PRACE Workshop: Deep Learning and GPU programming workshop

15 – 18 June 2020

MODULE TWO: PROFILING

Dr. Volker Weinberg | LRZ | 16.06.2020

MODULE OVERVIEW

Topics to be covered

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

COMPILING SEQUENTIAL CODE

PGI COMPILER BASICS

pgcc, pgc++ and pgfortran

- The command to compile C code is ‘pgcc’
- The command to compile C++ code is ‘pgc++’
- The command to compile Fortran code is ‘pgfortran’
- The -fast flag instructs the compiler to optimize the code to the best of its abilities

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

PGI COMPILER BASICS

-Minfo flag

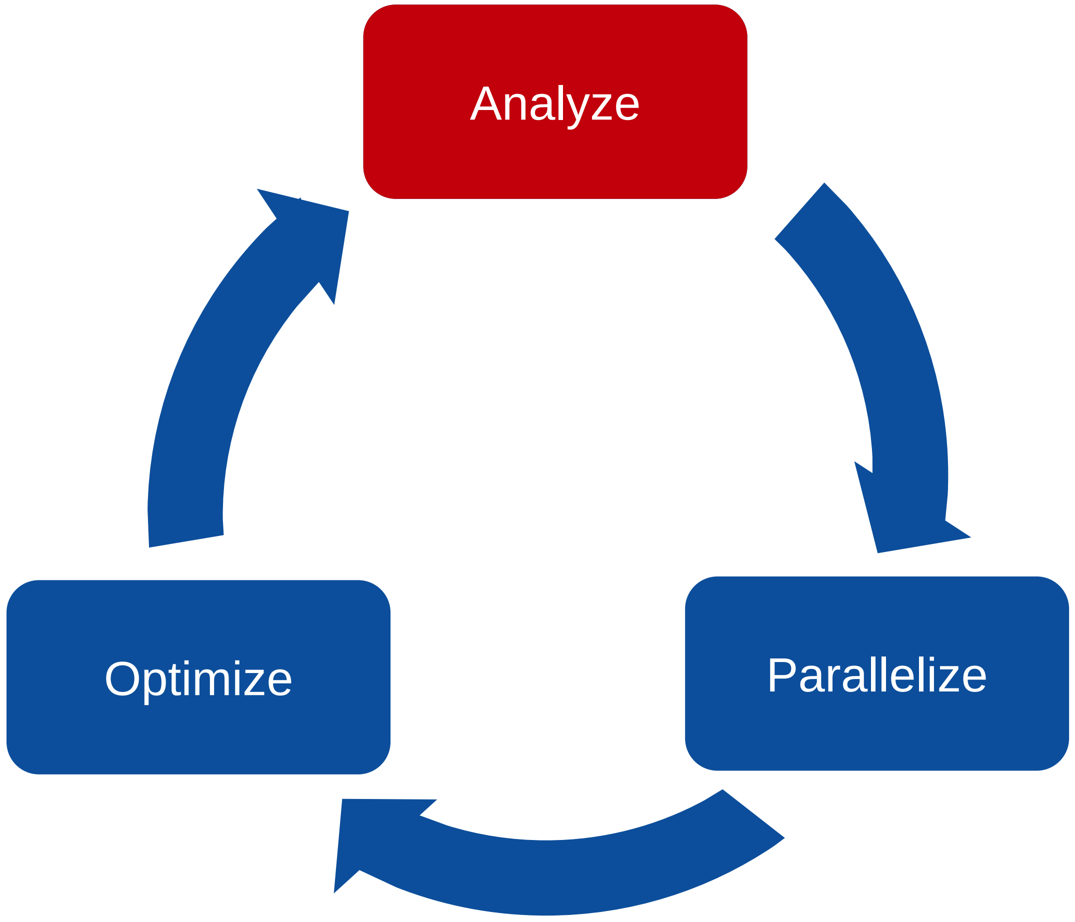
- The Minfo flag will instruct the compiler to print feedback about the compiled code
- -Minfo=accel will give us information about what parts of the code were accelerated via OpenACC
- -Minfo=opt will give information about all code optimizations
- -Minfo=all will give all code feedback, whether positive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

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

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

```
$ pgcc –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

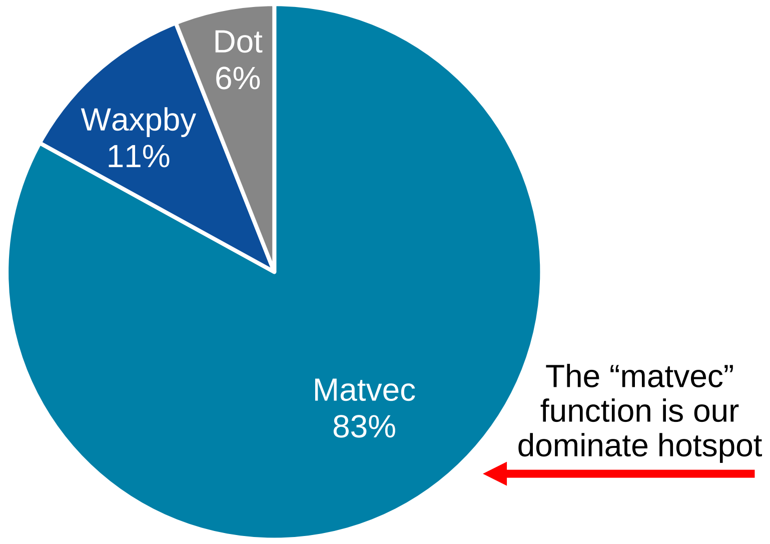
Obtain detailed information about how the code ran.

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.

Sample Code: Conjugate Gradient

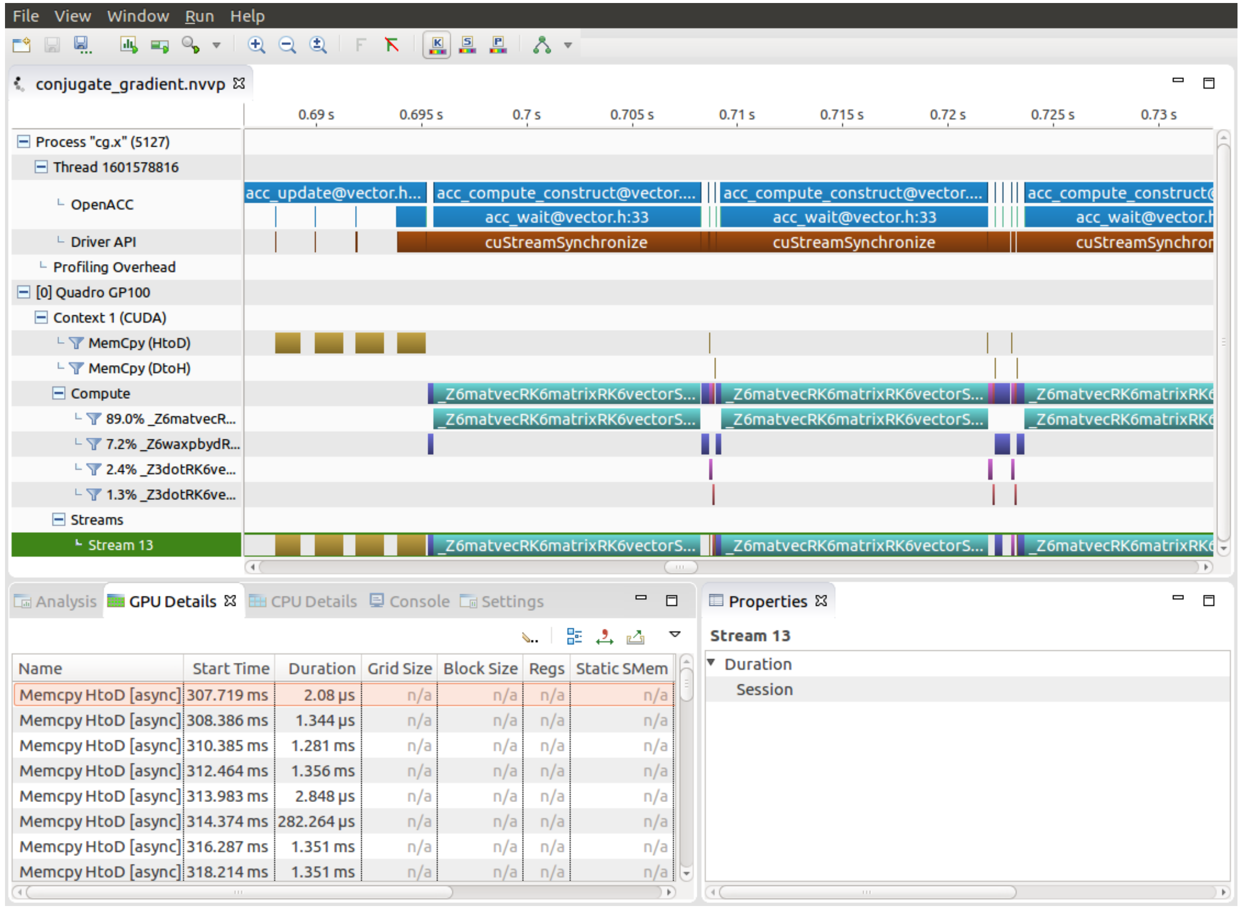
Total Runtime: 22.38 seconds



PROFILING SEQUENTIAL CODE

Introduction to PGProf

- Gives visual feedback of how the code ran
- Gives numbers and statistics, such as program runtime
- Also gives runtime information for individual functions/loops within the code
- Includes many extra features for profiling parallel code

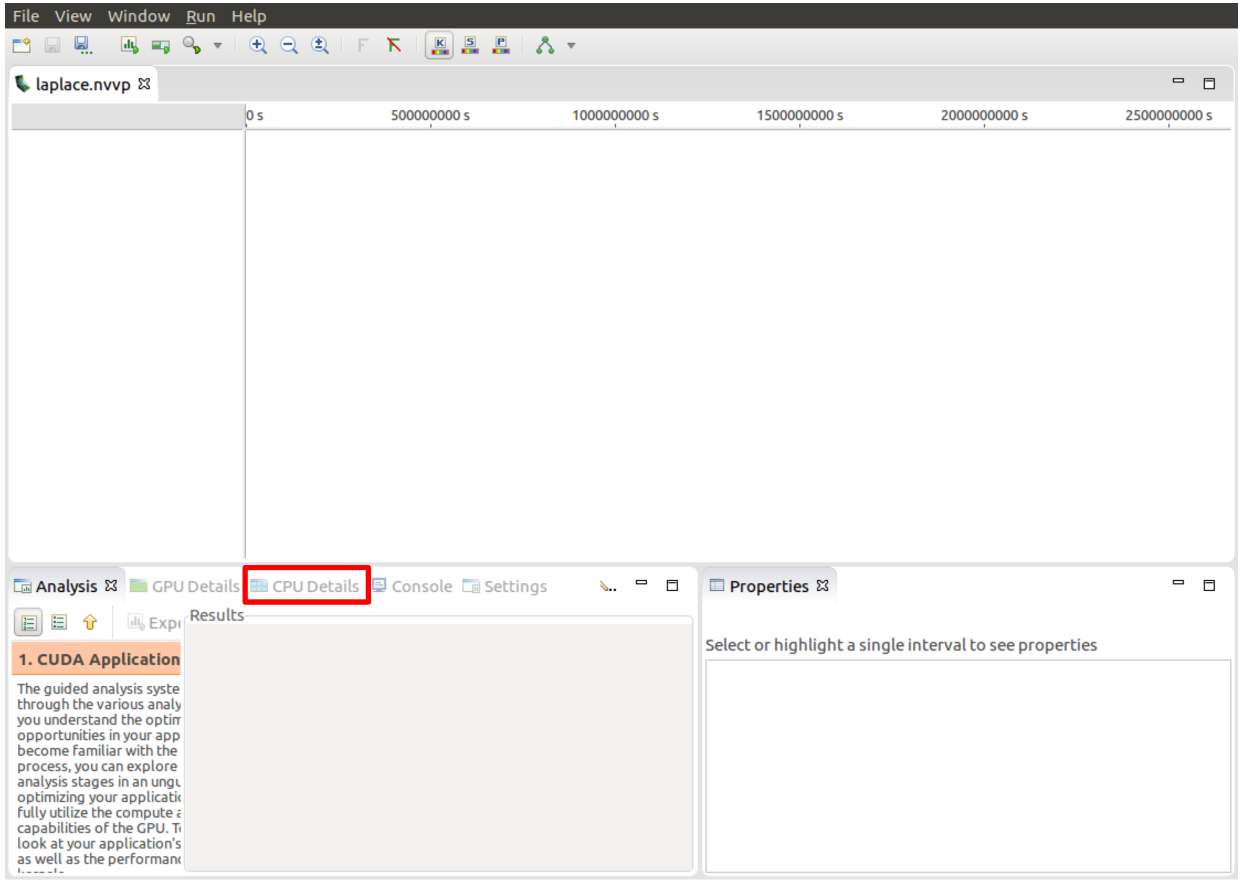


Name	Start Time	Duration	Grid Size	Block Size	Regs	Static SMem
Memcpy HtoD [async]	307.719 ms	2.08 μs	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	308.386 ms	1.344 μs	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	310.385 ms	1.281 ms	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	312.464 ms	1.356 ms	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	313.983 ms	2.848 μs	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	314.374 ms	282.264 μs	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	316.287 ms	1.351 ms	n/a	n/a	n/a	n/a
Memcpy HtoD [async]	318.214 ms	1.351 ms	n/a	n/a	n/a	n/a

PROFILING SEQUENTIAL CODE

First sight when using PGPROF

- Profiling a simple, sequential code
- Our sequential program will run on the CPU
- To view information about how our code ran, we should select the “CPU Details” tab



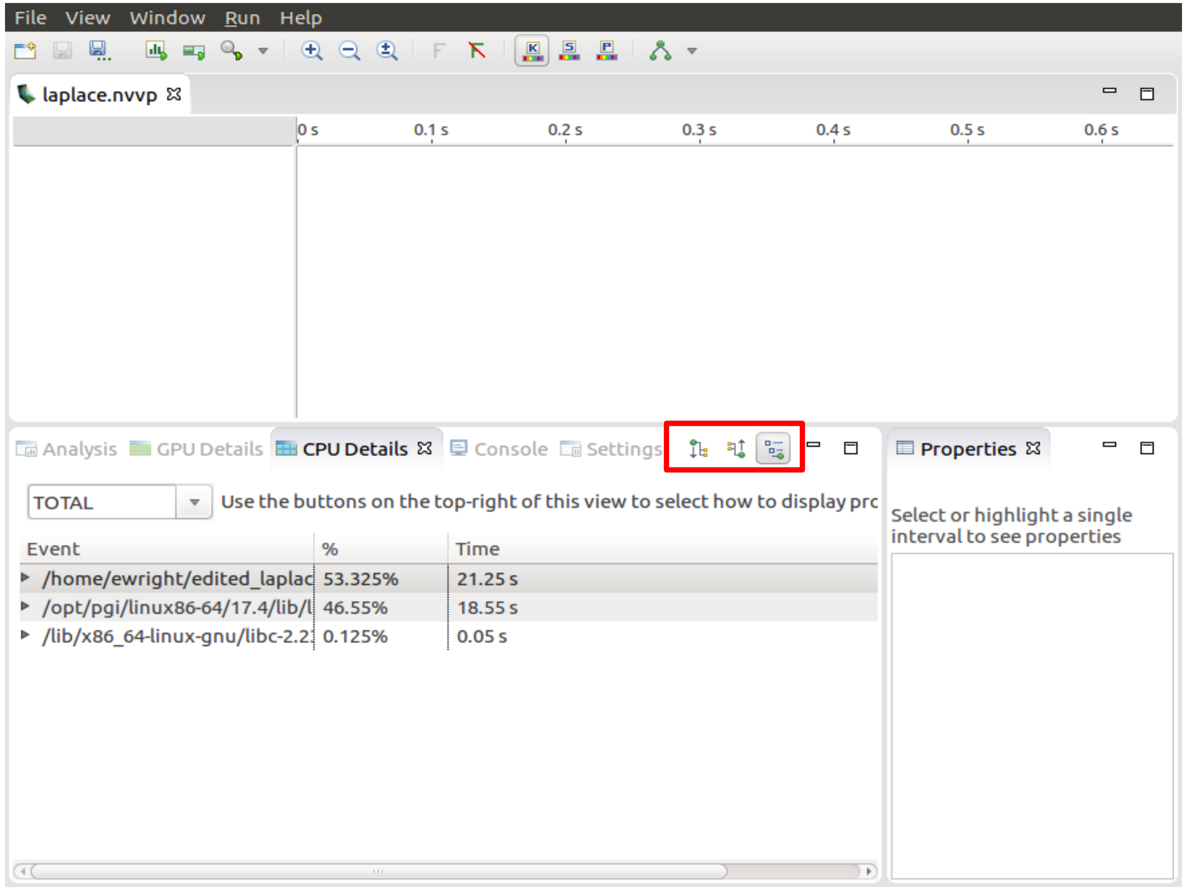
Select or highlight a single interval to see properties

The guided analysis syste
through the various analy
you understand the optim
opportunities in your app
become familiar with the
process, you can explore
analysis stages in an ungu
optimizing your applicati
fully utilize the compute a
capabilities of the GPU. T
look at your application's
as well as the performan

PROFILING SEQUENTIAL CODE

CPU Details

- Within the “CPU Details” tab, we can see the various parts of our code, and how long they took to run
- We can reorganize this info using the three options in the top-right portion of the tab
- We will expand this information, and see more details about our code



Use the buttons on the top-right of this view to select how to display prc

Event	%	Time
/home/ewright/edited_laplac	53.325%	21.25 s
/opt/pgi/linux86-64/17.4/lib/l	46.55%	18.55 s
/lib/x86_64-linux-gnu/libc-2.2	0.125%	0.05 s

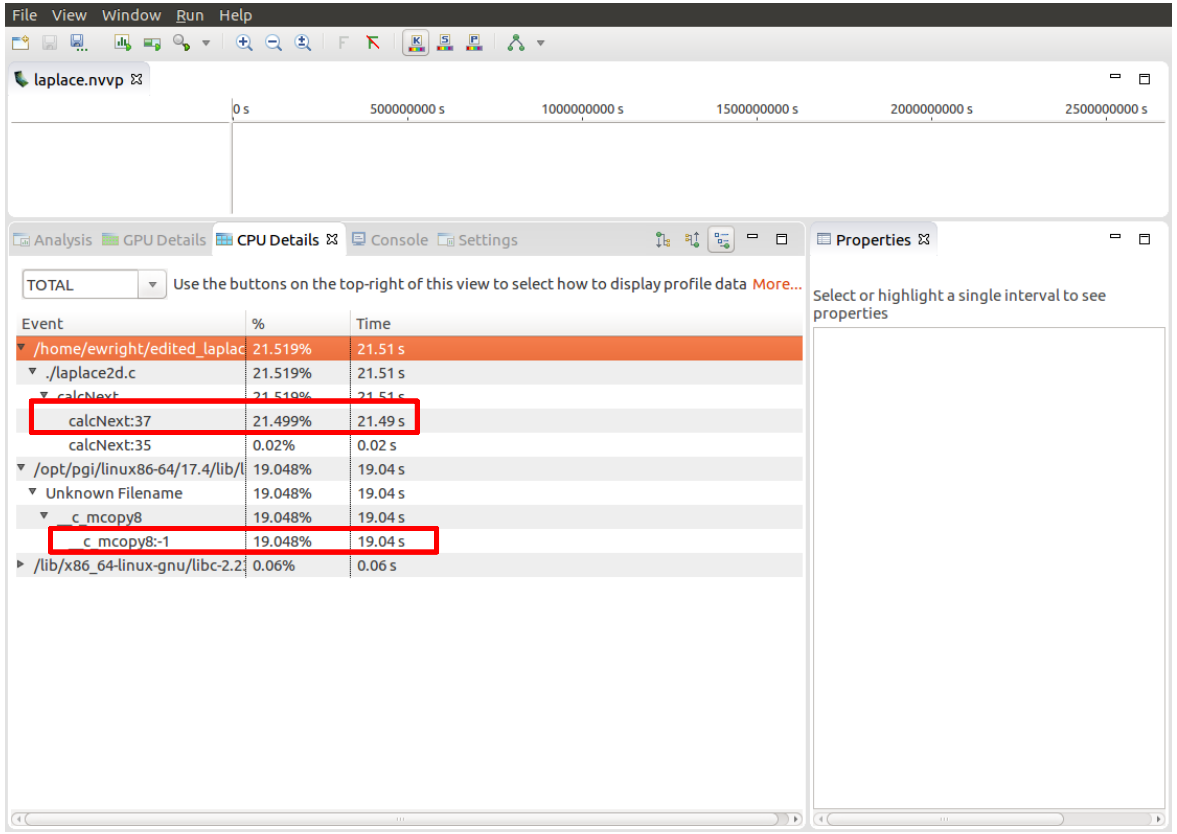
Select or highlight a single interval to see properties

OpenACC
More Science, Less Programming

PROFILING SEQUENTIAL CODE

CPU Details

- We can see that there are two places that our code is spending most of its time
- 21.49 seconds in the “calcNext” function
- 19.04 seconds in a memcpy function
- The c_mcopy8 that we see is actually a compiler optimization that is being applied to our “swap” function



Use the buttons on the top-right of this view to select how to display profile data More...

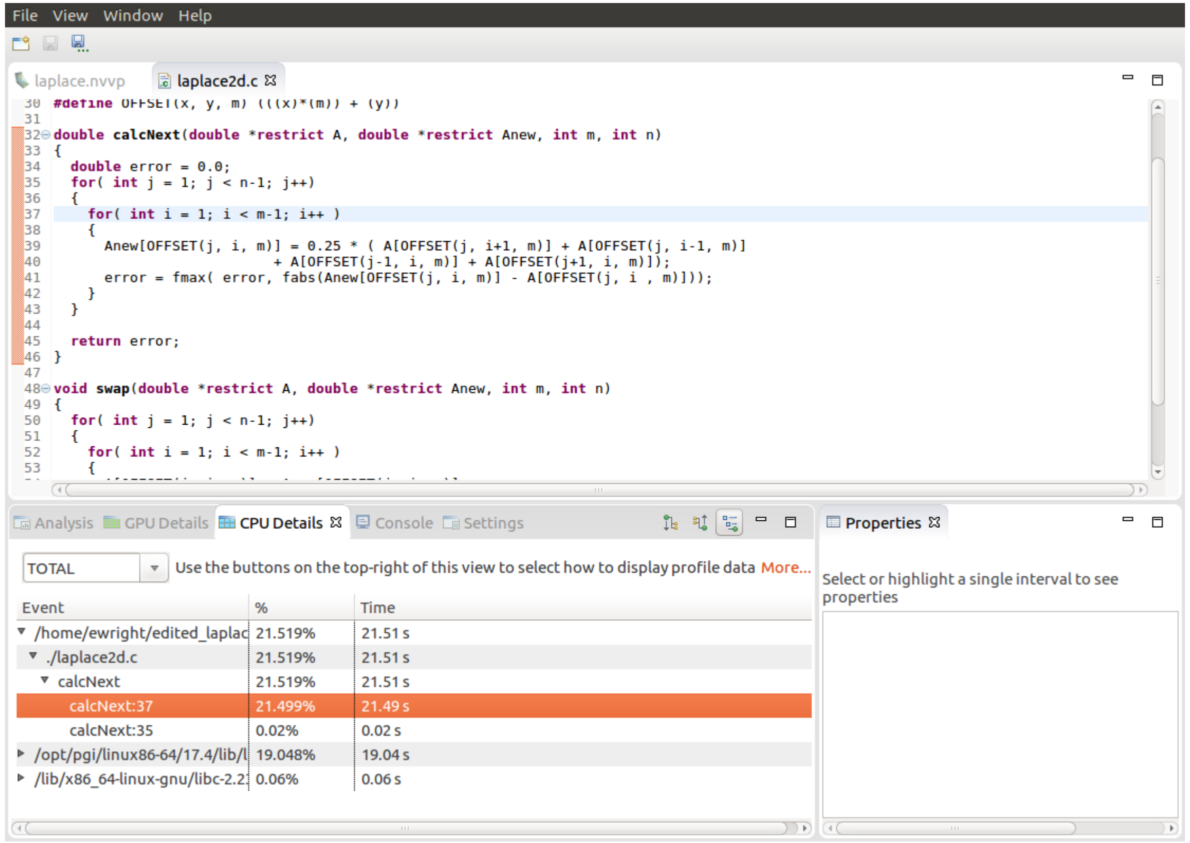
Event	%	Time
/home/ewright/edited_laplac	21.519%	21.51 s
./laplace2d.c	21.519%	21.51 s
calcNext	21.519%	21.51 s
calcNext:37	21.499%	21.49 s
calcNext:35	0.02%	0.02 s
/opt/pgi/linux86-64/17.4/lib/l	19.048%	19.04 s
Unknown Filename	19.048%	19.04 s
__c_mcopy8	19.048%	19.04 s
__c_mcopy8:-1	19.048%	19.04 s
/lib/x86_64-linux-gnu/libc-2.2	0.06%	0.06 s

Select or highlight a single interval to see properties

PROFILING SEQUENTIAL CODE

PGPROF

- We are also able to select the different elements in the CPU Details by double-clicking to open the associated source code
- Here we have selected the “calcNext:37” element, which opened up our code to show the exact line (line 37) that is associated with that element



```
30 #define OFFSET(x, y, m) (((x)*(m)) + (y))
31
32 double calcNext(double *restrict A, double *restrict Anew, int m, int n)
33 {
34   double error = 0.0;
35   for( int j = 1; j < n-1; j++)
36   {
37     for( int i = 1; i < m-1; i++ )
38     {
39       Anew[OFFSET(j, i, m)] = 0.25 * ( A[OFFSET(j, i+1, m)] + A[OFFSET(j, i-1, m)]
40                                     + A[OFFSET(j-1, i, m)] + A[OFFSET(j+1, i, m)]);
41       error = fmax( error, fabs(Anew[OFFSET(j, i, m)] - A[OFFSET(j, i , m)]));
42     }
43   }
44
45   return error;
46 }
47
48 void swap(double *restrict A, double *restrict Anew, int m, int n)
49 {
50   for( int j = 1; j < n-1; j++)
51   {
52     for( int i = 1; i < m-1; i++ )
53     {
```

TOTAL — Use the buttons on the top-right of this view to select how to display profile data More...

Event	%	Time
▾ /home/ewright/edited_laplac	21.519%	21.51 s
▾ ./laplace2d.c	21.519%	21.51 s
▾ calcNext	21.519%	21.51 s
calcNext:37	21.499%	21.49 s
calcNext:35	0.02%	0.02 s
▸ /opt/pgi/linux86-64/17.4/lib/l	19.048%	19.04 s
▸ /lib/x86_64-linux-gnu/libc-2.2	0.06%	0.06 s

Select or highlight a single interval to see properties

PROFILING SEQUENTIAL CODE

Step 2: Profile Your Code

Obtain detailed information about how the code ran.

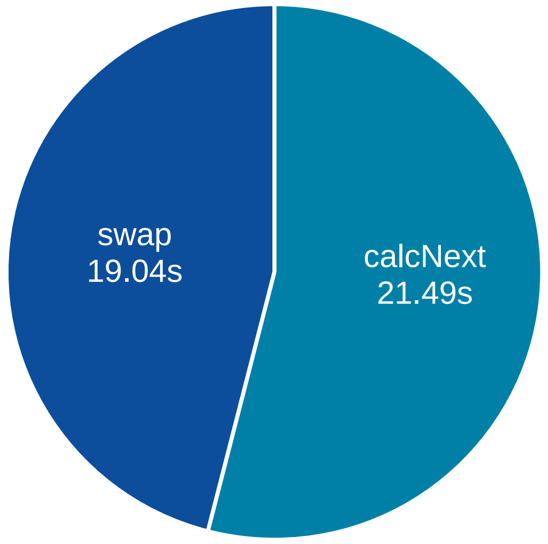
This can include information such as:

- 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



OpenACC
More Science, Less Programming

PROFILING SEQUENTIAL CODE

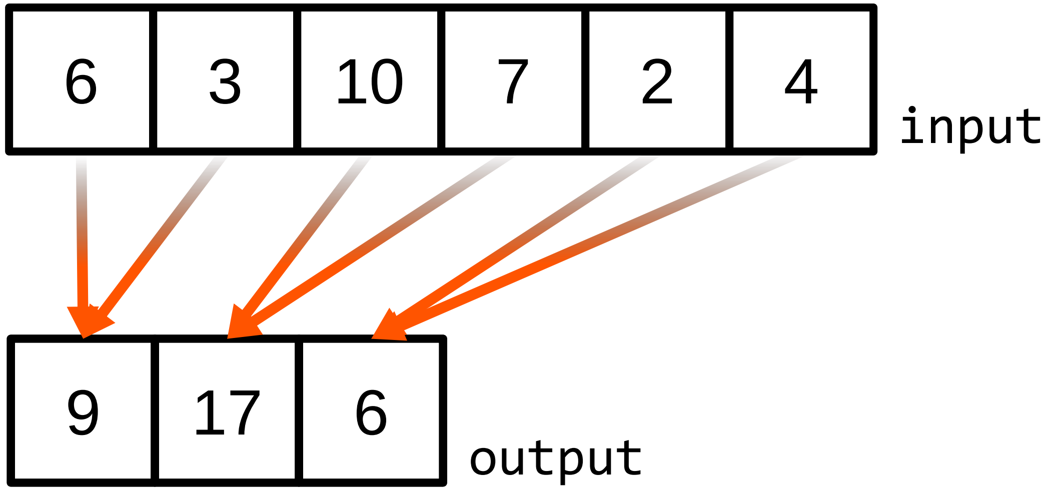
Step 3: Identify Parallelism

Observe the loops contained within the identified hotspots

Are these loops parallelizable?
Can the loop iterations execute independently of each other?
Are the loops multi-dimensional, and does that make them very large?

Loops that are good to parallelize tend to have a lot of iterations to map to parallel hardware.

```
void pairing(int *input, int *output, int N){

  for(int i = 0; i < N; i++)
    output[i] = input[i*2] + input[i*2+1];
}
```



OpenACC
More Science, Less Programming

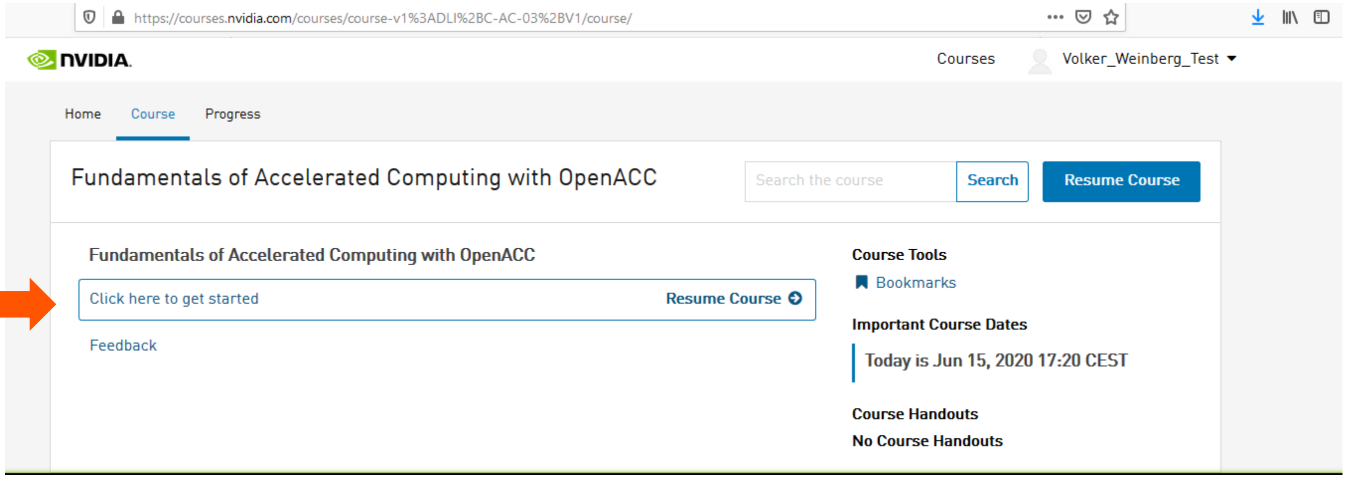
PLEASE START LAB NOW!

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“ (done yesterday)
- Visit http://courses.nvidia.com/dli-event and enter the event code

PRACE_OACC_AMBASSADOR_JU20

TRAINING SETUP


https://courses.nvidia.com/courses/course-v1%3ADLI%2BC-AC-03%2BV1/course/

Fundamentals of Accelerated Computing with OpenACC

Fundamentals of Accelerated Computing with OpenACC

Click here to get started

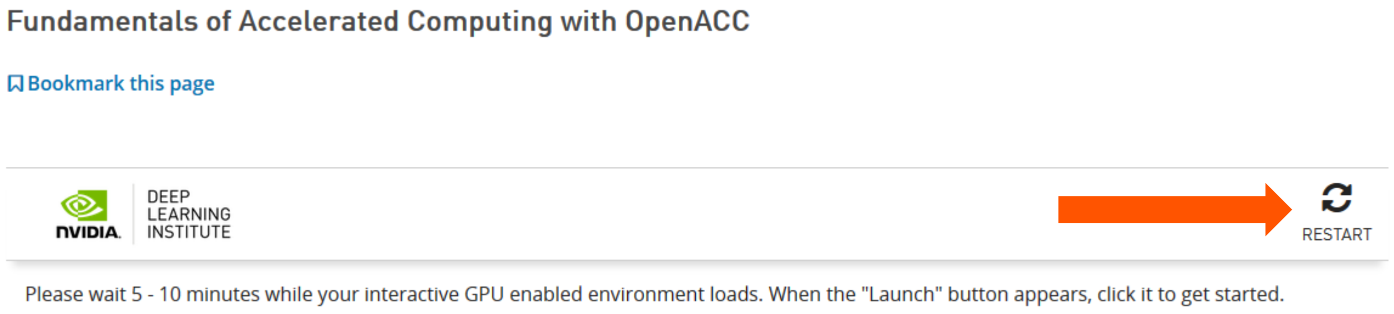
Course Tools

Important Course Dates

Today is Jun 15, 2020 17:20 CEST

Course Handouts

No Course Handouts


Fundamentals of Accelerated Computing with OpenACC

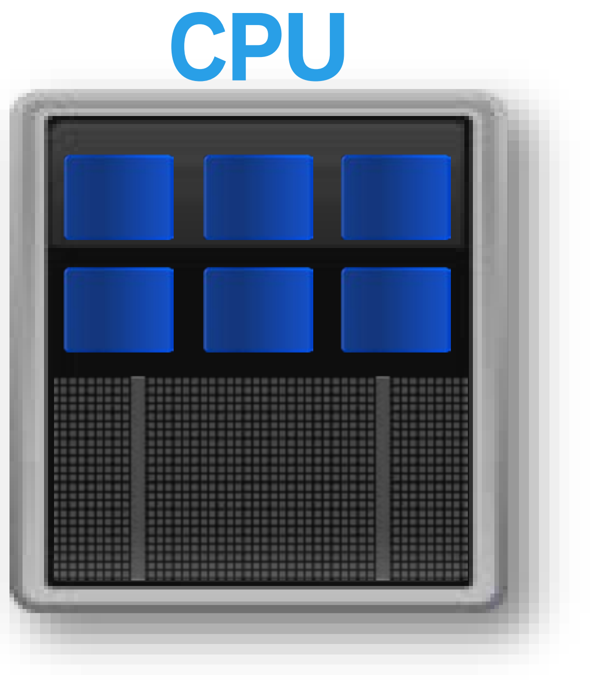
Please wait 5 - 10 minutes while your interactive GPU enabled environment loads. When the "Launch" button appears, click it to get started.

PROFILING MULTICORE CODE

PROFILING MULTICORE CODE

What is multicore?

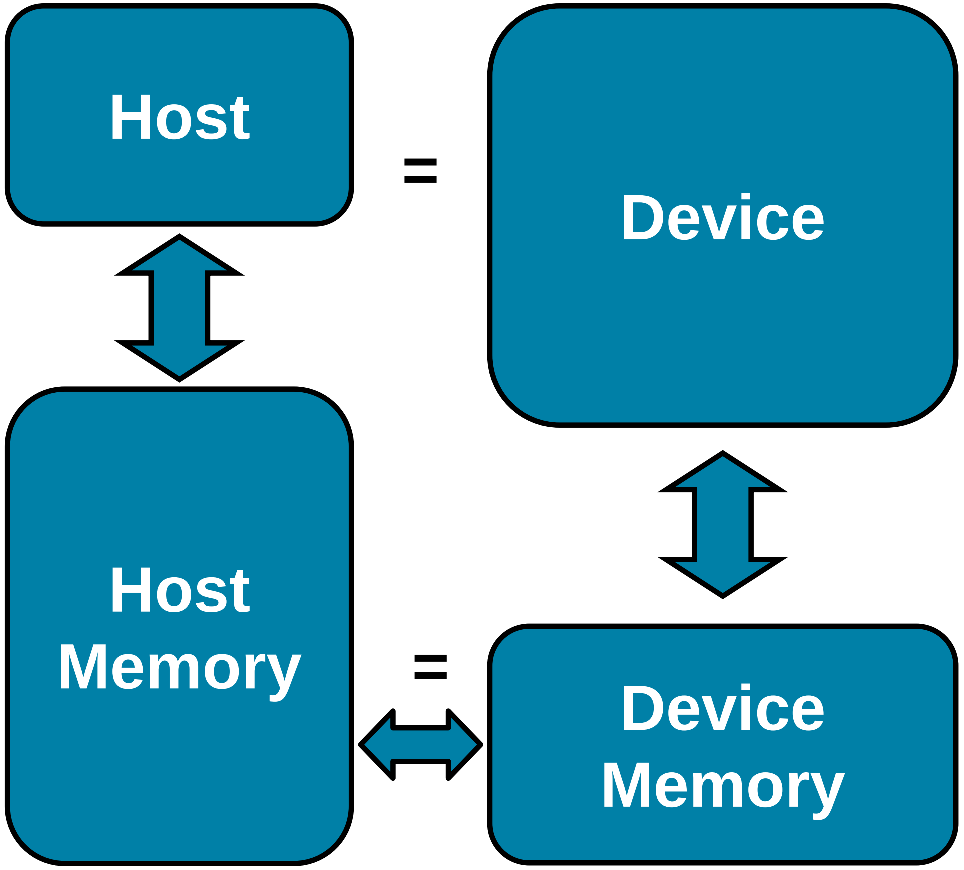
- *Multicore* refers to using a CPU with multiple computational cores as our parallel device
- These cores can run independently of each other, but have shared access to memory
- Loop iterations can be spread across CPU threads and can utilize SIMD/vector instructions (SSE, AVX, etc.)
- Parallelizing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



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

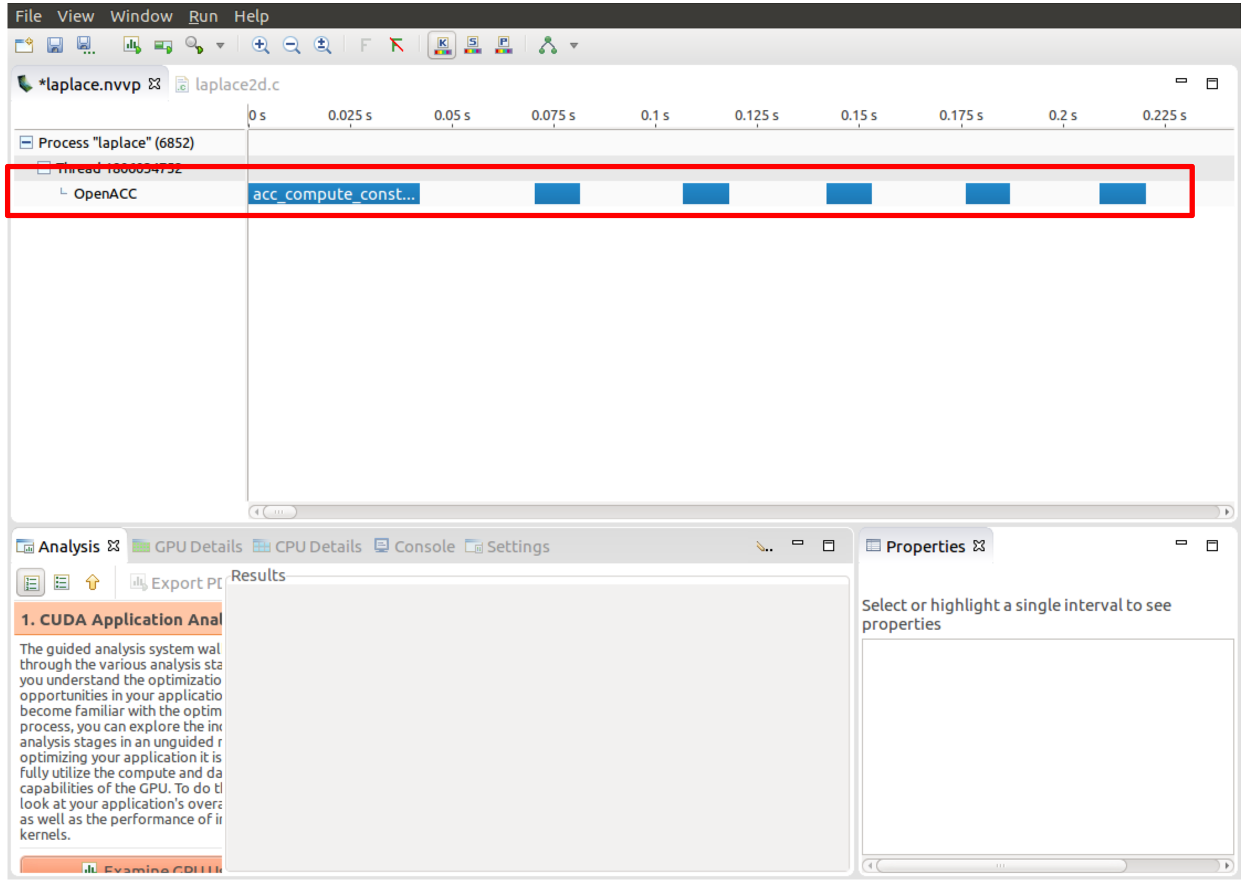
- The '-ta' flag will allow us to compile our code for a specific, target parallel hardware
- 'ta' stands for "Target Accelerator," an accelerator being another way to refer to a parallel hardware
- Our OpenACC code can be compiled for many different kinds of parallel hardware without having to change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

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

PROFILING MULTICORE CODE

PGPROF

- The first difference we see in this multicore profile is that there is now a “timeline”
- This timeline will show when our parallel hardware is being used, and how it is being used
- Each of the blue bars represent a portion of our program that was run on the multicore CPU



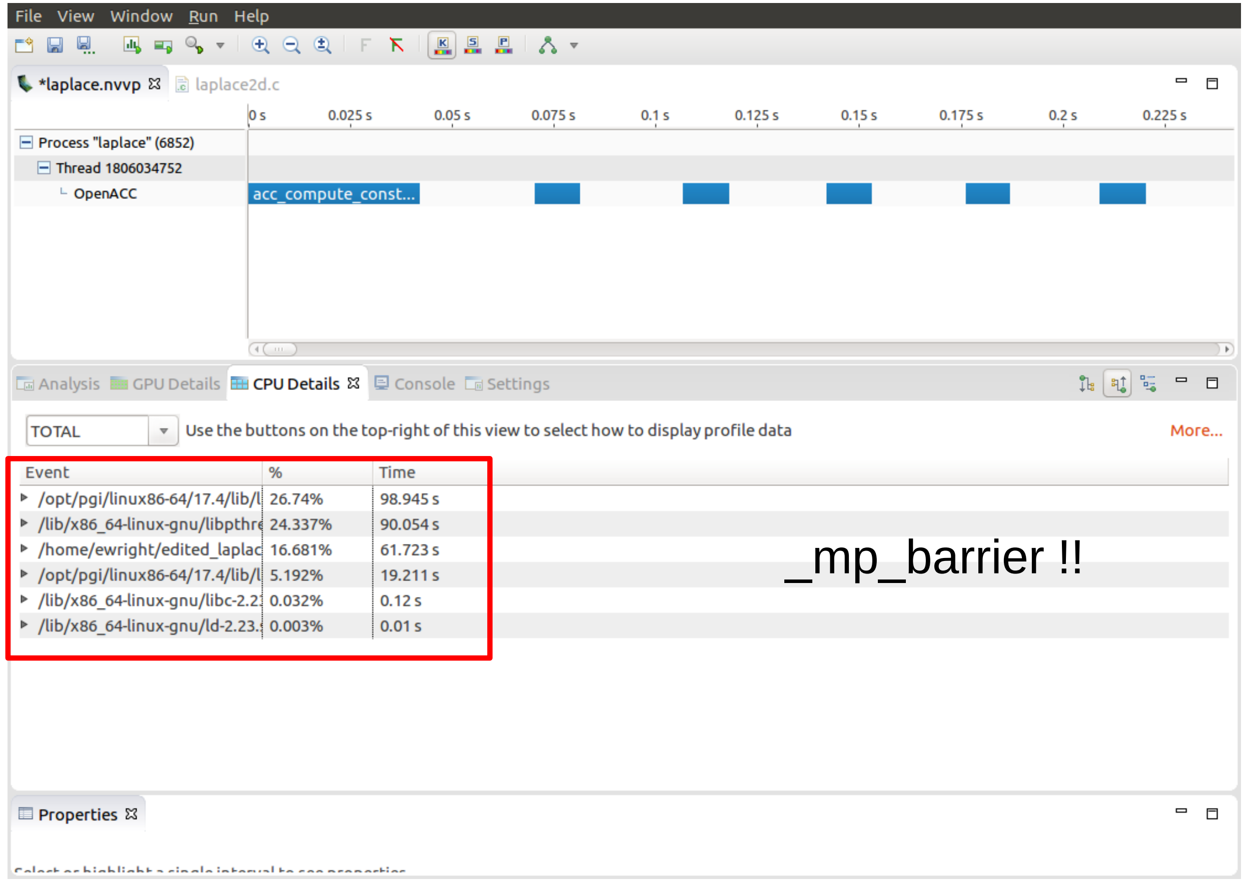
The guided analysis system wal
through the various analysis sta
you understand the optimizatio
opportunities in your applicatio
become familiar with the optim
process, you can explore the in
analysis stages in an unguided r
optimizing your application it is
fully utilize the compute and da
capabilities of the GPU. To do t
look at your application's overa
as well as the performance of i
kernels.

Select or highlight a single interval to see properties

PROFILING MULTICORE CODE

CPU Details

- Looking at our CPU Details, we can see that there is a lot more happening compared to our sequential program
- For the most part, these extra details revolve around the need for the CPU cores to communicate with each other



Use the buttons on the top-right of this view to select how to display profile data

Event	%	Time
/opt/pgi/linux86-64/17.4/lib/l	26.74%	98.945 s
/lib/x86_64-linux-gnu/libpthre	24.337%	90.054 s
/home/ewright/edited_laplac	16.681%	61.723 s
/opt/pgi/linux86-64/17.4/lib/l	5.192%	19.211 s
/lib/x86_64-linux-gnu/libc-2.2	0.032%	0.12 s
/lib/x86_64-linux-gnu/ld-2.23.	0.003%	0.01 s

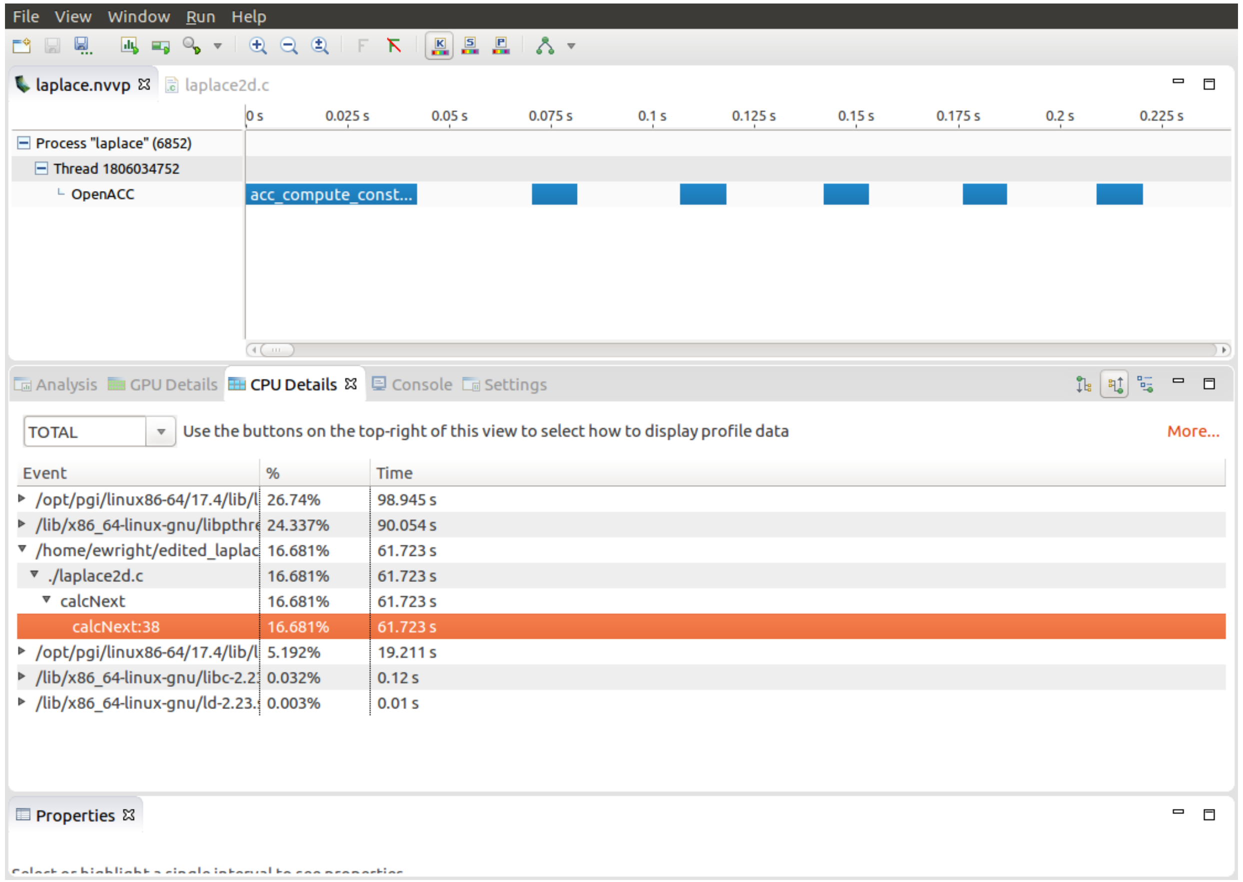
Select or highlight a single interval to see properties

_mp_barrier !!

PROFILING MULTICORE CODE

PGPROF

- Just like earlier, we see our “calcNext” function
- We also see that PGPROF is reporting this function to take 61.72 seconds to run
- Looking at the program now, it looks like it performs much worse than the sequential version



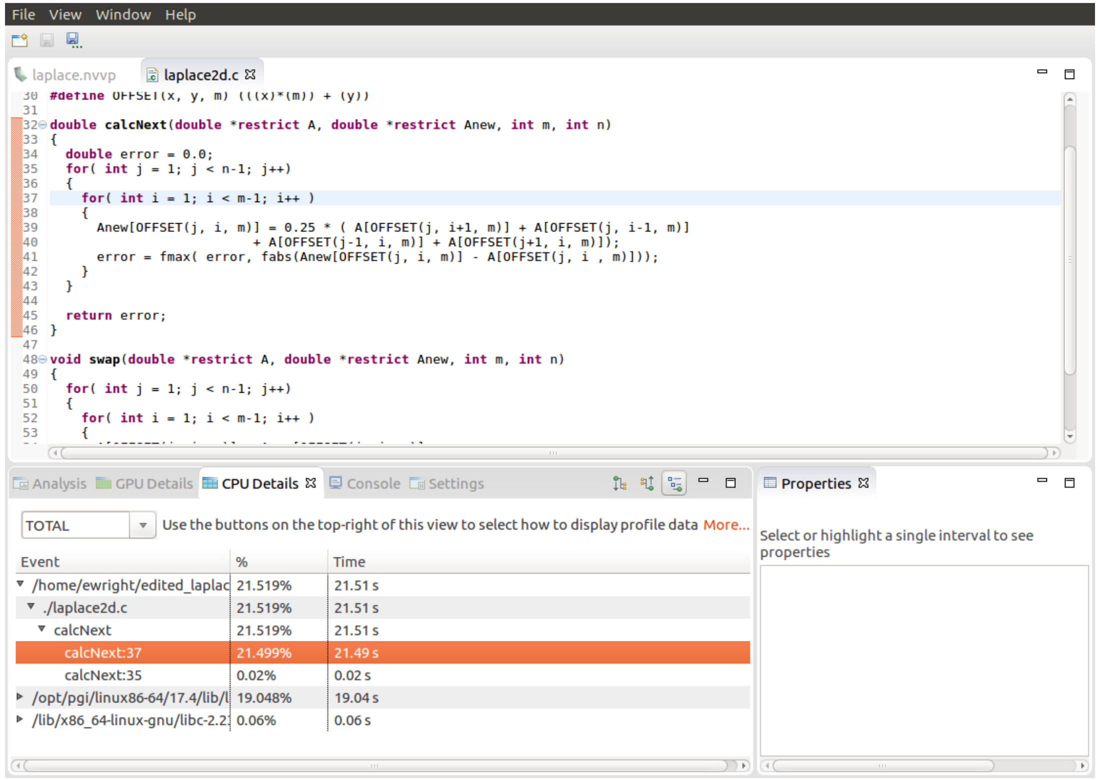
Use the buttons on the top-right of this view to select how to display profile data

Event	%	Time
/opt/pgi/linux86-64/17.4/lib/l	26.74%	98.945 s
/lib/x86_64-linux-gnu/libpthre	24.337%	90.054 s
/home/ewright/edited_laplac	16.681%	61.723 s
./laplace2d.c	16.681%	61.723 s
calcNext	16.681%	61.723 s
calcNext:38	16.681%	61.723 s
/opt/pgi/linux86-64/17.4/lib/l	5.192%	19.211 s
/lib/x86_64-linux-gnu/libc-2.2	0.032%	0.12 s
/lib/x86_64-linux-gnu/ld-2.23.	0.003%	0.01 s

Select or highlight a single interval to see properties

PROFILING MULTICORE CODE

PGPROF

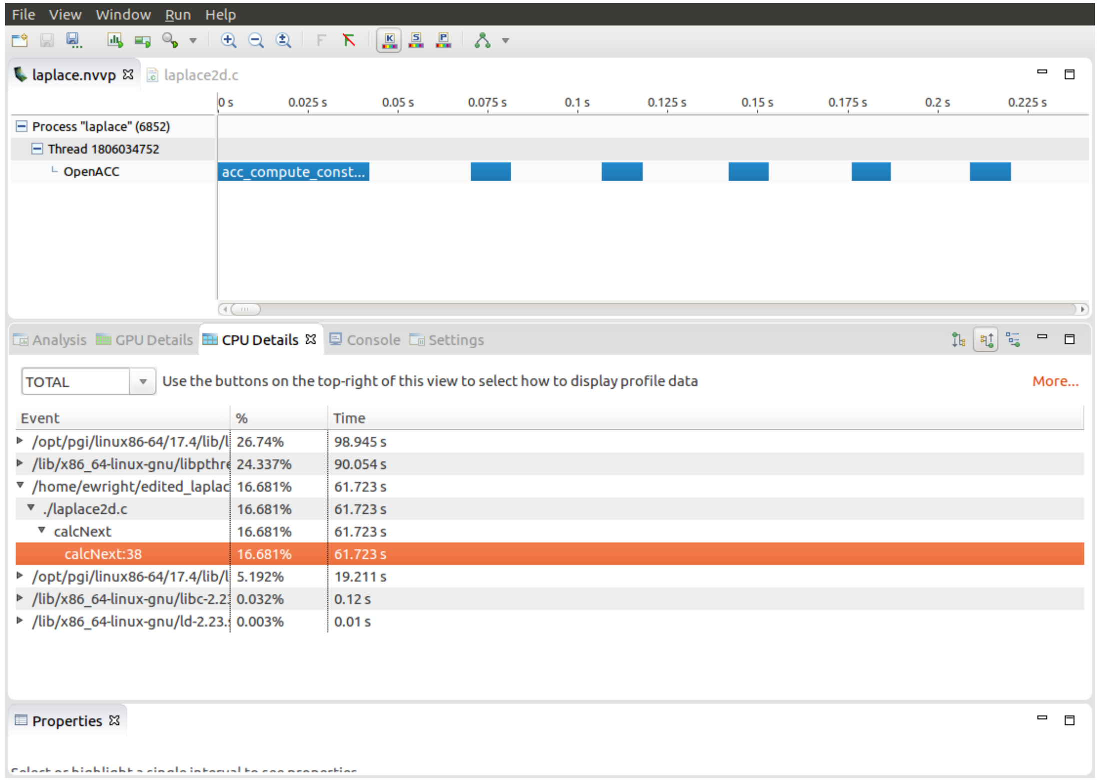


```c
#define OFFSET(x, y, m) (((x)*(m)) + (y))

double calcNext(double *restrict A, double *restrict Anew, int m, int n)
{
  double error = 0.0;
  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)]));
    }
  }

  return error;
}

void swap(double *restrict A, double *restrict Anew, int m, int n)
{
  for( int j = 1; j < n-1; j++)
  {
    for( int i = 1; i < m-1; i++ )
    {
```

TOTAL — Use the buttons on the top-right of this view to select how to display profile data More...

Event	%	Time
/home/ewright/edited_laplac	21.519%	21.51 s
./laplace2d.c	21.519%	21.51 s
calcNext	21.519%	21.51 s
calcNext:37	21.499%	21.49 s
calcNext:35	0.02%	0.02 s
/opt/pgi/linux86-64/17.4/lib/l	19.048%	19.04 s
/lib/x86_64-linux-gnu/libc-2.2	0.06%	0.06 s

Select or highlight a single interval to see properties



TOTAL — Use the buttons on the top-right of this view to select how to display profile data More...

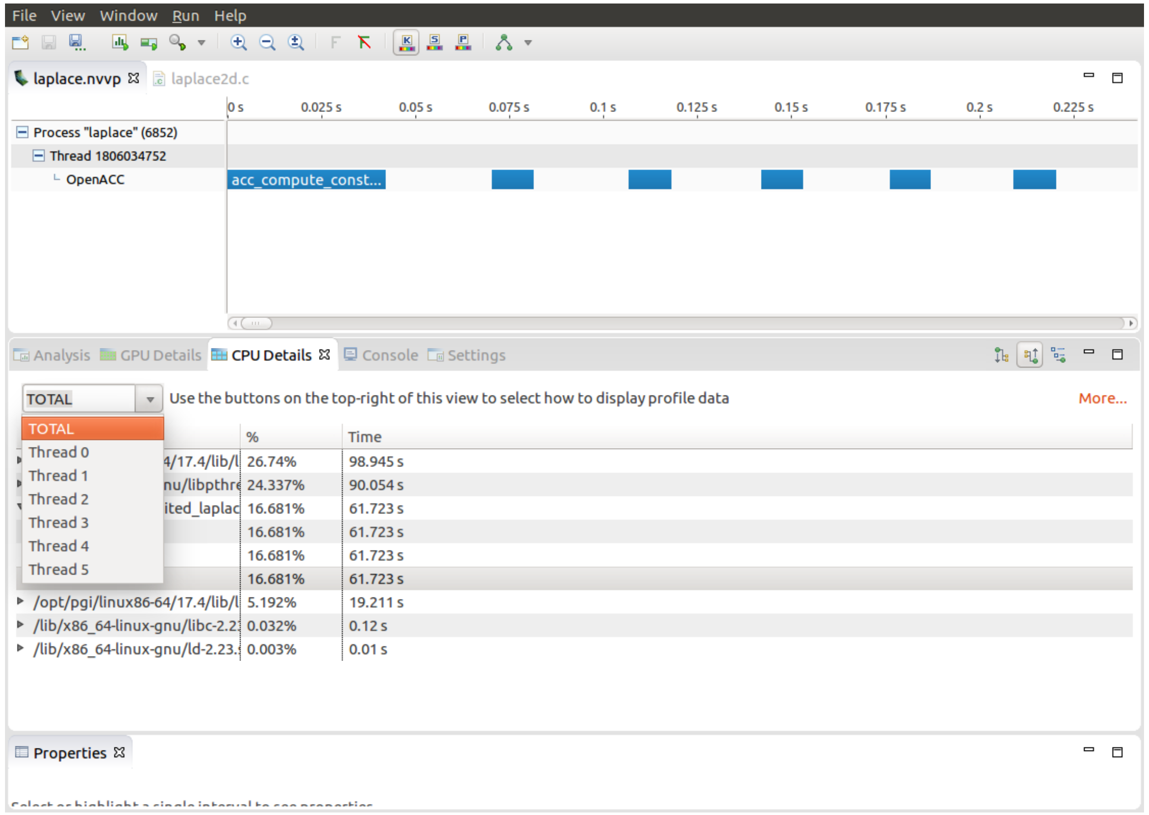
Event	%	Time
/opt/pgi/linux86-64/17.4/lib/l	26.74%	98.945 s
/lib/x86_64-linux-gnu/libpthre	24.337%	90.054 s
/home/ewright/edited_laplac	16.681%	61.723 s
./laplace2d.c	16.681%	61.723 s
calcNext	16.681%	61.723 s
calcNext:38	16.681%	61.723 s
/opt/pgi/linux86-64/17.4/lib/l	5.192%	19.211 s
/lib/x86_64-linux-gnu/libc-2.2	0.032%	0.12 s
/lib/x86_64-linux-gnu/ld-2.23.	0.003%	0.01 s

Select or highlight a single interval to see properties

PROFILING MULTICORE CODE

View of all computational threads

- The program is actually performing better than the sequential version
- We are only looking at the "TOTAL" view, which means that PGPROF is combining information from all of our CPU cores



Use the buttons on the top-right of this view to select how to display profile data

	%	Time
4/17.4/lib/l	26.74%	98.945 s
nu/libpthre	24.337%	90.054 s
ited_laplac	16.681%	61.723 s
	16.681%	61.723 s
	16.681%	61.723 s
	16.681%	61.723 s
/opt/pgi/linux86-64/17.4/lib/l	5.192%	19.211 s
/lib/x86_64-linux-gnu/libc-2.2	0.032%	0.12 s
/lib/x86_64-linux-gnu/ld-2.23.	0.003%	0.01 s

Select or highlight a single interval to see properties

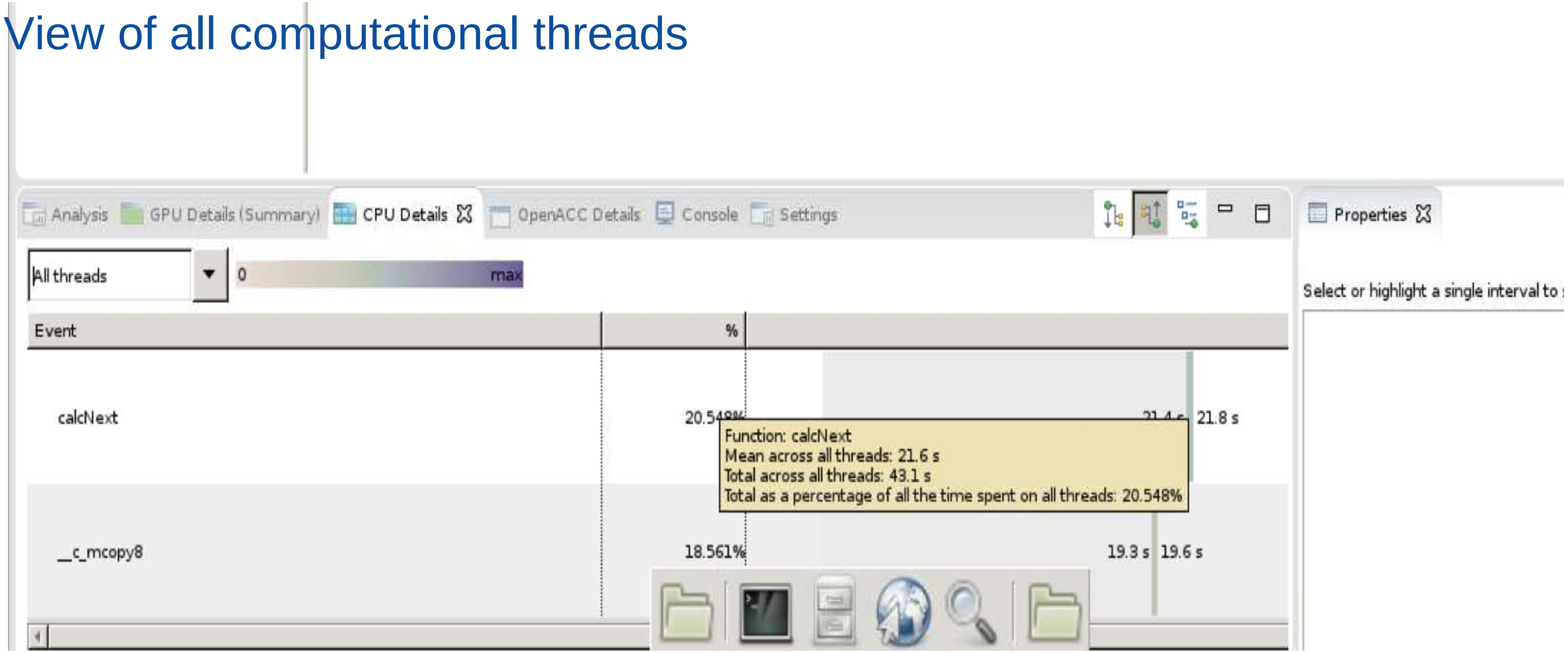
PROFILING MULTICORE CODE

View of all computational threads

- The new PGPROF 18.4 installed on VNC changed the dropdown box labeled TOTAL into “All threads” and displays min, max and mean values graphically.
- When moving the mouse on the % value, one can see
  - Mean across all threads
  - Total across all threads
  - Total as a percentage of all the time spent on all threads.

PROFILING MULTICORE CODE

View of all computational threads



Select or highlight a single interval to

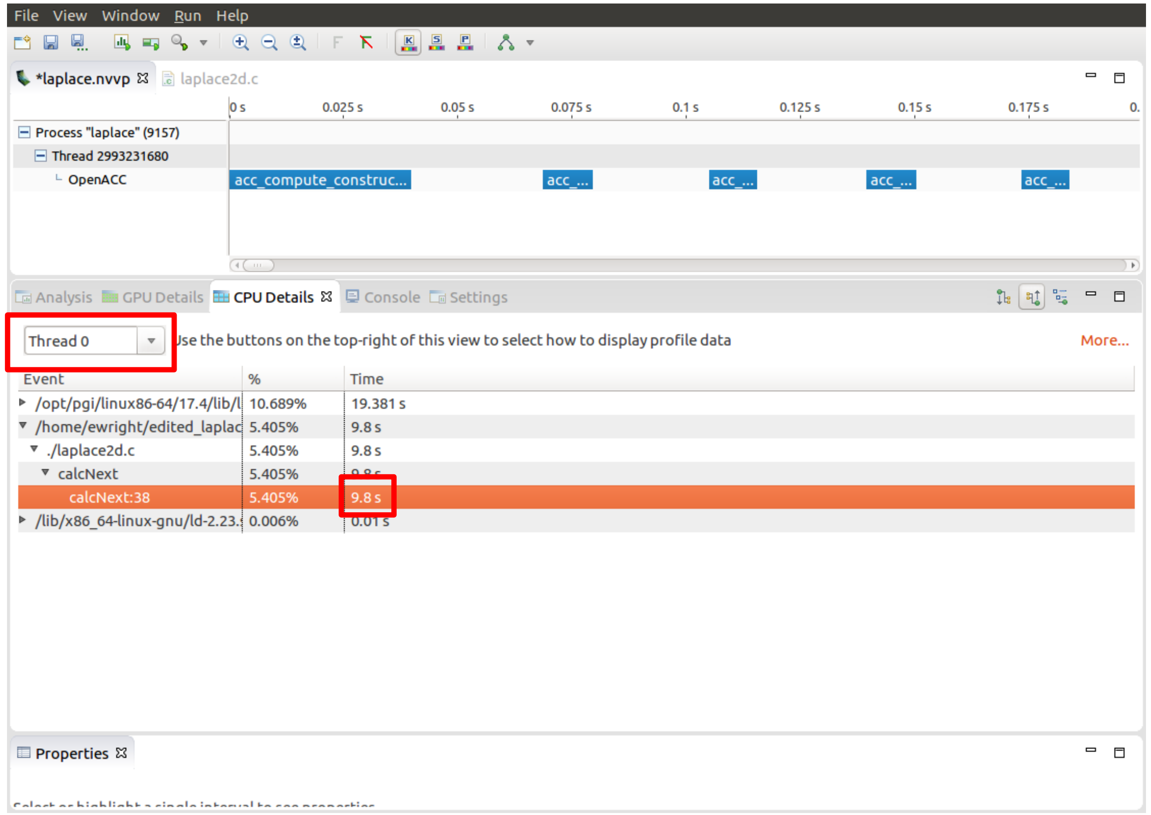
Event	%	
calcNext	20.548%	21.4 s 21.8 s
__c_mcopy8	18.561%	19.3 s 19.6 s

Function: calcNext
Mean across all threads: 21.6 s
Total across all threads: 43.1 s
Total as a percentage of all the time spent on all threads: 20.548%

PROFILING MULTICORE CODE

Observing a single thread

- Now we have selected to view a specific thread (for us, a thread would be a single CPU core)
- We can see that this single thread only spent 9.8 seconds running calcNext
- Each thread will take a similar amount of time and execute simultaneously, resulting in a faster run



Use the buttons on the top-right of this view to select how to display profile data

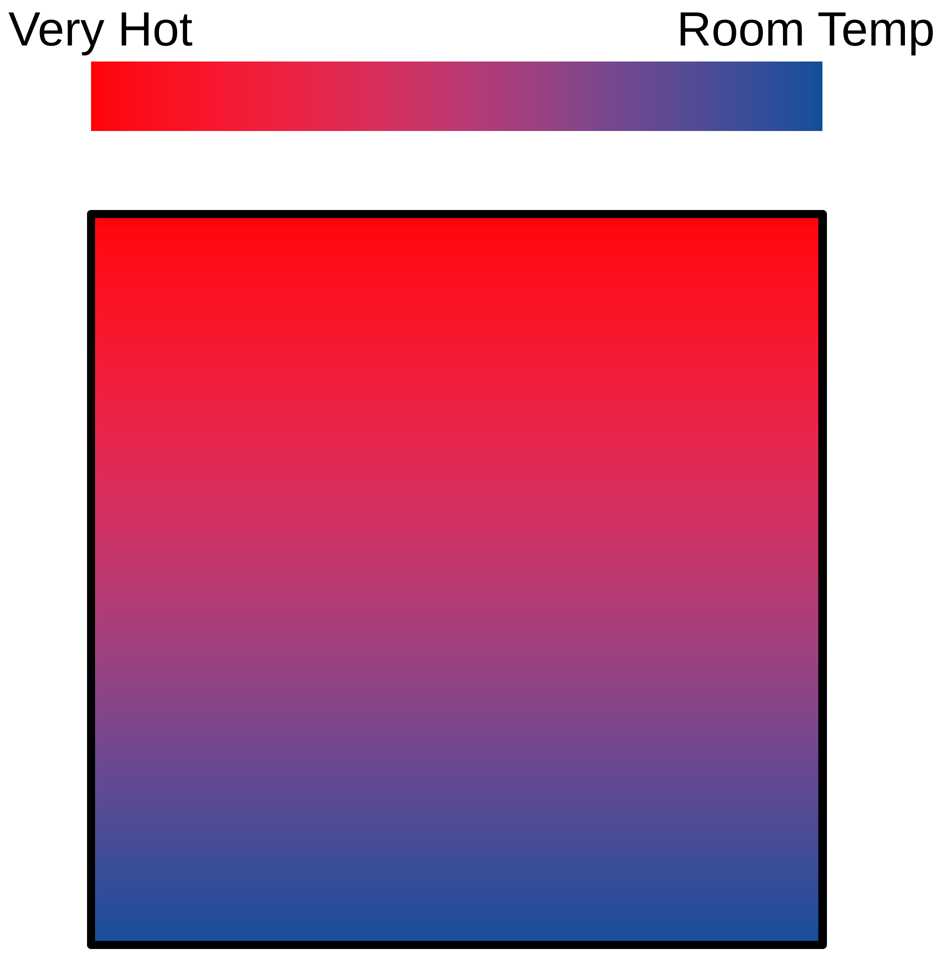
Event	%	Time
/opt/pgi/linux86-64/17.4/lib/l	10.689%	19.381 s
/home/ewright/edited_laplac	5.405%	9.8 s
./laplace2d.c	5.405%	9.8 s
calcNext	5.405%	9.8 s
calcNext:38	5.405%	9.8 s
/lib/x86_64-linux-gnu/ld-2.23.	0.006%	0.01 s

Select or highlight a single interval to see properties

OpenACC
More Science, Less Programming

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.

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

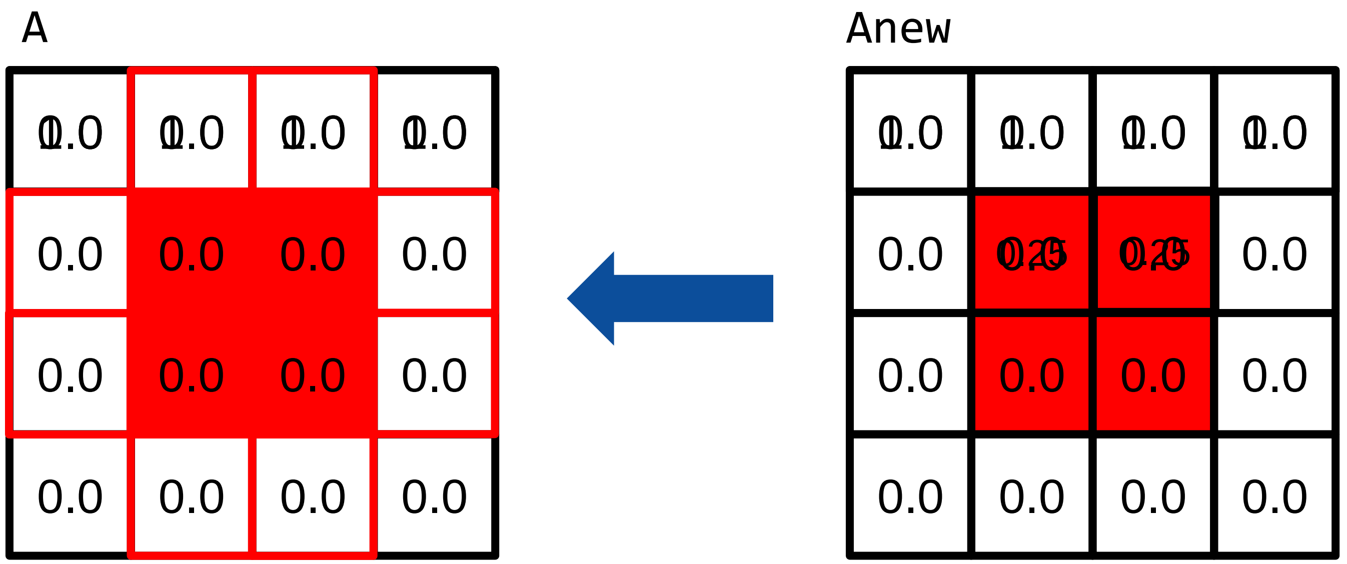
Introduction to lab code - technical

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

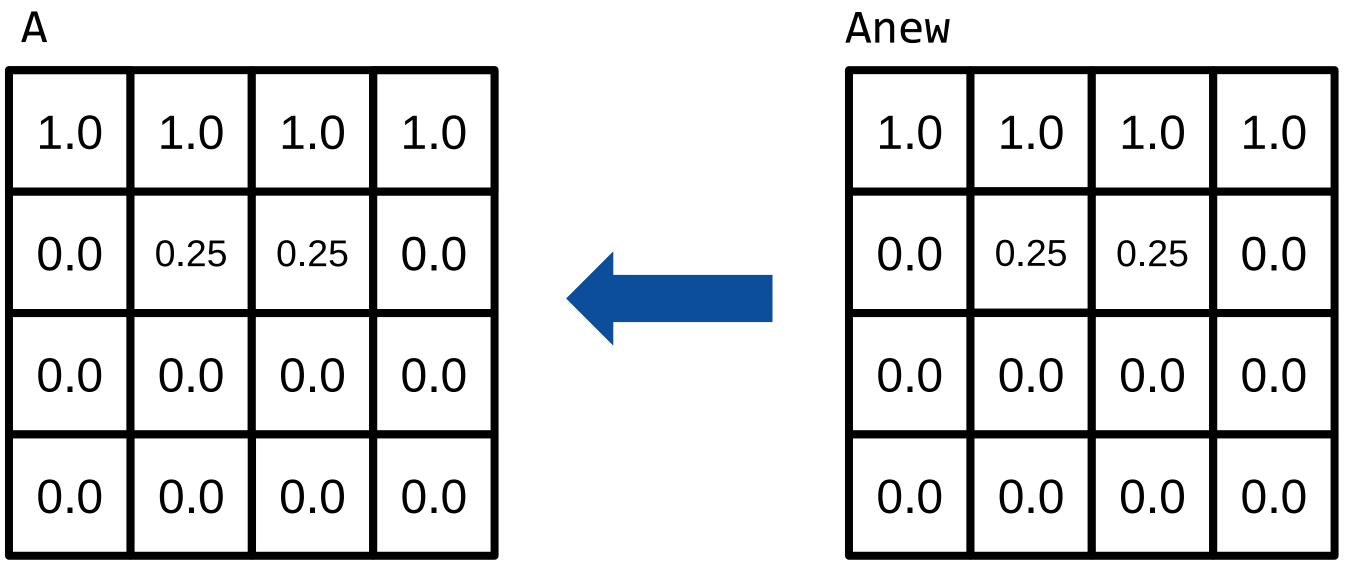
We will take the average of the neighboring cells, and record it in **Anew.**

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



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 PGI compiler
- Use PGProf to understand where the program spends its time

THANK YOU

OpenACC

More Science, Less Programming

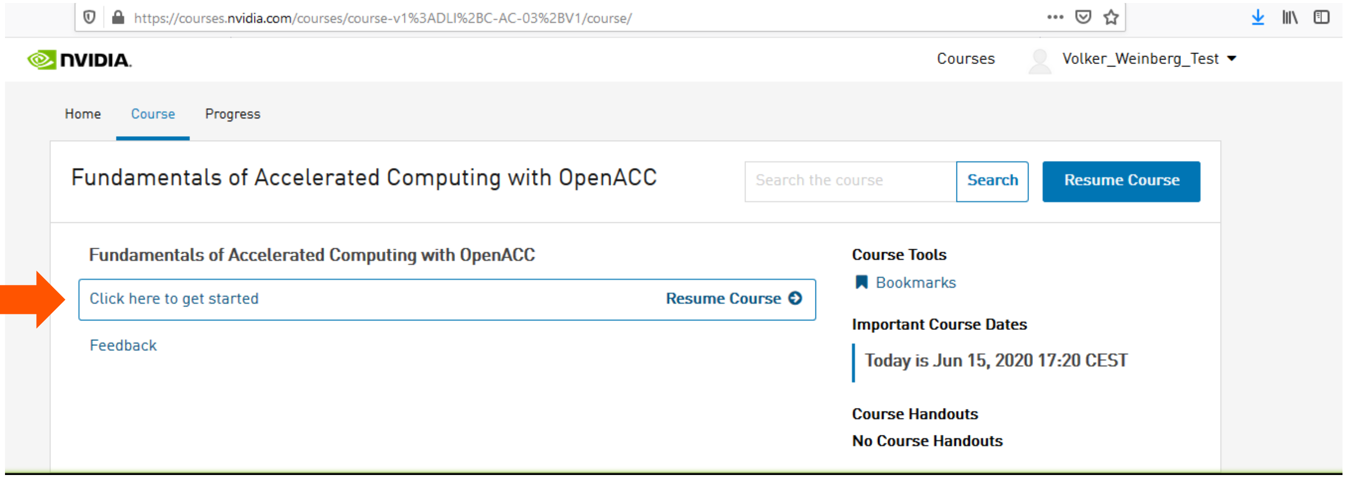
TRAINING SETUP

- 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“ (done yesterday)
- Visit http://courses.nvidia.com/dli-event and enter the event code

PRACE_OACC_AMBASSADOR_JU20

OpenACC
More Science, Less Programming

TRAINING SETUP



https://courses.nvidia.com/courses/course-v1%3ADLI%2BC-AC-03%2BV1/course/

Courses | Volker_Weinberg_Test

Home | Course | Progress

Fundamentals of Accelerated Computing with OpenACC

Search the course | Search | Resume Course

Fundamentals of Accelerated Computing with OpenACC

Click here to get started — Resume Course

Feedback

Course Tools

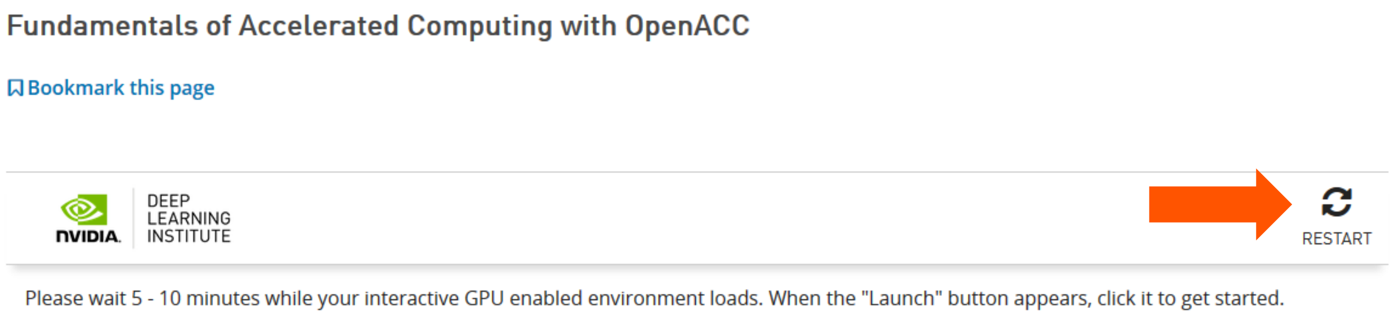
Bookmarks

Important Course Dates

Today is Jun 15, 2020 17:20 CEST

Course Handouts

No Course Handouts



Fundamentals of Accelerated Computing with OpenACC

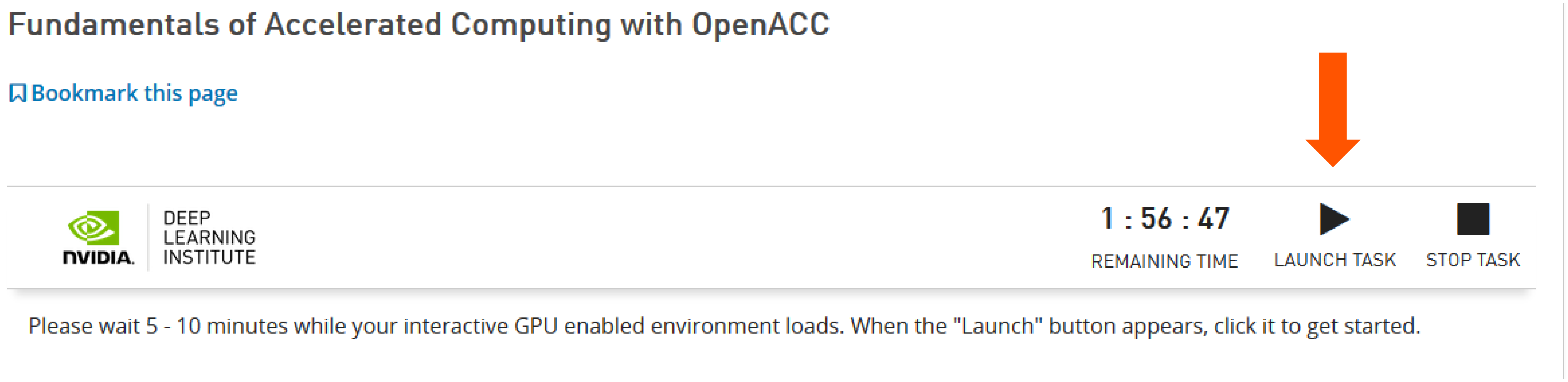
Bookmark this page

DEEP LEARNING INSTITUTE

RESTART

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

TRAINING SETUP

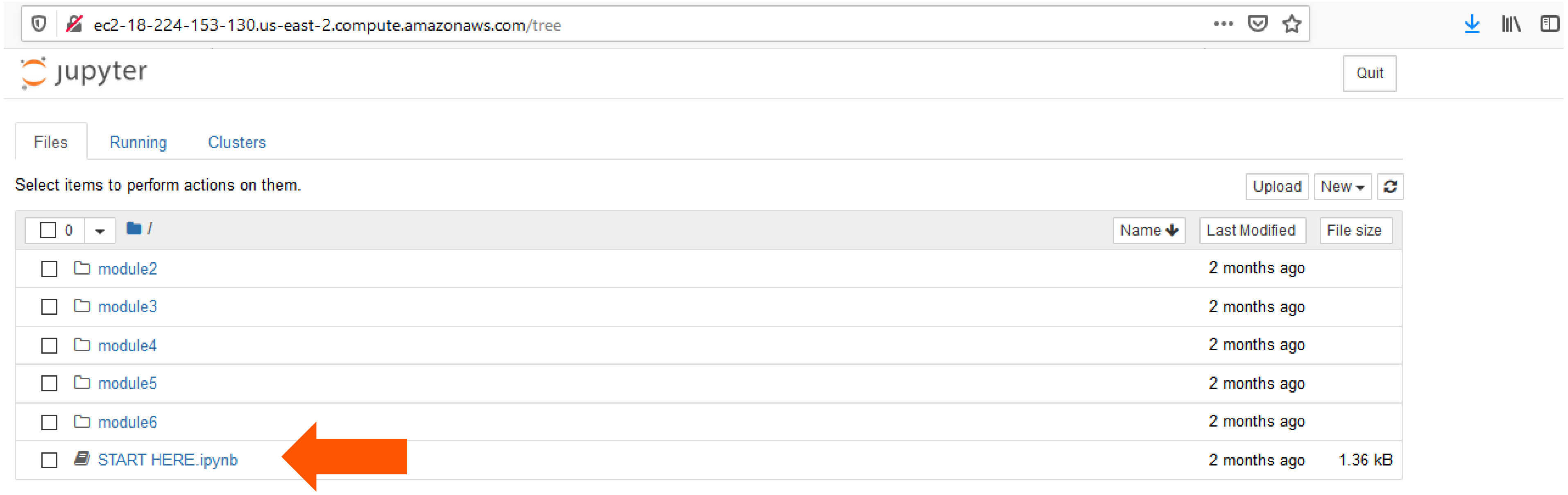


Fundamentals of Accelerated Computing with OpenACC

Bookmark this page

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

TRAINING SETUP

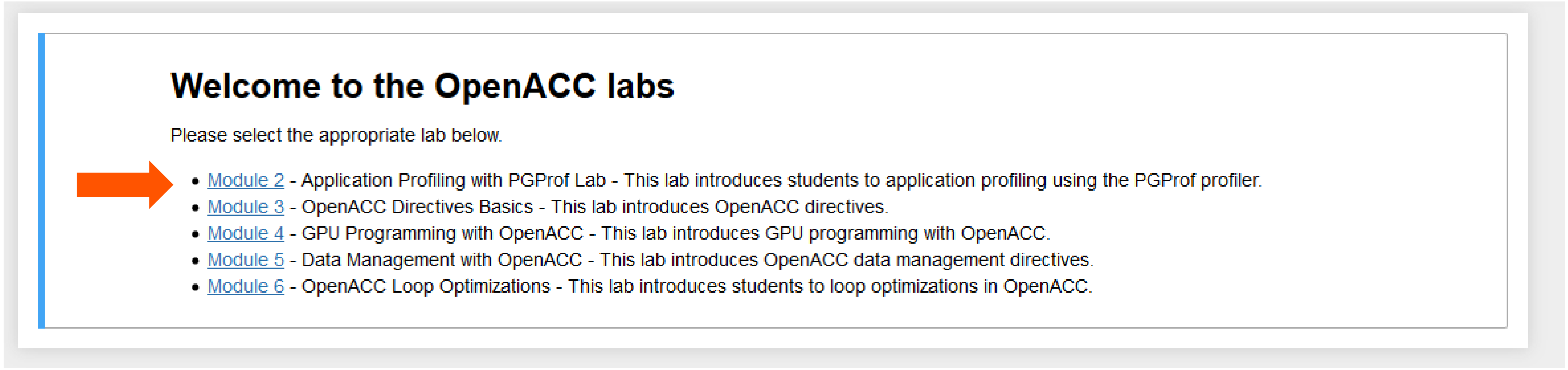


ec2-18-224-153-130.us-east-2.compute.amazonaws.com/tree

Select items to perform actions on them.

Name	Last Modified	File size
module2	2 months ago	
module3	2 months ago	
module4	2 months ago	
module5	2 months ago	
module6	2 months ago	
START HERE.ipynb	2 months ago	1.36 kB

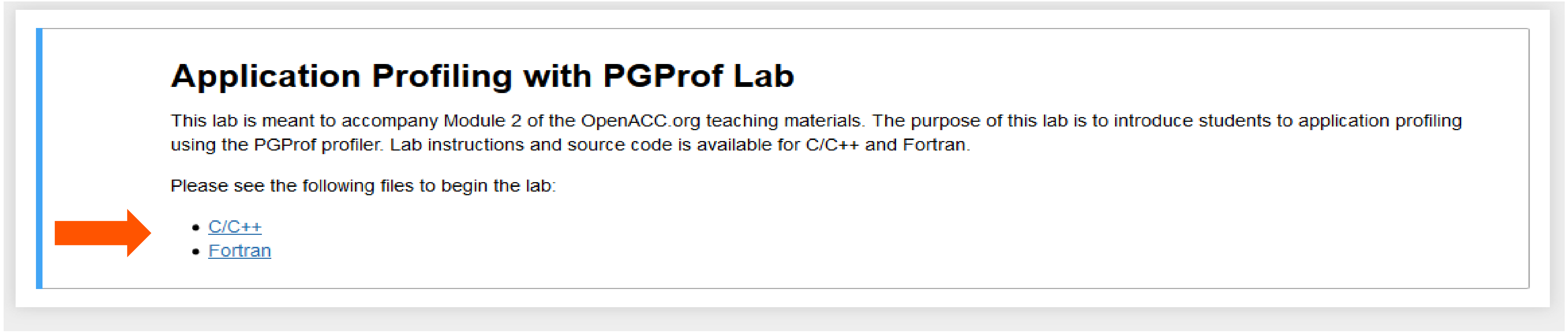
TRAINING SETUP



Welcome to the OpenACC labs

Please select the appropriate lab below.

- Module 2 - Application Profiling with PGProf Lab - This lab introduces students to application profiling using the PGProf 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.



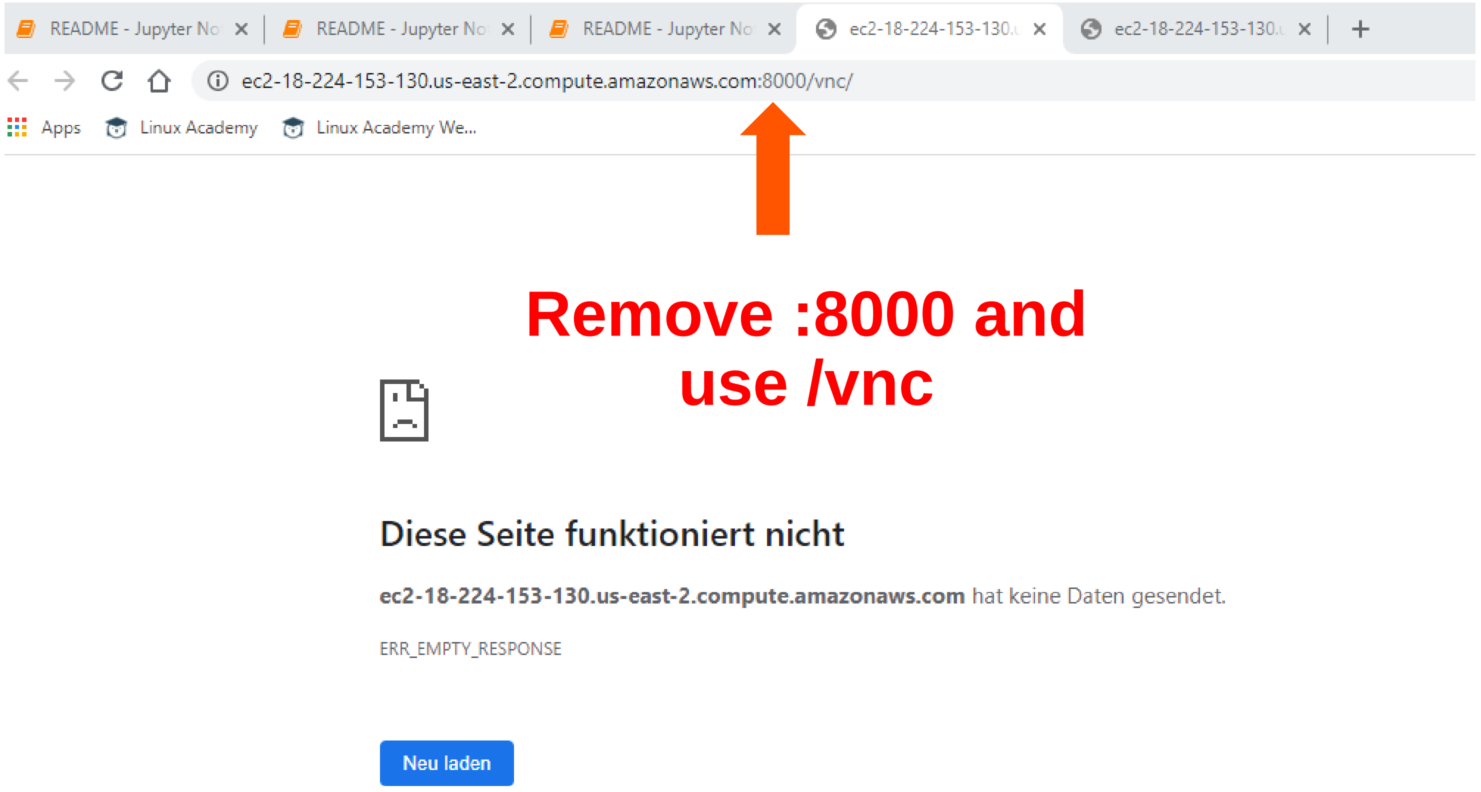
Application Profiling with PGProf Lab

This lab is meant to accompany Module 2 of the OpenACC.org teaching materials. The purpose of this lab is to introduce students to application profiling using the PGProf profiler. Lab instructions and source code is available for C/C++ and Fortran.

Please see the following files to begin the lab:

- C/C++
- Fortran

VNC USAGE



ec2-18-224-153-130.us-east-2.compute.amazonaws.com:8000/vnc/

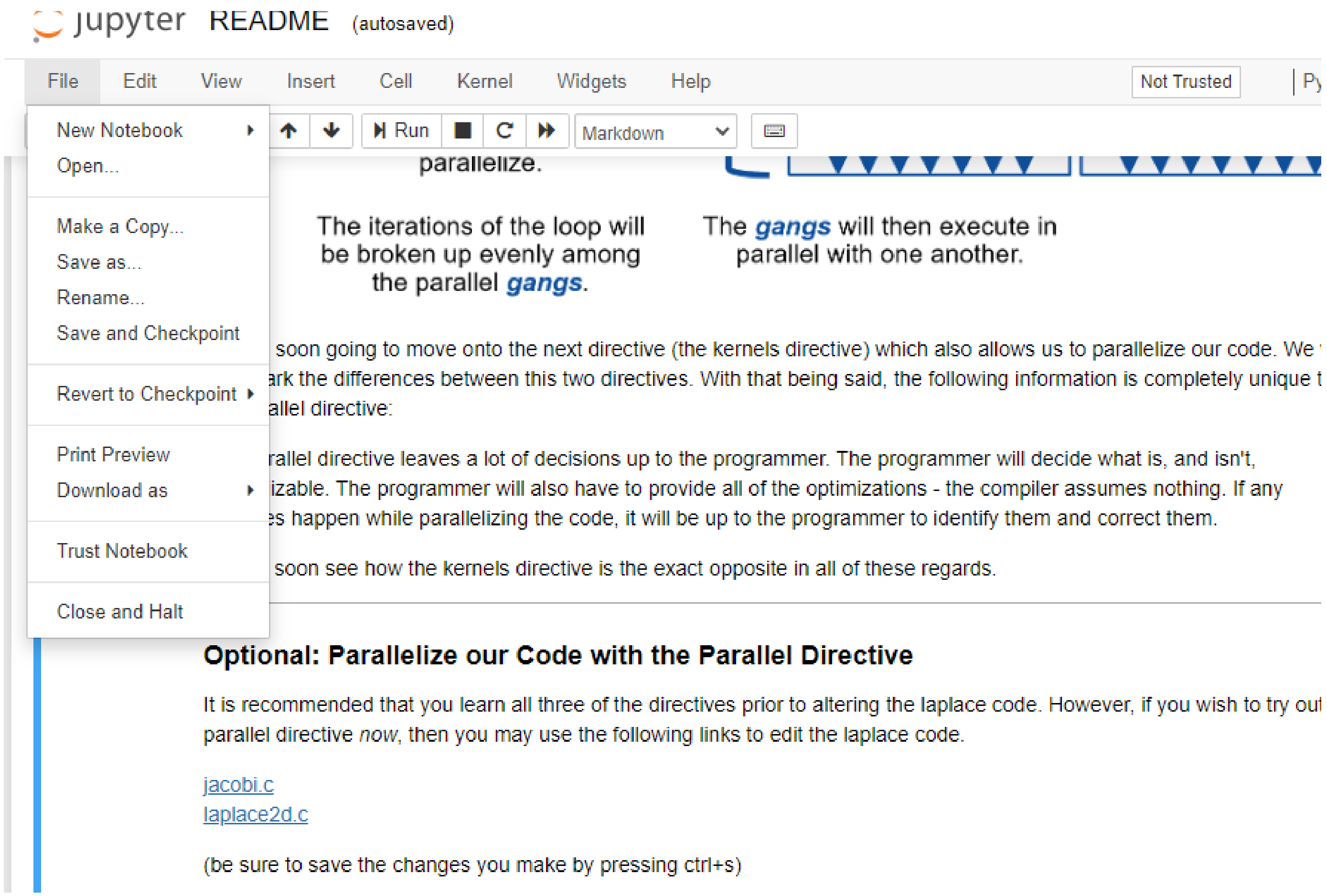
Diese Seite funktioniert nicht

ec2-18-224-153-130.us-east-2.compute.amazonaws.com hat keine Daten gesendet.

ERR_EMPTY_RESPONSE

Remove :8000 and use /vnc

HOW TO EDIT FILES IN MOD2 (METHOD 1)



soon going to move onto the next directive (the kernels directive) which also allows us to parallelize our code. We
ark the differences between this two directives. With that being said, the following information is completely unique t
allel directive:

rallel directive leaves a lot of decisions up to the programmer. The programmer will decide what is, and isn't,
izable. The programmer will also have to provide all of the optimizations - the compiler assumes nothing. If any
es happen while parallelizing the code, it will be up to the programmer to identify them and correct them.

soon see how the kernels directive is the exact opposite in all of these regards.

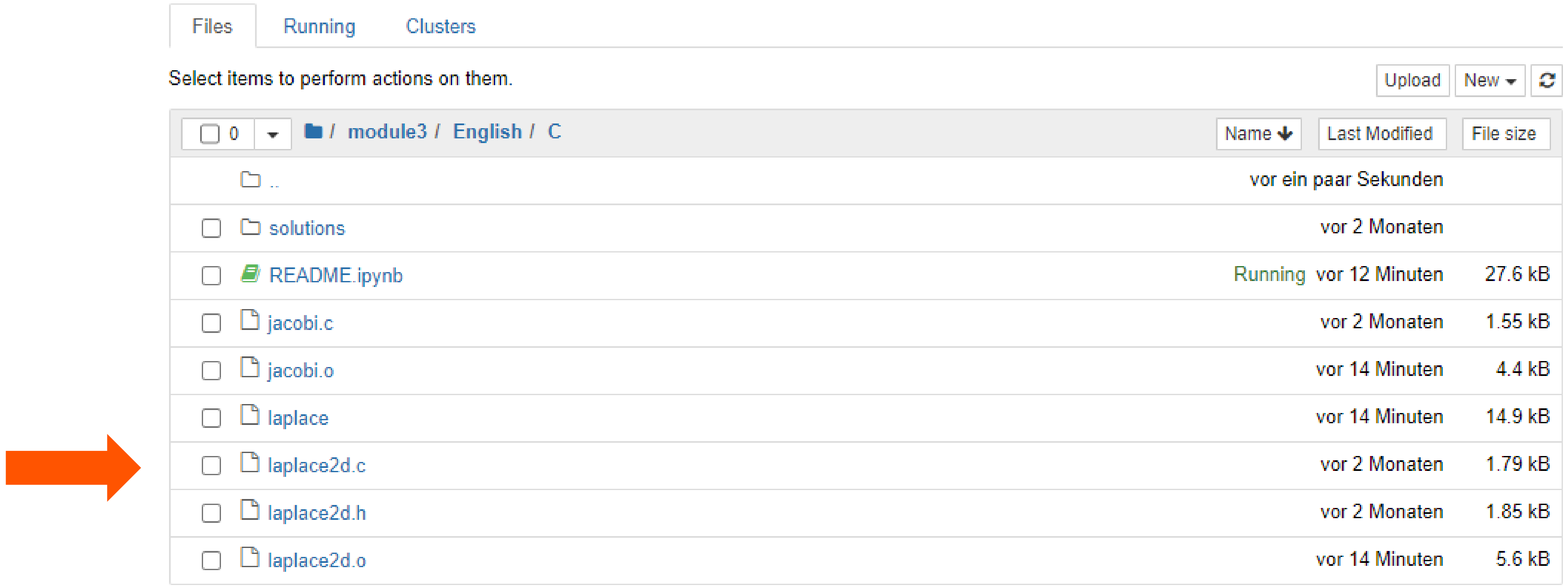
Optional: Parallelize our Code with the Parallel Directive

It is recommended that you learn all three of the directives prior to altering the laplace code. However, if you wish to try out
parallel directive *now*, then you may use the following links to edit the laplace code.

jacobi.c
laplace2d.c

(be sure to save the changes you make by pressing ctrl+s)

HOW TO EDIT FILES IN MOD2 (METHOD 1)

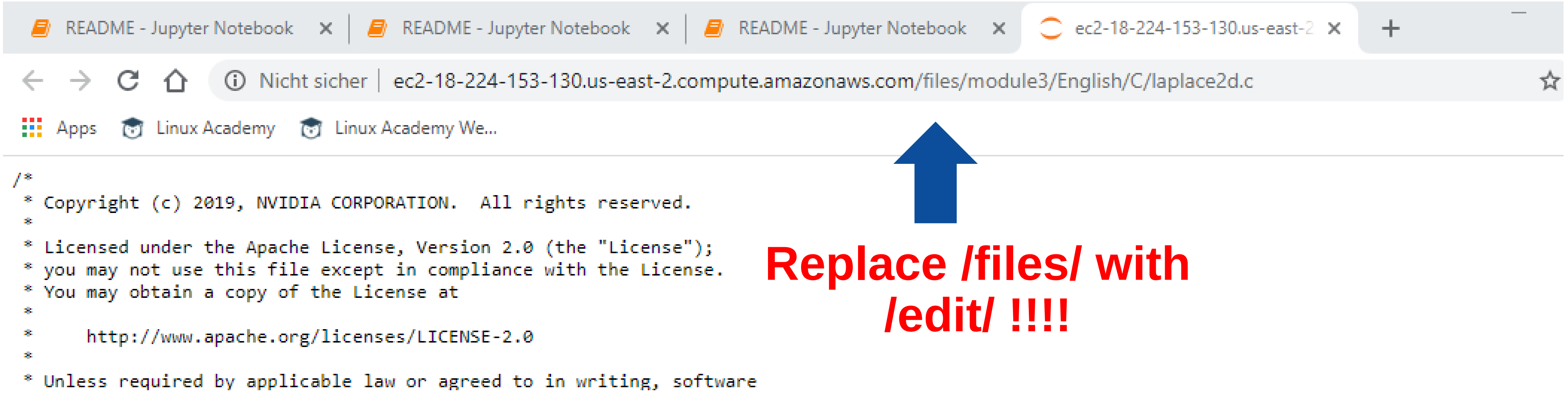


Select items to perform actions on them.

/ module3 / English / C

```
..                                   vor ein paar Sekunden
solutions                            vor 2 Monaten
README.ipynb          Running        vor 12 Minuten     27.6 kB
jacobi.c                             vor 2 Monaten      1.55 kB
jacobi.o                             vor 14 Minuten      4.4 kB
laplace                              vor 14 Minuten     14.9 kB
laplace2d.c                          vor 2 Monaten      1.79 kB
laplace2d.h                          vor 2 Monaten      1.85 kB
laplace2d.o                          vor 14 Minuten      5.6 kB
```

HOW TO EDIT FILES IN MOD2 (METHOD 2)


README - Jupyter Notebook
README - Jupyter Notebook
README - Jupyter Notebook
ec2-18-224-153-130.us-east-2
Nicht sicher
ec2-18-224-153-130.us-east-2.compute.amazonaws.com/files/module3/English/C/laplace2d.c
Apps
Linux Academy
Linux Academy We...

```
/*
 * Copyright (c) 2019, NVIDIA CORPORATION.  All rights reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
```

Replace /files/ with /edit/ !!!!