OpenACC Course

Lecture 1: Introduction to OpenACC

September 2015

Course Objective:

Enable *you* to accelerate *your* applications with OpenACC.

Course Syllabus

- Oct 1: Introduction to OpenACC
- Oct 6: Office Hours
- Oct 15: Profiling and Parallelizing with the OpenACC Toolkit
- Oct 20: Office Hours
- Oct 29: Expressing Data Locality and Optimizations with OpenACC
- Nov 3: Office Hours
- Nov 12: Advanced OpenACC Techniques
- Nov 24: Office Hours

Recordings:
https://developer.nvidia.com/openacc-course

Introduction to OpenACC

Jeff Larkin, NVIDIA Developer Technologies

Agenda

Why OpenACC?

Accelerated Computing Fundamentals

OpenACC Programming Cycle

Installing the OpenACC Toolkit

Accessing QwikLabs

Week 1 Homework

Why OpenACC?

OpenACC

Simple | Powerful | Portable

Fueling the Next Wave of Scientific Discoveries in HPC

```
main()
{
  <serial code>
  #pragma acc kernels
  //automatically runs on GPU
  {
    <parallel code>
  }
}
```

University of Illinois
PowerGrid- MRI Reconstruction

70x Speed-Up
2 Days of Effort

RIKEN Japan
NICAM- Climate Modeling

7-8x Speed-Up
5% of Code Modified

8000+

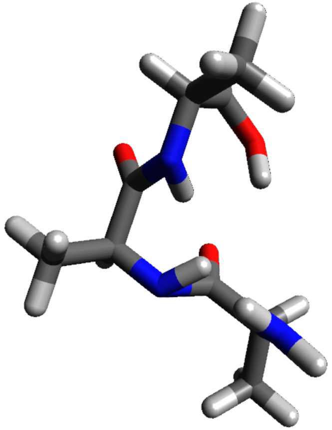
Developers

using OpenACC

http://www.cray.com/sites/default/files/resources/OpenACC_213462.12_OpenACC_Cosmo_CS_FNL.pdf
http://www.hpcwire.com/off-the-wire/first-round-of-2015-hackathons-gets-underway
http://on-demand.gputechconf.com/gtc/2015/presentation/S5297-Hisashi-Yashiro.pdf
http://www.openacc.org/content/experiences-porting-molecular-dynamics-code-gpus-cray-xk7

LS-DALTON

Large-scale application for calculating high-accuracy molecular energies



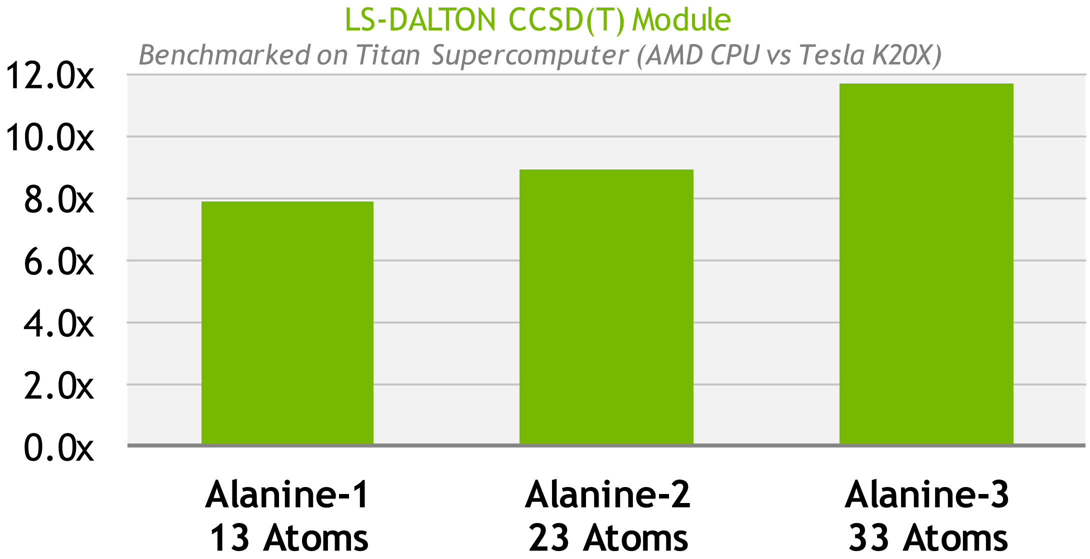
> “*OpenACC makes GPU computing approachable for domain scientists. Initial OpenACC implementation required only minor effort, and more importantly, no modifications of our existing CPU implementation.*”

Janus Juul Eriksen, PhD Fellow
qLEAP Center for Theoretical Chemistry, Aarhus University

Minimal Effort

Lines of Code Modified	# of Weeks Required	# of Codes to Maintain
<100 Lines	1 Week	1 Source

Big Performance



OpenACC Directives

Manage
Data
Movement

Initiate
Parallel
Execution

Optimize
Loop
Mappings

```
#pragma acc data copyin(a,b) copyout(c)
{
  ...
  #pragma acc parallel
  {
  #pragma acc loop gang vector
      for (i = 0; i < n; ++i) {
          z[i] = x[i] + y[i];
          ...
      }
  }
  ...
}
```

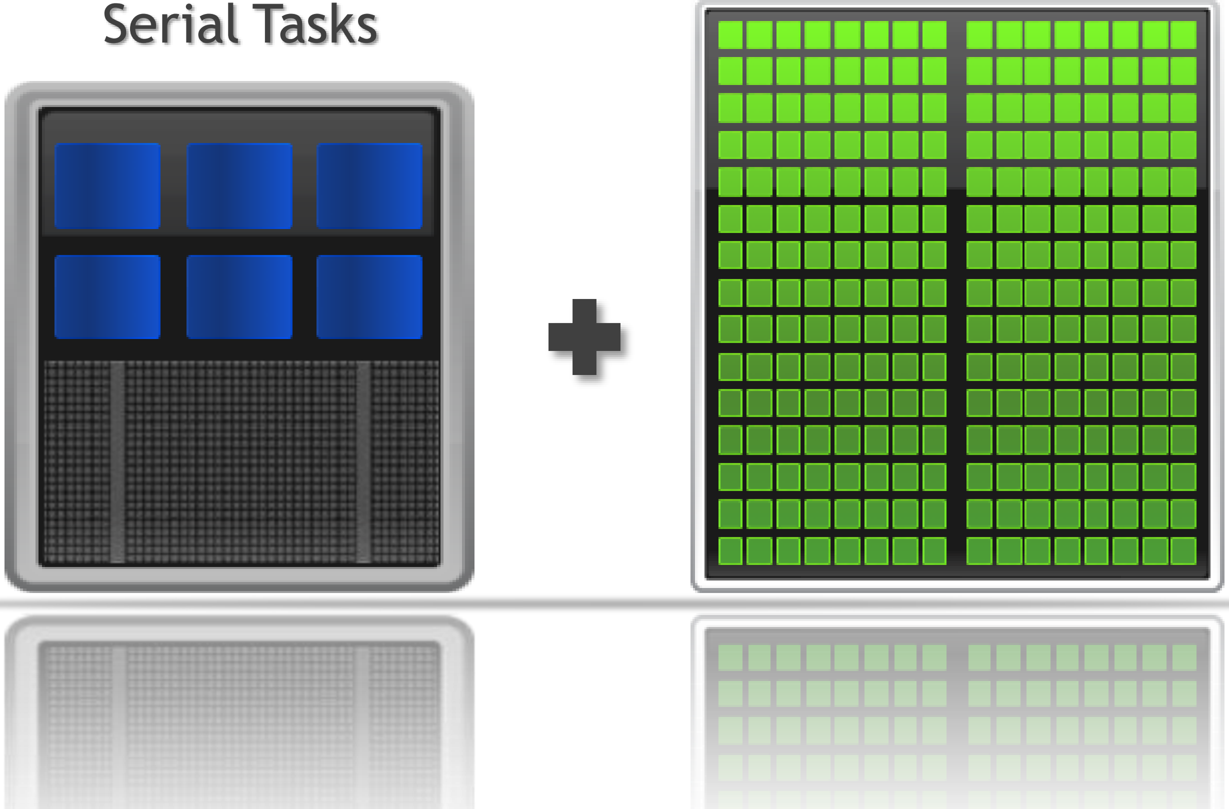
OpenACC
Directives for Accelerators

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, MIC

Accelerated Computing Fundamentals

Accelerated Computing

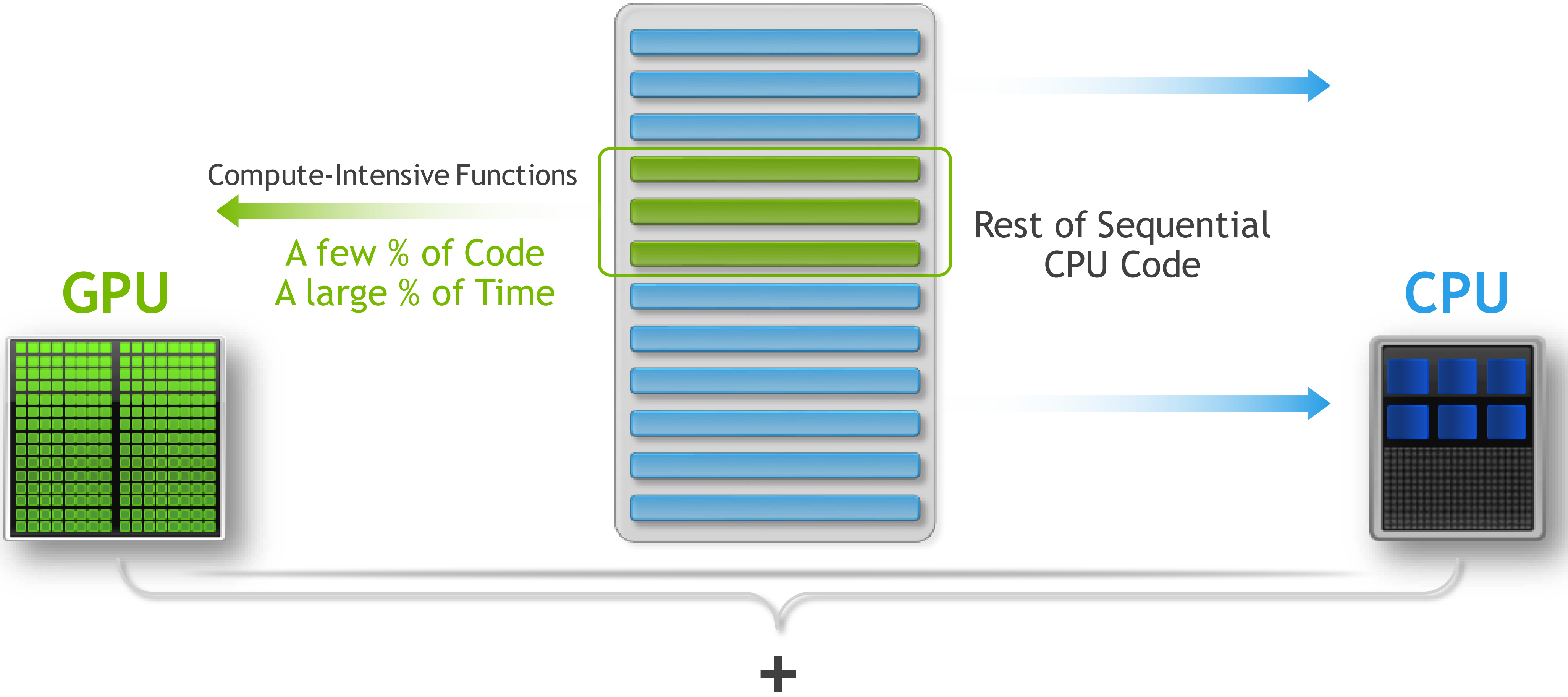
10x Performance & 5x Energy Efficiency for HPC



CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

What is Heterogeneous Programming?



Compute-Intensive Functions

A few % of Code
A large % of Time

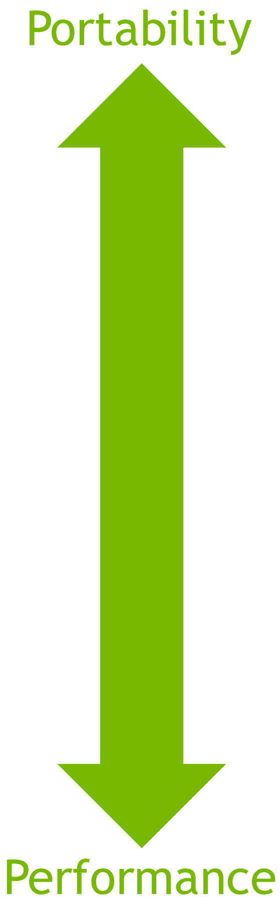
GPU

Rest of Sequential
CPU Code

CPU

+

Portability & Performance



Accelerated Libraries

- High performance with little or no code change
- Limited by what libraries are available

Compiler Directives

- High Level: Based on existing languages; simple, familiar, portable
- High Level: Performance may not be optimal

Parallel Language Extensions

- Greater flexibility and control for maximum performance
- Often less portable and more time consuming to implement

Code for Portability & Performance

Libraries

- Implement as much as possible using portable libraries

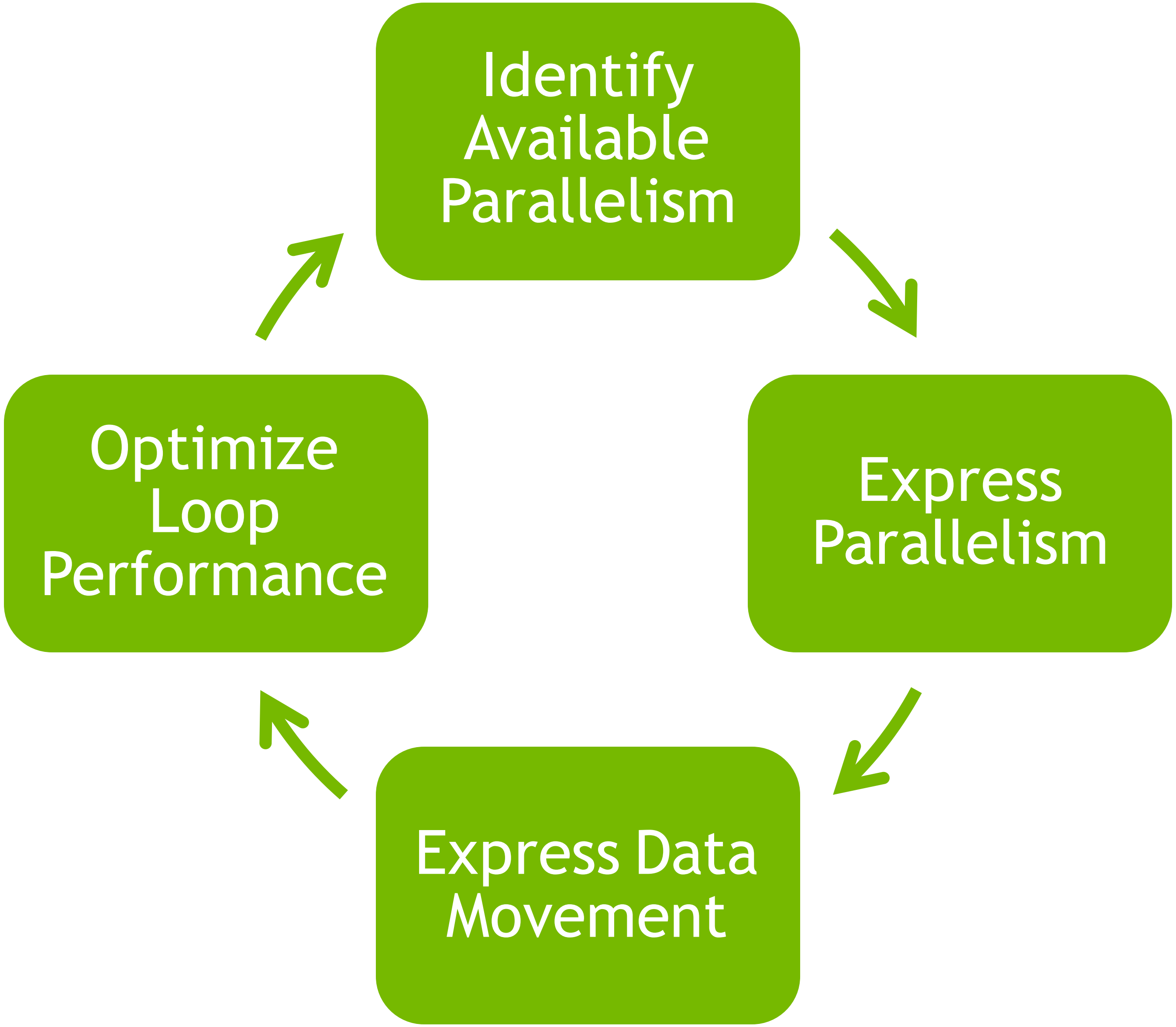
Directives

- Use directives for rapid and portable development

Languages

- Use lower level languages for important kernels

OpenACC Programming Cycle



Identify Available Parallelism

Express Parallelism

Express Data Movement

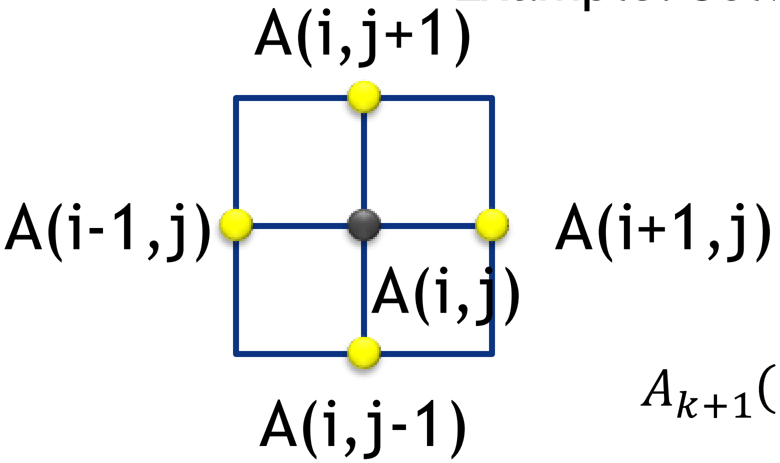
Optimize Loop Performance

Example: Jacobi Iteration

Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.

Common, useful algorithm

Example: Solve Laplace equation in 2D: $\boldsymbol{\nabla^2 f(x,y) = 0}$



$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$

Jacobi Iteration: C Code

```c
while ( err > tol && iter < iter_max ) {
  err=0.0;


  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }


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

  iter++;
}
```

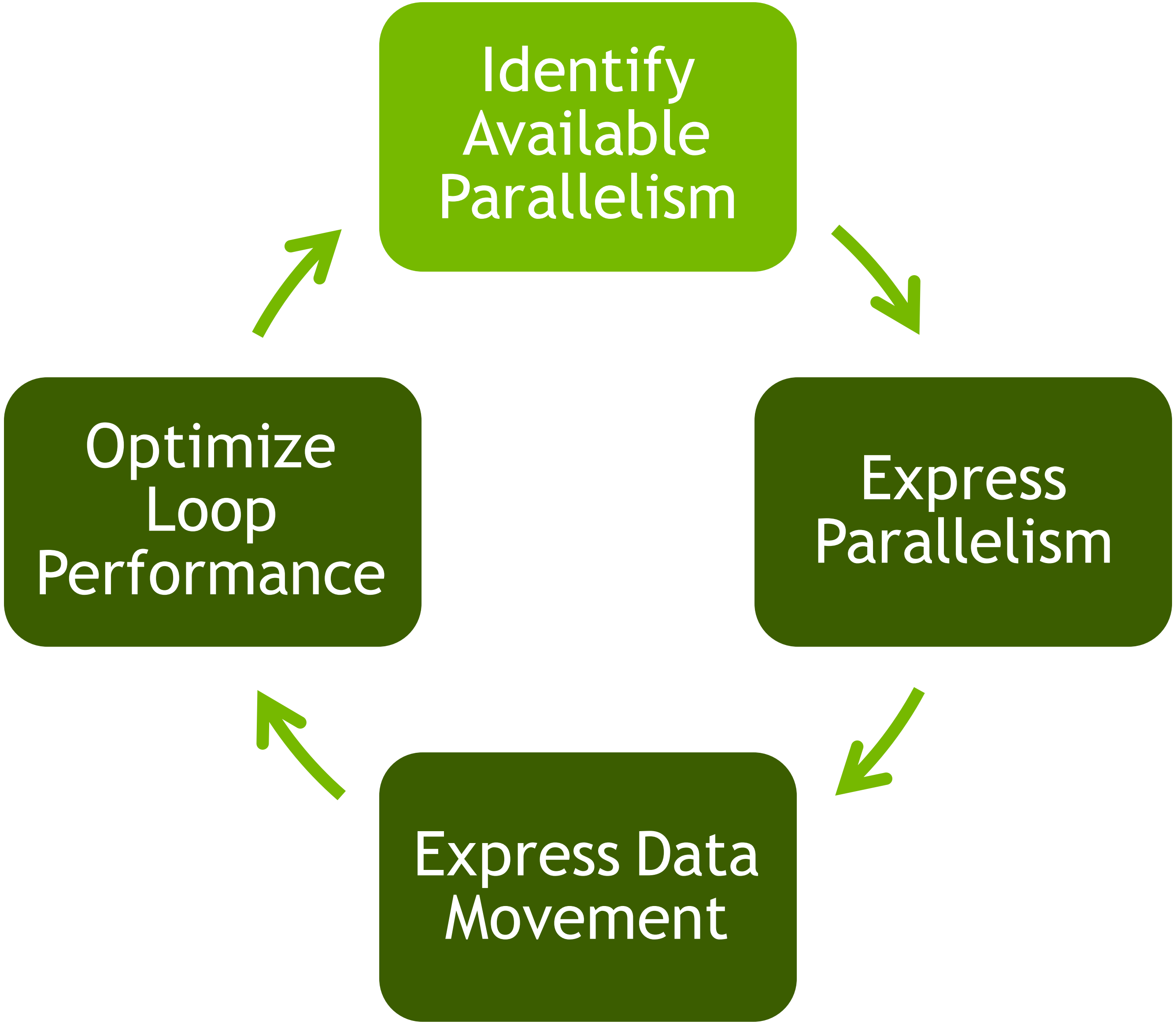
Iterate until converged

Iterate across matrix elements

Calculate new value from neighbors

Compute max error for convergence

Swap input/output arrays



Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance

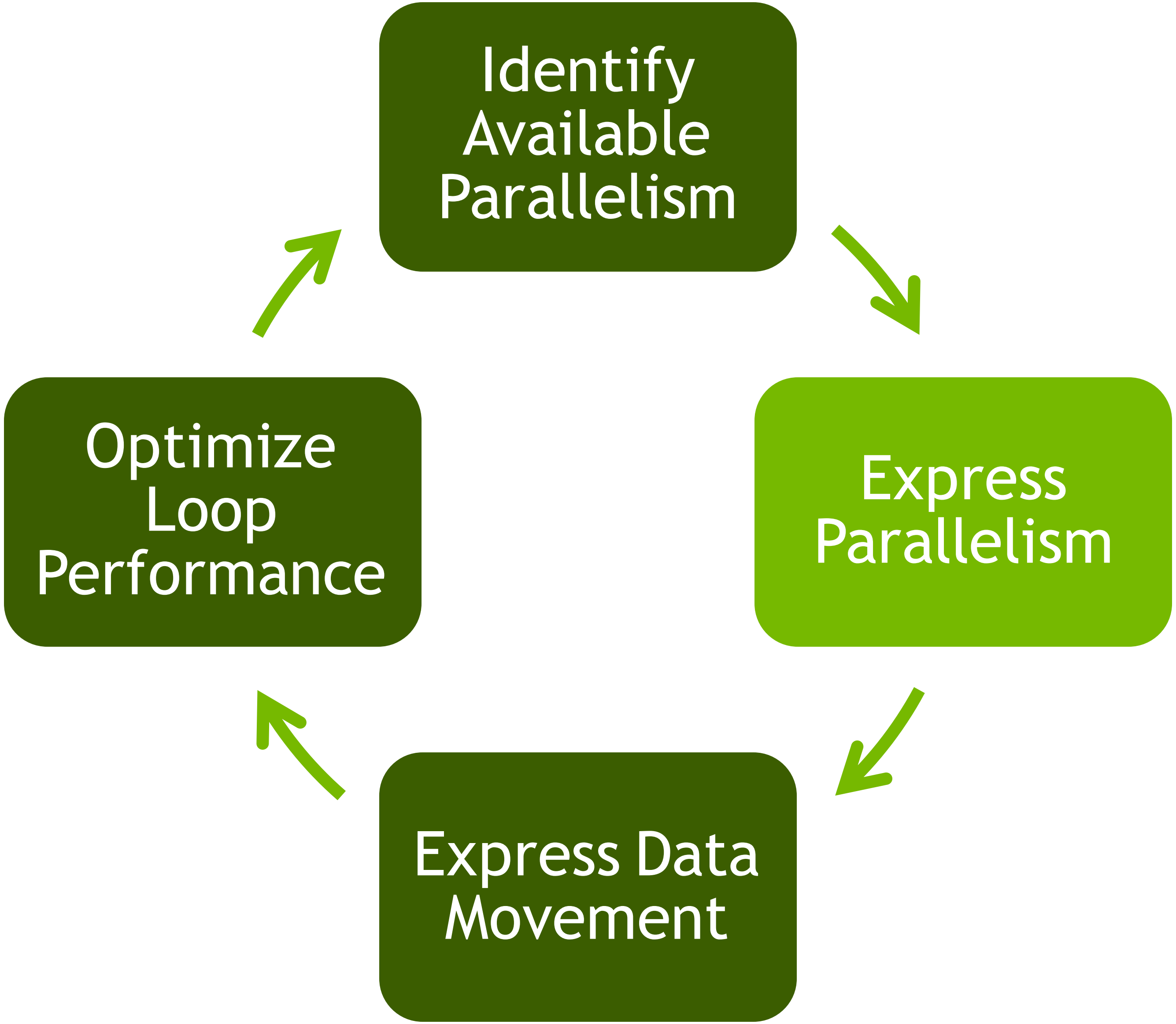
Identify Parallelism

```
while ( err > tol && iter < iter_max ) {
  err=0.0;


  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }


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

  iter++;
}
```

Data dependency between iterations.

Independent loop iterations

Independent loop iterations



Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance

OpenACC kernels Directive

The kernels directive identifies a region that may contain *loops* that the compiler can turn into parallel *kernels*.

```
#pragma acc kernels
{
for(int i=0; i<N; i++)
{
  x[i] = 1.0;
  y[i] = 2.0;
}

for(int i=0; i<N; i++)
{
  y[i] = a*x[i] + y[i];
}
}
```

kernel 1

kernel 2

The compiler identifies 2 parallel loops and generates 2 kernels.

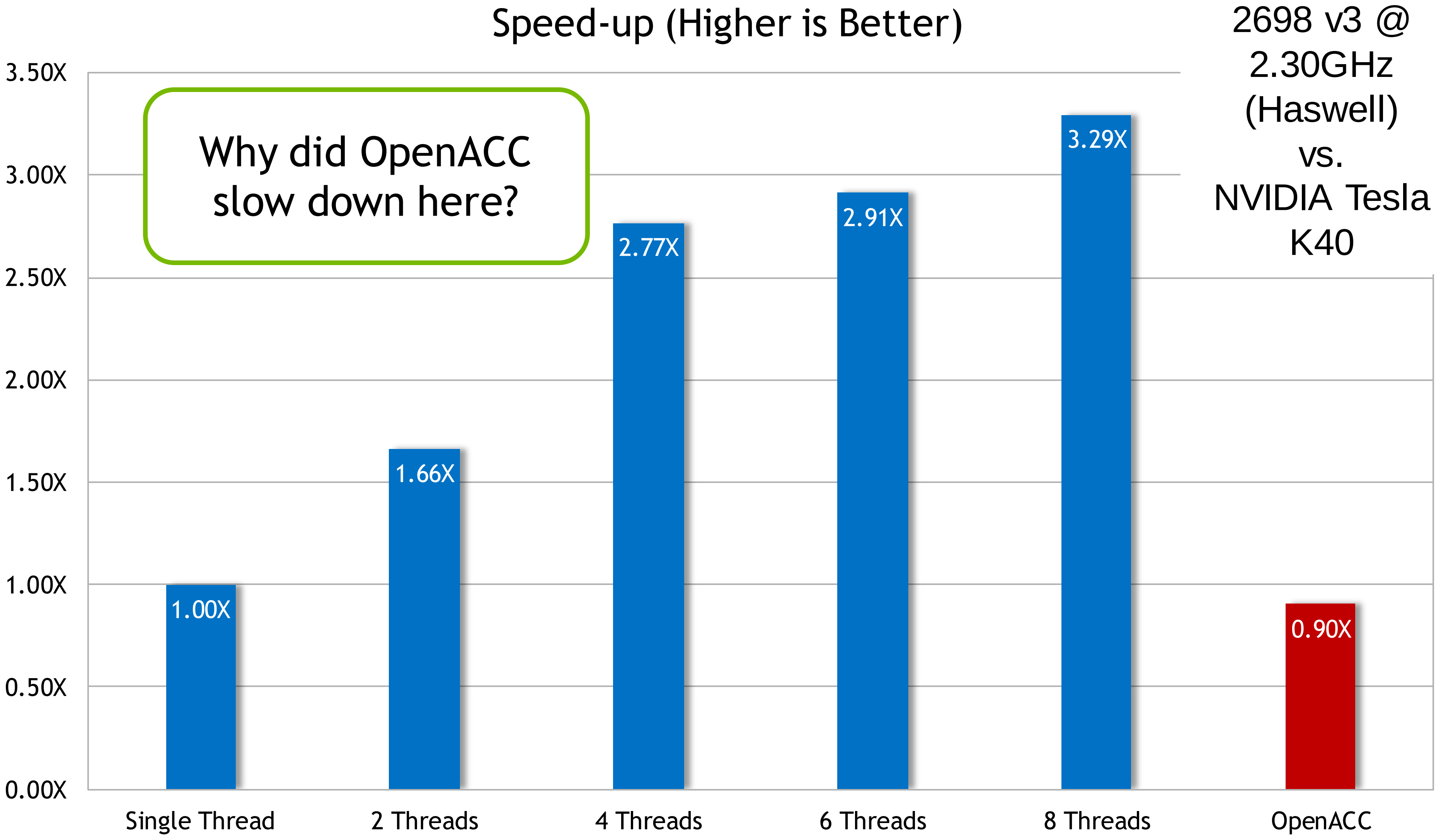
Parallelize with OpenACC kernels

```
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc kernels
{
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

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

Look for parallelism within this region.

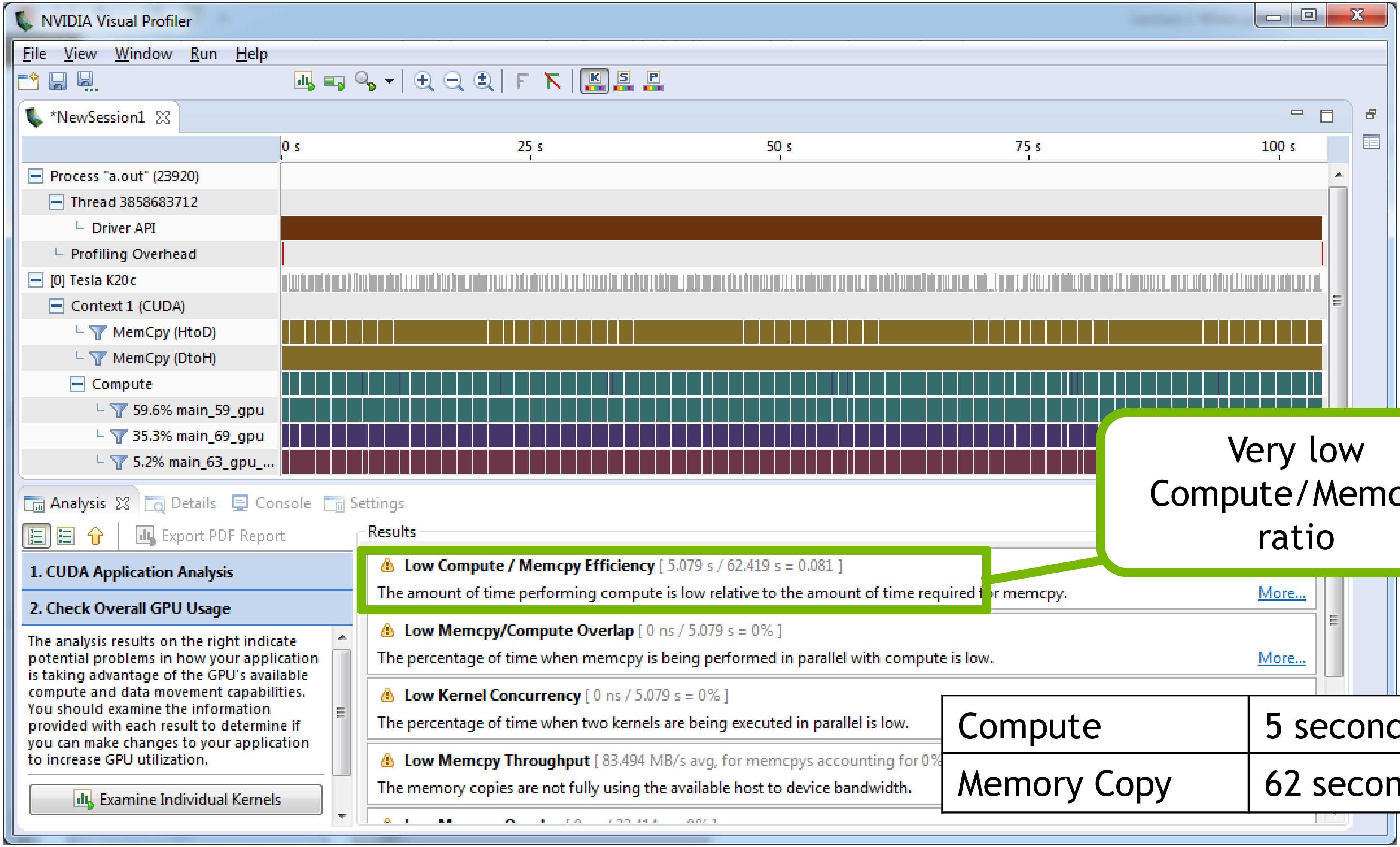
Building the code

```
$ pgcc -fast -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Loop not vectorized/parallelized: potential early exits
     55, Generating copyout(Anew[1:4094][1:4094])
         Generating copyin(A[:][:])
         Generating copyout(A[1:4094][1:4094])
         Generating Tesla code
     57, Loop is parallelizable
     59, Loop is parallelizable
         Accelerator kernel generated
         57, #pragma acc loop gang /* blockIdx.y */
         59, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
         63, Max reduction generated for error
     67, Loop is parallelizable
     69, Loop is parallelizable
         Accelerator kernel generated
         67, #pragma acc loop gang /* blockIdx.y */
         69, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
```



Why did OpenACC slow down here?

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell)
vs.
NVIDIA Tesla K40



The analysis results on the right indicate potential problems in how your application is taking advantage of the GPU's available compute and data movement capabilities. You should examine the information provided with each result to determine if you can make changes to your application to increase GPU utilization.

Low Compute / Memcpy Efficiency [5.079 s / 62.419 s = 0.081]
The amount of time performing compute is low relative to the amount of time required for memcpy.

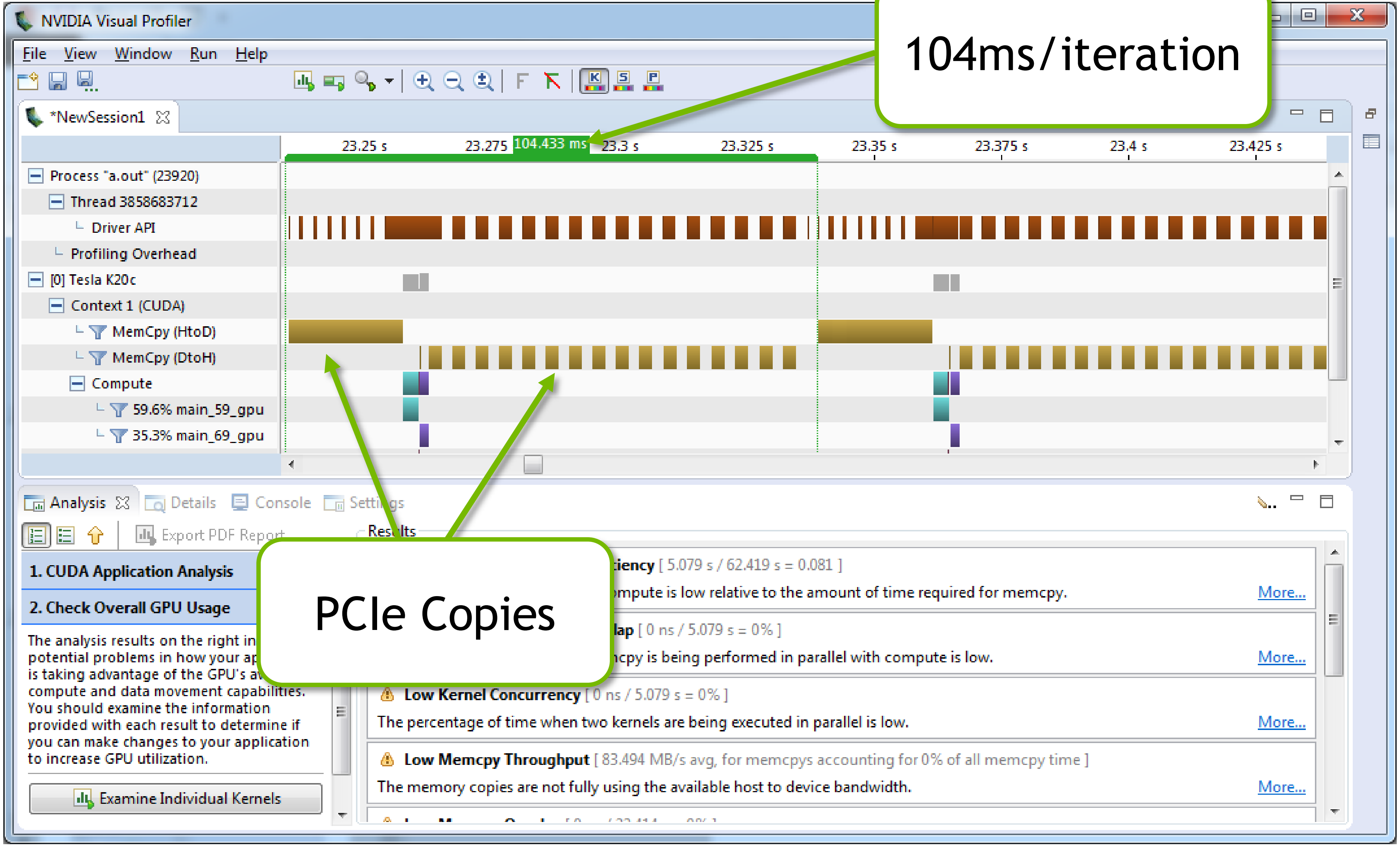
Low Memcpy/Compute Overlap [0 ns / 5.079 s = 0%]
The percentage of time when memcpy is being performed in parallel with compute is low.

Low Kernel Concurrency [0 ns / 5.079 s = 0%]
The percentage of time when two kernels are being executed in parallel is low.

Low Memcpy Throughput [83.494 MB/s avg, for memcpys accounting for 0%
The memory copies are not fully using the available host to device bandwidth.

Very low Compute/Memcpy ratio

Compute	5 seconds
Memory Copy	62 seconds



1. CUDA Application Analysis

2. Check Overall GPU Usage

The analysis results on the right in potential problems in how your ap is taking advantage of the GPU's a compute and data movement capabilities. You should examine the information provided with each result to determine if you can make changes to your application to increase GPU utilization.

iency [5.079 s / 62.419 s = 0.081]
mpute is low relative to the amount of time required for memcpy. More...

lap [0 ns / 5.079 s = 0%]
cpy is being performed in parallel with compute is low. More...

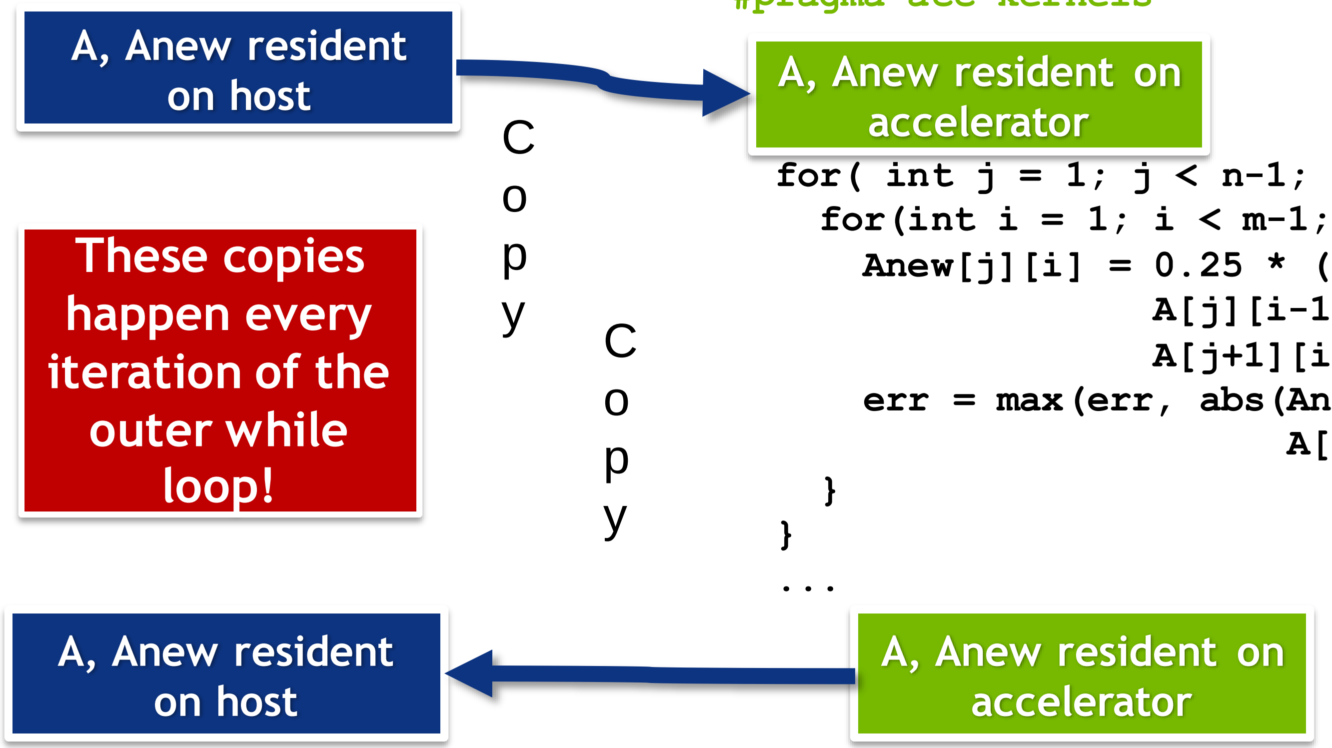
⚠ **Low Kernel Concurrency** [0 ns / 5.079 s = 0%]
The percentage of time when two kernels are being executed in parallel is low. More...

⚠ **Low Memcpy Throughput** [83.494 MB/s avg, for memcpys accounting for 0% of all memcpy time]
The memory copies are not fully using the available host to device bandwidth. More...

104ms/iteration

PCIe Copies

Excessive Data Transfers



```
while ( err > tol && iter < iter_max )
{
  err=0.0;
```

A, Anew resident on host

Copy

```
#pragma acc kernels
```

A, Anew resident on accelerator

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

These copies happen every iteration of the outer while loop!

Copy

A, Anew resident on accelerator

A, Anew resident on host

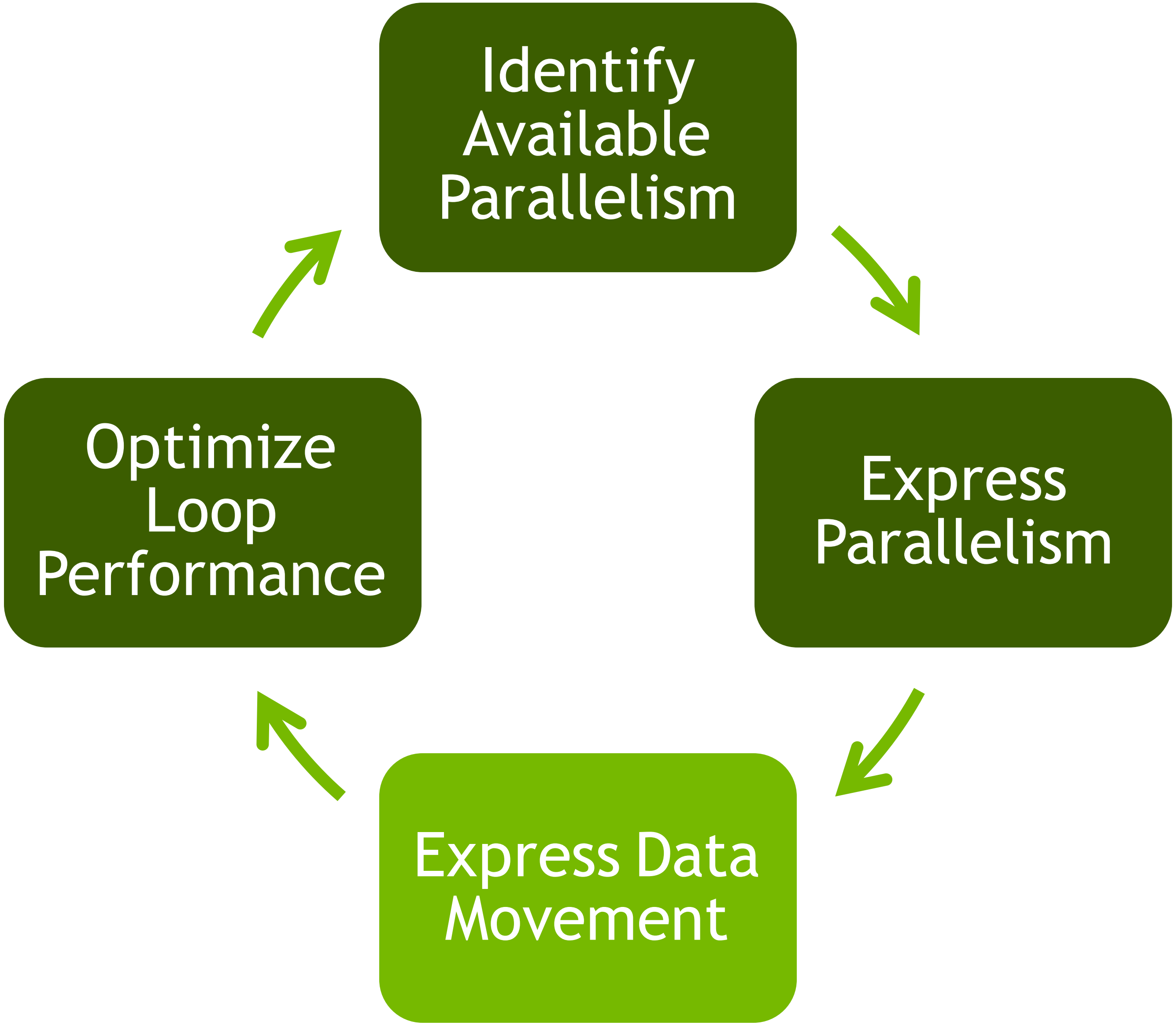
```
    ...
}
```

Identifying Data Locality

```
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc kernels
  {
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

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

  iter++;
}
```

Does the CPU need the data between these loop nests?

Does the CPU need the data between iterations of the convergence loop?



Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance

Data regions

The **`data`** directive defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```
#pragma acc data
{
#pragma acc kernels
...

#pragma acc kernels
...
}
```

Data Region

Arrays used within the data region will remain on the GPU until the end of the data region.

Data Clauses

`copy ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
`copyin ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region.
`copyout ( list )`	Allocates memory on GPU and copies data to the host when exiting region.
`create ( list )`	Allocates memory on GPU but does not copy.
`present ( list )`	Data is already present on GPU from another containing data region.
`deviceptr( list )`	The variable is a device pointer (e.g. CUDA) and can be used directly on the device.

Array Shaping

Compiler sometimes cannot determine size of arrays

- Must specify explicitly using data clauses and array “shape”

C/C++

```
#pragma acc data copyin(a[0:nelem]) copyout(b[s/4:3*s/4])
```

Fortran

```
!$acc data copyin(a(1:end)) copyout(b(s/4:3*s/4))
```

Note: data clauses can be used on `data`, `parallel`, or `kernels`

Express Data Locality

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;
#pragma acc kernels
{
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }

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

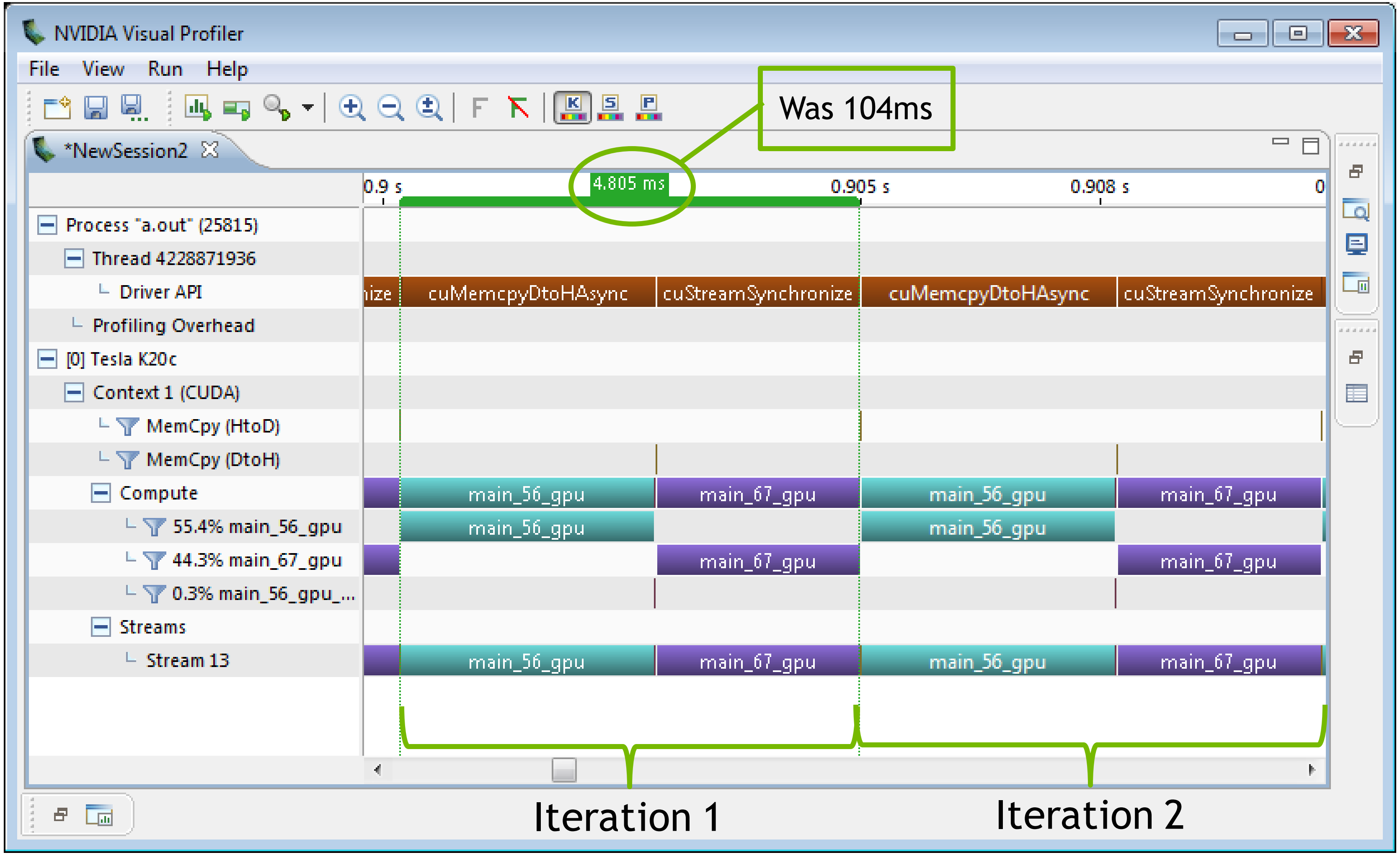
Copy A to/from the accelerator only when needed.

Create Anew as a device temporary.

Rebuilding the code

```
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:
     40, Loop not fused: function call before adjacent loop
         Generated vector sse code for the loop
     51, Generating copy(A[:][:])
         Generating create(Anew[:][:])
         Loop not vectorized/parallelized: potential early exits
     56, Accelerator kernel generated
         56, Max reduction generated for error
         57, #pragma acc loop gang /* blockIdx.x */
         59, #pragma acc loop vector(256) /* threadIdx.x */
     56, Generating Tesla code
     59, Loop is parallelizable
     67, Accelerator kernel generated
         68, #pragma acc loop gang /* blockIdx.x */
         70, #pragma acc loop vector(256) /* threadIdx.x */
     67, Generating Tesla code
     70, Loop is parallelizable
```

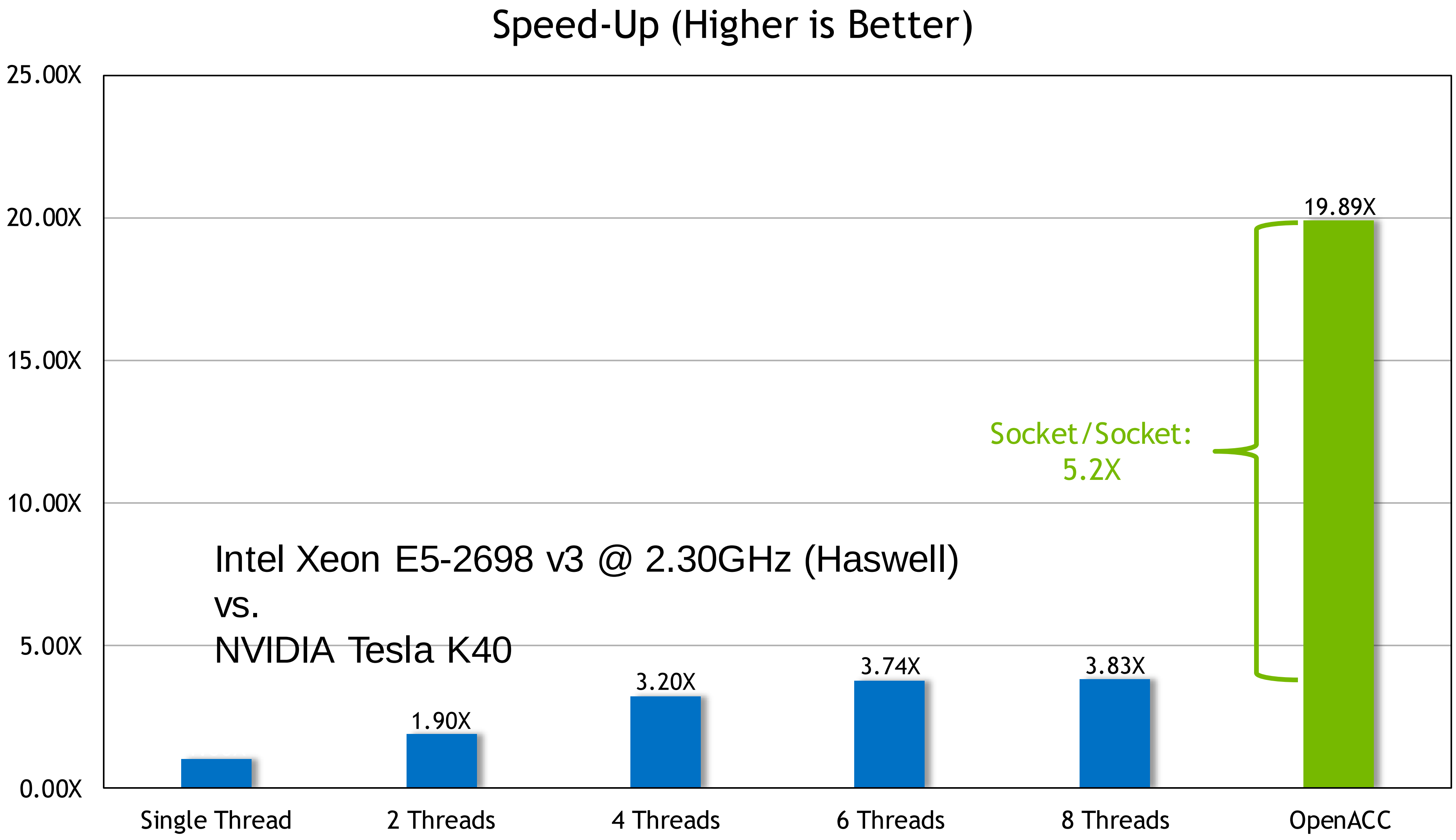
Visual Profiler: Data Region



Was 104ms

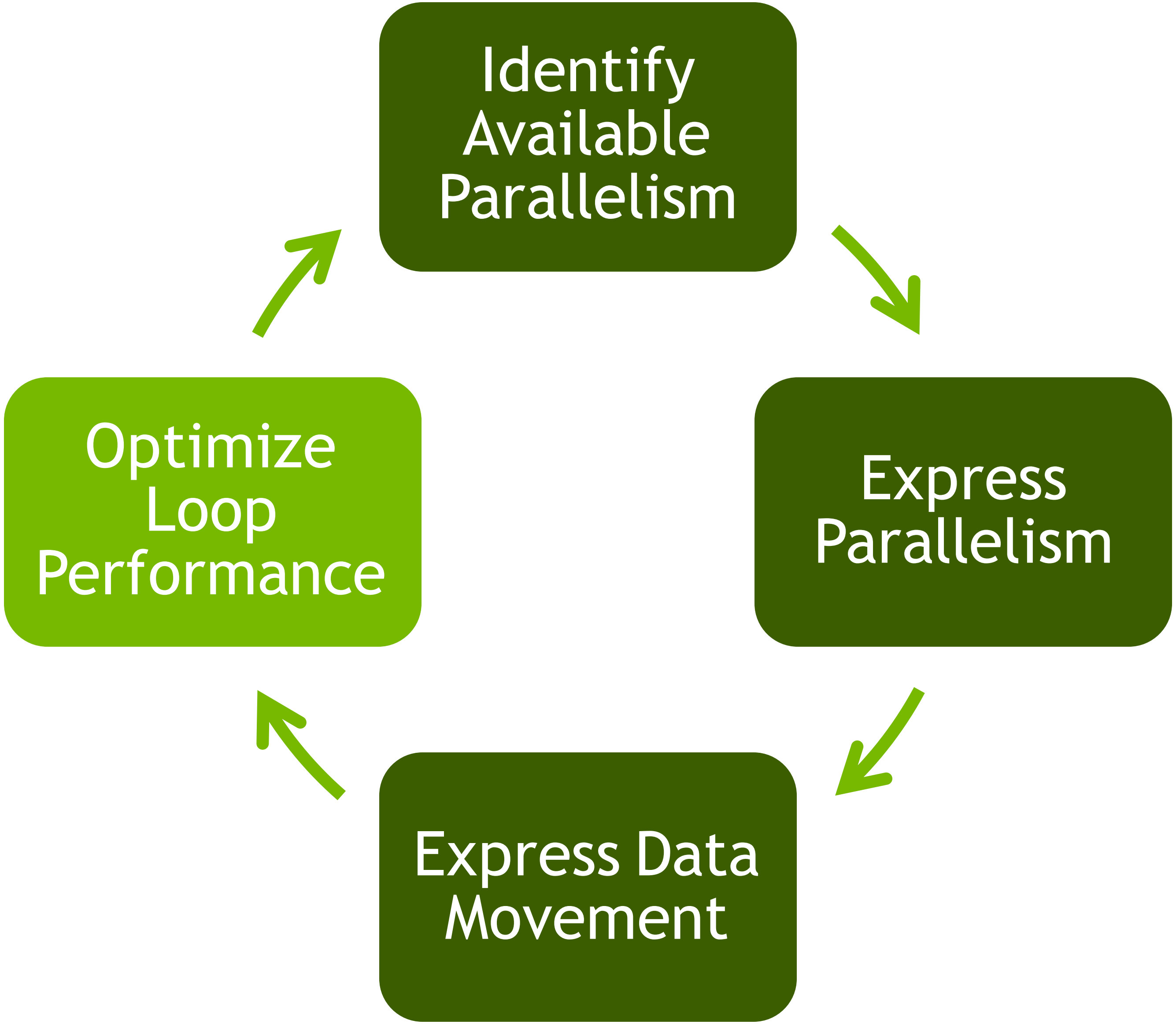
Iteration 1

Iteration 2



Socket/Socket:
5.2X

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell)
vs.
NVIDIA Tesla K40



Identify Available Parallelism

Express Parallelism

Express Data Movement

Optimize Loop Performance

The loop Directive

The `loop` directive gives the compiler additional information about the *next* loop in the source code through several clauses.

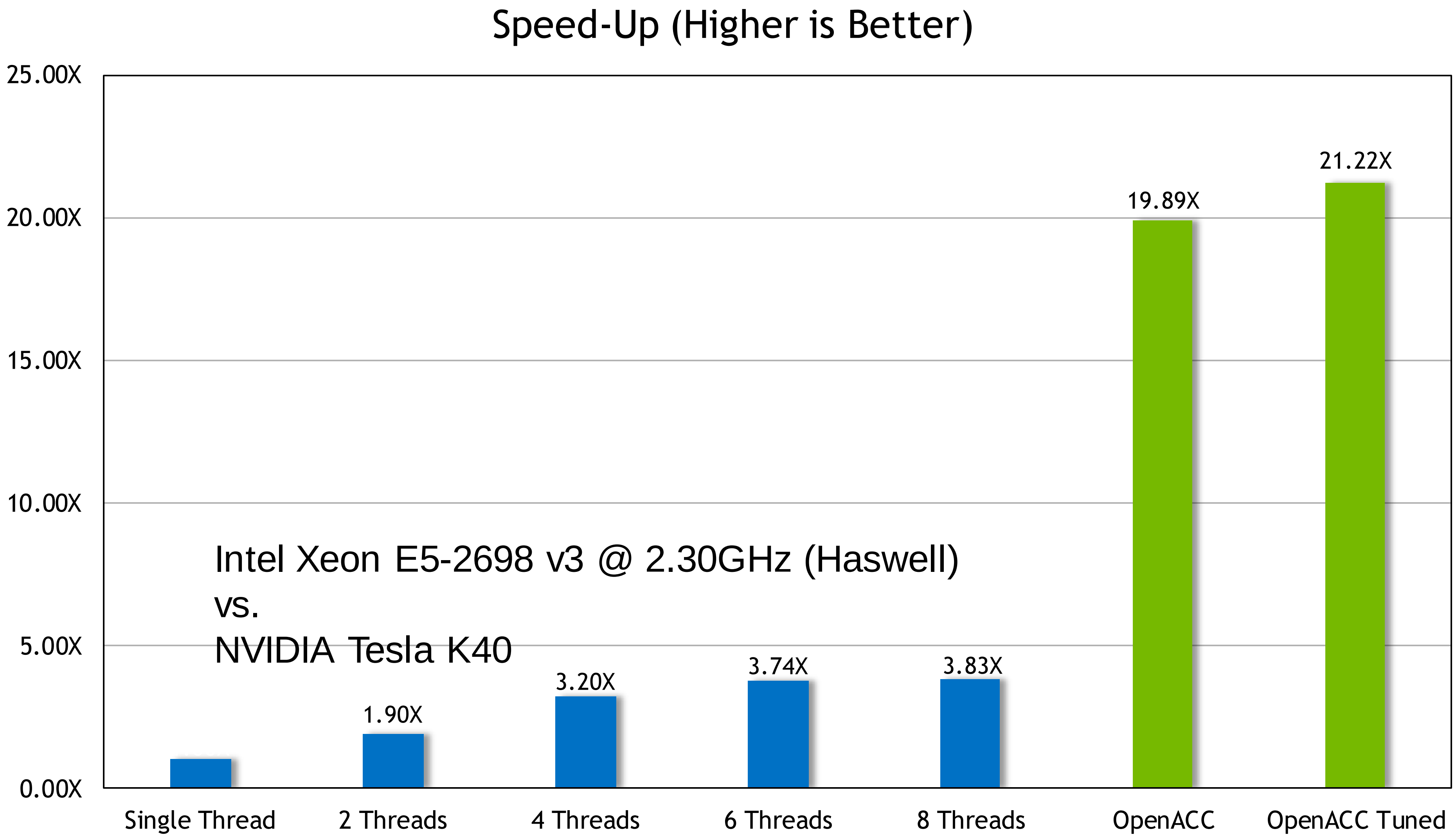
- **`independent`** – all iterations of the loop are independent
- **`collapse(N)`** – turn the next N loops into one, flattened loop
- **`tile(N[,M,…])`** - break the next 1 or more loops into *tiles* based on the provided dimensions.

These clauses and more will be discussed in greater detail in a later class.

Optimize Loop Performance

```
#pragma acc data copy(A) create(Anew)
while ( err > tol && iter < iter_max ) {
  err=0.0;
#pragma acc kernels
{
#pragma acc loop device_type(nvidia) tile(32,4)
  for( int j = 1; j < n-1; j++) {
    for(int i = 1; i < m-1; i++) {

      Anew[j][i] = 0.25 * (A[j][i+1] + A[j][i-1] +
                           A[j-1][i] + A[j+1][i]);

      err = max(err, abs(Anew[j][i] - A[j][i]));
    }
  }
#pragma acc loop device_type(nvidia) tile(32,4)
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
}
  iter++;
}
```

“Tile” the next two loops into 32x4 blocks, but only on NVIDIA GPUs.



Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell)
vs.
NVIDIA Tesla K40

The OpenACC Toolkit

Introducing the New OpenACC Toolkit

Free Toolkit Offers Simple & Powerful Path to Accelerated Computing

OpenACC
Directives for Accelerators

http://developer.nvidia.com/openacc

PGI Compiler
Free OpenACC compiler for academia

NVProf Profiler
Easily find where to add compiler directives

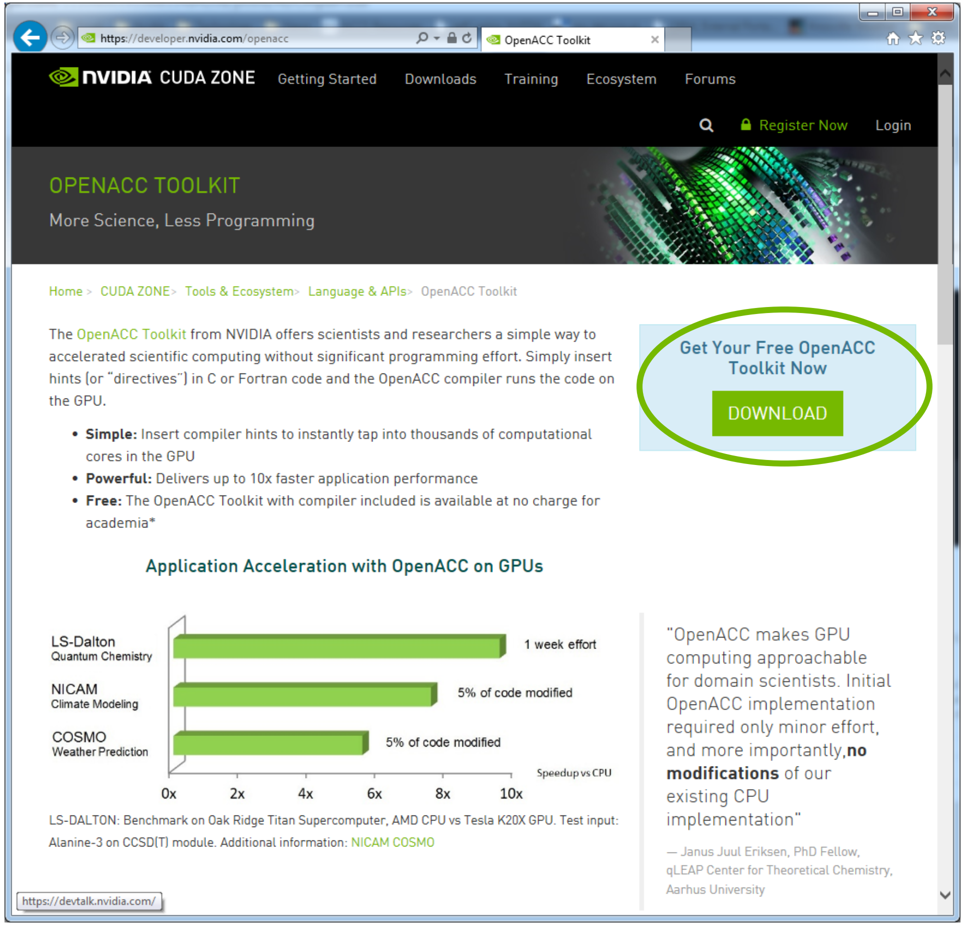
GPU Wizard
Identify which GPU libraries can jumpstart code

Code Samples
Learn from examples of real-world algorithms

Documentation
Quick start guide, Best practices, Forums

Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc



OPENACC TOOLKIT

More Science, Less Programming

Home > CUDA ZONE > Tools & Ecosystem > Language & APIs > OpenACC Toolkit

The OpenACC Toolkit from NVIDIA offers scientists and researchers a simple way to accelerated scientific computing without significant programming effort. Simply insert hints (or "directives") in C or Fortran code and the OpenACC compiler runs the code on the GPU.

- **Simple:** Insert compiler hints to instantly tap into thousands of computational cores in the GPU
- **Powerful:** Delivers up to 10x faster application performance
- **Free:** The OpenACC Toolkit with compiler included is available at no charge for academia*

Get Your Free OpenACC Toolkit Now

DOWNLOAD

Application Acceleration with OpenACC on GPUs

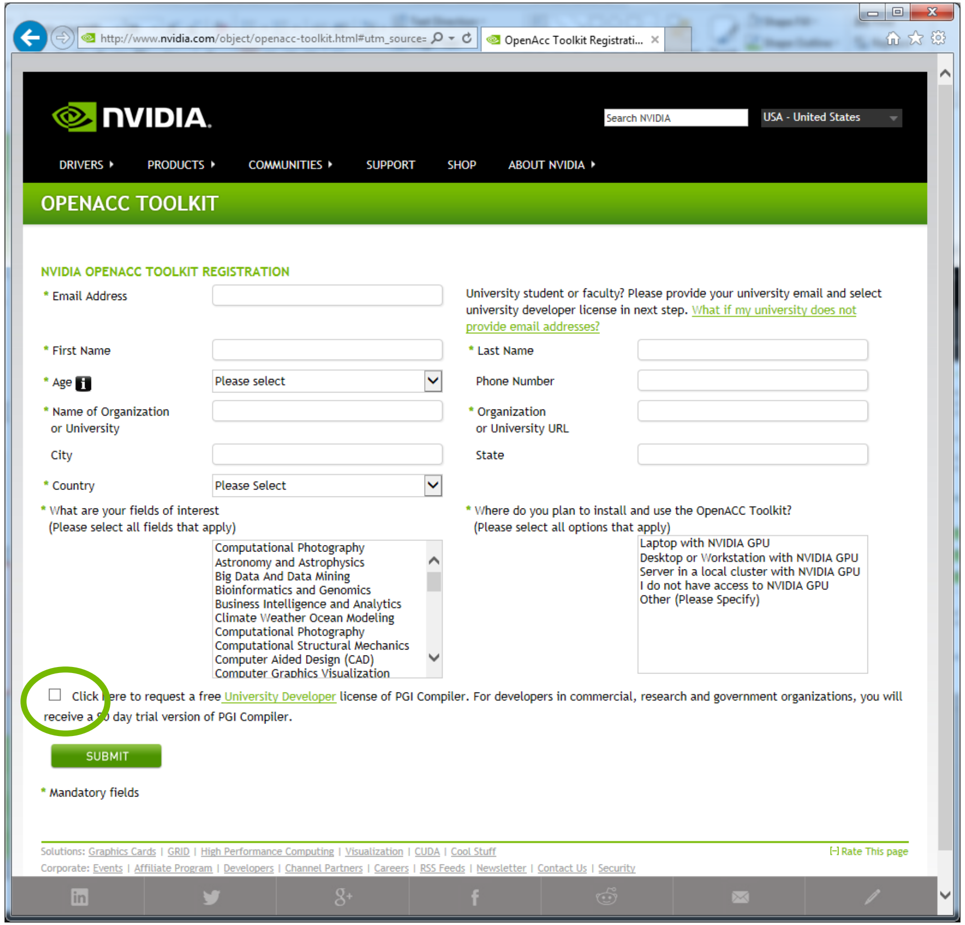
LS-DALTON: Benchmark on Oak Ridge Titan Supercomputer, AMD CPU vs Tesla K20X GPU. Test input: Alanine-3 on CCSD(T) module. Additional information: NICAM COSMO

"OpenACC makes GPU computing approachable for domain scientists. Initial OpenACC implementation required only minor effort, and more importantly, **no modifications** of our existing CPU implementation"

— Janus Juul Eriksen, PhD Fellow, qLEAP Center for Theoretical Chemistry, Aarhus University

Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc
- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.



OPENACC TOOLKIT

NVIDIA OPENACC TOOLKIT REGISTRATION

* Email Address

University student or faculty? Please provide your university email and select university developer license in next step. What if my university does not provide email addresses?

* First Name

* Last Name

* Age — Please select

Phone Number

* Name of Organization or University

* Organization or University URL

City

State

* Country — Please Select

* What are your fields of interest (Please select all fields that apply)

Computational Photography
Astronomy and Astrophysics
Big Data And Data Mining
Bioinformatics and Genomics
Business Intelligence and Analytics
Climate Weather Ocean Modeling
Computational Photography
Computational Structural Mechanics
Computer Aided Design (CAD)
Computer Graphics Visualization

* Where do you plan to install and use the OpenACC Toolkit? (Please select all options that apply)

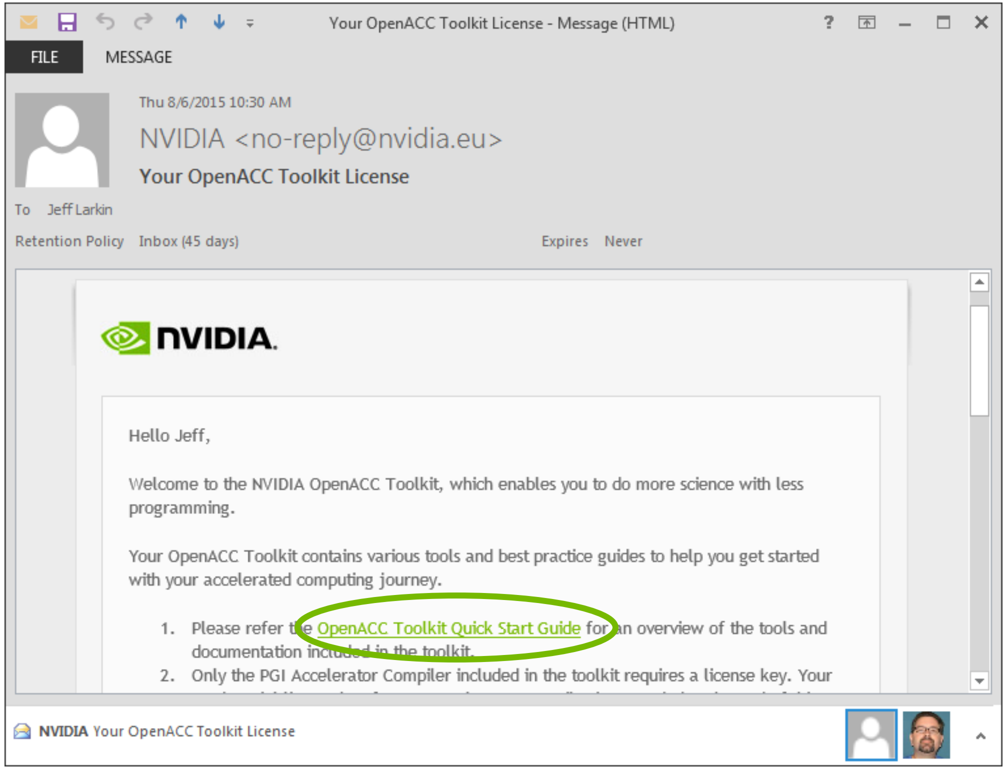
Laptop with NVIDIA GPU
Desktop or Workstation with NVIDIA GPU
Server in a local cluster with NVIDIA GPU
I do not have access to NVIDIA GPU
Other (Please Specify)

☐ Click here to request a free University Developer license of PGI Compiler. For developers in commercial, research and government organizations, you will receive a 90 day trial version of PGI Compiler.

* Mandatory fields

Download the OpenACC Toolkit

- Go to https://developer.nvidia.com/openacc
- Register for the toolkit
  - If you are an academic developer, be sure to click the check box at the bottom.
- You will receive an email from NVIDIA
  - Be sure to read the Quick Start Guide



Thu 8/6/2015 10:30 AM

NVIDIA <no-reply@nvidia.eu>

Your OpenACC Toolkit License

To Jeff Larkin

Retention Policy Inbox (45 days) Expires Never

Hello Jeff,

Welcome to the NVIDIA OpenACC Toolkit, which enables you to do more science with less programming.

Your OpenACC Toolkit contains various tools and best practice guides to help you get started with your accelerated computing journey.

1. Please refer to the OpenACC Toolkit Quick Start Guide for an overview of the tools and documentation included in the toolkit.
2. Only the PGI Accelerator Compiler included in the toolkit requires a license key. Your

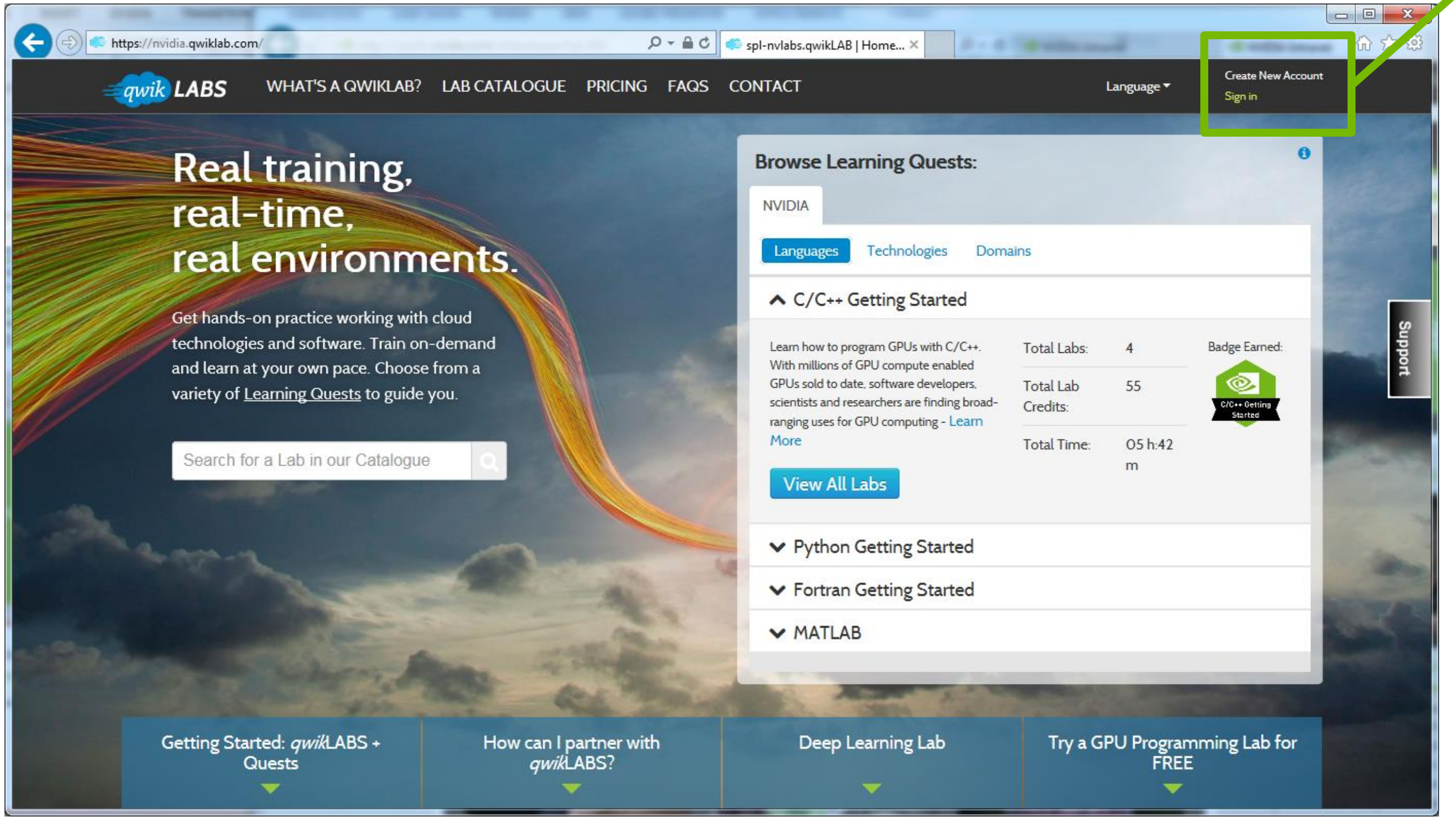
Windows/Mac Developers

- The OpenACC Toolkit is only available on Linux, however…
- The PGI compiler is available on Mac and Windows from http://www.pgroup.com/support/trial.htm
  - You should still register for the OpenACC Toolkit to get the 90 day license.
- The CUDA Toolkit contains the libraries and profiling tools that will be used in this course.
  - https://developer.nvidia.com/cuda-zone
- The OpenACC Programming Guide is available from http://bit.ly/openacc-guide
  - Obtaining all examples and guides from the toolkit will still require downloading the full OpenACC toolkit.

Using QwikLabs

Getting access

Go to nvidia.qwiklab.com, log-in or create an account



Real training, real-time, real environments.

Get hands-on practice working with cloud technologies and software. Train on-demand and learn at your own pace. Choose from a variety of Learning Quests to guide you.

Search for a Lab in our Catalogue

Browse Learning Quests:

Learn how to program GPUs with C/C++. With millions of GPU compute enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for GPU computing - Learn More

Total Labs:	4
Total Lab Credits:	55
Total Time:	05 h:42 m

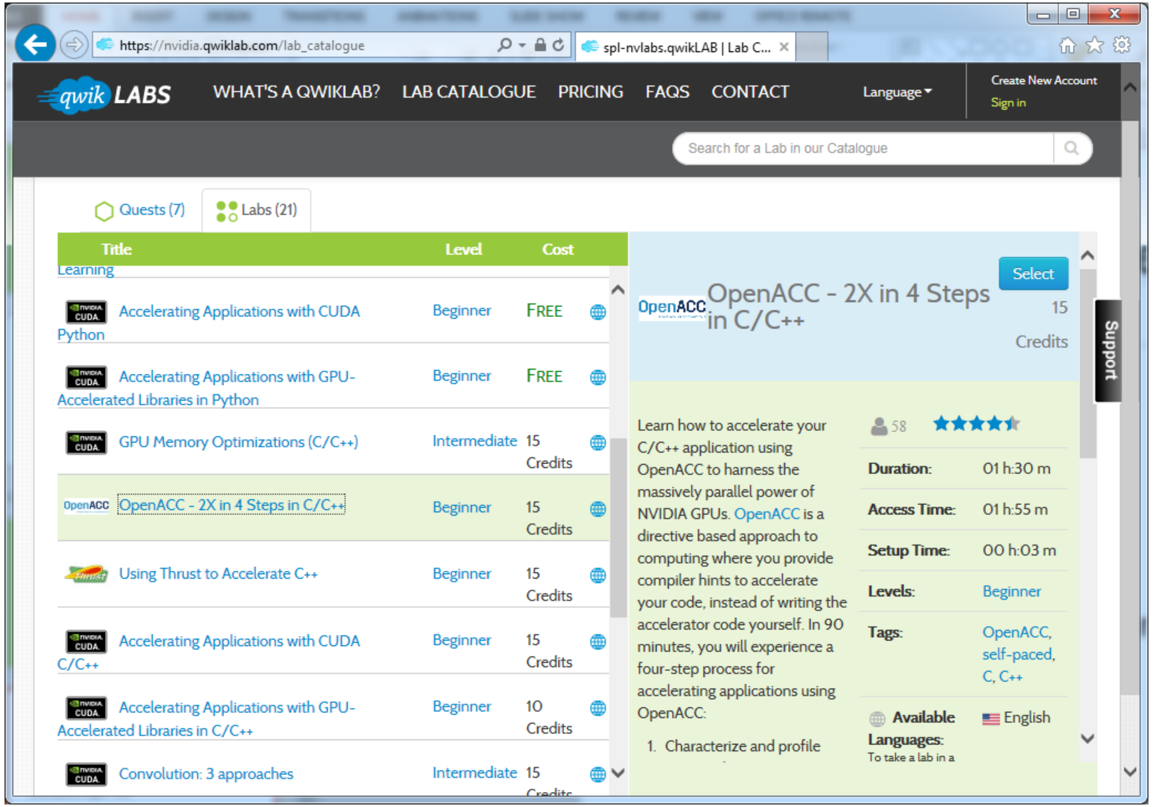
Sign In or Create a New Account

Homework

Complete “2X in 4 Steps” Qwiklab

- C: http://bit.ly/nvoacclab1c
- F90: http://bit.ly/nvoacclab1f

- This lab is browser-based and should take you roughly 1 hour.



Title	Level	Cost
Learning		
Accelerating Applications with CUDA Python	Beginner	FREE
Accelerating Applications with GPU-Accelerated Libraries in Python	Beginner	FREE
GPU Memory Optimizations (C/C++)	Intermediate	15 Credits
OpenACC - 2X in 4 Steps in C/C++	Beginner	15 Credits
Using Thrust to Accelerate C++	Beginner	15 Credits
Accelerating Applications with CUDA C/C++	Beginner	15 Credits
Accelerating Applications with GPU-Accelerated Libraries in C/C++	Beginner	10 Credits
Convolution: 3 approaches	Intermediate	15 Credits

OpenACC - 2X in 4 Steps in C/C++

15 Credits

Learn how to accelerate your C/C++ application using OpenACC to harness the massively parallel power of NVIDIA GPUs. OpenACC is a directive based approach to computing where you provide compiler hints to accelerate your code, instead of writing the accelerator code yourself. In 90 minutes, you will experience a four-step process for accelerating applications using OpenACC:

1. Characterize and profile

58 ★★★★★

Duration:	01 h:30 m
Access Time:	01 h:55 m
Setup Time:	00 h:03 m
Levels:	Beginner
Tags:	OpenACC, self-paced, C, C++
Available Languages:	English

To take a lab in a

Install the OpenACC Toolkit (Optional)

- Go to developer.nvidia.com/openacc
- Register for the OpenACC Toolkit
- Install on your personal machine. (Linux Only)

OPENACC TOOLKIT

More Science, Less Programming

The OpenACC Toolkit from NVIDIA offers scientists and researchers a simple way to accelerated scientific computing without significant programming effort. Simply insert hints (or "directives") in C or Fortran code and the OpenACC compiler runs the code on the GPU.

- **Simple:** Insert compiler hints to instantly tap into thousands of computational cores in the GPU
- **Powerful:** Delivers up to 10x faster application performance
- **Free:** The OpenACC Toolkit with compiler included is available at no charge for academia*

Get Your Free OpenACC Toolkit Now

Application Acceleration with OpenACC on GPUs

LS-DALTON: Benchmark on Oak Ridge Titan Supercomputer, AMD CPU vs Tesla K20X GPU. Test input: Alanine-3 on CCSD(T) module. Additional information: NICAM COSMO

"OpenACC makes GPU computing approachable for domain scientists. Initial OpenACC implementation required only minor effort, and more importantly, **no modifications** of our existing CPU implementation"

— Janus Juul Eriksen, PhD Fellow, qLEAP Center for Theoretical Chemistry, Aarhus University

Where to find help

- OpenACC Course Recordings - https://developer.nvidia.com/openacc-course
- OpenACC on StackOverflow - http://stackoverflow.com/questions/tagged/openacc
- OpenACC Toolkit - http://developer.nvidia.com/openacc

Additional Resources:

- Parallel Forall Blog - http://devblogs.nvidia.com/parallelforall/
- GPU Technology Conference - http://www.gputechconf.com/
- OpenACC Website - http://openacc.org/

Course Syllabus

Recordings:
https://developer.nvidia.com/openacc-course

Oct 1: Introduction to OpenACC

Oct 6: Office Hours

Oct 15: Profiling and Parallelizing with the OpenACC Toolkit

Oct 20: Office Hours

Oct 29: Expressing Data Locality and Optimizations with OpenACC

Nov 3: Office Hours

Nov 12: Advanced OpenACC Techniques

Nov 24: Office Hours