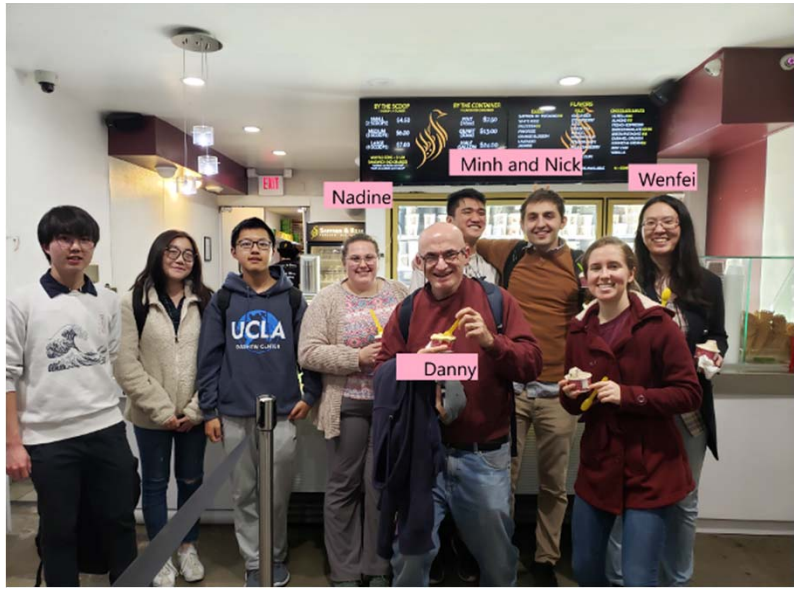


Stochastic Donkeys: StochasticGW at SDSC Hack 2020

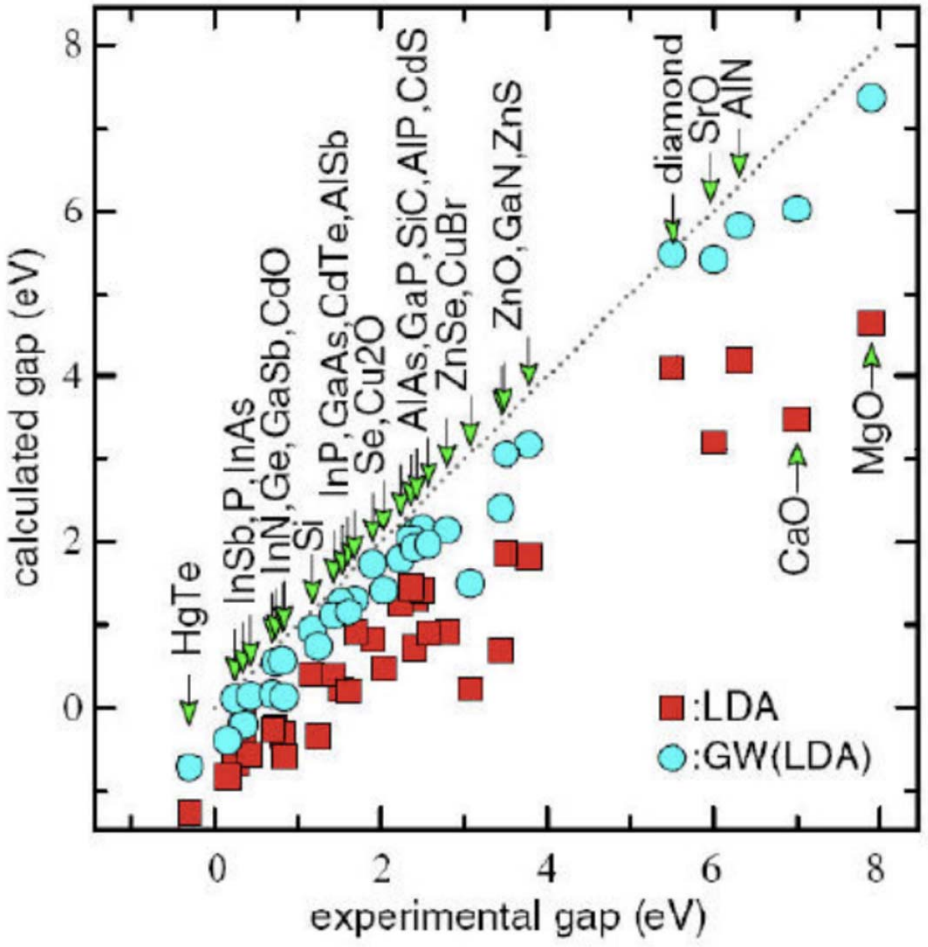
Team Members:
UCLA: Daniel Neuhauser,
Nadine Bradbury,
Wenfei Li,
Minh Nguyen,
Mykola (Nick) Sereda

LBL:
Jack DeSlippe,
Mauro Del Ben,
Chao Yang

Mentors:
LBL: Charlene Yang
Nvidia: Brent Leback
SDSC: Mahidhar Tatineni

Problems with current methods:

Motivation for StochasticGW

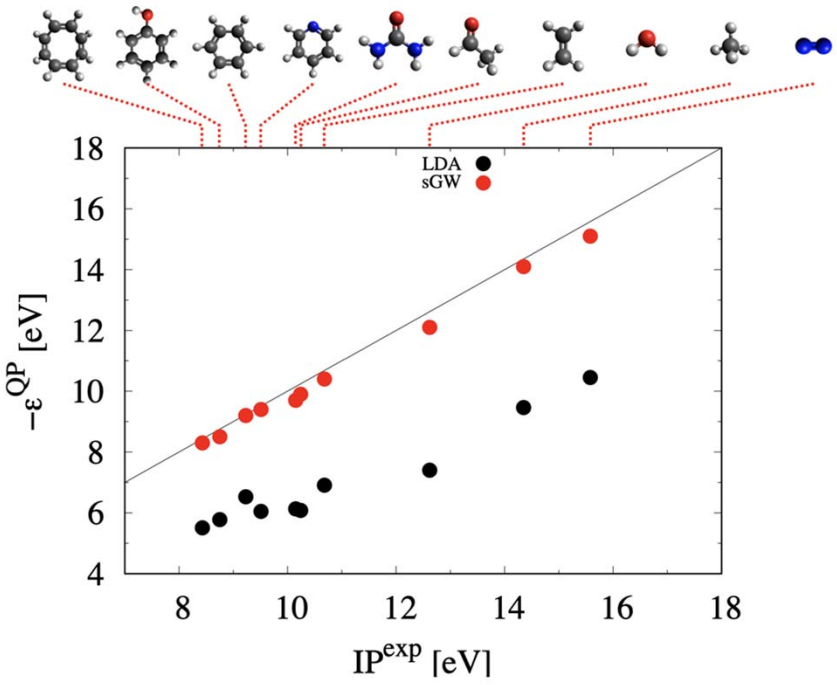


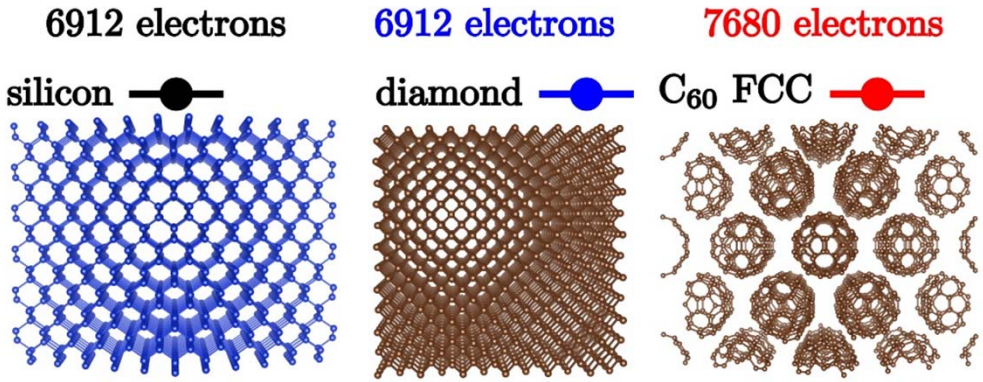
- Current QM chemistry methods like DFT(LDA) do not adequately describe electron interaction for important semiconductor systems
- Matrix Diagonalization based method, $O(N^3)$ not good for large systems. Need stochastic.

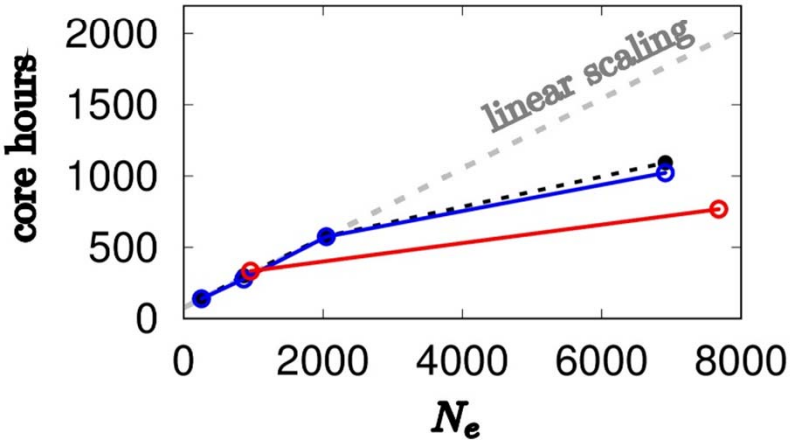
Schilfgaarde, PRL 96, 226402 (2006)

StochasticGW...

...is more accurate than traditional GW.	...scales efficiently with system size. Only Algorithm that can do 10,000+ electrons.



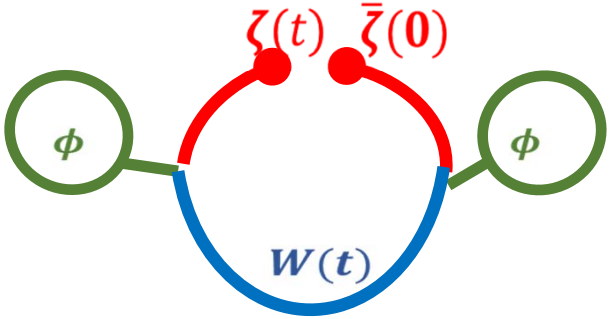




Vlček, J. Chem. Theory Comput. 2017, 13, 10, 4997–5003
Vlček, Phys. Rev. B 98, 075107

StochasticGW

Core Algorithm



Feynman Diagram of StochasticGW

For each CPU core, 1 Monte Carlo (MC) sample:

timeloop: Do t= 0, nt ! nt~2000

! "Propagate" once a matrix: **p(N, M) N~10^7** (grid), **M~10**

do i=1, M

- FFT back and forth p(:,i) ! apply kinetic energy via convolution
- **p(:,i) = p(:,i) + ∑ _a f(:,a)** ∗**sum(**f(:,a)∗p(:,i))

! BLAS to apply non-local potential energy

- Normalize p(:,i)

enddo

! Accumulate and overlap after each step, more BLAS

-Make v(:) from FFT of sum(|p|^2, dim=2) ! wfc * W --> v(r,t), store as v(w)

-From v(:) make many "segments" and store to make $\sum_p$! Get GW energies

! overlap with small basis --> easy to store

end timeloop

StochasticGW with OpenACC

OpenACC because:

- FORTRAN compatibility!
- Ease of learning
- Expert mentors available

Favorite Feature:

- private, copy, etc. simple control over memory copies

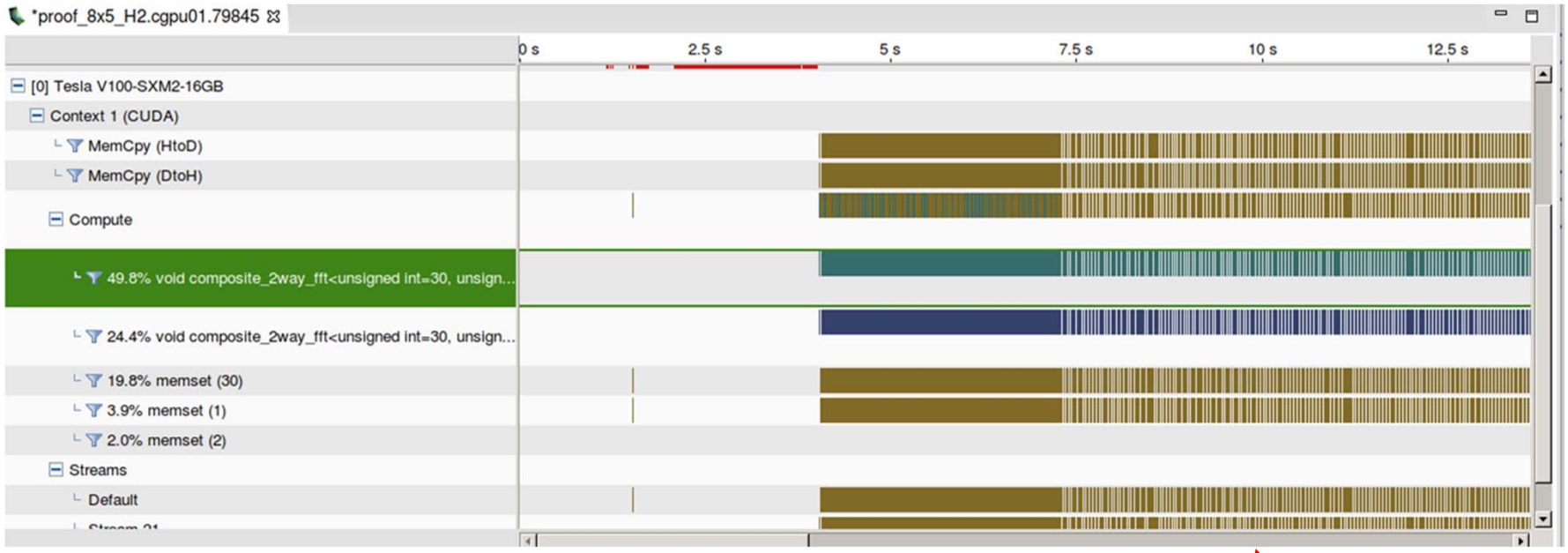
```
lloop : do l= 0,lpptop(ma)
   lif : if(l/=lpploc(ma)) then
      ib=l**2      ! 0,1,4,9
      it=ib+2*l    ! 0,3,8,15
      nbt = it-ib+1
      ! ov=<vphi|vphi> in 3d.
      !$acc loop private(j,js,dot)
      do j=ib,it
         js= ju(l) + (j-ib)

         dot = 0d0
         !$acc loop reduction(+:dot) private(igg)
         do igg=1,ngs(ia)
            dot= dot + vp(js+(igg-1)*nbt)**2
         enddo
         ov = max(dot*dv, toll_o)

         ce=0d0
         !$acc loop reduction(+:ce) private(igg)
         do igg=1,ngs(ia)
            ce= ce +  pt( mapkbg(igg,ia),is)*vp(js+(igg-1)*nbt)
         enddo

         ce = ce * dv /ov
         ce = ce * (exp(-ci*dt/2d0*ov/pvp(l,ma))-1d0)

         do igg=1,ngs(ia)
            !$acc atomic
            qr(mapkbg(igg,ia),is) = &
            qr(mapkbg(igg,ia),is) + &
                 dble(ce)*vp(js+(igg-1)*nbt)
            !$acc atomic
            qi(mapkbg(igg,ia),is) = &
            qi(mapkbg(igg,ia),is) + &
```

OpenACC Goals:

- One code base with all code on the GPU with OpenACC
- Comparisons 1 CPU (Nvidia) versus 1 GPU (Intel Xeon) on NERSC Cori
- Optimize OpenACC use, and Optimize cuFFT calls, and minimize memory copies

Initial Profile from 4/27

fftw --> cuFFT naively, NO OPEN ACC



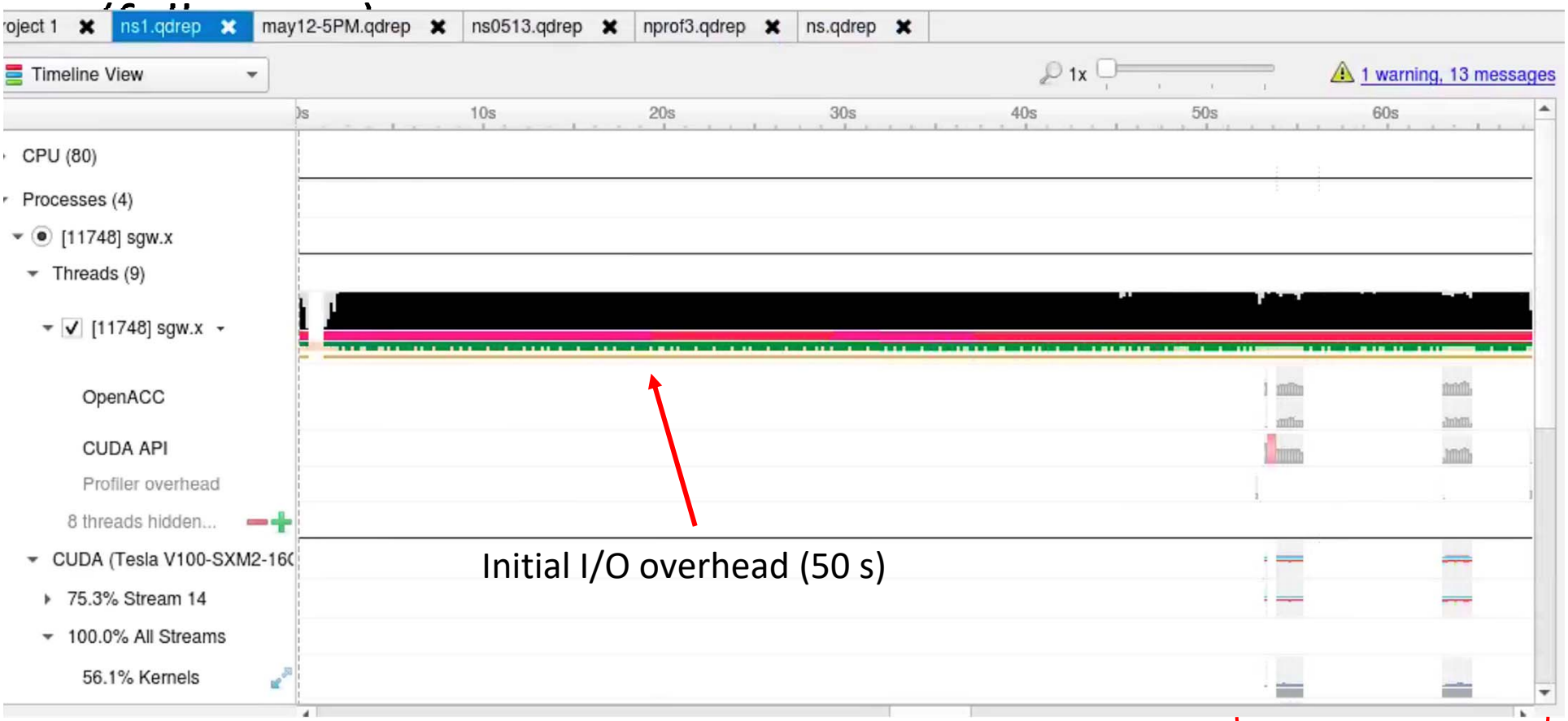
I/O overhead

Dominated by slow memory copies. (white bars |)

No initial speed up observed

(even for small systems)

Overview: 2 MC steps of thousands of C60

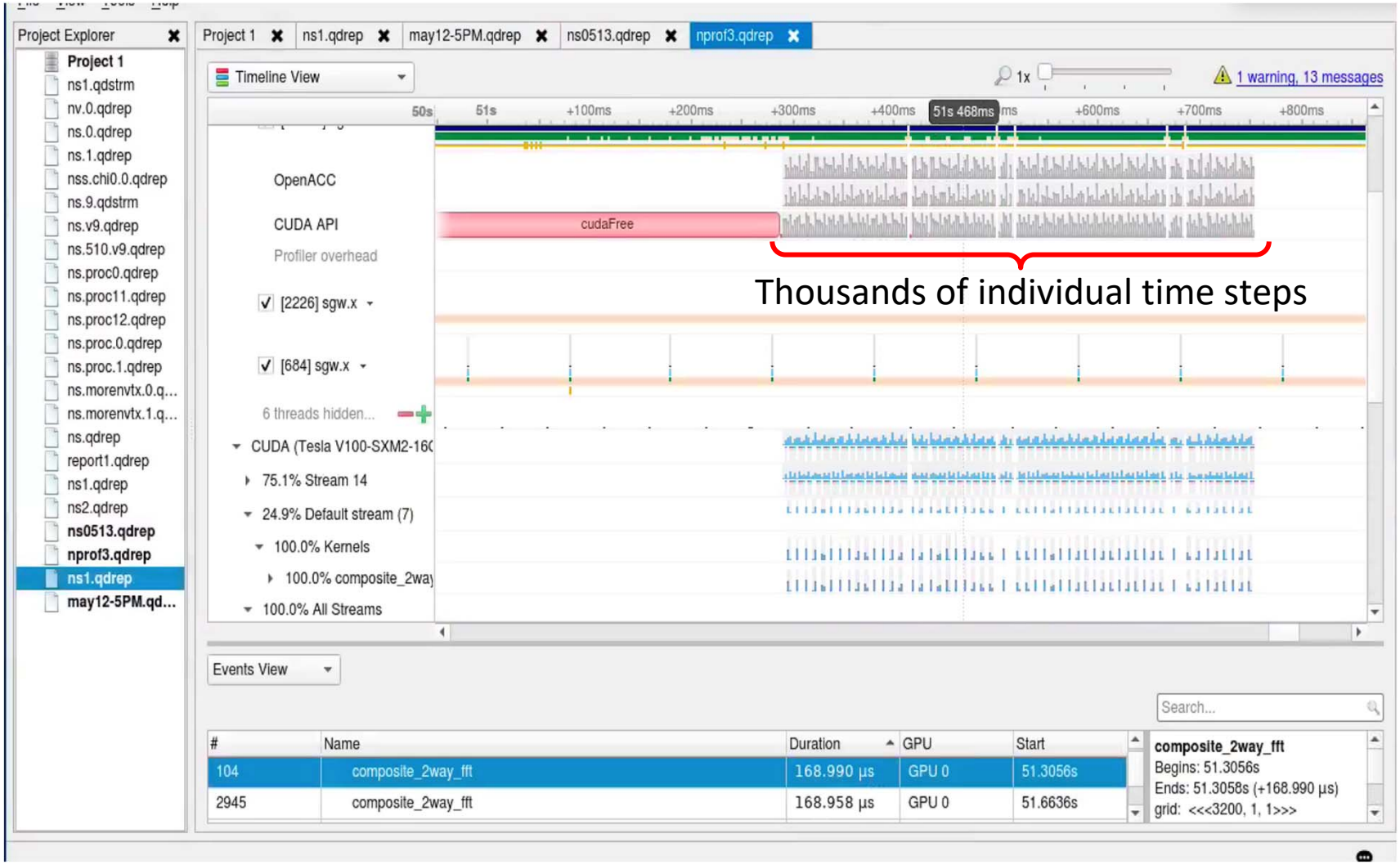


Initial I/O overhead (50 s)

First two MC samples
(~5 sec/step)

Within each Monte Carlo sample

with initial OpenACC and cuFFT kernels running for propogation loop

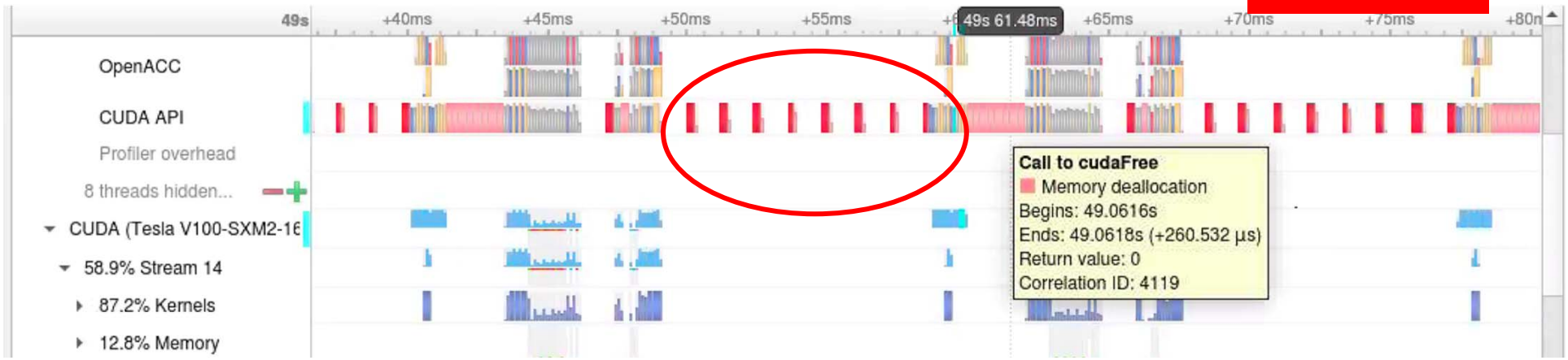


Opt. 1: Removing 3 Memory Copies from inner loop

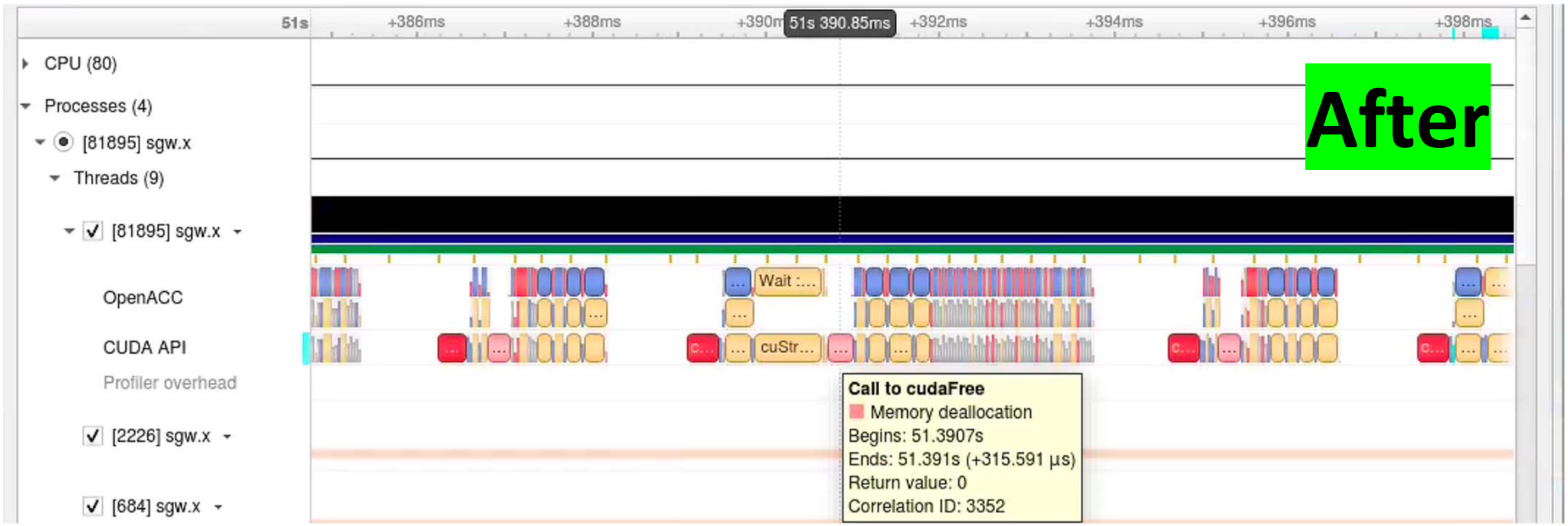
Opt. 2: Move cuFFT async stream --> cuFFT many

(8 cuMalloc/cuFree calls)

Before



After

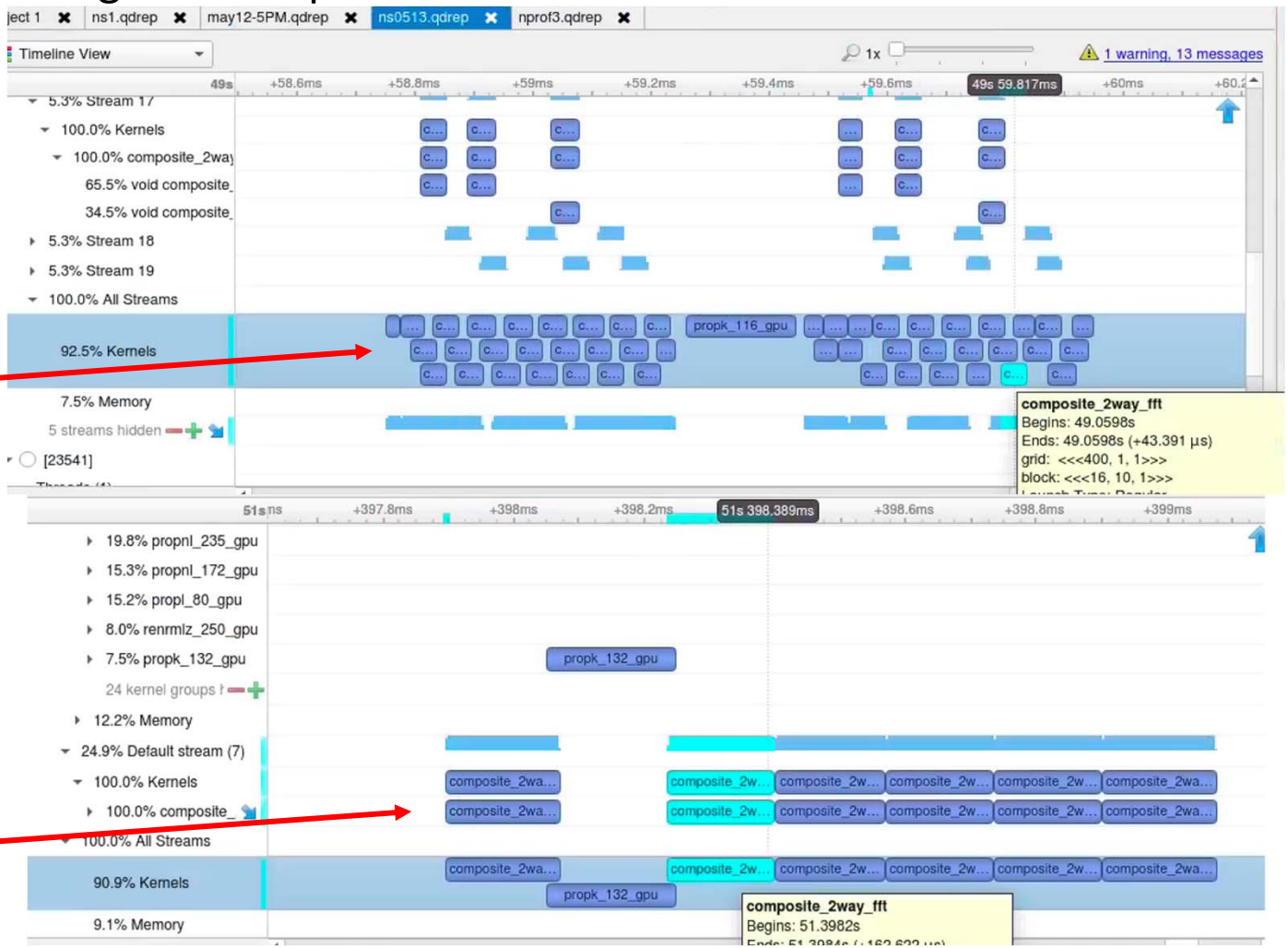


cuFFT with Streams vs cuFFTMany

propk convolution subroutine showed significant speed up using cuFFTMany vs cuFFT in asynchronous data streams.
Called 25x per single MC step.

~2.2 s over whole program

Now ~1.6 s



Open ACC Results and Final Profile

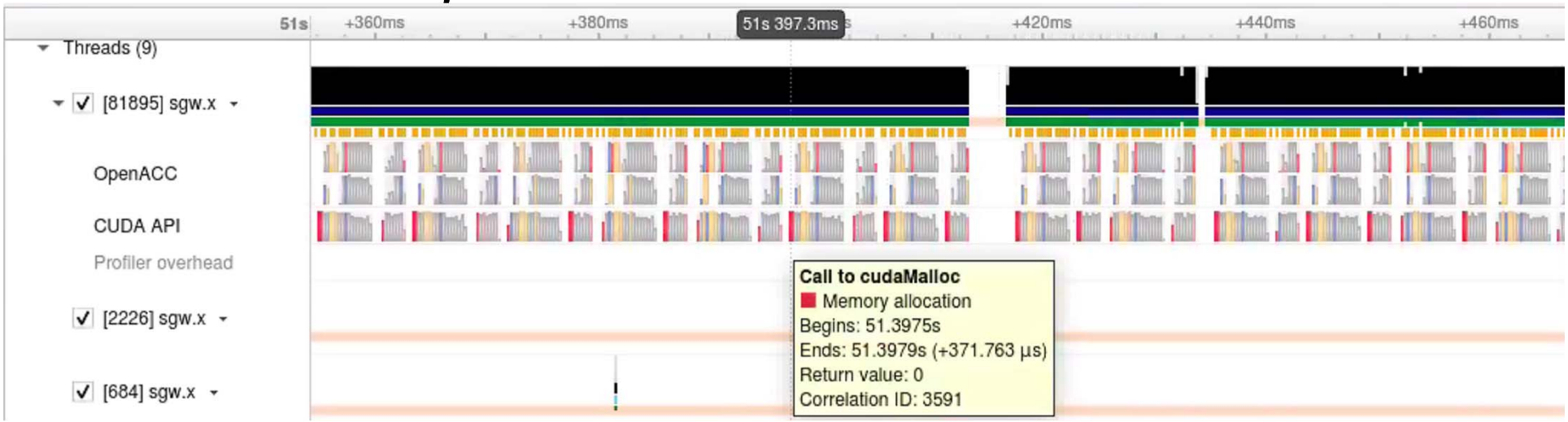
- Accomplishments:
  - OpenACC and cuFFT speedup:
    - 6.3x for whole code without I/O overhead
    - 30x to 40x for individual propogation routines

➔ 1 GPU (Nvidia) vs. 1 CPU (Intel Xeon) on NERSC Cori system

- Combination of GPU computing tools:
  - OpenACC directives
  - PGI compiler, NVIDIA profiling tools
  - cuFFT, cuBLAS

Future Work with **OpenACC**

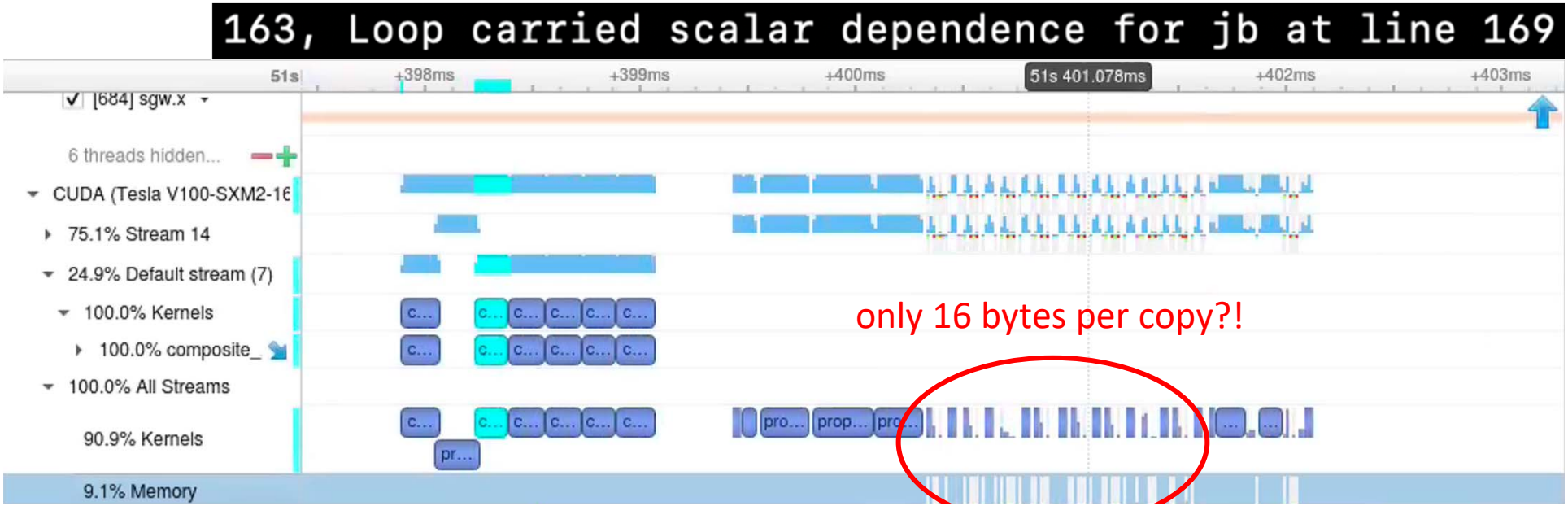
1. Move cuFFT plan creation further out



2. Reduce tiny remaining memory copies (1 double...)

OpenACC Bugs/Issues found

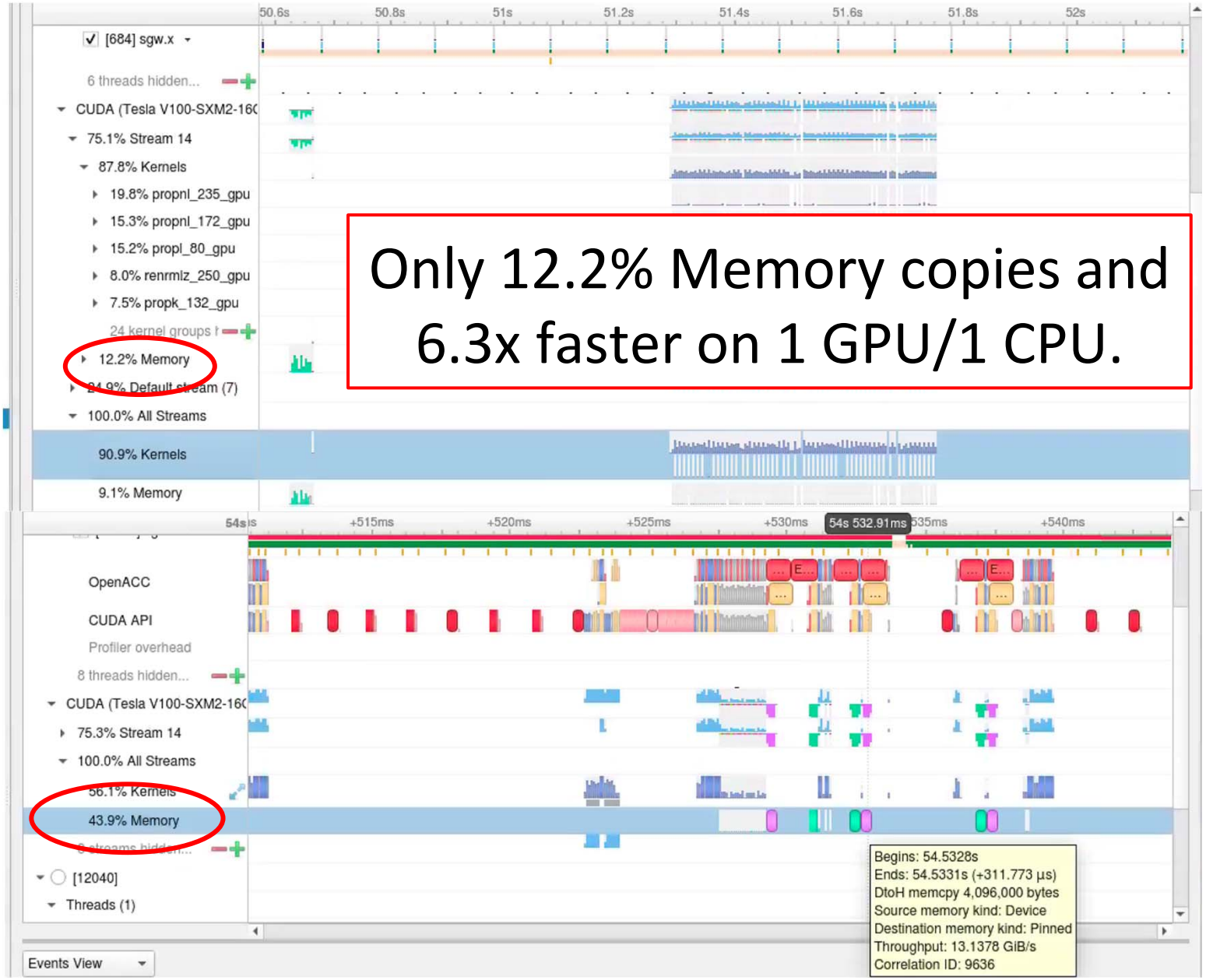
1. Miss-flagging loops as seq
   - Can work around with changing loop structure and explicit data declarations
2. Managed memory for floats/doubles
   - Constant values being perceived by compiler as changing and updating host every too often



Final Comparison (SDSCHack 2020)

Final

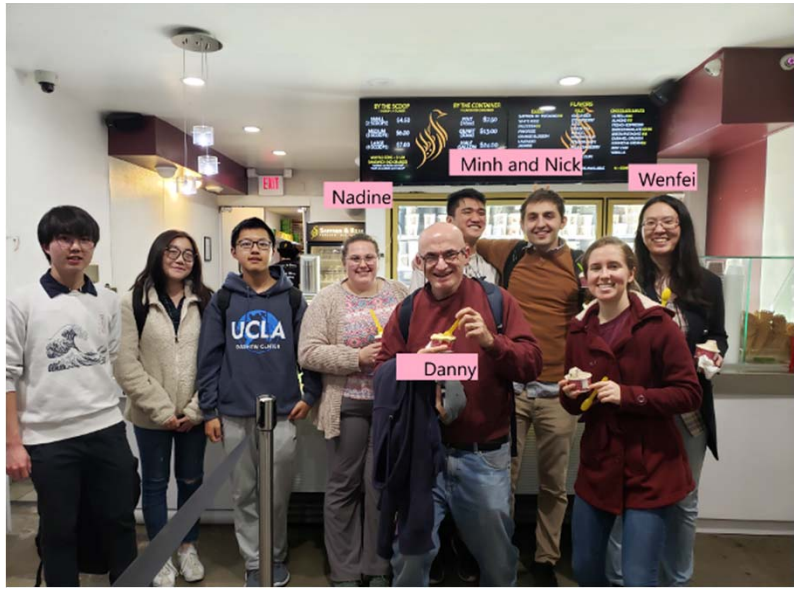
First GPU code



OpenACC Overall

- Breeze to learn!
  - (new coders ~5 days to port 90% of code)
- Simple commands
  - !$ acc loop, acc atomic...
- Good feedback from PGI compiler
  - know when optimization is working or is not
- Compatible with other GPU libraries

```
makerho:
      0, Generating Tesla code
    107, Generating implicit copyout(rho_p(:)) [if not already present]
    108, Loop is parallelizable
         Generating Tesla code
        108, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
    119, Generating copyin(a(:)) [if not already present]
    120, Generating implicit copy(rho_p(1:n)) [if not already present]
         Generating implicit copyin(pt(1:n,1:ns)) [if not already present]
    121, Loop is parallelizable
    123, Generating Tesla code
        121, !$acc loop gang, vector(128) ! blockidx%x threadidx%x
        123, !$acc loop seq
```



Stochastic Donkeys: StochasticGW at SDSC Hack 2020

Team Members:
UCLA: Daniel Neuhauser,
Nadine Bradbury,
Wenfei Li,
Minh Nguyen,
Mykola (Nick) Sereda

LBL:
Jack DeSlippe,
Mauro Del Ben,
Chao Yang

Mentors:
LBL: Charlene Yang
Nvidia: Brent Leback
SDSC: Mahidhar Tatineni