物性シミュレーションのための GPU コンピューティング

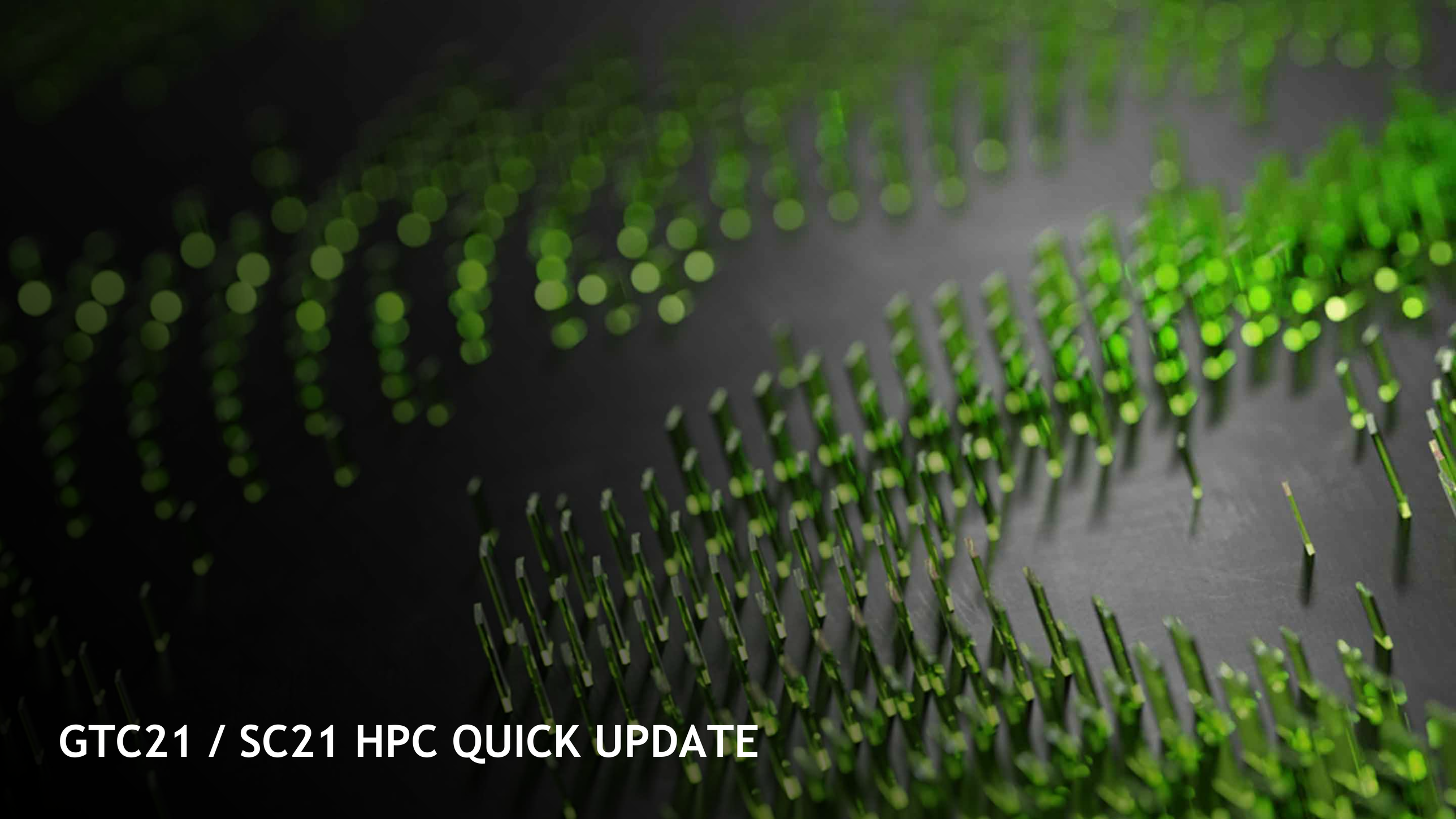
SHINNOSUKE FURUYA, PH.D., HPC DEVELOPER RELATIONS

2021/11/24

NVIDIA

Founded in 1993 | Jensen Huang, Founder & CEO | 50+ Offices | 19,000 Employees | $16.7B in FY21

NVIDIA


GTC21 / SC21 HPC QUICK UPDATE

スパコンランキング TOP500

上位 7 / 10 が NVIDIA GPU を搭載

システム名	概要	サイト	性能 (TFlops)
2: Summit	IBM POWER, NVIDIA V100, NVIDIA Mellanox IB EDR	アメリカ	148,600.0
3: Sierra	IBM POWER, NVIDIA V100, NVIDIA Mellanox IB EDR	アメリカ	94,640.0
5: Perlmutter	AMD EPYC, NVIDIA A100, HPE Slingshot	アメリカ	70,870.0
6: Selene	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	アメリカ	63,460.0
8: JUWELS Booster Module	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	ドイツ	44,120.0
9: HPC5	Intel Xeon, NVIDIA V100, NVIDIA Mellanox IB HDR	イタリア	35,450.0
10: Voyager-EUS2	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	アメリカ	30,050.0

https://www.top500.org

スパコンランキング GREEN500

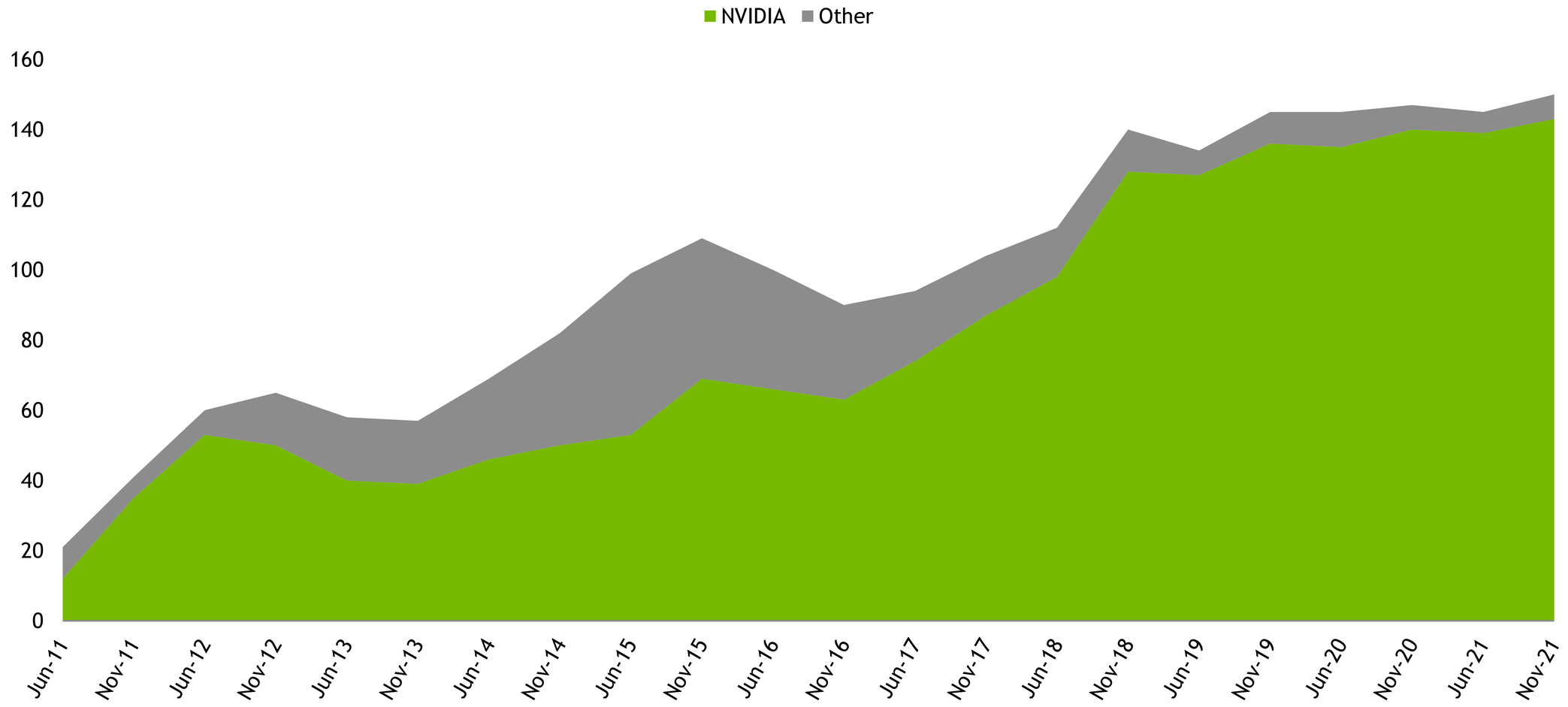
上位 9 / 10 が NVIDIA GPU を搭載

システム名	概要	サイト	電力効率 (GFlops/W)
2: SSC-21 Scalable Module	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR200	韓国	33.983
3: Tethys	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	アメリカ	31.538
4: Wilkes-3	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR200	イギリス	30.797
5: HiPerGator AI	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	アメリカ	29.521
6: Snellius Phase 1 GPU	Intel Xeon, NVIDIA A100, NVIDIA Mellanox IB HDR	オランダ	29.046
7: Perlmutter	AMD EPYC, NVIDIA A100, HPE Slingshot	アメリカ	27.374
8: Karolina, GPU partition	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR200	チェコ	27.213
9: MeluXina - Accelerator Module -	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	ルクセンブルク	26.957
10: NVIDIA DGX SuperPOD	AMD EPYC, NVIDIA A100, NVIDIA Mellanox IB HDR	アメリカ	26.195

https://www.top500.org

スパコンランキング TOP500

アクセラレータのトレンドは NVIDIA GPU



ANNOUNCING
NVIDIA cuNUMERIC

Accelerated Computing At-Scale for PyData and NumPy Ecosystem

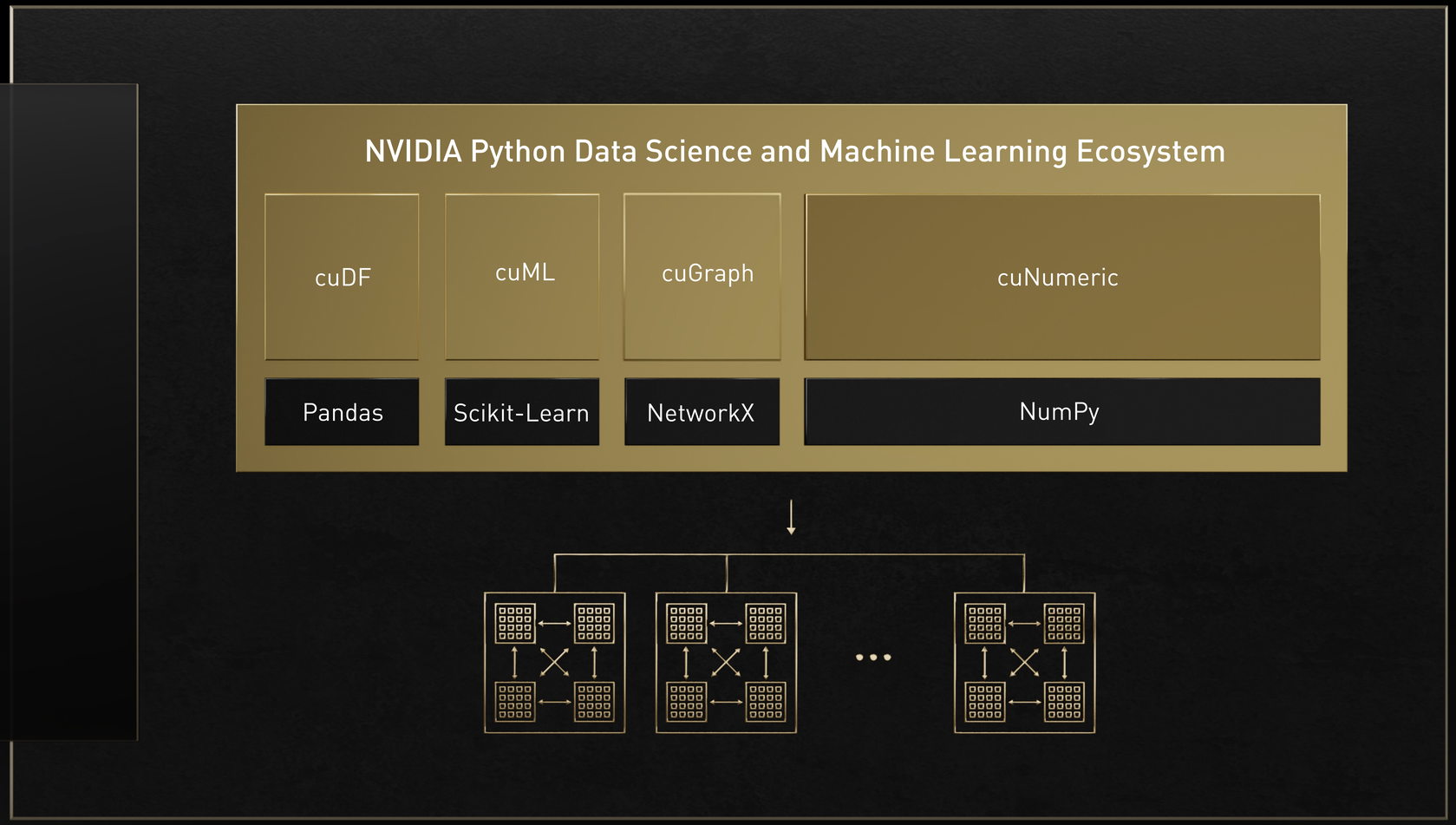
Transparently Accelerates and Scales NumPy Workflows

Zero Code Changes

Automatic Parallelism and Acceleration for Multi-GPU, Multi-Node Systems

Scales to 1,000s of GPUs

Available Now on GitHub and Conda



ANNOUNCING
NVIDIA MODULUS
Physics-ML Neural Simulation Framework

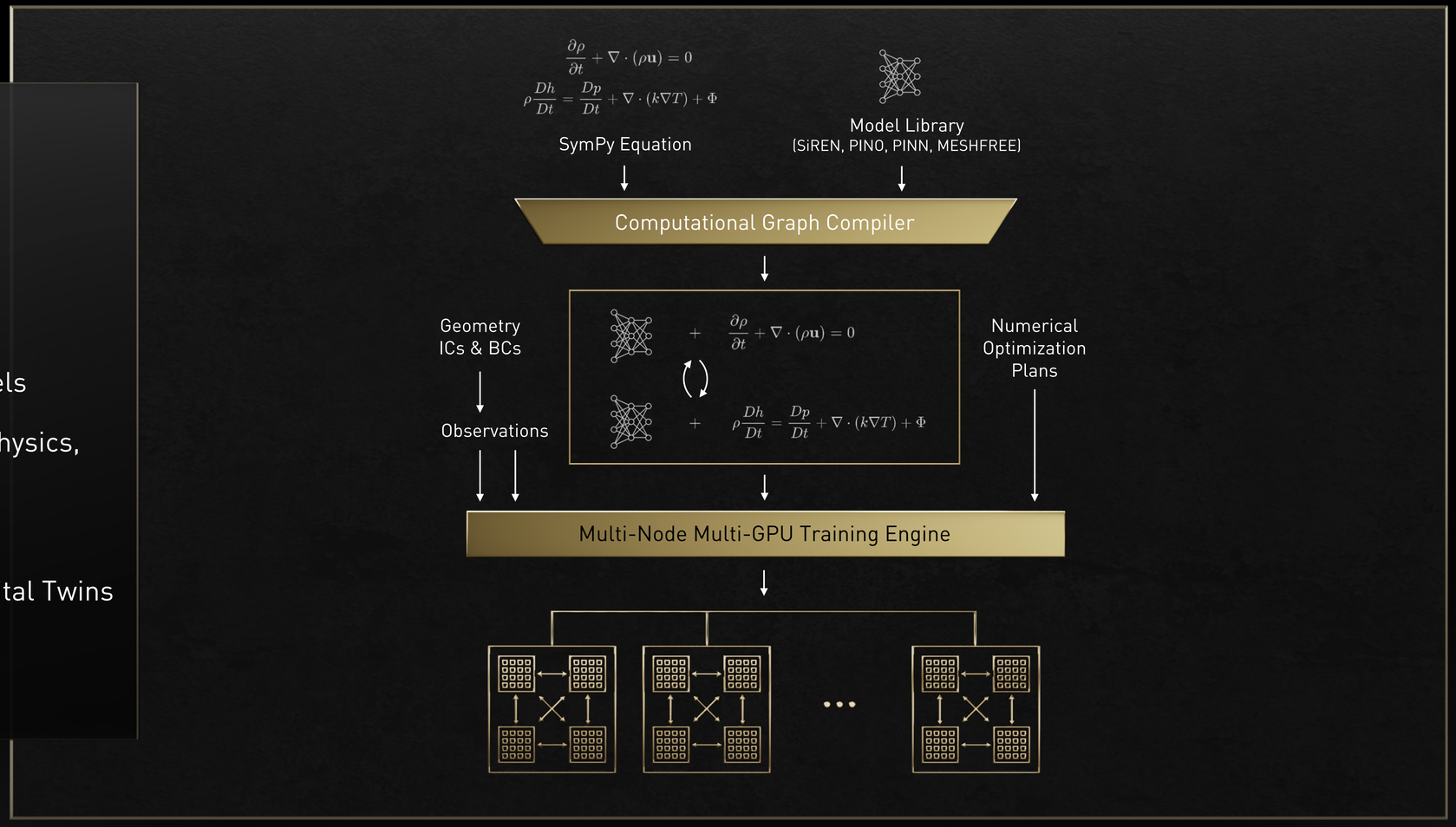
Framework for Developing Physics-ML Models

Train Physics-ML Models Using Governing Physics, Simulation, and Observed Data

Multi-GPU, Multi-Node Training

1,000-100,000X Speed Models – Ideal for Digital Twins

Available Now
developer.nvidia.com/modulus



HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Array Syntax and Intrinsics

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

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

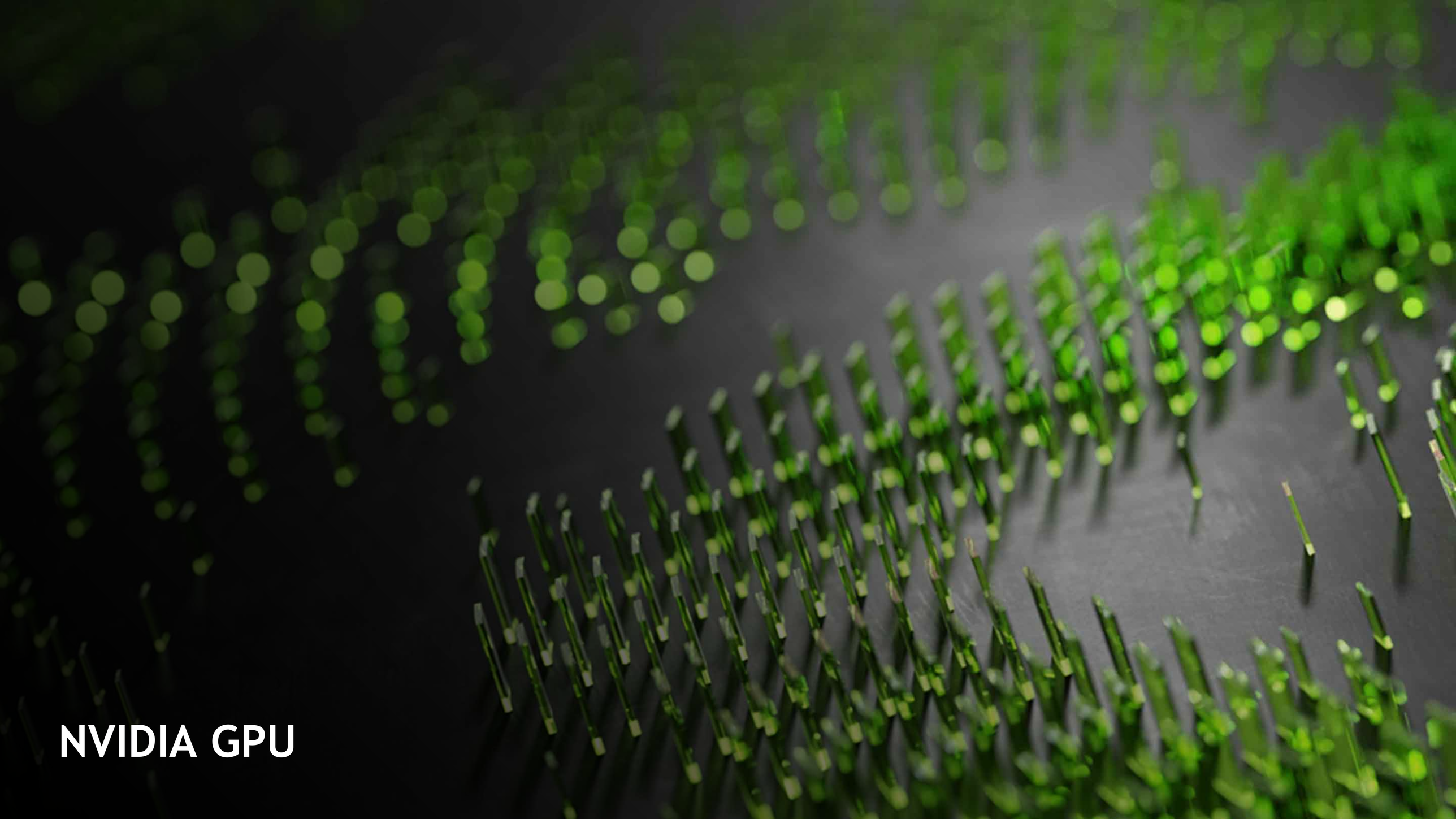
- Coming Soon
- Accelerated co-array images

Preview support available now in NVFORTRAN

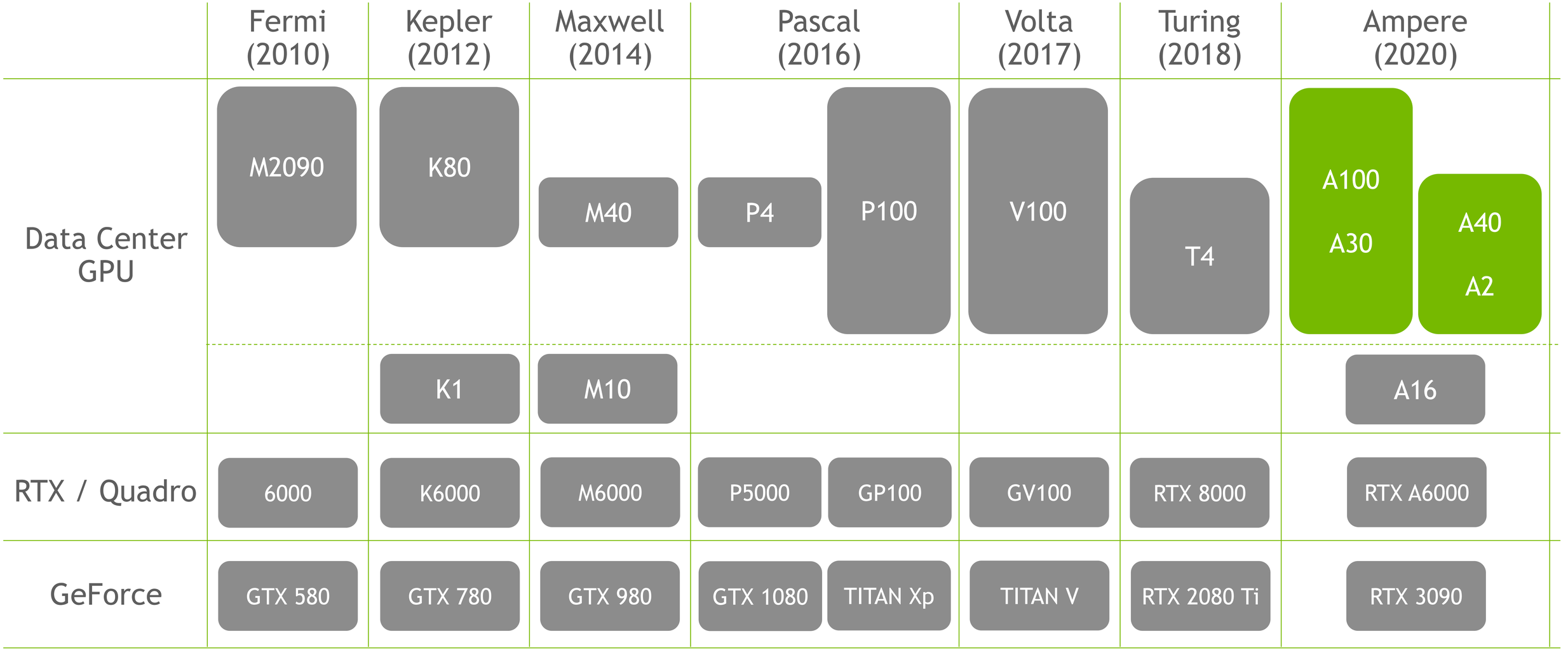
Fortran 202x

DO CONCURRENT Reductions

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


NVIDIA GPU

NVIDIA GPUS AT A GLANCE



DATA CENTER PRODUCT COMPARISON (NOV 2021)

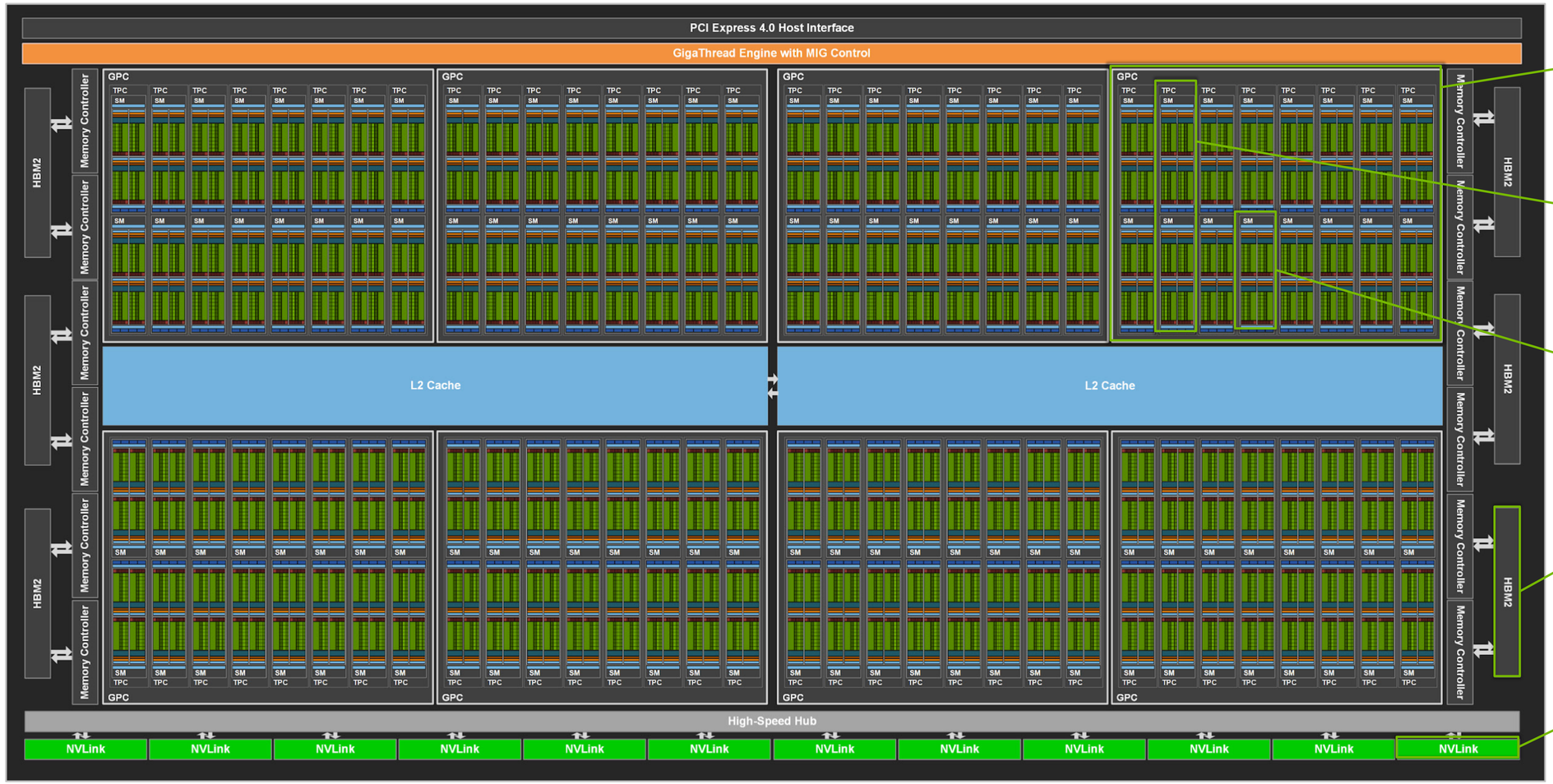
		A100		A30	A40	A2
Performance	FP64 (no Tensor Core)	9.7 TFlops		5.2 TFlops	-	-
	FP64 Tensor Core	19.5 TFlops		10.3 TFlops	-	-
	FP32 (no Tensor Core)	19.5 TFlops		10.3 TFlops	37.4 TFlops	4.5 TFlops
	TF32 Tensor Core	156 | 312* TFlops		82 | 165* TFlops	74.8 | 149.6* TFlops	9 | 18* TFlops
	FP16 Tensor Core	312 | 624* TFlops		165 | 330* TFlops	149.7 | 299.4* TFlops	18 | 36* TFlops
	BF16 Tensor Core	312 | 624* TFlops		165 | 330* TFlops	149.7 | 299.4* TFlops	18 | 36* TFlops
	Int8 Tensor Core	624 | 1248* TOPS		330 | 661* TOPS	299.3 | 598.6* TOPS	36 | 72* TOPS
	Int4 Tensor Core	1248 | 2496* TOPS		661 | 1321* TOPS	598.7 | 1197.4* TOPS	72 | 144* TOPS
Form Factor		SXM4 module on base board	x16 PCIe Gen4 2 Slot FHFL 3 NVLink bridge	x16 PCIe Gen4 2 Slot FHFL 1 NVLink bridge	x16 PCIe Gen4 2 Slot FHFL 1 NVLink bridge	x8 PCIe Gen4 1 Slot LP
GPU Memory		80 GB HBM2e	80 GB HBM2e	24 GB HBM2	48 GB GDDR6	16 GB GDDR6
GPU Memory Bandwidth		1935 GB/s	2039 GB/s	933 GB/s	696 GB/s	200 GB/s
Multi-Instance GPU		Up to 7	Up to 7	Up to 4	-	-
Media Acceleration		1 JPEG Decoder 5 Video Decoder		1 JPEG Decoder 4 Video Decoder	1 Video Encoder 2 Video Decoder (+AV1 decode)	1 Video Encoder 2 Video Decoder (+AV1 Decode)
Ray Tracing		No	No	No	Yes	Yes
Graphics		For in-situ visualization (no NVIDIA vPC or RTX vWS)			Best	Good
Max Power		400 W	300 W	165 W	300 W	40-60 W

* Performance with structured sparse matrix

NVIDIA

AMPERE GPU ARCHITECTURE

A100 Tensor Core GPU



7 GPCs

7 or 8 TPCs/GPC

2 SMs/TPC
(108 SMs/GPU)

5 HBM2 stacks

12 NVLink links

AMPERE GPU ARCHITECTURE

Streaming Multiprocessor (SM)

32 FP64 CUDA Cores

64 FP32 CUDA Cores

4 Tensor Cores

Up to 128 FP32 CUDA Cores

4 Tensor Cores

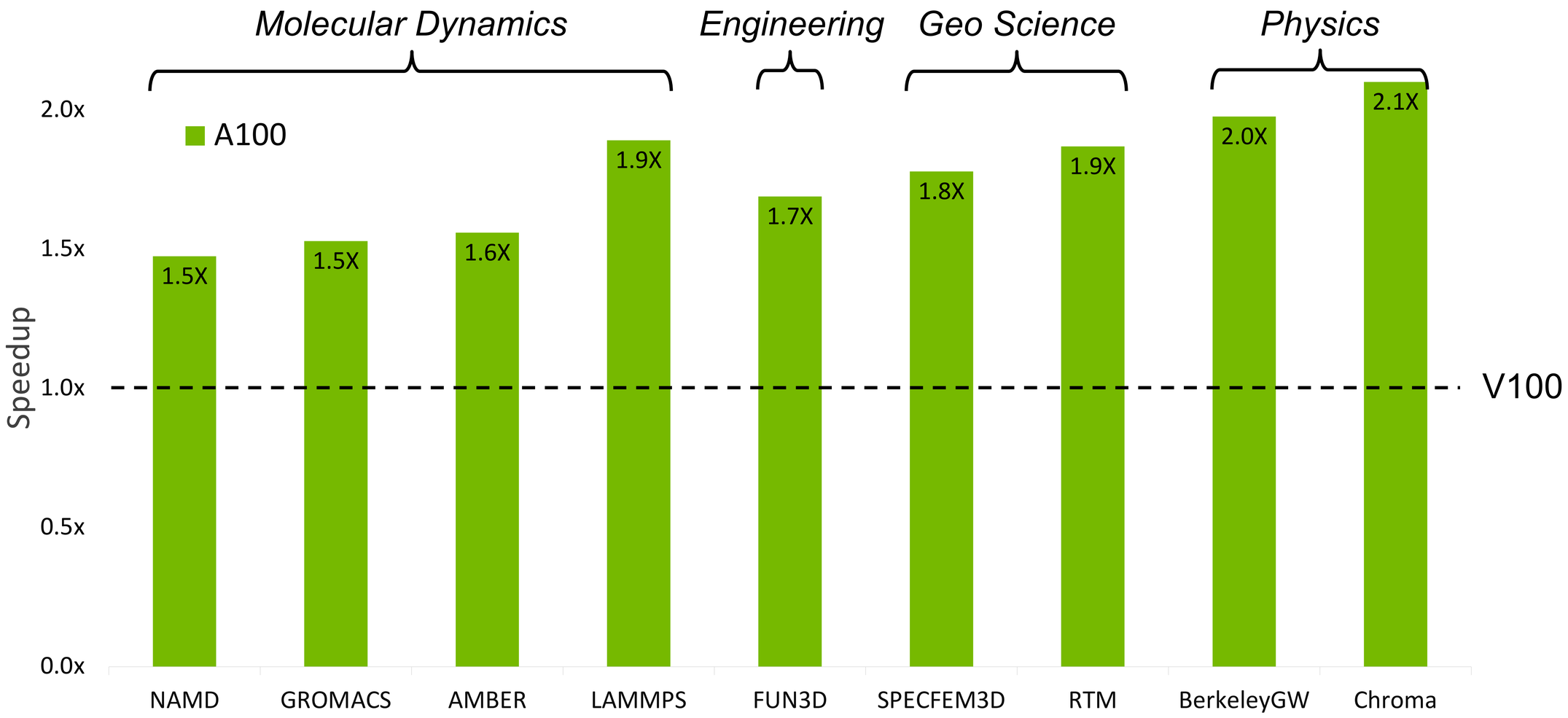
1 RT Core

GA100 (A100, A30)

GA102 (A40)

A100 HPC APPS PERFORMANCE

Over 2x More HPC Performance



All results are measured
Except BerkeleyGW, V100 used is single V100 SXM2. A100 used is single A100 SXM4
More apps detail: AMBER based on PME-Cellulose, GROMACS with STMV (h-bond), LAMMPS with Atomic Fluid LJ-2.5, NAMD with v3.0a1 STMV_NVE
Chroma with szscl21_24_128, FUN3D with dpw, RTM with Isotropic Radius 4 1024^3, SPECFEM3D with Cartesian four material model
BerkeleyGW based on Chi Sum and uses 8xV100 in DGX-1, vs 8xA100 40GB in DGX A100

NGC CATALOG - GPU-OPTIMIZED SOFTWARE

Build AI Faster, Deploy Anywhere

100 + CONTAINERS

HPC | DL | ML

50+ PRE-TRAINED MODELS

COMPUTER VISION | NLP | DLRM

INDUSTRY APP FRAMEWORKS

CLARA | Riva | ISAAC

HELM CHARTS

TRITON | GPU OPERATOR

COLLECTIONS

CLARA DISCOVERY | TLT-Riva | RECSYS

x86 | ARM | POWER

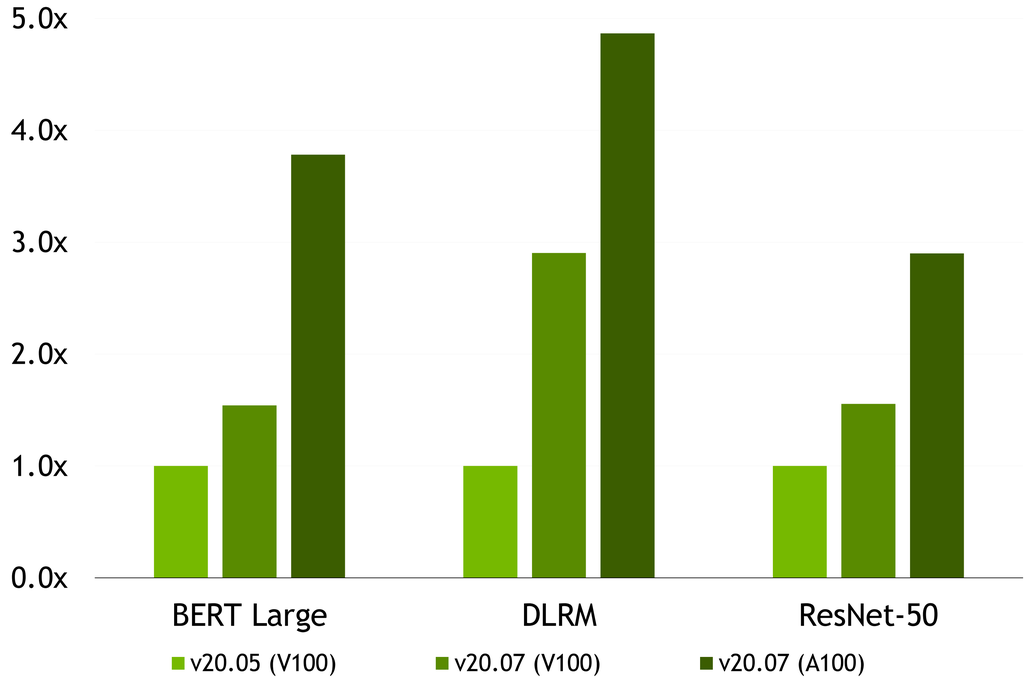
ON-PREM

CLOUD

HYBRID CLOUD

EDGE

NGC CONTAINERS ENABLE YOU TO FOCUS ON BUILDING AI



ENTERPRISE READY SOFTWARE

Scanned for CVEs, malware, crypto

Tested for reliability

Backed by Enterprise support

PERFORMANCE OPTIMIZED

Scalable

Updated Monthly

Better performance on the same system

DEPLOY ANYWHERE

Docker | cri-o | containerd | Singularity

Bare metal, VMs, Kubernetes

Multi-cloud, on-prem, hybrid, edge

BERT-Large and ResNet-50 v1.5 Training performance with TensorFlow on a single node 8x V100 (32GB) & A100 (40GB). Mixed Precision. Batch size for BERT: 10 (V100), 24 (A100), ResNet: 512 (V100, v20.05), 256 (v20.07)
DLRM Training performance with PyTorch on 1x V100 & 1x A100. Mixed Precision. Batch size 32768. DRLM trained with v20.03 and v20.07

ACCELERATING EVERY CLOUD

Over 30 Offerings Across USA and China

	K520	K80	P40	M60	P4	P100	T4	V100	A100	NGC
Alibaba Cloud					●	●	●	●		●
AWS	●	●		●			●	●	●	●
Baidu Cloud			●		●		●	●		
Google Cloud		●			●	●	●	●	●	●
IBM Cloud		●		●		●		●		
Microsoft Cloud		●	●	●		●		●	●	●
Oracle Cloud						●		●	●	●
Tencent Cloud			●		●		●	●		


PROGRAMMING

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

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

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}

#pragma omp target data map(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

PLATFORM SPECIALIZATION

CUDA

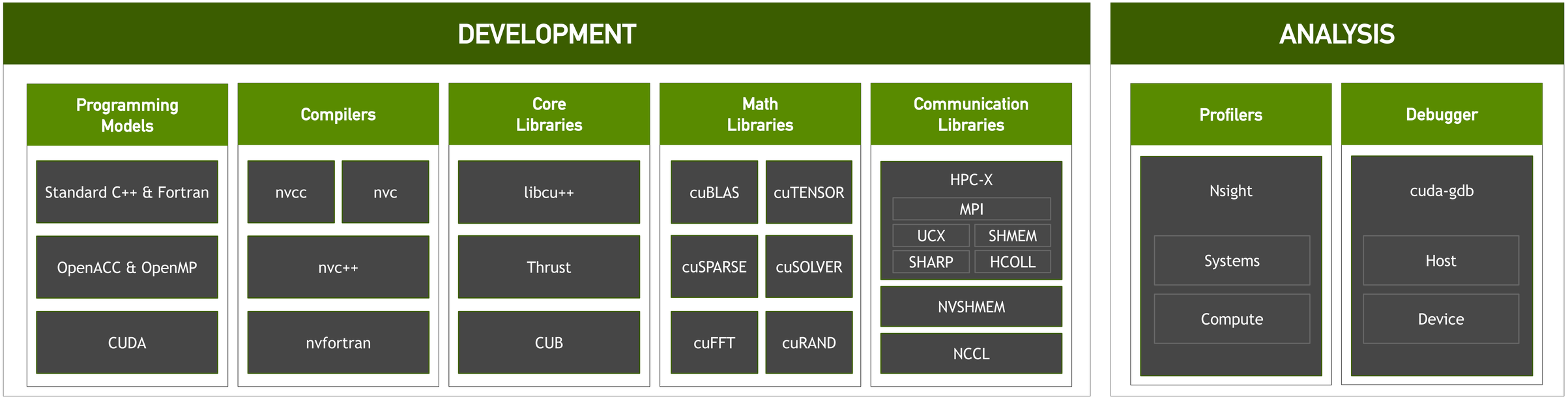
```
__global__
void saxpy(int n, float a,
            float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

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

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

ACCELERATION LIBRARIES

Core	Math	Communication	Data Analytics	AI	Quantum

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

HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

Preview support coming to NVC++

C++17

Parallel Algorithms

- In NVC++
- Parallel and vector concurrency

Forward Progress Guarantees

- Extend the C++ execution model for accelerators

Memory Model Clarifications

- Extend the C++ memory model for accelerators

C++20

Scalable Synchronization Library

- Express thread synchronization that is portable and scalable across CPUs and accelerators
- In libcu++:
  - `std::atomic<T>`
  - `std::barrier`
  - `std::counting_semaphore`
  - `std::atomic<T>::wait/notify_*`
  - `std::atomic_ref<T>`

C++23 and Beyond

Executors / Senders-Recievers

- Simplify launching and managing parallel work across CPUs and accelerators

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Range-Based Parallel Algorithms

- Improved multi-dimensional loops

Linear Algebra

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

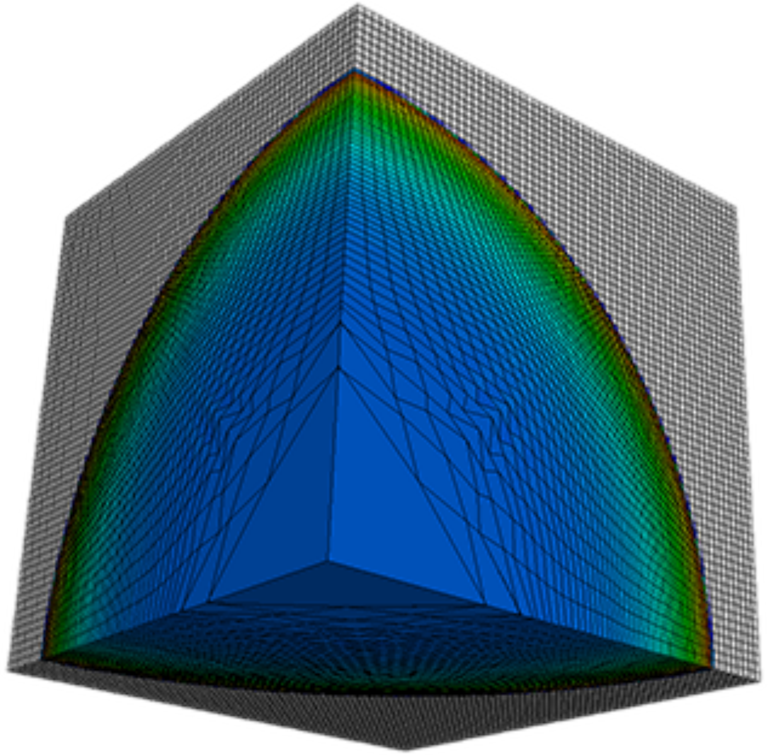
Extended Floating Point Types

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

C++17 PARALLEL ALGORITHMS

Lulesh Hydrodynamics Mini-app

- ~9000 lines of C++
- Parallel versions in MPI, OpenMP, OpenACC, CUDA, RAJA, Kokkos, ISO C++…
- Designed to stress compiler vectorization, parallel overheads, on-node parallelism



codesign.llnl.gov/lulesh

```
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

STANDARD C++

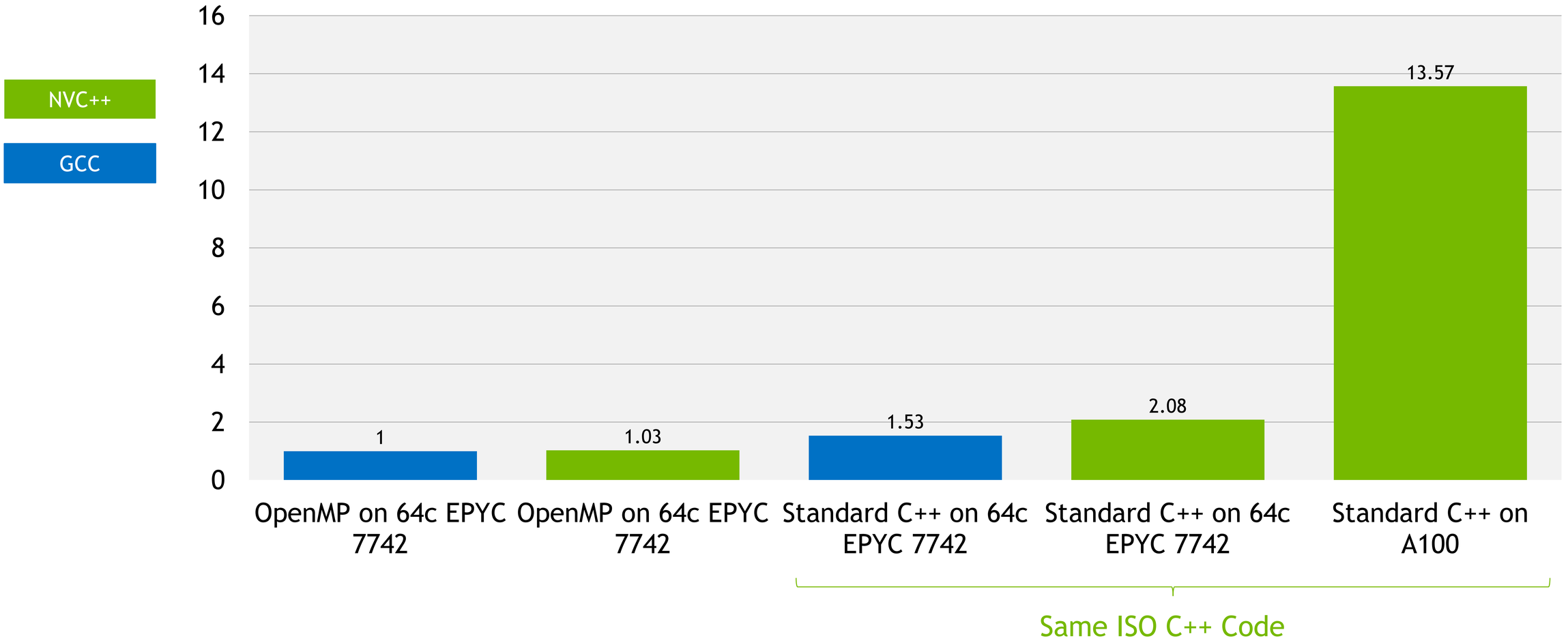
- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …

```
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));
      }
  });
}
```

Standard C++

C++ STANDARD PARALLELISM

Lulesh Performance



HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Array Syntax and Intrinsics

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

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

- Coming Soon
- Accelerated co-array images

Preview support available now in NVFORTRAN

Fortran 202x

DO CONCURRENT Reductions

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

ACCELERATED PROGRAMMING IN ISO FORTRAN

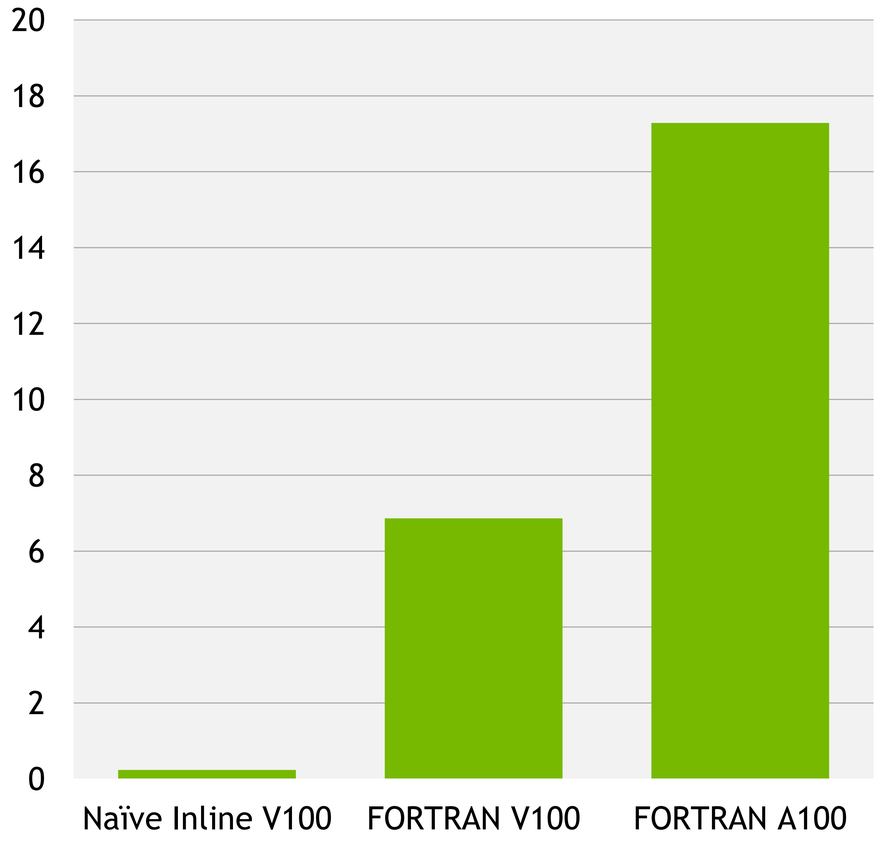
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
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

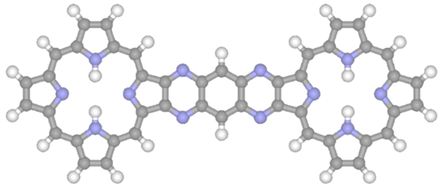


HPC PROGRAMMING IN ISO FORTRAN

Examples of Patterns Accelerated in NVFORTRAN

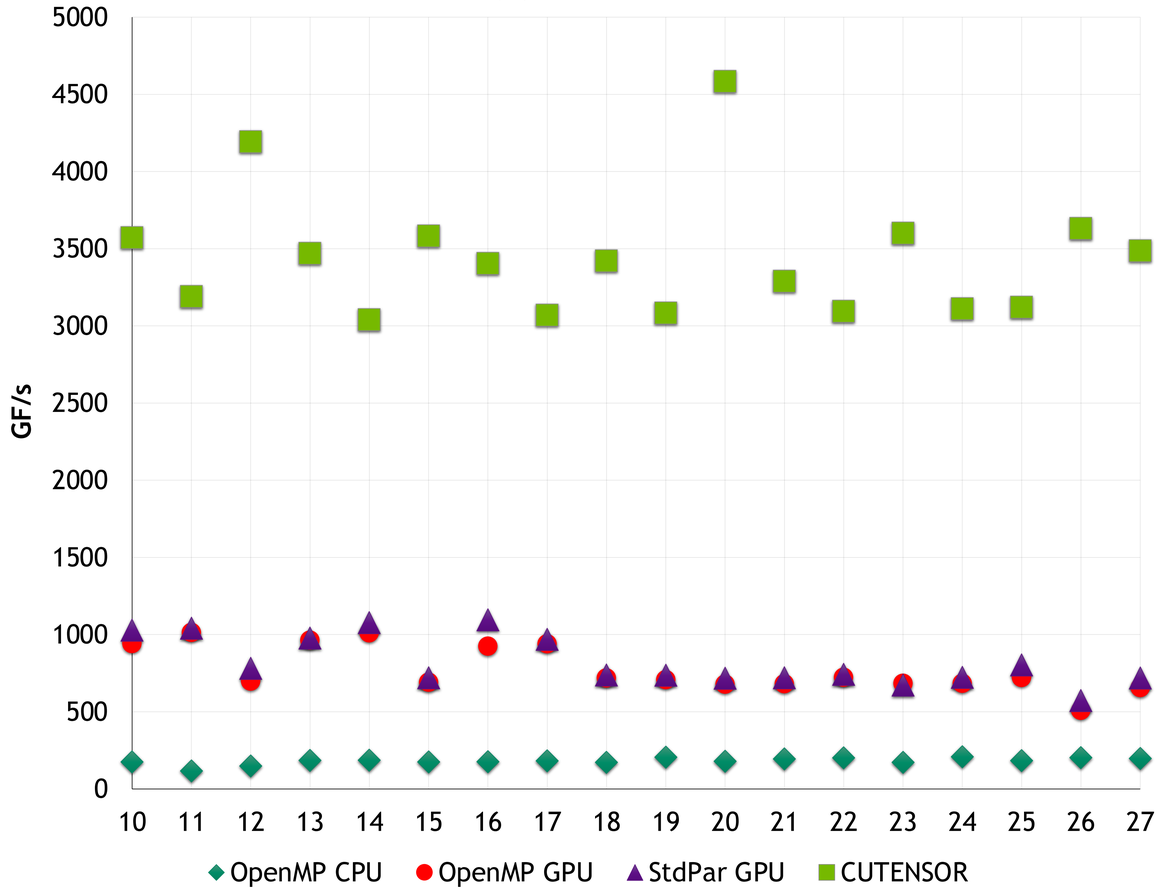
```
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(transpose(b))
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(b)
d = reshape(a,shape=[ni,nj,nk])
d = reshape(a,shape=[ni,nk,nj])
d = 2.5 * sqrt(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = alpha * conjg(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = reshape(a,shape=[ni,nk,nj],order=[1,3,2])
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
```

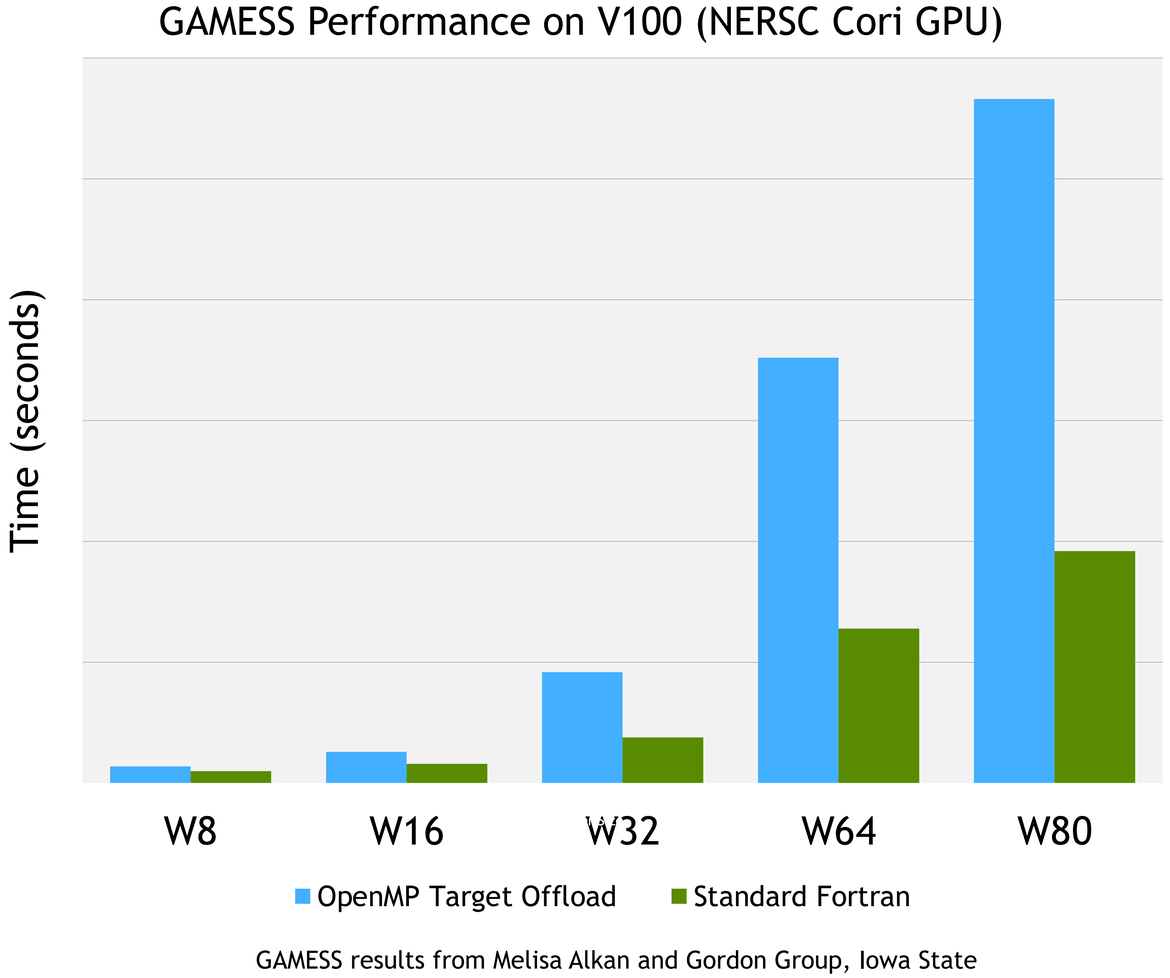


FORTRAN STANDARD PARALLELISM

NWChem and GAMESS with DO CONCURRENT



https://github.com/jeffhammond/nwchem-tce-triples-kernels/



GAMESS results from Melisa Alkan and Gordon Group, Iowa State

ACCELERATED STANDARD LANGUAGES

Parallel performance for wherever your code runs

ISO C++

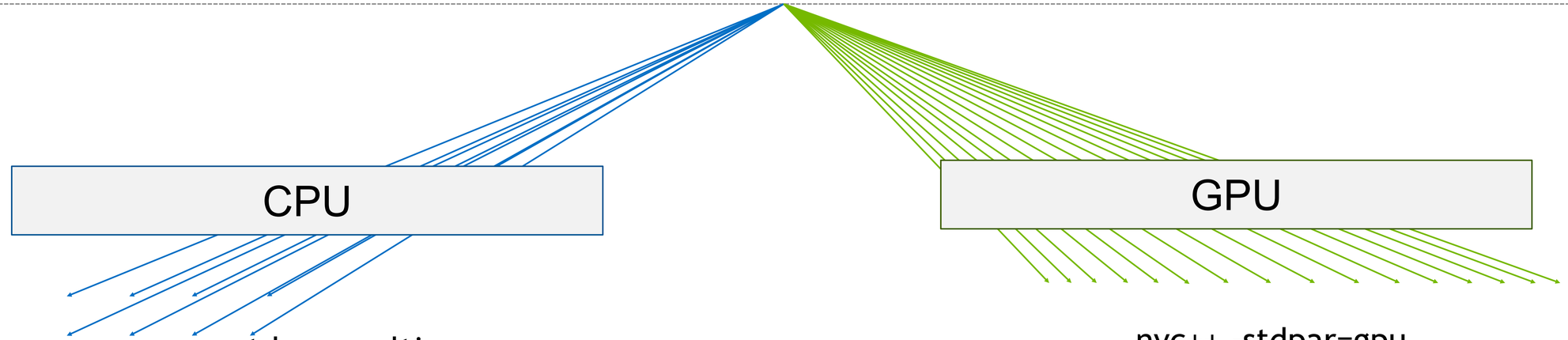
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

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

Python

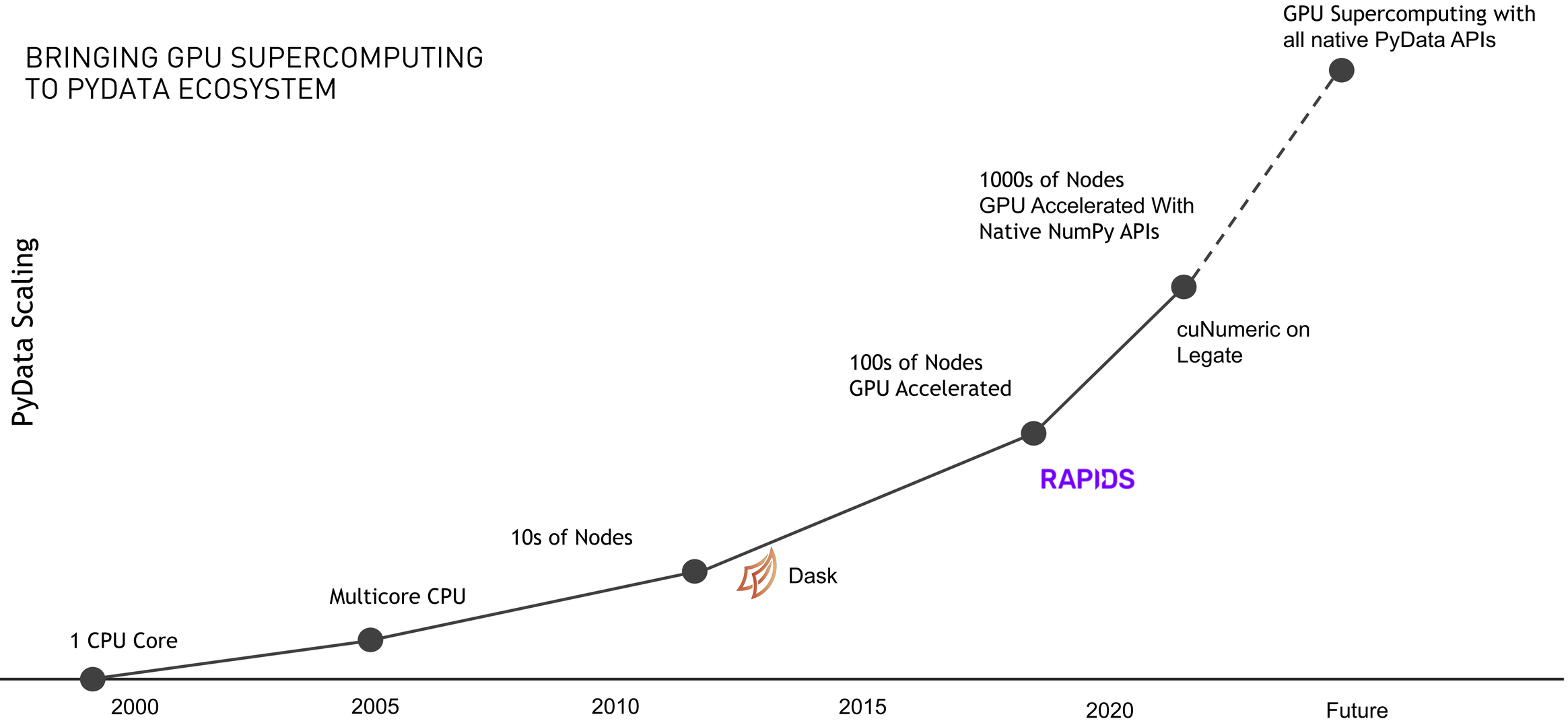
```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```



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

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

BRINGING GPU SUPERCOMPUTING TO PYDATA ECOSYSTEM




```
import numpy as np

a = np.random.randn(16).reshape(4, 4)
b = a + a.T
b
```

```
import dask.array as da
import numpy as np

a = da.from_array(
    np.random.randn(160_000).reshape(400, 400),
    chunks=(100, 100))
b = a + a.T
b.compute()
```

```
import dask.array as da
import cupy as cp

a = da.from_array(
    cp.random.randn(160_000).reshape(400, 400),
    chunks=(100, 100),
    asarray=False)
b = a + a.T
b.compute()
```

```
import cunumeric as np

a = np.random.randn(160_000).reshape(400, 400)
b = a + a.T
b
```

NVIDIA

PYTHON ECOSYSTEM GOALS

Have Your Cake and Eat It Too

Productivity

```
def cg_solve(A, b, conv_iters):
    x = np.zeros_like(b)
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    converged = False
    max_iters = b.shape[0]

    for i in range(max_iters):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)

        if i % conv_iters == 0 and \
            np.sqrt(rsnew) < 1e-10:
            converged = i
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
```

Performance

PRODUCTIVITY

Sequential and Composable Code

```
def cg_solve(A, b, conv_iters):
    x = np.zeros_like(b)
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    converged = False
    max_iters = b.shape[0]

    for i in range(max_iters):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)

        if i % conv_iters == 0 and \
            np.sqrt(rsnew) < 1e-10:
            converged = i
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
```

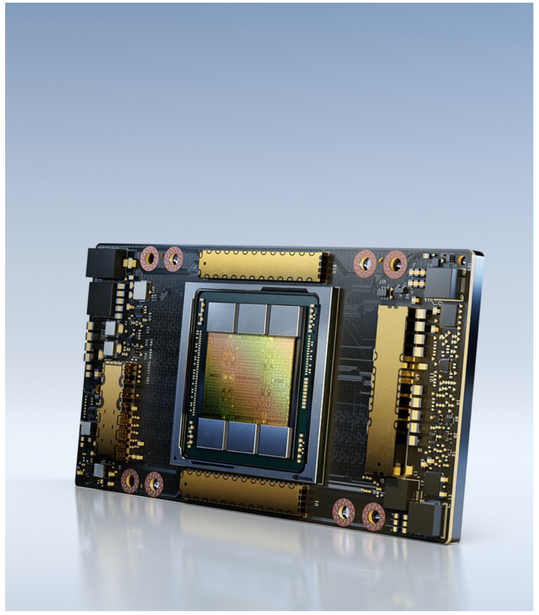
- Sequential semantics - no visible parallelism or synchronization
- Name-based global data – no partitioning
- Composable – can combine with other libraries and datatypes

PERFORMANCE

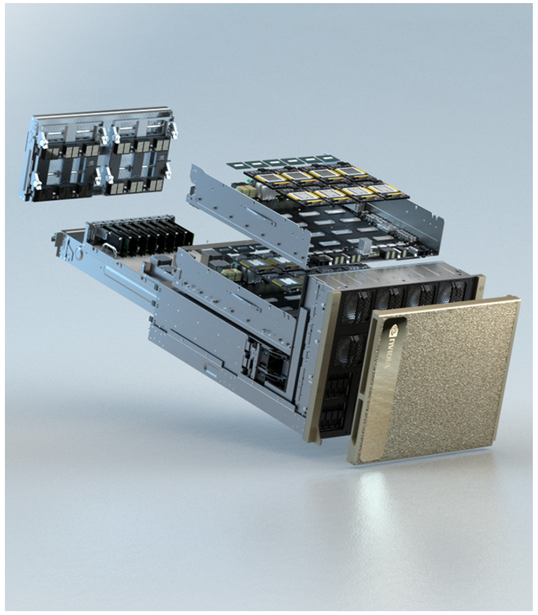
Transparent Acceleration

- Transparently run at any scale needed to address computational challenges at hand
- Automatically leverage all the available hardware

GPU

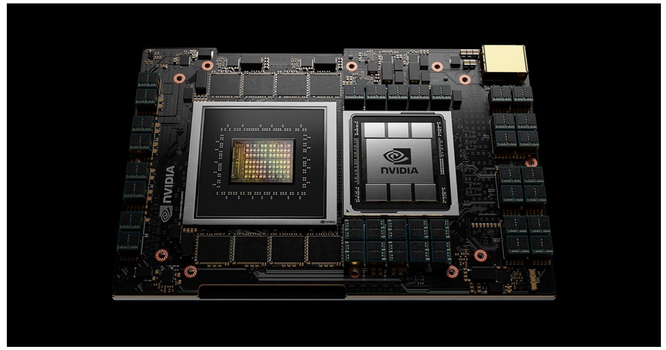


DGX-2

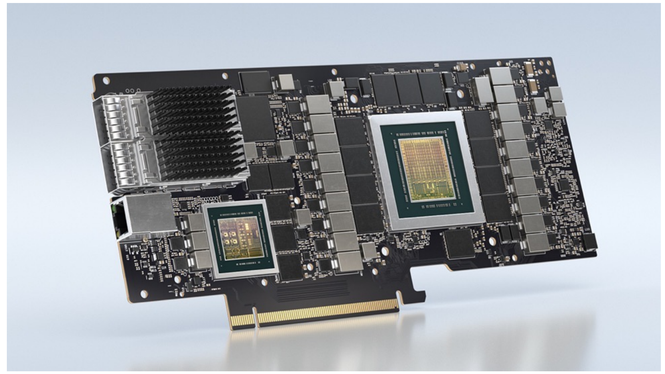


DGX SuperPod

Grace CPU

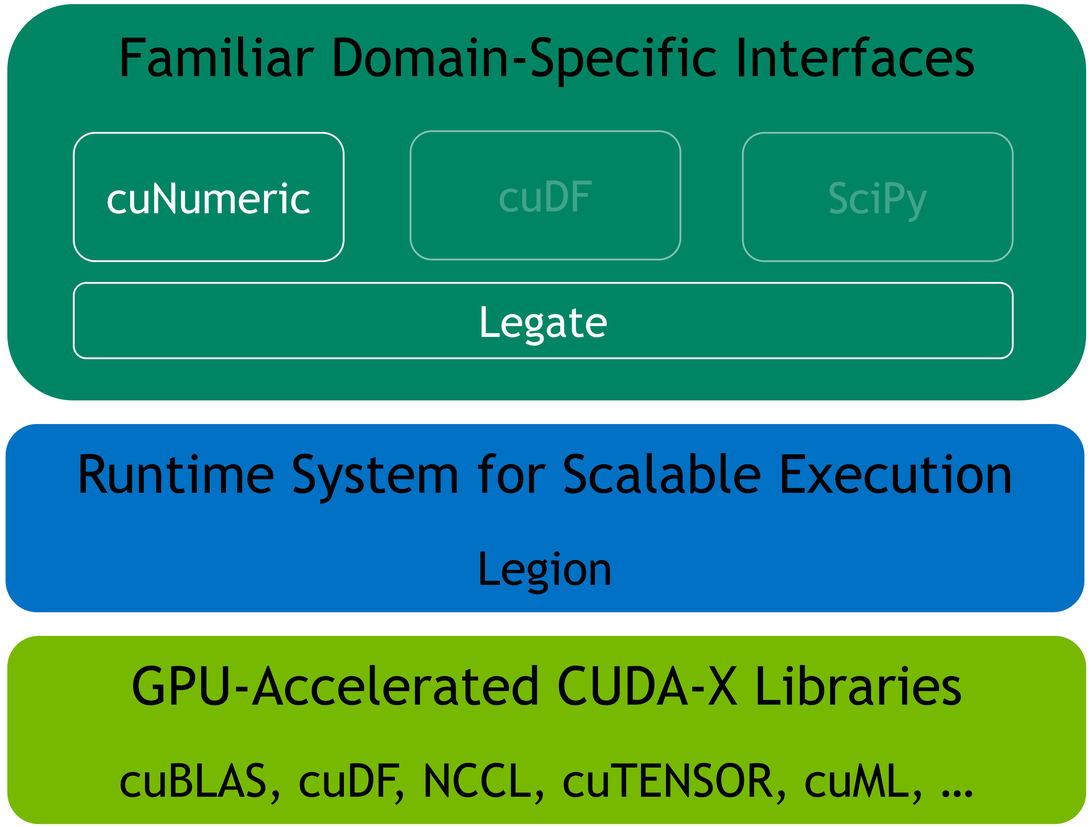


DPU



LEGATE ECOSYSTEM ARCHITECTURE

Scalable implementations of popular domain-specific APIs



CUNUMERIC

Automatic NumPy Acceleration and Scalability

cuNumeric

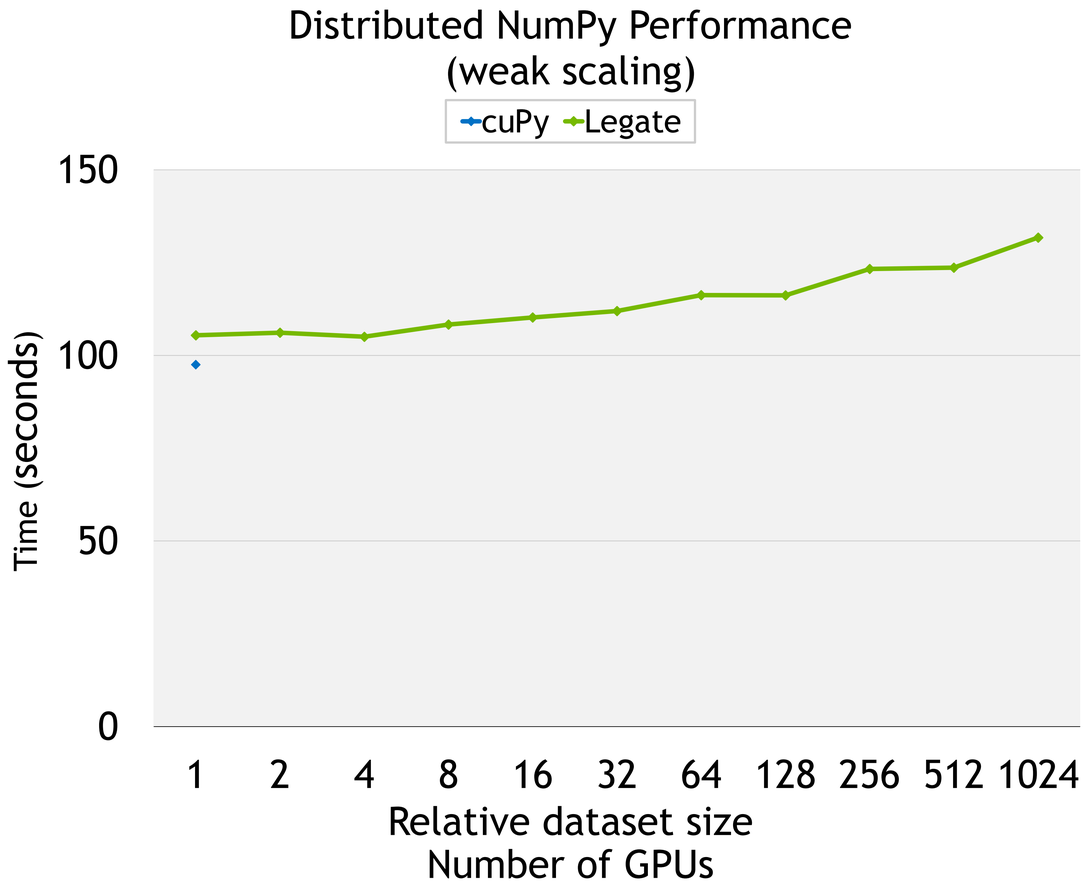
CuNumeric transparently accelerates and scales existing Numpy workloads

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

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

Alpha release available at github.com/nv-legate

```
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)

…
```

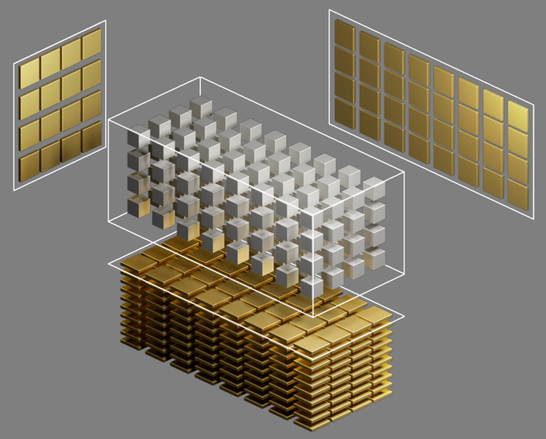


Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, **1**(9), 21, https://doi.org/10.21105/jose.00021

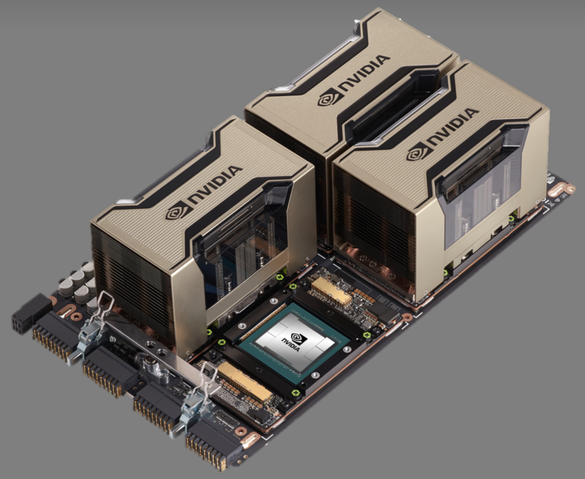
NVIDIA PERFORMANCE LIBRARIES

Major Directions

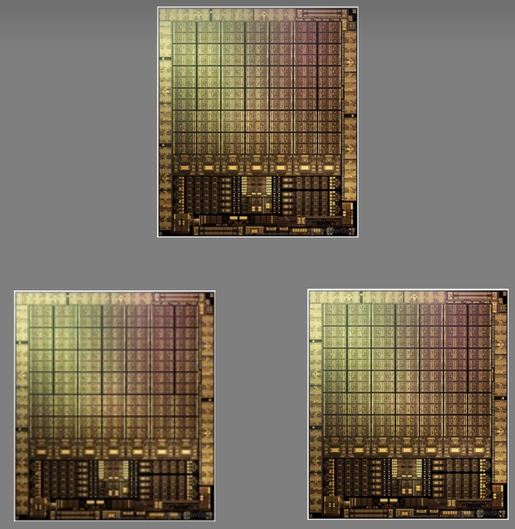
Seamless Acceleration
Tensor Cores, Enhanced L2$ & SMEM



Scaling Up
Multi-GPU and Multi-Node Libraries

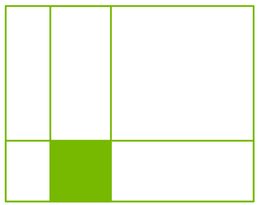


Composability
Device Functions

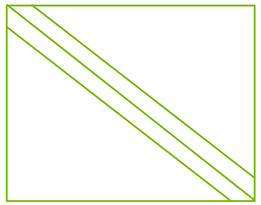


NVIDIA MATH LIBRARIES

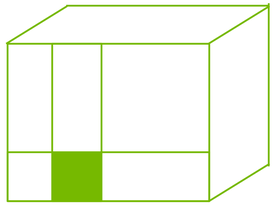
Linear Algebra, FFT, RNG and Basic Math



cuBLAS

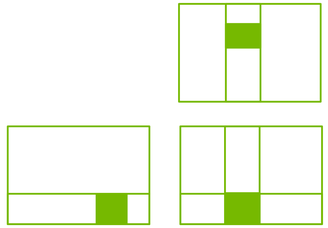


cuSPARSE

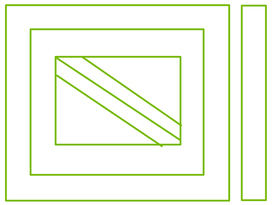


cuTENSOR

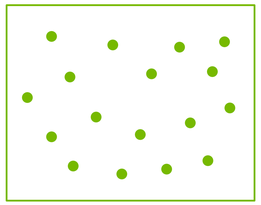
cuSOLVER



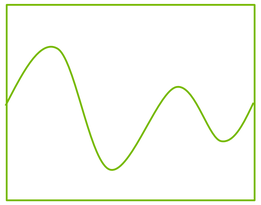
CUTLASS



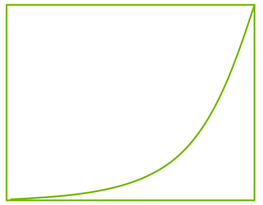
AMGX



cuRAND



cuFFT

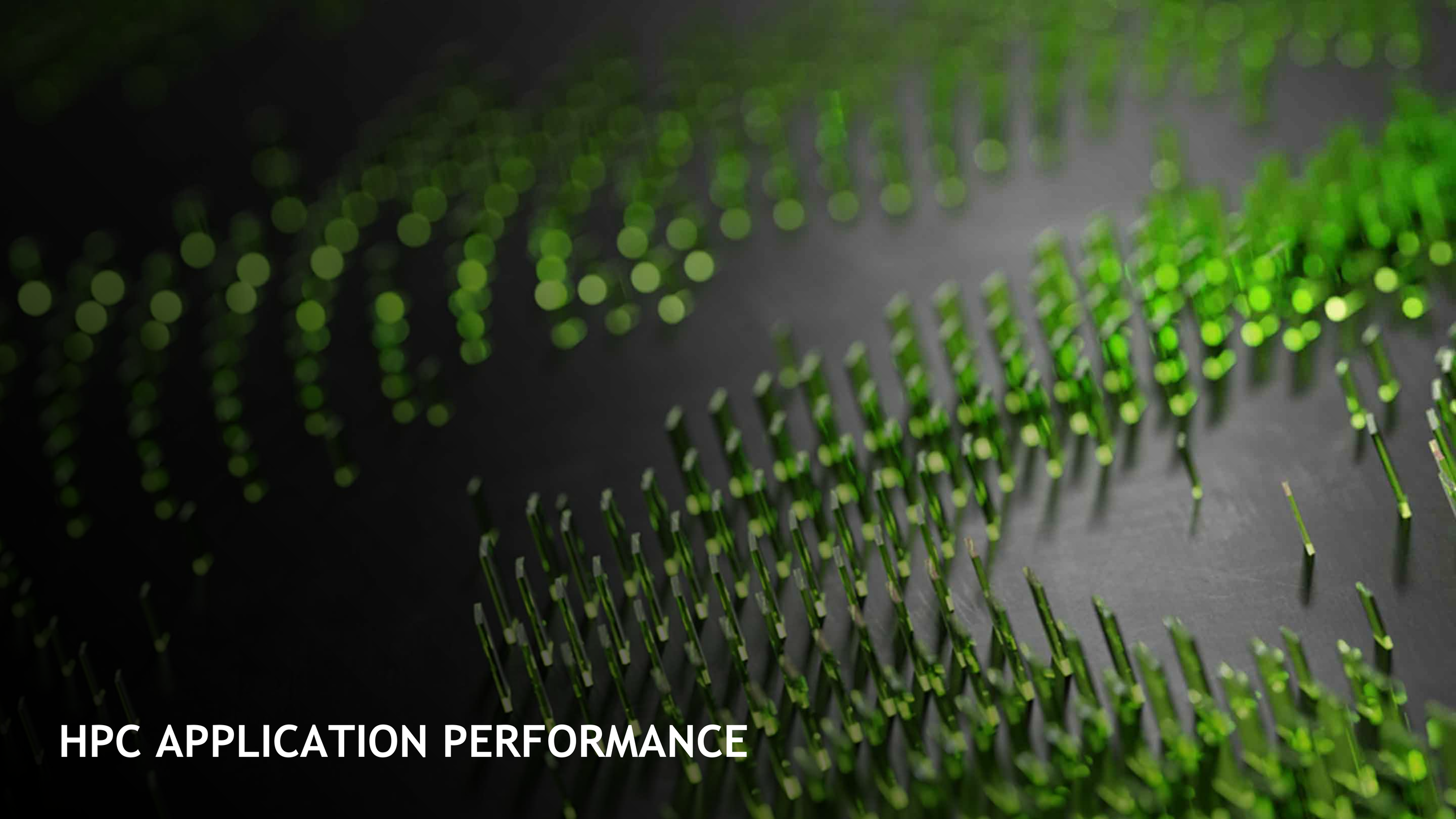


CUDA Math API

TENSOR CORE SUPPORT IN MATH LIBRARIES

High-level overview of supported functionality by each library

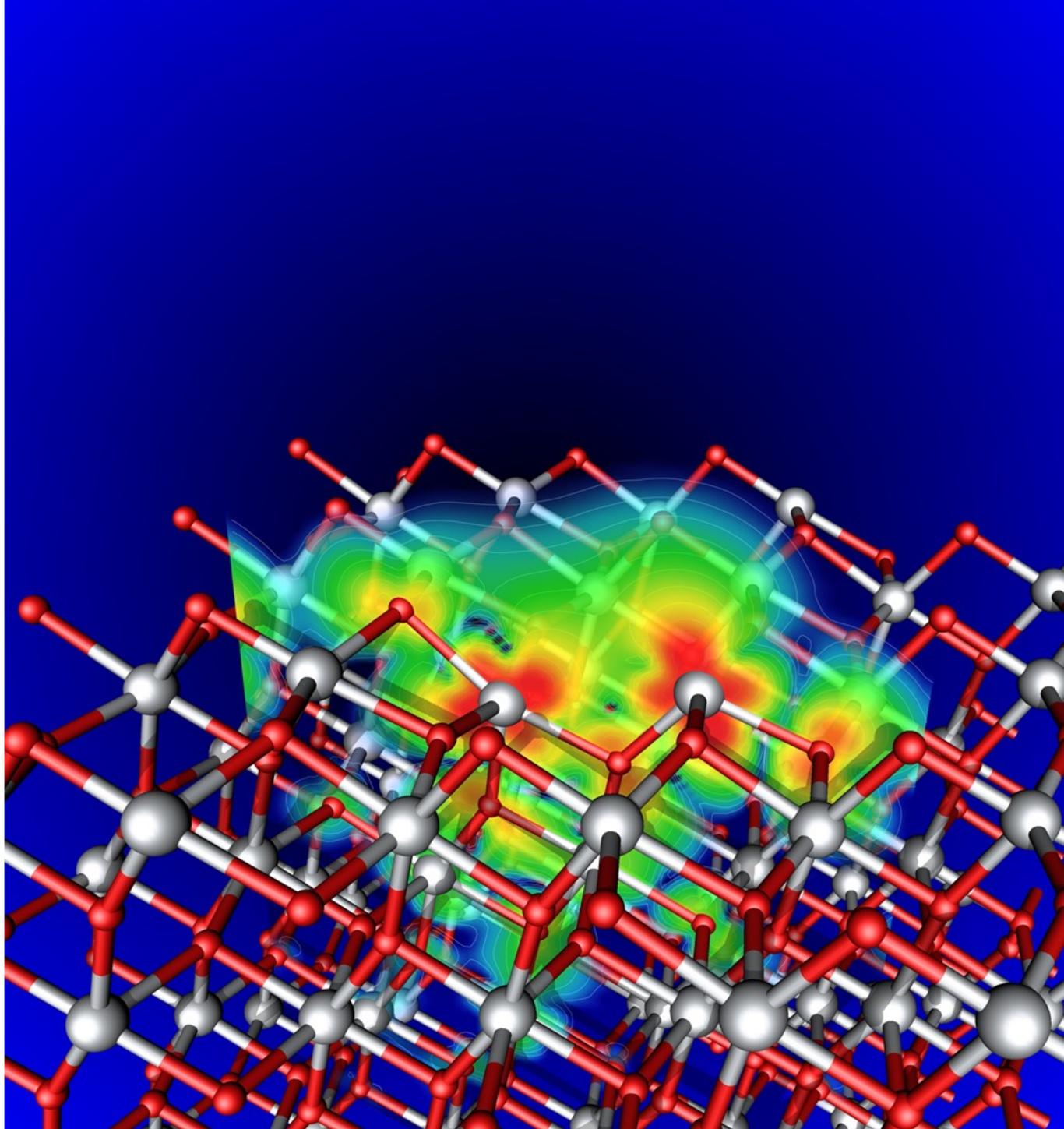
Library and Tensor Core Functionality	INT4		INT8		FP16		BF16		TF32		FP64
	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense
cuBLAS & cuBLASLt Dense GEMM			✅		✅		✅		✅		✅
cuTENSOR Tensor Contractions					✅		✅		✅		✅
cuSOLVER Linear System Solvers					✅		✅		✅		✅
cuSPARSE Block-SpMM			✅		✅		✅		✅		✅
cuSPARSELt SpMM				✅		✅		✅		✅	
CUTLASS Dense GEMM and SpMM	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅
CUTLASS Convolutions	✅		✅		✅		✅		✅		


HPC APPLICATION PERFORMANCE

INTRODUCTION TO VASP

Scientific Background

- Most widely used GPU-accelerated software for electronic structure of solids, surfaces, and interfaces
- Generates
  - Chemical and physical properties
  - Reactions paths
- Capabilities
  - First principles scaled to 1000s of atoms
  - Materials and properties - liquids, crystals, magnetism, semiconductors/insulators, surfaces, catalysts
  - Solves many-body Schrödinger equation
- Quantum-mechanical methods and solvers
  - Density Functional Theory (DFT)
  - Plane-wave based framework
  - New implementations for hybrid DFT (HF exact exchange)



FEATURES AVAILABLE AND ACCELERATED IN VASP 6.2

LEVELS OF THEORY

Standard DFT **(incl. meta-GGA, vdW-DFT)**
Hybrid DFT **(double buffered)**
Cubic-scaling RPA (ACFDT, GW)
Bethe-Salpeter Equations (BSE)

SOLVERS / MAIN ALGORITHM

Davidson **(+Adaptively Compressed Exch.)**
RMM-DIIS
Davidson+RMM-DIIS
Direct optimizers (Damped, All)
Linear response

PROJECTION SCHEME

Real space
Reciprocal space

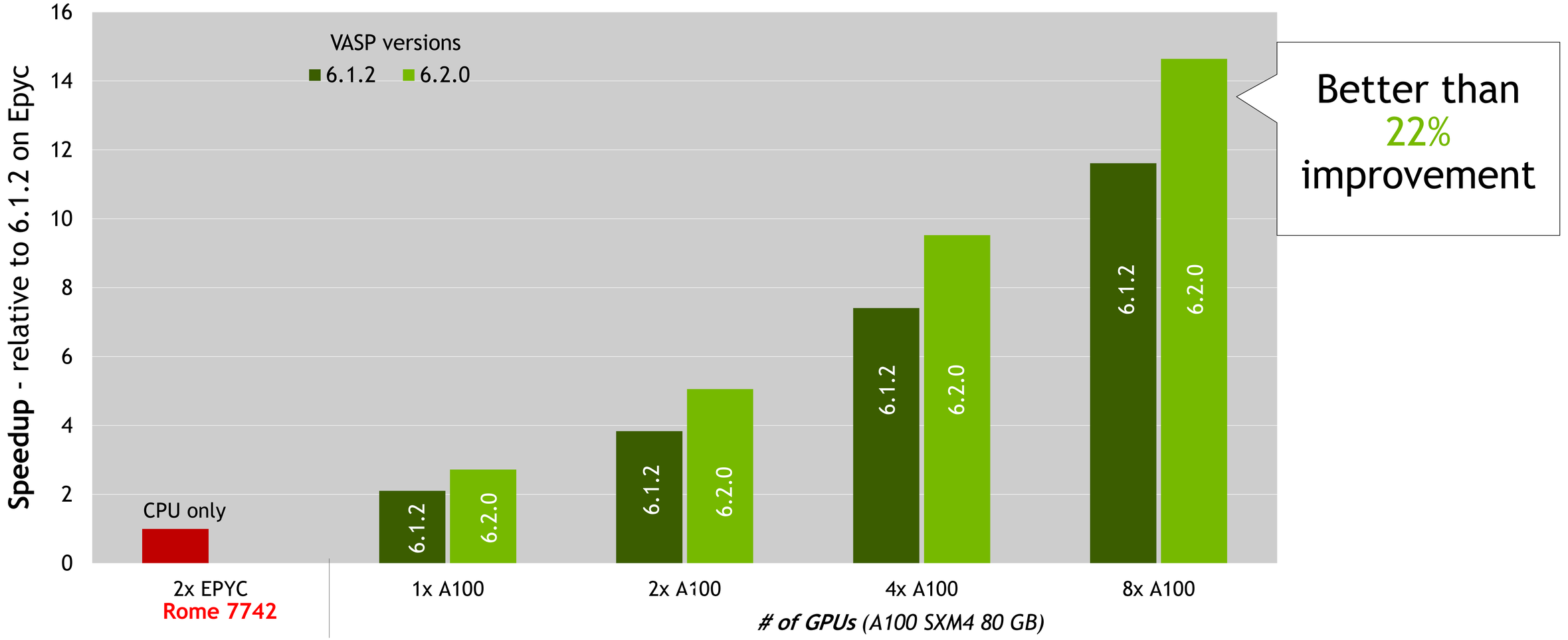
EXECUTABLE FLAVORS

Standard variant
Gamma-point simplification variant
Non-collinear spin variant

Existing acceleration *New acceleration* *Acceleration work in progress* *On acceleration roadmap*

VASP VERSION UPDATES BRING NEW ACCELERATION

Dataset: Si256_VJT_HSE06

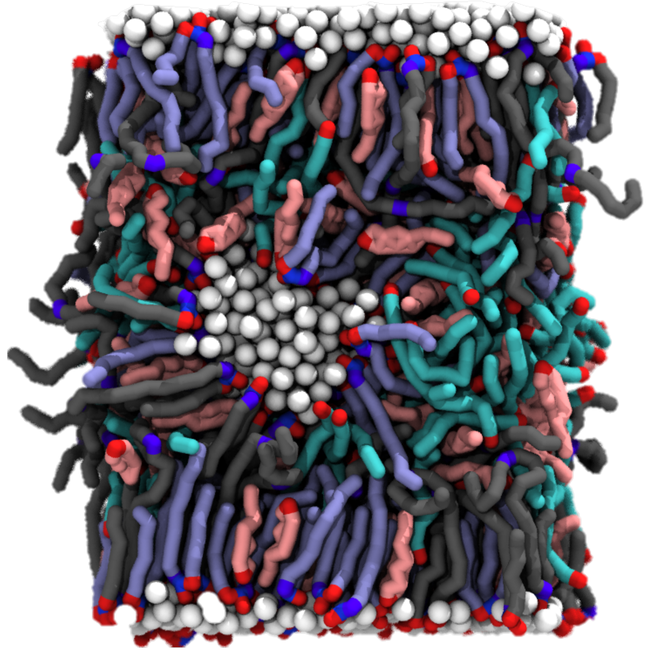


INTRODUCTION TO LAMMPS

Scientific Background

- LAMMPS stands for “Large-scale Atomic/Molecular Massively Parallel Simulator”. It is an open-source molecular dynamics simulation application for materials modeling both solid-state and soft matter. It can also do coarse grained simulations for larger particles. Development is funded by DOE and primary developers are at Sandia National Laboratory. This app is used all over the world by a wide variety of industries including semiconductors and pharmaceuticals.

- LAMMPS Distributions
  - Github
  - NGC container
  - MedeA by Materials Design



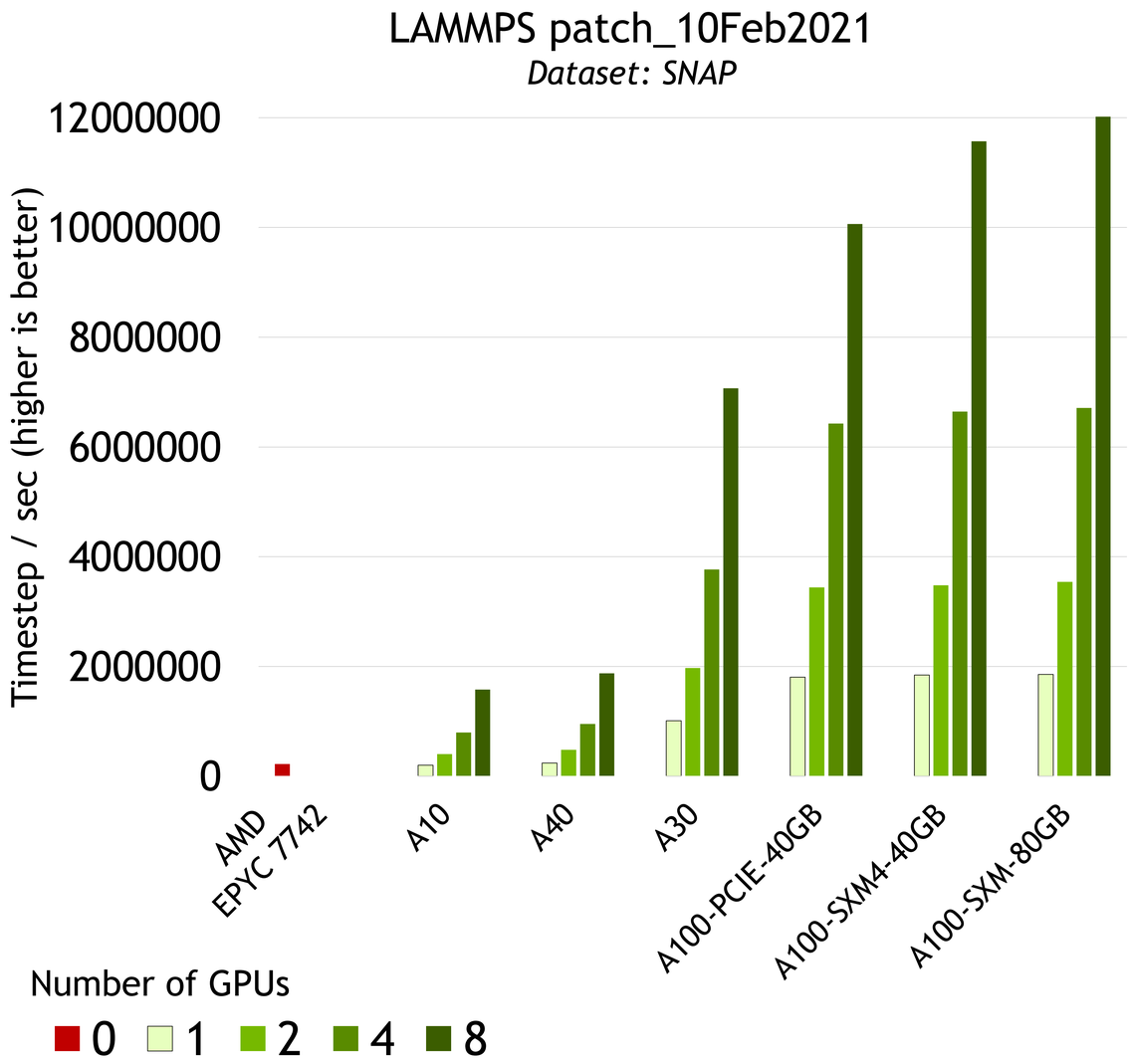
Lipids immobilizing water into droplets

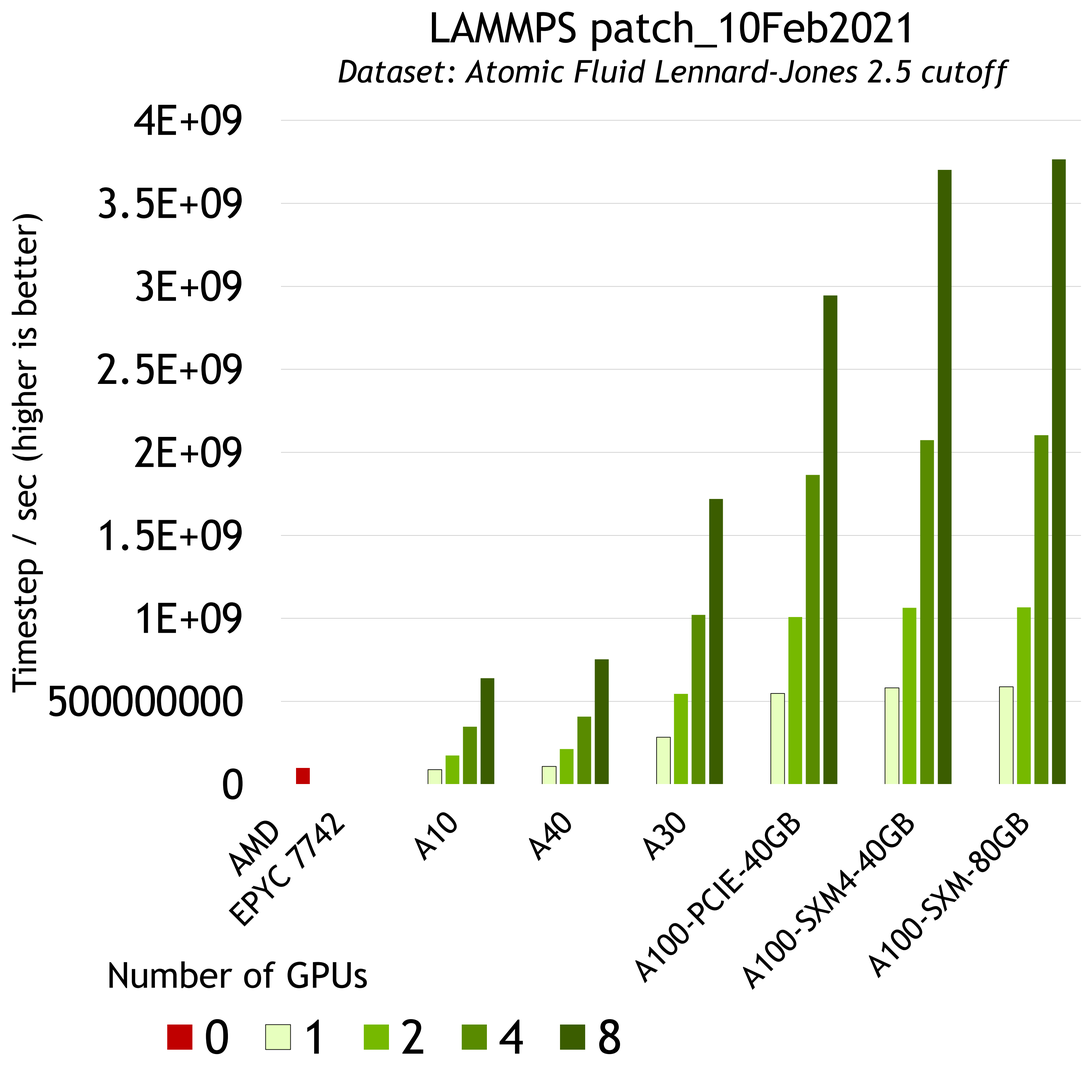
GPU ACCELERATED FEATURES IN LAMMPS

Primary GPU Acceleration Enabled in KOKKOS

- Virtually all features in LAMMPS are accelerated on NVIDIA GPUs using Kokkos. Performance varies by input and method.
- Most users will be familiar capabilities involved in “interatomic potentials” such as
  - Pairwise potentials like Lennard-Jones
  - Many-body potentials like EAM, ReaxFF, SNAP
  - Long-range interactions like PPPM for Ewald / particle mesh Ewald
  - Compatibility with force fields from CHARMM, AMBER, GROMACS, COMPASS
- Ongoing development of NVIDIA acceleration capabilities happens through partnership with LAMMPS developers and NVIDIA Devtech organization. NVIDIA leads include Evan Weinberg and Kamesh Arumugam. Each release has additional enhancements – so keep a look out for these updates.

LAMMPS CPU & GPU COMPARISON





OTHER HPC APPLICATIONS

NVIDIA HPC Application Performance | NVIDIA Developer

NVIDIA HPC Application Performance

For Deep Learning performance, please go here.

Modern HPC data centers are key to solving some of the world's most important scientific and engineering challenges. The NVIDIA A100, V100 and T4 GPUs fundamentally change the economics of the data center, delivering breakthrough performance with dramatically fewer servers, less power consumption, and reduced networking overhead, resulting in total cost savings of 5X-10X.

The number of CPU-only servers replaced by a single GPU-accelerated server is called the node replacement factor (NRF). To arrive at NRF, we measure application performance with up to 8 CPU-only servers. Then we use linear scaling to scale beyond 8 servers to calculate the NRF. The NRF will vary by application.

A100 | A30 | A40 | V100 | T4

Quantum Espresso

Material Science (Quantum Chemistry)

An Open-source suite of computer codes for electronic structure calculations and materials modeling at the nanoscale

VERSION

CPU 6.7; GPU 6.8

ACCELERATED FEATURES

- linear algebra (matix multiply)
- explicit computational kernels
- 3D FFTs

SCALABILITY

Multi-GPU and Multi-Node

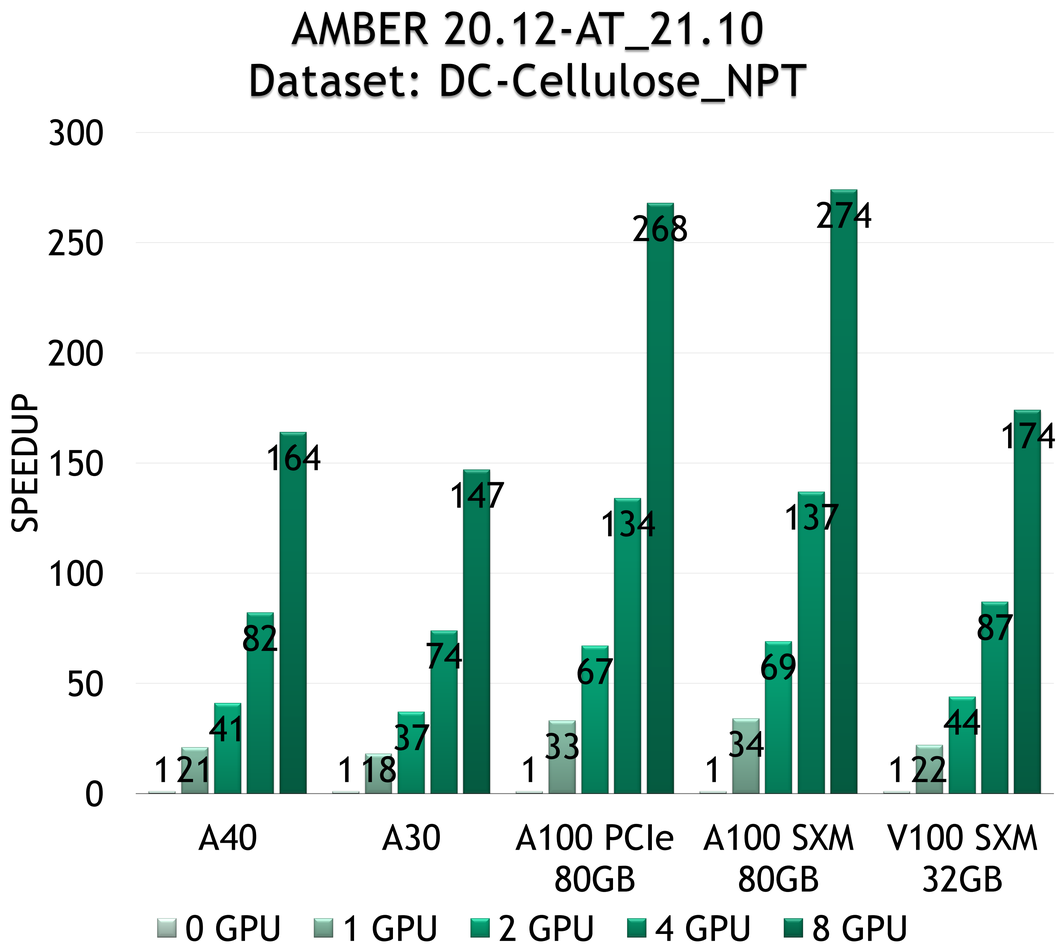
MORE INFORMATION

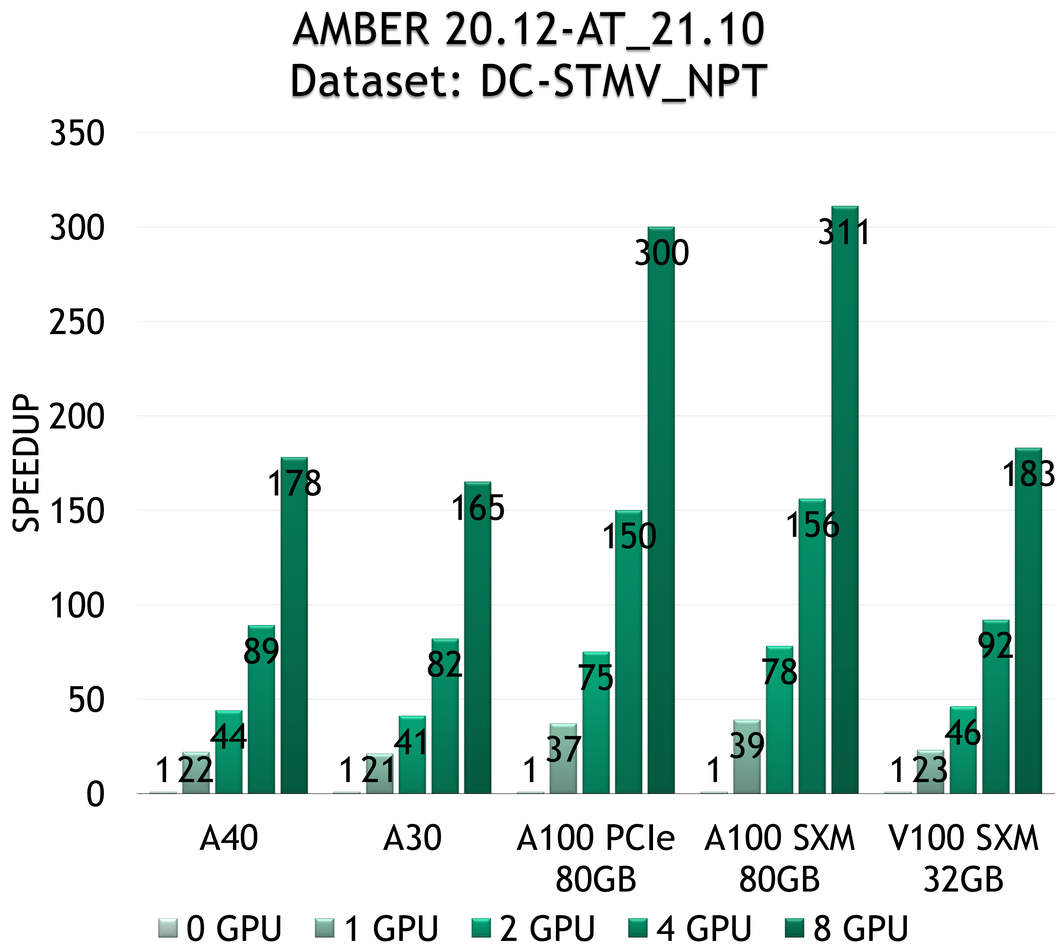
http://www.quantum-espresso.org

Application	Metric	Test Modules	Bigger is better	Dual Cascade Lake 6240 (CPU-Only)	1x A100 SXM 80GB	2x A100 SXM 80GB	4x A100 SXM 80GB	8x A100 SXM 80GB	1x A100 PCIe 80GB	2x A100 PCIe 800GB	4x A100 PCIe 80GB
Quantum Espressso	Total CPU Time (Sec)	AUSURF112-jR	no	765	135	84	58	49	147	92	78
Quantum Espressso	NRF	AUSURF112-jR	yes	1x	6x	10x	15x	17x	6x	9x	11x

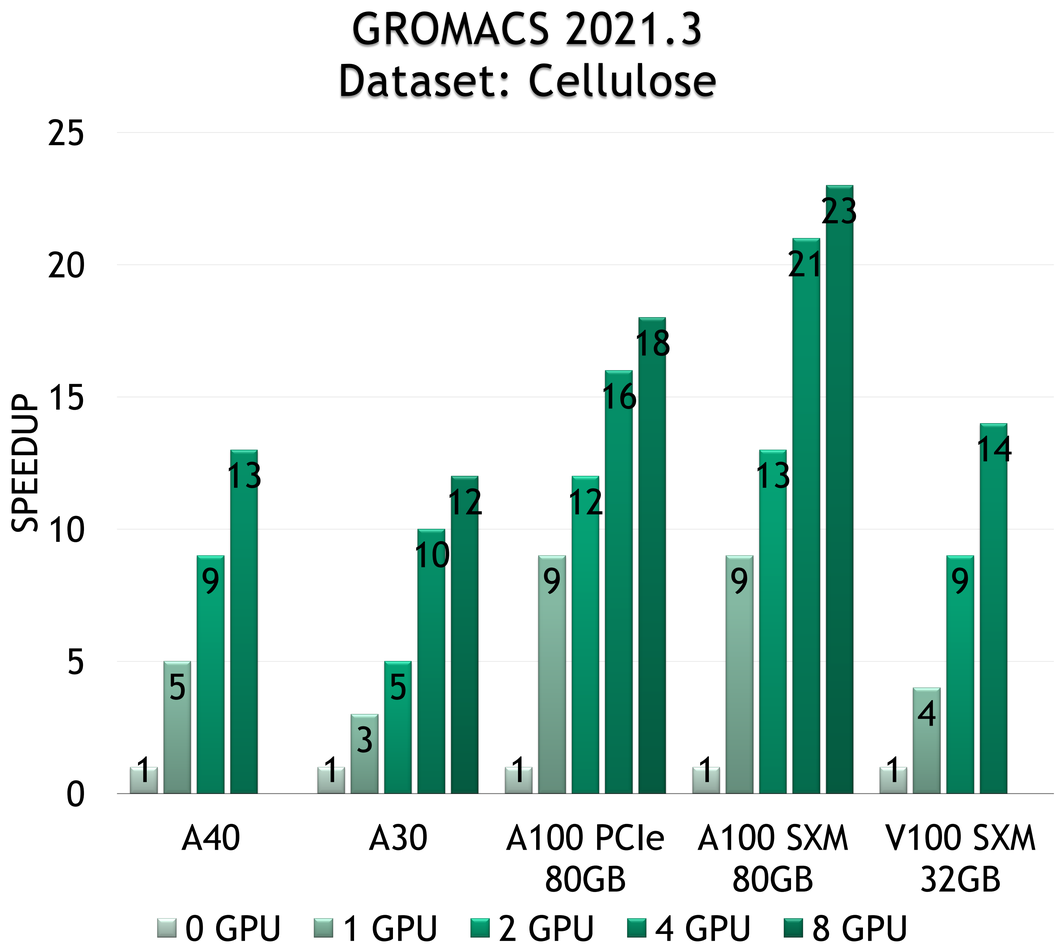
https://developer.nvidia.com/hpc-application-performance/

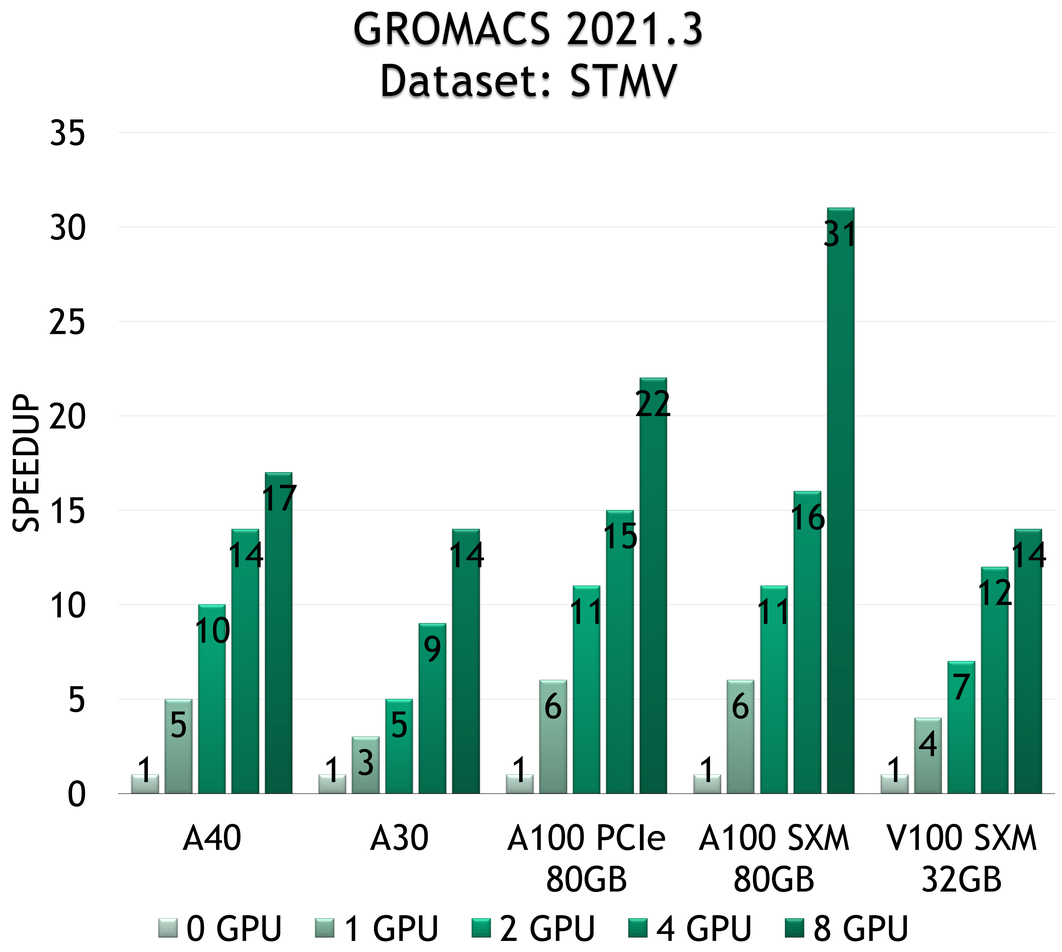
AMBER



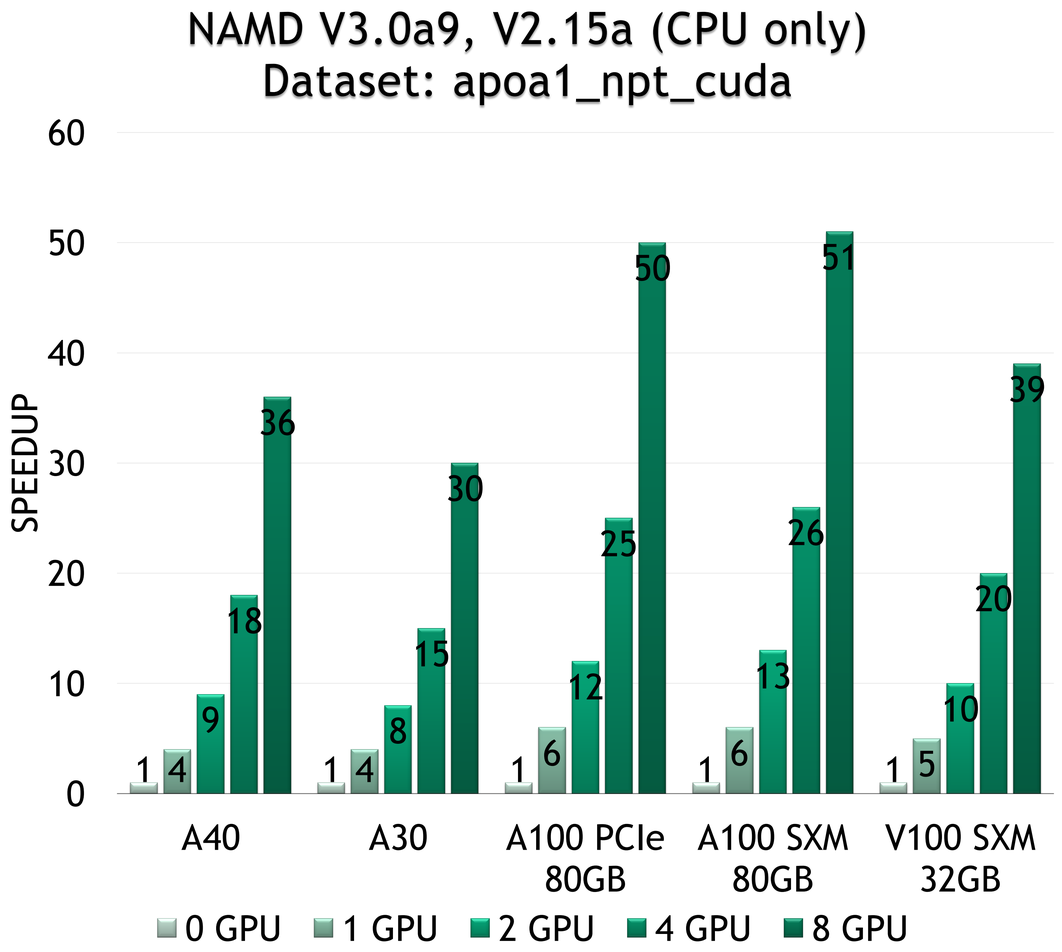


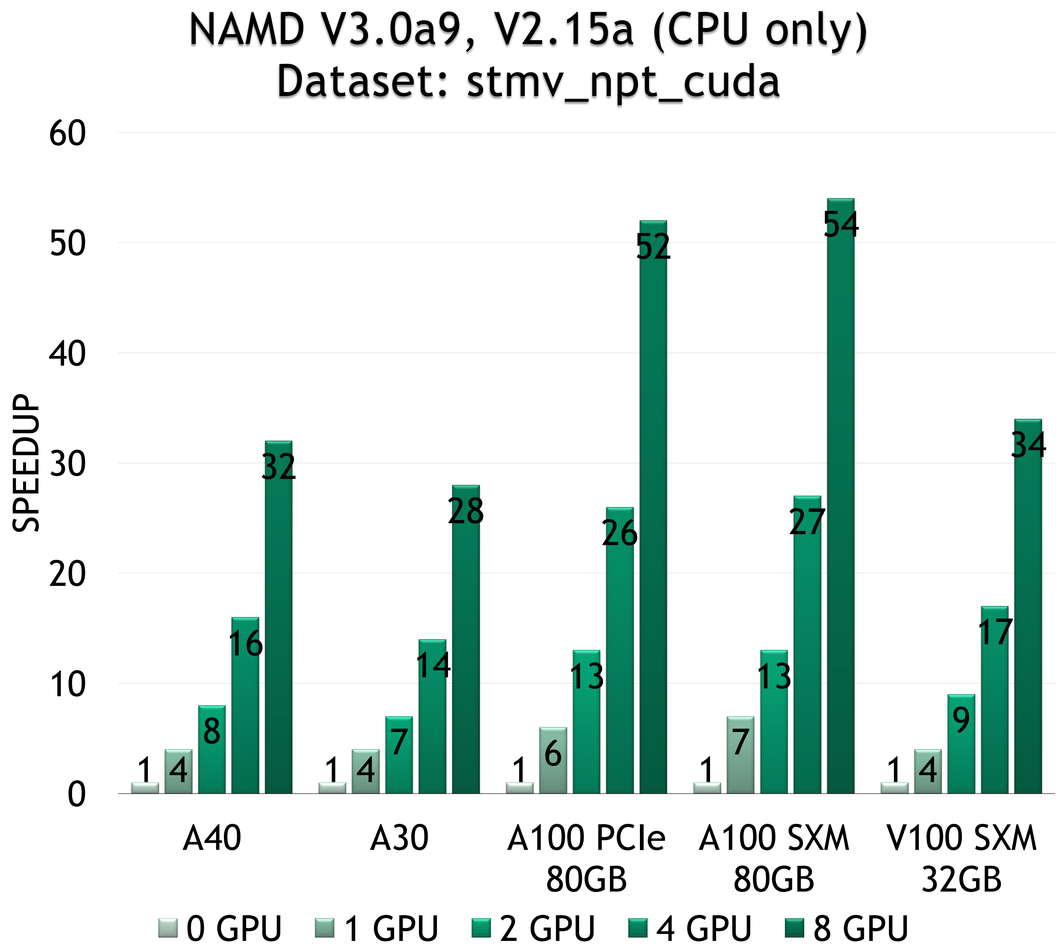
GROMACS



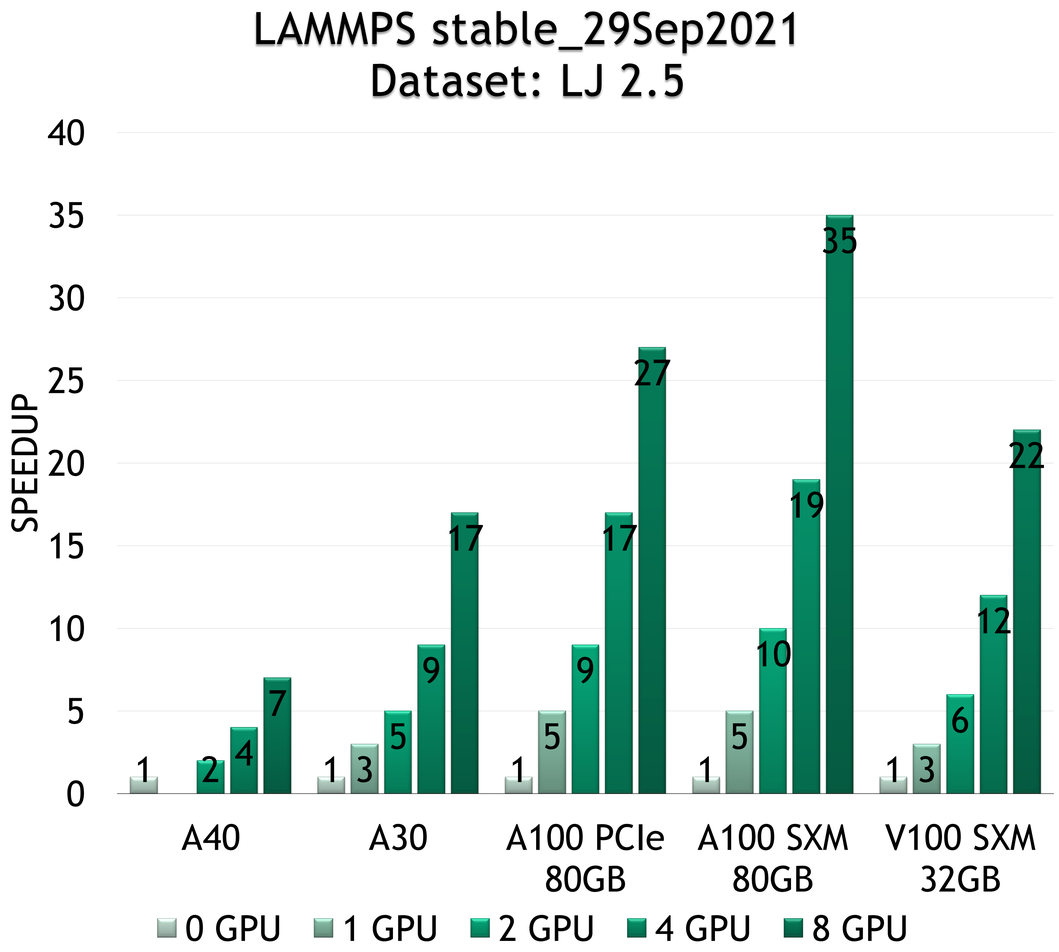


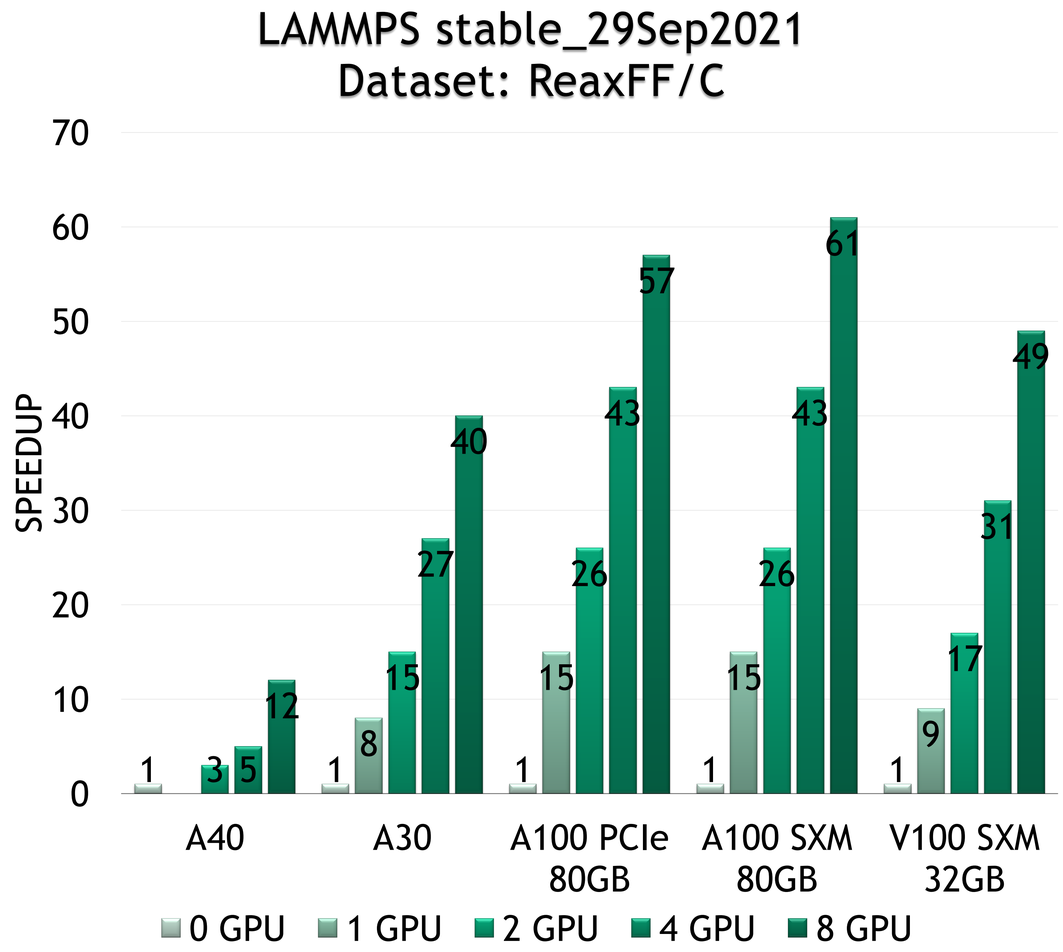
NAMD


NAMD V3.0a9, V2.15a (CPU only)
Dataset: apoa1_npt_cuda
SPEEDUP
60
50
40
30
20
10
0
1 4 9 18 36
1 4 8 15 30
1 6 12 25 50
1 6 13 26 51
1 5 10 20 39
A40
A30
A100 PCIe 80GB
A100 SXM 80GB
V100 SXM 32GB
0 GPU
1 GPU
2 GPU
4 GPU
8 GPU

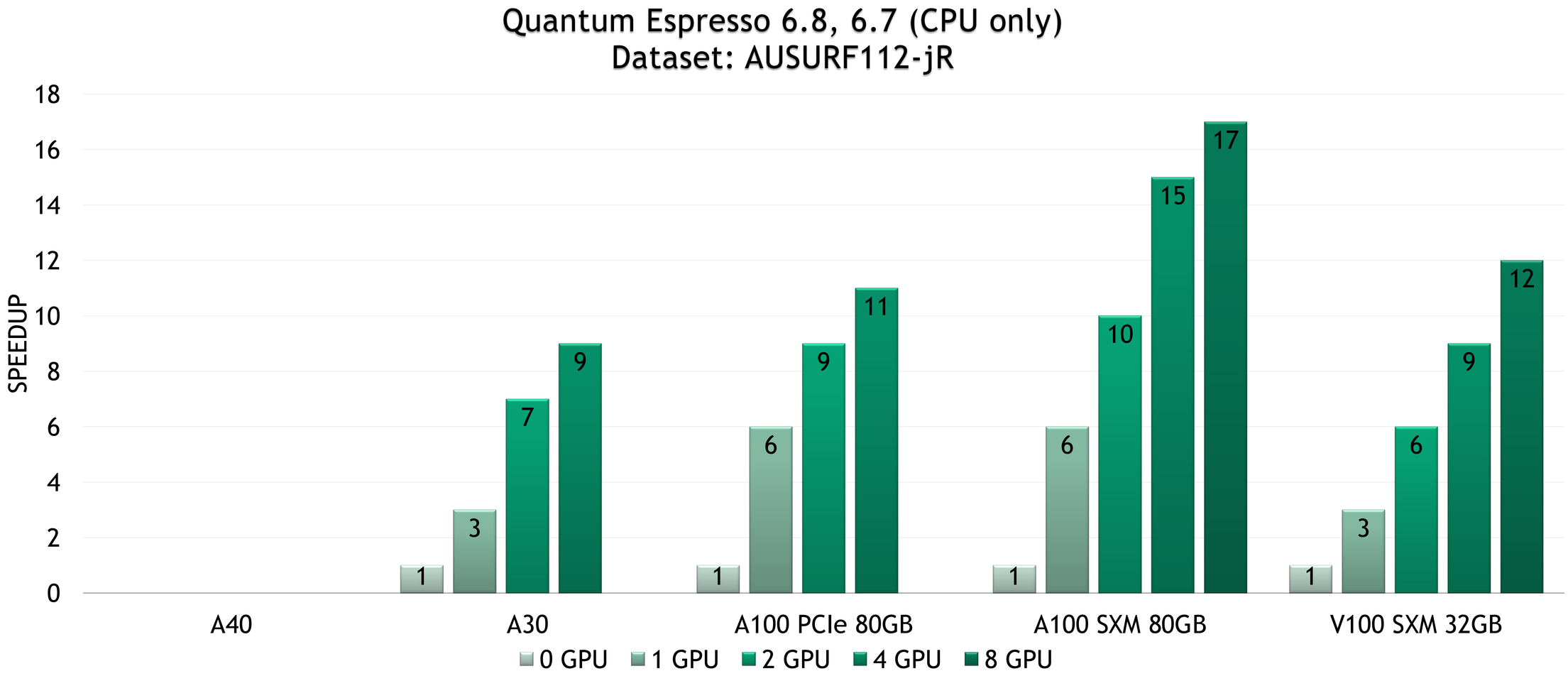

NAMD V3.0a9, V2.15a (CPU only)
Dataset: stmv_npt_cuda
SPEEDUP
60
50
40
30
20
10
0
1 4 8 16 32
1 4 7 14 28
1 6 13 26 52
1 7 13 27 54
1 4 9 17 34
A40
A30
A100 PCIe 80GB
A100 SXM 80GB
V100 SXM 32GB
0 GPU
1 GPU
2 GPU
4 GPU
8 GPU

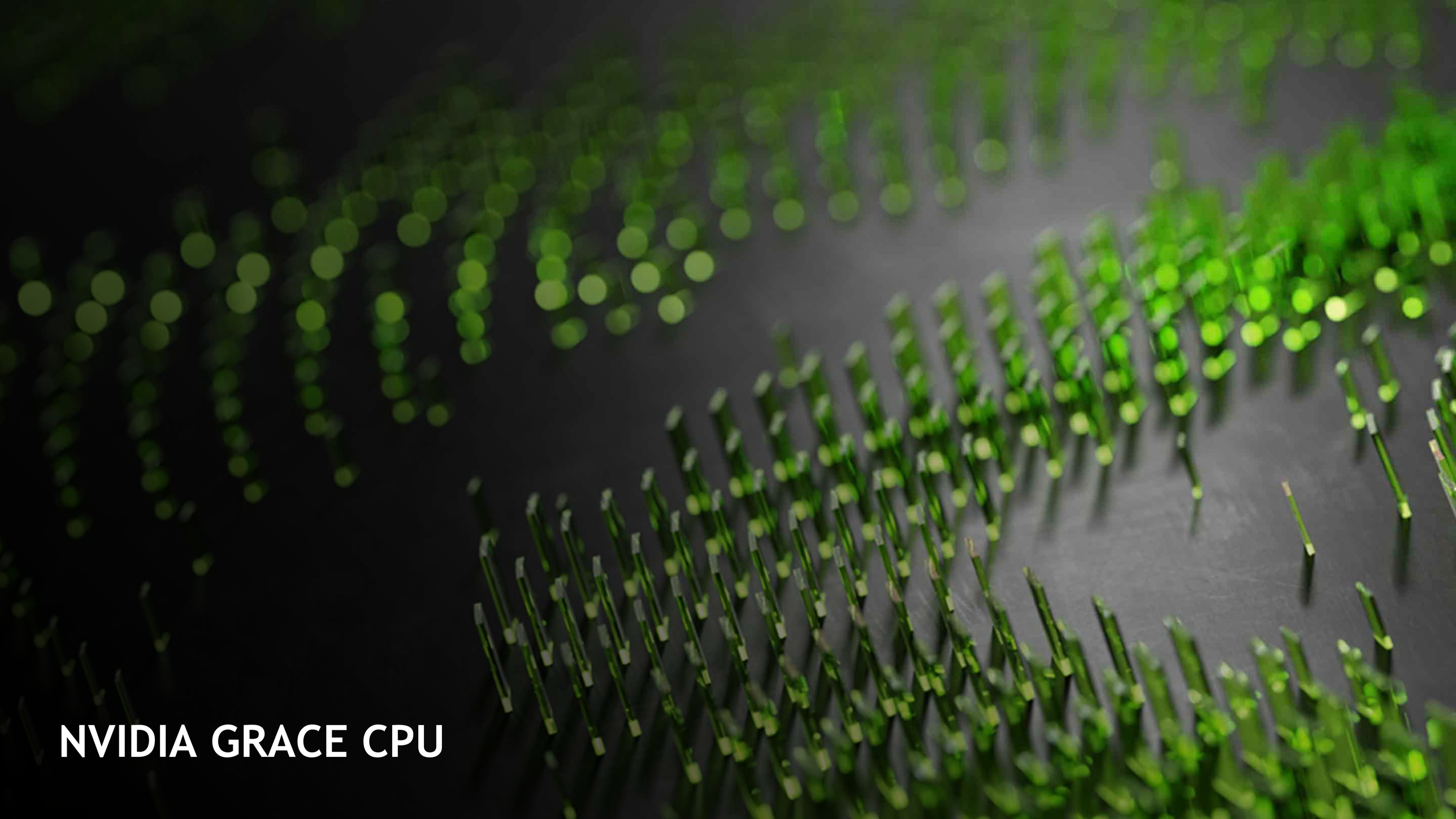
LAMMPS





QUANTUM ESPRESSO

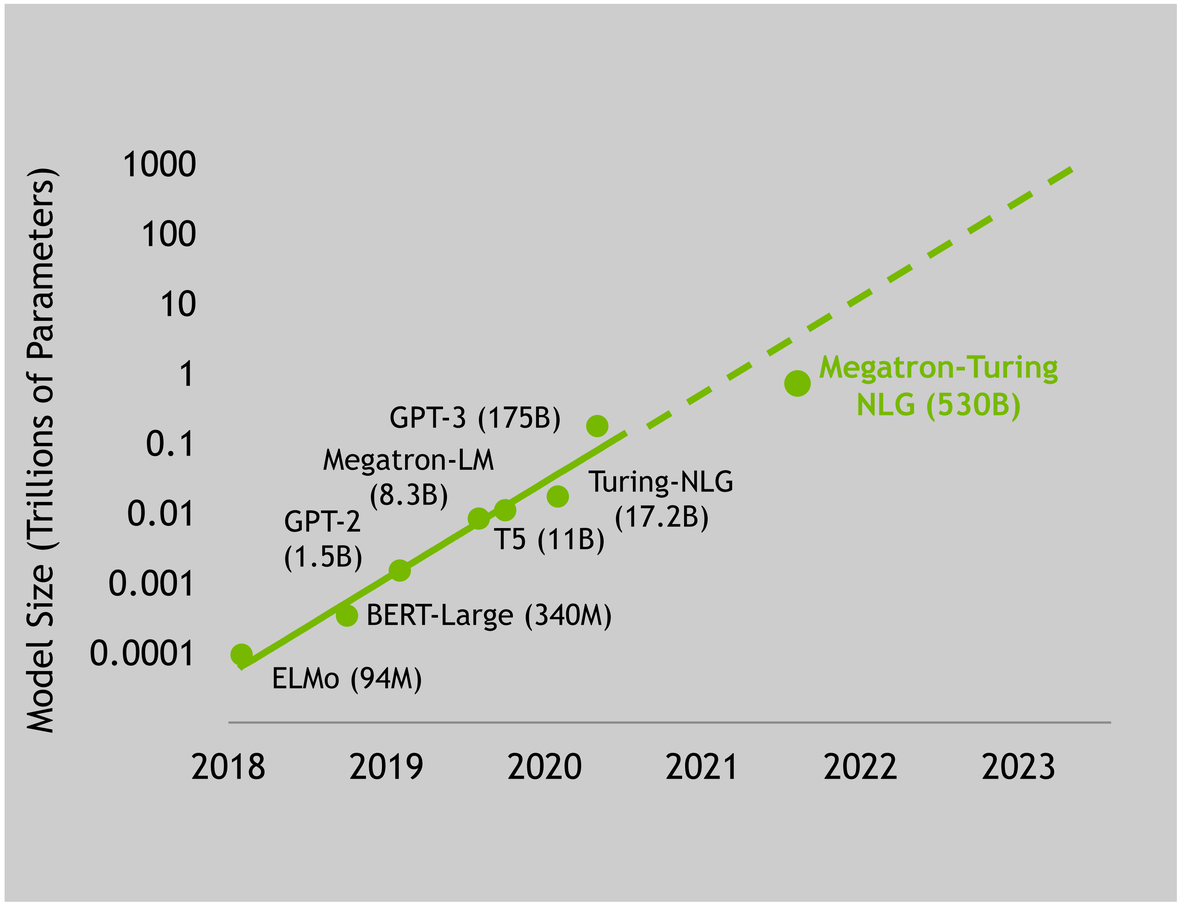



NVIDIA GRACE CPU

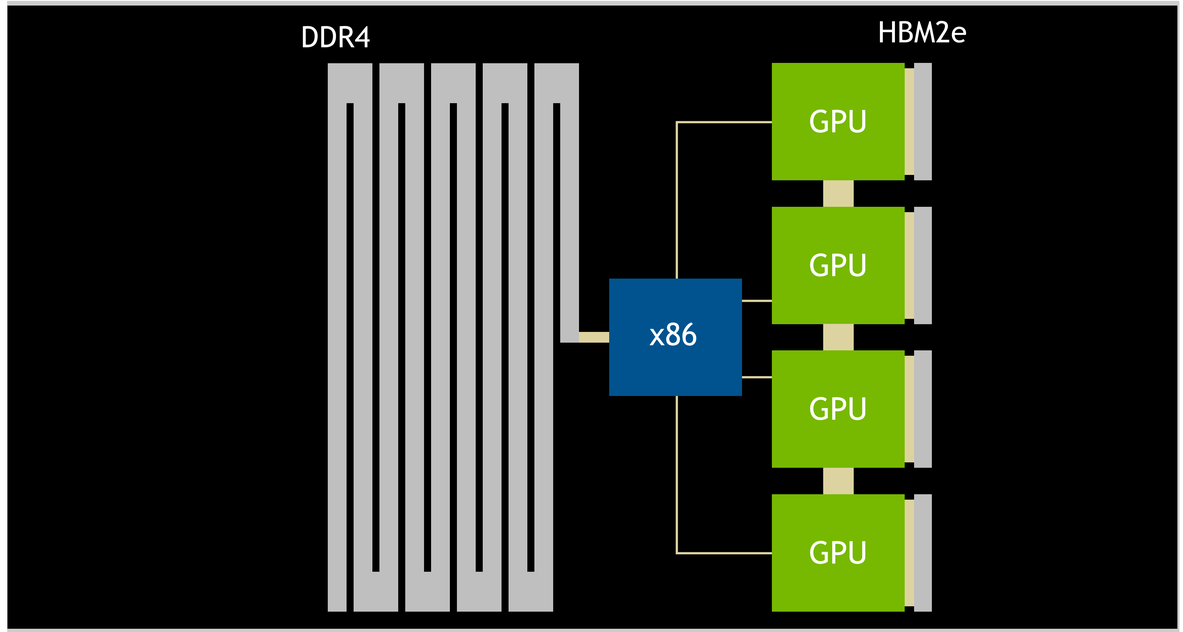
GIANT MODELS PUSHING LIMITS OF EXISTING ARCHITECTURE

Requires a New Architecture

100 TRILLION PARAMETER MODELS BY 2023



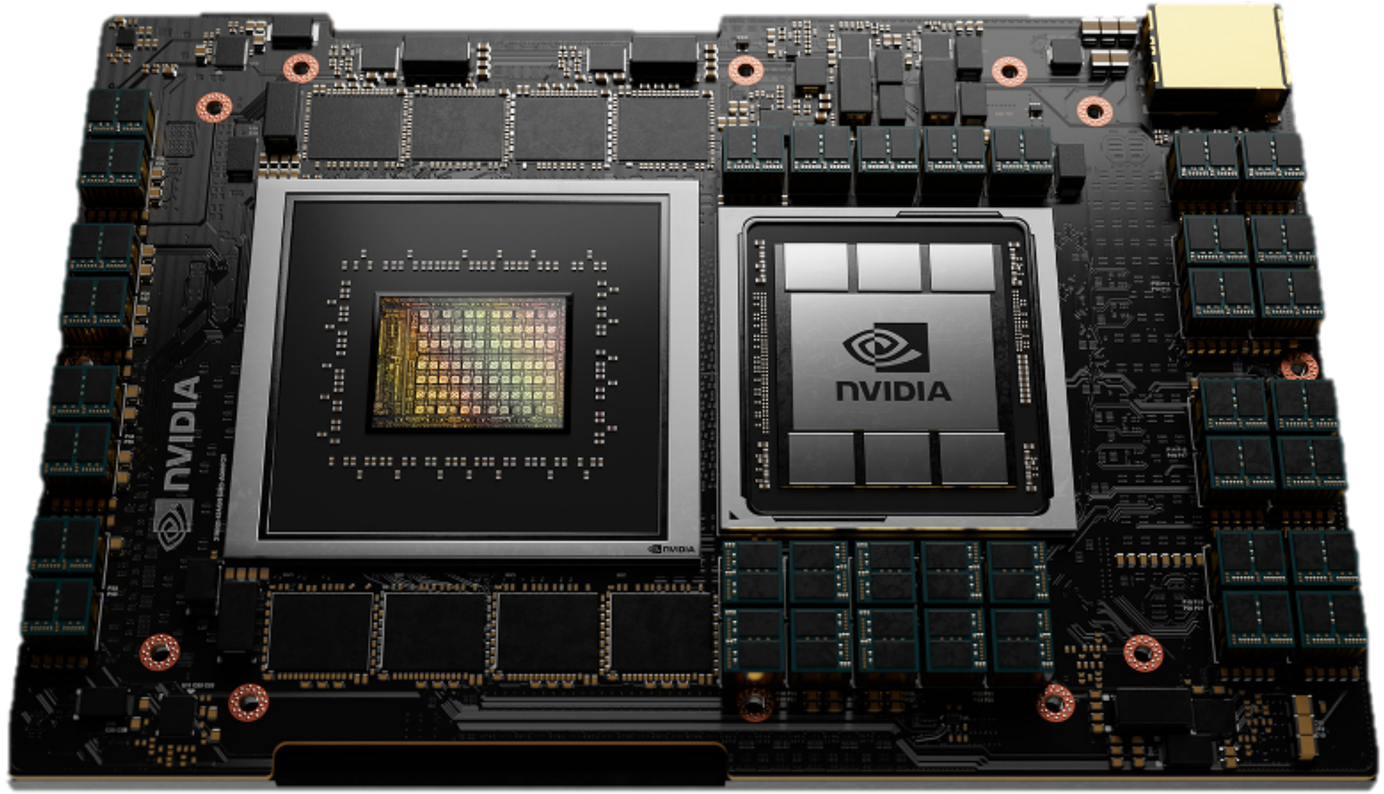
System Bandwidth Bottleneck



GPU	8,000	GB/sec
CPU	200	GB/sec
PCIE Gen4 (Effective Per GPU)	16	GB/sec
Mem-to-GPU	**64**	**GB/sec**

NVIDIA GRACE

Breakthrough CPU Designed for Giant-Scale AI and HPC Applications



FASTEST INTERCONNECTS

>900 GB/s Cache Coherent NVLink CPU To GPU (14x)
>600GB/s CPU To CPU (2x)

HIGHEST MEMORY BANDWIDTH

>500GB/s LPDDR5x w/ ECC
>2x Higher B/W
10x Higher Energy Efficiency

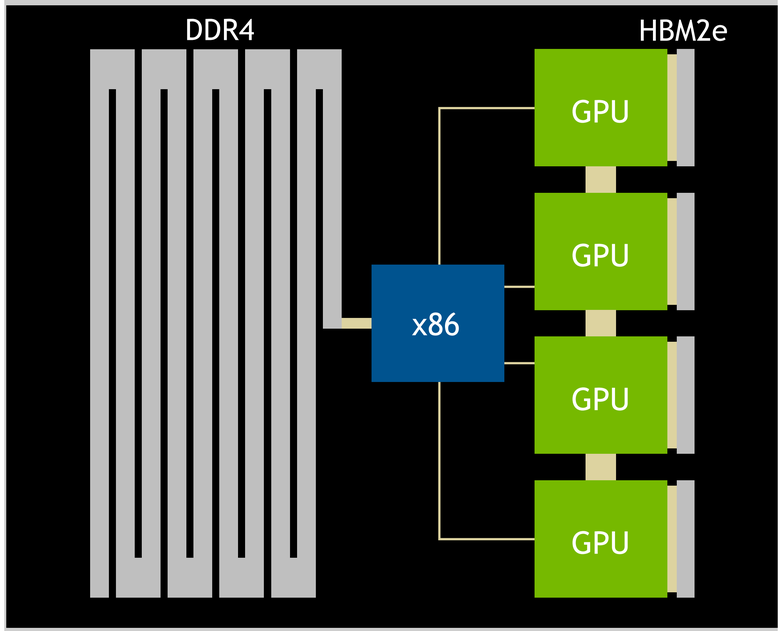
NEXT GENERATION ARM NEOVERSE CORES

>300 SPECrate2017_int_base est.
Availability 2023

TURBOCHARGED TERABYTE SCALE ACCELERATED COMPUTING

Evolving Architecture For New Workloads

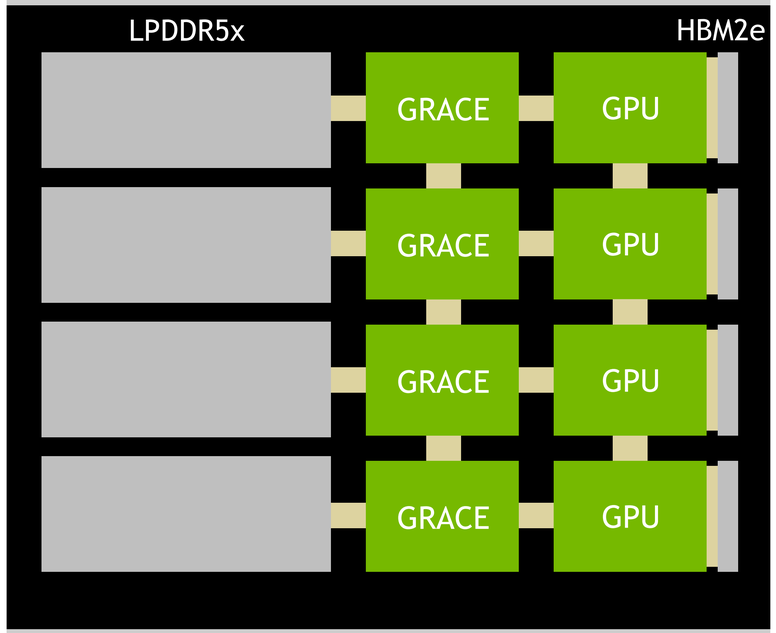
CURRENT x86 ARCHITECTURE



GPU	8,000	GB/sec
CPU	200	GB/sec
PCIE Gen4 (Effective Per GPU)	16	GB/sec
Mem-to-GPU	**64**	**GB/sec**

Transfer 2TB in 30 secs

INTEGRATED CPU-GPU ARCHITECTURE



GPU	8,000	GB/sec
CPU	500	GB/sec
NVLink	500	GB/sec
Mem-to-GPU	**2,000**	**GB/sec**

Transfer 2TB in 1 secs

3 DAYS FROM 1 MONTH
Fine-Tune Training of 1T Model

REAL-TIME INFERENCE
ON 0.5T MODEL
Interactive Single Node NLP Inference

Bandwidth claims rounded to nearest hundred for illustration.
Performance results based on projections on these configurations Grace : 8xGrace and 8xA100 with 4th Gen NVIDIA NVLink Connection between CPU and GPU and x86: DGX A100.
Training: 1 Month of training is Fine-Tuning a 1T parameter model on a large custom data set on 64xGrace+64xA100 compared to 8xDGXA100 (16xX86+64xA100)
Inference: 530B Parameter model on 8xGrace+8xA100 compared to DGXA100.

NVIDIA

ANNOUNCING THE WORLD'S FASTEST SUPERCOMPUTER FOR AI

20 Exaflops of AI

Accelerated w/ NVIDIA Grace CPU and NVIDIA GPU

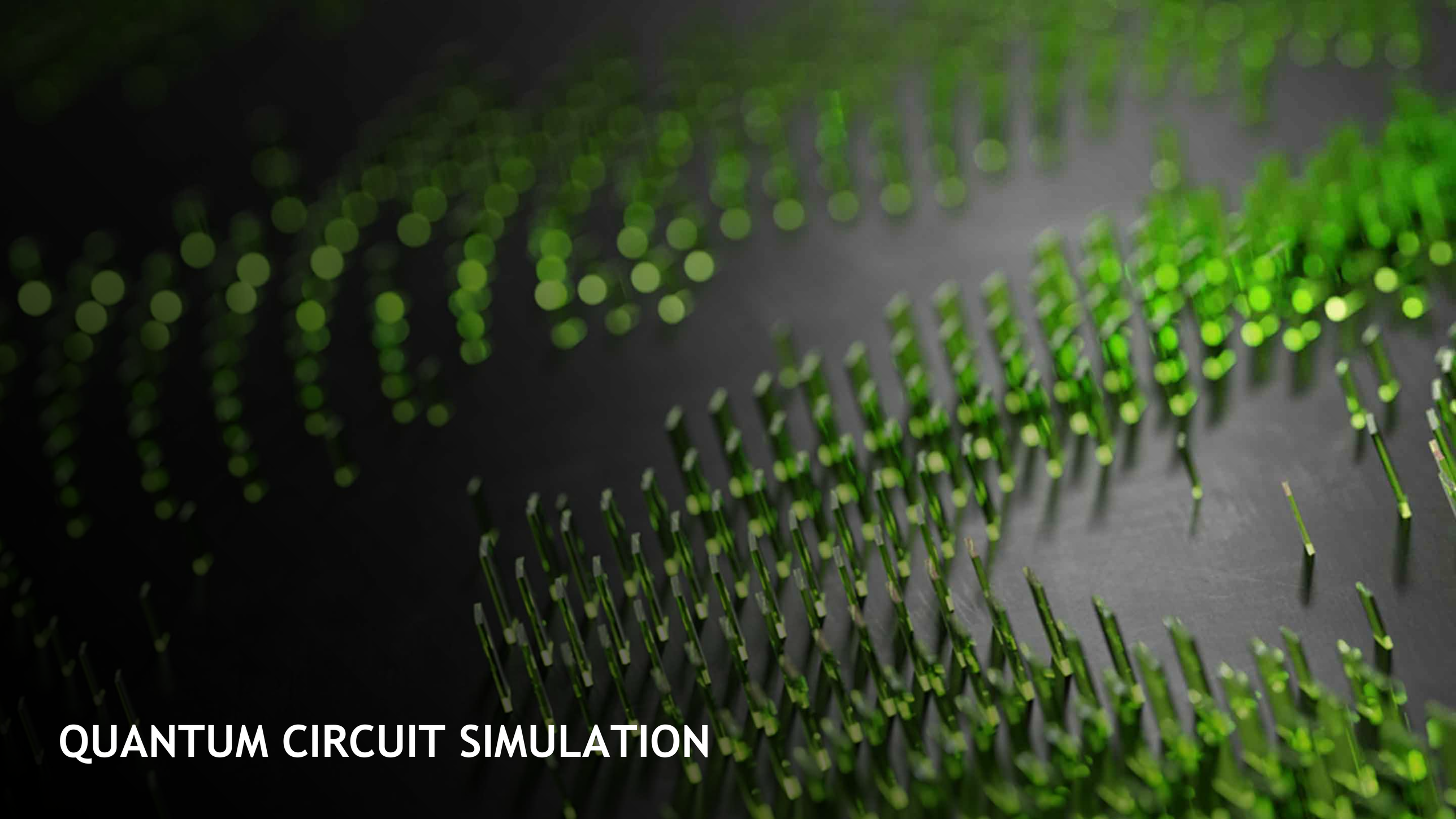
HPC and AI For Scientific and Commercial Apps

Advance Weather, Climate, and Material Science

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

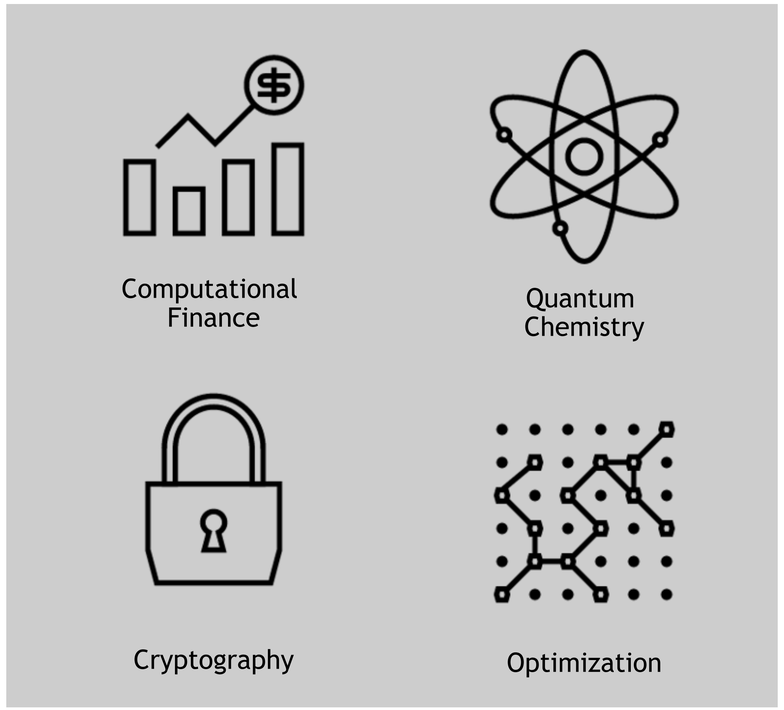
Hewlett Packard
Enterprise

NVIDIA

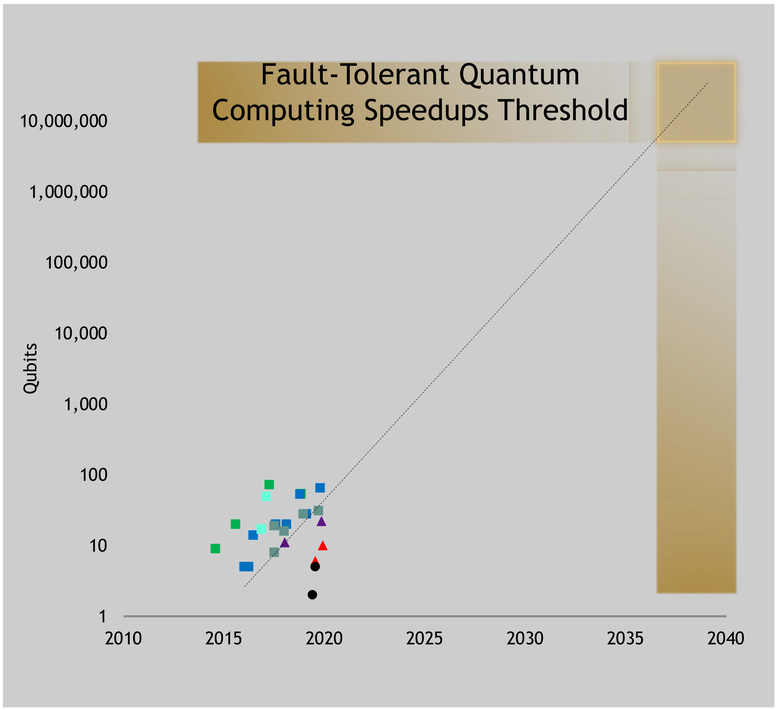

QUANTUM CIRCUIT SIMULATION

A NEW COMPUTING MODEL - QUANTUM

NEW COMPUTING MODEL



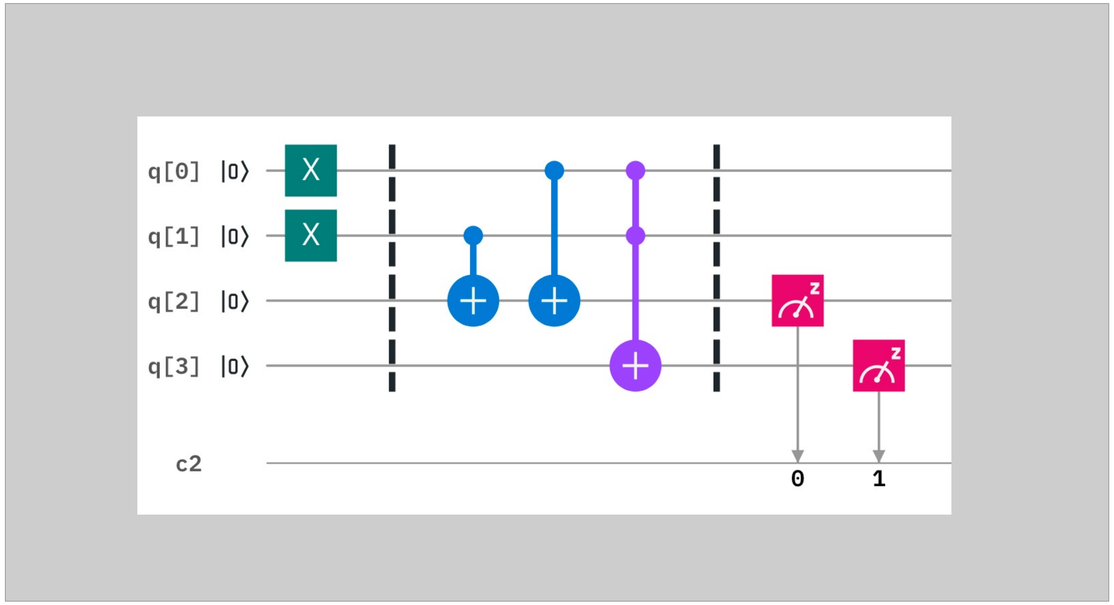
POTENTIAL USE CASES



QUANTUM SYSTEMS SCALING EXPONENTIALLY

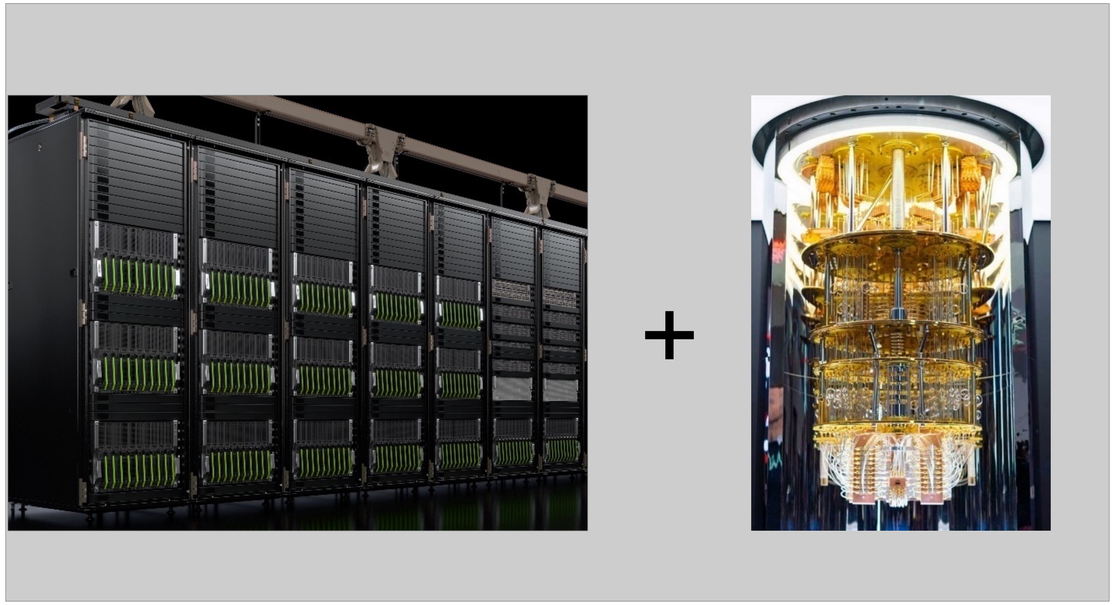
GPU-BASED SUPERCOMPUTING IN THE QC ECOSYSTEM

Researching the quantum computers of tomorrow with the supercomputers of today



Quantum Circuit Simulation

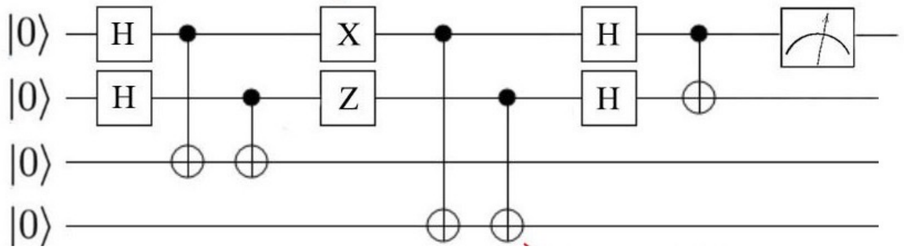
Critical tool for answering today's most pressing questions in Quantum Information Science (QIS):
Can entangled qubits be simulated efficiently on classical supercomputers?
Will NISQs have quantum advantage on useful workloads?
What are the best error correction algorithms for getting to fault tolerance?



Hybrid Classical/Quantum Applications

Impactful QC applications (e.g. simulating quantum materials and systems) will require classical supercomputers with quantum co-processors

TWO MOST POPULAR QUANTUM CIRCUIT SIMULATION APPROACHES



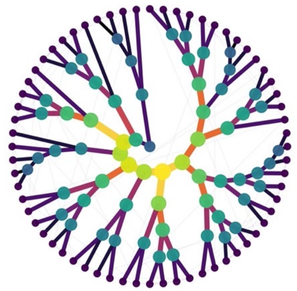
State vector simulation

“Gate-based simulation of a quantum computer”

- Maintain full 2^n qubit vector state in memory
- Update all states every timestep, probabilistically sample n of the states for measurement

Memory capacity & time grow exponentially w/ # of qubits - practical limit around 50 qubits on a supercomputer

Can model either ideal or noisy qubits



Tensor networks

“Only simulate the states you need”

- Uses tensor network contractions to dramatically reduce memory for simulating circuits
- Can simulate 100s or 1000s of qubits for many practical quantum circuits

GPUs are a great fit for either approach

Tensor Network image from Quimb: https://quimb.readthedocs.io/en/latest/index.html

CUQUANTUM

SDK of optimized libraries and tools for accelerating quantum computing workflows

Enabling Quantum Computing Research

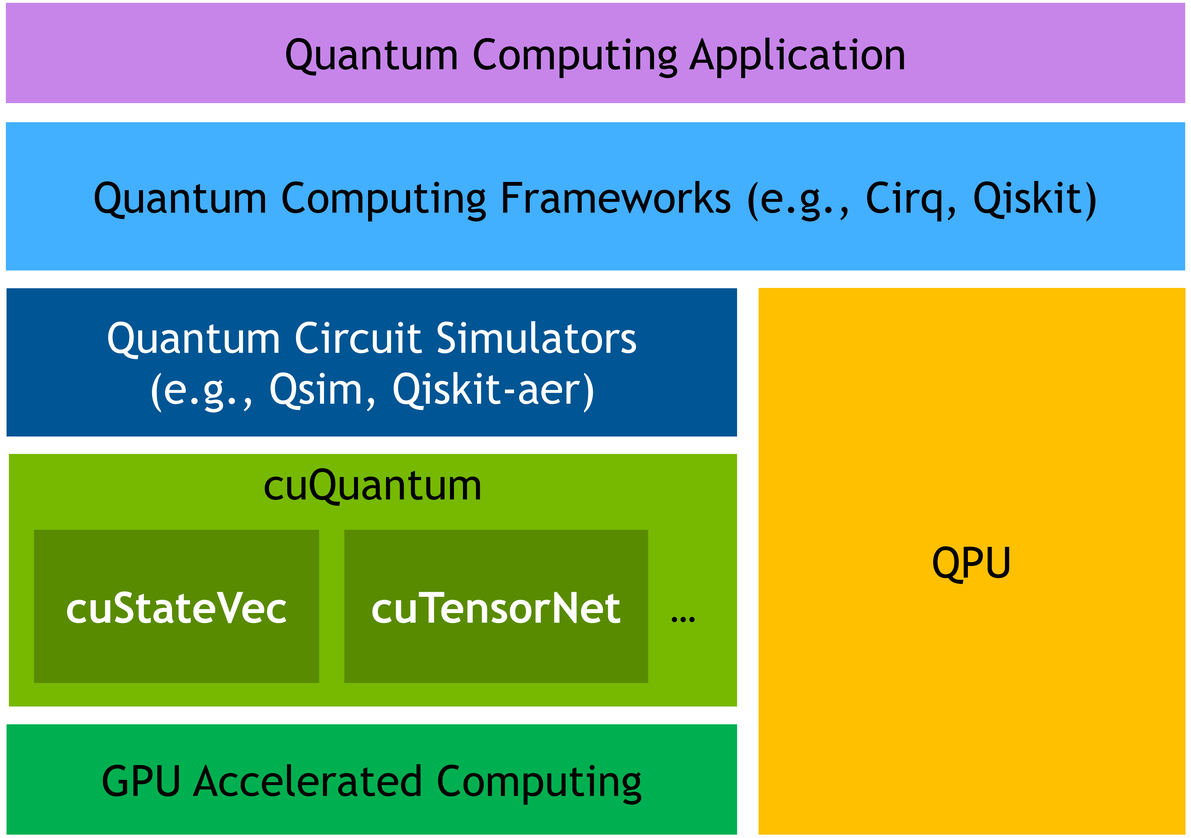
- Accelerate Quantum Circuit Simulators on GPUs
- Enable algorithms research with scale and performance not possible on quantum hardware, or on simulators today

Platform for Algorithm Development

- Leverage Accelerated Circuit Simulators or Quantum Processors
- Enable development of algorithms for scientific computing on hybrid Quantum/Classical systems

Open Beta Available Now

- Integrated in leading quantum computing frameworks Cirq and Qiskit
- Available today at developer.nvidia.com/cuquantum



まとめ

- GTC21 / SC21 HPC Quick Update
- NVIDIA GPU
- Programming
- HPC Application Performance
- NVIDIA Grace CPU
- Quantum Circuit Simulation

WE ARE HIRING !

Compute Performance Developer Technology Engineer

Japan, Tokyo

Apply

NVIDIA is hiring passionate, world-class computer scientists to work in its Compute Developer Technology (Devtech) team.

What you will be doing:

In this role, you will research and develop techniques to GPU-accelerate leading applications in high performance computing fields within scientific computing, computational engineering, and data science. You will be performing in-depth analysis and optimization to ensure the best possible performance on current and next-generation GPU architectures. This involves:

- Working directly with key application developers to understand the current and future problems they are solving, crafting and optimizing core parallel algorithms and data structures to provide the best solutions using GPUs, through both reference code development and direct contribution to the applications.
- Collaborating closely with diverse groups at NVIDIA such as the architecture, research, libraries, tools, and system software teams to influence the design of next-generation architectures, software platforms, and programming models, by investigating the impact on application performance and developer productivity.
- You will need to travel from time to time for conferences and for on-site visits with developers.

What we need to see:

- BS, MS, or PhD degree from a leading university in an engineering or computer science related discipline (or equivalent experience). While not a requirement, domain expertise in telecommunications, medical imaging, machine learning, deep learning, or natural sciences is helpful.
- Programming fluency in C/C++ and/or Fortran with a deep understanding of software design, programming techniques, and algorithms.
- Strong mathematical fundamentals, including linear algebra and numerical methods.
- Experience with parallel programming, ideally CUDA C/C++ and OpenACC.
- Strong communication and organization skills, with a logical approach to problem solving, good time management, and task prioritization skills.

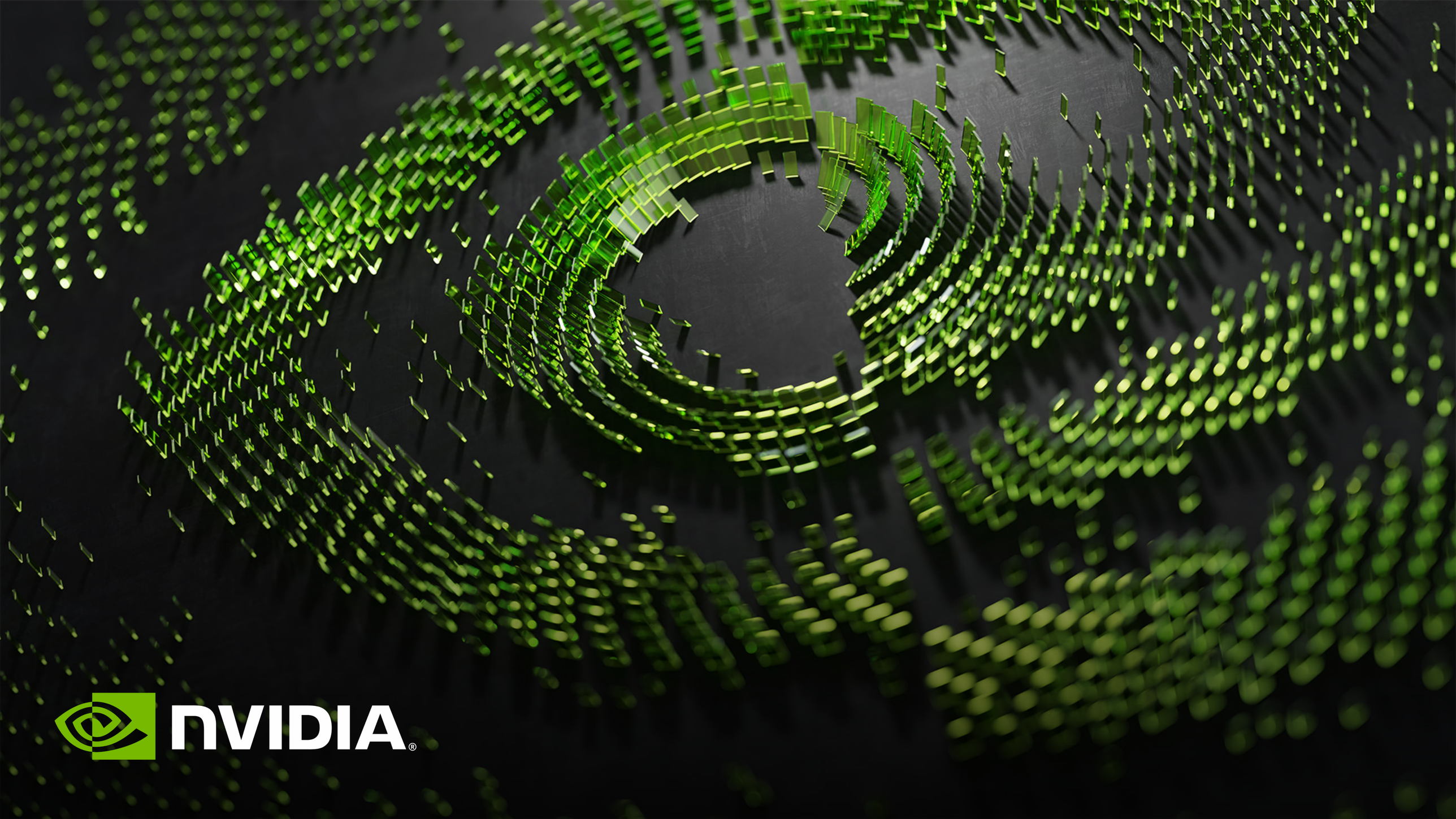
With highly competitive salaries and a comprehensive benefits package, NVIDIA is widely considered to be one of the technology industry's most desirable employers. We have some of the most brilliant and talented people in the world working with us and our engineering teams are growing fast in some of the hottest and state of the art fields: Deep Learning, Artificial Intelligence, Autonomous Vehicles, Supercomputing and more. Are you a creative and autonomous computer scientist with a real passion for parallel computing? If so, we want to hear from you.

https://nvidia.wd5.myworkdayjobs.com/en-US/NVIDIAExternalCareerSite/job/Japan-Tokyo/Compute-Performance-Developer-Technology-Engineer_JR1945486

NVIDIA

参考情報

- NVIDIA cuNumeric
  - https://developer.nvidia.com/cunumeric
- NVIDIA Modulus
  - https://developer.nvidia.com/modulus
- NVIDIA HPC SDK
  - https://developer.nvidia.com/hpc-sdk
- NGC Catalog
  - https://catalog.ngc.nvidia.com/
- NVIDIA HPC Application Performance
  - https://developer.nvidia.com/hpc-application-performance
- NVIDIA cuQuantum
  - https://developer.nvidia.com/cuquantum-sdk


NVIDIA