Developing for NVIDIA Superchips

Dr. John Linford, Principal Technical Product Manager

jlinford@nvidia.com
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 Grace Hopper / Blackwell Coherent Memory
NVIDIA AI Accelerated Computing Platform
Hardware and Software Acceleration Across Every Workload and Vertical

CUDA-X Libraries

Accelerated Computing
High Performance, Energy-Efficient Systems in TOP500/GREEN500

Grace Hopper is powering the next wave of new Exaflop AI systems around the globe

Powering Worlds Fastest, Greenest Supercomputers

TOP500
The platform of choice

76% use NVIDIA
1.5 EF added to top10

#1 powered by GH200
5 of the top10 powered by GH200
23 NVIDIA of the top 30

Green500
Energy Efficiency
NVIDIA GH200 Grace Hopper Superchip
Built for the New Era of AI Supercomputing

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

**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

Preliminary measured performance, subject to change.
Grace Hopper Powers AI Supercomputing Datacenters

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

Cumulative AI FLOPS

200 ExaFLOPS

AI Grace Hopper coming online 2024

80% of Hopper are Grace Hopper

2X More energy efficient

7 new GH200 supercomputers
First European Grace Hopper Supercomputer Online

- Fastest AI Supercomputer in Europe
- 20 Exaflops of AI
- 10X more energy efficient than Piz Daint
- Powered by 10,000 Grace Hopper Superchips
- HPC and AI to Advance Weather, Climate (1km global models), and Material Science
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.

Performance of ICON-openACC (ICON-22 model production at MeteoSwiss) measured on March 18, 2024 | 64 GPUs in each case | DGX-H100 80 GB SXM4 | GH200 (24 8 TB per GPU with power cap 360-480W per GPU & 128 GB LPDDR4)

coupled atmosphere at R2B8 with ocean at R2B9 resolution | Atm timestep 90 s, ocean step 5 minutes, coupling step 15 minutes. 90 atm levels, 72 ocean levels.

ICON is a flexible, scalable, high-performance modeling framework for weather, climate and environmental prediction. It provides actionable information for society and advances our understanding of the Earth’s climate system.
NVIDIA GH200 Delivers Breakthrough HPC Performance

Up to 50X more energy efficient

Relative Performance

<table>
<thead>
<tr>
<th></th>
<th>Physics</th>
<th>MILC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD NAMD</td>
<td>1X</td>
<td>6X</td>
</tr>
<tr>
<td>x86 CPU</td>
<td>1X</td>
<td>7X</td>
</tr>
<tr>
<td>x86 + H100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GH200 144GB</td>
<td>41X</td>
<td></td>
</tr>
</tbody>
</table>

Relative Energy Efficiency

<table>
<thead>
<tr>
<th></th>
<th>Physics</th>
<th>MILC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD NAMD</td>
<td>1X</td>
<td>1X</td>
</tr>
<tr>
<td>x86 CPU</td>
<td>1X</td>
<td>6X</td>
</tr>
<tr>
<td>x86 + H100</td>
<td>5X</td>
<td>11X</td>
</tr>
<tr>
<td>GH200 144GB</td>
<td>41X</td>
<td>52X</td>
</tr>
</tbody>
</table>
NVIDIA Grace CPU Superchip
Breakthrough Performance and Efficiency for the Modern Data Center

- **CPU + Memory Power**: 500W
- **Memory Bandwidth**: 1 TB/s
- **Green 500**: 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

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), WRF (CONUS12km), ICON (QUBICC 80 km resolution) specfn3d (four_material_simple_model) and Branson (3D_hohlraum_single_rede)
- Weather: WRF ICON (512km) Grace CPU Superchip vs 2S AMD EPYC 9654
- Graph Analytics: GAP BS Breadth First Search

144 Arm Neoverse V2 Cores  |  234MB L3 Cache  
3.2 TB/s NVIDIA Scalable Coherency Fabric  |  960GB LPDDR5X
Isambard 3
Grace CPU Supercomputer
University of Bristol

- Over 55,000 Arm Neoverse V2 cores
- 2.7 PF of HPC Performance, 270 kW of power
- 6X more performance and energy efficiency vs. predecessor
- Built by HPE
- Enabling breakthroughs in Climate Science, Drug Discovery, and Industrial HPC.
OpenRadioss Crash and Impact

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

**Application Performance**

<table>
<thead>
<tr>
<th></th>
<th>Taurus 10M</th>
<th>FV Model</th>
<th>SkyCAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 CPU A</td>
<td>1.6X</td>
<td>1.5X</td>
<td>1.6X</td>
</tr>
<tr>
<td>x86 CPU B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVIDIA Grace</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Energy Efficiency**

<table>
<thead>
<tr>
<th></th>
<th>Taurus 10M</th>
<th>FV Model</th>
<th>SkyCAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86 CPU A</td>
<td>2.5X</td>
<td>3.2X</td>
<td>3.4X</td>
</tr>
<tr>
<td>x86 CPU B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVIDIA Grace</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NVIDIA Grace Superchip vs x86 flagship 2-socket data center systems (x86 CPU A: Intel® Xeon® W-41700 and x86 CPU B: AMD EPYC™ 7642). Open Radioss Taurus, 10M Shells + Solids | SkyCAB FEA Model | 5.1M Elements | Results subject to change.
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 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
- 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

“Super” \(\rightarrow\) more than a “chip”
NVIDIA CPU + NVIDIA GPU w/o compromises

- **NVIDIA Grace CPU + LPDDR5 Memory**
  - 72 Arm-v9 Neoverse V2 CPU cores with SVE2.
    - Efficiency: 62pJ/DFMA (x86: \(\approx\)99); 1.6x more efficient
    - Performance: 3.6 FP64 TFLOP/s
  - Memory:
    - High capacity: \(\leq\) 480 GB LPDDR5X (5pJ/bit vs 36 DDR)
    - High bandwidth: \(\leq\) 500 GB/s
    - Low latency: less than competitors at peak bandwidth
NVIDIA Grace Hopper Superchip

“Super” → more than a “chip”
NVIDIA CPU + NVIDIA GPU w/o compromises

• NVIDIA Grace CPU + LPDDR5 Memory
  • 72 Arm-v9 Neoverse V2 CPU cores with SVE2.
    → Efficiency: 62pJ/DFMA (x86: ~99); 1.6x more efficient
    → Performance: 3.6 FP64 TFLOP/s
  • Memory:
    → High capacity: ≤ 480 GB LPDDR5X (5pJ/bit vs 36 DDR)
    → High bandwidth: ≤ 500 GB/s
    → Low latency: less than competitors at peak bandwidth

• NVIDIA Hopper GPU
  → High performance: 60 FP64 TC TFLOP/s
  • Memory:
    → High capacity: 96 GB HBM3
    → Extreme bandwidth ≤ 4000 GB/s
    → Threads are threads (not SIMD lanes)
**NVIDIA Grace Hopper Superchip**

Soul is the new **NVLink-C2C** CPU ↔ 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 ↔ 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.
NVIDIA Grace Hopper Superchip
Made ♥ for any programming model
Portable ISO C++, ISO Fortran, Python

• **Simplifies parallelization**: less SW changes

  → **ISO C++, ISO Fortran, Python**: Threads are “threads” (SIMD), memory consistency, automatic memory management, ...

  → **Applications**: complex code stays on CPU, infrequently used memory stays on DDR, large GPU memory capacity (600 GB).

• **Easiest system to**:
  → teach & learn heterogeneous programming
  → parallelize applications
  → use the right HW for each algorithm
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
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 $3.45x \pm 1.07$.

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-Hopper Performance Across a Range of Algorithms

- Faster CPU
  - 3.1 FP64 Tflops
  - 500 GB/s BW

- NVLINK C2C
  - 900 GB/s

- Extended GPU Memory
  - 480GB + 96GB

Relative Performance (x86+A100 = 1)

- Openfoam
- CP2K RPA
- NAMD + Colvars
- CuCo-find
- CuCo-insert
- Hash Join TPC-H Query 4
# Grace CPU Superchip

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

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

OEM-Provided Motherboard

Die 0
n0

Die 1
n1

500 GB/s worst-case n to n

500 Watts (CPU + MEM)

Die 0

Voltage Regulation

LPDDR5X

LPDDR5X

Die 1

500 GB/s

PCIe Peripherals

64 lanes PCIe Gen 5

64 lanes PCIe Gen 5

OEM-Provided Motherboard

Socket 0

n0

n1

n2

n3

Socket 1

n4

n5

n6

n7

x86 Package

x86 Package

8 NUMA Nodes

24 Compute Chiplets

500+ Watts (CPU + MEM)

Die 0

Voltage Regulation

LPDDR5X

LPDDR5X

DDR5 12 Channels

DDR5 12 Channels

500 GB/s worst-case n to n

460 GB/s

PCIe Peripherals

64 lanes PCIe Gen 5

64 lanes PCIe Gen 5

500 GB/s

460 GB/s
Grace Hopper Superchip

GPU can access CPU memory at CPU memory speeds

NVIDIA Grace Hopper Superchip

CPU LPDDR5X
120GB | 480GB

GPU HBM3
96 GB HBM3

500GB/s | 375GB/s

4000 GB/s

18x NVLINK 4
900 GB/s

4x 16x PCIe-5
512 GB/s

18x NVLINK 4
900 GB/s

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

- **NVIDIA Software Ecosystem**
  - Advancing the state-of-the-art standards (Standard Language Parallelism, CUDA, etc.)

- **Optimized OSS or Vendor Software (Armv9)**
  - Align with commercial momentum (CSP, Neoverse, etc.)

- **Arm Software Ecosystem (Armv8 SBSA)**
  - The most common computing architecture on the planet

---

*Optimal Executable*  
*Portable, Optimized, Accelerated Executable*  
*Optimized Executable*  
*Portable Executable*
Programming the NVIDIA Platform
Unmatched Developer Flexibility

Languages & Programming Models
- Accelerated Standard Languages
  - C++
  - Python
  - Fortran
- Incremental Optimization
  - OpenACC
  - OpenMP
- Platform Specialization
  - CUDA
  - C++ | Fortran | Python

Libraries & Frameworks
- Acceleration Libraries
  - Core
  - Math
  - Communication
  - Data Analytics
  - AI
  - DSLs
- Hardware
  - GPU
  - CPU
  - Interconnect

Wherever You Run
- PCs
- Workstations
- On Prem
- Cloud
- At the Edge
NVIDIA HPC SDK

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

Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
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

- **cuFFTMp**: 2D and 3D FFTs Decompositions: Slab (1D), Pencil (2D)
- **cuSOLVERMp**: Factorization, Symmetric Eigensolver
- **cuBLASMp**: GEMM, TRSM, SYRK at scale for FP16, FP32, FP64 types

**Communication Libraries**
- HPC-X
- MPI
- UCX
- SHMEM
- SHARP
- HCOLL
- NVSHMEM
- NCCL

- **DGX H100 8 GPUs**
- **Infiniband Between Nodes**

**DGX H100 Super POD**

**DGX GH200**

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

X86 + ARM support
HPC/AI Software Ecosystem for Grace
Developed by NVIDIA and the Arm OSS community

OS
All Major Linux Distributions

Arm ServerReady SR
Standard firmware and RAS

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

Applications and Frameworks
AI/ML/DL Frameworks, HPC, CSP, Commercial ISV, General Compute

Developer Tools
NVIDIA Nsight + Open Source and Commercial

Cluster Management
NVIDIA Bright Cluster Manager + Open Source and Commercial

Containers & Orchestration
NVIDIA NGC + Open Source and Commercial

Schedulers
Open Source and Commercial

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

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

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

Applications and Frameworks
AI/ML/DL Frameworks, HPC, CSP, Commercial ISV, General Compute

Developer Tools
NVIDIA Nsight + Open Source and Commercial

Cluster Management
NVIDIA Bright Cluster Manager + Open Source and Commercial

Containers & Orchestration
NVIDIA NGC + Open Source and Commercial

Schedulers
Open Source and Commercial

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

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
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
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

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Linux Distributions</th>
<th>CUDA Toolkit</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAarch64</td>
<td>Ubuntu 22.04</td>
<td>12.2U2 and later</td>
</tr>
<tr>
<td></td>
<td>RHEL 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CentOS 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLES 15-SP4</td>
<td></td>
</tr>
</tbody>
</table>

Download Now
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

- **Already Ported**: 43%
- **Non-trivial Port**: 8%
- **Trivial Port**: 49%

**Job done!**
Found on Arm at another HPC center

**Straightforward, easy work < 1 day**
Recompile and reconfigure runtime parameters

Nonstandard compilers, assembly language, vector intrinsics, or dependency
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, GNU Make, EasyBuild, Spack etc.
- Compiler & build system performance is excellent
  - Ampere Altra compilation performance is on is on-par with AMD EPYC 7742 – **you do not need to cross compile**

[https://www.anandtech.com/show/16315/the-ampere-altra-review](https://www.anandtech.com/show/16315/the-ampere-altra-review)
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](https://example.com) 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](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

Armv8: No SVE!

Correct ISA

```
.strength-profile-rng:register:sve2 bitperm:18mm:bf16
.library
```

-march=armv9-a
Correct instruction, limited ISA

-mtune=neoverse-v2
Library call instead of atomic instruction, limited ISA

-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/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

• 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/gtc.spring22-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
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](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

**Compilers**
- Auto-vectorization: NVIDIA, GCC, LLVM, ACfL, Cray...
- 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
- Arm Scalable Vector Extensions and application to Machine Learning

**Assembly**
- See SVE ISA Specification
- The Scalable Vector Extension for Armv8-A

**Libraries**
- BLAS
- LAPACK
- PBLAS
- SCALAPACK
- TENSOR
- SPARSE
- RAND
- FFTW

// Complex dot product
// (c[i] = a[i] + b[i] = (ac - bd) + i(ad + bc))
void complex_dot_product(Complex_t c[SIZE], Complex_t a[SIZE], Complex_t b[SIZE])
{
    wint32_t vl = svlenw();
    svbool_t p32_all = sv羿元_S32();
    for (int i=0; i<SIZE; i+=v1) {
        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 = svvadd_m(p32_all, vc.v0, va.v0, vb.v0); //c.re += a.re * b.re
        vc.v1 = svvadd_m(p32_all, vc.v1, va.v1, vb.v1); //c.i += a.im * b.re
        vc.v2 = svvadd_m(p32_all, vc.v2, va.v2, vb.v2); //c.re -= a.re * b.im
        vc.v3 = svvadd_m(p32_all, vc.v3, va.v3, vb.v3); //c.im += a.im * b.im
        svint32(p32_all, float32_t c[i], vc);
    }
}
Neoverse V2 Core Details
Arm Neoverse V2 Core

Overview

- Arm® Neoverse™ V2 Core Technical Reference Manual

- Arm Neoverse V2 implements Armv9.0-A architecture
  - Extends Armv8.0-5-A architecture
  - 128-bit vector length SVE & SVE2
  - 128-bit vector length ASIMD (a.k.a., NEON)
  - Learn the architecture - Understanding the Armv8.x and Armv9.x extensions

- Separate L1 data and instruction caches
  - L1 instruction memory system
    - 64KB, 4-way set associative, 64B cache line
    - Fully associative L1 instruction TLB, support for {4,16,64}KB and 2MB page sizes
    - 1536-entry, 4-way skewed associative L0 MOP cache
    - Dynamic branch predictor
  - L1 data memory system
    - 64KB, 4-way set associative, 64B cache line
    - Fully associative L1 data TLB, support for {4,16,64}KB page sizes and {2,512}MB block sizes

- Private, unified data and instruction L2 cache
  - 1-2MB (1MB for Grace), 8-way set associative
Arm Neoverse V2 Core
Core pipeline & general information

- **Arm Neoverse V2 Core Software Optimization Guide**
- Up to 8 instructions decoded into internal MOPs
- Each MOPs can be split into 2 uOPs
- Total of 16 uOP with some limitations
  - 2, 4 or 6 uOPs depending on the pipelines used
  - FP/ASIMD pipelines process FP, NEON, SVE and SVE2
- Some instructions might use more than one pipeline
  - e.g., gather load will use a Load and FP/ASIMD pipelines
- Instruction latency/throughput is variable
  - 4x Scalar/NEON/SVE FP64 FMA per cycle
- **Intrinsics – Arm Developer**
  - Includes intrinsic to assembly code information
Arm Neoverse V2 Core

Write streaming mode

• Avoids polluting cache when writing big chunks of data with no reads
  • e.g., memset to initialize data structures

• Enabled when core detects when a full cache line has been written before the line fill completes
  • i.e., you write to cache faster than you load cache lines with memory locations being written to

• While in streaming mode
  • Loads behave as normal
  • Writes lookup cache, if miss, write to L2 cache or system memory instead of generating a line fill

• Streaming mode is disabled when
  • System detects a cacheable write burst that is not a full cache line
  • There is a load operation from the same line that is being written to L2 cache

• Examples
  • \(a[i] = b[i] + c[i]\) will cause the core to go into streaming mode
  • \(a[i] = a[i] + c[i]\) will keep the core in non-streaming mode
Arm Neoverse V2 Core

Special considerations for compilers and low-level library developers

- Memory alignment
  - Generally, no penalty for unaligned memory accesses
  - Penalty can occur when
    - Crossing cache line (64B) boundary
    - Quad word loads that are not 4B aligned
    - Stores that cross a 32B boundary

- Memory routines
  - memcpy
    - Unroll loop to include multiple loads and store ops per iteration
    - Align loads to 16B boundary when possible
    - Use non-writeback forms of LDP/STP (load/store pair of registers)
  - memset
    - Unroll loop to include multiple stores per iteration
    - For memset to 0, use DC ZVA instructions instead of store (might be not recommended for small memset)

- Branch instruction alignment
  - Avoid placing more than four branch instructions within an aligned 32B instruction memory region

- AES encryption/decryption
  - At least 8 data chunks should be interleaved to achieve full pipeline utilization
  - Pairs of dependent AESE/ASEMC and AESD/AESIMC instructions should be adjacent in code and using the same destination register
Optimizing for Coherent Memory
Grace Hopper Superchip

GPU can access CPU memory at CPU memory speeds

NVIDIA Grace Hopper Superchip

CPU LPDDR5X
120GB | 480GB

500GB/s | 375GB/s

GPU HBM3
96 GB HBM3

4000 GB/s

18x NVLINK 4
900 GB/s

12x PCIe-5
512 GB/s

CPU LPDDR5X

GRACE CPU

NVLINK C2C
900 GB/s

HOPPER GPU

Hardware Consistency

4x 16x PCIe-5

512 GB/s

512 GB/s

NVLink NETWORK
≤ 256 GPUs

https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper
GPU Memory is Visible to the Operating System

Standard operating system commands work on the GPU

- Hopper GPU appears to the OS as a NUMA node with no CPU cores
- Total system memory capacity is CPU (480GB) + GPU (96GB)
- 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)

CPU

PHYSICAL MEMORY

Page A

GRACE CPU

NVLINK C2C

HOPPER GPU

CPU

PHYSICAL MEMORY

Page B

CPU resident access

Remote accesses

GPU resident access

System Page Table translates CPU malloc() to CPU or GPU
Grace Hopper Automatic Page Migration

CPU
PHYSICAL MEMORY

Page A

GRACE CPU

NVLINK C2C

HOPPER GPU

GPU
PHYSICAL MEMORY

Infrequent accesses

Frequent accesses

Migration improves performance

Page A

PTEA
High Bandwidth Memory Access & Automatic Data Migration

Hopper can access Grace memory at full CPU memory speed of 500 GB/sec. Note: 480GB CPUs have 25% lower LPDDR5x bandwidth.
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

- LPDDR5X: 120GB
- GRACE CPU
- NVLink C2C: 900 GB/s
- HOPPER GPU
- HBM3: 4000 GB/s

Achieved: 3732 GB/s
## Memory Allocators Impact Data Placement and Movement

### CUDA 12.4

<table>
<thead>
<tr>
<th></th>
<th>cudaMalloc</th>
<th>cudaMallocManaged</th>
<th>Malloc/mmap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Placement</td>
<td>GPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Page size</td>
<td>2MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Which processor can access?</td>
<td>GPU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>How does access happen?</td>
<td>GPU MMU</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What can the driver do for my app?</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>What can I do for my app?</td>
<td>Think how to potential leverage coherent systems</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Memory Allocators Impact Data Placement and Movement

CUDA 12.4

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

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**

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

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

<table>
<thead>
<tr>
<th>Memory</th>
<th>Placement</th>
<th>Access-based Migration</th>
<th>Accessible From</th>
</tr>
</thead>
<tbody>
<tr>
<td>System-allocated (malloc, mmap)</td>
<td>First-touch</td>
<td>✔️ ✔️ ✔️</td>
<td></td>
</tr>
<tr>
<td>CUDA managed (cudaMallocManaged)</td>
<td>(GPU</td>
<td>CPU)</td>
<td>✔️ ✔️ ✔️</td>
</tr>
<tr>
<td>CUDA device memory (cudaMalloc)</td>
<td>GPU</td>
<td>❌ ❌ ✔️</td>
<td></td>
</tr>
<tr>
<td>CUDA host memory (cudaMallocHost)</td>
<td>CPU</td>
<td>❌ ✔️ ✔️</td>
<td></td>
</tr>
</tbody>
</table>

...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 |
- **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 auto-vectorization
  - 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
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

36 GRACE CPUs
72 BLACKWELL GPUs
Fully Connected NVLink Switch Rack

<table>
<thead>
<tr>
<th>Feature</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>720 PFLOPs</td>
</tr>
<tr>
<td>Inference</td>
<td>1,440 PFLOPs</td>
</tr>
<tr>
<td>NVL Model Size</td>
<td>27T params</td>
</tr>
<tr>
<td>Multi-Node All-to-All</td>
<td>130 TB/s</td>
</tr>
<tr>
<td>Multi-Node All-Reduce</td>
<td>260 TB/s</td>
</tr>
</tbody>
</table>
Grace CPU Benchmarking Guide

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