# GPU Programming with OpenMP Part 4: Tools and advanced topics

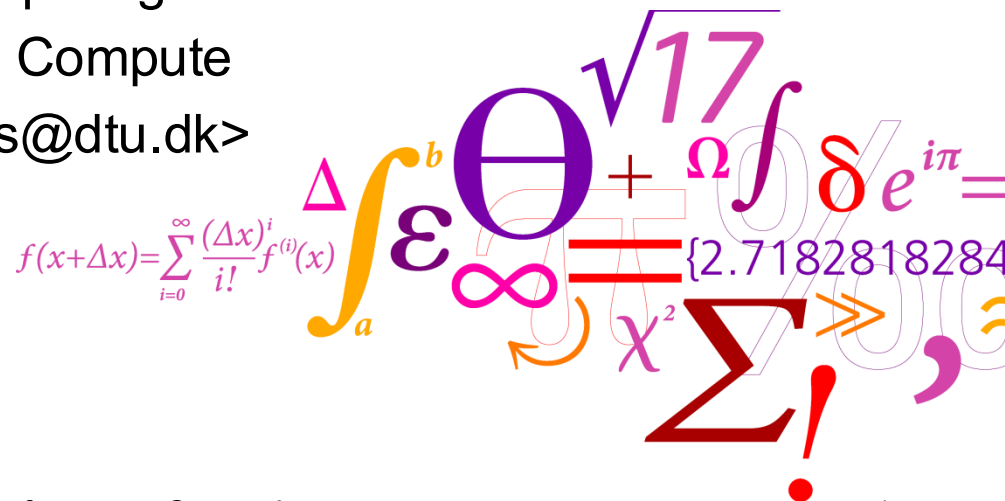Hans Henrik Brandenborg Sørensen

DTU Computing Center

DTU Compute

<hhbs@dtu.dk>

# Overview

- **Nsight profiling tools**
  - ❑ Nsight Systems
  - ❑ Nsight Compute
  - ❑ Using the command line interface
  - ❑ Running the profiler in batch mode

- **Asynchronous offloading**
  - ❑ `nowait` **clause**
  - ❑ `depend` **clause**

- **Pinned host memory**

- **Multi-GPU**

# Nsight profiling tools

Get started with NVIDIA Nsight Developer Tools:
https://www.youtube.com/playlist?list=PL5B692fm6--ukF8S7uI5NmceZhXLRv_IR

# Nsight profiling tools

- **Note**: To open displays for the user interfaces, you need to use X-window forwarding when you enter the interactive session:

```
$ hpcintrogpush -X
```

- Before you use the profiler, always load nvhpc:

```
$ module load nvhpc/25.11
```

- **Note**: Only one "profiler context" per GPU is allowed, so you get an error if another user is currently profiling on the same GPU.

# Nsight systems

- `nsys / nsys-ui`

  - ❑ You may generate a timeline for your application that shows (CUDA) API functions and offload execution

  - ❑ First you need to generate a profile report using the command line tool `nsys`, **e.g.;**

  ```
  $ nsys profile matmult_f.nvc++ \
  mnk_offload 2048 2048 2048
  ```
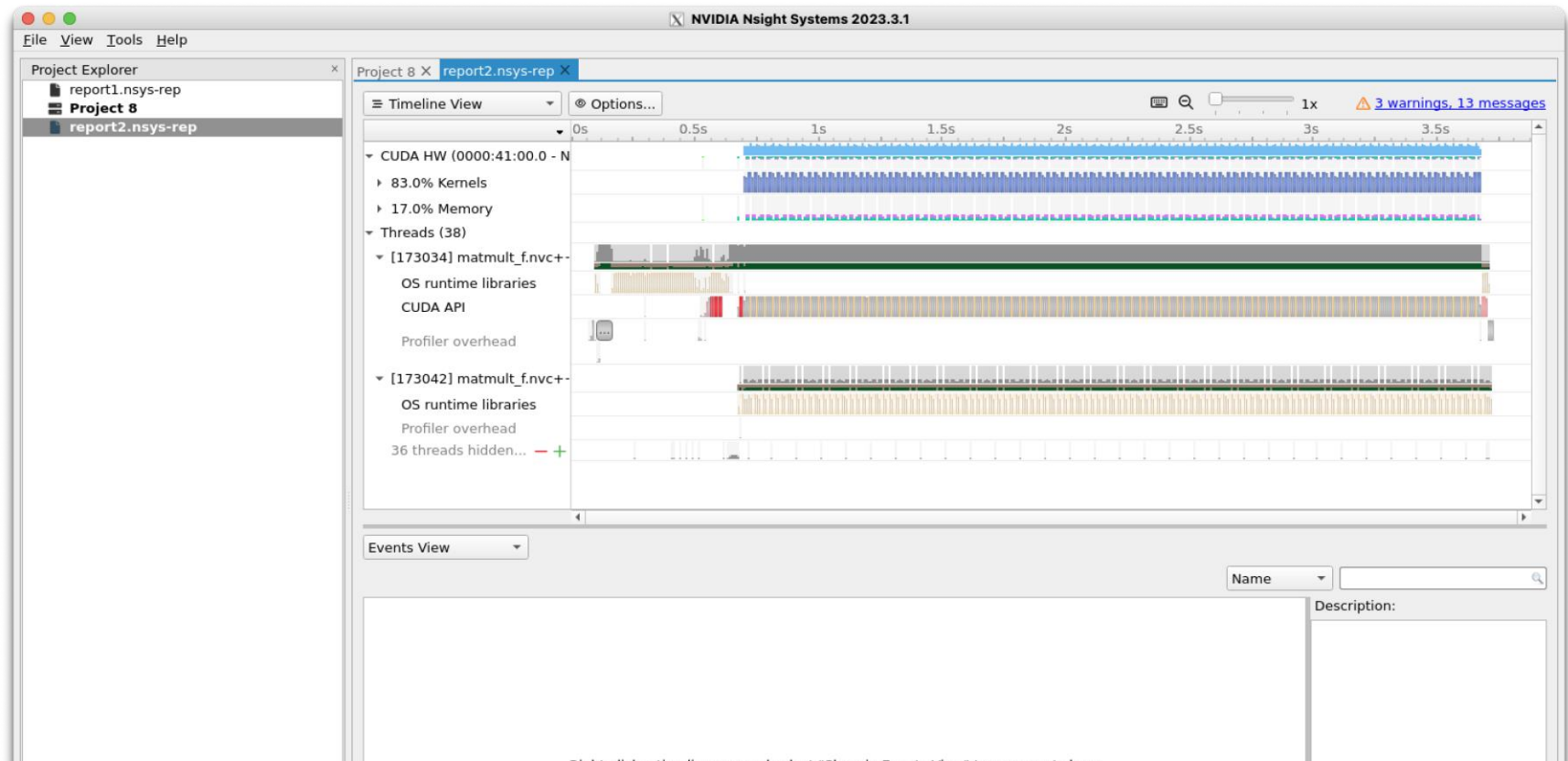
  - ❑ Generated file: `report1.nsys-rep`

# Nsight systems

- Now start the Nsight Systems user interface:

```
$ nsys-ui &
```

- Window opens – see next slide
- Open the file `report1.nsys-rep`. Click close on errors and warnings (we ignore these for now)!
- If you maximize the window you see the timeline for the driver running the `mnk_offload` function

# Nsight systems



❑ Here the first 0.8 secs correspond to the driver setup and the remaining time shows a large number of mnk_offload executions (as many as the driver can do within 3 secs)

# Nsight compute

- `ncu / ncu-ui`
  - ❑ The Nsight profiler can be used via a user interface that has customizable metric collection and presentations (tables, charts, etc.)
  - ❑ Built-in rules expertise and suggestions
- ## Standalone, IDE Integration, Remote Targets
  - ❑ OS: Linux (x86, Power, Tegra, Arm SBSA), Windows, MacOSX (host only)
  - ❑ GPUs: Volta, Turing, Ampere, Hopper (NVIDIA only!)
- https://developer.nvidia.com/nsight-compute

# Command line interface

- `ncu`

    - The simplest way to get quick textual performance information for your offload executions

    - ~20 specific output sections of interest available

    ```
    $ ncu --list-sections
    ```

    - Default: SpeedOfLight, LaunchStatistics, Occupancy

- **For example you can profile the driver running version `mnk_offload` in the matmult assignment**

```
$ TMPDIR=. MATMULT_COMPARE=0 MFLOPS_MAX_IT=1 \
ncu ./matmult_f.nvc++ mnk_offload 2048 2048 \
2048
```

```
$ TMPDIR=. MFLOPS_MAX_IT=1 ncu ./matmult_f.nvc++ mnk_offload 2048 2048 2048
==PROF== Connected to process 27597 (~/HPC_course_02614_OpenMP/src/assign3_matmult/matmult_f.nvc++)
==PROF== Profiling "nvkernel_matmult_mnk_offload_..." - 1: 0%....50%....100% - 11 passes
   98304.000   6528.841 300 # matmult_mnk_offload

Section: GPU Speed Of Light Throughput
    ---------------------------------------------------------------------- ------------- -----------------------------
    DRAM Frequency                                                         cycle/nsecond                         1.22
    SM Frequency                                                           cycle/usecond                       765.65
    Elapsed Cycles                                                         cycle                           19,854,813
    Memory [%]                                                             %                                   107.69
    DRAM Throughput                                                        %                                    48.55
    Duration                                                               msecond                              25.93
    L1/TEX Cache Throughput                                                %                                    50.59
    L2 Cache Throughput                                                    %                                   107.69
    SM Active Cycles                                                       cycle                        19,798,035.23
    Compute (SM) [%]                                                       %                                    25.05
    ---------------------------------------------------------------------- ------------- -----------------------------
    INF   The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To
          further improve performance, work will likely need to be shifted from the most utilized to another unit.
          Start by analyzing workloads in the Memory Workload Analysis section.

    Section: Launch Statistics
    ---------------------------------------------------------------------- ------------- -----------------------------
    Block Size                                                                                                    128
    Function Cache Configuration                                                               cudaFuncCachePreferNone
    Grid Size                                                                                                  32,768
    Registers Per Thread                                                   register/thread                         96
    Shared Memory Configuration Size                                       Kbyte                                16.38
    Driver Shared Memory Per Block                                         Kbyte/block                           1.02
    Dynamic Shared Memory Per Block                                        byte/block                               0
    Static Shared Memory Per Block                                         byte/block                               0
    Threads                                                                thread                           4,194,304
    Waves Per SM                                                                                                60.68
    ---------------------------------------------------------------------- ------------- -----------------------------

    Section: Occupancy
    ---------------------------------------------------------------------- ------------- -----------------------------
    Block Limit SM                                                         block                                   32
    Block Limit Registers                                                  block                                    5
    Block Limit Shared Mem                                                 block                                  164
    Block Limit Warps                                                      block                                   16
    Theoretical Active Warps per SM                                        warp                                    20
    Theoretical Occupancy                                                  %                                    31.25
    Achieved Occupancy                                                     %                                    30.96
    Achieved Active Warps Per SM                                           warp                                 19.81
    ---------------------------------------------------------------------- ------------- -----------------------------
```

# Command line interface

- **`--section MemoryWorkloadAnalysis`**
  - ❑ Outputs the achieved memory bandwidth etc.

```
$ MFLOPS_MAX_IT=0 nv-nsight-cu-cli --section MemoryWorkloadAnalysis ./matmult_f.nvc++ mnk_offload 2048 2048 2048
==PROF== Connected to process 16739 (~/HPC_course_02614_OpenMP/src/assign3_matmult/matmult_f.nvc++)
==PROF== Profiling "nvkernel_matmult_mnk_offload_..." - 0: 0%....50%....100% - 11 passes
  98304.000   5985.350 300 # matmult_mnk_offload
==PROF== Disconnected from process 16739
[16739] matmult_f.nvc++@127.0.0.1
  nvkernel_matmult_mnk_offload_F1L115_14, 2023-Jan-03 15:15:02, Context 1, Stream 14
    Section: Memory Workload Analysis
    --------------------------------------------------------------- ------------- -----------------------------
    Memory Throughput                                               Gbyte/second                         733.82
    Mem Busy                                                        %                                    107.41
    Max Bandwidth                                                   %                                     71.43
    L1/TEX Hit Rate                                                 %                                     10.43
    L2 Compression Success Rate                                     %                                          0
    L2 Compression Ratio                                                                                       0
    L2 Hit Rate                                                     %                                     65.11
    Mem Pipes Busy                                                  %                                     24.73
    --------------------------------------------------------------- ------------- -----------------------------
```

- **Advanced options**
  - ❑ `--set detailed` – more details and recommendations
  - ❑ `--target-processes all` – child processes are also profiled

# Nsight compute
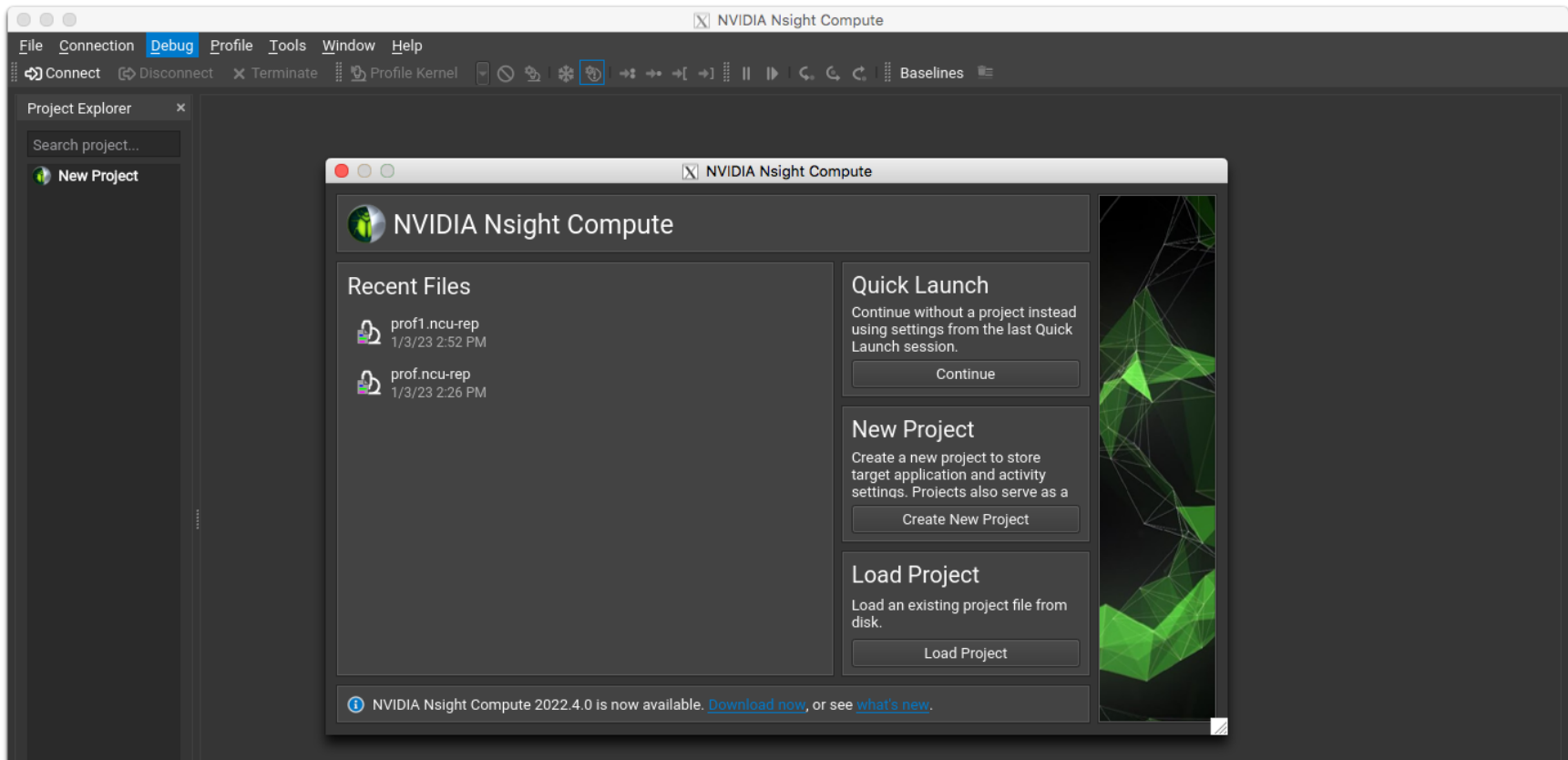


❑ For the easiest start, you can click on "Continue" under "Quick Launch". (Alternatively, you can create a project by selecting the "Create New Project" button under "New Project".) Next, a profiling configuration window should open ("Connect to process").

# Nsight compute
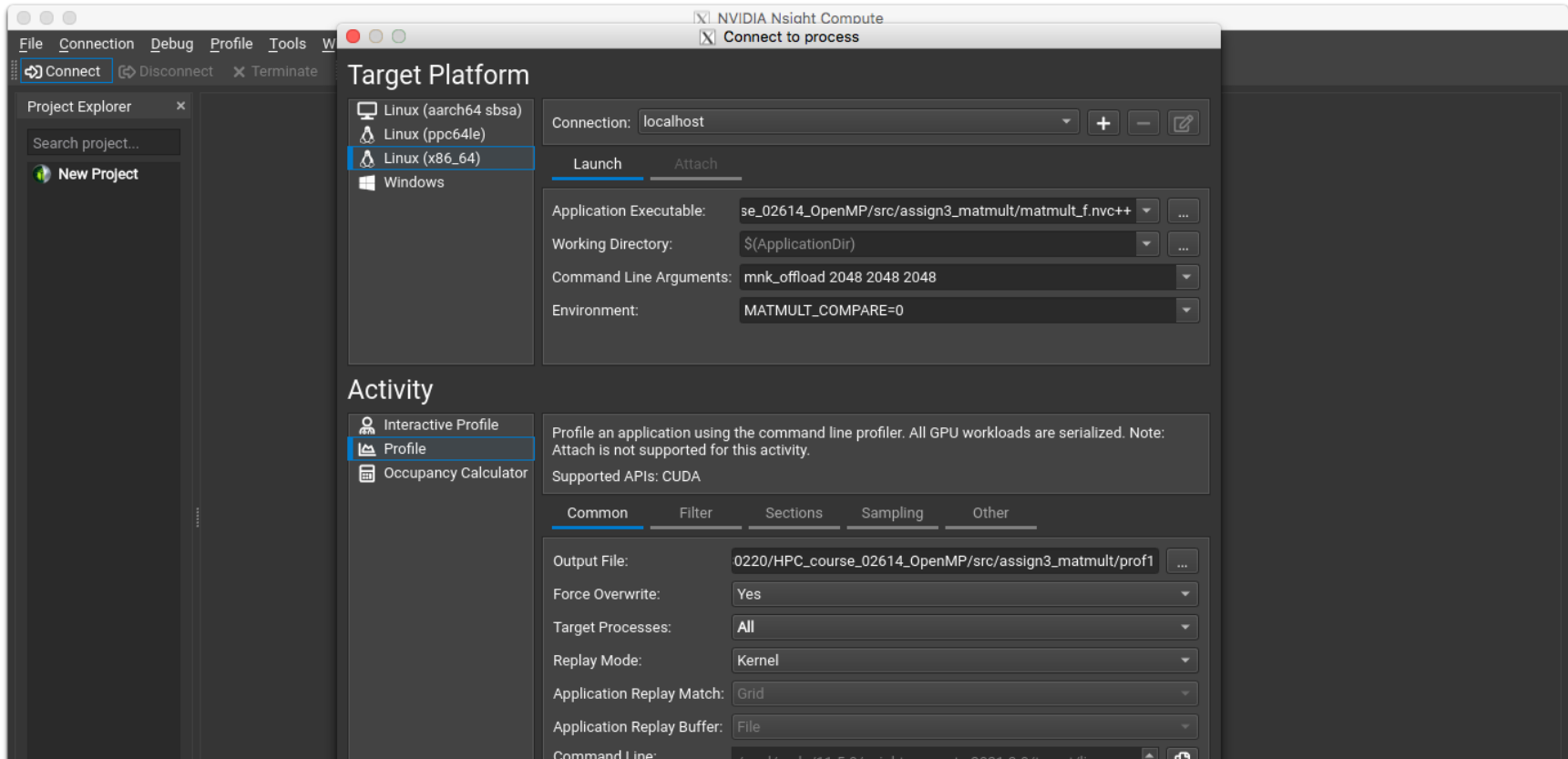


❑ Here you need to enter the path and name of the executable to be profiled ("Application Executable") and the file name, where the profiler can store the metric results ("Output File").

# Nsight compute
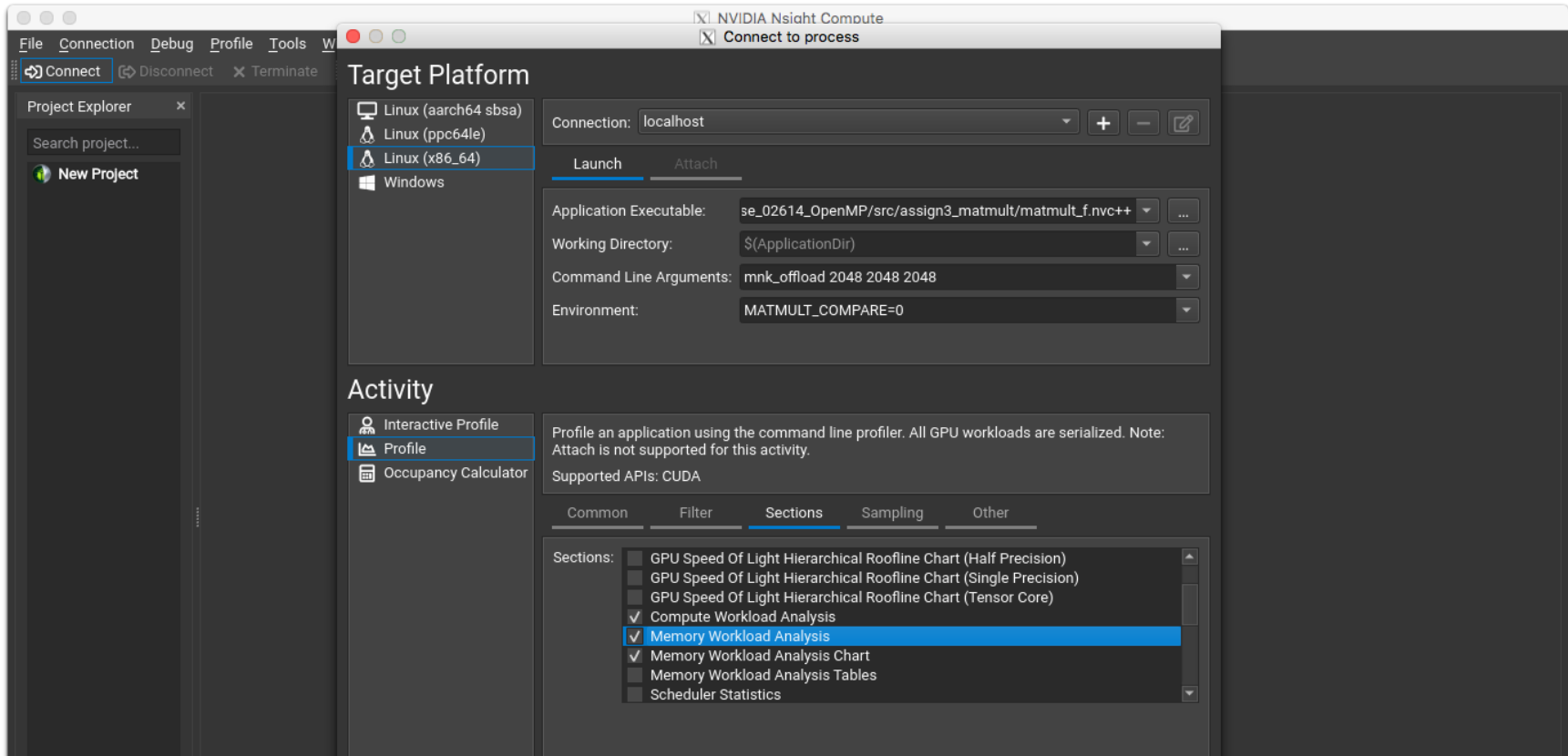


❑ In the "Activity" part, when you click the "Sections" tab, you can also select specific profiling sections you are interested in. You may also enter a comma-separated list of metric names in the "Other" tab. Clicking the "Launch" button will start the profiling of the kernel and it will run several times in order to collect all the requested metrics.

# Nsight compute
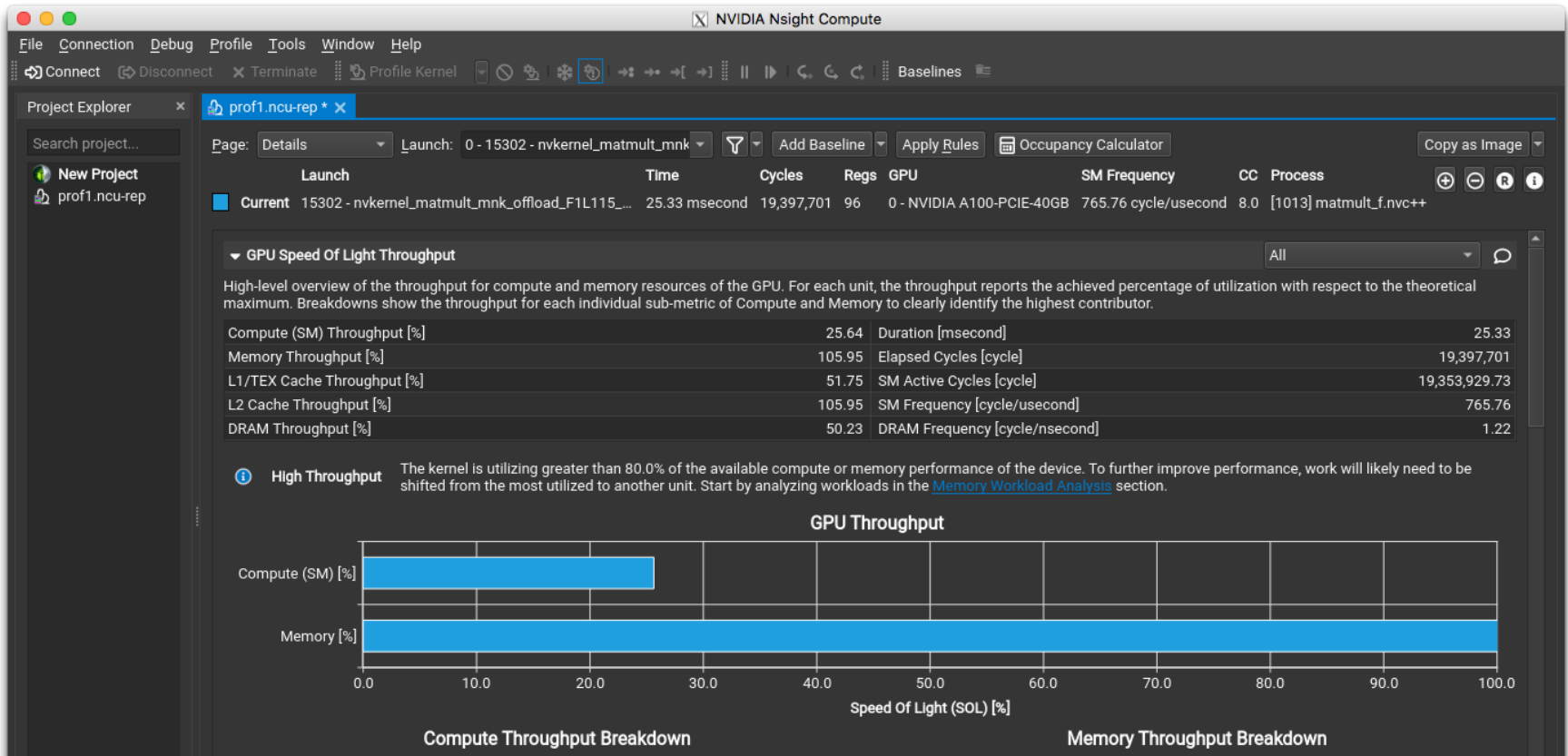


❑ When done the window will show the profiling results section by section (typically the "GPU Speed of Light" section at the top, if selected). The "GPU Speed of Light" section outputs, e.g., the SM utilization and the Memory utilization, which indicates whether the kernel is compute or memory bound.

# Nsight compute
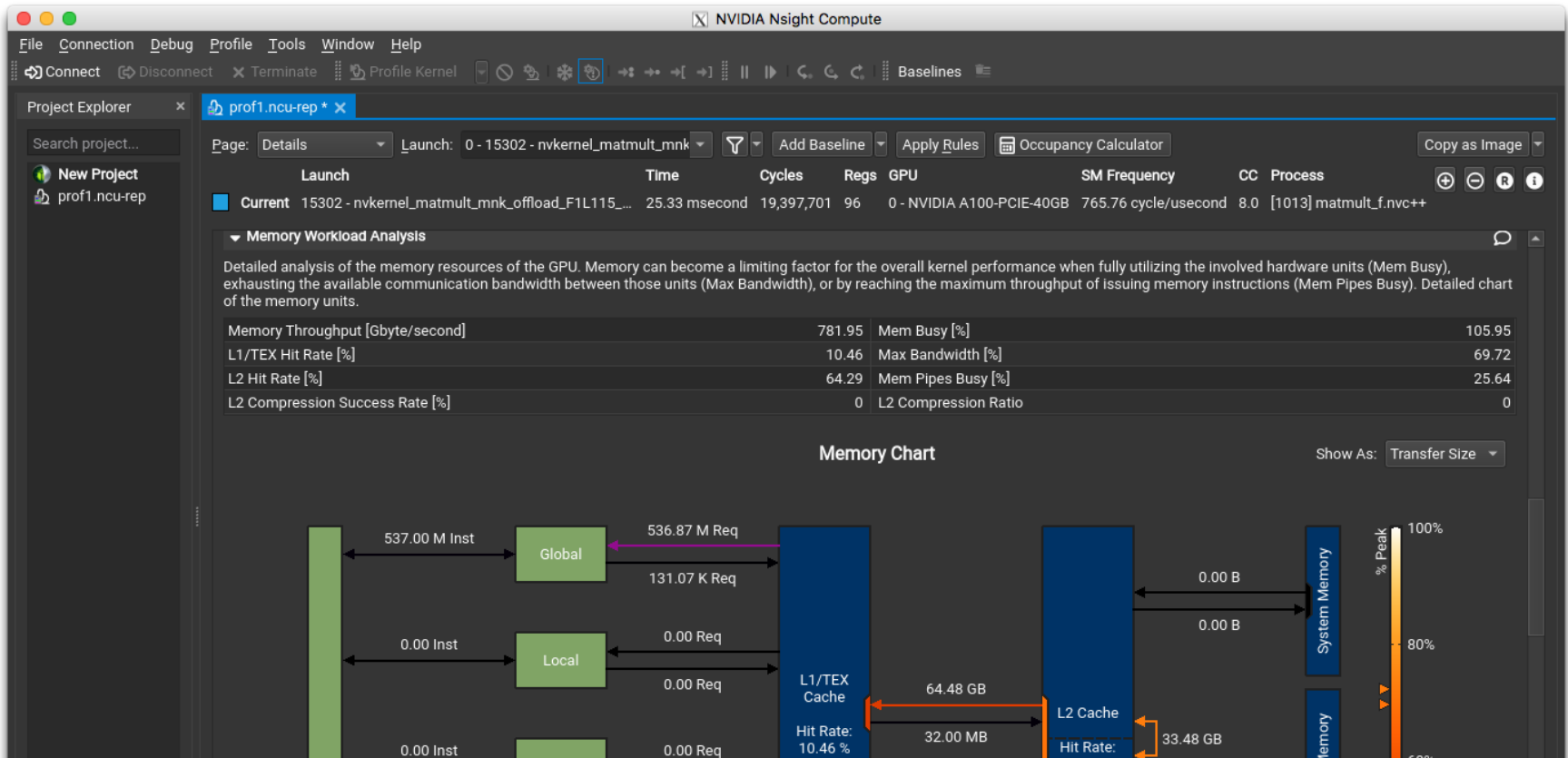


❏ You can use the slider to the right to move down through the sections that you have selected for profiling. For example you can click the "Memory Workload Analysis" title to expand details and see the Memory Chart (if selected). Each section also offers an detailed analysis that gives advice and optimization suggestions.

# Running the profiler in batch mode

- **Batch script to get exclusive profiling on a GPU**
  - ❏ Recommended – gives more reliable results

```sh
#!/bin/sh
#BSUB -J proftest
#BSUB -q hpcintrogpu
#BSUB -n 4
#BSUB -R "span[hosts=1]"
#BSUB -gpu "num=1:mode=exclusive_process"
#BSUB -W 10
#BSUB -R "rusage[mem=2048]"

export TMPDIR=$__LSF_JOB_TMPDIR__
module load nvhpc/25.11


export MATMULT_COMPARE=0 MFLOPS_MAX_IT=1


ncu -o profile_$LSB_JOBID \
    --section MemoryWorkloadAnalysis \
    --section MemoryWorkloadAnalysis_Chart \
    --section ComputeWorkloadAnalysis \
./matmult_f.nvc++ mnk_offload 2048 2048 2048
```

# Running the profiler in batch mode

- **You can modify the sections of interest by adding/removing them as needed**
- **Submit with:** `$ bsub < scriptname`
  - After the job is finished, you will find a profile with the name `profile_JOBID.ncu-rep` in your folder
  - This profile can be now opened in the Nsight GUI in your interactive session with the command:

```
TMPDIR=. ncu --open-in-ui \
-i profile_9172161.ncu-rep
```

  - Note: the JOBID 9172161 is appended by the profiler

# Asynchronous offloading

# Synchronous offloading

- ## By default `target` constructs are synchronous
  - ❑ Host thread waits until the offloaded region is finished
  - ❑ Host thread is occupied while waiting
    - ■ We risk that the CPU is idle while offloading
  - ❑ All transfers with the `map` clause are synchronous
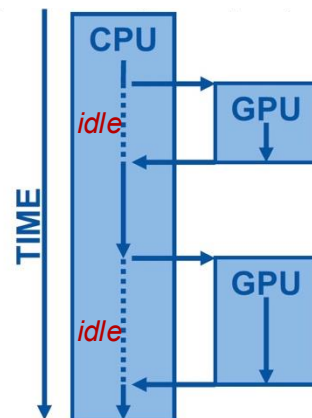
# Asynchronous offloading

- **Asynchronous offloading allows**
  - ❑ Simultaneous computing on host and device
  - ❑ Overlap of data transfers and computation
  - ❑ Overlap of data transfers in both directions
  - ❑ Simultaneous execution of several offload regions (if resources are available)
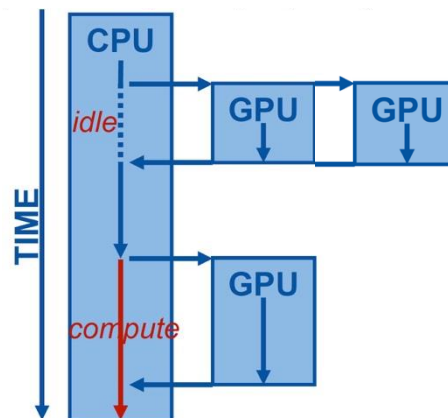
# OpenMP offload syntax

- Syntax C/C++:

```
#pragma omp target nowait
#pragma omp taskwait
```

- The `nowait` clause overrides any synchroniza-tion that would otherwise occur at the end of a construct, including the `target` construct

- For constructs that generate a task, the `nowait` clause specifies that the task may be deferred

- The `taskwait` construct specifies a wait on the completion of child tasks of the current task

# OpenMP offload syntax

- Syntax C/C++:

```
depend([depend-modifier,]depend-type : list)
```

- The `depend` clause enforces constraints on the scheduling of tasks or loop iterations
- A task cannot be executed until all its dependen-cies from predecessor tasks are completed

> Not currently fully supported for target constucts in `nvc++`

- `depend-modifier`: out-of-scope for 02614

# OpenMP offload syntax

- `dependence-type` can be

<div style="border: 1px solid red;">Not currently supported in `nvc++`</div>

  - ❏ `in`
    - Task will depend on all previously generated sibling tasks that reference the list-item in an `out` or `inout` dependence
  - ❏ `out/inout`
    - Task will depend on all previously generated sibling tasks that reference the list-item in an `in`, `out` or `inout` dependence
  - ❏ ~~`inoutset/mutexinoutset`~~
    - Specifies a set of tasks that depend in the same list-item and can be executed in any order possibly with mutual exclusion
  - ❏ ~~`depobj`~~
    - Task dependences are derived from the dependences represented by the depend objects (user-computed)

# Asynchronous offloading

- Simultaneous computing on host and device

```
#pragma omp target map(...) nowait depend(out: gpu_data)
/* do work on device -> gpu_data */

#pragma omp task nowait depend(out: cpu_data)
/* do work on host -> cpu_data */

#pragma omp task depend(in: cpu_data) depend(in: gpu_data)
/* combined work on host */

#pragma omp taskwait // wait for all tasks
```

# Asynchronous offloading

■ Example – vector addition v1

In order to make compute time longer – Do not do this!

```
void vecadd(double *a, double *b, double *c)
{
    #pragma omp target teams distribute parallel for \
        num_teams(N) thread_limit(1)                 \
        map(to: a[0:N], b[0:N]) map(from: c[0:N])
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

# Asynchronous offloading

■ **Example – vector addition v1**

In order to make compute time longer – Do not do this!

```cpp
void vecadd(double *a, double *b, double *c)
{
    #pragma omp target teams distribute parallel for \
        num_teams(N) thread_limit(1)                 \
        map(to: a[0:N], b[0:N]) map(from: c[0:N])
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

main.cpp

```cpp
#define N 1000000
double t, tmin = 1;
memset(c, 0, N * sizeof(double));
for (int n = 0; n < 100; ++n) {
    t = omp_get_wtime();
    vecadd(a, b, c);
    t = omp_get_wtime() - t;
    if (tmin > t) tmin = t;
}
check(c);
printf("Runtime: %f\n", tmin);
```

```
$ make && ./vecadd
Runtime: 0.006749
```

| Version | v1 |
|---------|-----|
| Time [ms] | 6.75 |

# Asynchronous offloading

- **Example – vector addition v1**

```
$ nsys profile ./vecadd
Runtime: 0.006700
Generating '/tmp/nsys-report-4b5d.qdstrm'
[1/1] [=========================100%] report1.nsys-rep
```
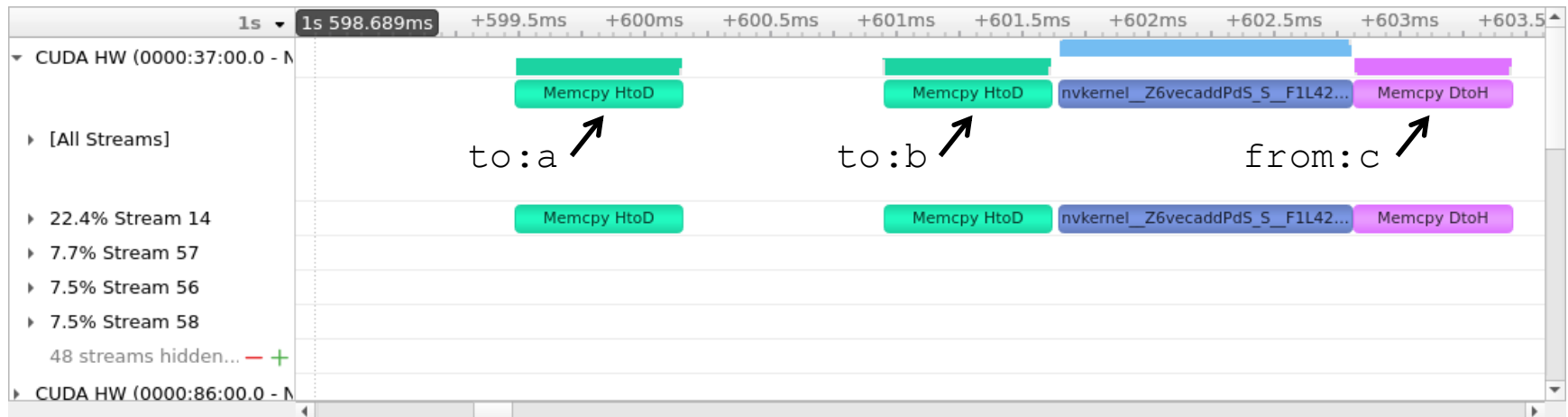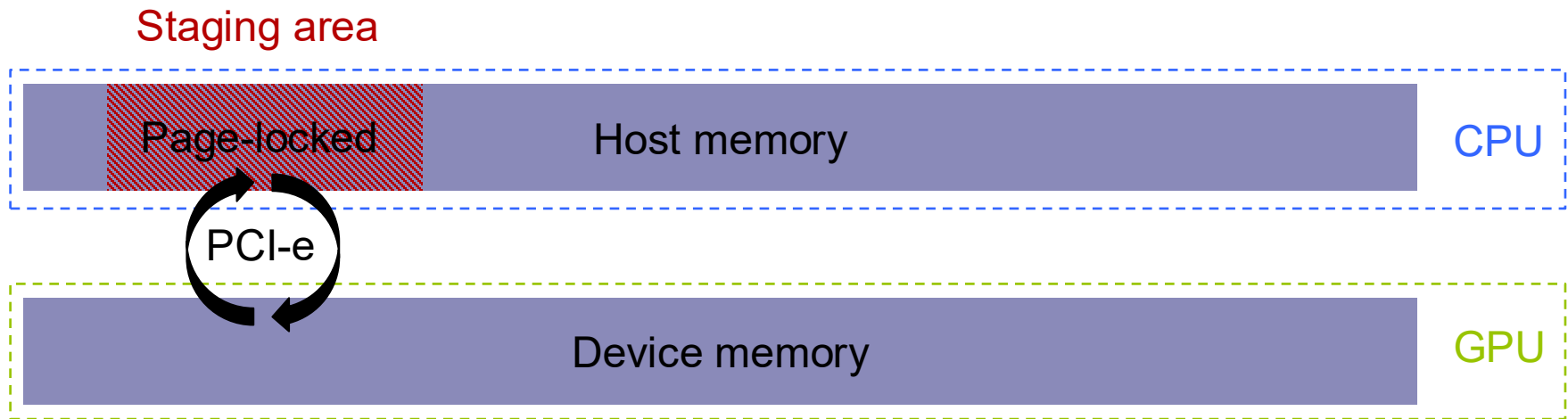


| Version | v1 |
|---------|-----|
| Time [ms] | 6.75 |

# Asynchronous offloading

- **Example – vector addition v2**

```
void vecadd2(double *a, double *b, double *c)
{
    #pragma omp target teams distribute parallel for nowait \
        num_teams(N/2) thread_limit(1)                       \
        map(to: a[0:N/2], b[0:N/2]) map(from: c[0:N/2])
    for (int i = 0; i < N/2; i++)
        c[i] = a[i] + b[i];

    #pragma omp target teams distribute parallel for nowait \
        num_teams(N/2) thread_limit(1)                       \
        map(to: a[N/2:N/2], b[N/2:N/2]) map(from: c[N/2:N/2])
    for (int i = N/2; i < N; i++)
        c[i] = a[i] + b[i];

    #pragma omp taskwait
}
```

# Asynchronous offloading

- ## Example – vector addition v2

```
$ nsys profile ./vecadd2
Runtime: 0.005060
Generating '/tmp/nsys-report-7242.qdstrm'
[1/1] [=========================100%] report2.nsys-rep
```



Overlapping transfers
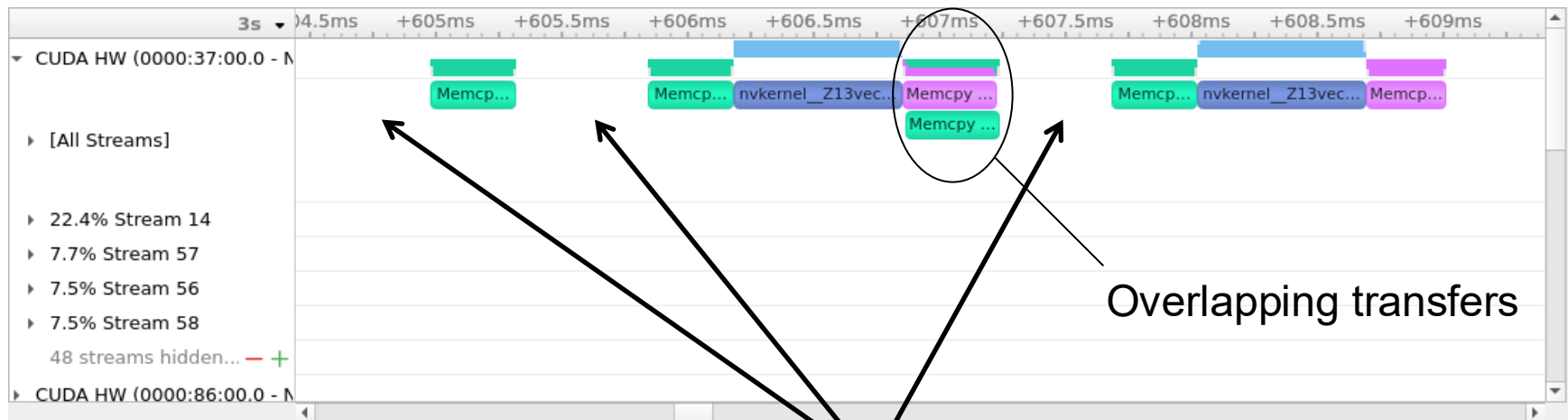
But what's going on here?

| Version | v1 | v2 |
|---------|------|------|
| Time [ms] | 6.75 | 5.06 |

# Pinned host memory

Staging area

Page-locked | Host memory | CPU

PCI-e

Device memory | GPU

# Pinned host memory

memcpy()

Staging area

Page-locked    Host memory    CPU

PCI-e

Device memory    GPU

# Pinned host memory

memcpy()

Staging area

Pinned

Page-locked | Host memory | CPU

PCI-e | PCI-e

Device memory | GPU

# Pinned host memory

Memory transfers (NVIDIA A100-PCIE-40GB)

PINNED Memory Transfer
PAGEABLE Memory Transfer

Bandwidth (GB/s)

Size (KB)
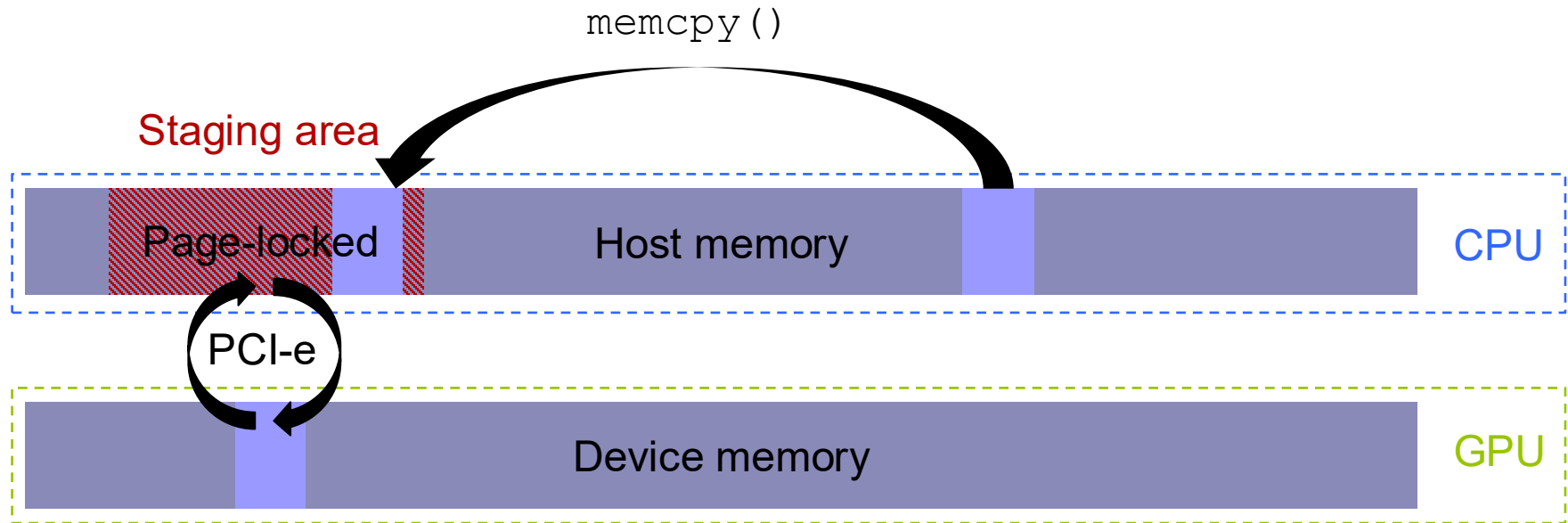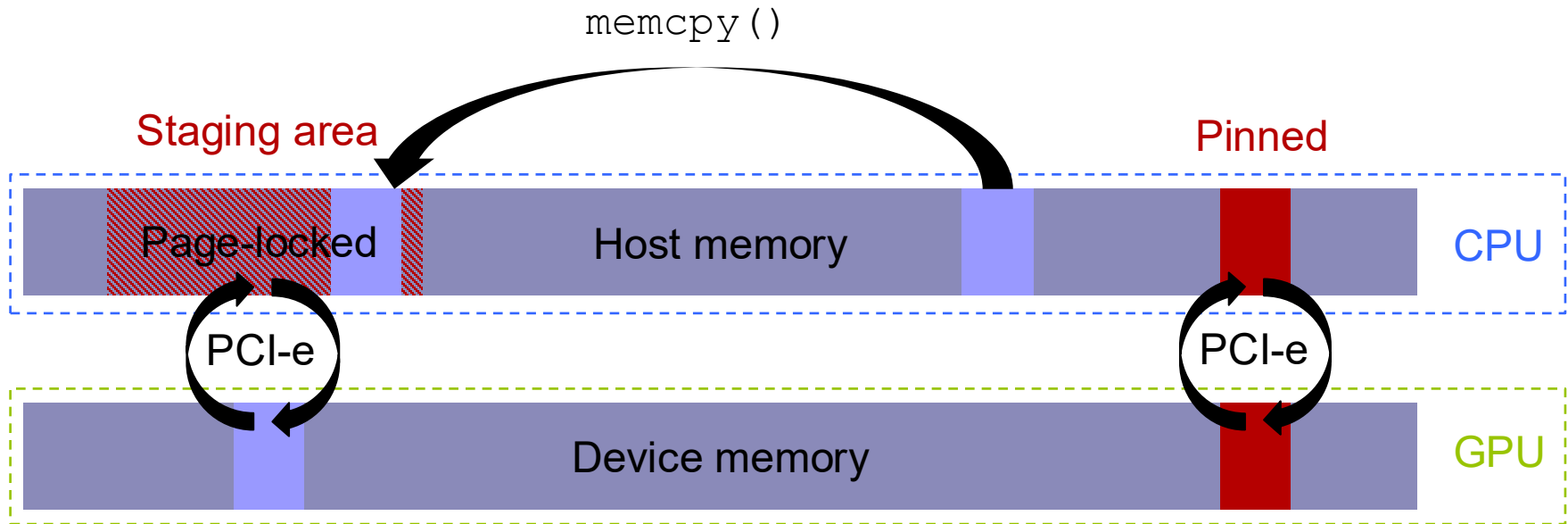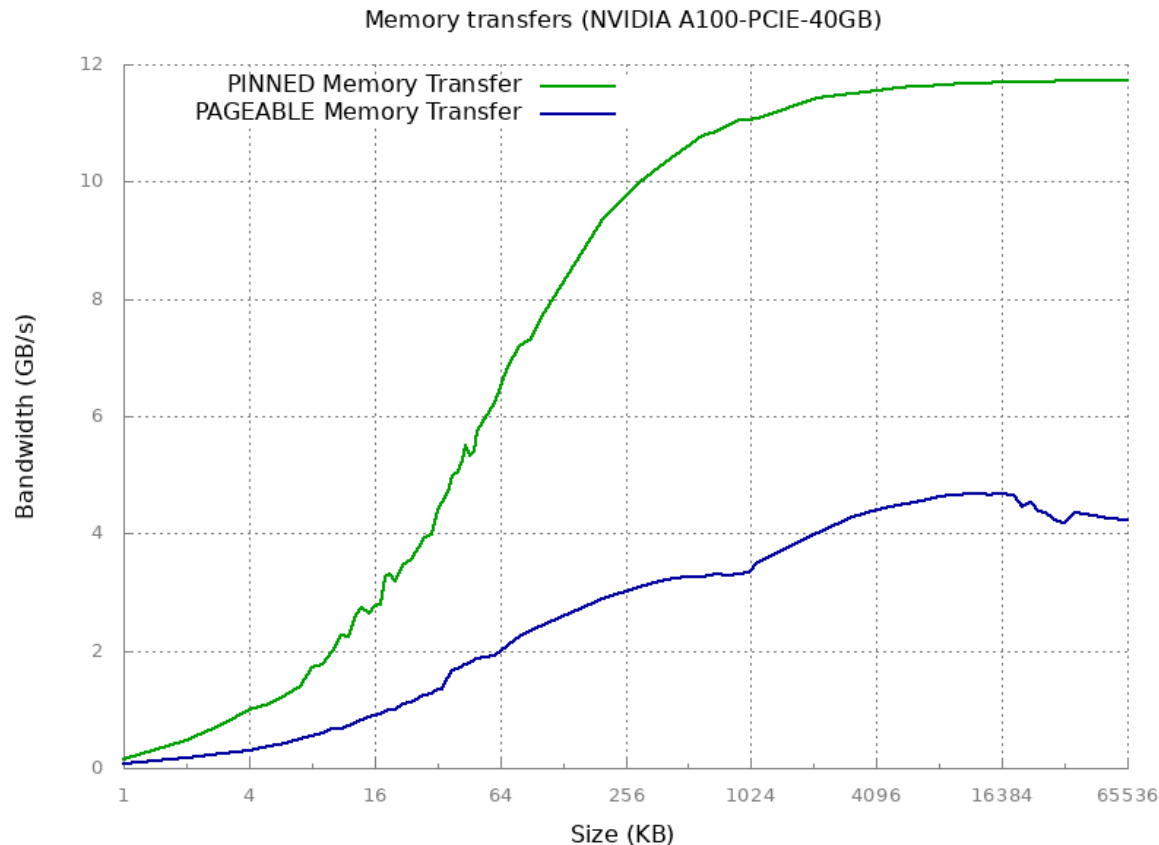
Use `nvc++` **option**
`-gpu=mem:`
`separate:`
`pinnedalloc`
**vs.**
`-gpu=mem:`
`separate:`
`nopinnedalloc`
(default)

■ **Drawback: The pinned allocation is much more expensive than a standard allocation with malloc**

# Asynchronous offloading

- ■ **Example – vector addition v2 pinned**

```
$ nsys profile ./vecadd2_pinned
Runtime: 0.002053
Generating '/tmp/nsys-report-bc06.qdstrm'
[1/1] [========================100%] report3.nsys-rep
```
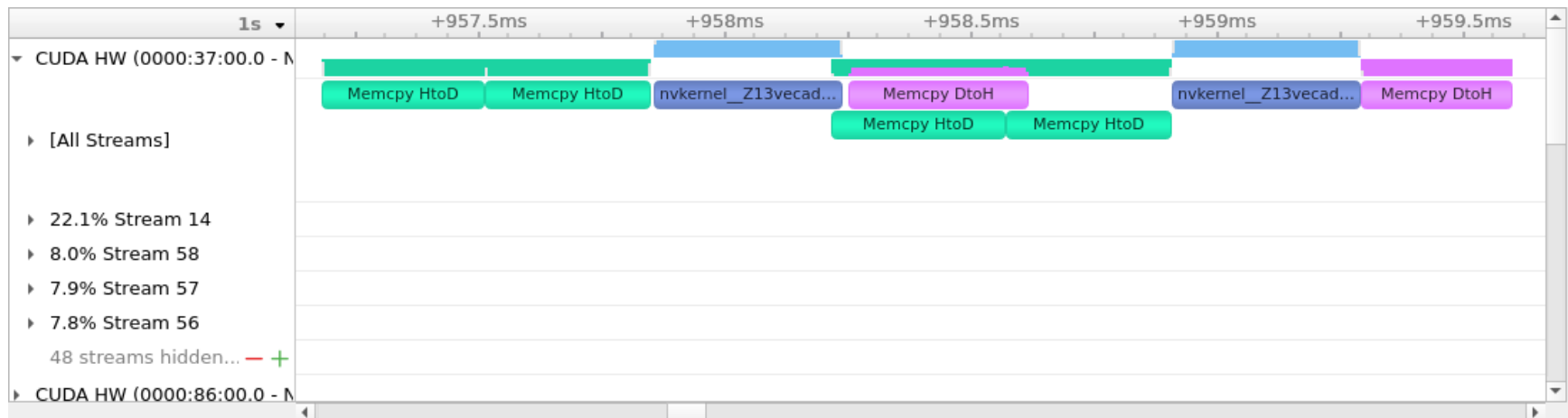


| Version | v1 | v2 | v2 pin. |
|---|---|---|---|
| Time [ms] | 6.75 | 5.06 | 2.05 |

# Asynchronous offloading

- Example – vector addition v3

```
void vecadd3(double *a, double *b, double *c)
{
    #define SPLITS 8

    for (int s = 0; s < SPLITS; ++s) {
        int length = N / SPLITS;
        int lower = s * length;
        #pragma omp target teams distribute parallel for nowait \
            num_teams(length) thread_limit(1)                    \
            map(to: a[lower:length], b[lower:length])            \
            map(from: c[lower:length])
        for (int i = lower; i < lower + length; i++)
            c[i] = a[i] + b[i];
    }
    #pragma omp taskwait
}
```

# Asynchronous offloading

■ Example – vector addition v3

```
$ nsys profile ./vecadd3_pinned
Runtime: 0.001716
Generating '/tmp/nsys-report-94aa.qdstrm'
[1/1] [===========================100%] report4.nsys-rep
```
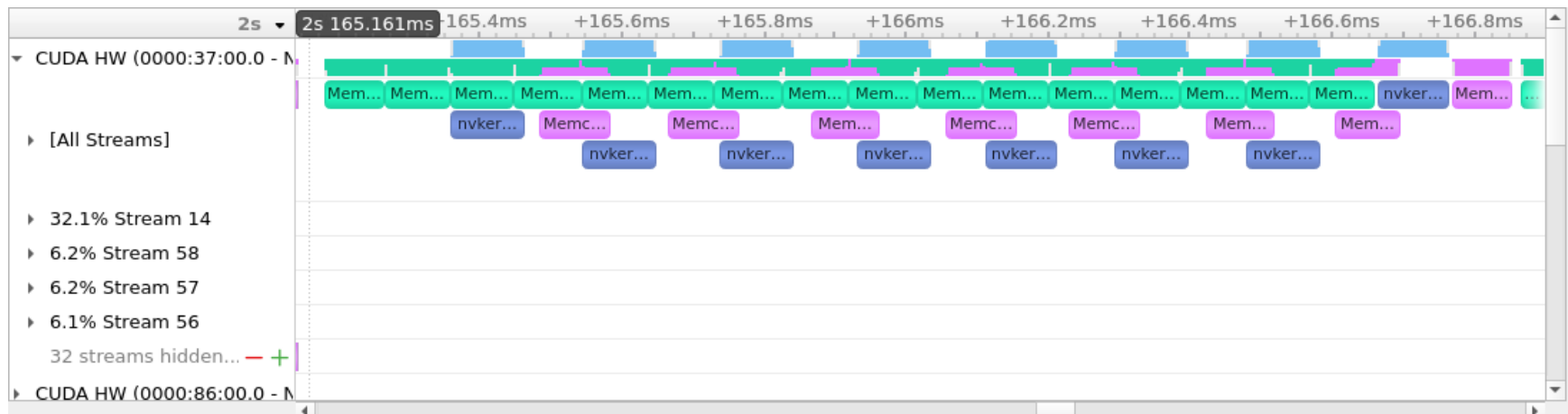


| Version | v1 | v2 | v2 pin. | v3 |
|---------|------|------|---------|------|
| Time [ms] | 6.75 | 5.06 | 2.05 | 1.72 |

# Asynchronous offloading

- Example – vector addition v3 alternative

```
void vecadd3(double *a, double *b, double *c)
{
    #define SPLITS 8

    #pragma omp parallel for
    for (int s = 0; s < SPLITS; ++s) {
        int length = N / SPLITS;
        int lower = s * length;
        #pragma omp target teams distribute parallel for \
            num_teams(length) thread_limit(1)            \
            map(to: a[lower:length], b[lower:length])    \
            map(from: c[lower:length])
        for (int i = lower; i < lower + length; i++)
            c[i] = a[i] + b[i];
    }
}
```

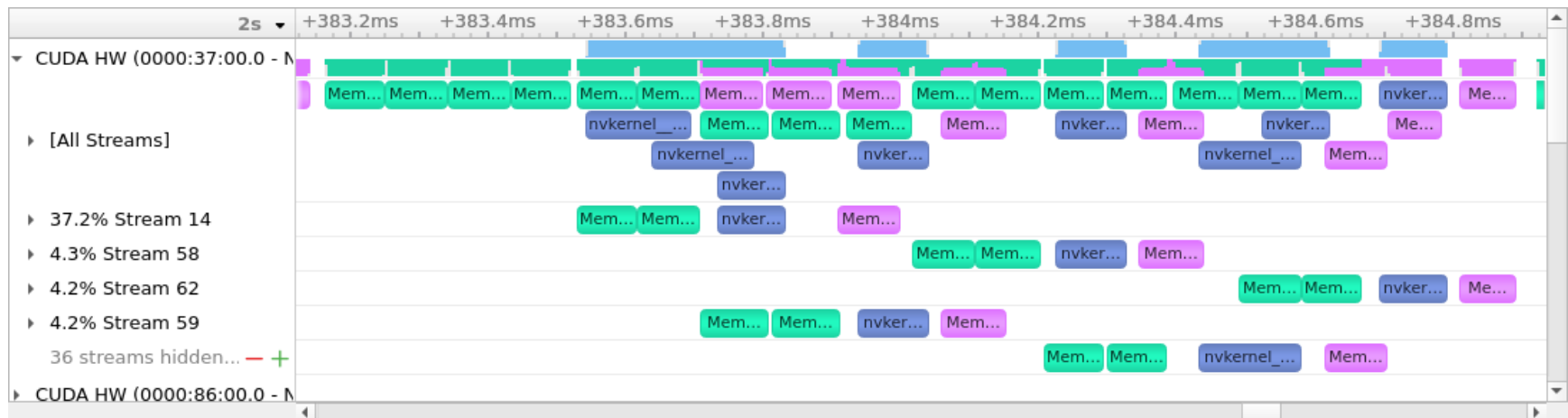OpenMP standard parallel threads as an alternative to nowait

# Asynchronous offloading

- **Example – vector addition alternative**

```
$ nsys profile ./vecadd3alt_pinned
Runtime: 0.001852
Generating '/tmp/nsys-report-3a3d.qdstrm'
[1/1]  [========================100%] report5.nsys-rep
```



| Version | v1 | v2 | v2 pin. | v3 | v3 alt. |
|---------|------|------|---------|------|---------|
| Time [ms] | 6.75 | 5.06 | 2.05 | 1.72 | 1.85 |

# Multi-GPU

02614 – High Performance Computing

# Multi-GPU systems

- **Multi-GPU systems appear in several flavors**

Server

NVIDIA Tesla K80

HPC Cluster (via MPI)



- ❑ NVIDIA Tesla K80, while physically occupying a single expansion slot, will appear to OpenMP applications as two separate GPUs

# Multi-GPU systems

- **Using multiple GPUs within the same application can improve the performance**
  - Splitting the task (extra level of parallelism)
  - Scales the peak performance
  - Scales the memory bandwidth
  - Does NOT always scale the PCI-e bandwidth!

# Multi-GPU with OpenMP

- **omp_get_num_devices()**
  - ❏ Gets the number of available GPUs
- **Clause device()**
  - ❏ Selects the device on all target contructs

```c
// Run independent offloads on each device
int numDevs = omp_get_num_devices();
for (int d = 0; d < numDevs; d++) {

    #pragma omp target nowait device(d)
    {
     ...
    }

}
```

# Multi-GPU with OpenMP

■ Example using `device()` clause

```
void vecadd_multi(double *a, double *b, double *c)
{
    #pragma omp target teams distribute parallel for nowait \
        num_teams(N/2) thread_limit(1) device(0)                \
        map(to: a[0:N/2], b[0:N/2]) map(from: c[0:N/2])
    for (int i = 0; i < N/2; i++)
        c[i] = a[i] + b[i];

    #pragma omp target teams distribute parallel for nowait \
        num_teams(N/2) thread_limit(1) device(1)                \
        map(to: a[N/2:N/2], b[N/2:N/2]) map(from: c[N/2:N/2])
    for (int i = N/2; i < N; i++)
        c[i] = a[i] + b[i];

    #pragma omp taskwait
}
```
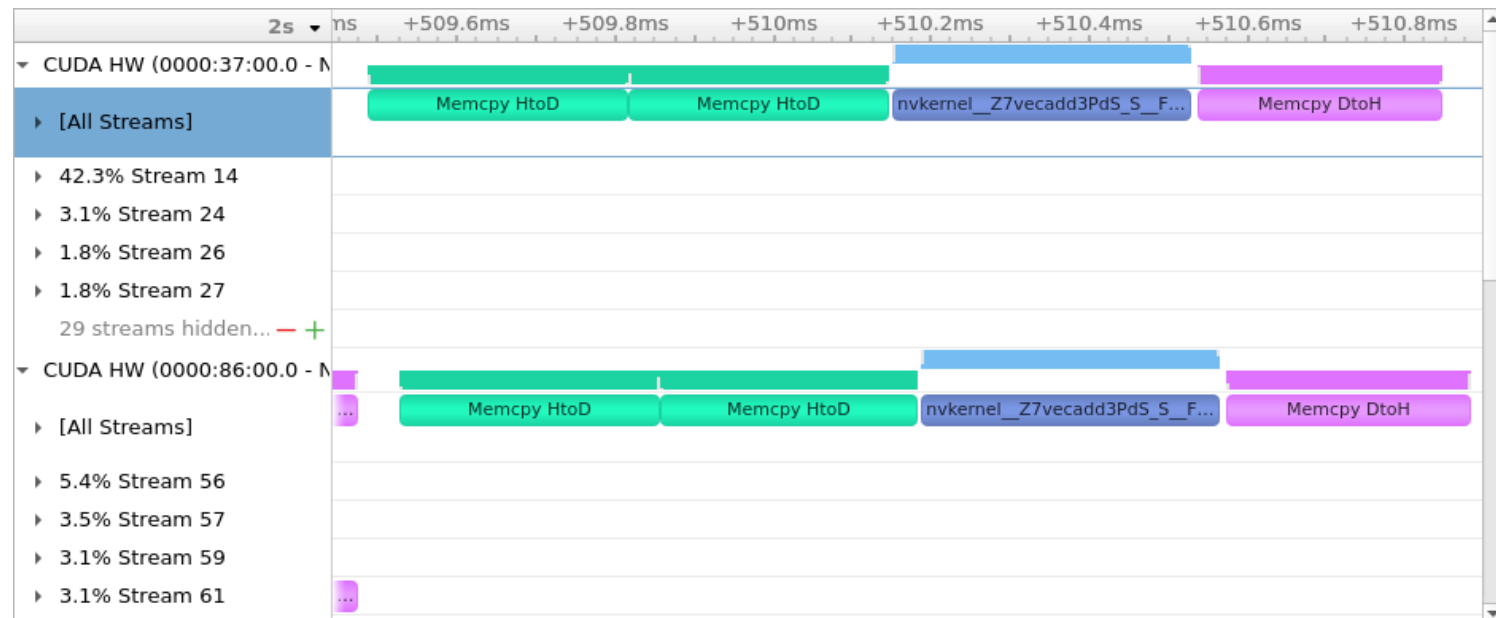
# Multi-GPU with OpenMP

- **Example using `device()` clause**

```
$ nsys profile ./vecadd_multi
Runtime: 0.001564
Generating '/tmp/nsys-report-3ba8.qdstrm'
[1/1] [=========================100%] report6.nsys-rep
```

# Multi-GPU with OpenMP API

- **omp_set_default_device()**
  - ❑ Sets the device to run on

- **omp_get_default_device()**
  - ❑ Gets the current device

```c
// Allocate memory on two devices
double **a_d0, **a_d1, *data_d0, *data_d1;

omp_set_default_device(0);
a_d0 = malloc_2d_dev(N, N, &data_d0);
omp_set_default_device(1);
a_d1 = malloc_2d_dev(N, N, &data_d1);

... // clause: is_device_ptr(a_d0/1)

omp_set_default_device(0);
free_2d_dev(a_d0, data_d0);
omp_set_default_device(1);
free_2d_dev(a_d1, data_d1);
```

```c
/* Routine for allocating two-dimensional
   array on the device */
double **malloc_2d_dev(int m, int n,
                       double **data)
{
    if (m <= 0 || n <= 0)
        return NULL;
    double **A =(double**)omp_target_alloc(
                m*sizeof(double *),
                omp_get_default_device());
    ...
}
```

# Multi-GPU peer-to-peer access

- **Trick: This is not currently part of OpenMP!**

- Use `cudaDeviceEnablePeerAccess()` to get unidirectional peer access to other GPUs

```
// Enable peer-to-peer access and offload
cudaSetDevice(0);
cudaDeviceEnablePeerAccess(1, 0); // (dev 1, future flag)
cudaSetDevice(1);
cudaDeviceEnablePeerAccess(0, 0); // (dev 0, future flag)
cudaSetDevice(0);
#pragma omp target is_device_ptr(a_d0, a_d1) device(0)
{
   ... // both devices can access both arrays a_d0 and a_d1!
}
```

- Check peer access support with `deviceQuery`

# End of lecture