



JCAHPC Open Hackathon

Day 0

2025/1/31



Agenda

- NVIDIA GH200 Hardware Overview

- NVIDIA GH200 Programming Model Overview

- NVIDIA Developer Tools

- Hands-on (Nsight Systems)



NVIDIA GH200 Hardware Overview

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

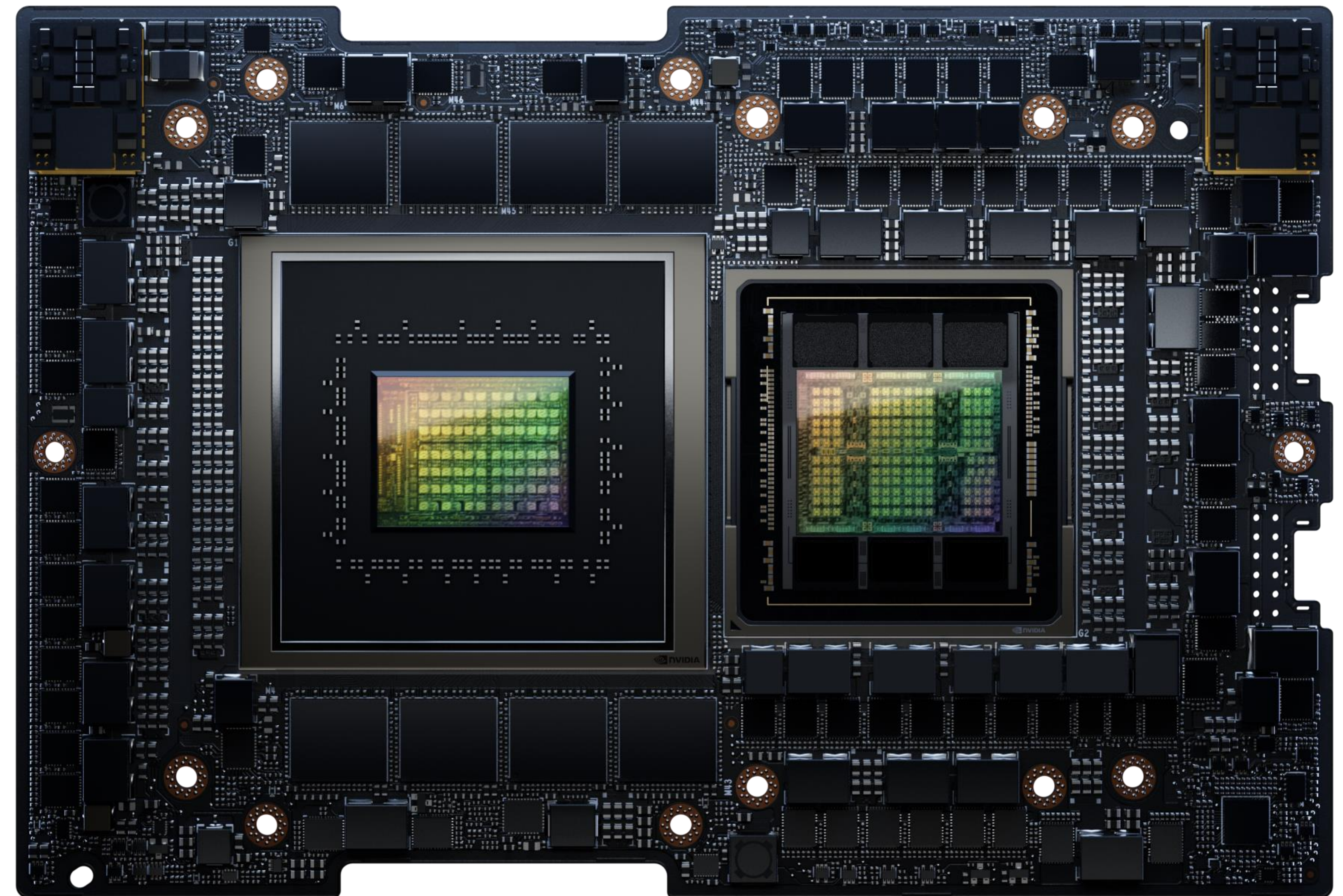
900GB/s bi-directional bandwidth CPU to GPU
7X faster than PCIe Gen 5

~600GB of Fast Access Memory

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

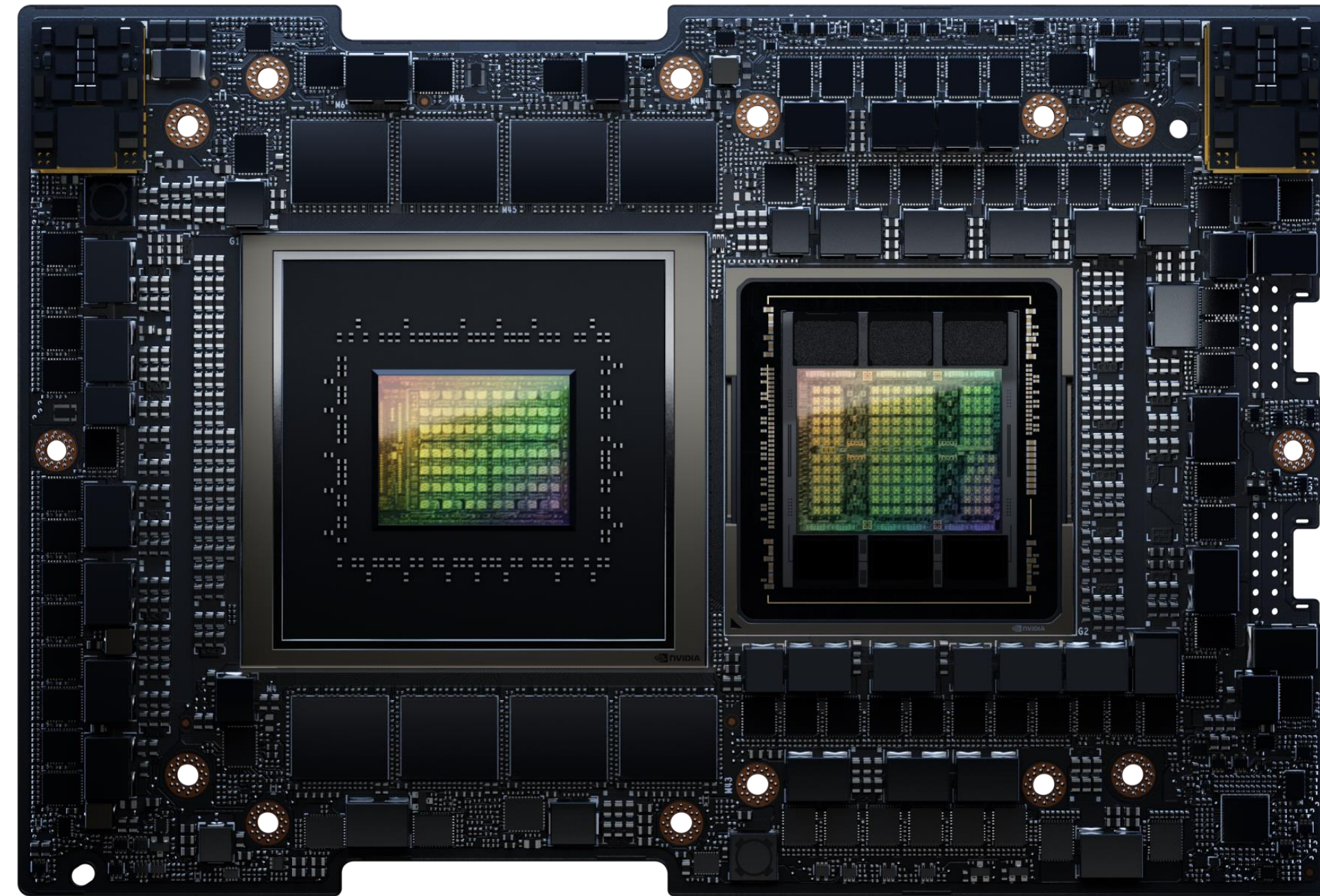
Full NVIDIA Compute Stack

AI, Omniverse



NVIDIA GH200 Grace Hopper Superchip

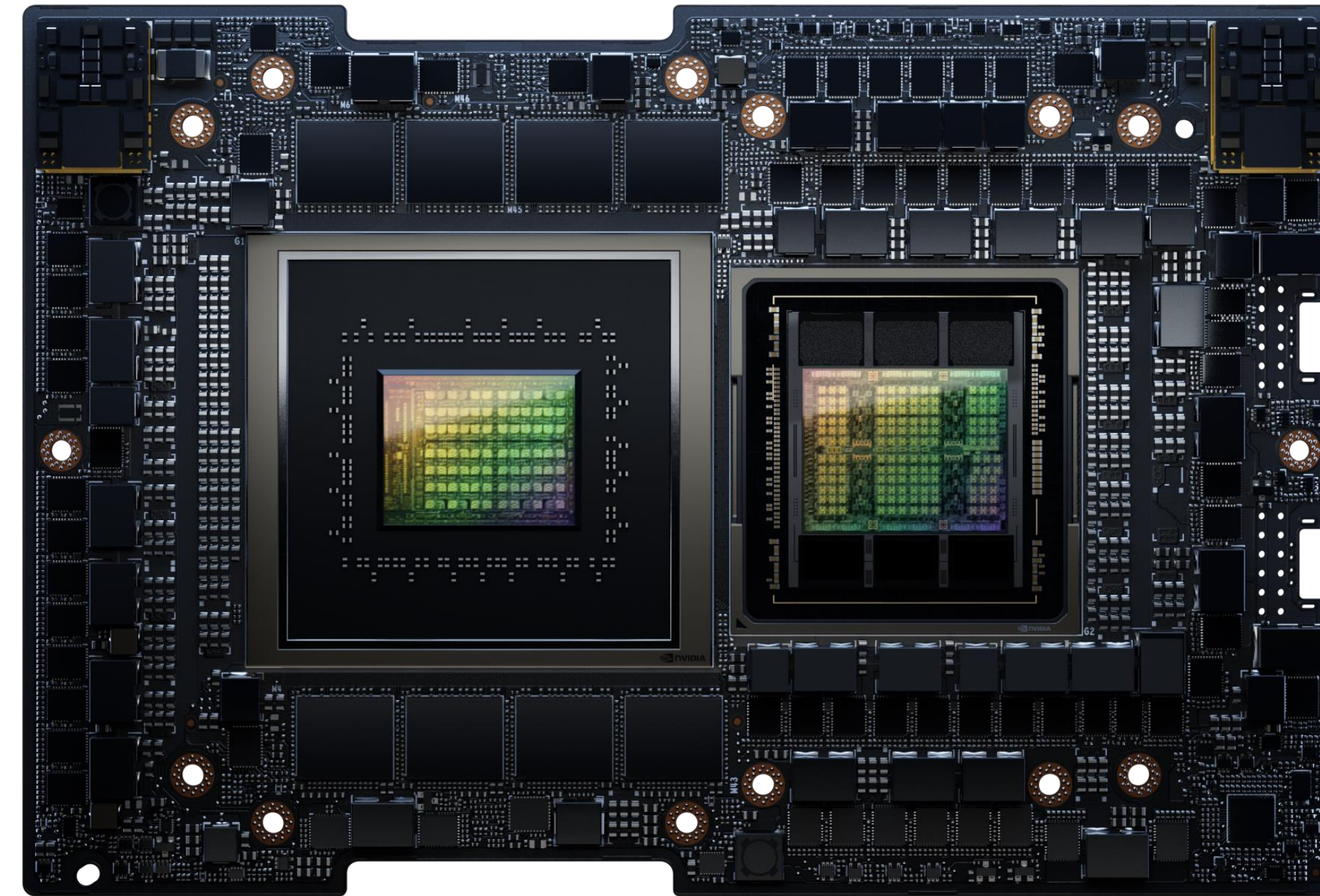
Processor For The Era of Accelerated Computing And Generative AI



72 Core Grace CPU | 4 PFLOPS Hopper GPU
96 GB HBM3 | 4 TB/s | 900 GB/s NVLink-C2C

- 7X bandwidth to GPU vs PCIe Gen 5
- Combined 576 GB of fast memory
- 1.2x capacity and bandwidth vs H100
- Full NVIDIA Compute Stack

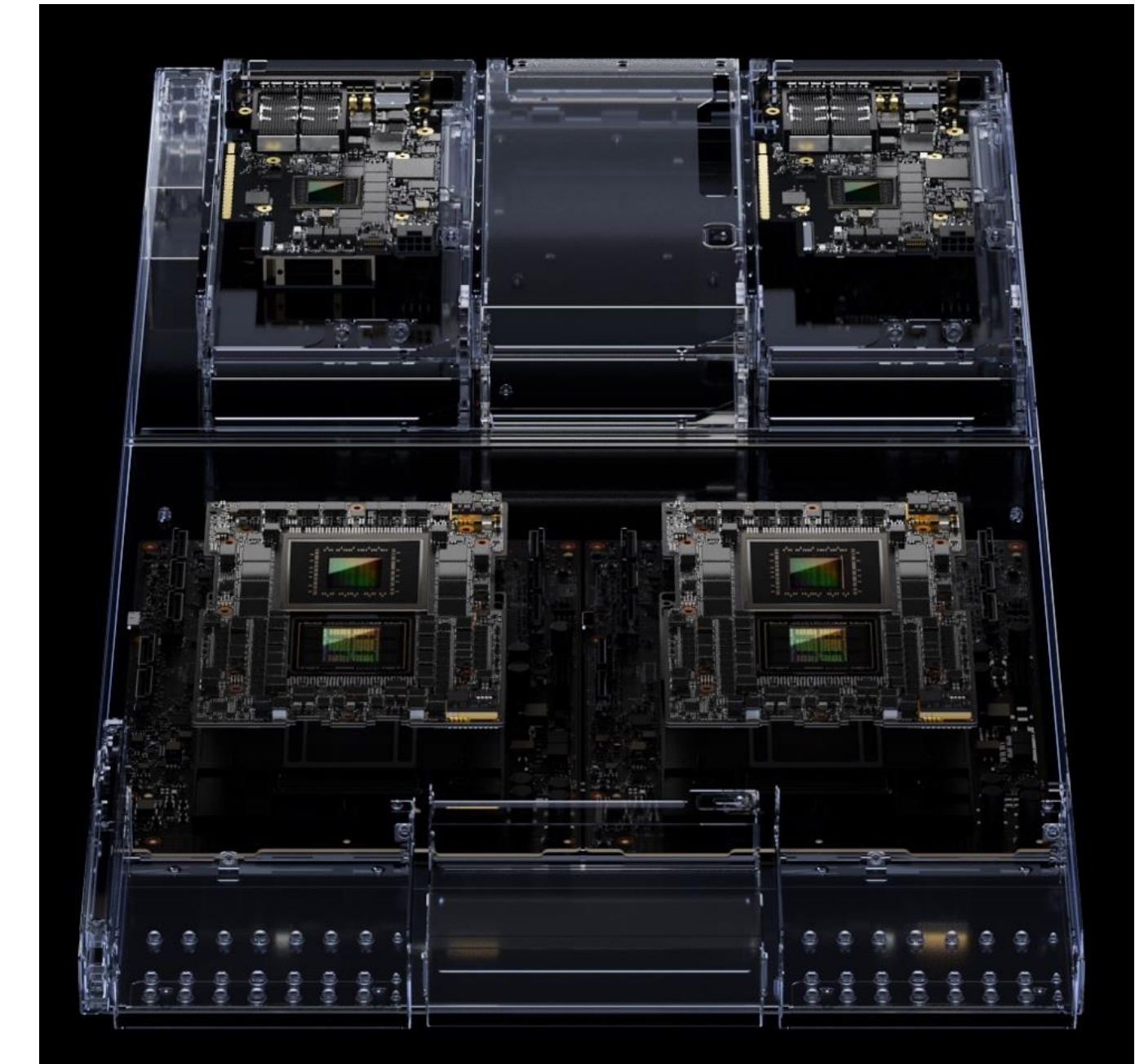
GH200 with HBM3



72 Core Grace CPU | 4 PFLOPS Hopper GPU
144 GB HBM3e | 5 TB/s | 900 GB/s NVLink-C2C

- World's first HBM3e GPU
- Combined 624 GB of fast memory
- 1.7x capacity and 1.5x bandwidth vs H100
- Full NVIDIA Compute Stack

GH200 with HBM3e



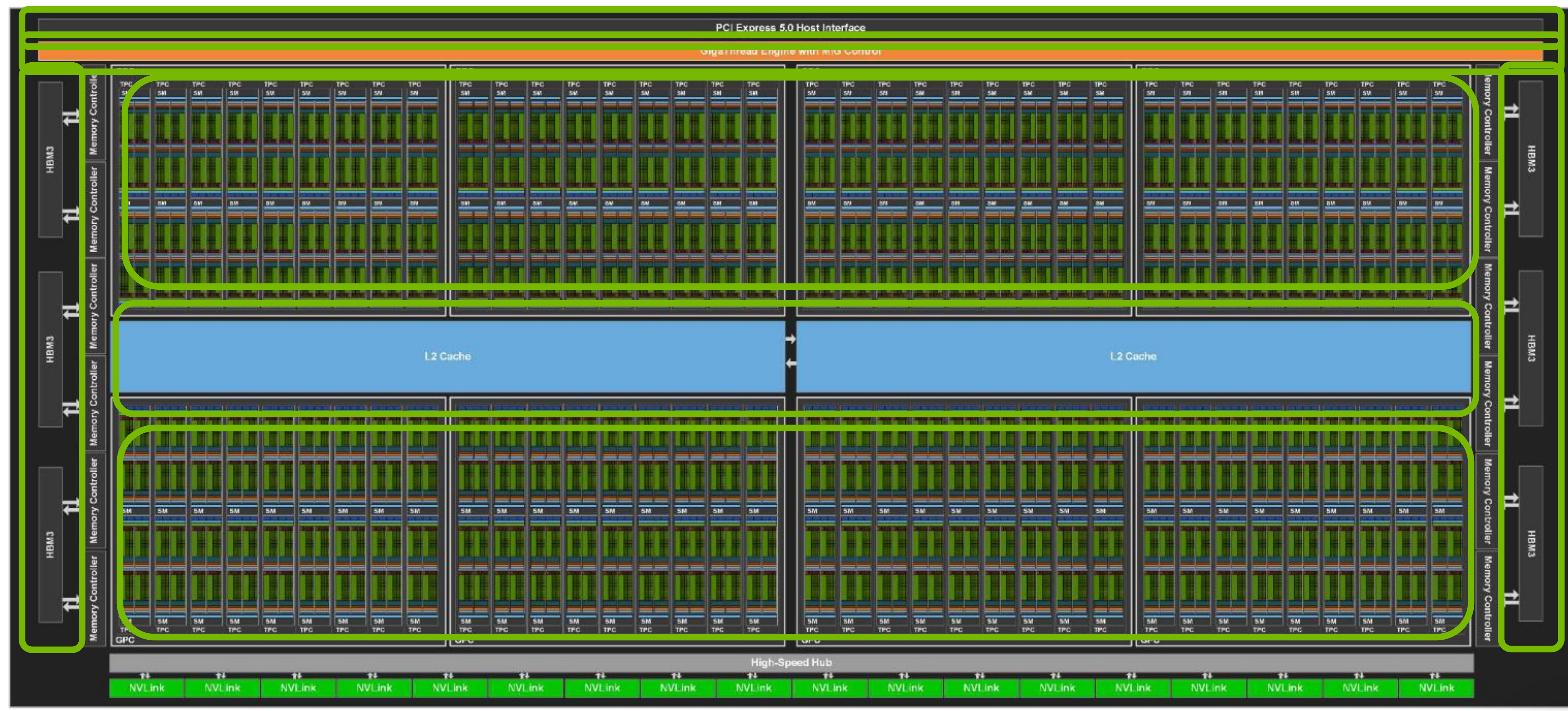
144 Core Grace CPU | 8 PFLOPS Hopper GPU
288 GB HBM3e | 10 TB/s | 900 GB/s NVLink-C2C

- Simple to deploy MGX-compatible design
- Combined 1.2 TB fast memory
- 3.5x capacity and 3x bandwidth vs H100
- Full NVIDIA Compute Stack

GH200 NVL2

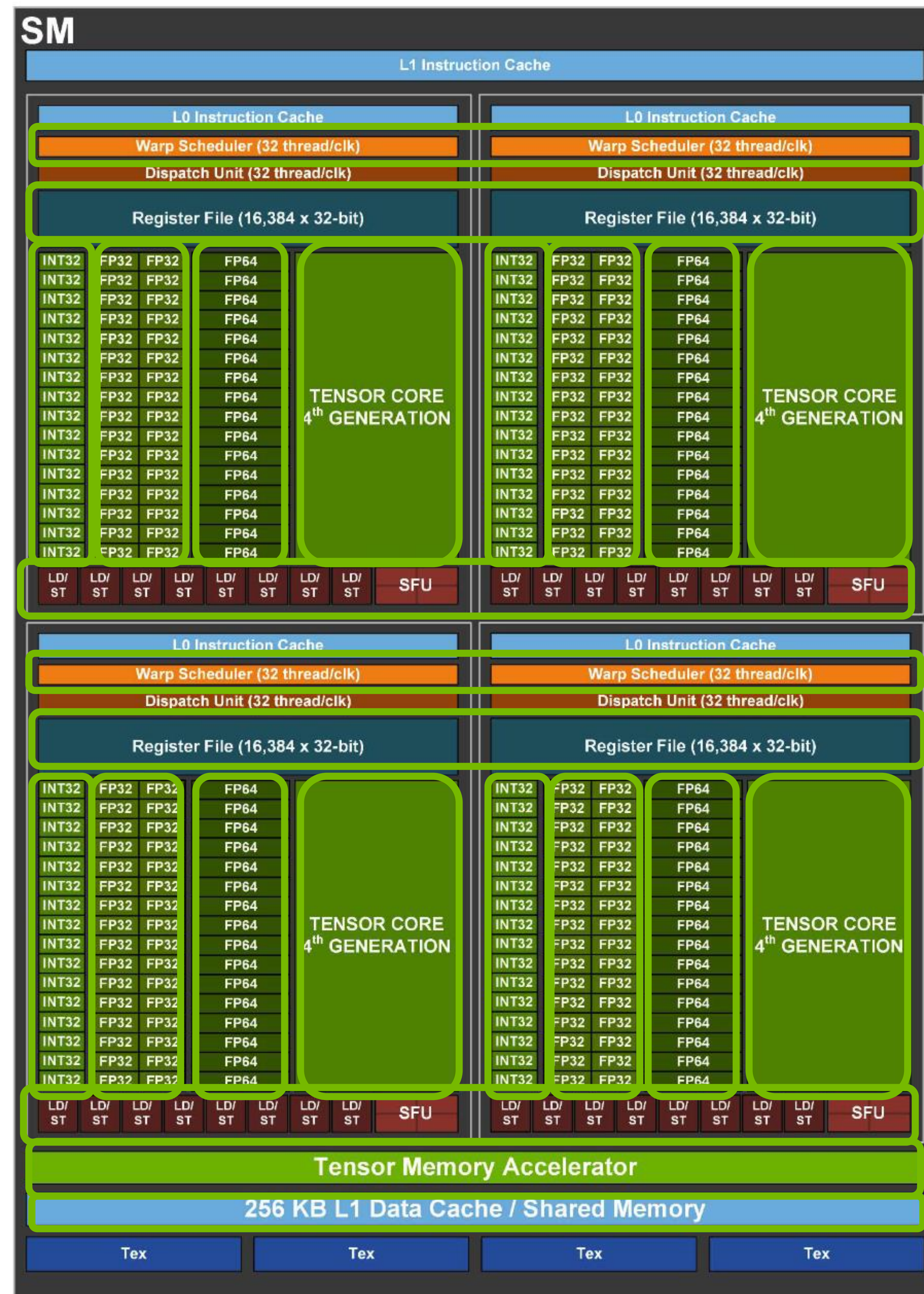
Hopper Architecture

NVIDIA H100



- PCI I/F
 - ホスト接続インタフェース
- Giga Thread Engine
 - SM に処理を割り振るスケジューラ
- DRAM I/F
 - 全 SM、PCI I/F からアクセス可能なメモリ (デバイスメモリ, フレームバッファ)
- L2 cache (50 MB)
 - 全 SM からアクセス可能な R/W キャッシュ
- SM (Streaming Multiprocessor)
 - 「並列」プロセッサ、H100 : 132

SM (Streaming Multi-processor)

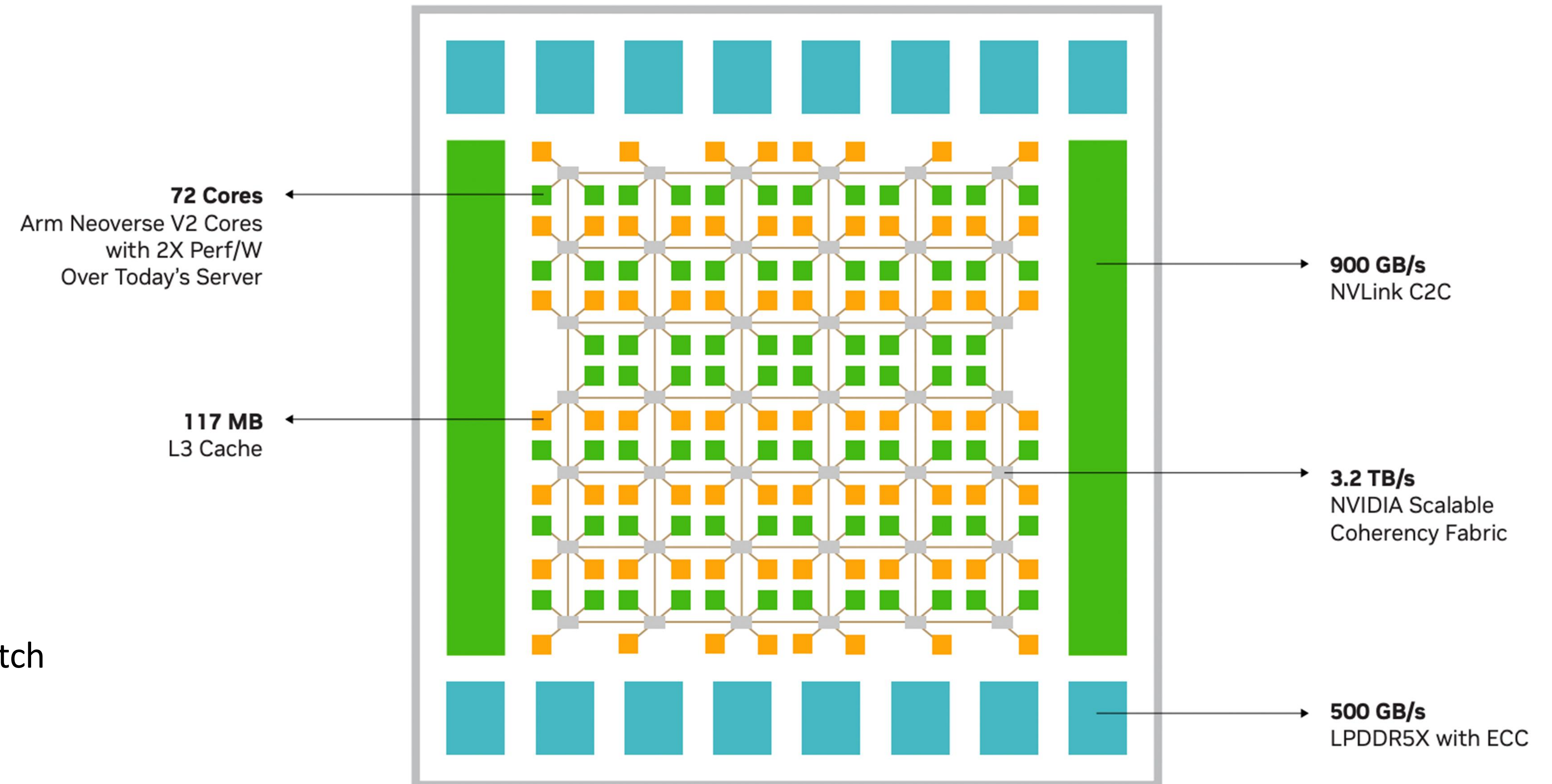


- ワーブスケジューラ
- 演算ユニット
 - INT32: 64 個
 - FP32: 128 個
 - FP64: 64 個
 - TensorCore: 4 個
- Other units
 - LD/ST, SFU, etc
- レジスタ (32 bit): 64K 個
- 共有メモリ/L1 キャッシュ: 256 KB
- Tensor Memory Accelerator

GRACE IS A COMPUTE & DATA MOVEMENT ARCHITECTURE

NVIDIA Scalable Coherency Fabric (SCF) and distributed cache design

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

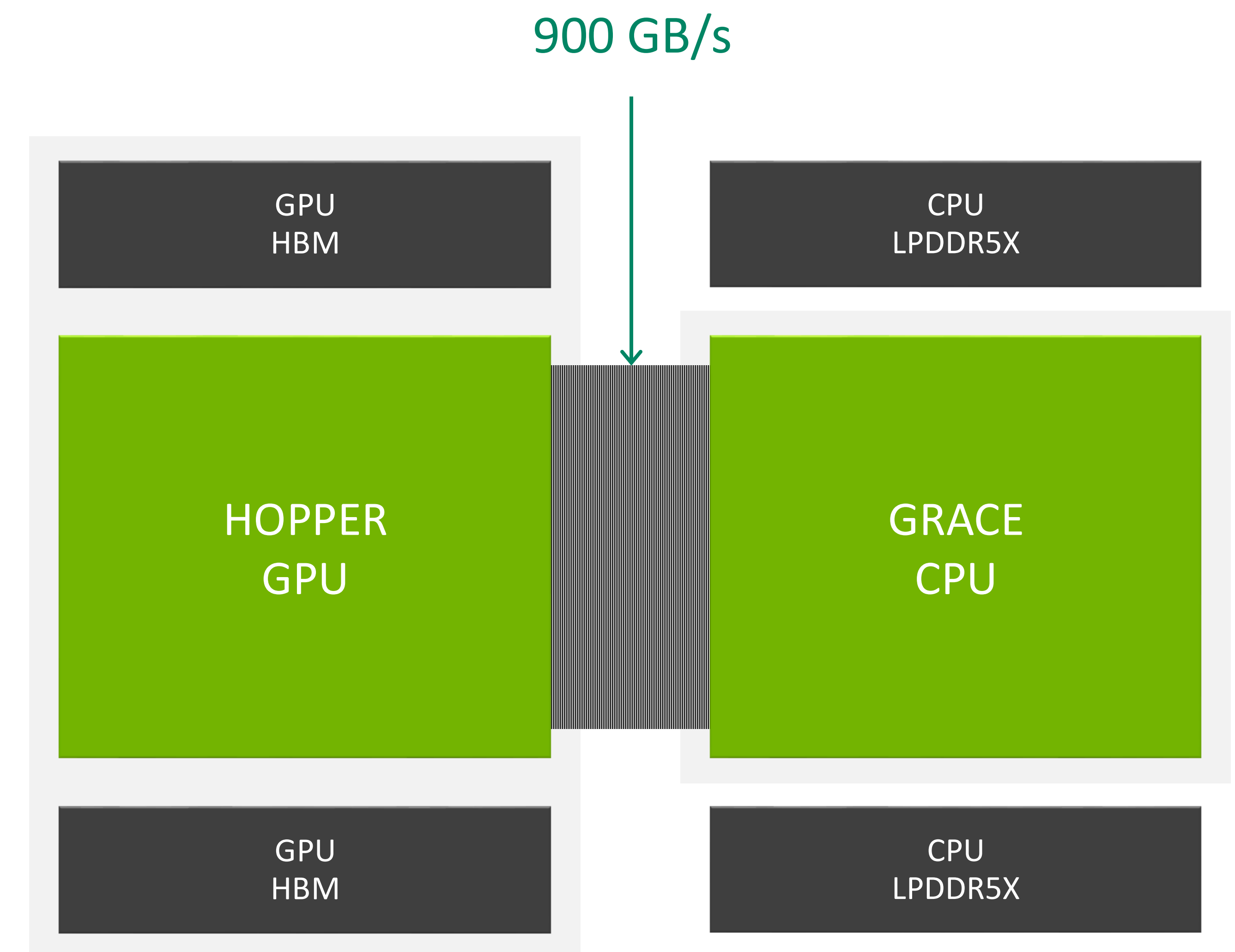


Example possible fabric topology for illustrative purposes

NVLINK-C2C

High Speed Chip to Chip Interconnect

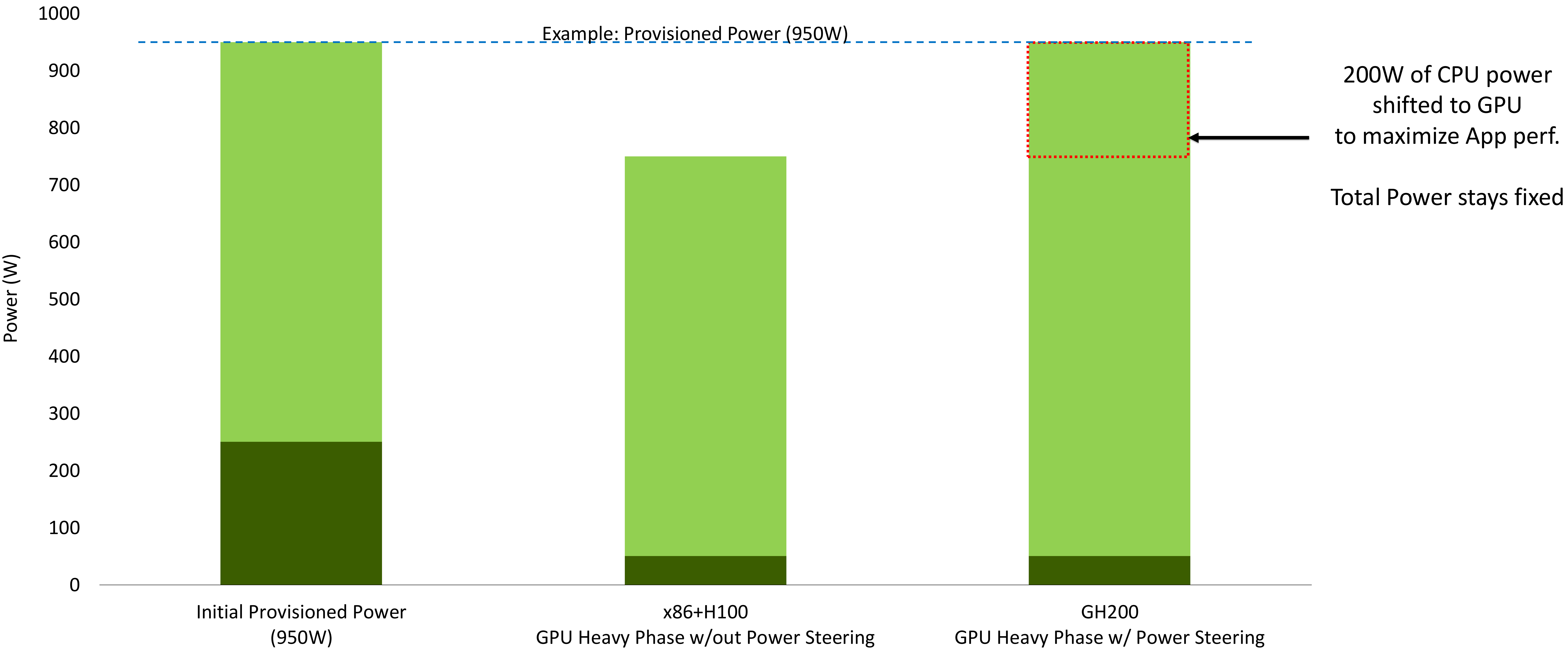
- Used to create the Grace Hopper, and Grace Superchips
 - Native atomics, including standard C++ atomic support
 - **Enables coherency**
- Up to 900 GB/s of raw bidirectional BW
 - Same BW as GPU to GPU NVLINK on Hopper
- Low power interface - 1.3 pJ/bit
 - **More than 5x more power efficient than PCIe**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



Optimizing Performance Through Power Steering

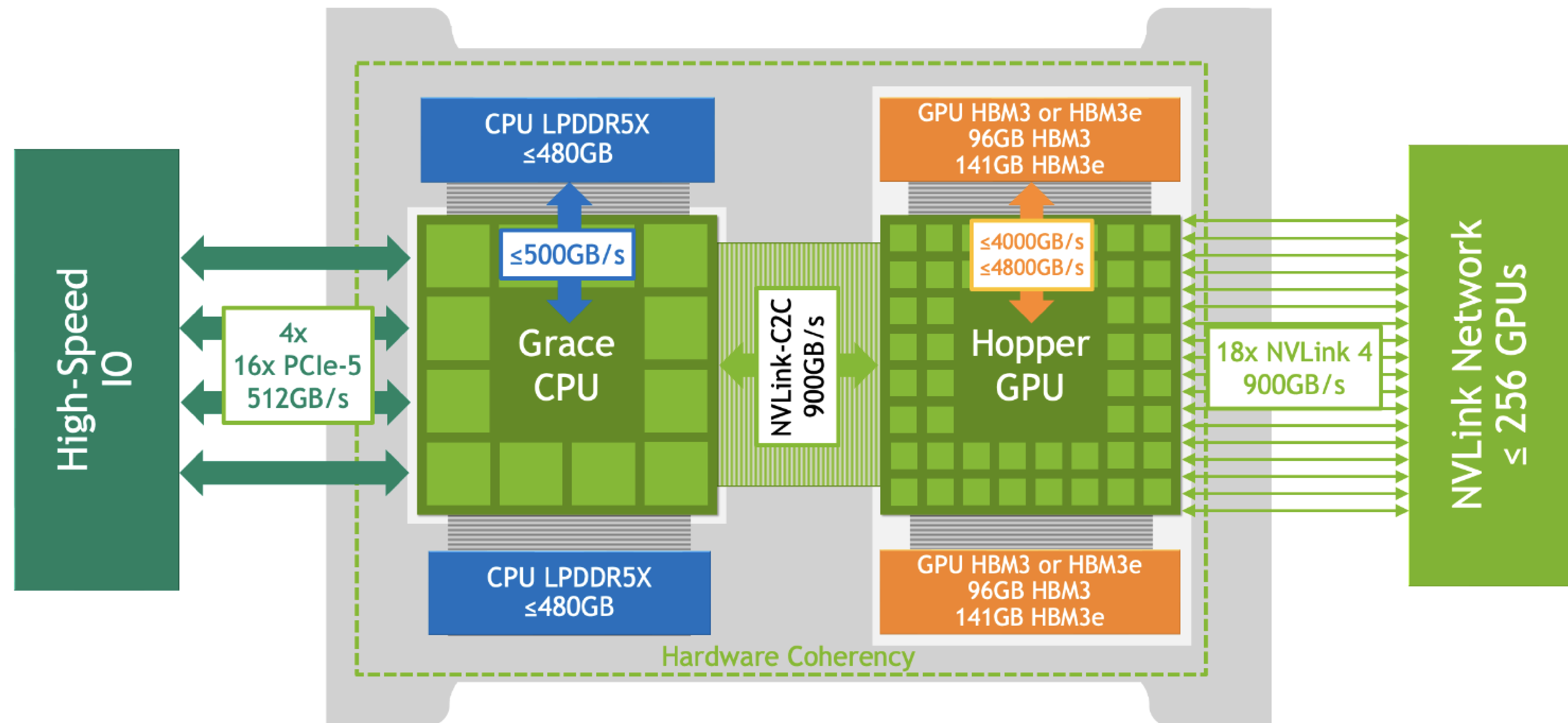
Getting the Most Out of provisioned power

CPU GPU



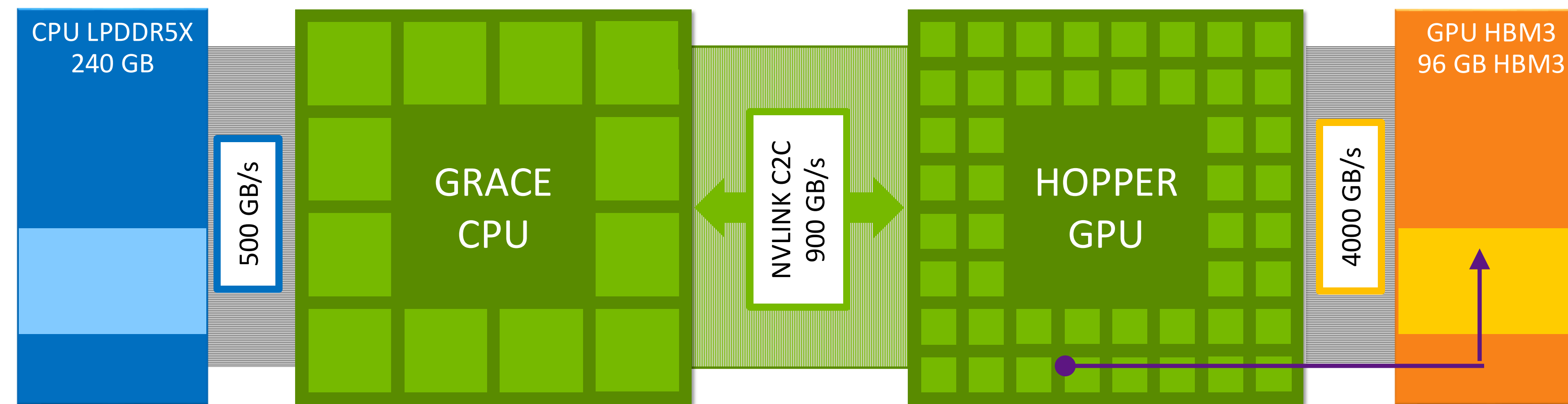
NVIDIA GH200

GPU can access CPU memory at CPU memory speeds

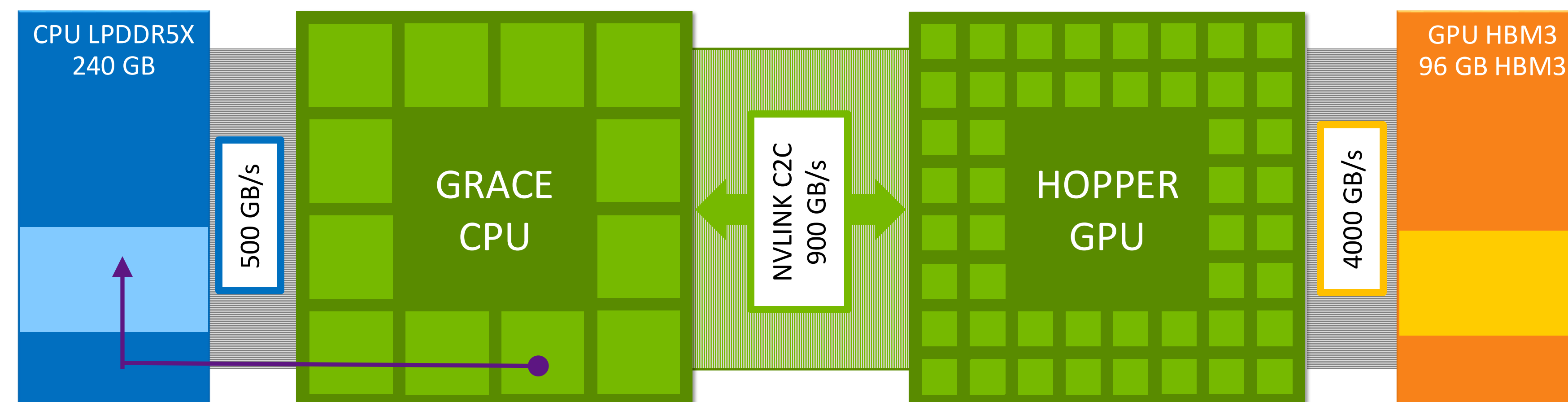
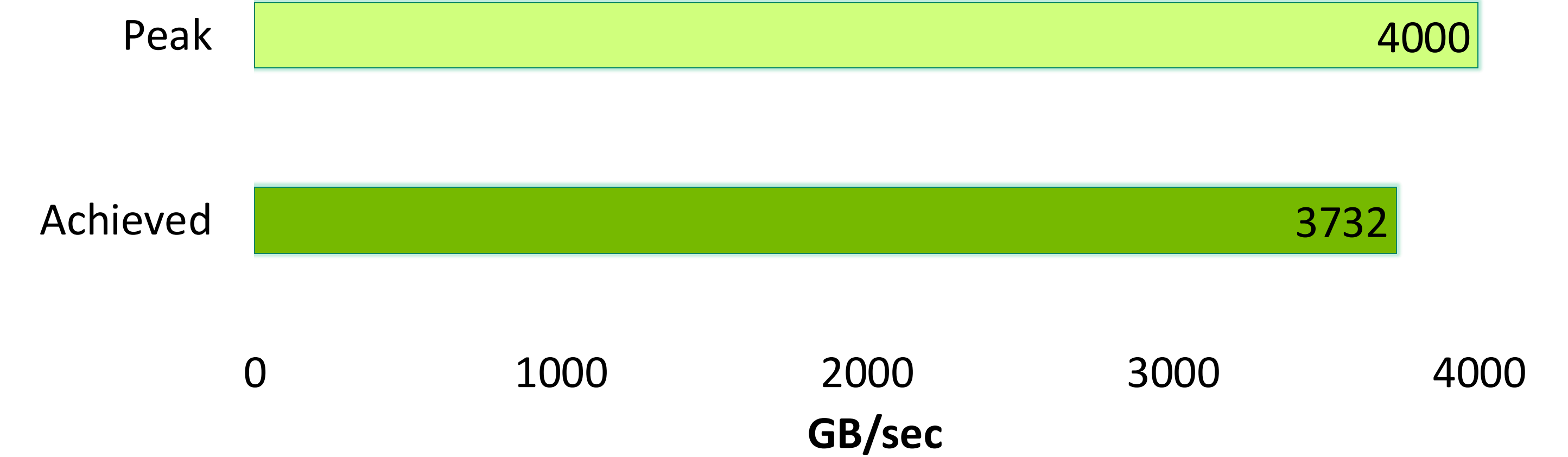


HW/SW memory view on GH200

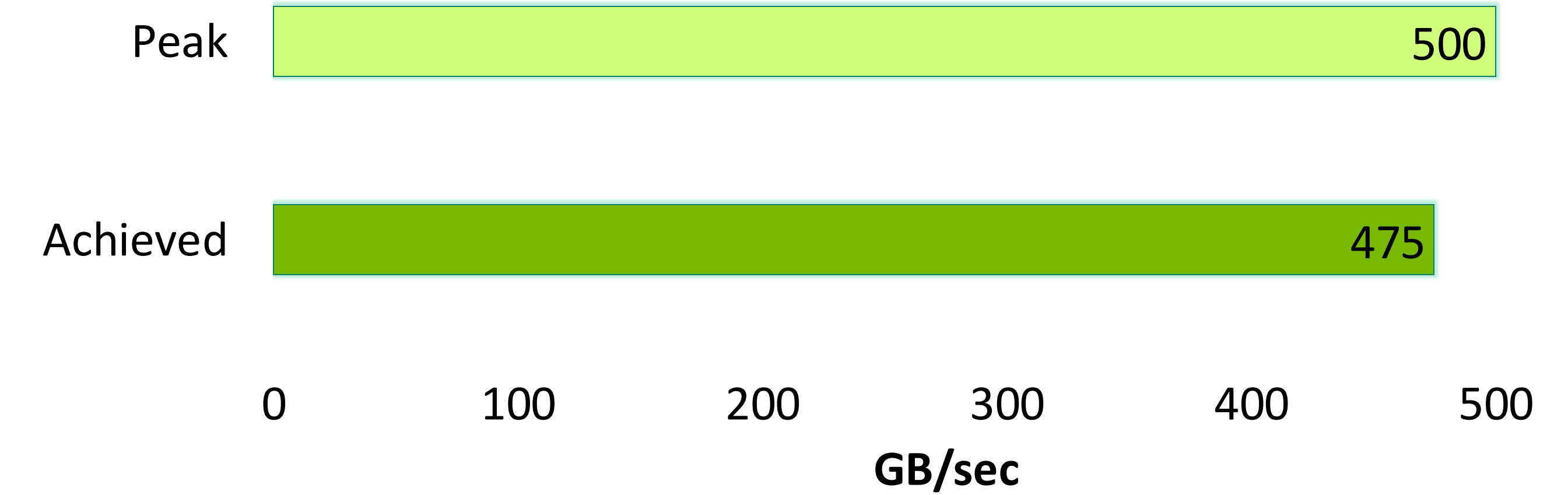
Local memory pool accesses at link speeds



Bandwidth for GPU stream triad kernel accessing GPU memory

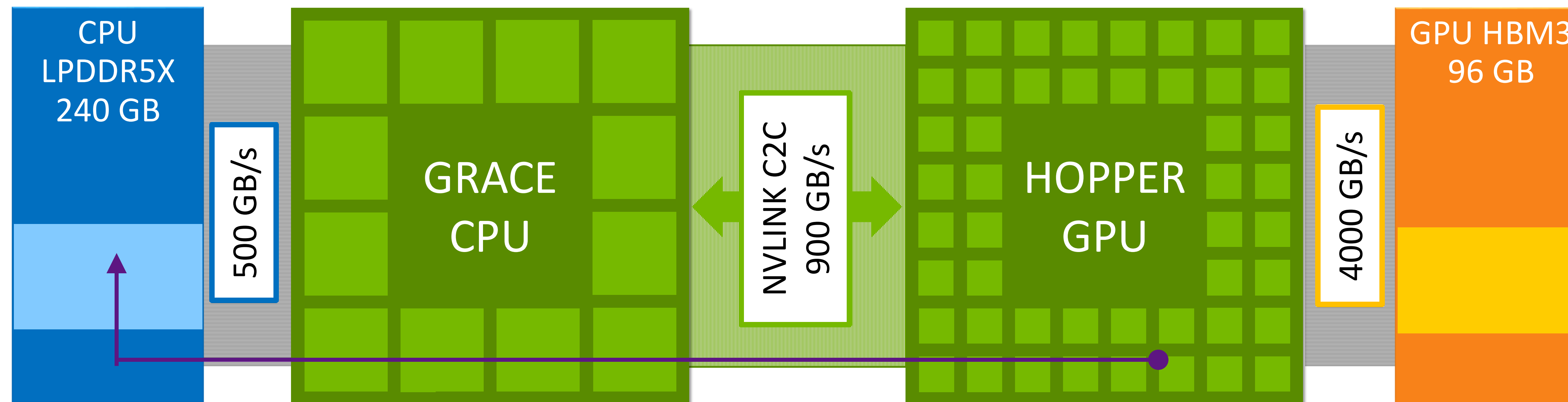


Bandwidth for CPU stream accessing CPU memory

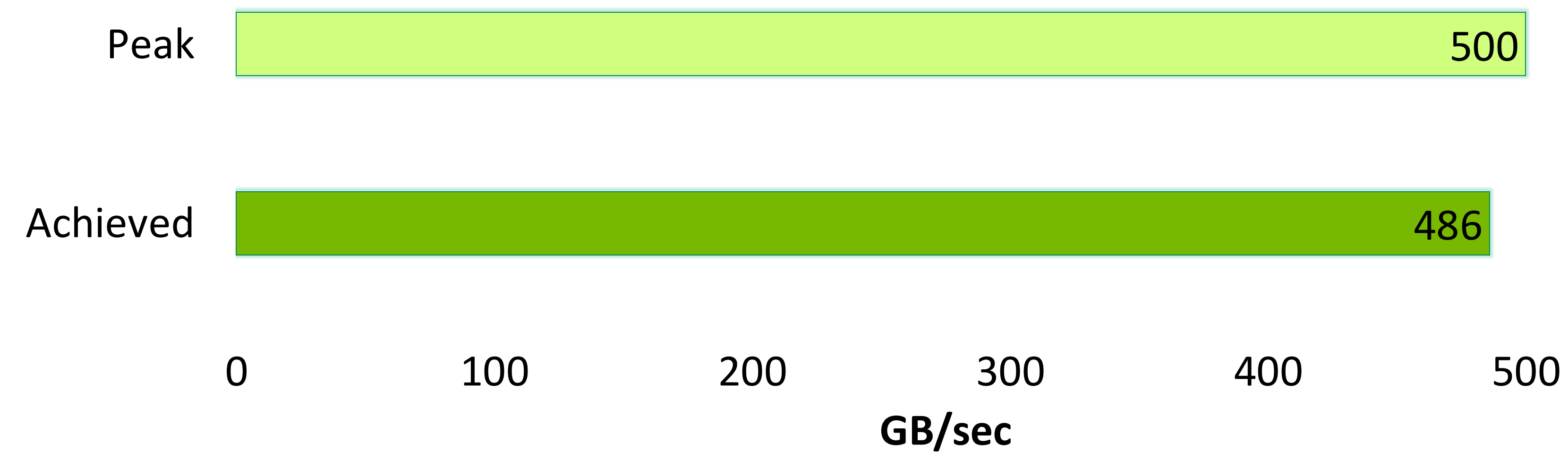


HW/SW memory view on GH200

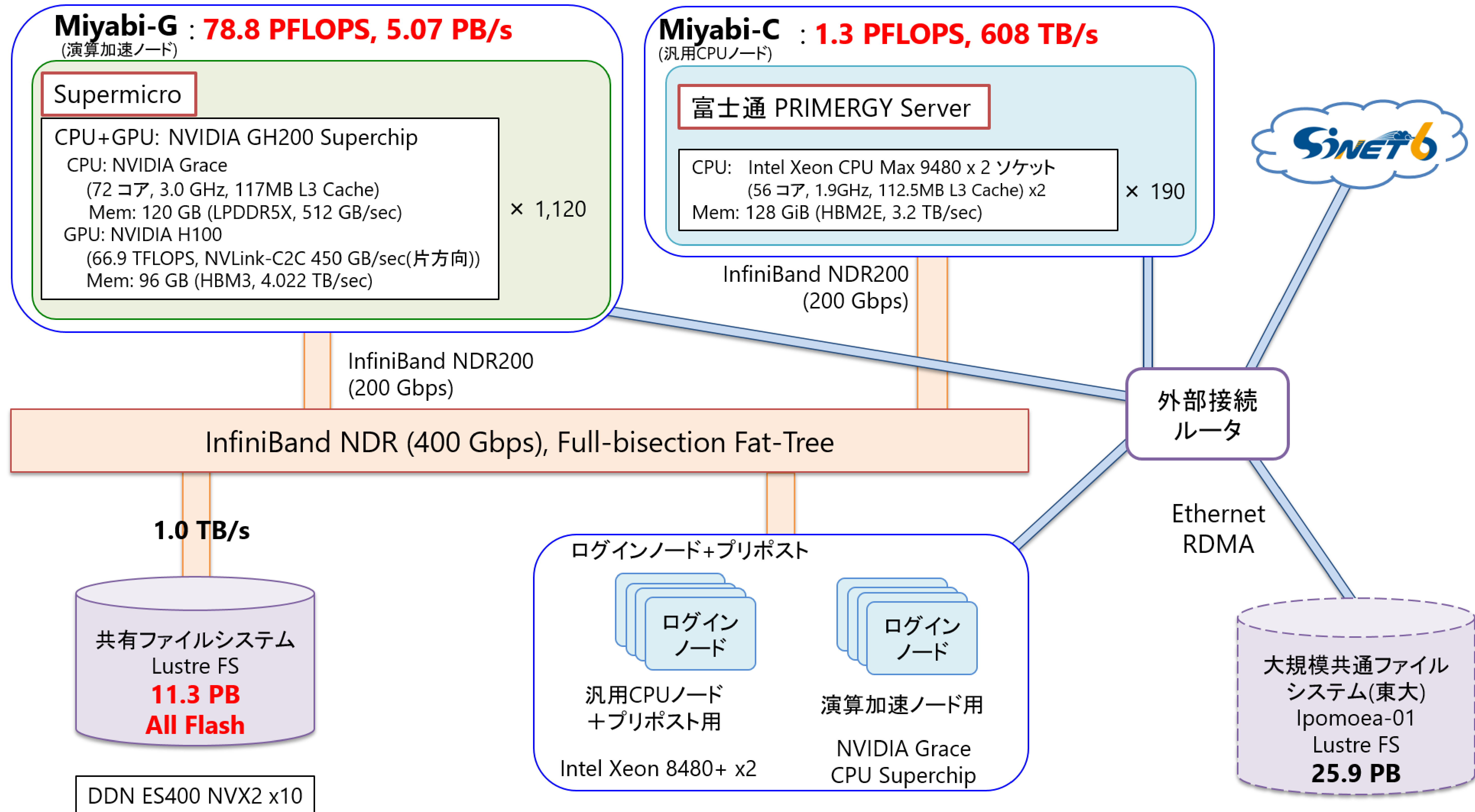
C2C enabling remote accesses



Bandwidth for GPU stream kernel accessing CPU memory



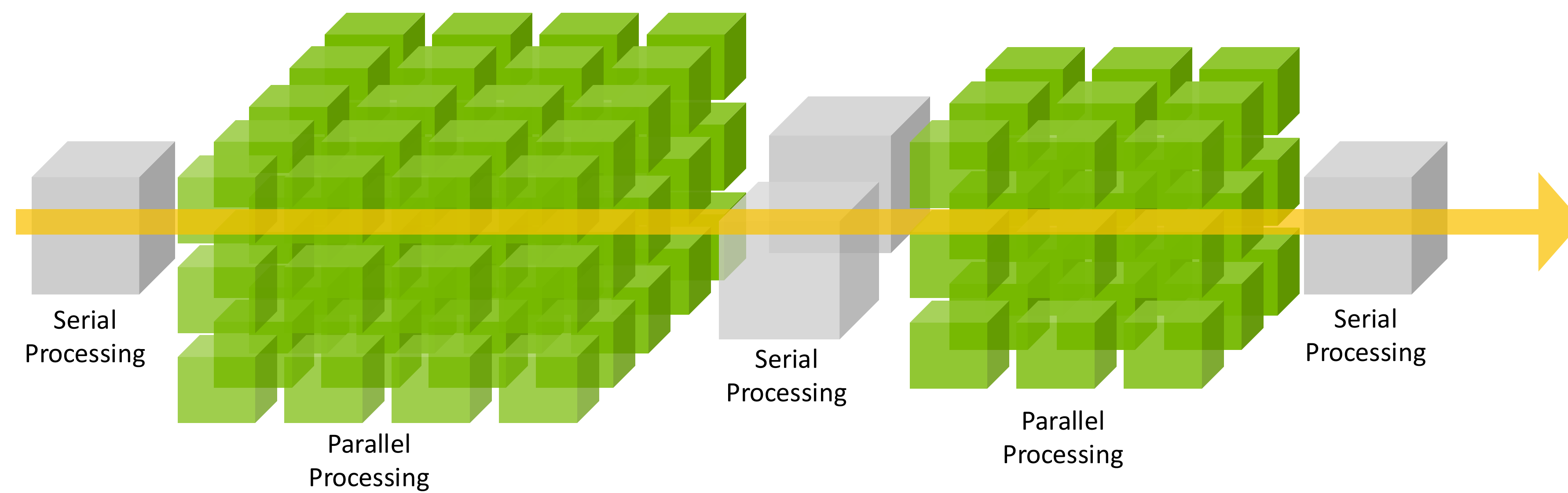
Miyabi System



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

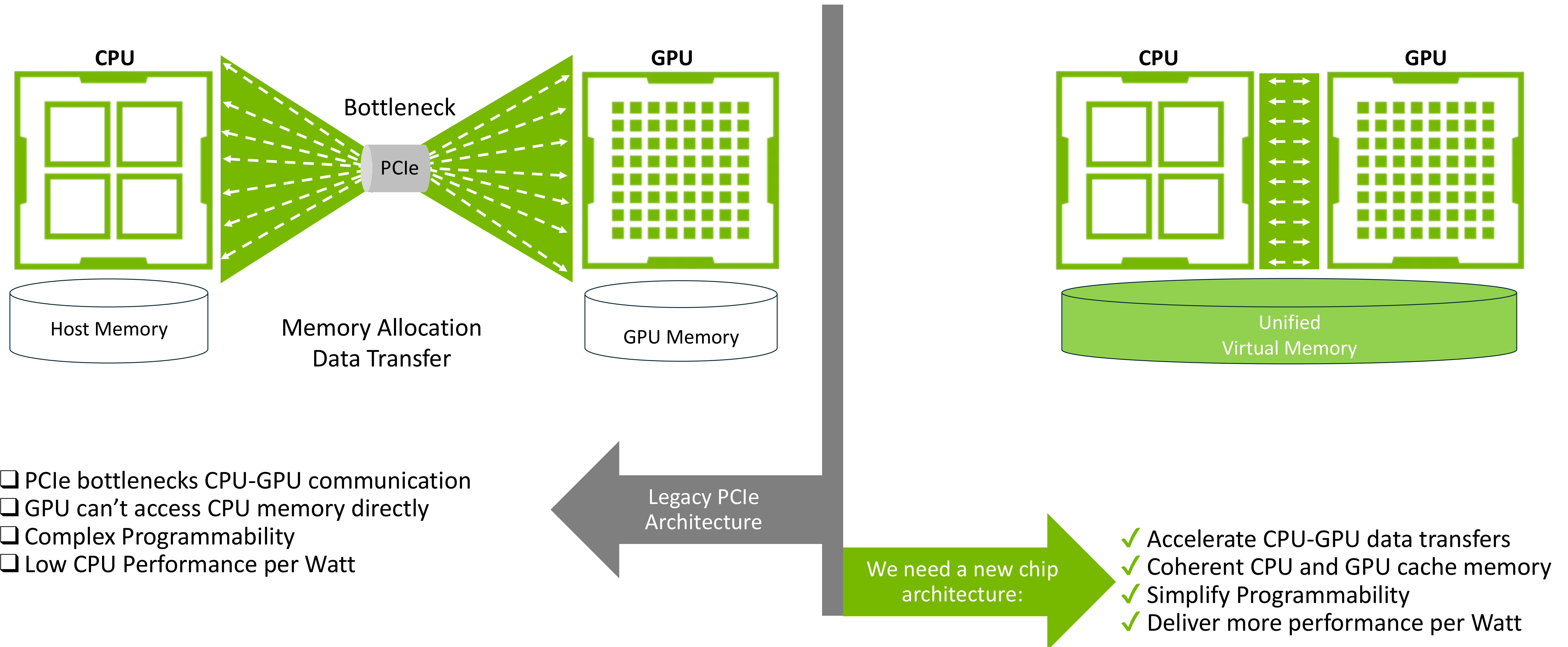


Excels at Parallel Processing

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

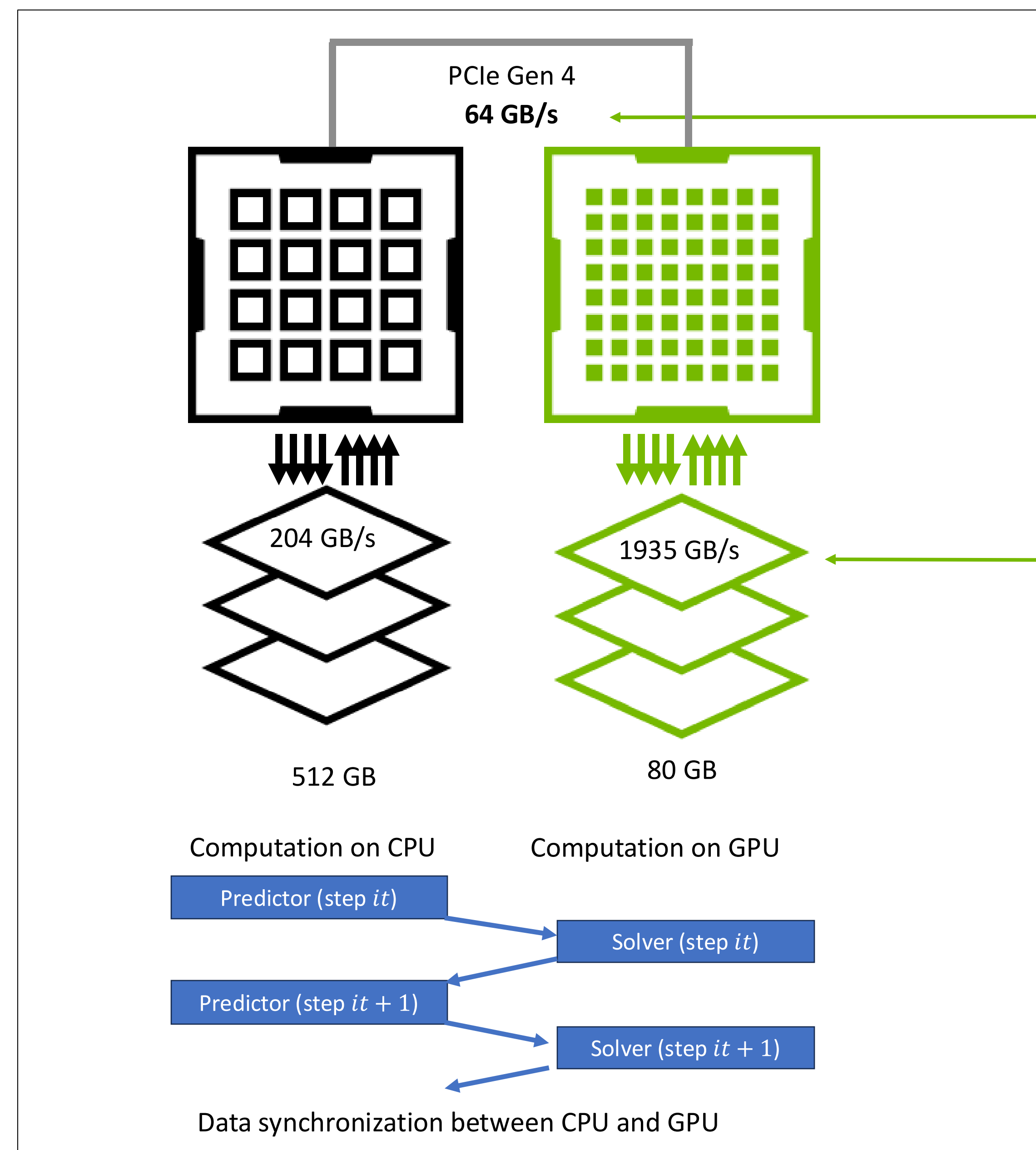
Challenges with Traditional Accelerated Systems

Bridging the GAP: Unifying CPU and GPU Memory



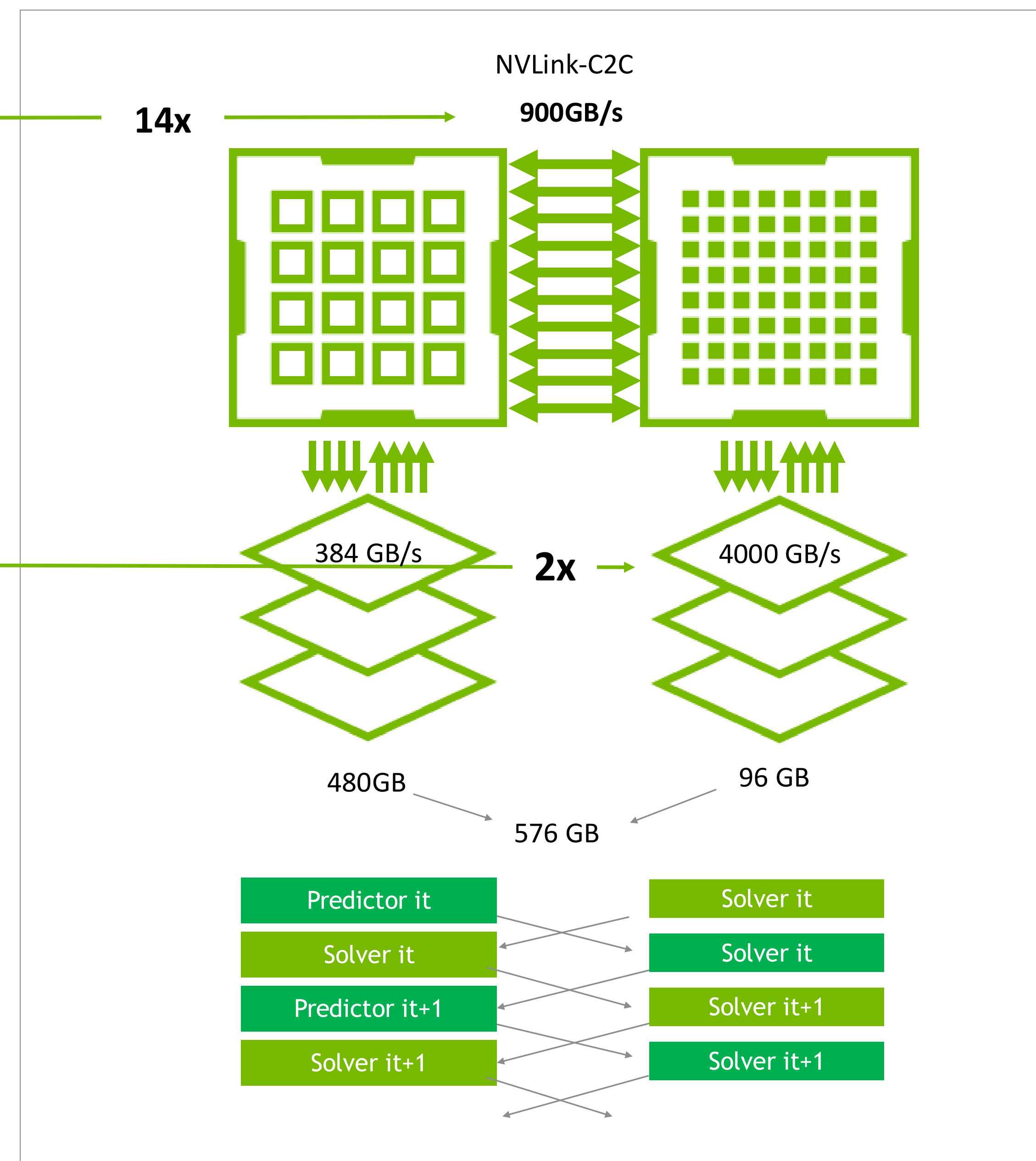
6x Faster Equation-based Modeling with GH200

University of Tokyo used Hybrid Computing to interleave CPU and GPU computing



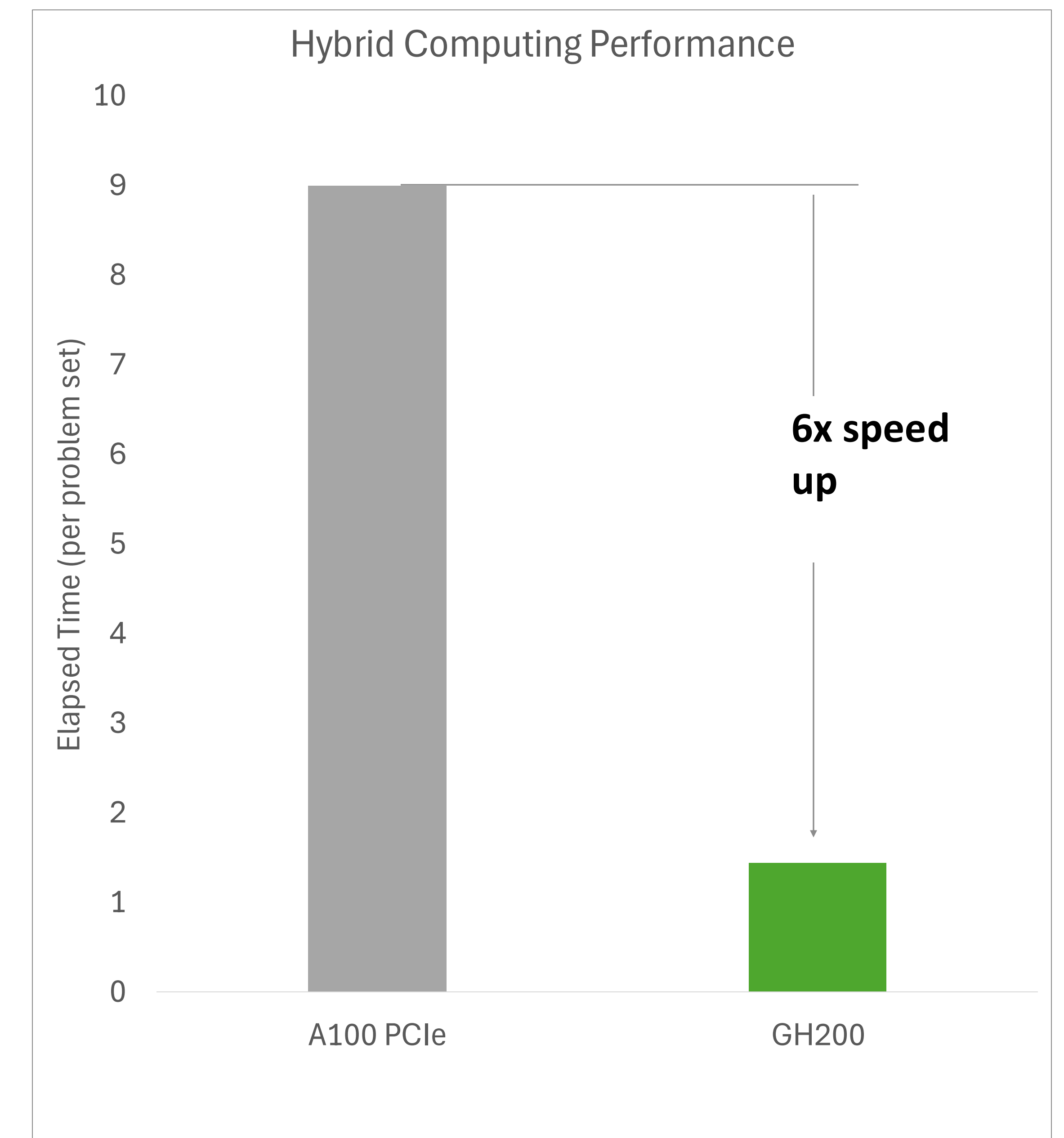
X86 + A100

I/O is bottlenecked by PCIe Gen 4



GH200

Concurrent use of CPU and GPU to solve 2 sets of problems.



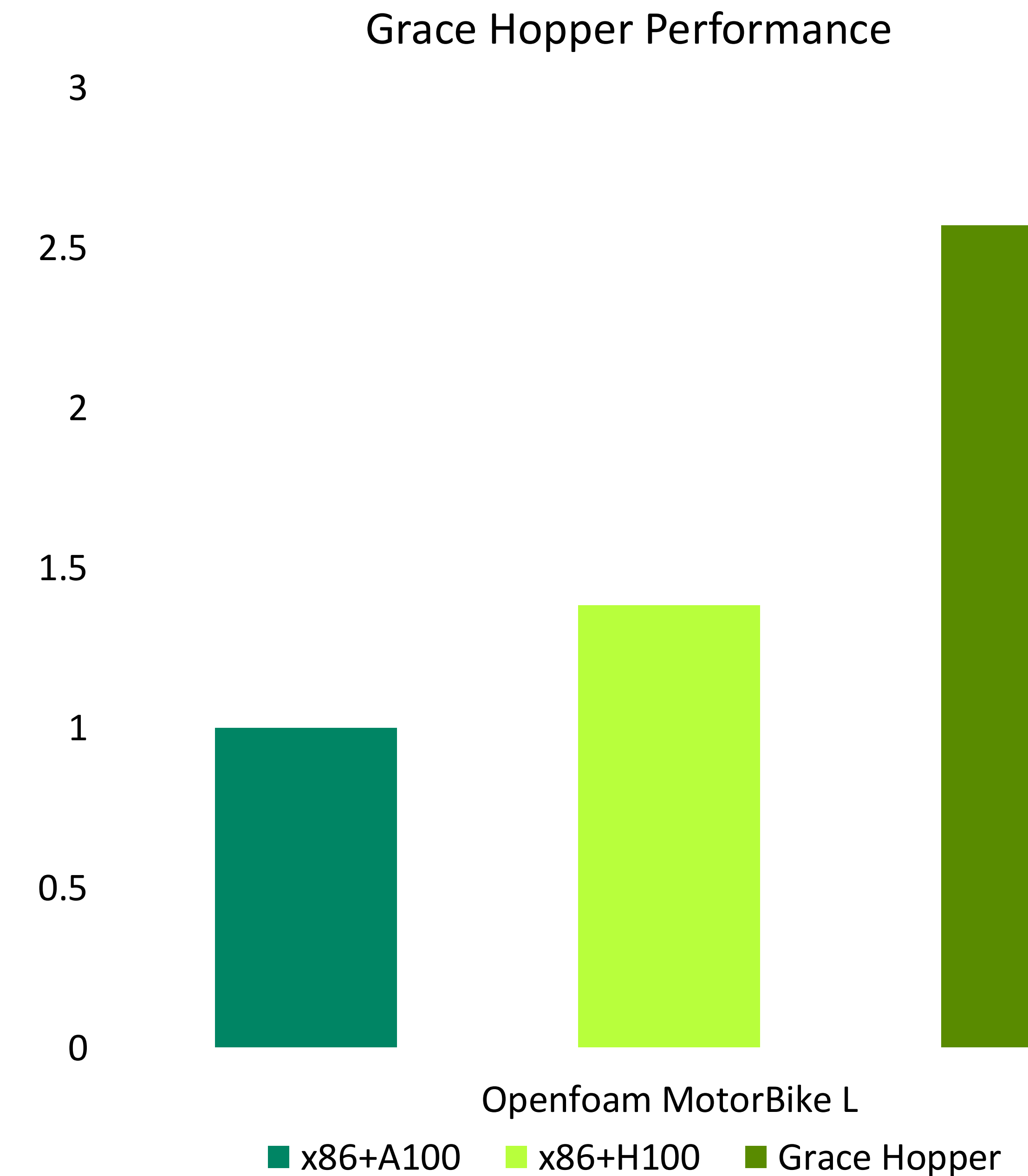
PDE based time evolution problems

Numerous applications in physics, engineering and other fields.

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

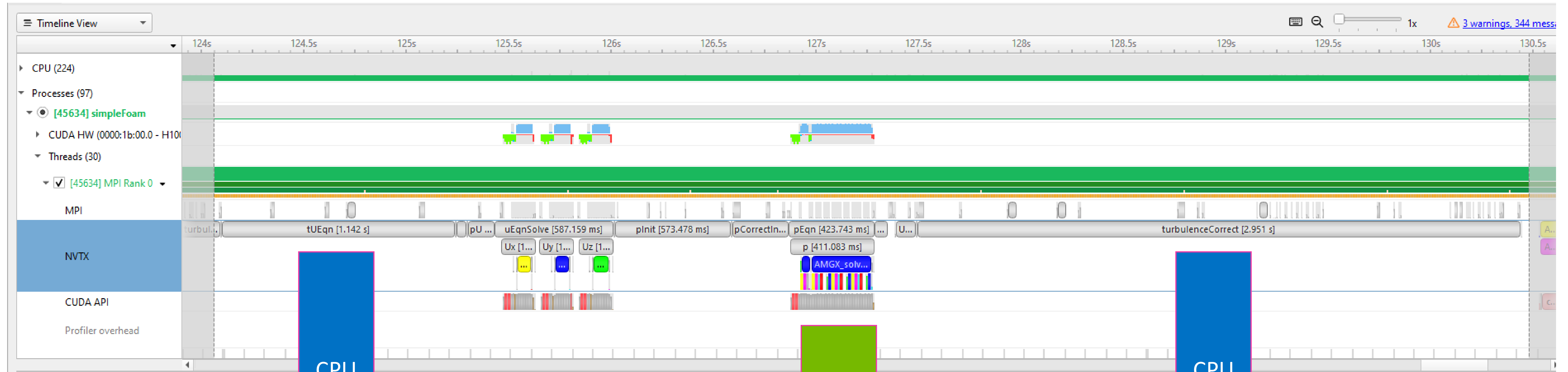


~35M cells benchmark designed by
OpenFOAM HPC technical committee

OpenFOAM

Motorbike L

x86 + H100
(H100 80GB
HBM3)

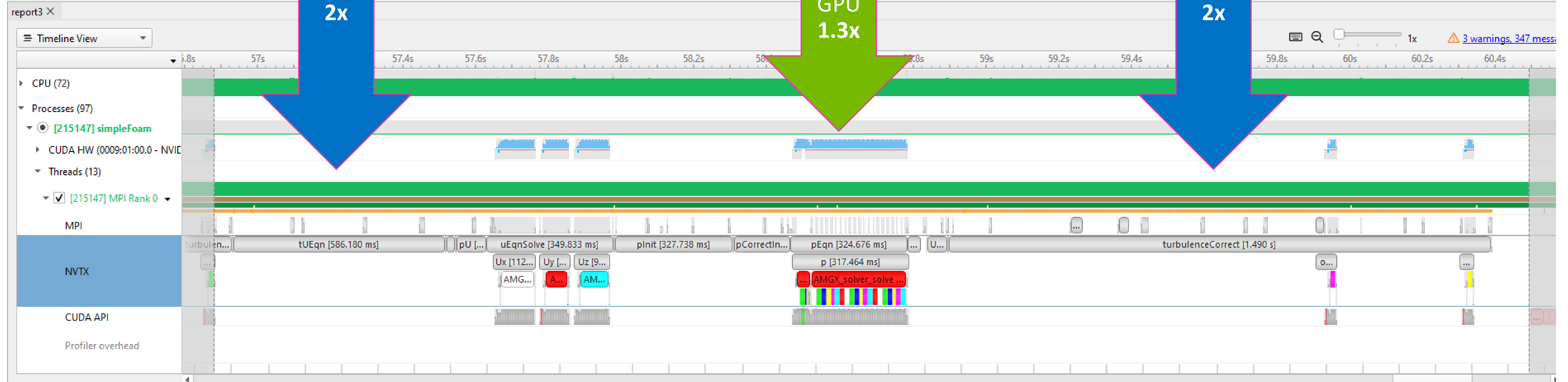


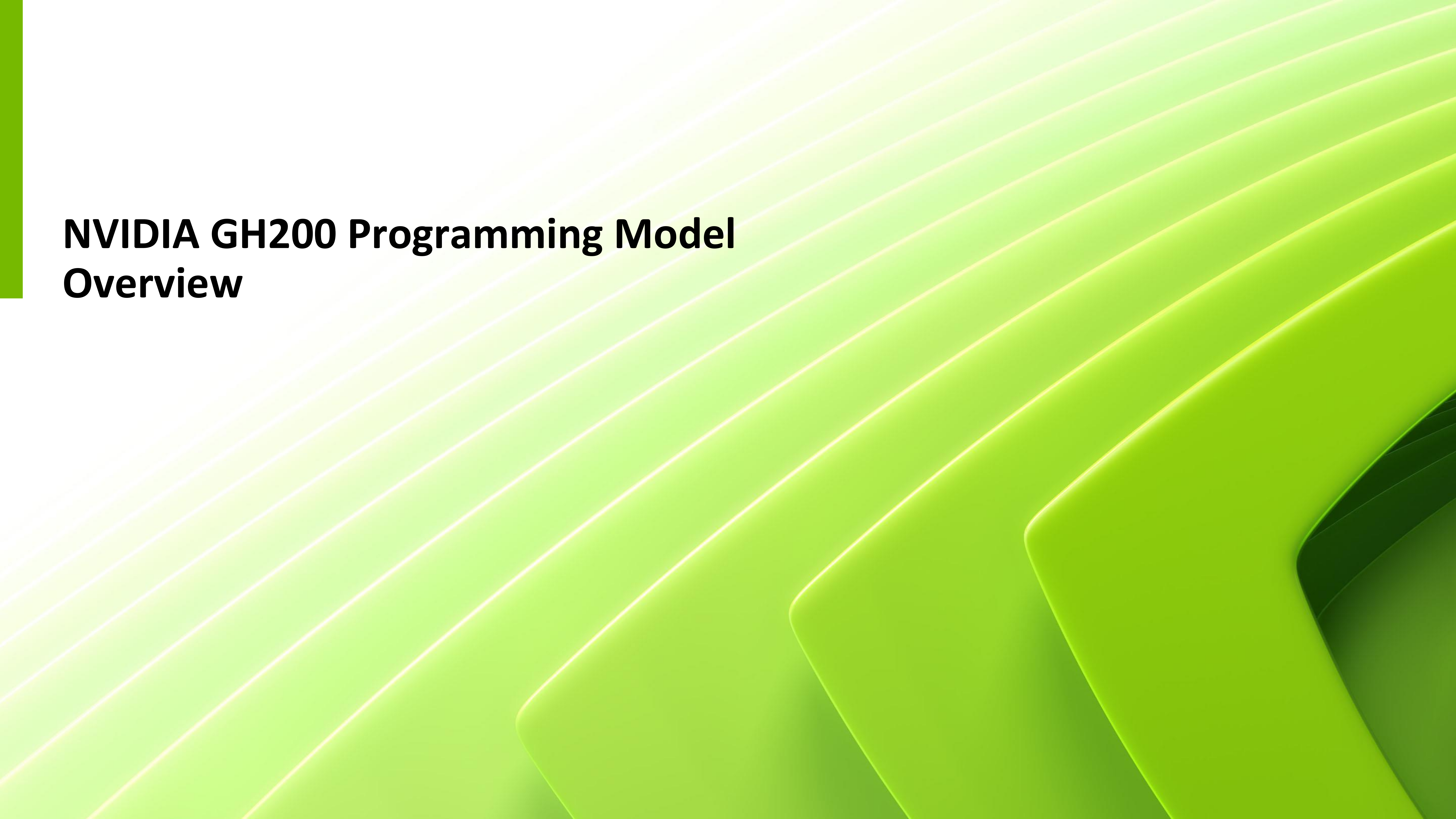
CPU
2x

GPU
1.3x

CPU
2x

Grace
Hopper
(H100 96GB
HBM3)





NVIDIA GH200 Programming Model Overview

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



Programming the NVIDIA platform

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

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

```
do concurrent (i = 1:n)  
  y(i) = y(i) + a*x(i)  
enddo
```

```
import cunumeric as np  
...  
def saxpy(a, x, y):  
  y[:] += a*x
```

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {  
...  
std::transform(par, x, x+n, y, y,  
              [=](float x, float y){  
                return y + a*x;  
              }  
);  
...  
}  
  
#pragma omp target data map(x,y) {  
...  
std::transform(par, x, x+n, y, y,  
              [=](float x, float y){  
                return y + a*x;  
              }  
);  
...  
}
```

PLATFORM SPECIALIZATION

CUDA

```
__global__  
void saxpy(int n, float a,  
          float *x, float *y) {  
  int i = blockIdx.x*blockDim.x +  
          threadIdx.x;  
  if (i < n) y[i] += a*x[i];  
}  
  
int main(void) {  
  ...  
  cudaMemcpy(d_x, x, ...);  
  cudaMemcpy(d_y, y, ...);  
  
  saxpy<<<(N+255)/256,256>>>(...);  
  
  cudaMemcpy(y, d_y, ...);
```

ACCELERATION LIBRARIES

Core

Math

Communication

Data Analytics

AI

Quantum

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	

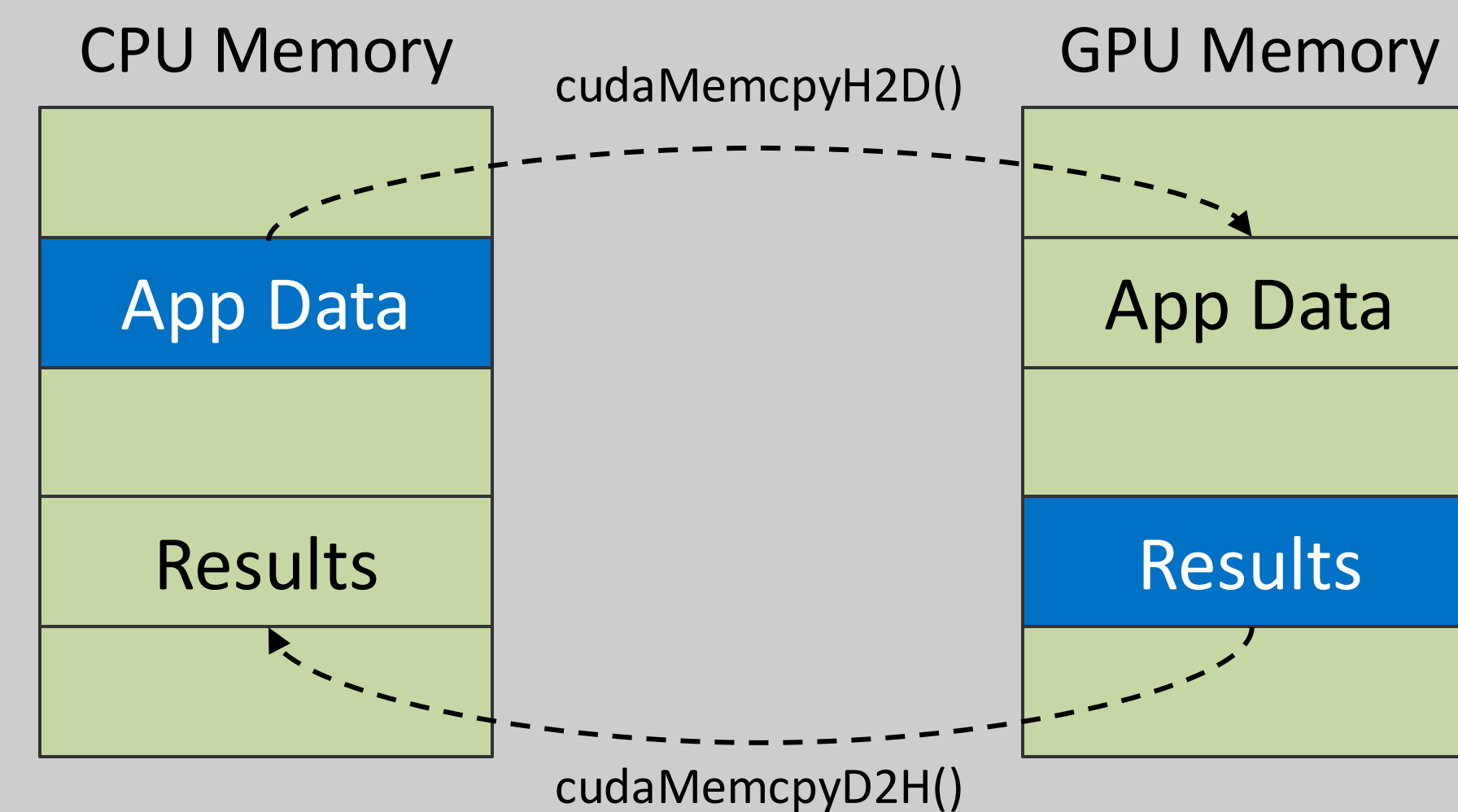
By design these approaches are interoperable so developers can choose the right balance for their needs.

ADVANTAGES OF THE GRACE HOPPER MEMORY MODEL

Full CUDA support with additional Grace memory extensions

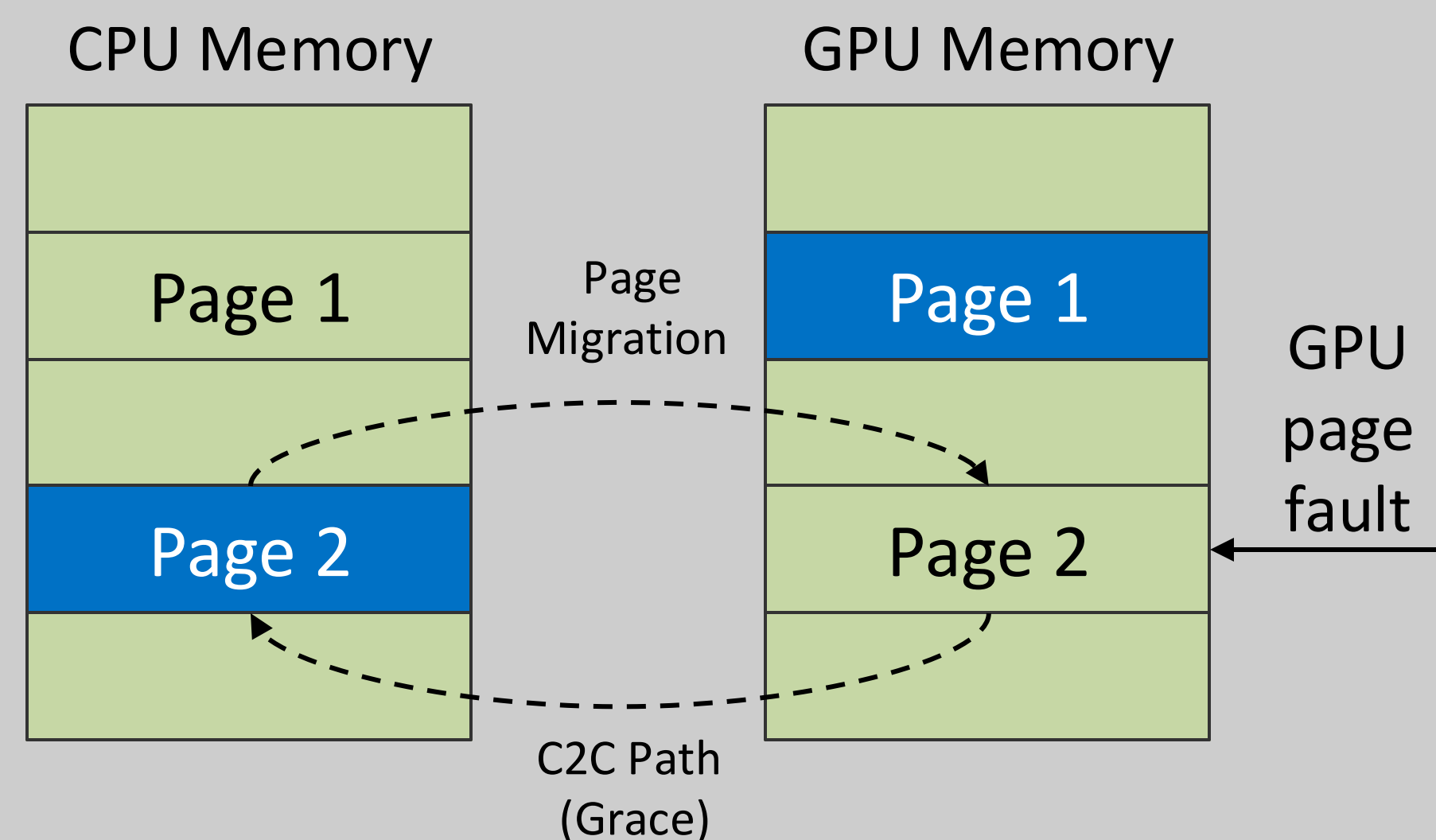
Explicit Copy

Application explicitly moves data between CPU & GPU as needed



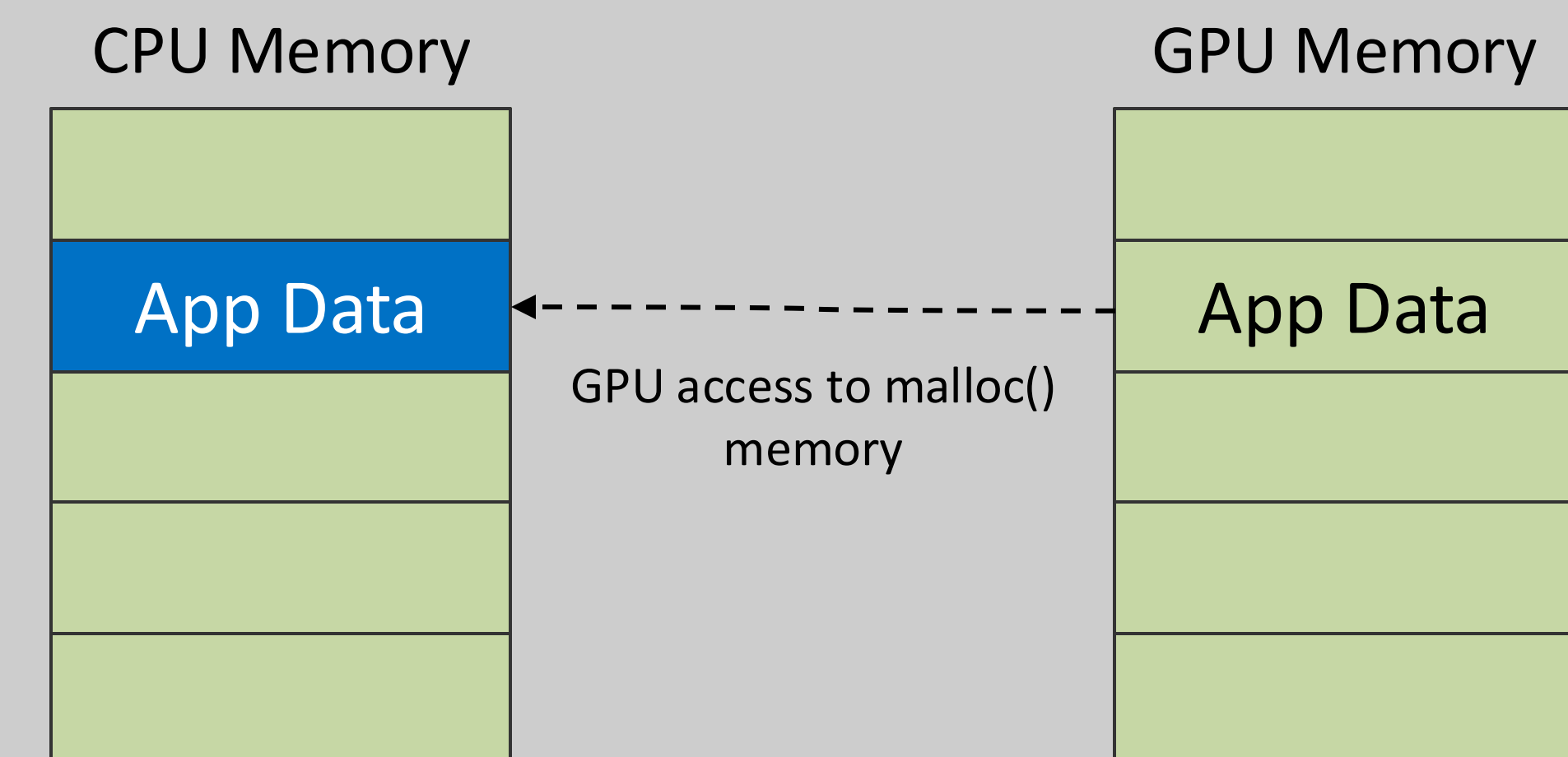
Managed Memory

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



System Allocated

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

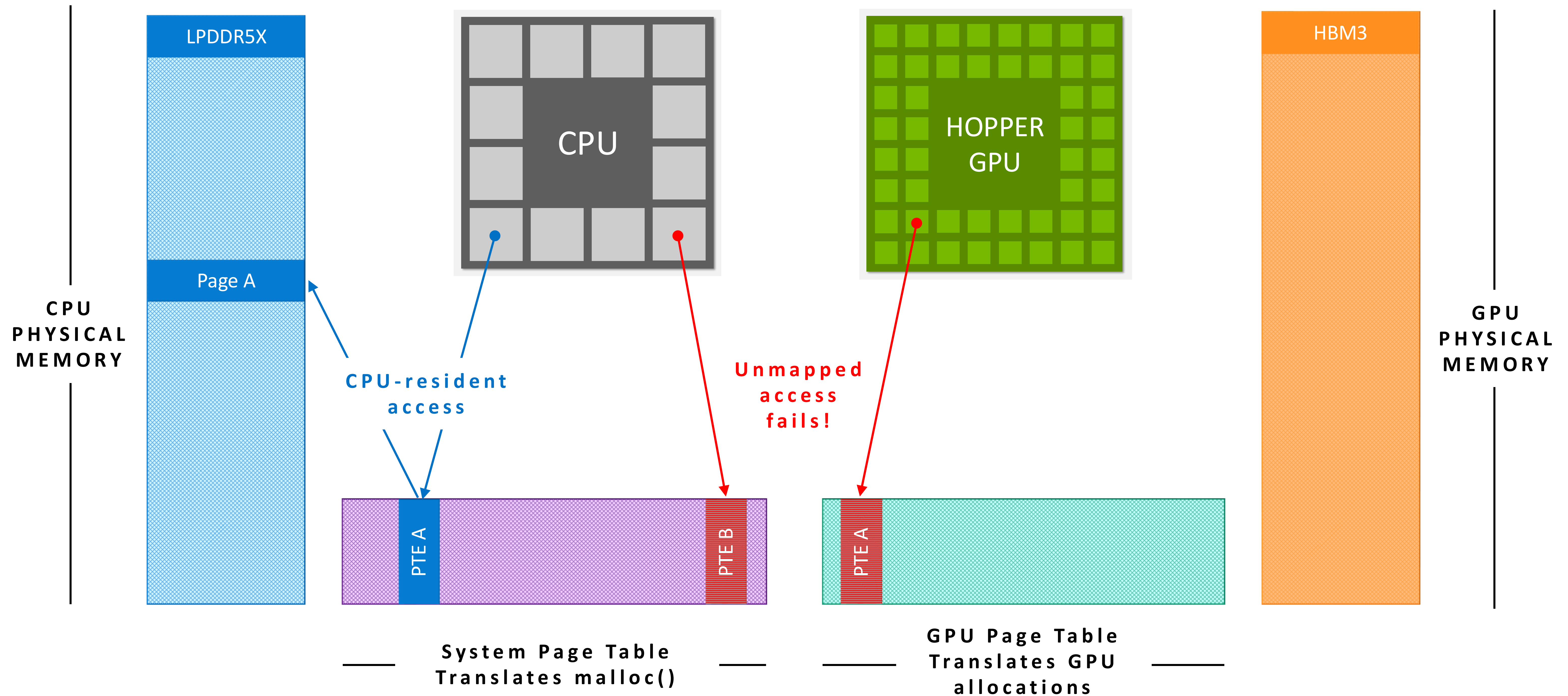
7x faster transfers, up to 450 GB/s (NVLink C2C)

Migrations not required and faster migrations when they happen at NVLink C2C speed

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

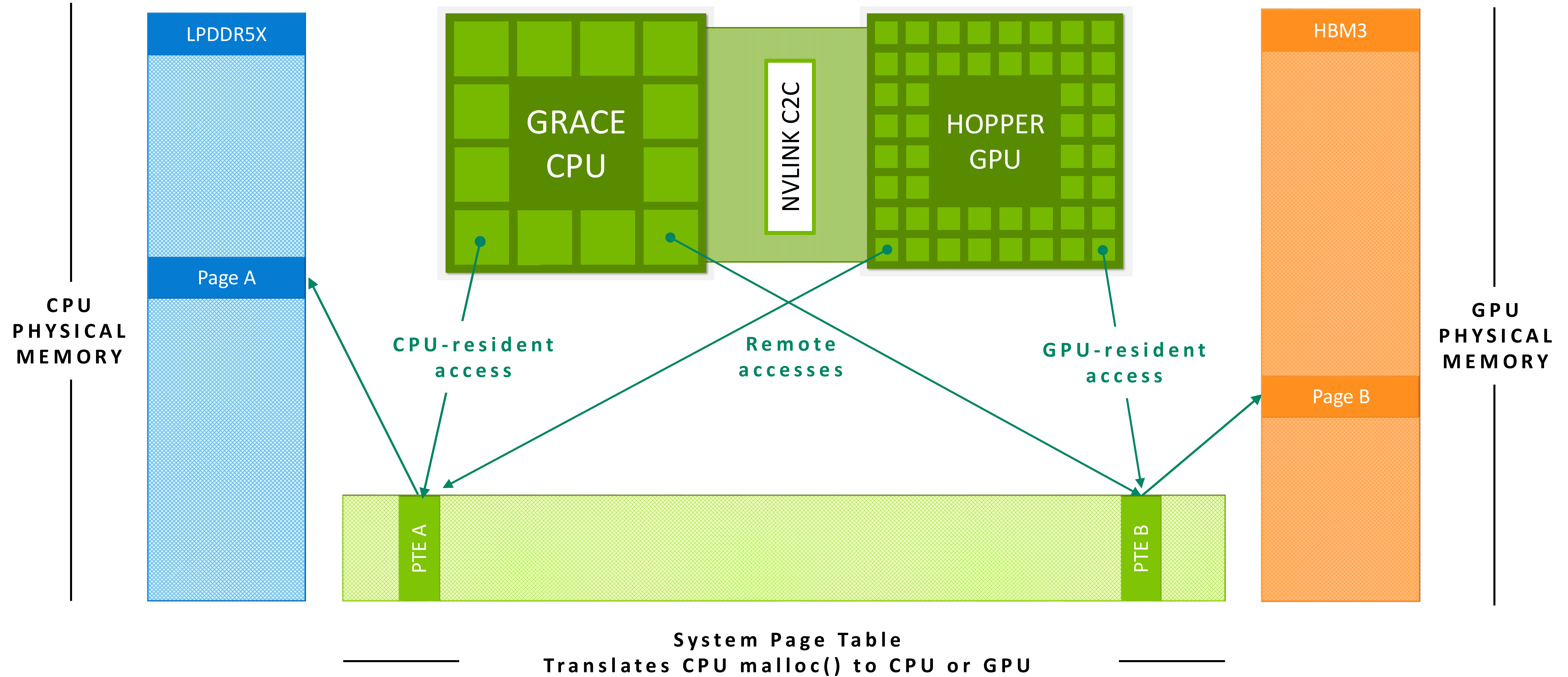
HW/SW memory view on X86 + GPU

Separate page tables



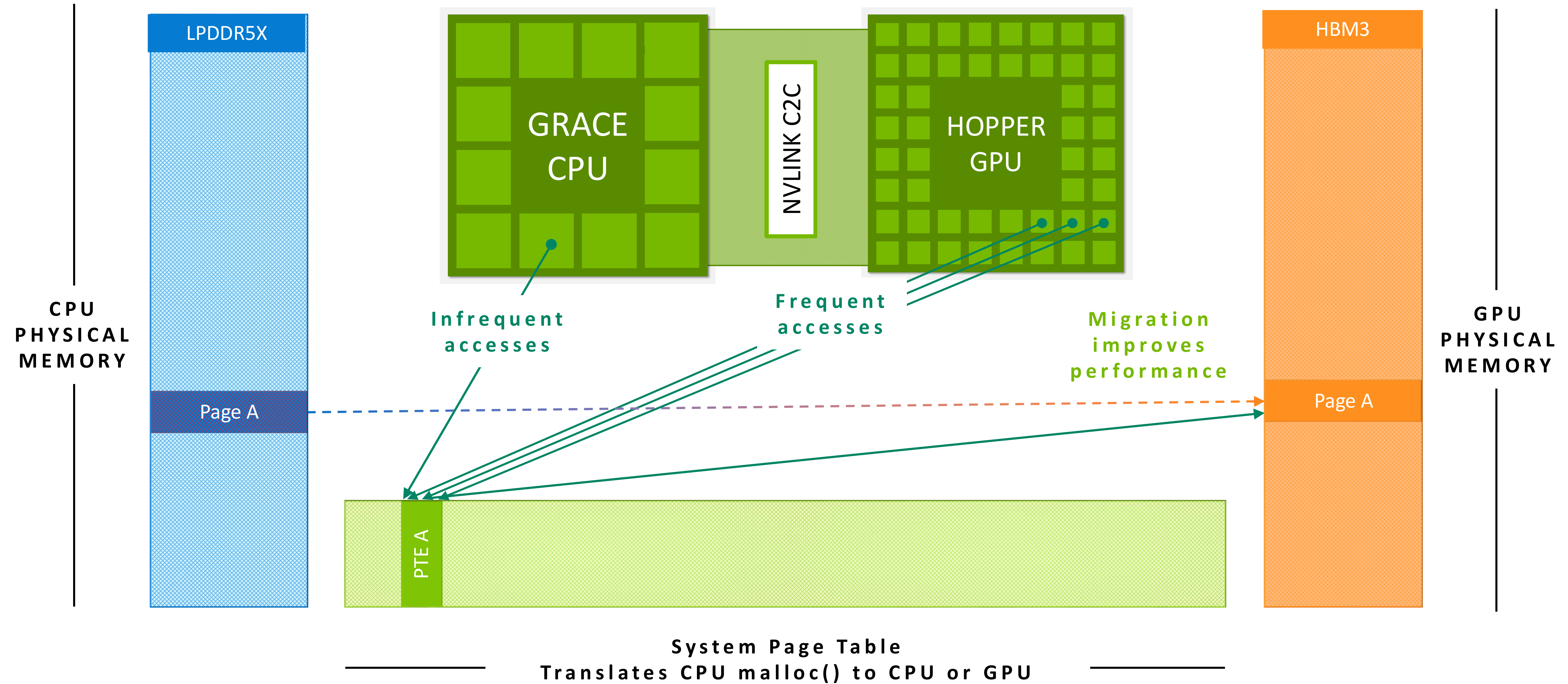
HW/SW memory view on GH200

Address Translation Service (ATS)



HW/SW memory view on GH200

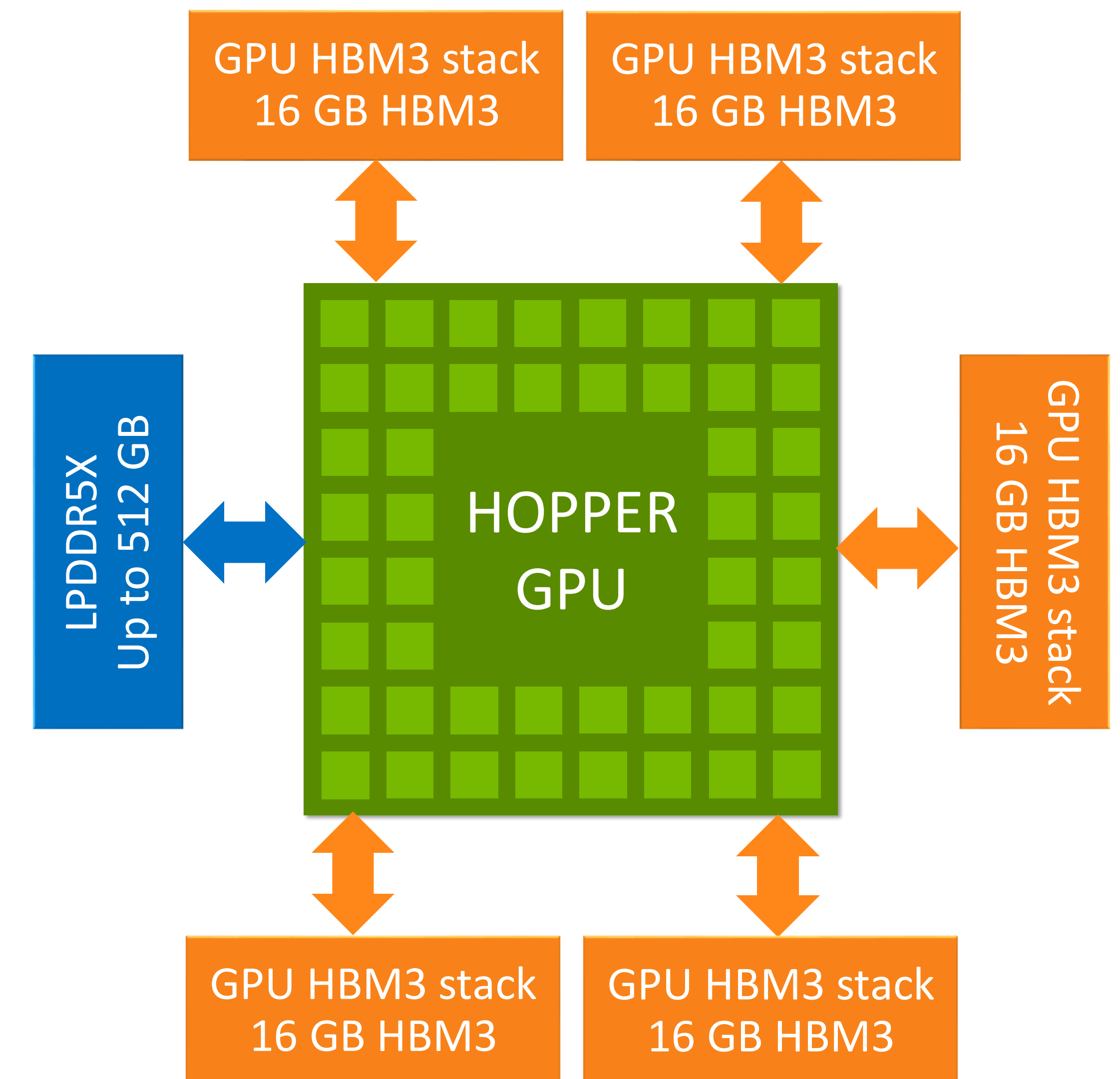
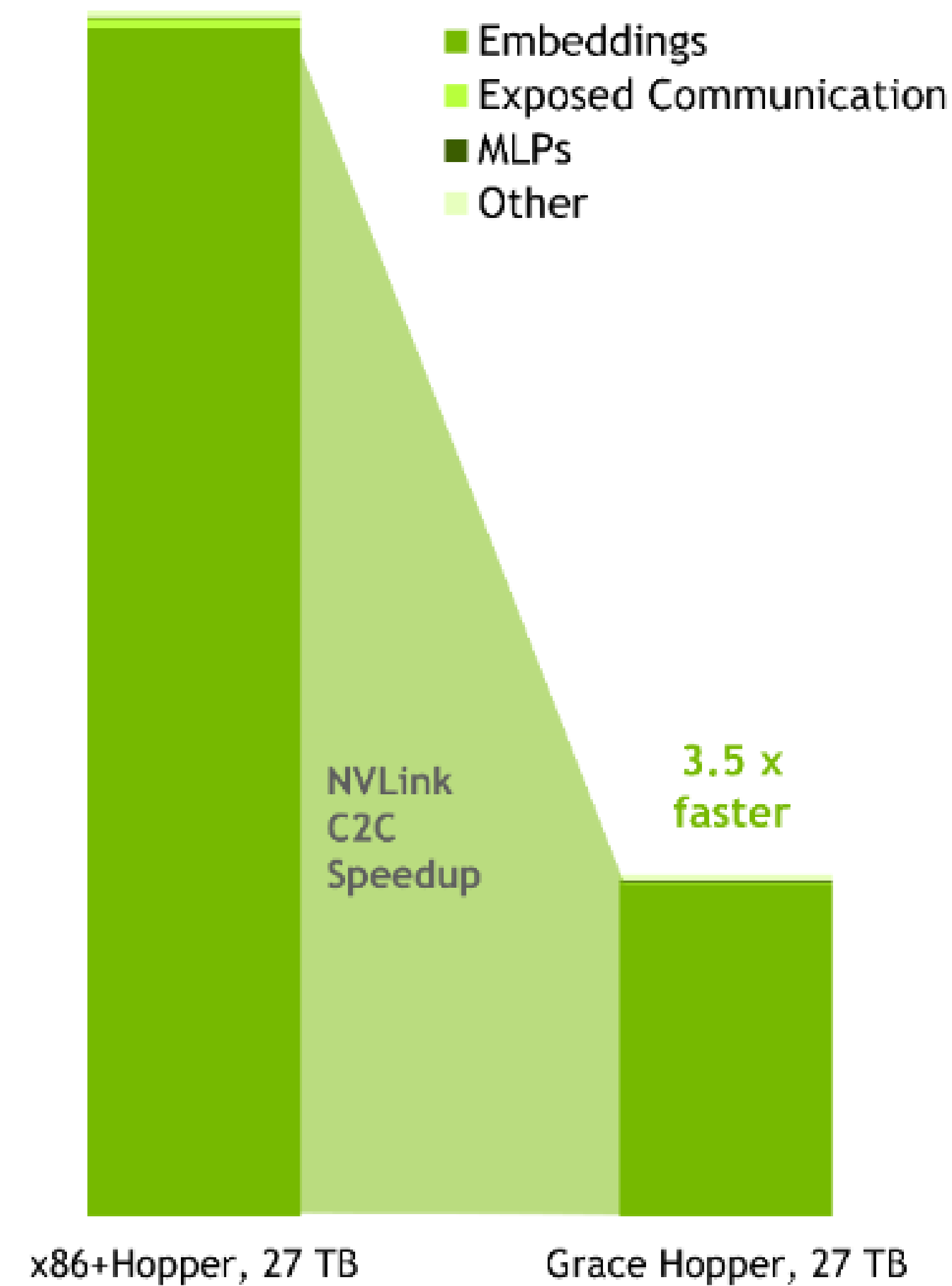
Simplified Unified Memory via **Address Translation Service (ATS)** & automatic migrations



Using CPU as a memory pool

CPU Memory can be thought of large fast access pool

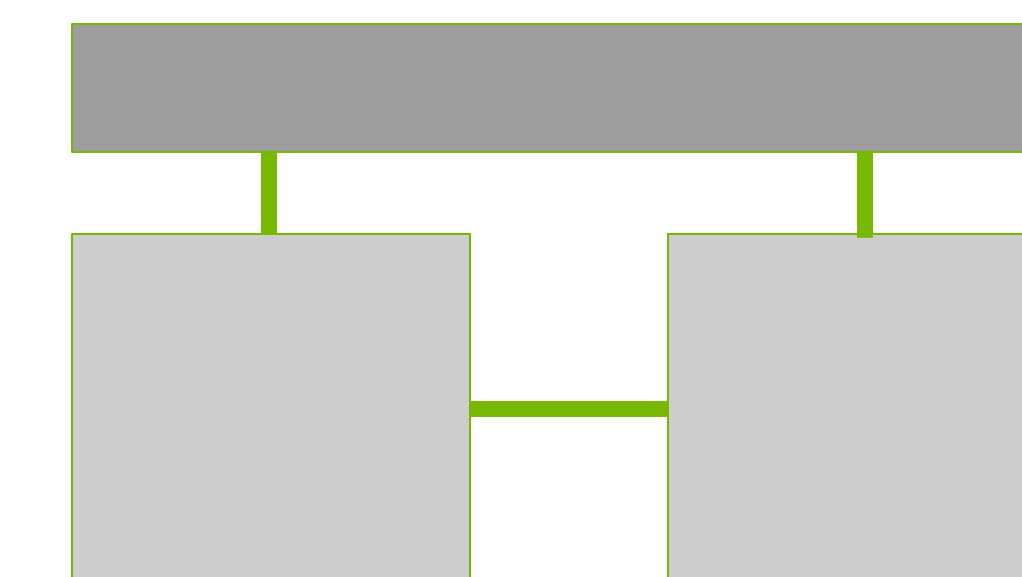
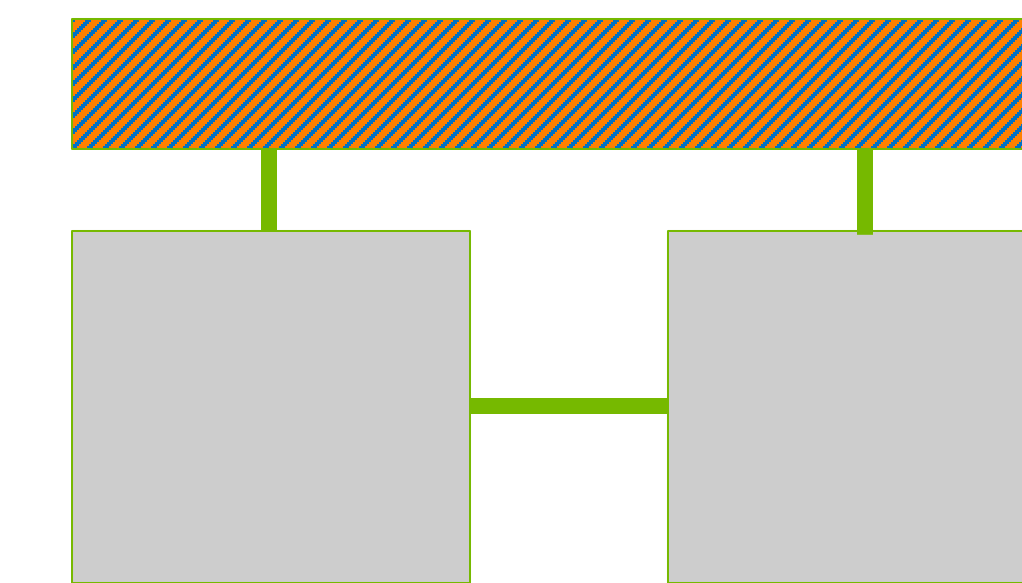
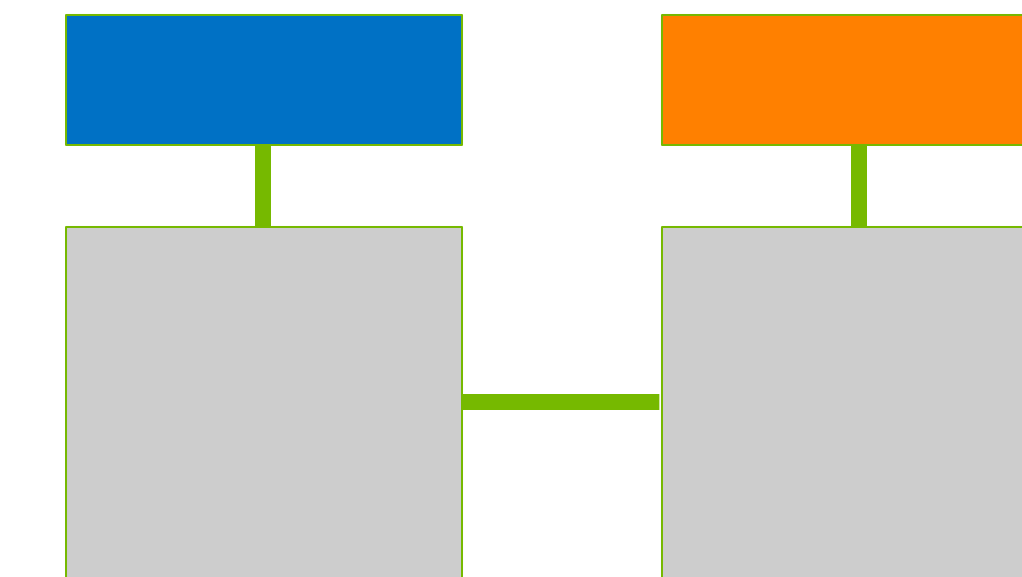
- CPU memory can be thought as “extra stacks” of accessible memory
- Large capacity and slightly further away than HBM3
- Perfect for Deep Recommendation Models (DLRMs)
 - Large embeddings



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.

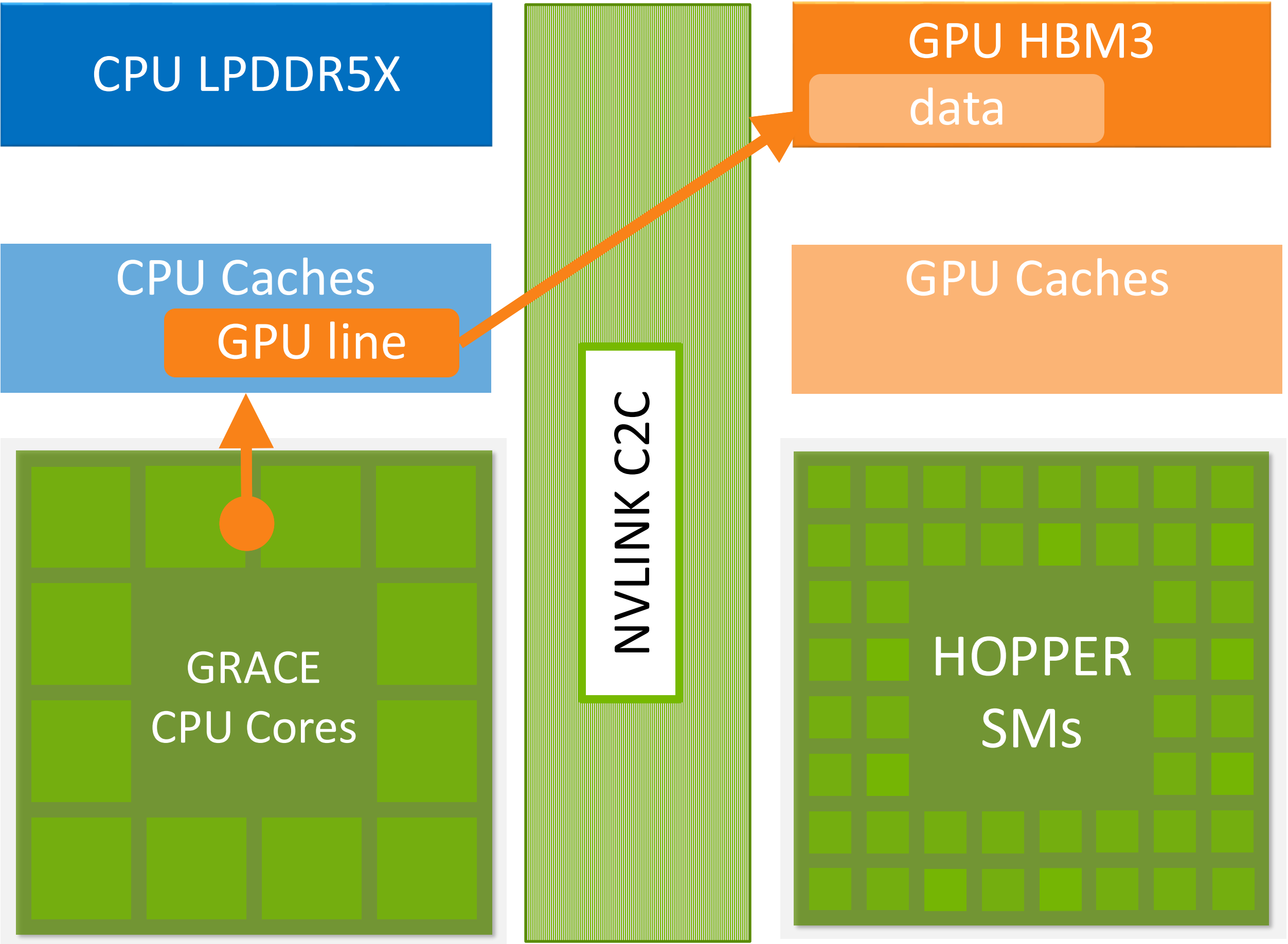


NVHPC Compilers GPU Memory Model Flags

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 explicitly 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

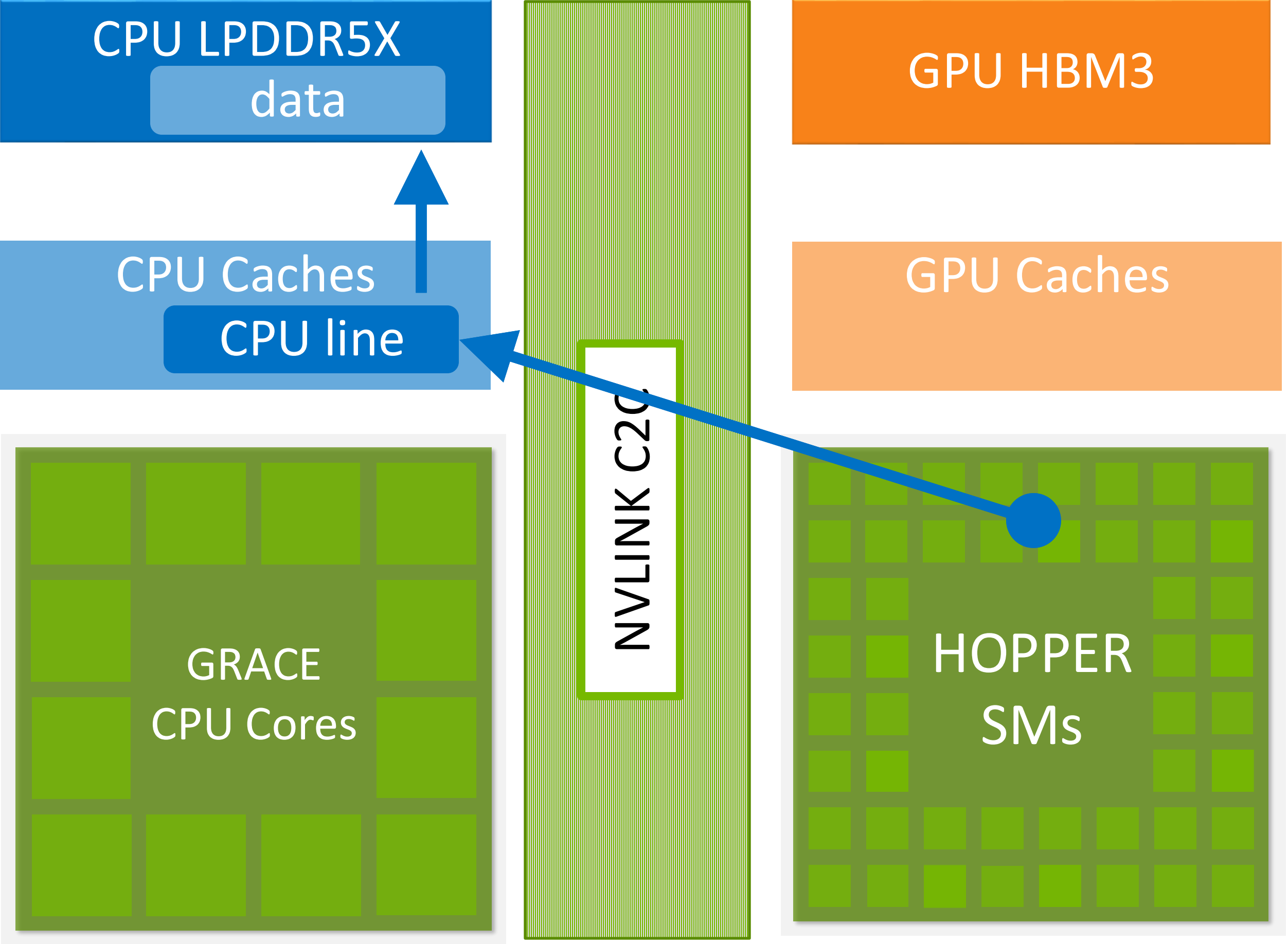
Global Access to All Data

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

Explicit Data Movement using CUDA (bulk/blocking)

Works on both x86 and Grace Hopper

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

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

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

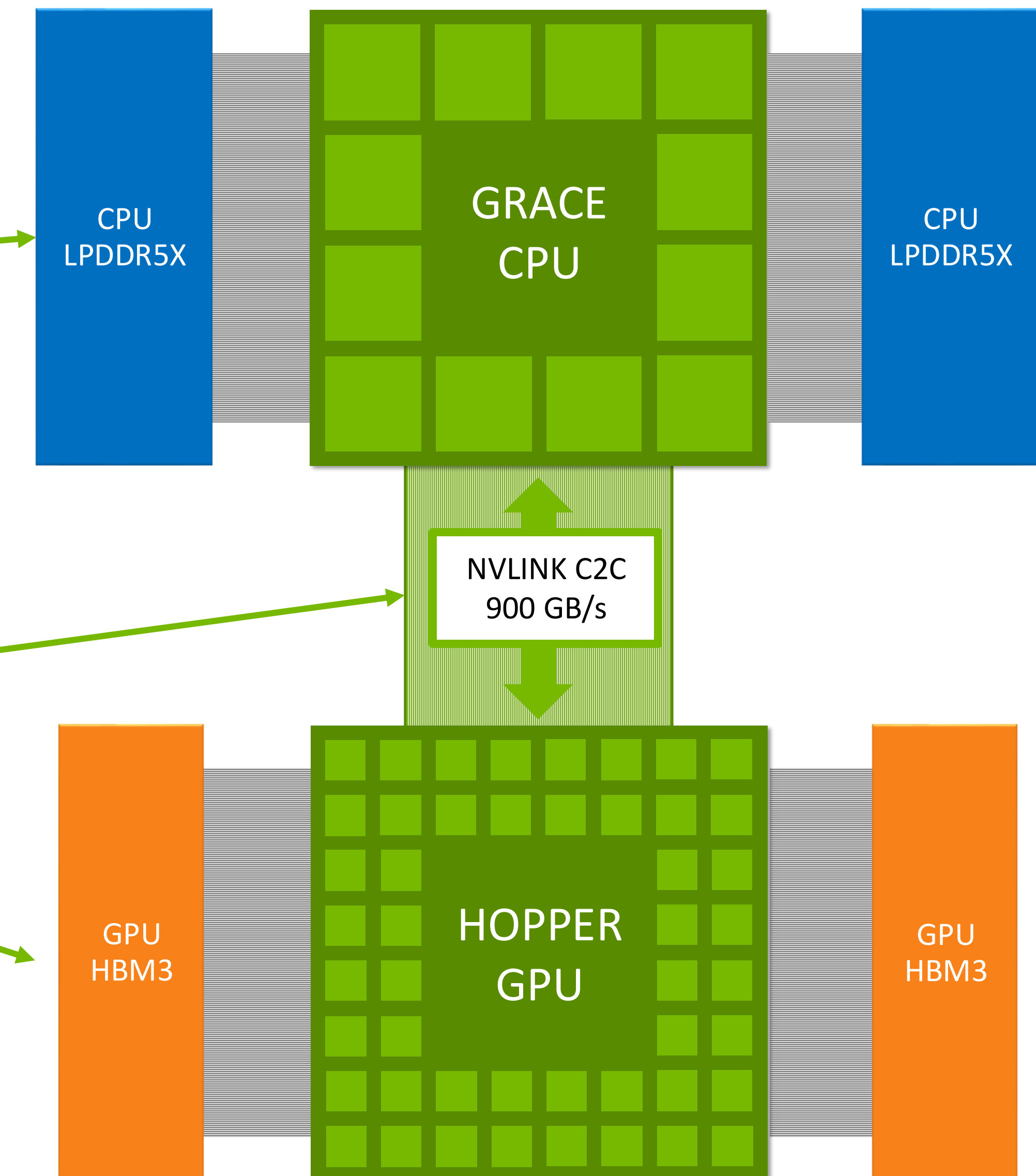
    cudaMalloc((void **)&d_data, N * sizeof(int));
    cudaMemcpy(d_data, data, N * sizeof(int), cudaMemcpyHostToDevice);

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

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

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

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



Managed Memory using CUDA

Portable codes across x86 and Grace Hopper

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

int main()
{
    int N = 128;
    int *data;
    cudaMallocManaged((void **)&data, N * sizeof(int));

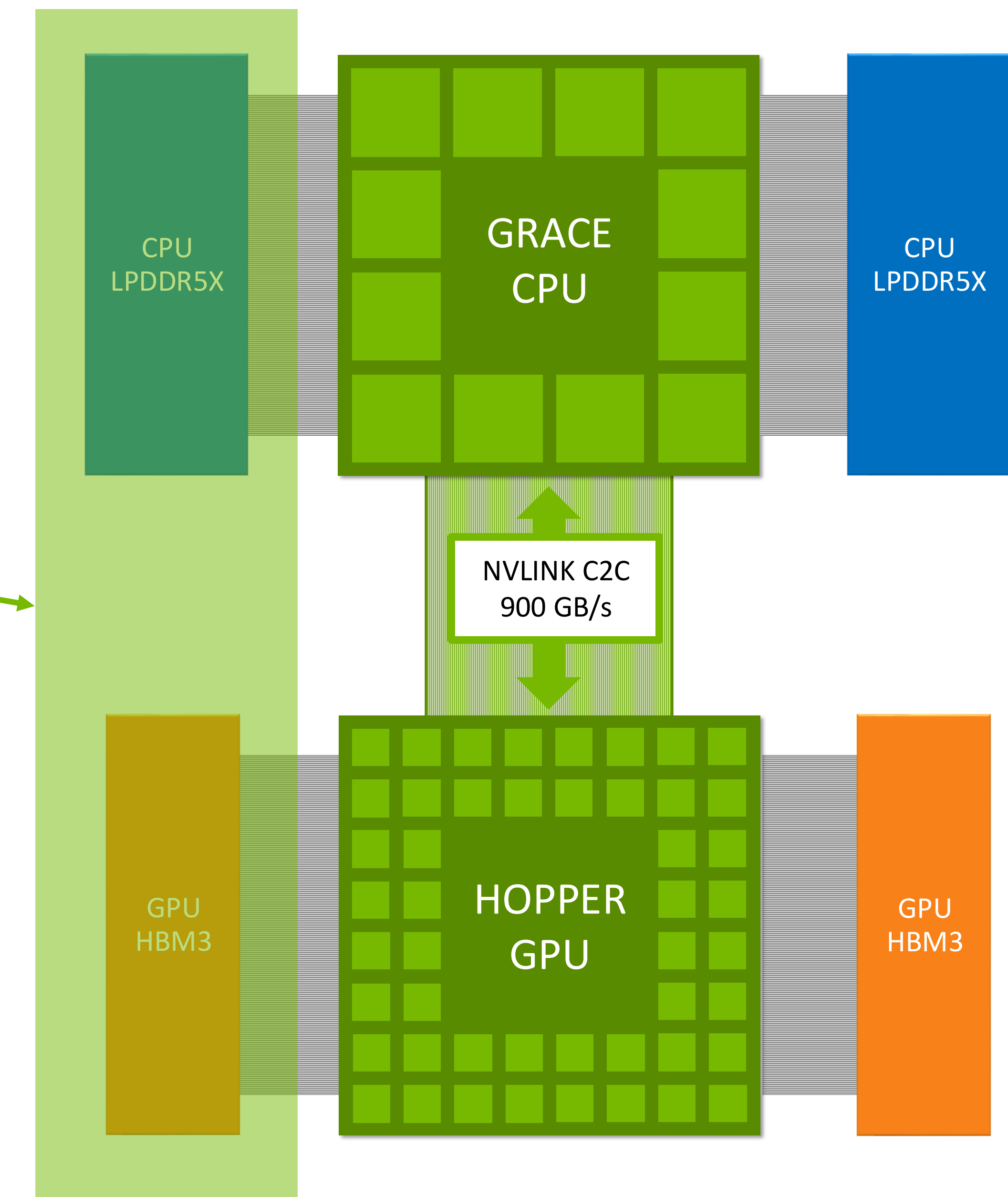
    for (int i = 0; i < N; i++)
    { data[i] = i; }

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

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

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

    cudaFree(data);
}
```



Managed Memory using CUDA

Portable codes across x86 and Grace Hopper

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

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

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

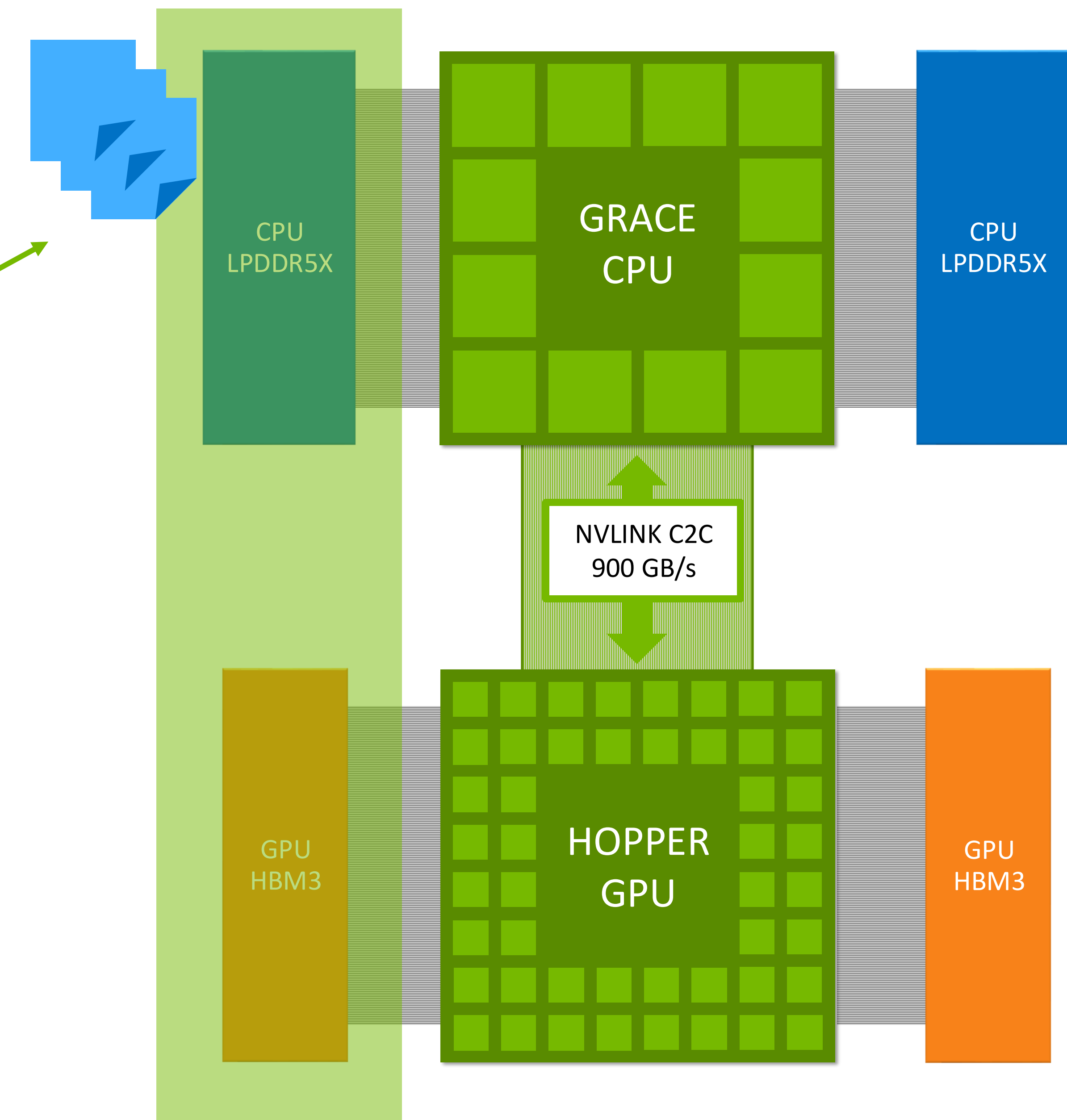
    for (int i = 0; i < N; i++)
    { data[i] = i; }

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

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

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

    cudaFree(data);
}
```



Managed Memory using CUDA

Portable codes across x86 and Grace Hopper by direct access

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

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

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

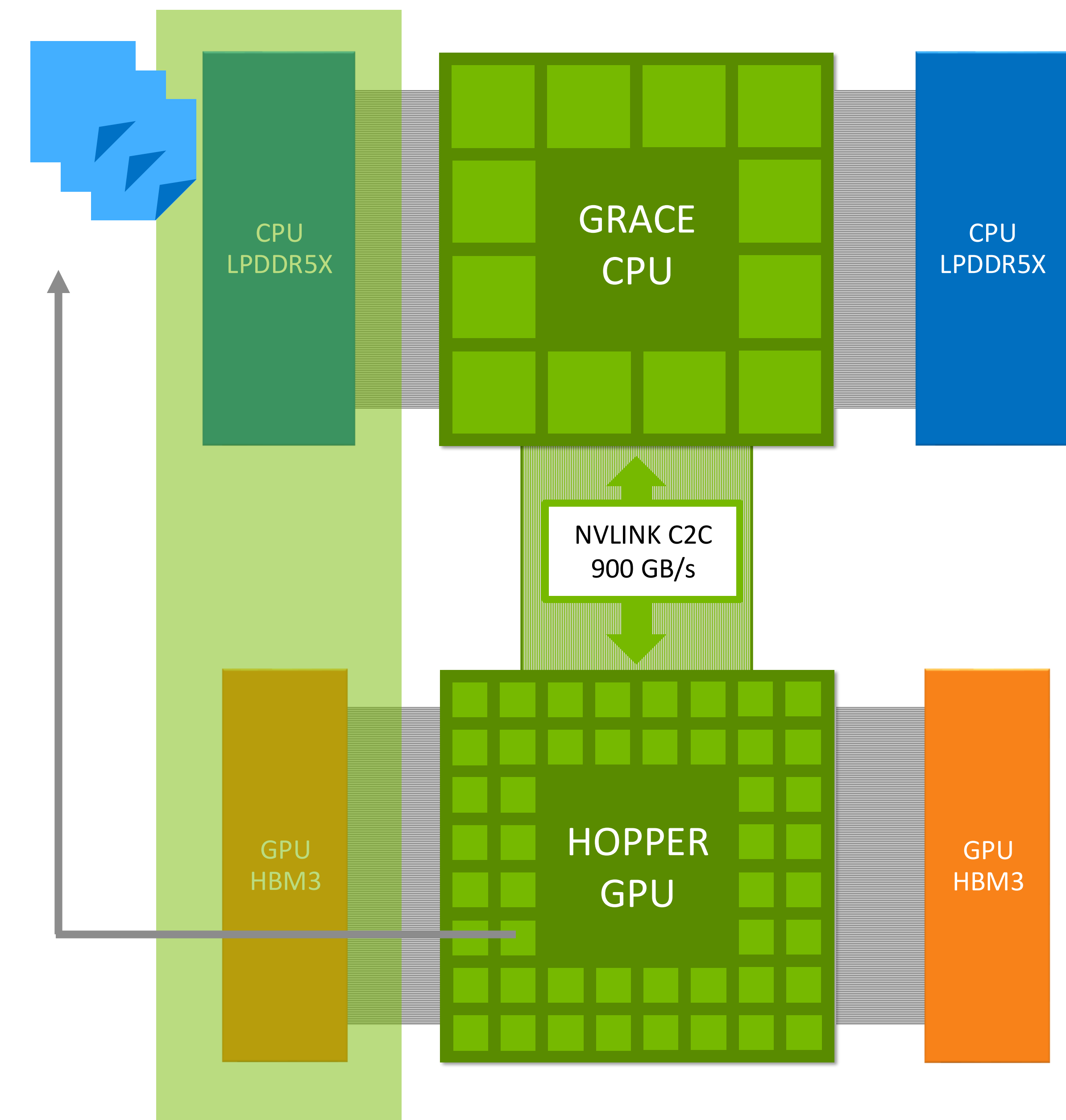
    for (int i = 0; i < N; i++)
    { data[i] = i; }

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

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

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

    cudaFree(data);
}
```



Managed Memory using CUDA

Portable codes across x86 and Grace Hopper by direct access or moving pages

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

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

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

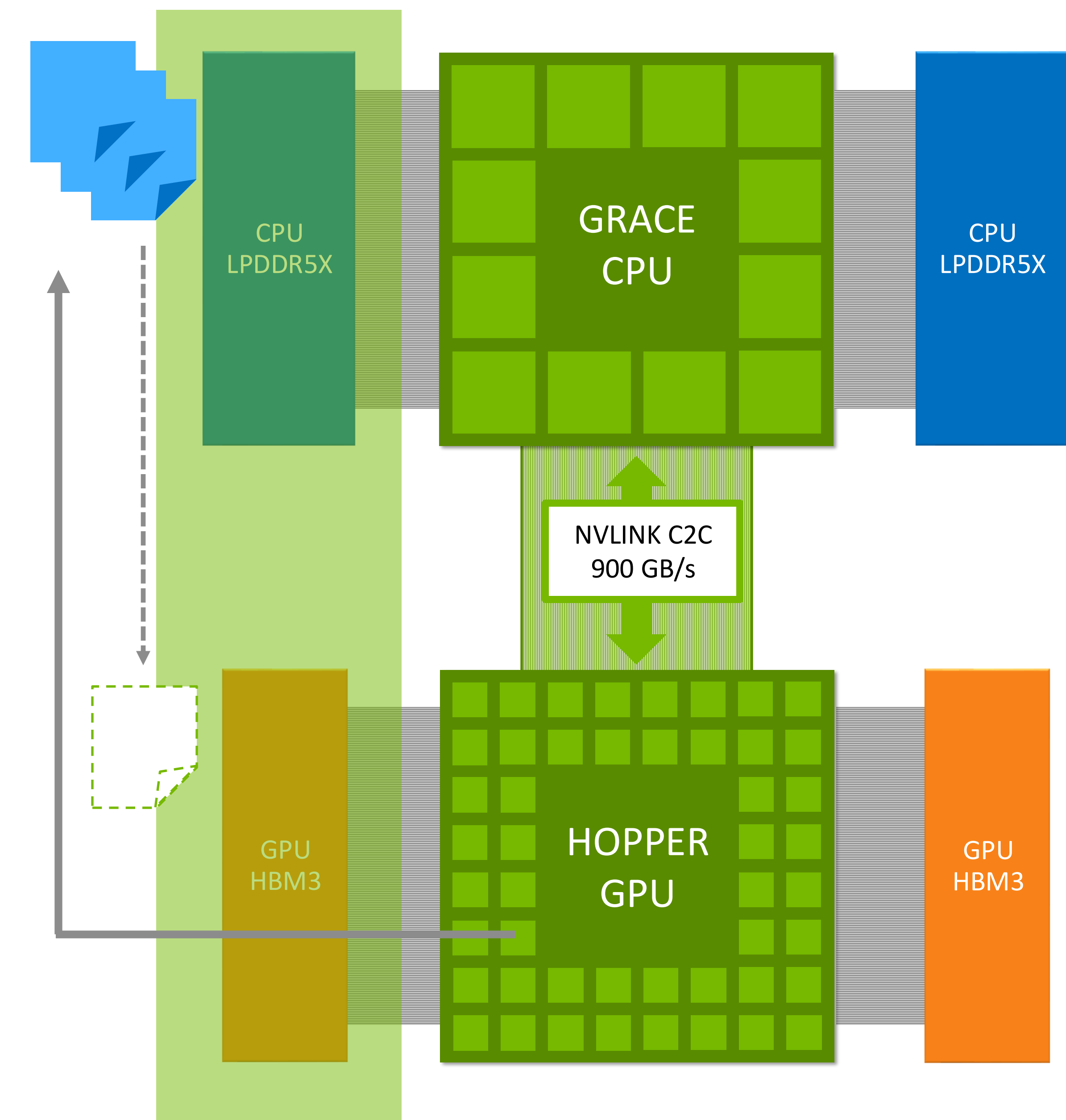
    for (int i = 0; i < N; i++)
    { data[i] = i; }

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

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

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

    cudaFree(data);
}
```



Using System Allocator using CUDA

Easy to move codes to GPU. Uses Address Translation Service (ATS)

```
__global__ void kernel(int *data1, int *data2)
{
    data1[threadIdx.x] = threadIdx.x;
    data2[threadIdx.x] = threadIdx.x;
}

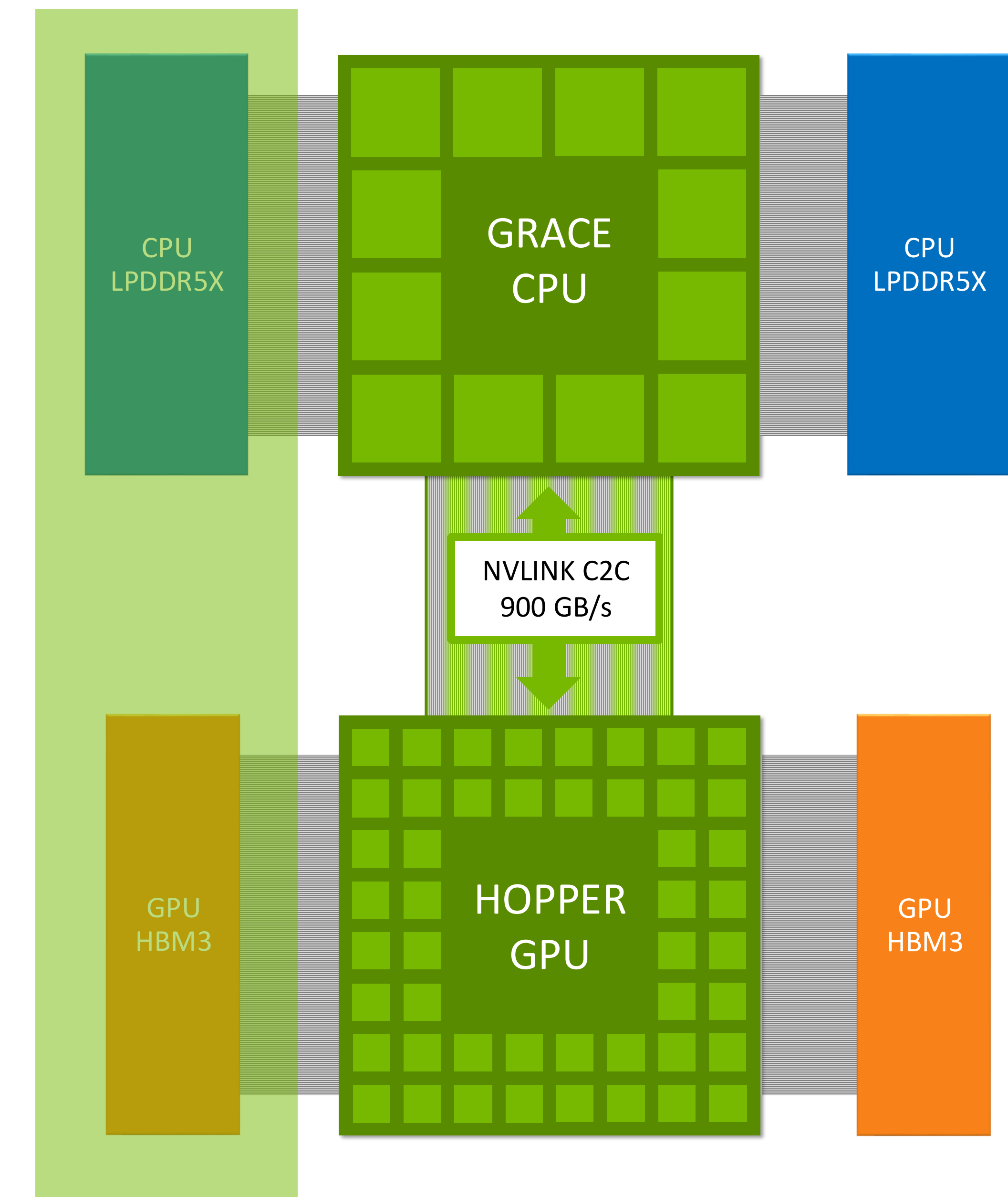
int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));
    int data2[N];

    kernel<<<1, N>>>(data, data2);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("(%d, %d) ", data[i], data2[i]); }

    free(data);
}
```

Physical pages are
not yet allocated



Using System Allocator using CUDA

Easy to move codes to GPU. Uses Address Translation Service (ATS)

```
__global__ void kernel(int *data1, int *data2)
{
    data1[threadIdx.x] = threadIdx.x;
    data2[threadIdx.x] = threadIdx.x;
}

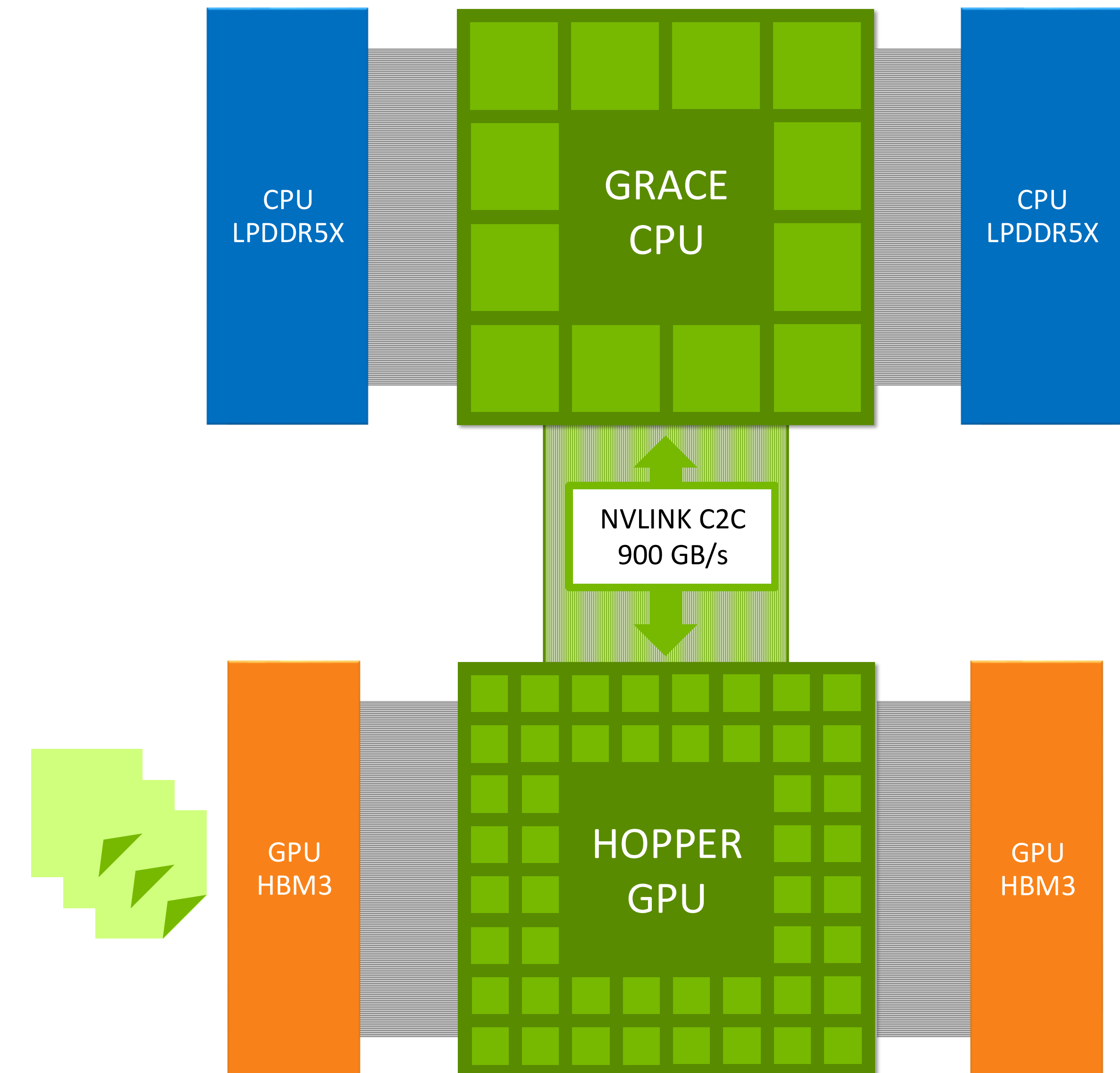
int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));
    int data2[N];

    kernel<<<1, N>>>(data, data2);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("(%d, %d) ", data[i], data2[i]); }

    free(data);
}
```

Physical pages are
allocated on first
touch



Using System Allocator using CUDA

Easy to move codes to GPU. Uses Address Translation Service (ATS)

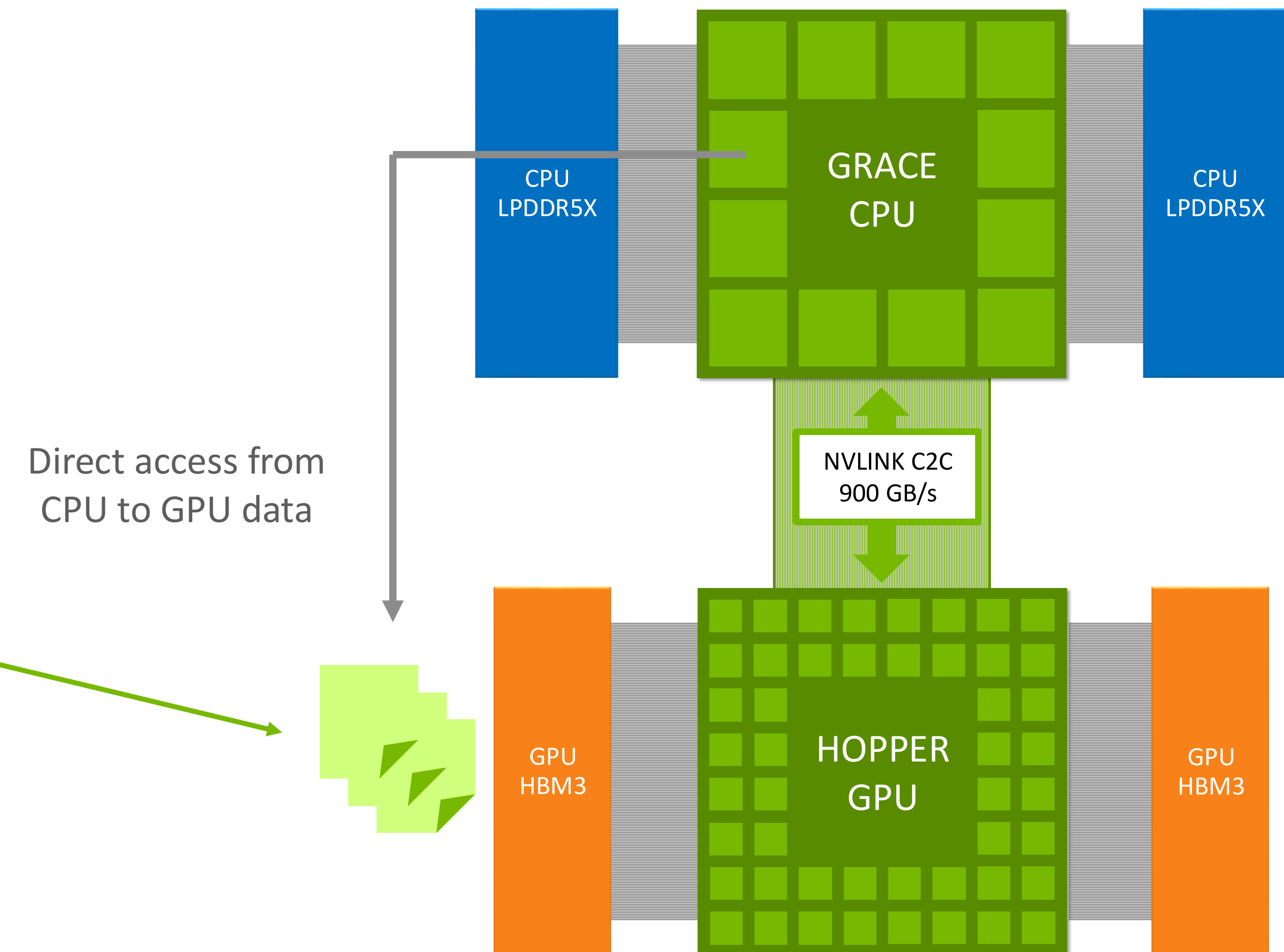
```
__global__ void kernel(int *data1, int *data2)
{
    data1[threadIdx.x] = threadIdx.x;
    data2[threadIdx.x] = threadIdx.x;
}

int main()
{
    int N = 128;
    int *data = (int *) malloc(N * sizeof(int));
    int data2[N];

    kernel<<<1, N>>>(data, data2);
    cudaDeviceSynchronize();

    for (int i = 0; i < 128; i++)
    { printf("(%d, %d) ", data[i], data2[i]); }

    free(data);
}
```



What do Programmers Need to Know About Grace-Hopper

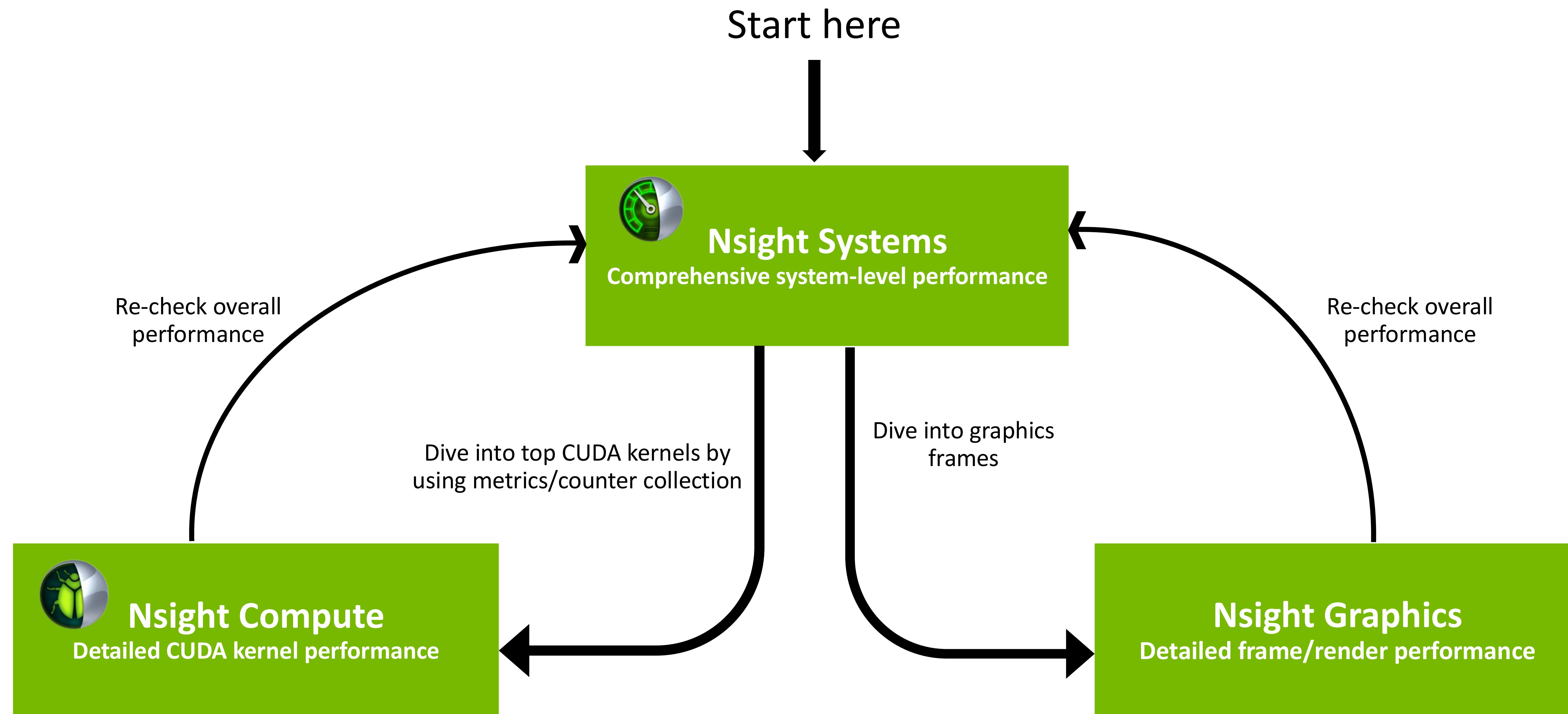
- **Existing GPU applications require no changes for Grace Hopper**
 - No new APIs
 - No restructuring
 - No new programming model
 - Developers who choose to can optimize for the Grace Hopper platform
- **Existing GPU applications (fully or partially ported) will run better on Grace Hopper**
 - Data 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



NVIDIA Developer Tools

- Nsight Systems -

Nsight Tools のワークフロー

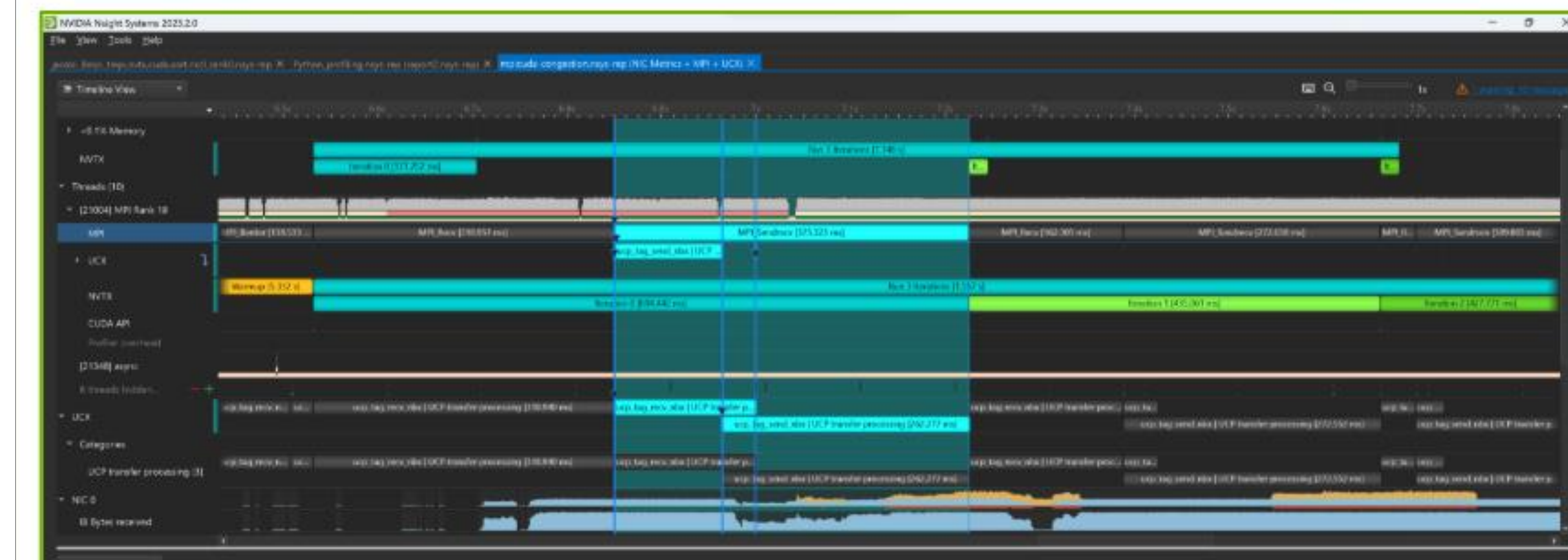
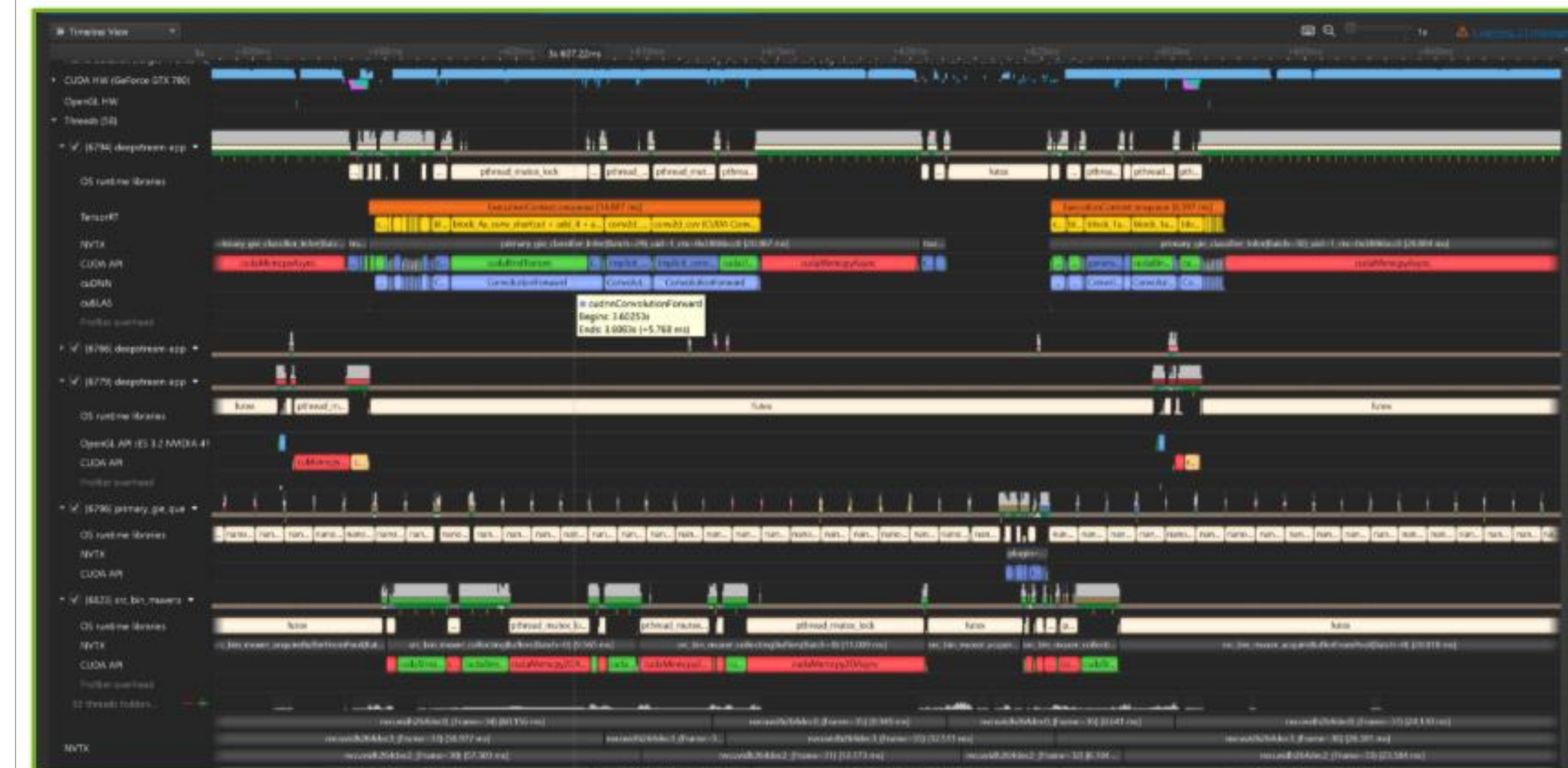
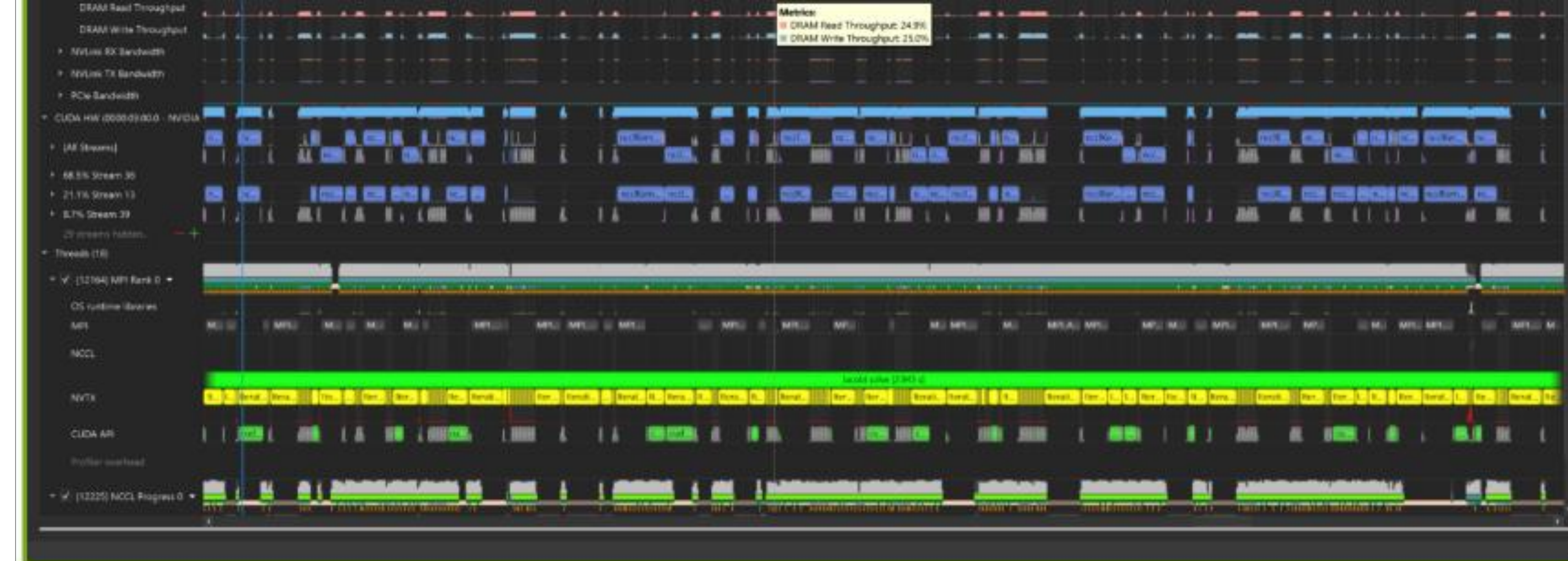


Nsight Systems

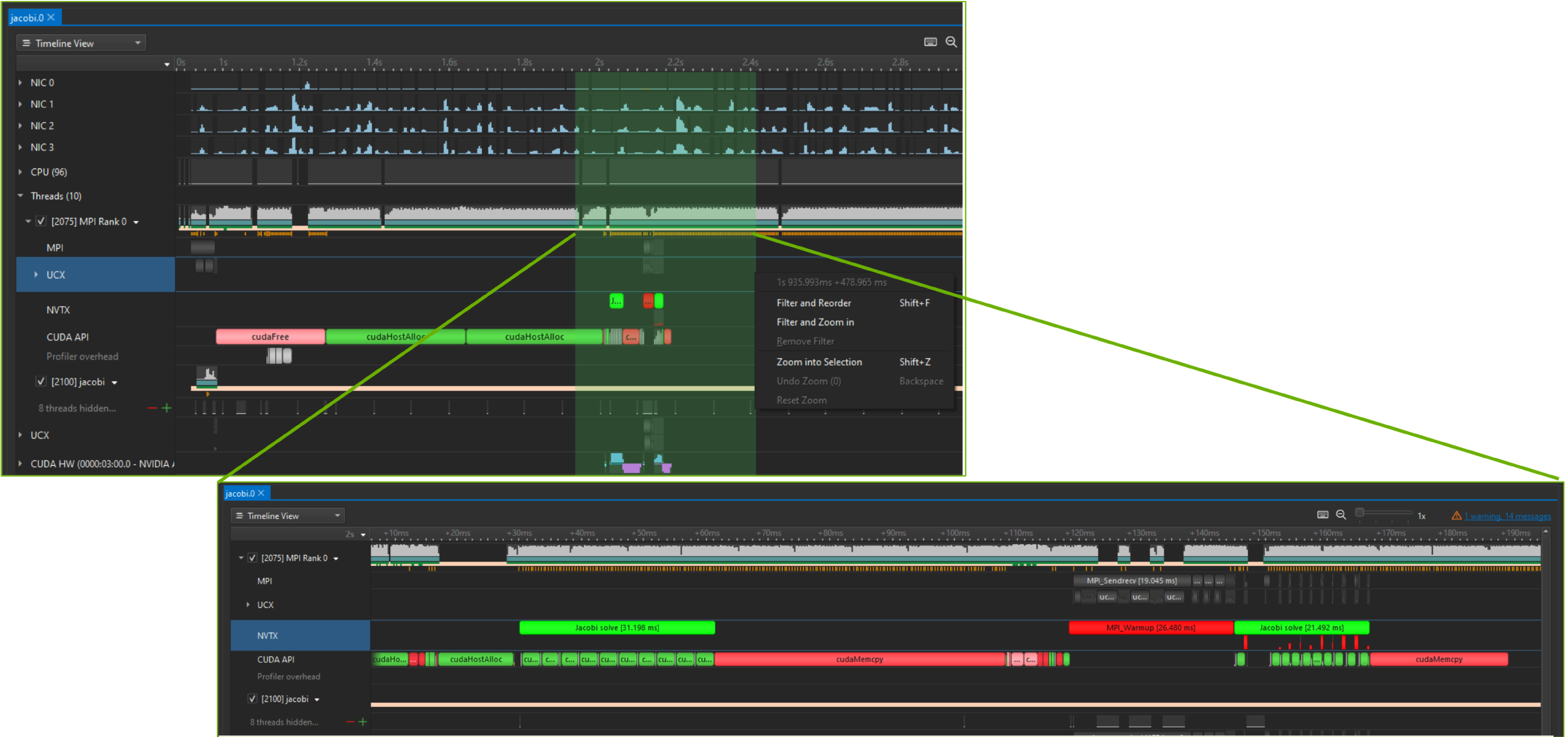
System Profiler

Key Features:

- System-wide application algorithm tuning
 - Multi-process tree support
- Locate optimization opportunities
 - Visualize millions of events on a very fast GUI timeline
 - Identify gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
 - CPU algorithms, utilization and thread state
 - GPU streams, kernels, memory transfers, etc
- Command line, Standalone, IDE Integration
- OS: Linux (x86, ARM Server, Tegra), Windows, MacOS X (host)
- GPUs: Pascal+
- Docs/Product: <https://developer.nvidia.com/nsight-systems>



Zoom/Filter to Exact Areas of Interest



NVIDIA Tools eXtensions (NVTX)

Nsight Systems と合わせて使うと便利なライブラリ

- ソースコードにアノテーションするための、CUDA ライブラリ
- NVIDIA HPC SDK にも同梱
- Nsight Systems と併用することで、アノテーションしたレンジがタイムライン上に表示される
- CPU 側の処理も含めて、タイムライン上での処理が把握しやすくなる
- Header-only ライブラリ
 - C/C++: `#include <nvtx3/nvToolsExt.h>`
 - Fortran: プログラム中で `use nvtx` して、コンパイル時に `-lnvhpcwrapnvtx`
- マクロで NVTX の無効化が可能
 - コンパイル時に `-DNVTX_DISABLE` を付加

NVTX:

<https://github.com/NVIDIA/NVTX/tree/release-v3/c>

Nsight Systems 101

Quick start

- Nsight Systems CLI によるプロファイリング

```
$ nsys profile [options] <application> [application-arguments]
```

- `-t <parameters>` : トレースする API を指定。デフォルトは、`cuda, opengl nvtx, osrt`
- `--stats <true|false>` : `true` でプログラム実行時の統計情報を標準出力に表示
- `-o <filename>` : 出力ファイル名を指定
- `-f <true|false>` : `true` で出力ファイルの上書きを許可、デフォルトは `false`

など... 詳細は、`nsys --help` or `nsys [specific command] --help` で確認可能

- `<filename>.nsys-rep` が出力される
- ローカル PC に `<filename>.nsys-rep` を転送し、Nsight Systems UI で可視化

Nsight Systems user guide:

<https://docs.nvidia.com/nsight-systems/UserGuide/index.html>

Nsight Systems のローカル端末 へのインストール

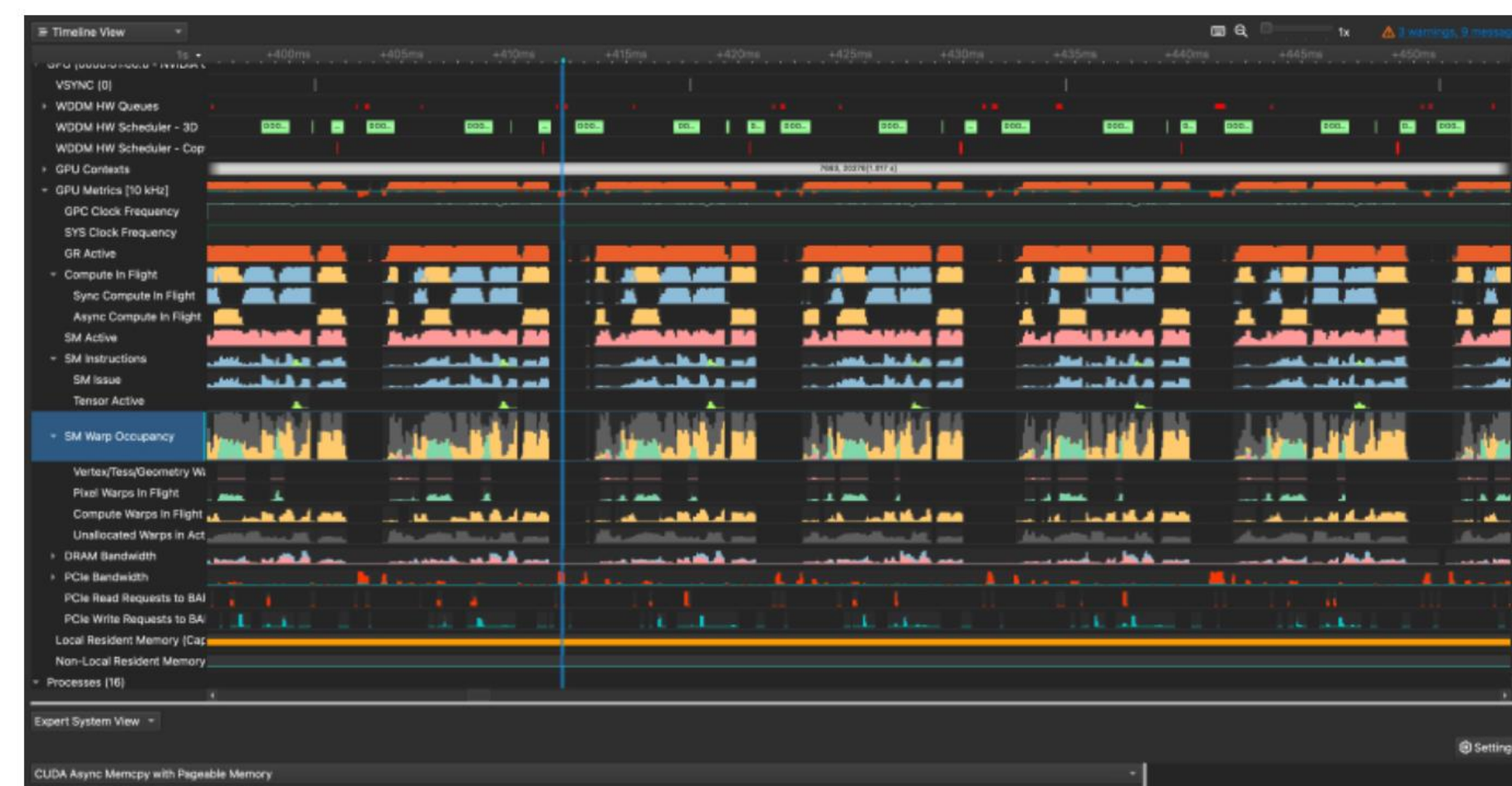
Win/Mac/Linux 用を提供

NVIDIA Nsight Systems

NVIDIA Nsight™ Systems is a system-wide performance analysis tool designed to visualize an application's algorithms, identify the largest opportunities to optimize, and tune to scale efficiently across any quantity or size of CPUs and GPUs, from large servers to our smallest system on a chip (SoC).

Get started

Nsight Systems 2024.1 is available now.



Download

Latest version

Supported
Platforms

System
Requirements

Release Notes

Feature Table

Archives

Resources

Documentation

Self-Paced
Training

Tutorial Sessions

Video Series

Support

Download NVIDIA Nsight Systems

Nsight Systems 2024.1 is Available Now

Review the [supported platforms](#) for NVIDIA Nsight™ Systems to choose the correct version for your host and profiling target.

If profiling from the CLI, pick your platform based on where the CLI will be run. If using the GUI (Full Version) to view reports, do profiling, or do remote profiling, pick your platform based on the host PC architecture where the GUI will be run.

Also review the [system requirements](#) before downloading.

Desktop, workstation, and server platforms:

Download for Windows on x86_64

Download for Linux on x86_64

Download for Linux on Power9

Download for Linux on Arm Servers and NVIDIA Grace

Download for macOS

<https://developer.nvidia.com/nsight-systems>

サンプルコード

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )
```

```
error = calcNext(A, Anew, m, n)
```

```
call swap(A, Anew, m, n)
```

```
if(mod(iter,100).eq.0 ) write(*,'(i5,f10.6)'), it
```

```
iter = iter + 1
```

```
end do
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    Anew(i,j) = 0.25_fp_kind * ( A(i+1,j) + A(i-1,j) + &  
                                A(i,j-1) + A(i,j+1) )  
    error = max( error, abs(Anew(i,j)-A(i,j)) )  
  end do  
end do  
...
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    A(i,j) = Anew(i,j)  
  end do  
end do  
...
```

サンプルコード

C

```
while ( error > tol && iter < iter_max ) {
```

```
    error = calcNext(A, Anew, m, n);
```

```
    swap(A, Anew, m, n);
```

```
    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter,
```

```
    iter++;
```

```
}
```

```
...
for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {
        Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)]
                                         + A[OFFSET(j, i-1, m)]
                                         + A[OFFSET(j-1, i, m)]
                                         + A[OFFSET(j+1, i, m)] );
        error = fmax( error, fabs(Anew[OFFSET(j, i, m)]
                                   - A[OFFSET(j, i, m)]));
    }
}
...
```

```
...
for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
    }
}
...
```

サンプルコード

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )
```

```
error = calcNext(A, Anew, m, n)
```

```
call swap(A, Anew, m, n)
```

```
if(mod(iter,100).eq.0 ) writ
```

```
iter = iter + 1
```

```
end do
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    Anew(i,i) = 0.25*fn*kind * ( A(i+1,j) + A(i-1,j) + &  
      A(i,j-1) + A(i,j+1) )  
    A(i,j) = 0.25*kind * ( A(i+1,j) + A(i-1,j) + &  
      A(i,j-1) + A(i,j+1) )  
  end do  
end do  
...  
$ nsys profile -t cuda,nvtx,openacc,osrt \   
-f true \   
-o my_report \   
./my_app
```

* 長時間実行すると巨大な結果ファイルが生成されます
イタレーション (ステップ) 数を必要最低限に設定することをオススメします

```
do i=1,n-2  
  A(i,j) = Anew(i,j)  
end do  
end do  
...
```

サンプルコード

C

```
while ( error > tol && iter < iter_max ) {
```

```
    error = calcNext(A, Anew, m, n);
```

```
    swap(A, Anew, m, n);
```

```
    if(iter % 100 == 0) printf("iter: %d, error: %f\n", iter, error);
```

```
    iter++;
```

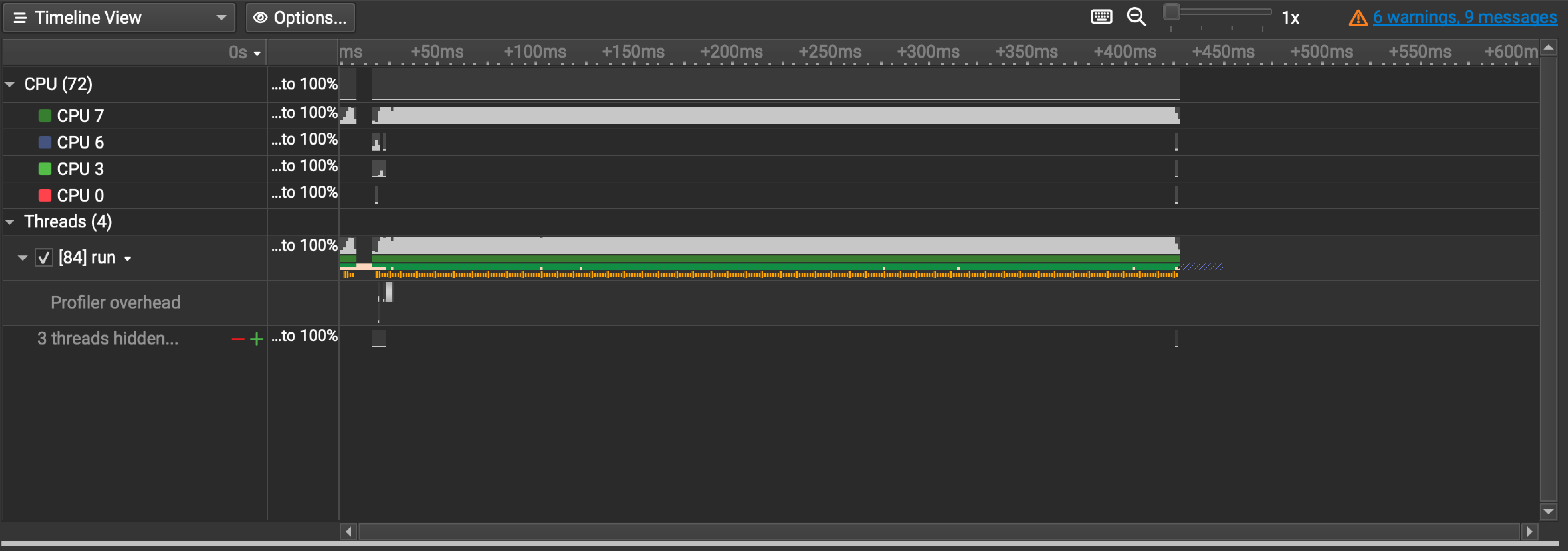
```
}
```

```
...
for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {
        Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)]
                                         + A[OFFSET(j, i-1, m)]
                                         + A[OFFSET(j-1, i, m)]
                                         + A[OFFSET(j+1, i, m)] );
        A[OFFSET(j, i, m)] = (Anew[OFFSET(j, i, m)]
                               + A[OFFSET(j, i, m)] ) / 2;
    }
}
...
```

* 長時間実行すると巨大な結果ファイルが生成されます
イタレーション (ステップ) 数を必要最低限に設定することをオススメします

```
for( int i = 1; i < m-1; i++ ) {
    A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
}
}
...
```

サンプルコードのタイムライン



サンプルコード With NVTX

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )  
  call nvtxStartRange("main loop")  
  
  call nvtxStartRange("calcNext")  
  error = calcNext(A, Anew, m, n)  
  call nvtxEndRange  
  
  call nvtxStartRange("swap")  
  call swap(A, Anew, m, n)  
  call nvtxEndRange  
  
  if(mod(iter,100).eq.0 ) write(*,'(i5,f10.6)'), it  
  
  iter = iter + 1  
  
  call nvtxEndRange  
end do
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    Anew(i,j) = 0.25_fp_kind * ( A(i+1,j) + A(i-1,j) + &  
                                A(i,j-1) + A(i,j+1) )  
    error = max( error, abs(Anew(i,j)-A(i,j)) )  
  end do  
end do  
...
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    A(i,j) = Anew(i,j)  
  end do  
end do  
...
```

サンプルコード With NVTX

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )  
  call nvtxStartRange("main loop")
```

```
  call nvtxStartRange("calcNext")  
  error = calcNext(A, Anew, m, n)  
  call nvtxEndRange
```

```
  call nvtxStartRange("swap")  
  call swap(A, Anew, m, n)  
  call nvtxEndRange
```

```
  if(mod(iter,100).eq.0 ) writ
```

```
  iter = iter + 1
```

```
  call nvtxEndRange  
end do
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    Anew(i,i) = 0.25*fn kind * ( A(i+1,j) + A(i-1,j) + &  
                                \ A(i ,j-1) + A(i ,j+1) )  
                                \ w(i,j)-A(i,j)) )  
  ...  
$ nsys profile -t cuda,nvtx,openacc,osrt \-f true  
-o my_report  
./my_app  
...
```

```
...  
do j=1,m-2  
  do i=1,n-2  
    A(i,j) = Anew(i,j)  
  end do  
end do  
...
```

サンプルコード With NVTX

C

```
while ( error > tol && iter < iter_max ) {  
    nvtxRangePush("main loop");  
  
    nvtxRangePush("calcNext");  
    error = calcNext(A, Anew, m, n);  
    nvtxRangePop();  
  
    nvtxRangePush("swap");  
    swap(A, Anew, m, n);  
    nvtxRangePop();  
  
    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter,  
    iter++;  
  
    nvtxRangePop();  
}
```

```
...  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)]  
                                         + A[OFFSET(j, i-1, m)]  
                                         + A[OFFSET(j-1, i, m)]  
                                         + A[OFFSET(j+1, i, m)] );  
  
        error = fmax( error, fabs(Anew[OFFSET(j, i, m)]  
                                - A[OFFSET(j, i, m)]));  
    }  
}  
...
```

```
...  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];  
    }  
}  
...
```

サンプルコード With NVTX

C

```
while ( error > tol && iter < iter_max ) {  
    nvtxRangePush("main loop");
```

```
    nvtxRangePush("calcNext");  
    error = calcNext(A, Anew, m, n);  
    nvtxRangePop();
```

```
    nvtxRangePush("swap");  
    swap(A, Anew, m, n);  
    nvtxRangePop();
```

```
    if(iter % 100 == 0) printf("iter %d error %f\n", iter, error);
```

```
    iter++;
```

```
    nvtxRangePop();
```

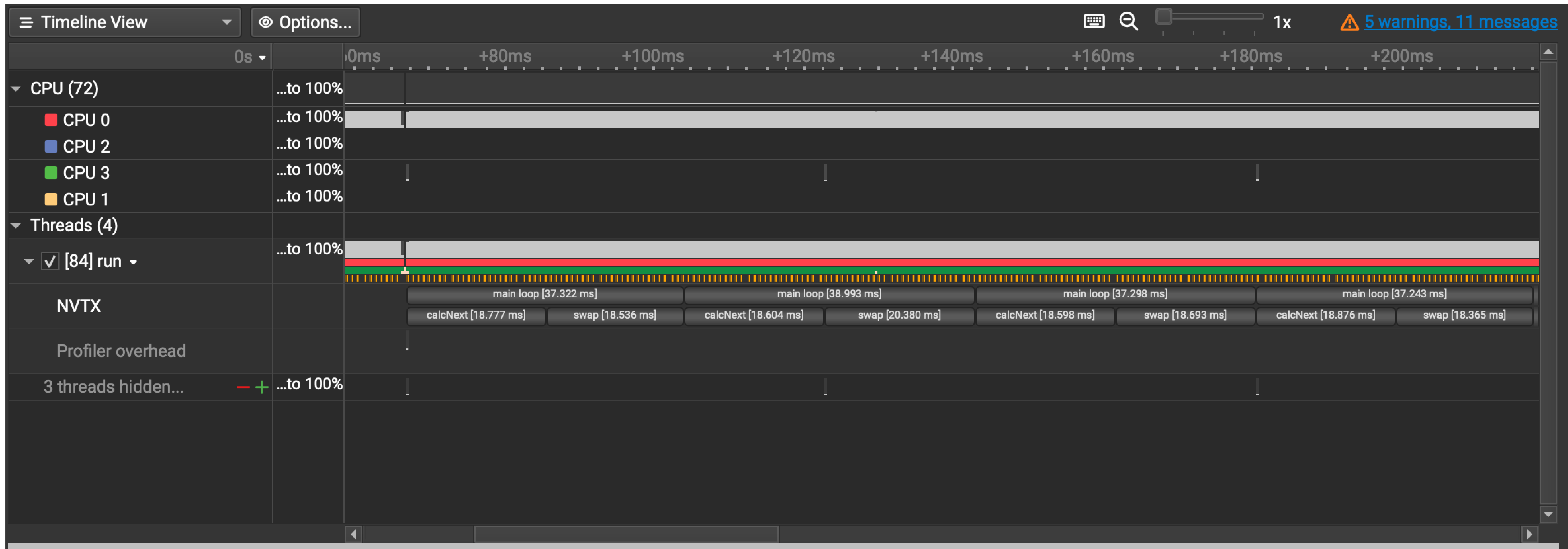
```
}
```

```
...  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)]  
                                         + A[OFFSET(j, i-1, m)]  
                                         + A[OFFSET(j-1, i, m)]  
                                         + A[OFFSET(j+1, i, m)] );  
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];  
    }  
}  
...
```

```
$ nsys profile -t cuda,nvtx,openacc,osrt \-f true \-o my_report \./my_app
```

```
...  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];  
    }  
}  
...
```

サンプルコード With NVTX のタイムライン



サンプルコード With NVTX and OpenACC Directives

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )
  call nvtxStartRange("main loop")

  call nvtxStartRange("calcNext")
  error = calcNext(A, Anew, m, n)
  call nvtxEndRange

  call nvtxStartRange("swap")
  call swap(A, Anew, m, n)
  call nvtxEndRange

  if(mod(iter,100).eq.0 ) write(*,'(i5,f10.6)'), it
  iter = iter + 1

  call nvtxEndRange
end do
```

```
...
!$acc parallel loop reduction(max:error)
do j=1,m-2
  do i=1,n-2
    Anew(i,j) = 0.25_fp_kind * ( A(i+1,j) + A(i-1,j) + &
                                A(i,j-1) + A(i,j+1) )
    error = max( error, abs(Anew(i,j)-A(i,j)) )
  end do
end do
!$acc end parallel
...
```

```
...
!$acc parallel loop
do j=1,m-2
  do i=1,n-2
    A(i,j) = Anew(i,j)
  end do
end do
!$acc end parallel
...
```

サンプルコード With NVTX and OpenACC Directives

Fortran

```
do while ( error .gt. tol .and. iter .lt. iter_max )  
  call nvtxStartRange("main loop")
```

```
  call nvtxStartRange("calcNext")  
  error = calcNext(A, Anew, m, n)  
  call nvtxEndRange
```

```
  call nvtxStartRange("swap")  
  call swap(A, Anew, m, n)  
  call nvtxEndRange
```

```
  if(mod(iter,100).eq.0 ) writ
```

```
  iter = iter + 1
```

```
  call nvtxEndRange  
end do
```

```
...  
!$acc parallel loop reduction(max:error)
```

```
do j=1,m-2  
  do i=1,n-2
```

```
$ nsys profile -t cuda,nvtx,openacc,osrt \ ( A(i+1,j ) + A(i-1,j ) + &  
-f true \ A(i ,j-1) + A(i ,j+1) )  
-o my_report \ w(i,j)-A(i,j)) )  
./my_app
```

```
!$acc end parallel
```

```
...
```

```
...  
!$acc parallel loop  
do j=1,m-2  
  do i=1,n-2  
    A(i,j) = Anew(i,j)  
  end do  
end do  
!$acc end parallel  
...
```

サンプルコード With NVTX and OpenACC Directives

C

```
while ( error > tol && iter < iter_max ) {  
    nvtxRangePush("main loop");  
  
    nvtxRangePush("calcNext");  
    error = calcNext(A, Anew, m, n);  
    nvtxRangePop();  
  
    nvtxRangePush("swap");  
    swap(A, Anew, m, n);  
    nvtxRangePop();  
  
    if(iter % 100 == 0) printf("%5d, %0.6f\n", iter,  
    iter++;  
  
    nvtxRangePop();  
}
```

```
...  
#pragma acc parallel loop reduction(max:error)  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)]  
                                         + A[OFFSET(j, i-1, m)]  
                                         + A[OFFSET(j-1, i, m)]  
                                         + A[OFFSET(j+1, i, m)] );  
  
        error = fmax( error, fabs(Anew[OFFSET(j, i, m)]  
                                - A[OFFSET(j, i, m)]));  
    }  
}  
...
```

```
...  
#pragma acc parallel loop  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];  
    }  
}  
...
```

サンプルコード With NVTX and OpenACC Directives

C

```
while ( error > tol && iter < iter_max ) {  
    nvtxRangePush("main loop");
```

```
    nvtxRangePush("calcNext");  
    error = calcNext(A, Anew, m, n);  
    nvtxRangePop();
```

```
    nvtxRangePush("swap");  
    swap(A, Anew, m, n);  
    nvtxRangePop();
```

```
    if(iter % 100 == 0) printf("iter %d error %f\n", iter, error);  
    iter++;
```

```
    nvtxRangePop();  
}
```

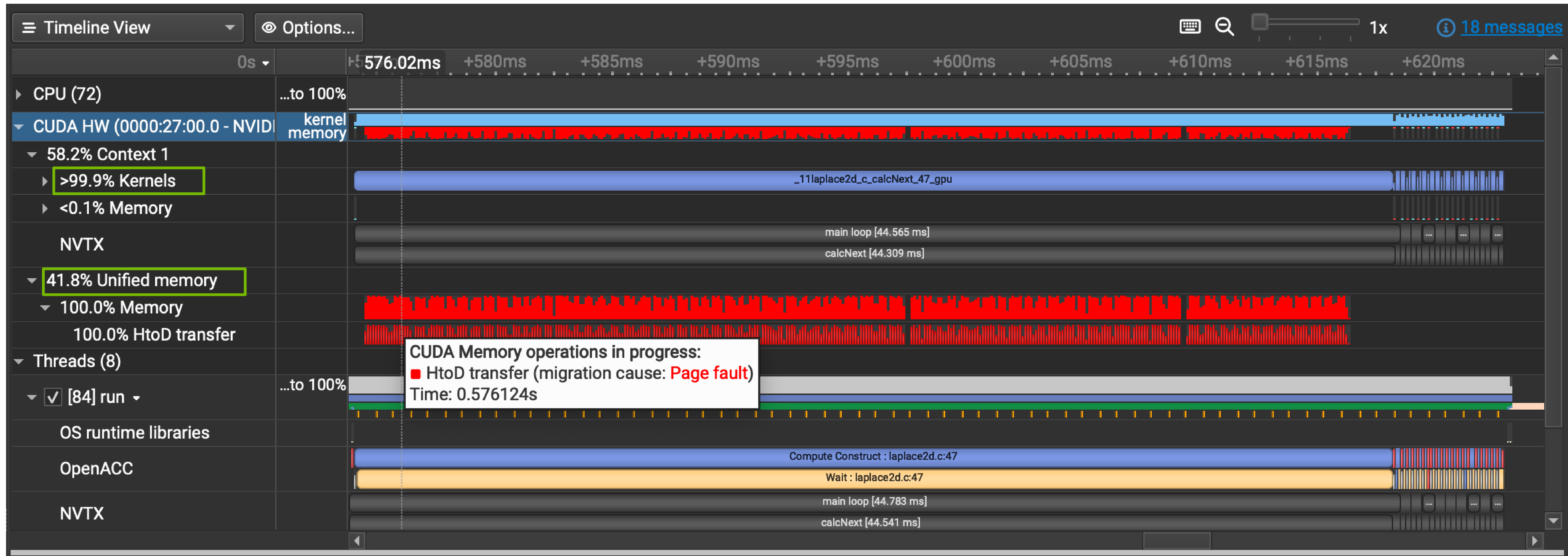
```
$ nsys profile -t cuda,nvtx,openacc,osrt  
-f true  
-o my_report  
./my_app
```

```
...  
#pragma acc parallel loop reduction(max:error)  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        \0.25 * ( A[OFFSET(j, i+1, m)]  
                + A[OFFSET(j, i-1, m)]  
                + A[OFFSET(j-1, i, m)]  
                + A[OFFSET(j+1, i, m)] );  
        error = max( error, fabs(Anew[OFFSET(j, i, m)]  
                                - A[OFFSET(j, i, m)]));  
    }  
}  
...
```

```
...  
#pragma acc parallel loop  
for( int j = 1; j < n-1; j++ ) {  
    for( int i = 1; i < m-1; i++ ) {  
        A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];  
    }  
}  
...
```

サンプルコード With NVTX and OpenACC Directives

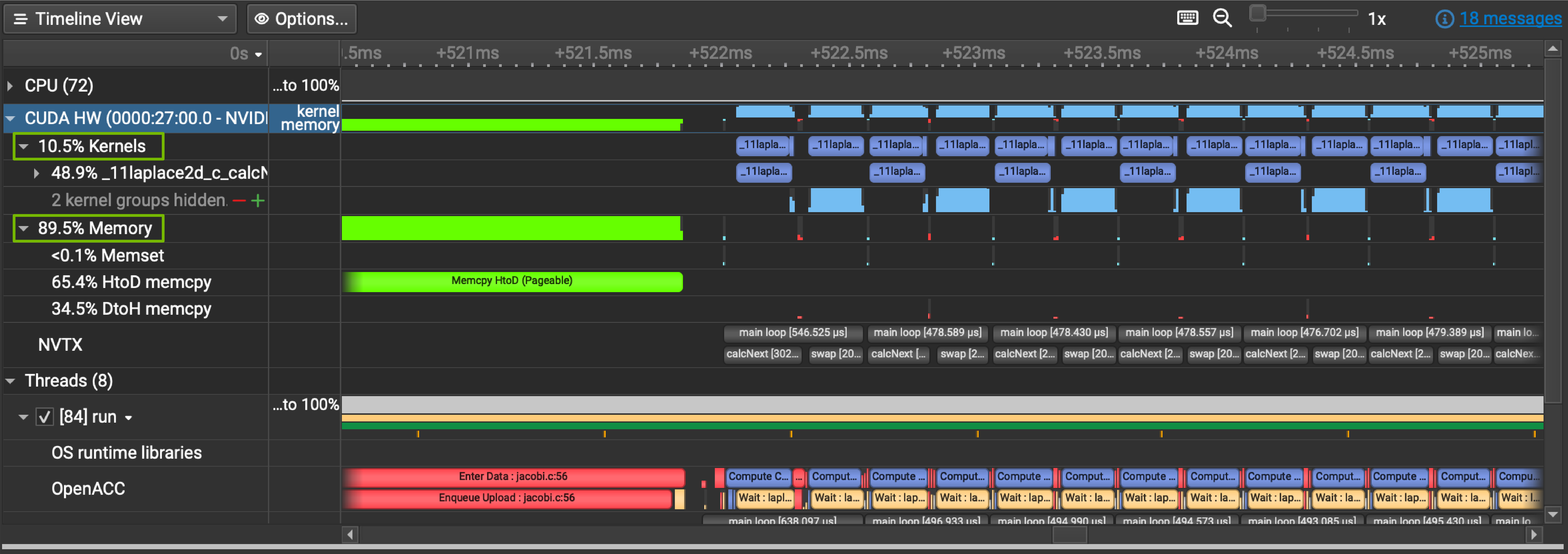
With CUDA Managed Memory



1 回目の iteration

サンプルコード With NVTX and OpenACC Directives

With Data Directive



Nsight Systems

統計情報の取得

```
$ nsys profile -t cuda,nvtx,openacc,mpi \
               --stats=true           \
               -f true                 \
               -o my_report            \
               ./my_app
```

または、*nsys-rep ファイルを nsys stats コマンドで後処理

```
$ nsys stats my_report.nsys-rep
```

[3/8] Executing 'nvtx_sum' stats report

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Style	Range
50.0	492,240,564	1,000	492,240.6	489,954.0	484,485	846,943	12,123.2	PushPop	main loop
27.3	268,612,338	1,000	268,612.3	266,471.5	262,708	593,590	10,948.2	PushPop	calcNext
22.7	222,937,020	1,000	222,937.0	222,801.0	219,211	228,587	1,472.5	PushPop	swap

[4/8] Executing 'cuda_api_sum' stats report

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
90.9	458,545,384	6,002	76,398.8	9,397.5	1,219	654,061	100,531.5	cuStreamSynchronize
3.9	19,871,245	1	19,871,245.0	19,871,245.0	19,871,245	19,871,245	0.0	cuMemHostAlloc
1.6	8,164,380	3,000	2,721.5	2,692.0	2,180	18,312	469.4	cuLaunchKernel
1.2	6,255,682	1,032	6,061.7	1,419.0	1,369	713,125	55,271.5	cuEventSynchronize
0.8	4,182,583	1	4,182,583.0	4,182,583.0	4,182,583	4,182,583	0.0	cuModuleLoadDataEx
0.5	2,497,926	1,009	2,475.6	2,428.0	2,235	6,605	280.0	cuMemcpyDtoHAsync_v2
0.3	1,747,163	1,000	1,747.2	1,711.0	1,532	6,313	218.9	cuMemsetD32Async
0.2	1,170,436	1,034	1,131.9	1,105.5	978	3,616	183.3	cuEventRecord
0.2	1,006,885	7	143,840.7	5,930.0	3,051	670,018	244,141.9	cuMemAlloc_v2
0.1	724,040	1	724,040.0	724,040.0	724,040	724,040	0.0	cuMemAllocHost_v2
0.0	106,349	18	5,908.3	5,042.0	4,531	15,614	2,682.1	cuMemcpyHtoDAsync_v2
0.0	8,105	4	2,026.3	1,667.5	575	4,195	1,536.8	cuEventCreate
0.0	1,545	3	515.0	388.0	237	920	358.8	cuCtxSetCurrent

[5/8] Executing 'cuda_gpu_kern_sum' stats report

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
48.9	209,006,029	1,000	209,006.0	208,989.0	206,397	213,373	1,118.9	laplace2d_calcnex_58_gpu
48.6	207,578,272	1,000	207,578.3	207,485.0	204,254	211,710	1,247.7	laplace2d_swap_76_gpu
2.5	10,609,245	1,000	10,609.2	10,048.0	9,504	12,736	951.0	laplace2d_calcnex_58_gpu__red

[6/8] Executing 'cuda_gpu_mem_time_sum' stats report

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
55.9	10,797,638	18	599,868.8	674,712.0	1,888	680,664	217,215.0	[CUDA memcpy HtoD]
38.0	7,347,941	1,009	7,282.4	1,920.0	1,855	702,872	58,687.8	[CUDA memcpy DtoH]
6.1	1,172,678	1,000	1,172.7	1,056.0	1,024	1,568	150.0	[CUDA memset]

[7/8] Executing 'cuda_gpu_mem_size_sum' stats report

Total (MB)	Count	Avg (MB)	Med (MB)	Min (MB)	Max (MB)	StdDev (MB)	Operation
268.436	18	14.913	16.777	0.000	16.777	5.425	[CUDA memcpy HtoD]
134.226	1,009	0.133	0.000	0.000	16.777	1.489	[CUDA memcpy DtoH]
0.008	1,000	0.000	0.000	0.000	0.000	0.000	[CUDA memset]

[8/8] Executing 'openacc_sum' stats report

Time (%)	Total Time (ns)	Num Calls	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
22.6	233,998,834	3,000	77,999.6	8,413.5	1,897	229,657	102,523.0	Wait@laplace2d.f90:58
22.3	230,826,836	1,000	230,826.8	229,934.5	226,312	416,634	6,301.2	Compute Construct@laplace2d.f90:58
21.0	217,731,630	1,000	217,731.6	217,615.5	214,092	223,364	1,431.4	Compute Construct@laplace2d.f90:76
20.8	215,468,752	2,000	107,734.4	108,030.5	1,878	218,154	105,483.4	Wait@laplace2d.f90:76
4.3	44,679,372	1	44,679,372.0	44,679,372.0	44,679,372	44,679,372	0.0	Enter Data@jacobi.f90:46
1.9	19,804,302	2,000	9,902.2	10,034.0	735	50,939	9,168.4	Exit Data@laplace2d.f90:58

プロファイリングの範囲を指定

- NVTX

- プロファイリングしたい NVTX の範囲名を指定

```
$ nsys profile -o report -f true -t cuda,openacc,nvtx,osrt \
  --capture-range=nvtx --capture-range-end=stop --nvtx-capture=my_range@* \
  -e NSYS_NVTX_PROFILER_REGISTER_ONLY=0 ./my_app
```

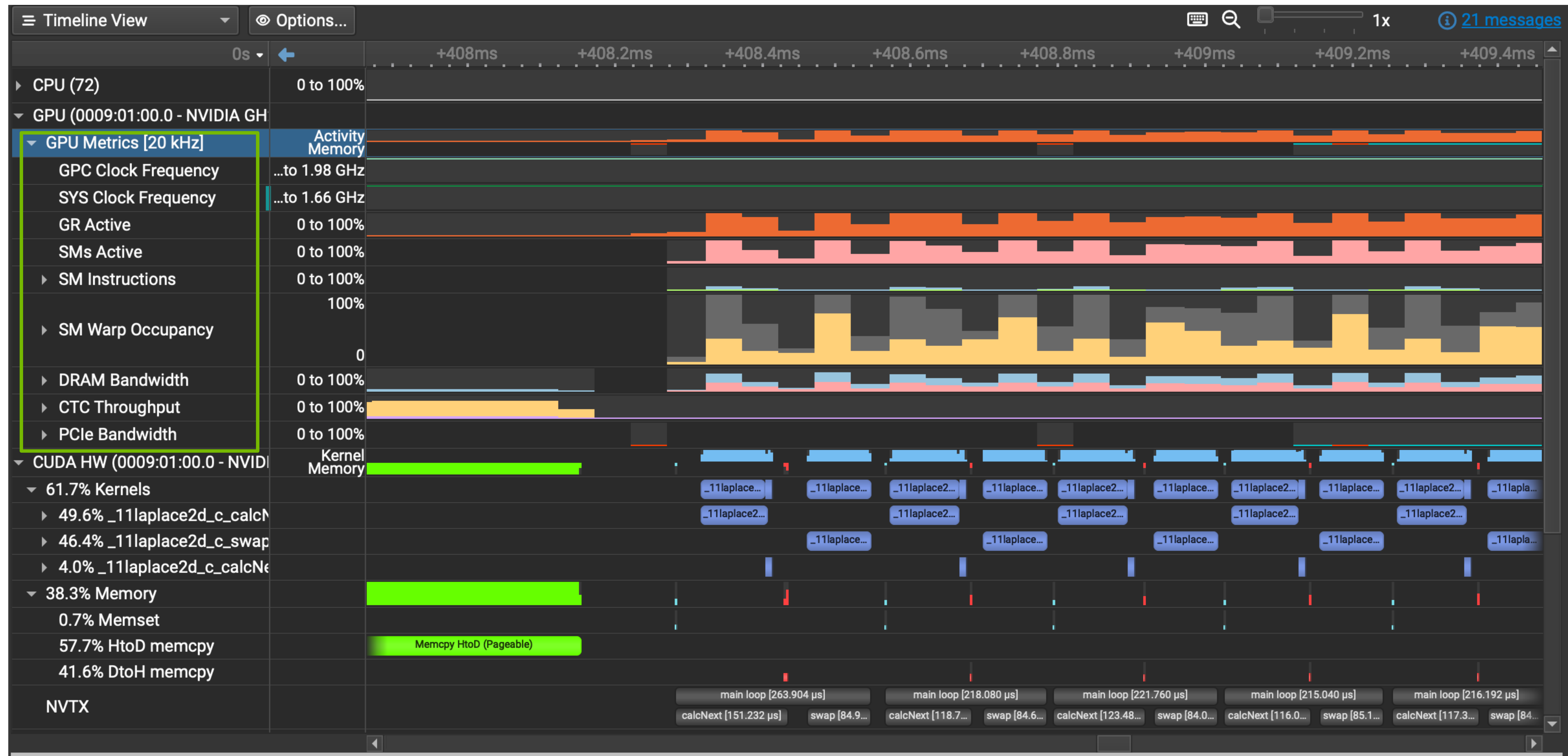
- cudaProfilerApi

- `#include <cuda_profiler_api.h>`
- プロファイリングしたい範囲を `cudaProfilerStart()`, `cudaProfilerStop()` で囲む

```
$ nsys profile -o report -f true -t cuda,openacc,nvtx,osrt \
  --capture-range=cudaProfilerApi --capture-range-end=stop ./my_app
```

各種 GPU メトリクスの取得

- `--gpu-metrics-device=[GPU ID, help, all, none]` を指定することで、以下のような GPU メトリクスの取得が可能 (SMs activity, instructions issued, warp occupancy, DRAM bandwidth, PCIe bandwidth, etc...)



各種 CPU メトリクスの取得

- `--cpu-core-events=help` で CPU コアイベント、`--cpu-core-metrics=help` で CPU コアメトリクスを取得可能

```
$ nsys profile --cpu-core-events=help
```

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

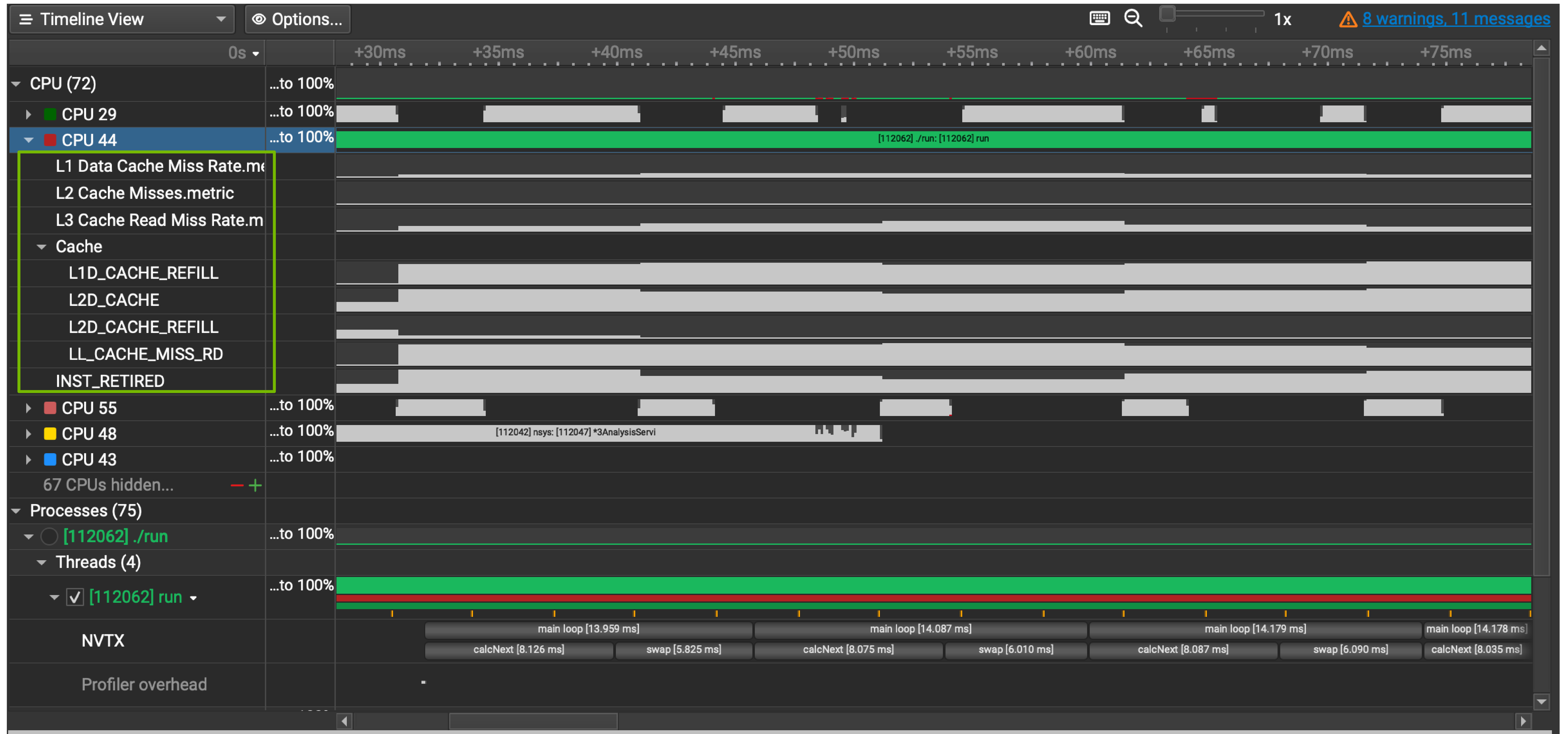
Miyabi で取得可能な CPU コアイベント

'64' OP_RETIRED (Micro-operation architecturally executed.)
'65' OP_SPEC (Micro-operation speculatively executed.)
'66' STALL_SLOT (No operation sent for execution on a slot.)
'69' FP_DP_SPEC (Double-precision floating-point operation speculatively executed.)
'70' FP_SCALE_OPS_SPEC (Scalable floating-point element operations speculatively executed.)
'71' FP_FIXED_OPS_SPEC (Non-scalable floating-point element operations speculatively executed.)
'72' ASE_INST_SPEC (Advanced SIMD operations speculatively executed.)
'73' SVE_INST_SPEC (SVE operations speculatively executed.)
'74' FP_SP_SPEC (Single-precision floating-point operation speculatively executed.)
'75' L1D_CACHE_RD (L1 data cache access, read.)
'76' L1D_CACHE_WR (L1 data cache access, write.)
'77' L1D_CACHE (L1 data cache access.)
'78' L1D_CACHE_REFILL (L1 data cache refill.)
'79' L1D_CACHE_WB (L1 data cache Write-Back.)
'80' L2D_CACHE_REFILL (L2 cache refill.)
'81' L2D_CACHE_WB (L2 cache write-back.)
'82' L3D_CACHE_REFILL (Attributable L3 cache refill.)
'83' STALL_BACKEND (No operation issued because of the backend.)
'84' STALL_FRONTEND (No operation issued because of the frontend.)
'85' BR_MIS_PRED (Mispredicted or not predicted branch speculatively executed.)
'86' LD_SPEC (Operation speculatively executed, load.)
'87' INST_SPEC (Operation speculatively executed.)
'88' ST_SPEC (Operation speculatively executed, store.)
'89' BR_IMMED_SPEC (Branch speculatively executed, immediate branch.)
'90' BR_INDIRECT_SPEC (Branch speculatively executed, indirect branch.)
'91' DP_SPEC (Operation speculatively executed, integer data-processing.)
'92' VFP_SPEC (Operation speculatively executed, floating-point instruction.)
'93' ISB_SPEC (Barrier speculatively executed, ISB.)
'94' DSB_SPEC (Barrier speculatively executed, DSB.)
'95' DMB_SPEC (Barrier speculatively executed, DMB.)
'96' CRYPTO_SPEC (Operation speculatively executed, Cryptographic instruction.)
'97' FP_HP_SPEC (Half-precision floating-point operation speculatively executed.)
'98' SVE_PRED_FULL_SPEC (SVE predicated operations speculatively executed with all active predicates.)
'99' SVE_PRED_PARTIAL_SPEC (SVE predicated operations speculatively executed with partially active predicates.)
'100' SVE_PRED_EMPTY_SPEC (SVE predicated operations with no active predicates speculatively executed.)
'101' L1I_CACHE_REFILL (L1 instruction cache refill.)
'102' L1I_CACHE (L1 instruction cache access or Macro-op (MOP) cache access.)
'103' L2D_CACHE (L2 cache access.)
'104' LL_CACHE_RD (Last level cache access, read.)
'105' LL_CACHE_MISS_RD (Last level cache miss, read.)
'106' L1D_TLB_REFILL (L1 data TLB refill.)
'107' L1D_TLB (Level 1 data TLB access.)
'108' L1I_TLB_REFILL (L1 instruction TLB refill.)
'109' L1I_TLB (Level 1 instruction TLB access.)
'110' BR_MIS_PRED_RETIRED (Instruction architecturally executed, mispredicted branch.)
'111' BR_RETIRED (Instruction architecturally executed, branch.)
'112' CPU_CYCLES (CPU cycles.)
'113' INST_RETIRED (Instruction architecturally executed.)
'114' ASE_SPEC (Operation speculatively executed, Advanced SIMD instruction.)

Miyabi で取得可能な CPU コアメトリクス

```
'0' IPC (Instructions Retired per CPU Cycle)
    INST_RETIRED / CPU_CYCLES
'1' Retiring (Percentage of total slots that are retired operations)
    100*(OP_RETIRED / OP_SPEC) * (1 - (STALL_SLOT / (CPU_CYCLES*8)))
'2' Backend Stalls (Fraction of total cycles that were stalled because of resource constraints in the processor backend)
    STALL_BACKEND / CPU_CYCLES
'3' Frontend Stalls (Fraction of total cycles that were stalled because of resource constraints in the processor frontend)
    STALL_FRONTEND / CPU_CYCLES
'6' SVE FLOPS (Floating point operations per second in any precision performed by the SVE instructions)
    FP_SCALE_OPS_SPEC / TIME
'7' Non-SVE FLOPS (Floating point operations per second in any precision performed by an instruction that is not an SVE instruction)
    FP_FIXED_OPS_SPEC / TIME
'8' FLOPS (Floating point operations per second in any precision performed by any instruction)
    (FP_SCALE_OPS_SPEC + FP_FIXED_OPS_SPEC) / TIME
'9' Load Percentage (Fraction of total instructions that were speculatively executed because of the load instructions)
    LD_SPEC / INST_SPEC
'10'    Store Percentage (Fraction of total instructions speculatively executed because of the store instructions)
    ST_SPEC / INST_SPEC
'11'    Branch Percentage (Fraction of total instructions that were speculatively executed because of the branch instructions)
    (BR_IMMED_SPEC + BR_INDIRECT_SPEC) / INST_SPEC
'12'    Scalar Integer Percentage (Fraction of total instructions that were speculatively executed because of the scalar integer instructions)
    DP_SPEC / INST_SPEC
'13'    Scalar Floating Point Percentage (Fraction of total instructions speculatively executed because of the scalar floating point instructions)
    VFP_SPEC / INST_SPEC
'14'    Synchronization Percentage (Fraction of total instructions that were speculatively executed because of the synchronization instructions)
    (ISB_SPEC + DSB_SPEC + DMB_SPEC) / INST_SPEC
'15'    Crypto Percentage (Fraction of total instructions that were speculatively executed because of the crypto instructions)
    CRYPTO_SPEC / INST_SPEC
'16'    SVE SIMD Percentage (Fraction of total instructions that were speculatively executed because of the integer or floating point SVE SIMD instructions)
    SVE_INST_SPEC / INST_SPEC
'17'    NEON SIMD Percentage (Fraction of total instructions that were speculatively executed because of the integer or floating point NEON SIMD instructions)
    ASE_INST_SPEC / INST_SPEC
'18'    SIMD Percentage (Fraction of total instructions that were speculatively executed because of the integer or floating point vector/SIMD instructions)
    (SVE_INST_SPEC + ASE_INST_SPEC) / INST_SPEC
'19'    FP16 Percentage (Fraction of total instructions that were speculatively executed because of the half-precision floating point instructions)
    FP_HP_SPEC / INST_SPEC
'20'    FP32 Percentage (Fraction of total instructions that were speculatively executed because of the single-precision floating point instructions)
    FP_SP_SPEC / INST_SPEC
'21'    FP64 Percentage (Fraction of total instructions that were speculatively executed because of the double-precision floating point instructions)
    FP_DP_SPEC / INST_SPEC
'22'    Full SVE Instructions (Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions with all active predicates)
    SVE_PRED_FULL_SPEC / INST_SPEC
'23'    Partial SVE Instructions (Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions in which at least one element is FALSE)
    SVE_PRED_PARTIAL_SPEC / INST_SPEC
'24'    Empty SVE Instructions (Fraction of total instructions that were speculatively executed because of the SVE SIMD instructions with no active predicate)
    SVE_PRED_EMPTY_SPEC / INST_SPEC
'25'    L1 Data Cache Misses (Fraction of total level 1 data cache read or write accesses that miss)
    L1D_CACHE_REFILL / L1D_CACHE
'26'    L1 Data Cache Miss Rate (Count of level 1 data cache read or write accesses that miss per kilo-instructions executed)
    L1D_CACHE_REFILL / (INST_RETIRED / 1000)
'27'    L1 Instruction Cache Misses (Fraction of total level 1 instruction cache accesses that miss)
    L1I_CACHE_REFILL / L1I_CACHE
'28'    L1 Instruction Cache Miss Rate (Count of level 1 instruction cache accesses missed per kilo-instructions executed)
    L1I_CACHE_REFILL / (INST_RETIRED / 1000)
'29'    L2 Cache Misses (Fraction of total level 2 cache read or write accesses that miss)
    L2D_CACHE_REFILL / L2D_CACHE
'30'    L2 Cache Miss Rate (Count of level 2 cache read or write accesses that miss per kilo-instructions executed)
    L2D_CACHE_REFILL / (INST_RETIRED / 1000)
'31'    L3 Cache Read Hits (Fraction of L3 cache read accesses that hit)
    (LL_CACHE_RD - LL_CACHE_MISS_RD) / LL_CACHE_RD
'32'    L3 Cache Read Misses (Fraction of L3 cache read accesses that miss)
    LL_CACHE_MISS_RD / LL_CACHE_RD
'33'    L3 Cache Read Miss Rate (Count of L3 cache read accesses missed per kilo-instructions executed)
    LL_CACHE_MISS_RD / (INST_RETIRED / 1000)
'34'    L1 Data TLB Misses (Fraction of total level 1 data TLB accesses that miss)
    L1D_TLB_REFILL / L1D_TLB
'35'    L1 Data TLB Miss Rate (Count of level 1 data TLB accesses that miss per kilo-instructions executed)
    L1D_TLB_REFILL / (INST_RETIRED / 1000)
'36'    L1 Instruction TLB Misses (Fraction of total level 1 instruction TLB accesses that miss)
    L1I_TLB_REFILL / L1I_TLB
'37'    L1 Instruction TLB Miss Rate (Count of level 1 instruction TLB accesses that miss per kilo-instructions executed)
    L1I_TLB_REFILL / (INST_RETIRED / 1000)
'38'    Branch Mispredictions (Fraction of architecturally executed branches that were mispredicted)
    BR_MIS_PRED_RETIRED / BR_RETIRED
```

CPU コアメトリクス



MPI コードのプロファイリング

mpirun の中から nsys を呼ぶ

```
$ mpirun -np NUM_PROCESSES nsys profile -t cuda,nvtx,openacc,mpi \
--stats=true \
-f true \
-o my_report_%q{OMPI_COMM_WORLD_RANK} \
./my_app
```

- -t
 - cuda: CUDA
 - nvtx: NVIDIA Tools Extension
 - openacc: OpenACC
 - mpi: MPI (OpenMPI と MPICH をサポート)
- --stats=true
 - プロファイル結果のサマリーを標準出力に表示
- -f
 - 結果ファイルの上書き可否 true | false
- -o
 - 結果ファイル名
- %q{OMPI_COMM_WORLD_RANK}
 - MPI ランクに置き換わる (ランク数分だけ .nsys-rep ファイルが書き出される)

MPI コードのプロファイリング

mpirun の中から nsys を呼んで Rank0 のみプロファイルを取得

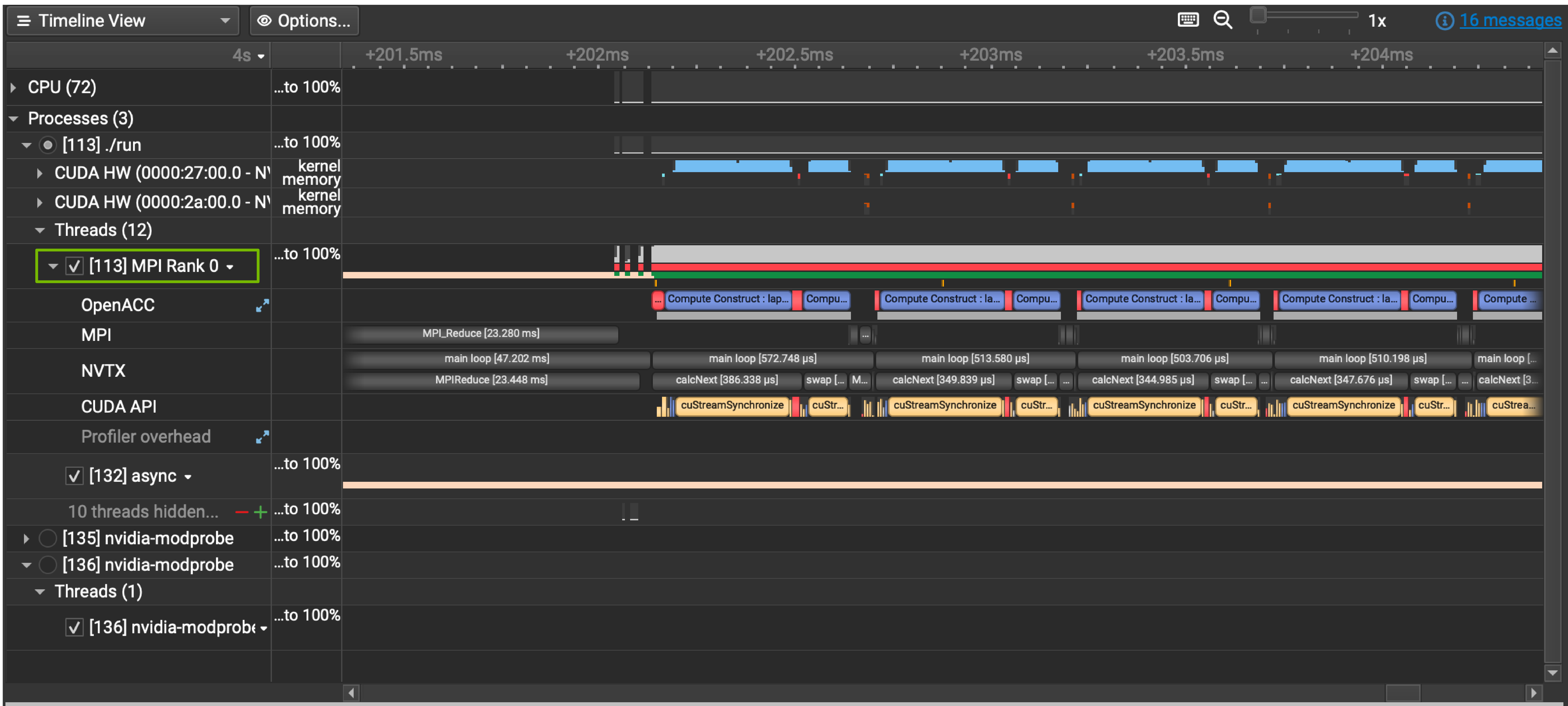
```
$ mpirun -np NUM_PROCESSES ./wrap.sh ./my_app
```

(例) wrap.sh

```
#!/bin/bash
if [[ $OMPI_COMM_WORLD_RANK == 0 ]]; then
    nsys profile -t cuda,nvtx,openacc,mpi --stats=true -f true -o my_report $@
else
    $@
fi
```

MPI コードのプロファイリング

mpirun の中から nsys を呼んで Rank0 のみプロファイルを取得



MPI コードのプロファイリング

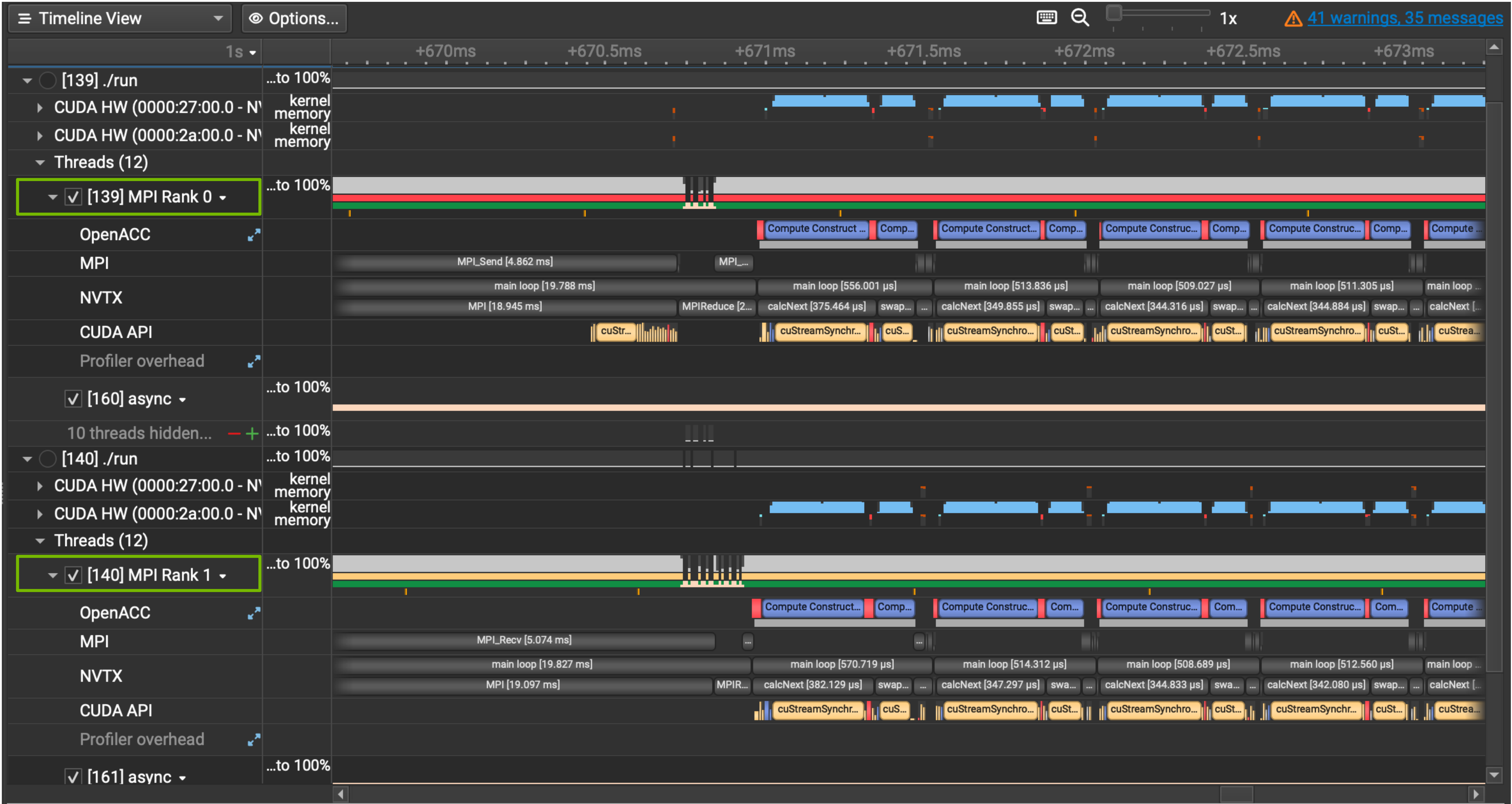
mpirun の外で nsys を呼ぶ

```
$ nsys profile -t cuda,nvtx,openacc,mpi \
               --stats=true             \
               -f true                   \
               -o my_report              \
               mpirun -np NUM_PROCESSES ./my_app
```

- 全ランクの情報が 1 ファイルに集約
- 単一ノード内での並列処理でのみ利用可能

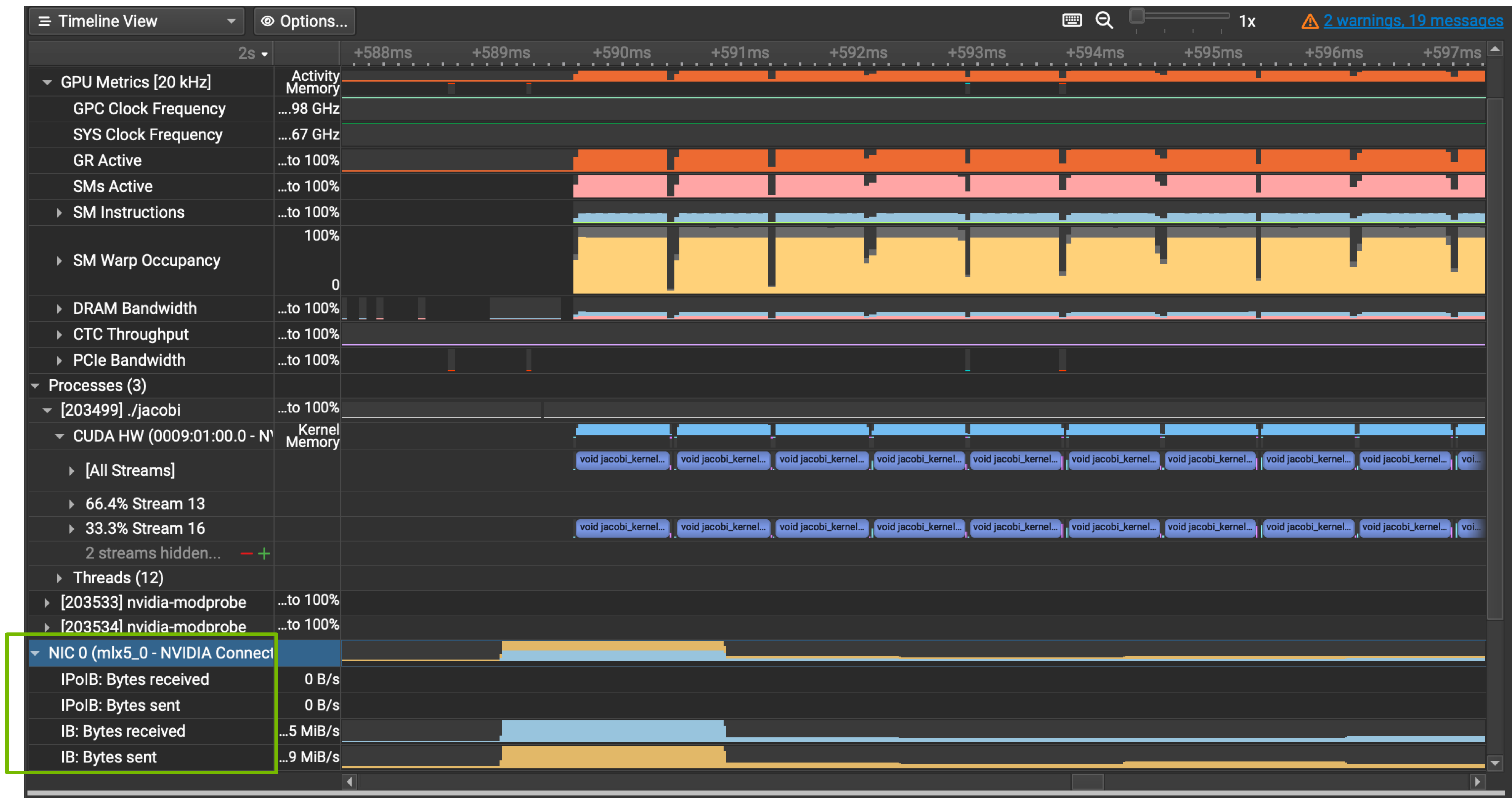
MPI コードのプロファイリング

mpirun の外で nsys を呼ぶ



NVIDIA Networking NIC Metrics Sampling

- `--nic-metrics=true`
- Correlate with expected network traffic and other system activities





Agenda

- NVIDIA GH200 Hardware Overview

- NVIDIA GH200 Programming Model Overview

- NVIDIA Developer Tools

- Hands-on (Nsight Systems)

