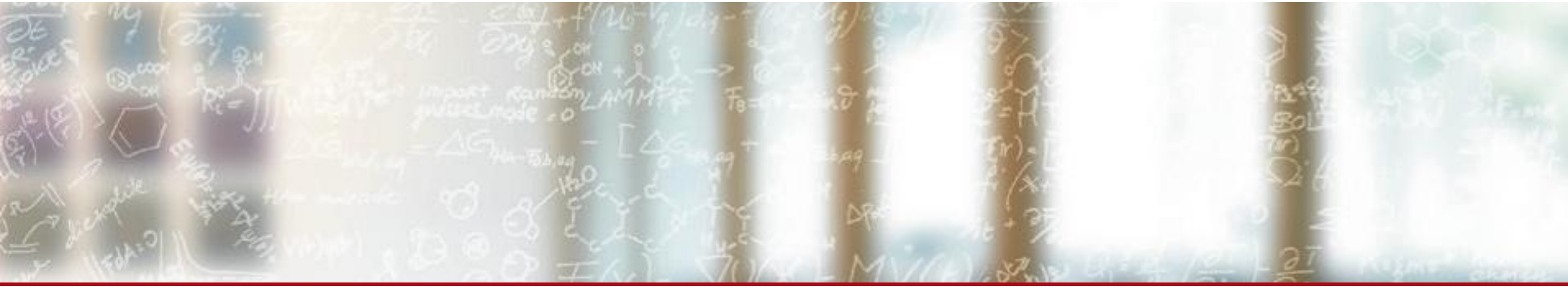


Preparing for the Migration from Daint-GPU to Grace-Hopper

Ben Cumming

User Lab Webinar

June 6, 2023

An Introduction:

who am I?

A Disclaimer:

The information presented here is from discussions with NVIDIA.

CSCS has, as of today, access to a Grace-Hopper module for development.

Agenda

The focus of the webinar is to understand the impact of migrating from Daint-GPU to Grace-Hopper on Alps - specifically on applications.

We plan more practical hands-on events when Grace-Hopper is available.

Start with an overview of what the new system and how it differs from the old:

- Alps Phase 2 - what is changing for users of Daint-GPU?
- Comparison of Daint-GPU and Grace-Hopper nodes
- An overview of Grace-Hopper

Before diving into the impact this will have on applications and developers:

- Status of Application and library readiness for Grace-Hopper
- Getting the most out of Grace-Hopper
- The user environment on Alps

Alps Phase 2

Alps Phase 2

Alps HPC infrastructure is a HPE Cray-EX system

- Phase 0 and Phase 1 are already installed
  - **2020: Phase 0** – AMD Rome CPU nodes currently used in Eiger
  - **2022: Phase 1** – NVIDIA A100, AMD Mi250x and AMD Milan CPU nodes
- Phase 2
  - **Q1 2023: Phase 2** - NVIDIA Grace-Hopper GH200 nodes

The User Lab scale out Grace-Hopper platform will be the largest **tenant** on Alps

- It will replace Daint-GPU with the same number of Grace-Hopper modules (>5000) as there are on Daint-GPU today.

Alps Phase 2 – some perspective

Replacing a system is can be disruptive - particularly when changing architecture

- Migration from Rosa to Daint was disruptive
  - CSCS and the wider community invested years of effort beforehand porting to GPUs
    - My first years at CSCS were spent on this effort
    - 2013: Daint-GPU was installed with Sandy-Bridge CPU + K20x GPUs
    - 2016: Daint-GPU was upgraded to Haswell + P100 GPUs
  - CSCS and the HPC community have 10 years of development and experience under our belts.

Accelerator-based systems are now standard

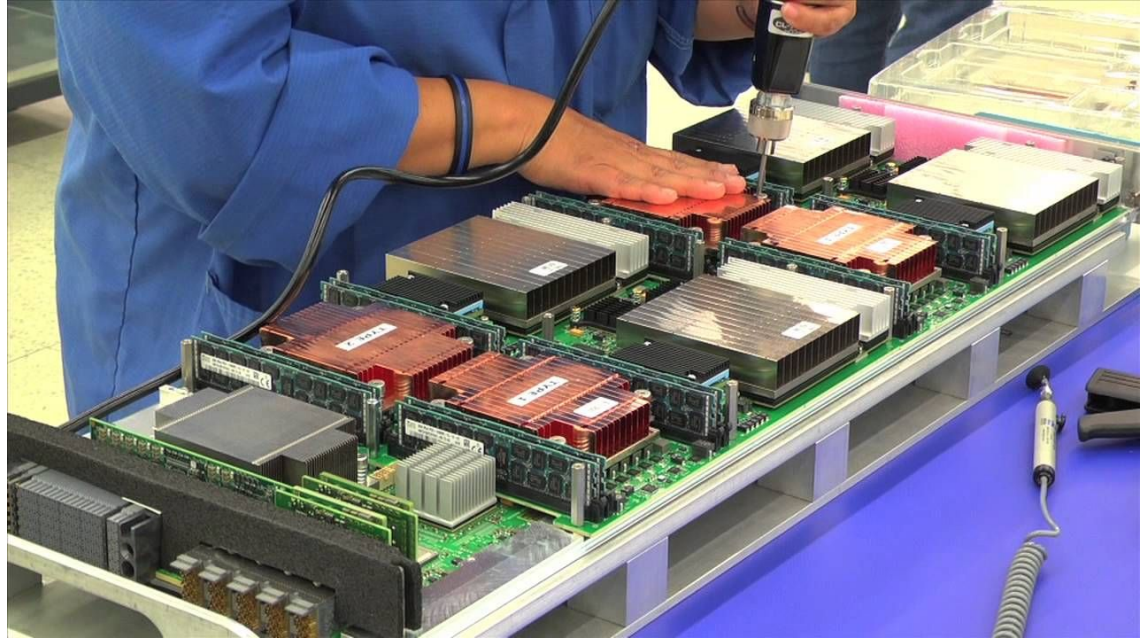
- EU and US flagship systems use accelerators
- Migration effort depends on the accelerator architecture

CSCS expect no negative impact on application availability

- Applications running on Daint-GPU will run with little porting effort on Grace-Hopper

Grace-Hopper

Daint-GPU nodes



Daint-GPU node has a simple architecture

- 1 Haswell CPU socket
- 1 P100 GPU
- PCI-E connection between host-device
- 1 NIC

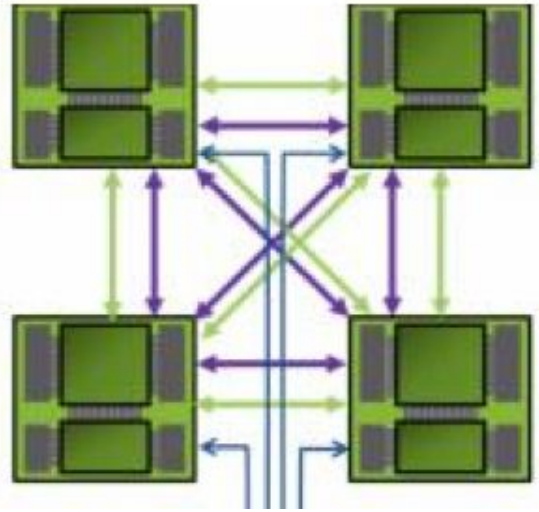
The ratio of 1-1 made allocating MPI ranks relatively simple:

- One rank per GPU + CPU
- Or multiple ranks sharing the GPU using CUDA MPS (multi-process service)

Assigning CPU resources to GPU on x86+A100/H100 nodes is more challenging

- A single CPU socket with multiple NUMA regions is divided between GPUs.

Alps Phase II nodes



Grace-Hopper modules are *conceptually similar*

- 1 Grace CPU socket and one Hopper GPU per module
- Cache-coherent NVLINK connection between host and device memory
- One NIC per module

Each node will have 4 Grace-Hopper modules

- All-to-all cache-coherent memory NVLINK between all host and device memory

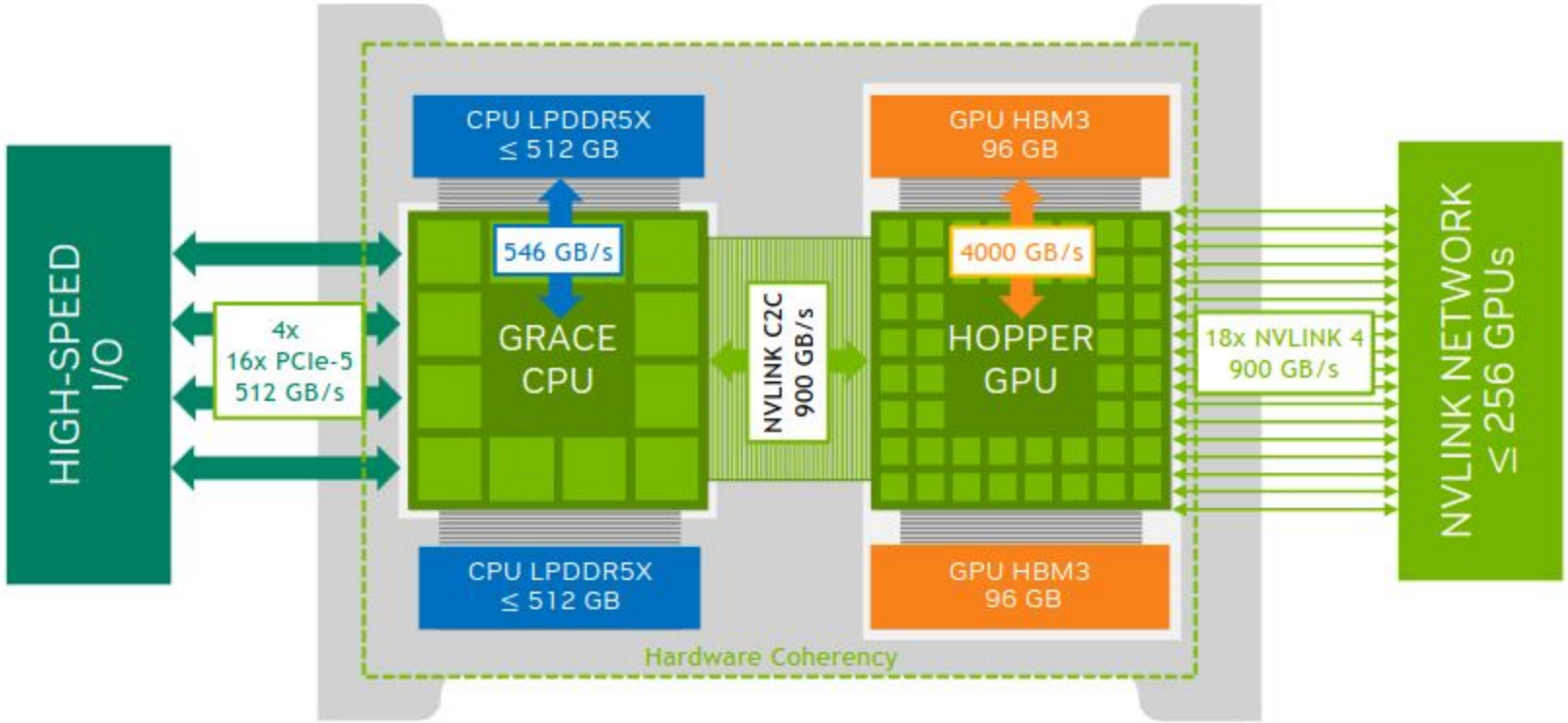
The one-to-one CPU to GPU ratio remains

- The 4 modules on a node form an optimised communication network.

The Grace-Hopper “Super Chip”

NVIDA are releasing are two super chips:

1. Grace-Grace: dual-socket Grace CPU with NVLINK C2C
2. Grace-Hopper: Grace CPU + Hopper GPU with NVLINK C2C → Alps Phase II



Feeds and speeds: Daint-GPU node vs. one GH module

Comparing the raw speeds and feeds of the CPU and GPU

GPU	P100	Hopper	Increase
Bandwidth	700 GB/s	4000 GB/s	5.7x
FP64	4.7 TFlops	34/67 TFlops	7-14x
Memory	**16 GB HBM**	**96 GB HBM3**	**6x**

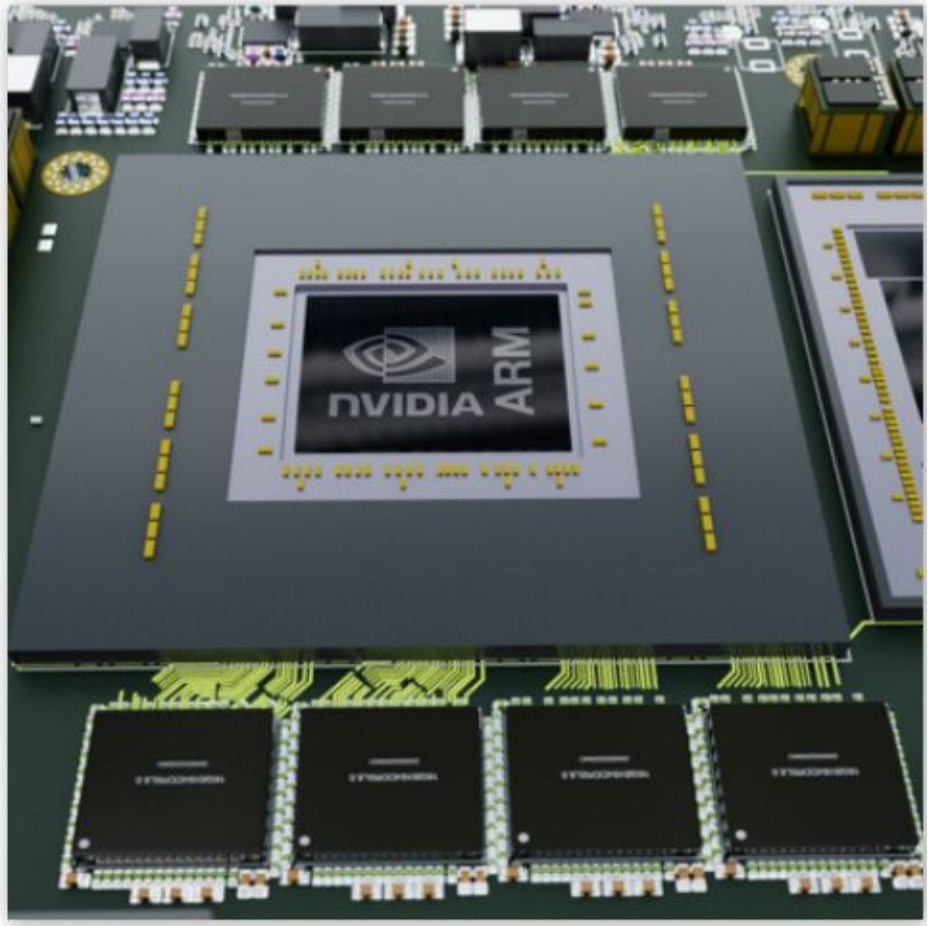
CPU	Haswell	Grace	Increase
Cores	12	72	6x
Bandwidth	**60 GB/s**	**475 GB/s**	**8x**
FP64	0.49 TFlops	> 2.5 TFlops	5x
Memory	64 GB DDR3	128 GB LPDDR	2x

Data Movement	Daint-GPU	Alps Phase II	Increase
Host-Device	**22 GB/s**	**480 GB/s**	**20x**
Device-Device on node	-	900 GB/s	-
node-node	11 GB/s	23 GB/s	2x

- The Grace-Hopper module delivers 5-10x improvement across the board
- Speedup may be lower or higher depending on the existing bottlenecks.

Grace: Server Class ARM CPU

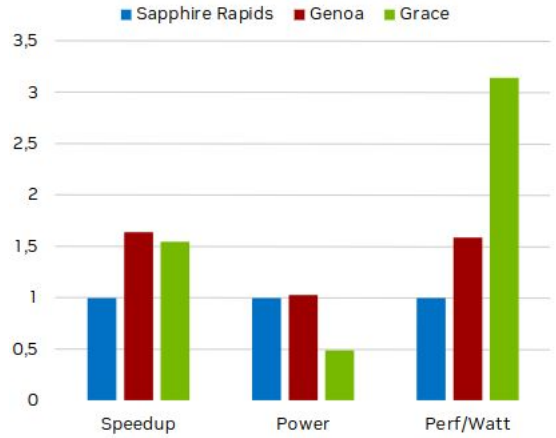
- 64bit Server Class Core and SoC
  - Arm V9.0 ISA Compliant aarch64 core (Neoverse V2 “Demeter”architecture)
  - Full SVE-2 Vector Extensions support, inclusive of NEON instructions
  - Supports 48-bit Virtual and 48-bit Physical address space
- Implemented on 5nm Process technology
- Balanced architecture between Single Core Perf, Core count, Memory and IO subsystems

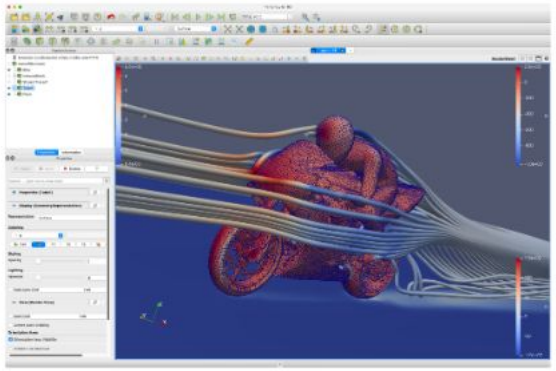


Grace Performance

OPENFOAM 2206

MotorBike 5M





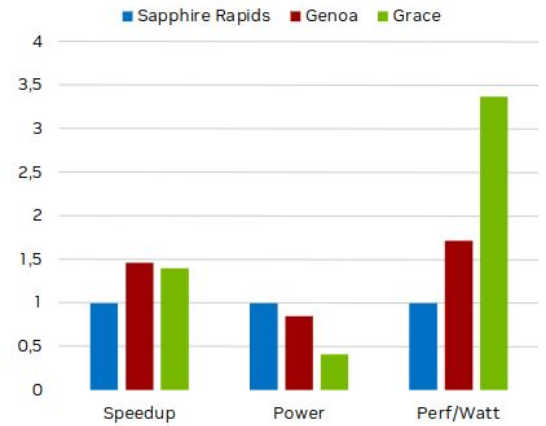
Sapphire Rapids: Intel Xeon Platinum 8470Q, 52c @ 2.1GHz - 3.8GHz
Genoa: AMD EPYC 9654, 96c @ 1.5GHz – 3.7GHz
Grace: NVIDIA Engineering Sample, 72c @ 3.2GHz
Best single socket time using best compilers (GCC, ICC, AOCC) and best rank/thread decomposition

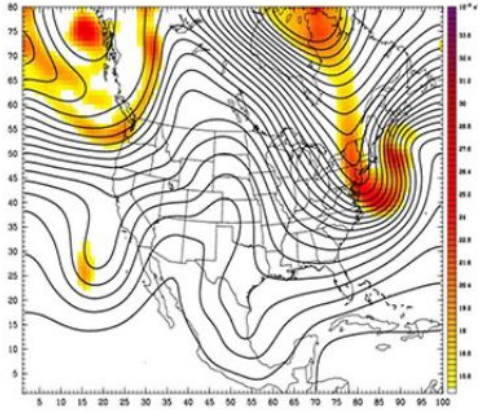
Results based on early engineering samples of Grace

- Grace will be competitive with x86 HPC CPU architectures.
- Each GPU will have a full CPU socket – workloads that can "reverse offload" or have CPU-intensive components will benefit.

WRF 4.4.2

CONUS12, 24hr simulation time





Sapphire Rapids: Intel Xeon Platinum 8470Q, 52c @ 2.1GHz - 3.8GHz
Genoa: AMD EPYC 9654, 96c @ 1.5GHz – 3.7GHz
Grace: NVIDIA Engineering Sample, 72c @ 3.2GHz
Best single socket time using best compilers (GCC, ICC, AOCC) and best rank/thread decomposition

Application Readiness

Will applications be ready?

- Most applications and libraries will work out of the box
- For example, we asked NVIDIA about the status of CSCS supported applications:

Application	Status	Comments
Amber	A	
LAMMPS	A	
NAMD	A	
CP2K	A	
GROMACS	A	
Quantum Espresso	B	Waiting for a perf bug fix; good perf on Arm+A100 & x86+H100
CPMD	B	Running on GH, perf analysis to do.
VASP	C	Has been run on ARM H100

*** ML/AI Frameworks, ICON, Openfoam and many others have been tested and tuned internally by NVIDIA*

A	Tested & validated with good perf
B	Tested and validated
C	Not tested

Will applications be ready?

CSCS will soon have access to a Grace-Hopper module to test programming environments and application readiness

- The most likely cause of issues will be moving from x86 to ARM CPU
  - CSCS has evaluated ARM Graviton 3 on AWS - a Neoverse V1 processor
  - In a short time we could compile our applications
    - C, C++, Python and Fortran are all supported
  - There are some rough edges in the ecosystem - e.g. Spack packages and compiler flags need updating.
- The two causes of issues will be:
  - x86 intrinsics assembly and intrinsics
    - These won't be used so frequently for GPU-accelerated codes
  - x86 specific libraries like MKL
    - OpenBLAS and BLIS will provide BLAS/LAPACK implementations for Grace

Will applications be ready?

GPU-enabled applications will not need significant porting

- Some fixes and tuning will be required for many applications
- With very good performance
- Some applications will benefit significantly from the new host-device memory model

There are three categories of GPU accelerated applications

- Fully GPU accelerated
- Partially GPU accelerated
- Coherently GPU accelerated

Fully GPU accelerated

Compute and data are on the GPU

- Little or no limitation from CPU or data transfers

Grace-Hopper and x86+Hopper will be equivalent

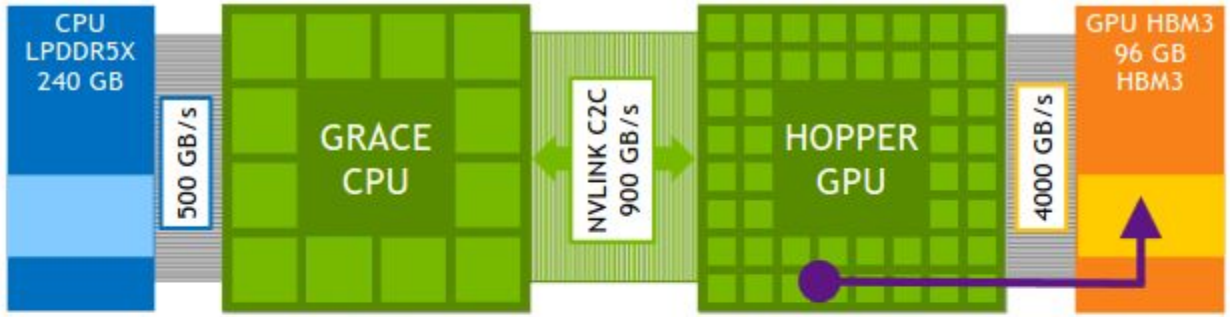
Partially GPU accelerated

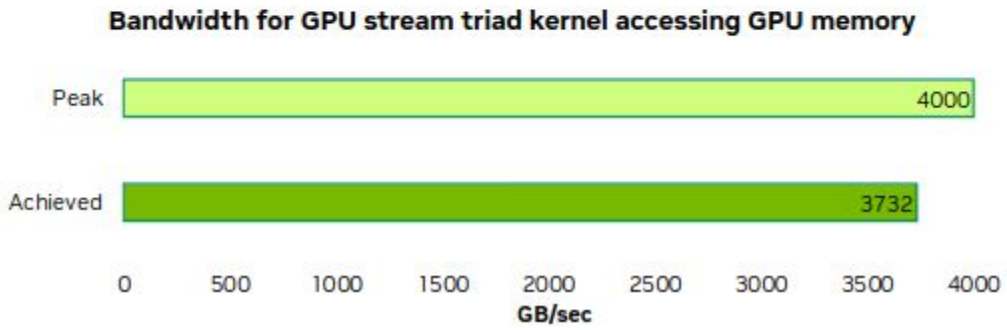
More powerful GPUs lead to applications spending a larger proportion of time on non-GPUoperations

- Data transfer (PCIe) limited

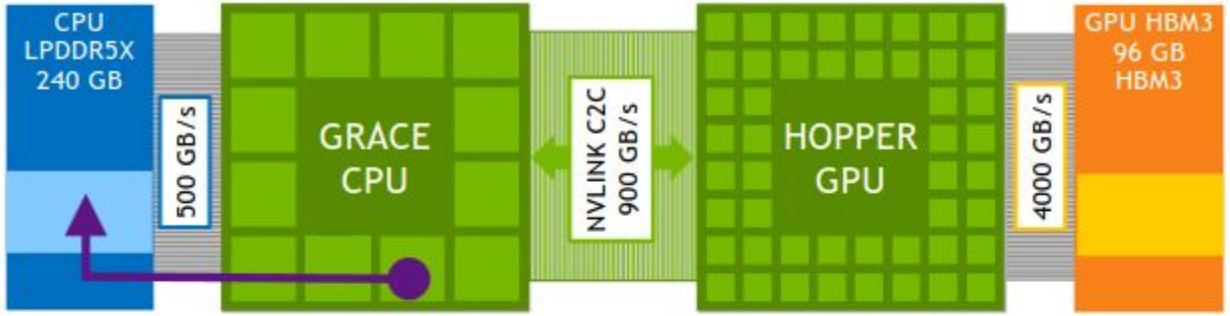
- CPU limited:

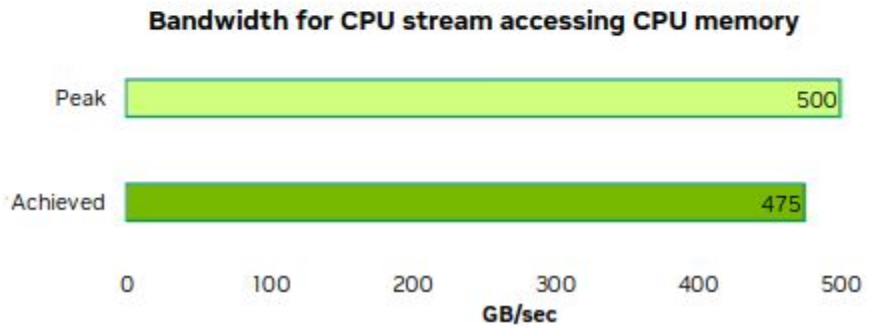
Hopper-HBM3


CPU
LPDDR5X
240 GB
500 GB/s
GRACE
CPU
NVLINK C2C
900 GB/s
HOPPER
GPU
4000 GB/s
GPU HBM3
96 GB
HBM3

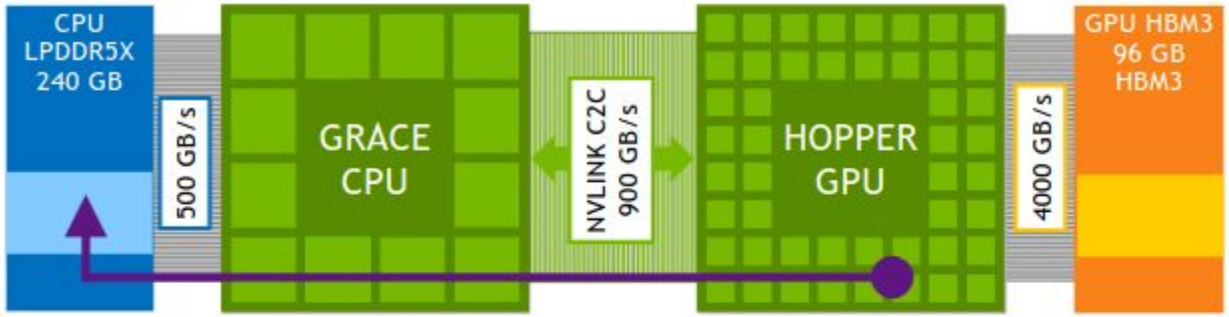

Bandwidth for GPU stream triad kernel accessing GPU memory
Peak
4000
Achieved
3732
0
500
1000
1500
2000
2500
3000
3500
4000
GB/sec

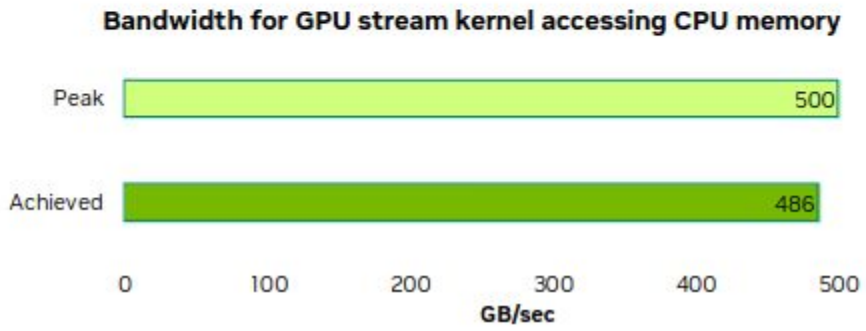
Grace-LPPDR5





Hopper-LPDDR





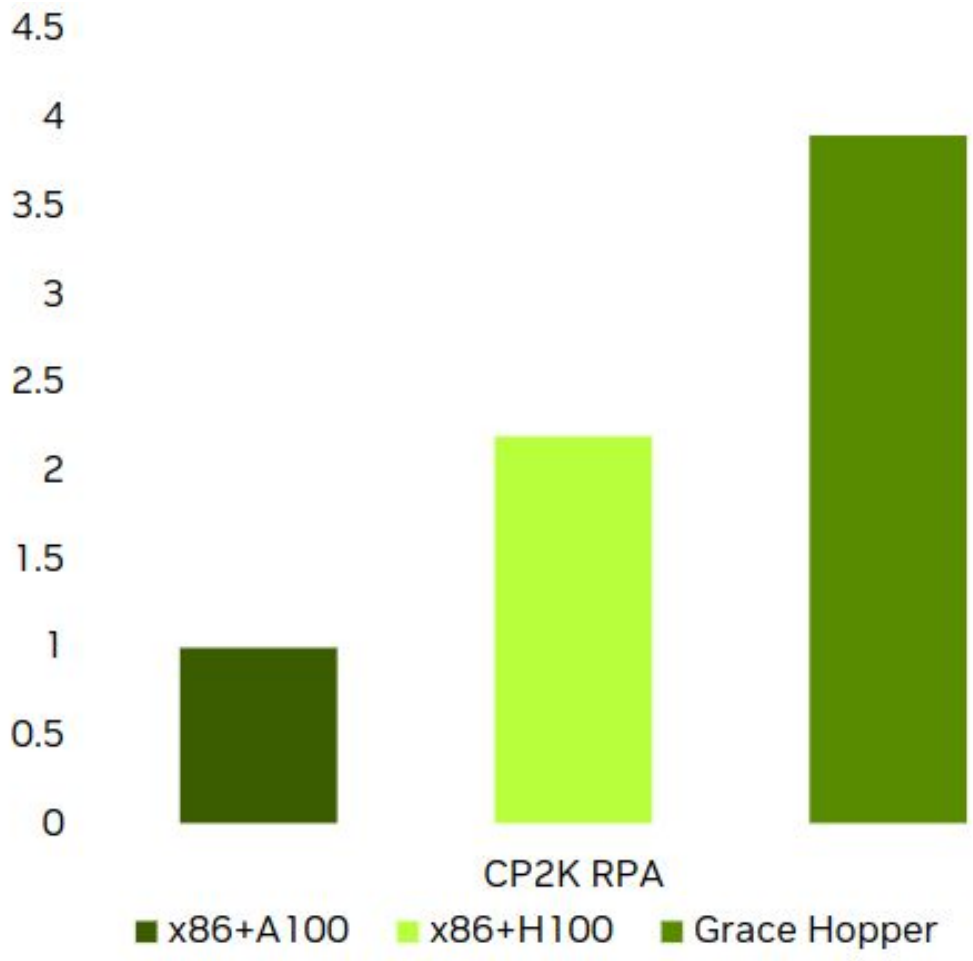
CP2K: Partially GPU Accelerated – mostly data transfer limited

CP2K Quantum chemistry application

- Implements a suite of methods - many not yet GPU accelerated
- Memory constraints require

Dataset **128-H20** with random-phase approximation (RPA) method

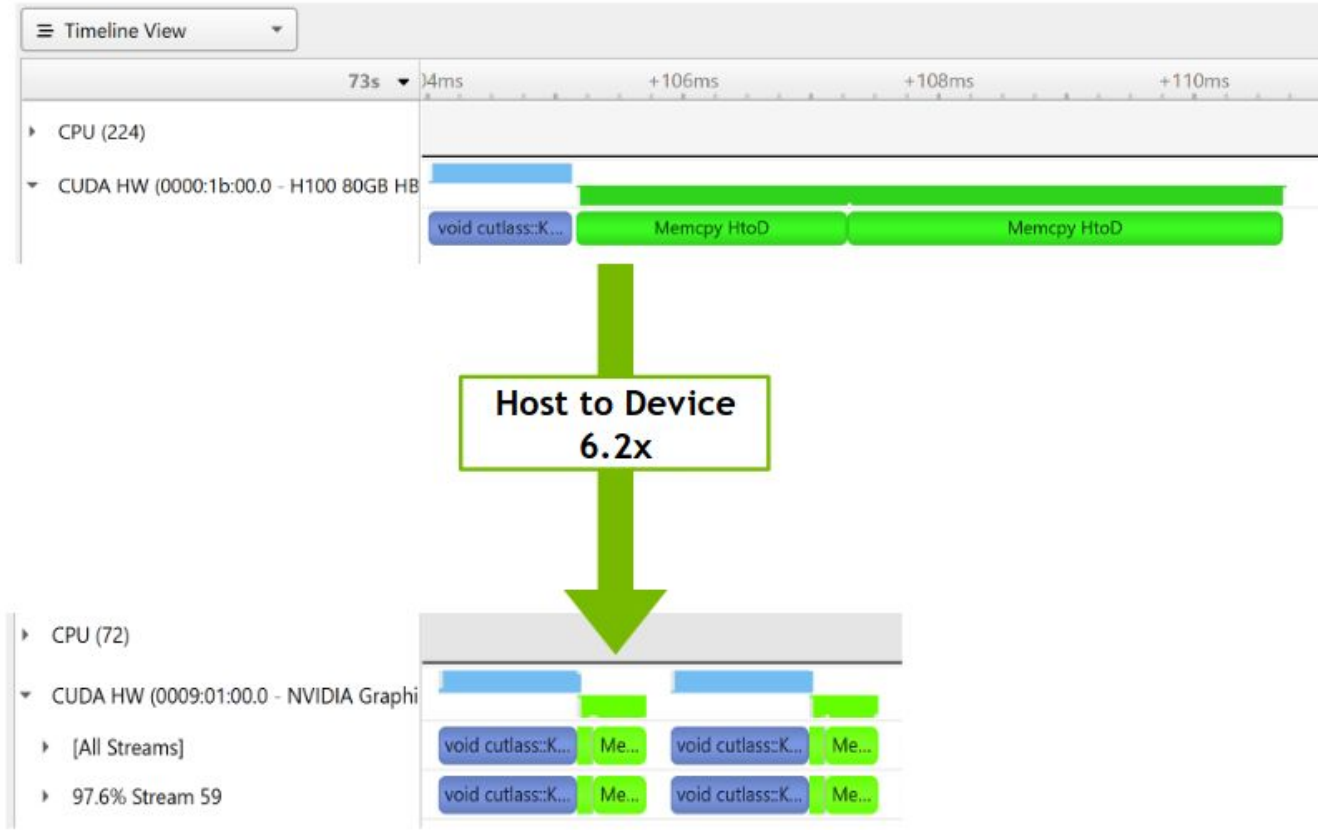
- PDGEMM dominates -> use GPU PDGEMM
- Performance bounded by CPU memory BW, PCIe and MPI



CP2K: Partially GPU Accelerated – mostly data transfer limited

x86 + H100 (H100 80GB HBM3)

Grace Hopper (H100 96GB HBM3)



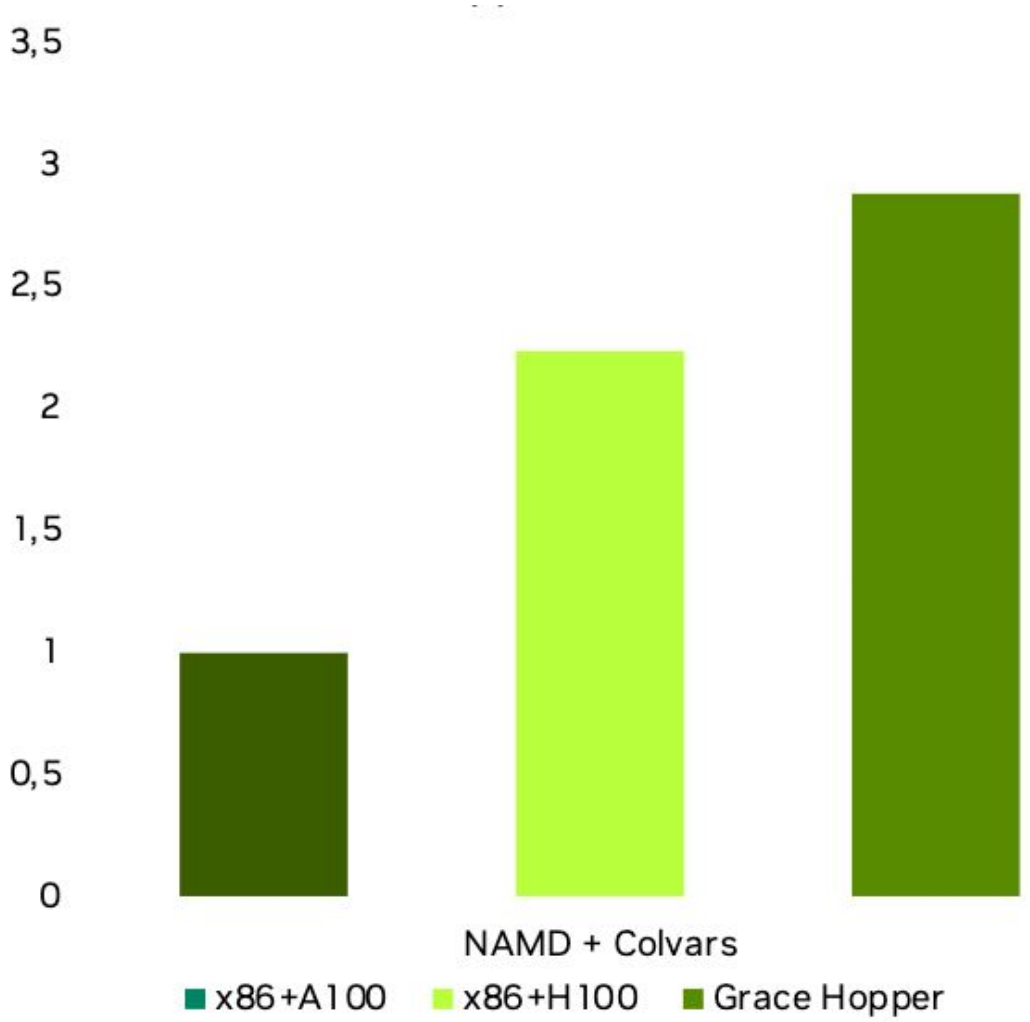
NAMD: Partially GPU Accelerated – mostly CPU limited

Molecular Dynamics simulation

- Collective Variables provided by 3rd party Colvars
- CPU-only also used by Gromacs, LAMMPS

Glucose transporter 3 system (143k atoms)

- Majority of forces are GPU accelerated
- Additional Colvars computed on CPU
- Major bottleneck was CPU memory bandwidth bound.



Coherently GPU accelerated – Optimised for coherent architectures

Exploit GPU / CPU coherency

- Use all available system features
- Use fine-grained synchronisation primitives
- “Reverse-offload” operations that no longer need to run on the GPU to avoid PCIe

This is a new model

- Enabled by coherent CPU-GPU memory architectures
- Optimisation will occur over a longer period of time
- Also applies to AMD Mi300a GPU architecture

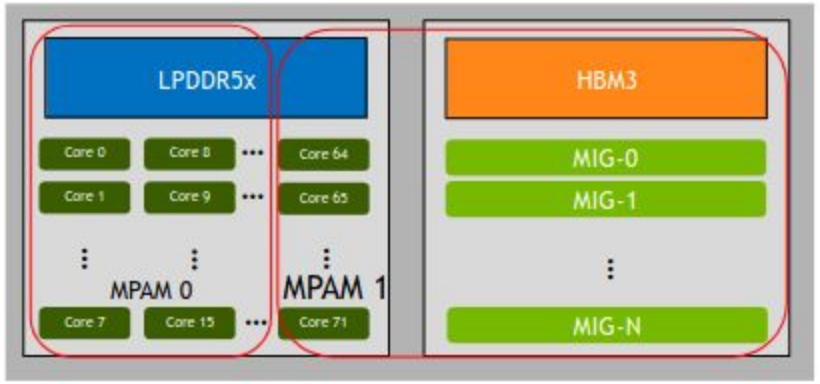
- ML/AI users will see the biggest boost
  - Features like tensor cores and lower precision types were first introduced in V100
  - The ML/AI software ecosystem is the highest priority for NVIDIA
  - Inter-node scaling over SlingShot 11 will be CSCS' focus

- Will migration mitigate your main bottleneck on Daint-GPU?
  - Memory capacity
    - Squeeze your workload onto fewer GPUs - a single GPU or single 4x node
    - Leave some data on the host and access using C2C NVLINK
  - Host-Device migration
    - C2C NVLINK will reduce the overhead
    - More opportunities to leave data in place and compute where it makes sense.
  - Dense linear algebra or Memory Bandwidth
    - Easy win!
  - Not enough work for a single GPU
    - CPU and GPU resources to fit multiple jobs per GPU

Flexible distribution of resources

Grace-Hopper supports flexible allocation of CPU and GPU resources over multiple jobs and tasks

One Job exclusive to the node



Job A - 64 Grace CPU MPAM
Job B - 8 Grace Cores MPAM + Hopper GPU

Job A - 8 Grace Cores MPAM + Hopper MIG
Job B - 8 Grace Cores MPAM + Hopper MIG
Etc.

- Assumptions about which data to move, which compute to move and how to synchronise them optimally will change
  - Applications that have been optimised to move all data and compute to GPU will still perform very well out of the box
  - Applications for which it has been impossible or impractical to port fully to GPU will be the biggest beneficiary
    - Removing the memory bottleneck enables performing compute where it makes sense.

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

User Environments on Alps

Programming Toolchains

There will be three compiler toolchains

1. GCC + CUDA
   - C, C++, Fortran, nvcc
2. NVIDIA HPC SDK
   - C, C++, Fortran, OpenACC
3. Cray
   - C, C++, Fortran, OpenACC

Python Frameworks will also be ready to use

- Pytorch, TensorFlow, Jax, etc

The Piz Daint user environment: one size fits all

Interactions with Daint use the well-established, tried-and-trusted HPC interface:

- Log in via ssh to a shell with a programming environment loaded
- Manipulate the environment using modules
  - Select Cray-provided compilers, libraries and tools
  - Select CSCS-provided libraries, tools and applications
- Use shared resources including storage and job scheduling.

All workflows, such as JupyterLab, FirecREST present a layer over this

- Underneath, all access is via a standard user account

As the number and variety of use cases expand

is the “one size fits all model” a good fit?

Stability vs. Bug Fixes and New Features

Any feedback that in your opinion can help us improve the HPC environment?

Would it be possible to keep older versions?

I am very satisfied with the HPC environment on Piz Daint

Compilers that support the newest C++ standards as well as possible

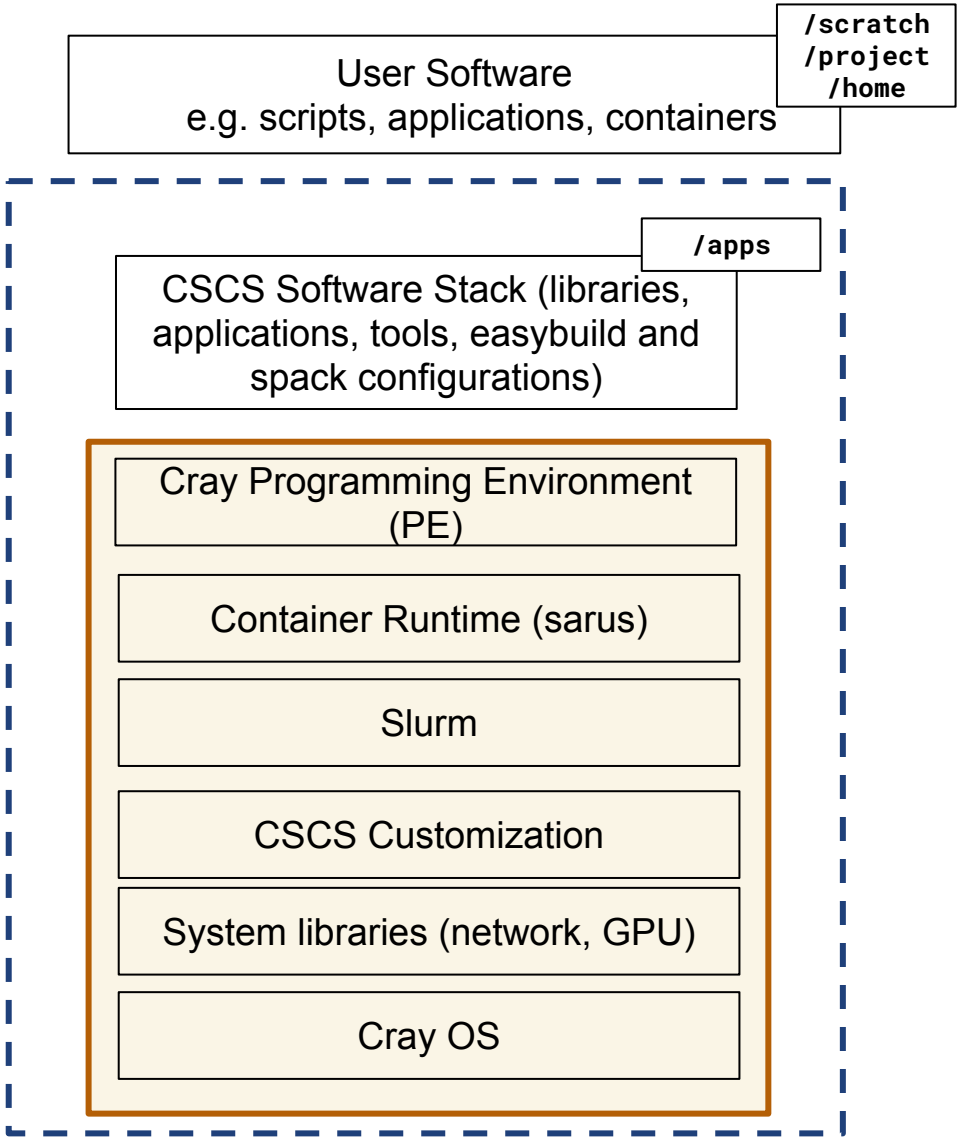
I need a stabler environment: older versions of the software tools disappear too quickly, which means I have to rebuild my stack every few months.

Please regularly update C++ and CUDA compilers

By providing an environment on CPE it is very difficult to meet all requirements

- Regular updates are required to fix bugs, maintain security and provide updated versions of tools.
- The latest versions of compilers can't be installed before they are packaged by HPE and tested by CSCS.
- It is impractical to maintain:
  - Full stacks on top of more than one CPE
  - More than 2-3 CPE on a system

The user environment on Daint: layer cake



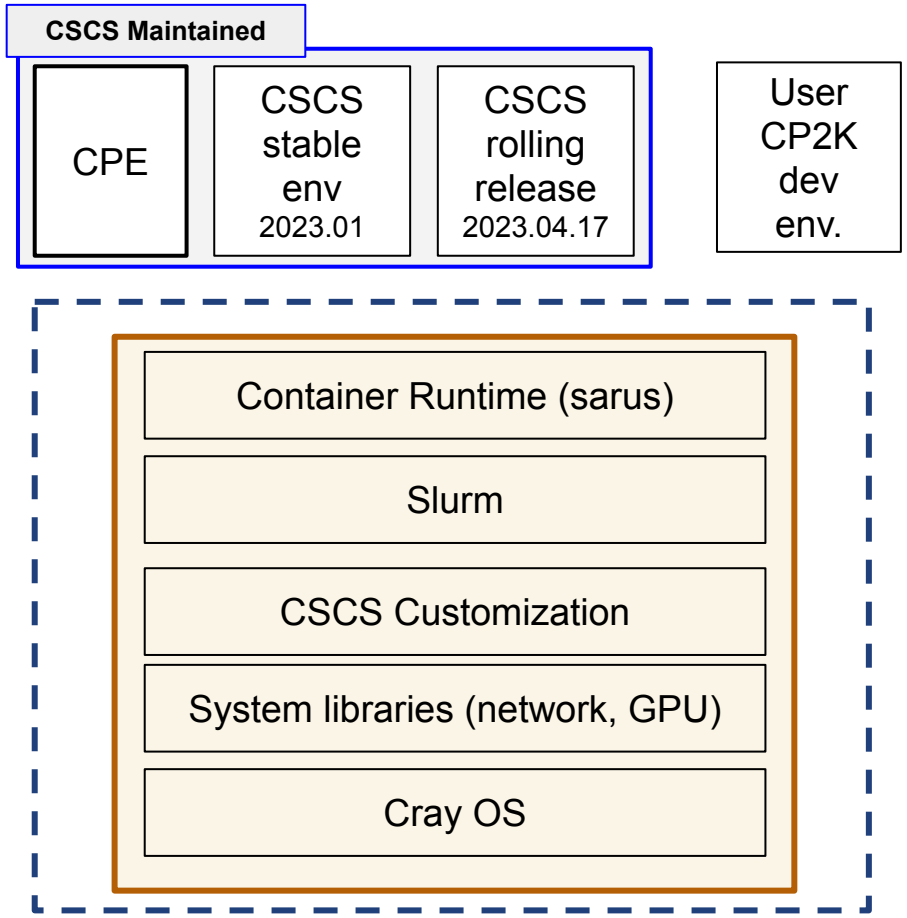
1. Essential services are installed on top of Cray OS
2. The Cray PE is installed as part of the underlying "node image"
3. CSCS provides a rich set of software products built with the Cray PE
4. Users build their software and workflows on top of that.

A change to one layer during an upgrade affects every layer above:

- A new Cray PE change often requires rebuilding the CSCS stack and user software

CSCS has developed tooling (ReFrame) and a very comprehensive test suite and CI/CD for maintaining the quality of this integration

The User Environment Initiative



Provide multiple user environments

- Login to a clean environment
  - Cray OS with slurm + container runtime + drivers
- Multiple environments are available:
  - The familiar CPE with `env-load craype-22.08`
  - CSCS-provided user environments
  - User-built user environments
- Each environment is contained in a single file
  - Shared in an artifactory or stored on a filesystem.

The environments are **independent**

The environments are built on top of the base-image - not the Cray PE.

User Environments in Practice

There are 3 components:

1. A standard approach for packaging environments
   - Not tied to any method for building or describing environments (e.g. Modules, Spack, Easybuild)
   - Simple to understand.
   - Simple to copy, store and version.
2. Tooling to describe and build environments
   - Provide a flat description of an environment as code
   - Efficiently build a packaged environment from the description
   - **Reproducible**
3. Tooling to load a user environment
   - Simple: one command to load an environment
   - Can be integrated with SLURM
   - Ease of use of tools like debuggers and profilers

Packaging and Loading an Environment

Environments are stored in a single compressed file that contains a directory structure.

Starting an environment: squashfs-run $SCRATCH/gromacs-cuda.squashfs bash

1. Mount: the directory tree in a fixed /user-environment path
2. Execute: a built-in prologue script that configures the environment
3. Launch: a shell in the new environment

Some key properties

- Environments are simply a compressed directory tree
  - Can be created with Spack, Easybuild, Conda or by hand.
- Per-process: other users on the system can mount their own environments without affecting one-another.
- Much lower latency for configure-then-make development workflows than HPC file systems.

Creating an environment with Spack

CSCS is developing tooling to build bespoke programming environments with compilers, MPI, and libraries defined in a git repository

- A matrix of compiler versions and libraries (e.g. MPI, netcdf, fftw) are described in YAML files.

Fast builds: a full stack with multiple gcc and clang versions, NVIDIA HPC SDK, libraries, MPI, and tools in less than an hour.

Environments can be versioned:

- rebuilt when the underlying node image is updated.

CSCS and users can configure their own environments

- CSCS can offer stable and cutting edge environments in our service catalogue

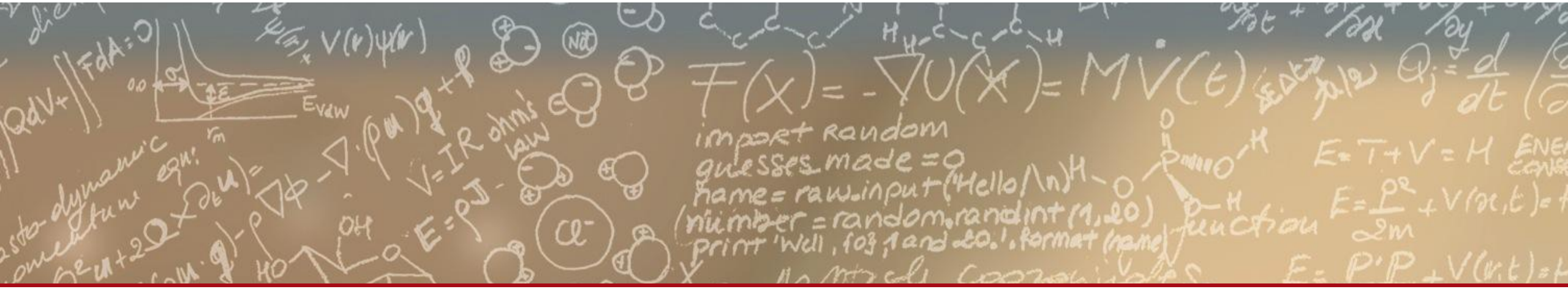
github.com/eth-cscs/stackinator

Status

CSCS is busy working on making user environments easy to use and intuitive user experience.

Features include:

- Slurm integration
- CI/CD for automatic build and push to the CSCS artifactory when:
  - The target node description is updated
  - The software description is updated
- Cray-mpich bundled outside of
- Environments for Python and Julia.
- Simplified recipes for whole programming environments
  - Can be versioned in Git and delivered through CI/CD
- Currently being used by one tennant.


import random
guesses_made = 0
name = raw_input('Hello! What is your name?\n')
number = random.randint(1, 20)
print 'Well, ' + name + ', I am thinking of a number between 1 and 20.'

Thank you!