Deep Dive into NVIDIA Grace CPU

architecture, performance, optimizations

NCHC 2025

- NVIDIA Grace Superchip
- Grace Software Ecosystem
- Getting started with Grace
- Performance Analysis
- GH200 “Grace Hopper” Superchip

Arm is the most pervasive and adopted computer architecture

Arm is Building the Future of Computing

- Arm is the world's most pervasive CPU architecture
- Everything today is a computer – CPUs needed everywhere
- Strong growth, highly profitable and cash generative company

285+ Billion
Arm-based chips shipped since inception

28.6 Billion
Arm-based chips reported as shipped in FYE24

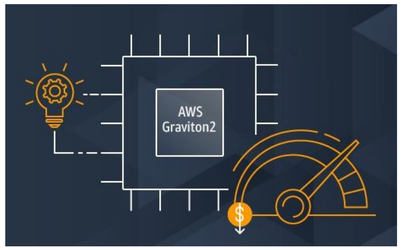
18M+
Software Developers on Arm

arm

Source: Arm Holdings plc Q4 FYE24 Results (May 2024) -- https://investors.arm.com/static-files/4f2fc46b-34a5-4bc5-94af-f13fbc348f0e

The Arm future is here already

ARM is on the desktop and in the datacenter **today**



Expanding availability of ARM CPUs

Alibaba.com

AMPERE

aws

FUJITSU

NVIDIA

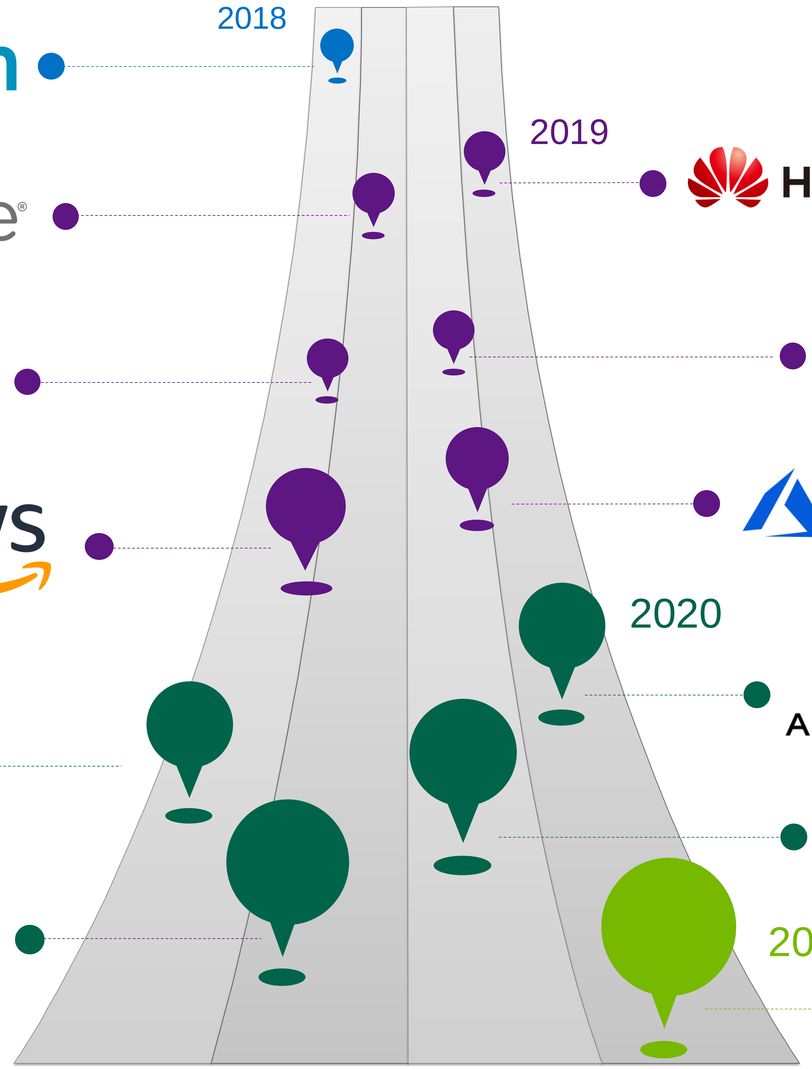
SIPEARL

Broad SW ecosystem support backed by Arm standards

>97%

Native Package Coverage on DEBIAN & UBUNTU

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

Ain’t our first rodeo …

Pioneering Arm CPU as host: Marvell ‘ThunderX2’ CPU + NVIDIA V100 @ ORNL [2019]

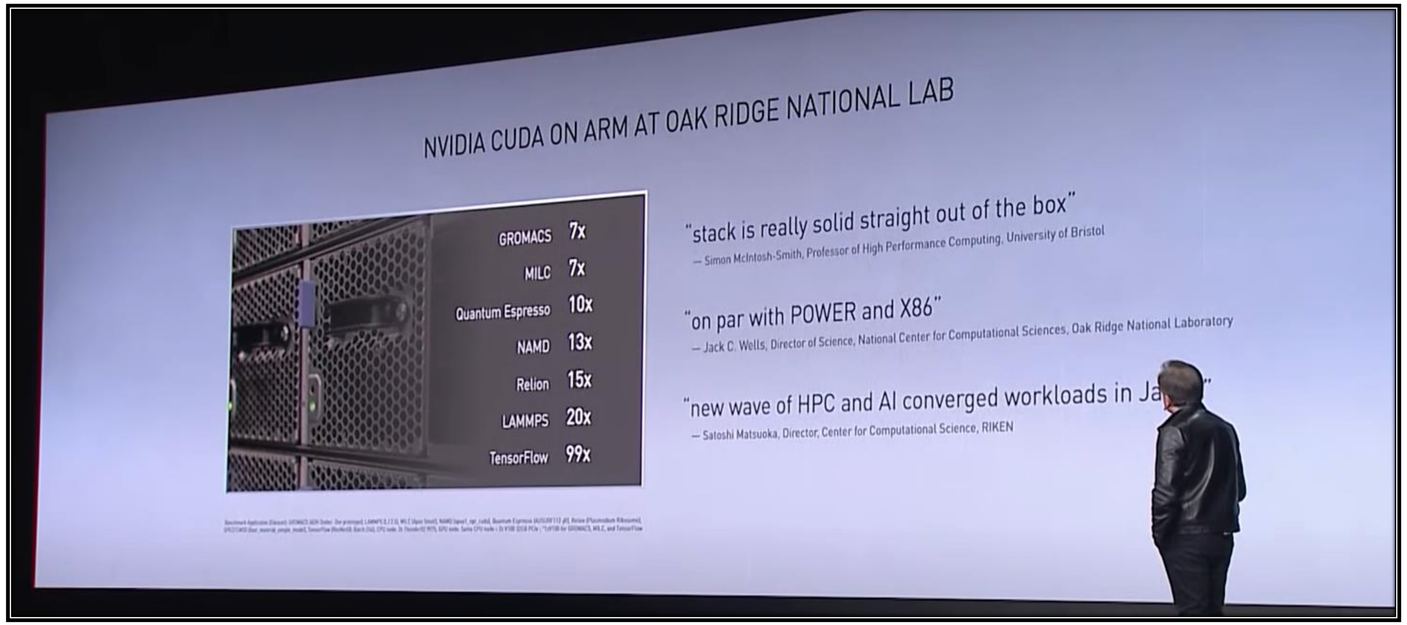
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

OLCF upgrades Arm test bed with NVIDIA GPUs and new NVIDIA CUDA software to explore new avenues in HPC

Alongside the regular high-performance computing (HPC) resources at supercomputing centers are test beds, small computing clusters that offer HPC experts a chance to explore new architectures before they are considered for deployment at a larger scale. Last year, the Oak Ridge Leadership Computing Facility (OLCF) ventured into new HPC territory when it installed the Arm-based test bed Wombat, powered by pre-production Cavium (now Marvell) ThunderX2 CPU processors and AMD GPUs.

Now, the OLCF has upgraded Wombat, installing production Marvell ThunderX2 CPUs and NVIDIA V100 GPUs to test NVIDIA’s new CUDA software stack purpose-built for Arm CPU systems. In late October, immediately after the upgrade, eight teams successfully ported their codes to the new system in the days leading up to the 2019 Supercomputing Conference, SC19. In less than 2 weeks, eight codes in a variety of scientific domains were running smoothly on Wombat.

“At the end of the day, everything that we tried was successful,” said Oscar Hernandez, a computer scientist at the OLCF. “We didn’t have any applications that failed or couldn’t deliver at the time we wanted them to.”

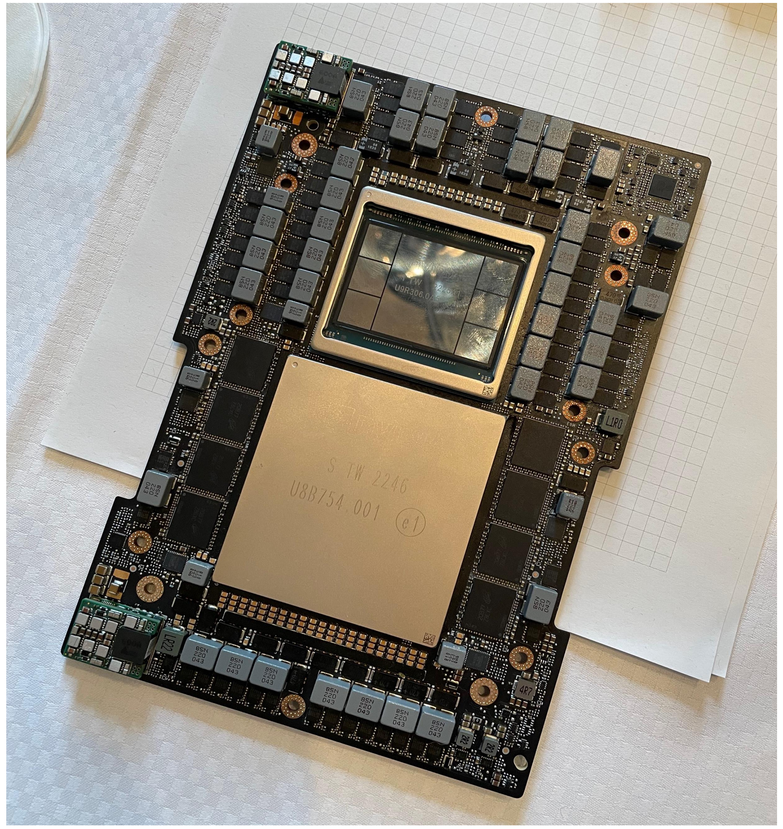


SC19 Jensen Huang special address

Source: https://www.olcf.ornl.gov/2020/01/17/eight-teams-successfully-running-on-first-nvidia-and-arm-test-bed-wombat-at-the-olcf/

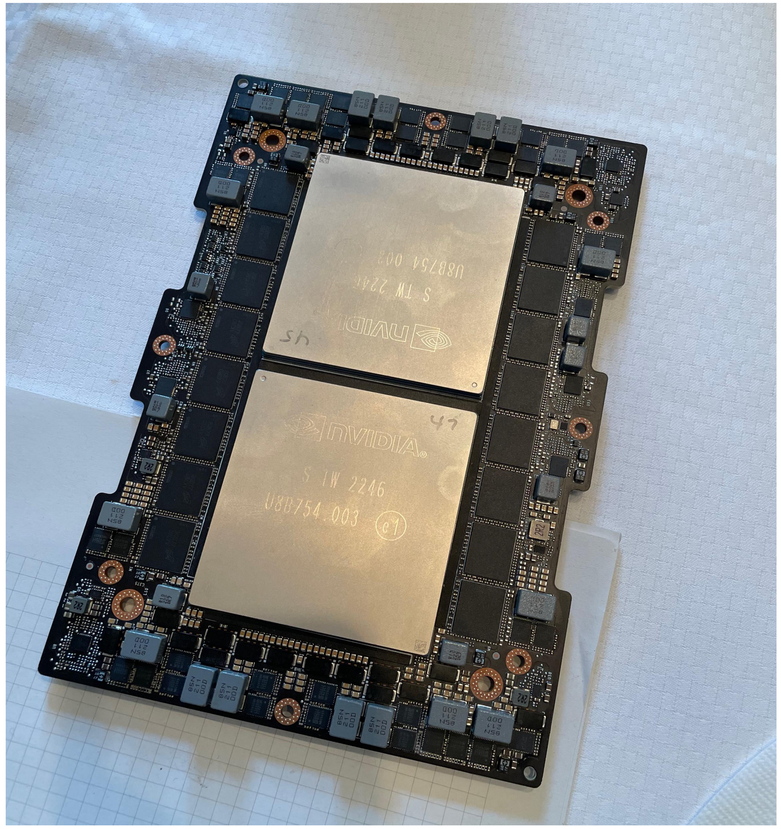
NVIDIA Grace Superchip

NVIDIA Grace for HPC & AI Infrastructure



Grace Hopper Superchip

Giant Scale AI & HPC



Grace CPU Superchip

CPU Computing

NVIDIA Grace CPU Superchip

2X Performance at the Same Power for the Modern Data Center

High Performance Power Efficient Cores

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

Fast On-Chip Fabric

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

High-Bandwidth Low-Power Memory

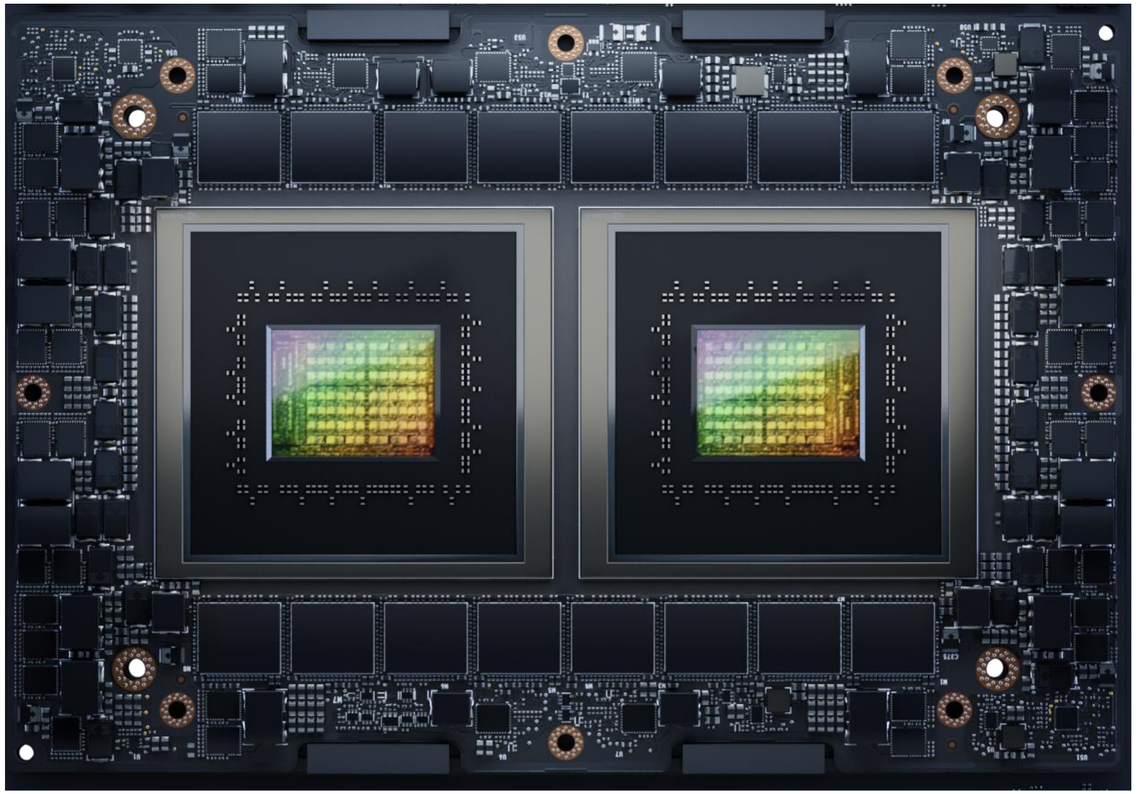
Up to 960GB of data center enhanced LPDDR5X Memory that
delivers up to1TB/s of memory bandwidth

Fast and Flexible CPU IO

Up to 8x PCIe Gen5 x16 interface. PCIe Gen 5 up to 128GB/s
2X more bandwidth compared to PCIe Gen 4

Full NVIDIA Software Stack

AI, Omniverse



NVIDIA Grace vs. Fujitsu A64FX

A64FX is special in many way – Grace is mainstream

MAINSTREAM LEADERSHIP HPC

Familiar design	• High single-thread performance • Simple memory hierarchy
Large user community	• Runs key HPC applications out-of-the-box • Standard best practices hold true
Significant fraction of peak w/o tuning	• OSS toolchains (i.e. GNU) are tuned for u-arch • Performance curves generally follow expectation

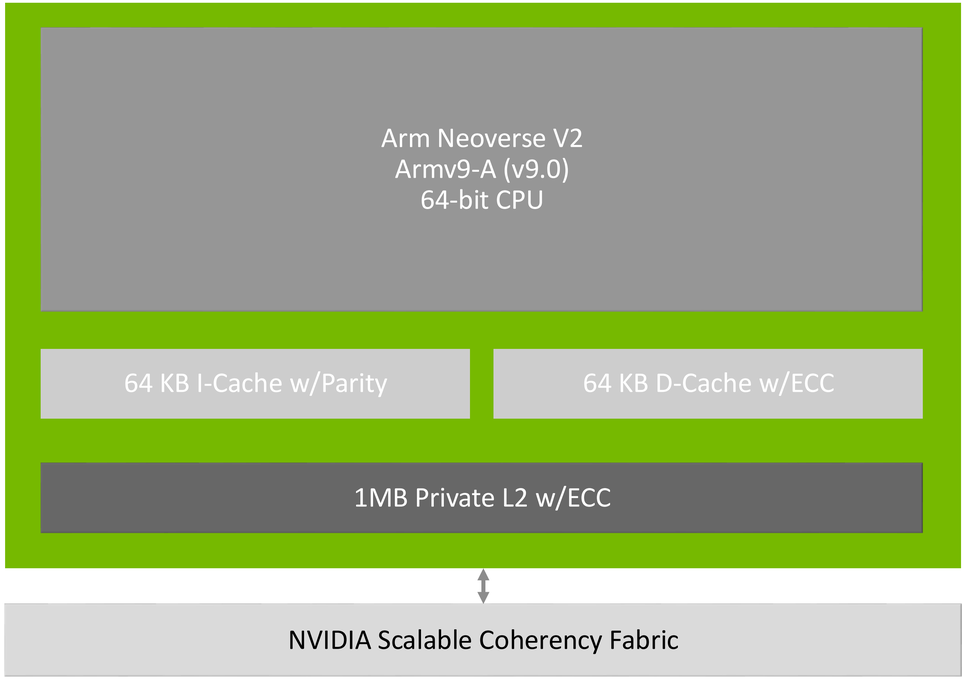
EXTREME HPC CODESIGN

Codesigned for specific application	• Custom hardware or software • Trades generality for performance
Small userbase of extreme experts	• Nonstandard software environments • Common assumptions may hurt performance
Significant tuning effort required	• OSS toolchains unlikely to be performant • Plan for man-months of optimization effort

NVIDIA Grace

Introducing Neoverse V2

- **Arm Neoverse V2 core – Arm v9.0**
  - Extends Armv8.[0-5]-A architecture
- 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)

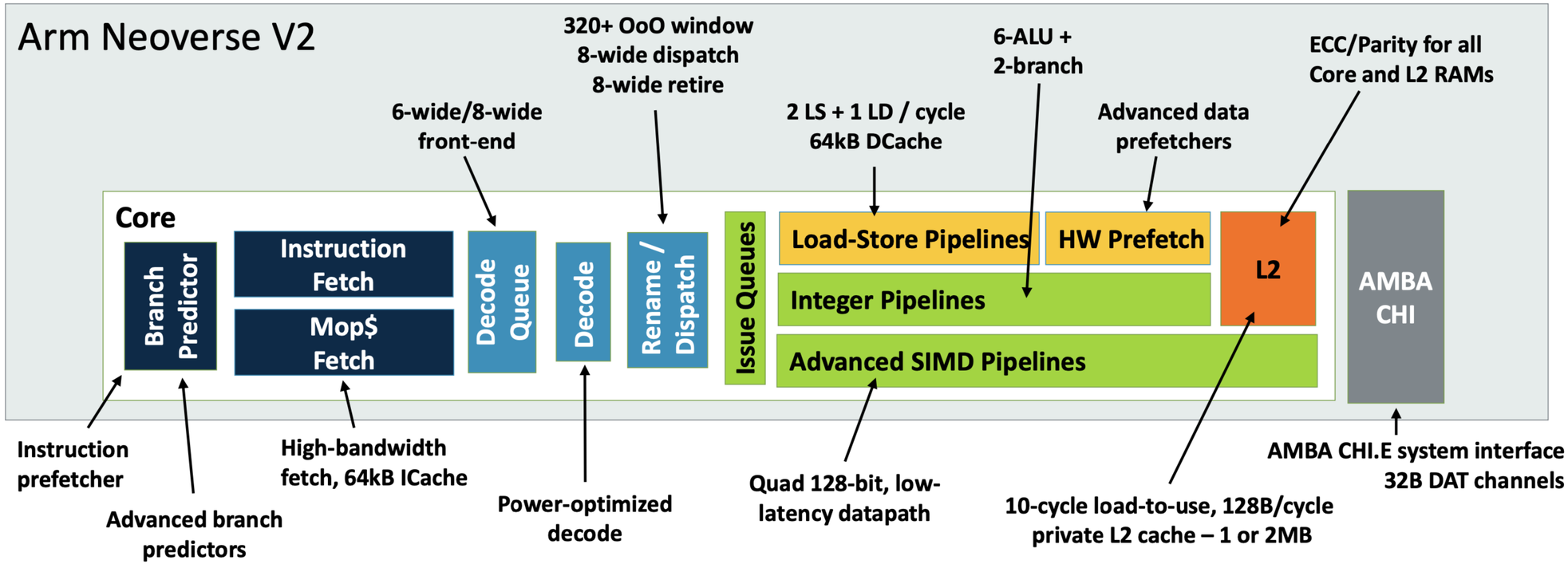


Arm Neoverse V2

- Grace CPU Neoverse V2 core implements the Armv9.0-A architecture:
  - Binary compatible with Armv8
  - Recompile for optimal performance
- Single instruction multiple data (SIMD) vector instruction sets in a 4x128-bit configuration:
  - Scalable Vector Extension version 2 (SVE2)
  - Advanced SIMD (NEON)
- Large System Extension (LSE) for atomic operations (Armv8.1):
  - Compare and Swap instructions, CAS, and CASP
  - Atomic memory operation instructions, LD<OP> and ST<OP>, where <OP> is ADD, CLR, EOR, SET, SMAX, SMIN, UMAX, or UMIN
  - Swap procedure, SWP
- Additional Armv9 Features:
  - Cryptographic acceleration, scalable profiling extension, virtualization extensions, secure boot

High-Level Core Microarchitecture

Arm Neoverse V2 IP



Double-digit gains over Neoverse V1 on cloud infrastructure workloads

- 13% uplift on SPEC CPU® 2017 Integer1
- 15% to 100% uplift across a range of server workloads (caching, web, database)

Up to 2x the performance of Neoverse V1 on HPC and ML workloads

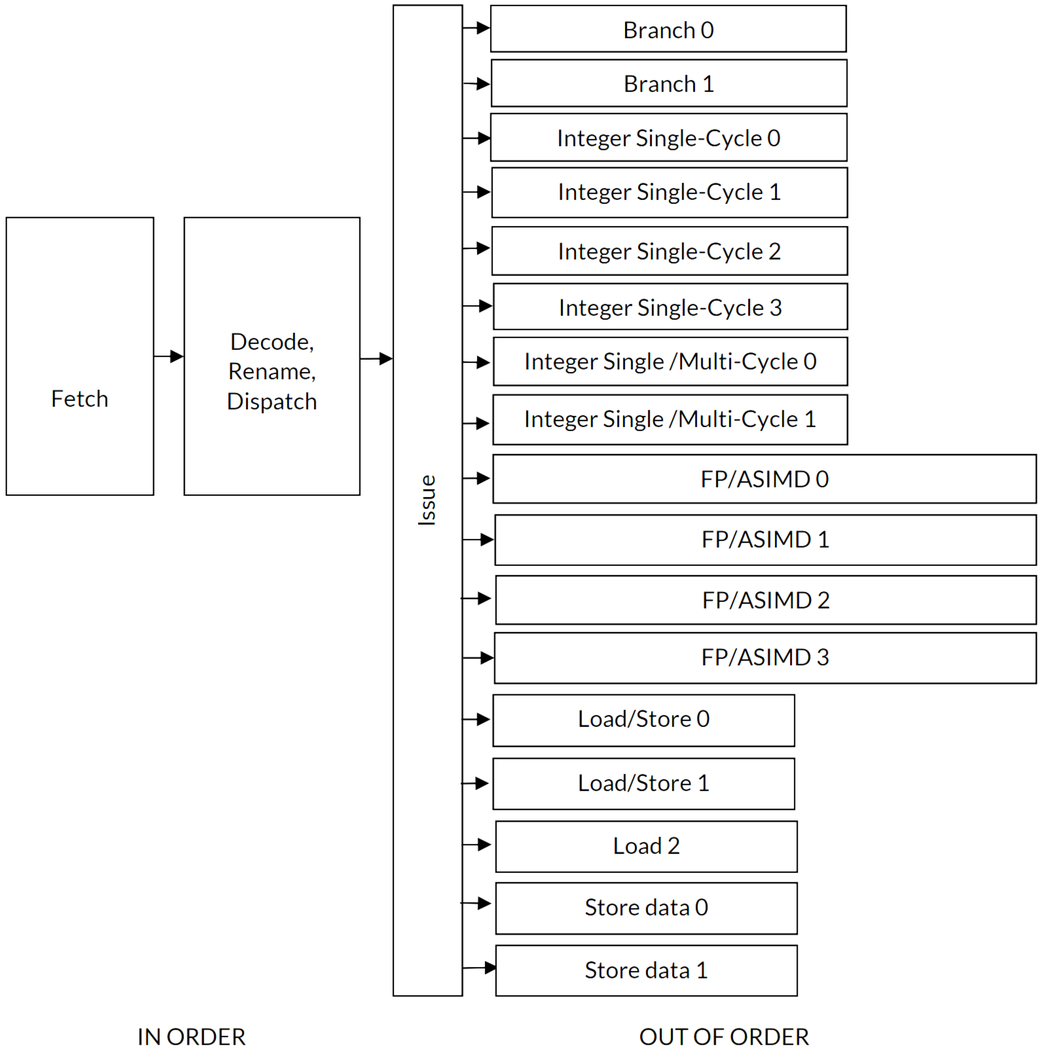
- Up to 114% uplift on XGBoost (83% average)
- Meets or exceeds leading x86-CPUs on performance with up to 2x the performance efficiency

Neoverse V2 Deep Dive by Arm Ltd @ HotChips 2023: https://hc2023.hotchips.org/assets/program/conference/day1/CPU1/HC2023.Arm.MagnusBruce.v04.FINAL.pdf

NVIDIA Grace SIMD Highlights

Neoverse V2 Arm IP core

- 4x128b SIMD units = 512b SIMD vector bandwidth total
- Each SIMD unit can retire NEON or SVE2 instructions
- **On this architecture, SVE2 and NEON have the same peak performance**
  - *Different from Fujitsu A64FX where SVE 4x faster than NEON*
- SVE2 can vectorize more complex codes and supports more data types than NEON.
- NEON doesn't require predicate calculation
  - *Neither does VLS SVE, but that's an advanced topic*

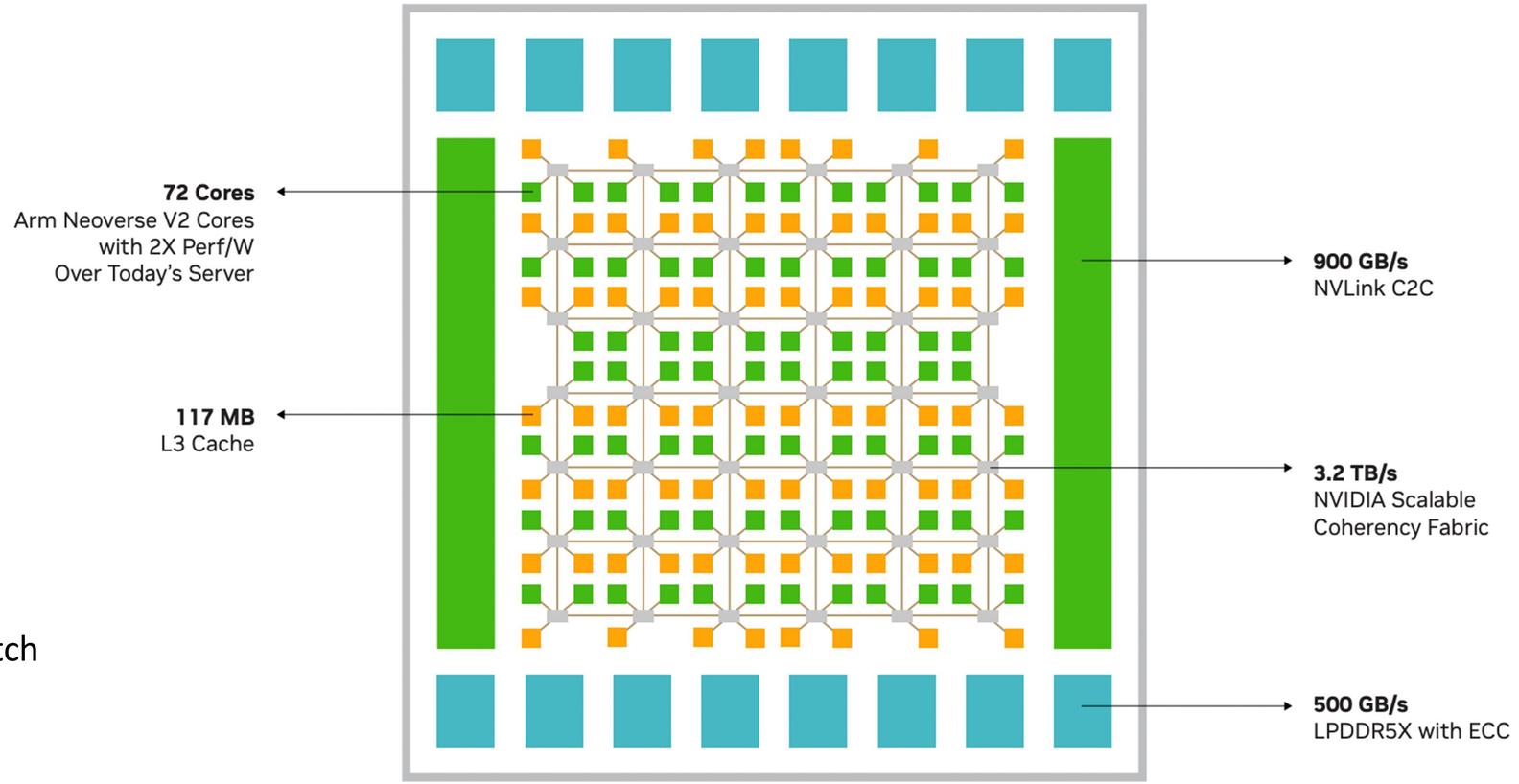


For more in-depth core u-arch details: Arm® Neoverse™ V2 Core Technical Reference Manual

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

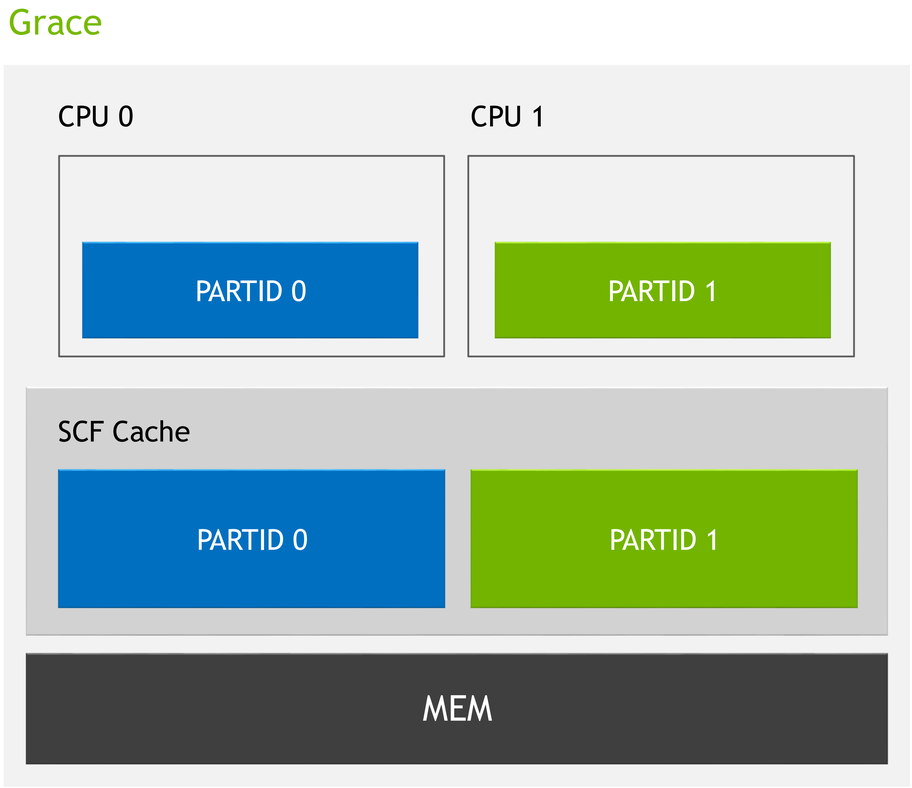


Example possible fabric topology for illustrative purposes

Scalable Coherent Fabric (SCF)

Memory Partitioning and Monitoring

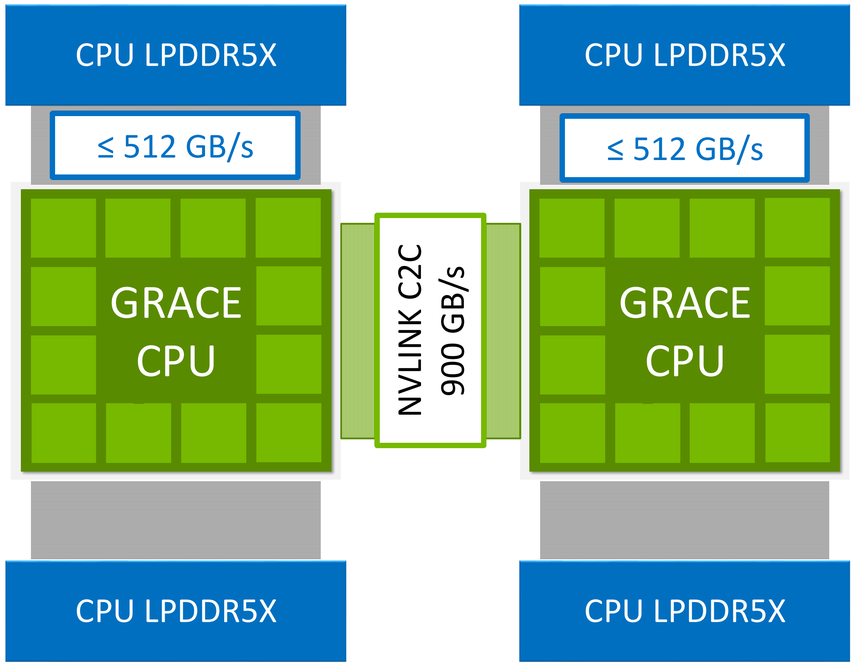
- Arm standard for partitioning system resources
- Partition IDs (PARTID) are assigned to entities making requests to memory
- SCF Cache resources can be partitioned between different PARTIDs
- Partition both Cache capacity, and Memory Bandwidth
- Performance Monitor Groups (PMG) can be used to monitor resource usage



Low-Power High-Bandwidth Memory Subsystem

LPDDR5X Data Center Enhanced Memory

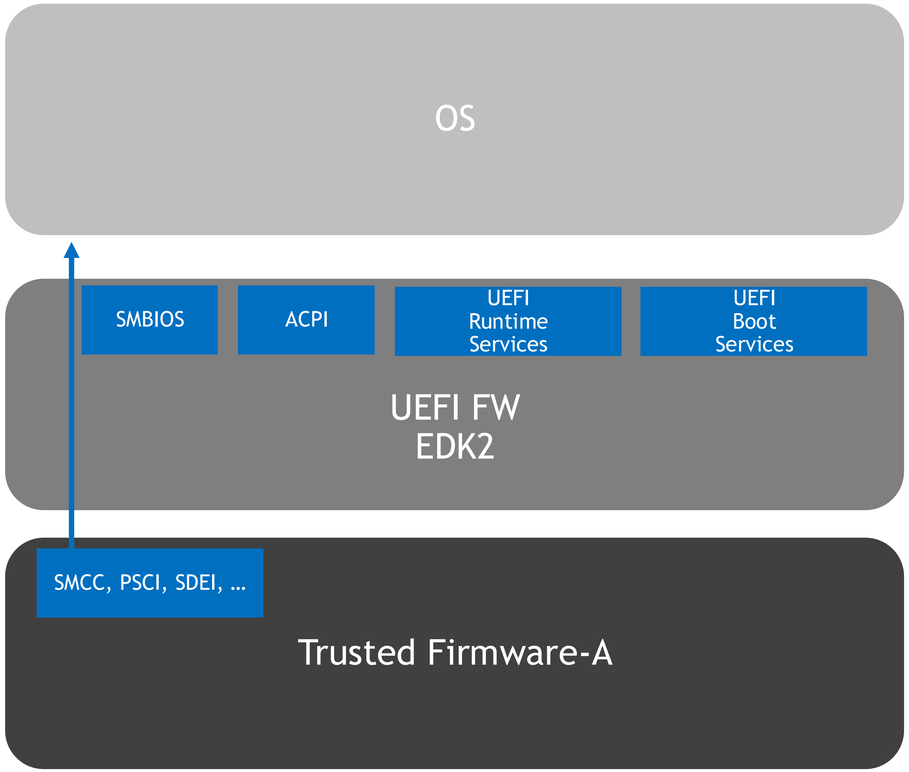
- Optimal balance between bandwidth, energy efficiency and capacity
- Up to 1TB/s of raw bidirectional BW
- 1/8th power per GB/s vs conventional DDR memory
- Similar cost / bit to conventional DDR memory
- Data Center class memory with error code correction (ECC)



Grace

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

- Standard set of platform requirements and recommendations to enable off-the-shelf OS support
- 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

Grace Simplifies System Design and Workload Optimization

Reduces NUMA Bottlenecks

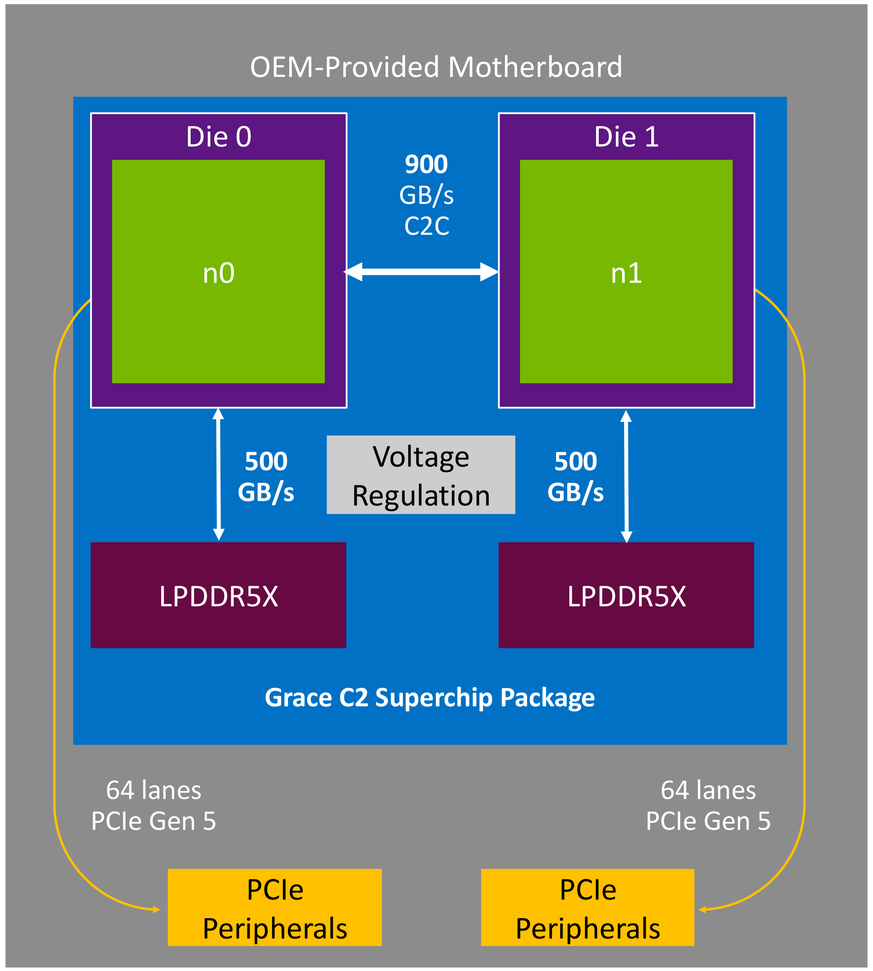
2
NUMA Nodes

2
Compute Dies

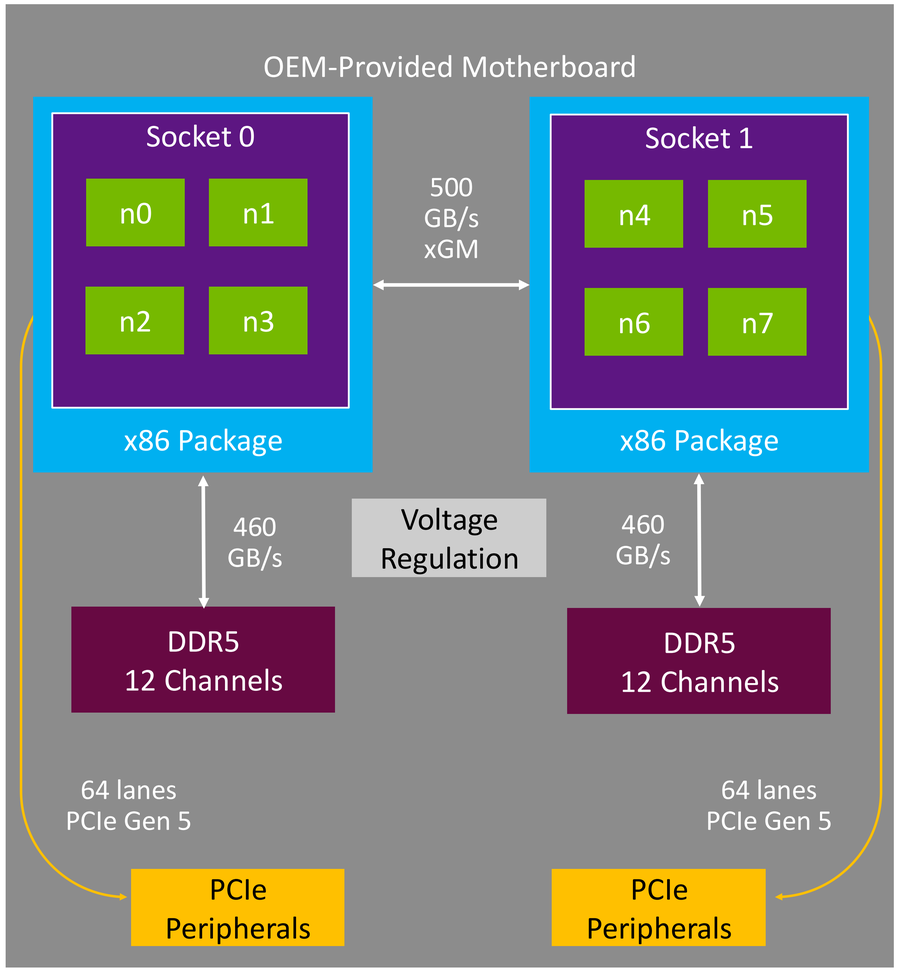
500
Watts
(CPU + MEM)

900
GB/s n-to-n

Grace Server
Grace C2 Superchip



Conventional 2-Socket Server
Example: 2x AMD Genoa, Native NPS=4



8
NUMA Nodes

24
Compute Chiplets

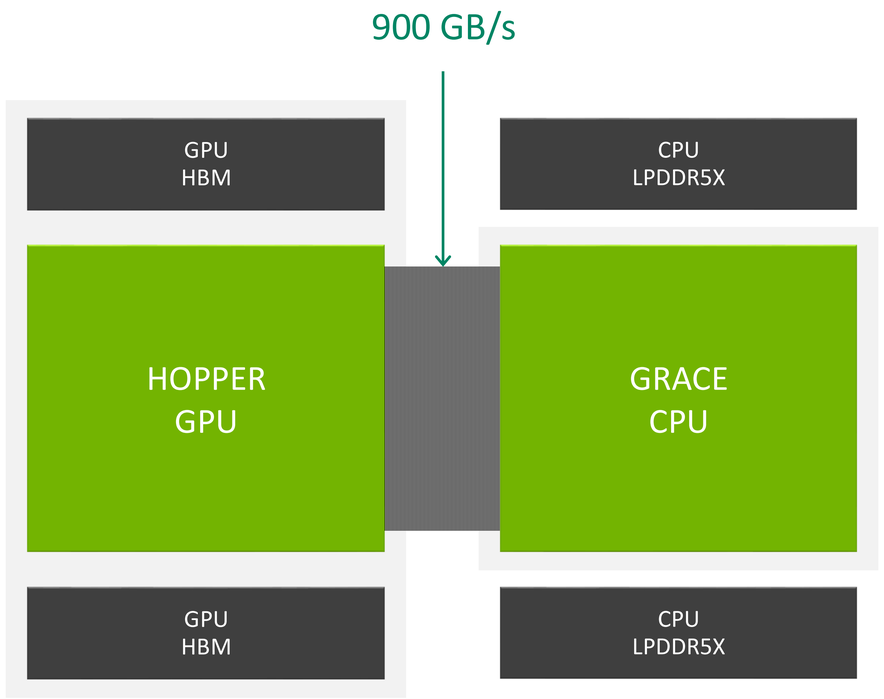
900+
Watts
(CPU + MEM)

500
GB/s n-to-n

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)**



Two Memory Systems, Each Optimized For Its Processor

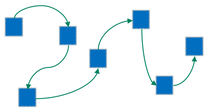
Act as a Single Memory Pool for Data Intensive Applications

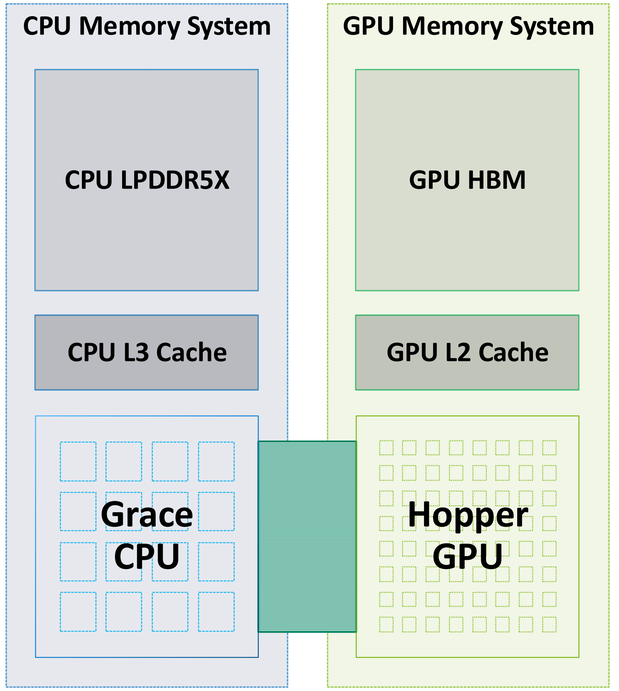
Benefits

- Faster data transfer
- Memory coherency with simplified programming
- Power efficient performance

CPU memory system is optimized for **low latency** and **deep cache hierarchy**

Run **latency-sensitive** code on the CPU, e.g., a linked list



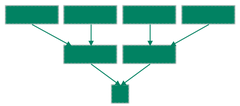


NVLink-C2C

CPU and GPU each have **full coherent access** to memory

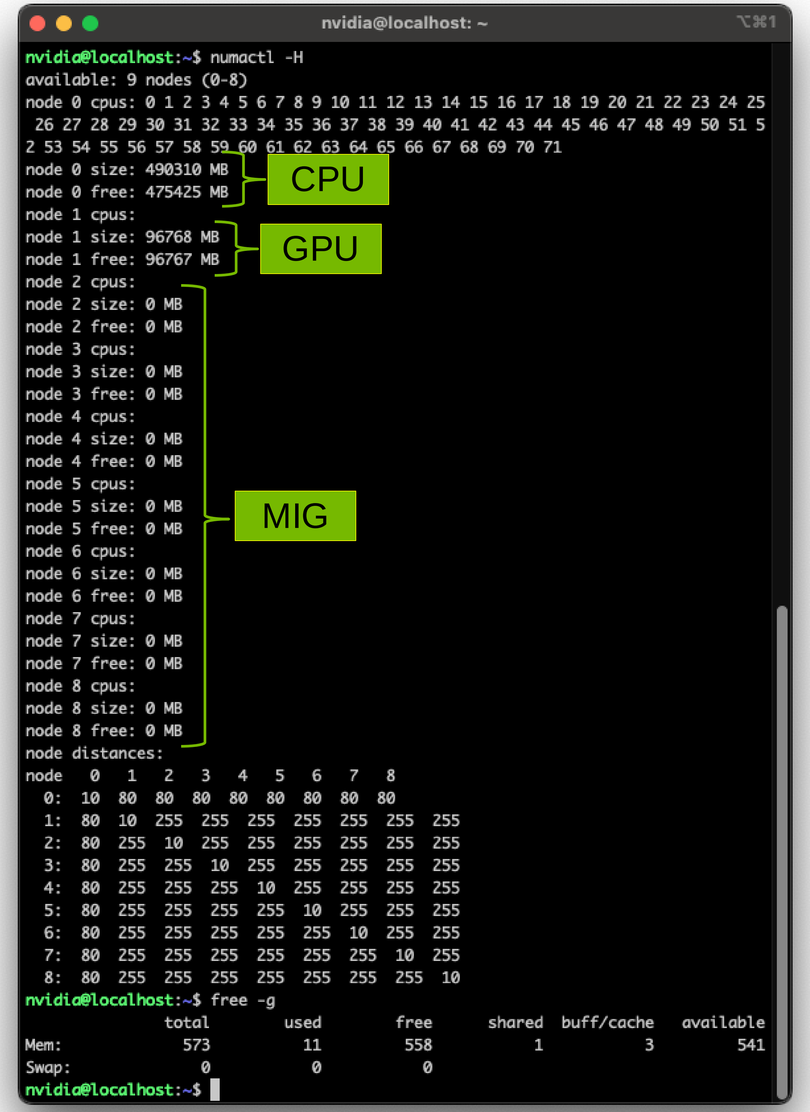
GPU memory system is optimized for **high throughput** and **high bandwidth cache**

Data- and math-intensive code on the GPU, e.g., vector reduction



GPU Memory is Visible to the Operating System

Standard operating system commands work on the GPU




```
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
```

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

GH200 GRACE HOPPER SUPERCHIP

The breakthrough accelerated CPU for Large-Scale AI and HPC applications

Grace CPU + H100 GPU

72 Arm Neoverse V2 Cores with SVE2 4x128b
Transformer Engine and ~4PFLOPS of FP8

Fast NVLink-C2C Connection

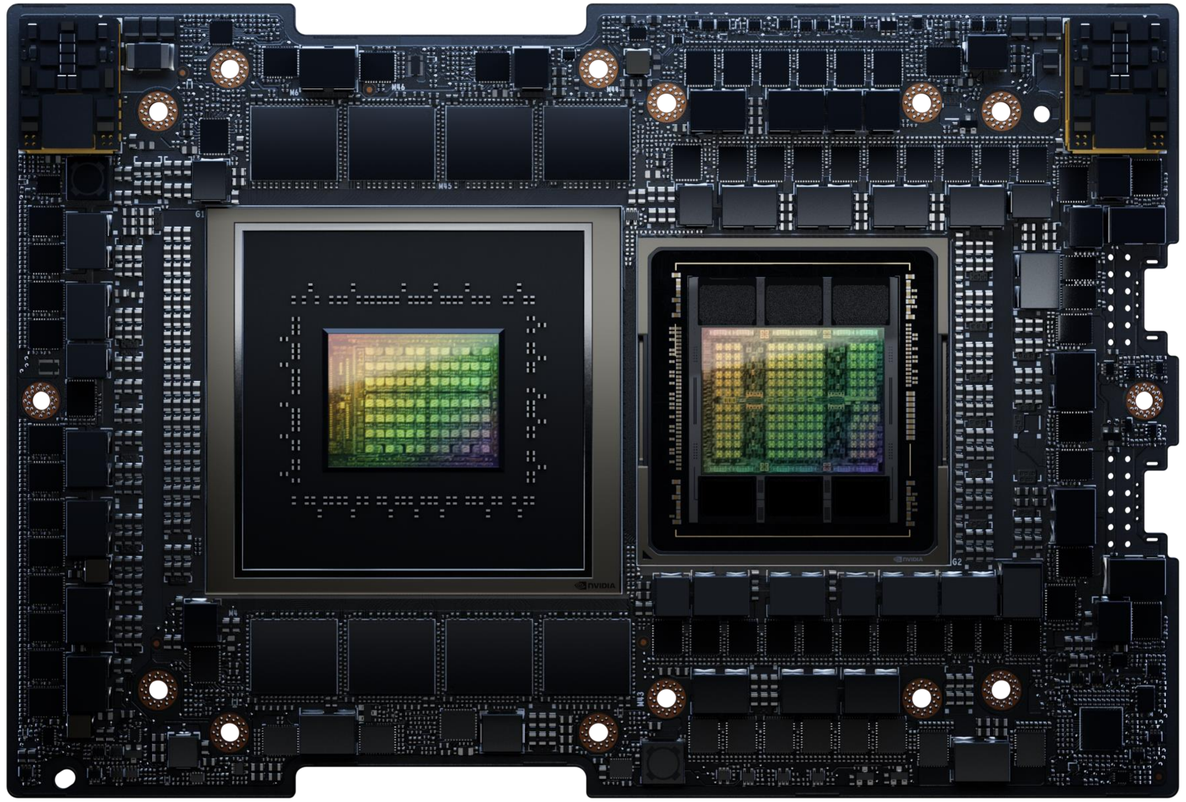
900GB/s bi-directional bandwidth CPU to GPU
7X faster than PCIe Gen 5

~600GB of Fast Access Memory

Up to 96GB HBM3, 4TB/s bandwidth
Up to 480GB LPDDR5X, 512GB/s bandwidth

Full NVIDIA Compute Stack

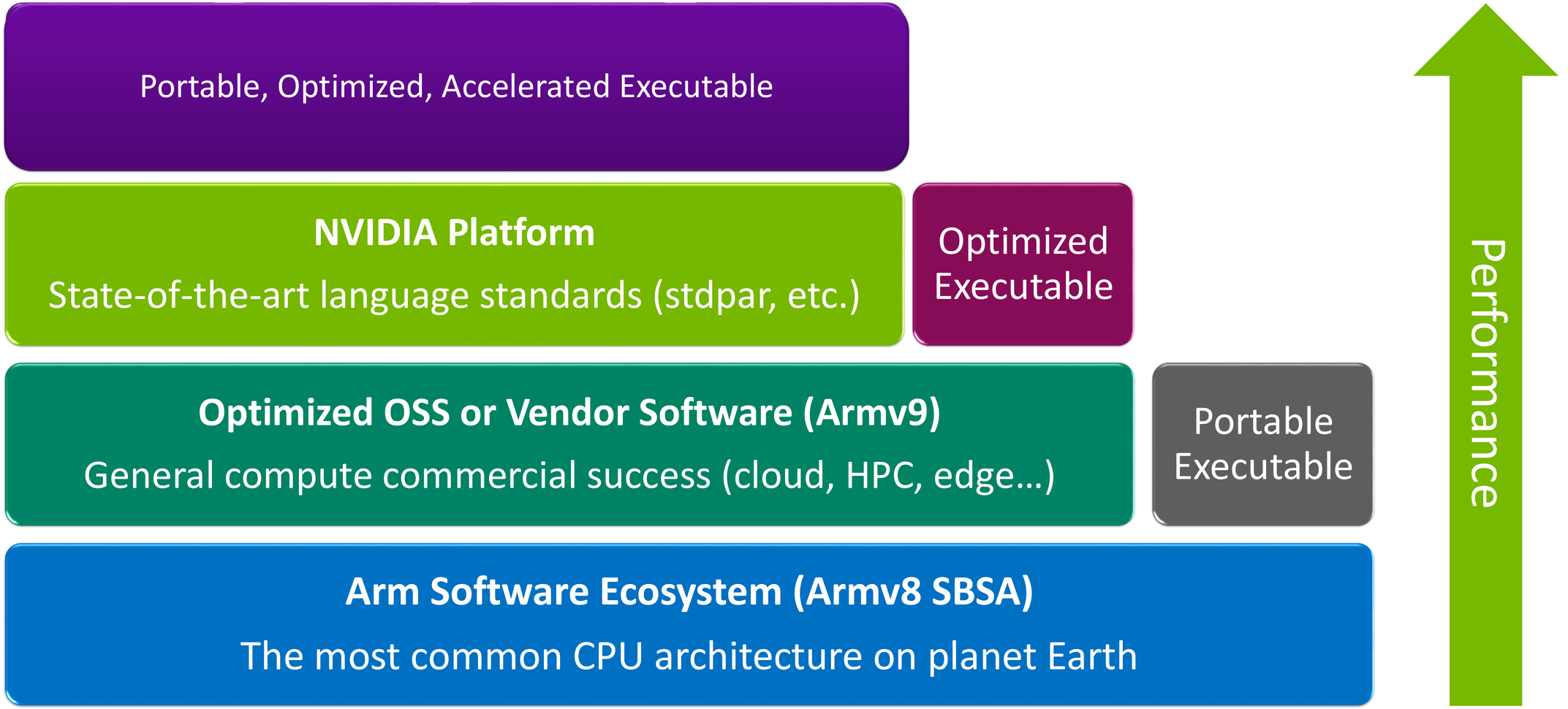
AI, Omniverse



Grace Software Ecosystem

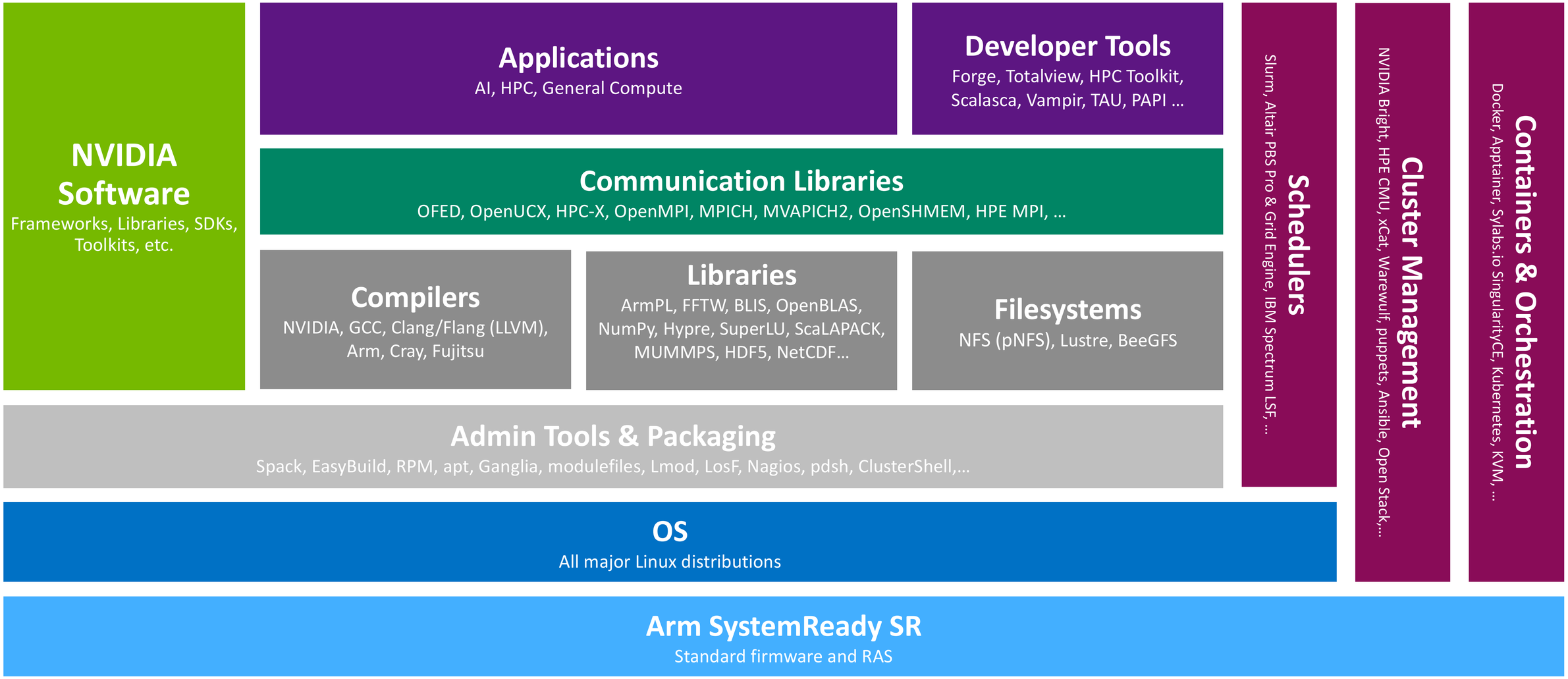
Grace Software Ecosystem is Built on Standards + NVIDIA's Ecosystem

Grace brings the full NVIDIA software stack to Arm.



NVIDIA Grace HPC/AI Software Ecosystem

Full support for the broad Arm software ecosystem, both open source and commercial



Arm Neoverse Software Ecosystem Supports NVIDIA Grace

100s of OSS

Native Build Projects

100+ ISVs

Commercial Support

90% CNCF

Graduated & Incubated

Docker Build Cloud

Native Multi-Arch builds

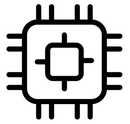
PERFORMANCE

Scalable Neoverse CPU Cores

EFFICIENCY

Best $/throughput in the industry

PLATFORM DIVERSITY



Widest possible choice of Platforms

OPTIMIZATION

Purpose built use cases

Sources:
Getting Started with Grace: https://docs.nvidia.com/grace/index.html
AWS Graviton Resources: https://github.com/aws/aws-graviton-getting-started
Arm Blog: https://cloudnativenow.com/topics/building-the-future-of-cloud-native-software-on-a-multi-architecture-infrastructure/
Docker Hub: https://hub.docker.com/

arm NVIDIA

Recommended Software Support

SW Configurations recommended for the best performance for NVIDIA CPUs
Compliant with all Arm ecosystem software

Operating System	Virtualization	Compilers	NVIDIA Tools / Libraries (Optional)
For custom kernels: Kernel 6.4[1]	Additional information coming in upcoming updated	GCC 12.3+	NVIDIA Performance Libraries (NVPL)[3]
Ubuntu LTS 24.04		LLVM 16.0.5+[2]	NVIDIA NSIGHT
RHEL 9.4			NVIDIA HPC SDK
SUSE 15 SP6			

NVIDIA CPUs are fully supported on most of our NGC and NIMs containers[4]

[1] For details on the recommended software patches and configurations
[2] Recommend using optimized build of LLVM Clang for Grace. Certified CUDA host compiler
[3] Optimized math libraries for NVIDIA CPUs. Easily port applications to Nvidia CPUs, drop-in replacement for any math library. New interfaces for high perf libraries
[4] Grace CPU support for NIM containers available ~Q1'25

NVIDIA’s Approach to CPU Compilers for Grace

Use the compilers you know and love

- **NVHPC and NVCC**
  - **23.5+ supports Grace and Neoverse V2**
  - Continuously improving performance
- **LLVM Clang**
  - **LLVM16+ supports -mcpu=neoverse-v2**
  - NVIDIA will provide supported builds of Clang as drop-in replacements for mainline Clang
  - NVIDIA contributing to LLVM in open source to maximize Grace application performance
  - Engaging with Arm (Inc.) to improve performance for key CPU benchmarks and applications
- **GCC**
  - **GCC12.2+ supports -mcpu=neoverse-v2**
  - Engaging with Arm (Inc.)
  - Already finding and fixing bugs in mainline GCC

NVHPC

- NVIDIA’s innovation space
- Focus on value for workloads

LLVM / Clang

- Required for an excellent CPU software ecosystem
- A foundation for potential compiler innovations

GCC

- Required for an excellent CPU software ecosystem
- Often the default; sets performance expectations

NVIDIA Performance Libraries (NVPL)

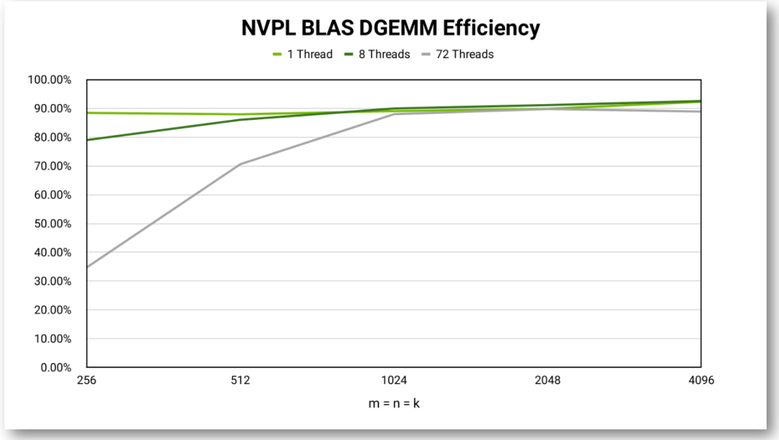
Optimized math libraries for NVIDIA CPUs

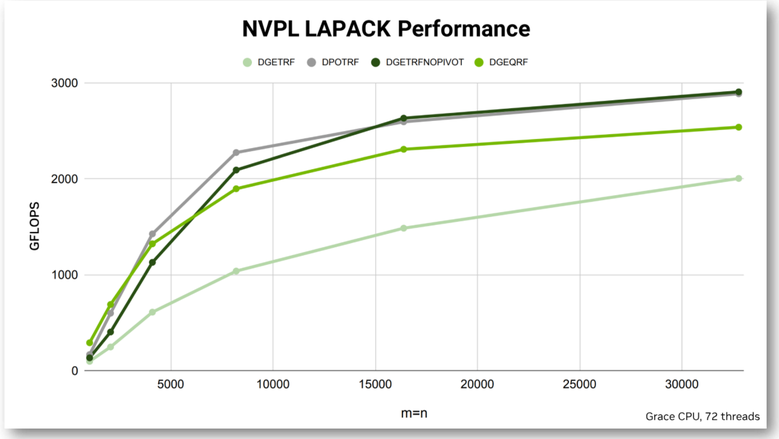
- Easily port applications to NVIDIA's Arm CPUs
- Drop-in replacement for any math library implementing standard interfaces (e.g. Netlib, FFTW)
- New interfaces for high-performance libraries

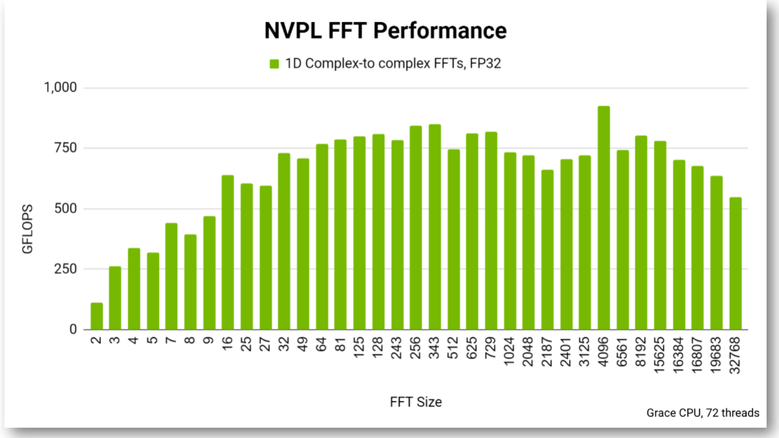
BLAS	LAPACK	PBLAS	SCALAPACK
TENSOR	SPARSE	RAND	FFT

Download Now

www.developer.nvidia.com/nvpl



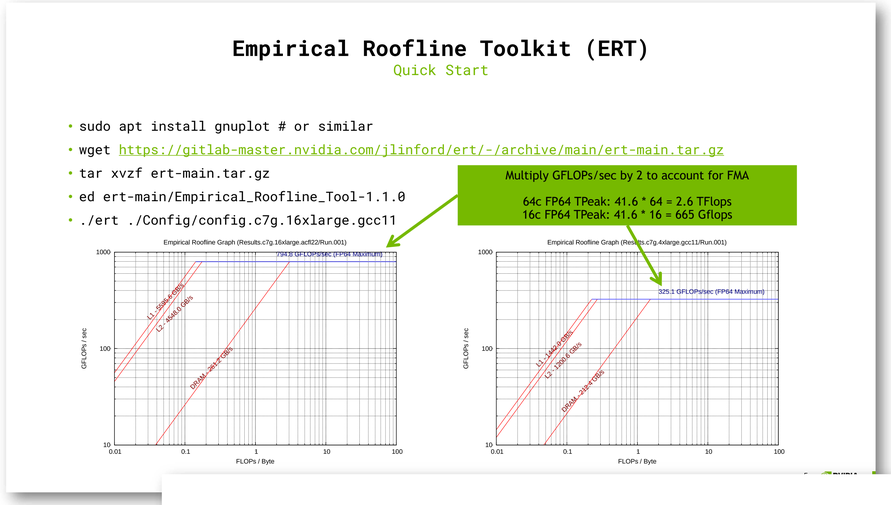


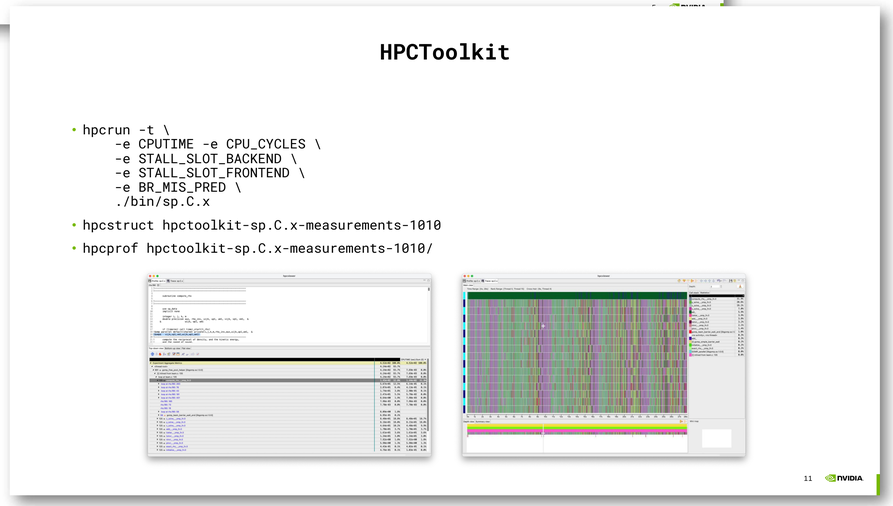


Performance Engineering Tools

A mix of NVIDIA tools and community tools – developers want to use the same tools on Grace

- **Objective**: Enable Grace developers to the same extent that they are enabled on x86
- The NVIDIA NSIGHT team are defining CPU profiling workflows that enable Grace developers
- NVIDIA is working with community tools to better support Grace
  - E.g. PAPI and libpfm4 now support Grace in-core counters





PAPI & libpfm4 with Arm Neoverse Support

Upstreamed and maintained by the community

- `cd src`
- `./configure --prefix=$PWD/..`
- `make -j && make install`

```
[jlinford@c7g-4xlarge-dy-c7g4xlarge-2 papi-jlinford-arm-neoverse]$ ./bin/papi_native_avail | grep SVE
|           Data memory read accesses (includes SVE)                          |
|           Data memory write accesses (includes SVE)                         |
|           Unaligned accesses (includes speculatively executed SVE load and s|
| SVE_INST_SPEC                                                               |
|           SVE operations sepculatively executed                             |
| SVE_PRED_SPEC                                                               |
|           SVE predicated operations speculatively executed                  |
| SVE_PRED_EMPTY_SPEC                                                         |
|           SVE predicated operations with no active predicates speculatively |
| SVE_PRED_FULL_SPEC                                                          |
|           SVE predicated operations with all active predicates speculatively|
| SVE_PRED_PARTIAL_SPEC                                                       |
|           SVE predicated operations with partially active predicates specula|
| SVE_LDFF_SPEC                                                               |
|           SVE first-fault load operations speculatively executed            |
| SVE_LDFF_FAULT_SPEC                                                         |
|           SVE first-fault load operations speculatively executed which set F|
```

```
[jlinford@c7g-4xlarge-dy-c7g4xlarge-2 papi-jlinford-arm-neoverse]$ ./bin/papi_avail | grep Yes
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   No   Level 2 data cache misses
PAPI_TLB_DM  0x80000014  Yes   No   Data translation lookaside buffer misses
PAPI_L2_LDM  0x80000019  Yes   No   Level 2 load misses
PAPI_STL_ICY 0x80000025  Yes   Yes  Cycles with no instruction issue
PAPI_HW_INT  0x80000029  Yes   Yes  Hardware interrupts
PAPI_BR_MSP  0x8000002e  Yes   No   Conditional branch instructions mispredicted
PAPI_BR_PRC  0x8000002f  Yes   Yes  Conditional branch instructions correctly predicted
PAPI_TOT_INS 0x80000032  Yes   No   Instructions completed
PAPI_FP_INS  0x80000034  Yes   No   Floating point instructions
PAPI_LD_INS  0x80000035  Yes   No   Load instructions
PAPI_SR_INS  0x80000036  Yes   No   Store instructions
PAPI_BR_INS  0x80000037  Yes   No   Branch instructions
PAPI_VEC_INS 0x80000038  Yes   Yes  Vector/SIMD instructions (could include integer)
PAPI_RES_STL 0x80000039  Yes   No   Cycles stalled on any resource
PAPI_TOT_CYC 0x8000003b  Yes   No   Total cycles
PAPI_LST_INS 0x8000003c  Yes   Yes  Load/store instructions completed
PAPI_SYC_INS 0x8000003d  Yes   Yes  Synchronization instructions completed
PAPI_L1_DCA  0x80000040  Yes   No   Level 1 data cache accesses
PAPI_L2_DCA  0x80000041  Yes   Yes  Level 2 data cache accesses
PAPI_L1_DCR  0x80000043  Yes   No   Level 1 data cache reads
PAPI_L2_DCR  0x80000044  Yes   No   Level 2 data cache reads
PAPI_L1_DCW  0x80000046  Yes   No   Level 1 data cache writes
PAPI_L2_DCW  0x80000047  Yes   No   Level 2 data cache writes
PAPI_L1_ICH  0x80000049  Yes   Yes  Level 1 instruction cache hits
PAPI_L1_ICA  0x8000004c  Yes   No   Level 1 instruction cache accesses
PAPI_L2_TCA  0x80000059  Yes   No   Level 2 total cache accesses
```

Getting started with Grace

Grace Documentation

https://docs.nvidia.com/grace

MUST READ
Grace Performance Tuning Guide

NVIDIA Grace

Search NVIDIA Grace

NVIDIA Docs Hub › NVIDIA Grace

Grace is NVIDIA's first datacenter CPU. All Grace products start with a system-on-chip (SoC) that comprise 72 high-performance Arm v9 cores and feature the NVIDIA-proprietary Scalable Coherency Fabric (SCF) network-on-chip for incredible core-to-core communication, memory bandwidth, and new GPU I/O capabilities. Grace provides a high-performance compute foundation in a low-power system-on-chip.

Built on standards such as Arm SystemReady SR, the Grace CPU is compatible with a wide variety of Arm-compatible operating systems, PCIe and USB peripherals, drivers, and application software already commonplace in existing Arm deployments— whether in the datacenter or the public cloud—including NVIDIA's CUDA and GPU driver ecosystem.

Grace is available in a variety of platforms for traditional and accelerated compute— including Grace Hopper products like GH200, which integrate a single 72-core Grace CPU with a H100 GPU on a new common memory subsystem to enable the next frontier of accelerated workloads—and the Grace CPU Superchip, which features a dual-CPU configuration with 144 cores, delivering the performance of today's highest-end conventional 2-socket CPU-based servers while improving datacenter efficiency by 2x.

Getting Started with NVIDIA Grace | Grace CPU Systems | Grace Hopper Systems

NVIDIA Grace Performance Tuning Guide

11/09/23

The Grace Performance Tuning guide provides best practices, software, and hardware configuration suggestions—

Grace OS Installation Guides

Grace systems can run a variety of Linux distributions that support the AArch64 architecture. With the proper kernel support and configurations, you can run one of the following Linux distros and take advantage

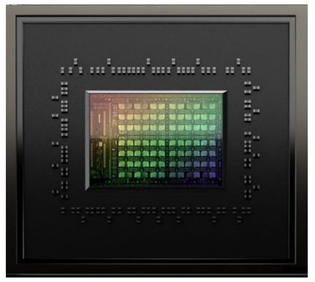
NVIDIA Grace Platform Support Software Patches and Configurations

12/18/23

Grace systems are composed of multiple hardware components that require support across different

It Just Works

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



Reuse

- NGC
- Your Linux Distros 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

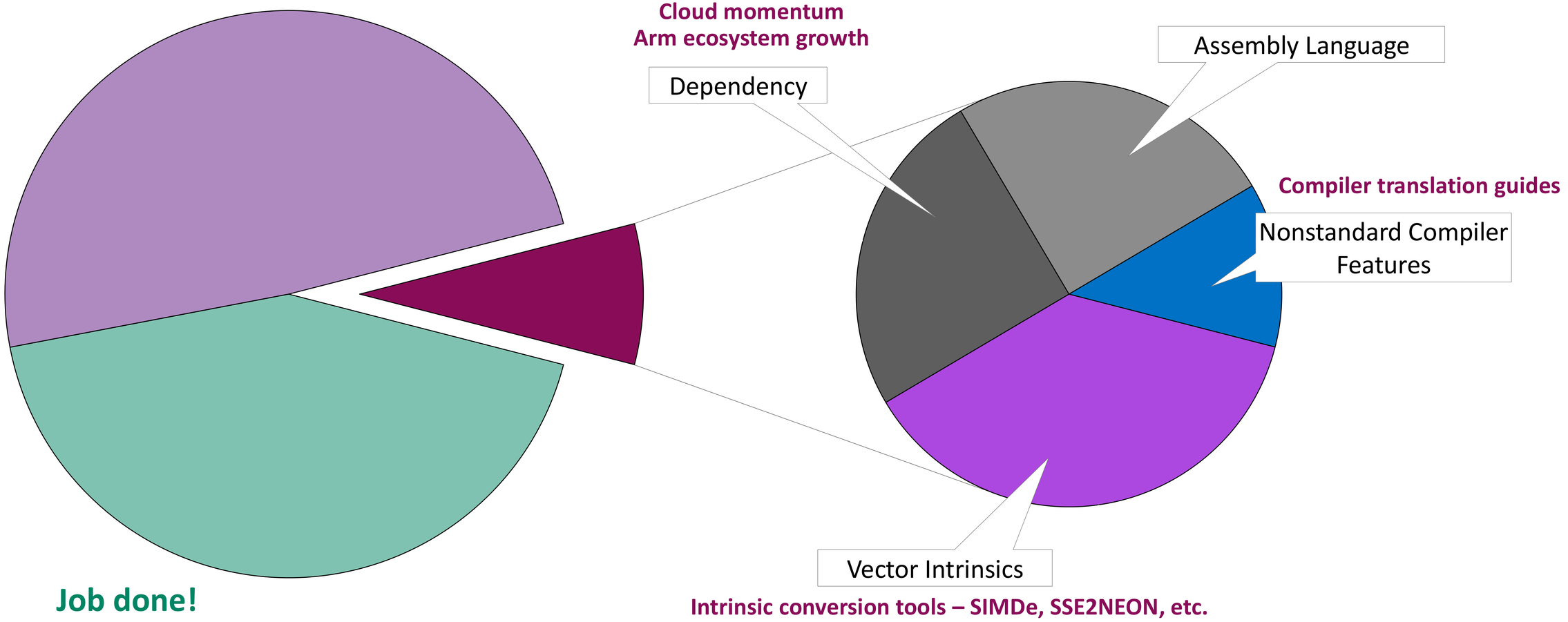
- Most compiler flags are the same:
  - Use `-mcpu=native`
  - Don’t use `-march` or `–mtune`
  - You may need `–fsigned-char`
- Update your unit tests:
  - Aarch64 floating point is as accurate as all other platforms

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

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

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/TO/nvpl/include \
    -L/PATH/TO/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.

Getting started on Grace

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 12.3.0** or later 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
  - Make sure default compiler commands (`cc`, `fc`, `gcc`, etc.) invoke a recent cross-platform compiler
  - Use `` `mpicc -show` `` or similar to verify that MPI compiler wrappers are invoking the right compiler
- Log the build, then check the log afterward

Getting started on Grace

Recommended software stack

Package	Minimum Version	Link
NVIDIA HPC SDK	23.11	https://developer.nvidia.com/hpc-sdk
NVIDIA Clang for Grace	16.0.5	https://developer.nvidia.com/grace/clang
GNU Binutils	2.41	https://ftp.gnu.org/gnu/binutils/binutils-2.41.tar.xz
GNU GCC	12.3	https://ftp.gnu.org/gnu/gcc/gcc-12.3.0/gcc-12.3.0.tar.xz
UCX	1.14.1	https://github.com/openucx/ucx/releases/tag/v1.14.1
OpenMPI	4.1.5	https://www.open-mpi.org/software/ompi/v4.1/
MVAPICH2	2.3.7	https://mvapich.cse.ohio-state.edu/download/mvapich/mv2/mvapich2-2.3.7-1.tar.gz
PAPI	7.1.0	https://icl.utk.edu/papi/
Arm Compiler for Linux	23.04	https://developer.arm.com/downloads/-/arm-compiler-for-linux

IMPORTANT NOTE: sometimes a OS base installation do not provide the most up-to-date system libraries and compilers (e.g. GNU Binutils). For optimal performance it is recommended to rebuild some of these libraries from source. See "NVIDIA Grace CPU Benchmarking Guide" at http://docs.nvidia.com/grace

Getting started with Grace

Selecting GNU and LLVM Compiler Flags for Arm

- Start with `-Ofast -mcpu=native`
  - Avoid `-march` or `-mtune` as these flags have a different meaning on Arm.
  - See https://www.stonybrook.edu/commcms/ookami/_pdf/Linford_OokamiUGM_2022.pdf for details.
- 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.
- Fortran apps building with gfortran may need `-fno-stack-arrays`
- Apps may need `-fsigned-char` or `-funsigned-char` depending on developer assumption
- 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)

Getting started on Grace

Architecture-specific flags

```
arch value : Architecture : Includes by default
armv8-a : Armv8-A : +fp, +simd
armv8.1-a : Armv8.1-A : armv8-a, +crc, +lse, +rdma
armv8.2-a : Armv8.2-A : armv8.1-a
armv8.3-a : Armv8.3-A : armv8.2-a, +pauth
armv8.4-a : Armv8.4-A : armv8.3-a, +flagm, +fp16fml, +dotprod
armv8.5-a : Armv8.5-A : armv8.4-a, +sb, +ssbs, +predres
armv8.6-a : Armv8.6-A : armv8.5-a, +bf16, +i8mm
armv8.7-a : Armv8.7-A : armv8.6-a, +ls64
armv8.8-a : Armv8.8-a : armv8.7-a, +mops
armv9-a : Armv9-A : armv8.5-a, +sve, +sve2
armv8-r : Armv8-R : armv8-r
```

- Careful with –mcpu/-march/-mtune! On Arm, **-mcpu is preferred**
  - -mcpu → sets –march and –mtune appropriately
  - -march → sets which instructions can be emitted by the compiler
  - -mtune → tells the compiler target system to optimize to
- Depending on compiler, -mcpu and –march can accept architecture extension modifiers
  - e.g., -march=armv9-a+crypto to enable crypto extension, which is not enabled by default
- -msve-vector-bits
  - Can take exact VL in bits (128, 256, etc.) or scalable keyword
  - Compiler auto vectorization can be heavily affected depending on this value
- On GCC, you can easily specify which vector extension to use
  - --param aarch64-autovec-preference={0..4} ← check man gcc for details on each value
- Non-specific x86 compiler flags should work the same, but always a good idea to check first!

Getting started on Grace

Consideration on Arm LSE Atomics

- Arm vs x86 differences in memory model
  - x86 is *strongly ordered*, with sequentially consistent (SC) behavior *almost* everywhere.
  - ARMv8, and thus AArch64, is *weakly ordered* and requires additional synchronization to behave like SC [1].
- Arm Large System Extensions (LSE)
  - Available in all server-class Arm64 CPUs (armv8.1-a mandatory extension)
  - Provides low-cost atomic operations
  - Improvement can be up to an order of magnitude in recent Arm64 CPUs

How to code with atomics?

1. Use C11 atomics [2], C++11 atomics [3], OpenMP atomic pragmas [4], or GCC __atomic intrinsics [5].
2. Do not use GCC __sync intrinsics (SC only) or any form of assembly (non-portable).

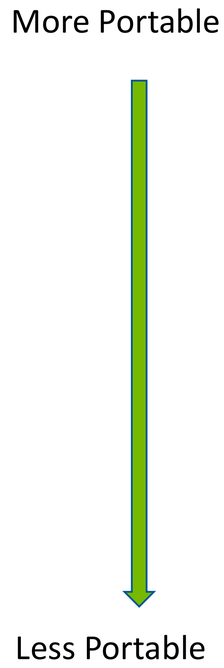
References

1. https://www.cl.cam.ac.uk/~pes20/recent_abstracts.html#armv8-mca
2. https://en.cppreference.com/w/c/atomic
3. https://en.cppreference.com/w/cpp/atomic
4. https://www.openmp.org/spec-html/5.0/openmpsu95.html
5. https://gcc.gnu.org/onlinedocs/gcc/_005f_005fatomic-Builtins.html

Arm SIMD Programming Approaches

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

- Compilers
  - Auto-vectorization: NVIDIA, GCC, LLVM, ACfL
  - Compiler directives, e.g. OpenMP
    - `#pragma omp parallel for simd`
    - `#pragma vector always`
- Libraries
  - NVIDIA Math Libraries
  - Arm Performance Library (ArmPL)
  - Open Source Scientific Libraries (BLIS, FFTW, PETSc, etc.)
- Intrinsics (ACLE)
  - Arm C Language Extensions for SVE
- Assembly
  - SVE ISA Specification: The Scalable Vector Extension for Armv8-A



- Porting and Optimizing HPC Applications for ARM: https://developer.arm.com/documentation/101725/0200
- Arm SVE Instruction Reference: https://developer.arm.com/docs/ddi0596/i/a64-sve-instructions-alphabetic-order
- SVE programming examples: https://developer.arm.com/documentation/dai0548/latest

Getting started on Grace

Porting Assembly and Vector Intrinsics

- 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
  - Tutorial using BWA-MEM2: 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]
  - 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

DYNQCD

Heavily x86 optimized Lattice QCD code

- Uses AVX2 / SSE optimization

```
4   4   /* Intel intrinsics, use MP_SINGLE (->sse) or MP_DOUBLE (->avx2) for required precision */
5       -
6       - #include <immintrin.h>
        5 + #define SIMDE_ENABLE_NATIVE_ALIASES
        6 + #include "simde/x86/avx2.h"
        7 + #include "simde/x86/fma.h"
        8 + //#include <immintrin.h>
```

SIMD_ENABLE_NATIVE_ALIASES allows to use x86 intrinsics and replace them with SIMDE versions
otherwise need to use ***simde*** prefixed versions, e.g. ***simde_mm_fnmadd_ps***

- Use appropriate compiler flags

```
29  29  ifeq ($(SIMD),4)
30      -     CFLAGS+=-mavx2 -mfma
        30 +  CFLAGS+=-mcpu=native
31  31  endif
```

Complete and Mature Toolchain Support for SVE

GCC, all LLVM-based, and many vendor toolchains are optimized for SVE autovectorization

- Arm actively posting SVE open source patches upstream since 2016
  - Beginning with first public announcement of SVE at HotChips 2016

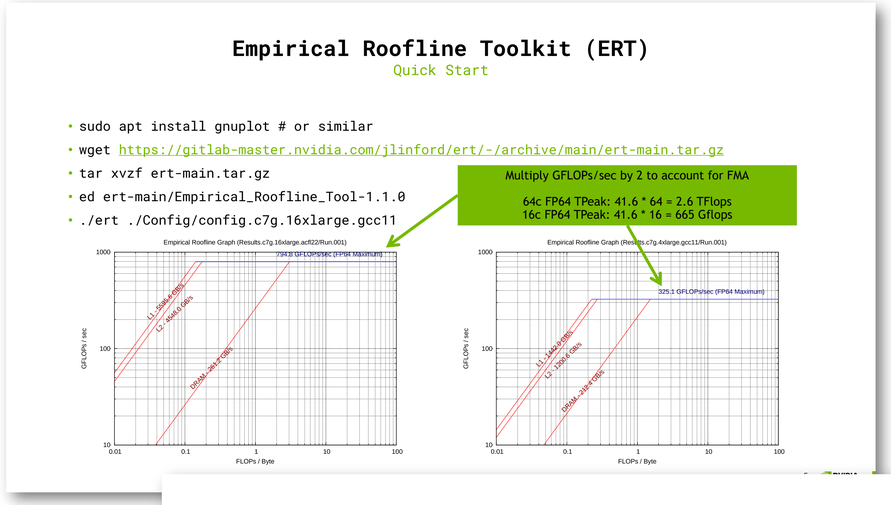
- Available upstream
  - GNU Binutils-2.28 → Released Feb 2017, includes SVE assembler & disassembler
  - GCC 8 → Full assembly, disassembly and basic auto-vectorization
  - LLVM 7 → Full assembly, disassembly
  - QEMU 3 → User space SVE emulation
  - GDB 8.2 → HPC use cases fully included
  - LLVM → Since Nov 2016, as presented at LLVM conference
  - Linux_kernel → Since Mar 2017

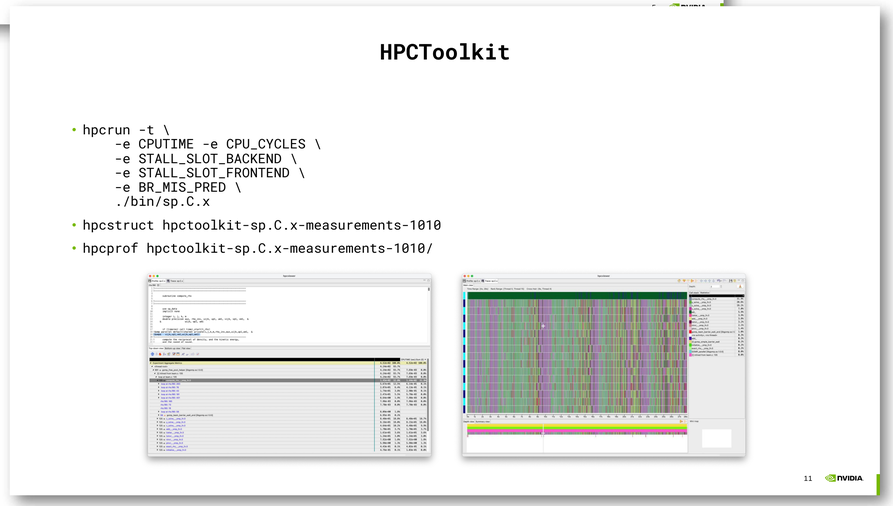
Performance Analysis

Performance Engineering Tools

A mix of NVIDIA tools and community tools – developers want to use the same tools on Grace

- **Objective**: Enable Grace developers to the same extent that they are enabled on x86
- The NVIDIA NSIGHT team are defining CPU profiling workflows that enable Grace developers
- NVIDIA is working with community tools to better support Grace
  - E.g. PAPI and libpfm4 now support Grace in-core counters





PAPI & libpfm4 with Arm Neoverse Support

Upstreamed and maintained by the community

- `cd src`
- `./configure --prefix=$PWD/..`
- `make -j && make install`

```
[jlinford@c7g-4xlarge-dy-c7g4xlarge-2 papi-jlinford-arm-neoverse]$ ./bin/papi_native_avail | grep SVE
|           Data memory read accesses (includes SVE)                          |
|           Data memory write accesses (includes SVE)                         |
|           Unaligned accesses (includes speculatively executed SVE load and s|
| SVE_INST_SPEC                                                               |
|           SVE operations sepculatively executed                             |
| SVE_PRED_SPEC                                                               |
|           SVE predicated operations speculatively executed                  |
| SVE_PRED_EMPTY_SPEC                                                         |
|           SVE predicated operations with no active predicates speculatively |
| SVE_PRED_FULL_SPEC                                                          |
|           SVE predicated operations with all active predicates speculatively|
| SVE_PRED_PARTIAL_SPEC                                                       |
|           SVE predicated operations with partially active predicates specula|
| SVE_LDFF_SPEC                                                               |
|           SVE first-fault load operations speculatively executed            |
| SVE_LDFF_FAULT_SPEC                                                         |
|           SVE first-fault load operations speculatively executed which set F|
```

```
[jlinford@c7g-4xlarge-dy-c7g4xlarge-2 papi-jlinford-arm-neoverse]$ ./bin/papi_avail | grep Yes
PAPI_L1_DCM  0x80000000  Yes   No   Level 1 data cache misses
PAPI_L1_ICM  0x80000001  Yes   No   Level 1 instruction cache misses
PAPI_L2_DCM  0x80000002  Yes   No   Level 2 data cache misses
PAPI_TLB_DM  0x80000014  Yes   No   Data translation lookaside buffer misses
PAPI_L2_LDM  0x80000019  Yes   No   Level 2 load misses
PAPI_STL_ICY 0x80000025  Yes   Yes  Cycles with no instruction issue
PAPI_HW_INT  0x80000029  Yes   Yes  Hardware interrupts
PAPI_BR_MSP  0x8000002e  Yes   No   Conditional branch instructions mispredicted
PAPI_BR_PRC  0x8000002f  Yes   Yes  Conditional branch instructions correctly predicted
PAPI_TOT_INS 0x80000032  Yes   No   Instructions completed
PAPI_FP_INS  0x80000034  Yes   No   Floating point instructions
PAPI_LD_INS  0x80000035  Yes   No   Load instructions
PAPI_SR_INS  0x80000036  Yes   No   Store instructions
PAPI_BR_INS  0x80000037  Yes   No   Branch instructions
PAPI_VEC_INS 0x80000038  Yes   Yes  Vector/SIMD instructions (could include integer)
PAPI_RES_STL 0x80000039  Yes   No   Cycles stalled on any resource
PAPI_TOT_CYC 0x8000003b  Yes   No   Total cycles
PAPI_LST_INS 0x8000003c  Yes   Yes  Load/store instructions completed
PAPI_SYC_INS 0x8000003d  Yes   Yes  Synchronization instructions completed
PAPI_L1_DCA  0x80000040  Yes   No   Level 1 data cache accesses
PAPI_L2_DCA  0x80000041  Yes   Yes  Level 2 data cache accesses
PAPI_L1_DCR  0x80000043  Yes   No   Level 1 data cache reads
PAPI_L2_DCR  0x80000044  Yes   No   Level 2 data cache reads
PAPI_L1_DCW  0x80000046  Yes   No   Level 1 data cache writes
PAPI_L2_DCW  0x80000047  Yes   No   Level 2 data cache writes
PAPI_L1_ICH  0x80000049  Yes   Yes  Level 1 instruction cache hits
PAPI_L1_ICA  0x8000004c  Yes   No   Level 1 instruction cache accesses
PAPI_L2_TCA  0x80000059  Yes   No   Level 2 total cache accesses
```

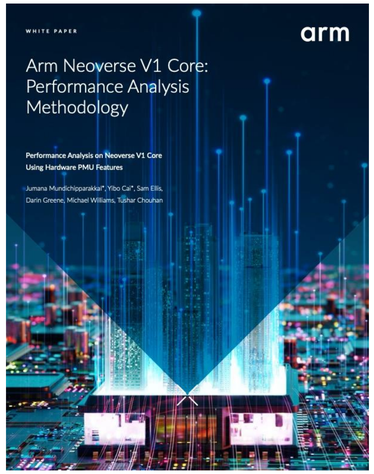
Top-Down Methodology

Workload characterization using PMU

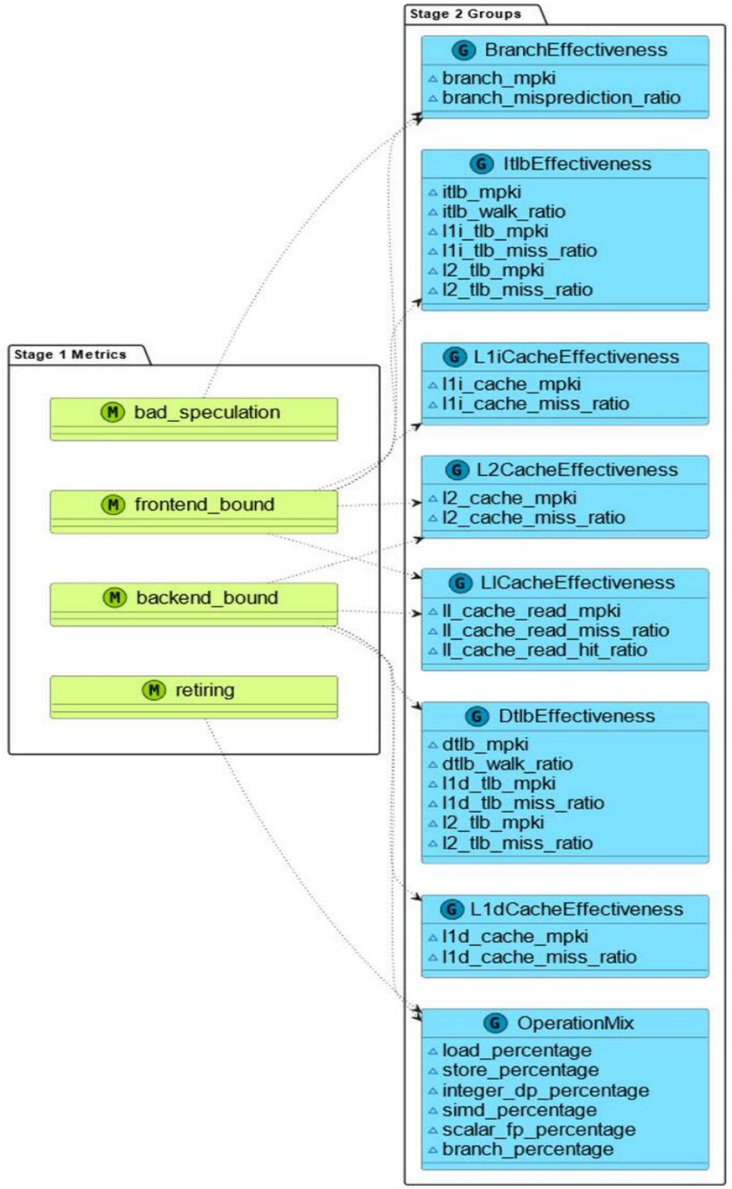
- Consider in core performance limiters
  - Frontend-bound
  - Backend-bound

- Arm Neoverse V1 Performance Analysis whitepaper
  https://community.arm.com/arm-community-blogs/b/infrastructure-solutions-blog/posts/arm-neoverse-v1-top-down-methodology

  - Most of the content applies for Grace (Neoverse V2)
  - Stage 1 Metrics tell you about performance boundness
    - Bad speculation, frontend/backend bound or retiring
  - Stage 2 helps you identify why code is bound by X
    - L1 instruction/data cache, L2 cache, DTLB, etc.



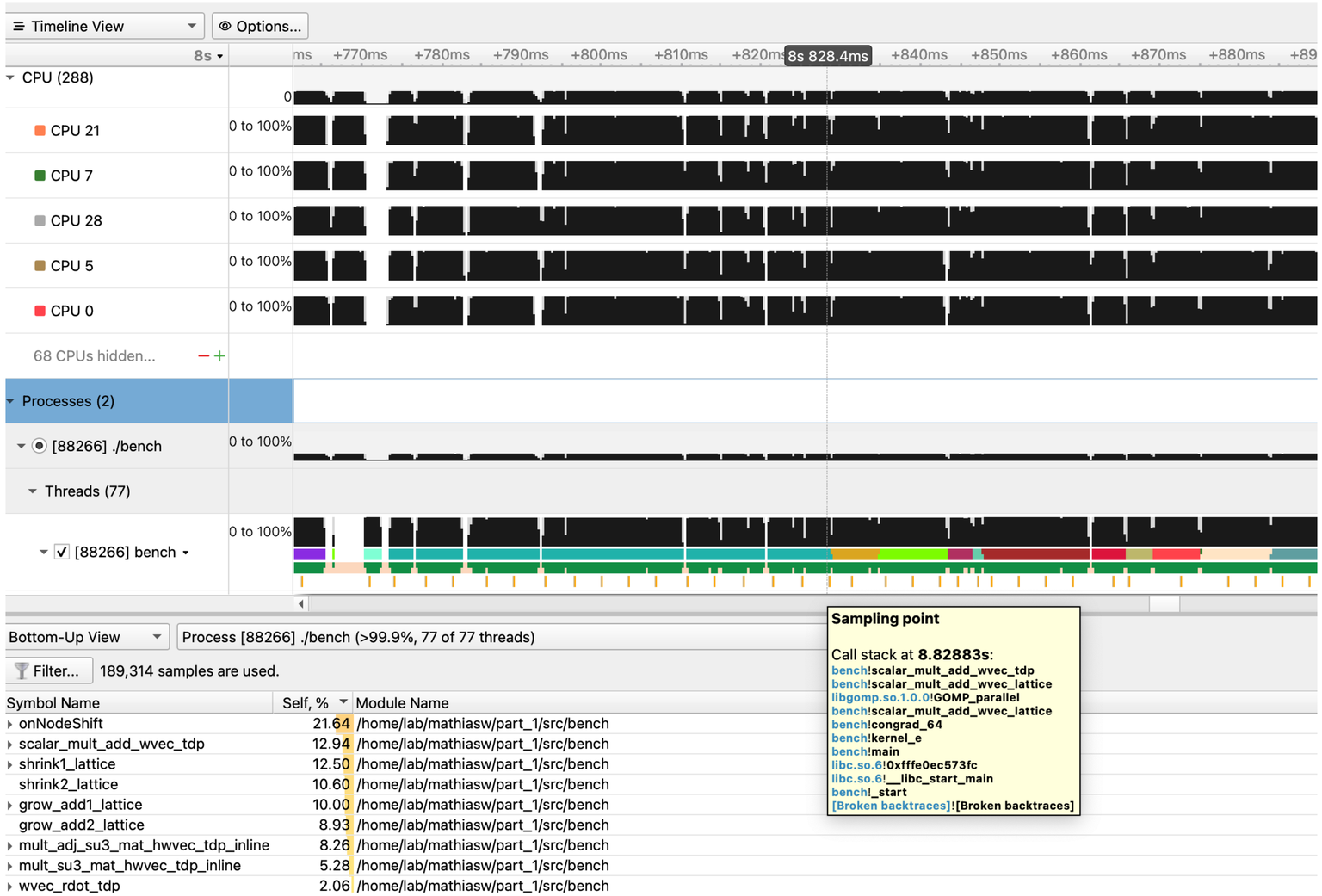
- Arm Neoverse V2 PMU Documentation
  https://developer.arm.com/documentation/PJDOC-1063724031-660094/1/?lang=en
- Arm Neoverse V2 Software Optimization Guide
  https://developer.arm.com/documentation/pjdoc466751330-593177/latest



Nsight Systems

Timeline

- Nsight Systems Documentation
- https://docs.nvidia.com/nsight-systems/UserGuide/index.html
  - → **CPU Profiling on Linux**
  - → **Profiling from the CLI**
- Collect profile for timeline offline
  - `nsys profile ./app`
  - https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profiling
- Open generated `*.nsys-rep` in Nsight Systems



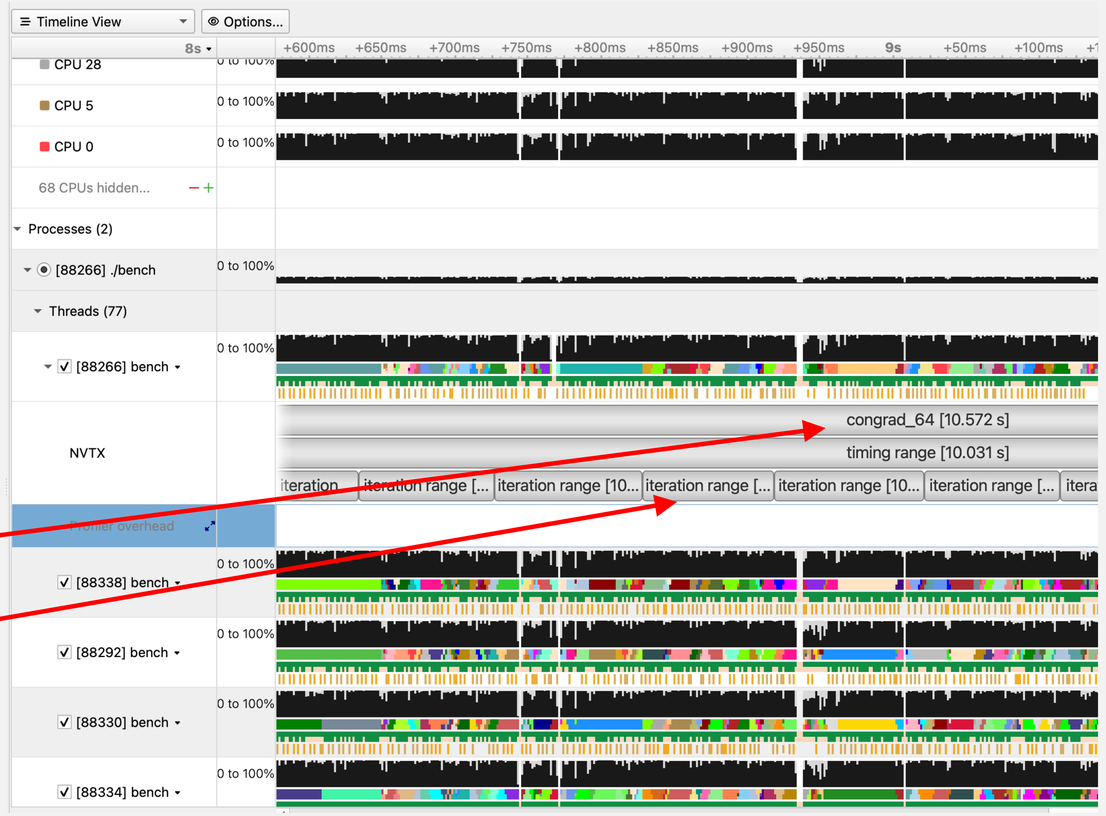
NVTX

Markup your code

- https://github.com/NVIDIA/NVTX/
  - Also part of CUDA Toolkit
  - Available for C, C++, Python, Fortran (NVHPC TK)

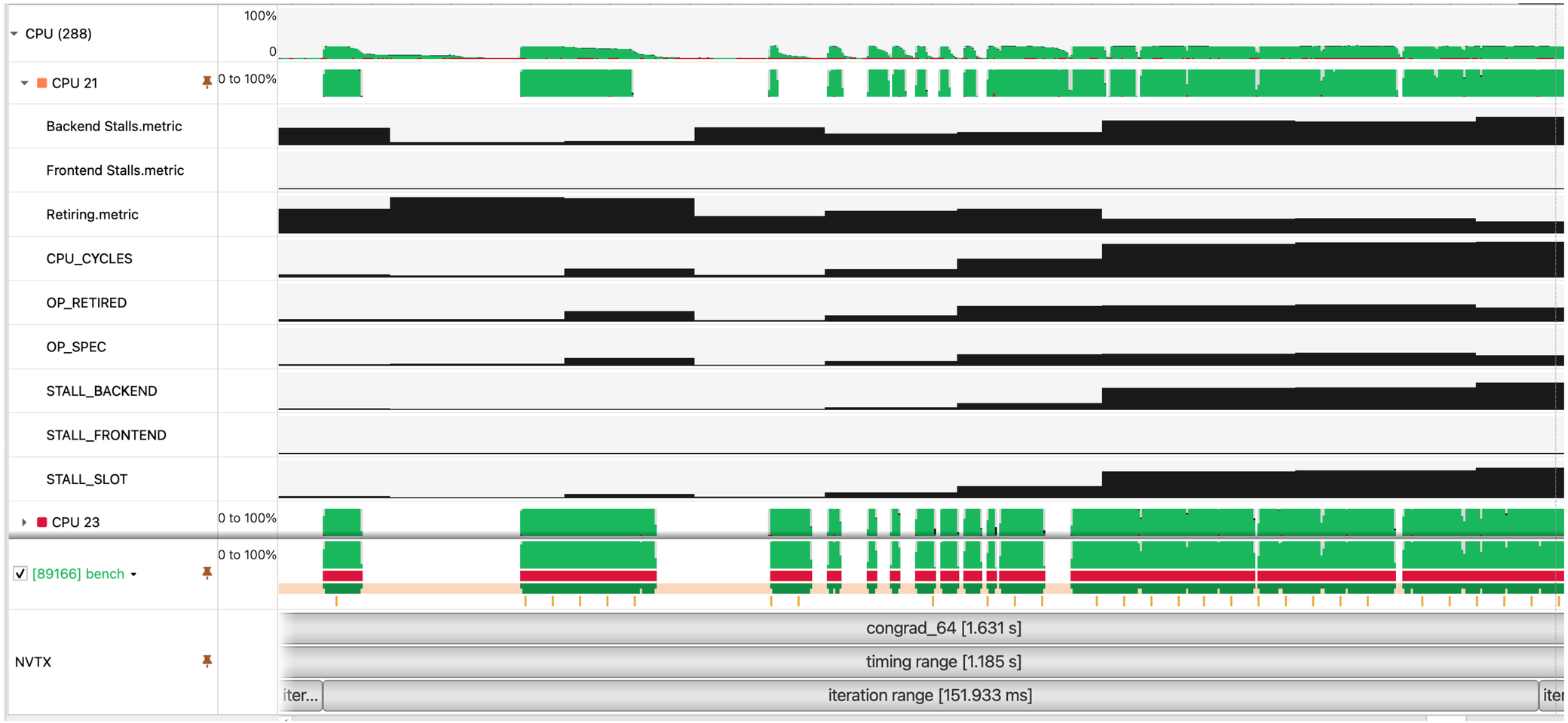
```
#include <nvtx3/nvToolsExt.h>

void congrad_64()
{
  nvtxRangePush(__func__); // Range around the whole function
  for (int i = 0; i < 6; ++i) {
    nvtxRangePush("loop range"); // Range for iteration
    // Do ab iteration
    nvtxRangePop(); // End the inner range
  }
nvtxRangePop(); // End the outer range
}
```



Nsight Systems – Core metrics

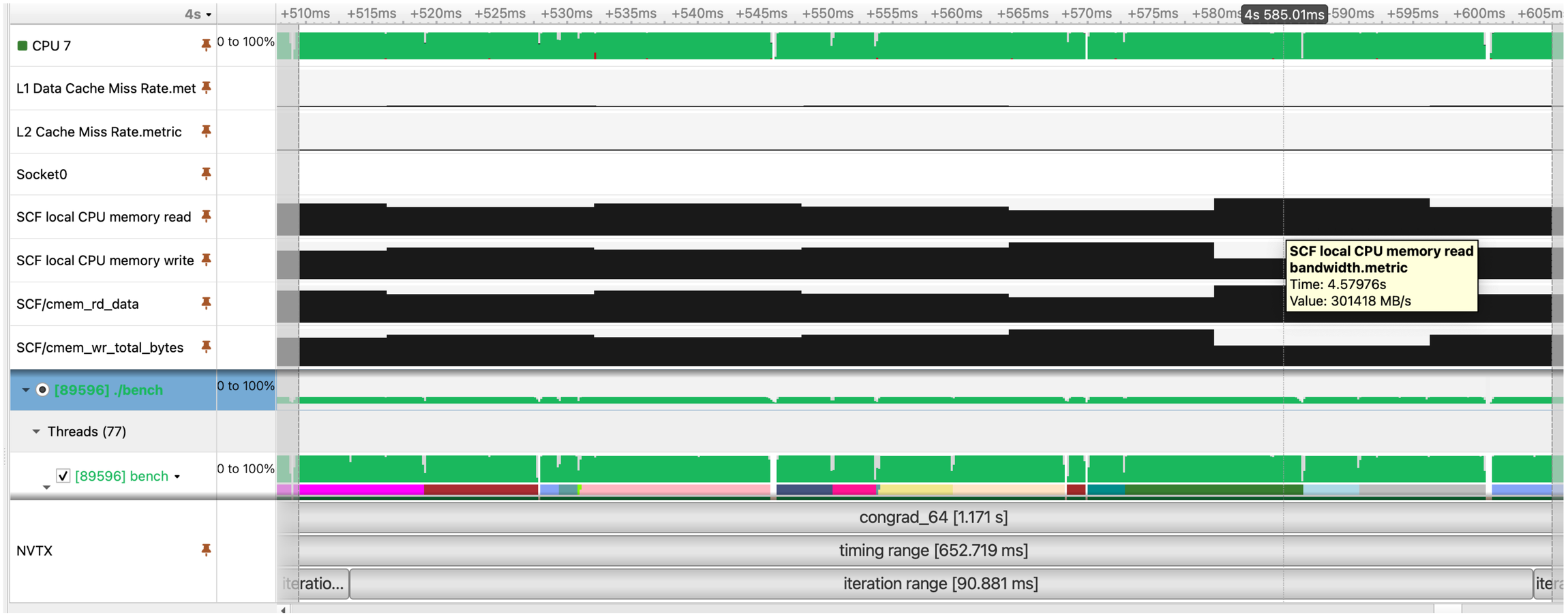
Collect Core performance metrics




```
nsys profile --cpu-core-metrics=help
```

Nsight Systems – Uncore metrics

Collect uncore performance metrics

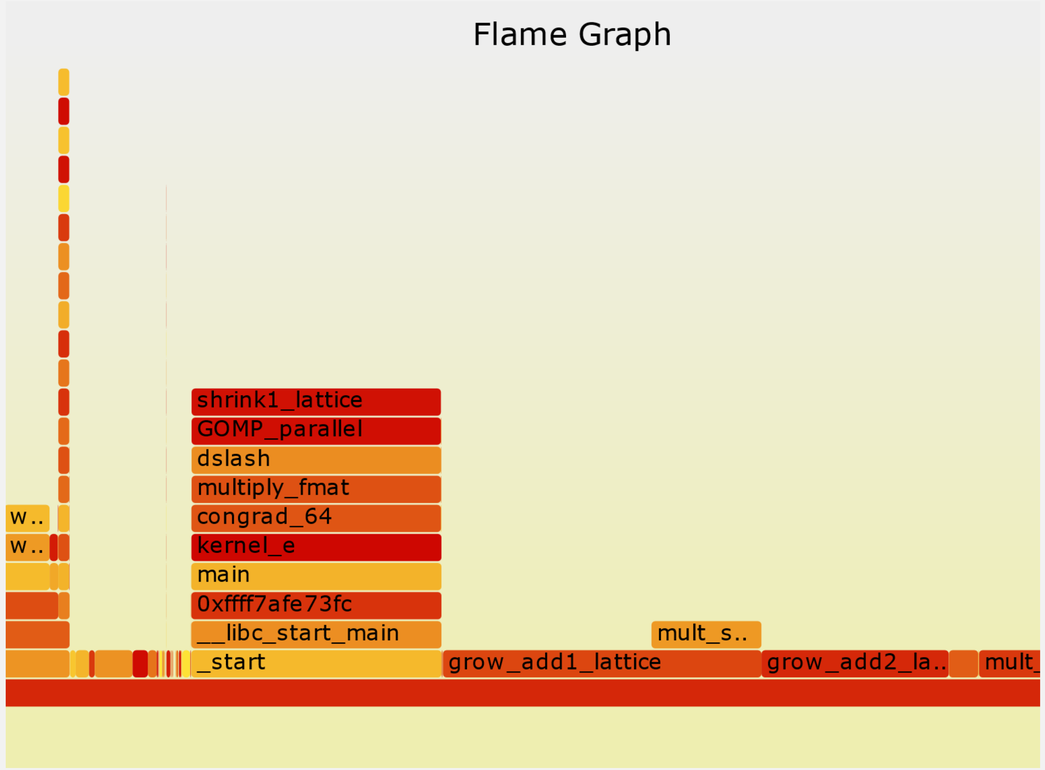



```
nsys profile --cpu-socket-metrics=help
```

Identify Hotspots

https://github.com/brendangregg/FlameGraph

- Using Nsight Systems
  - `Scripts/Flamegraph` directory in NSight Systems installation directory
  - `nsys profile –o report ./app`
  - `python3 stackcollapse_nsys.py report.nsys-rep | ./flamegraph.pl > result_flamegraph.svg`
- Using perf
  - `perf record -a -g ./app`
  - `perf script | stackcollapse-perf.pl > out.perf-folded`
  - `flamegraph.pl out.perf-folded > perf.svg`



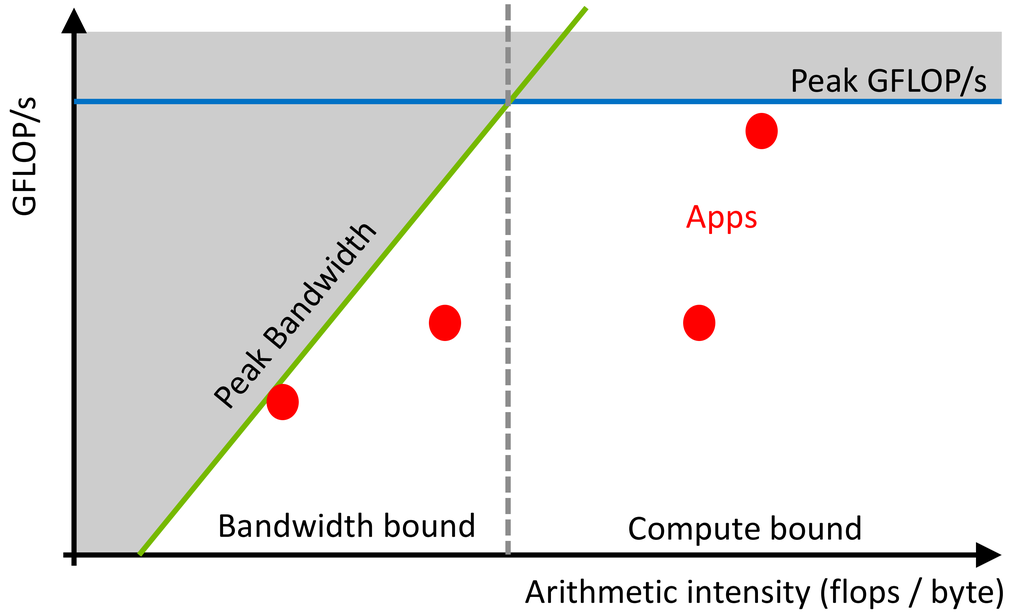
Roofline

- Consider compute and data throughput
- Hierarchical model considers also
  - cache bandwidths
  - Different peak GFLOP/Ss (precision, vectorization, FMA)
- Arithmetic intensity AI = (Flops/Byte)

- Define the roofline
  - either theoretical peaks for bandwidth and flops
  - Or measured values
    - (pro-tip: use same operation types and load/store ratios as your benchmark)
    - Use stream, likwid, NVIDIA Arm-Kernels,

- Know your workload:
  - Calculate expected AI from your code
  - Measure using tools

- Distance to the roofline indicates achieved performance level

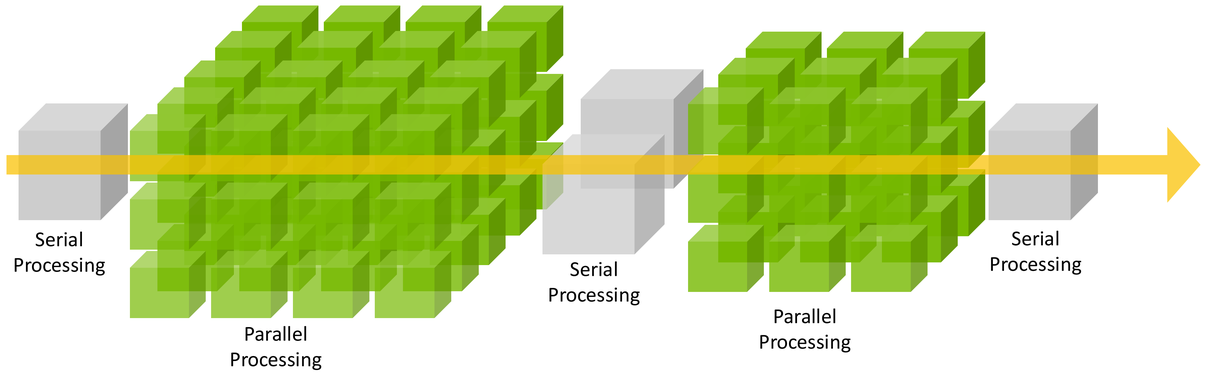


GH200 “Grace Hopper” Superchip

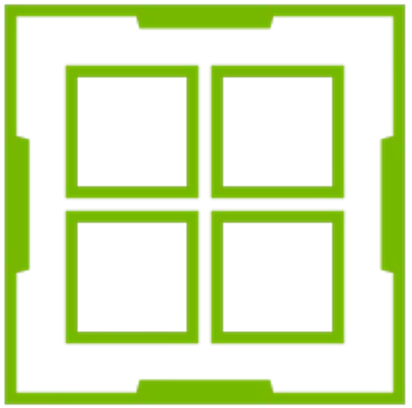
Modern Applications Integrate Sequential and Parallel Computing

Both CPU and GPU are Critical for Accelerated Compute

Modern Application Workflows



CPU



Excels at Serial Processing

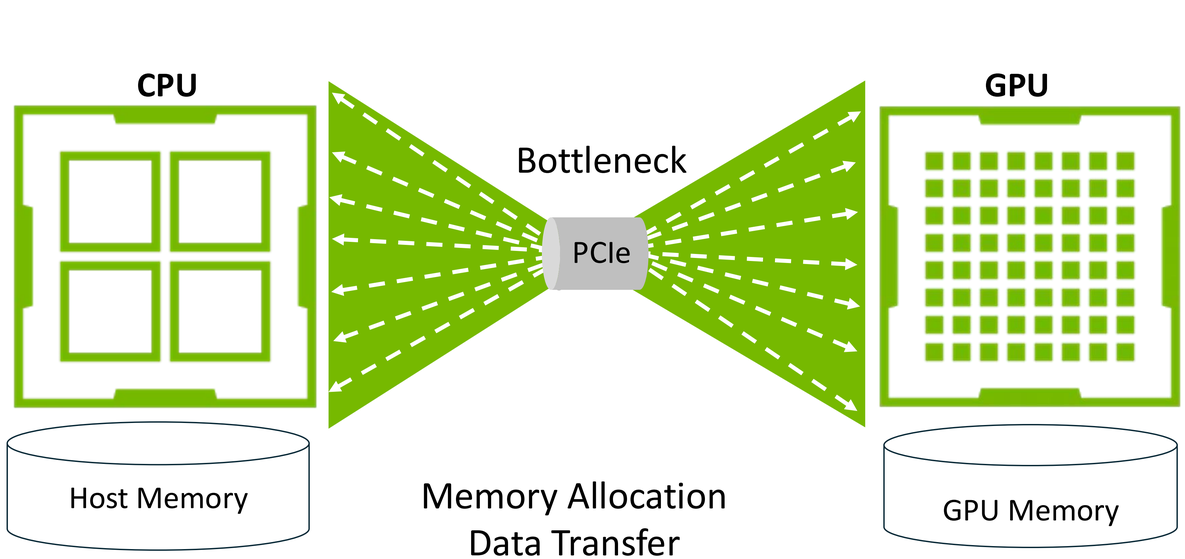
GPU

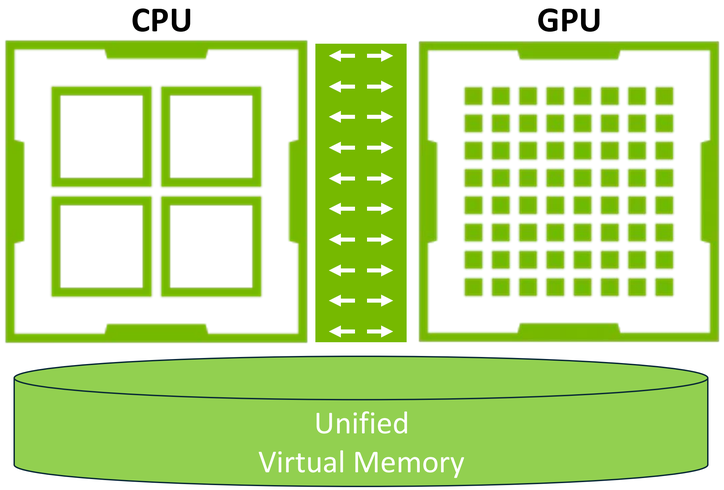
Excels at Parallel Processing

To further accelerate compute, and make it accessible to every workload we need to address CPU bottlenecks

Challenges with Traditional Accelerated Systems

Bridging the GAP: Unifying CPU and GPU Memory





- ❑ PCIe bottlenecks CPU-GPU communication
- ❑ GPU can't access CPU memory directly
- ❑ Complex Programmability
- ❑ Low CPU Performance per Watt

Legacy PCIe Architecture

We need a new chip architecture:

- ✓ Accelerate CPU-GPU data transfers
- ✓ Coherent CPU and GPU cache memory
- ✓ Simplify Programmability
- ✓ Deliver more performance per Watt

Two Memory Systems, Each Optimized For Its Processor

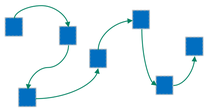
Act as a Single Memory Pool for Data Intensive Applications

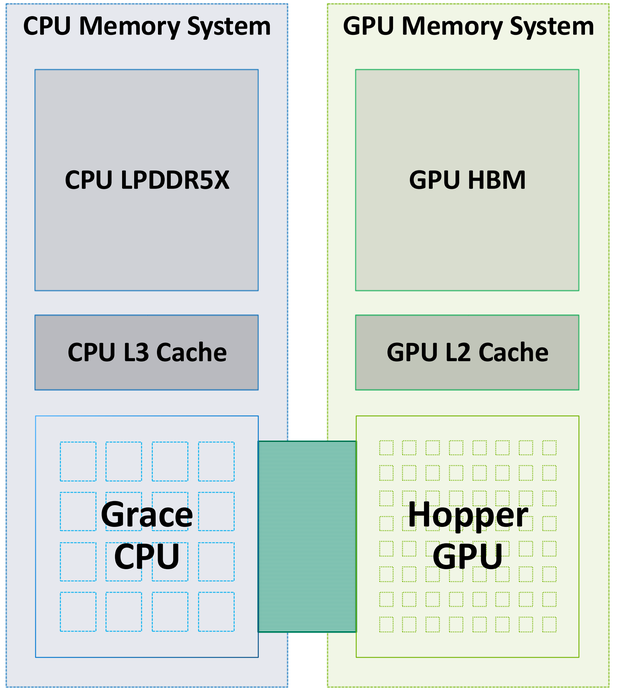
Benefits

- Faster data transfer
- Memory coherency with simplified programming
- Power efficient performance

CPU memory system is optimized for **low latency** and **deep cache hierarchy**

Run **latency-sensitive** code on the CPU, e.g., a linked list



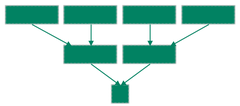


NVLink-C2C

CPU and GPU each have **full coherent access** to memory

GPU memory system is optimized for **high throughput** and **high bandwidth cache**

Data- and math-intensive code on the GPU, e.g., vector reduction



GH200 GRACE HOPPER SUPERCHIP

The breakthrough accelerated CPU for Large-Scale AI and HPC applications

Grace CPU + H100 GPU

72 Arm Neoverse V2 Cores with SVE2 4x128b
Transformer Engine and ~4PFLOPS of FP8

Fast NVLink-C2C Connection

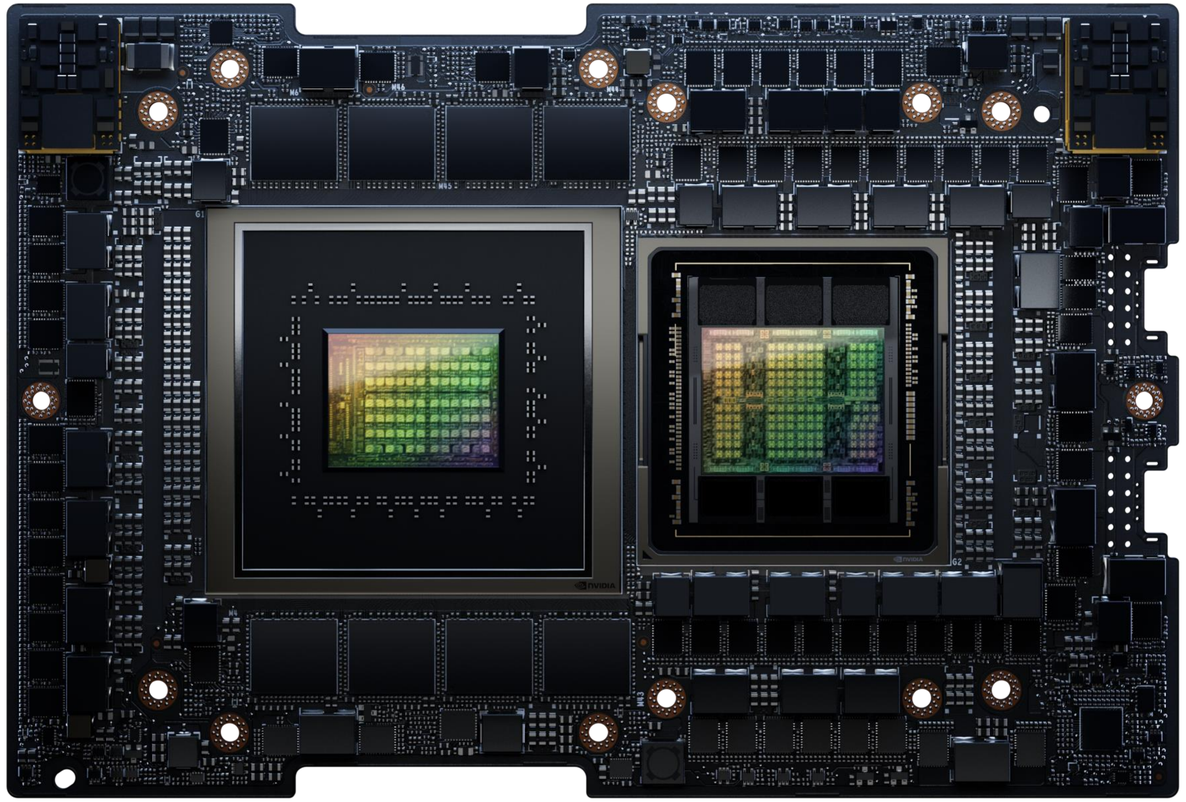
900GB/s bi-directional bandwidth CPU to GPU
7X faster than PCIe Gen 5

~600GB of Fast Access Memory

Up to 96GB HBM3, 4TB/s bandwidth
Up to 480GB LPDDR5X, 512GB/s bandwidth

Full NVIDIA Compute Stack

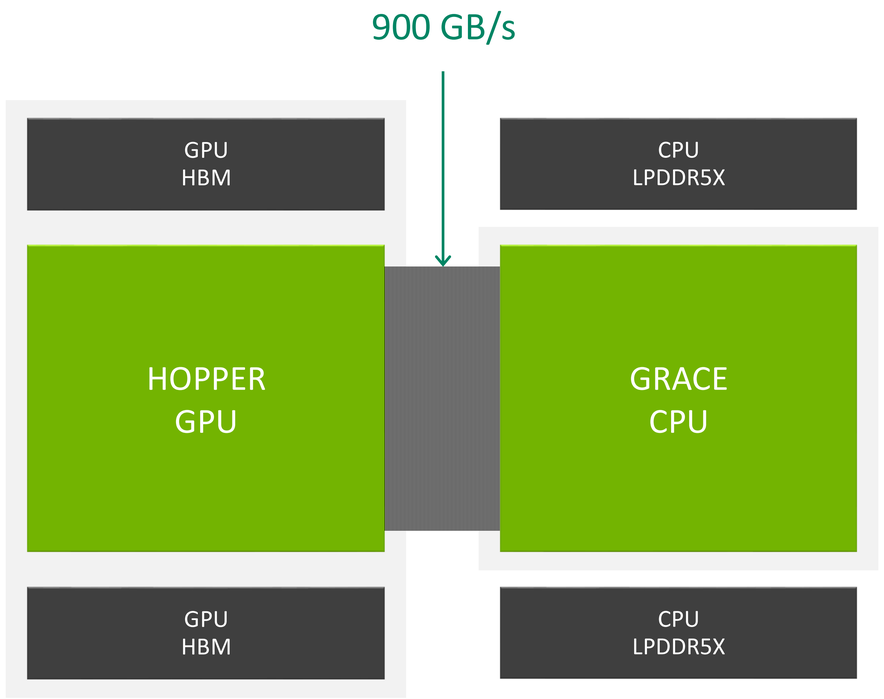
AI, Omniverse



NVLINK-C2C

High Speed Chip to Chip Interconnect

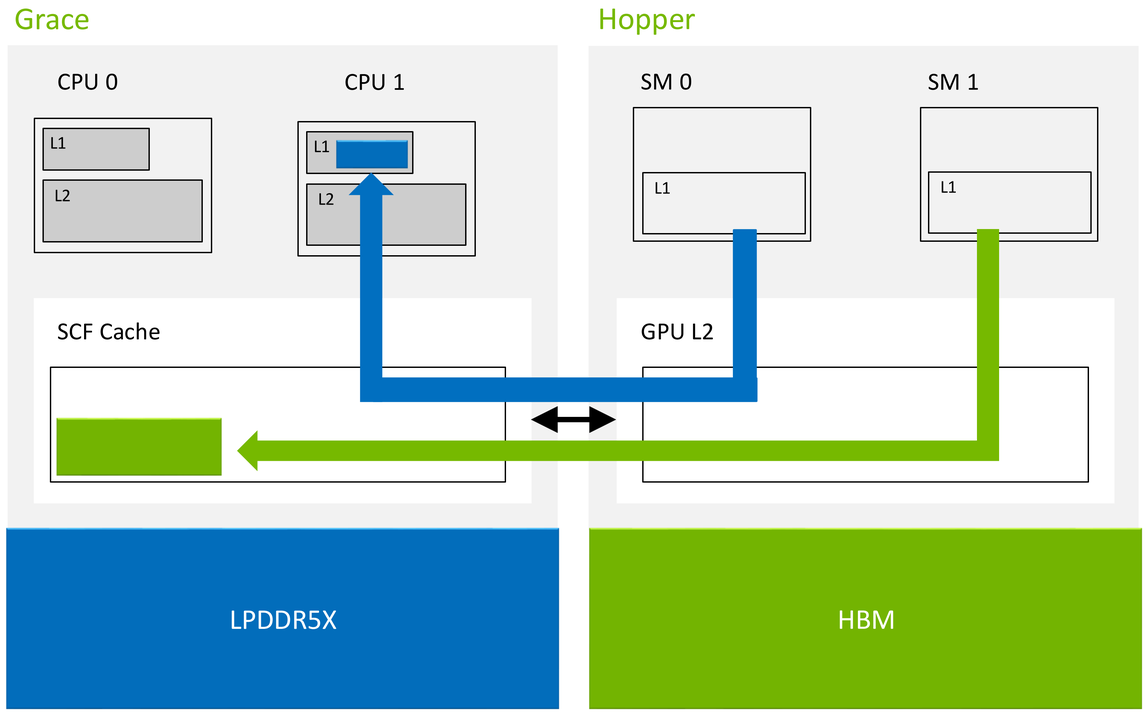
- Used to create the Grace Hopper, and Grace Superchips
  - Native atomics, including standard C++ atomic support
  - **Enables coherency**
- Up to 900 GB/s of raw bidirectional BW
  - Same BW as GPU to GPU NVLINK on Hopper
- Low power interface - 1.3 pJ/bit
  - **More than 5x more power efficient than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



Grace Hopper Coherency

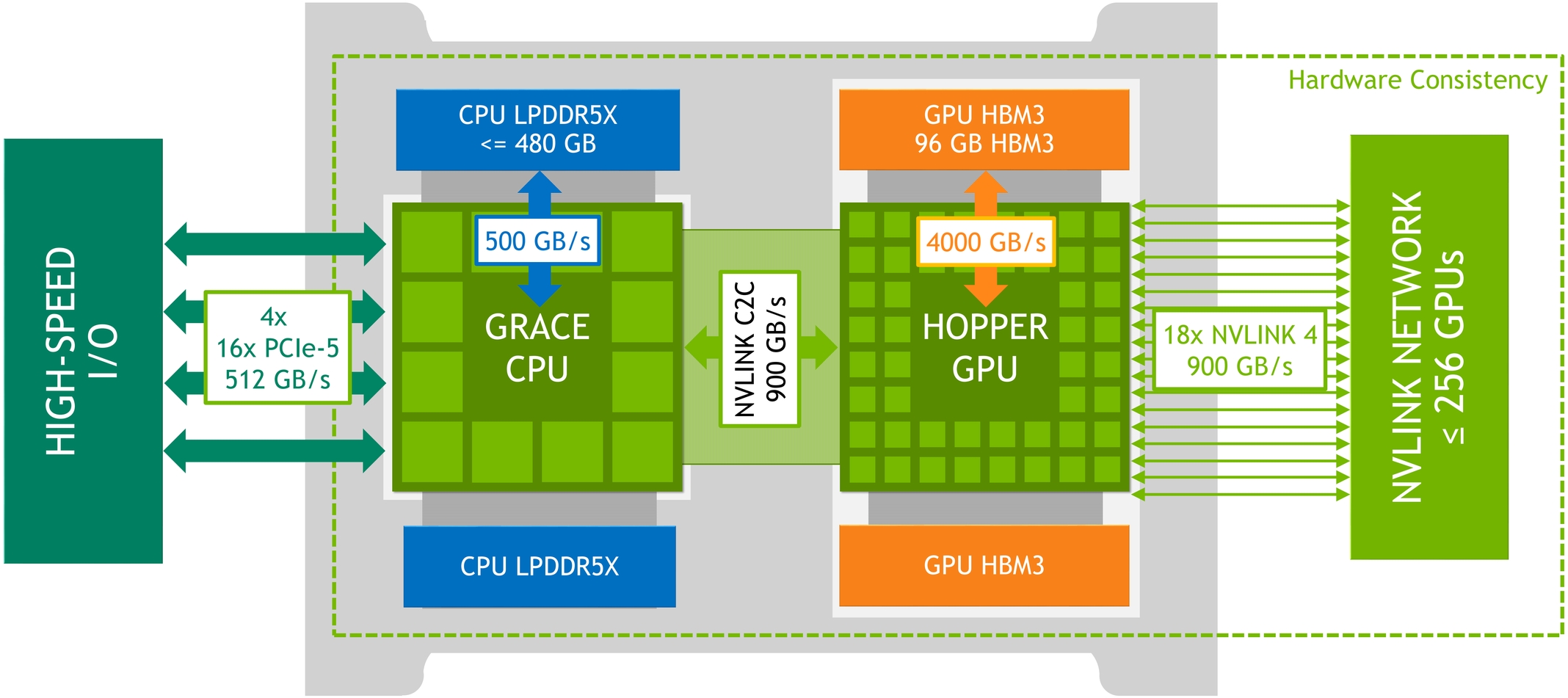
Heterogeneous Coherency

- **Grace follows the Arm memory model**
  - Coherently caches Hopper's memory
  - Data can be cached in CPU caches or SCF Cache
  - Same programming model as any other CPU memory
- **Hopper follows the CUDA memory model**
  - GPU L2 tracks if Grace has a line and will snoop Grace for data
  - GPU will snoop CPU caches for LPDDR5X memory access



Grace Hopper Superchip

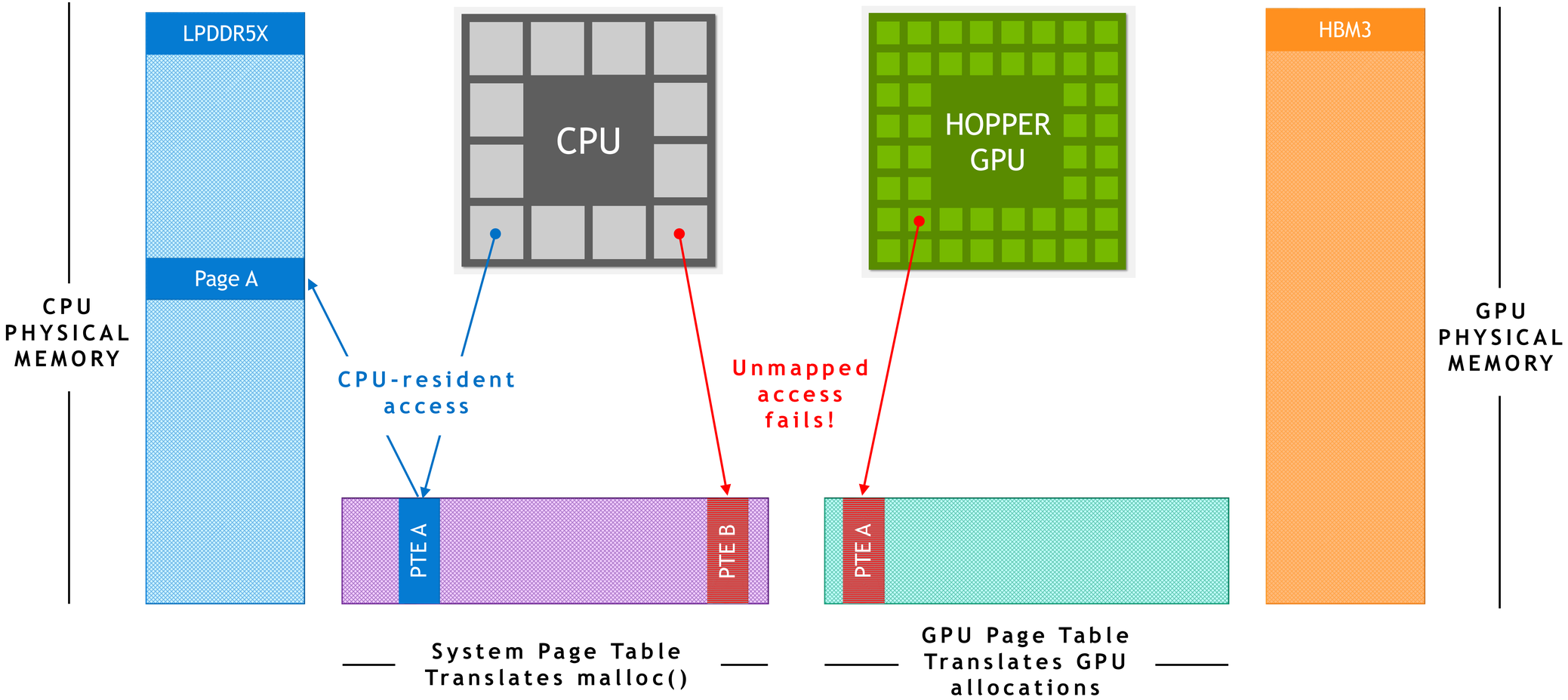
GPU can access CPU memory at CPU memory speeds



https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

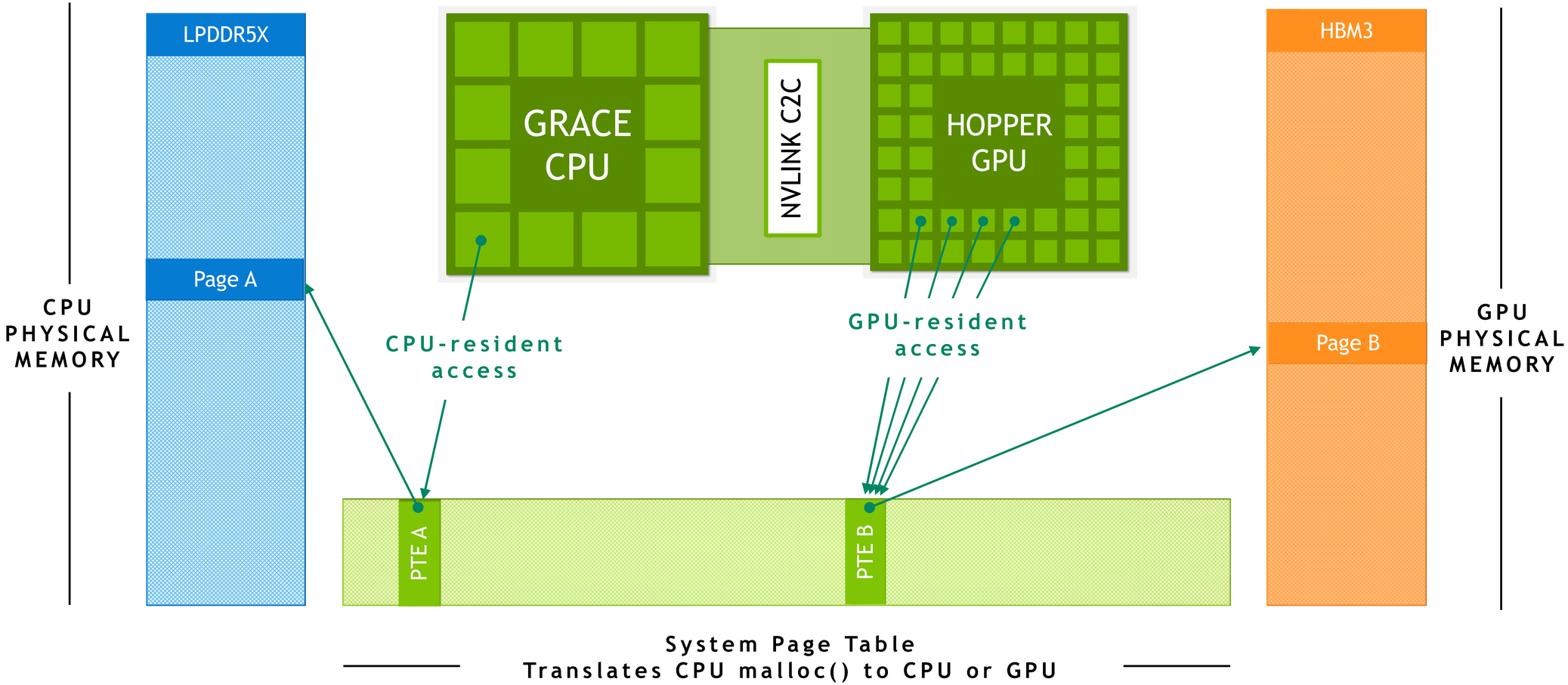
HW/SW memory view on X86 + GPU

Separate page tables



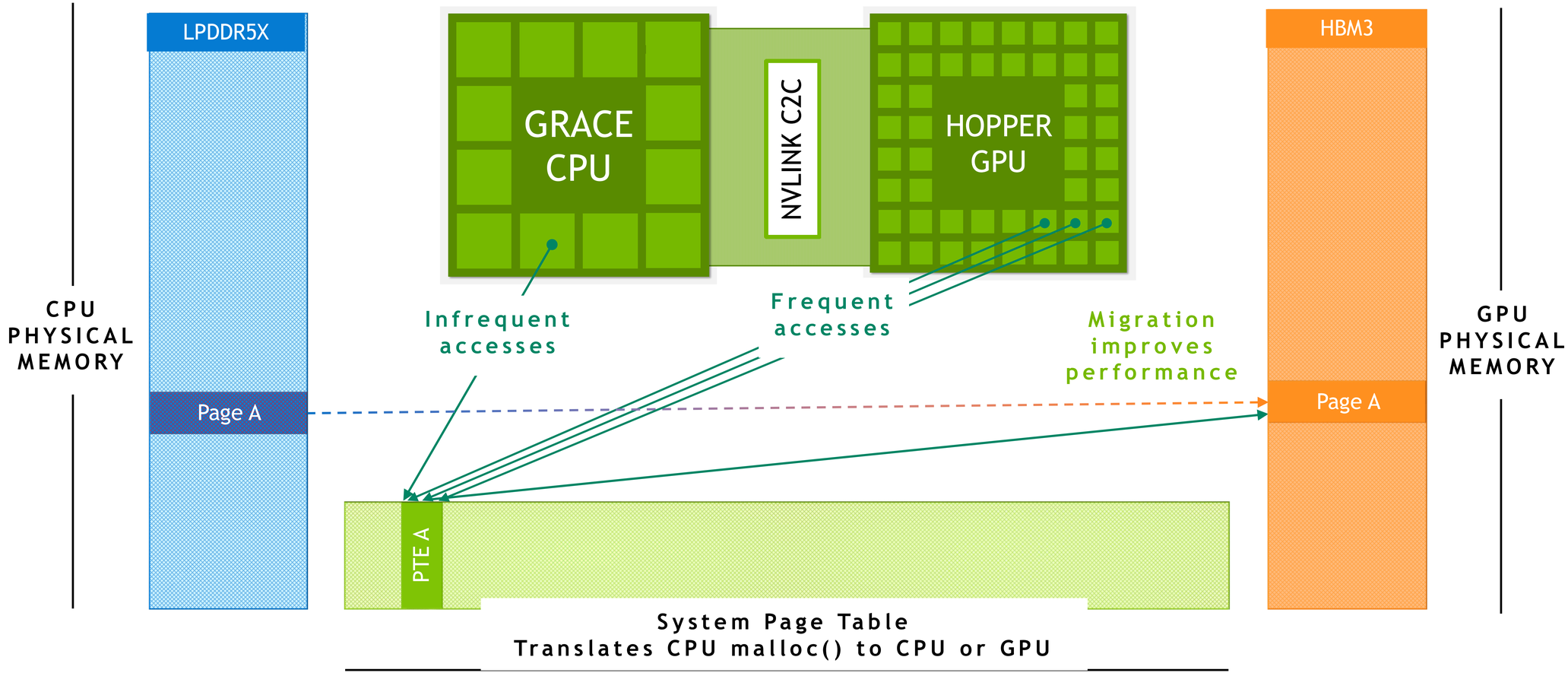
HW/SW memory view on GH200

Local memory pool accesses at link speeds



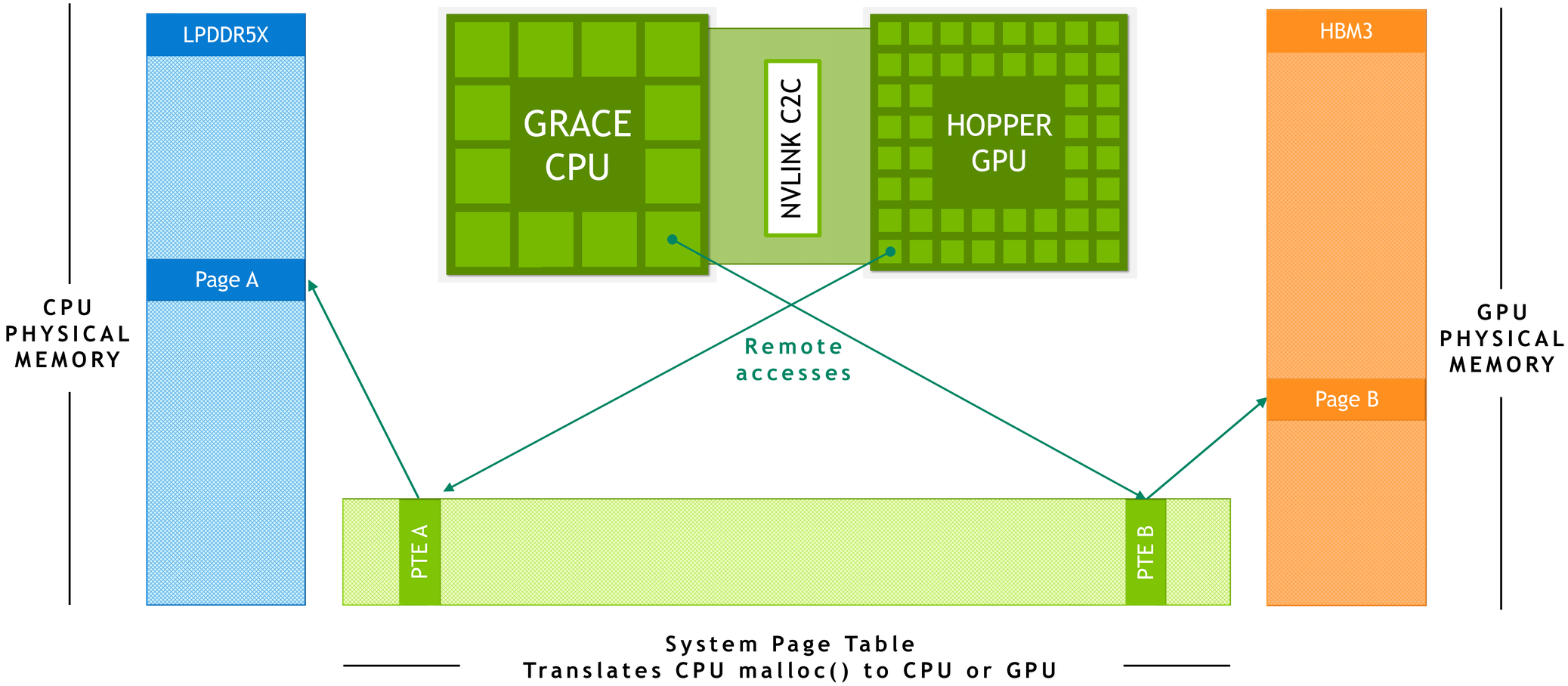
HW/SW memory view on GH200

Simplified Unified Memory via **Address Translation Service (ATS)** & automatic micrations



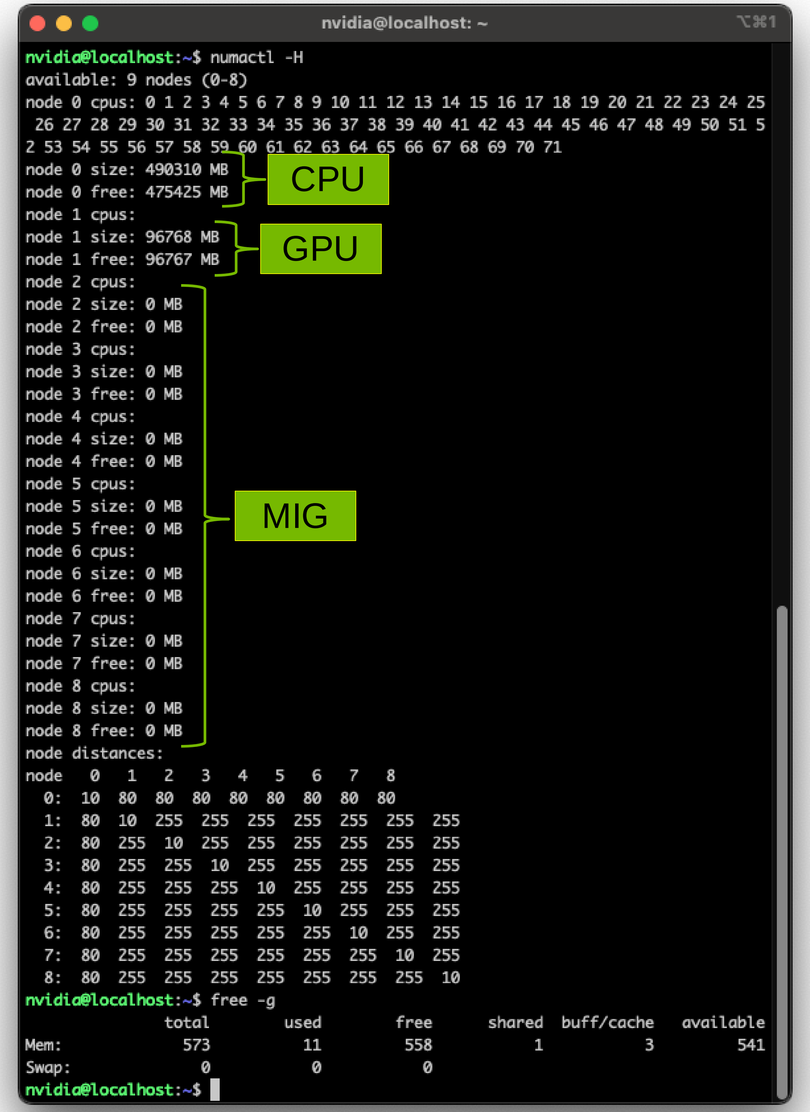
HW/SW memory view on GH200

C2C enabling remote accesses



GPU Memory is Visible to the Operating System

Standard operating system commands work on the GPU




```
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
```

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

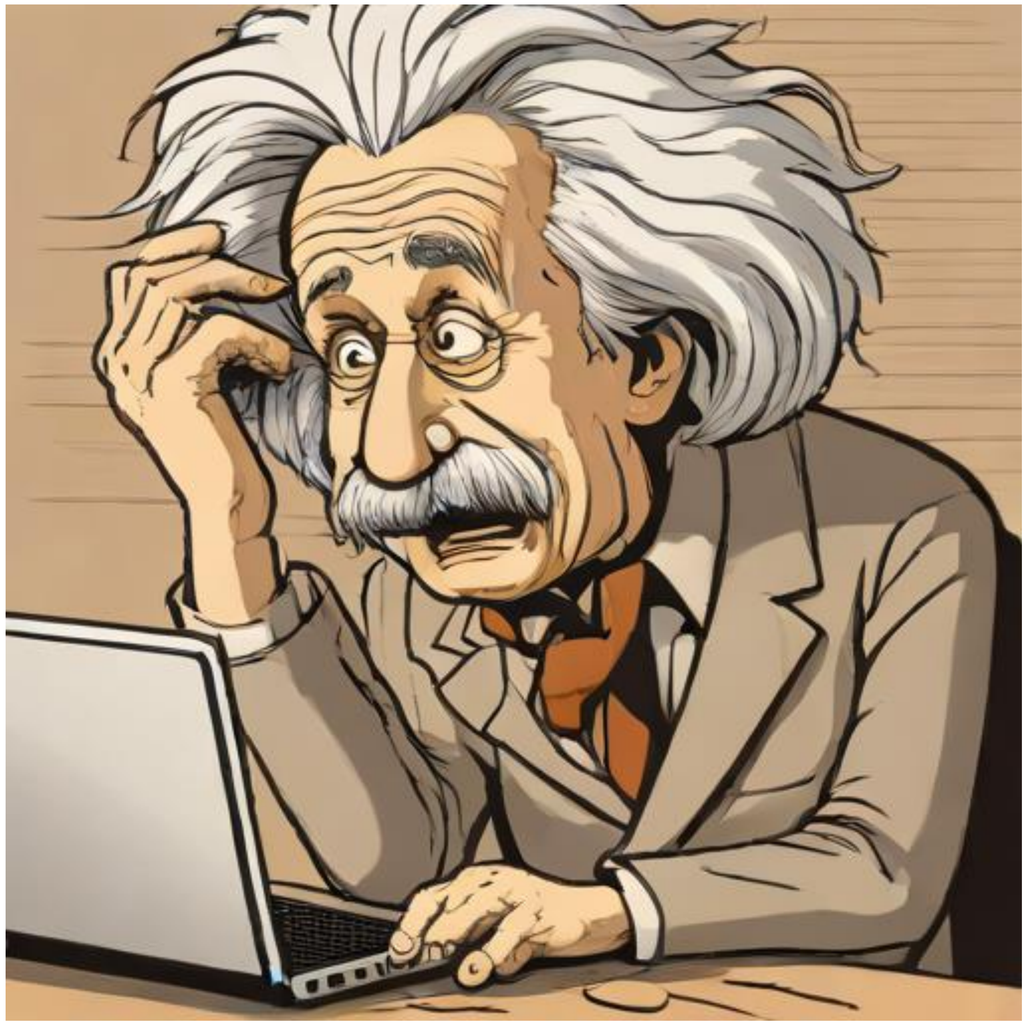
What do Programmers Need to Know About Grace-Hopper

- **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**
  - Managing and processing larger data sets
  - Workflows that utilize both halves

Dispelling Some Myths

- GPU Programming...
  - …is hard
  - …is time-consuming
  - …isn't portable
  - …requires a special language or extensions
  - …can only be done in C++
  - …is only for ninja programmers

FALSE



Choosing A Programming Model

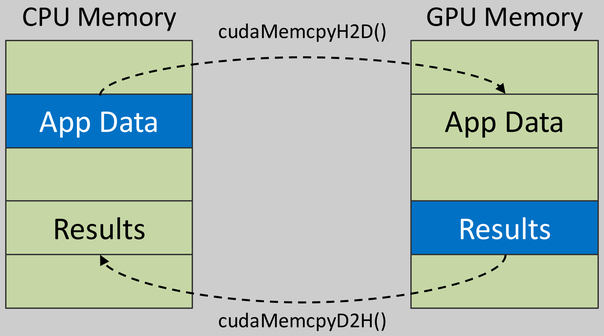
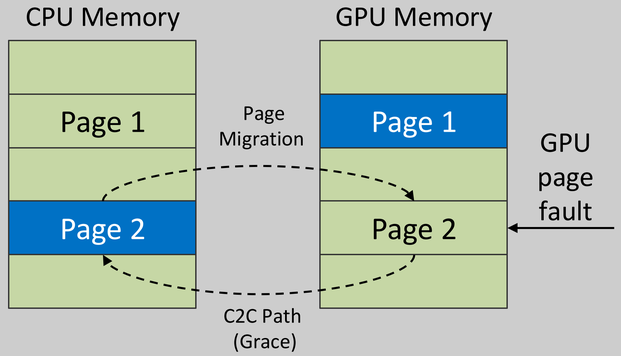
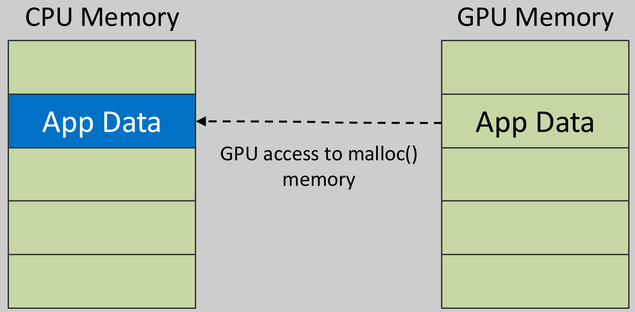
There can be ~~only~~ ***more than*** one.

Libraries	Standard Languages	Compiler Directives	CUDA Languages
• Accelerate common operations with little/no code changes. • Expert-tuned performance. • Forward support guarantees.	• Strong cross-platform support. • Single source code for multiple platforms. • Reduced learning curve.	• High cross-platform support. • Single source code for multiple platforms. • Reduced learning curve. • Additional programmer control.	• Exposes full GPU capabilities. • Trades portability for performance. • Distinct GPU/CPU code paths. • Full programmer control.
Programmer Productivity			Programmer Control

By design these approaches are interoperable so developers can choose the right balance for their needs.

The Grace Hopper Advantage

Full CUDA support with additional Grace memory extensions

	Explicit Copy	Managed Memory	System Allocated
	Application explicitly moves data between CPU & GPU as needed	CPU and GPU can access memory on-demand and data migrated locally for higher BW access	GPU can access memory allocated from `malloc()`, `mmap()`, etc.
			
HGX	~60 GB/s PCIe Gen5 transfers (H2D/D2H)	Requires migration to GPU	Access possible with explicit call to `cudaHostRegister()` at PCIe speeds Requires HMM patch in Linux Kernel
G+H	7x faster transfers, up to 450 GB/s (NVLink C2C)	Migrations not required and faster migrations when they happen at NVLink C2C speed	`cudaHostRegister()` not needed; access at NVLink C2C speeds

CUDA Explicit Memory Allocators

Maximum **portable** performance Out-of-the-box

- **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.
  - Work with all memory, e.g., including `malloc`.

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`

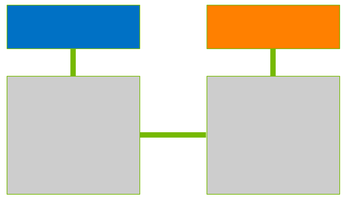
```
cudaMemPrefetchASync(ptr, nbytes, destination);
```

Destinations	`GPU id`	`CPU`

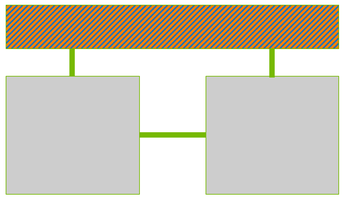
NVHPC Compilers GPU Memory Model

Abstraction over HW simplifying GPU programming

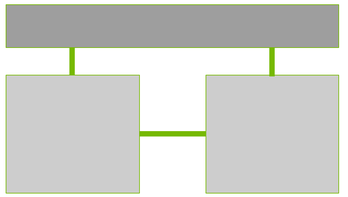
- **Separate** - CPU and GPU have distinct memories when data are shared between the two explicit copy is needed



- **Managed –** CPU and GPU have a single address space for dynamically-allocated data, data is migrated automatically on-demand. Other memory remains separate.



- **Unified** - CPU and GPU have single address space which allows accessing all data locations from both processors, data may be migrated or accessed in-place.



NVHPC Compilers GPU Memory Model Flags

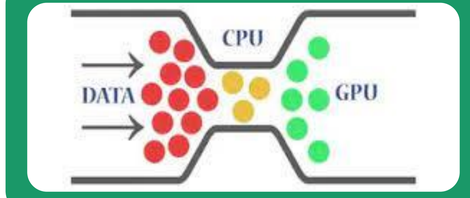
Memory Mode	Flag	Default (no mem flags)	Description
Separate	**-gpu=mem:separate**	OpenACC OpenMP CUF	- All data used from GPU placed in GPU memory and eplixity moved b/w CPU-GPU memories. - Requires explicit data annotations or compiler detection.
Managed	**-gpu=mem:managed**	Stdpar on systems with CUDA Managed Memory only	- Data from explicit dynamic allocations are managed by CUDA Managed Memory - Other data are in separate memories as above.
Unified	**-gpu=mem:unified**	Stdpar on systems with full CUDA Unified Memory (HMM/ATS)	- All data are managed by CUDA Unified Memory automatically

Where is GH200 better than X86+hopper?

HPC and AI cases

Partially Accelerated Applications bound by CPU perf

- OpenFOAM: solver only



Fully or Partially accelerated Applications that bottleneck on PCI connectivity

- CP2K, ABINIT
- Large AI Training

Apps that can leverage tight cache coherence

- Data Assimilation step in weather models can stay on Grace
- AMR methods

New-to-GPU Apps

- Brand-new redesign more effectively leverage standard language acceleration

Large portfolio of applications are partially accelerated

As GPUs become faster applications become increasingly limited by non-GPU factors

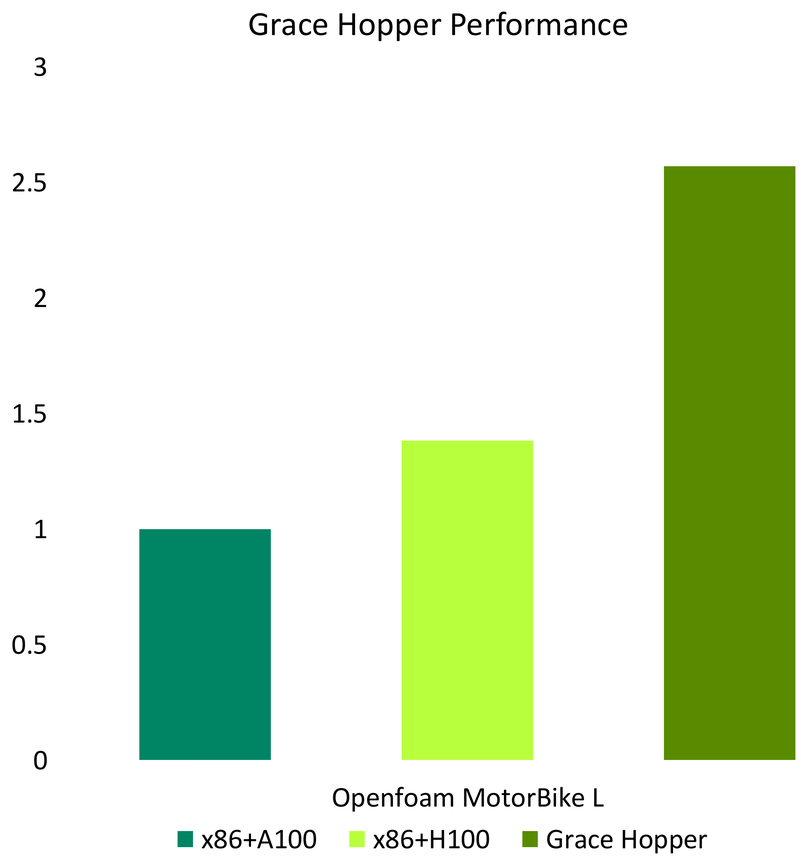
Mostly data transfer (PCIe) limited

Mostly CPU limited

OpenFOAM (Computational Fluid Mechanics)

Partially GPU Accelerated – mostly CPU limited

- Computational fluid dynamics (CFD) toolbox developed by OpenCFD
  - Popular in automotive and other engineering sectors
  - Highly configurable fluid flow solvers with turbulence / heat transfer / etc.
  - Leverage GPU accelerated AMGX linear solvers
- HPC motorbike problem (Large)
  - Around 30% of CPU-only execution is spent in linear solves
- Performance on Grace Hopper
  - High CPU and GPU memory bandwidth improve compute performance
  - C2C bandwidth minimises the cost of migrating CPU matrix data



~35M cells benchmark designed by
OpenFOAM HPC technical committee

OpenFOAM

Motorbike L

x86 + H100 (H100 80GB HBM3)

Grace Hopper (H100 96GB HBM3)