

# MODULE TWO: PROFILING

Dr. Volker Weinberg | LRZ

# MODULE OVERVIEW

## Topics to be covered

- Compiling and profiling sequential code
- Explanation of multicore programming
- Compiling and profiling multicore code

# COMPILING SEQUENTIAL CODE

# NVIDIA'S HPC COMPILERS (AKA PGI)

## NVIDIA Compiler Names (PGI names still work)

- `nvc` - The command to compile C code (formerly known as 'pgcc')
- `nvc++` - The command to compile C++ code (formerly known as 'pgc++')
- `nvfortran` - The command to compile Fortran code (formerly known As `pgfortran/pgf90/pgf95/pgf77`)
- The `-fast` flag instructs the compiler to optimize the code to the best of its abilities

```
$ nvc -fast main.c  
$ nvc++ -fast main.cpp  
$ nvfortran -fast main.F90
```

```
$ pgcc -fast main.c  
$ pgc++ -fast main.cpp  
$ pgfortran -fast main.F90
```

# NVIDIA'S HPC COMPILERS (AKA PGI)

## -Minfo flag

- The Minfo flag will instruct the compiler to print feedback about the compiled code
- -Minfo=accel will give us information about what parts of the code were accelerated via OpenACC
- -Minfo=opt will give information about all code optimizations
- -Minfo=all will give all code feedback, whether positive or negative

```
$ pgcc -fast -Minfo=all main.c  
$ pgc++ -fast -Minfo=all main.cpp  
$ pgfortran -fast -Minfo=all main.f90
```

# NVIDIA NSIGHT FAMILY

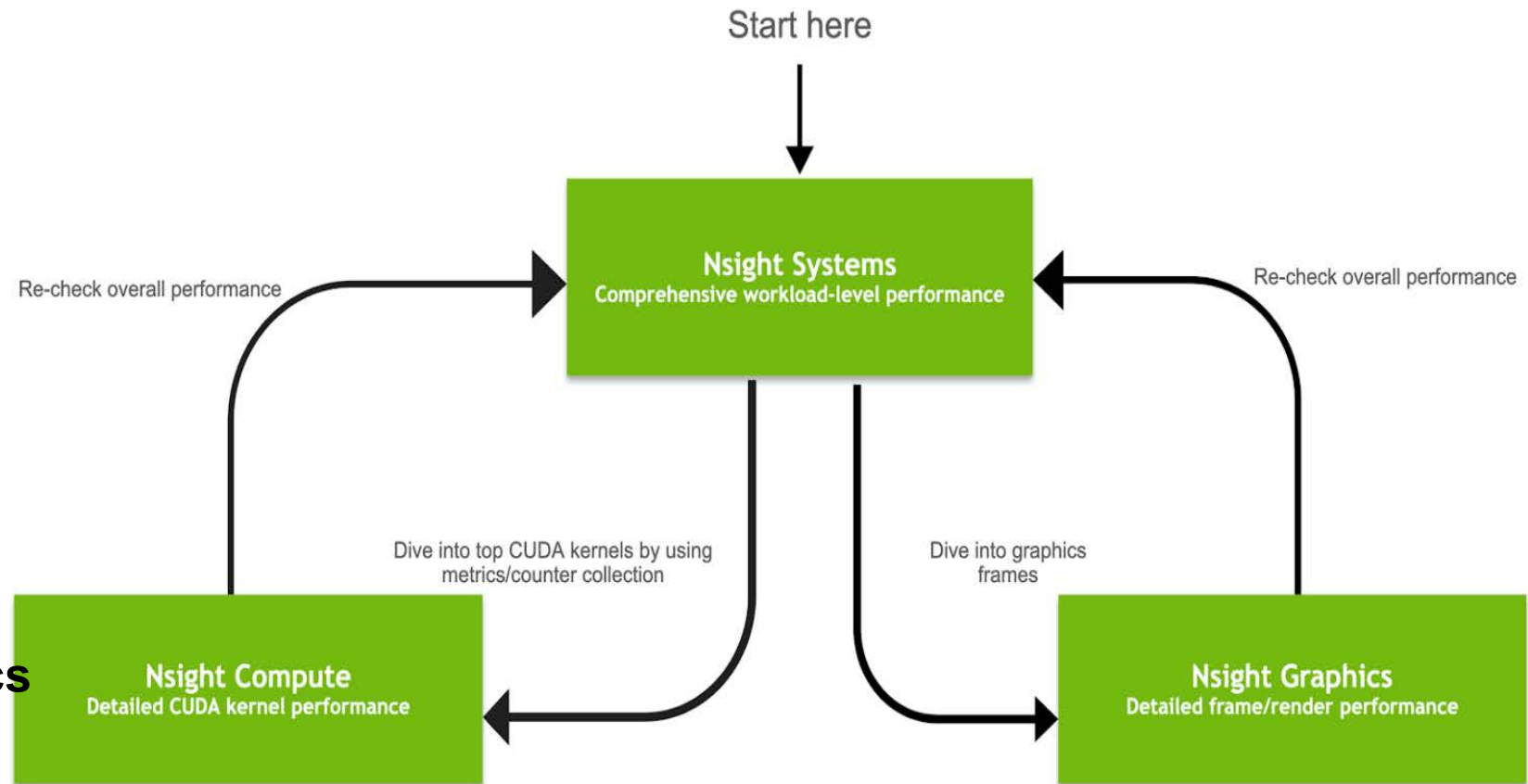
# Nsight Product Family

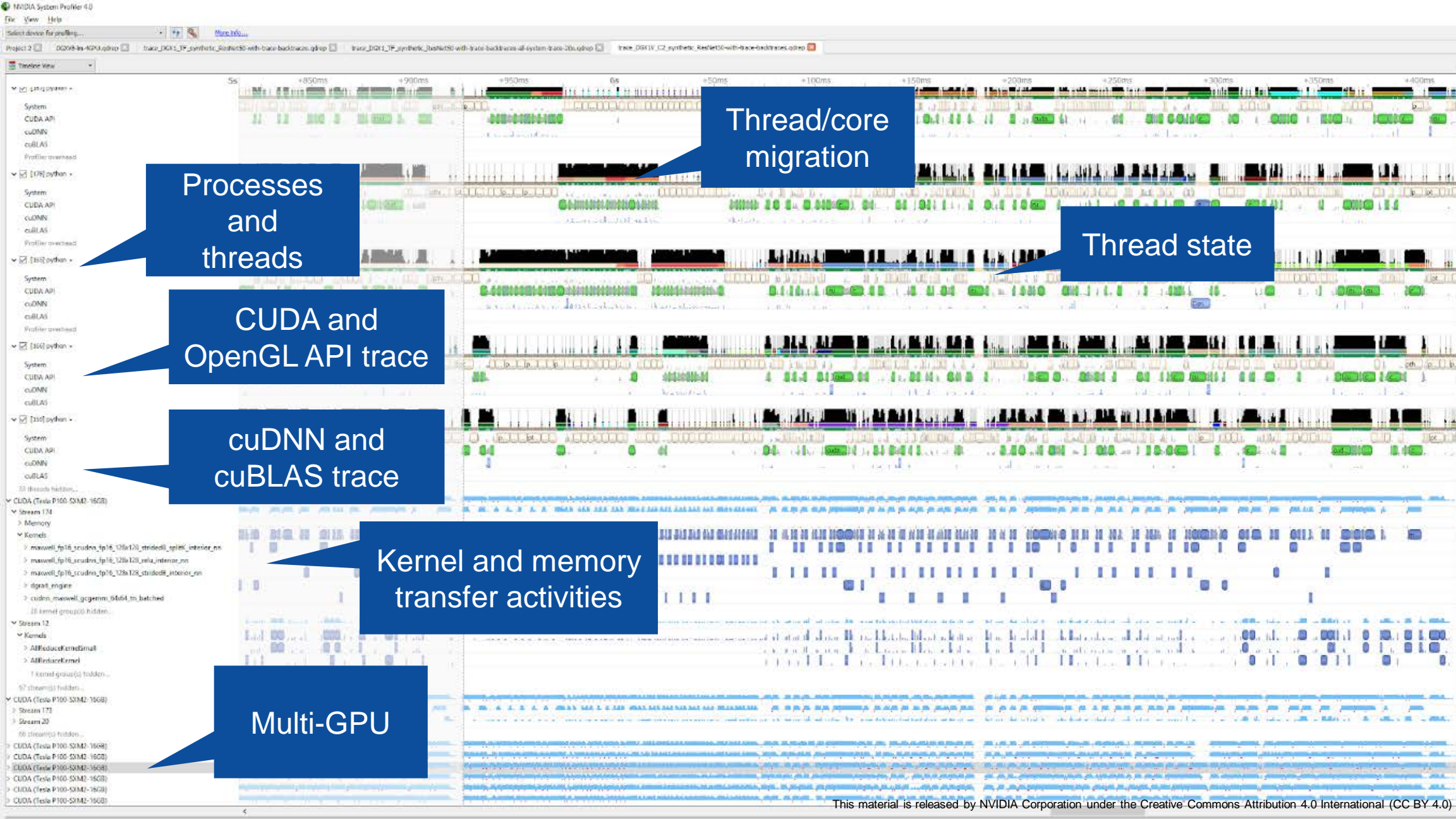
## Workflow

**Nsight Systems** -  
Analyze application  
algorithm system-wide

**Nsight Compute** -  
Debug/optimize CUDA  
kernel

**Nsight Graphics** -  
Debug/optimize graphics  
workloads



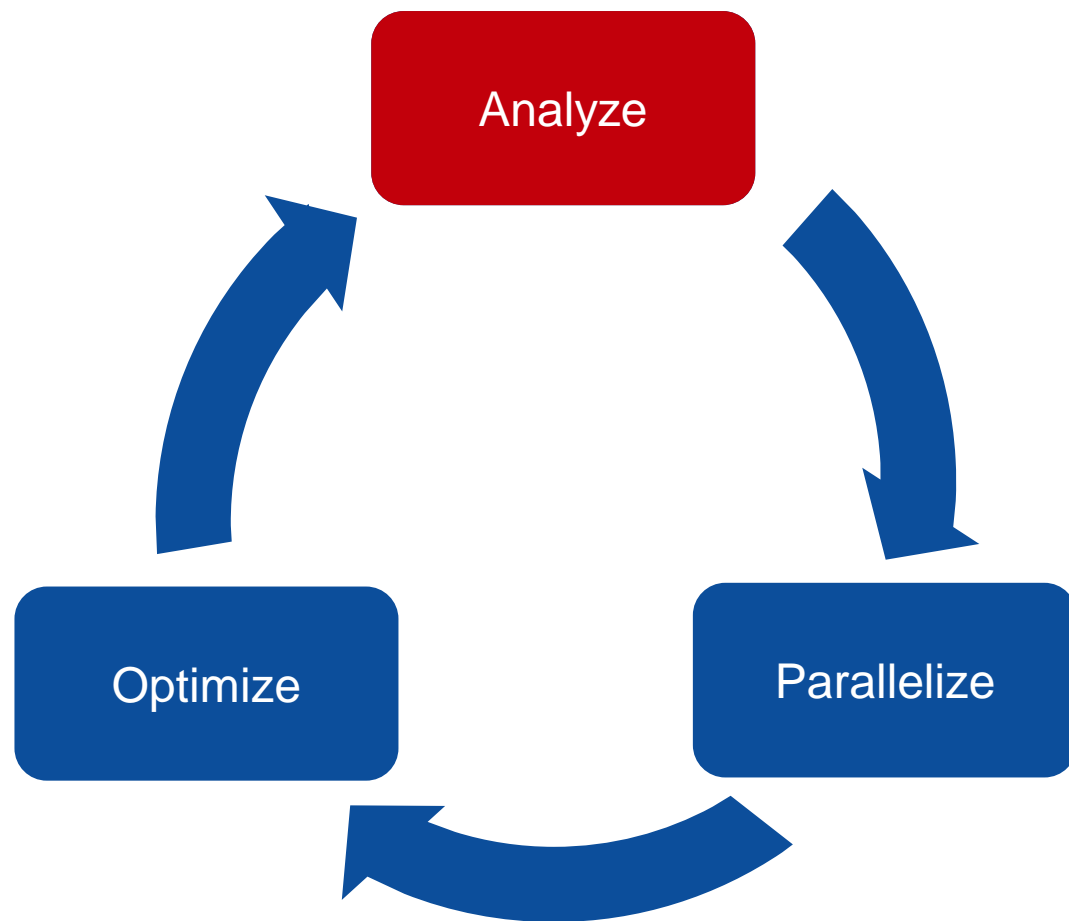




# PROFILING SEQUENTIAL CODE

# OPENACC DEVELOPMENT CYCLE

- **Analyze** your code to determine most likely places needing parallelization or optimization.
- **Parallelize** your code by starting with the most time consuming parts, check for correctness and then analyze it again.
- **Optimize** your code to improve observed speed-up from parallelization.



# PROFILING SEQUENTIAL CODE

## Step 1: Run Your Code

Record the time it takes for your sequential program to run.

Note the final results to verify correctness later.

Always run a problem that is representative of your real jobs.

## Terminal Window

```
$ gcc -fast jacobi.c laplace2d.c
$ ./a.out
  0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 39.432648 s
```

# PROFILING SEQUENTIAL CODE

## Step 2: Profile Your Code

Obtain detailed information about how the code ran.

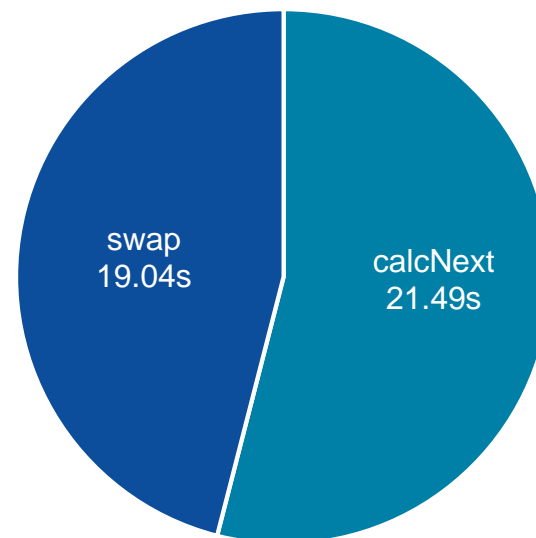
This can include information such as:

- Total runtime
- Runtime of individual routines
- Hardware counters

Identify the portions of code that took the longest to run. We want to focus on these “hotspots” when parallelizing.

## Lab Code: Laplace Heat Transfer

Total Runtime: 39.43 seconds



# PROFILING WITH NSIGHT SYSTEM AND NVTX

# PROFILING SEQUENTIAL CODE

## Using Command Line Interface (CLI)

NVIDIA Nsight Systems CLI provides

- Simple interface to collect data
- Can be copied to any system and analysed later
- Profiles both serial and parallel code
- For more info enter `nsys --help` on the terminal

To profile a serial application with NVIDIA Nsight Systems, we use NVIDIA Tools Extension (NVTX) API functions in addition to collecting backtraces while sampling.

# PROFILING SEQUENTIAL CODE

## NVIDIA Tools Extension API (NVTX) library

### What is it?

- A C-based Application Programming Interface (API) for annotating events
- Can be easily integrated to the application
- Can be used with NVIDIA Nsight Systems

### Why?

- Allows manual instrumentation of the application
- Allows additional information for profiling (e.g: tracing of CPU events and time ranges)

### How?

- Import the header only C library `nvToolsExt.h`
- Wrap the code region or a specific function with `nvtxRangePush()` and `nvtxRangePop()`

```

#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include "laplace2d.h"
#include <nvtx3/nvToolsExt.h>

int main(int argc, char** argv)
{
    const int n = 4096;
    const int m = 4096;
    const int iter_max = 1000;

    const double tol = 1.0e-6;
    double error = 1.0;

    double *restrict A = (double*)malloc(sizeof(double)*n*m);
    double *restrict Anew = (double*)malloc(sizeof(double)*n*m);

    nvtxRangePushA("init");
    initialize(A, Anew, m, n);
    nvtxRangePop();

    printf("Jacobi relaxation Calculation: %d x %d mesh\n", n, m);

    double st = omp_get_wtime();
    int iter = 0;

    nvtxRangePushA("while");
    while ( error > tol && iter < iter_max )
    {
        nvtxRangePushA("calc");
        error = calcNext(A, Anew, m, n);
        nvtxRangePop();

        nvtxRangePushA("swap");
        swap(A, Anew, m, n);
        nvtxRangePop();

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

        iter++;
    }
    nvtxRangePop();

    double runtime = omp_get_wtime() - st;

    printf(" total: %f s\n", runtime);

    deallocate(A, Anew);

    return 0;
}

```

jacobi.c  
(starting and ending of ranges are highlighted with the same color)

- t Selects the APIs to be traced (nvtx in this example)
- status if true, generates summary of statistics after the collection
- b Selects the backtrace method to use while sampling. The option dwarf uses DWARF's CFI (Call Frame Information).
- force-overwrite if true, overwrites the existing results
- o sets the output (qdrep) filename

```

mozhgank@prn-dgx-32:~/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel$ nsys profile -t nvtx --stats=true -b dwarf --force-overwrite true -o laplace-seq ./laplace-seq
Collecting data...
Jacobi relaxation calculation: 4096 x 4096 mesh
  0, 0.250000
 100, 0.002397
 200, 0.001204
 300, 0.000804
 400, 0.000603
 500, 0.000483
 600, 0.000403
 700, 0.000345
 800, 0.000302
 900, 0.000269
total: 55.754501 s
Processing events...
Capturing symbol files...
Saving intermediate "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdstrm" file to disk...
Importing [=====100%]
Saved report file to "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.qdrep"
Exporting 70802 events: [=====100%]
Exported successfully to
/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-seq.sqlite
Generating NVTX Push-Pop Range Statistics...
NVTX Push-Pop Range Statistics (nanoseconds)
Time(%)  Total Time  Instances  Average  Minimum  Maximum  Range
-----
 49.9    55754497966      1  55754497966.0  55754497966  55754497966  while
 26.5    29577817696    1000  29577817.7    29092956    65008545  calc
 23.4    26163892482    1000  26163892.5    25761418    60129514  swap
  0.1         137489808      1    137489808.0    137489808    137489808  init

```

NVTX range statistics

"calc" region (calcNext function) takes 26.6%  
"swap" region (swap function) takes 23.4% of total execution time

Open laplace-seq.qdrep with Nsight System GUI to view the timeline

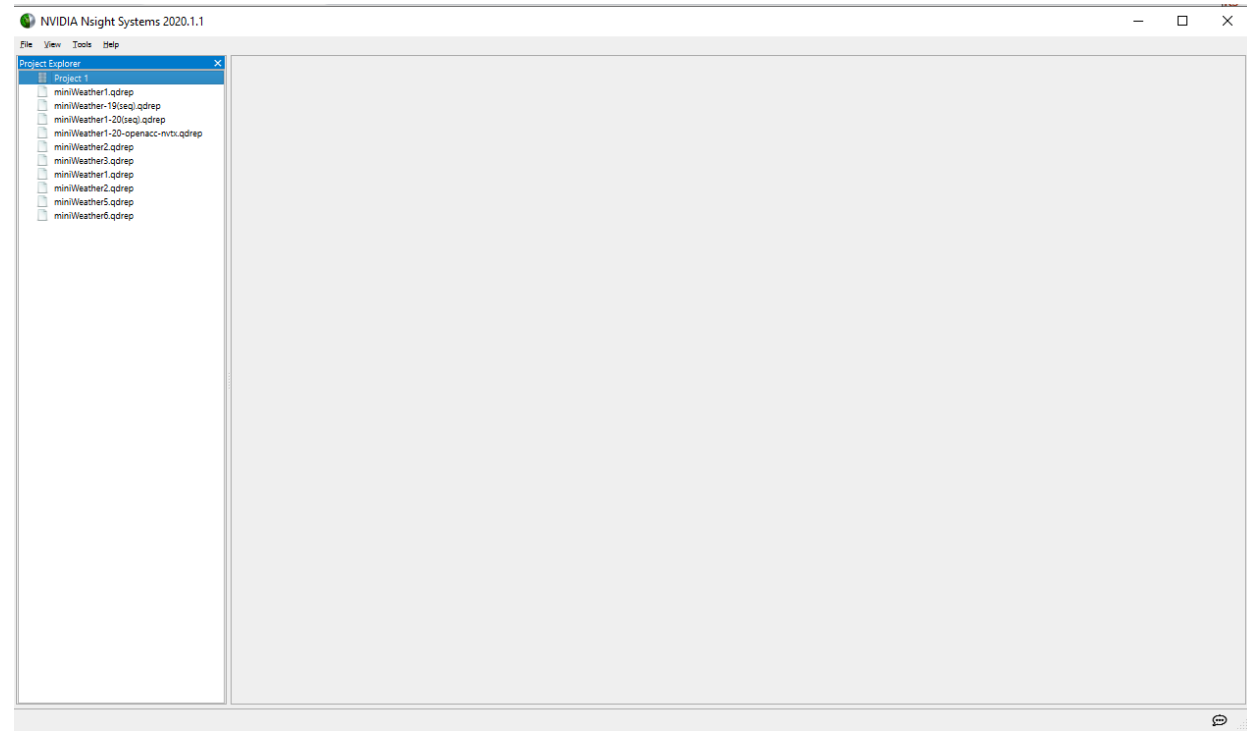


# PROFILING SEQUENTIAL CODE

## Using Nsight Systems

Open the generated report files (\*.qdrep) from command line in the Nsight Systems profiler.

File > Open



# PROFILING SEQUENTIAL CODE

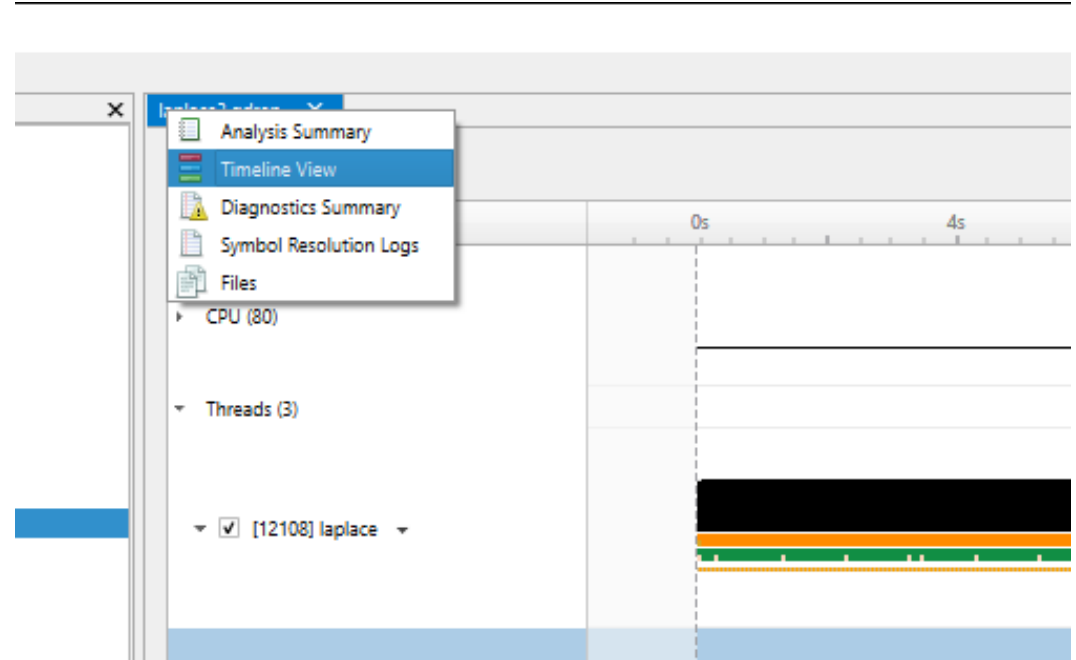
## Using Nsight Systems

Navigate through the “view selector”.

“Analysis summary” shows a summary of the profiling session. To review the project configuration used to generate this report, see next slide.

“Timeline View” contains the timeline at the top, and a bottom pane that contains the events view and the function table.

Read more: <https://docs.nvidia.com/nsight-systems>



# PROFILING SEQUENTIAL CODE

## Using Nsight Systems

**Analysis Summary**

Profiling session duration: 00:55.623

Total number of threads	3
Number of events collected	70,773
Report size	851.22 KiB
Report capture date	19 March 2020 08:01:16
Host computer	prm-dgx-28
Profiling stop reason	Stopped by user
Imported from	/home/mozhgank/Codes/opensource/training-materials/fabs/mods/ks2/English/C/ajplace3.qdstrm
Import host computer	prm-dgx-28
CLI command used	nsys profile -t mtr -stats=true --force-overwrite true -o ajplace3_ajplace

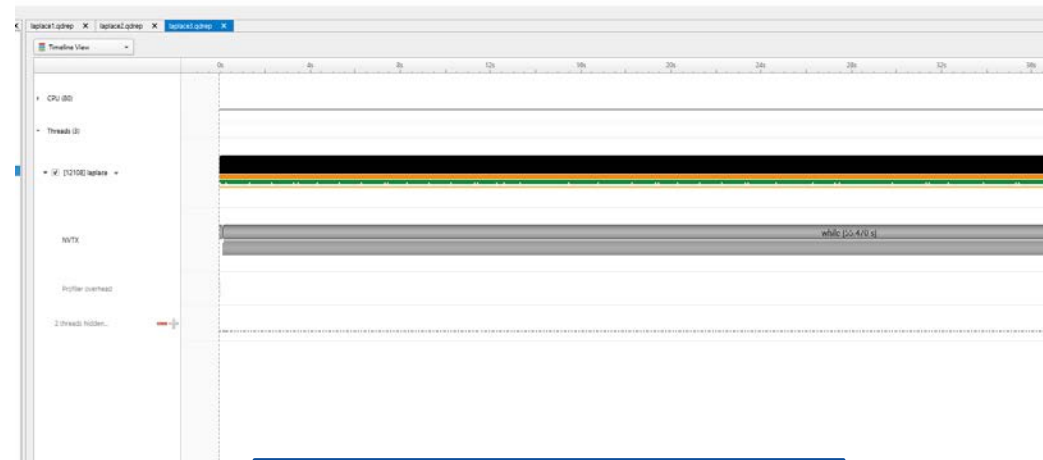
[Show report file in folder](#)

**prm-dgx-28 (0:1)**

**Target**

Target name	prm-dgx-28
Platform	Linux
OS	Ubuntu 18.04.3 LTS
Hardware platform	x86_64
Serial number	Local (CLI)
CPU description	Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz
CUDA driver version	10.2
NVIDIA driver version	440.33.01
GPU context switch	unsupported

Analysis Summary



Timeline view  
(charts and the hierarchy on the top pane)

**Events View**

ID	Name	Duration	TID	Start
1	int [1010E]ajplace	133.102 ms	1010E	0.00000000s
2	while [0-4/0]	15.470 s	1010E	0.14700000s

Timeline view  
(event view and function table on the bottom pane)

# PROFILING SEQUENTIAL CODE

## Using Nsight Systems

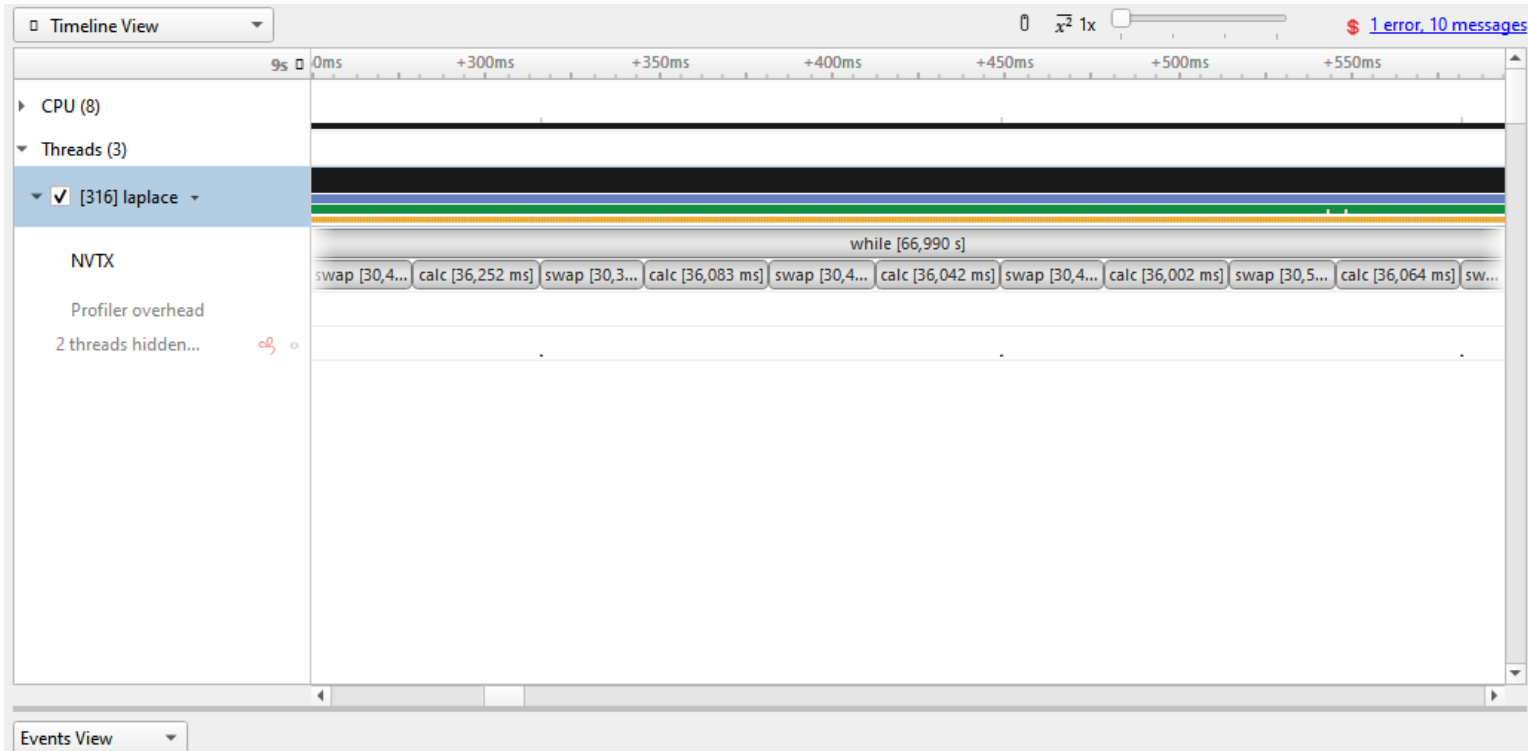
Enlarge view!

The screenshot displays the NVIDIA Nsight Systems 2021.2.1 interface. The main window shows a timeline view for a project named 'laplace.qdrep'. The timeline is divided into sections for CPU (8), Threads (3), and NVTX. A specific region is highlighted in green, spanning from 9s 585,729ms to 10,957ms. A context menu is open over this region, listing options: Filter and Reorder (Shift+F), Filter and Zoom in, Remove Filter, Zoom into Selection (highlighted), Undo Zoom (1), and Reset Zoom. A blue arrow points to the 'Zoom into Selection' option. Another blue arrow points to the top of the window with the text 'Enlarge view!'.

right click in  
selected region  
and Zoom into  
selection!

# PROFILING SEQUENTIAL CODE

## Using Nsight Systems

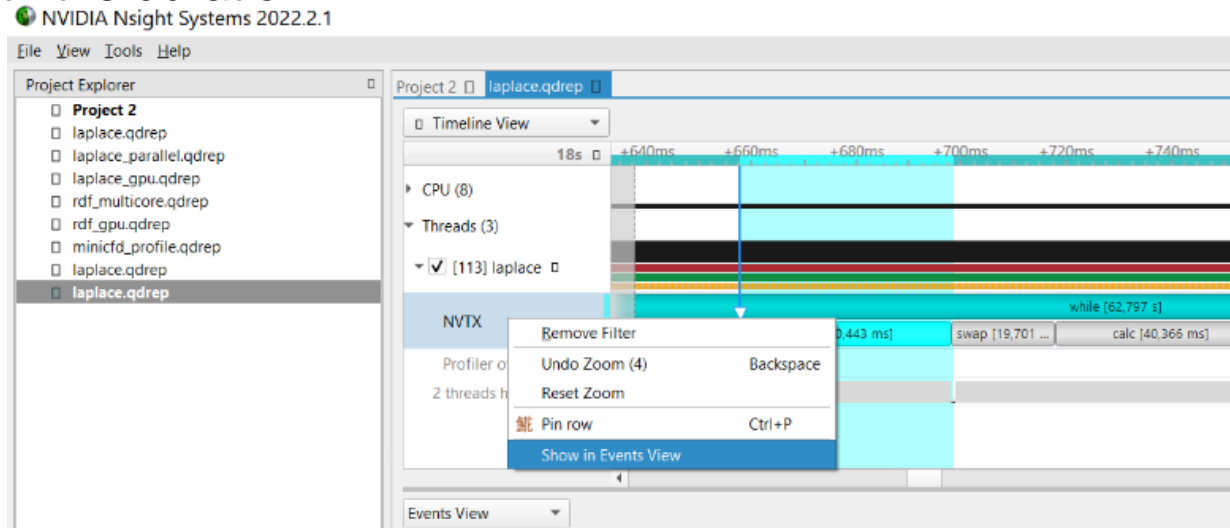


# PROFILING SEQUENTIAL CODE

Viewing captured NVTX events and time ranges via Nsight Systems GUI

From the Timeline view, right click on the “NVTX” from the top pane and choose “Show in Events View”.

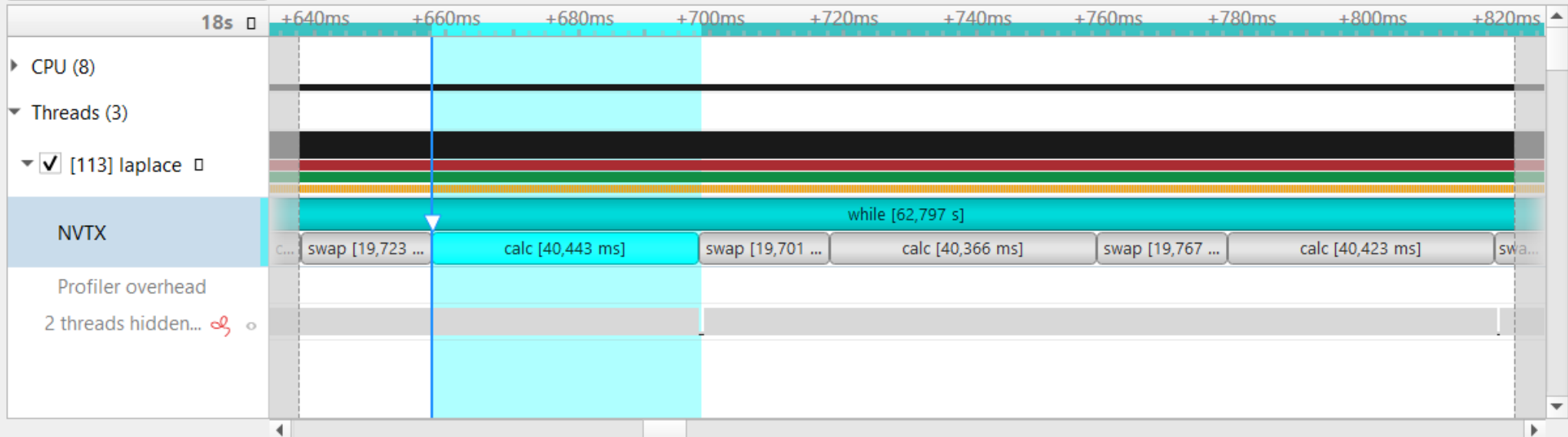
From the bottom pane, you can now see name of the events captured with the duration.



Timeline View

0  $\bar{x}^2$  1x

\$ 1 error, 10 messages



Events View

Name

#	Name	Start	Duration	TID	Category	Description:
1	while	0,193229s	62,797 s	113		<b>while</b> Begins: 0,193229s Ends: 62,9905s (+62,797 s) Thread: 113
2	calc	18,5981s	40,468 ms	113		
3	swap	18,6386s	19,723 ms	113		
4	calc	18,6583s	40,443 ms	113		
5	swap	18,6988s	19,701 ms	113		
6	calc	18,7185s	40,366 ms	113		
7	swap	18,7589s	19,767 ms	113		

PLEASE START LAB NOW!

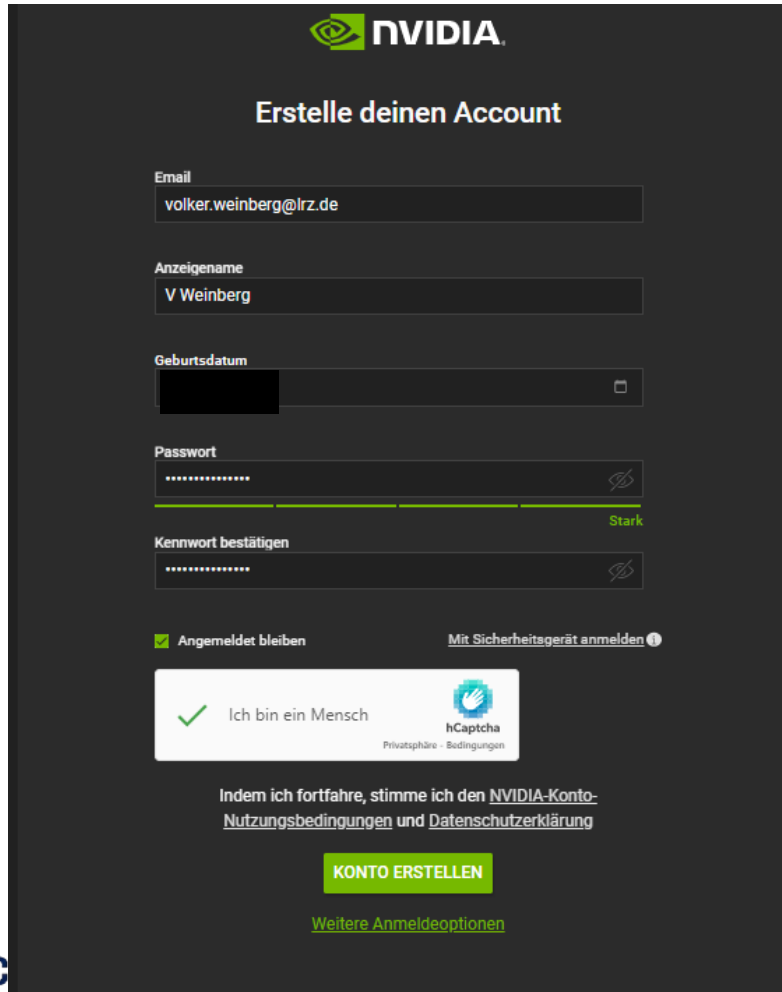


# TRAINING SETUP

- To get started, follow these steps:
- Create an NVIDIA Developer account at <http://courses.nvidia.com/join> Select "Log in with my NVIDIA Account" and then "Create Account"
- Visit <http://courses.nvidia.com/dli-event> and enter the event code

**LRZ\_OPENACC\_AMBASSADOR\_MY22**

# TRAINING SETUP



**NVIDIA**

## Erstelle deinen Account

Email  
volker.weinberg@lrz.de


Anzeigename  
V Weinberg

Geburtsdatum  
[Redacted]

Passwort  
[Redacted] **Stark**

Kennwort bestätigen  
[Redacted]

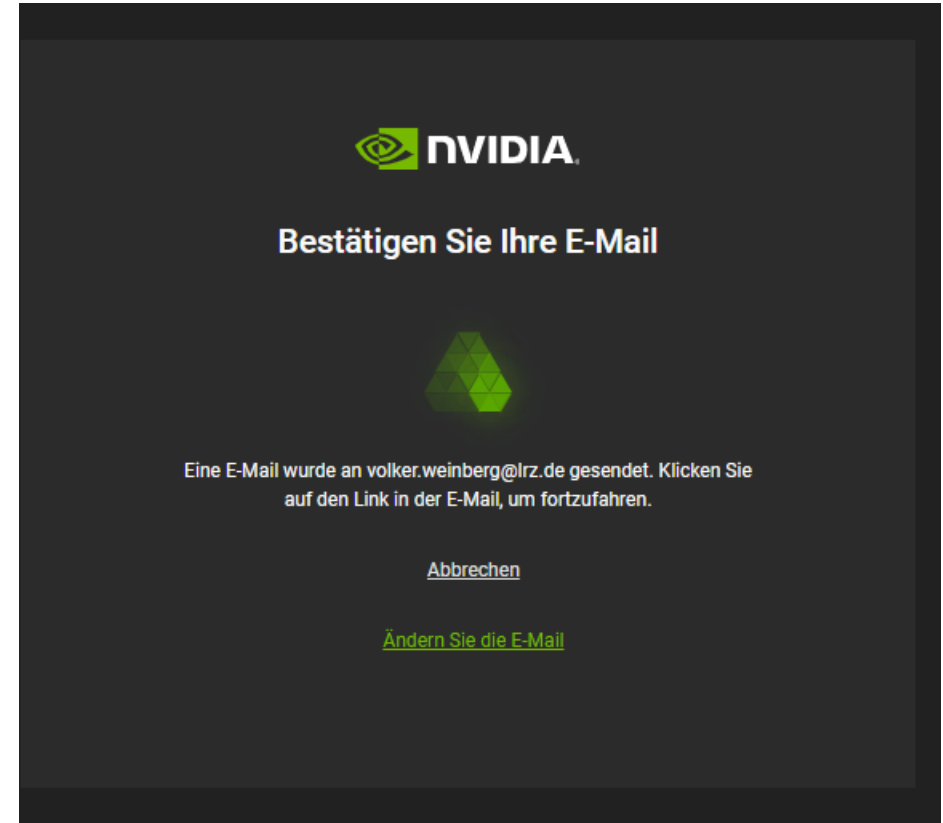
Angemeldet bleiben [Mit Sicherheitsgerät anmelden](#)

Ich bin ein Mensch  Privatsphäre · Bedingungen

Indem ich fortfahre, stimme ich den [NVIDIA-Konto-Nutzungsbedingungen](#) und [Datenschutzerklärung](#)

**KONTO ERSTELLEN**

[Weitere Anmeldeoptionen](#)



# TRAINING SETUP

← → ↻ 🏠 [courses.nvidia.com/dli-event/](https://courses.nvidia.com/dli-event/) ☆ V ⋮

📱 Apps 🎓 Linux Academy 🎓 Linux Academy We... 🗓️ DLI Event | Deep Le...



VOLKER ▾

DEEP LEARNING INSTITUTE

ONLINE COURSES

INSTRUCTOR-LED WORKSHOPS

DLI Event

Event Code

CSC\_OPENACC\_AMBASS

Enter your event code.

ENROLL

# TRAINING SETUP



VOLKER ▾

DEEP LEARNING INSTITUTE

ONLINE COURSES

INSTRUCTOR-LED WORKSHOPS

EDUCATOR PROGRAMS

ENTERPRISE SOLUTIONS

Course Progress

Fundamentals of Accelerated Computing with OpenACC

Search the course

Search

Start Course



Welcome to *Fundamentals of Accelerated Computing with OpenACC*.



Head over to the **"Course"** tab to get started!

#### Course Tools

Bookmarks

Updates

#### Course Handouts

No Course Handouts

Expand All

▾ Fundamentals of Accelerated Computing with OpenACC

🔗 [Click here to get started \(1 Question\)](#)

Assessment

Feedback

# TRAINING SETUP



VOLKER ▾

DEEP LEARNING INSTITUTE

ONLINE COURSES

INSTRUCTOR-LED WORKSHOPS

EDUCATOR PROGRAMS

ENTERPRISE SOLUTIONS

Course Progress

Course > Fundamentals of Accelerated Computing with... > Click here to get started > Fundamentals of Accelerated Computing with...

< Previous



Next >

## Fundamentals of Accelerated Computing with OpenACC

🔖 Bookmark this page



DEEP  
LEARNING  
INSTITUTE



START

To get started with this live GPU enabled interactive content please click the "Start" button on the top right of this block.

This will launch a pre-configured GPU workstation, it may take 5-10 minutes.

# TRAINING SETUP

DEEP LEARNING INSTITUTE

ONLINE COURSES

INSTRUCTOR-LED WORKSHOPS

EDUCATOR PROGRAMS

ENTERPRISE SOLUTIONS

Course

Progress

Course > [Fundamentals of Accelerated Computing with...](#) > [Click here to get started](#) > Fundamentals of Accelerated Computing with...

< Previous



Next >

## Fundamentals of Accelerated Computing with OpenACC

[Bookmark this page](#)



DEEP  
LEARNING  
INSTITUTE

This Lab -1 : -2 : -4 / 8 : 00 : 00

▶  
LAUNCH

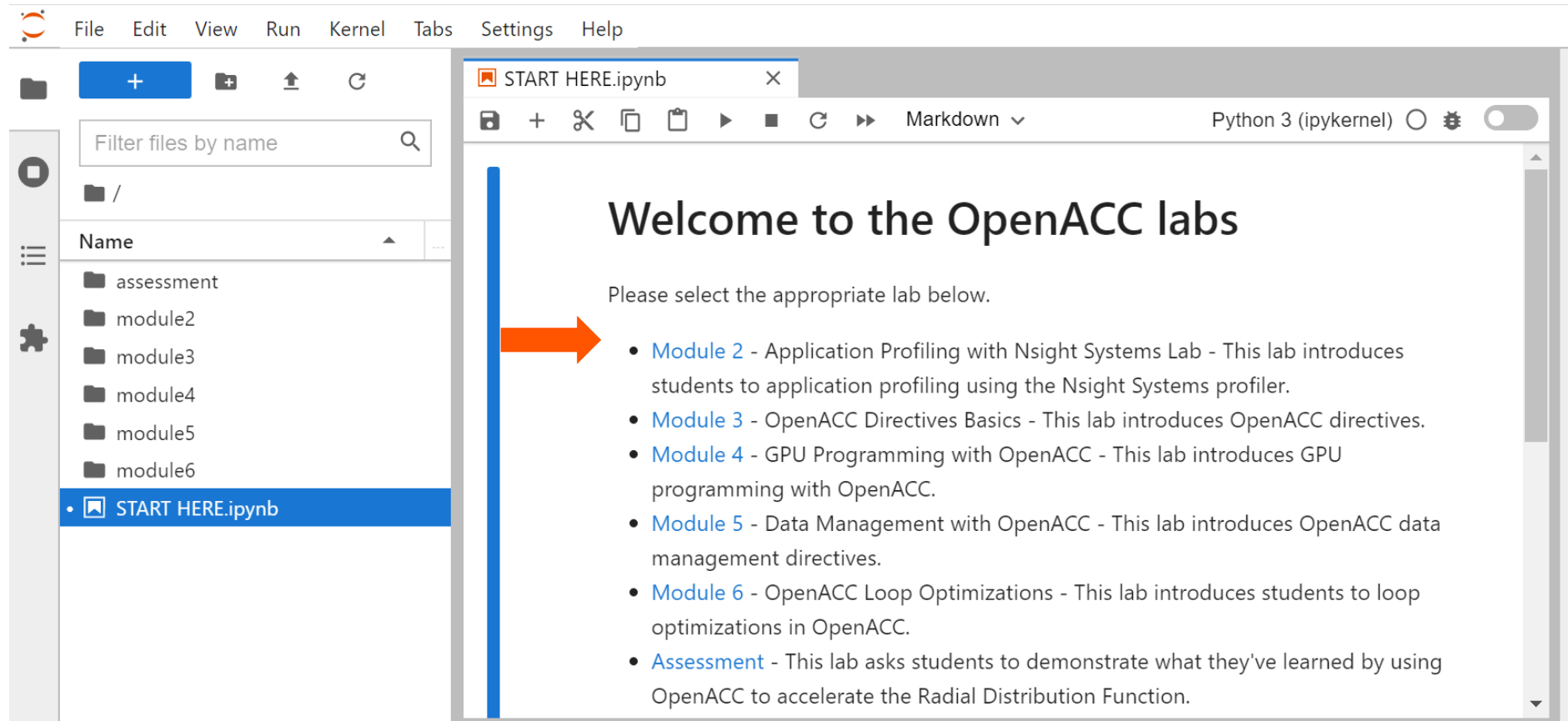
■  
STOP TASK

✓  
ASSESS TASK

Please wait 5 - 10 minutes while your interactive GPU enabled environment loads. When the "Launch" button appears, click it to get started.



# TRAINING SETUP



The screenshot displays the OpenACC training setup interface. On the left, a file browser shows a directory structure with folders named 'assessment', 'module2', 'module3', 'module4', 'module5', and 'module6'. The 'START HERE.ipynb' file is highlighted in blue. The main window shows the notebook content, which includes a title 'Welcome to the OpenACC labs' and a list of modules to be selected. An orange arrow points to the 'Module 2' entry in the list.

File Edit View Run Kernel Tabs Settings Help

START HERE.ipynb

Python 3 (ipykernel)

## Welcome to the OpenACC labs

Please select the appropriate lab below.

- [Module 2](#) - Application Profiling with Nsight Systems Lab - This lab introduces students to application profiling using the Nsight Systems profiler.
- [Module 3](#) - OpenACC Directives Basics - This lab introduces OpenACC directives.
- [Module 4](#) - GPU Programming with OpenACC - This lab introduces GPU programming with OpenACC.
- [Module 5](#) - Data Management with OpenACC - This lab introduces OpenACC data management directives.
- [Module 6](#) - OpenACC Loop Optimizations - This lab introduces students to loop optimizations in OpenACC.
- [Assessment](#) - This lab asks students to demonstrate what they've learned by using OpenACC to accelerate the Radial Distribution Function.

# TRAINING SETUP

- To be able to visualise Nsight System profiler output during the course, please install Nsight System latest version on your local system before the course. The software can be downloaded from <https://developer.nvidia.com/nsight-systems>.



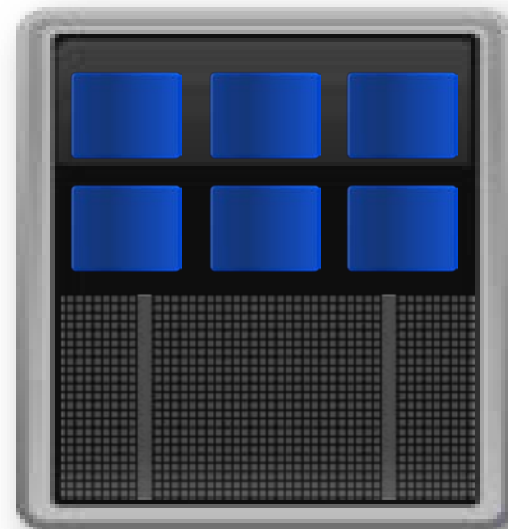
# PROFILING MULTICORE CODE

# PROFILING MULTICORE CODE

## What is multicore?

- *Multicore* refers to using a CPU with multiple computational cores as our parallel device
- These cores can run independently of each other, but have shared access to memory
- Loop iterations can be spread across CPU threads and can utilize SIMD/vector instructions (SSE, AVX, etc.)
- Parallelizing on a multicore CPU is a good starting place, since data management is unnecessary

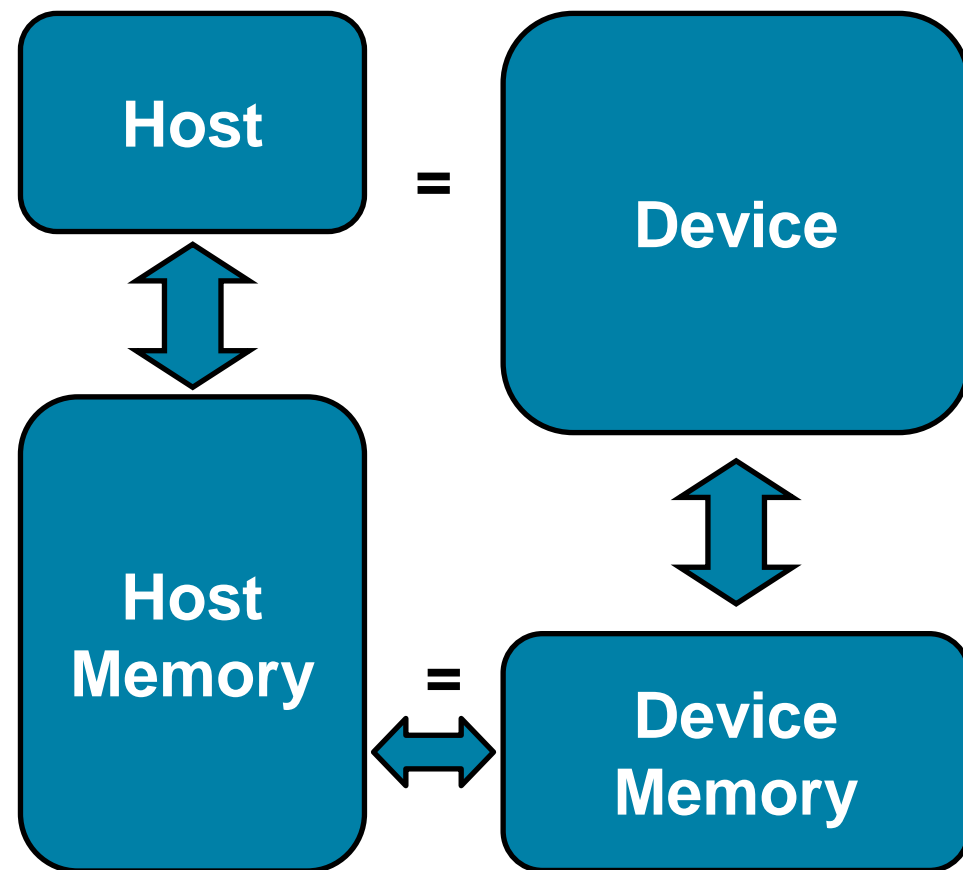
CPU



# PROFILING MULTICORE CODE

## Using a multicore CPU with OpenACC

- OpenACC's generic model involves a combination of a host and a device
- Host generally means a CPU, and the device is some parallel hardware
- When running with a multicore CPU as our device, typically this means that our host/device will be the same
- This also means that their memories will be the same



# PROFILING MULTICORE CODE

## Compiling code for a specific parallel hardware

- The '-ta' flag will allow us to compile our code for a specific, target parallel hardware
- 'ta' stands for "Target Accelerator," an accelerator being another way to refer to a parallel hardware
- Our OpenACC code can be compiled for many different kinds of parallel hardware without having to change the code

```
$ pgcc -fast -Minfo=accel -ta=multicore laplace2d.c  
calcNext:  
    35, Generating Multicore code  
    36, #pragma acc loop gang
```

# PROFILING MULTICORE CODE

## Compiling code for a specific parallel hardware

- nsys profile -t nvtx --stats=true --force-overwrite true -o laplace\_parallel ./laplace\_parallel

NVTX Push-Pop Range Statistics:

Time (%)	Total Time (ns)	Instances	Average	Minimum	Maximum	Range
49.9	24908340742	1	24908340742.0	24908340742	24908340742	while
26.4	13167317033	1000	13167317.0	9986457	52044034	calc
23.4	11711313301	1000	11711313.3	8693117	62627309	swap
0.4	175394843	1	175394843.0	175394843	175394843	init

Report file moved to `"/home/openacc/labs/module2/English/C/laplace_parallel.qdrep"`

Report file moved to `"/home/openacc/labs/module2/English/C/laplace_parallel.sqlite"`

# PROFILING OPENACC CODE

# PARALLEL VS SEQUENTIAL

## Compiler feedback

Have a close look at the PGI compiler feedback for both sequential and parallel implementation of the application.

It provides information about how your program was optimized or why a particular optimization was not made.

Note: Adding `-Minfo` flag or `-Minfo=accel` or `-Minfo=all` when compiling, will enable compiler feedback messages, giving details about the parallel code generated.

```
jacobi.c:
main:
    59, Loop not vectorized/parallelized: potential early exits
laplace2d.c:
"laplace2d.c", line 24: warning: function "memset" declared implicitly
    memset(A, 0, n * m * sizeof(double));
    ^
initialize:
    27, Generated vector simd code for the loop
calcNext:
    40, Generated vector simd code for the loop containing reductions
swap:
    56, Memory copy idiom, loop replaced by call to __c_mcopy8
```

Sequential

```
jacobi.c:
main:
    59, Loop not vectorized/parallelized: potential early exits
laplace2d.c:
"laplace2d.c", line 24: warning: function "memset" declared implicitly
    memset(A, 0, n * m * sizeof(double));
    ^
initialize:
    27, Generated vector simd code for the loop
calcNext:
    35, Generating Tesla code
    37, #pragma acc loop gang /* blockIdx.x */
        Generating implicit reduction(max:error)
    40, #pragma acc loop vector(128) /* threadIdx.x */
    35, Generating implicit copyin(A[:]) [if not already present]
        Generating implicit copy(error) [if not already present]
        Generating implicit copyout(Anew[:]) [if not already present]
    40, Loop is parallelizable
swap:
    51, Generating Tesla code
    53, #pragma acc loop gang /* blockIdx.x */
    56, #pragma acc loop vector(128) /* threadIdx.x */
    51, Generating implicit copyout(A[:]) [if not already present]
        Generating implicit copyin(Anew[:]) [if not already present]
    56, Loop is parallelizable
```

Parallel

```

#include <math.h>
#include <stdlib.h>

#define OFFSET(x, y, m) (((x)*(m)) + (y))

void initialize(double *restrict A, double *restrict Anew, int m, int n)
{
    memset(A, 0, n * m * sizeof(double));
    memset(Anew, 0, n * m * sizeof(double));

    for(int i = 0; i < m; i++){
        A[i] = 1.0;
        Anew[i] = 1.0;
    }
}

double calcNext(double *restrict A, double *restrict Anew, int m, int n)
{
    double error = 0.0;
    #pragma acc parallel loop reduction(max:error)

    for( int j = 1; j < n-1; j++)
    {
        #pragma acc loop
        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 = max( error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i, m)]));
        }
    }
    return error;
}

void swap(double *restrict A, double *restrict Anew, int m, int n)
{
    #pragma acc parallel loop
    for( int j = 1; j < n-1; j++)
    {
        #pragma acc loop
        for( int i = 1; i < m-1; i++ )
        {
            A[OFFSET(j, i, m)] = Anew[OFFSET(j, i, m)];
        }
    }
}

void deallocate(double *restrict A, double *restrict Anew)
{
    free(A);
    free(Anew);
}

```

laplace2d.c  
 (Parallelised using OpenACC parallel  
 directives (pragmas highlighted))

```

Importing [=====100%]
Saved report file to "/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-par.qdrep"
Exporting 59427 events: [=====100%]

Exported successfully to
/home/mozhgank/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel/laplace-par.sqlite

Generating CUDA API Statistics...
CUDA API Statistics (nanoseconds)

```

Time (%)	Total Time	Calls	Average	Minimum	Maximum	Name
93.1	1107821913	4000	276955.5	2205	63176580	cuStreamSynchronize
2.2	25667004	1	25667004.0	25667004	25667004	cuMemHostAlloc
1.8	21314088	1	21314088.0	21314088	21314088	cuMemAllocManaged
1.6	18919205	3000	6306.4	4141	38827	cuLaunchKernel
0.6	6575301	1000	6575.3	5719	37100	cuMemcpyDtoHAsync_v2
0.4	4672638	1000	4672.6	3898	23307	cuMemcpyD32Async
0.2	1957950	1000	1958.0	1656	9112	cuEventRecord
0.2	1835300	1000	1835.3	1685	15602	cuEventSynchronize
0.1	1013630	1	1013630.0	1013630	1013630	cuMemAllocHost_v2
0.0	498778	3	166259.3	7731	462241	cuMemAlloc_v2
0.0	251874	1	251874.0	251874	251874	cuModuleLoadDataEx
0.0	9531	1	9531.0	9531	9531	cuStreamCreate
0.0	3936	2	1968.0	1136	2800	cuEventCreate

Generating CUDA Kernel Statistics...  
 CUDA Kernel Statistics (nanoseconds)

Time (%)	Total Time	Instances	Average	Minimum	Maximum	Name
58.9	643260973	1000	643261.0	563448	63174901	calcNext_35_gpu
40.4	441862005	1000	441862.0	434490	452729	swap_51_gpu
0.7	7350076	1000	7350.1	6880	7936	calcNext_35_gpu_red

Generating CUDA Memory Operation Statistics...  
 CUDA Memory Operation Statistics (nanoseconds)

Time (%)	Total Time	Operations	Average	Minimum	Maximum	Name
94.0	47346144	8191	5780.3	2688	71456	[CUDA Unified Memory memcpy HtoH]
3.1	1567908	1000	1567.9	1535	2816	[CUDA memcpy DtoH]
2.9	1447303	1000	1447.3	1407	2080	[CUDA memset]

CUDA Memory Operation Statistics (KiB)

Total	Operations	Average	Minimum	Maximum	Name
7.813	1000	0.008	0.008	0.008	[CUDA memcpy DtoH]
7.813	1000	0.008	0.008	0.008	[CUDA memset]
260072.000	8191	31.751	4.000	732.000	[CUDA Unified Memory memcpy HtoH]

Generating NVTX Push-Pop Range Statistics...  
 NVTX Push-Pop Range Statistics (nanoseconds)

Time (%)	Total Time	Instances	Average	Minimum	Maximum	Range
47.6	1199130726	1	1199130726.0	1199130726	1199130726	while
29.2	734660957	1000	734661.0	627348	89307285	calc
18.3	461447578	1000	461447.6	454563	490929	swap
4.9	122270944	1	122270944.0	122270944	122270944	init

CUDA API statistics

CUDA Kernel statistics

CUDA Memory Operation statistics

NVTX range statistics

"calc" region (calcNext function) takes 29.2%  
 "swap" region (swap function) takes 18.3% of  
 total execution time

Open laplace-par.qdrep  
 with Nsight System GUI to  
 view the timeline



# PARALLEL VS SEQUENTIAL SPEEDUP

## Viewing captured NVTX events

Have a close look at the captured NVTX events for both serial and parallel implementations.

Time spent in “while” loop has significantly decreased.

Achieved speedup: ~47

```
mozhgank@prm-dgx-32:~/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel$ nsys profile -t nvtx,openacc --stats=true -b dwarf --force-overwrite true -o laplace-par ./laplace-par
WARNING: openACC, cuDNN and cuBLAS rely on CUDA. CUDA tracing has been automatically enabled.
Collecting data...
Jacobi relaxation calculation: 4096 x 4096 mesh
0, 0.250000
100, 0.002397
200, 0.001204
300, 0.000804
400, 0.000603
500, 0.000483
600, 0.000403
700, 0.000345
800, 0.000302
900, 0.000269
total: 1.199133 s
```

Parallel

```
mozhgank@prm-dgx-32:~/Code/openacc-training-materials/labs/module4/English/C/solutions/parallel$ nsys profile -t nvtx --stats=true -b dwarf --force-overwrite true -o laplace-seq ./laplace-seq
Collecting data...
Jacobi relaxation calculation: 4096 x 4096 mesh
0, 0.250000
100, 0.002397
200, 0.001204
300, 0.000804
400, 0.000603
500, 0.000483
600, 0.000403
700, 0.000345
800, 0.000302
900, 0.000269
total: 55.754501 s
```

Sequential

#	Name	Duration	TID	Start
1	init [122.271 ms]	122.271 ms	45547	0.84707s
3	while [1.199 s]	1.199 s	45547	0.969399s

Parallel

#	Name	Duration	TID	Start
1	init [137.490 ms]	137.490 ms	46133	0.00975564s
3	while [55.754 s]	55.754 s	46133	0.147279s

Sequential

# PROFILING PARALLEL CODE

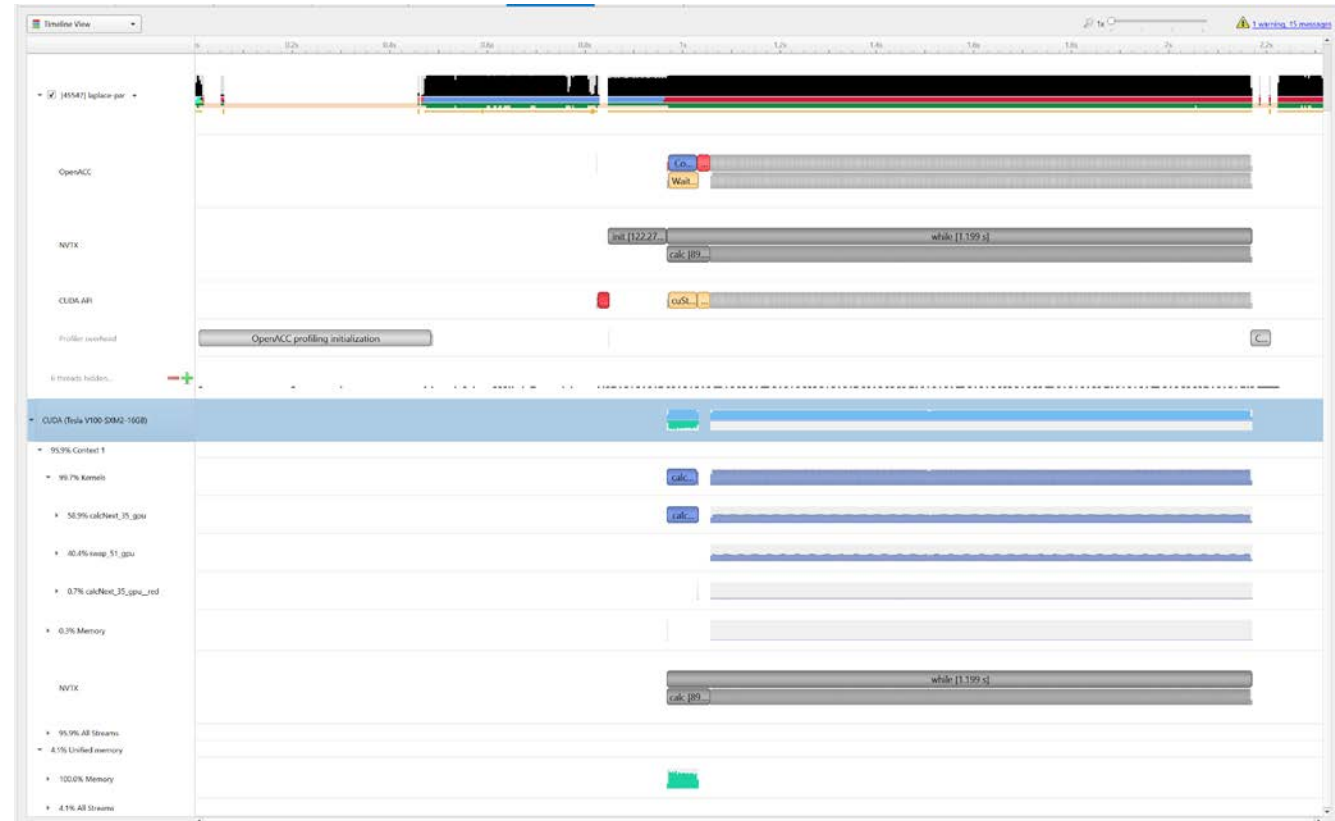
## Viewing timeline via Nsight Systems

Contents of the tree-like hierarchy on the left depend on the project settings used to collect this report.

If a certain feature has not been enabled, corresponding rows will not be shown on the timeline.

In this example, we chose to trace NVTX and OpenACC while sampling.

Note: Kernel launches are represented by **blue** and memory transfers are displayed in **green**.



# LAB CODE

# LAPLACE HEAT TRANSFER

## Introduction to lab code - visual

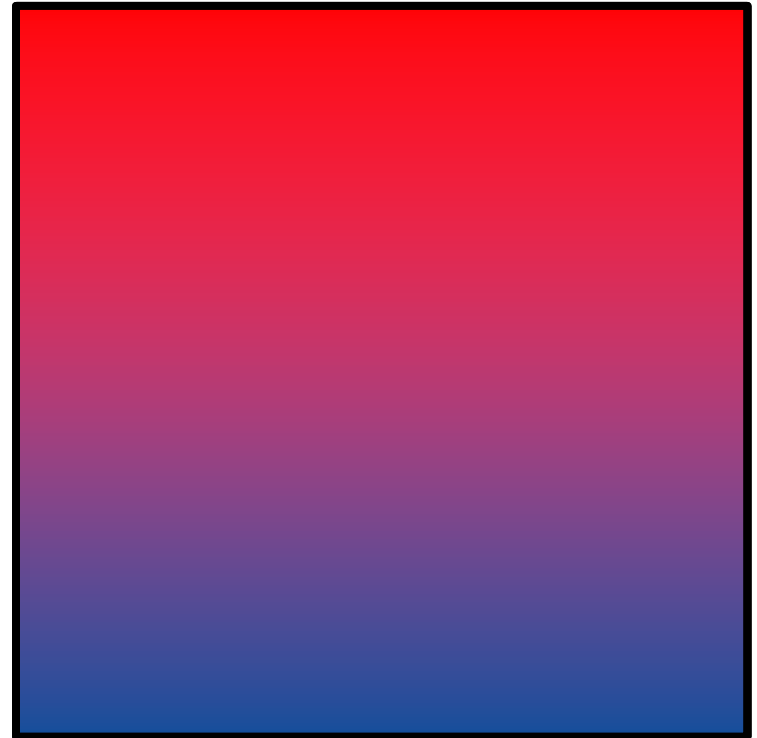
We will observe a simple simulation of heat distributing across a metal plate.

We will apply a consistent heat to the top of the plate.

Then, we will simulate the heat distributing across the plate.

Very Hot

Room Temp



# LAPLACE HEAT TRANSFER

## Introduction to lab code - technical

The lab simulates a very basic 2-dimensional heat transfer problem. We have two 2-dimensional arrays, **A** and **Anew**.

The arrays represent a 2-dimensional, metal plate. Each element in the array is a **double** value that represents temperature.

We will simulate the distribution of heat until a **minimum change value** is achieved, or until we exceed a **maximum number of iterations**.

A

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Anew

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

# LAPLACE HEAT TRANSFER

## Introduction to lab code - technical

We initialize the top row to be a temperature of 1.0

The **calcNext** function will iterate through all of the inner elements of array A, and update the corresponding elements in Anew

We will take the average of the neighboring cells, and record it in **Anew**.

The **swap** function will copy the contents of Anew to A

A

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Anew

0.0	0.0	0.0	0.0
0.0	0.25	0.25	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

# LAPLACE HEAT TRANSFER

## Introduction to lab code

A

1.0	1.0	1.0	1.0
0.0	0.25	0.25	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



Anew

1.0	1.0	1.0	1.0
0.0	0.25	0.25	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

The **swap** function will copy the contents of Anew to A

# KEY CONCEPTS

In this module we discussed...

- Compiling sequential and parallel code
- CPU profiling for sequential and parallel execution
- Specifics of our Laplace Heat Transfer lab code



# LAB GOALS

In this lab you will do the following...

- Build and run the example code using the NVIDIA's HPC compiler
- Use Nsight Systems to understand where the program spends its time

# THANK YOU