



# Developing for NVIDIA Superchips

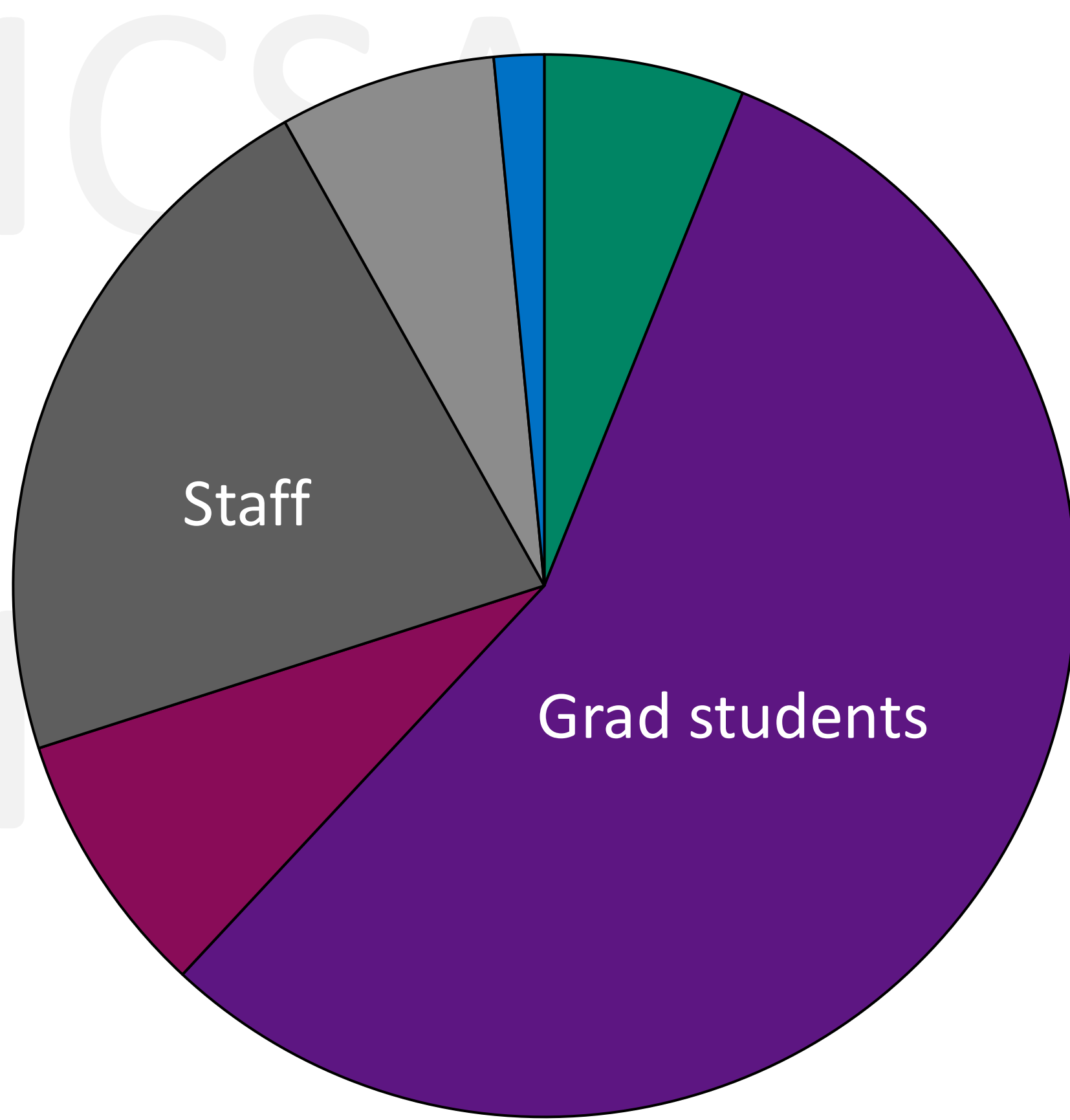
Dr. John Linford, Principal Technical Product Manager

[jlinford@nvidia.com](mailto:jlinford@nvidia.com)



# Who are you and what do you want to learn?

Thanks for the feedback! It is very much appreciated!



*“Grace Hopper architecture”*

*“Best way to develop / build software for Grace”*

*“How to run simulations/applications on Grace Hopper”*

*“New programming model and APIs”*

*“Distributed CPU/GPU programming”*

*“Everything about computer science”*

*“I’m a relatively new NCSA staff member in MarComm, and I am getting more familiar with what we do.*

*I am a sponge.”*





# Agenda

- Grace Hopper and Grace CPU Headlines in HPC

---
- Introduction to NVIDIA Superchips

---
- Programming Grace Hopper and Grace CPU Superchip

---
- Porting and Optimizing for Grace CPU

---
- Optimizing for GH200 and GB200 Coherent Memory



# Not on the Agenda

- Delta AI system specifics: logging in, running jobs, ...

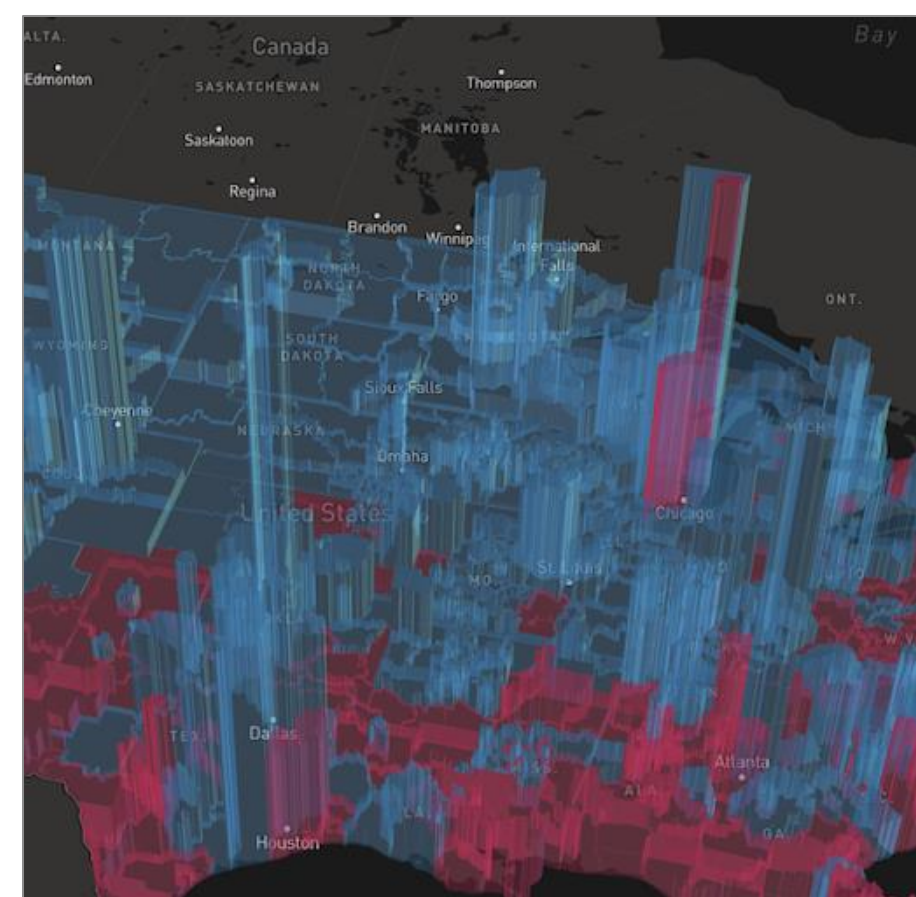
---
- Optimizing for third-party tech: Slingshot, Lustre, ...

---
- CUDA fundamentals (but we will cover new APIs)

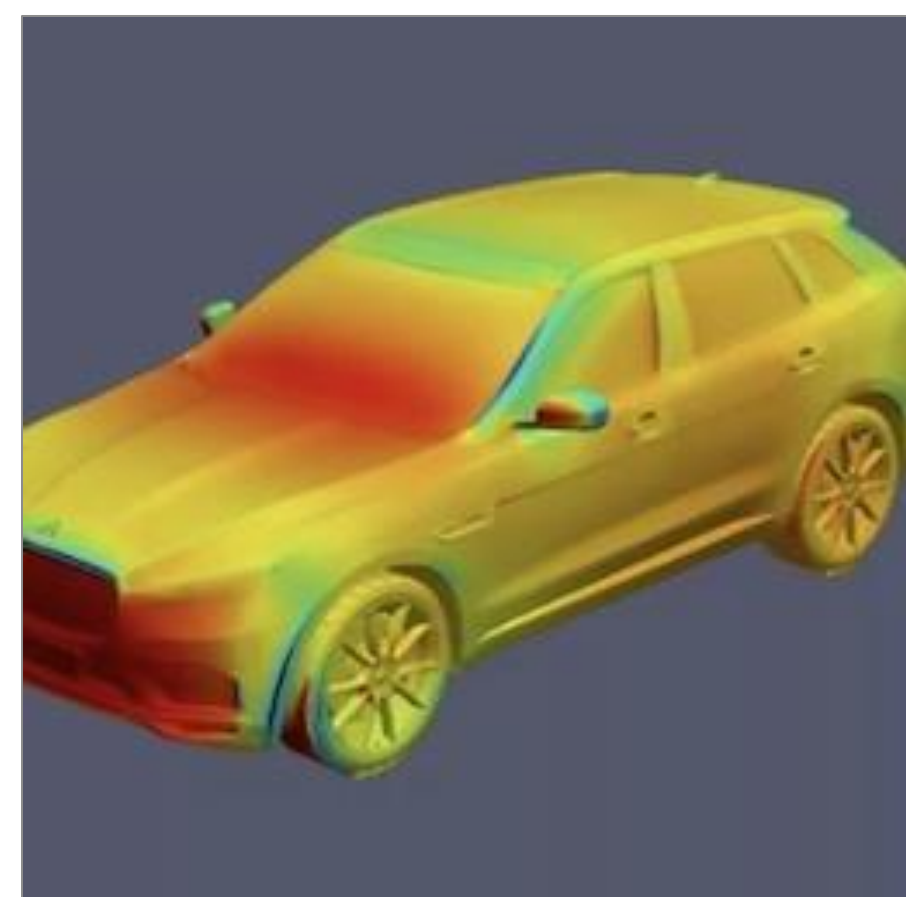


# NVIDIA AI Accelerated Computing Platform

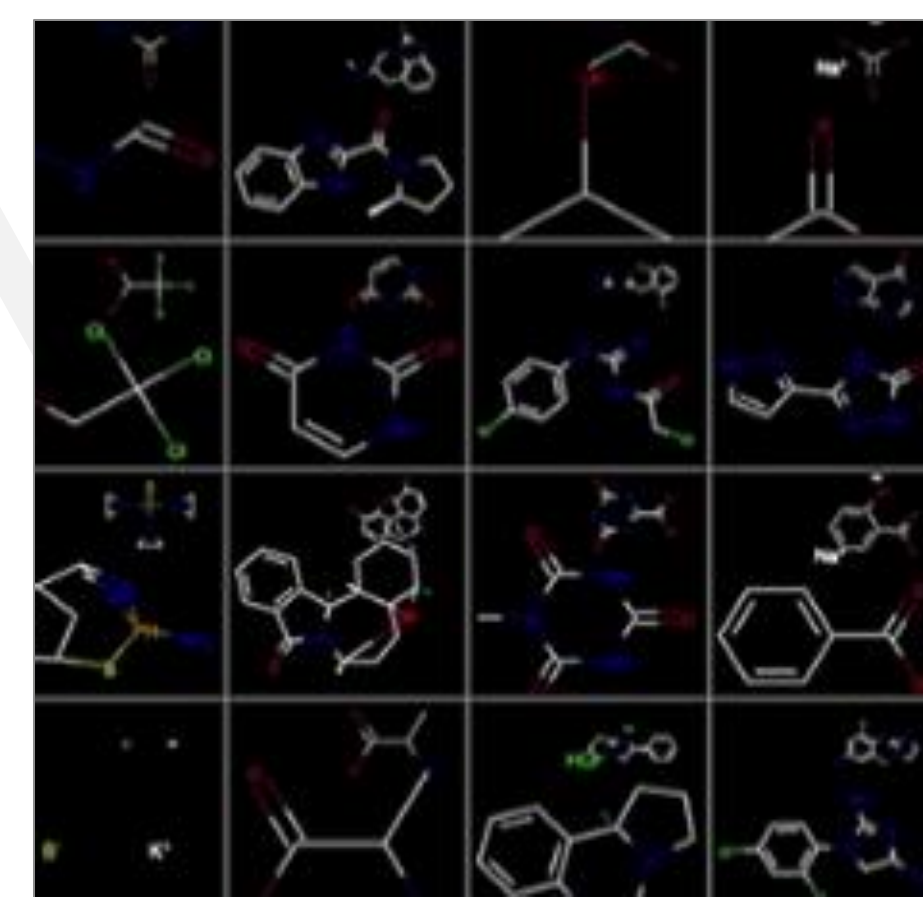
Hardware and Software Acceleration Across Every Workload and Vertical



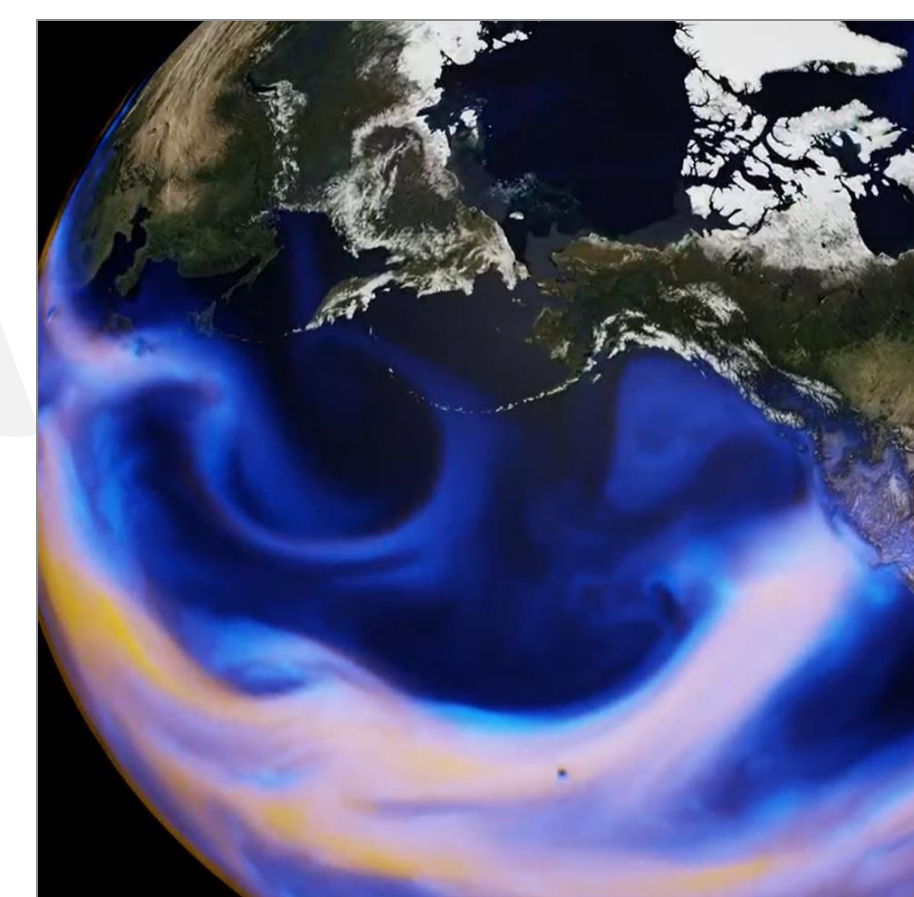
Data Processing



CAD, CAE, SDA



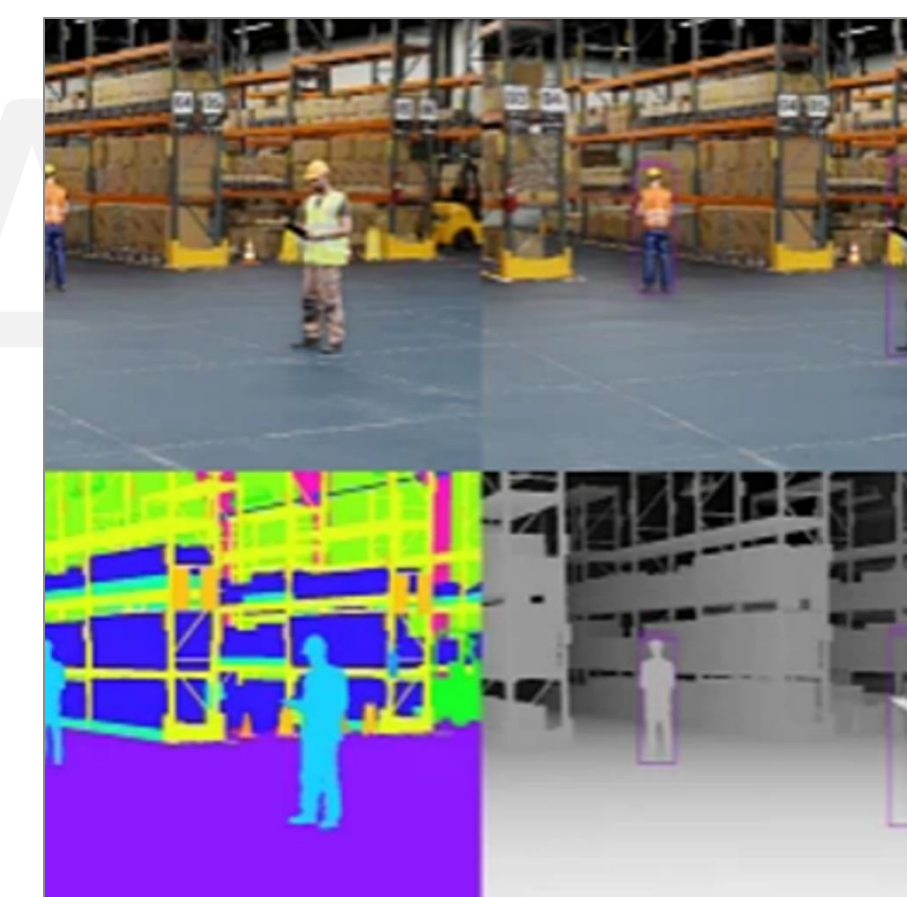
Computer-aided Drug Design



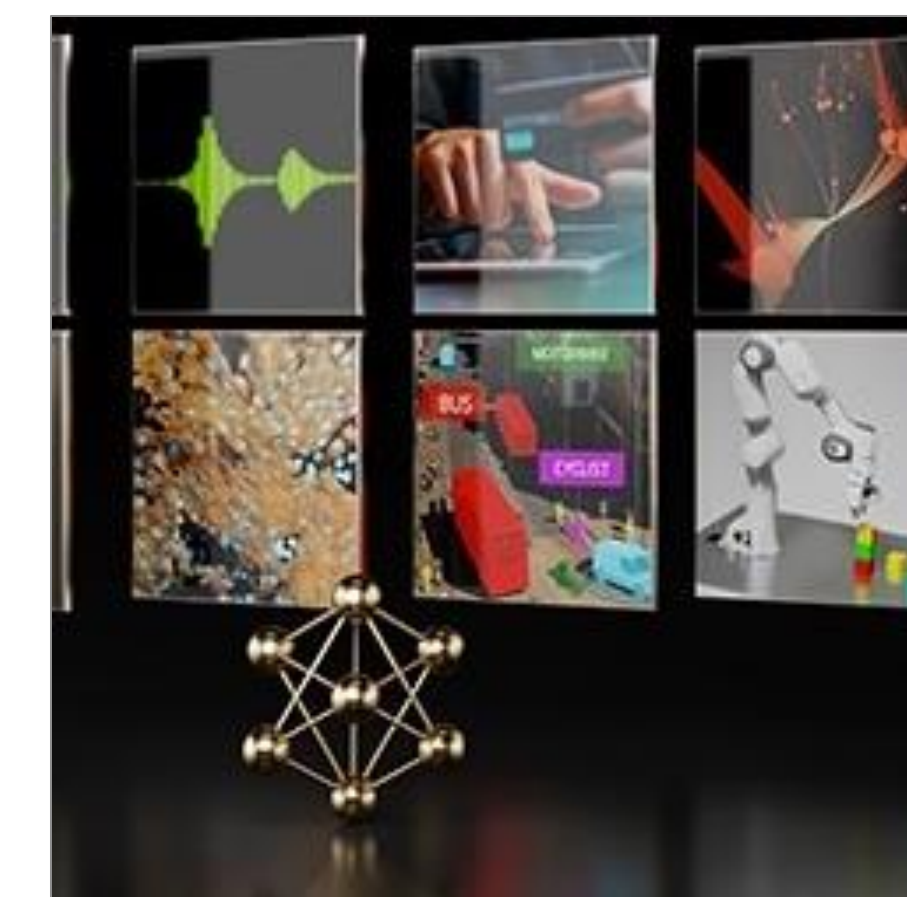
Climate Simulation



Quantum Simulation

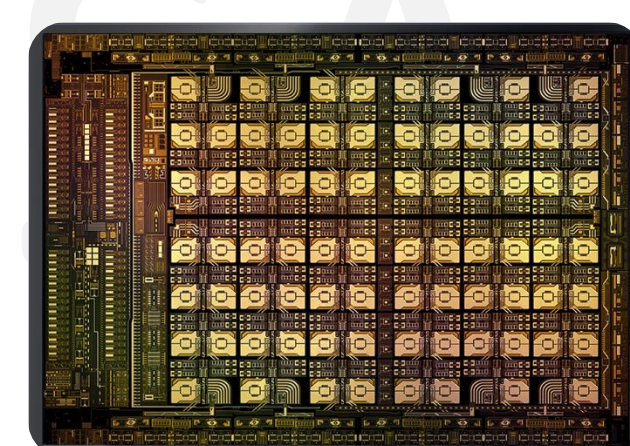


Robotics & Industrial Digital Twins

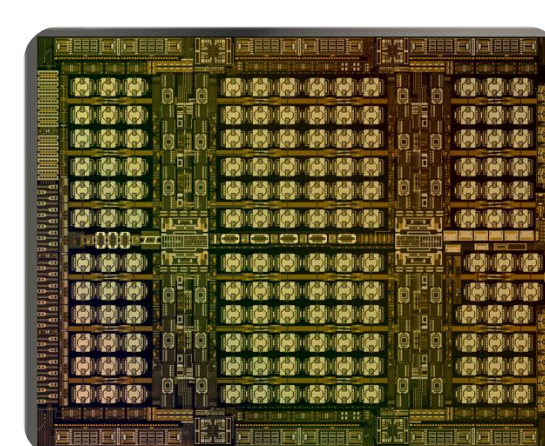


Enterprise AI

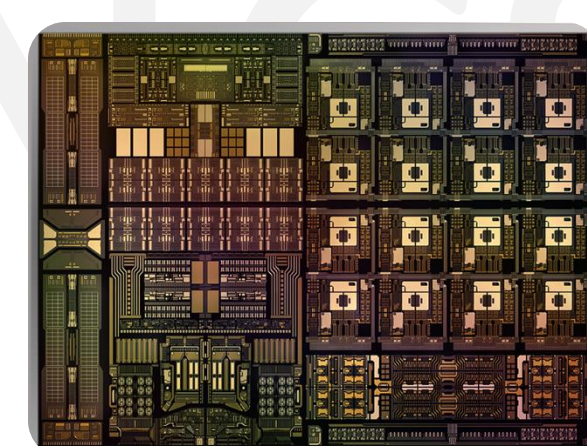
CUDA-X Libraries



CPU

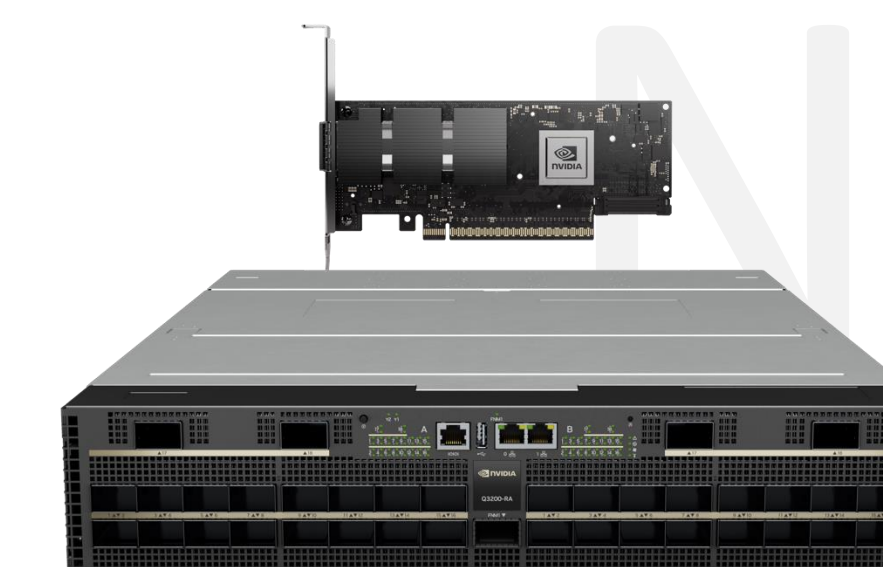
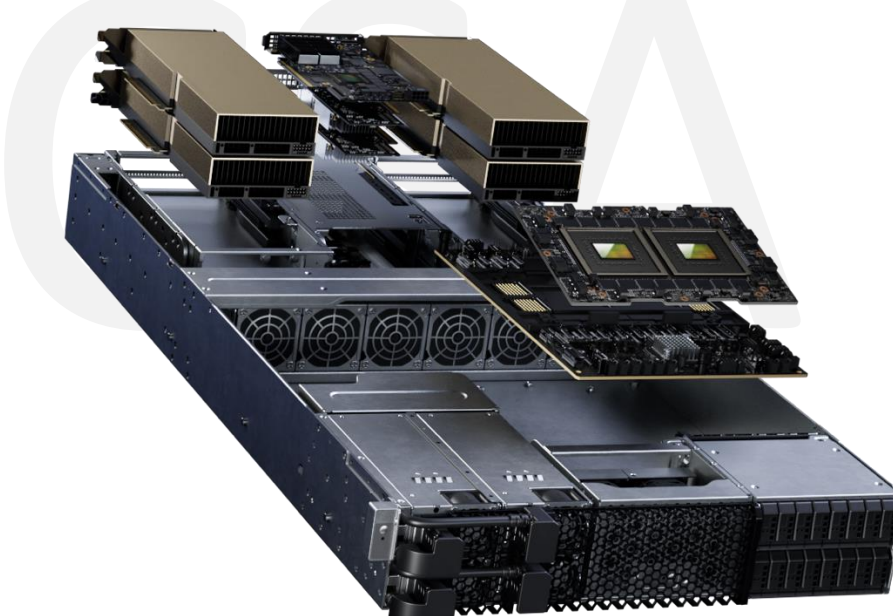


GPU



DPU

Accelerated Computing





# NVIDIA GH200 Grace Hopper Superchip

Built for the New Era of AI Supercomputing

CPU to GPU Bandwidth

**900GB/s**

NVLink-C2C

GPU Memory Bandwidth

**5TB/s**

HBM3e

Energy Efficiency

**50X**

MILC Efficiency vs 2S x86 CPUs

QFT Quantum Simulation

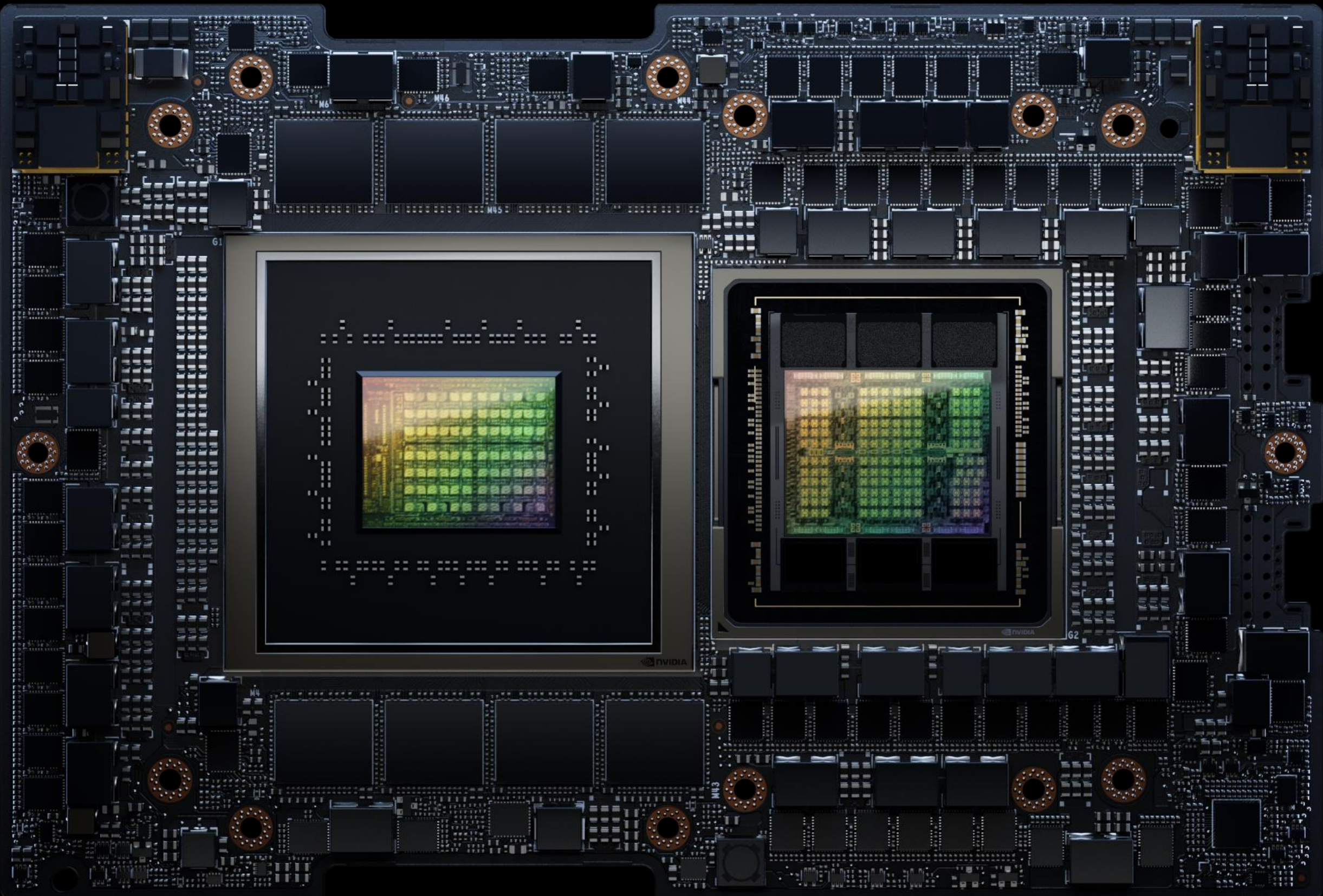
**90X**

Performance vs 2S x86 CPUs

LLM Inference

**200X**

Performance vs H100 80GB



624GB High-Speed Memory | 4 PF AI Perf | 72 Arm Cores

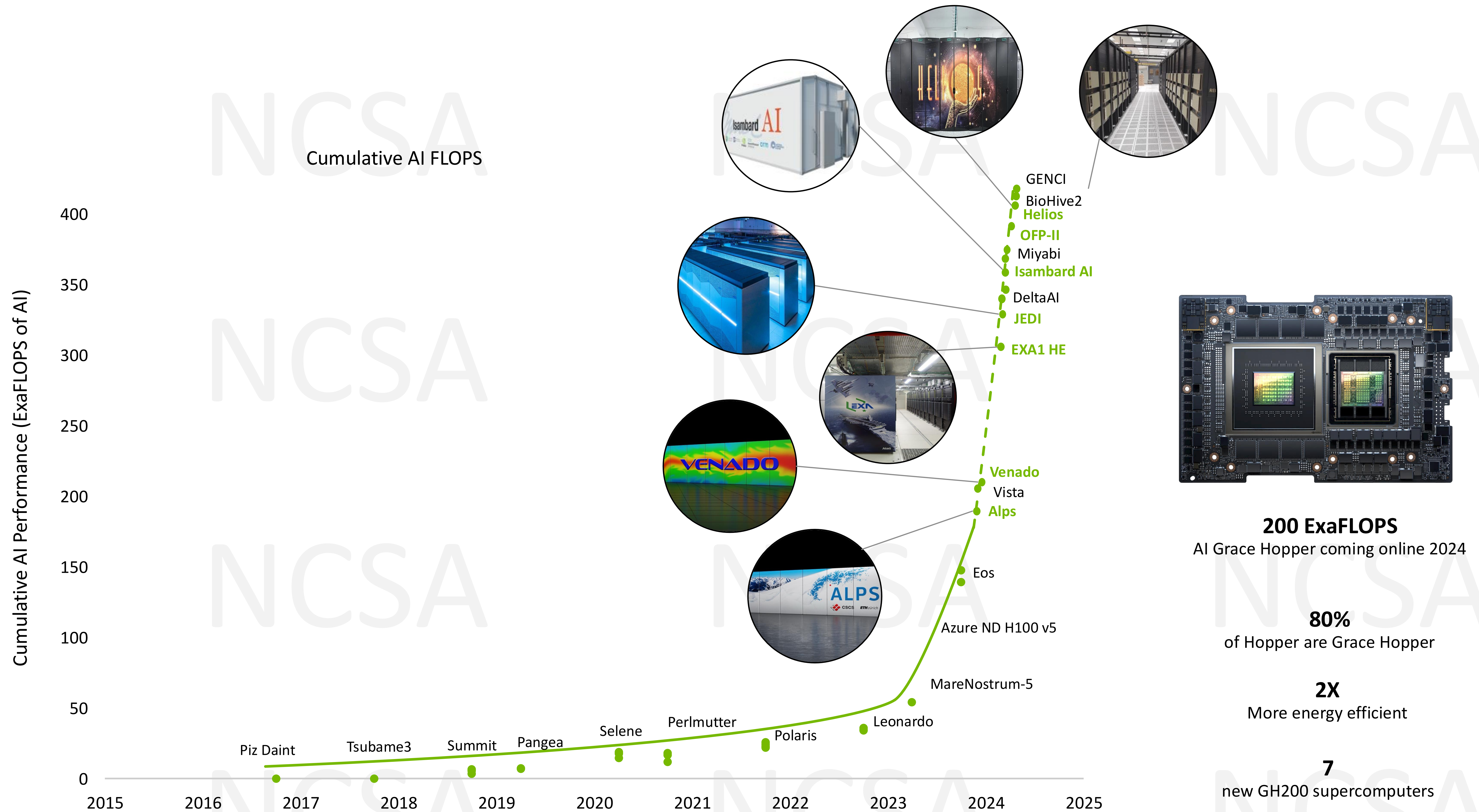


Preliminary measured performance, subject to change  
Energy Efficiency: GH200 144GB vs 2S Xeon 8480+ CPU for MILC running dataset: Apex Medium  
QFT Quantum Simulation: QFT 2S Xeon 8480+ vs GH200 144GB  
LLM Inference: Llama.cpp (2S 8480+) and TensorRT-LLM (GH200, H100) | Max Batch Llama2 70B | Throughput includes time to first token + token generation time



# Grace Hopper Powers AI Supercomputing Datacenters

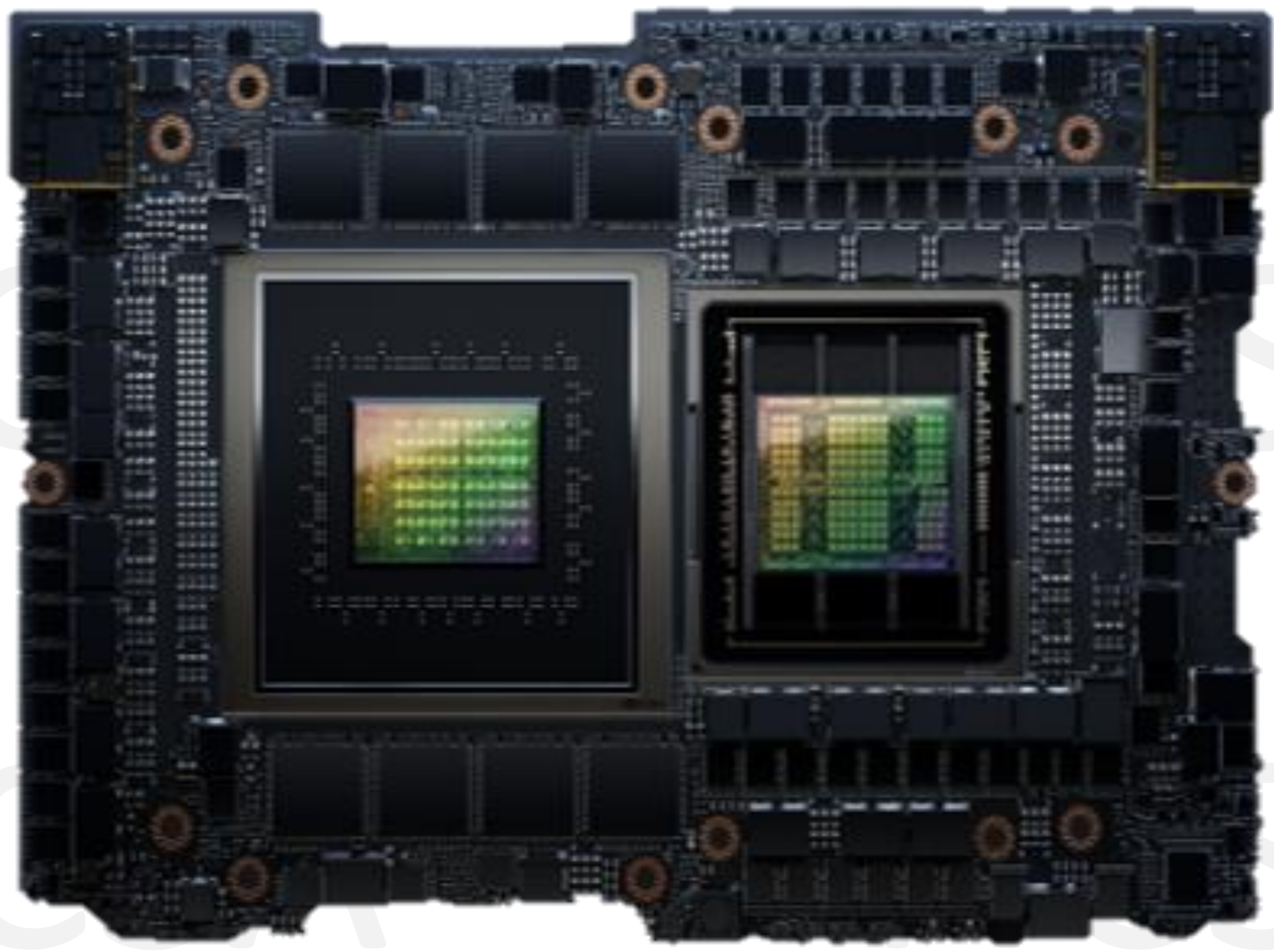
Grace Hopper Will Deliver 200 Exaflops of AI performance for Groundbreaking Research



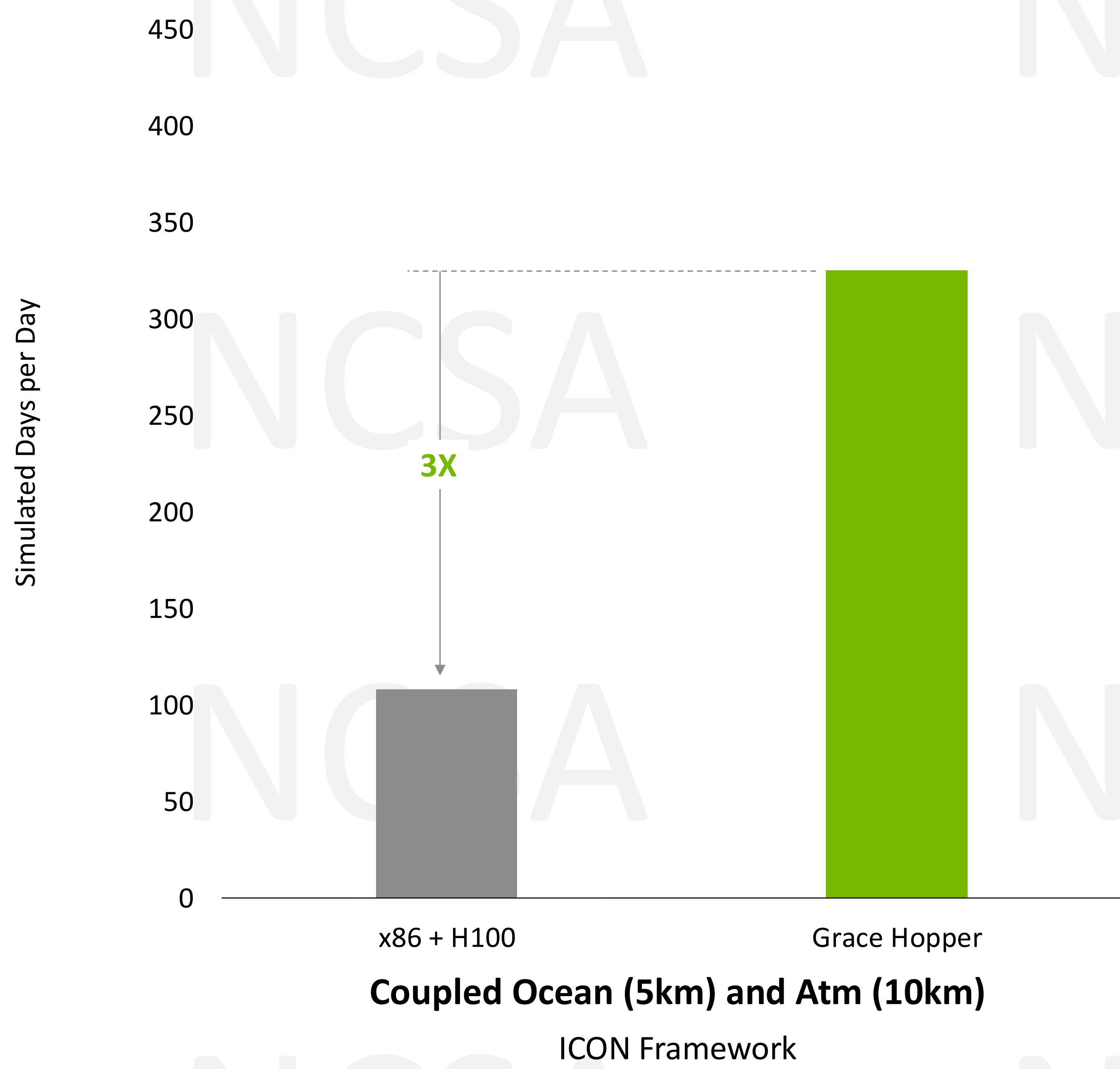


# CSCS Alps Climate Science Results on Grace Hopper

Up to 4.5X more performance for climate science



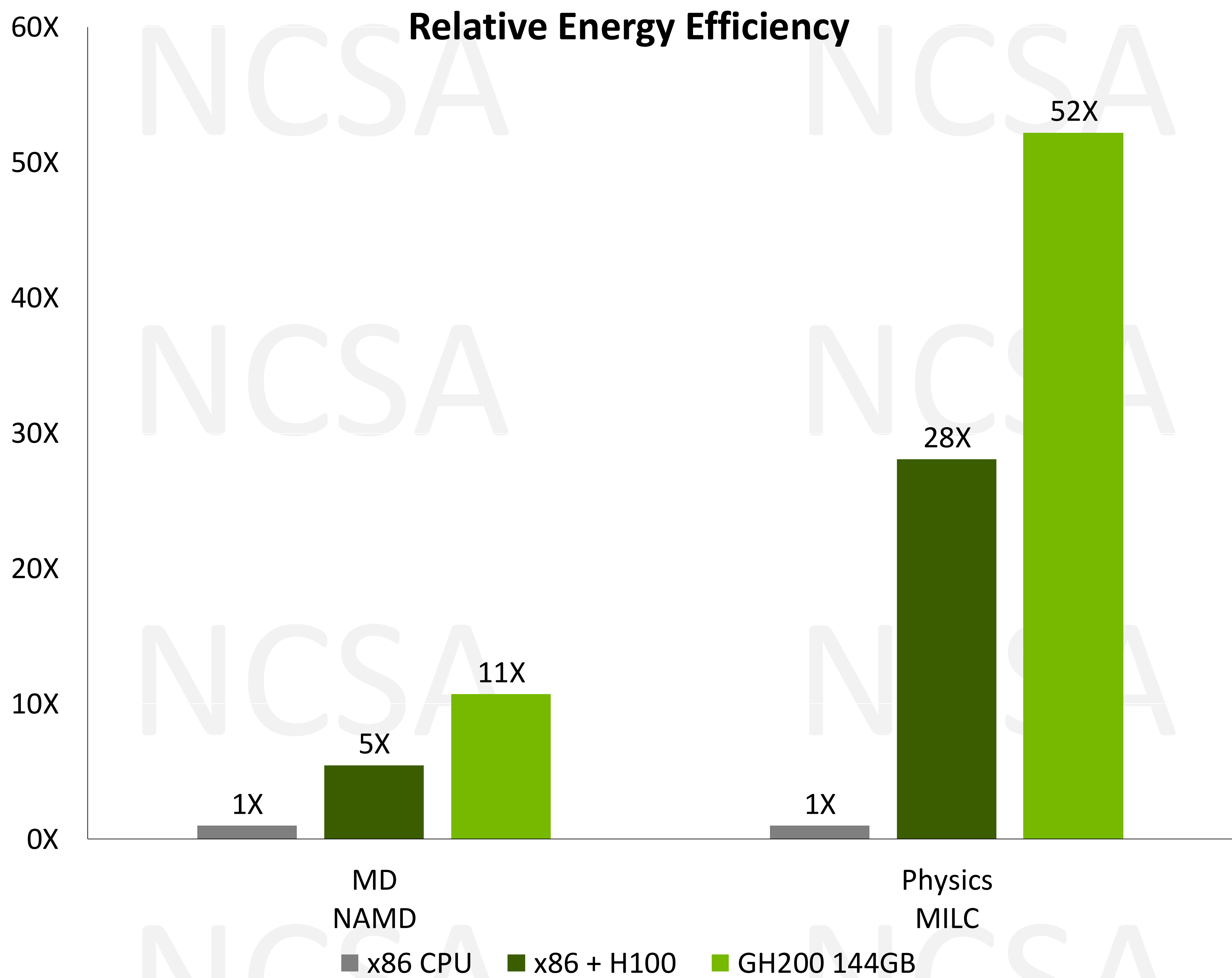
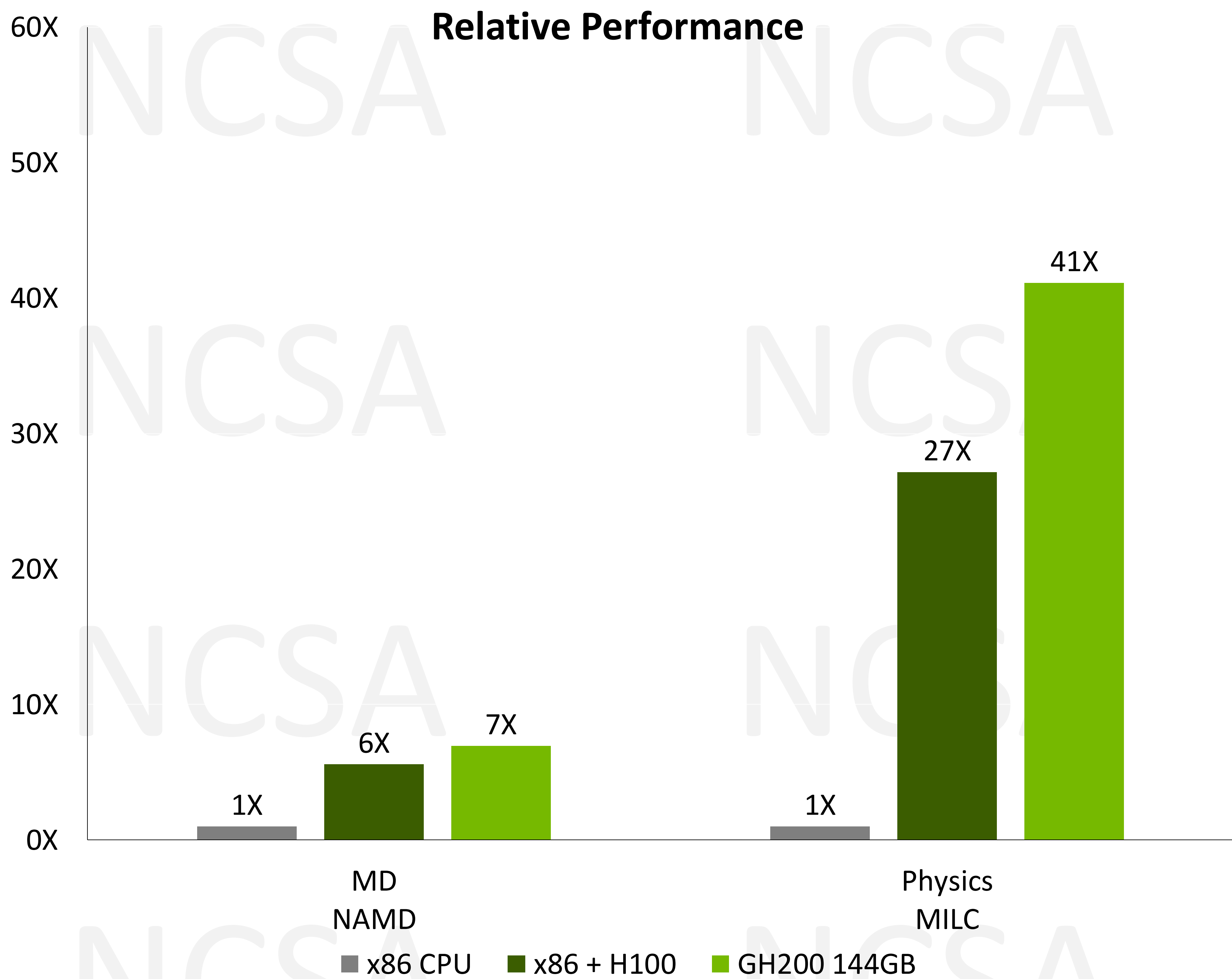
Alps GH200 has 4X more, faster CPUs accelerating Ocean simulations. Coupled model waits less and runs faster.





# NVIDIA GH200 Delivers Breakthrough HPC Performance

Up to 50X more energy efficient





# NVIDIA Grace CPU Superchip

Breakthrough Performance and Efficiency for the Modern Data Center

CPU + Memory Power

**500W**

Grace CPU Superchip TDP

Memory Bandwidth

**1 TB/s**

LPDDR5X

Green 500

**7** Grace systems

Performance, Energy Efficiency with GH200

Energy Efficiency

**2X**

Performance vs x86 CPU

Weather

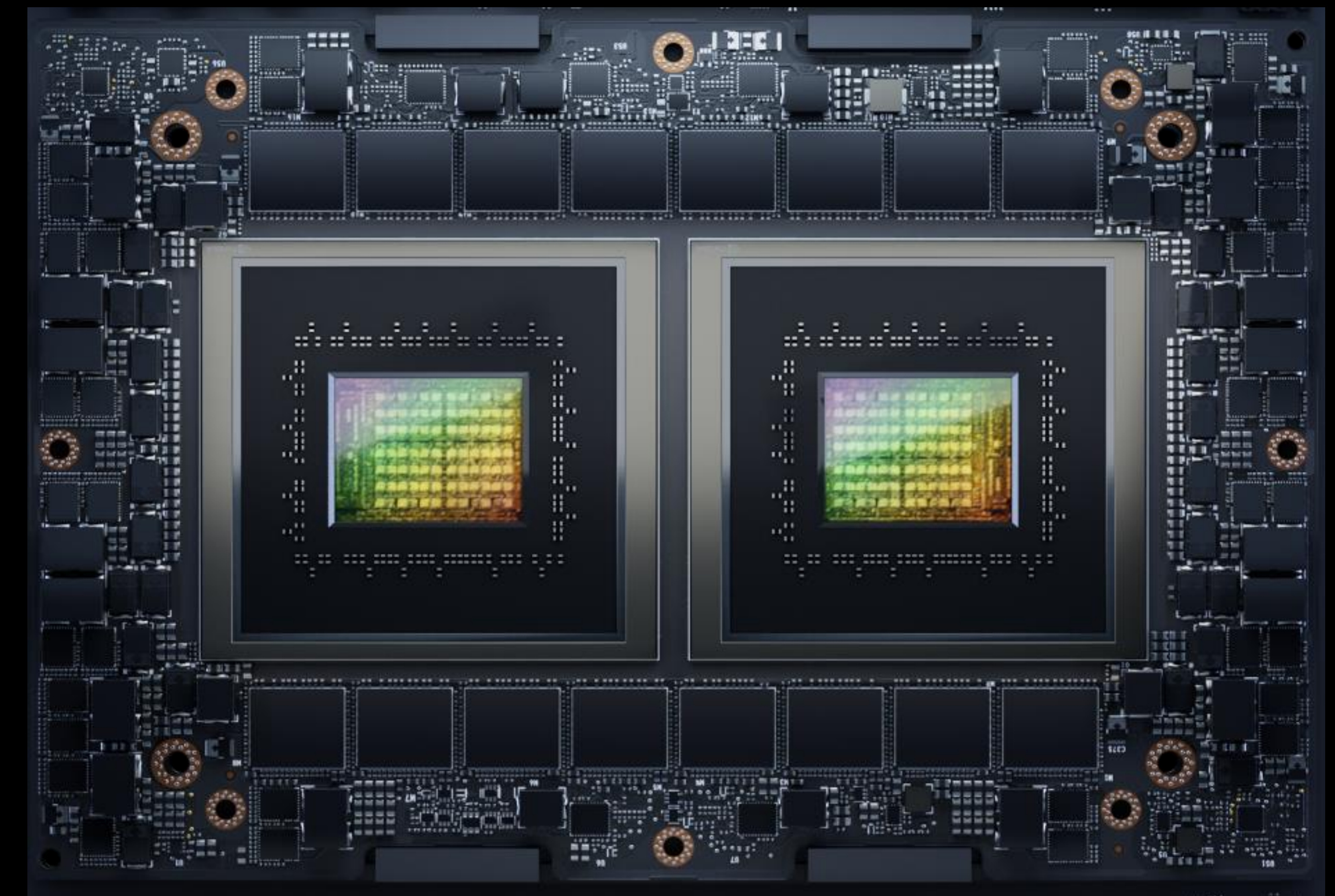
**1.3X**

Performance vs x86 CPU

Graph Analytics

**2X**

Performance vs x86 CPUs



144 Arm Neoverse V2 Cores | 234MB L3 Cache  
3.2 TB/s NVIDIA Scalable Coherency Fabric | 960GB LPDDR5X

Preliminary measured performance, subject to change

Energy Efficiency: Grace CPU Superchip vs 2S AMD EPYC 9654 and Xeon Platinum 8480+. Geomean of OpenFOAM (Motorbike Large), WFR (CONUS12km), ICON (QUBICC 80 km resolution) specfm3d (four\_material\_simple\_model) and Branson (3D\_hohlraum\_single\_node)

Weather: WRF (CONUS12km) Grace CPU Superchip vs 2S AMD EPYC 9654

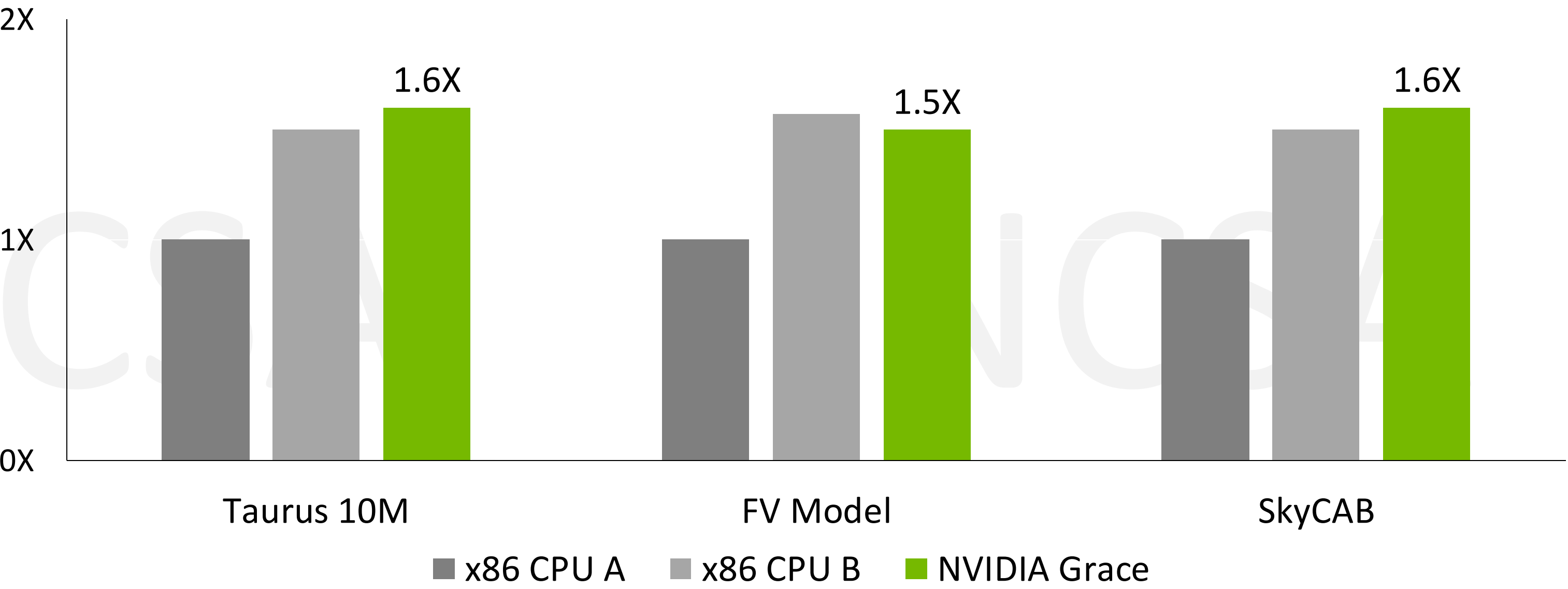
Graph Analytics: GAP BS Breadth First Search



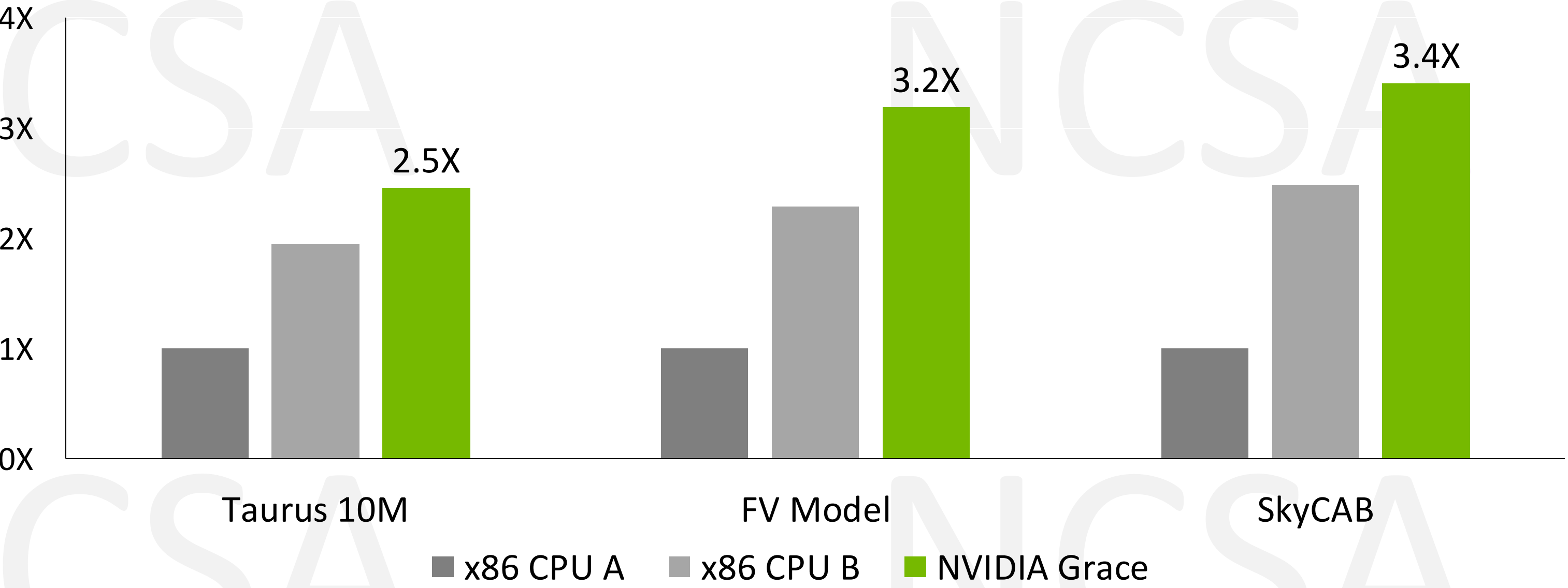
# OpenRadioss Crash and Impact

NVIDIA Grace CPU delivers up **3.4X** energy efficiency

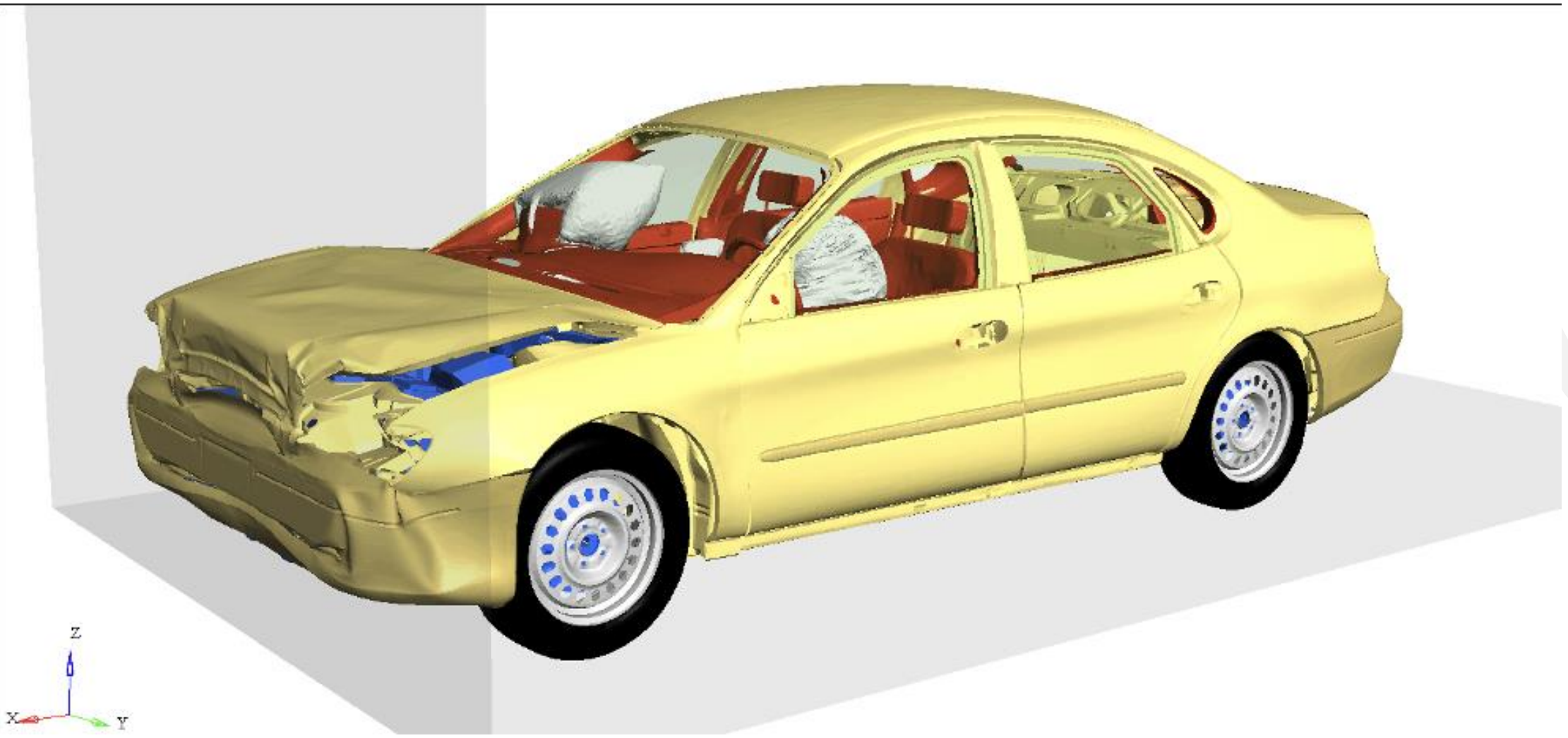
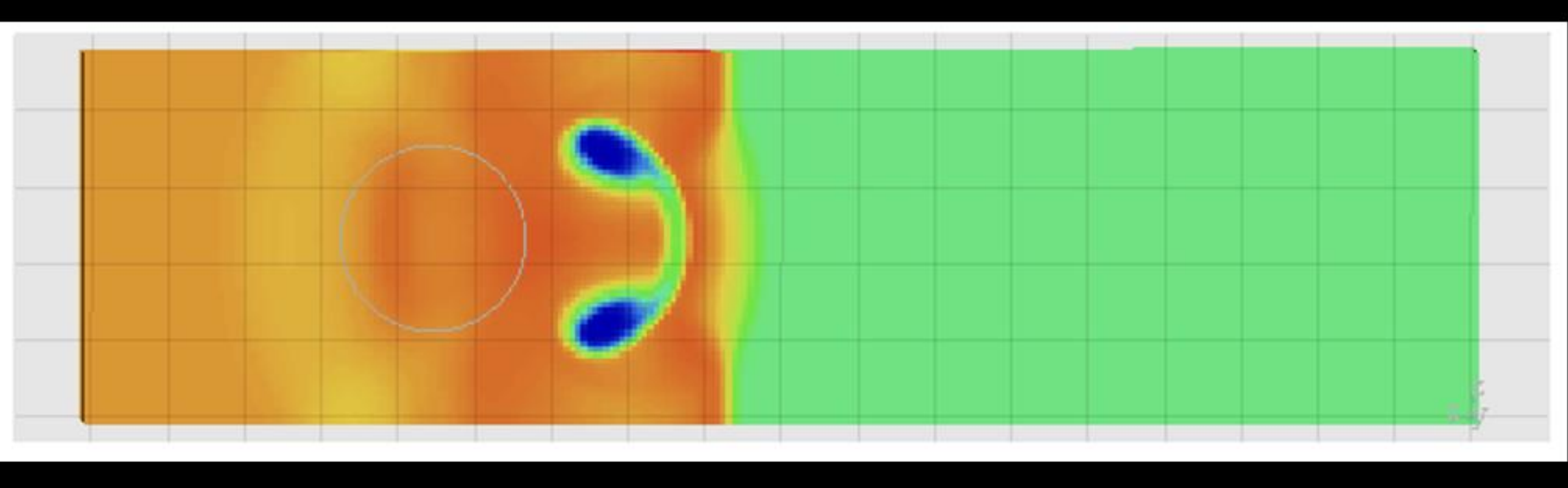
Application Performance



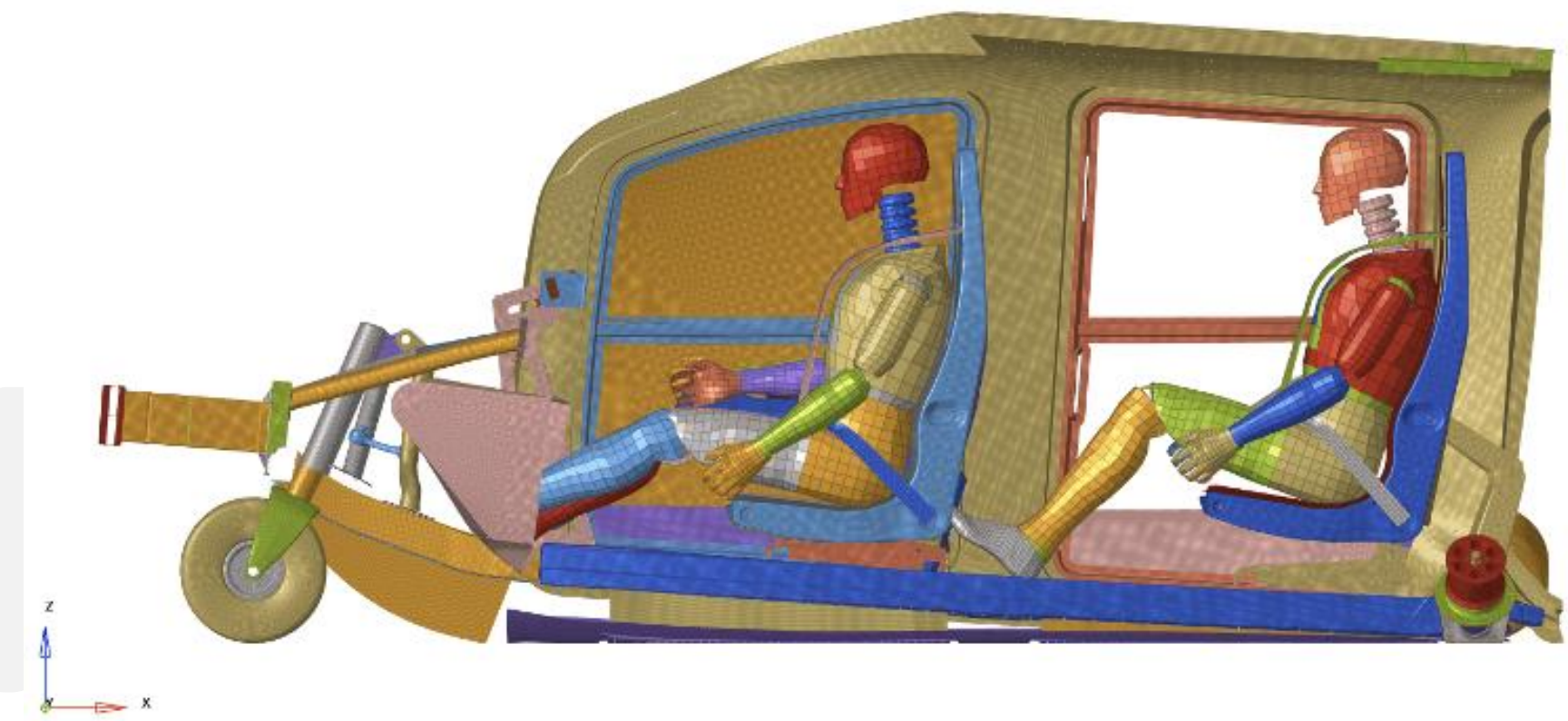
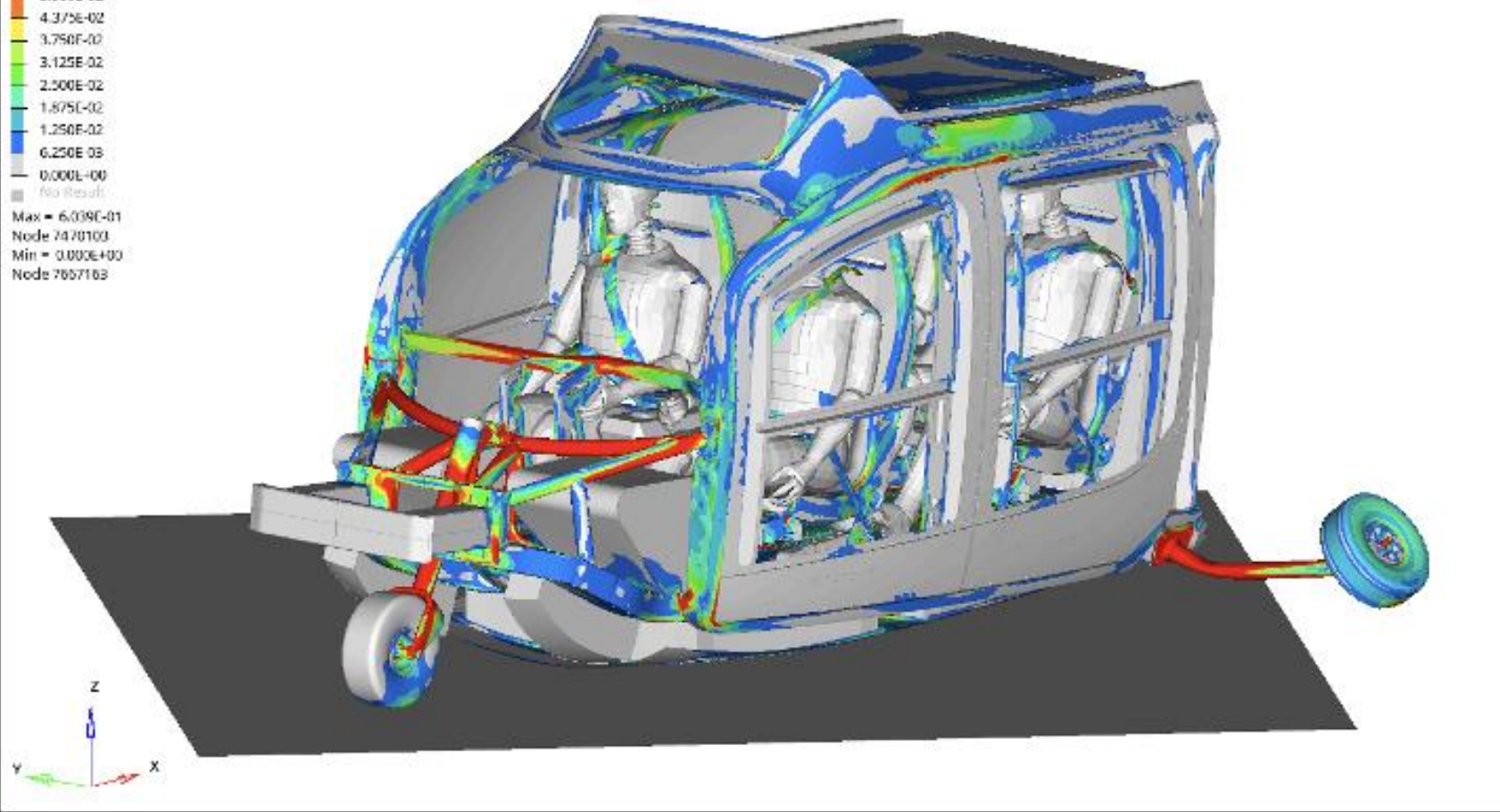
Energy Efficiency



Contour Plot  
Density (Scalar value, Mid)  
2.738E+00  
2.464E+00  
2.190E+00  
1.915E+00  
1.641E+00  
1.367E+00  
1.092E+00  
8.179E-01  
5.436E-01  
2.692E-01  
No result  
Local Max = 2.738E+00  
Solid 1119940  
Local Min = 2.692E-01  
Solid 384067



Contour Plot  
VonMises (Scalar value, Mid)  
Simple Average  
1.000E-01  
5.000E-02  
4.375E-02  
3.750E-02  
3.125E-02  
2.500E-02  
1.875E-02  
1.250E-02  
6.25E-03  
0.000E+00  
No result  
Max = 6.000E-01  
Node 18.00103  
Min = 0.000E+00  
Node 7007183





The background features a series of overlapping, diagonal, light green rectangular shapes that create a sense of depth and movement. The shapes are arranged in a way that they appear to be receding into the distance, with the top-left shape being the most prominent and the bottom-right shape being the least. The overall color palette is a range of light greens, from pale yellow-green to a slightly darker, more saturated green.

# **NVIDIA Grace Superchips**



# The NVIDIA Grace CPU

The building block of the superchip

## High Performance Power Efficient Cores

72 flagship Arm Neoverse V2 Cores with  
SVE2 4x128b SIMD per core

**3.5 FP64 TFLOP/s TPeak**

## Fast On-Chip Fabric

**3.2 TB/s of bisection bandwidth** connects  
CPU cores, NVLink-C2C, memory, and system IO

## High-Bandwidth Low-Power Memory

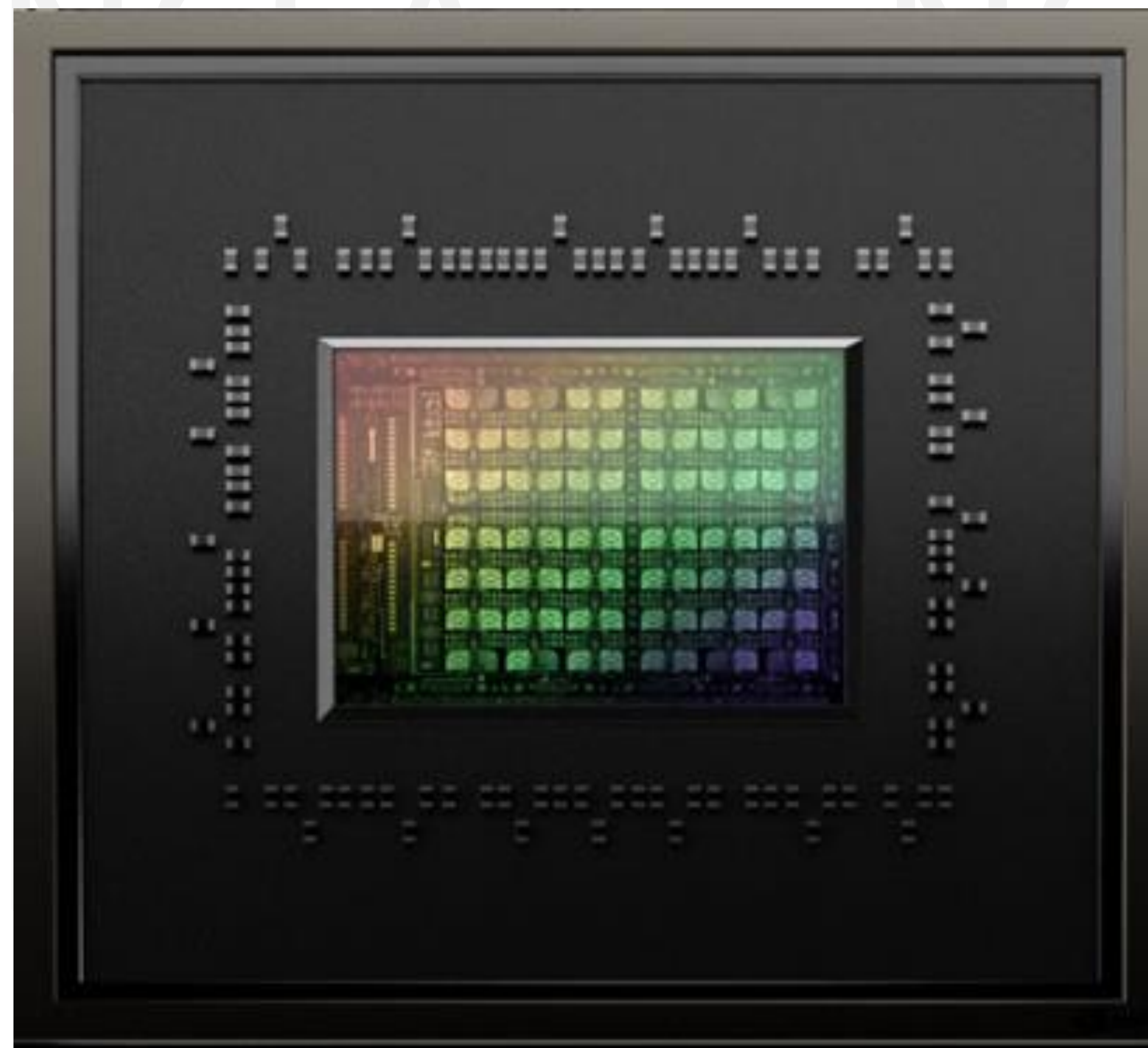
Up to 480 GB of data center enhanced LPDDR5X Memory that  
delivers **up to 500 GB/s of memory bandwidth**

## Coherent Chip-to-Chip Connections

NVLink-C2C with **900 GB/s bandwidth** for coherent  
connection to CPU or GPU

## Industry Leading Performance Per Watt

Up to 2X perf / W over today's leading servers

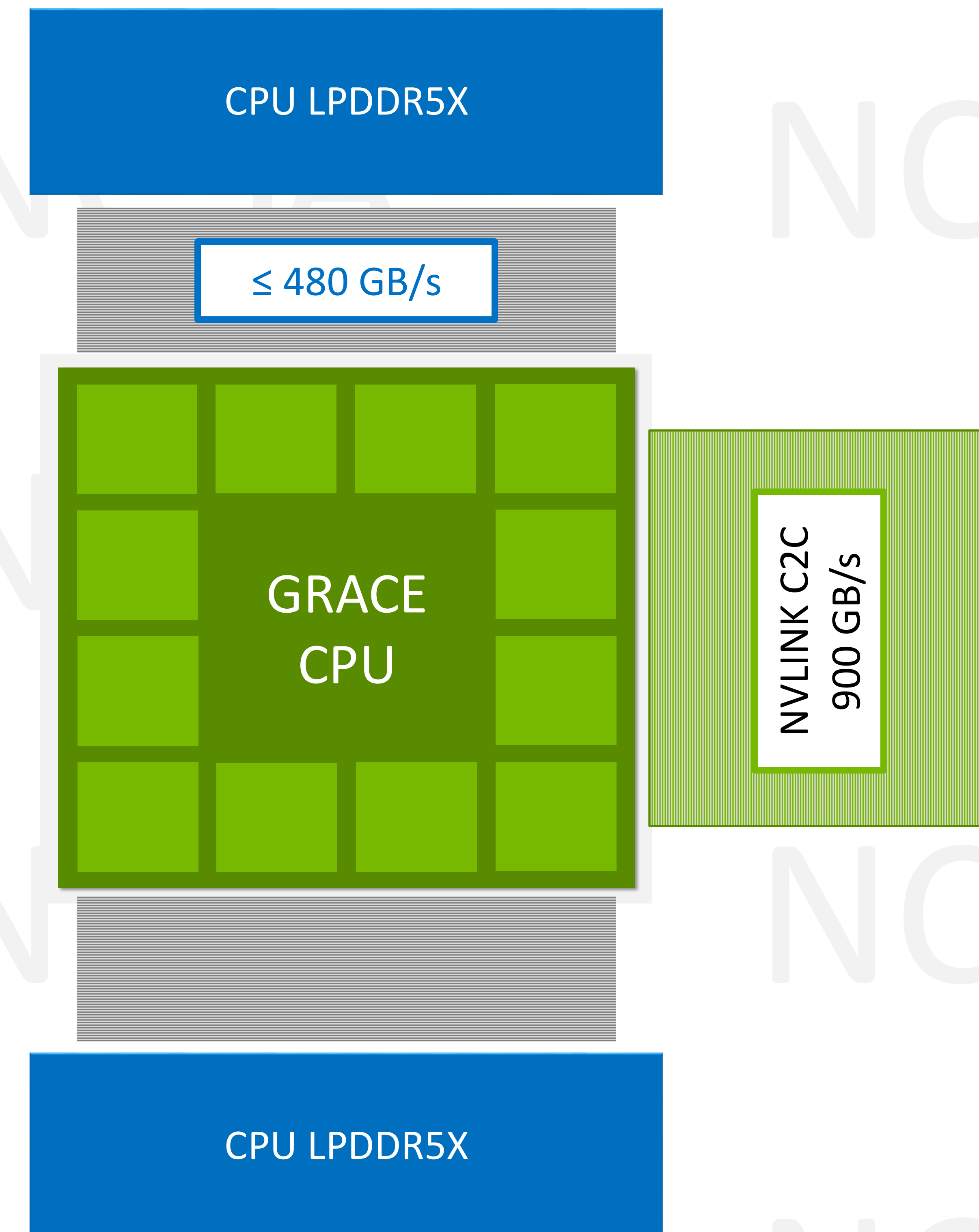




# NVLINK-C2C

High Speed Chip to Chip Interconnect

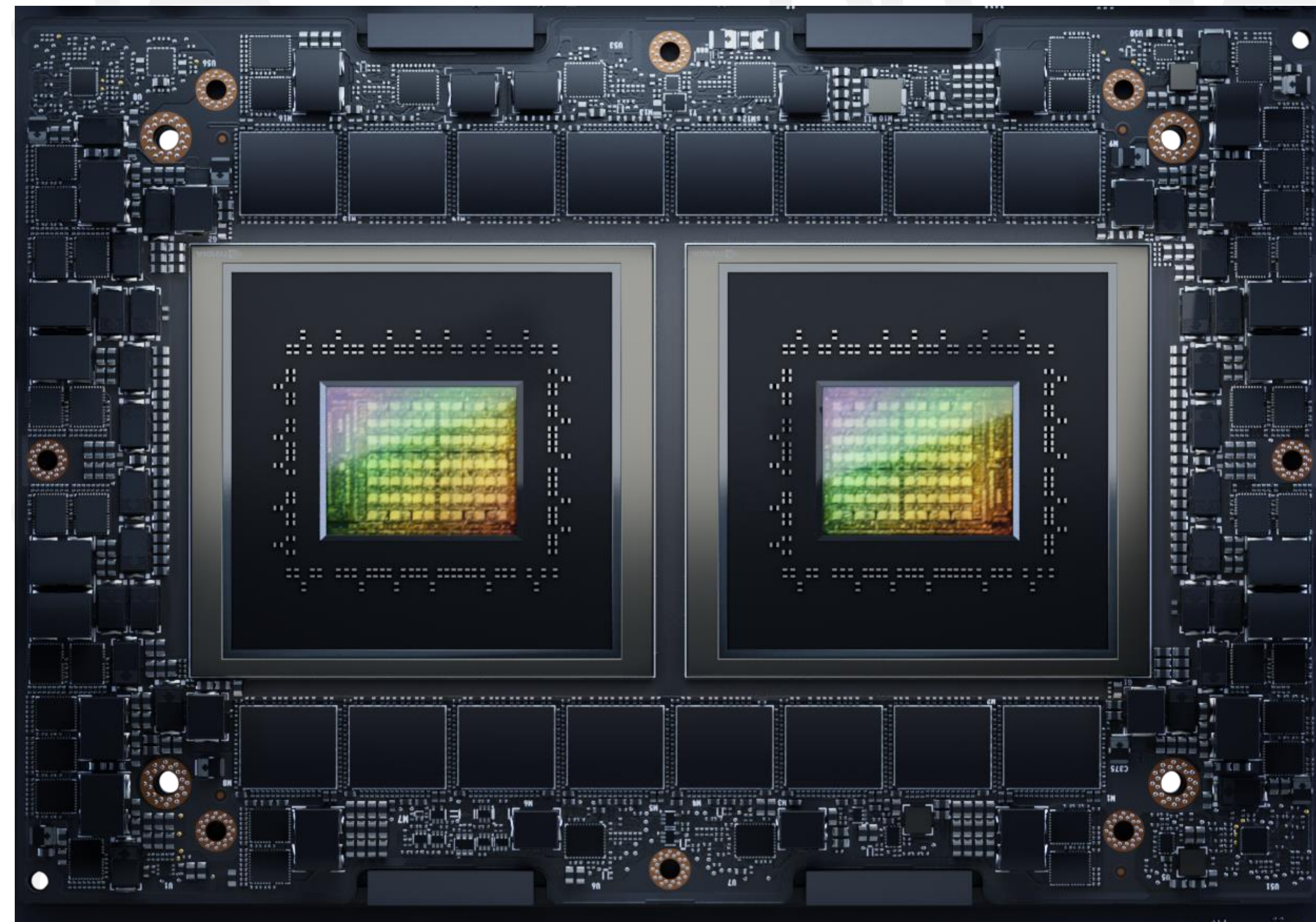
- Grace Hopper and Grace Superchips
- Removes the typical cross-socket bottlenecks
- Up to 900GB/s of raw bidirectional BW
  - Same BW as GPU to GPU NVLINK on Hopper
- Low power interface - 1.3 pJ/bit
  - More than 5x more power efficient than PCIe
- Enables coherency for both Grace and Grace Hopper superchips





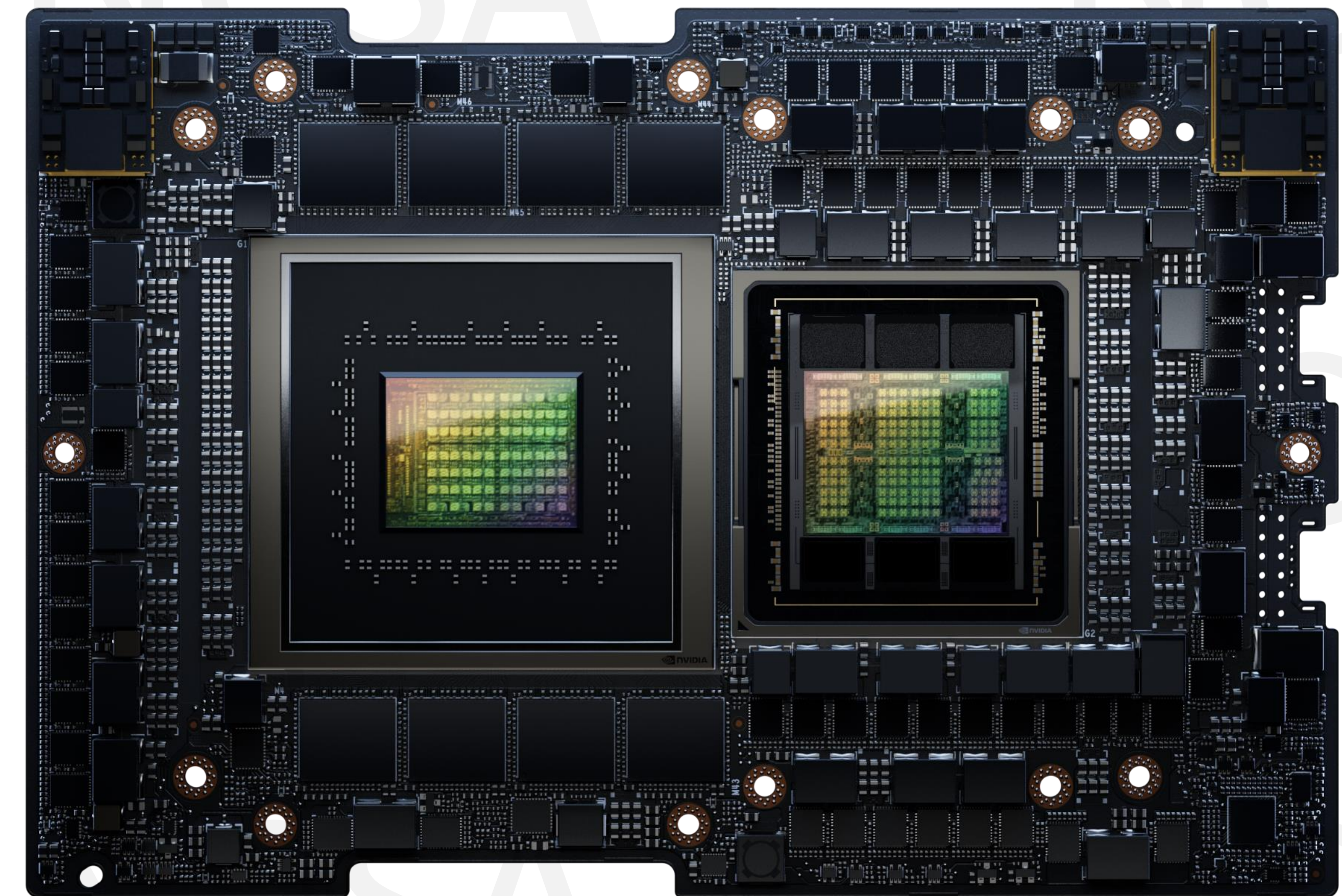
# One Powerful CPU – Two Superchip Configurations

Grace CPU Superchip  
CPU Computing



More than “2x Grace”

GH200 Grace Hopper Superchip  
Large Scale AI & HPC



More than “Grace + Hopper”





# NVIDIA Grace Hopper Superchip

Soul is the new **NVLink-C2C** CPU  $\leftrightarrow$  GPU interconnect

- **Memory coherency**: ease of use
  - All threads – GPU and CPU – access system memory: C++ new, malloc, mmap'ed files, atomics, ...
  - Fast automatic page migrations HBM3  $\leftrightarrow$  LPDDR5X.
  - Threads cache peer memory → Less migrations.
- **High-bandwidth**: 900 GB/s (same as peer NVLink 4)
  - GPU reads or writes local/peer LPDDR5X at ~peak BW
- **Low-latency**: GPU → HBM latency
  - GPU reads or writes LPDDR5X at ~HBM3 latency

For all threads in the system  
**memory is memory**  
expected behavior + latency + bandwidth.



# The Grace Hopper Advantage for Developers

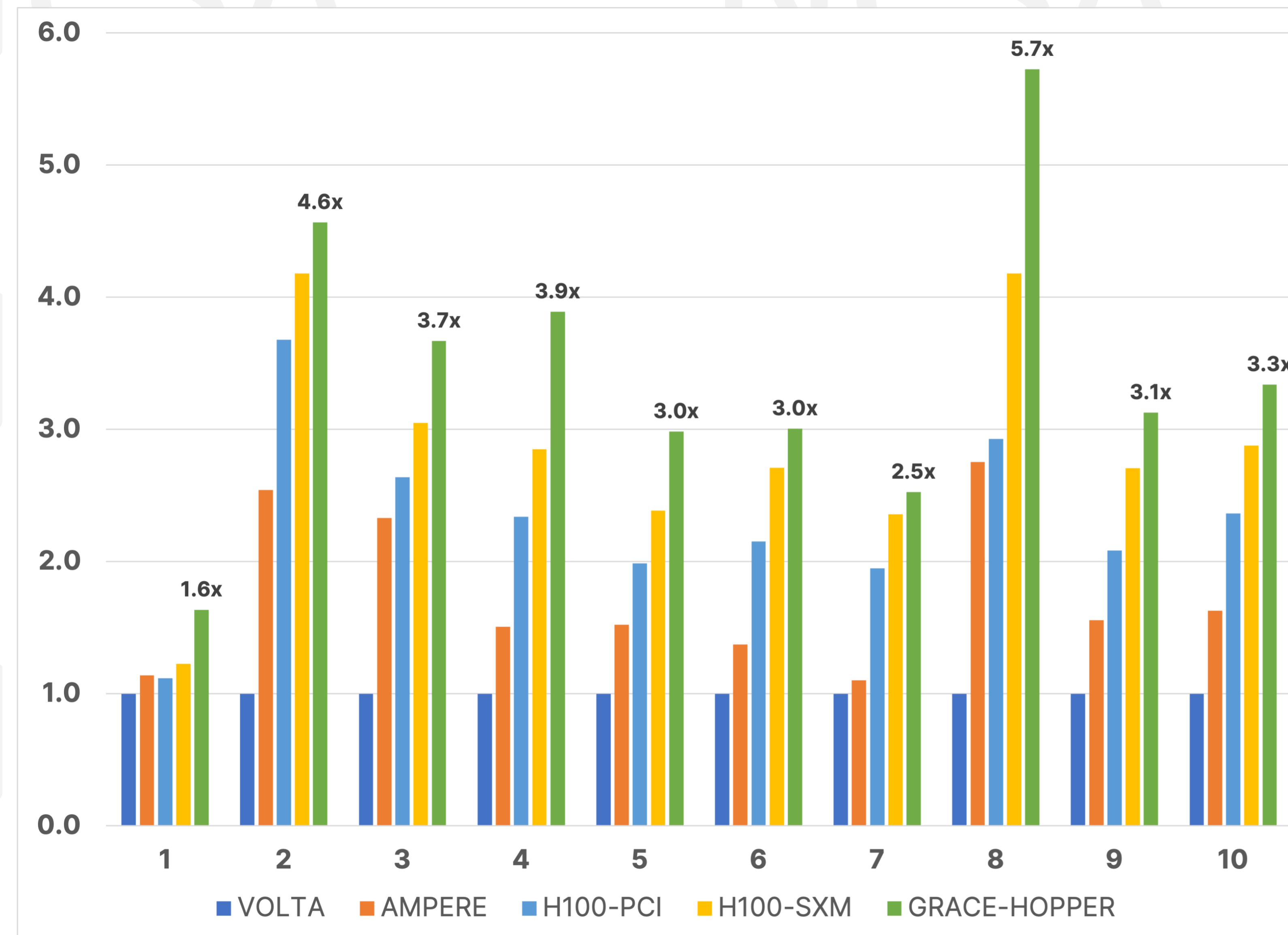
- Existing GPU applications require no changes for Grace Hopper
  - No new APIs
  - No restructuring
  - No new programming model
  - Developers who choose to can optimize for the Grace Hopper platform
- Existing GPU applications (fully or partially ported) will run better on Grace Hopper
  - Data migration no longer required, may still be a performance optimization
  - When data migrations happen, they happen faster due to C2C interconnect
  - CPU code will benefit from higher bandwidth memory, high thread performance, coherent accesses
  - Existing, stable Unified Memory APIs may be used for performance optimization
- Non-GPU applications will run unmodified and benefit from Grace architecture
- Porting from CPU to GPU is made simpler by Grace Hopper
  - Coherent Memory Subsystem
  - C2C interconnect
  - Programming model choice
- Some new capabilities may be unlocked
  - Larger data sets
  - Workflows that utilize both halves





# Stone Ridge Technology “Leap Ahead with Hopper”

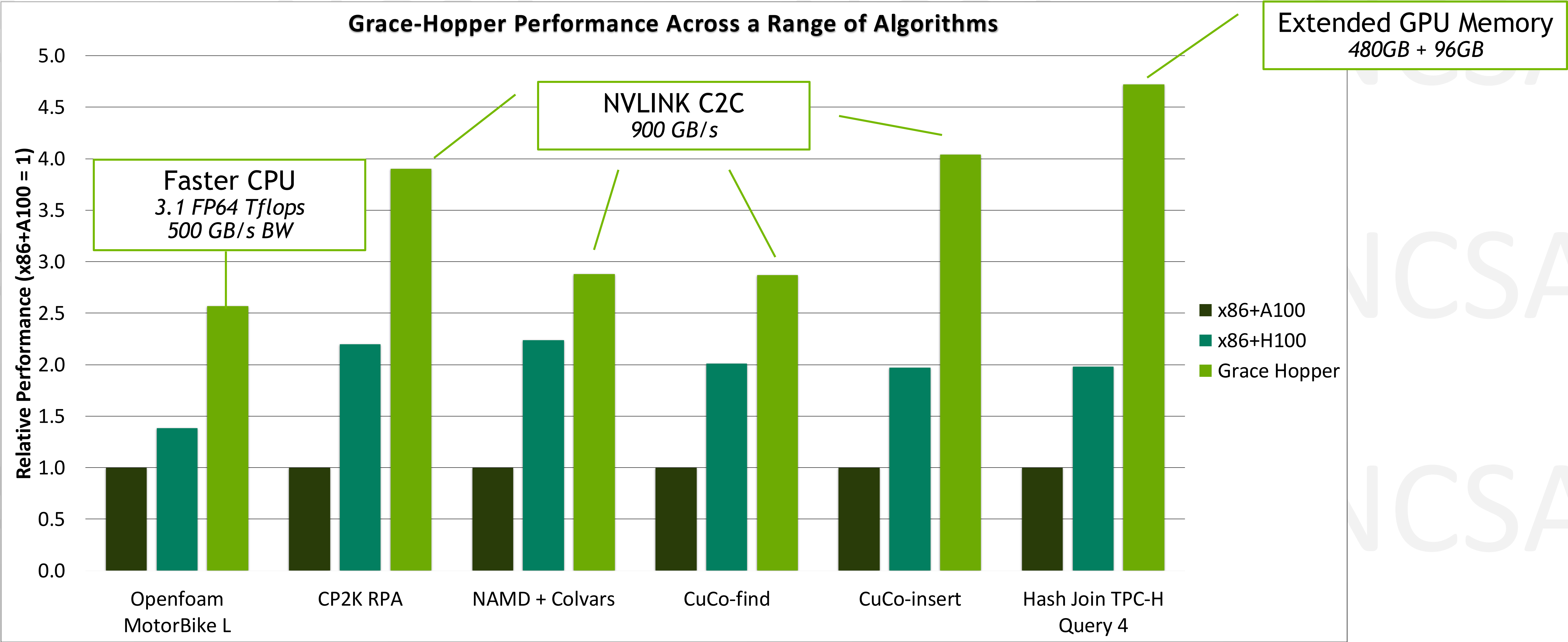
<https://stoneridgetechnology.com/company/blog/leap-ahead-with-hopper-2/>



- For ECHELON, **rebuilding was a trivial exercise**, and the resulting binary “just worked” on the Grace Hopper Superchip with no further tweaking required
- The performance gains were realized with **no modifications to the code**
- The average performance gain is **3.45x ± 1.07**
- The GPU portion of the code is performing so rapidly that whatever remains on CPU may be starting to illustrate Amdahl’s law behavior
- It is also possible that the improved performance on Grace Hopper is due to the **increased bandwidth between GPU and CPU**
- Further optimization for the Grace Hopper system may provide more gains



# Grace-Hopper Superchip Workload Performance





# Grace CPU Superchip

Grace CPU Superchip	
Architecture	Arm Neoverse V2 Armv9.0-A, <b>SVE2 4x128b SIMD</b>
Cores / Speed	144 cores, up to 3.2GHz
Memory	LPDDR5x soldered down, 1TB/s BW Up to 480GB per superchip
Cache	L1: 64KB i\$ + 64KB d\$ per core L2: 1MB per core L3: 234MB per superchip
Power	<b>500W including LPDDR5x memory</b>
Interfaces	Up to 8x PCIe Gen5 x16 HS interface
Process Node	TSMC 4N
Availability	Q3 2023

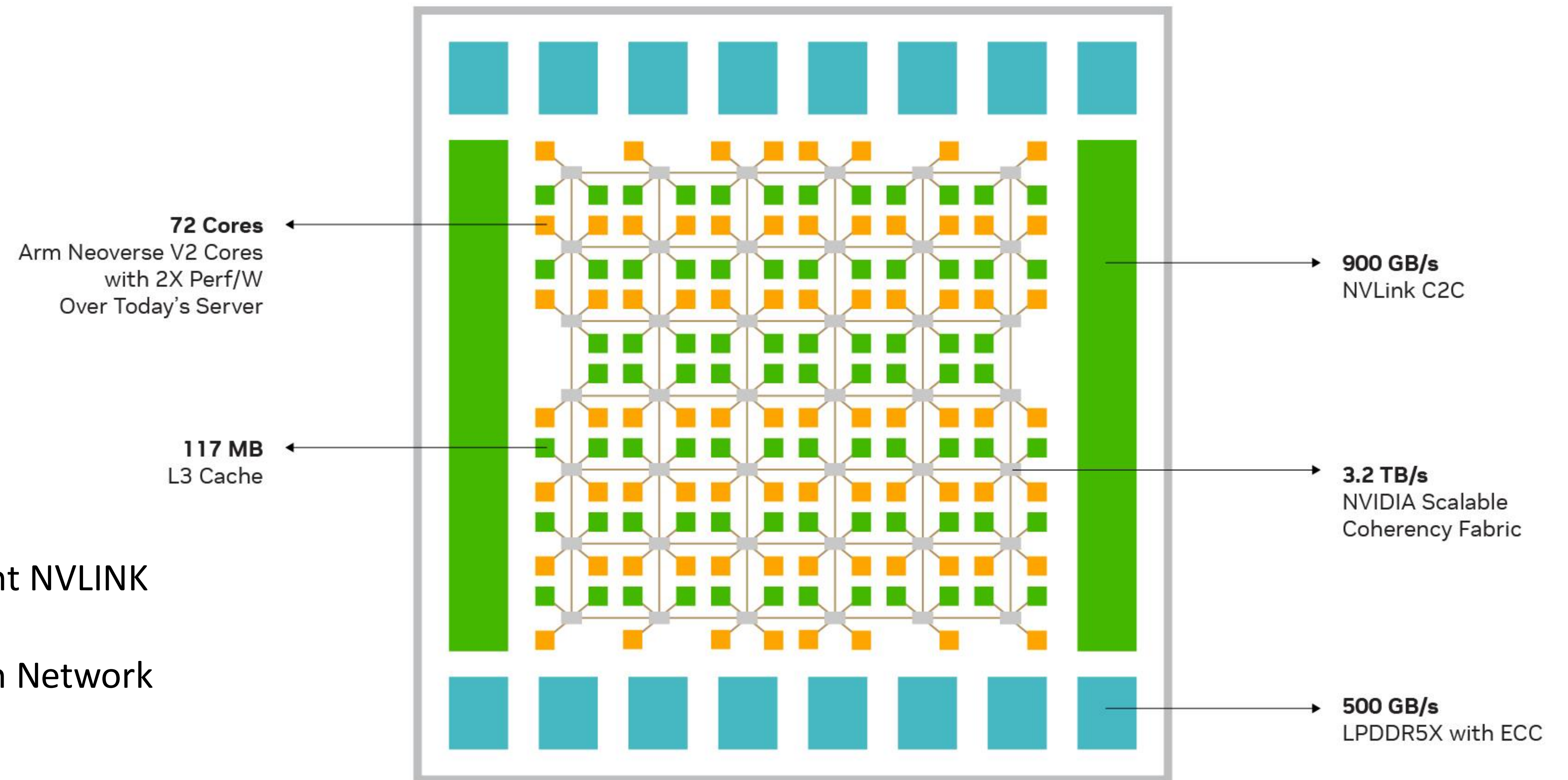




# Grace is a Compute and Data Movement Architecture

NVIDIA Scalable Coherency Fabric (SCF) and distributed cache design

- **Single Die:** More efficient use of power
- 3,225.6 GB/s Bi-section BW
- 117MB of L3 cache
- Scalable to 72+ cores per die
- Local caching of remote die memory
- Supports up to 4-die coherency over Coherent NVLINK
- Background data movement via Cache Switch Network



Example possible fabric topology for illustrative purposes



# Grace Simplifies System Design and Workload Optimization

A high-performance server on a single superchip package

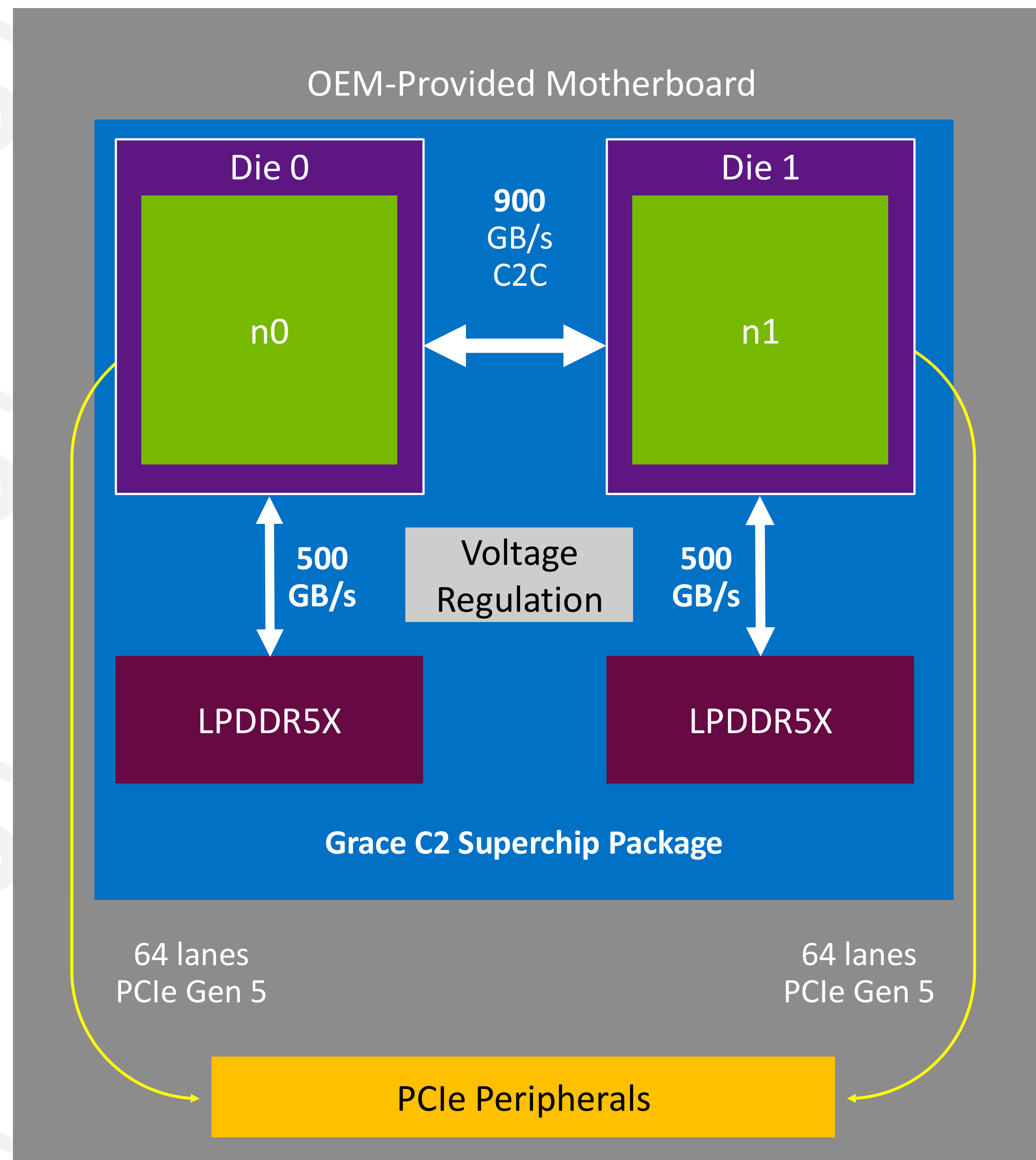
Grace Server  
Single Grace C2 Superchip

2  
NUMA Nodes

2  
Compute  
Dies

500  
Watts  
(CPU + MEM)

900  
GB/s worst-  
case n to n



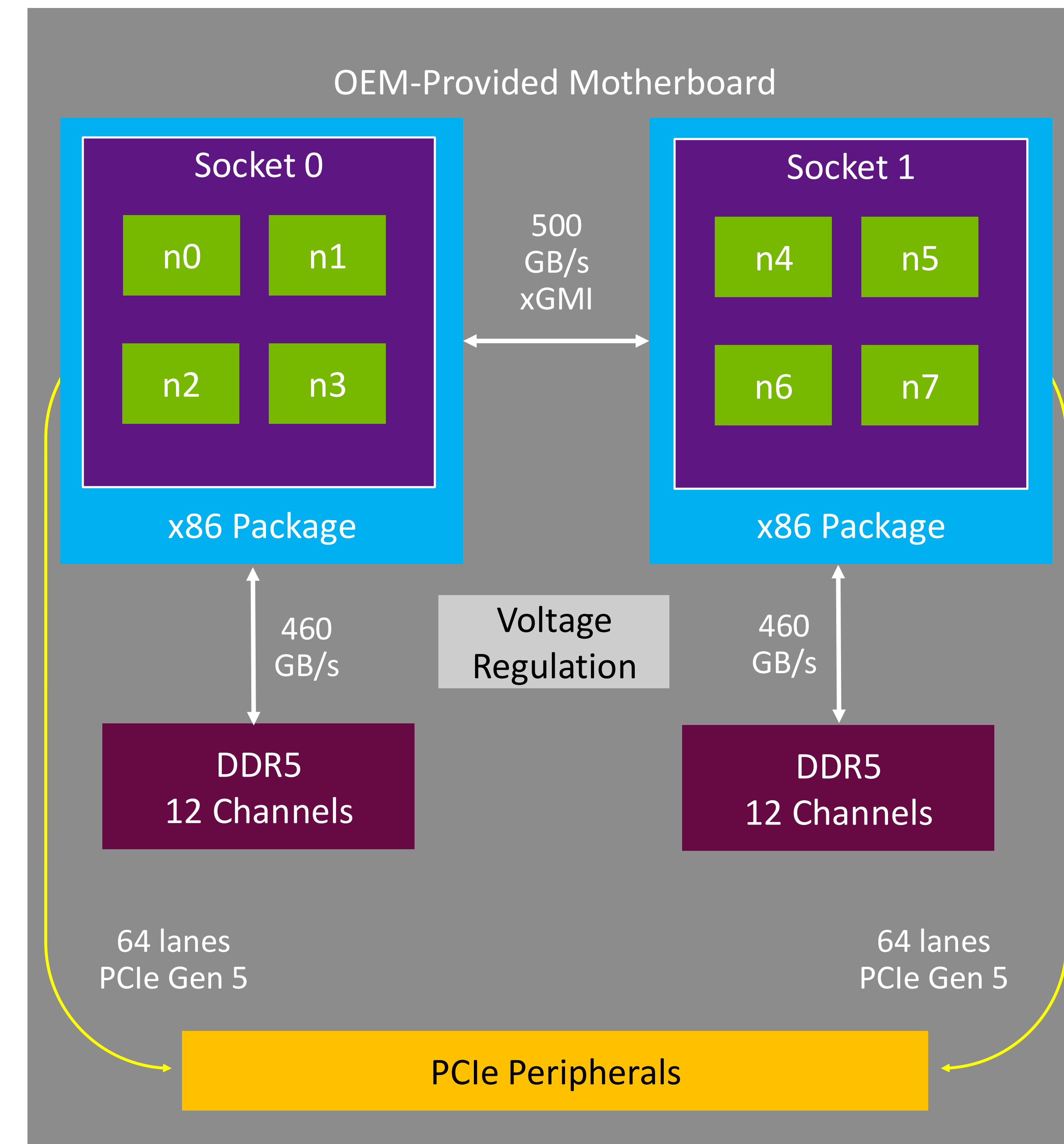
Conventional 2-Socket Server  
Dual-socket x86, NPS=4

8  
NUMA Nodes

24  
Compute  
Chiplets

900+  
Watts  
(CPU + MEM)

500  
GB/s worst-  
case n to n

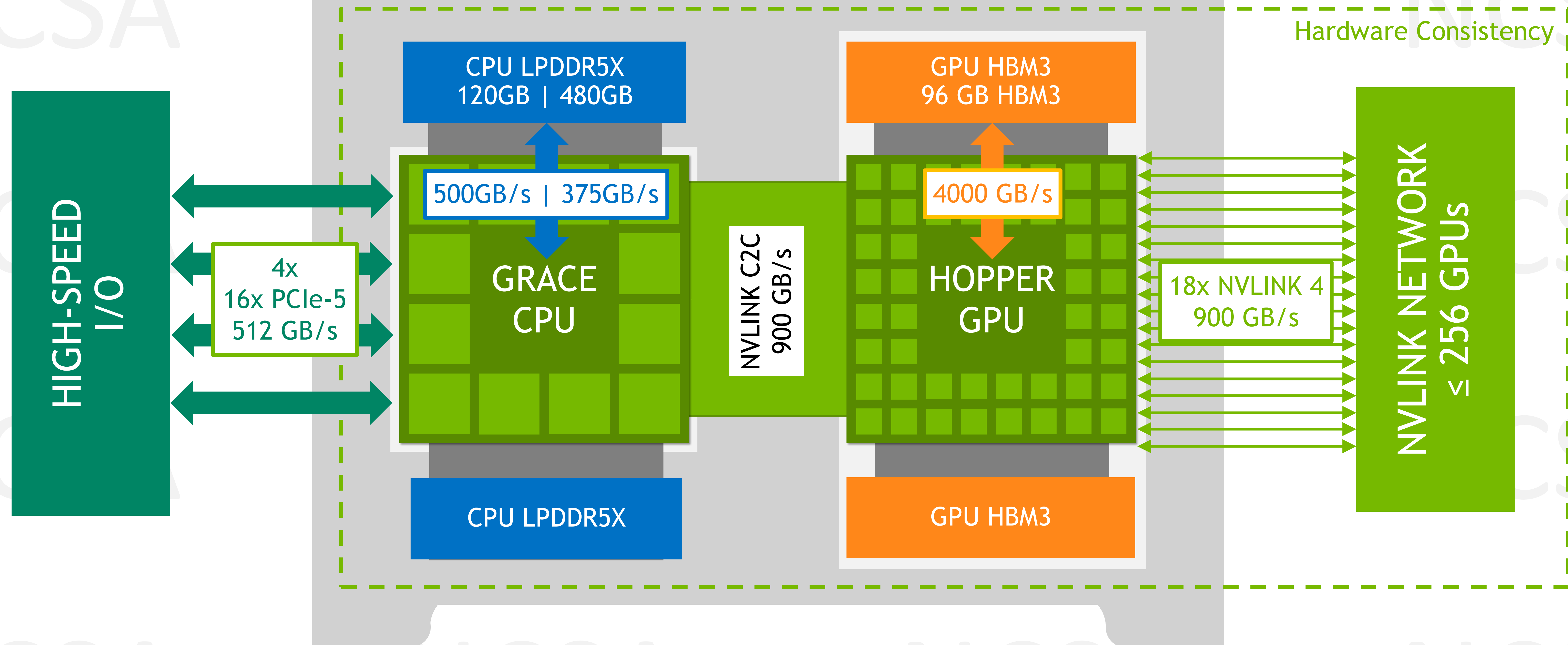




# Grace Hopper Superchip

GPU can access CPU memory at CPU memory speeds

## NVIDIA Grace Hopper Superchip





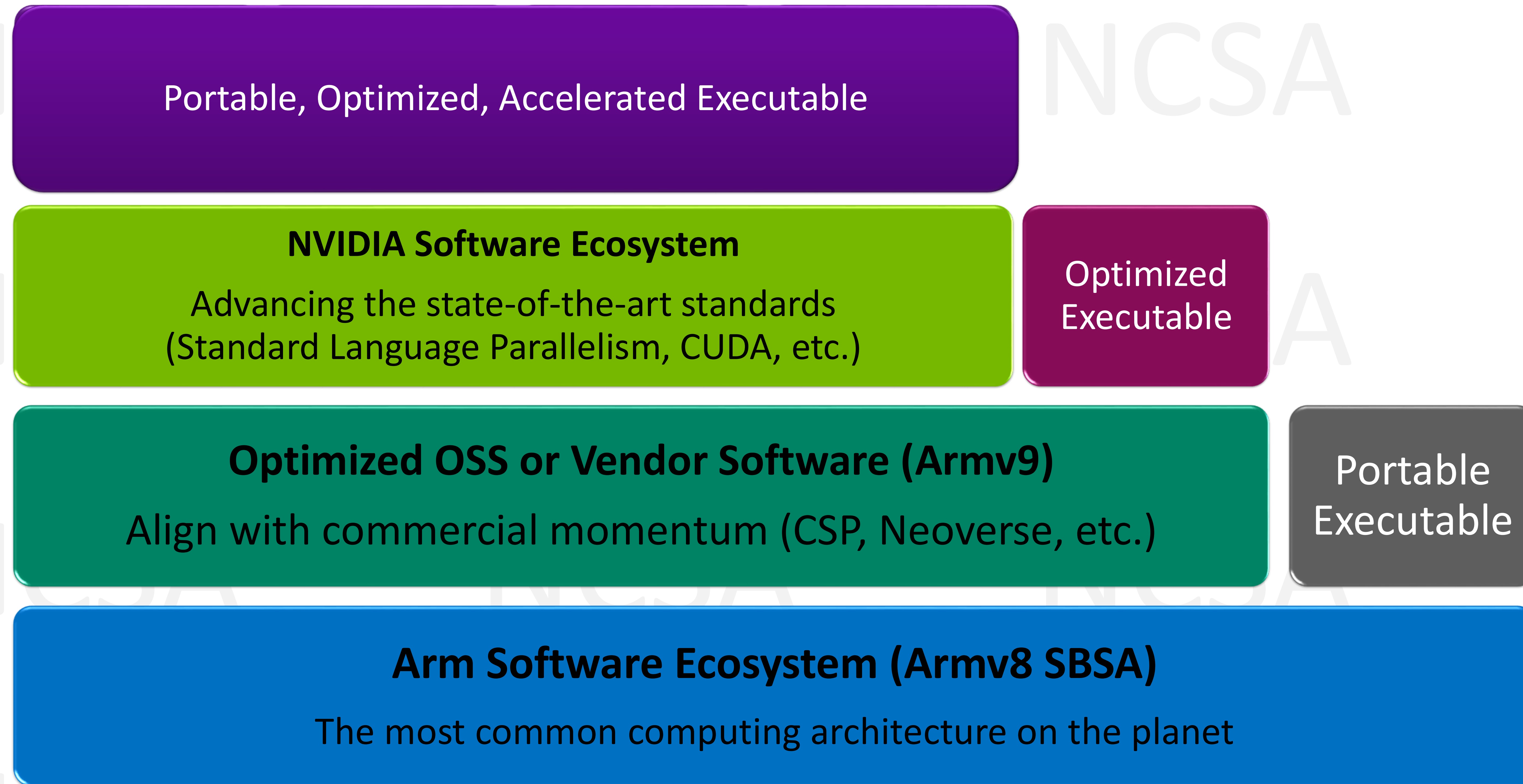


# Programming the NVIDIA Platform



# The Grace Software Ecosystem is Built on Standards

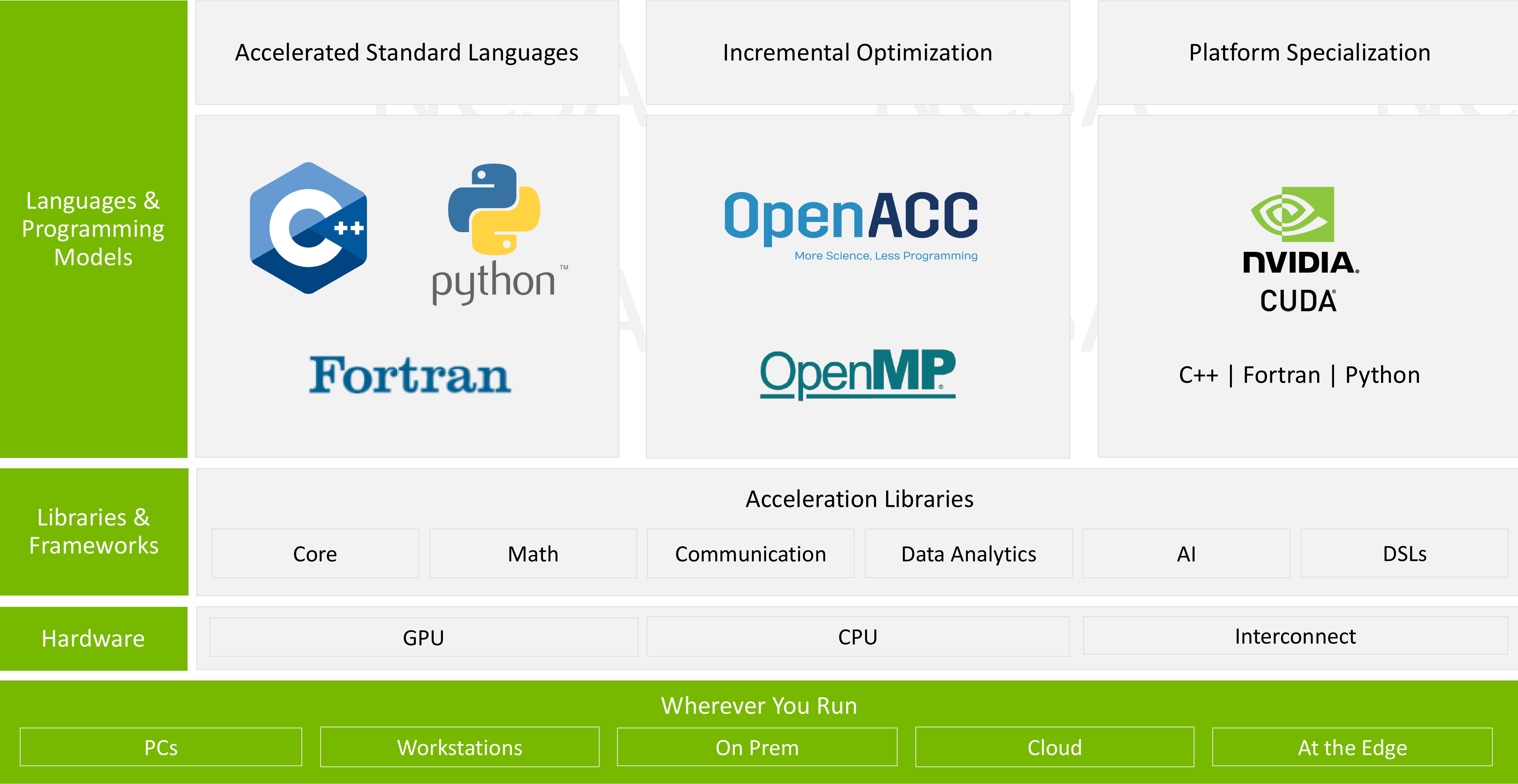
The NVIDIA platform builds on optimized software from the broad Arm software ecosystem





# Programming the NVIDIA Platform

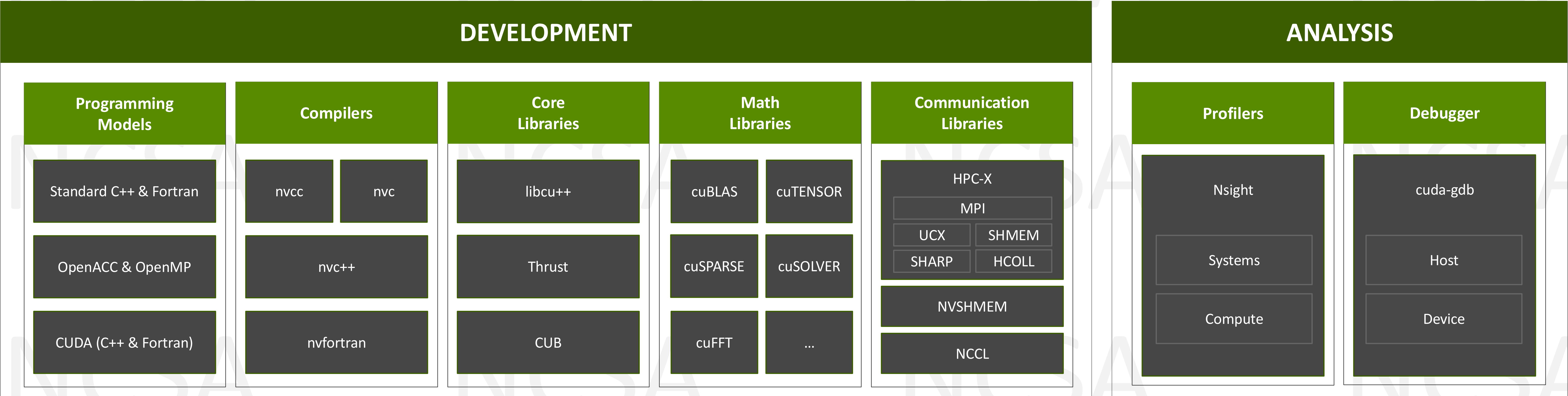
Unmatched Developer Flexibility





# NVIDIA HPC SDK

Available at [developer.nvidia.com/hpc-sdk](https://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  
x86\_64 | Arm  
6 Releases Per Year | Freely Available



# HPC SDK Updates

Grace Hopper, unified memory, and more

- **HPC SDK 23.11:**

- Unified memory support for stdpar, OpenACC, and CUDA C++/Fortran
- NVTX improvements for stdpar codes
  - Now you can see your stdpar in NSight: improved tools support, developer experience, performance optimizations
- C-Fortran Interface
  - Better multi-paradigm interoperability for mixed C, C++, and Fortran codes
  - F2008 MPI bindings for nvfortran
- C++20 Coroutines for CPU
  - Future GPU support will enable alternative async models for stdpar
- Support for Grace Hopper in all bundled components
  - Compilers, Math Libraries, Networking, Tools.
- HPC-X is the default MPI implementation optimized for NV platform
- Grace(/Arm) performance (-tp=neoverse-v2)
  - Re-engineered vectorizer, intrinsics, system math library functions

- **HPC SDK 24.3:**

- Improved compile speed for nvc++
  - Up to 1.15x - 2x faster for some workloads
- Unified memory support for OpenMP Target Offload
- Integrated NVIDIA Performance Library (NVPL) for Grace CPUs
- CUDA Fortran `unified` attribute

- **HPC SDK 24.5:**

- New NVPL integrations
- Ubuntu 24.04 support
- Improved memory model CLI for HPC Compilers

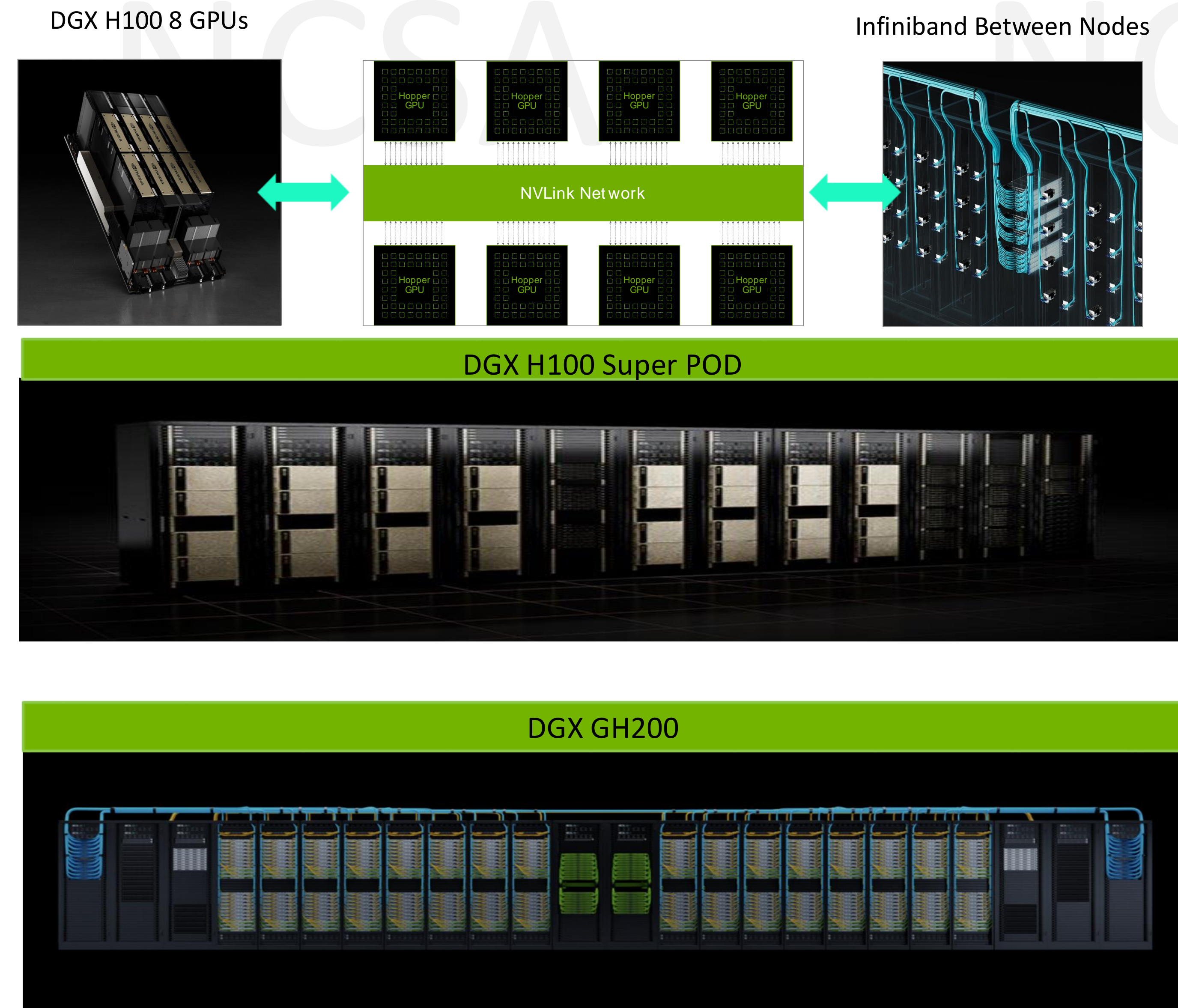
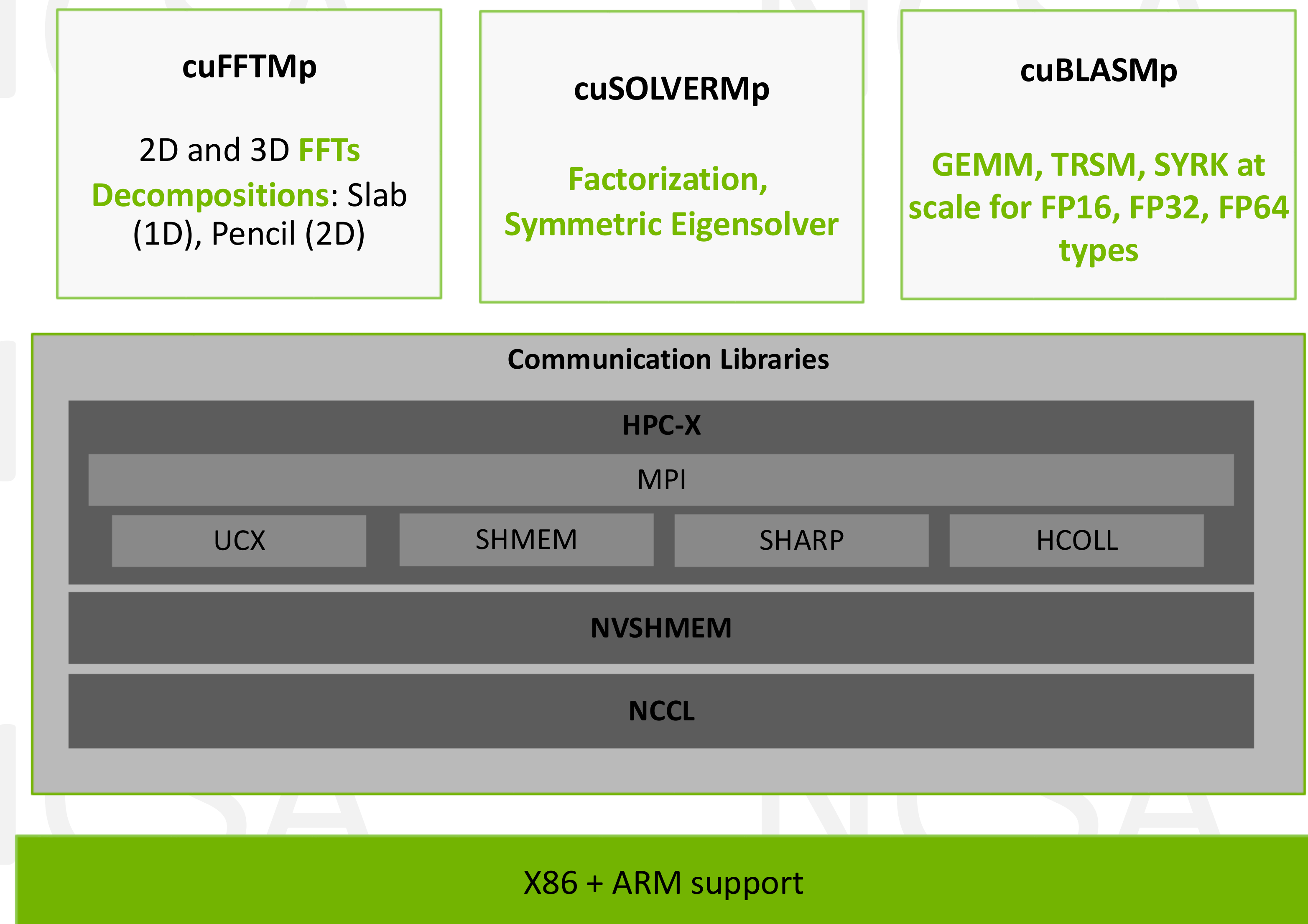
## Unified Memory

- C++ stdpar improvements
- Fortran stdpar improvements
- OpenACC improvements
- CUDA Fortran
- OpenMP Target Offload
- Unified Functions



# Multi GPU Multi Node APIs

Scalable and Grace Hopper Support

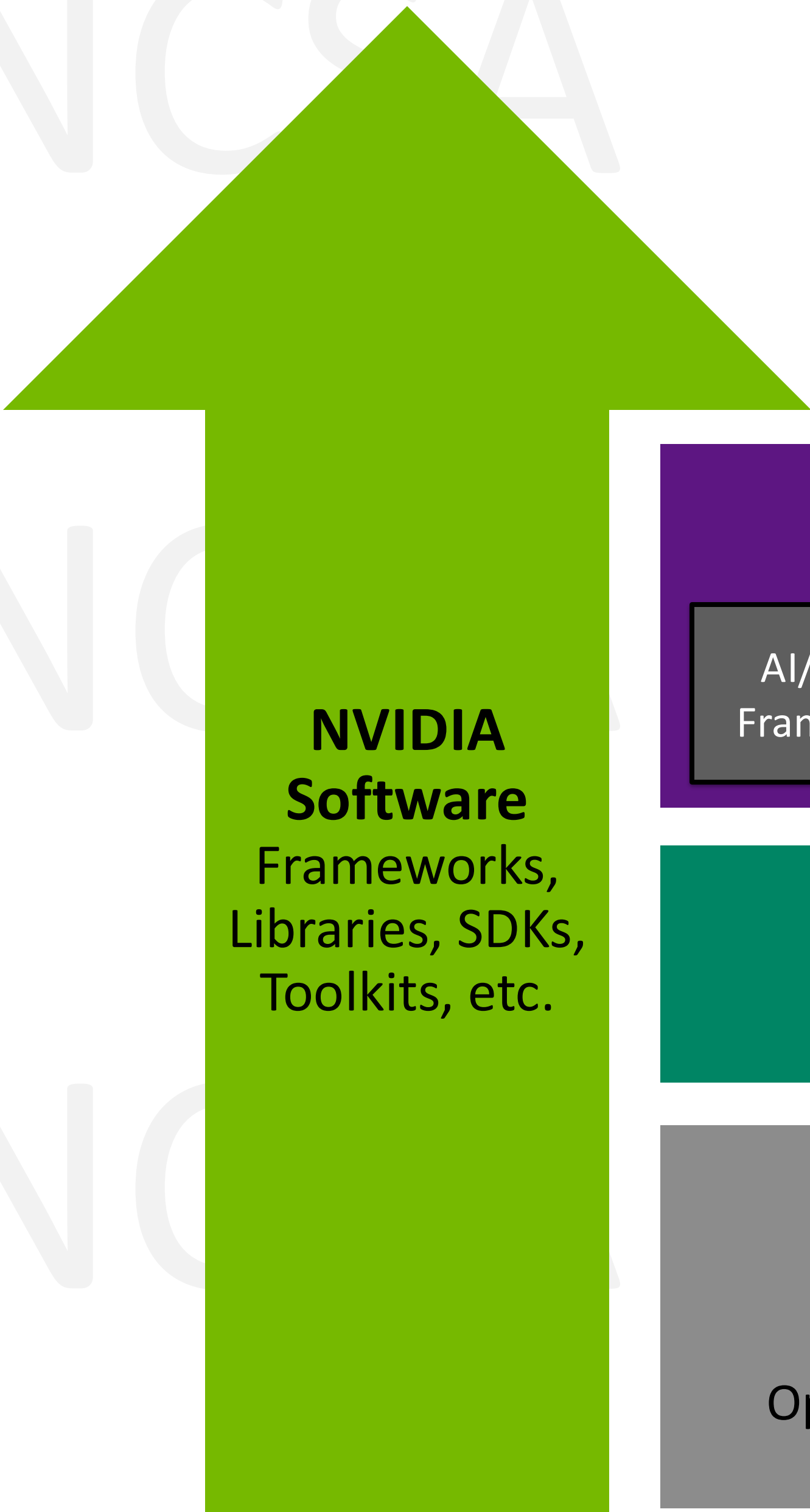


**256** Grace Hopper Superchips | **1EFLOPS** AI Performance | **144TB** unified fast memory  
| **900 GB/s** GPU-to-GPU bandwidth | **128 TB/s** bisection bandwidth



# HPC/AI Software Ecosystem for Grace

Developed by NVIDIA and the Arm OSS community



**NVIDIA Software**  
Frameworks, Libraries, SDKs, Toolkits, etc.

## Applications and Frameworks

- AI/ML/DL Frameworks
- HPC
- CSP
- Commercial ISV
- General Compute

## Developer Tools

NVIDIA Nsight +  
Open Source and Commercial

## Communication Libraries

HPC-X, OpenMPI, MPICH, MVAPICH2 + Open Source and Commercial

## Compilers

NVIDIA, GCC, LLVM +  
Open Source and Commercial

## Libraries

NVPL, ArmPL, FFTW, BLIS, OpenBLAS  
+  
Open Source and Commercial

## Filesystems

Lustre, BeeGFS,  
Spectrum Scale +  
Open Source and Commercial

## Admin Tools & Packaging

Spack, EasyBuild, Conda, PyPi + Open Source and Commercial

## OS

All Major Linux Distributions

## Arm ServerReady SR

Standard firmware and RAS

**Schedulers**  
Open Source and Commercial

**Cluster Management**  
NVIDIA Bright Cluster Manager + Open Source and Commercial

**Containers & Orchestration**  
NVIDIA NGC + Open Source and Commercial



# NVIDIA Performance Libraries (NVPL)

Optimized math libraries for NVIDIA CPUs

- Easily port applications to NVIDIA's Arm CPUs
- Drop-in replacement for any math library implementing standard interfaces (e.g. Netlib, FFTW)
- New interfaces for high-performance libraries

BLAS

LAPACK

PBLAS

SCALAPACK

TENSOR

SPARSE

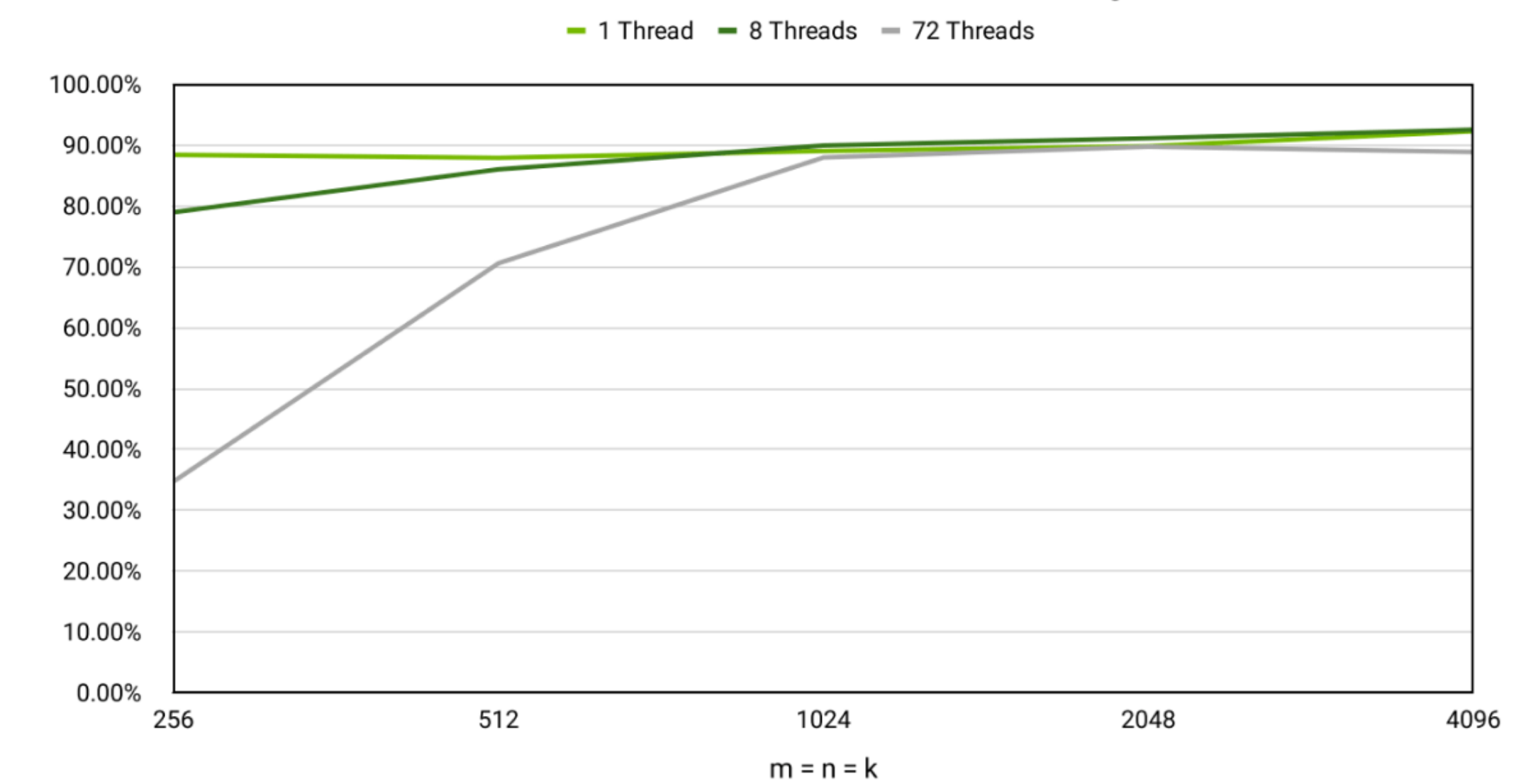
RAND

FFT

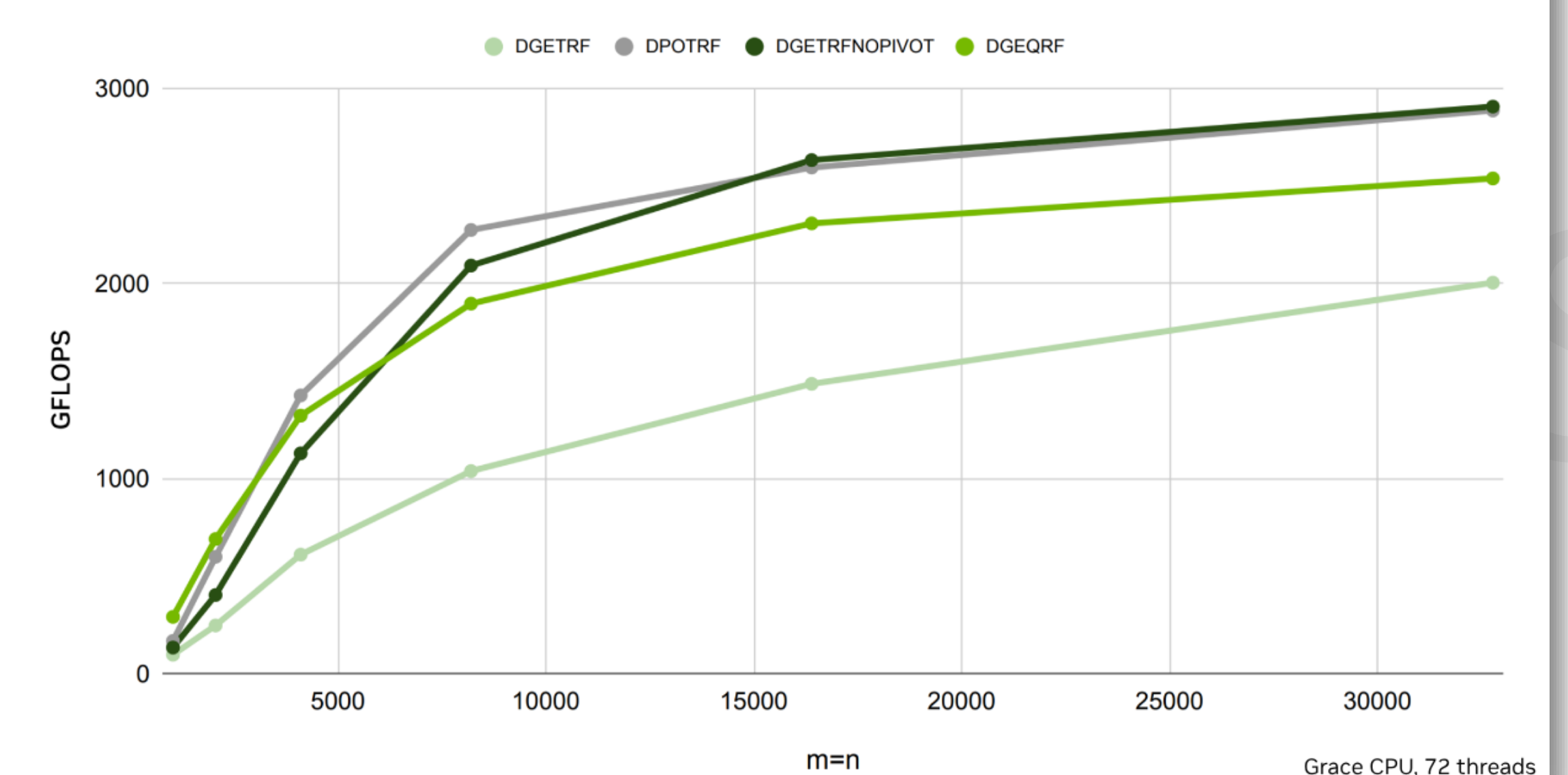
Download Now

[www.developer.nvidia.com/nvpl](http://www.developer.nvidia.com/nvpl)

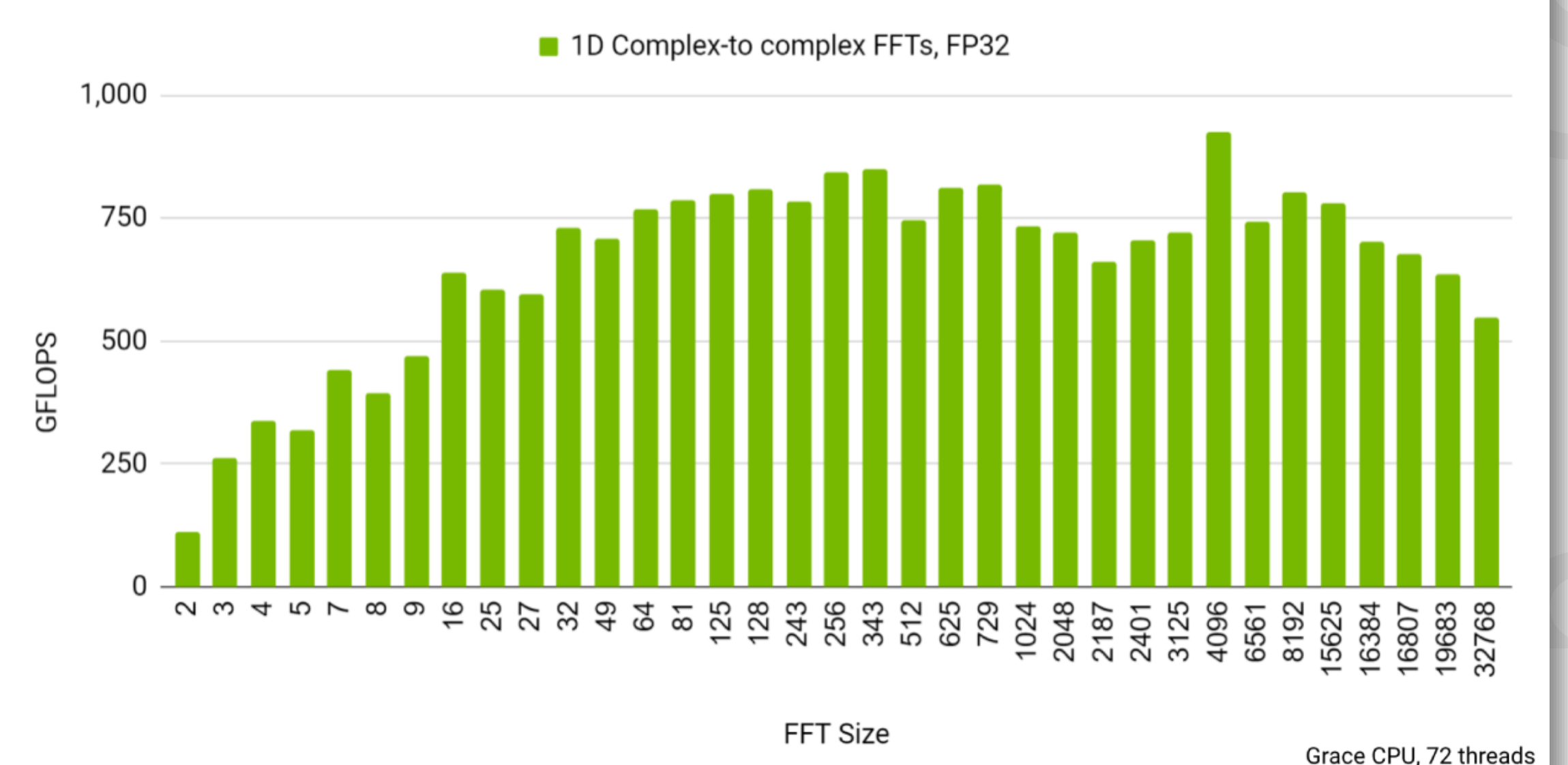
NVPL BLAS DGEMM Efficiency



NVPL LAPACK Performance



NVPL FFT Performance



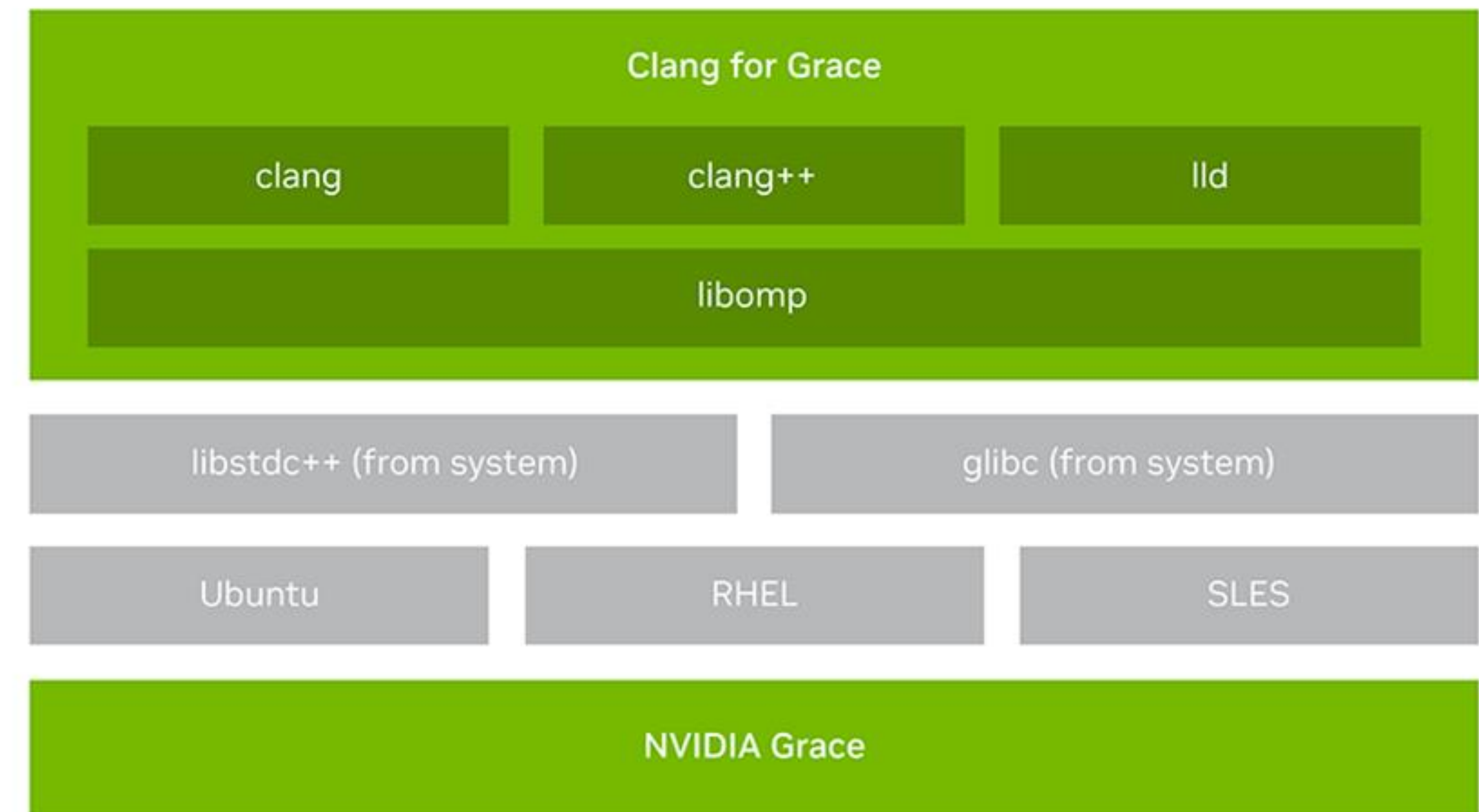


# Clang for NVIDIA Grace

An optimized build of LLVM Clang for the NVIDIA Grace CPU

- Optimized builds of the open-source LLVM Clang compiler for rapid access to the latest LLVM improvements for the Grace CPU
- Certified CUDA host compiler
- Optimized compile times: 15% faster vs. mainline LLVM
- Current release based on **LLVM 18.1.1**
  - C compiler driver binary - **clang**
  - C++ compiler driver binary - **clang++**
  - LLVM Linker - **lld**
  - OpenMP Runtime support - **libomp**
- [www.developer.nvidia.com/grace/clang](http://www.developer.nvidia.com/grace/clang)

Architecture	Linux Distributions	CUDA Toolkit
AArch64	<ul style="list-style-type: none"><li>• Ubuntu 22.04</li><li>• RHEL 9</li><li>• CentOS 9</li><li>• SLES 15-SP4</li></ul>	12.2U2 and later



Download Now

[www.developer.nvidia.com/grace/clang](http://www.developer.nvidia.com/grace/clang)



# Advancing the State-of-the-Art in Compilers

NVIDIA invests in open source and commercial compilers for NVIDIA Grace

- **NVIDIA HPC Compilers**

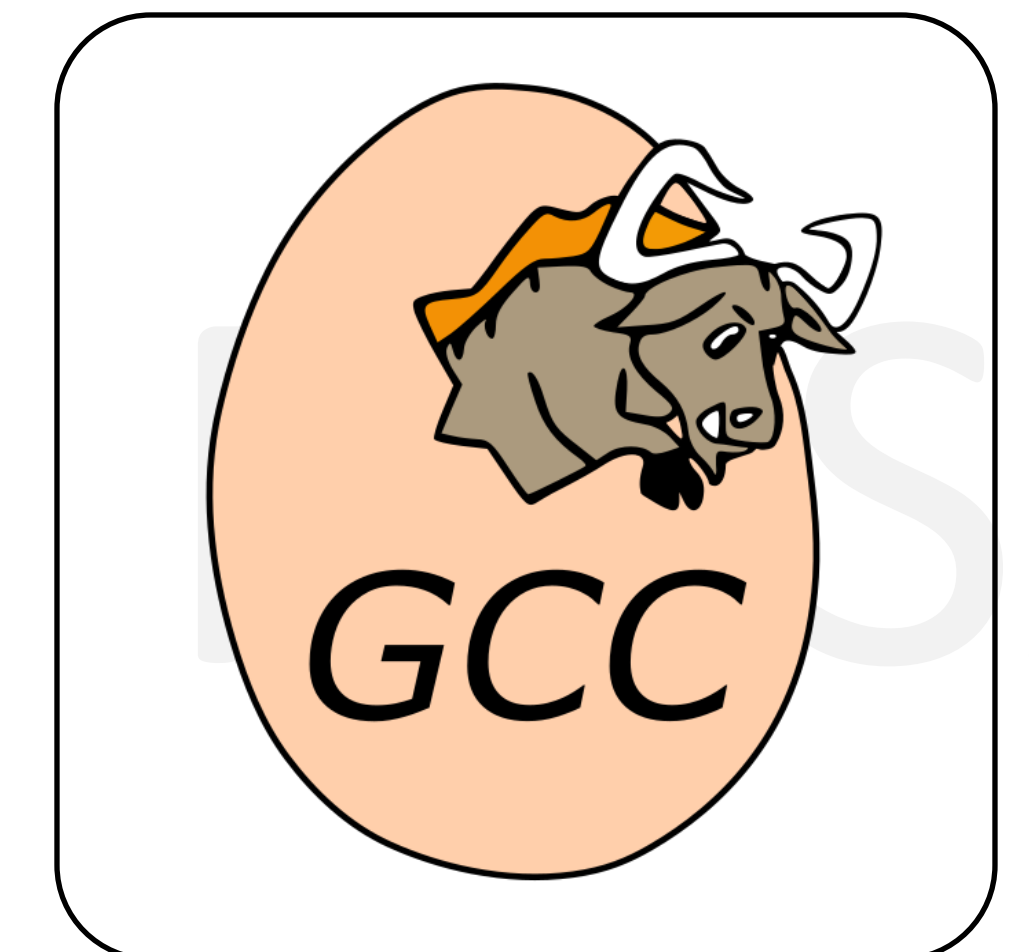
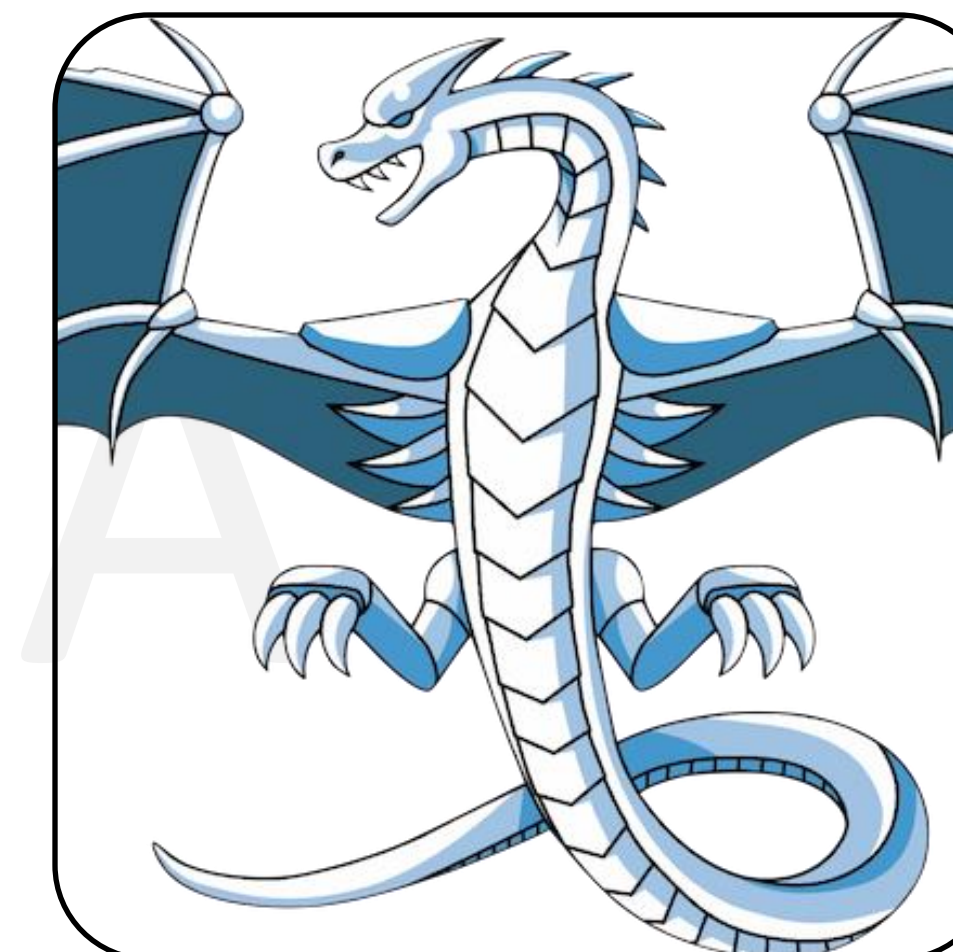
- Focused on application performance and programmer productivity
- High velocity, constant innovation
- Freely available with commercial support option

- **LLVM and Clang**

- NVIDIA provides builds of Clang for Grace
  - <https://developer.nvidia.com/grace/clang>
- Drop-in replacement for mainline Clang
- 100% of Clang enhancements for Grace are contributed to mainline LLVM

- **GCC**

- NVIDIA contributes to mainline GCC to support Grace
- Working with all major Linux distros to improve availability of Grace optimizations in GCC

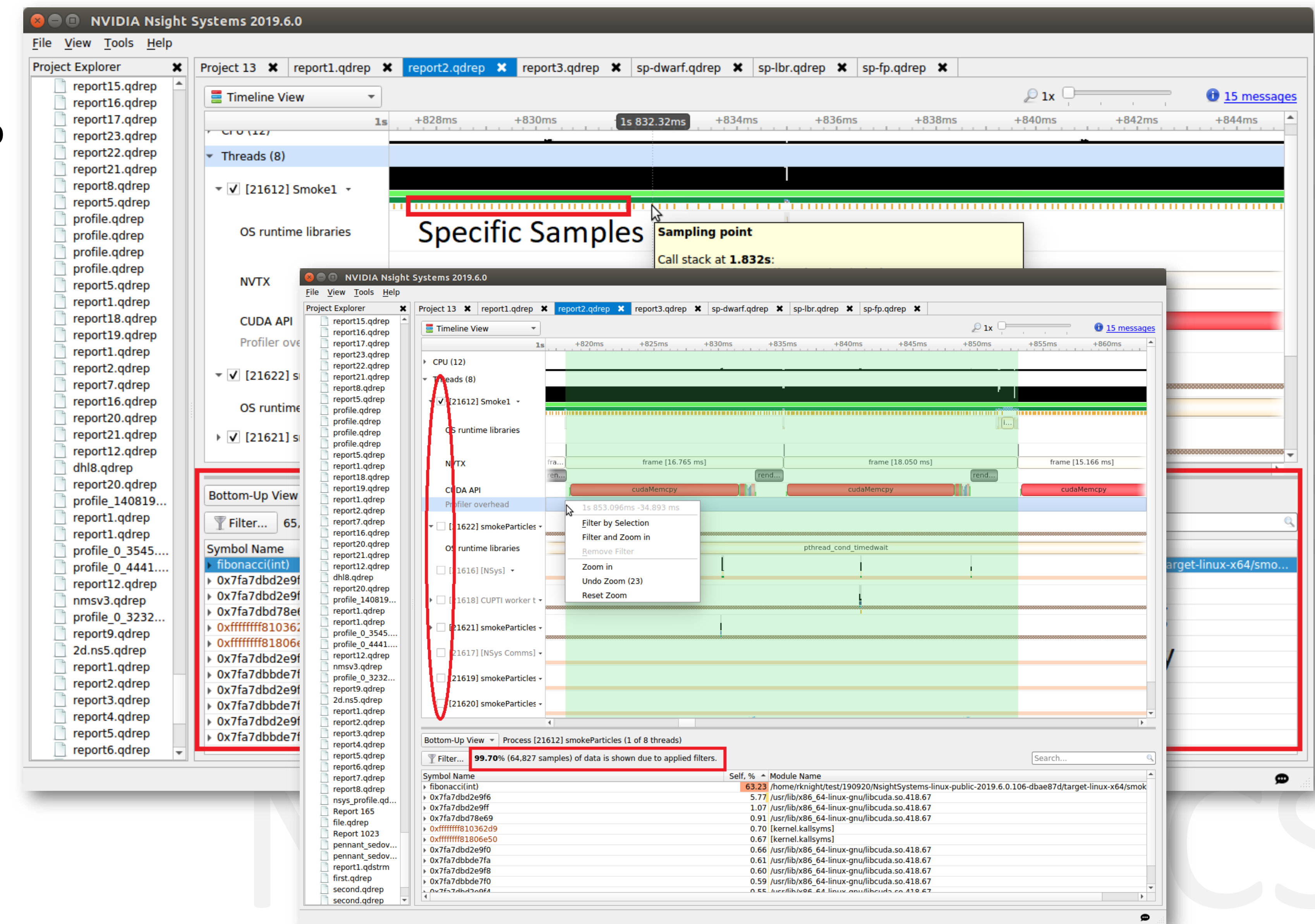




# Debuggers and Profilers for GH200 and Grace CPU Superchip

Full capability on Grace-Hopper

- **NVIDIA Nsight has full feature-parity on GH200**
  - Anything you can do with Nsight tools on x86+Hopper, you can do on GH200 with the same workflow
- **GH200 has hundreds of performance counters (PMUs)**
  - **Computational intensity, bandwidth, instruction mix...**
- **Generally, all major debugging and profiling tools for x86+Hopper are available on GH200**
  - Similar capabilities are provided by other tools on Grace





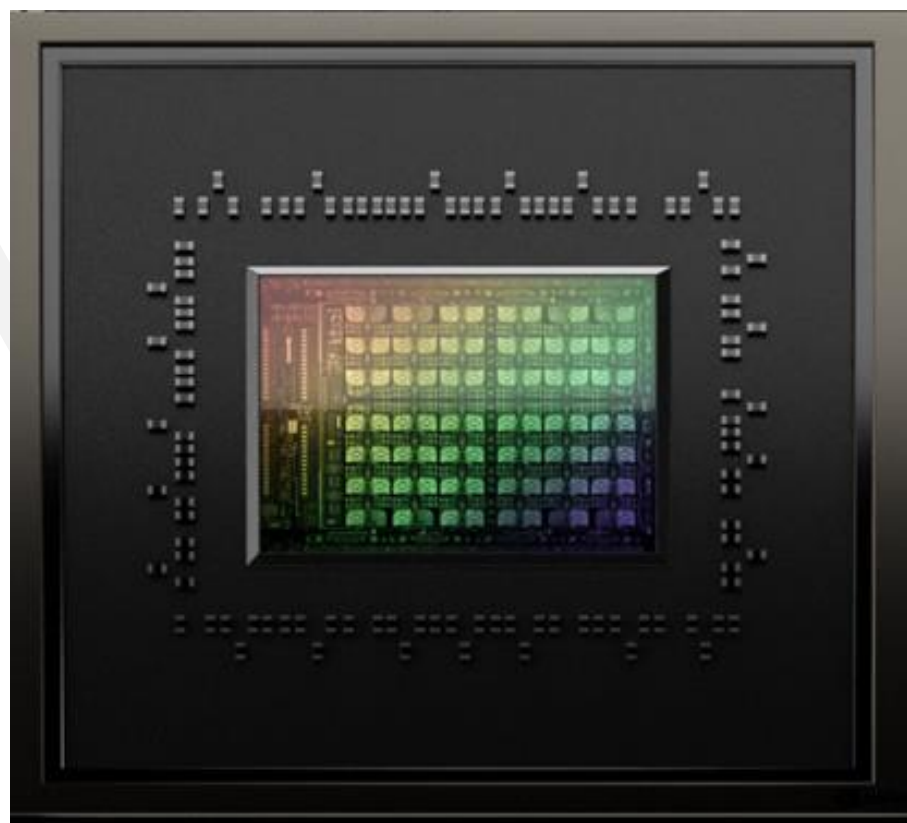


# **Porting and Optimizing for NVIDIA Grace CPU**



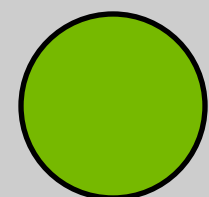
# Expectation: It Just Works

Most applications will recompile easily and work “out of the box”



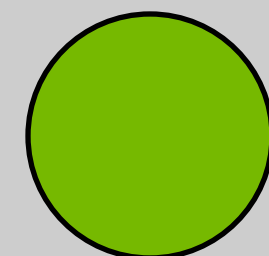
## Reuse

- NGC
- Your Linux Distro's Package Manager



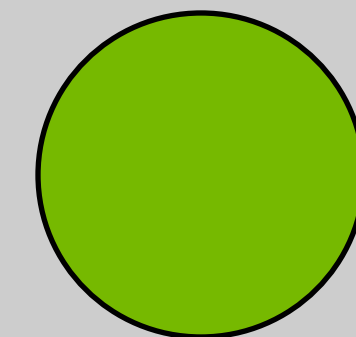
## Recompile

- NVIDIA Compiler
- GCC
- LLVM
- Spack or EasyBuild



## Run

- High core count
- Threads-per-process
- Update tests



## Optimize

- NVIDIA Nsight
- Arm Forge
- Perf, PAPI, TAU, Score-P, ...

## Quick Launch

- NGC containerized applications, frameworks, and toolkits
- `./configure && make install`

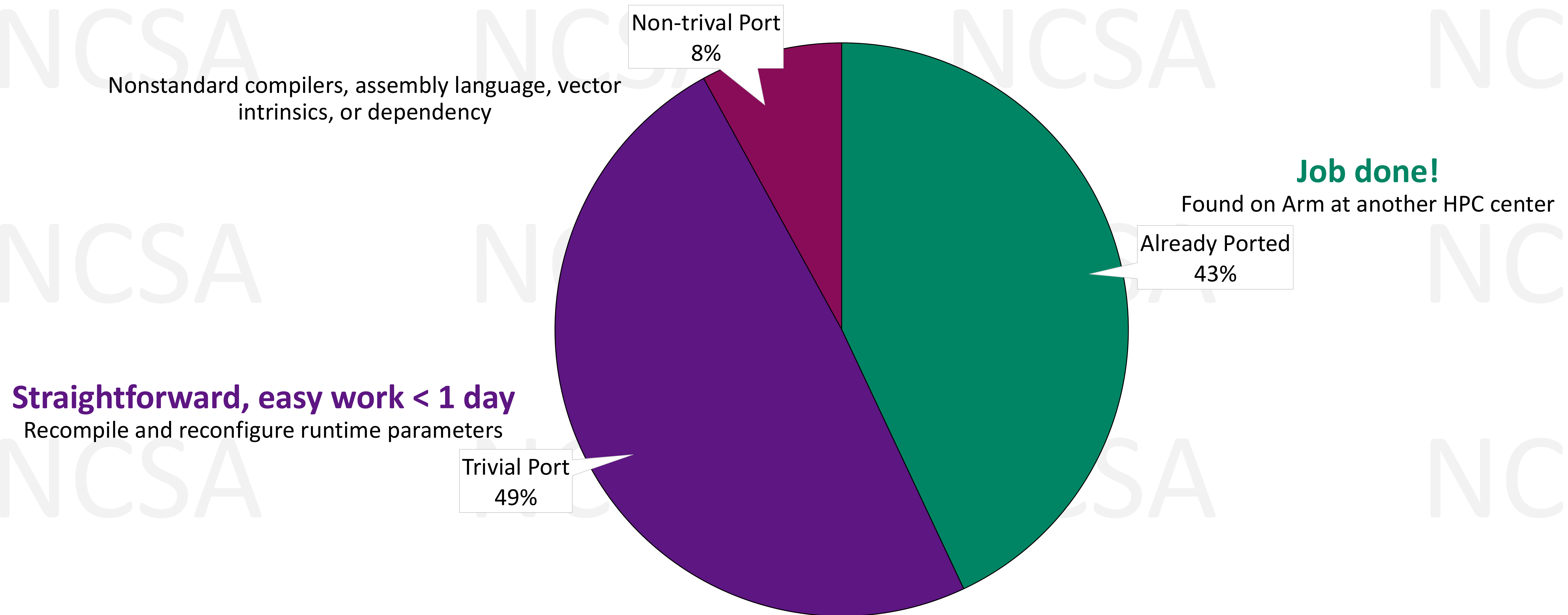
## Compilation Tips

- Most compiler flags are the same:
  - Use `-mcpu=native`
  - Don't use `-march` or `-mtune`
  - You may need `-fsigned-char`
- Update your unit tests:
  - Aarch64 floating point is as accurate as all other platforms



# Workloads of a National Computing Center

Annual core hours

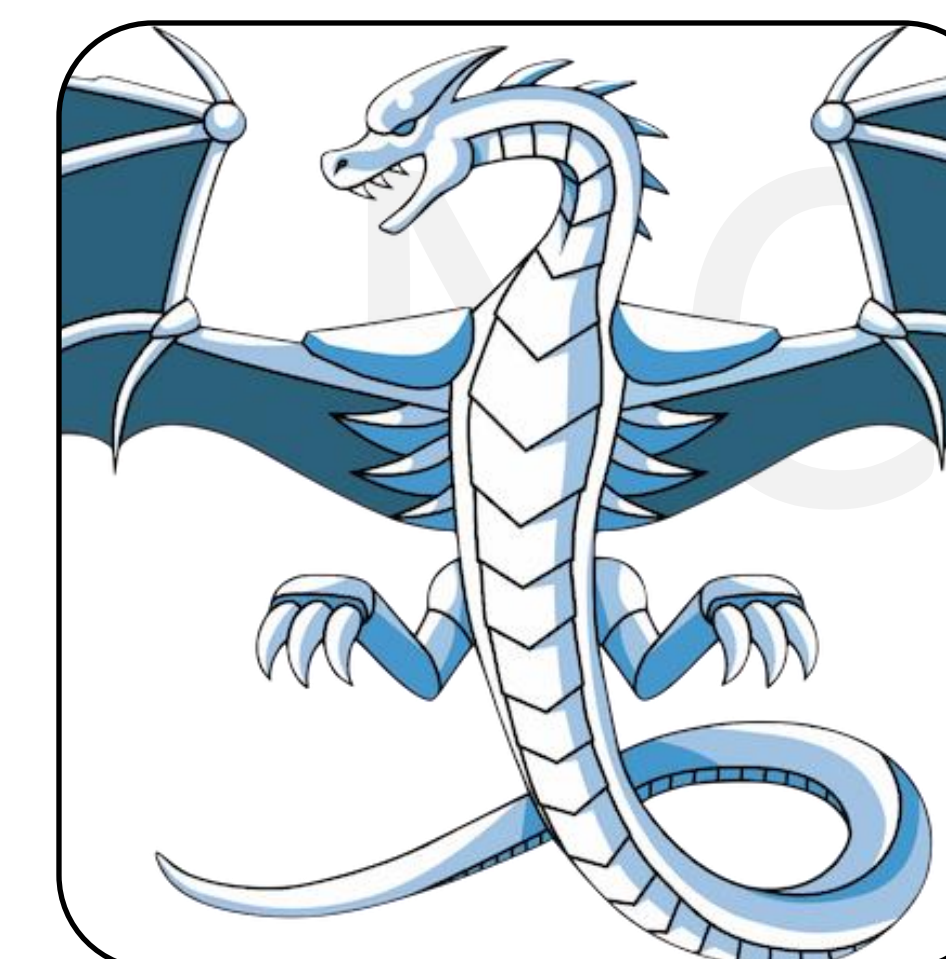
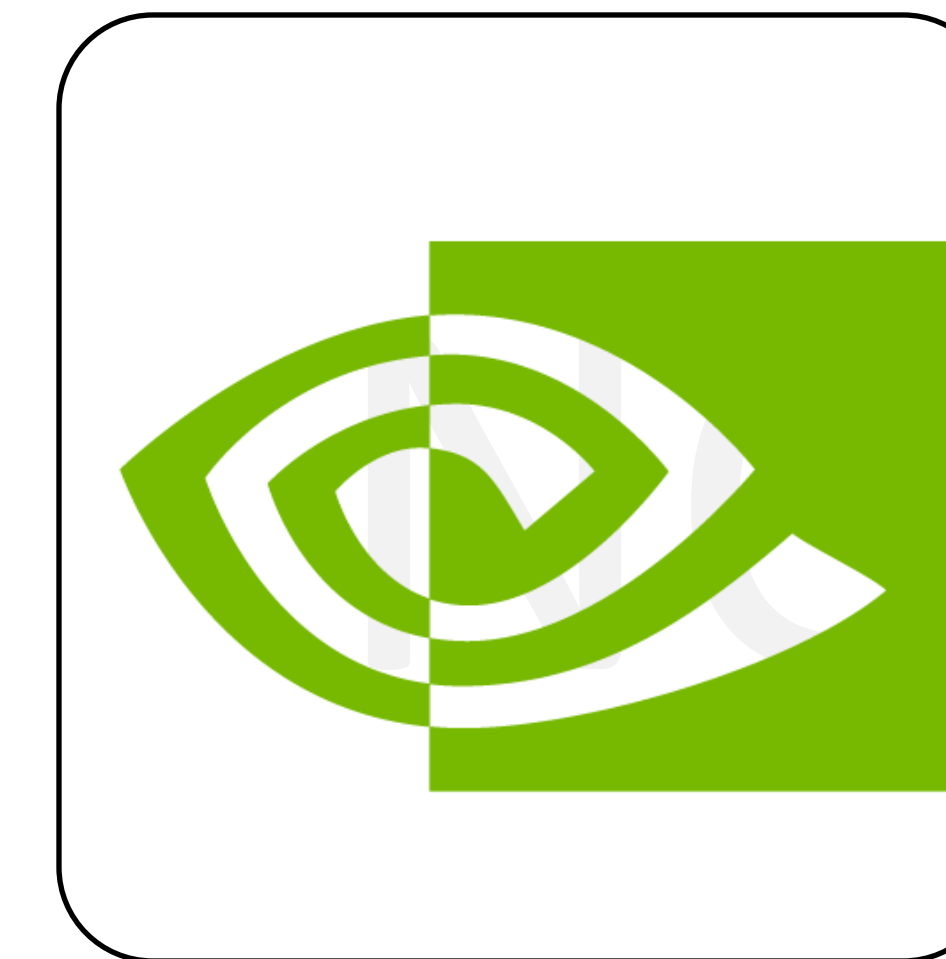




# Use Standards-compliant Multi-platform Compilers

You're not porting to Arm. You're porting away from ifort, xlf, etc.

- Use any portable multi-platform compiler: NVIDIA, GCC, LLVM, etc.
- Use the most recent compiler possible. **GCC 12+** is strongly recommended.
- Beware of non-standard build systems
  - `icc`, `ifort`, `xlf`, etc. may be hard-coded into the build system
  - Be explicit about which compiler to use. Don't let the build system make assumptions
- Beware of non-standard default compilers
  - Check default compiler commands (`cc`, `fc`, `gcc`, etc.) invoke a recent compiler
  - Use `mpicc -show` to verify that MPI compiler wrappers invoke the right compiler
- Log the build, then check the log afterward

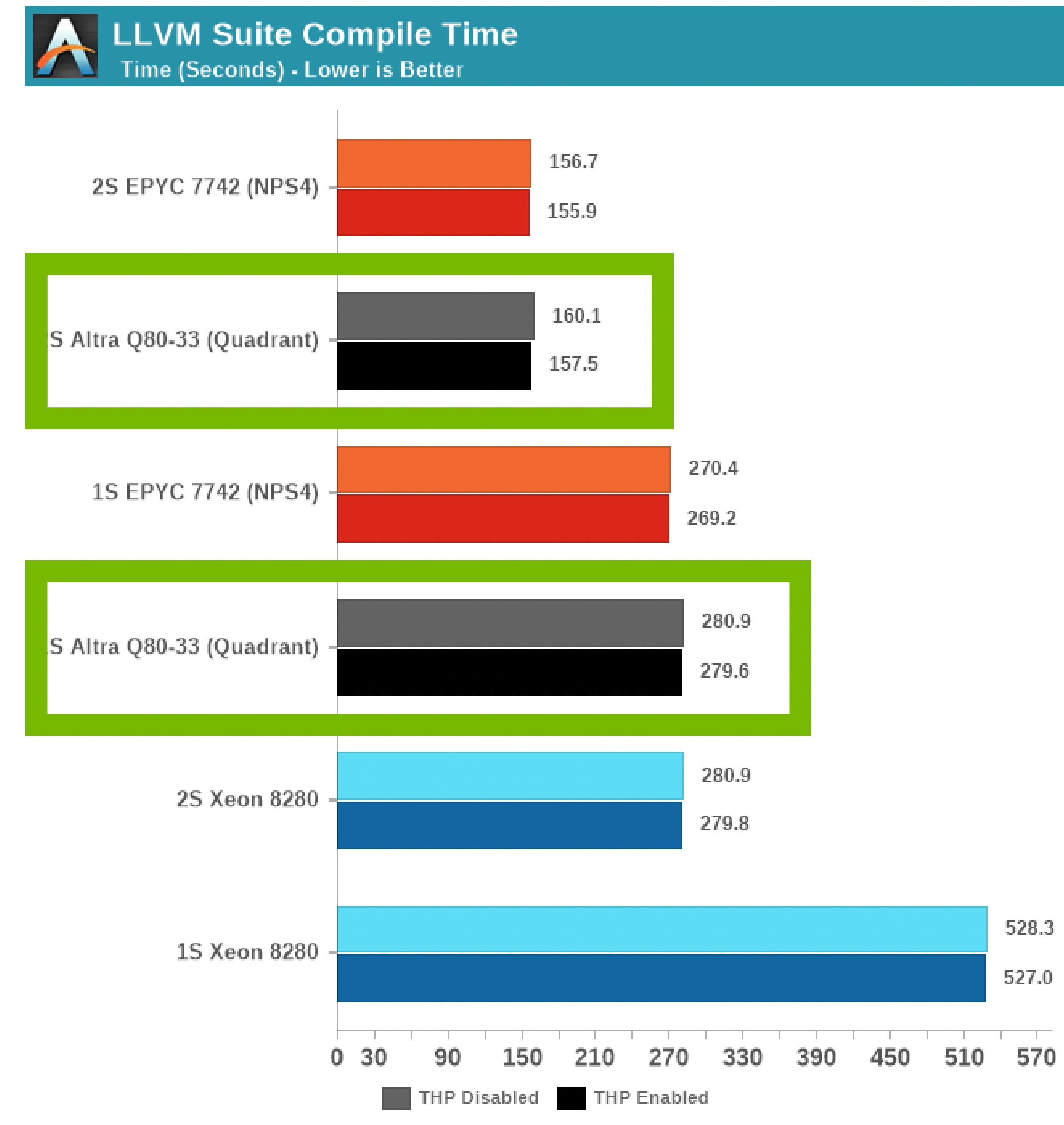
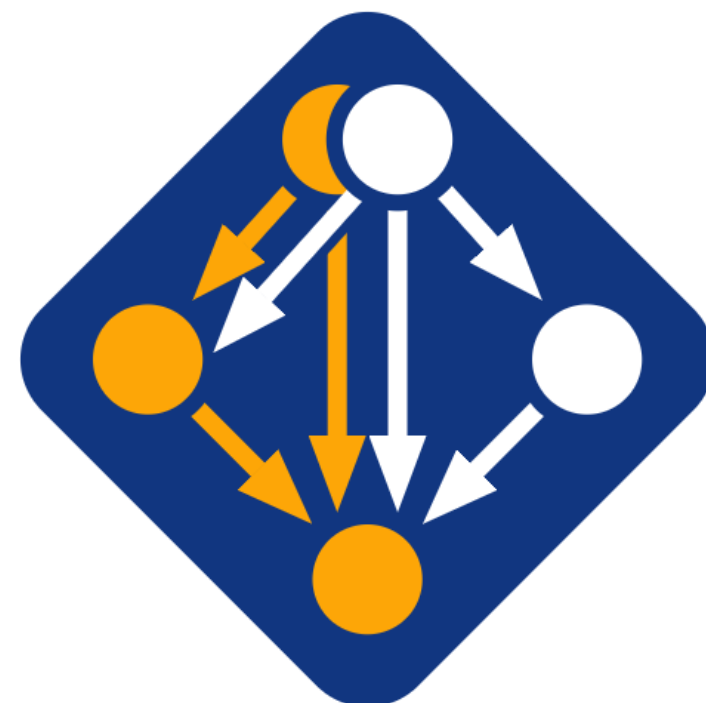




# No Cross Compiling! Just Don't.

All popular build systems are supported – and *performant* – on Arm

- GCC and LLVM are excellent Arm compilers
  - Auto-vectorizing, auto-parallelizing, tested, in production
  - Arm & partners are the majority of GCC contributors
- All major build systems and tools work on Arm
  - CMake, Make, GNUMake, EasyBuild, Spack etc.
- Compiler & build system performance is excellent
  - Ampere Altra compilation performance is on-par with AMD EPYC 7742 – **you do not need to cross compile**



<https://www.anandtech.com/show/16315/the-ampere-altra-review/8>



# Selecting GNU and LLVM Compiler Flags for Grace

Similar flags have different meanings across compilers and across platforms

- Remove all architecture-specific flags: `-mavx`, `-mavx2`, etc.
- Remove `-march` and `-mtune` flags
  - These flags have a different meaning on aarch64
  - See [How to Optimize for Arm and not get Eaten by a Bear](#) for details
- Use `-Ofast -mcpu=native`
  - If fast math optimizations are not acceptable, use `-O3 -ffp-contract=fast`
  - For even more accuracy, use `-ffp-contract=off` to disable floating point operation contraction (e.g. FMA)
  - Can also use `-mcpu=neoverse-v2`, but `-mcpu=native` will “port forward”
- Use `-flto` to enable link-time optimization
  - The benefits of link-time optimization vary from code to code, but can be significant
  - See <https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html> for details
- Apps may need `-fsigned-char` or `-funsigned-char` depending on the developer’s assumption
- gfortran may benefit from `-fno-stack-arrays`



# \_\_atomic\_add\_fetch(&var, num, \_\_ATOMIC\_RELAXED)

GCC 12.3 on Grace

Missing ISA Extensions: i8mm and bf16

```
.arch armv9-a+crc
.file "foo.c"
.text
.align 2
.global main
.type main, %function

main:
.LFB0:

.cfi_startproc
sub    sp, sp, #16
.cfi_def_cfa_offset 16
str    wzr, [sp, 8]
mov    w0, 1
str    w0, [sp, 12]
ldr    w1, [sp, 12]
add    x0, sp, 8
ldadd  w1, w0, [x0]
mov    w0, 0
add    sp, sp, 16
.cfi_def_cfa_offset 0
ret
.cfi_endproc

.LFE0:
.size   main, .-main
.ident  "GCC: (GNU) 12.3.0"
.section .note.GNU-stack,"",@progbits
```

Atomic Add

-march=armv9-a

Correct instruction, limited ISA

Armv8: No SVE!

```
.arch armv8-a
.file "foo.c"
.text
.global __aarch64_ldadd4_relax
.align 2
.global main
.type main, %function

main:
.LFB0:

.cfi_startproc
stp    x29, x30, [sp, -32]!
.cfi_def_cfa_offset 32
.cfi_offset 29, -32
.cfi_offset 30, -24
mov    x29, sp
str    wzr, [sp, 24]
mov    w0, 1
str    w0, [sp, 28]
ldr    w2, [sp, 28]
add    x0, sp, 24
mov    x1, x0
mov    w0, w2
bl     __aarch64_ldadd4_relax
mov    w0, 0
ldp    x29, x30, [sp], 32
.cfi_restore 30
.cfi_restore 29
.cfi_def_cfa_offset 0
ret
.cfi_endproc
```

libgcc call

-mtune=neoverse-v2

Library call instead of atomic instruction, limited ISA

Correct ISA

```
.arch armv9-a+crc+profile+rng+memtag+sve2-bitperm+i8mm+bf16
.file "foo.c"
.text
.align 2
.global main
.type main, %function

.cfi_startproc
sub    sp, sp, #16
.cfi_def_cfa_offset 16
str    wzr, [sp, 8]
mov    w0, 1
str    w0, [sp, 12]
ldr    w1, [sp, 12]
add    x0, sp, 8
ldadd  w1, w0, [x0]
mov    w0, 0
add    sp, sp, 16
.cfi_def_cfa_offset 0
ret
.cfi_endproc

.size   main, .-main
.ident  "GCC: (GNU) 12.3.0"
.section .note.GNU-stack,"",@progbits
```

Atomic Add

-mcpu=neoverse-v2 (or -mcpu=native)

Correct instruction, correct ISA



# Porting Applications that use Math Libraries: MKL, OpenBLAS, etc.

Several library options to choose from

- Prefer Netlib BLAS/LAPACK and FFTW interfaces
  - Building on these interfaces enables compatibility

- **NVPL**

- gcc **-DUSE\_CBLAS** -ffast-math -mcpu=native -O3 \
  - I/PATH/T0/nvpl/include** \
  - L/PATH/T0/nvpl/lib** \
  - o mt-dgemm.nvpl mt-dgemm.c \
  - lnvpl\_blas\_lp64\_gomp**

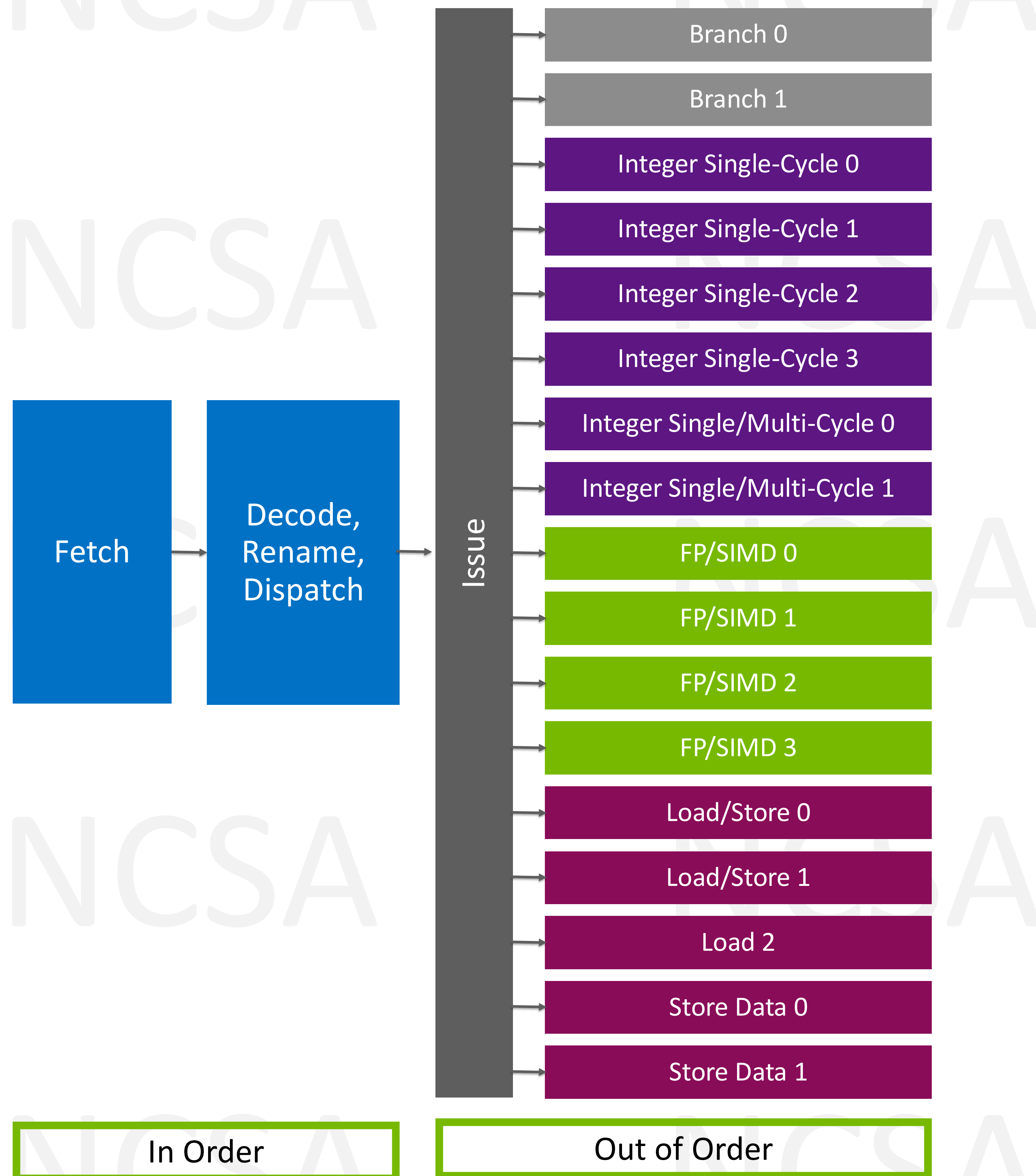
- **ArmPL**

- gcc -DUSE\_CBLAS -ffast-math -mcpu=native -O3 \
  - I/opt/arm/armpl-23.10.0\_Ubuntu-22.04\_gcc/include \
  - L/opt/arm/armpl-23.10.0\_Ubuntu-22.04\_gcc/lib \
  - o mt-dgemm.armpl mt-dgemm.c \
  - larmpl\_lp64

```
libnvpl_blas_ilp64_gomp.so
libnvpl_blas_ilp64_seq.so
libnvpl_blas_lp64_gomp.so
libnvpl_blas_lp64_seq.so
libnvpl_fftw.so
libnvpl_lapack_ilp64_gomp.so
libnvpl_lapack_ilp64_seq.so
libnvpl_lapack_lp64_gomp.so
libnvpl_lapack_lp64_seq.so
libnvpl_rand_mt.so
libnvpl_rand.so
libnvpl_scalapack_ilp64.so
libnvpl_scalapack_lp64.so
libnvpl_sparse.so
libnvpl_tensor.so
```

- **ATLAS**, **OpenBLAS**, **BLIS**, ... Community supported with some optimizations for Neoverse V2.
  - Works on Grace, but unlikely to outperform NVPL and ArmPL. A good compatibility option.





## SIMD in NVIDIA Grace

- 4x128b SIMD units = 512b SIMD vector bandwidth
- Full core frequency at 100% 512b SIMD utilization
  - With all cores at 100%, a fully loaded socket may downclock about 200MHz
- Each SIMD unit can retire NEON or SVE2 instructions
- On this architecture, SVE2 and NEON have the **same peak performance** ...
- ... but SVE2 can vectorize more complex codes and supports more data types than NEON
- In practice, SVE2 typically outperforms NEON



# Porting Assembly and Vector Intrinsics

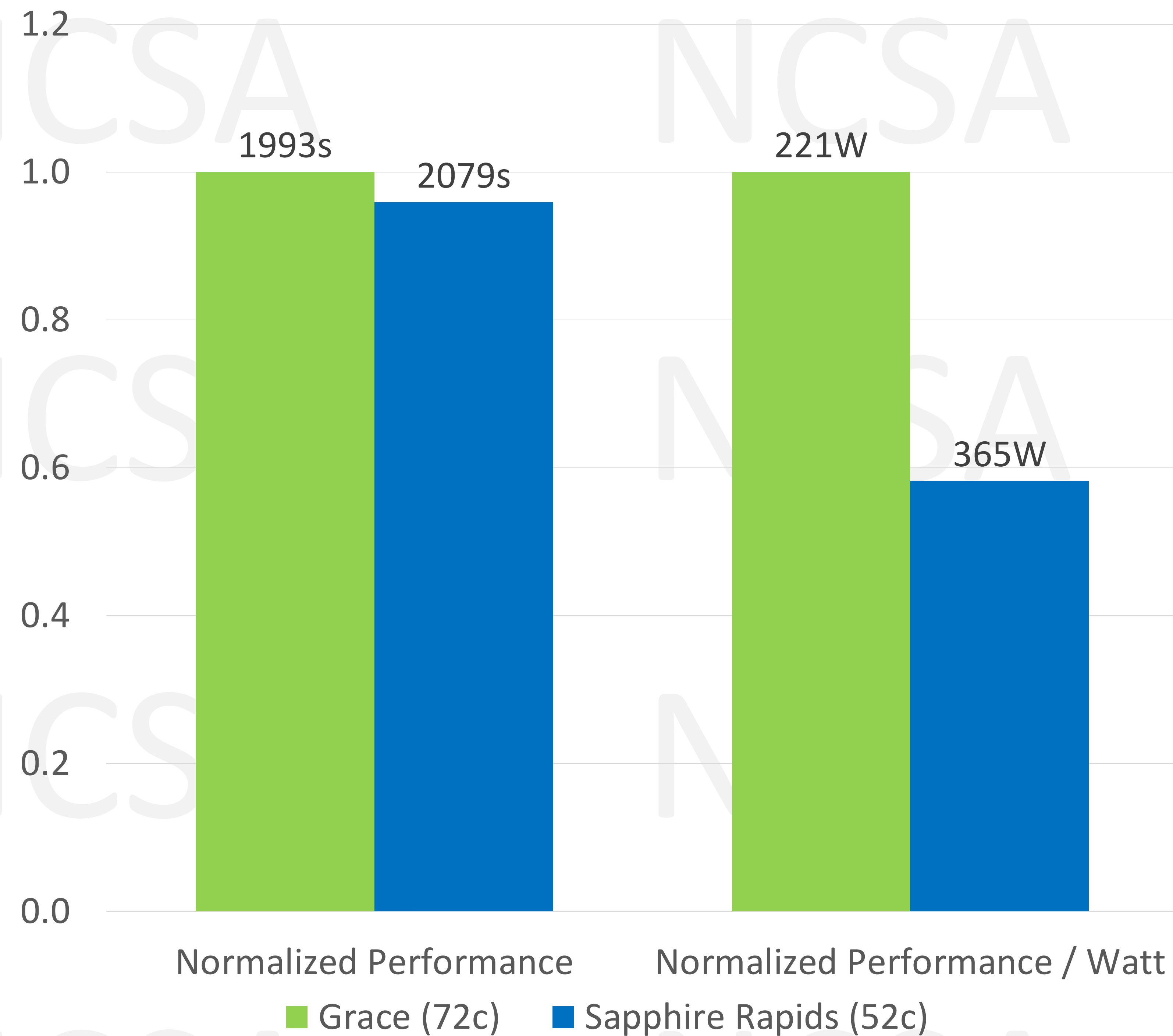
Translate intrinsics to port functionality, then focus on performance tuning

- For a quick fix, use a drop-in header-based intrinsics translator
  - SIMD Everywhere (SIMDe): <https://github.com/simd-everywhere/simde>
  - SSE2NEON: <https://github.com/DLTcollab/sse2neon>
  - **Demonstration:** <https://www.nvidia.com/en-us/on-demand/session/gtcspring22-s41702/>
- Follow Arm's documentation on rewriting x86 vector intrinsics
  - [Porting and Optimizing HPC Applications for Arm SVE](https://developer.arm.com/documentation/101726/latest) [https://developer.arm.com/documentation/101726/latest]
  - [Coding for NEON](https://developer.arm.com/documentation/101725/0300/Coding-for-Neon) [https://developer.arm.com/documentation/101725/0300/Coding-for-Neon]
- Arm assembly is simpler than x86
  - Arm processors have a much simpler and general set of registers than x86. Just assign a one-to-one mapping from an x86 register to an Arm register when porting code.
  - Complex x86 instructions will become multiple Arm instructions



# Porting x86 Intrinsics: BWA-MEM2

Auto-translation overhead is offset by CPU performance advantage



## Scope of “porting” work, no optimization done:

~3 hours of developer time to investigate and add:

#include: AVX -> Vendor Nonspecific SIMD Wrapper

- <https://github.com/simd-everywhere/simde>

## Tools:

- Compilers: Clang 16 (NVIDIA)
- Compile options: GCC 12 and SIMDe

## Comparison:

- Precompiled binaries on x86
- HG002 dataset from Illumina paired-end sequencers
- Complete human genome at 30x coverage

## Run configuration:

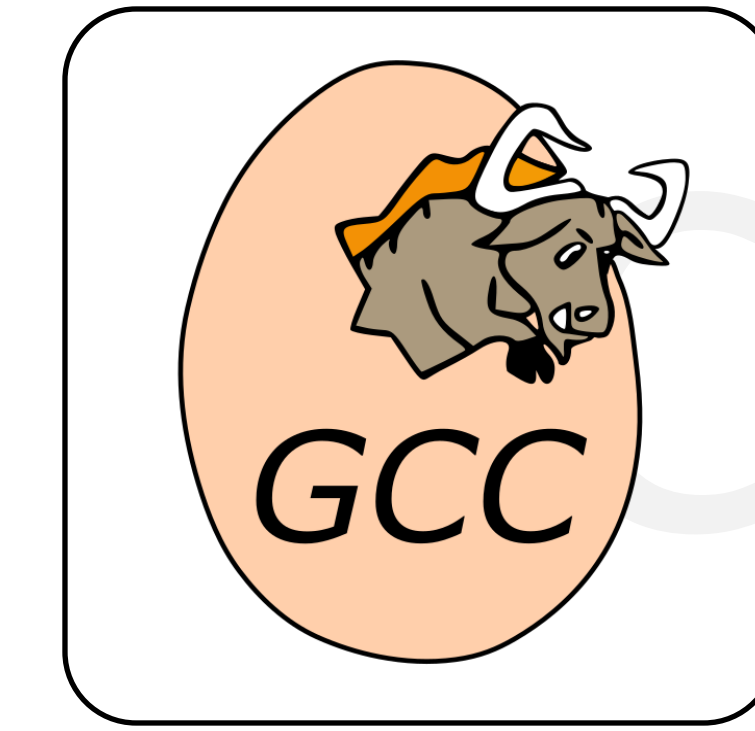
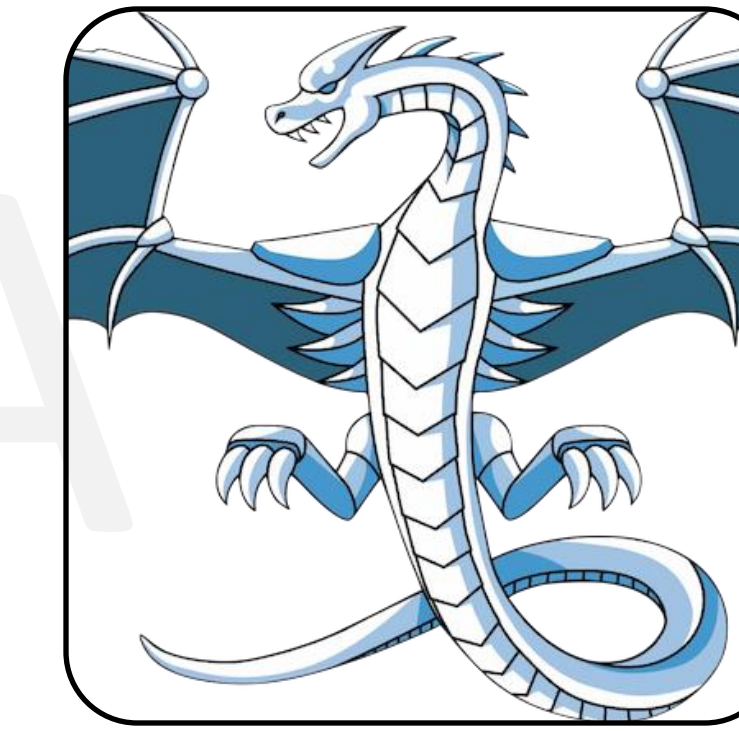
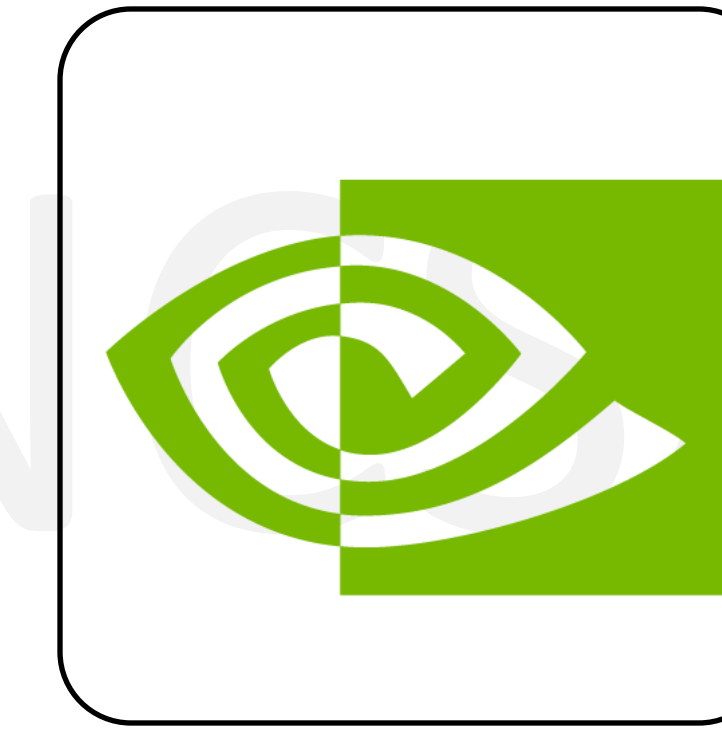
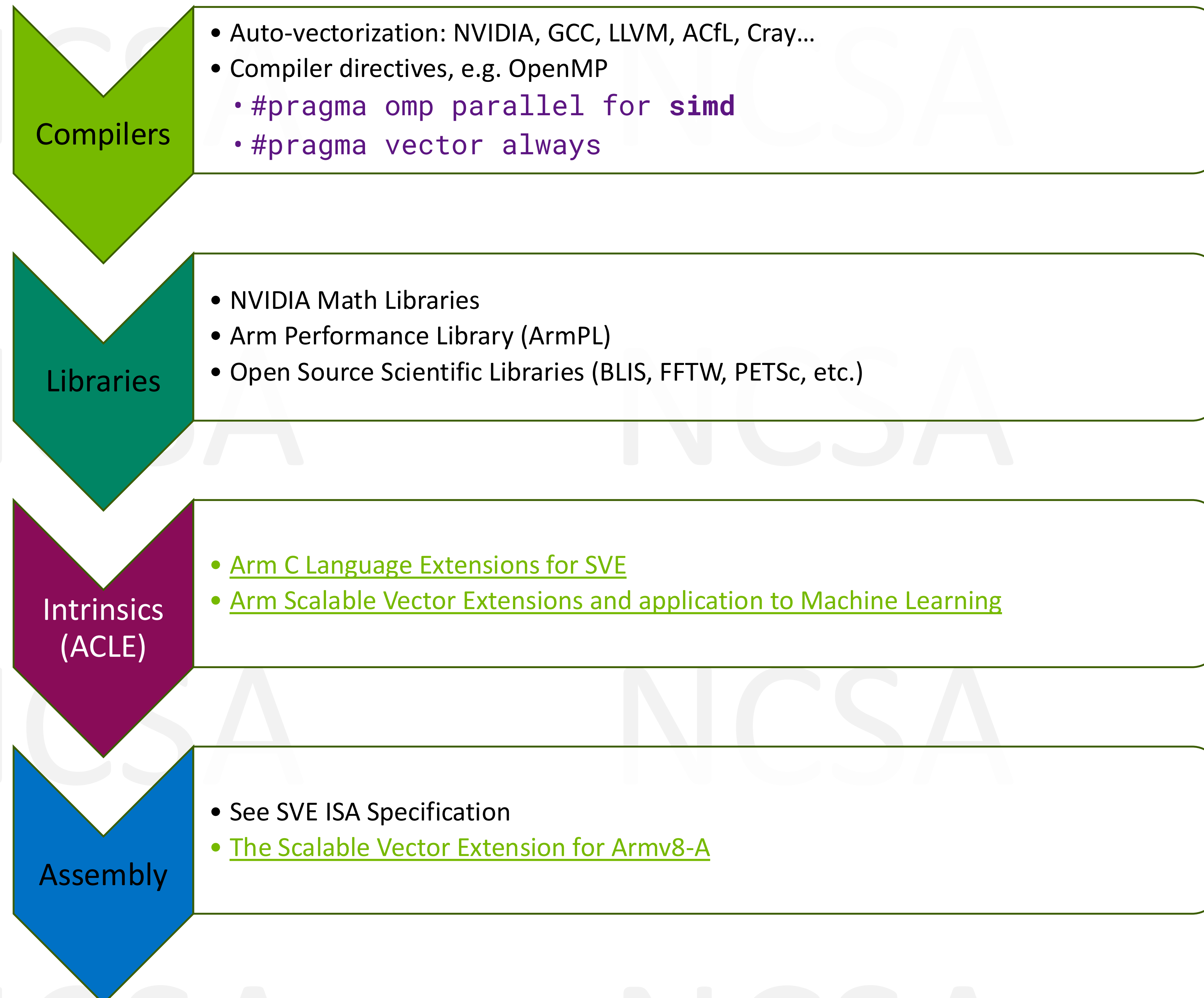
Similar configuration overhead to moving between Intel & AMD

- Grace: jemalloc + transparent huge pages
- Intel: AVX512 intel-optimized version on SPR



# SIMD Programming Approaches

Follow these recommendations in order, e.g. prefer auto-vectorization over intrinsics



BLAS

LAPACK

PBLAS

SCALAPACK

TENSOR

SPARSE

RAND

FFTW

```
// Complex dot product
// (a+ib)*(c+id) = (ac - bd) + i(ad + bc)
void complex_dot_product(Complex_t c[SIZE], Complex_t a[SIZE], Complex_t b[SIZE])
{
    uint32_t vl = svcntw();


    svbool_t p32_all = svptrue_b32();

    for (int i=0; i<SIZE; i+=vl) {
        svfloat32x2_t va = svld2(p32_all, (float32_t*)&a[i]);
        svfloat32x2_t vb = svld2(p32_all, (float32_t*)&b[i]);
        svfloat32x2_t vc = svld2(p32_all, (float32_t*)&c[i]);

        vc.v0 = svmla_m(p32_all, vc.v0, va.v0, vb.v0); //c.re += a.re * b.re
        vc.v1 = svmla_m(p32_all, vc.v1, va.v1, vb.v0); //c.im += a.im * b.re
        vc.v0 = svmls_m(p32_all, vc.v0, va.v1, vb.v1); //c.re -= a.im * b.im
        vc.v1 = svmla_m(p32_all, vc.v1, va.v0, vb.v1); //c.im += a.re * b.im

        svst2(p32_all, (float32_t*)&c[i], vc);
    }
}
```





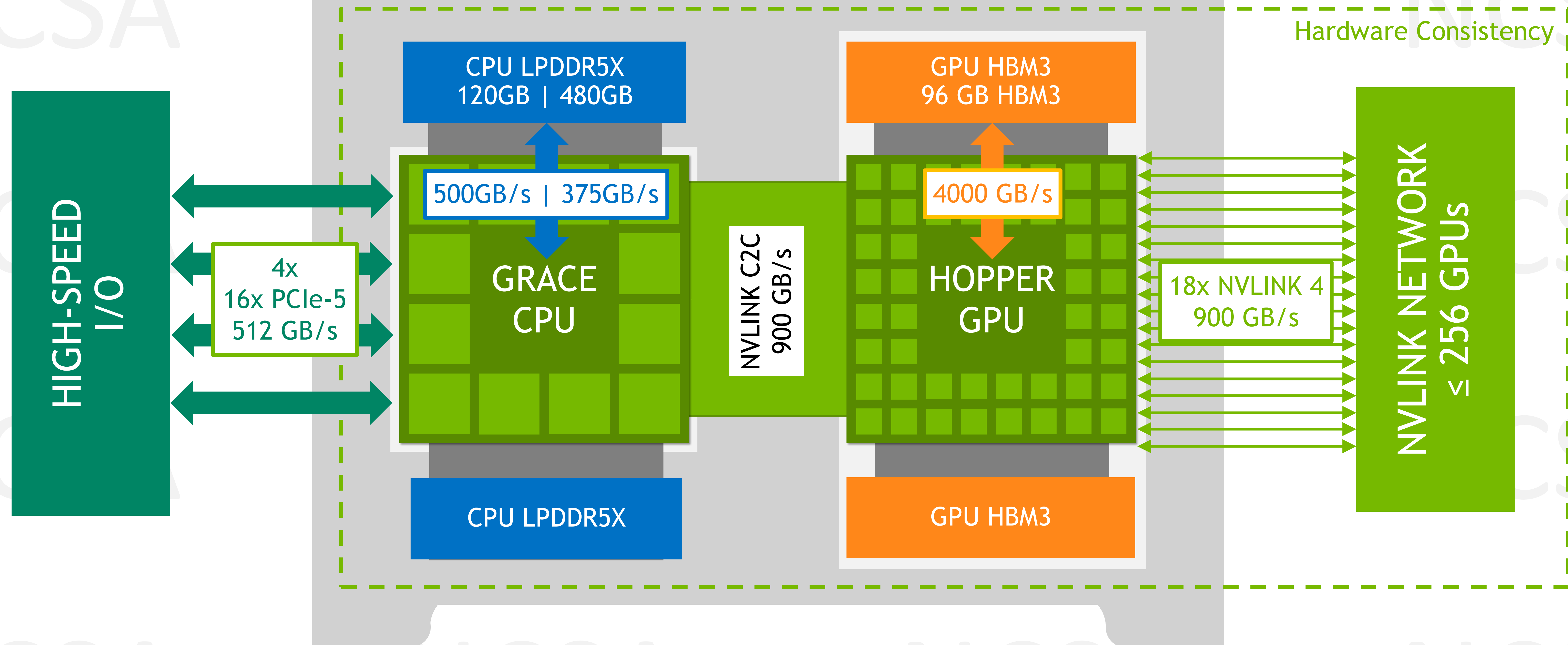
# **Optimizing for Coherent Memory**



# Grace Hopper Superchip

GPU can access CPU memory at CPU memory speeds

## NVIDIA Grace Hopper Superchip





# GPU Memory is Visible to the Operating System

Standard operating system commands work on the GPU

```
nvidia@localhost: ~  
nvidia@localhost:~$ numactl -H  
available: 9 nodes (0-8)  
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 5  
2 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71  
node 0 size: 490310 MB  
node 0 free: 475425 MB  
node 1 cpus:  
node 1 size: 96768 MB  
node 1 free: 96767 MB  
node 2 cpus:  
node 2 size: 0 MB  
node 2 free: 0 MB  
node 3 cpus:  
node 3 size: 0 MB  
node 3 free: 0 MB  
node 4 cpus:  
node 4 size: 0 MB  
node 4 free: 0 MB  
node 5 cpus:  
node 5 size: 0 MB  
node 5 free: 0 MB  
node 6 cpus:  
node 6 size: 0 MB  
node 6 free: 0 MB  
node 7 cpus:  
node 7 size: 0 MB  
node 7 free: 0 MB  
node 8 cpus:  
node 8 size: 0 MB  
node 8 free: 0 MB  
node distances:  
node 0 1 2 3 4 5 6 7 8  
0: 10 80 80 80 80 80 80 80 80  
1: 80 10 255 255 255 255 255 255 255  
2: 80 255 10 255 255 255 255 255 255  
3: 80 255 255 10 255 255 255 255 255  
4: 80 255 255 255 10 255 255 255 255  
5: 80 255 255 255 255 10 255 255 255  
6: 80 255 255 255 255 255 10 255 255  
7: 80 255 255 255 255 255 255 10 255  
8: 80 255 255 255 255 255 255 255 10  
nvidia@localhost:~$ free -g  
              total        used        free      shared  buff/cache   available  
Mem:           573          11         558           1           3         541  
Swap:           0           0           0
```

CPU

GPU

MIG

```
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 5  
2 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71  
node 0 size: 490310 MB  
node 0 free: 475425 MB  
node 1 cpus:  
node 1 size: 96768 MB  
node 1 free: 96767 MB  
node 2 cpus:
```

Hopper GPU appears to the OS as a NUMA node with no CPU cores

Total system memory capacity is CPU (480GB) + GPU (96GB)

```
nvidia@localhost:~$ free -g  
              total        used        free      shared  buff/cache   available  
Mem:           573          11         558           1           3         541  
Swap:           0           0           0
```

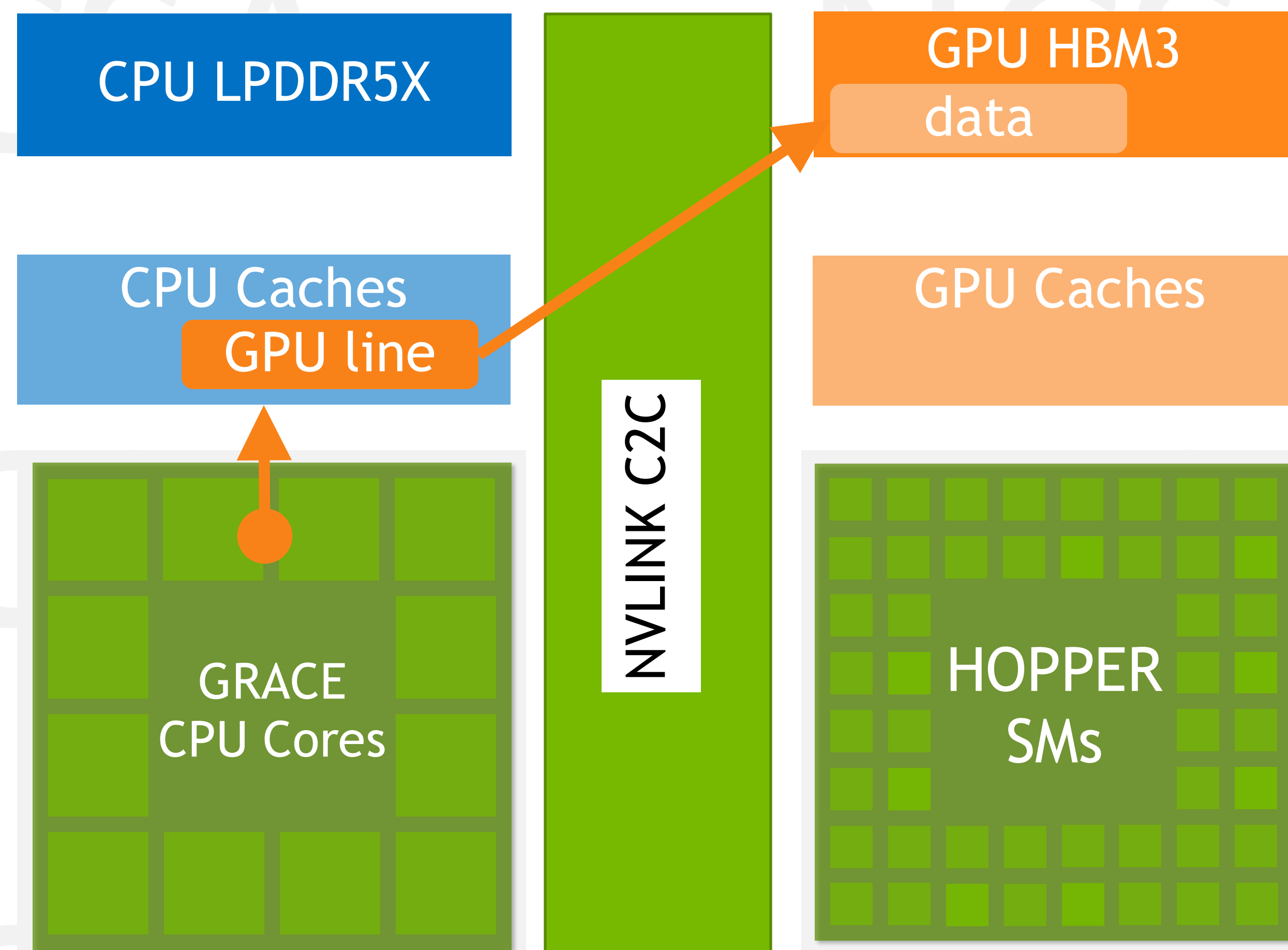
```
nvidia@localhost:/home/nvidia/jlinford/mt-dgemm/src$ numactl -m1 ./mt-dgemm.nvpl 5000 1 1 1 0 1 1  
Matrix size input by command line: 5000  
Repeat multiply 1 times.
```

Can use numactl to put CPU application data in GPU memory



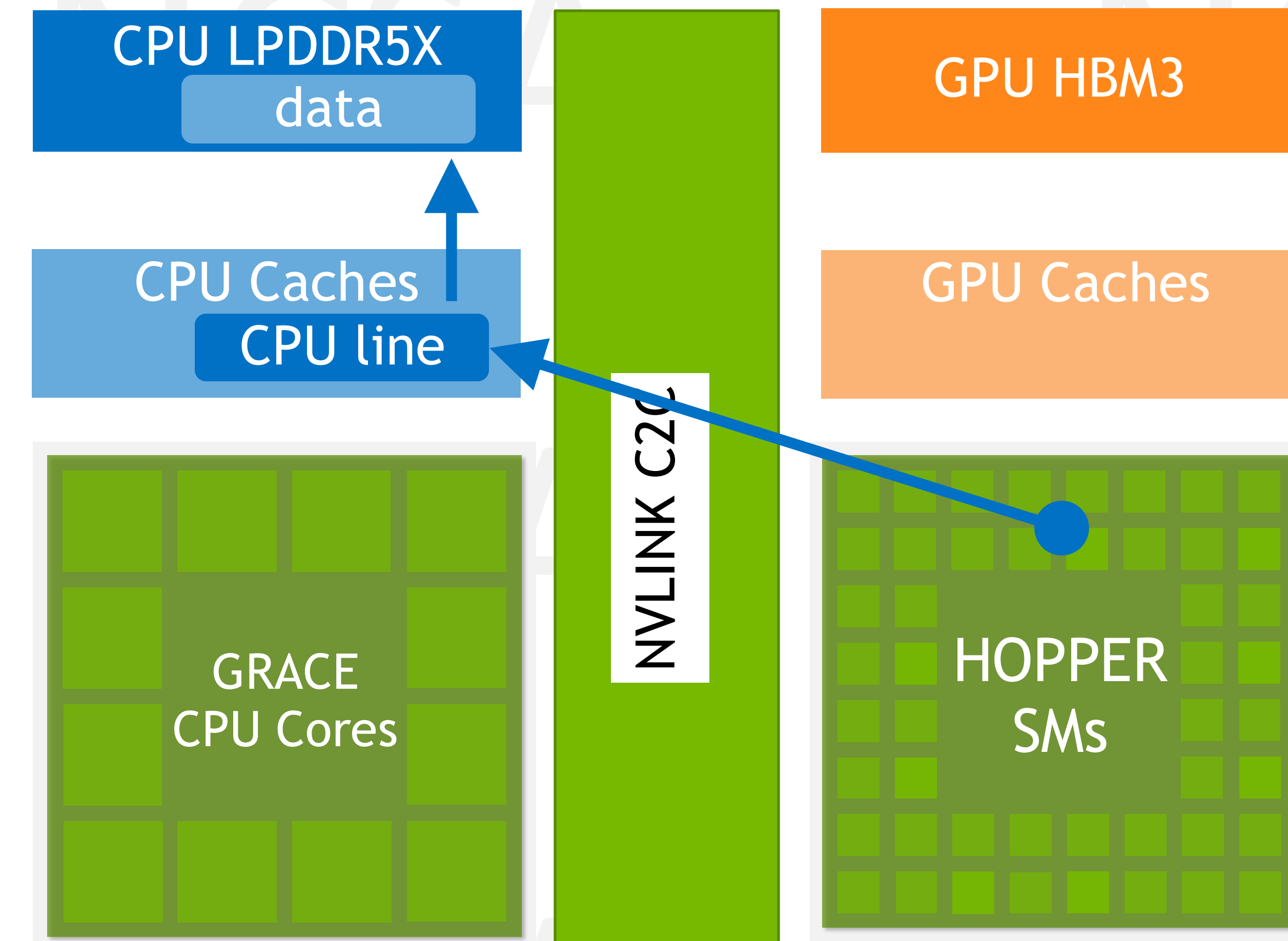
# Global Access to All Data

Cache-coherent access via NVLink C2C from either processor to either physical memory



Grace directly reading Hopper's memory

CPU fetches GPU data into CPU L3 cache  
Cache remains **coherent** with GPU memory  
Changes to GPU memory **evict** cache line

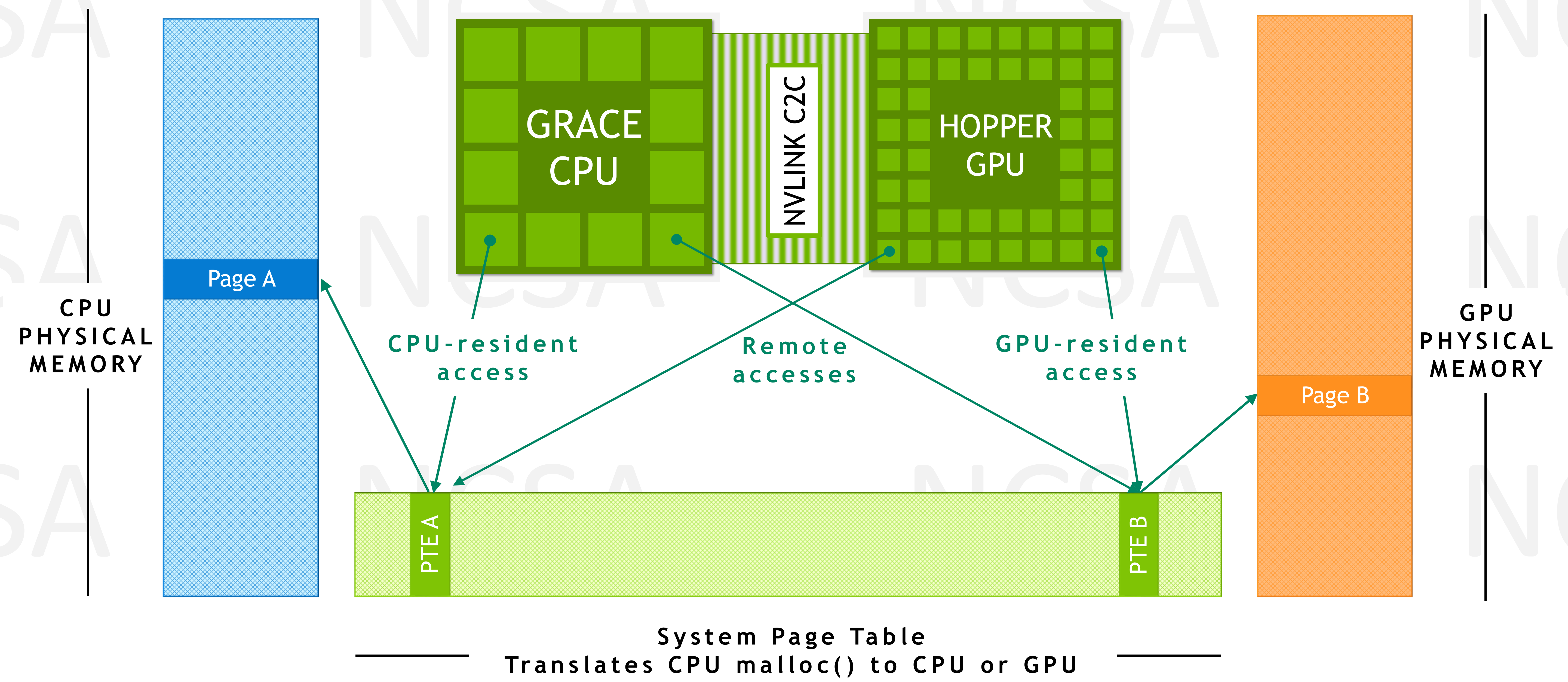


Hopper directly reading Grace's memory

GPU loads CPU data via CPU L3 cache  
CPU and GPU **can both hit** on cached data  
Changes to CPU memory **update** cache line

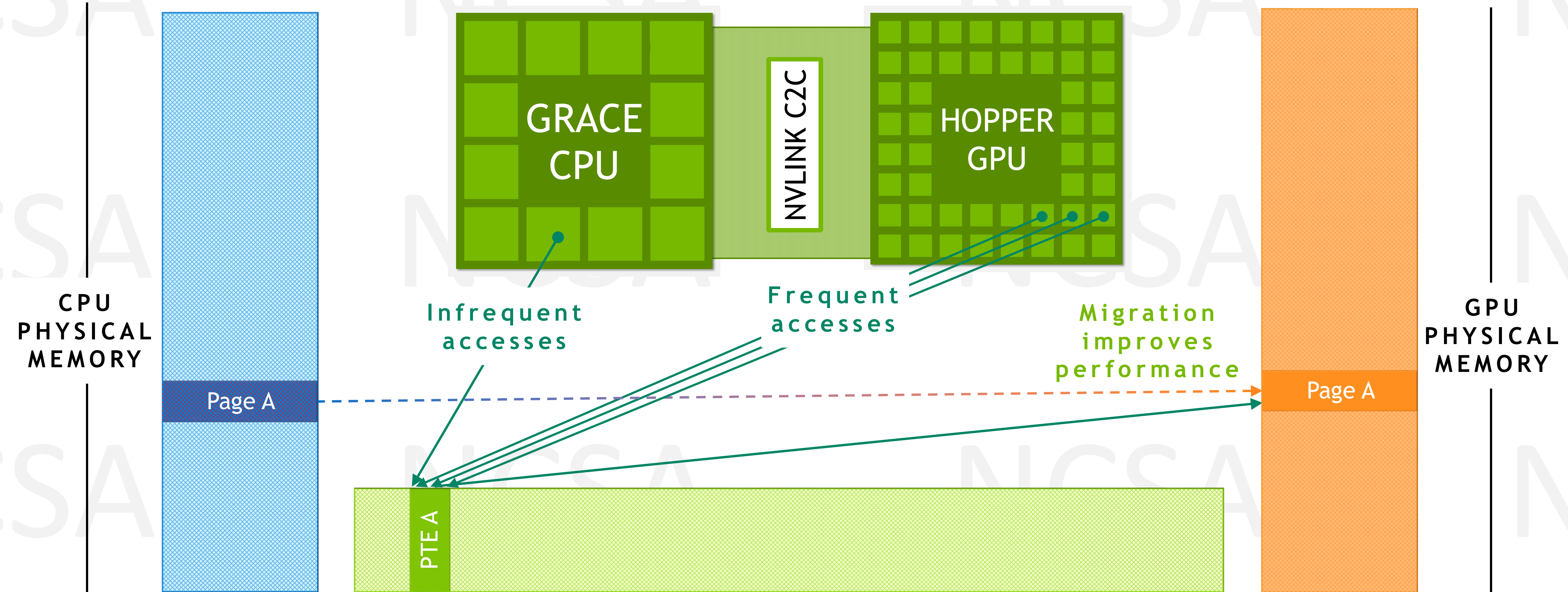


# Grace Hopper Address Translation Service (ATS)



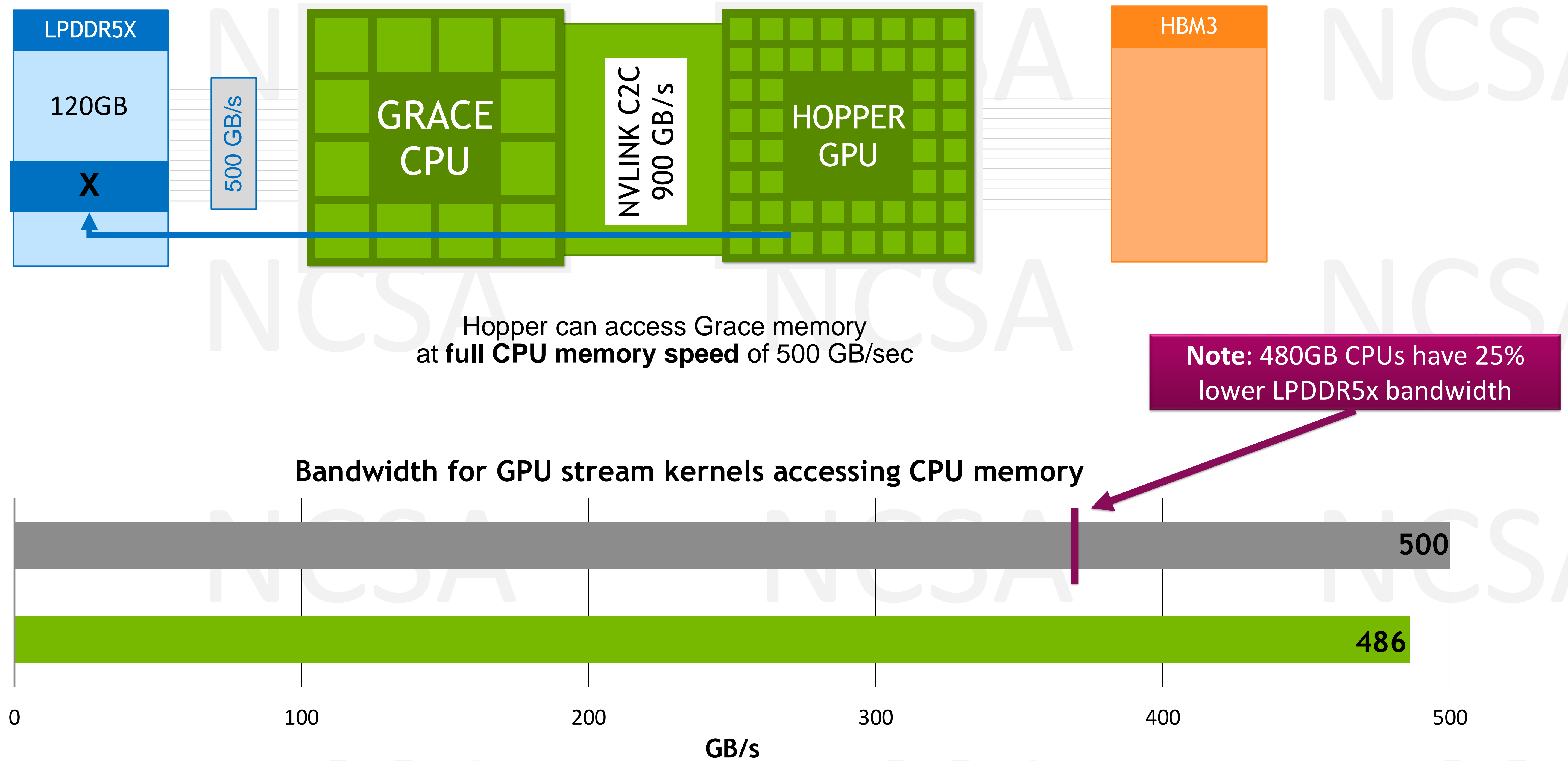


# Grace Hopper Automatic Page Migration



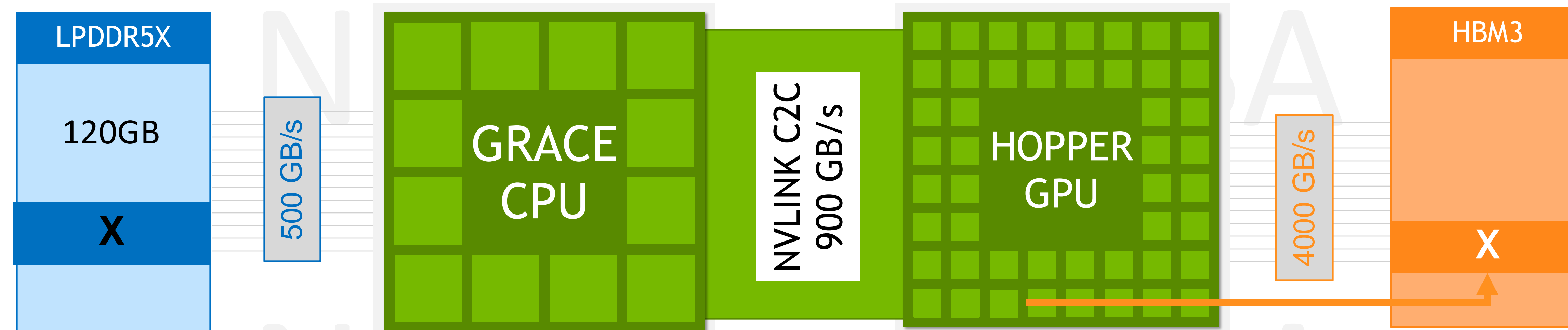


# High Bandwidth Memory Access & Automatic Data Migration



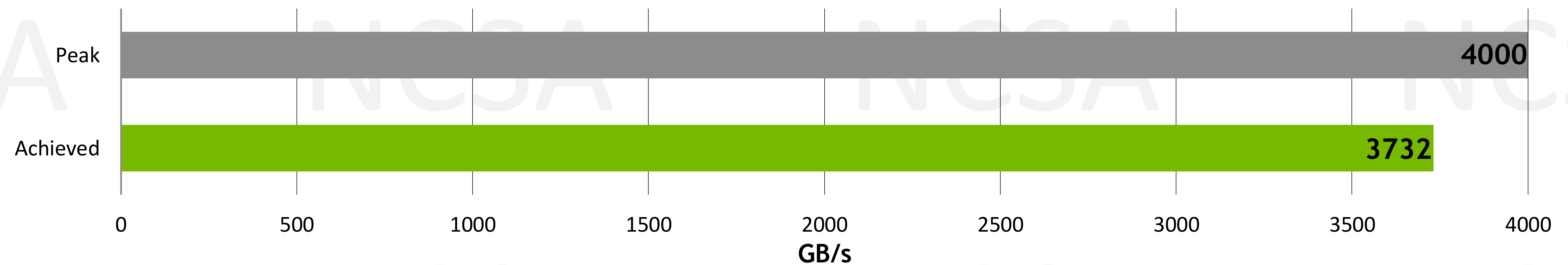


# High Bandwidth Memory Access & Automatic Data Migration



But Hopper can access its own memory  
at full HBM speed of 4000 GB/sec

Bandwidth for GPU stream kernels accessing GPU memory





# Memory Allocators Impact Data Placement and Movement

CUDA 12.4

	cudaMalloc	cudaMallocManaged	Malloc/mmap
Placement	GPU		
Page size	2MB		
Which processor can access ?	GPU		
How does access happen ?	GPU MMU		
What can the driver do for my app ?	-		
What can I do for my app ?	Think how to potential leverage coherent systems		



# Memory Allocators Impact Data Placement and Movement

CUDA 12.4

	cudaMalloc	cudaMallocManaged	Malloc/mmap
Placement	GPU	First touch	
Page size	2MB	hybrid, 64K for CPU and 2MB for GPU	
Which processor can access ?	GPU	Both CPU and GPU	
How does access happen ?	GPU MMU	Fault on first access and move page [2]	
What can the driver do for my app ?	-	Fault or Access counters [2]	
What can I do for my app ?	Think how to potential leverage coherent systems	Use CUDA APIs to manage memory	

- 1. mTHP [will allow 2MB page sizes](#) Linux kernel 6.9 patch or hugeTLB
- 2. Unless Memadvise with preferred location and setAccessedBy are set
- 3. Pages don't migrate back to CPU due to lack of access counters



# Memory Allocators Impact Data Placement and Movement

CUDA 12.4

	cudaMalloc	cudaMallocManaged	System (malloc/mmap/...)
Placement	GPU	First touch	First touch
Page size	2MB	hybrid, 64K for CPU and 2MB for GPU	64K (system page) [1]
Which processor can access ?	GPU	Both CPU and GPU	Both CPU and GPU
How does access happen ?	GPU MMU	Fault on first access and move page [2]	Direct access over C2C using ATS [2]
What can the driver do for my app ?	-	Fault or Access counters [2]	Using access counter to migrate memory CPU -> GPU [3]
What can I do for my app ?	Think how to potential leverage coherent systems	Use CUDA APIs to manage memory	Use CUDA APIs to manage memory

- 1. mTHP will allow 2MB page sizes Linux kernel 6.9 patch or hugeTLB
- 2. Unless Memadvise with preferred location and setAccessedBy are set
- 3. Pages don't migrate back to CPU due to lack of access counters



# Grace UVM Migration Enhancements: CUDA C++ & CUDA Fortran

Maximum portable performance to NVIDIA HW  
out-of-the-box & without any changes

- No programming model changes!
  - No new APIs
  - No changes to existing APIs
  - No source code changes
- Unified Memory
  - Available on *most* platforms supported by CUDA 12.x: GH, P9+V100, PCIe x86 & Arm, etc.
  - Same Unified Memory Programming Model for all platforms: "memory accesses just work" + "hints".
- Unified Memory **Hints**
  - *Hints* only impact performance, not results.
  - **cudaMemAdvise** hints: PreferredLocation, AccessedBy.
  - **cudaMemPrefetch** hints: prefetch to NUMA node.
  - Works with cudaMallocManaged memory on all supported Unified Memory platforms
  - Work with system allocated memory (e.g. malloc) on Grace Hopper and systems with HMM

## CUDA Explicit Memory Allocators

Memory	Placement	Access-based Migration	Accessible From	
			CPU	GPUs
System-allocated (malloc, mmap)	First-touch (GPU   CPU)	✓	✓	✓
CUDA managed (cudaMallocManaged)		✓	✓	✓
CUDA device memory (cudaMalloc)	GPU	✗	✗	✓
CUDA host memory (cudaMallocHost)	CPU	✗	✓	✓
...and many others: interprocess, virtual, fabric, ...				

## CUDA Unified Memory Hints

```
cudaMemAdvise(ptr, nbytes, advice, device);
```

Advices	PreferredLocation	AccessedBy	ReadMostly
---------	-------------------	------------	------------

Devices	GPU id	CPU	CPU Numa Node
---------	--------	-----	---------------

```
cudaMemPrefetchAsync(ptr, nbytes, destination, stream);
```

Destinations	GPU id	CPU	CPU Numa Node
--------------	--------	-----	---------------



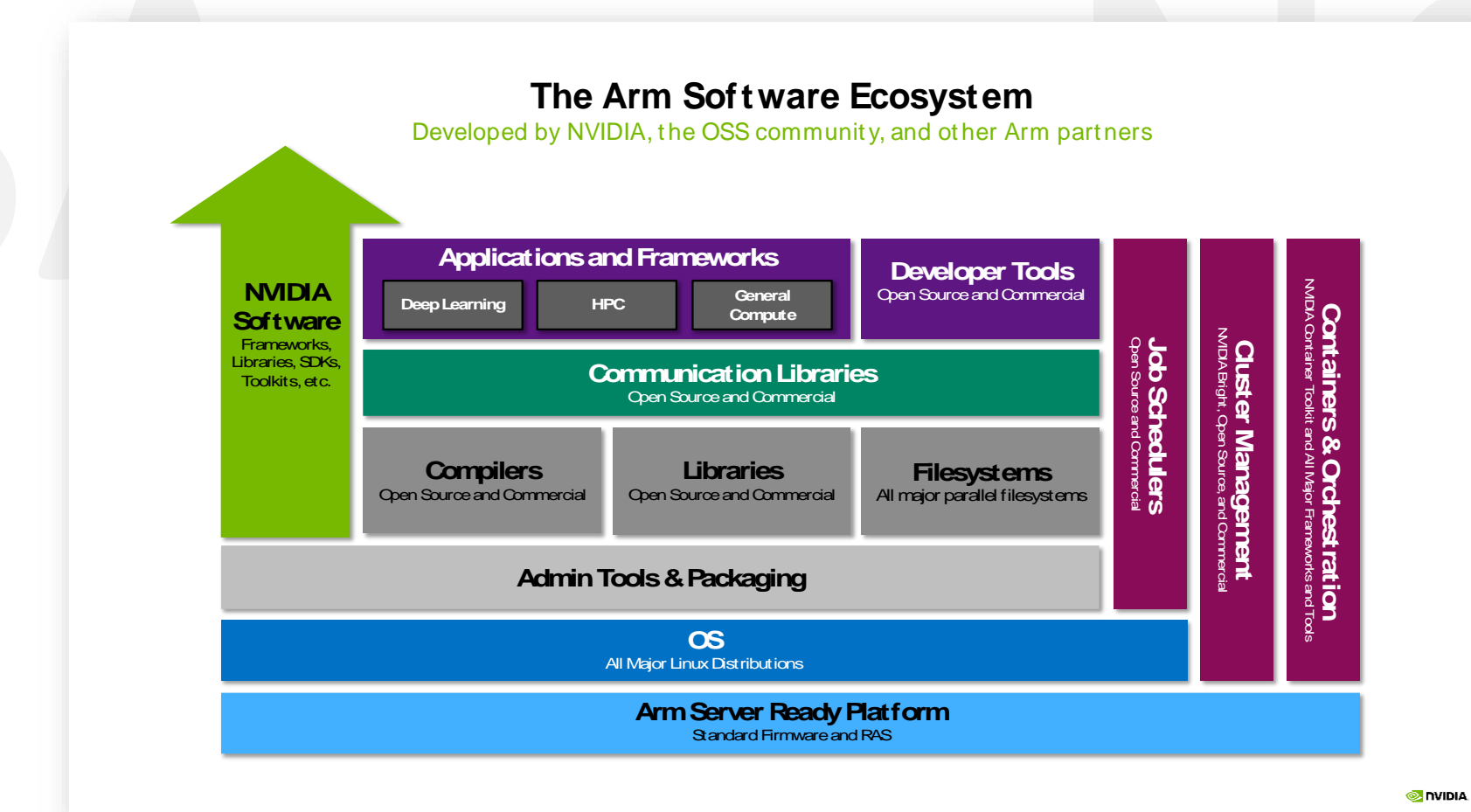
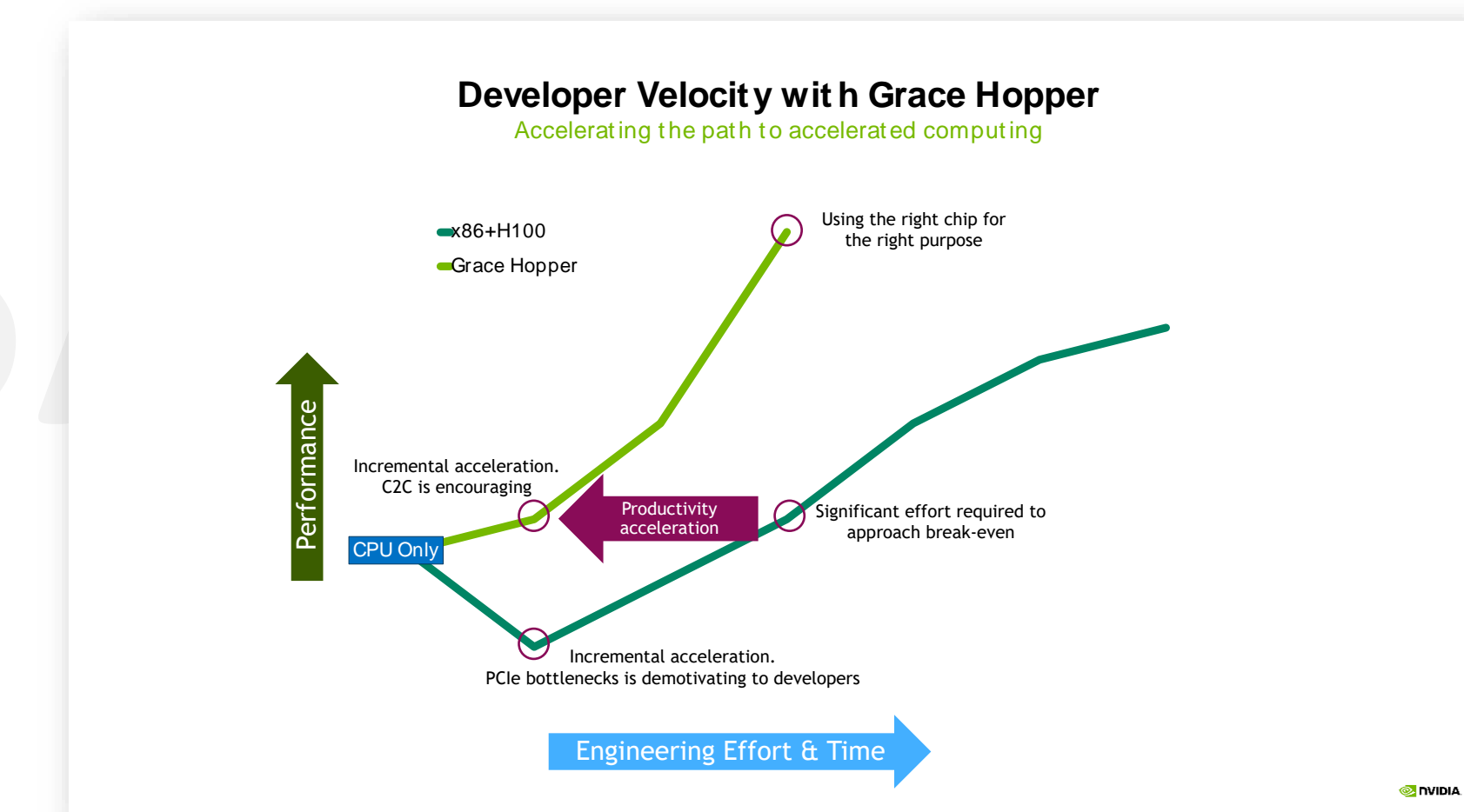


**Wrap Up**



# Conclusions

- NVIDIA Grace Hopper improves developer velocity
- Use the compilers, libraries, and tools you already use
  - ... as long as they are standards-compliant and multi-platform
- Expect software to work; expect software to perform well
- Tune Grace CPU performance with compilers and libraries
  - Update compiler flags
  - Use compiler autovectorization
  - Use de-facto standard library APIs like Netlib BLAS and FFTW
- Tune for coherent memory with memory allocators and CUDA UM hints
  - Use CUDA-managed memory to unlock coherent memory capability
  - Use CUDA unified memory hints to improve performance



**Grace UVM Migration Enhancements: CUDA C++ & CUDA Fortran**  
Maximum portable performance to NVIDIA HW out-of-the-box & without any changes

- No programming model changes!
  - No new APIs
  - No changes to existing APIs
  - No source code changes
- Unified Memory
  - Available on most platforms supported by CUDA 12.x: GH, P9+V100, PCIe x86 & Arm, etc.
  - Same Unified Memory Programming Model for all platforms: "memory accesses just work" + "hints".
- Unified Memory Hints
  - Hints only impact performance, not results.
  - `cudaMemAdvise` hints: PreferredLocation, AccessedBy, ReadMostly.
  - `cudaMemPrefetch` hints: prefetch to NUMA node.
  - Works with `cudaMallocManaged` memory on all supported Unified Memory platforms.
  - Work with system allocated memory (e.g. `malloc`) on Grace Hopper and systems with HMM.

Memory	Placement	Access-based Migration	Accessible From	
			CPU	GPUs
System-allocated (malloc, mmap)	First-touch (GPU   CPU)	❌	❌	❌
CUDA managed (cudaMallocManaged)		✅	✅	✅
CUDA device memory (cudaMalloc)	GPU	❌	❌	✅
CUDA host memory (cudaMallocHost)	CPU	❌	✅	❌

...and many others: interprocess, virtual, fabric, ...

**CUDA Unified Memory Hints**

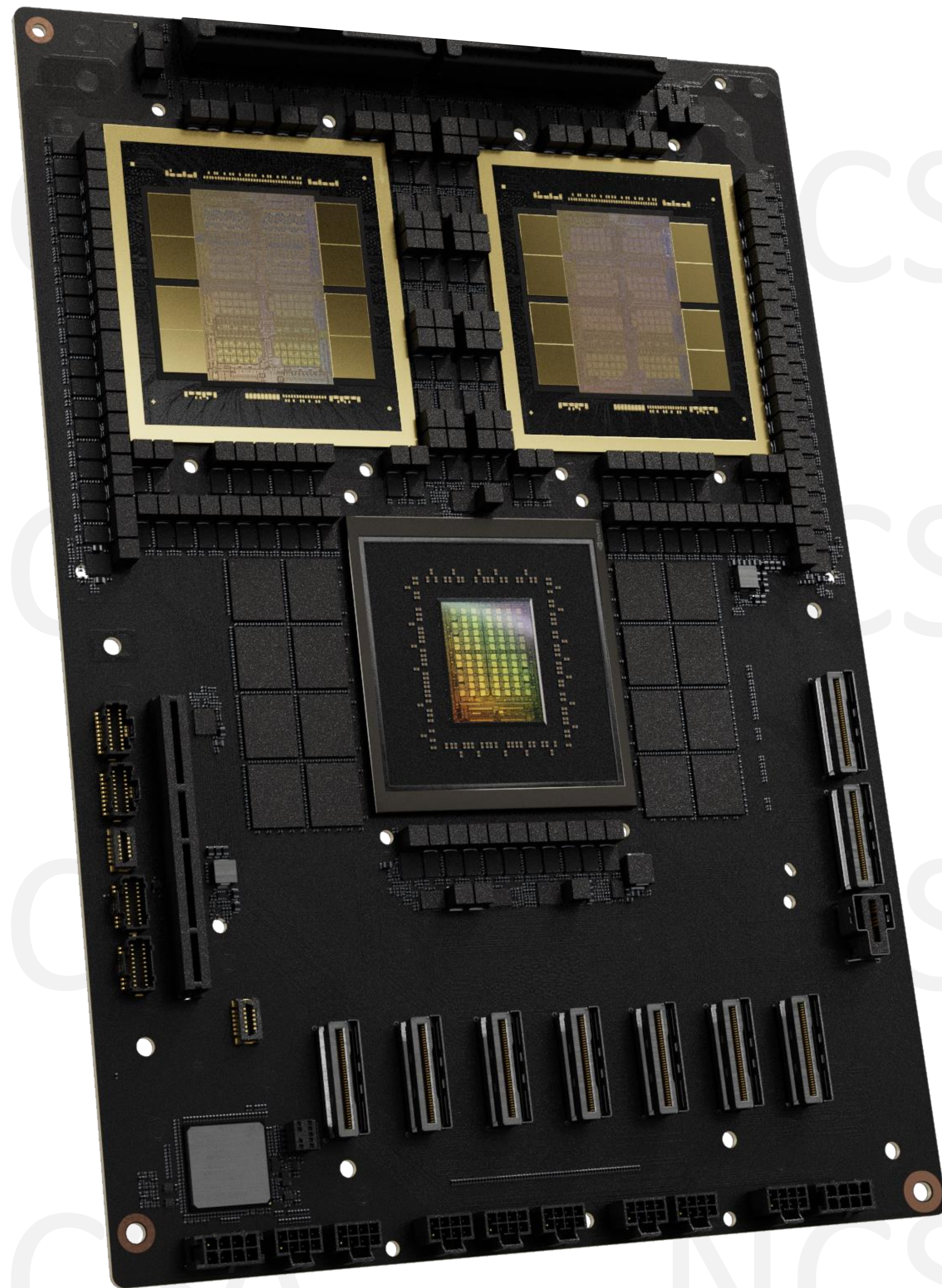
```

cudaMemAdvise(ptr, nbytes, advice, device);
Advices PreferredLocation AccessedBy ReadMostly
Devices GPU_id CPU CPU_Numa_Node
cudaMemPrefetchAsync(ptr, nbytes, destination, stream);
Destinations GPU_id CPU CPU_Numa_Node
  
```



# GB200 SUPERCHIP

Optimized for Supercomputer-Scale Science



72 Grace CPU Arm cores

40 PetaFLOPS FP4 AI Inference

20 PetaFLOPS FP8 AI Training

16 TB/s of GPU memory bandwidth

864 GB Fast Memory



# DGX GB200

Delivers New Unit of Compute



## DGX GB200

36 GRACE CPUs  
72 BLACKWELL GPUs  
Fully Connected NVLink Switch Rack

Training  
Inference  
NVL Model Size  
Multi-Node All-to-All  
Multi-Node All-Reduce

720 PFLOPs  
1,440 PFLOPs  
27T params  
130 TB/s  
260 TB/s







# Grace CPU Benchmarking Guide

<https://nvidia.github.io/grace-cpu-benchmarking-guide/>

