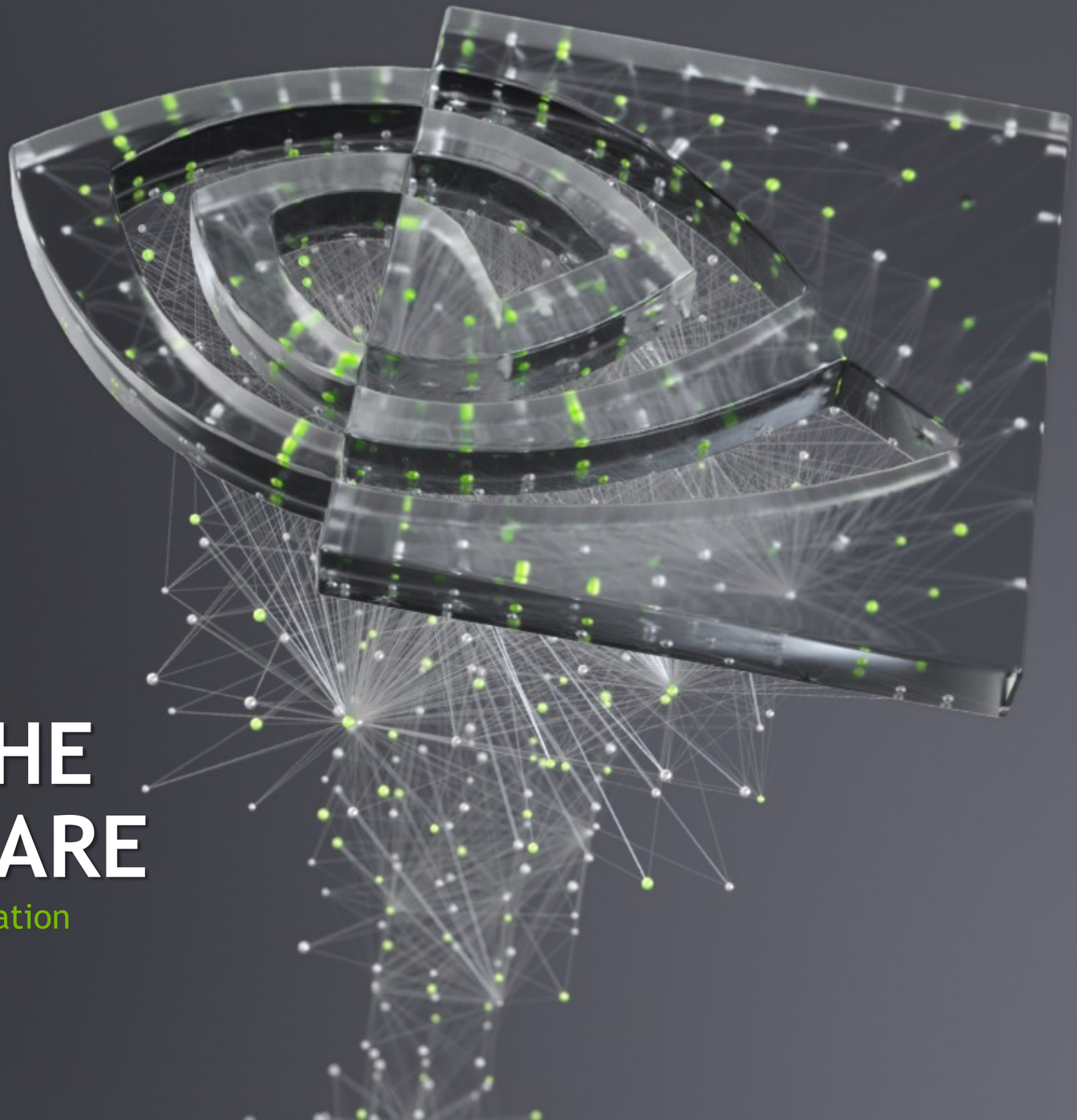


A DEEP DIVE INTO THE LATEST HPC SOFTWARE

Robbie Searles | Solutions Architect, NVIDIA Corporation

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

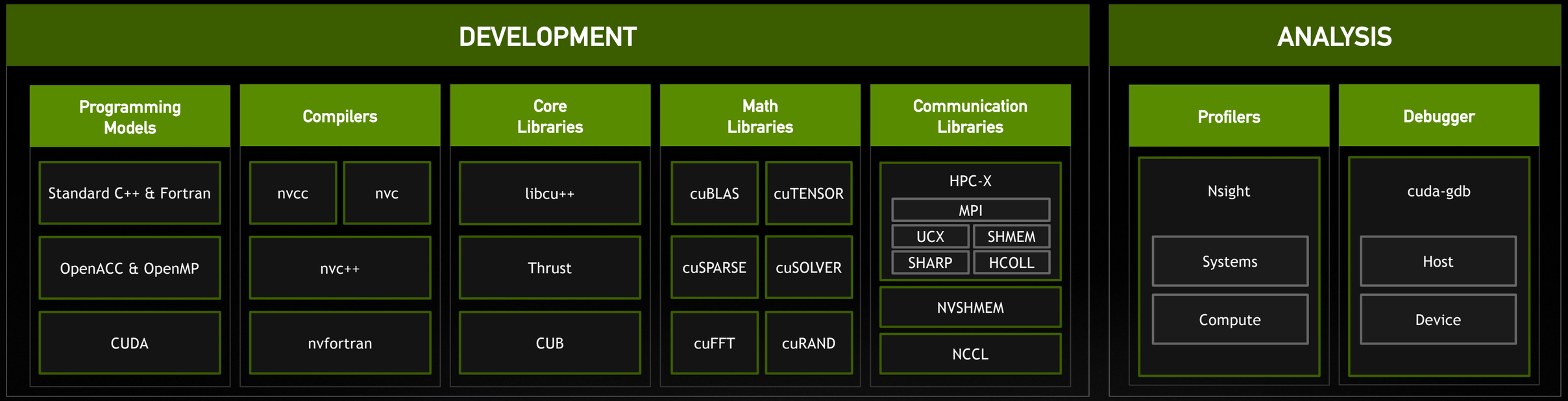
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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	Quantum

Acceleration Libraries

NVIDIA HPC SDK

Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud



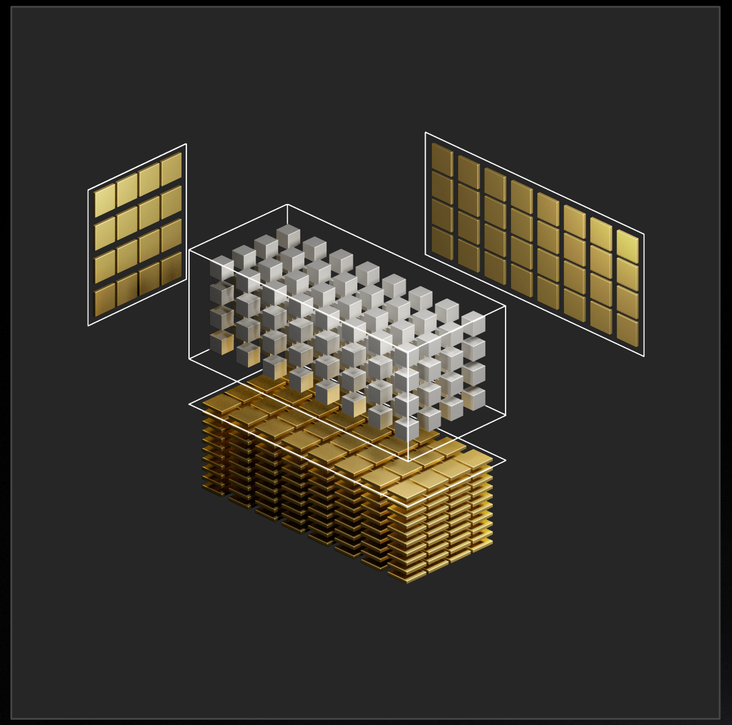
Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

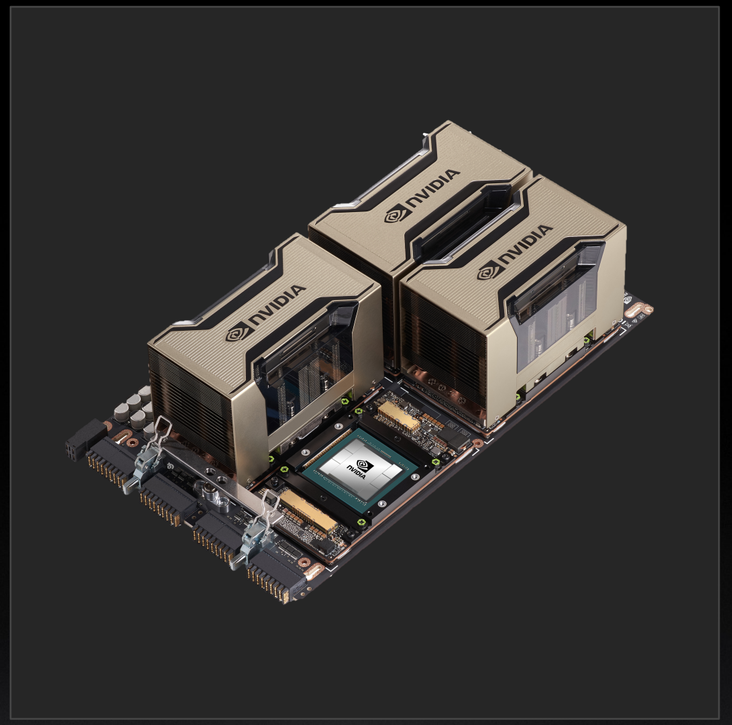
7-8 Releases Per Year | Freely Available

NVIDIA PERFORMANCE LIBRARIES

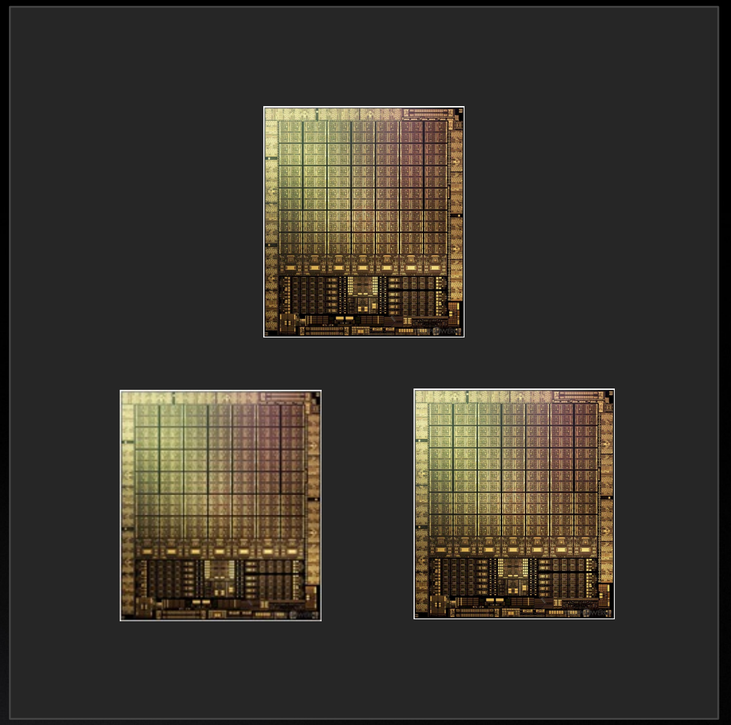
Math Library Directions



Seamless Acceleration
Tensor Cores, Enhanced L2$ & SMEM



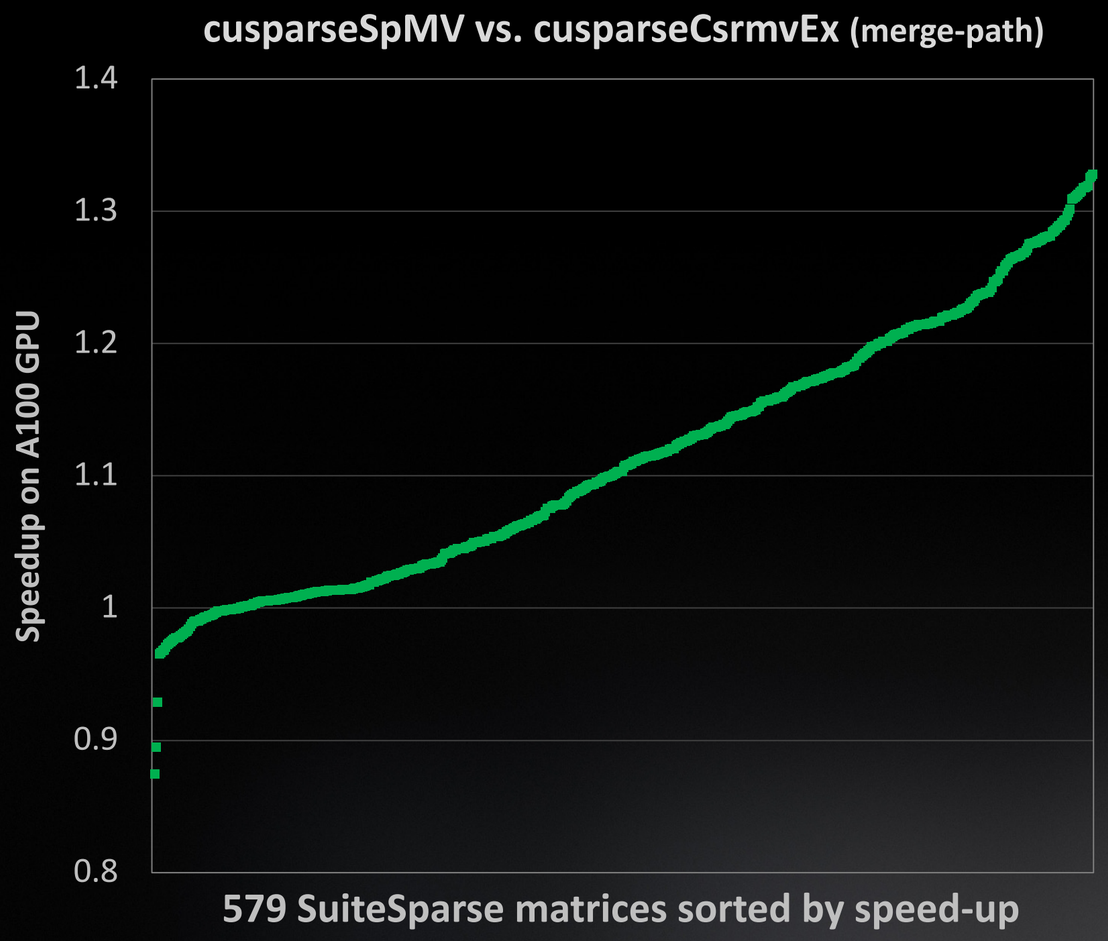
Scaling Up
Multi-GPU and Multi-Node Libraries

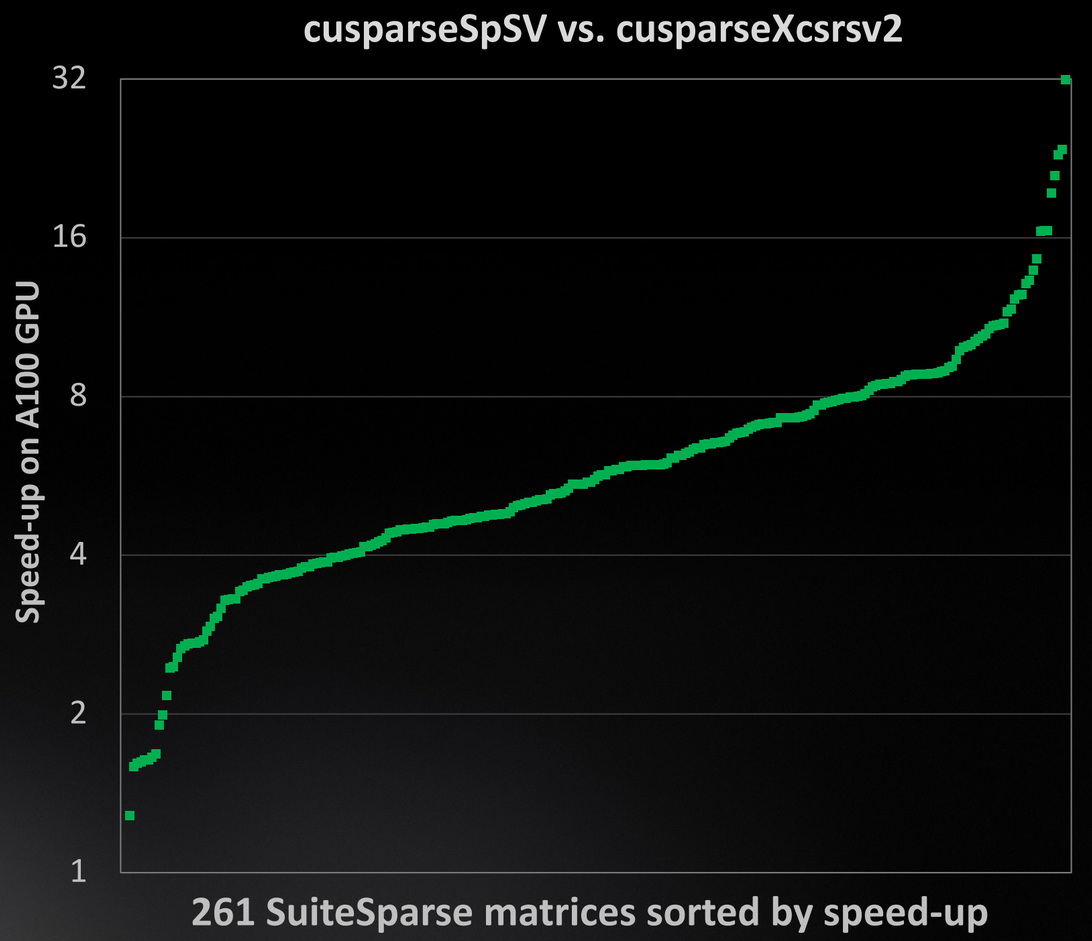


Composability
Device Functions

IMPROVED PERFORMANCE ON KEY HPC KERNELS

Sparse Matrix x Vector and Sparse Triangular Solver

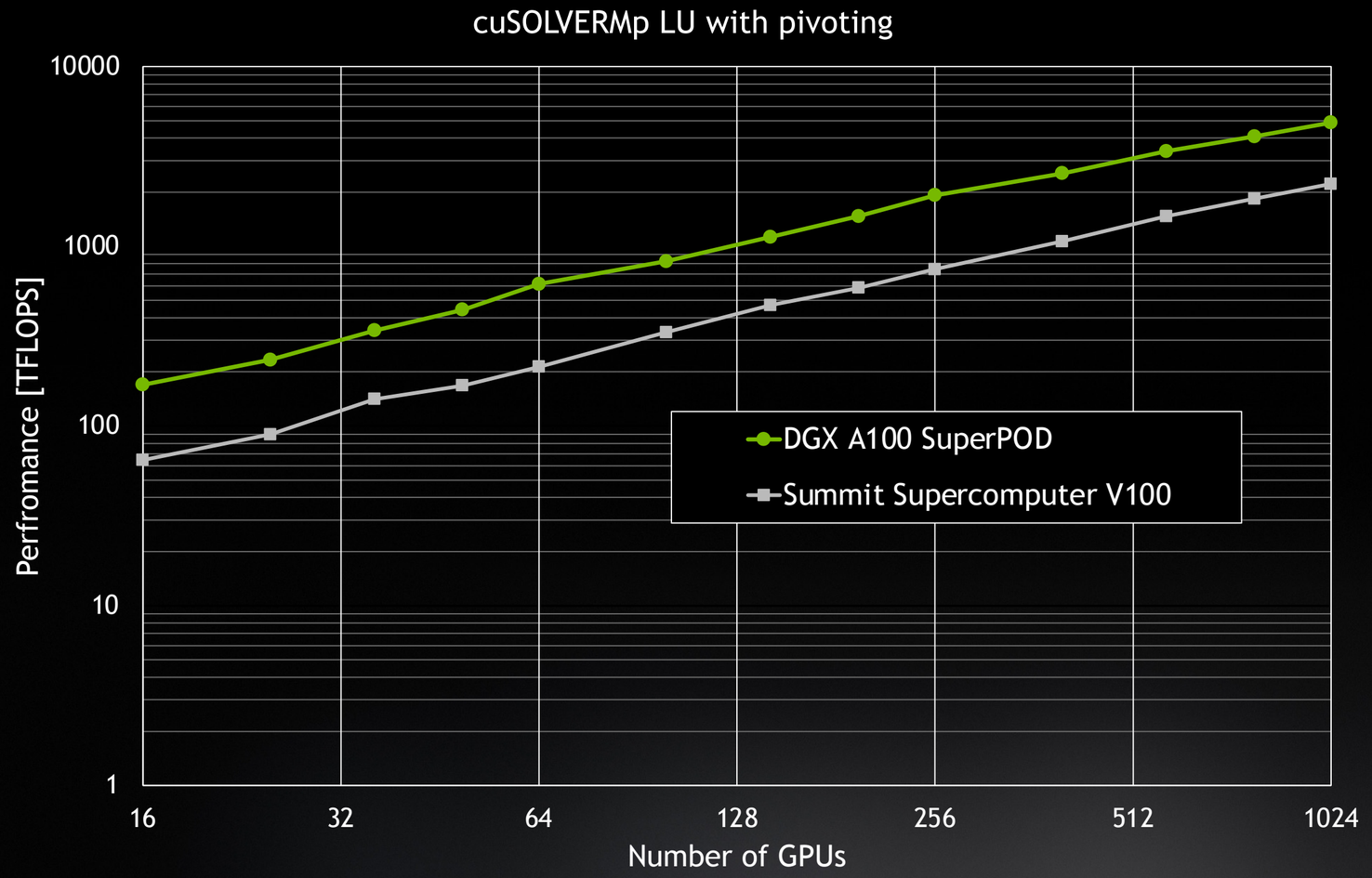




NVIDIA.

cuSOLVER DISTRIBUTED MULTI-GPU LU SOLVER

Scalability on Top 5 supercomputers



- Supports standard 2D block cyclic distribution between processes
- > 2.2X speed-up A100 over V100
- Early Access available https://developer.nvidia.com/cudamathlibraryea

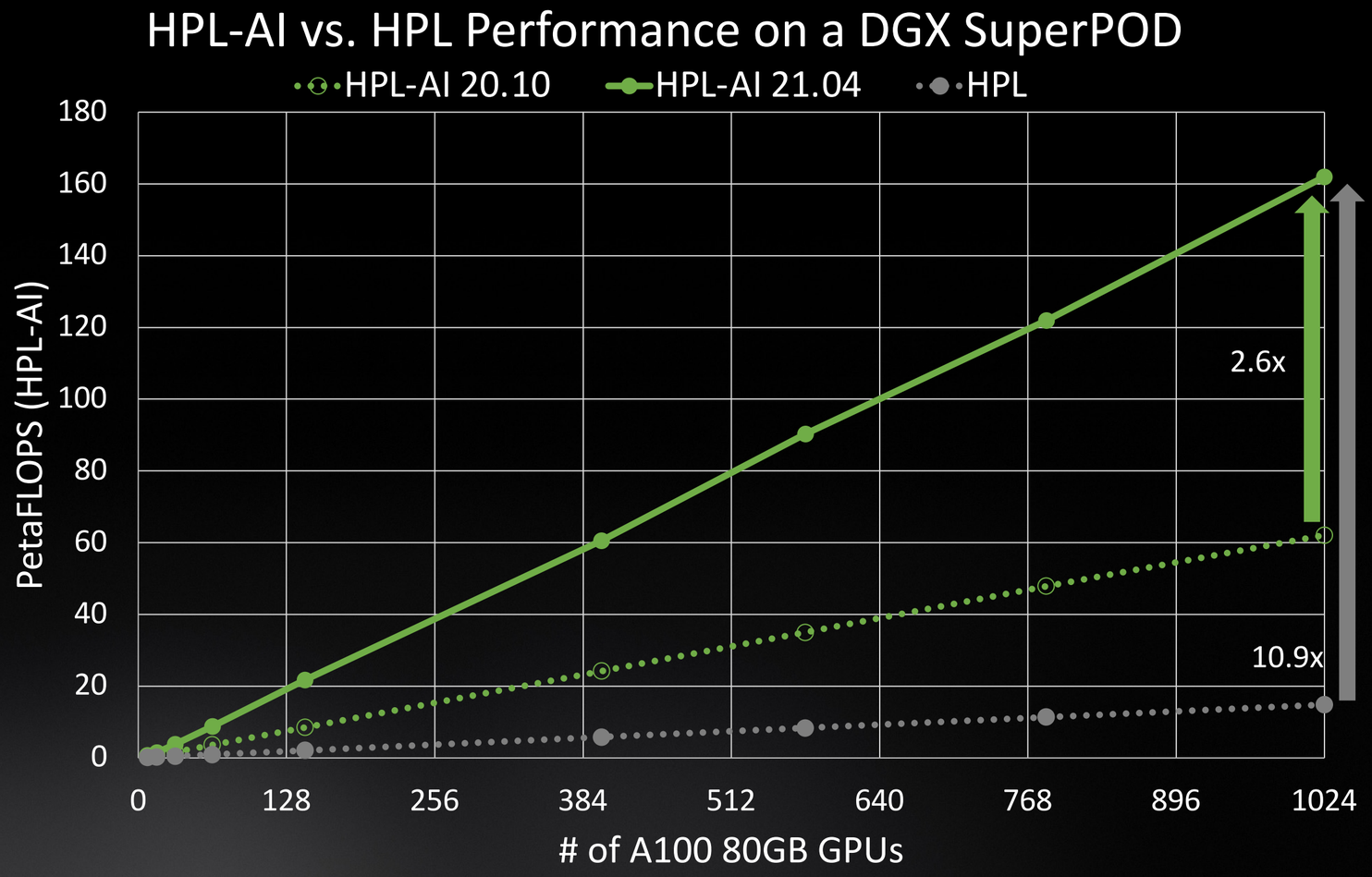
Data shown is for double complex LU factorization for 27882 rows/cols per device

HPL-AI UPDATE

Available on NGC as part of the HPC Benchmarks Container

- First HPC Benchmarks release 2020.10 on NGC provided: HPL, HPL-AI, HPCG benchmarks with optimized performance on NVIDIA accelerated HPC systems
- 2021.04 release delivers a 2.6X speed-up for HPL-AI on a DGX SuperPOD

https://ngc.nvidia.com/catalog/containers/nvidia:hpc-benchmarks



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	Quantum

Acceleration Libraries

```cpp
static inline
void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                  Index_t *regElemlist, Real_t dvovmax, Real_t& dthydro)
{
#if _OPENMP
  const Index_t threads = omp_get_max_threads();
  Index_t hydro_elem_per_thread[threads];
  Real_t dthydro_per_thread[threads];
#else
  Index_t threads = 1;
  Index_t hydro_elem_per_thread[1];
  Real_t dthydro_per_thread[1];
#endif
#pragma omp parallel firstprivate(length, dvovmax)
  {
    Real_t dthydro_tmp = dthydro ;
    Index_t hydro_elem = -1 ;
#if _OPENMP
    Index_t thread_num = omp_get_thread_num();
#else
    Index_t thread_num = 0;
#endif
#pragma omp for
    for (Index_t i = 0 ; i < length ; ++i) {
      Index_t indx = regElemlist[i] ;

      if (domain.vdov(indx) != Real_t(0.)) {
        Real_t dtdvov = dvovmax / (FABS(domain.vdov(indx))+Real_t(1.e-20)) ;

        if ( dthydro_tmp > dtdvov ) {
          dthydro_tmp = dtdvov ;
          hydro_elem = indx ;
        }
      }
    }
    dthydro_per_thread[thread_num] = dthydro_tmp ;
    hydro_elem_per_thread[thread_num] = hydro_elem ;
  }
  for (Index_t i = 1; i < threads; ++i) {
    if(dthydro_per_thread[i] < dthydro_per_thread[0]) {
      dthydro_per_thread[0] = dthydro_per_thread[i];
      hydro_elem_per_thread[0] = hydro_elem_per_thread[i];
    }
  }
  if (hydro_elem_per_thread[0] != -1) {
    dthydro = dthydro_per_thread[0] ;
  }
  return ;
}
```

C++ with OpenMP

PARALLEL C++

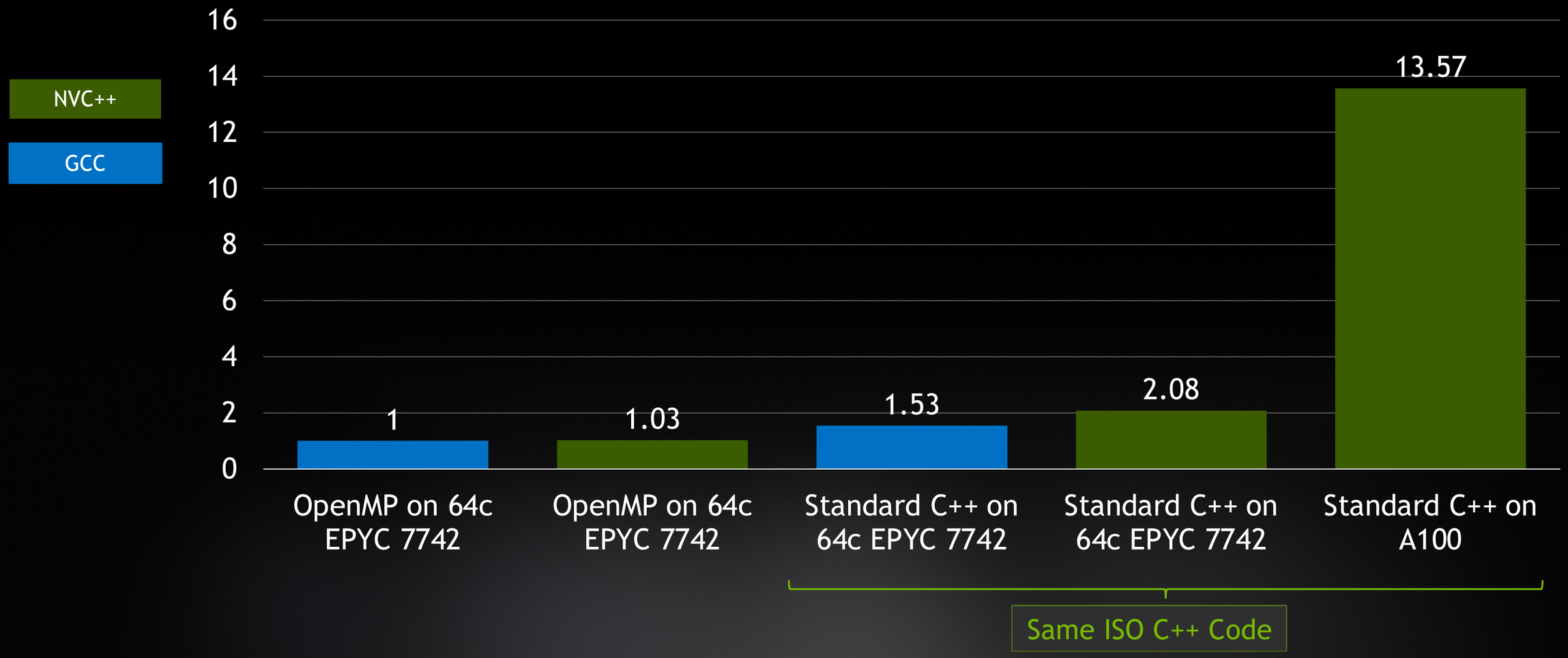
- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …

```cpp
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                                               Index_t *regElemlist,
                                               Real_t dvovmax,
                                               Real_t &dthydro)
{
  dthydro = std::transform_reduce(
    std::execution::par, counting_iterator(0), counting_iterator(length),
    dthydro, [](Real_t a, Real_t b) { return a < b ? a : b; },
    [=, &domain](Index_t i)
  {
      Index_t indx = regElemlist[i];
      if (domain.vdov(indx) == Real_t(0.0)) {
        return std::numeric_limits<Real_t>::max();
      } else {
        return dvovmax / (std::abs(domain.vdov(indx)) + Real_t(1.e-20));
      }
  });
}
```

Parallel C++17

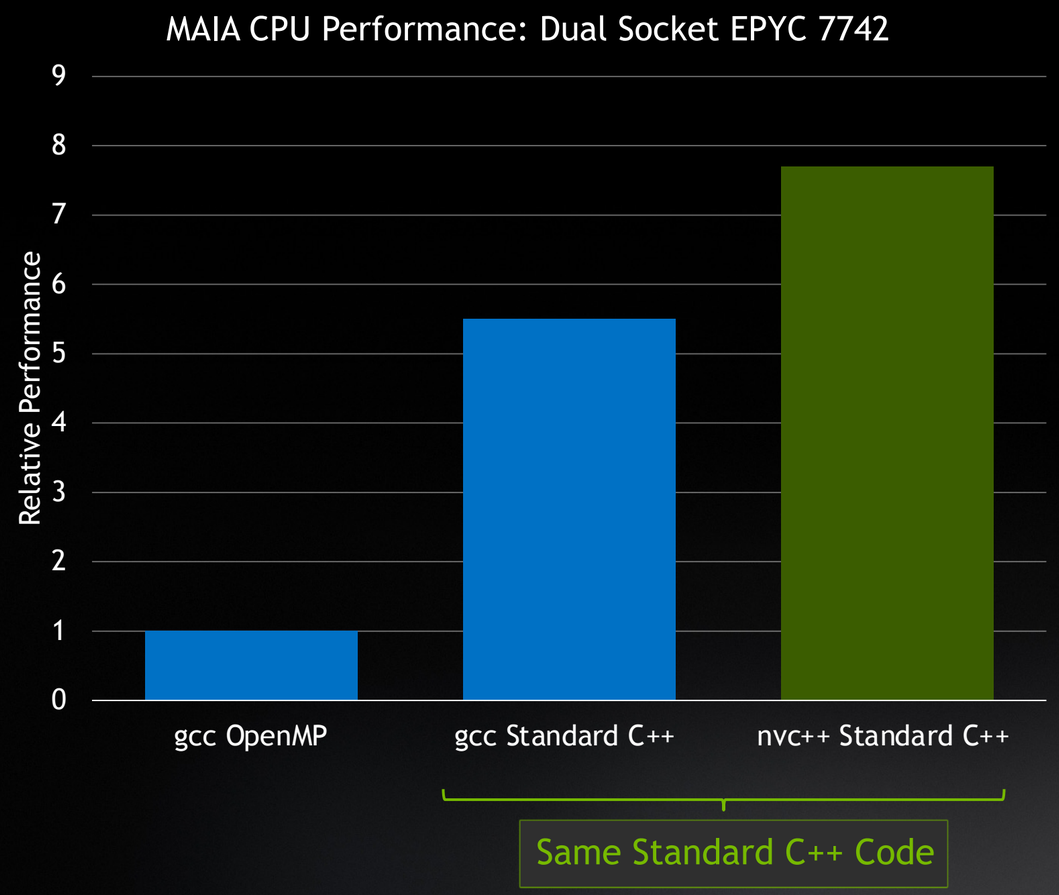
LULESH PERFORMANCE

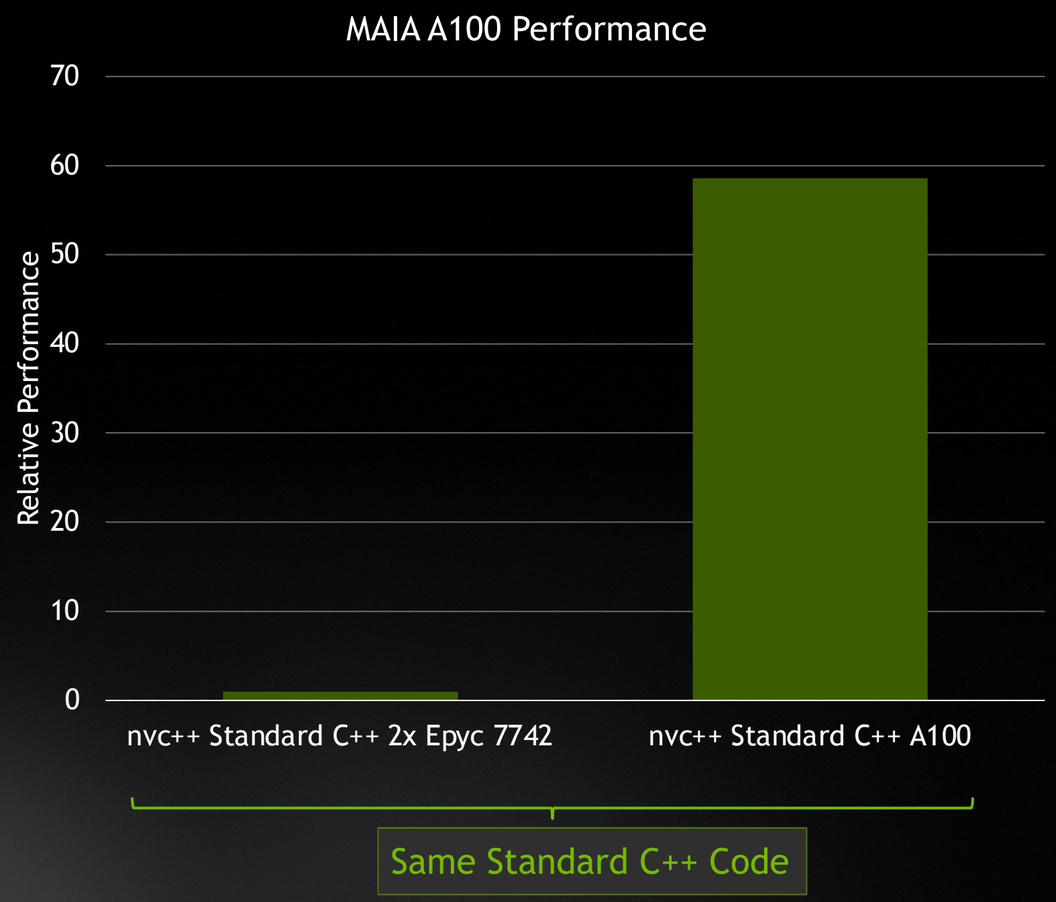
Relative Performance – Higher is Better



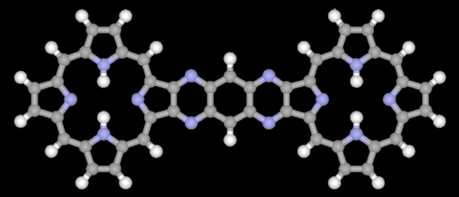
C++ STANDARD PARALLELISM

MAIA





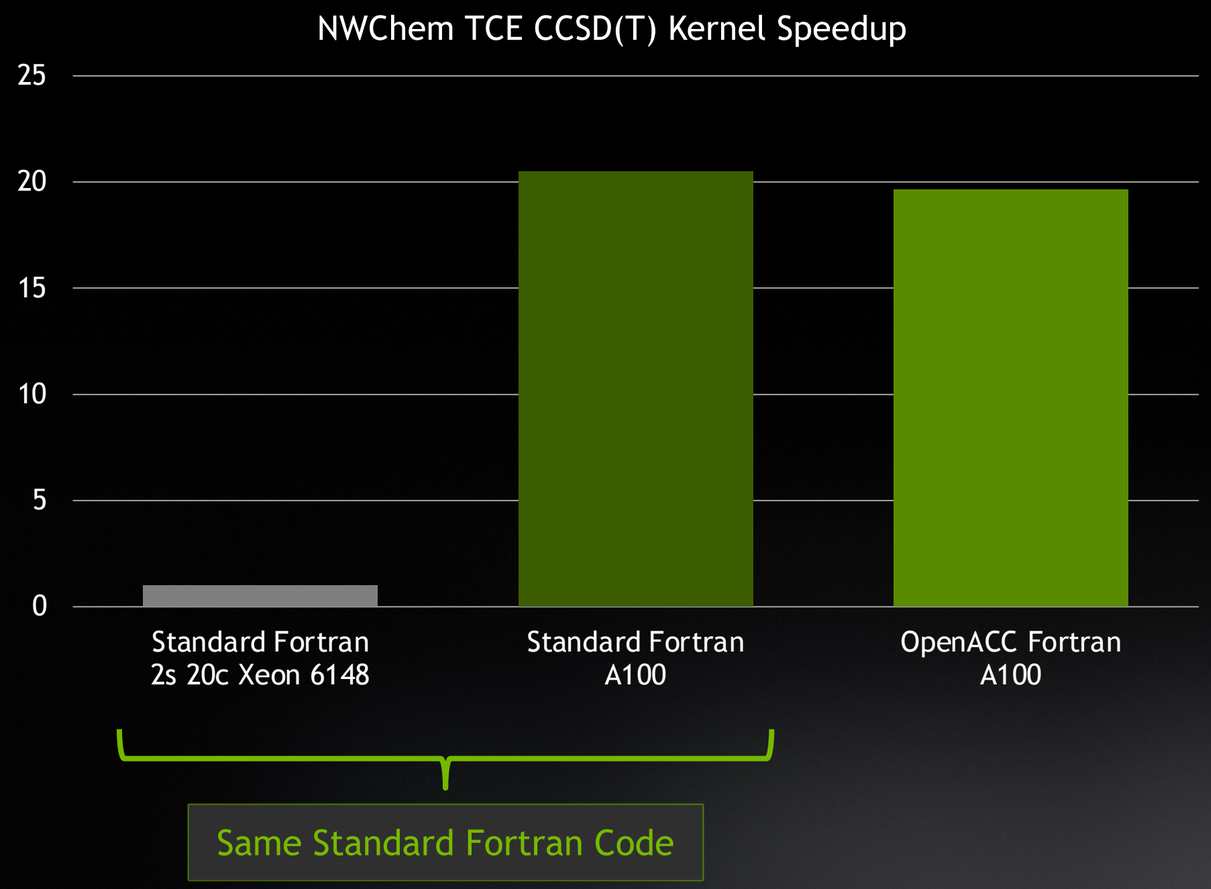
NVIDIA

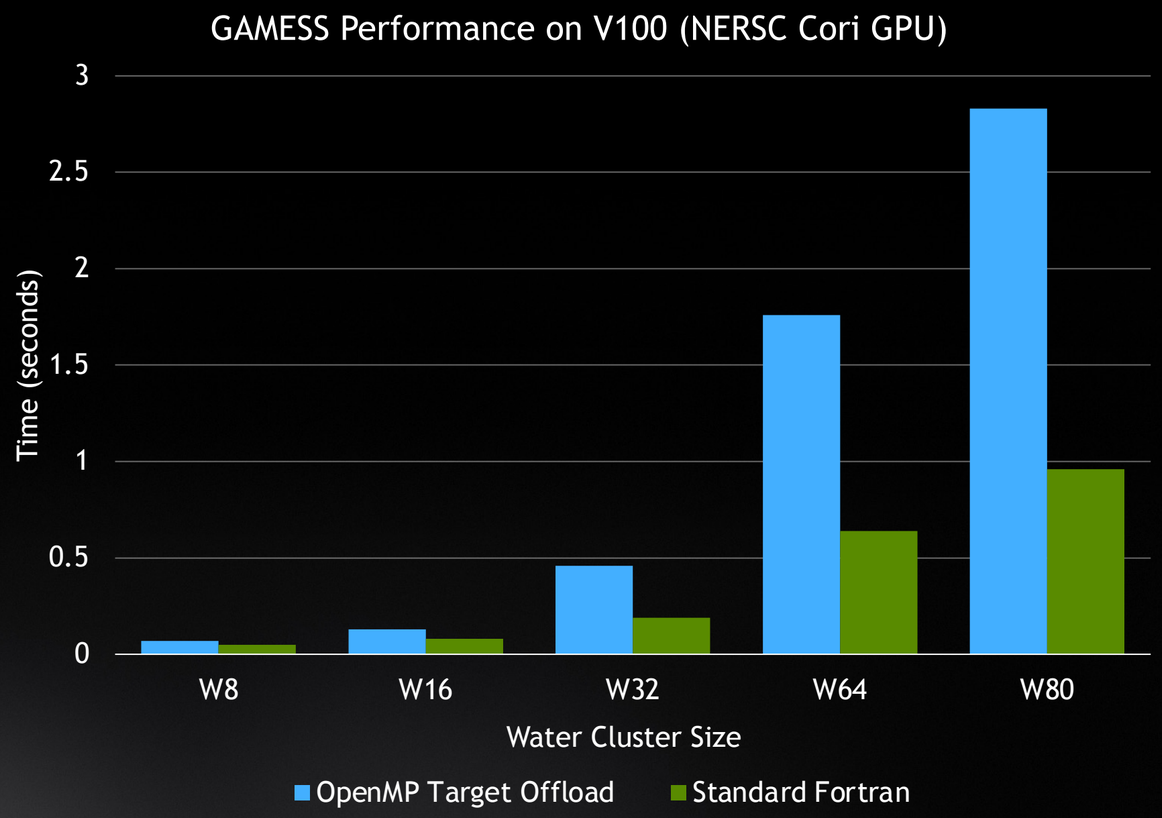


FORTRAN STANDARD PARALLELISM

NWChem and GAMESS with DO CONCURRENT

GAMESS





GAMESS results from Melisa Alkan and Gordon Group, Iowa State

NVIDIA

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU and Network

```python
import numpy as np
import 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
payload_r = np.random.randn(size) * 100.0
rhs = pd.DataFrame({"key": key_r, "payload": payload_r})

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

Accelerated Standards

```python
import cudart

alloc1 = cudart.cudaMalloc(10)
alloc2 = cudart.cudaMalloc(10)
cudart.cudaMemset(alloc1, 5, 10)
cudart.cudaMemcpy(alloc2, alloc1, 10,
                  cudart.cudaMemcpyDeviceToDevice)
```

Platform Specialization

Core	Math	Communication	Data Analytics	AI	Quantum

Acceleration Libraries

PYTHON STANDARD PARALLELISM

Introducing Legate

Legate

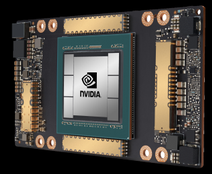
Legate transparently accelerates and scales existing Numpy and Pandas workloads

Program from the edge to the supercomputer in Python by changing 1 import line

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

Alpha release available at github.com/nv-legate

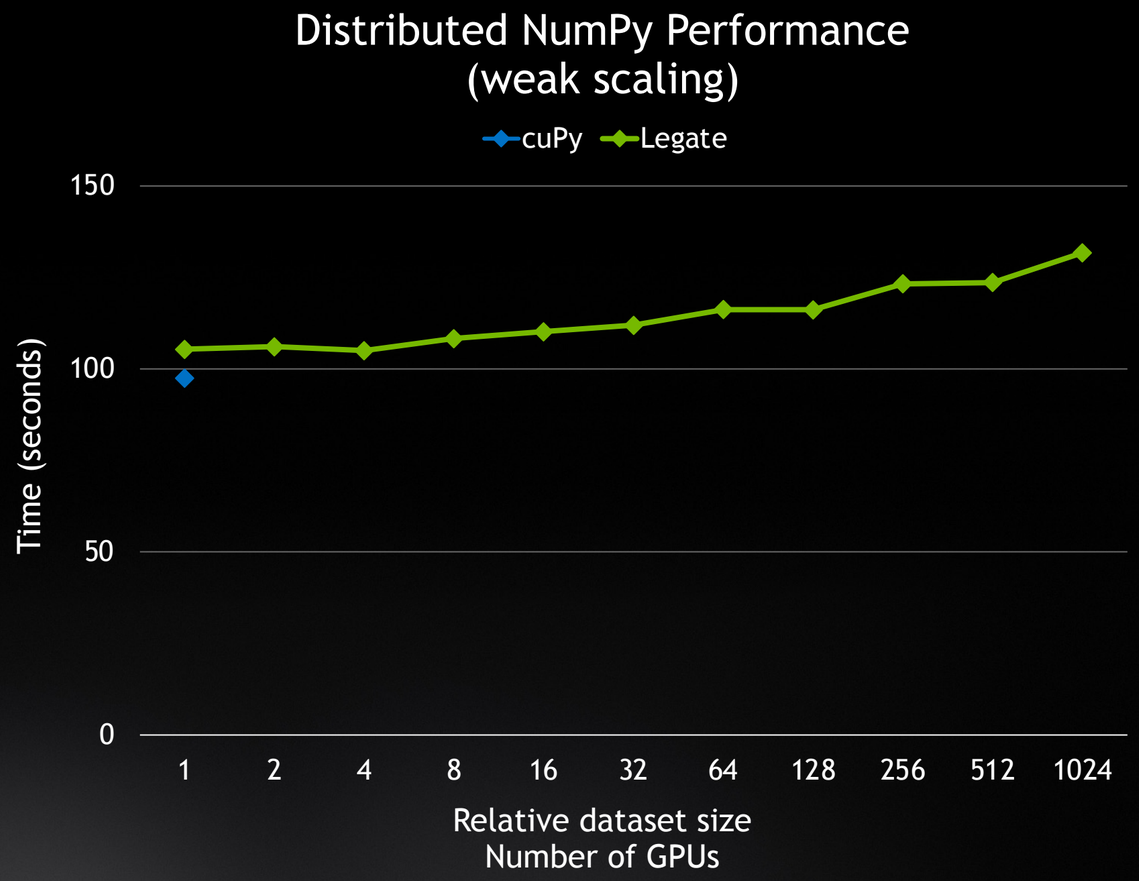
```
# Import numpy as np
import legate.numpy as np

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)

…
```



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
    )
```



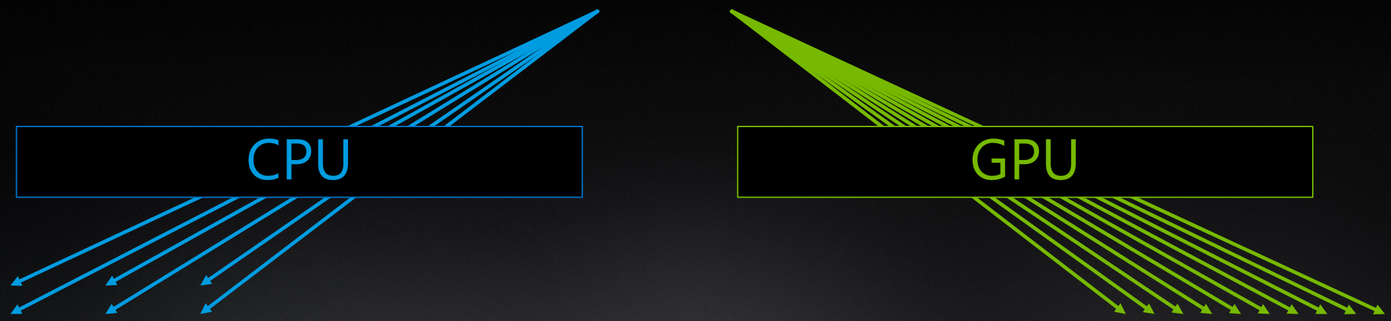
Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations.
Journal of Open Source Education, **1**(9), 21, https://doi.org/10.21105/jose.00021

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
```

THE NVIDIA HPC SDK IS READY FOR ARM

Auto Vectorization

Vector Intrinsics

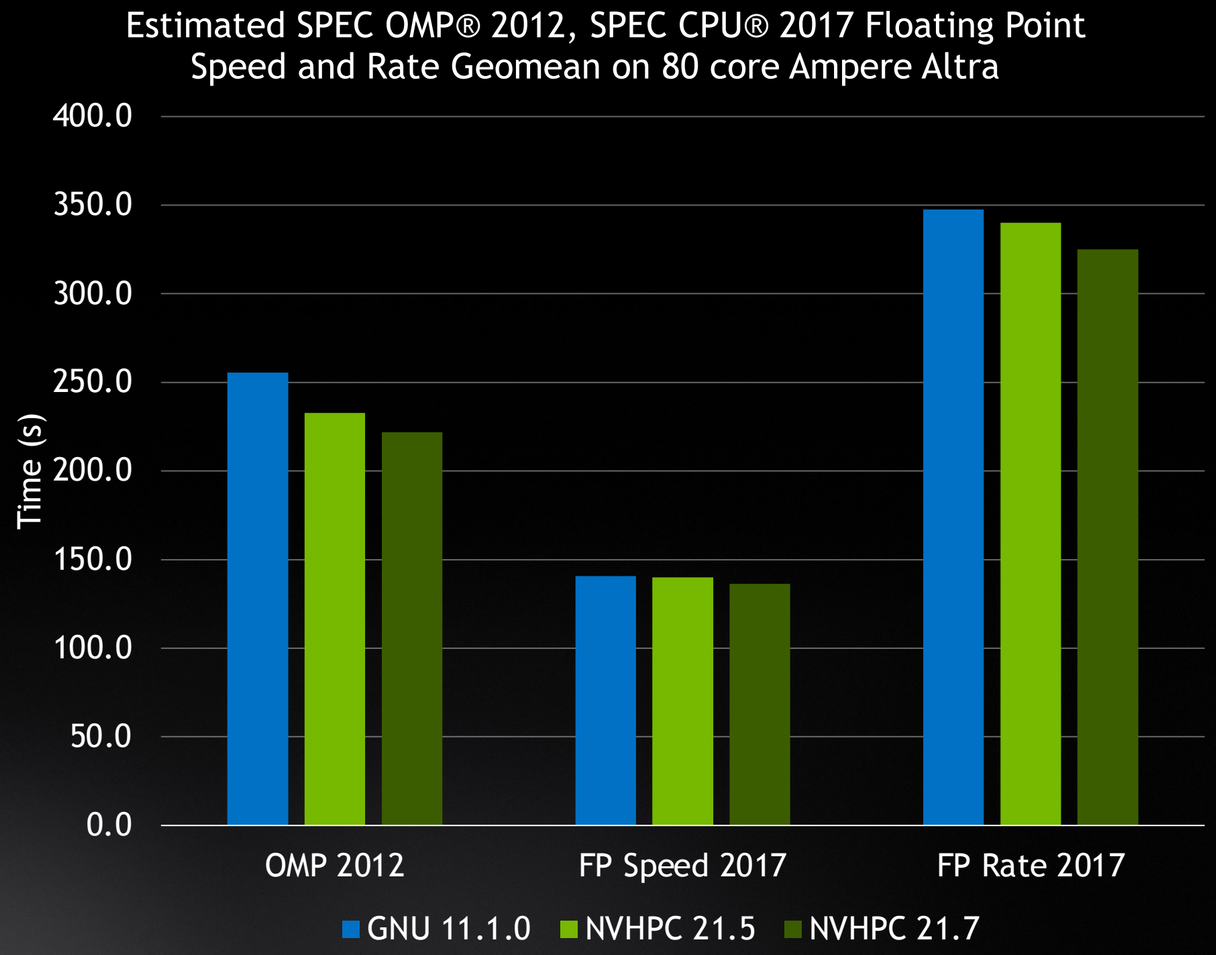
```
float32x2_t vadd_f32 (float32x2_t a, float32x2_t b);
float32x4_t vdupq_n_f32 (float32_t value);
float32x2_t vget_high_f32 (float32x4_t a);
```

Host Parallelism

- Standard Parallelism
- OpenACC
- OpenMP

FP Models

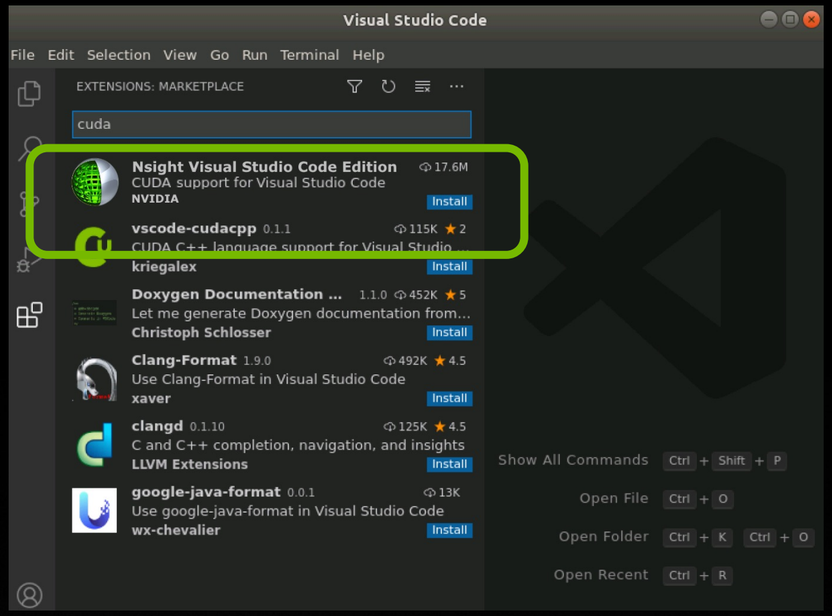
- Precise
- Fast
- Relaxed

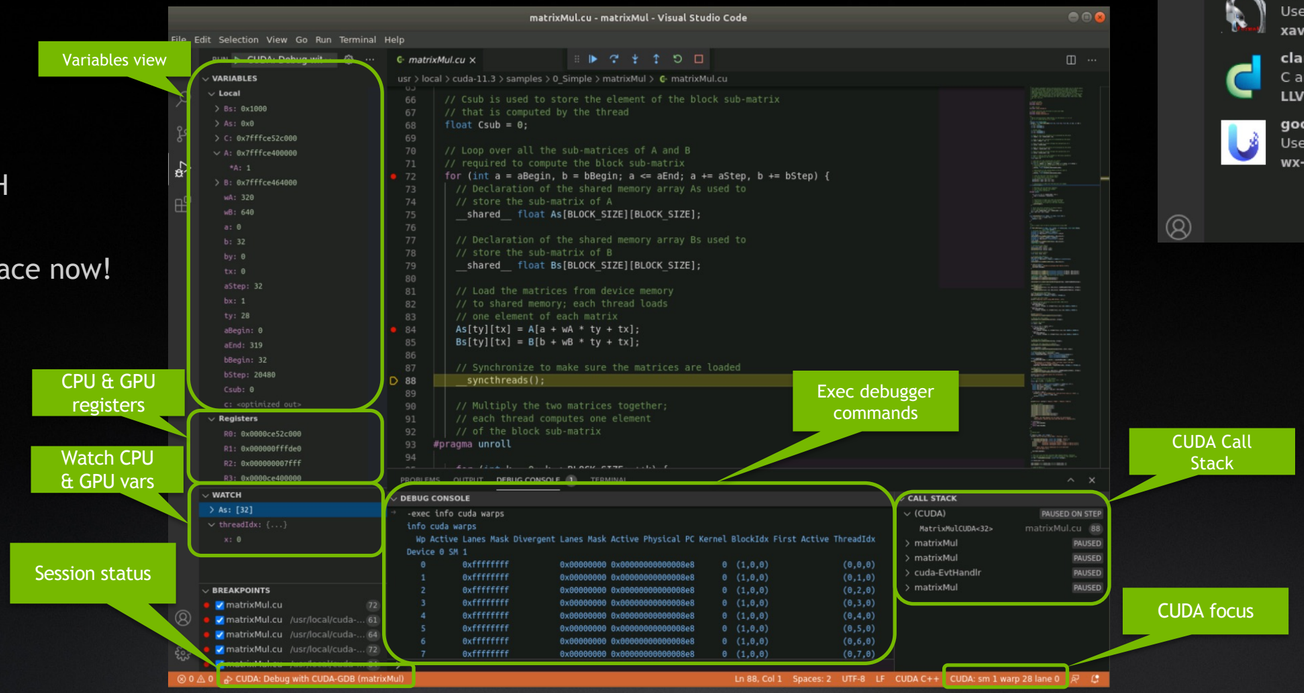


NSIGHT VISUAL STUDIO CODE EDITION

Visual Studio Code extensions that provides:

- CUDA code syntax highlighting
- CUDA code completion
- Build warning/errors
- Debug CPU & GPU code
- Remote connection support via SSH
- Available on the VS Code Marketplace now!



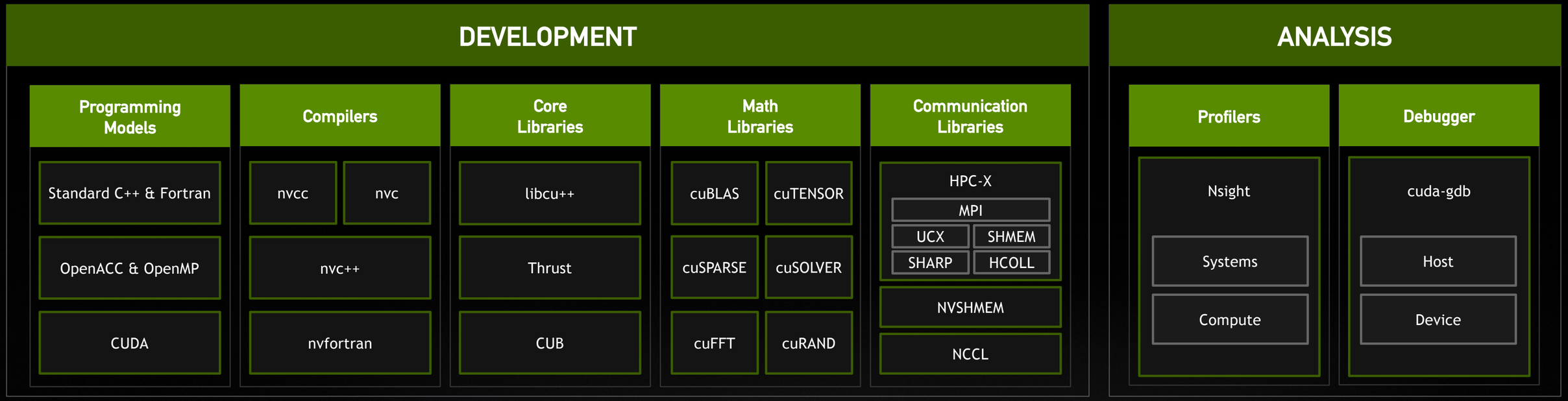


https://developer.nvidia.com/nsight-visual-studio-code-edition

NVIDIA

NVIDIA HPC SDK

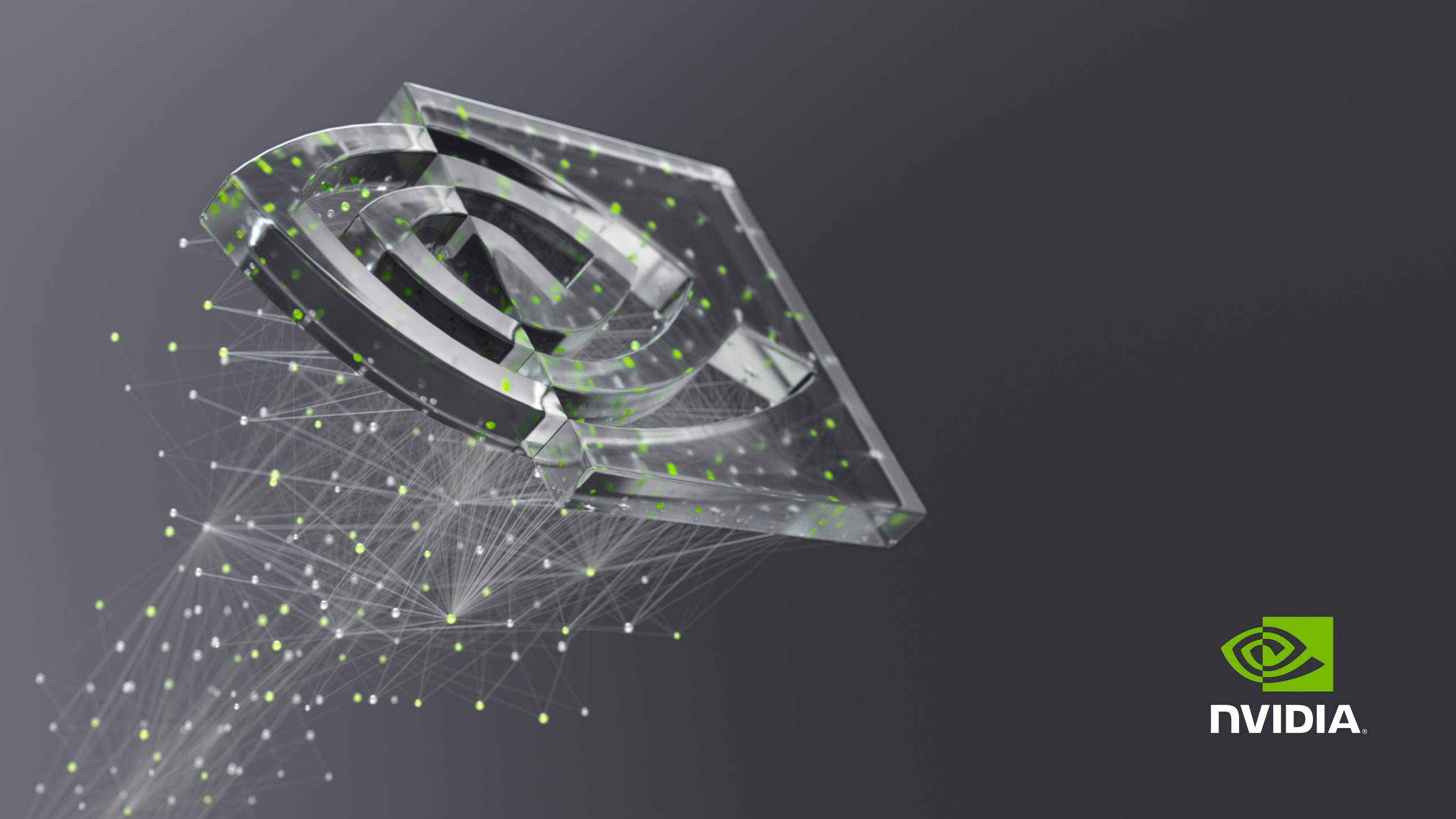
Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud



Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

7-8 Releases Per Year | Freely Available


NVIDIA