A Deep Dive into the Latest HPC Software

Axel Koehler | Principal Solutions Architect | NVIDIA

Cray User Group - Helsinki - May 10th 2023

Agenda

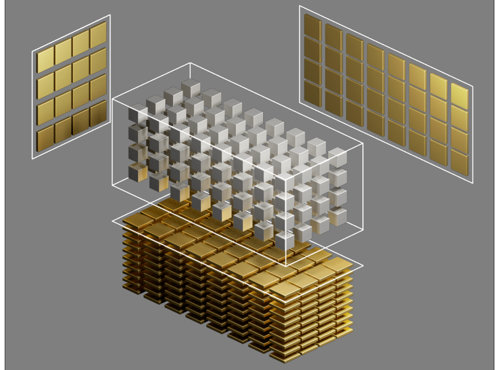
- Programming the NVIDIA Platform
- Standard Language Parallelism
- Accelerated Libraries
- Arm Software Stack

NVIDIA HPC Software

Major Initiatives

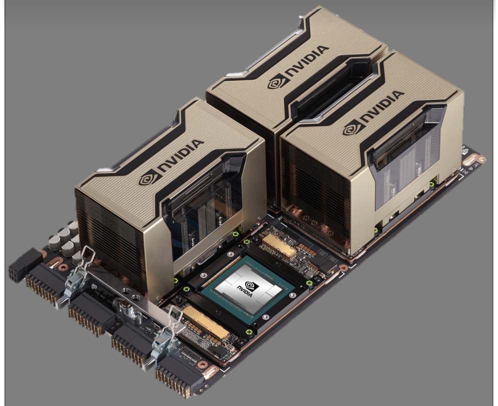
Seamless Acceleration

STDPAR, Tensor Cores, GH C2C



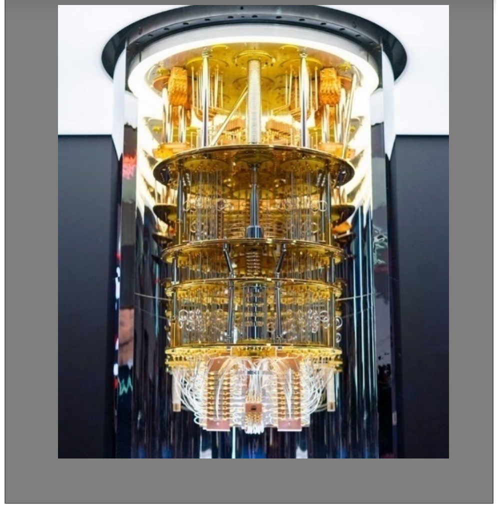
Scaling Up

Multi-GPU and Multi-Node Libraries



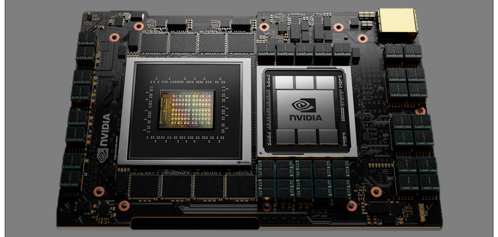
Domain Libraries

Quantum, Signal Processing, LQCD, etc



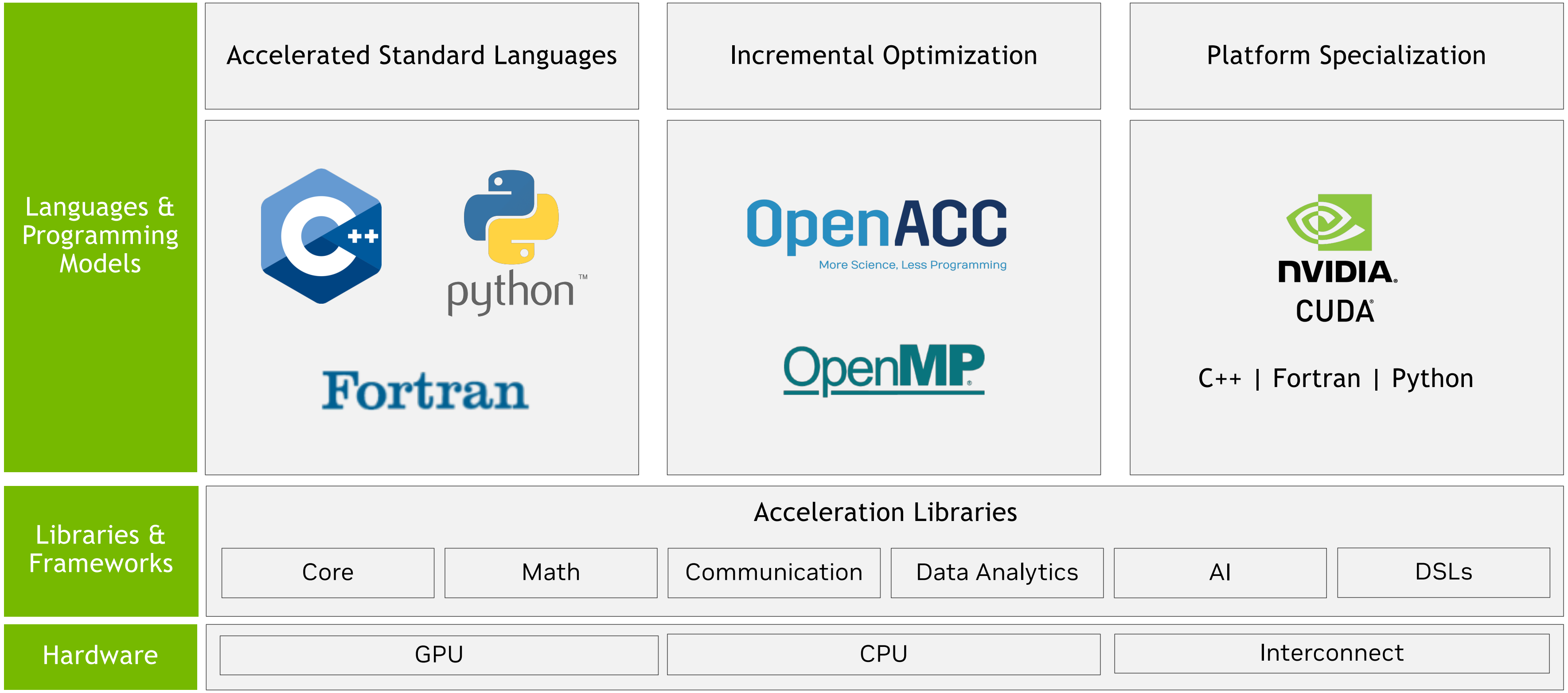
Arm Software

Compilers, Libraries, Ecosystem



Programming the NVIDIA Platform

Programming The NVIDIA Platform



Languages & Programming Models

Accelerated Standard Languages

C++

python™

Fortran

Incremental Optimization

OpenACC

More Science, Less Programming

OpenMP

Platform Specialization

NVIDIA

CUDA

C++ | Fortran | Python

Libraries & Frameworks

Acceleration Libraries

Core | Math | Communication | Data Analytics | AI | DSLs

Hardware

GPU | CPU | Interconnect

Accelerated Standard Languages

Parallel performance for wherever your code runs

ISO C++

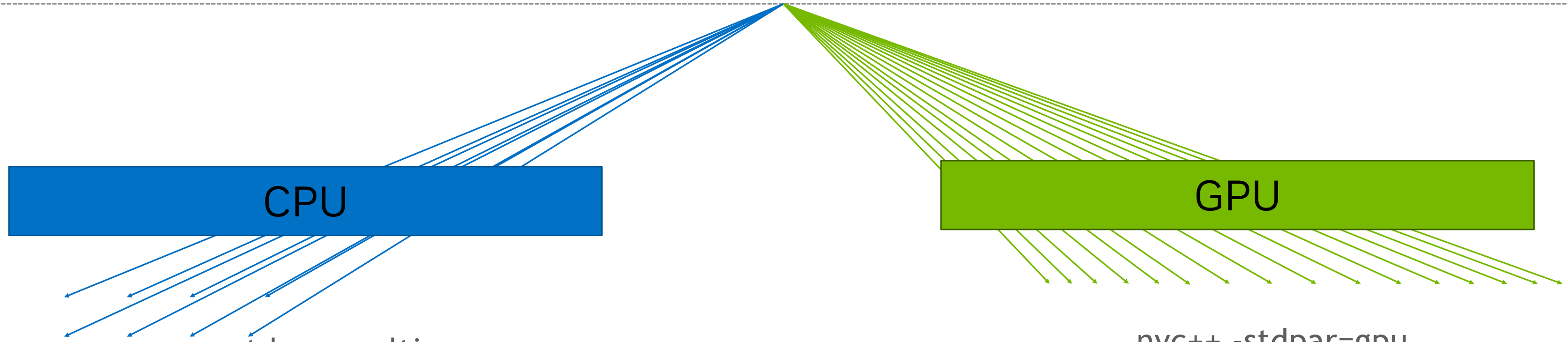
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

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

Python

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

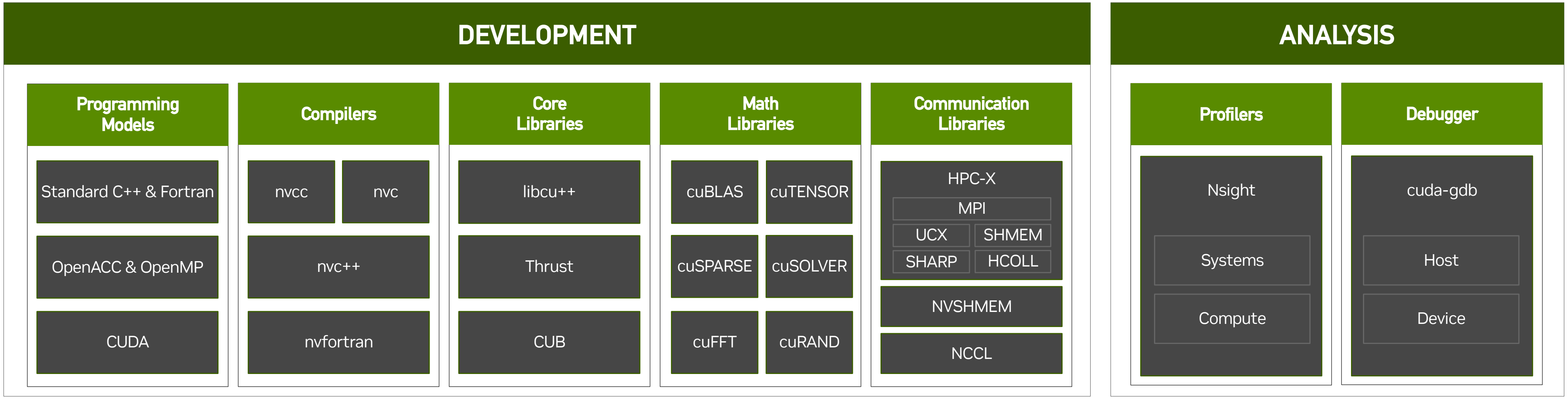


nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
legate -cpus 16 saxpy.py

nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
legate -gpus 1 saxpy.py

NVIDIA HPC SDK

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



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

x86_64 | Arm | OpenPOWER

7-8 Releases Per Year | Freely Available

Standard Language Parallelism

HPC Programming In ISO C++

ISO is the place for portable concurrency and parallelism

C++17 & C++20

Parallel Algorithms

- Parallel and vector concurrency

Forward Progress Guarantees

- Extend the C++ execution model for accelerators

Memory Model Clarifications

- Extend the C++ memory model for accelerators

Ranges

- Simplifies iterating over a range of values

Scalable Synchronization Library

- Express thread synchronization that is portable and scalable across CPUs and accelerators

Preview support coming to NVC++

C++23

`std::mdspan`

- HPC-oriented multi-dimensional array abstractions.
- Preview Available Now

Range-Based Parallel Algorithms

- Improved multi-dimensional loops

Extended Floating Point Types

- First-class support for formats new and old: `std::float16_t/float64_t`

And Beyond

Senders/Receivers

- Standardized mechanism for asynchrony in the C++ standard library
- Simplify launching and managing parallel work across CPUs and accelerators
- Preview Available Now

Linear Algebra

- C++ standard algorithms API to linear algebra
- Maps to vendor optimized BLAS libraries
- Preview Available Now

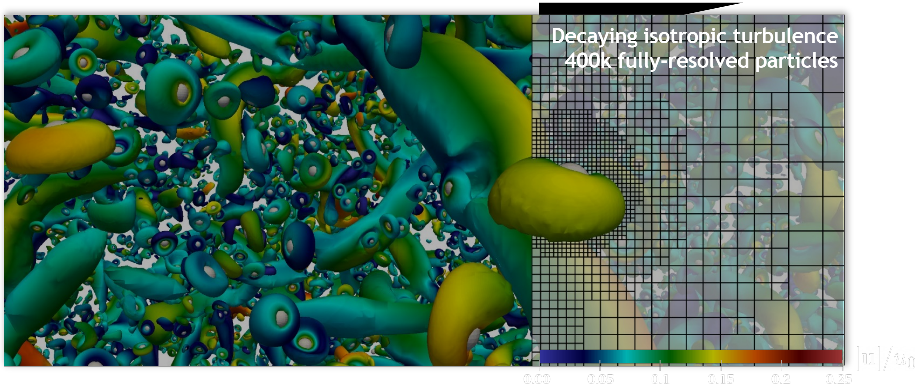
MDArray and SubMDSpan

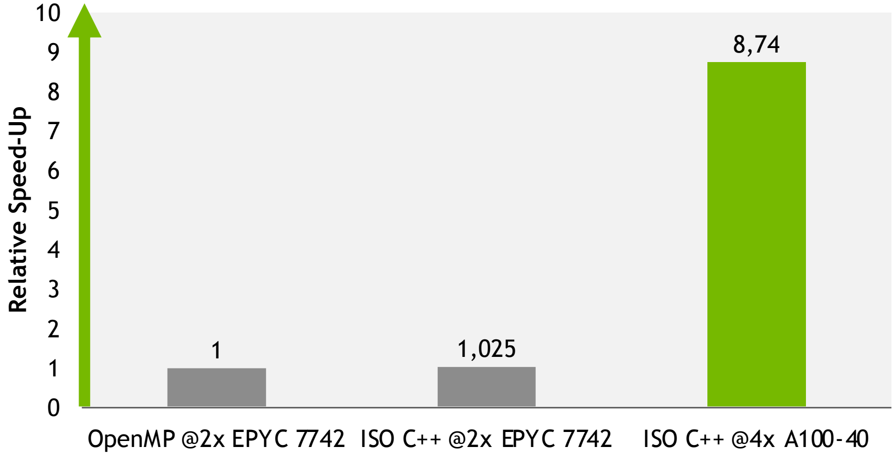
- Expands the capabilities of C++23 MDSpan
- Preview Available Now

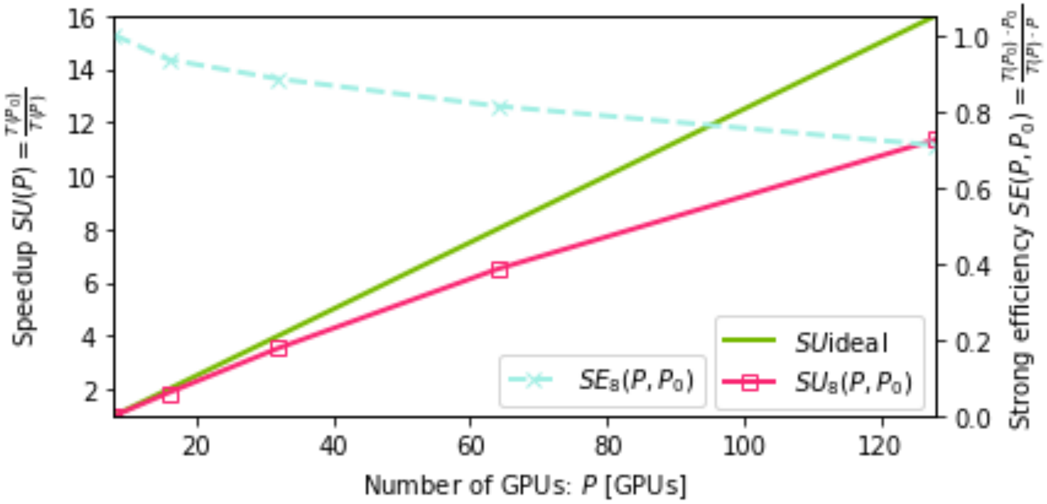
M-AIA

Multi-physics simulation framework developed at the Institute of Aerodynamics, RWTH Aachen University

- Hierarchical grids, complex moving geometries
- Adaptive meshing, load balancing
- Numerical methods: FV, DG, LBM, FEM, Level-Set, ...
- Physics: aeroacoustics, combustion, biomedical, ...
- Developed by ~20 PhDs (Mech. Eng.), ~500k LOC++
- **Programming model:** MPI + **ISO C++ parallelism**







HPC Programming in ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Fortran Array Math Intrinsics

- Since NVFORTRAN 20.5
- Accelerated matmul, reshape, spread, …

DO CONCURRENT

- Since NVFORTRAN 20.11
- Auto-offload & multi-core with vectorization

Co-Arrays

- Not currently available
- Accelerated co-array images

Preview support available now in NVFORTRAN

Fortran 2023

DO CONCURRENT Reductions

- Since NVFORTRAN 21.11
- **REDUCE** subclause added
- Support for `+`, `*`, `MIN`, `MAX`, `IAND`, `IOR`, `IEOR`.
- Support for `.AND.`, `.OR.`, `.EQV.`, `.NEQV` on `LOGICAL` values

GAMESS

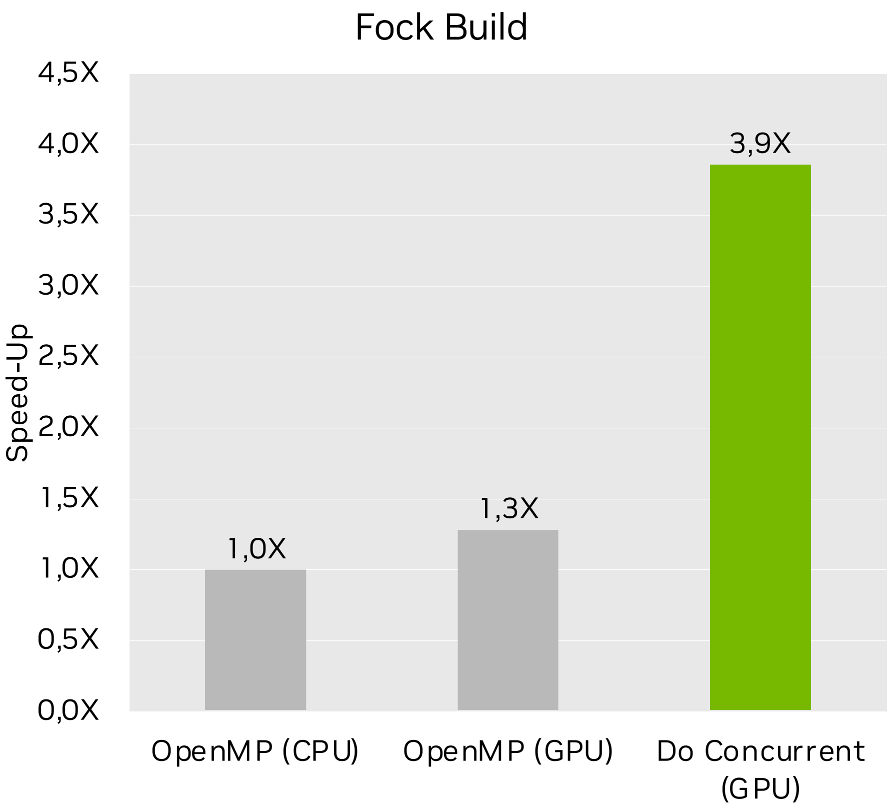
Computational Chemistry with Fortran Do Concurrent

- GAMESS is a popular Quantum Chemistry application.
- More than 40 years of development in Fortran and C
- MPI + OpenMP baseline code
- Hartree-Fock rewritten in Do Concurrent

```
!pre-sorting, screening

!$omp target teams distribute
             parallel do &
!$omp shared() private()
do iquart = 1, ssdd_quarts
!recover shell index
ish=IDX(s_sh)
jsh=IDX(s_sh)
ksh=IDX(d_sh)
lsh=IDX(d_sh)
  !compute ints
  !digest ints
enddo
!$omp end target teams distribute
          parallel do
```

```
!pre-sorting, screening


DO CONCURRENT (iquart=1::ssdd_quarts)&
   SHARED() LOCAL()
!recover shell index
ish=IDX(s_sh)
jsh=IDX(s_sh)
ksh=IDX(d_sh)
lsh=IDX(d_sh)
  !compute ints
  !digest ints
enddo
```

* Courtesy of Melisa Alkan, Iowa State University. Not yet published.



nvfortran 22.7, NVIDIA A100 GPU, AMD “Milan” CPU

cuNumeric

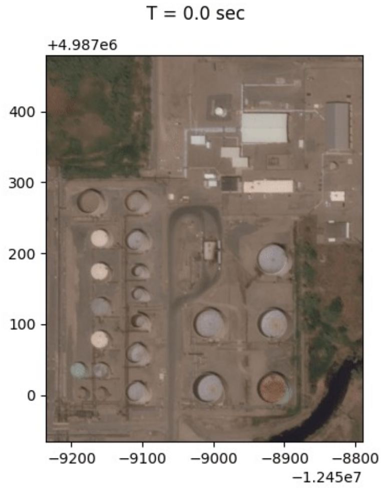
Automatic NumPy Acceleration and Scalability

cuNumeric

cuNumeric transparently accelerates and scales existing Numpy workloads

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

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

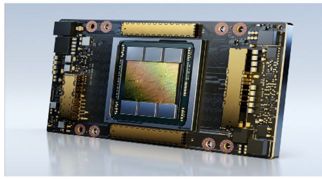


```
for _ in range(iter):
    un = u.copy()

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

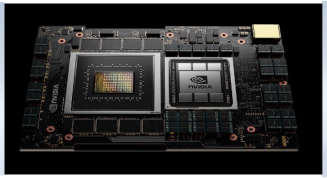
…

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

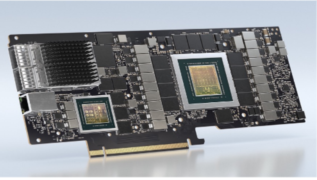
Run **everywhere**!



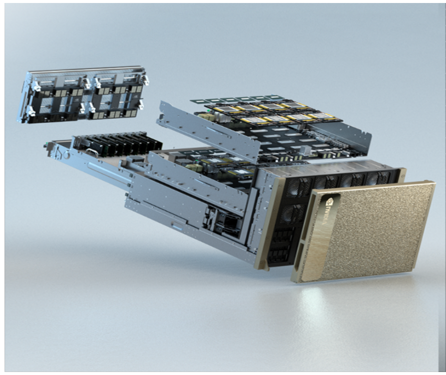
GPU



Grace CPU



DPU



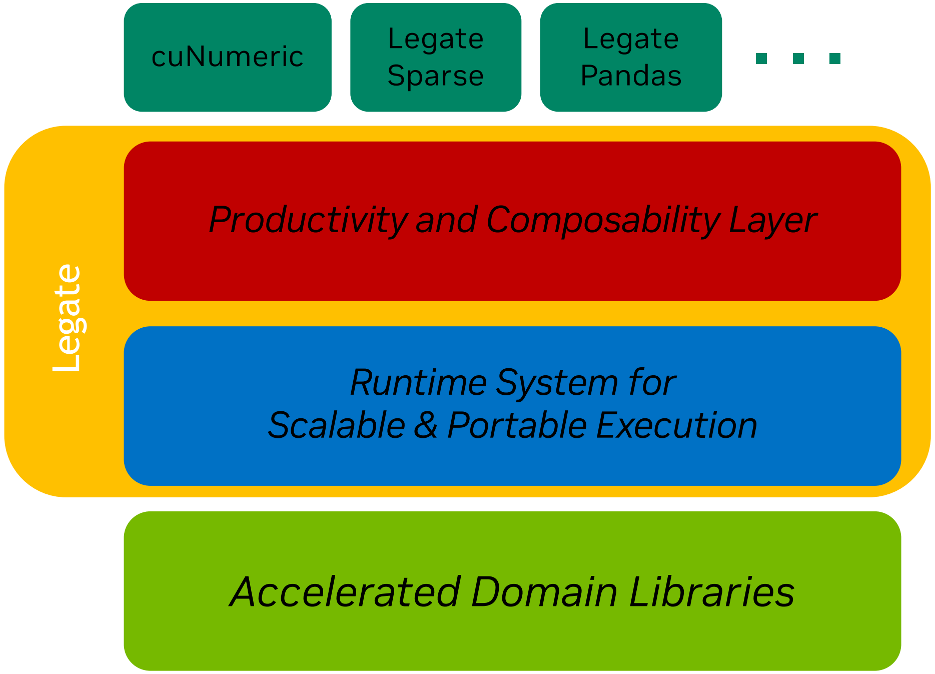
DGX

DGX SuperPod

Behind the Curtain: Legate

Powerhouse of cuNumeric and all other Legate libraries

Vision: build an **ecosystem of composasble and easy-to-use libraries**

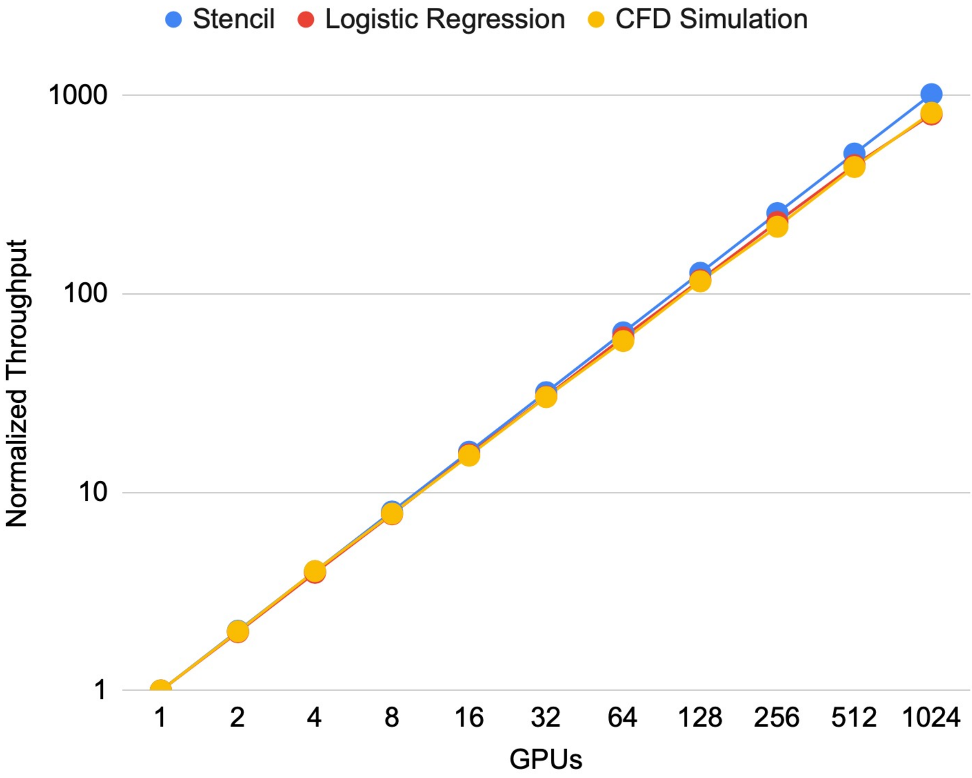

cuNumeric
Legate Sparse
Legate Pandas
. . .
Legate
Productivity and Composability Layer
Runtime System for Scalable & Portable Execution
Accelerated Domain Libraries

Productivity layer that ***accelerates*** library development

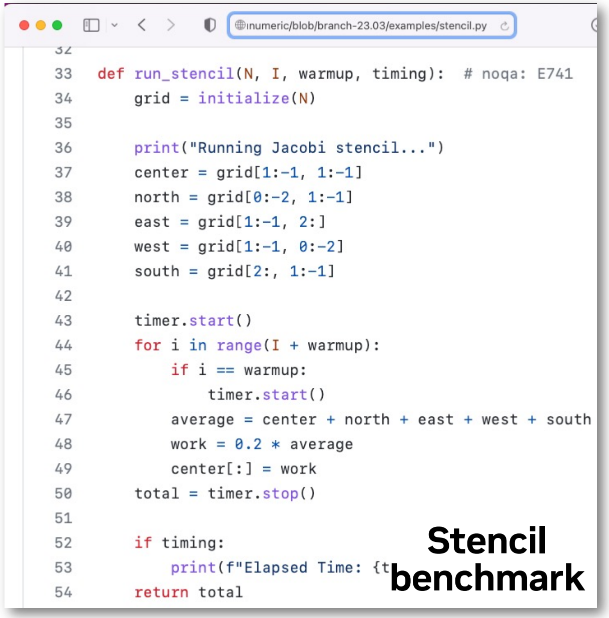
Common runtime system for *scalable* extraction of implicit parallelism

Accelerated domain libraries for excellent single-accelerator performance

Weak Scaling Performance



- No modifications required to scale the benchmarks to a thousand GPUs



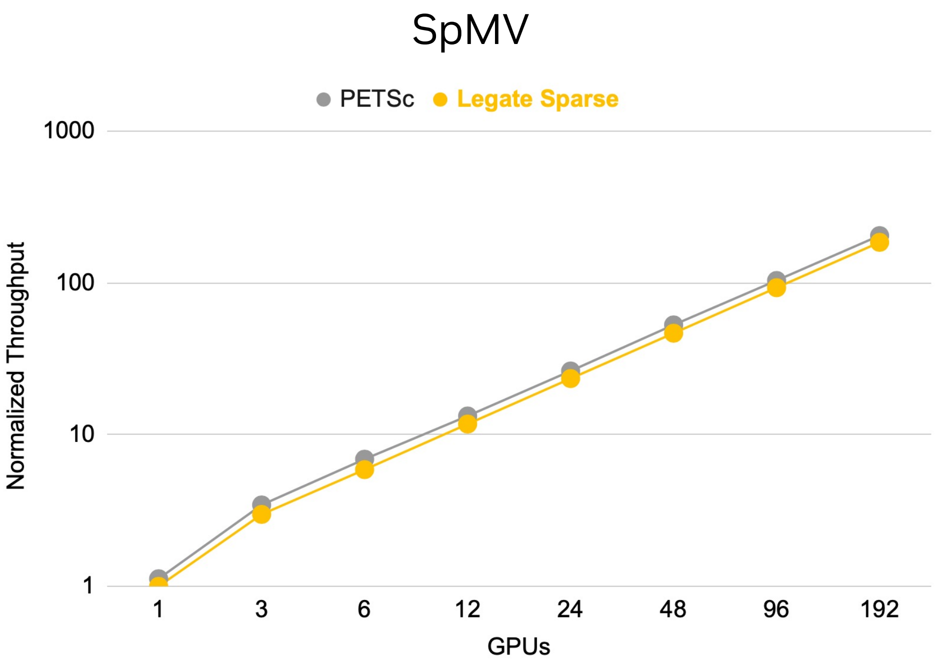
```python
def run_stencil(N, I, warmup, timing):  # noqa: E741
    grid = initialize(N)

    print("Running Jacobi stencil...")
    center = grid[1:-1, 1:-1]
    north = grid[0:-2, 1:-1]
    east = grid[1:-1, 2:]
    west = grid[1:-1, 0:-2]
    south = grid[2:, 1:-1]

    timer.start()
    for i in range(I + warmup):
        if i == warmup:
            timer.start()
        average = center + north + east + west + south
        work = 0.2 * average
        center[:] = work
    total = timer.stop()

    if timing:
        print(f"Elapsed Time: {t
    return total
```

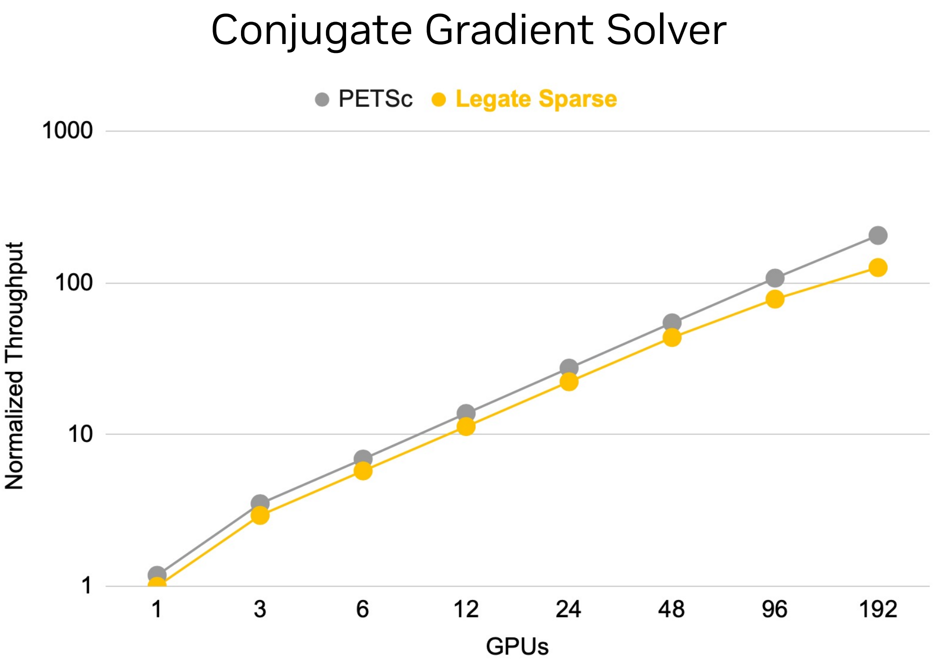
Stencil benchmark

- Single GPU performance comparable to / better than CuPy’s

Interoperation Example

Weak scaling performance





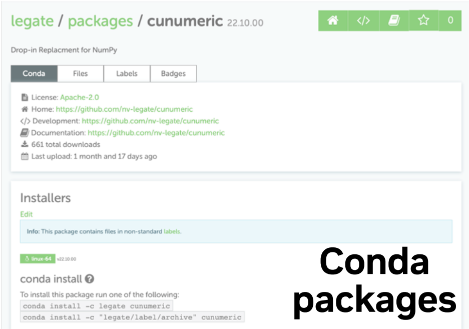
Pure Python implementations of the benchmarks have competitive performance as PETSc, a state-of-the-art MPI-based implementation

cuNumeric Beta Release

What’s packed in the release

- Coverage on ~60% NumPy API
  - Advanced indexing
  - Tensor contraction
  - Multi-dimensional sorting
  - 96% of ufuncs
  - 80% of RNGs

- Ergonomics



Drop-in Replacment for NumPy

License: Apache-2.0
Home: https://github.com/nv-legate/cunumeric
Development: https://github.com/nv-legate/cunumeric
Documentation: https://github.com/nv-legate/cunumeric
661 total downloads
Last upload: 1 month and 17 days ago

Installers

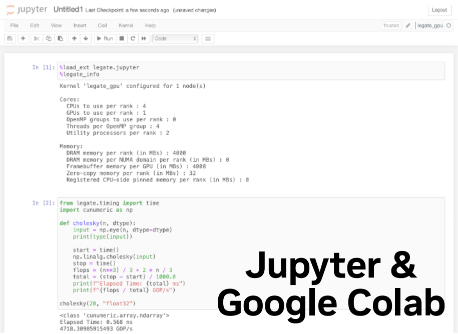
Info: This package contains files in non-standard labels.

conda install

To install this package run one of the following:

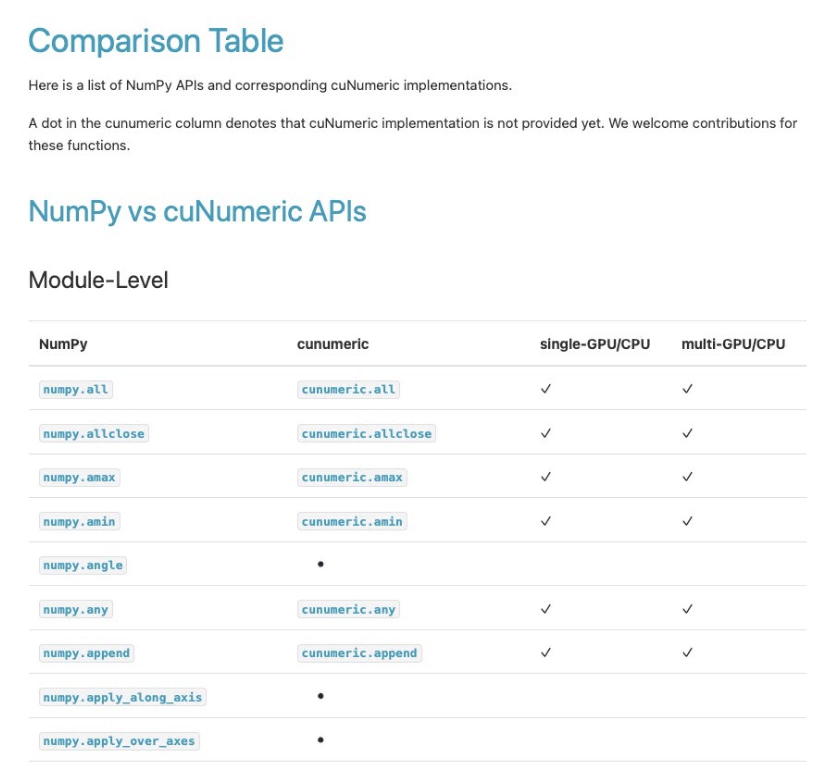
```
conda install -c legate cunumeric
conda install -c "legate/label/archive" cunumeric
```

Conda packages



```
In [1]: %load_ext legate.jupyter
        %legate_info

Kernel 'legate_gpu' configured for 1 node(s)

Cores:
  CPUs to use per rank : 4
  GPUs to use per rank : 1
  OpenMP groups to use per rank : 0
  Threads per OpenMP group : 4
  Utility processors per rank : 2

Memory:
  DRAM memory per rank (in MBs) : 4000
  DRAM memory per NUMA domain per rank (in MBs) : 0
  Framebuffer memory per GPU (in MBs) : 4000
  Zero-copy memory per rank (in MBs) : 32
  Registered CPU-side pinned memory per rank (in MBs) : 0

In [2]: from legate.timing import time
        import cunumeric as np

        def cholesky(n, dtype):
            input = np.eye(n, dtype=dtype)
            print(type(input))

            start = time()
            np.linalg.cholesky(input)
            stop = time()
            flops = (n**3) / 3 + 2 * n / 3
            total = (stop - start) / 1000.0
            print(f"Elapsed Time: {total} ms")
            print(f"{flops / total} GOP/s")

        cholesky(20, "float32")

<class 'cunumeric.array.ndarray'>
Elapsed Time: 0.568 ms
4718.30985915493 GOP/s
```

Jupyter & Google Colab



Comparison Table

Here is a list of NumPy APIs and corresponding cuNumeric implementations.

A dot in the cunumeric column denotes that cuNumeric implementation is not provided yet. We welcome contributions for these functions.

NumPy vs cuNumeric APIs

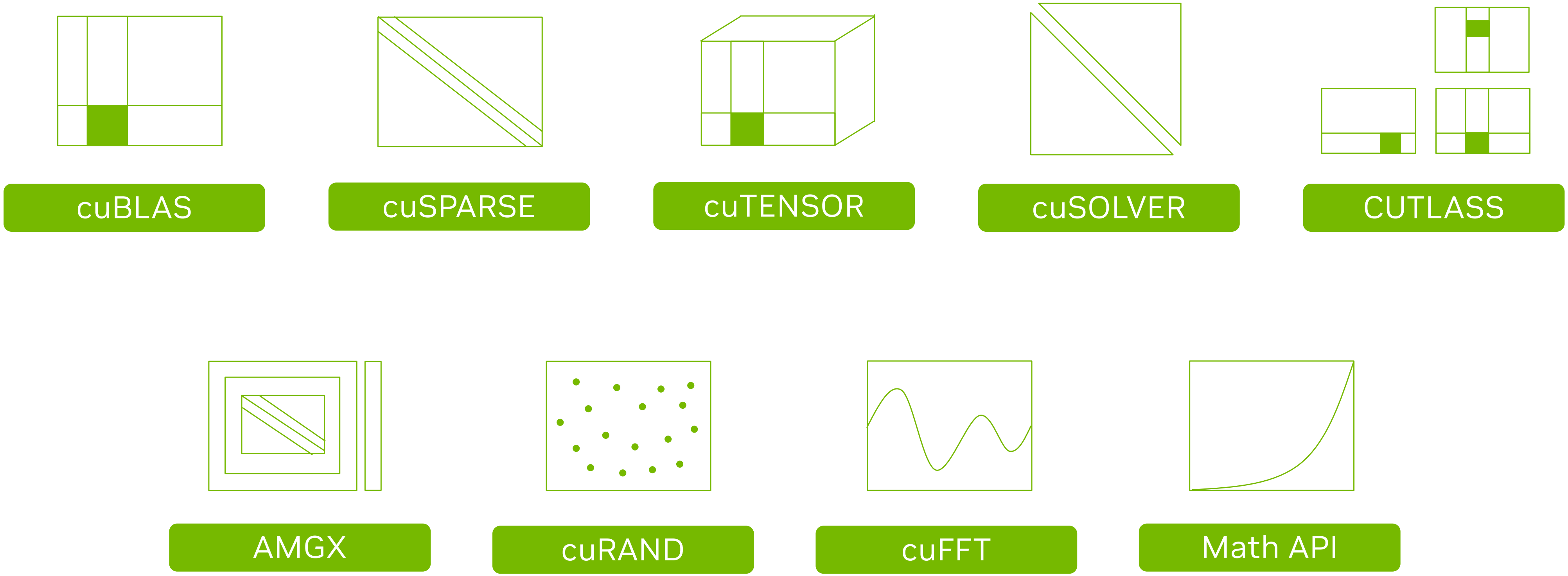
Module-Level

NumPy	cunumeric	single-GPU/CPU	multi-GPU/CPU
`numpy.all`	`cunumeric.all`	✓	✓
`numpy.allclose`	`cunumeric.allclose`	✓	✓
`numpy.amax`	`cunumeric.amax`	✓	✓
`numpy.amin`	`cunumeric.amin`	✓	✓
`numpy.angle`	•		
`numpy.any`	`cunumeric.any`	✓	✓
`numpy.append`	`cunumeric.append`	✓	✓
`numpy.apply_along_axis`	•		
`numpy.apply_over_axes`	•		

Accelerated Computing Libraries

NVIDIA Math Libraries

Linear Algebra, FFT, RNG, and Basic Math



cuBLAS

cuSPARSE

cuTENSOR

cuSOLVER

CUTLASS

AMGX

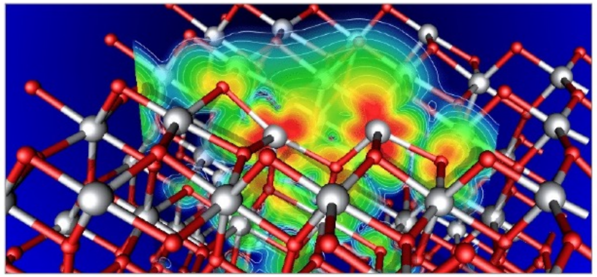
cuRAND

cuFFT

Math API

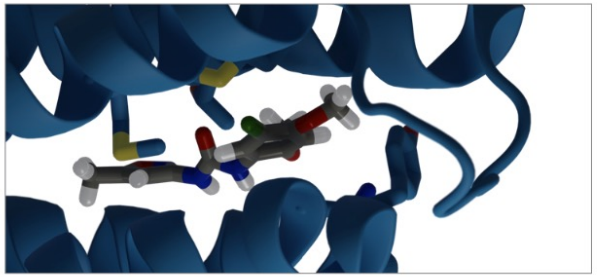
Multi GPU Multi-Node (MGMN) libraries

Enable Science At Scale



VASP

Atomic scale materials modelling



GROMACS

Molecular Dynamics Simulation

JAX

High-Performance Scalable Python

cuSOLVERMp

- Linear solvers (LU, Cholesky, QR)
- Symmetric Eigenvalue Solver
- UCC support

cuFFTMp

- Hopper support
- Improved interop via NVSHMEM

Distributed Symmetric Eigenvalue Solver

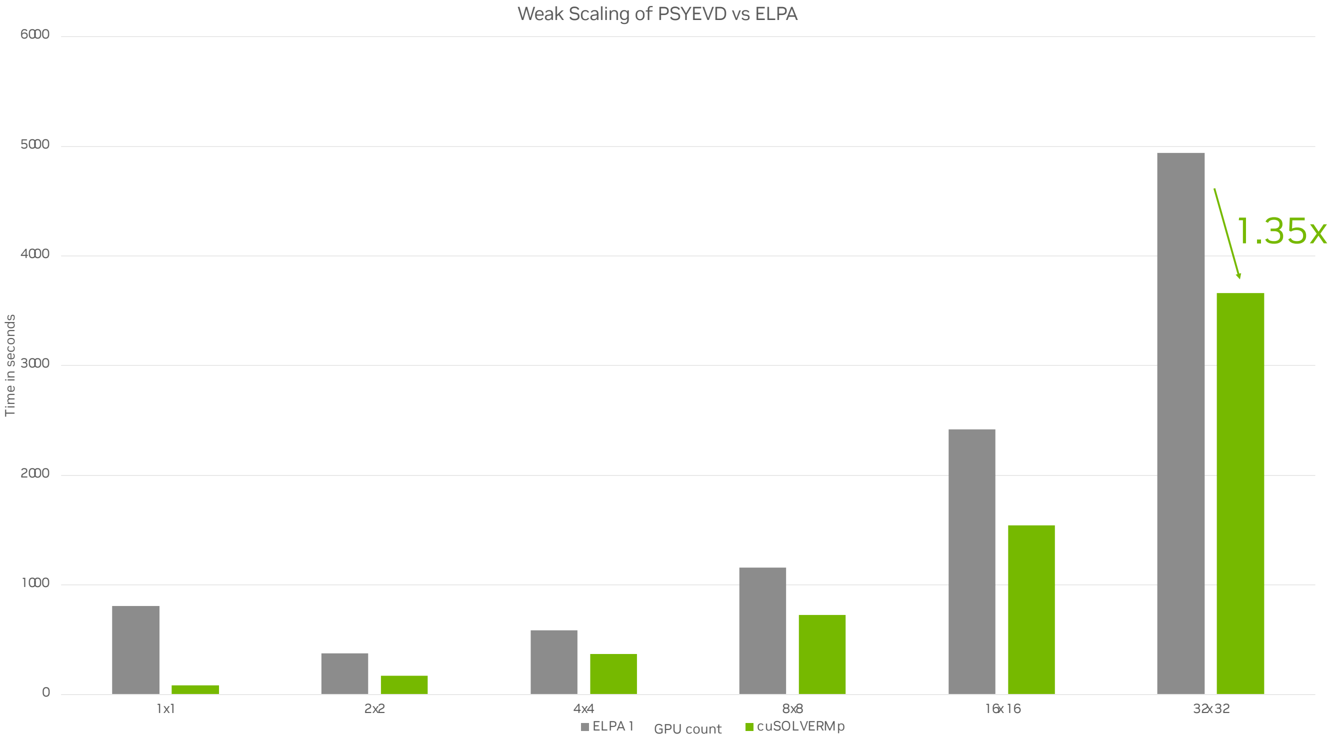
State of the art performance

ScaLAPACK

```
call blacs_get
Call blacs_gridinit

call descinit

call pzheevd(,-1, ,-1)
call pzheevd

call blacs_gridexit
call blasc_exit
```

cuSOLVERMp

```
cal_comm_create
cusolverMpCreate
cusolverMpCreateDeviceGrid

cusolverMpCreateMatrixDesc

cusolverMpSyevd_bufferSize
cusolverMpSyevd

cusolverMpDestroyMatrixDesc
cusolverMpDestroyGrid
cusolverMpDestroy
cal_comm_destroy
```



Performance measured on NVIDIA A100 DGX Super POD
~32k x 32k, real fp64 input matrix per GPU

cuSOLVERMp library samples on github

cuSOLVERMp is coming to VASP

Up to 1.5x faster than ELPA

Coming soon in VASP!

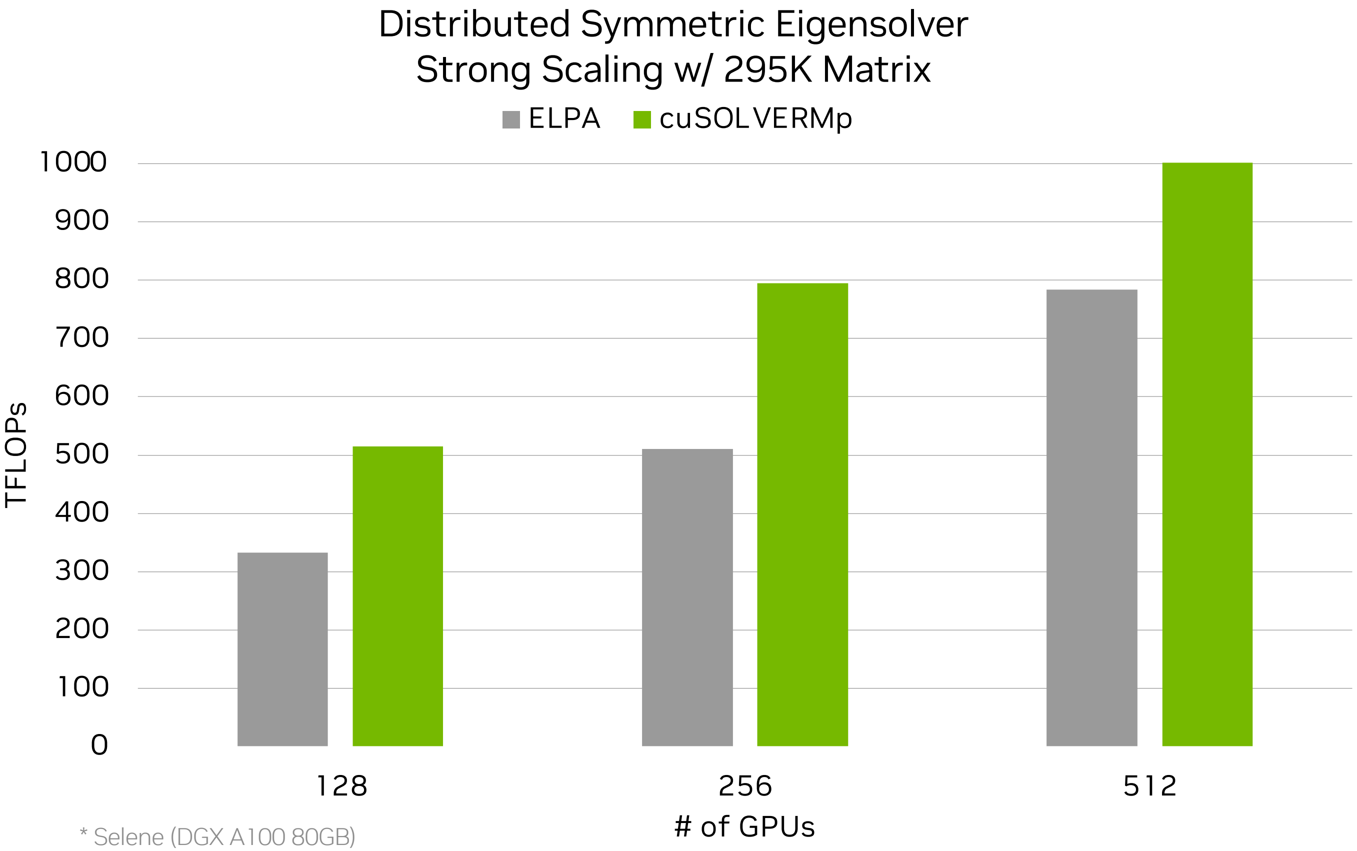
- Enables running the largest BSE calculation (576K) ever computed by the VASP group

Current Features

- LU with and without pivoting
- Cholesky

New Features for Q1’23

- QR Factorization
- LU and Cholesky support for multiple RHS
- Symmetric Eigensolver
  - Up to 1.5x faster than ELPA



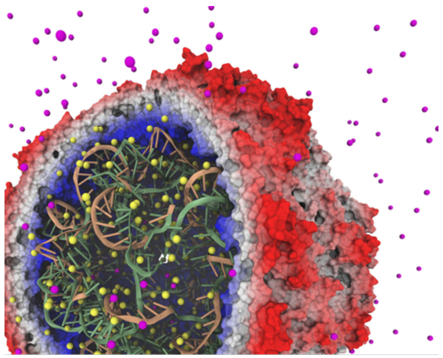
* Selene (DGX A100 80GB)

Large-Scale BSE Calculations for Solar-Panel Materials in VASP on GPUs with cuSolverMp

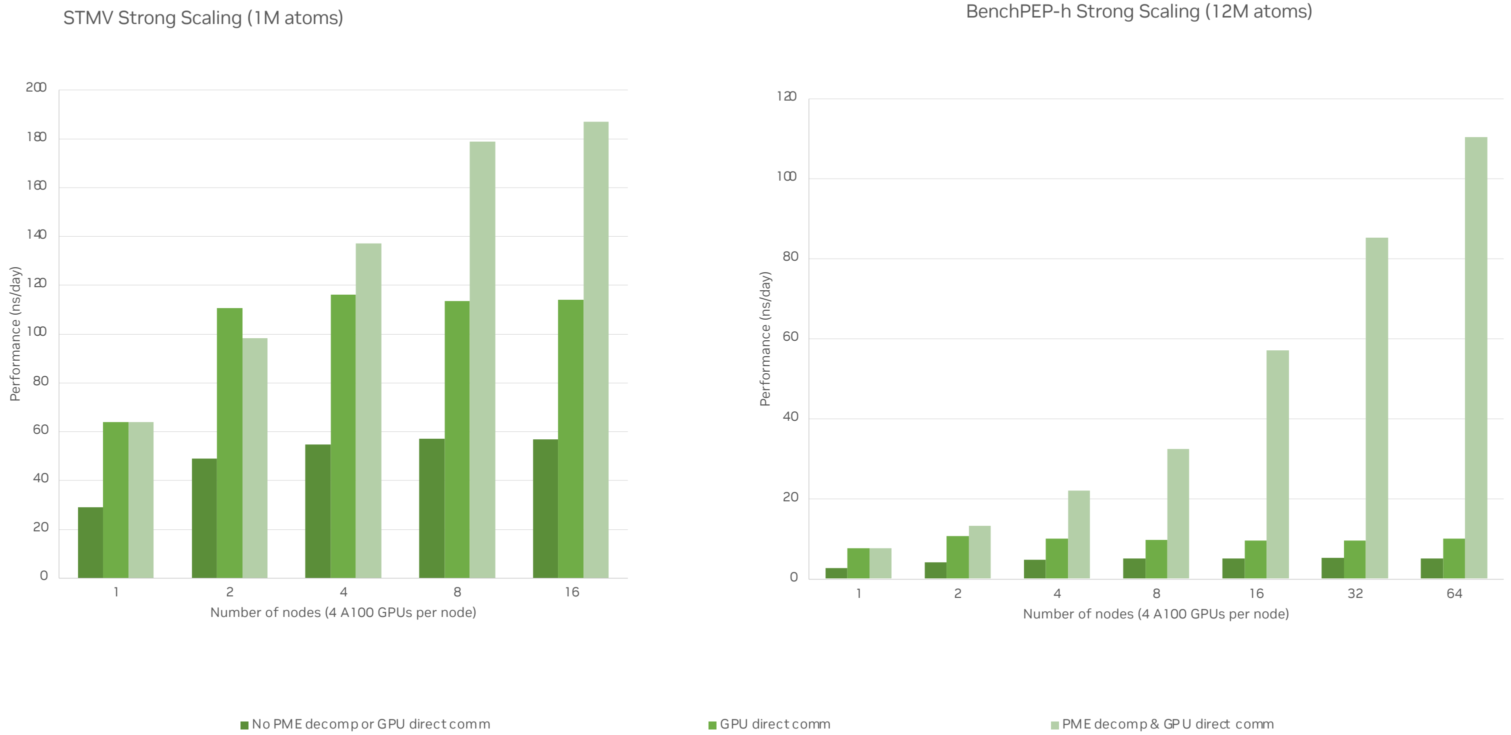
cuFFTMp + GROMACS

Accelerating Molecular Dynamics

FAST. FLEXIBLE. FREE.
GROMACS



cuFFTMp



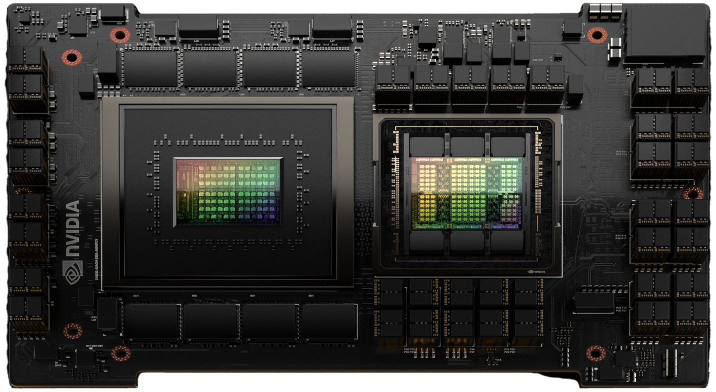
Performance measured on NVIDIA A100 DGX Super POD

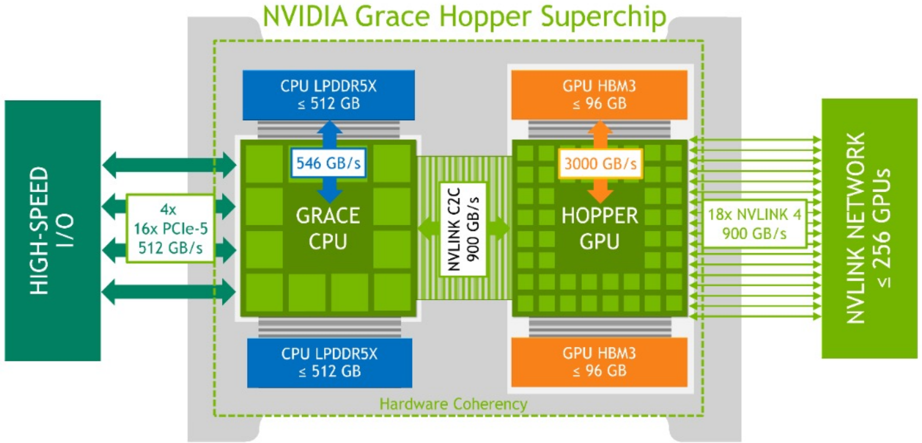
Massively Improved Multi-node NVIDIA GPU Scalability with GROMACS

Arm Software Stack

Grace Hopper Superchip

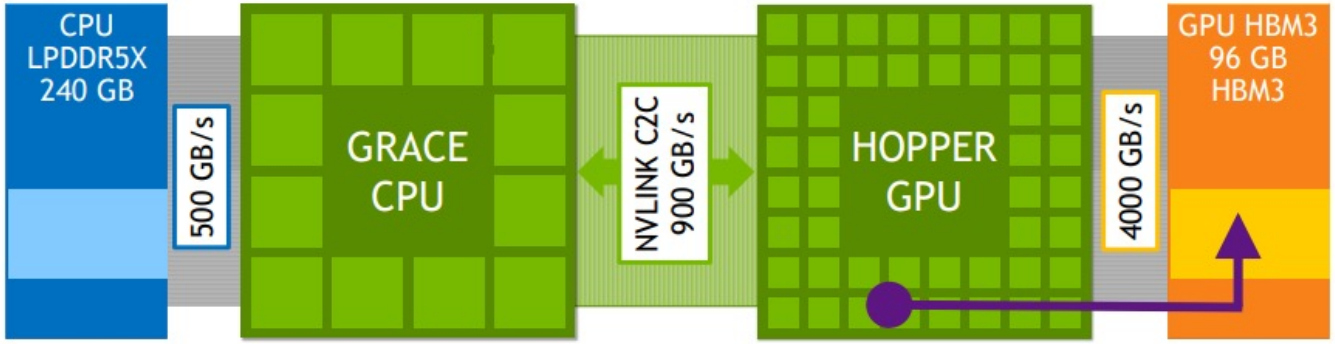
Programming Model and Applications for the Grace Hopper Superchip

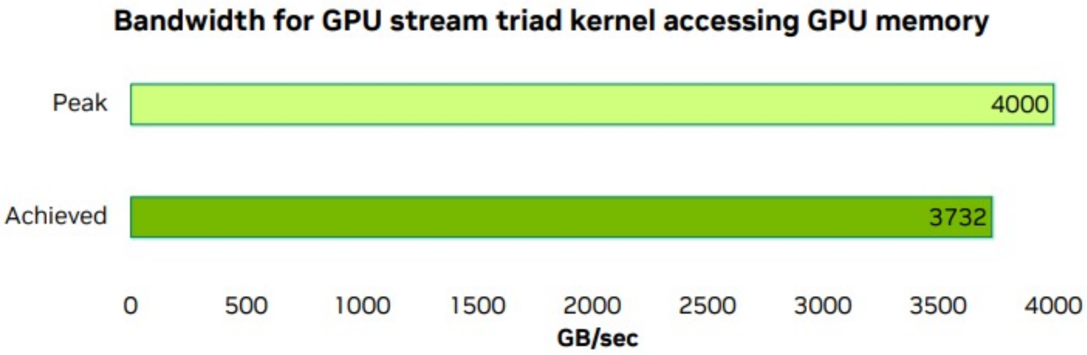




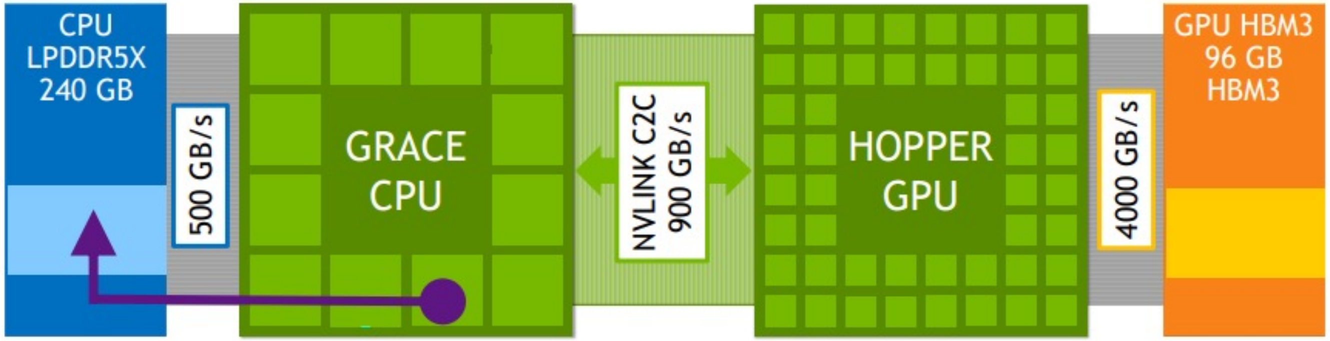
Grace Hopper Superchip	
GPU	Hopper 96GB HBM3
GPU Memory Bandwidth	4 TB/s
CPU	72 Arm Neoverse-V2 Cores
CPU Memory	Up to 480GB LPDDR5X
CPU Memory Bandwidth	Up to 500 GB/s
CPU to GPU NVLink C2C	900GB/s, cache coherent
TDP	700W

High Bandwidth Memory Access & Automatic Data Migration

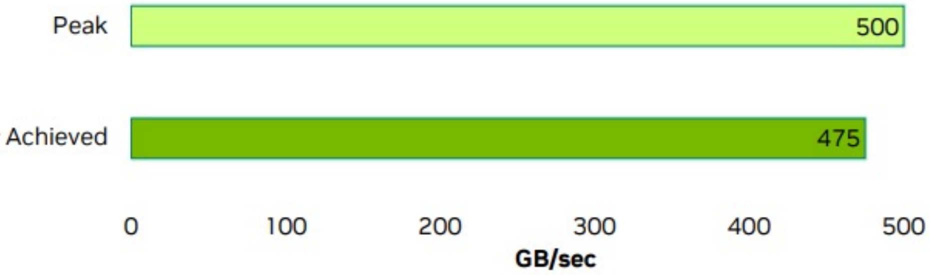




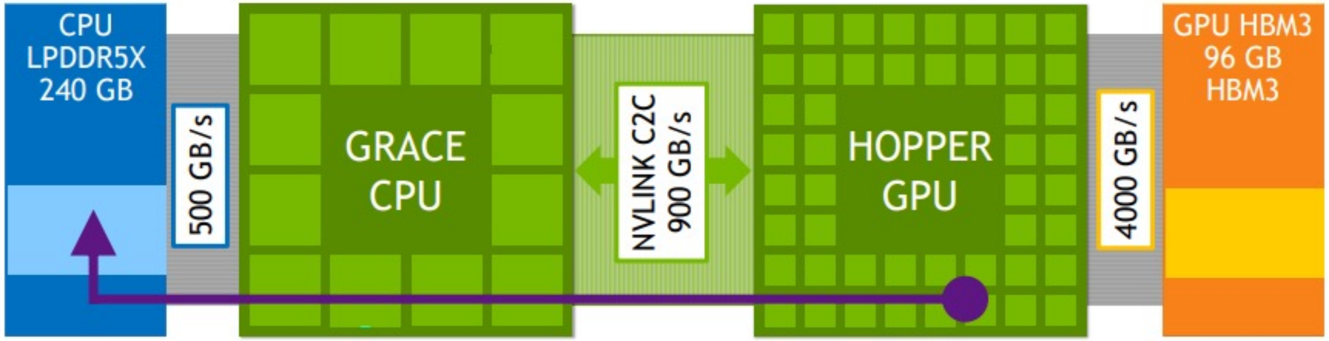
High Bandwidth Memory Access & Automatic Data Migration



Bandwidth for CPU stream accessing CPU memory



High Bandwidth Memory Access & Automatic Data Migration



CPU LPDDR5X 240 GB

500 GB/s

GRACE CPU

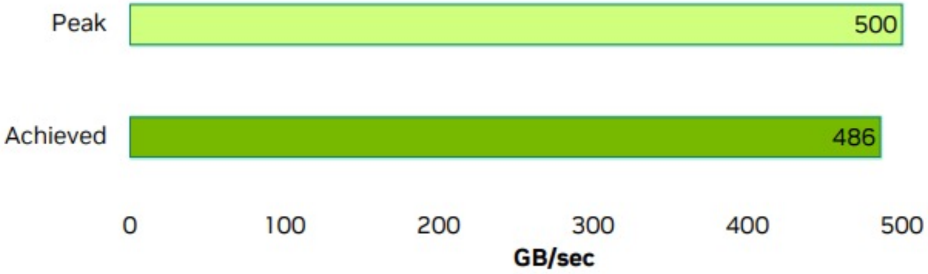
NVLINK C2C 900 GB/s

HOPPER GPU

4000 GB/s

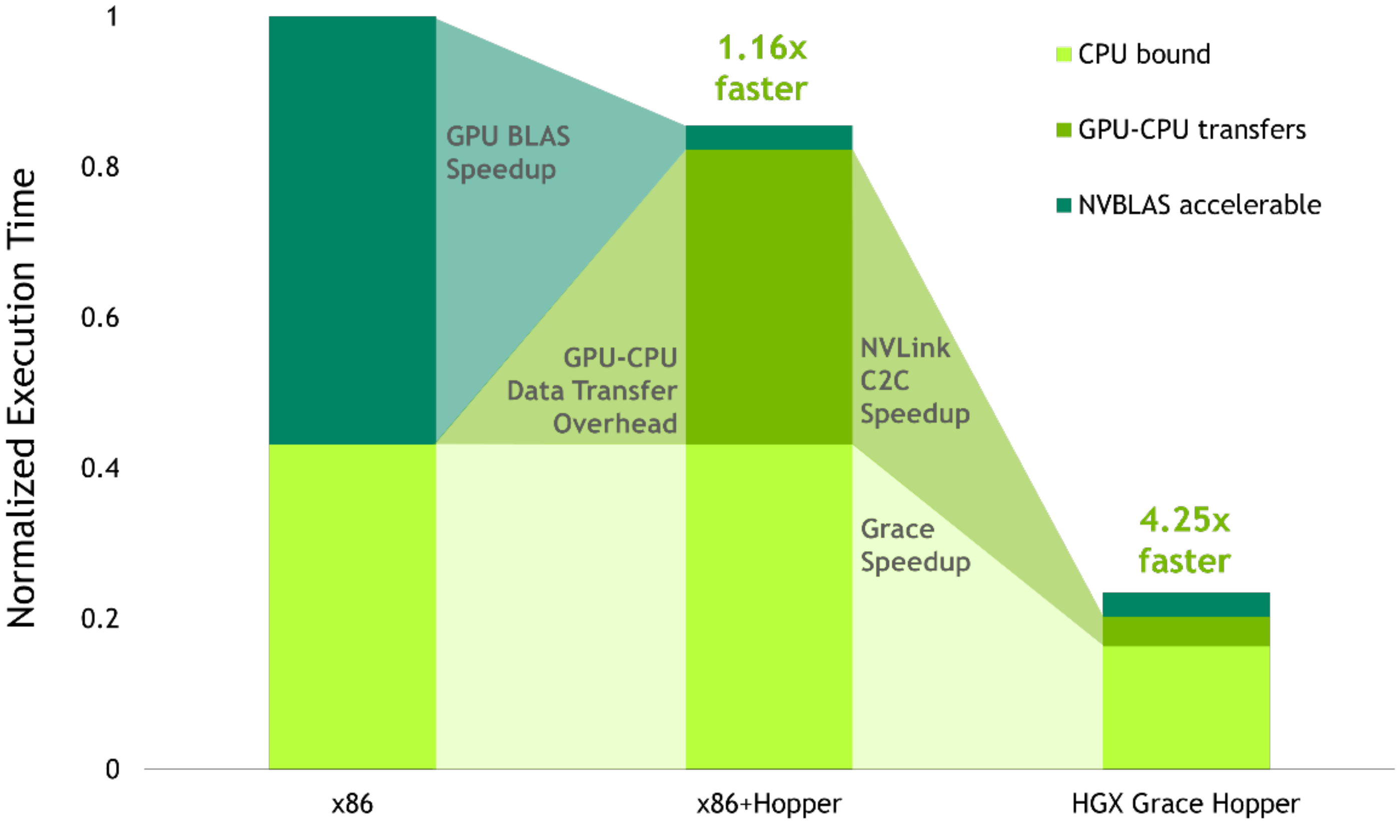
GPU HBM3 96 GB HBM3

Bandwidth for GPU stream kernel accessing CPU memory



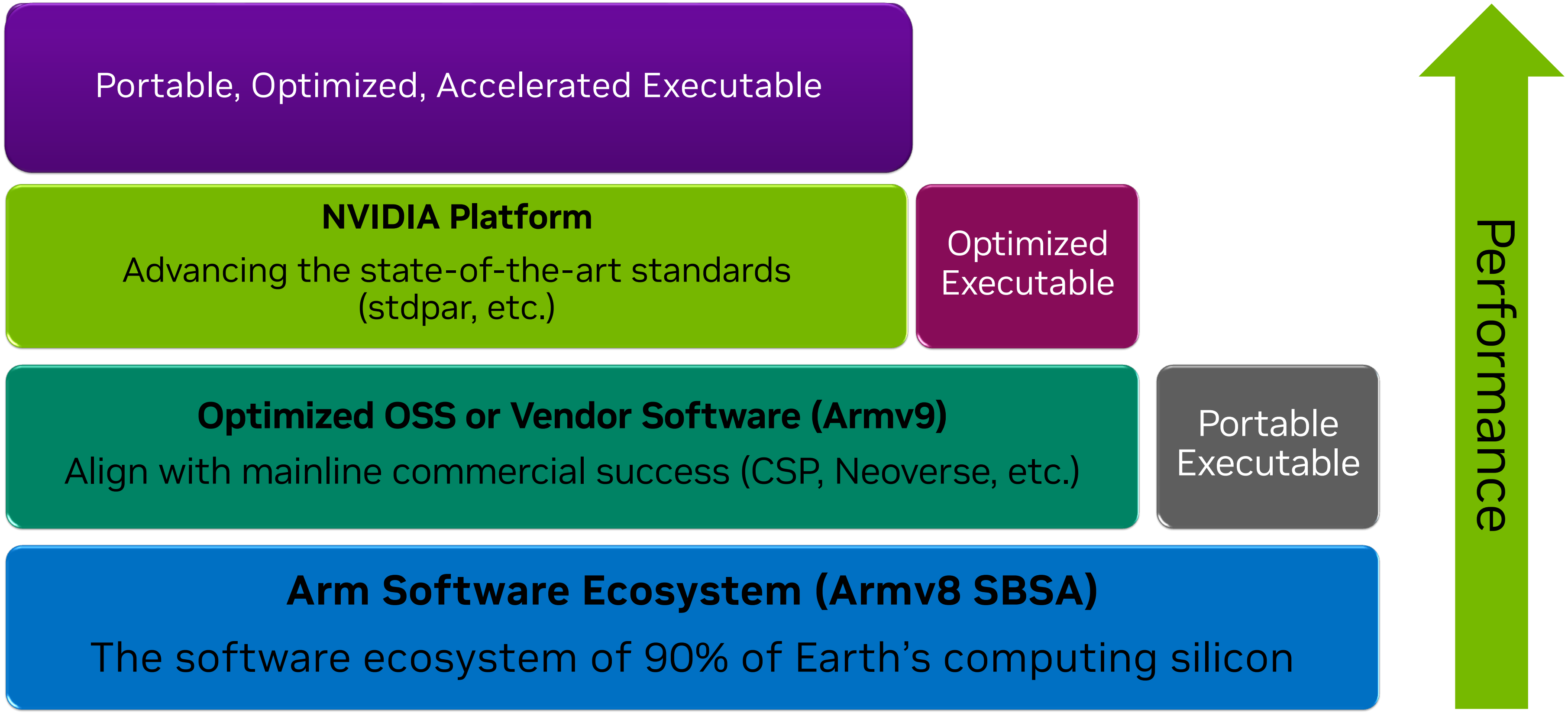
ABINIT

Titanium 255 Atoms using the LOBPCG algorithm



Grace Software Ecosystem is Built on Standards

The NVIDIA platform builds on optimized software from the broad Arm software ecosystem



Portable, Optimized, Accelerated Executable

NVIDIA Platform

Advancing the state-of-the-art standards (stdpar, etc.)

Optimized Executable

Optimized OSS or Vendor Software (Armv9)

Align with mainline commercial success (CSP, Neoverse, etc.)

Portable Executable

Arm Software Ecosystem (Armv8 SBSA)

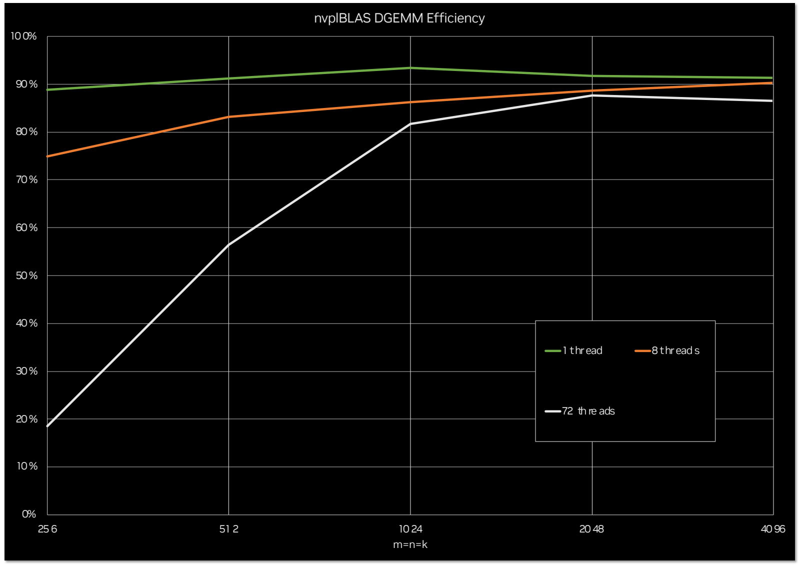
The software ecosystem of 90% of Earth's computing silicon

Performance

NVIDIA Performance Libraries (NVPL)

Math Libraries Optimized for Arm CPUs

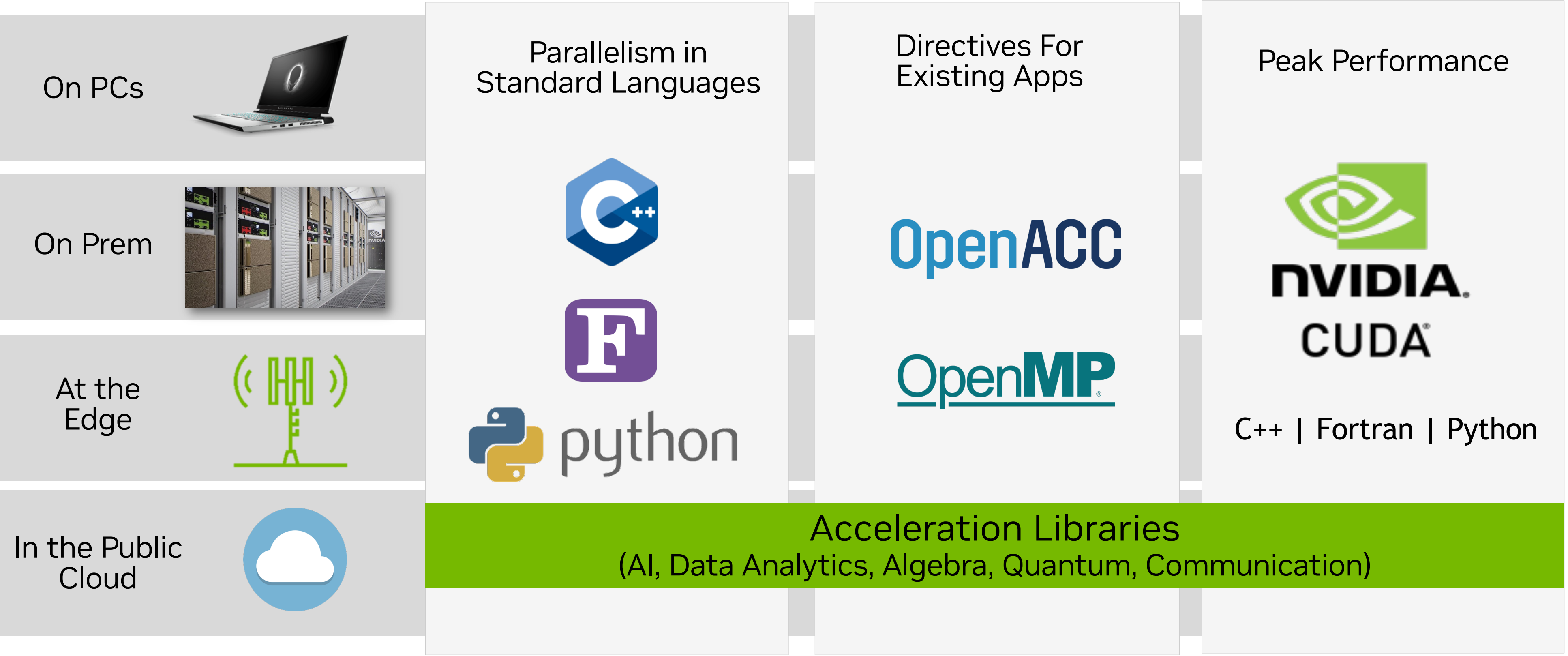
- Enable easy porting of HPC applications to NVIDIA Grace CPU based platforms to achieve industry leading performance and efficiency
  - Standard interfaces (e.g., BLAS, FFTW)
  - New interfaces (e.g., SPARSE, TENSOR)
- Early access in H2'2023



BLAS	LAPACK	PBLAS	SCALAPACK
TENSOR	SPARSE	RNG	FFTW

Programming the NVIDIA Platform

Unmatched Developer Flexibility



On PCs

On Prem

At the Edge

In the Public Cloud

Parallelism in Standard Languages

C++

F

python

Directives For Existing Apps

OpenACC

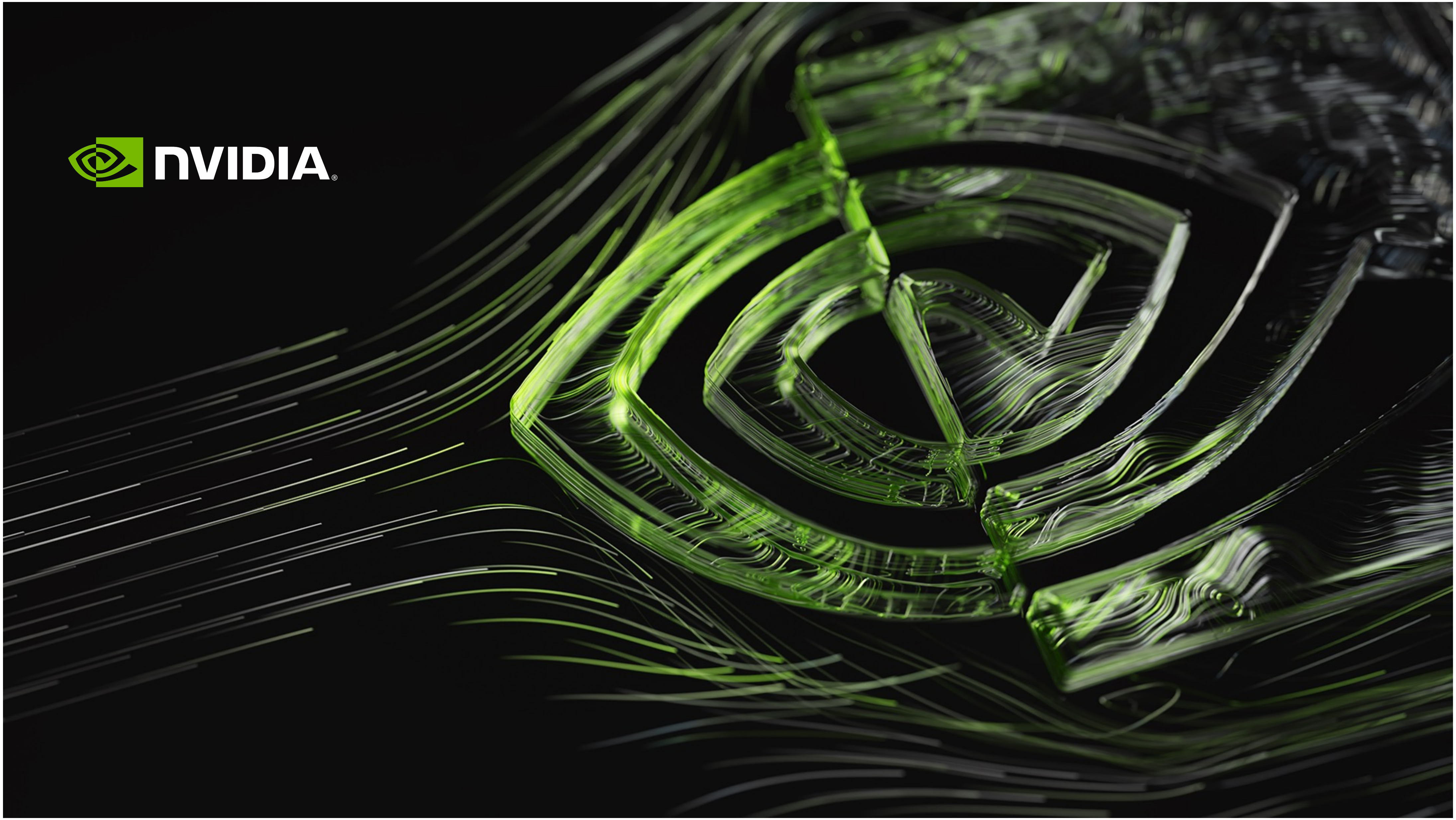
OpenMP

Peak Performance

NVIDIA CUDA

C++ | Fortran | Python

Acceleration Libraries
(AI, Data Analytics, Algebra, Quantum, Communication)



NVIDIA®