NVIDIA'S SOFTWARE ECOSYSTEM FOR HPC

Max Katz

May 5, 2021

NVIDIA HPC SDK

DEVELOPMENT

Programming Models
- Standard C++ & Fortran
- OpenACC & OpenMP
- CUDA

Compilers
- nvcc
- nvc
- nvc++
- nvfortran

Core Libraries
- libcu++
- Thrust
- CUB

Math Libraries
- cuBLAS
- cuTENSOR
- cuSPARSE
- cuSOLVER
- cuFFT
- cuRAND

Communication Libraries
- HPC-X
  - MPI
  - UCX
  - SHMEM
  - SHARP
  - HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
  - Systems
  - Compute

Debugger
- cuda-gdb
  - Host
  - Device

Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
7-8 Releases Per Year | Freely Available

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU and Network

Accelerated Standard Languages

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){ return y + a*x; }
);
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
import legate.numpy as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

Incremental Portable Optimization

```
#pragma acc data copy(x,y)
{

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});

...

}
```

Platform Specialization

```
__global__
void saxpy(int n, float a,
            float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

Core	Math	Communication	Data Analytics	AI

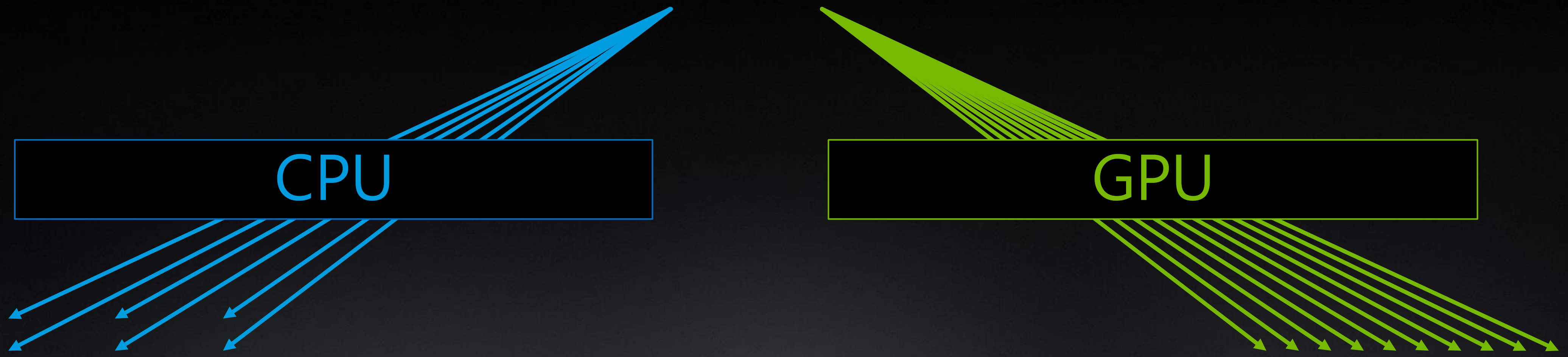
Acceleration Libraries

ACCELERATED STANDARDS

Parallel performance for wherever you code needs to run

```
std::transform(std::execution::par, x, x+n, y, y,
  [=] (auto xi, auto yi) { return y + a*xi; });
```

```
do concurrent (i = 1:n)
  y(i) = y(i) + a*x(i)
enddo
```




```
nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
```

```
nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
```

ACCELERATED PROGRAMMING IN ISO FORTRAN

NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c
...
!$acc enter data copyin(a,b,c) create(d)

do nt = 1, ntimes
  !$acc kernels
  do j = 1, nj
    do i = 1, ni
      d(i,j) = c(i,j)
      do k = 1, nk
        d(i,j) = d(i,j) + a(i,k) * b(k,j)
      end do
    end do
  end do
    !$acc end kernels
end do

!$acc exit data copyout(d)
```

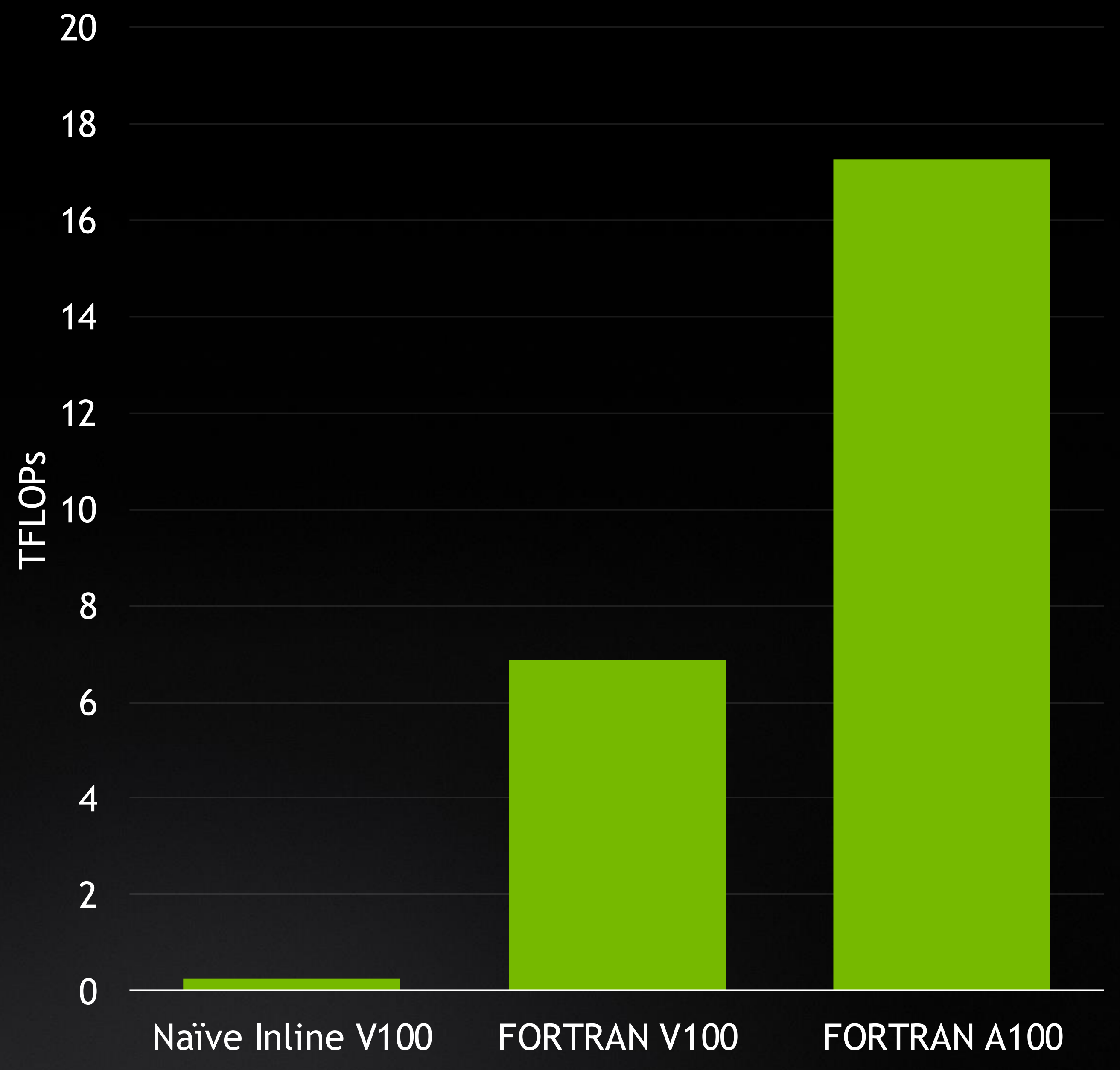
Inline FP64 matrix multiply

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c

...

do nt = 1, ntimes
  d = c + matmul(a,b)
end do
```

MATMUL FP64 matrix multiply



NVIDIA

HPC PROGRAMMING IN ISO FORTRAN

Examples of Patterns Accelerated in NVFORTRAN

```
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(transpose(b))
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(b)
d = reshape(a,shape=[ni,nj,nk])
d = reshape(a,shape=[ni,nk,nj])
d = 2.5 * sqrt(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = alpha * conjg(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = reshape(a,shape=[ni,nk,nj],order=[1,3,2])
d = reshape(a,shape=[nk,ni,nj],order=[2,3,1])
d = reshape(a,shape=[ni*nj,nk])
d = reshape(a,shape=[nk,ni*nj],order=[2,1])
d = reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1])
d = abs(reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1]))
c = matmul(a,b)
c = matmul(transpose(a),b)
c = matmul(reshape(a,shape=[m,k],order=[2,1]),b)
c = matmul(a,transpose(b))
c = matmul(a,reshape(b,shape=[k,n],order=[2,1]))
```

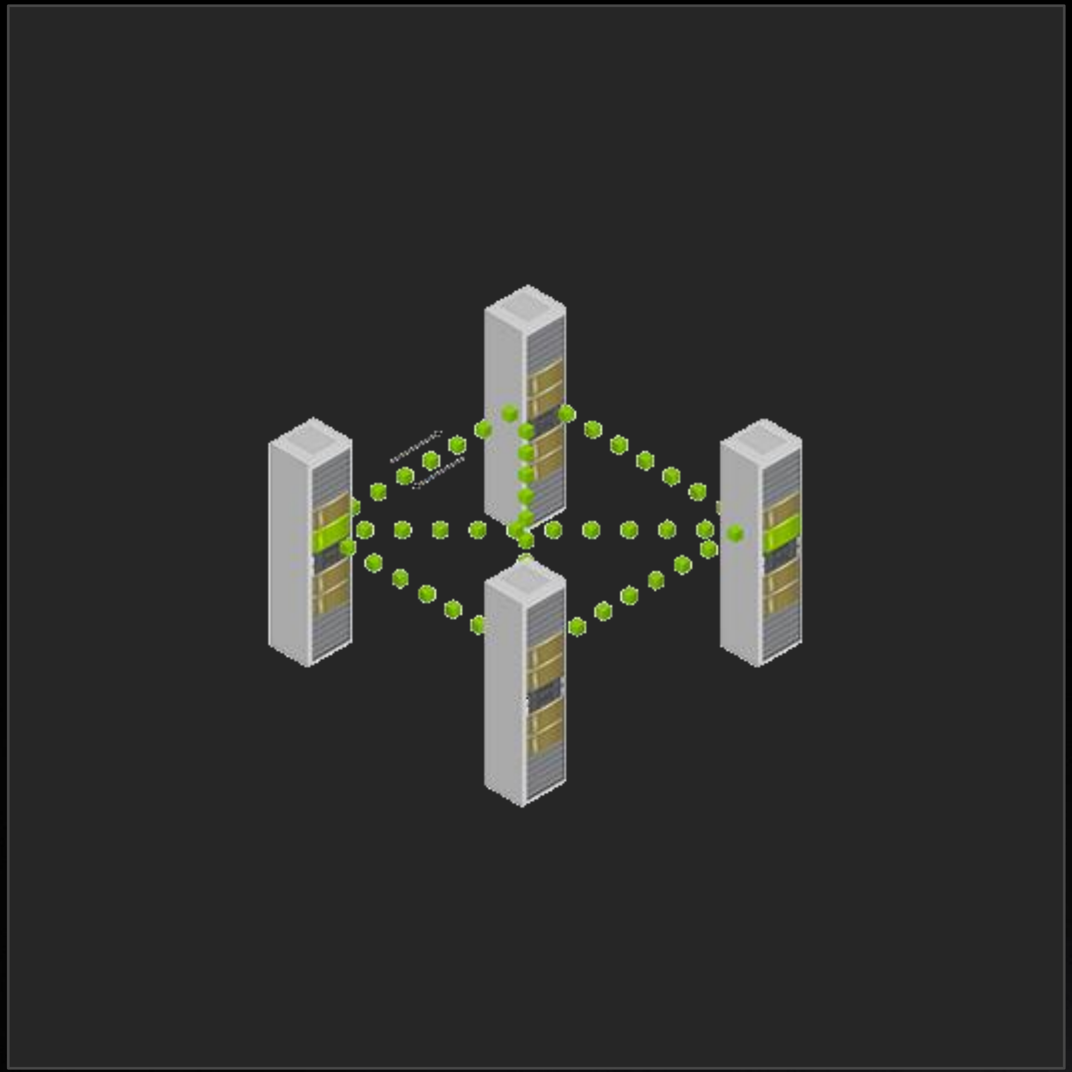
```
c = matmul(transpose(a),transpose(b))
c = matmul(transpose(a),reshape(b,shape=[k,n],order=[2,1]))
d = spread(a,dim=3,ncopies=nk)
d = spread(a,dim=1,ncopies=ni)
d = spread(a,dim=2,ncopies=nx)
d = alpha * abs(spread(a,dim=2,ncopies=nx))
d = alpha * spread(a,dim=2,ncopies=nx)
d = abs(spread(a,dim=2,ncopies=nx))
d = transpose(a)
d = alpha * transpose(a)
d = alpha * ceil(transpose(a))
d = alpha * conjg(transpose(a))
c = c + matmul(a,b)
c = c - matmul(a,b)
c = c + alpha * matmul(a,b)
d = alpha * matmul(a,b) + c
d = alpha * matmul(a,b) + beta * c
```

NVIDIA PERFORMANCE LIBRARIES

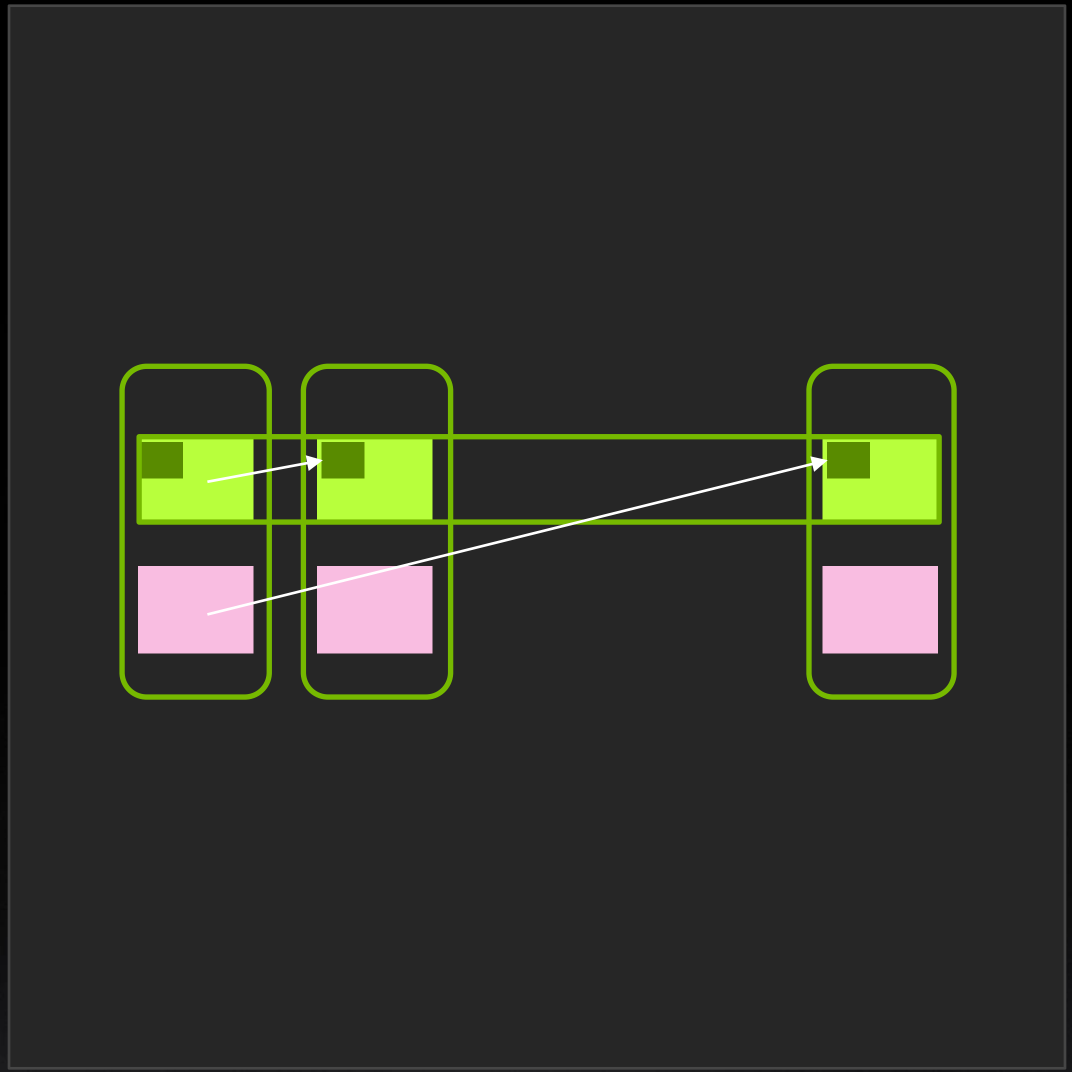
Core and Math

Core	Math
libcu++	cuBLAS
Thrust	cuSPARSE
CUB	cuTENSOR
	cuSOLVER
	cuRAND
	cuFFT
	CUDA Math API

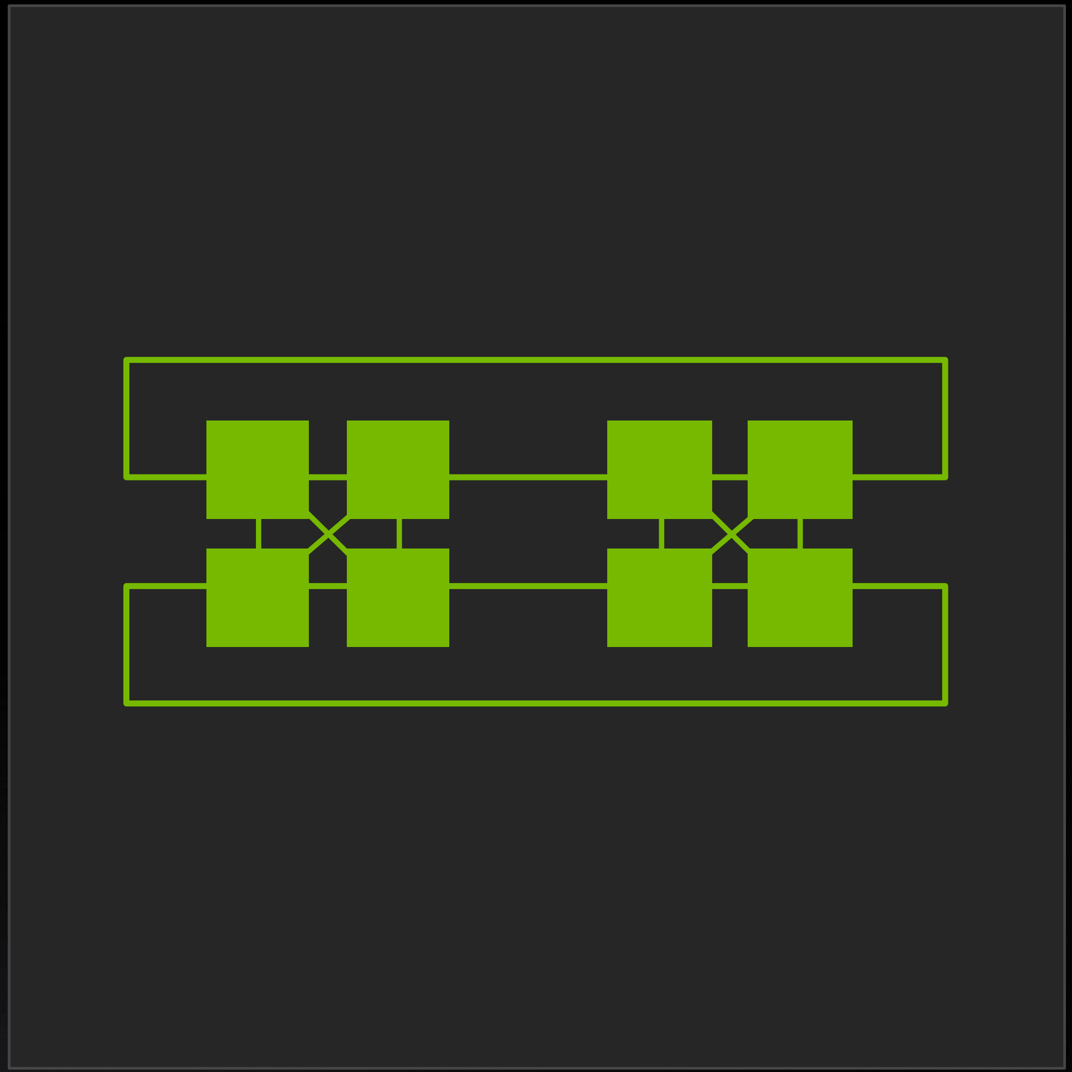
NVIDIA COMMUNICATION LIBRARIES



HPC-X
Optimized whole-system communications



NVSHMEM
Low-latency PGAS programming



NCCL
Multi-node collectives for accelerators

INTRODUCING LEGATE

Accelerated and Distributed

A framework for programming large numbers of GPUs as if they were a single processor

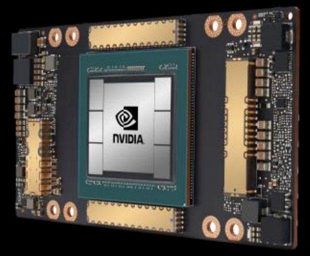
Pass data between Legate libraries without worrying about distribution or synchronization requirements

Legate NumPy and Pandas aim to transparently scale existing Numpy and Pandas workloads

Legate Numpy and Legate Pandas available now and opensource!

```
import legate.numpy as np

def cg_solve(A, b, tol=1e-10):
    x = np.zeros(A.shape[1])
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    for i in xrange(b.shape[0]):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)
        if np.sqrt(rsnew) < tol:
          break
        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
        return x
```



LEGATE NUMPY

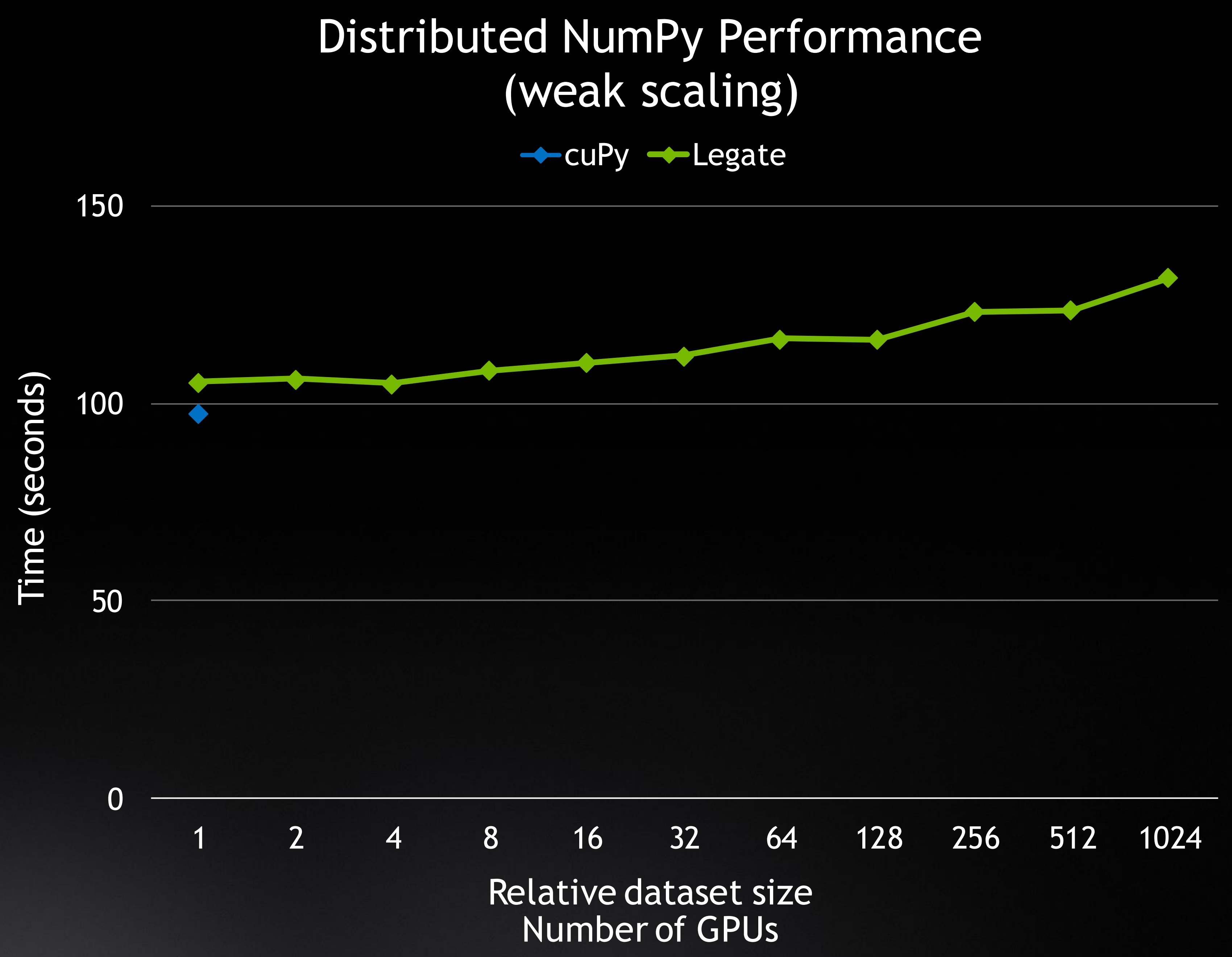
Results from "CFD Python"
https://github.com/barbagroup/CFDPython

```
import legate.numpy as np

for _ in range(iter):
    un = u.copy()

    vn = v.copy()
    b = build_up_b(rho, dt, dx, dy, u, v)
    p = pressure_poisson_periodic(b, nit, p, dx, dy)

    u[1:-1, 1:-1] = (
        un[1:-1, 1:-1]
        - un[1:-1, 1:-1] * dt / dx * (un[1:-1, 1:-1] - un[1:-1, 0:-2])
        - vn[1:-1, 1:-1] * dt / dy * (un[1:-1, 1:-1] - un[0:-2, 1:-1])
        - dt / (2 * rho * dx) * (p[1:-1, 2:] - p[1:-1, 0:-2])
        + nu
        * (
            dt
             / dx ** 2
            * (un[1:-1, 2:] - 2 * un[1:-1, 1:-1] + un[1:-1, 0:-2])
            + dt
             / dy ** 2
            * (un[2:, 1:-1] - 2 * un[1:-1, 1:-1] + un[0:-2, 1:-1])
        )
        + F * dt
    )
```



Extracted from "CFD Python" course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, 1(9), 21, https://doi.org/10.21105/jose.00021

LEGATE PANDAS

Pandas join micro-benchmark – 300M rows/GPU

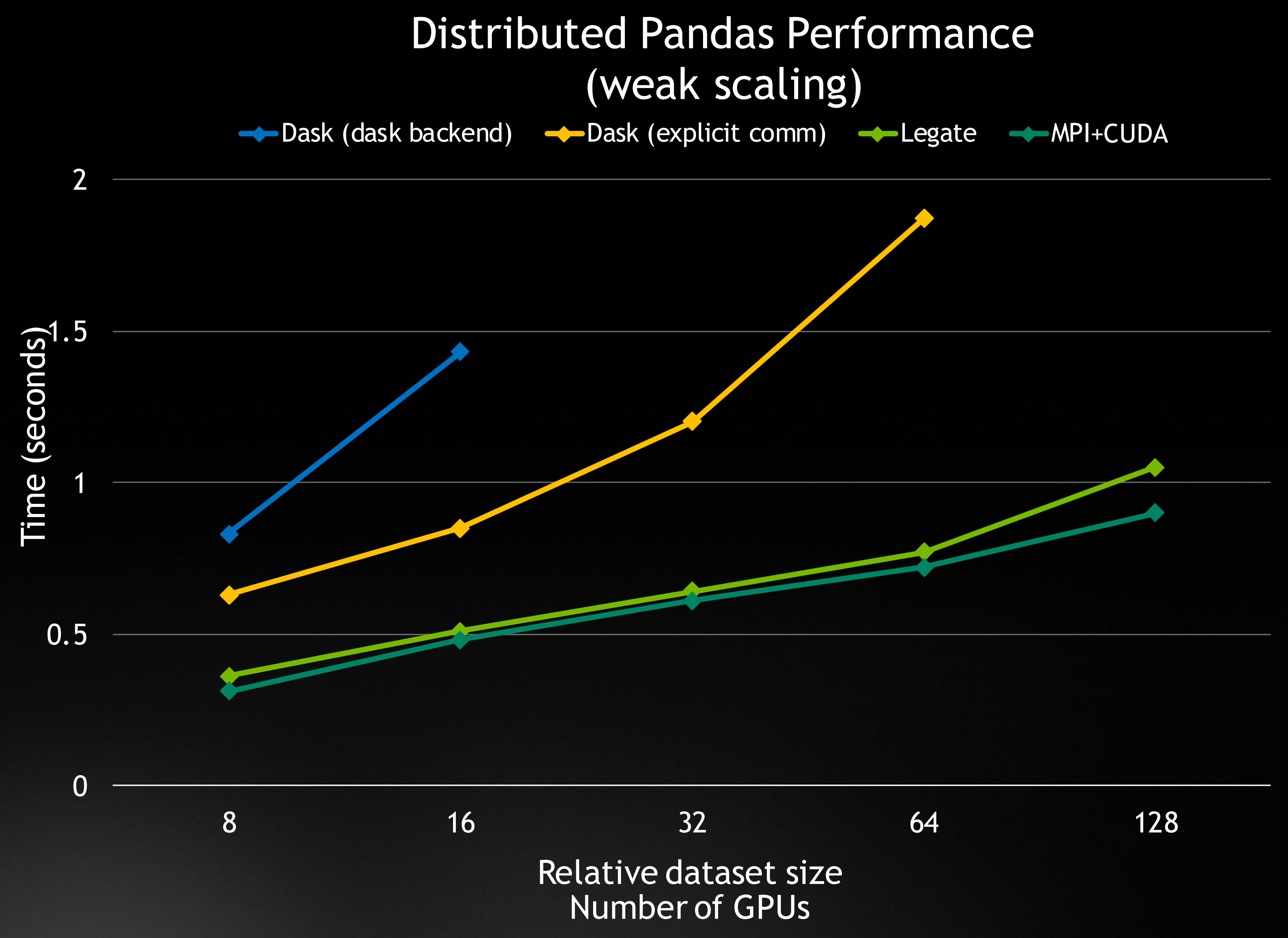
```
import legate.numpy as np
import legate.pandas as pd

size = num_rows_per_gpu * num_gpus

key_l = np.arange(size)
payload_l = np.random.randn(size) * 100.0
lhs = pd.DataFrame({"key": key_l, "payload": payload_l})

key_r = key_l // 3 * 3    # selectivity: 0.33
payload_r = np.random.randn(size) * 100.0
rhs = pd.DataFrame({"key": key_r, "payload": payload_r})

out = lhs.merge(rhs, on="key")
```



NSIGHT PRODUCT FAMILY

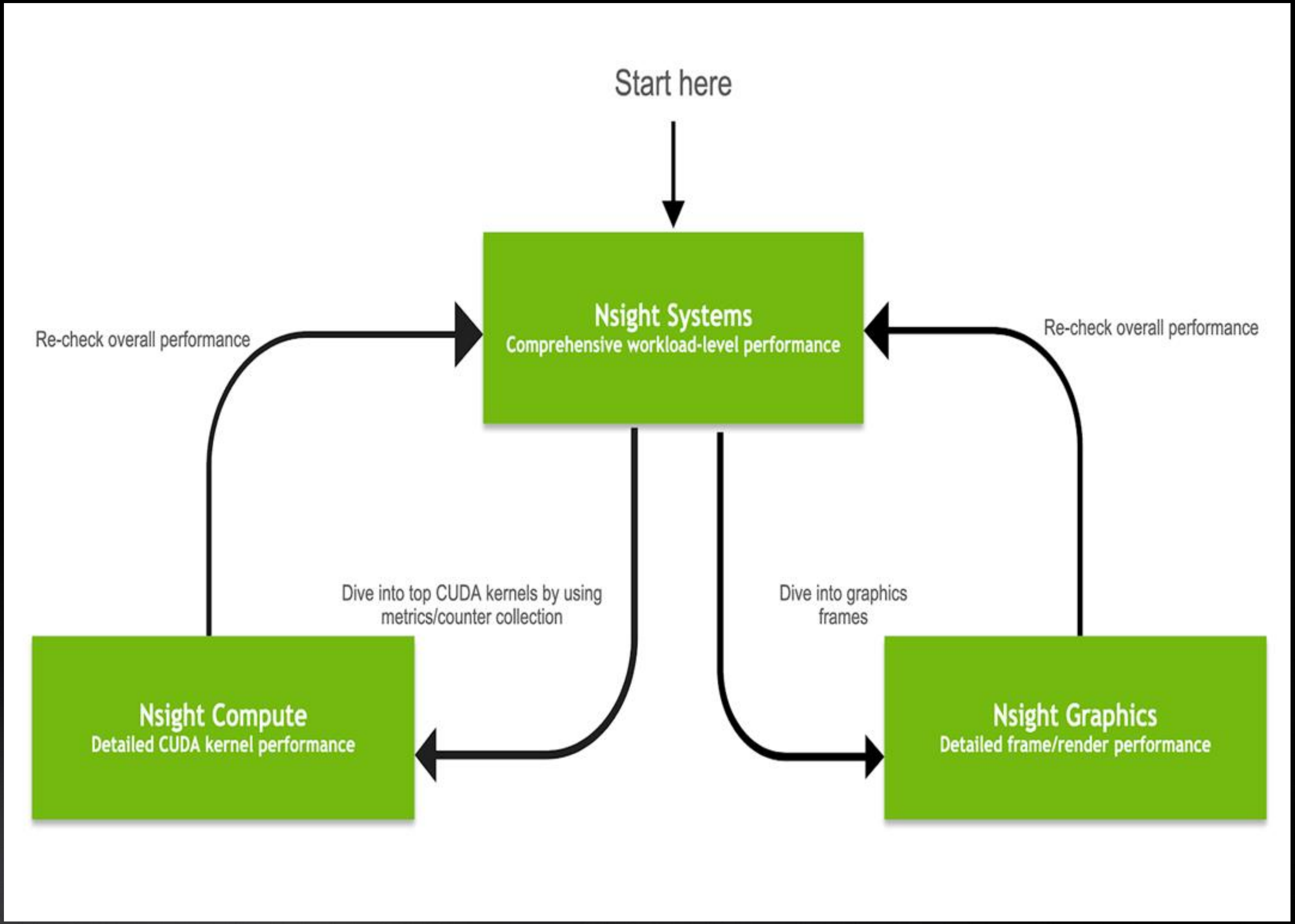
Nsight Systems - Analyze application performance (system-wide)

Nsight Compute - Optimize CUDA kernel

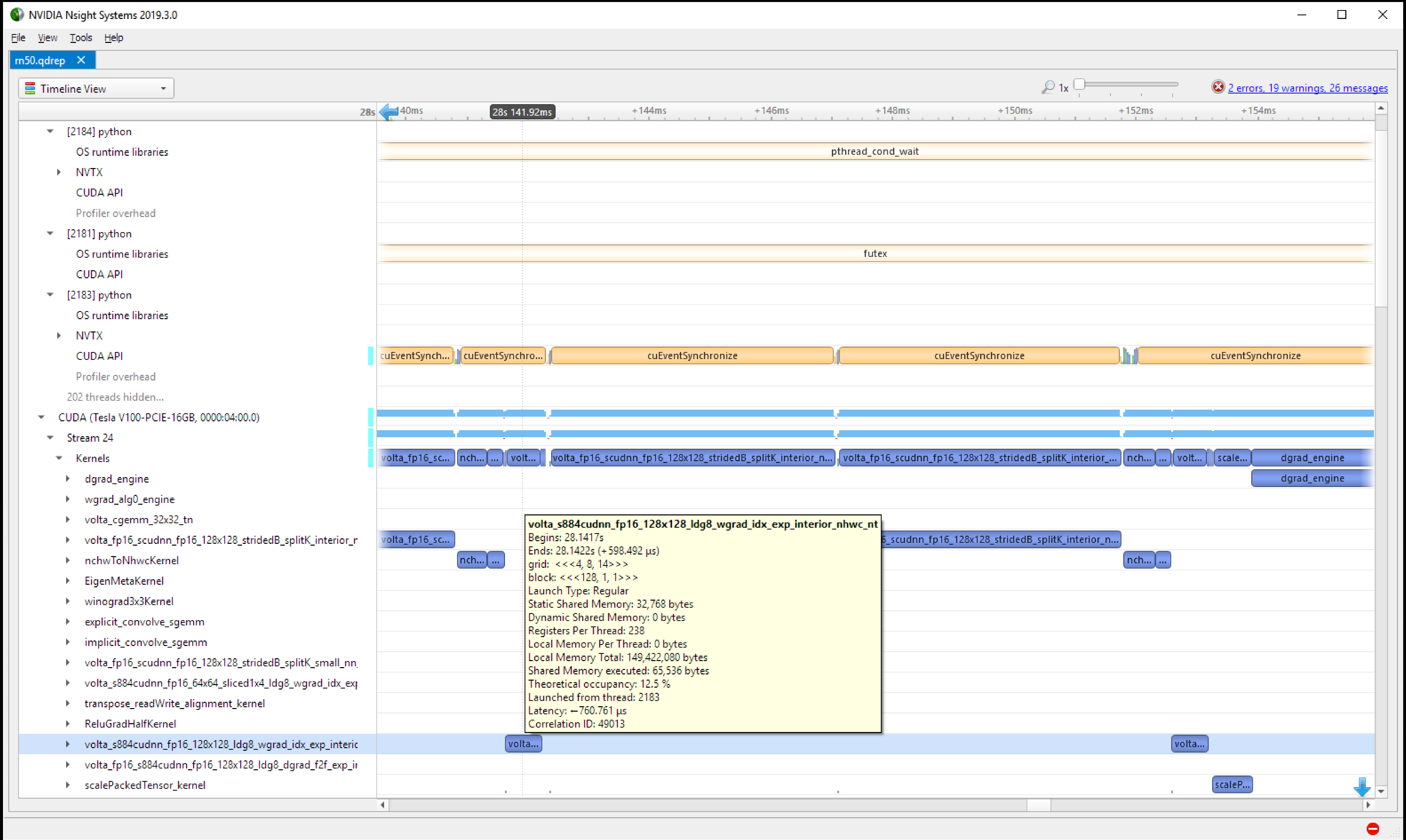
Other developer tools:

Compute Sanitizer: memory debugging similar to valgrind for GPUs

cuda-gdb: CUDA-aware extension of gdb




NVIDIA Nsight Systems 2018.0.0
File View Help
Select device for profiling...
More info...
aggcpy-yeast-CUDA.qdrep
Timeline View
1 warning, 5 messages
System
NVTX
CUDA API
[1999] lm
[2002] lm
[1998] lm
[1987] lm
11 threads hidden...
CUDA (Tesla V100-SXM2-32GB)
Stream 70
Memory
Kernels
1 stream(s) hidden...
MPD_TIMESTEP [524.355 µs]
MCLKR Barrier [194.213 µs]
MPD_TIMESTEP [475.814 µs]
MPD_TIMESTEP [466.524 µs]
MPD_TIMESTEP [446.305 µs]
MPD_TIMESTEP [452.747 µs]
MCLKR Barrier [193.247 µs]
MPD_TIMESTEP [526.627 µs]
MPD_TIMESTEP [540.467 µs]
MPD_TIMESTEP [573.056 µs]
MPD_TIMESTEP [555.258 µs]
MPD_TIMESTEP [578.180 µs]
MPD_TIMESTEP [489.153 µs]
MPD_TIMESTEP [466.427 µs]
MPD_TIMESTEP [583.928 µs]
MPD_TIMESTEP [466.209 µs]
MGPU_x_kernel_unpack
MGPU_y_kernel
cudaMemcpyAs...
Waiting
Duration: 36.206 µs
Call stack at 3.639s:
libpthread-2.23.so!__pthread_mutex_lock
libcuda.so.384.125!cuMemcpyAsync
libcudart.so.9.0.176!cudaMemcpyAsync
lm!lm::rdme::MGPUMpdRdmeSolver::run_next_timestep(...)
lm!lm::rdme::MGPUMpdRdmeSolver::run_thread(...)
libpthread-2.23.so!start_thread
libc-2.23.so!__clone
Bottom-Up View
Process [1979] lm (10 of 21 threads)
Filter...
99.01% of data is shown due to applied filters.
Search...
Symbol Name | Self, % | Module Name
lm::rdme::MGPUMpdRdmeSolver::run_thread(int) | 38.59 | /opt/lm/bin/lm
0x7fd568442726 | 8.26 | /usr/lib/x86_64-linux-gnu/libcuda.so.384.125
0x7fd568373448 | 2.97 | /usr/lib/x86_64-linux-gnu/libcuda.so.384.125
0x7fd568373436 | 2.25 | /usr/lib/x86_64-linux-gnu/libcuda.so.384.125



NVIDIA Nsight Systems 2019.3.0
File View Tools Help
rn50.qdrep
Timeline View
1x
2 errors, 19 warnings, 26 messages
28s
40ms
28s 141.92ms
+144ms
+146ms
+148ms
+150ms
+152ms
+154ms
[2184] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
pthread_cond_wait
[2181] python
OS runtime libraries
CUDA API
futex
[2183] python
OS runtime libraries
NVTX
CUDA API
Profiler overhead
cuEventSynch...
cuEventSynchro...
cuEventSynchronize
202 threads hidden...
CUDA (Tesla V100-PCIE-16GB, 0000:04:00.0)
Stream 24
Kernels
volta_fp16_sc...
nch...
volt...
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_n...
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_...
scale...
dgrad_engine
wgrad_alg0_engine
volta_cgemm_32x32_tn
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_interior_r
nchwToNhwcKernel
EigenMetaKernel
winograd3x3Kernel
explicit_convolve_sgemm
implicit_convolve_sgemm
volta_fp16_scudnn_fp16_128x128_stridedB_splitK_small_nn
volta_s884cudnn_fp16_64x64_sliced1x4_ldg8_wgrad_idx_exp
transpose_readWrite_alignment_kernel
ReluGradHalfKernel
volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interic
volta_fp16_s884cudnn_fp16_128x128_ldg8_dgrad_f2f_exp_ir
scalePackedTensor_kernel
volta...
scaleP...
volta_s884cudnn_fp16_128x128_ldg8_wgrad_idx_exp_interior_nhwc_nt
Begins: 28.1417s
Ends: 28.1422s (+598.492 µs)
grid: <<<4, 8, 14>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 32,768 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 238
Local Memory Per Thread: 0 bytes
Local Memory Total: 149,422,080 bytes
Shared Memory executed: 65,536 bytes
Theoretical occupancy: 12.5 %
Launched from thread: 2183
Latency: ←760.761 µs
Correlation ID: 49013

NVTX

NVIDIA Tools Extension for Profiling

NVTX can be used to manually instrument applications, for example in C:

```
nvtxRangePush("region_name");

nvtxRangePop();
```

These names are then shown on the nsys timeline (can also be used with ncu)

Headers provided with CUDA toolkit for C/C++; can also be used with Fortran, generic Python (e.g. provided by CuPy), TensorFlow, and PyTorch

COLLECT A PROFILE WITH NSIGHT SYSTEMS

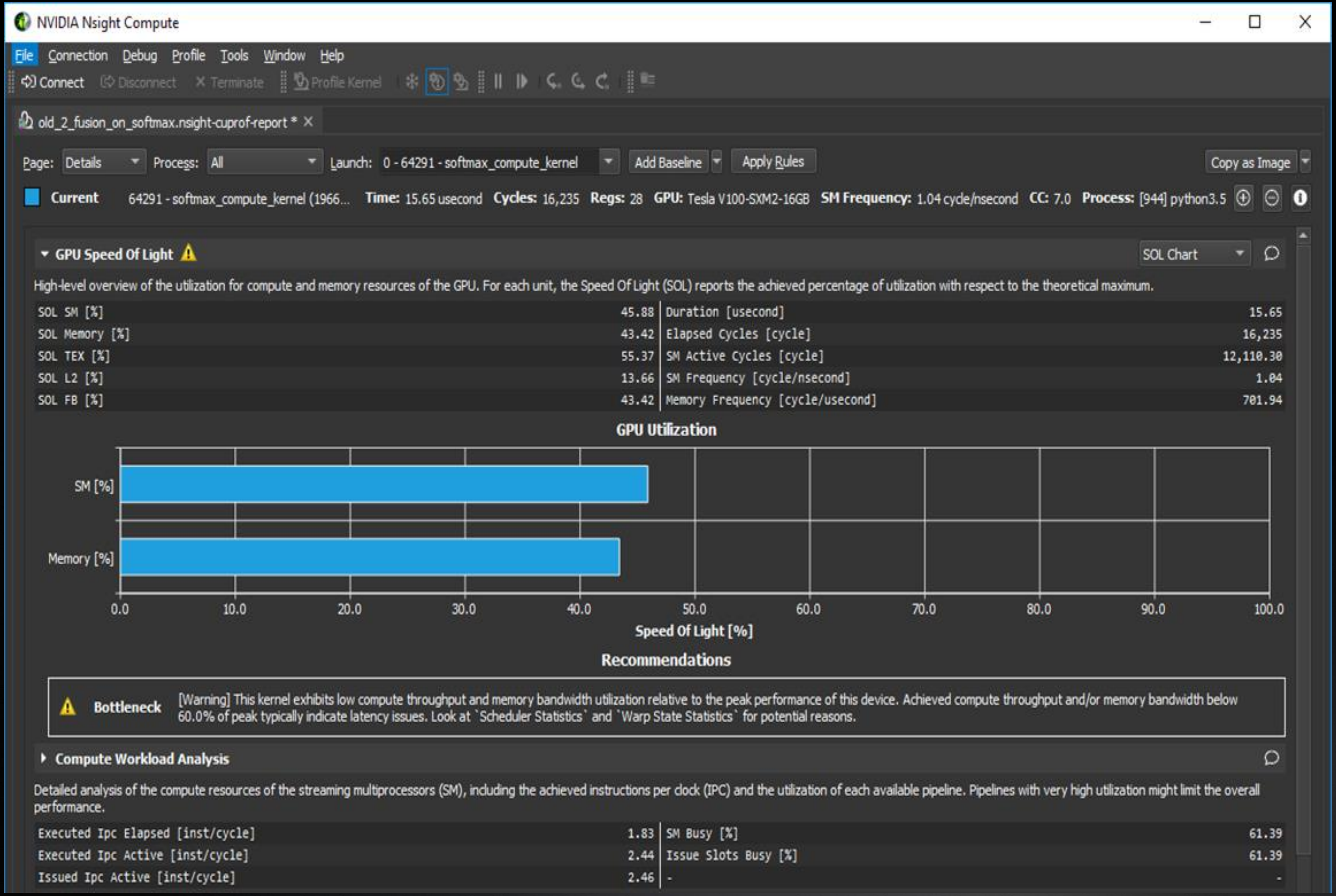
```
$ nsys profile -o report --stats=true ./myapp.exe
```

Generated file: `report.qdrep;` open for viewing in the Nsight Systems UI

Can be done inside a container if the container has `nsys`:

```
$ mpirun -n 4 singularity run --nv -B $CONTAINER_IMG nsys profile python train.py
```

NSIGHT COMPUTE



CUDA kernel profiler

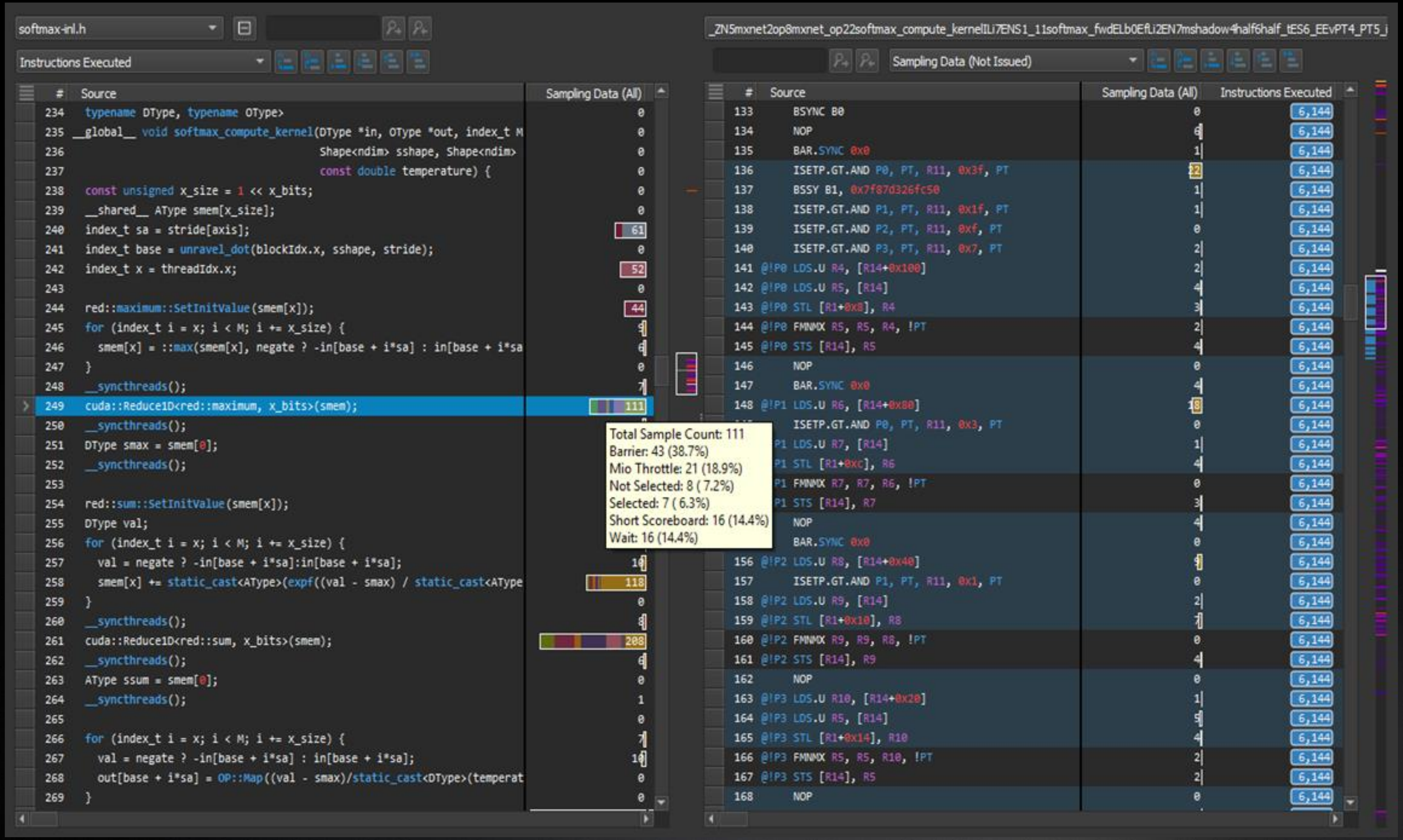
Targeted metric sections for various performance aspects

Customizable data collection and presentation (tables, charts, …)

UI and Command Line

Python-based rules for guided analysis (or post-processing)

NSIGHT COMPUTE



Source/PTX/SASS analysis and correlation

Source metrics per instruction and aggregated (e.g. PC sampling data)

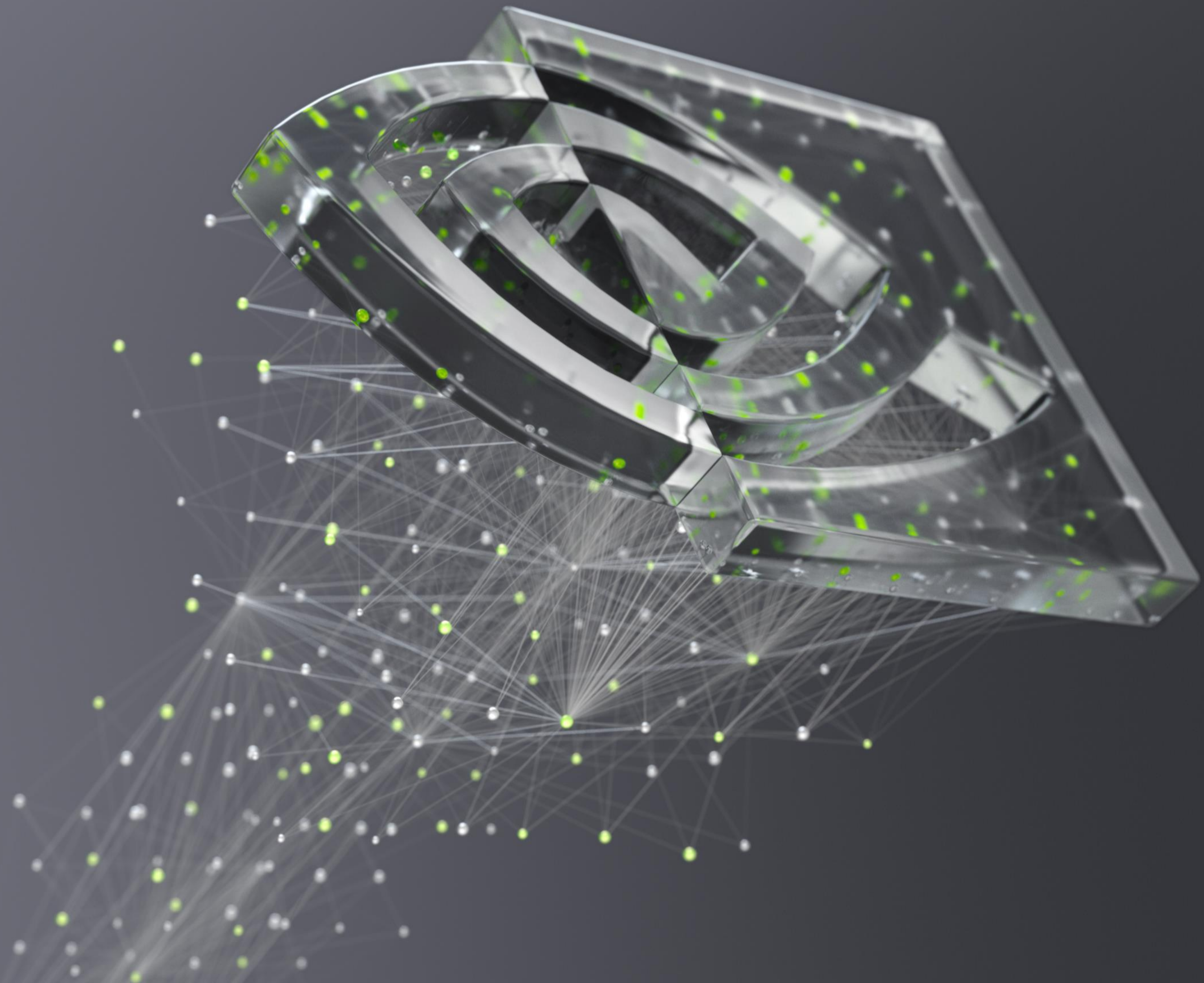
Metric heatmap

KERNEL PROFILES WITH NSIGHT COMPUTE

```
$ ncu -k mykernel -o report ./myapp.exe
```

Generated file: `report.ncu-rep;` open for viewing in the Nsight Compute UI

(Without the –k option, Nsight Compute will profile everything and take a long time!)



NVIDIA