R0, R16, c[0x0][0x2c8], RZ                           0
154 00002b4b 6aff5c90      IMAD R10, R16, c[0x0][0x2d8], RZ                          4
155 00002b4b 6aff5ca0      IMAD R11, R17, c[0x0][0x2cc], R0                          3
156 00002b4b 6aff5cb0      IADD3 R0, P1, R32, R6, RZ                                 1
157 00002b4b 6aff5cc0      IMAD R13, R17, c[0x0][0x2dc], R10                         3
158 00002b4b 6aff5cd0      IMAD.MOV.U32 R8, RZ, RZ, c[0x0][0x2c0]                    4
159 00002b4b 6aff5ce0      IADD3.X R6, R33, R9, R11, P0, !PT                         0
160 00002b4b 6aff5cf0      IMAD.MOV.U32 R9, RZ, RZ, c[0x0][0x2c4]                    1
161 00002b4b 6aff5d00      IADD3.X R7, R33, R7, R13, P1, !PT                         2
162 00002b4b 6aff5d10      IMAD R20, R22, c[0x0][0x2b8], RZ                          1
163 00002b4b 6aff5d20      LEA R14, P1, R0, c[0x0][0x1e8], 0x3                       1
```

Lots of Profiler Data… What’s Important?

Amount of information from a profiler is overwhelming. Whether you do a deep dive into these reports is up to you, but try and focus on important metrics

1. Occupancy = # active threads / max # threads per compute unit
    a. Important to check whether GPU SMs are kept busy
    b. If low occupancy, try adjusting workgroup size or kernel resource usage

2. Issue Efficiency
    a. Measurement of instructions issued per cycle vs max possible per cycle
    b. There are profiler counters to check for kernel stalls related to…
        i. Memory dependency, execution dependency, synchronization, memory throttle, constant miss, texture busy, or pipeline busy

3. Peak Bandwidth Percentage
    a. Compares your achieved bandwidth to the theoretical max.
    b. Look for opportunities to...
        i. coalesce memory loads
        ii. store values in local cache memory (OpenACC cache directive)
        iii. reuse data to minimize host to device memory movement

Takeways

1. Refactor GPU codes in **incremental steps** to avoid bugs, guarantee correctness. `-ta=autocompare` is a good tool.

2. Minimize data movement. **Consider wrapping whole compute intensive region(s) in a data region** if there is enough GPU memory.

3. Ensure performant code by **running `nsys` first (maybe Arm Map/others).** Check idle regions and/or kernel launches that use excessive amount of time.

4. **Compilers can still implement code inefficiently** when using OpenACC. Use `ncu` to verify.

5. Organizing and setting aside dedicated time for a small-medium size team to focus on improving model, such as via gpuhackathons.org, is a great option to consider. We made significant progress ourselves here.

6. Excessive time spent optimizing is less important than following good Software Engineering practices.

Some useful resources...

1. Textbooks
   a. Parallel and High Performance Computing, 9e Robey & Zamora - Chapter 13 on GPU Profiling
   b. Programming Massively Parallel Processors, 3e Kirk & Hwu

2. Courses and Webinars
   a. NCSA - Intro to Nsight Systems (2018) - video
   b. NCSA - Intro to Nsight Compute (2018) - video
   c. OLCF - Nsight Compute (2020) - video / slides
   d. UIUC ECE 408 - 4 part series on Nsight profilers (2020), Youtube
      Session 3 of 4 goes most in depth on Nsight Compute
   e. EU POP CoE - Profiling GPU Apps w/ Nsight (2020) - video / slides

3. Guides and Blog posts
   a. NVIDIA Blog - Using Nsight Compute to Inspect your Kernels
   b. NVIDIA Blog - Custom Profiles with NVTX (C/C++), post on Fortran

4. Developer Materials
   a. NVIDIA Documentation for Nsight Systems
   b. NVIDIA Documentation for Nsight Compute