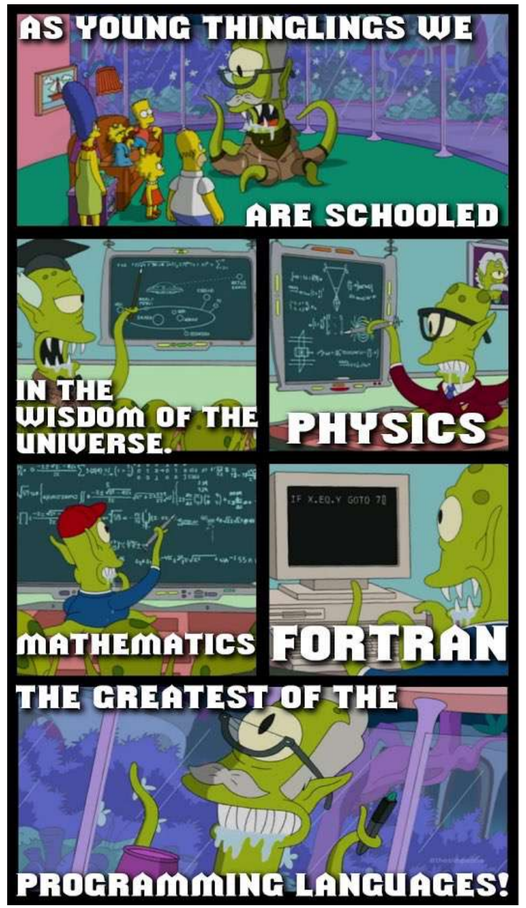
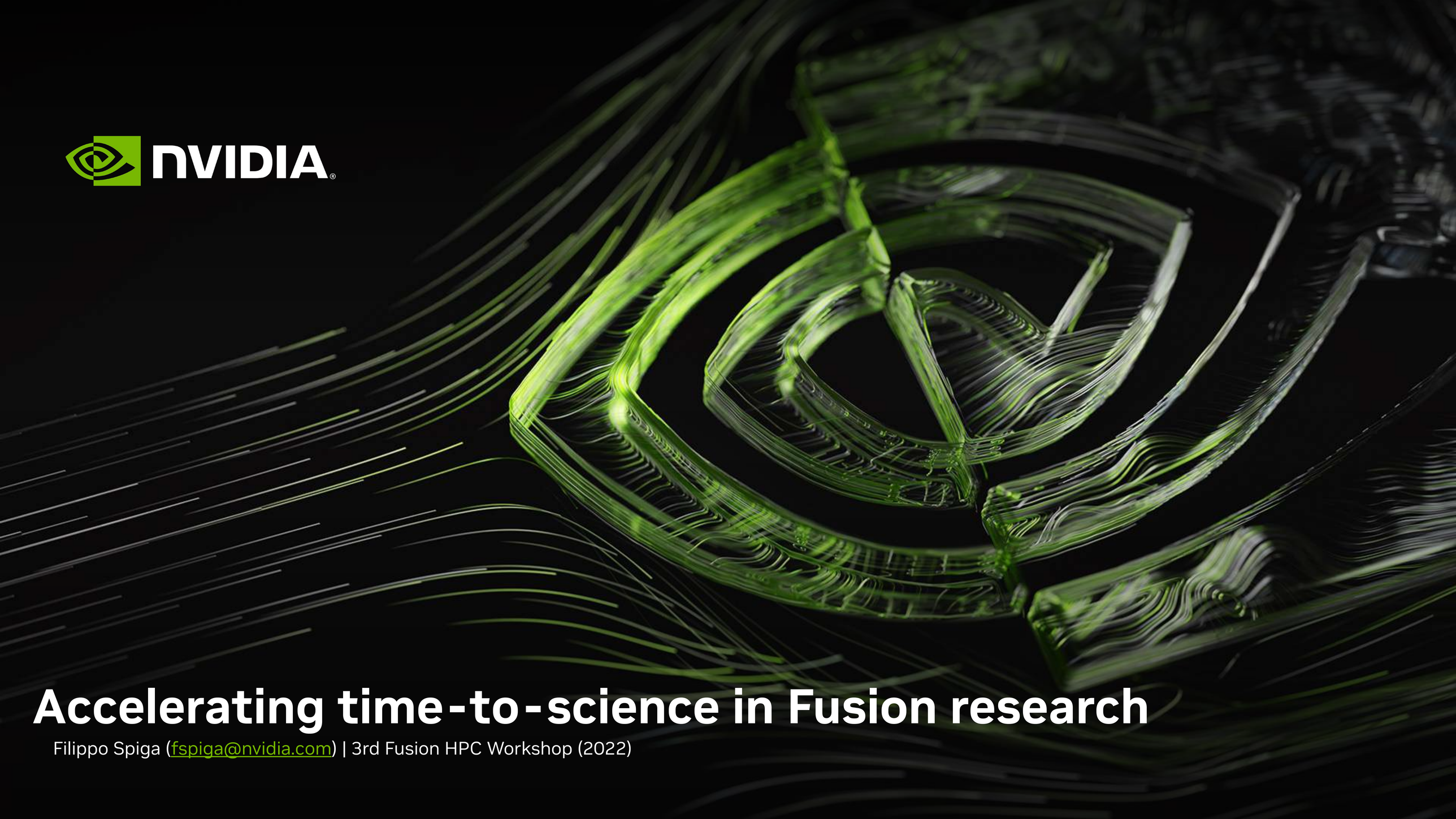
About Me

Filippo Spiga

- Joined NVIDIA February 2020 starting as **EMEA HPC Developer Relations**, now also covering also **WW Arm HPC Ecosystem Readiness** for Tesla Business Unit.
- Previously at Arm Research (2 years), University of Cambridge (5 years) and few other places (IBM Research, ICHEC, CINECA, INFN/CERN).
- Graduated MSc in Computer Science, converted Computational Scientist working mainly on Quantum Chemistry and CFD.
- Outspoken supporter and true believer of Fortran.
- Based in Cambridge (UK)



https://github.com/fspiga https://www.linkedin.com/in/filippospiga/ https://twitter.com/filippospiga



Accelerating time-to-science in Fusion research

Filippo Spiga (fspiga@nvidia.com) | 3rd Fusion HPC Workshop (2022)

Contents

A tiny bit of HW

NVIDIA H100

Unprecedented Performance, Scalability, and Security for Every Data Center

Highest AI and HPC Performance

4PF FP8 (6X)| 2PF FP16 (3X)| 1PF TF32 (3X)| 67TF FP64 (3.4X)
3.35TB/s (1.5X), 80GB HBM3 memory

Transformer Model Optimizations

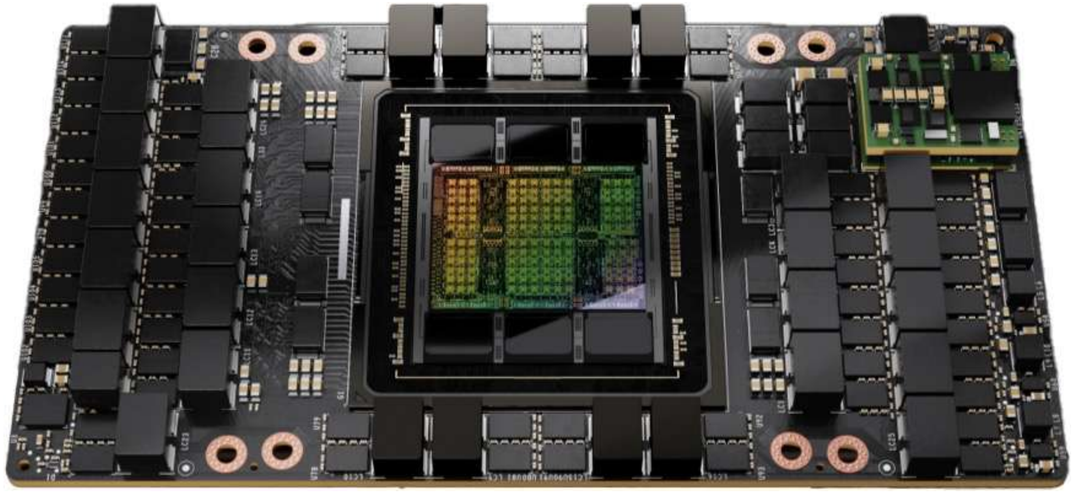
6X faster on largest transformer models

Highest Utilization Efficiency and Security

7 Fully isolated & secured instances, guaranteed QoS
2nd Gen MIG | Confidential Computing

Fastest, Scalable Interconnect

900 GB/s GPU-2-GPU connectivity (1.5X)
up to 256 GPUs with NVLink Switch | 128GB/s PCI Gen5



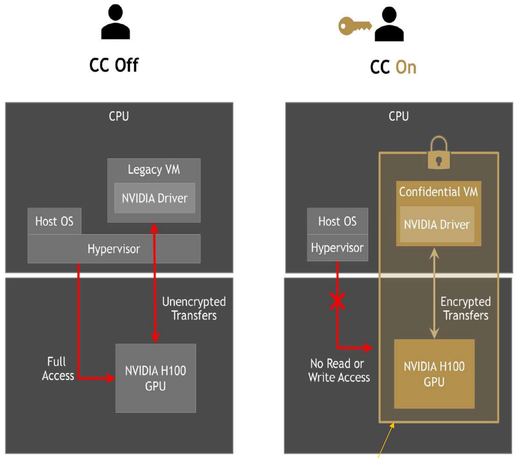
Blog "*NVIDIA Hopper Architecture In-Depth*": https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

FP8, FP16, TF32 performance include sparsity. X-factor compared to A100

HOPPER TECHNOLOGICAL BREAKTHROUGHS

CONFIDENTIAL COMPUTING

Secure Data and AI Models In-Use

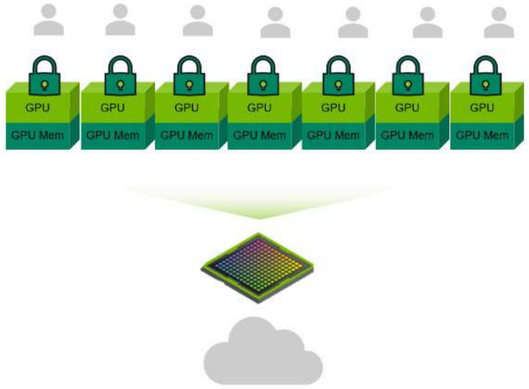


Confidential VM
(virtualization-based TEE)

Secure during compute, not just in storage or in motion

MULTI-GPU INSTANCE

7 Secure Tenants on 1 GPU



New vs A100:

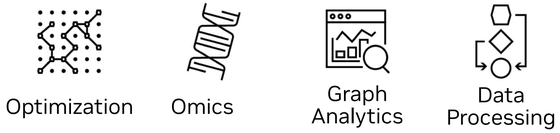
- *6x compute, more mem bandwidth*
- *Dedicated NVDEC & NVJPG per MIG*
- *Each instance has its own telemetry*
- *Secure confidential compute MIG*

3x Compute
1.5x Bandwidth

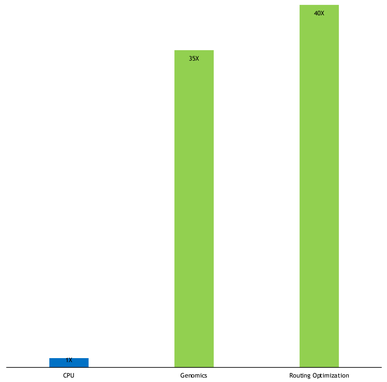
NEW DYNAMIC PROGRAMING INSTRUCTIONS

Accelerate Dynamic Programming Algorithms

A BROAD RANGE OF USE CASES



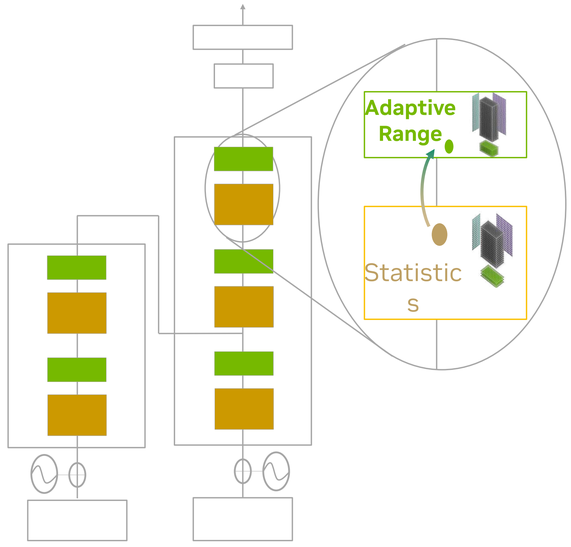
REAL-TIME PERFORMANCE



40x vs CPU
7x vs A100

TRANSFORMER ENGINE

Tensor Core Optimized for Transformer Models



Statistics and Adaptive Range Tracking

16-bit 8-bit

6x on LLM

Dynamic Programming Instructions HGX H100 4-GPU vs dual socket 32 core IceLake. HGX H100 performance projections.

SCALING THE CUDA PROGRAMMING MODEL

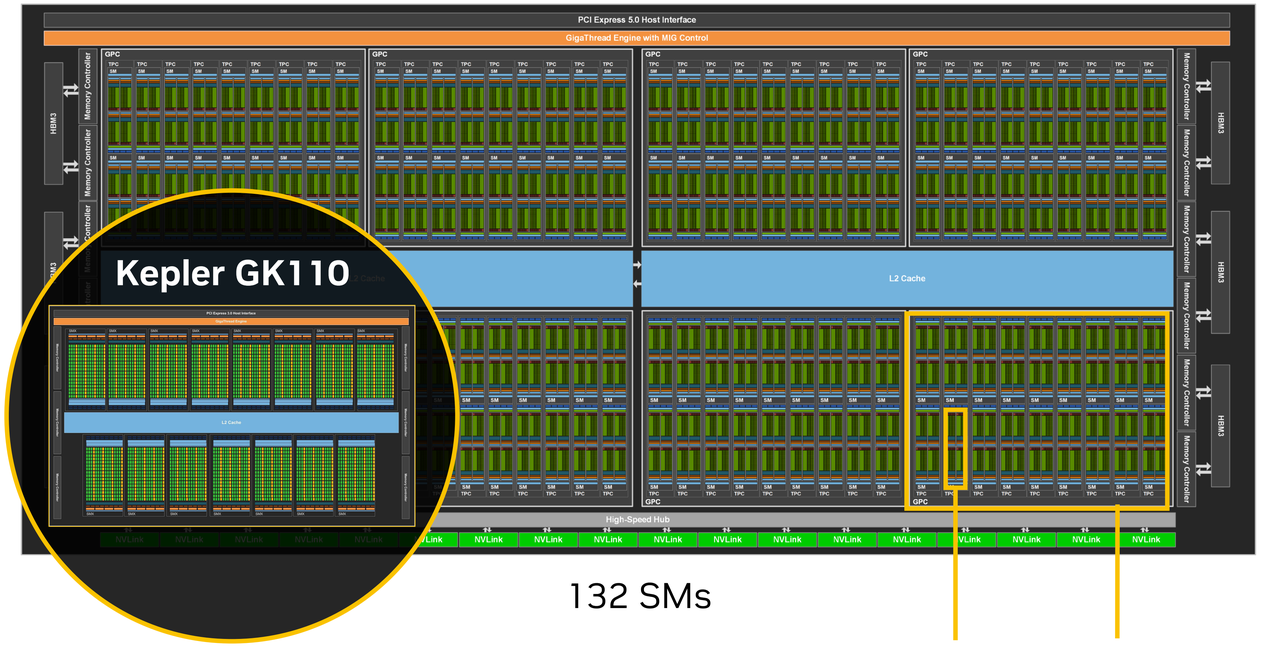
Kepler GK110 GPU, 2012

15 SMs

SM

3.52 TFLOPS single
1.17 TFLOPS double

Hopper H100 GPU, 2022



132 SMs

SM **GPC**

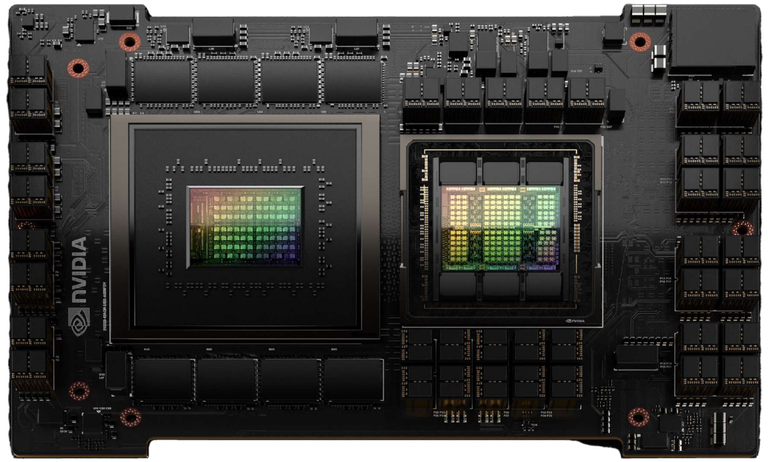
67 TFLOPS single **[19x]**
34 TFLOPS double **[29x]**
67 TFLOPS double with TC **[57x]**

NVIDIA

NVIDIA GRACE FOR HPC & AI INFRASTRUCTURE

Grace Hopper Superchip

Giant Scale AI & HPC



Accelerated applications where CPU performance and system memory BW are critical; extreme and highly atomic collaboration between CPU & GPU contexts for flagship AI & HPC

Grace CPU Superchip

CPU Computing

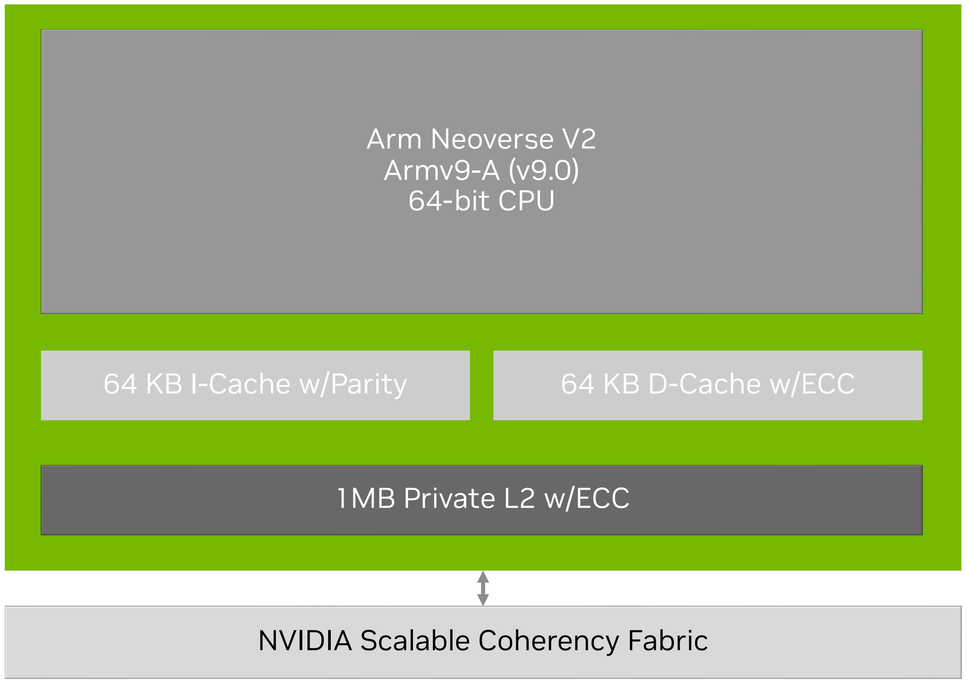
Applications that run on CPU but where absolute performance, energy efficiency, and datacenter density matter, such as in scientific computing, data analytics, and hyperscale computing applications

Designed from the ground-up to be a Superchip, always paired

NVIDIA Grace

Introducing Neoverse V2

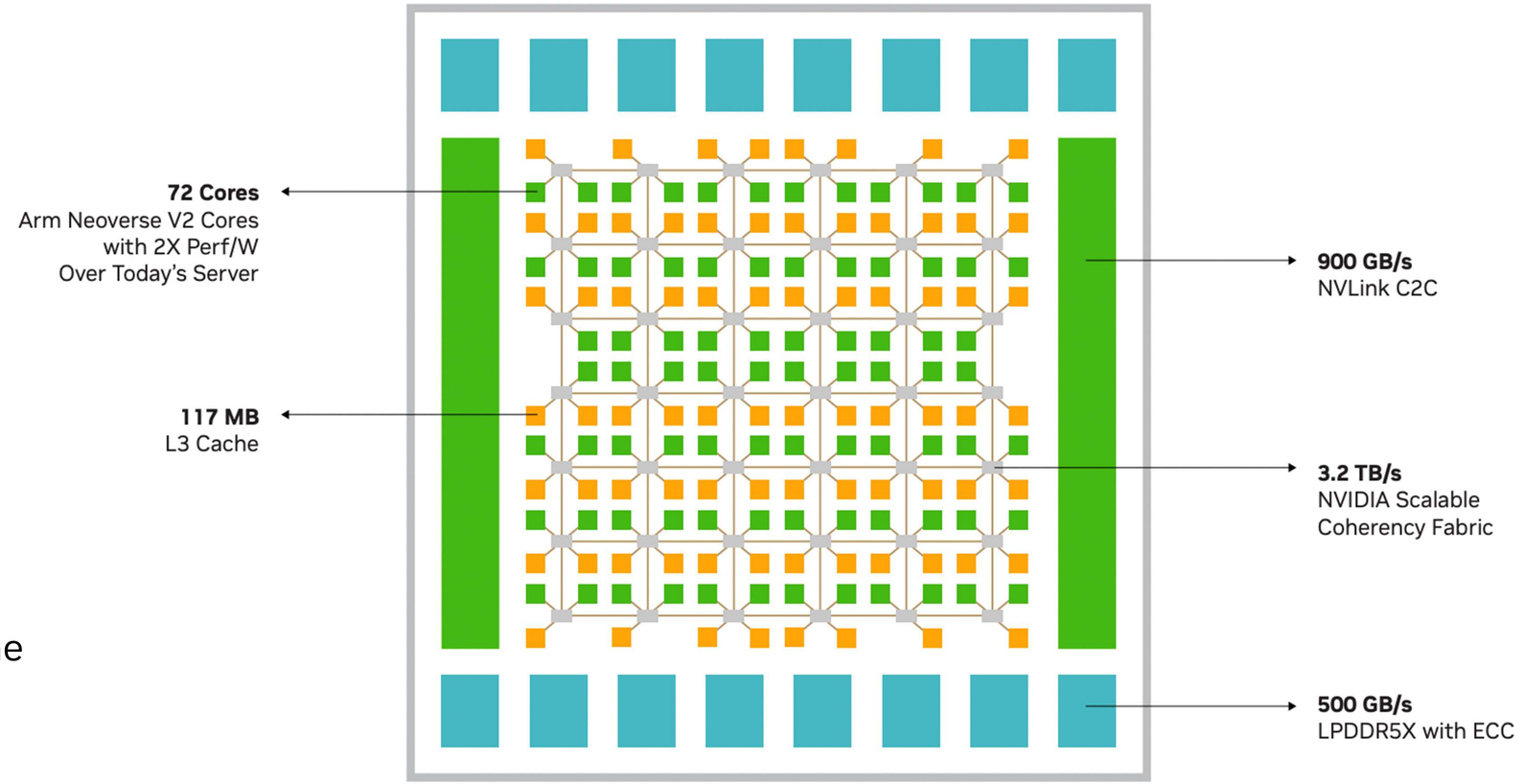
- **Arm Neoverse V2 core – Arm v9.0**
- AARCH64 at all ELs
- v9.0 scalable vector extensions
  - **Scalable Vector Extension 2 (SVE2) - 4 x 128b**
  - Scalable Vector AES (SVE_AES)
  - Scalable Vector PMULL (SVE_PMULL)
  - Scalable Vector SHA3 (SVE_SHA3)
  - Scalable Vector Pit Permutes (SVE_BitPerm)
- V9.0 debug
  - Embedded Trace Extension (ETE)
  - Trace Buffer Extension (TBE)



GRACE IS A COMPUTE & DATA MOVEMENT ARCHITECTURE

NVIDIA Scalable Coherency Fabric (SCF) and distributed cache design

- Up to 512GB of LPDDR5X memory
  - **32 channels**
  - **Up to 546 GB/s of memory BW**
  - **Competitive power/perf**
- NVIDIA Scalable Coherency Fabric
  - 3,225.6 GB/s bi-section BW
  - **117MB of distributed L3 cache**
  - Scalable to 72+ cores per die
  - Background data movement via Cache Switch Network
- Supports up to 4-die coherency over Coherent NVLINK

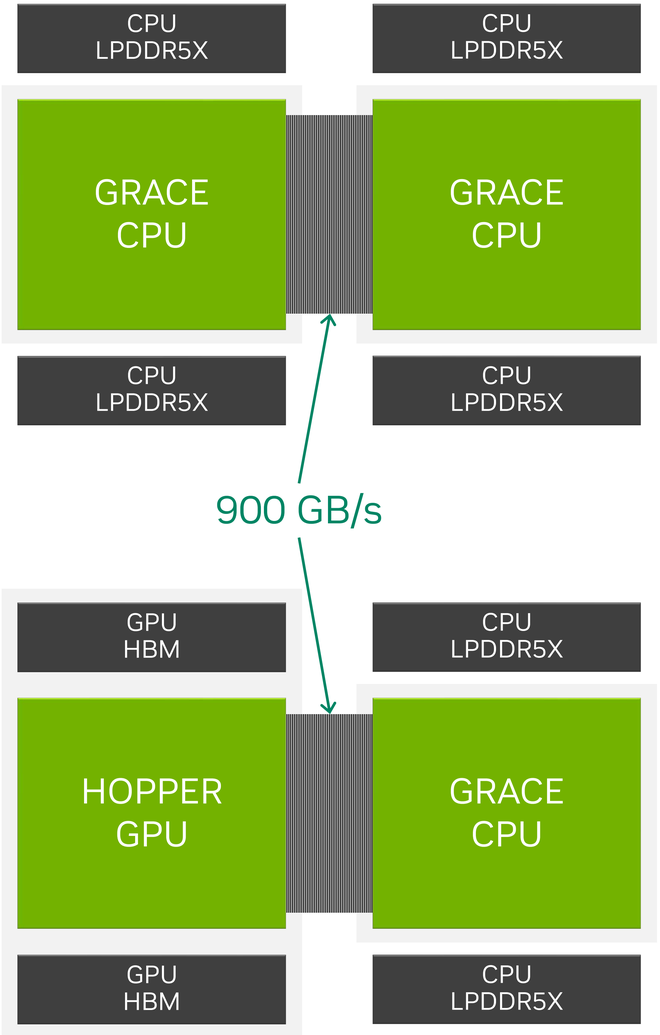


Example possible fabric topology for illustrative purposes

NVLINK-C2C

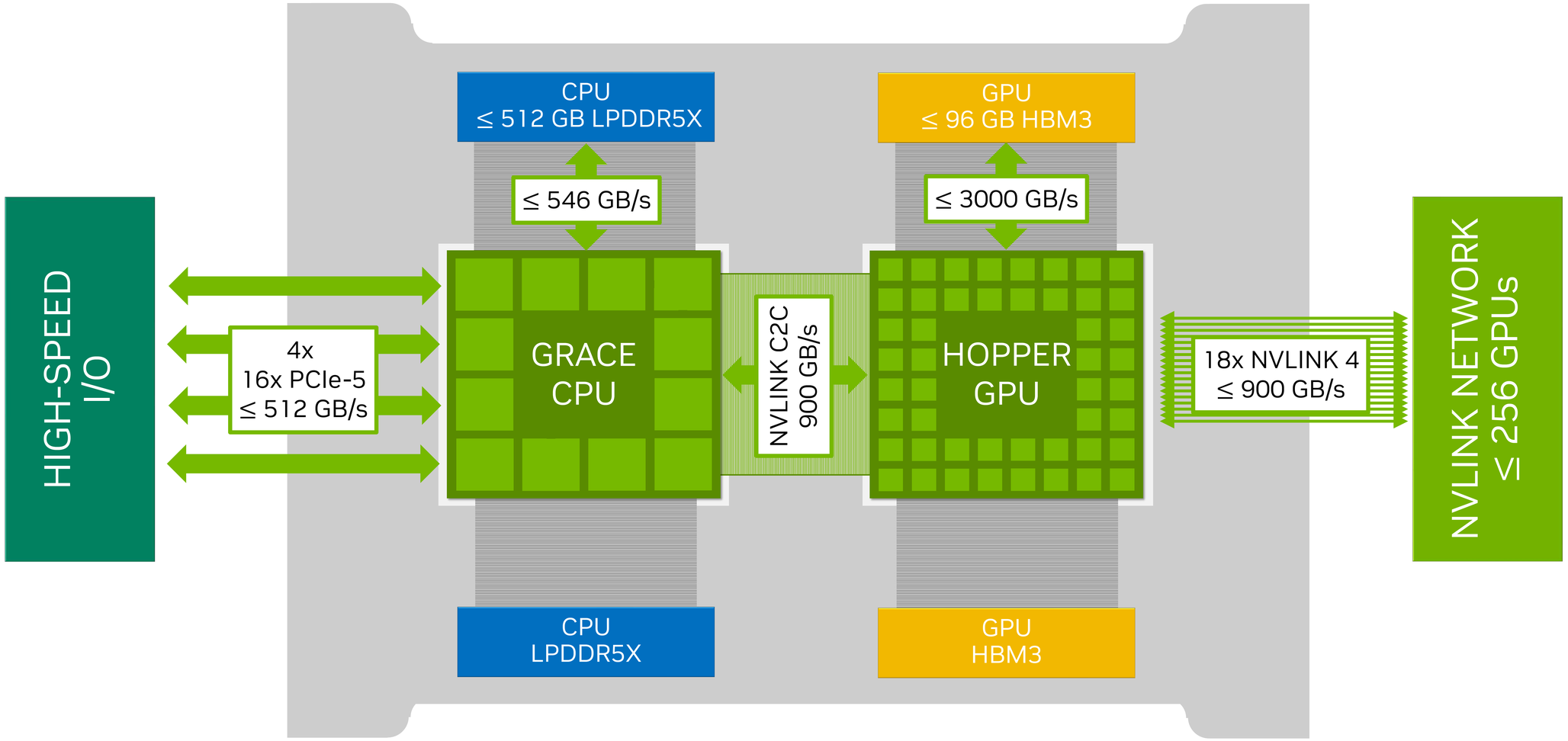
High Speed Chip to Chip Interconnect

- Used to create the Grace Hopper, and Grace Superchips
  - **Native atomics, including standard C++ atomic support**
  - Enables coherency
- Up to 900 GB/s of raw bidirectional BW
  - Same BW as GPU to GPU NVLINK on Hopper
- Low power interface - 1.3 pJ/bit
  - **More than 5x more power efficient than PCIe**
- Unified Memory with shared page tables
  - **Shared CPU and GPU virtual address space (AST)**



Grace Hopper Superchip Platform

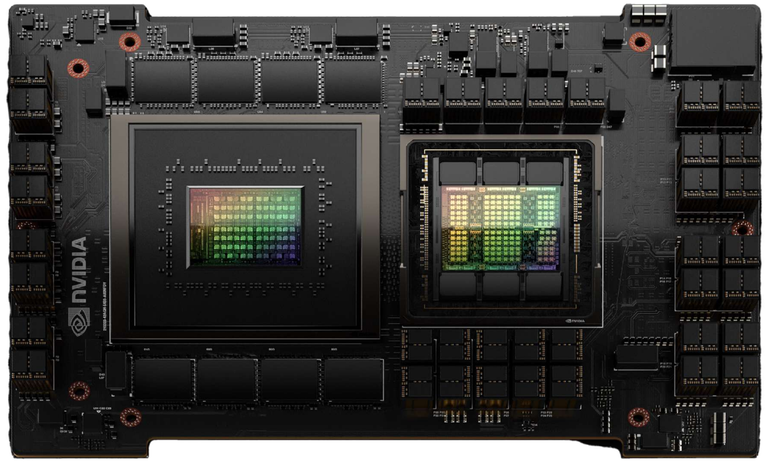
Speeds and Feeds



NVIDIA GRACE FOR HPC & AI INFRASTRUCTURE

Grace Hopper Superchip

Giant Scale AI & HPC



1000W max
(including LPDDR5x and HBM3 memory)

Grace CPU Superchip

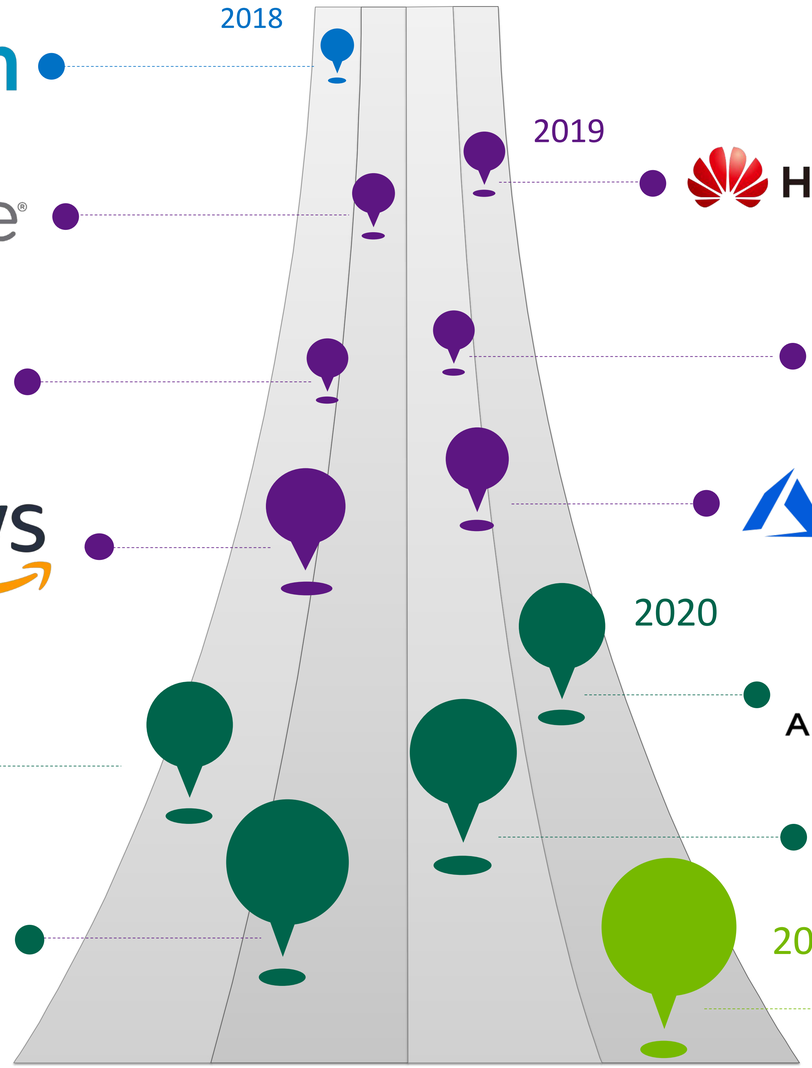
CPU Computing

500W max
(including LPDDR5x memory)

Designed from the ground-up to be a Superchip, always paired

Arm HPC ecosystem

ARM'S RISE IN HPC



2018

Arm Neoverse Announced

VMware demo'd ESXi on 64b Arm

Neoverse N1 & E1 Platform announced

AWS announced 2nd Generation Arm-based Graviton2 Server CPU

2019

Huawei released Kunpeng 920 CPU & TaiShan server platform

Nvidia CUDA stack to Arm Platforms

Marvell's ThunderX2 Solution for Microsoft Azure

2020

Ampere announced industry's 1st 80-Core Server SoC (128 Altra Max)

Neoverse V1 License

Marvell ThunderX2 Server SoC

Fujitsu Fugaku Riken #1 on Top500

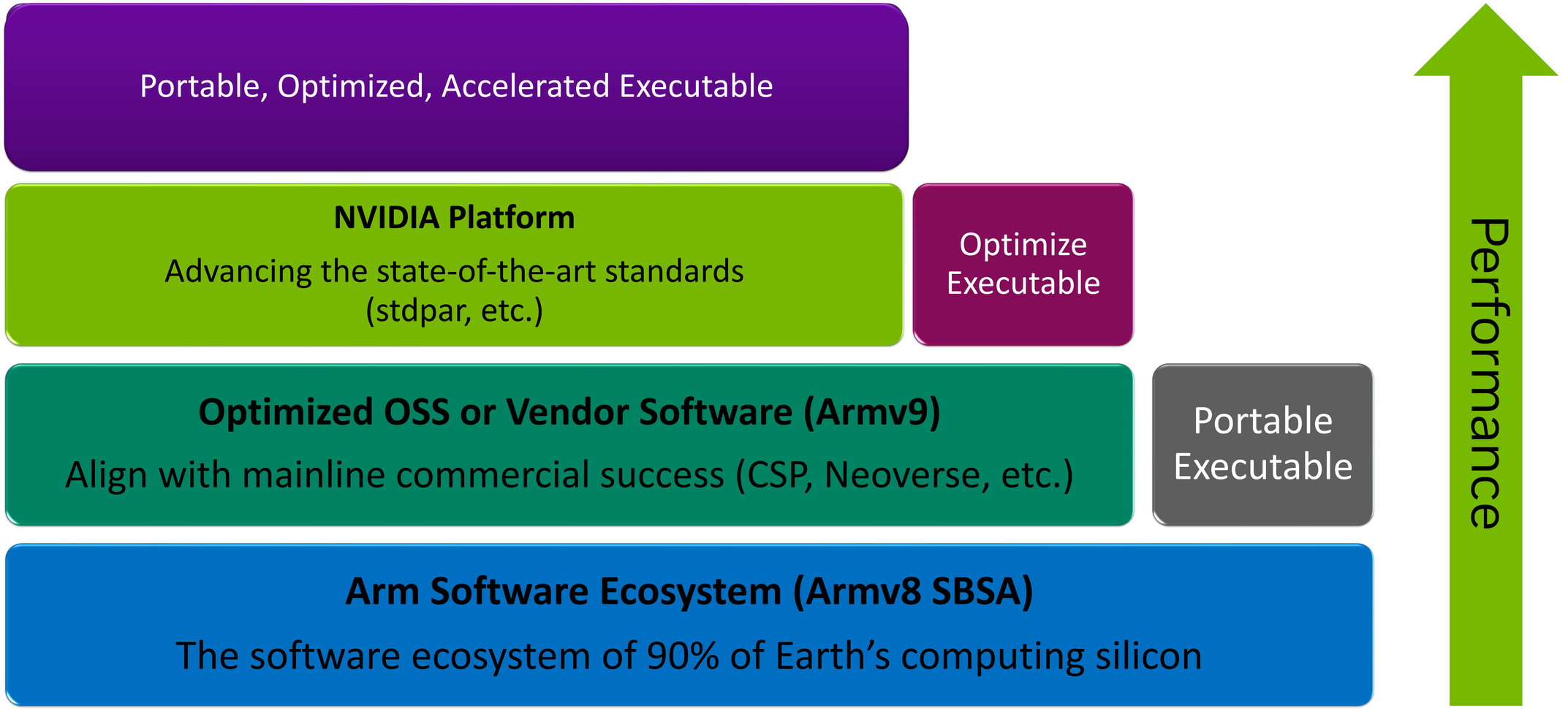
2021

Grace is announced

Source: https://gtc21.event.nvidia.com/media/HPC%20Applications%20on%20ARM%2BNVIDIA%20A100%20%5BS32758%5D/1_cynpwxsp

GRACE SOFTWARE ECOSYSTEM IS BUILT ON STANDARDS

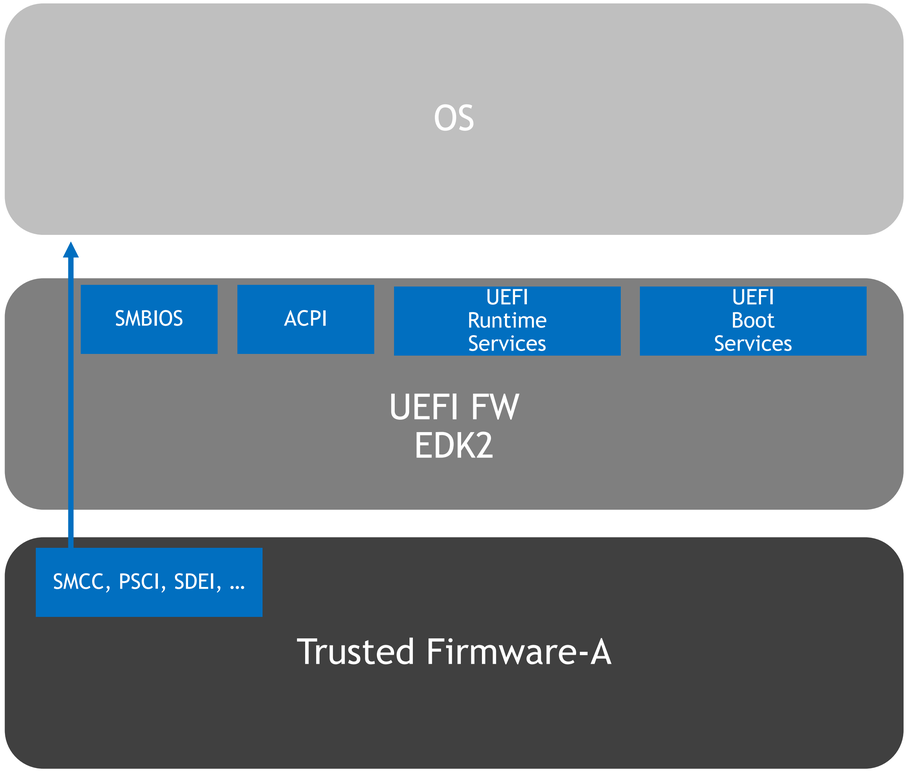
The NVIDIA platform builds on optimized software from the broad Arm software ecosystem



GRACE

Server Base System Architecture (SBSA), and Base Boot Requirements (BBR)

- Standard set of platform requirements and recommendations to enable off-the-shelf OS support
- **“It just works” with a standard OS**
- SBBR recipe support from BBR
- Allows OS and system SW to expect consistency across different SOCs
  - Standard Private Peripheral Interrupt (PPI) assignments
  - Standard UART
  - PCIe – ECAM, ITS for MSI(-X)



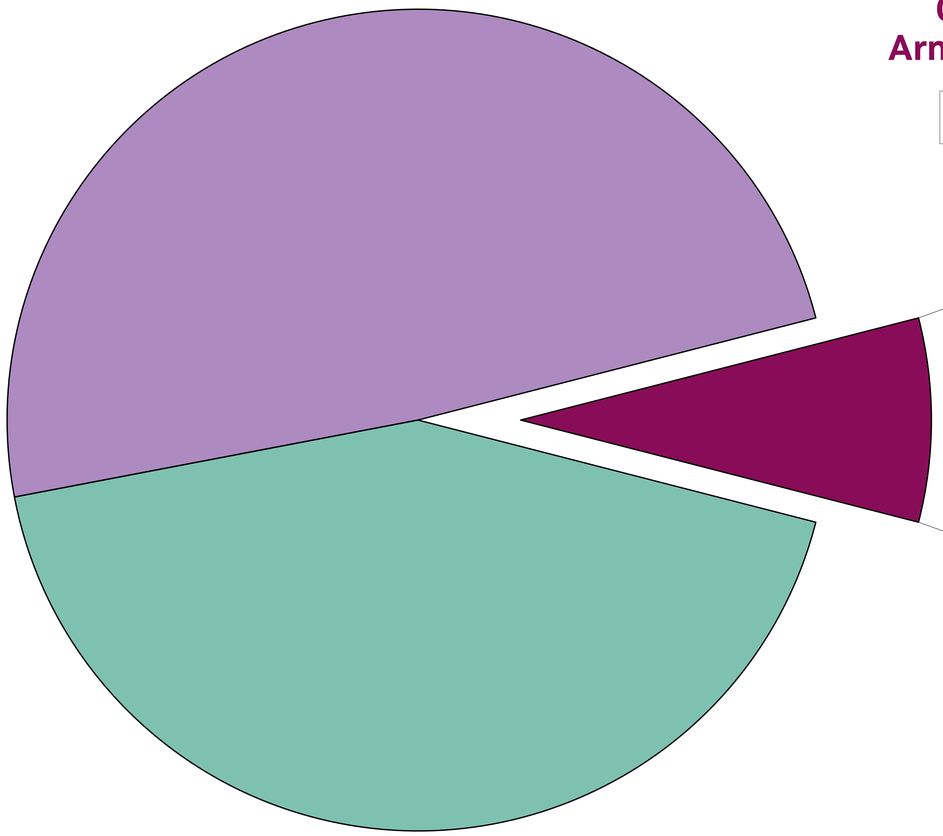
arm SystemReady

APPLICATION PORTING: MANY NON-TRIVIAL CASES REALLY ARE TRIVIAL

Vector intrinsics, dependencies, and nonstandard features are easily ported

Straightforward, easy work < 1 day

Recompile and reconfigure runtime parameters

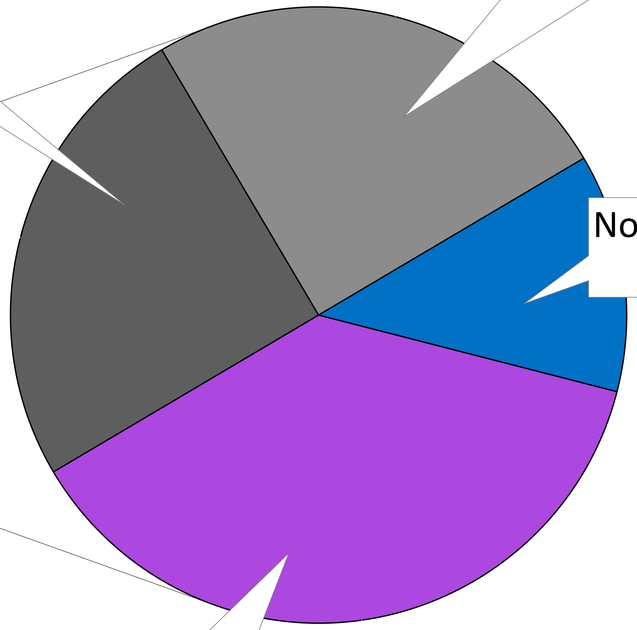


Job done!

Found on Arm at another HPC center

Cloud momentum
Arm ecosystem growth

Dependency



Assembly Language

Compiler translation guides

Nonstandard Compiler Features

Vector Intrinsics

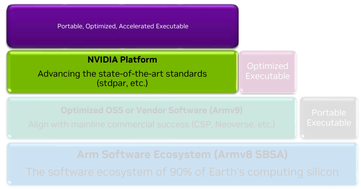
Intrinsic conversion tools – SIMDe, SSE2NEON, etc.

**Indicative workload mix inspired by an US DoE lab usage*

ARM HPC DEVKIT DEPLOYMENTS

~100 DevKit worldwide

Programming the NVIDIA Platform



The NVIDIA HPC Software Platform

HPC Applications

Simulation

Data Analytics

AI

HPC SDK

COMPILERS

NVC++ | NVC | NVFORTRAN | NVCC

CORE C++ LIBRARIES

CUB | Thrust | libcu++

PROFILING AND DIAGNOSTICS

CUDA-gdb | NSight Compute | NSight Systems | CUPTI

COMMUNICATION LIBRARIES

MPI | **HPC-X**: OpenSHMEM, UCX, SHARP, HCOLL | NCCL | NVSHMEM

MATH LIBRARIES

cuBLAS | cuTENSOR | cuSOLVER | cuSPARSE

cuFFT | cuRAND | MATH API | OpenBLAS

CUDA Runtime

RAPIDS

cuDF

cuxfilter

cuSpatial

cuSignal

cuGRAPH

cuML

DL INFERENCE

Triton Inference Server

TensorRT

TRAINING (DLFW)

TensorFlow 1/2 | DIGITS | NVCaffe

Pytorch | MxNet | Kaldi

DL LIBRARIES AND SOFTWARE

DALI | CASK | cuDNN

Myelin | ONNX | DLA

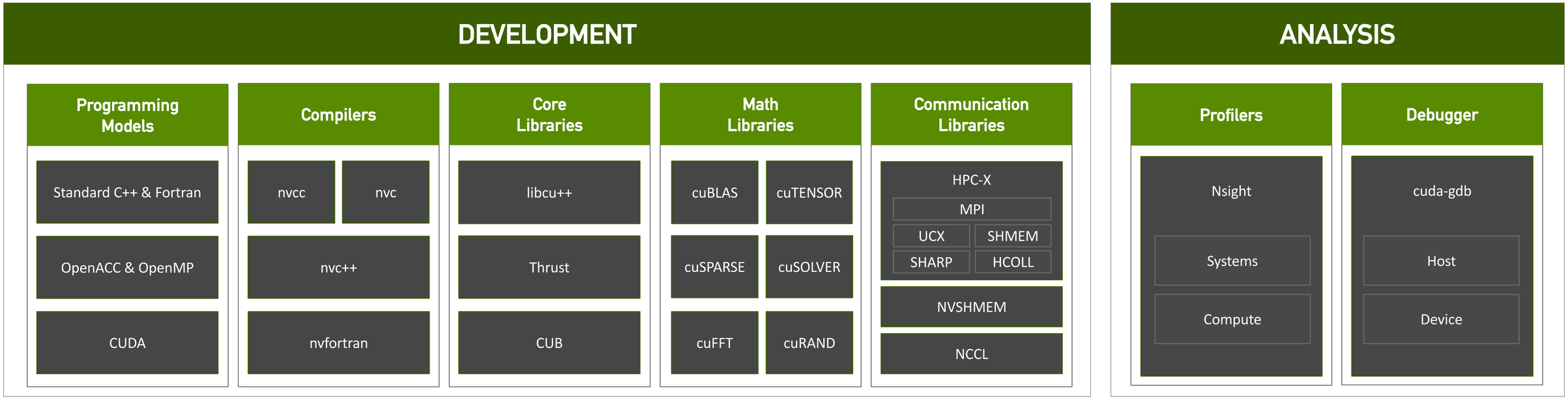
Platform Software

Fabric Manager | GPUDirect | NVML | nvidia-smi | NVIDIA Linux GPU Driver for ARM | DCGM | Slurm | Docker | Singularity

7-8 Releases Per Year | Freely Available

NVIDIA

NVIDIA HPC SDK

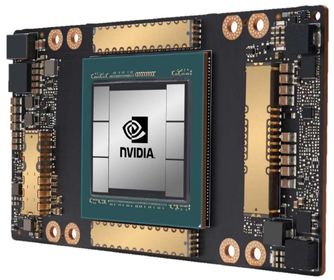


Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
x86_64 | Arm | OpenPOWER
7-8 Releases Per Year | Freely Available

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

HPC Compilers

NVC | NVC++ | NVFORTRAN



Accelerated

H100
Automatic

Fortran

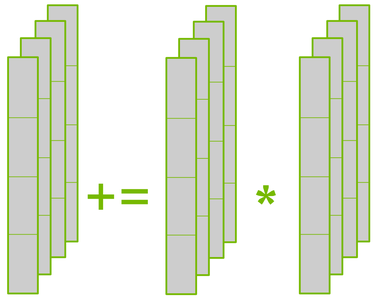
NVIDIA
CUDA

OpenACC
More Science, Less Programming

OpenMP

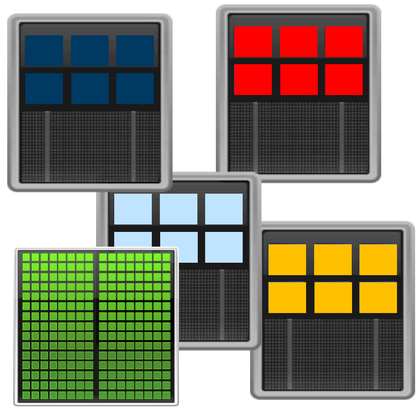
Programmable

Standard Languages
Directives
CUDA



CPU Optimized

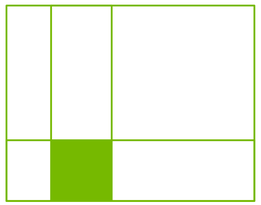
Directives
Vectorization



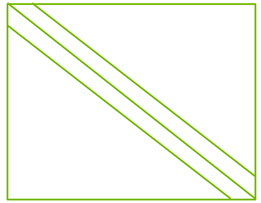
Multi-Platform

x86_64
AArch64
OpenPOWER

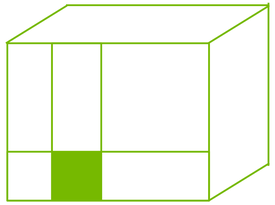
GPU accelerated Math libraries



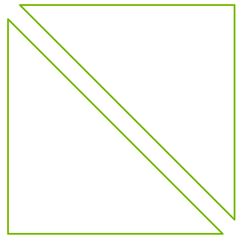
cuBLAS



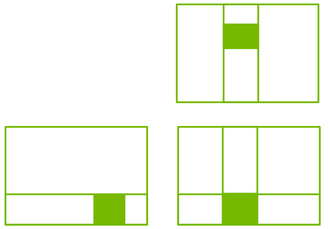
cuSPARSE



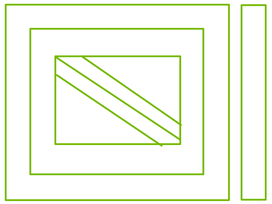
cuTENSOR



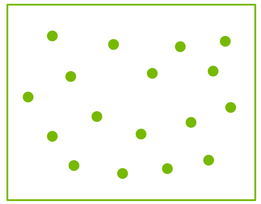
cuSOLVER



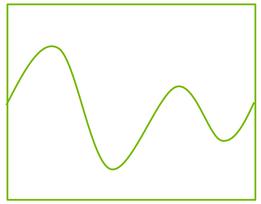
CUTLASS



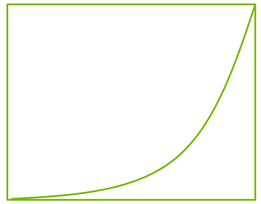
AMGX



cuRAND



cuFFT



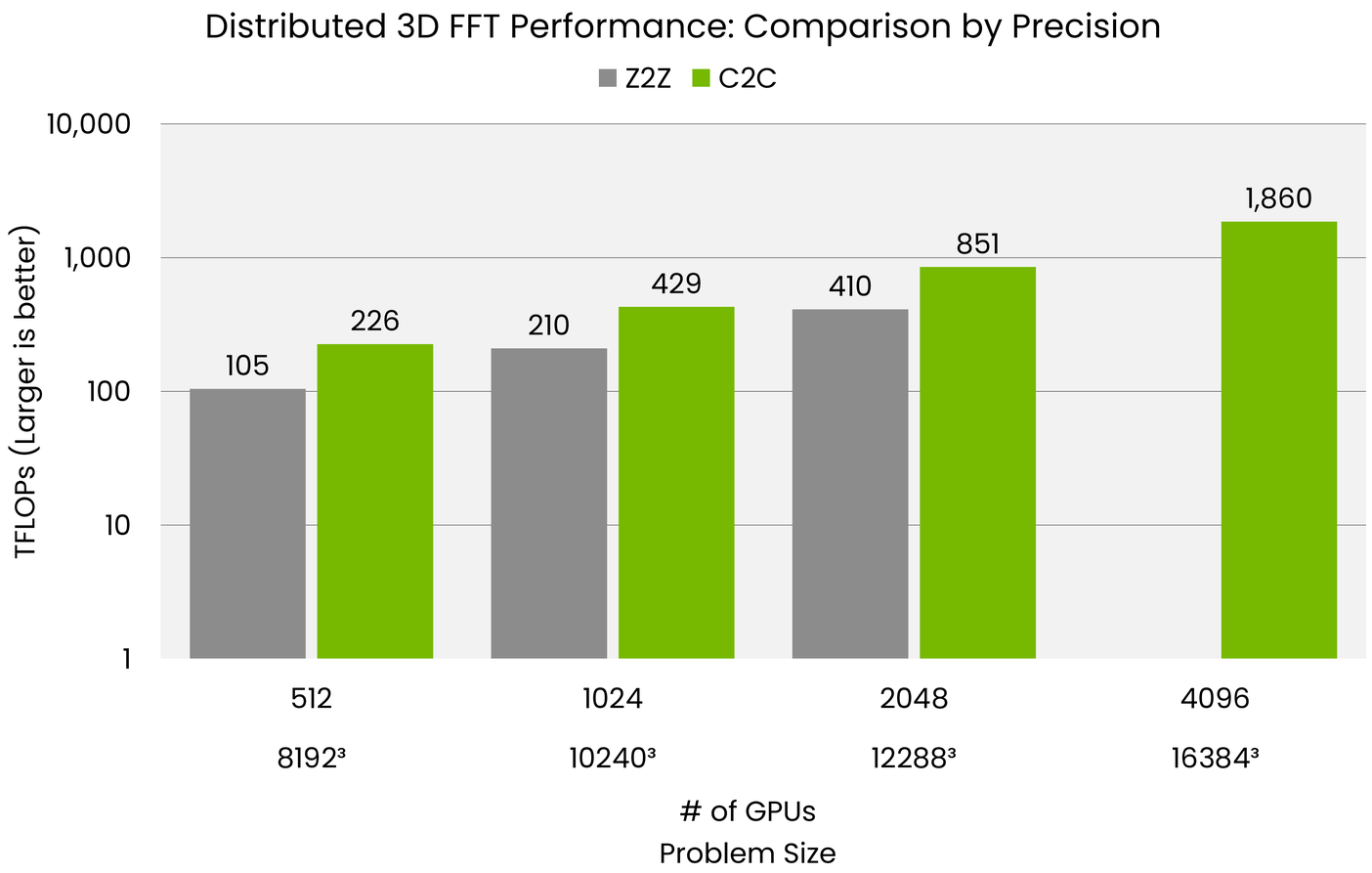
CUDA Math API

Single-Process / Multi-GPU Multi-Process / Multi-GPU Batched APIs

Support to Mixed Precisions Support to TensorCore

Multinode FFTs

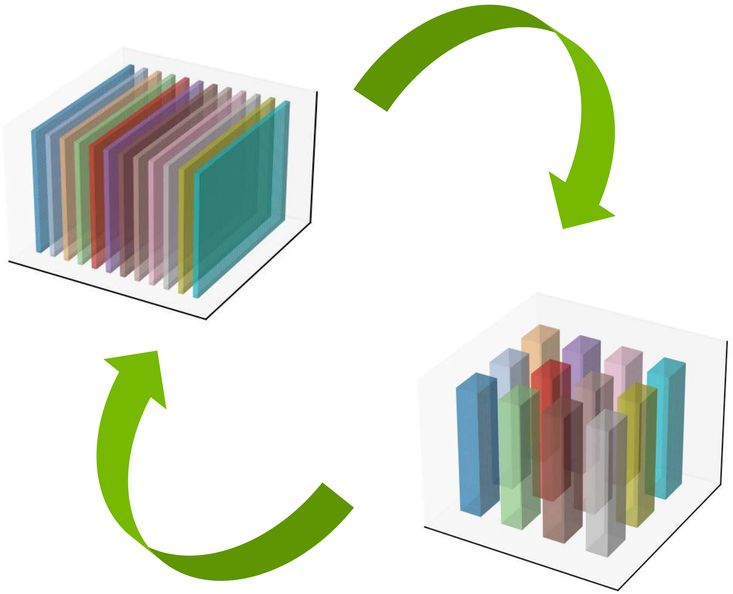
Distributed 2D & 3D FFTs at scale



* Selene: A100 80GB @ 1410 MHz

Released in HPC SDK

- Distributed 2D/3D FFTs
- Slab Decomposition
- Pencil Decomposition (Preview)
- Helper functions: Pencils <-> Slabs



Programming the NVIDIA platform

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran [38]

```
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

The Role of Directives for Parallel Programming

Directives convey additional information to the compiler.

Accelerated Standard Languages

Parallel performance for wherever your code runs

ISO C++

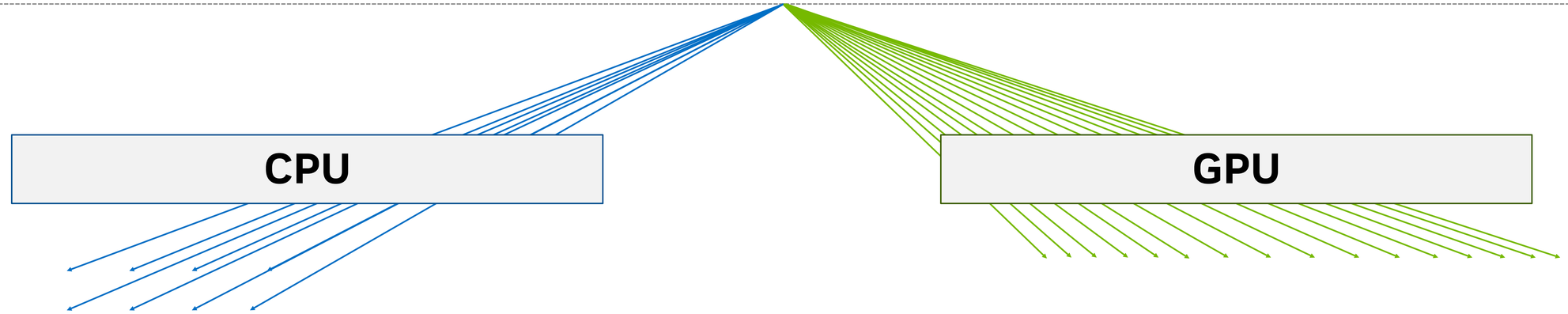
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

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
```

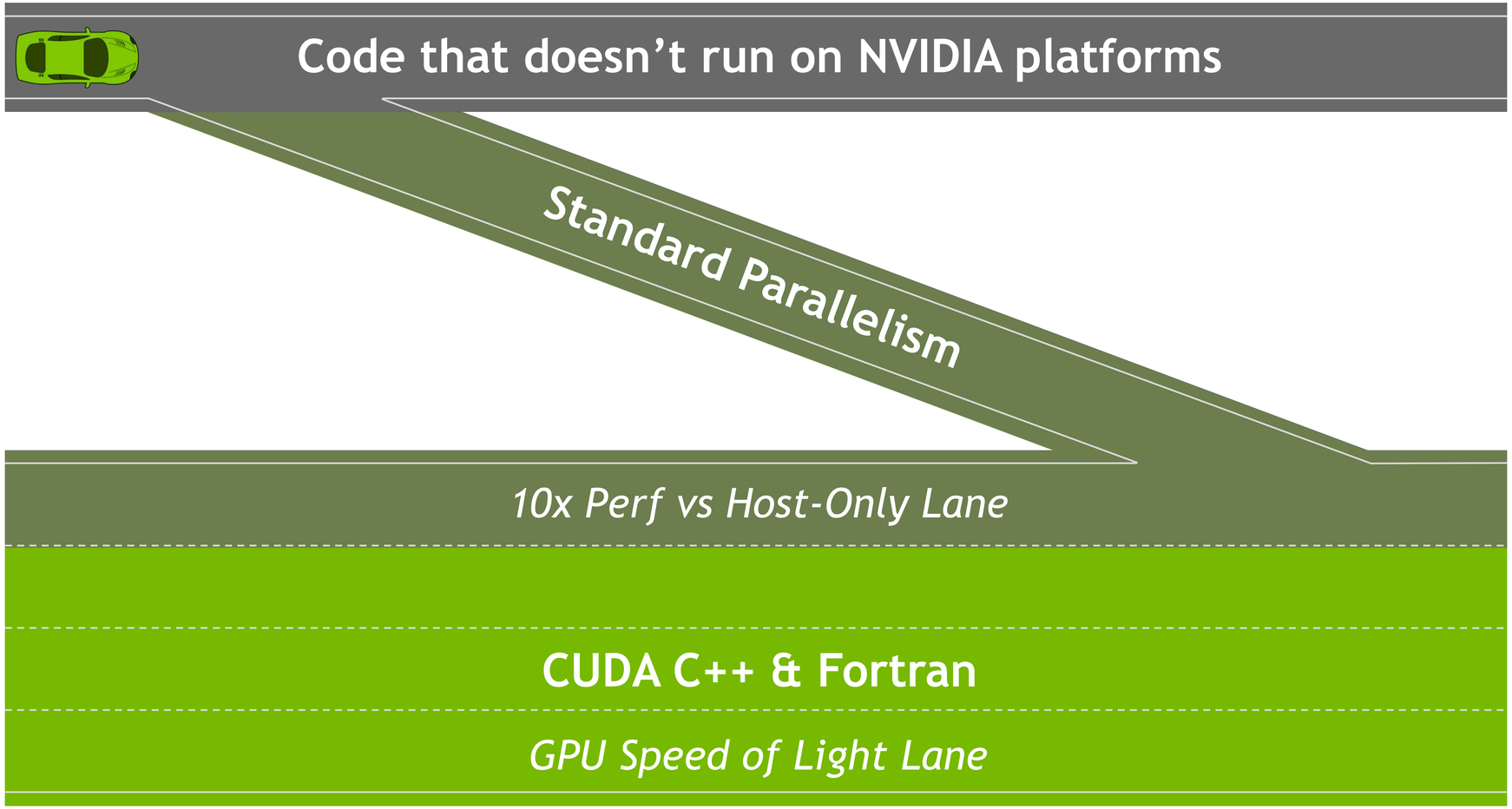



```
nvc++ -stdpar=multicore
nvfortran –stdpar=multicore
legate –cpus 16 saxpy.py
```

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

Scientists Need On-Ramps

Promote Parallelism, not Heterogeneity



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/mdarray`

- 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

Std::Execution

- Simplify launching and managing parallel work across CPUs and accelerators
- Preview Available Now

Linear Algebra

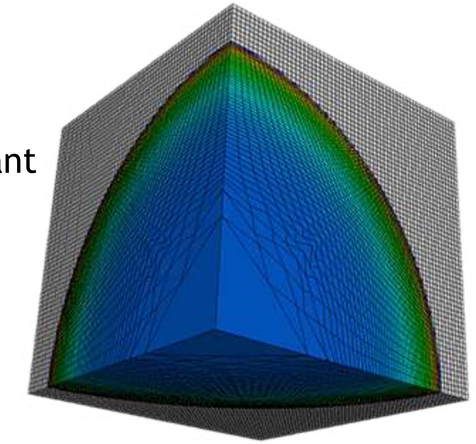
- C++ standard algorithms API to linear algebra
- Maps to vendor optimized BLAS libraries
- Preview Available Now

LULESH WITH STANDARD C++

C++ Hydrodynamics Mini-app from LLNL

Rewritten from OpenMP to ISO C++

- More composable, compact, and elegant
- Easier 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)
{
#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

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

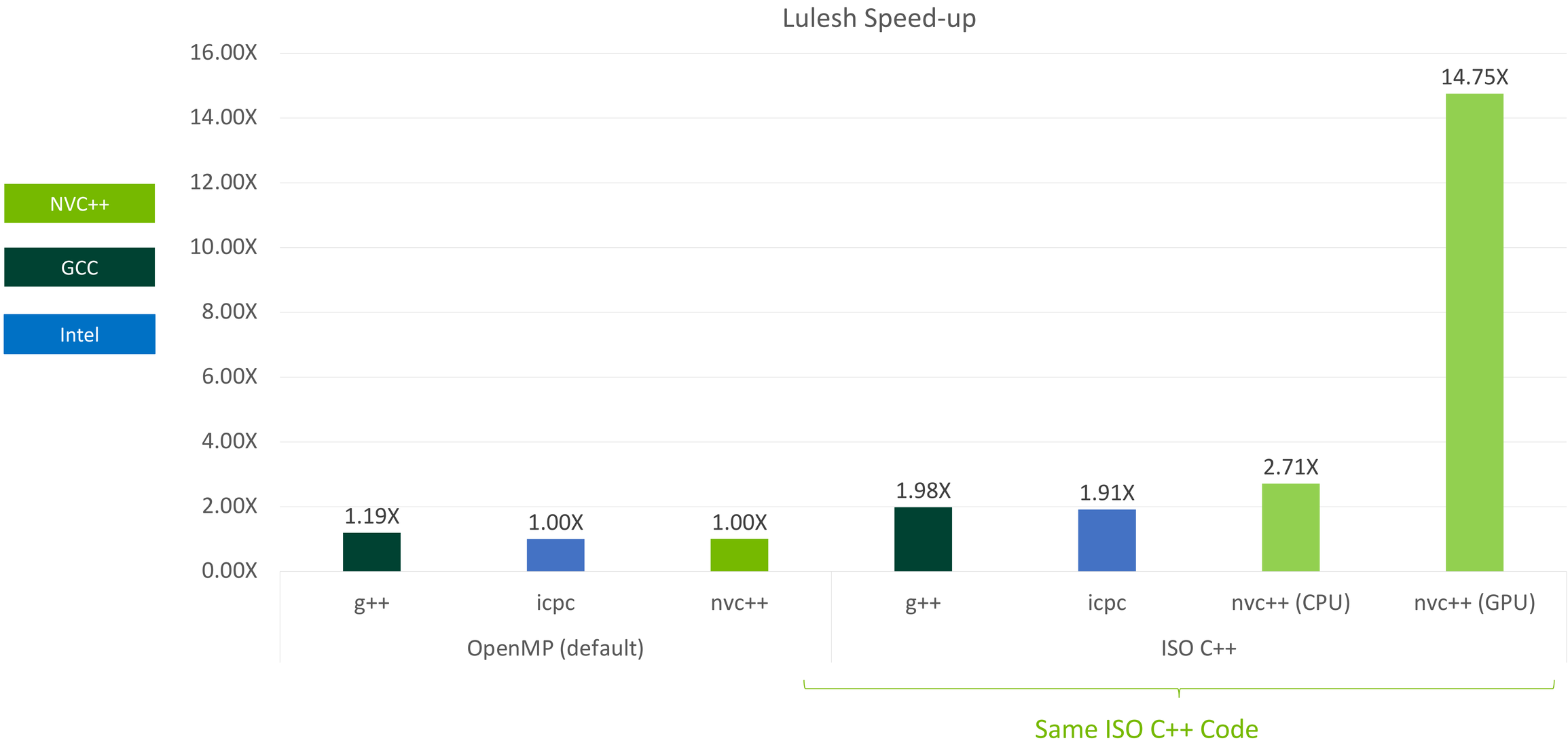
Standard C++

codesign.llnl.gov/lulesh

NVIDIA

C++ STANDARD PARALLELISM

Lulesh Performance

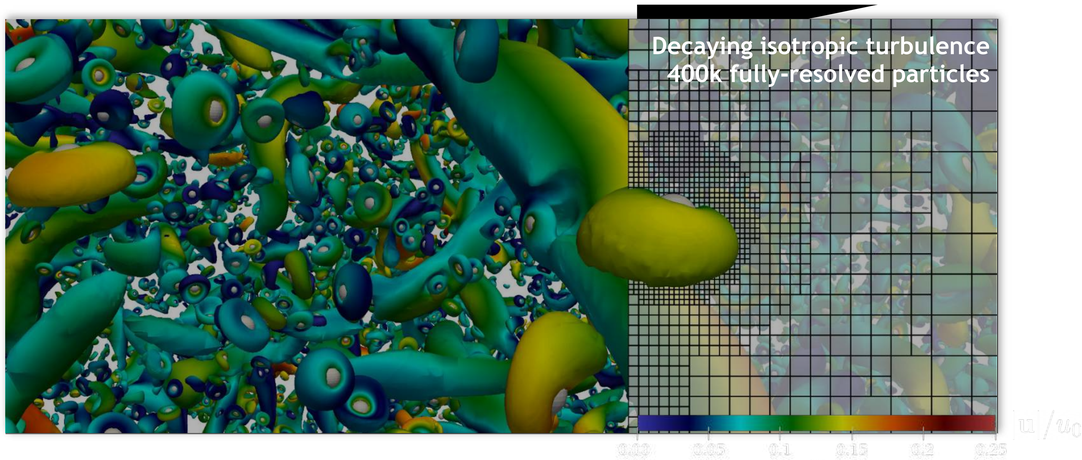


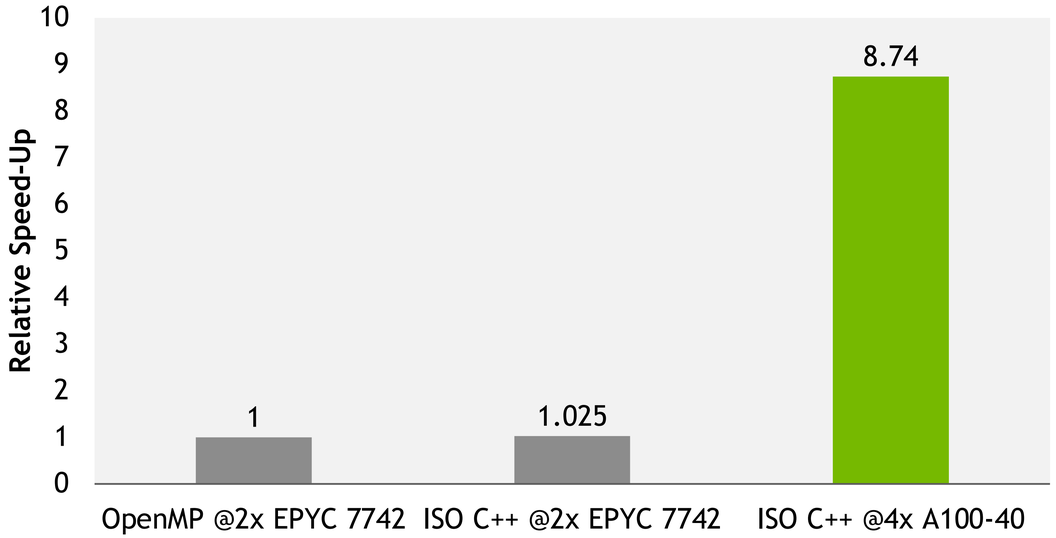
AMD EPYC 7742 CPU, NVIDIA A100 GPU. g++ version 10.3.0, icpc version 2021.5.0, nvc++ version 22.3

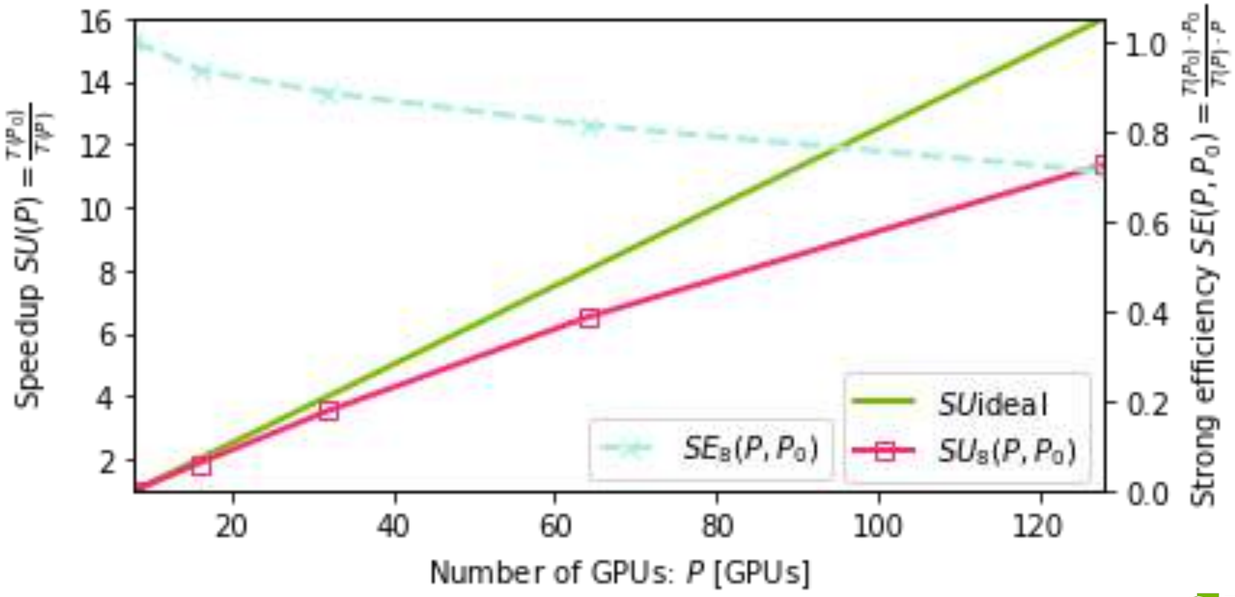
M-AIA

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**





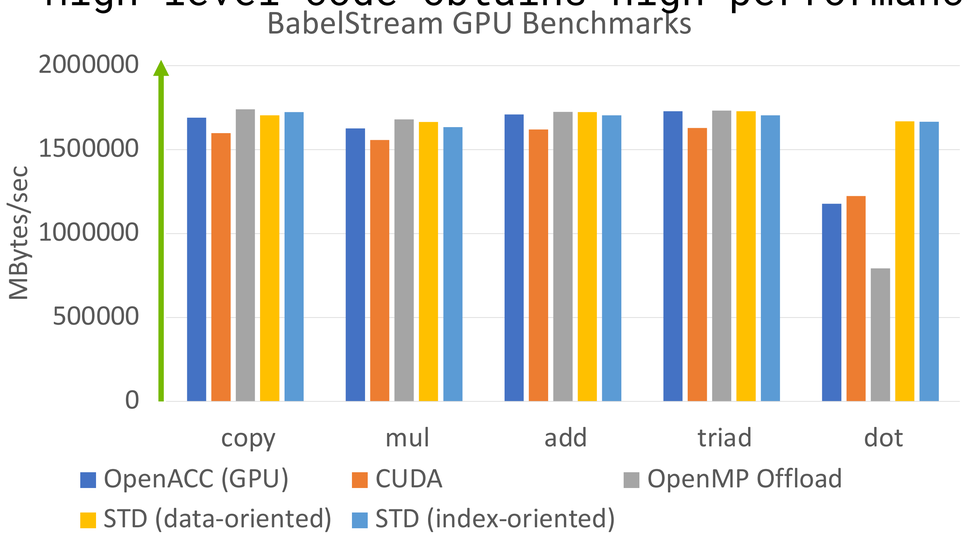


NVIDIA.

BabelStream with ISO C++

ISO C++ for Memory-Bound Applications

- Stream is a widely-accepted high-water for bandwidth-bound kernels
- Standard Parallel Algorithms are the building-blocks
- High-level code obtains high-performance



* Results from NVIDIA A100-80G

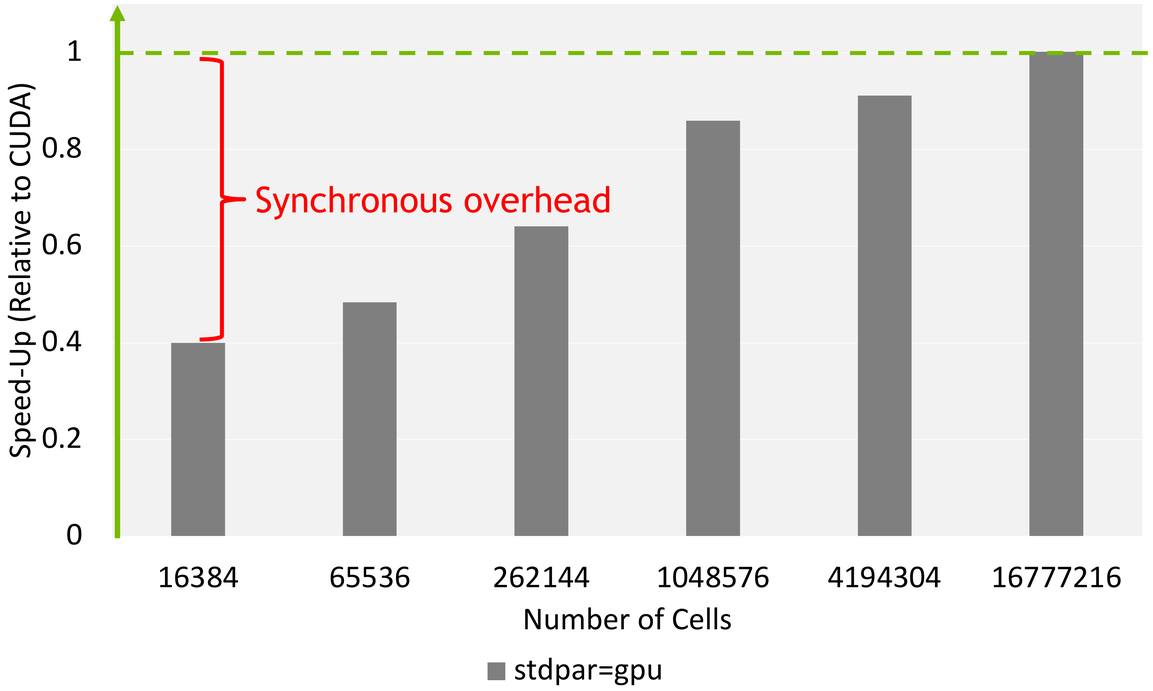
```
std::vector<T> a, b, c;

// COPY - c[i] = a[i]
std::copy(exe_policy, a.begin(), a.end(), c.begin());

// MUL - b[i] = scalar * c[i];
std::transform(exe_policy, range.begin(), range.end(), b.begin(),
  [&, scalar = startScalar](int i) {
  return scalar * c[i];
});

//  ADD - c[i] = a[i] + b[i];
std::transform(exe_policy, range.begin(), range.end(), c.begin(),
  [&](int i) {
  return a[i] + b[i];
});

// TRIAD - a[i] = b[i] + scalar * c[i];
std::transform(exe_policy, range.begin(), range.end(), a.begin(),
  [&, scalar = startScalar](int i) {
  return b[i] + scalar * c[i];
});

// DOT - sum = 0; sum += a[i]*b[i];
std::transform_reduce(exe_policy, a.begin(), a.end(), b.begin(), 0.0);
```


Work by Simon McIntosh-Smith, Tom Deakin, and Tom Lin from University of Bristol. https://github.com/UoB-HPC/BabelStream

Standard Parallelism

The Cost of Synchrony

- Current C++ Parallel Algorithms are synchronous, incurring extra overheads when launched on a GPU



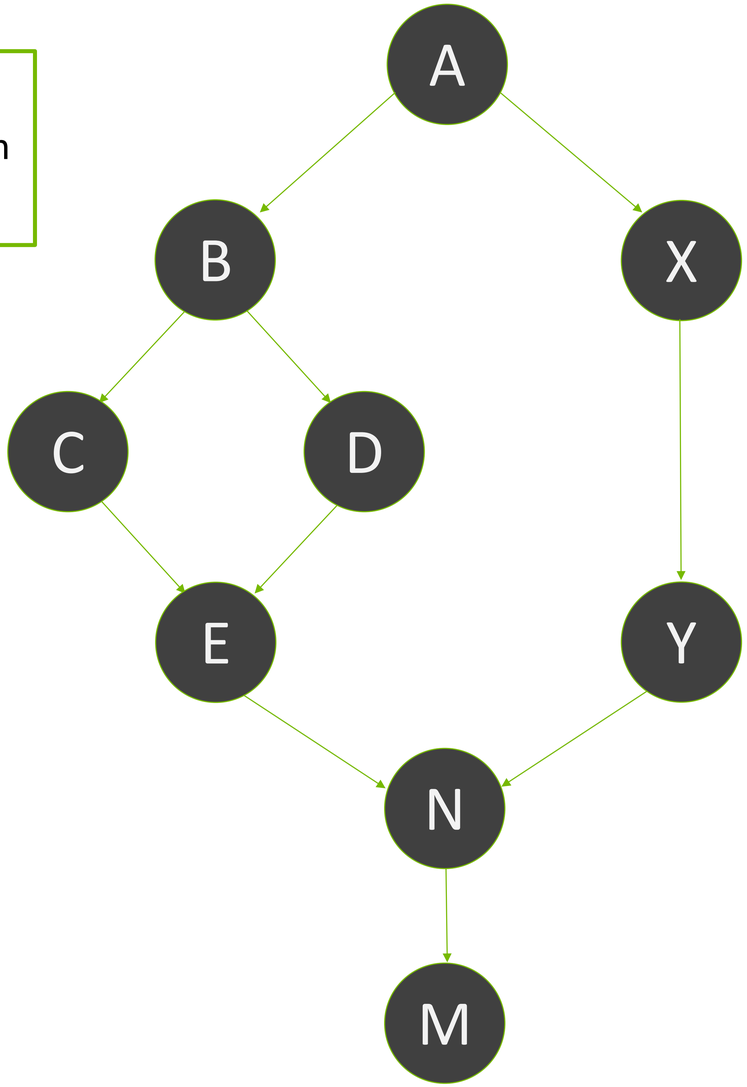

```
// C++17 Parallel Algorithms are Synchronous
for (int step = 0; step < n_steps; step++) {
  std::for_each(par_unseq, cells_begin, cells_end, update_h);
  std::for_each(par_unseq, cells_begin, cells_end, update_e);
}

// ------------------------------------------------------

// CUDA enables asynchronously enqueuing kernels
for (int step = 0; step < n_steps; step++) {
  kernel<<<blocks, threads, 0, stream>>>(n_cells, update_h);
  kernel<<<blocks, threads, 0, stream>>>(n_cells, update_e);
}
cudaStreamSynchronize(stream);
```

STD::EXECUTION

Lazily build large graphs of computation

```
sender auto async_graph(sender auto s) {
    auto A = s | std::then(printer{'A’}) | std::split();
    auto B = A | std::then(printer{'B'}) | std::split();
    auto C = B | std::then(printer{'C'});
    auto D = B | std::then(printer{'D'});
    auto E = std::when_all(C, D) | std::then(printer{'E'});
    auto X = A | std::then(printer{'X'})
               | std::then(printer{'Y'});
    return std::when_all(E, X);
```

Express a device-agnostic execution graph.




```
cuda::execution::scheduler gpu_scheduler;
sender auto work = std::schedule(gpu_scheduler) | async_graph | async_algo;
auto [r] = std::this_thread::sync_wait(work).value();
assert(r == 55);
```

Apply a specific scheduler to execute that graph.

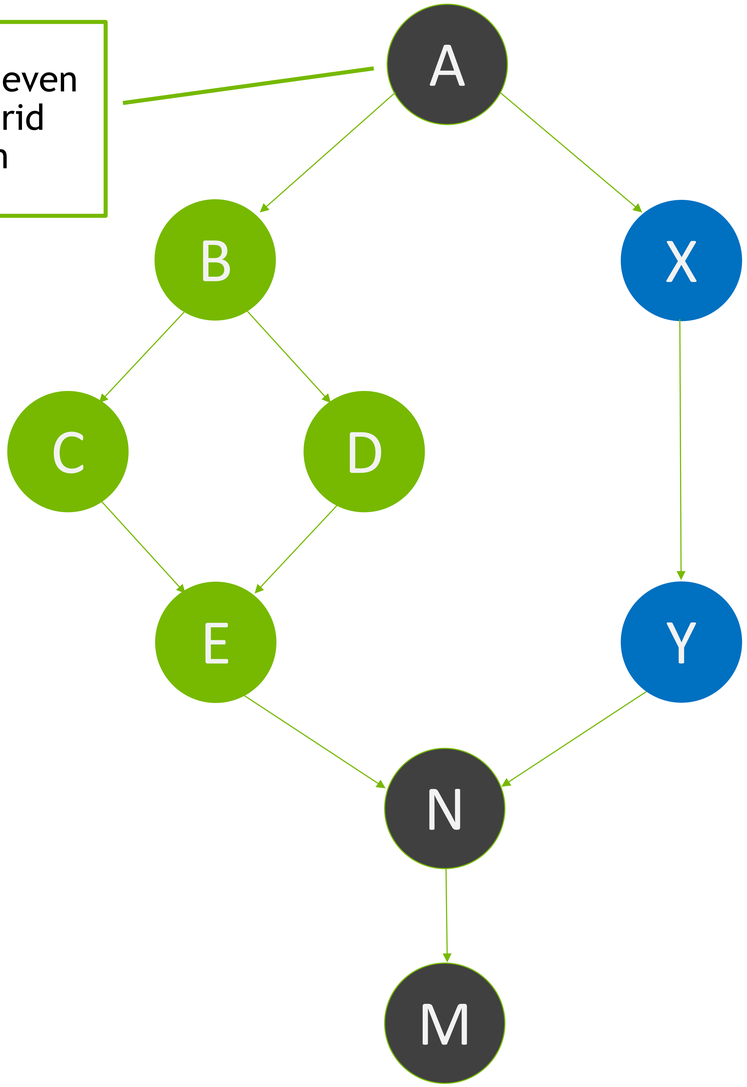
Wait for the graph to complete.

STD::EXECUTION

Lazily build large graphs of computation

```
sender auto async_graph(sender auto s) {
    auto A = s | std::then(printer{'A’}) | std::split();
    auto B = A | std::then(printer{'B'}) | std::split();
    auto C = B | std::then(printer{'C'});
    auto D = B | std::then(printer{'D'});
    auto E = std::when_all(C, D) | std::then(printer{'E'});
    auto X = A | std::then(printer{'X'})
               | std::then(printer{'Y'});
    return std::when_all(E, X);
```

Graphs could even include hybrid execution



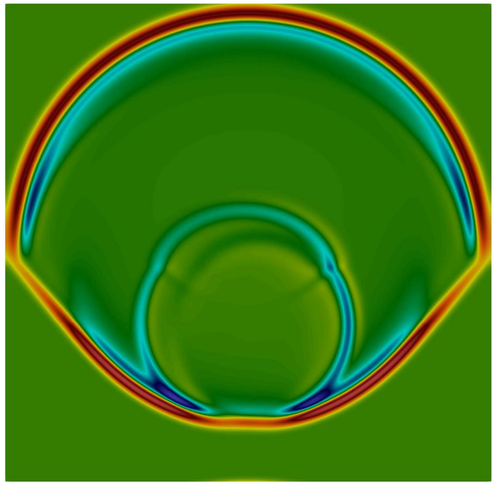
Instantiate a custom scheduler.

```
my_hybrid_scheduler hybrid_scheduler;
sender auto work = std::schedule(hybrid_scheduler) | async_graph | async_algo;
auto [r] = std::this_thread::sync_wait(work).value();
assert(r == 55);
```

Maxwell's Equations with Senders

Electromagnetism Solver

- Simplify Work Across CPUs and Accelerators
- Standardized way to express asynchrony and concurrency in C++
- Uniform abstraction between code and diverse resources
- Write once, run everywhere

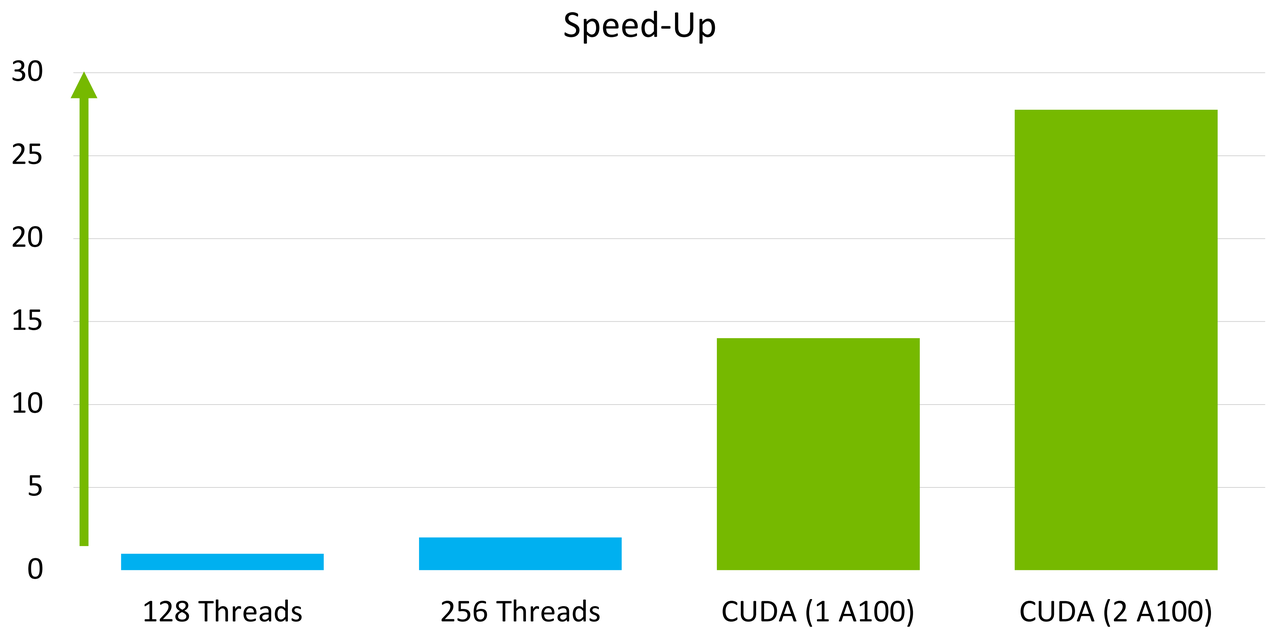


```
auto maxwell(auto &&scheduler, auto &&problem) {
  return ex::just()
    | exec::on(scheduler,
      repeat_n(problem.n_iterations,
        ex::bulk(data.n_cells, update_h(problem))
        | ex::bulk(data.n_cells, update_e(problem))))
    | ex::then(write_vtk(problem));
}
```

ELECTROMAGNETISM

Portable Algorithm, Performant Specialization

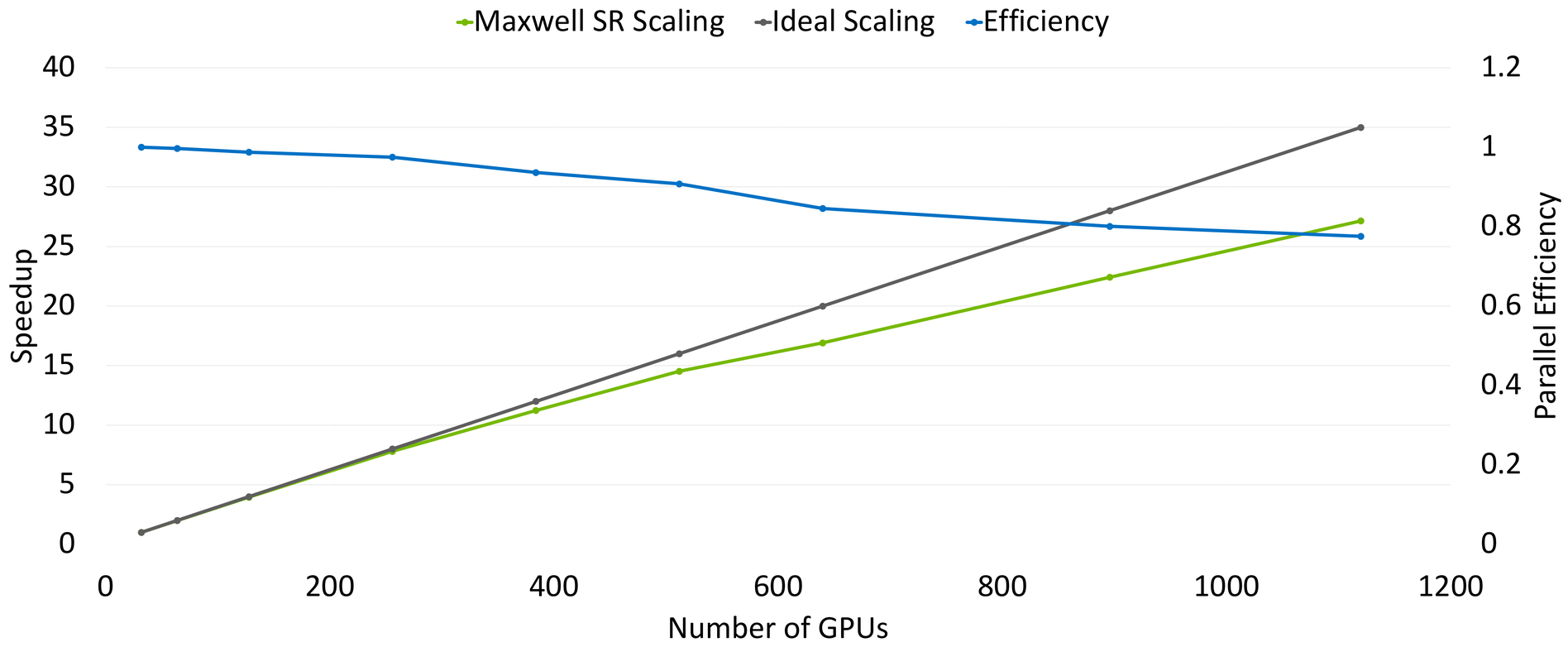
```
stdexec::sync_wait(maxwell(inline_scheduler, problem));
stdexec::sync_wait(maxwell(pool_scheduler,   problem));
stdexec::sync_wait(maxwell(gpu_scheduler,    problem));
```



- CPUs: AMD EPYC 7742 CPUs, GPUs: NVIDIA A100-SXM4-80
- OpenMP-128 (1x CPU), OpenMP-256 (2x CPUs), Graph (1x GPU), Multi-GPU (2x GPUs)

STRONG SCALING USING ISO STANDARD C++

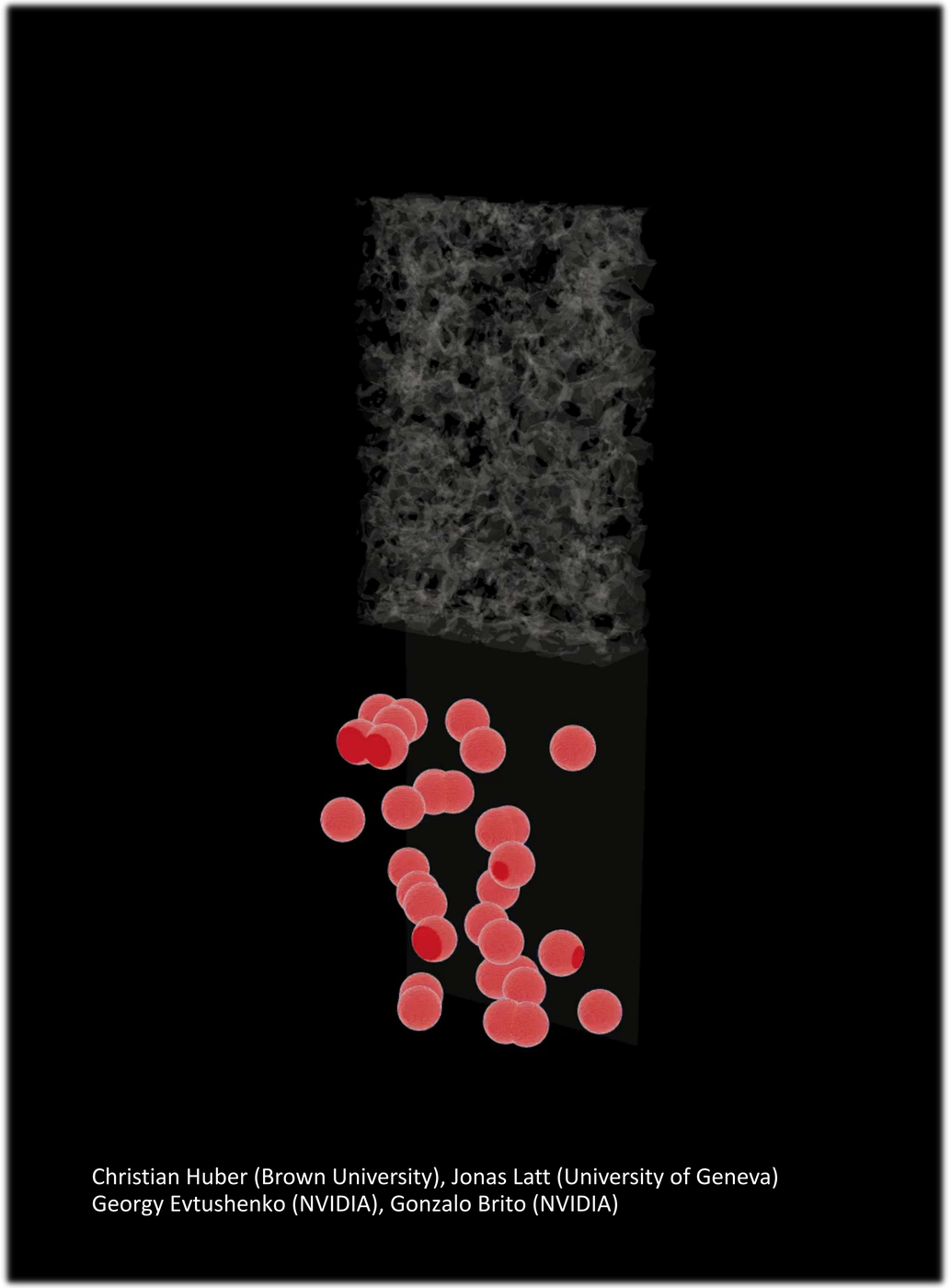
PARALLEL ALGORITHMS AND STD::EXECUTION



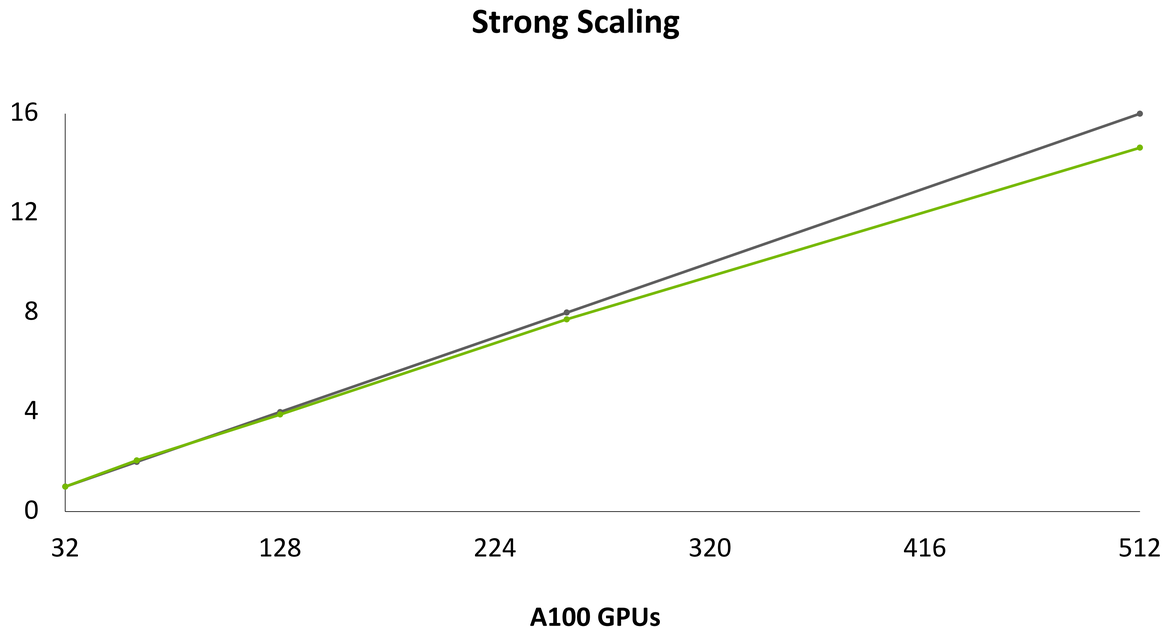
NVIDIA SUPERPOD

- 140x NVIDIA DGX-A100 640
- 1120x NVIDIA A100-SXM4-80 GPUs

PALABOS CARBON SEQUESTRATION



Christian Huber (Brown University), Jonas Latt (University of Geneva)
Georgy Evtushenko (NVIDIA), Gonzalo Brito (NVIDIA)



- Palabos is a framework for fluid dynamics simulations using Lattice-Boltzmann methods.
- Code for multi-component flow through a porous media ported to C++ Std::Execution.
- Application: simulating carbon sequestration in sandstone.

HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Fortran Array Intrinsics

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

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

- Not currently available
- Accelerated co-array images

Preview support available now in NVFORTRAN

Fortran 2023

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

ACCELERATED PROGRAMMING IN ISO FORTRAN

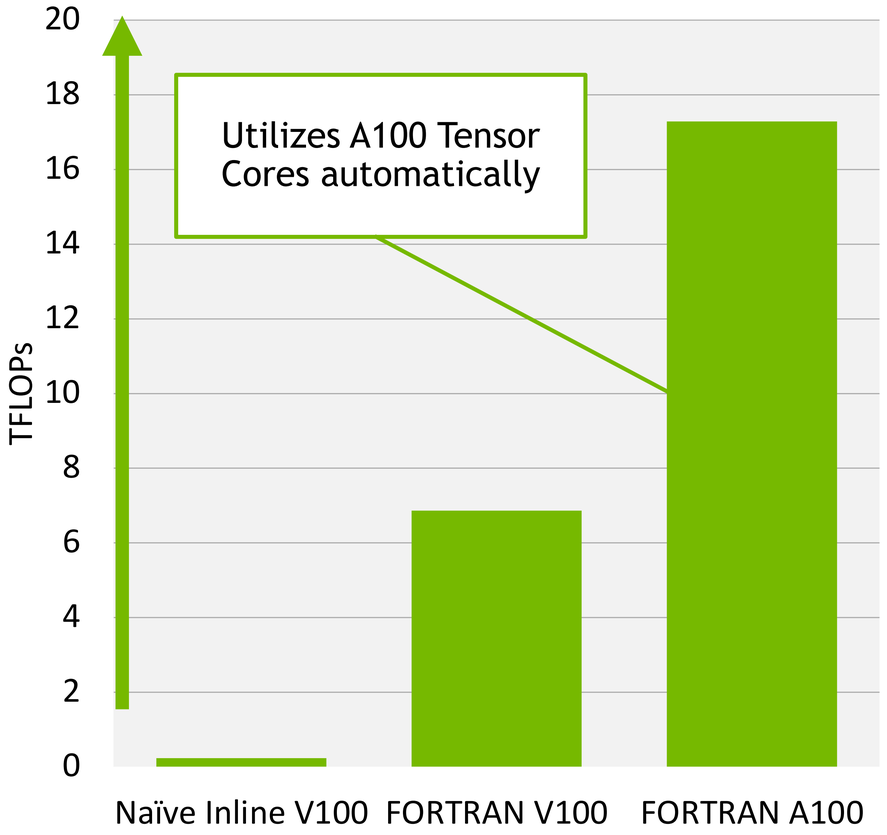
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
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



MINIWEATHER

Standard Language Parallelism in Climate/Weather Applications

MiniWeather

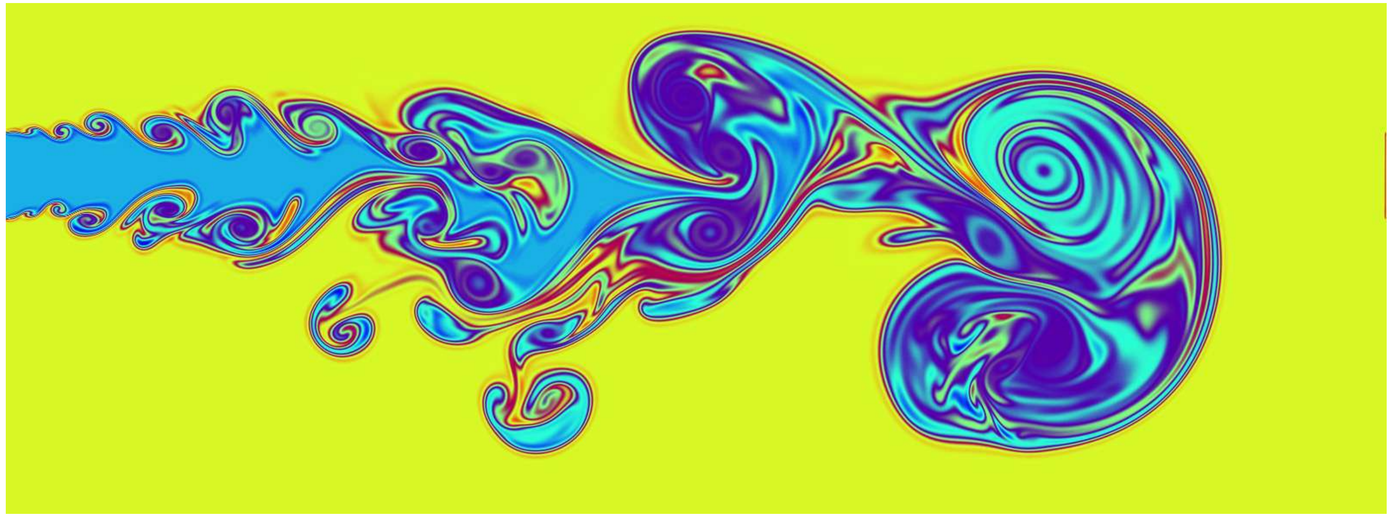
Mini-App written in C++ and Fortran that simulates weather-like fluid flows using Finite Volume and Runge-Kutta methods.

Existing parallelization in MPI, OpenMP, OpenACC, …

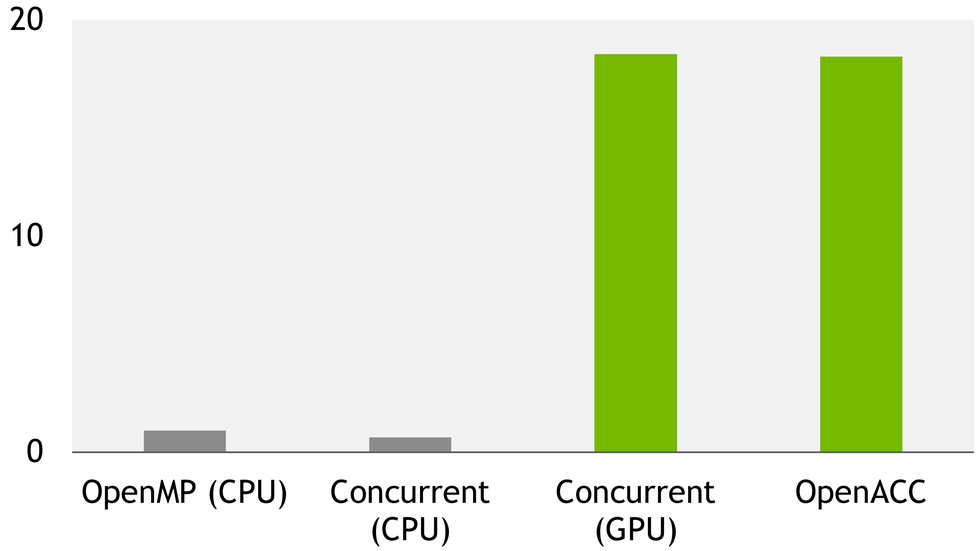
Included in the SPEChpc benchmark suite*

Open-source and commonly-used in training events.

https://github.com/mrnorman/miniWeather/



```
do concurrent (ll=1:NUM_VARS, k=1:nz, i=1:nx)
      local(x,z,x0,z0,xrad,zrad,amp,dist,wpert)

  if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
    x = (i_beg-1 + i-0.5_rp) * dx
    z = (k_beg-1 + k-0.5_rp) * dz
        x0 = xlen/8
    z0 = 1000
    xrad = 500
    zrad = 500
    amp = 0.01_rp
    dist = sqrt( ((x-x0)/xrad)**2 + ((z-z0)/zrad)**2 )
         * pi / 2._rp
    if (dist <= pi / 2._rp) then
      wpert = amp * cos(dist)**2
    else
      wpert = 0._rp
    endif
    tend(i,k,ID_WMOM) = tend(i,k,ID_WMOM)
                      + wpert*hy_dens_cell(k)
  endif
  state_out(i,k,ll) = state_init(i,k,ll)
                    + dt * tend(i,k,ll)

enddo
```



Source: HPC SDK 22.1, AMD EPYC 7742, NVIDIA A100. MiniWeather: NX=2000, NZ=1000, SIM_TIME=5. OpenACC version uses -gpu=managed option.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

POT3D: DO CONCURRENT + LIMITED OPENACC

POT3D

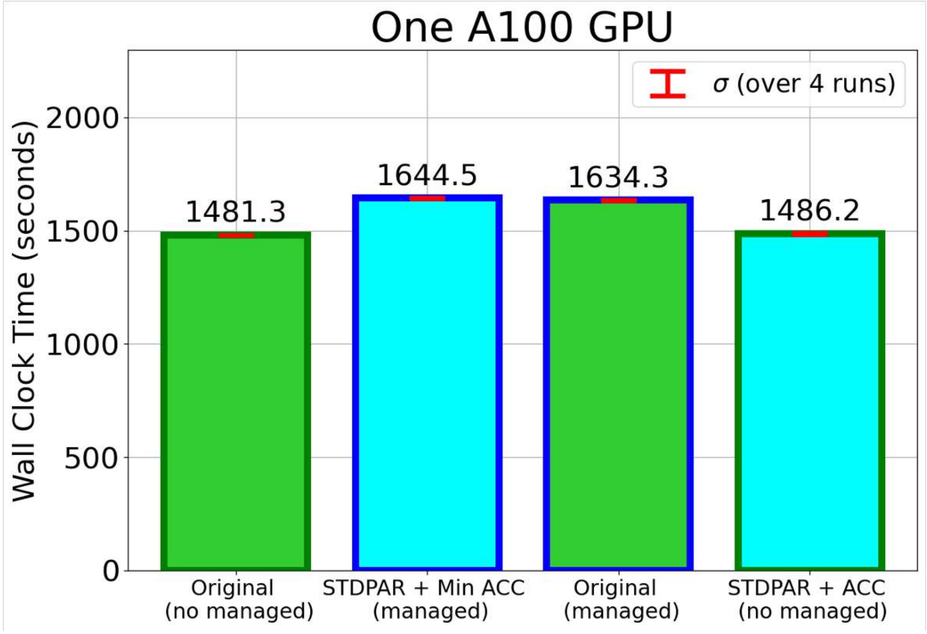
POT3D is a Fortran application for approximating solar coronal magnetic fields.

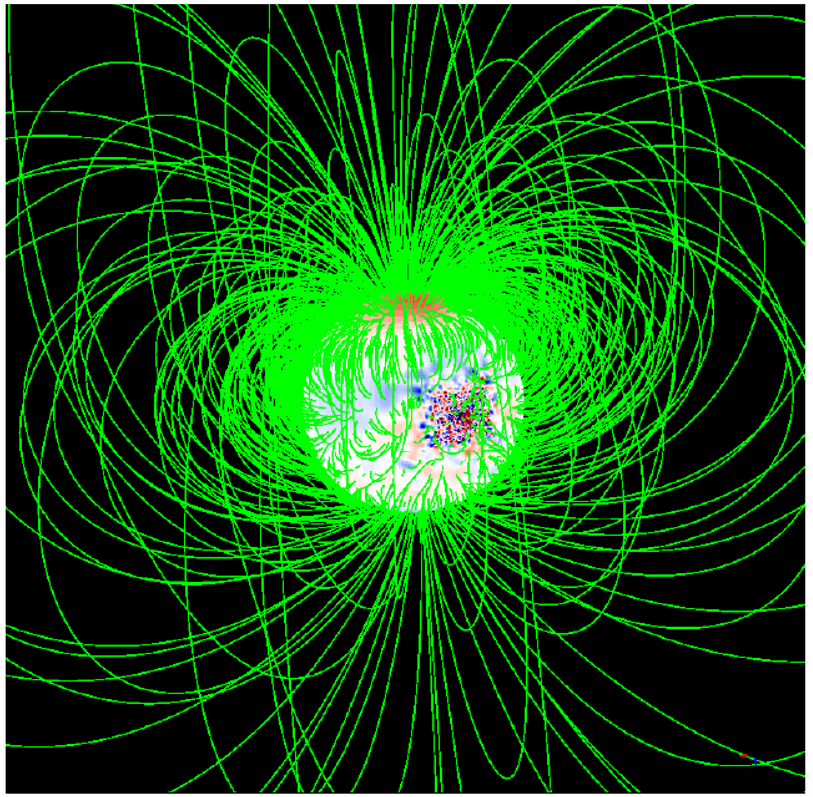
Included in the SPEChpc benchmark suite*

Existing parallelization in MPI & OpenACC

Optimized the DO CONCURRENT version by using OpenACC solely for data motion and atomics

https://github.com/predsci/POT3D





```
!$acc enter data copyin(phi,dr_i)
!$acc enter data create(br)
do concurrent (k=1:np,j=1:nt,i=1:nrm1)
  br(i,j,k)=(phi(i+1,j,k)-phi(i,j,k ))*dr_i(i)
enddo
!$acc exit data delete(phi,dr_i,br)
```

Data courtesy of Predictive Science Inc.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

NVIDIA

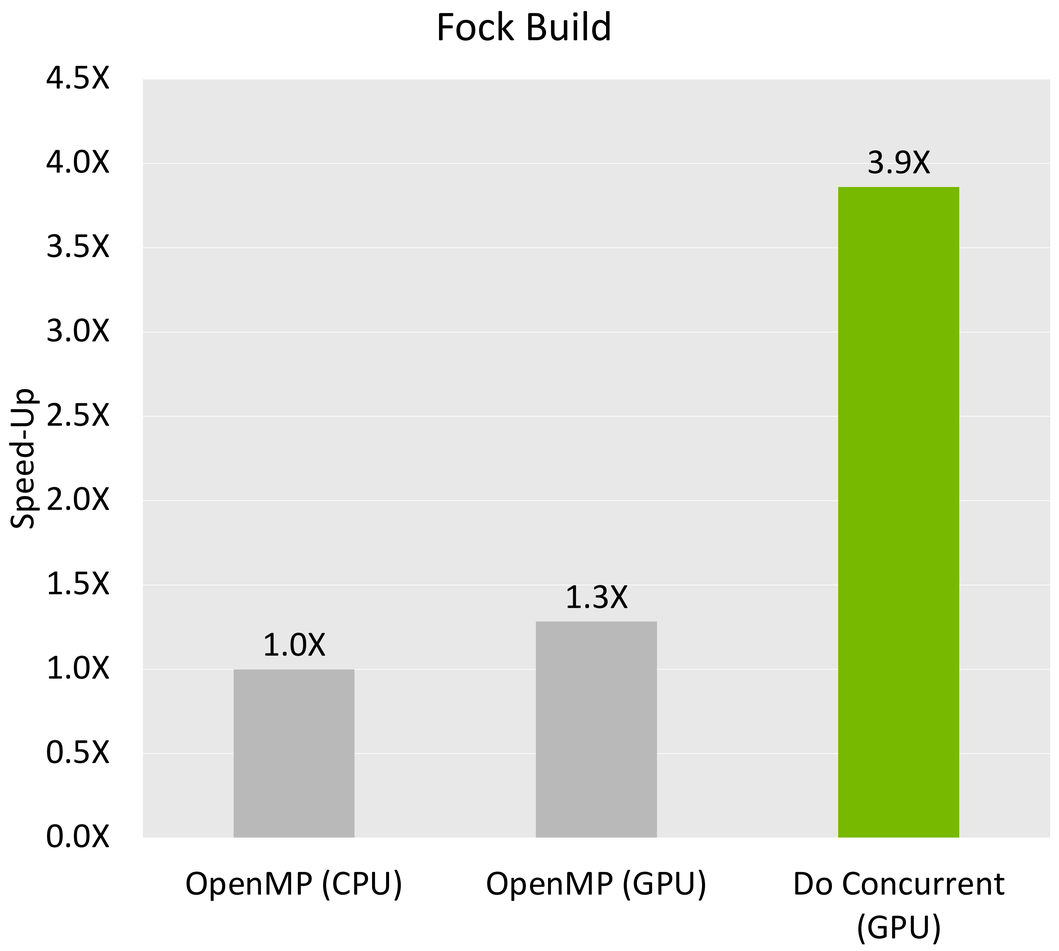
GAMESS

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
```



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

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

https://commons.wikimedia.org/wiki/File:Ferrari_F355_Spider_-_Flickr_-_The_Car_Spy_(16).jpg

What is the GPU Gearbox?

The GPU gearbox is a mental model for thinking about programming models, to deliver the best performance at different levels of developer effort and specialization.

Think about torque, not speed…

First Gear

ISO standard parallelism: Easiest to adopt. Maximum portability. Good performance in a subset of use cases.

Second Gear

Performance libraries: Peak performance for supported features, which include a wide range of common patterns in linear algebra, machine learning and data analysis.

Third Gear

Directives and Pragmas: Easy to adopt. Good portability. Great performance in many use cases.

Fourth Gear

CUDA languages: Exposes full hardware capability and enables maximum performance. Supported on all NVIDIA GPUs.

Interoperability by Design

From Zero to Hero without interrupting science

```
__global__ void  times_two(float* first, uint64_t n) {
             auto t= blockIdx.x * blockDim.x + threadIdx.x;
             if (t < n) first[t] *= 2;
}

void compute (std::vector<float>& x) {
             auto const b = ((x.size() – 1)/64 +1 );
             times_two<<<b, 64>>>(x.data(), x.size());
             cudaDeviceSyncronize();

             #pragma omp target teams loop
             for (auto& e : x) e *=e;

             #pragma acc parallel loop
             for (auto& e : x) e +=e;

             std::sort(std::execution::par,
                          x.begin(), x.end());
}
```

Full programming models **interoperability**

- All can be used in the same codebase (some caveats apply)
- ABI compatible with each other and with NVCC
- By leveraging Managed Memory, easy switch on/off selectively programming models

```
nvc++ -cuda –acc –mp –stdpar […]
```

Programming the NVIDIA Superchip

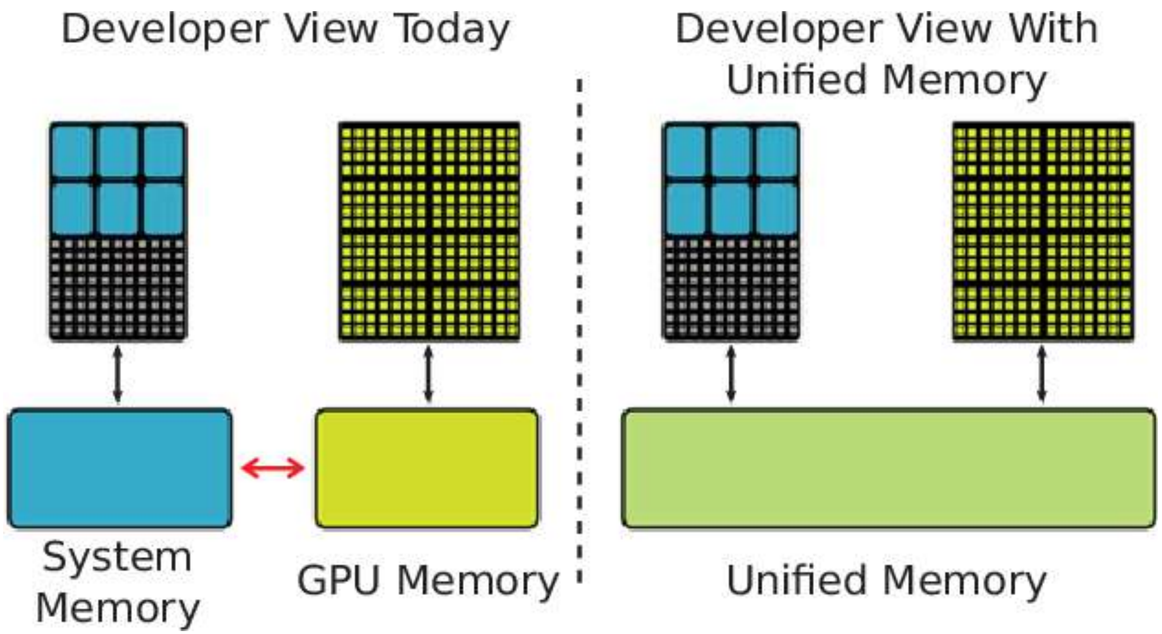
NVIDIA

Unified Memory

Dramatically Lowering Developer Effort

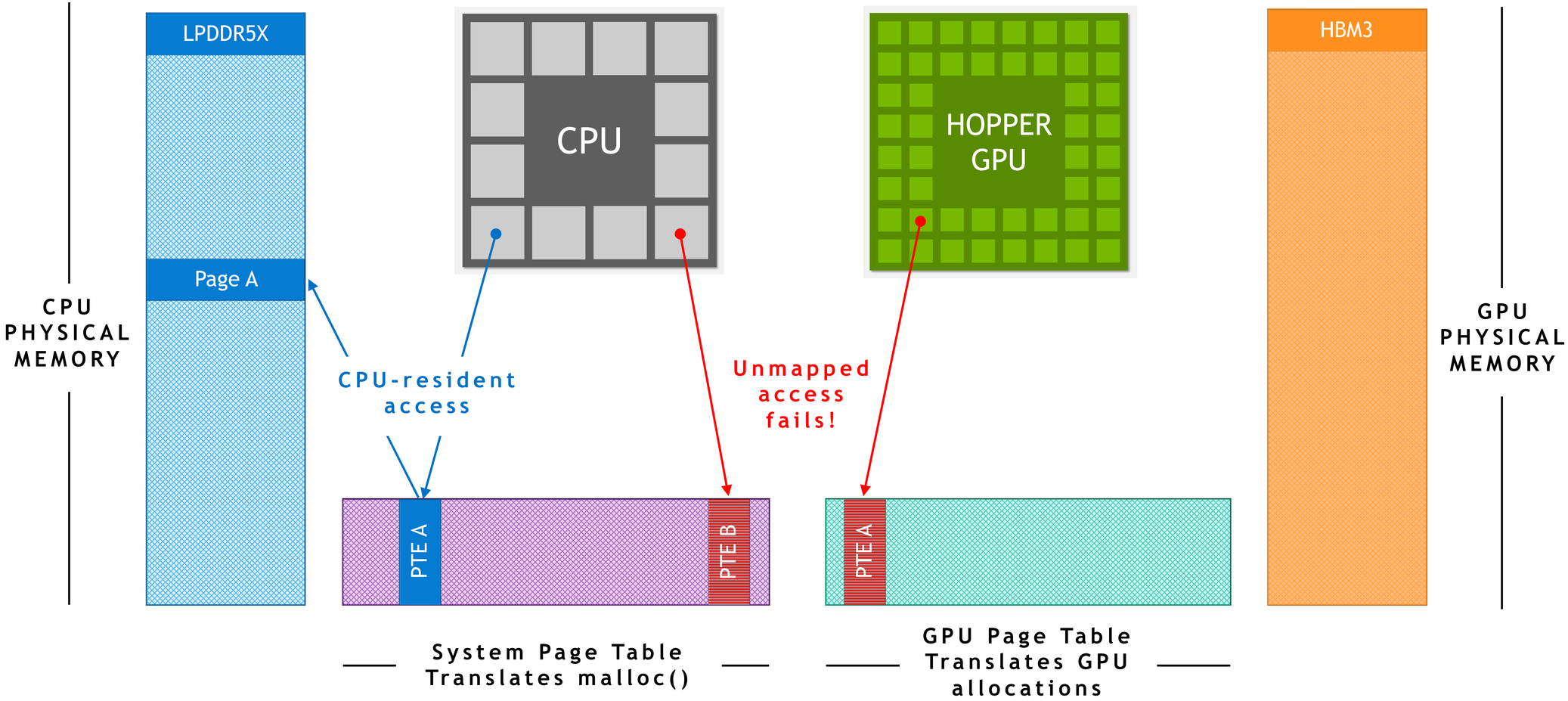
CUDA **Unified Memory** allows a program to dynamically allocate data that can be accessed from the CPU or from the GPU, with the same pointer and address

- Introduced with CUDA 6 / Kepler GK10x
- Leverages UVA (Unified Virtual Addressing) beyond GPU memory pools
- CUDA driver managed page migration CPU <--> GPU on-demand when data is requested (*first touch*)
- Can be controlled / tweaked with prefetching APIs and by define User Policy
  - ReadMostly
  - PreferredLocation
  - AccessedBy



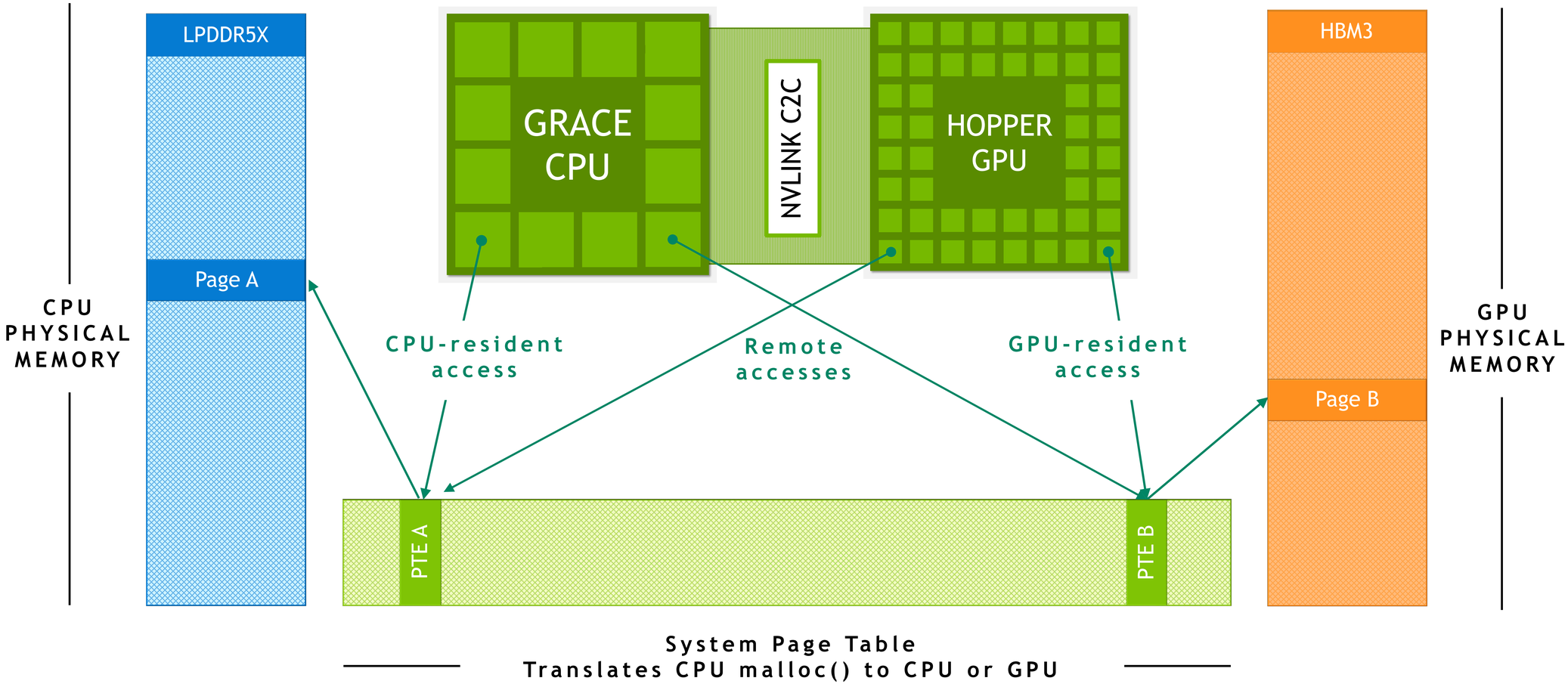
HW/SW memory view on X86 + GPU

Separate page tables



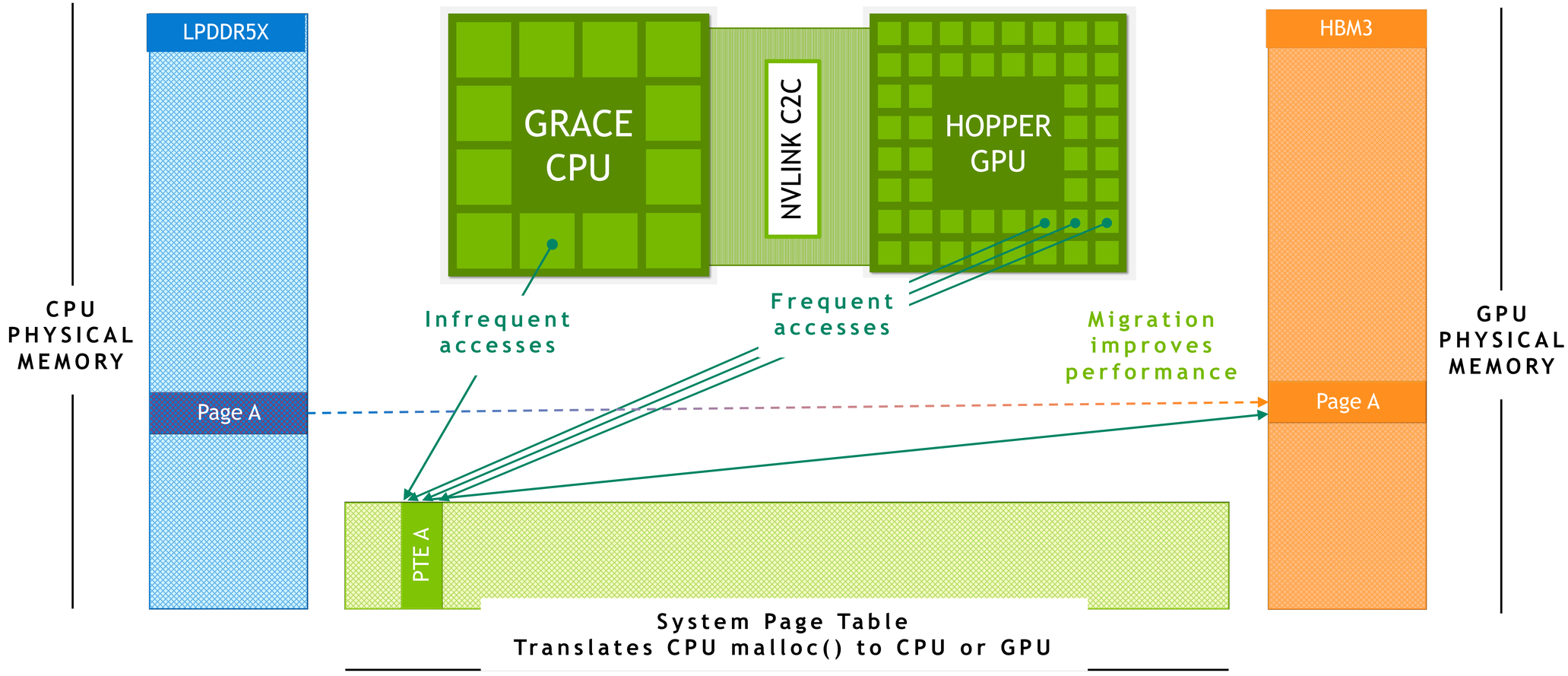
HW/SW memory view on Grace Hopper Superchip

Simplified Unified Memory via Address Translation Service (ATS)



HW/SW memory view on Grace Hopper Superchip

Automatic Migrations



Grace-Hopper Memory Model

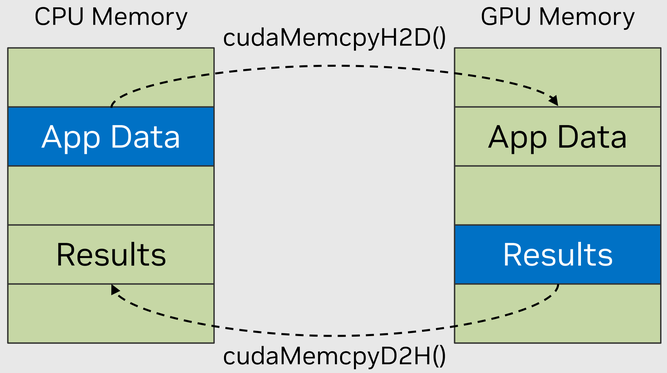
Full CUDA support with additional Grace memory extensions

Explicit Copy

Application explicitly moves data between CPU & GPU as needed

PCIE: ~60 GB/s PCIE transfers (H2D/D2H)

Grace: Faster transfers; up to 450 GB/s C2C transfers (per direction)

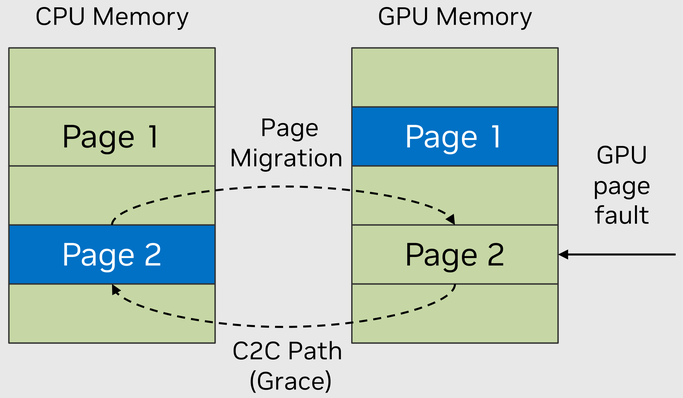


Managed Memory

CPU and GPU can access memory on-demand and data migrated locally for higher BW access

PCIE: Requires migration to GPU

Grace: Migrations not required and faster migrations when they happen

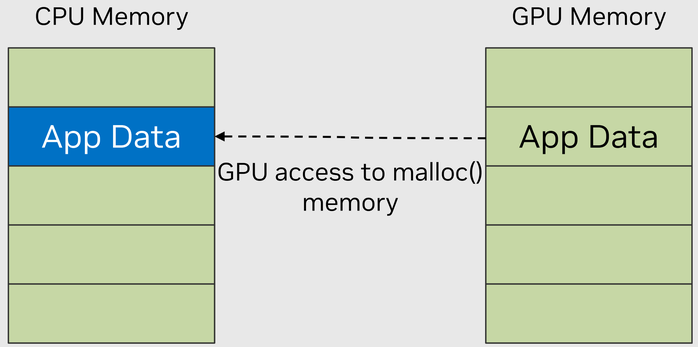


System Allocated

GPU can access memory allocated from `malloc()`, `mmap()`, etc.

PCIE: Access possible with explicit call to `cudaHostRegister()` at PCIe speeds

Grace: `cudaHostRegister()` not needed; access at NVLink C2C speeds



EXPLICIT DATA MOVEMENT

Works on both x86 and Grace Hopper

CUDA C

```
#include <stdio.h>
#include <stdlib.h>

__global__ void kernel(int *data)
{
    data[threadIdx.x] += 2;
}

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));
    int *d_data;

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    cudaMalloc((void **)&d_data, N * sizeof(int));
    cudaMemcpy(d_data, data, N * sizeof(int), cudaMemcpyHostToDevice);

    kernel<<<1, N>>>(d_data);

    cudaMemcpy(d_data, data, N * sizeof(int), cudaMemcpyDeviceToHost);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }

    cudaFree(d_data);
    free(data);
}
```

OpenACC

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    #pragma acc data
    {
        #pragma acc parallel loop
        for (int i = 0; i < N; i++)
        { data[i] += 2; }
    }

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }

    free(data);
}
```

MANAGED MEMORY

Portable codes across x86 and Grace Hopper

CUDA C

```
#include <stdio.h>
#include <stdlib.h>

__global__ void kernel(int *data)
{
    data[threadIdx.x] += 2;
}

int main()
{
    int N = 128;
    int *data;

    cudaMallocManaged((void **)&data, N * sizeof(int));

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    // Optional
    cudaMemPrefetchAsync(data, N * sizeof(int), 0);

    kernel<<<1, N>>>(data);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }

    cudaFree(data);
}
```

OpenACC

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    #pragma acc parallel loop
    for (int i = 0; i < N; i++)
    { data[i] += 2; }

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }

    free(data);
}
```

“-acc -gpu=managed” flag used

USING SYSTEM ALLOCATOR

Super easy to move codes to GPU, portable with HMM on x86

CUDA C

```cpp
#include <stdio.h>
#include <stdlib.h>

__global__ void kernel(int *data)
{
    data[threadIdx.x] += 2;
}

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    kernel<<<1, N>>>(data);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }

    free(data);
}
```

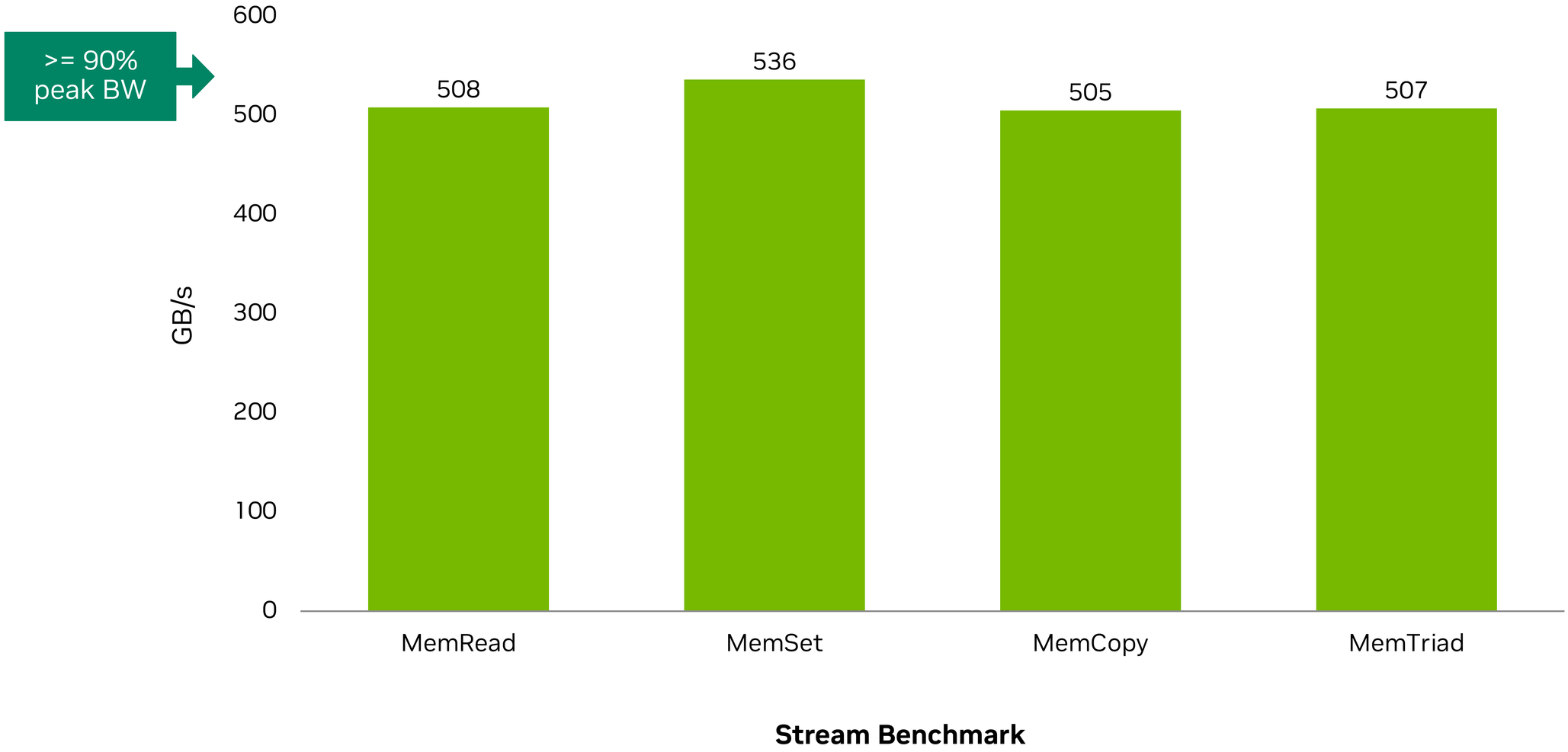
OpenACC

```cpp
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));
    int *d_data;

    for (int i = 0; i < N; i++)
    { data[i] = i; }

    #pragma acc parallel loop
    for (int i = 0; i < N; i++)
    { data[i] += 2; }

    for (int i = 0; i < 128; i++)
    { printf("%d ", data[i]); }
    free(data);
}
```

“-acc -gpu=managed” flag used

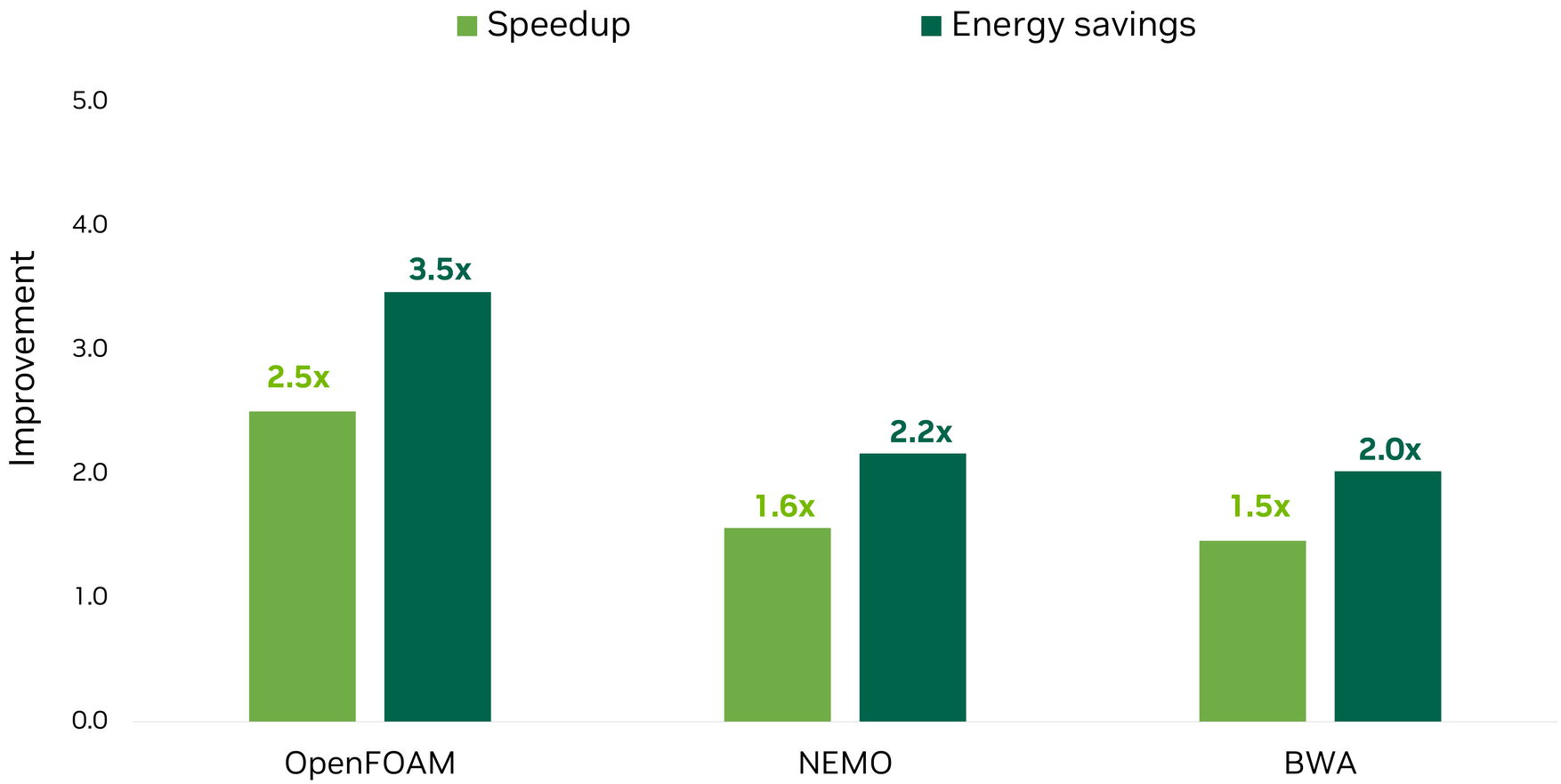
Performance of the NVIDIA Superchip

Grace CPU Memory Benchmark



Source: NVIDIA Grace pre-silicon results for single Grace SOC, subject to change

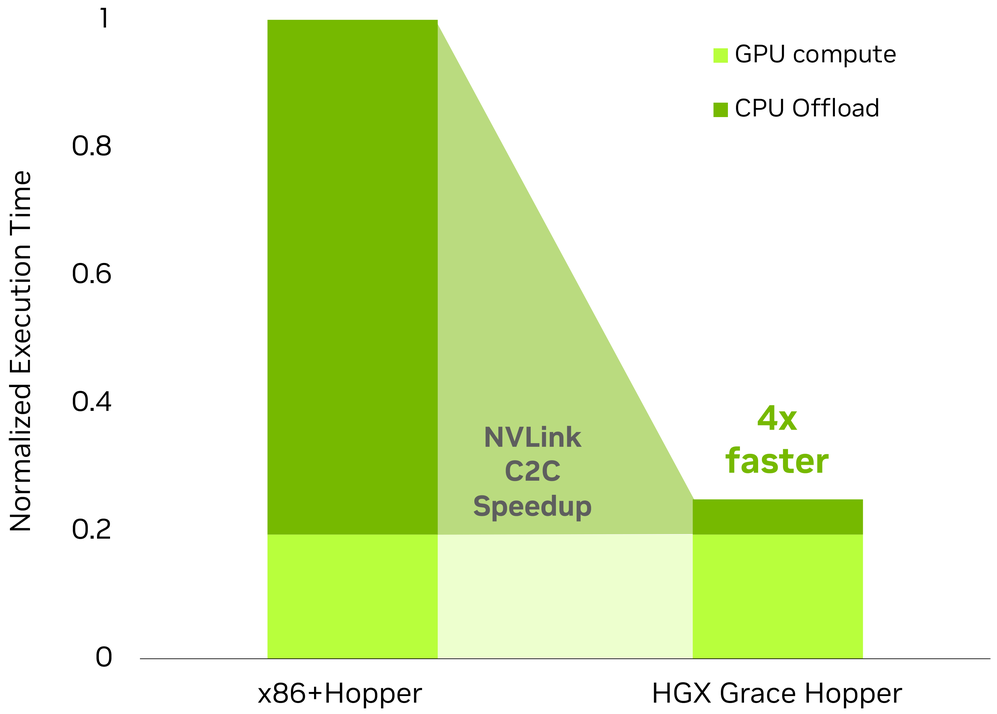
Grace Single CPU vs AMD Milan 7763 single socket



Projected performance subject to change. A100 cluster: HDR IB network. H100 cluster: NDR IB network with NVLink Network where indicated.
GPUs: Climate Modelling 1K, LQCD 1K, Genomics 8, 3D-FFT 256, MT-NLG 32 (batch sizes: 4 for A100, 60 for H100 at 1sec, 8 for A100 and 64 for H100 at 1.5 and 2sec), MRCNN 8 (batch 32), GPT-3 16B 512 (batch 256), DLRM 128 (batch 64K), GPT-3 175B 16K (batch 512), MoE 8K (batch 512, one expert per GPU)

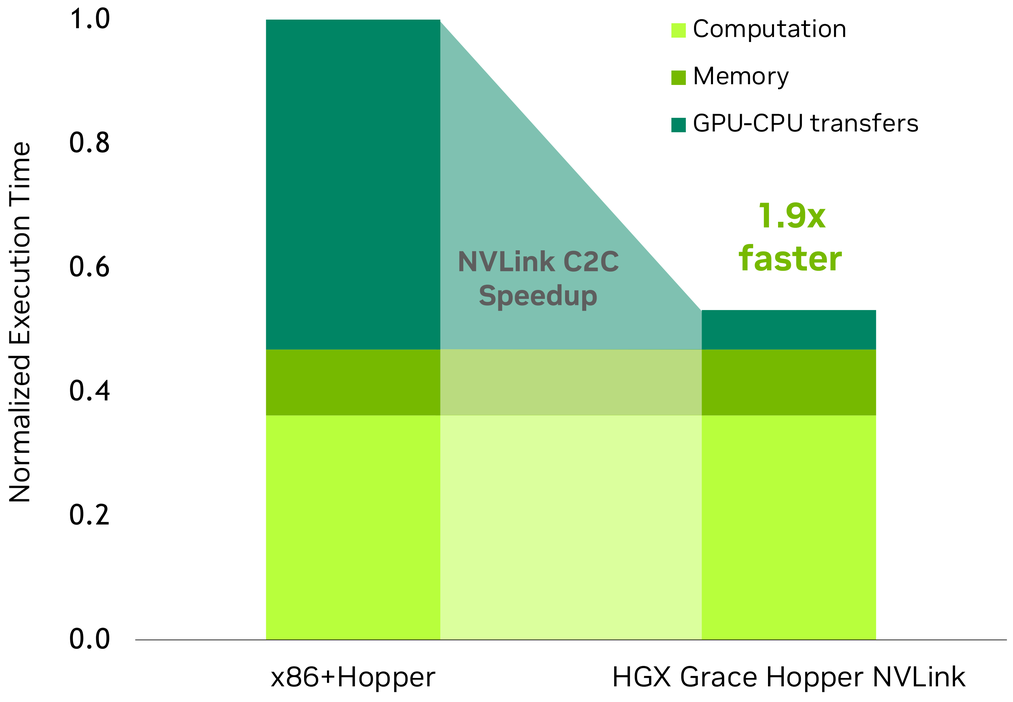
Deep Learning / AI workloads

Natural Language Processing Networks



Workload/Model: P-tune GPT3
Dataset: 175B NLP Model implemented with offloading tensors to CPU memory during the training iteration.

Graph Neural Networks



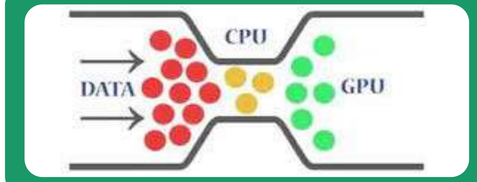
Workload/Model: GraphSAGE
Dataset: augmented version of the ogbn-products dataset having 626M nodes and 31B edges, coming to about 500 GB in size.

Where is Grace+Hopper better than X86+hopper?

HPC use-cases

Partially Ported Apps

- OpenFOAM – solver only (bar is lower to better price/perf)



Apps that bottleneck on PCI connectivity

- ABINIT example with pencil-shaped ZGEMM
- Large AI Training

Apps that can leverage tight cache coherence

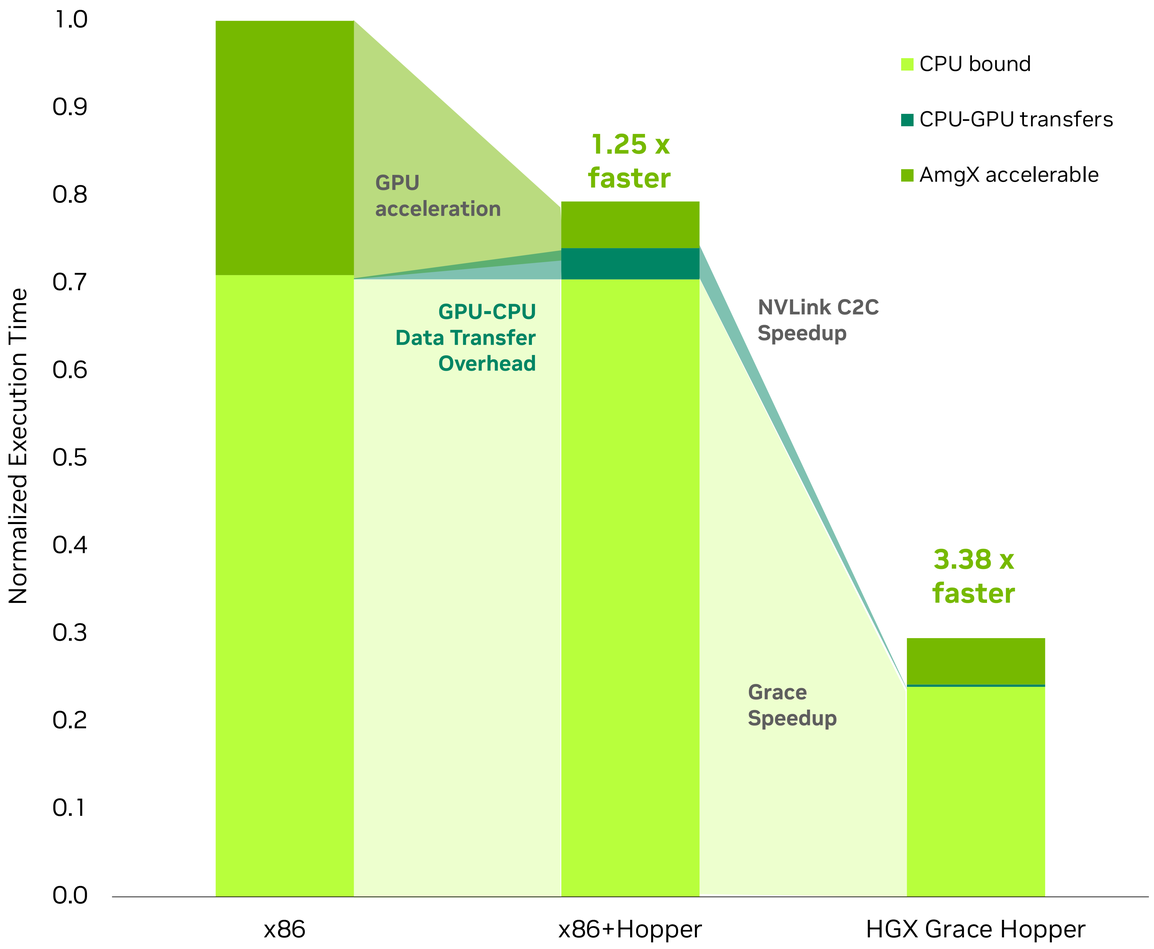
- Data Assimilation step in weather models can stay on Grace
- Multigrid solvers

New-to-GPU Apps

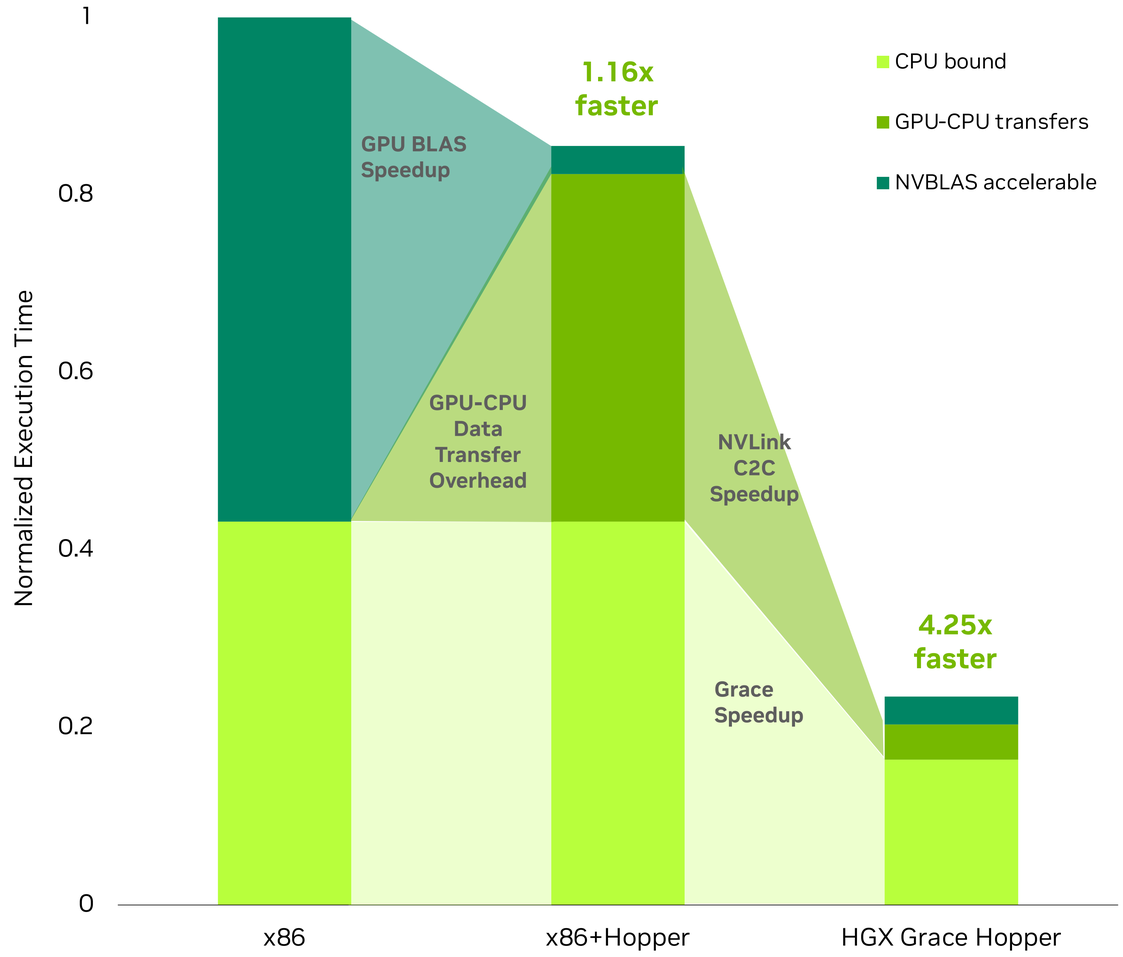
- Can more effectively leverage standard language acceleration

Partially accelerated HPC apps

OpenFOAM



ABINIT

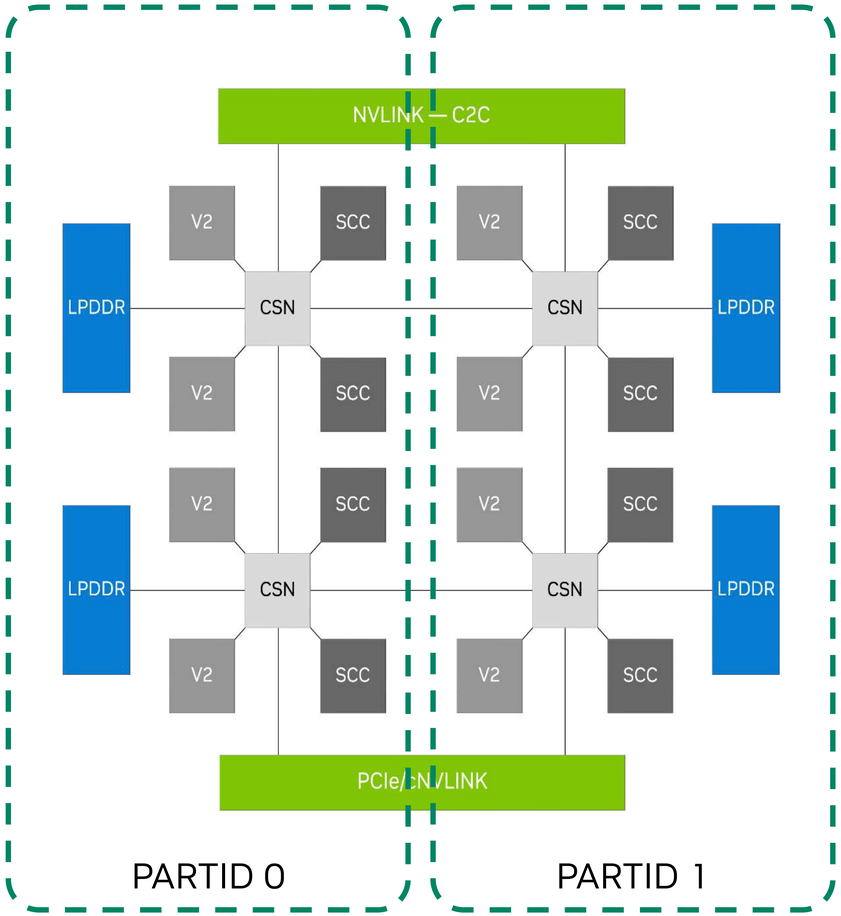


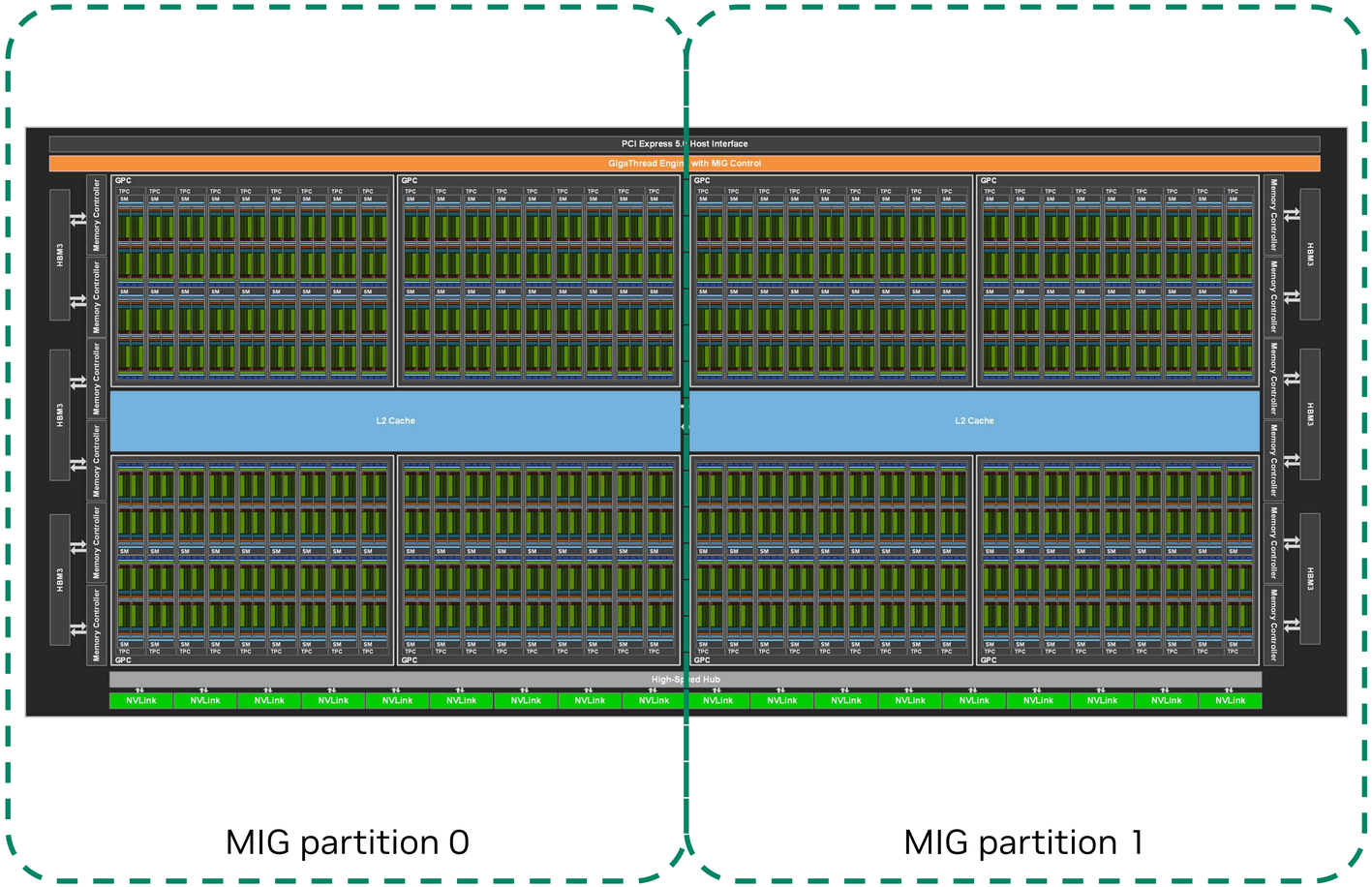
NVIDIA

GRACE HOPPER UTILIZATION

Flexible computing

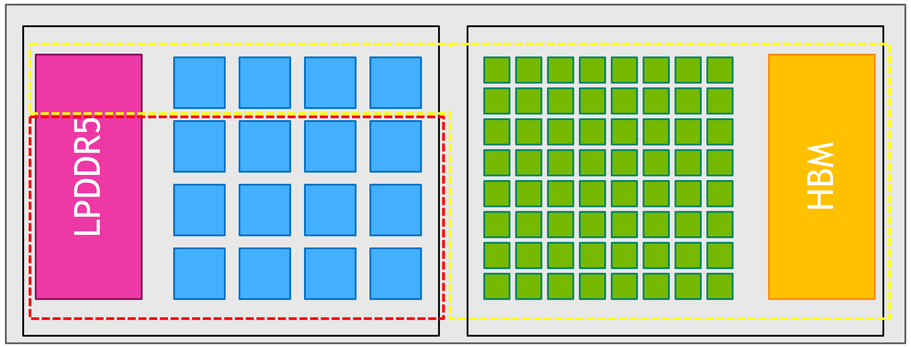
System administrations can combine CPU partitions and GPU MIG instance. NVLink C2C can be partitioned with CPU partitioning



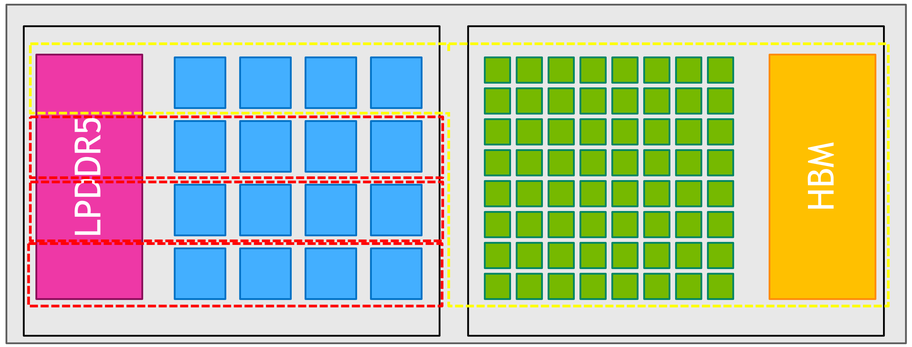


Co-scheduling opportunities for complex workflows

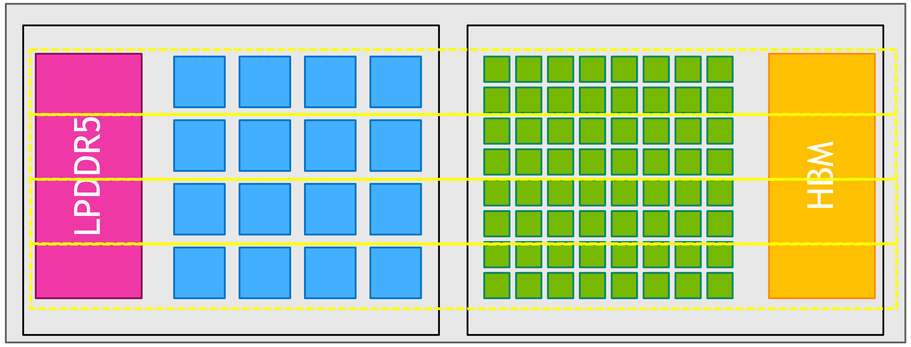
Maximize utilization, not only performance



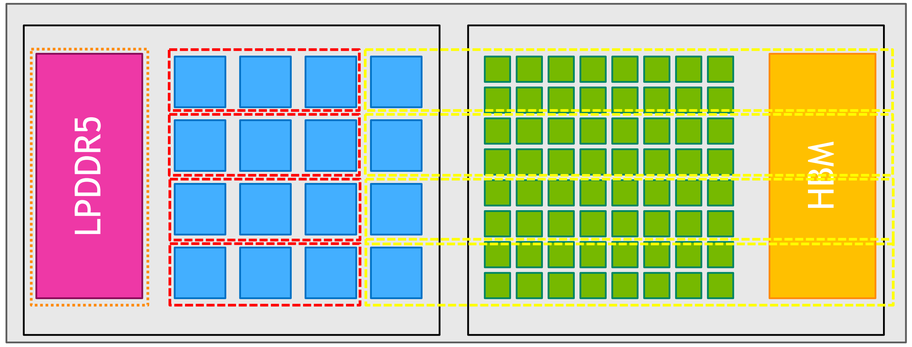
Use-case: 1 CPU job, 1 GPU job (1 MPI)
CPU: cgroup + MPAM
GPU: full control



Use-case: N CPU job, 1 GPU job (multiple MPI)
CPU: cgroup + MPAM
GPU: full control + MPS



Use-case: multiple GPU jobs
CPU: multiple process, cgroup + MPAM
GPU: partitioned with MIG



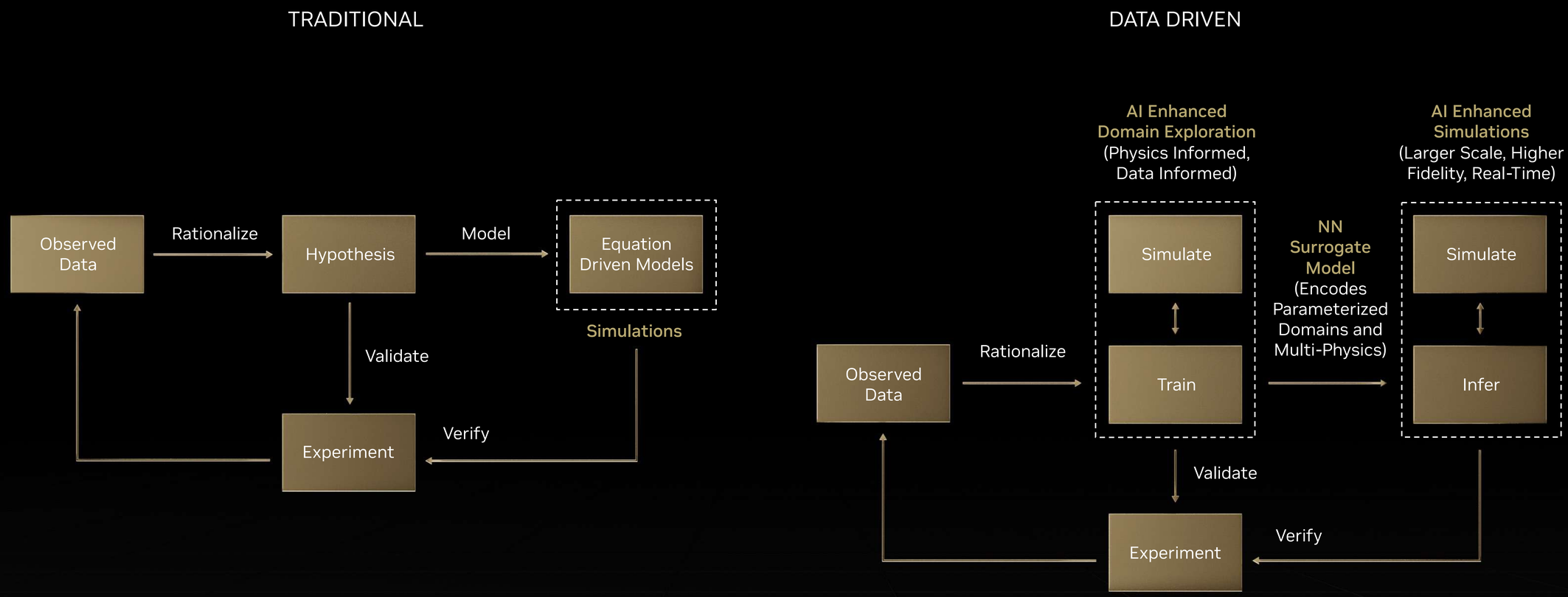
Use-case: multiple GPU jobs, multiple CPU jobs
CPU: multiple processes, cgroup + MPAM
GPU: partitioned with MIG


NVIDIA

Beyond HPC

HPC+AI, Edge, Digital Twins

AI IS THE 4TH PILLAR OF SCIENTIFIC DISCOVERY

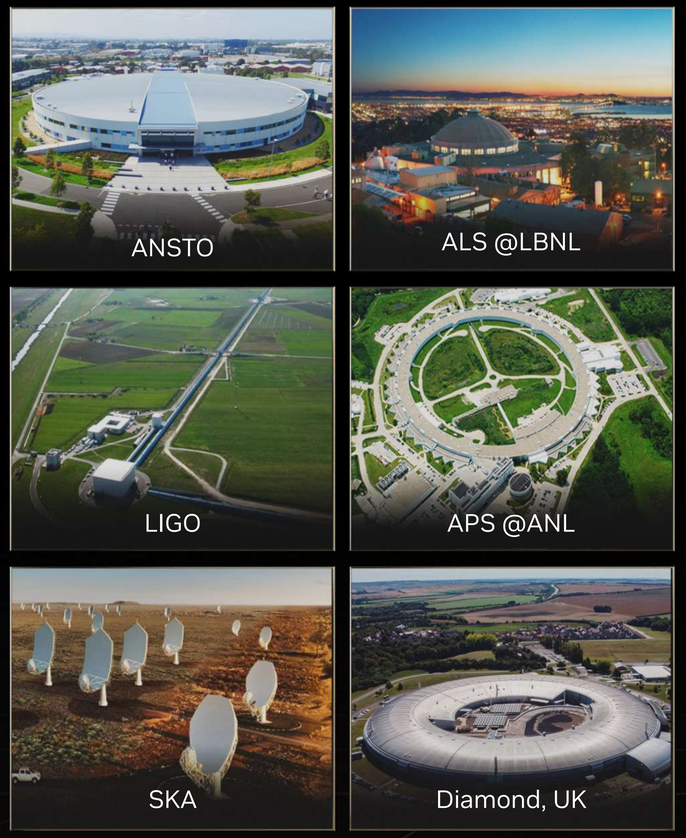


POSING A NEW SET OF CHALLENGES FOR HPC

Rise of HPC at the Edge

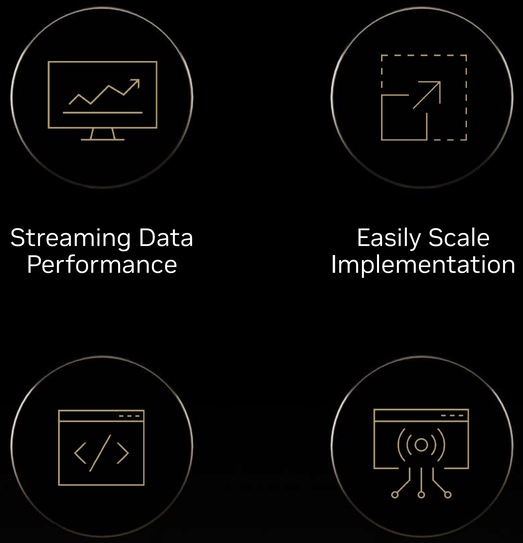
10X – 1000X MORE DATA

50+ Giant Scale Instruments



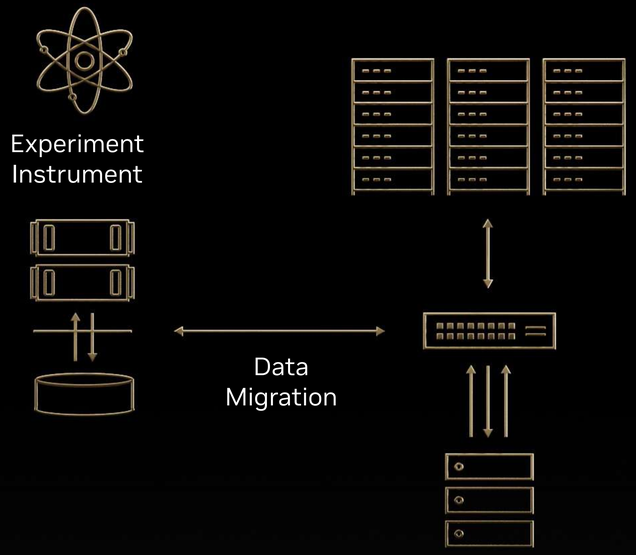
ACCELERATING STREAMING DATA CHALLENGES

For Data Scientists, Researchers and DevOps



DATA MOVEMENT CHALLENGES

Enables Real-Time Insights and Control



AUTOMATING THE AI WORKFLOW FOR SCIENTIFIC DISCOVERY

New C++ and Python APIs | Scalable from IGX to Cloud | Sample Applications | ETA Mid-December 2022

HOLOSCAN / WORKFLOW MANAGEMENT SYSTEM

UCF / HOLOSCAN / WORKFLOW MANAGEMENT SYSTEM

Edge Compute

Pre-Processing

Reconstruction

Visualize

Feedback to Sensor

Sensor Steering / Control

300X

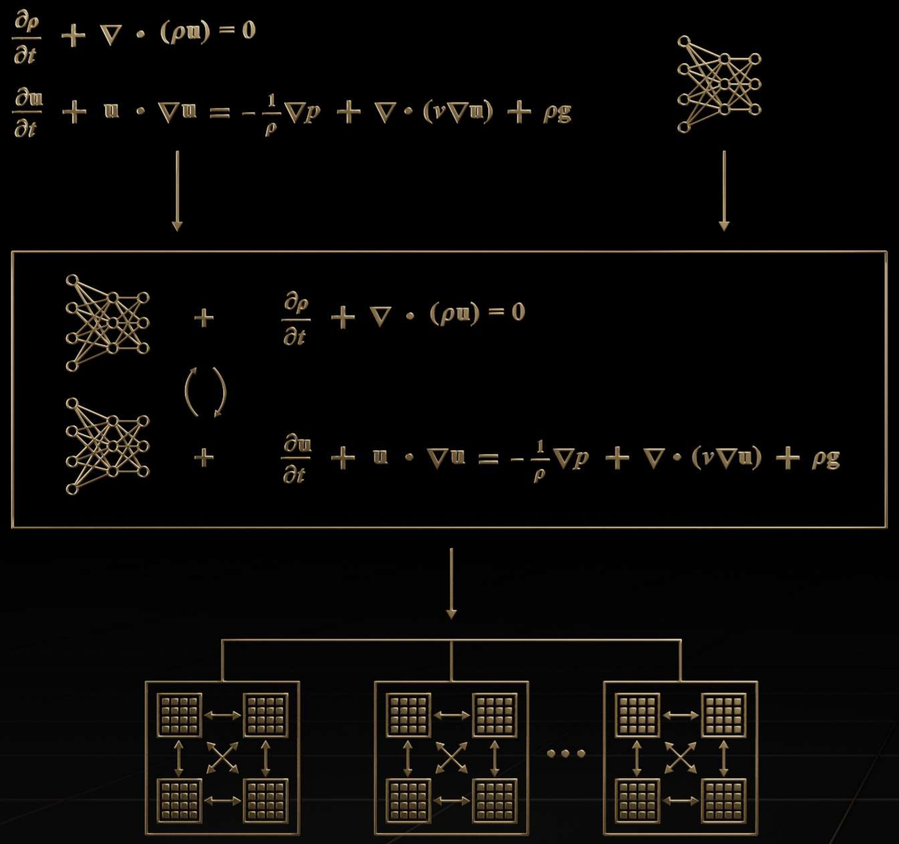
Performance Increase*

UCF: Unified Compute Framework

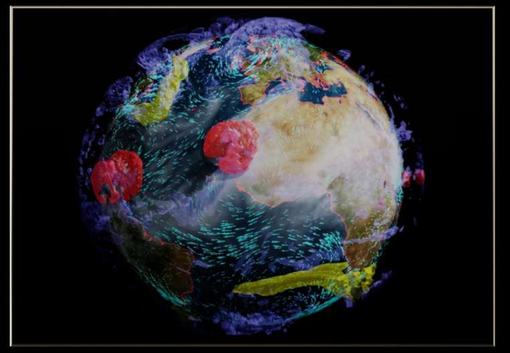
* With PtychoNN, See Real-time sparse-sampled Ptychographic imaging through deep neural networks

NVIDIA MODULUS

Shortcut to Surrogate Models for Interactive Simulation | Development Platform for AI Surrogate Models

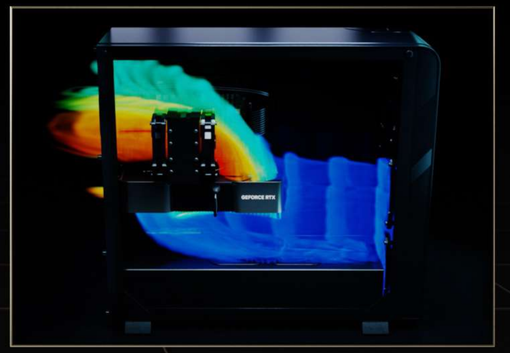


WEATHER & CLIMATE MODELING
FourCastNet



RENEWABLE ENERGY
Siemens Gamesa Wind Farm

PRODUCT DEVELOPMENT
Thermal Airflow Digital Twin

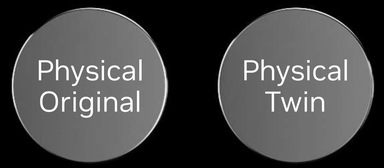


INDUSTRIAL DIGITAL TWIN
Data Center CFD Acceleration

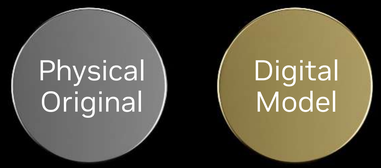
Now Available on NVIDIA LaunchPad and Major CSPs

SCIENTIFIC DIGITAL TWINS ARE EVOLVING TO ENABLE NEW WORKFLOWS

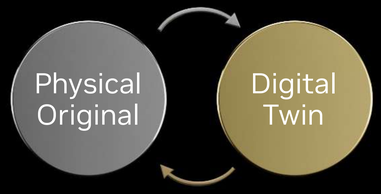
PHYSICAL

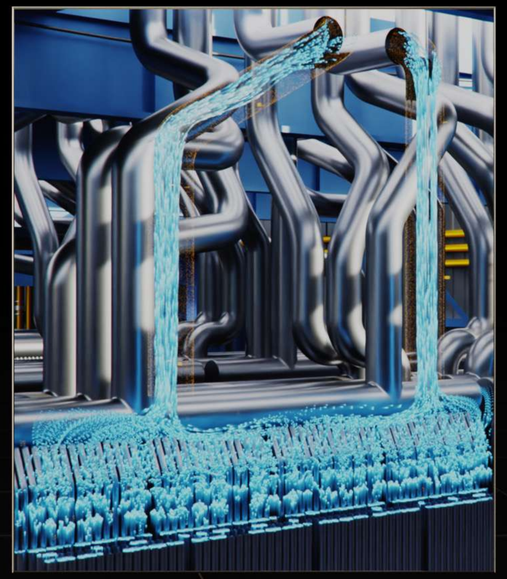


BATCH SIMULATION
Months-to-Days

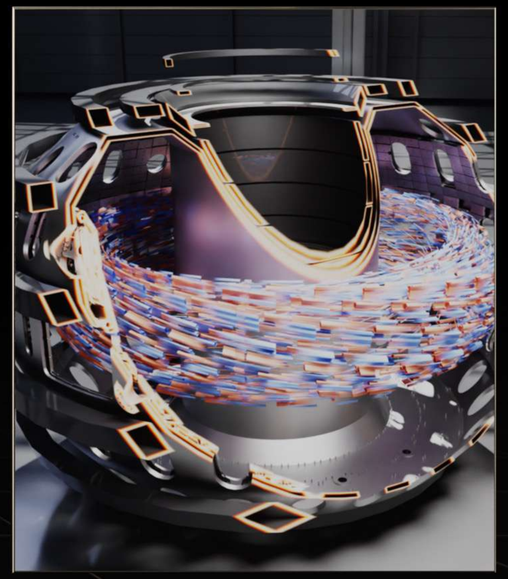


INTERACTIVE SIMULATION
Hours-to-Minutes



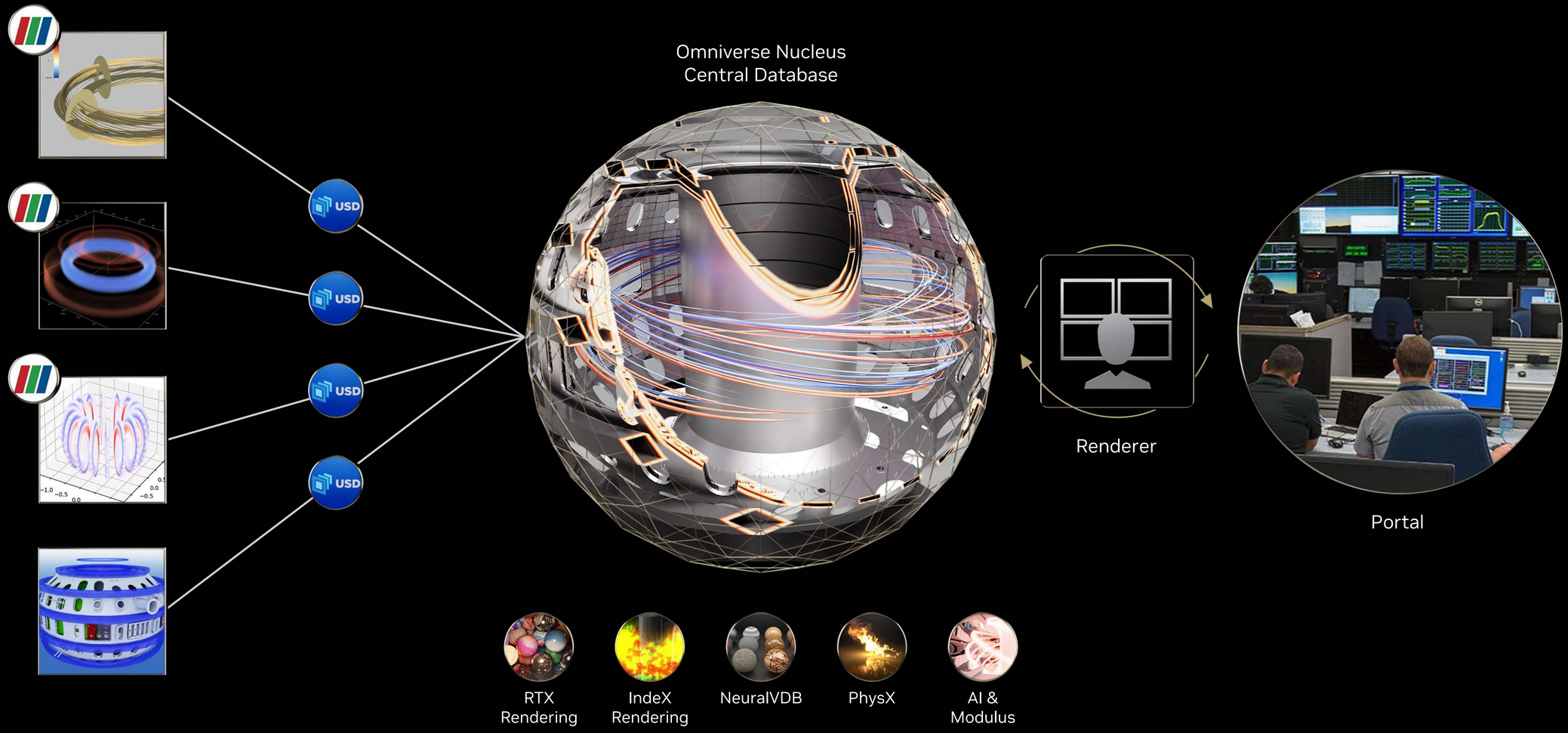


VIRTUAL/PHYSICAL STEERING
Real-Time



NVIDIA OMNIVERSE FOR SCIENTIFIC COMPUTING

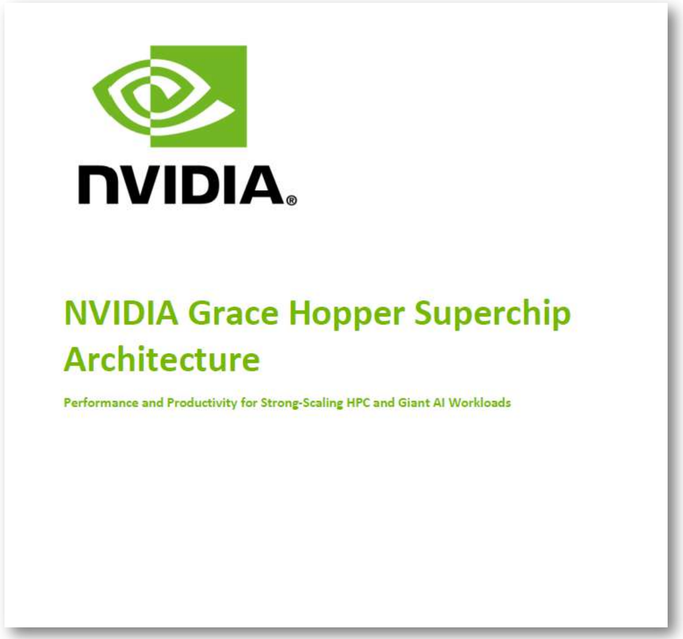
Connecting Complex HPC 3D and Simulation Workflows




NVIDIA

Grace Hopper Superchip Whitepaper

- Grace Hopper platform
- Architecture with more details
- Programming model
- Latest applications and performance
- Whitepaper link



NVIDIA Developer Blog "*NVIDIA Grace Hopper Superchip Architecture In-Depth*"
https://developer.nvidia.com/blog/nvidia-grace-hopper-superchip-architecture-in-depth/