OPENACC ONLINE COURSE

Module 1 – Introduction to OpenACC

Robert Searles
NVIDIA Corporation

OpenACC
More Science, Less Programming

NVIDIA

ABOUT THIS COURSE

3 Part Introduction to OpenACC

- Module 1 – Introduction to OpenACC
- Module 2 – Data Management with OpenACC
- Module 3 – Optimizations with OpenACC

Each module will have a corresponding lab

COURSE OBJECTIVE

Enable ***YOU*** to accelerate ***YOUR*** applications with OpenACC.

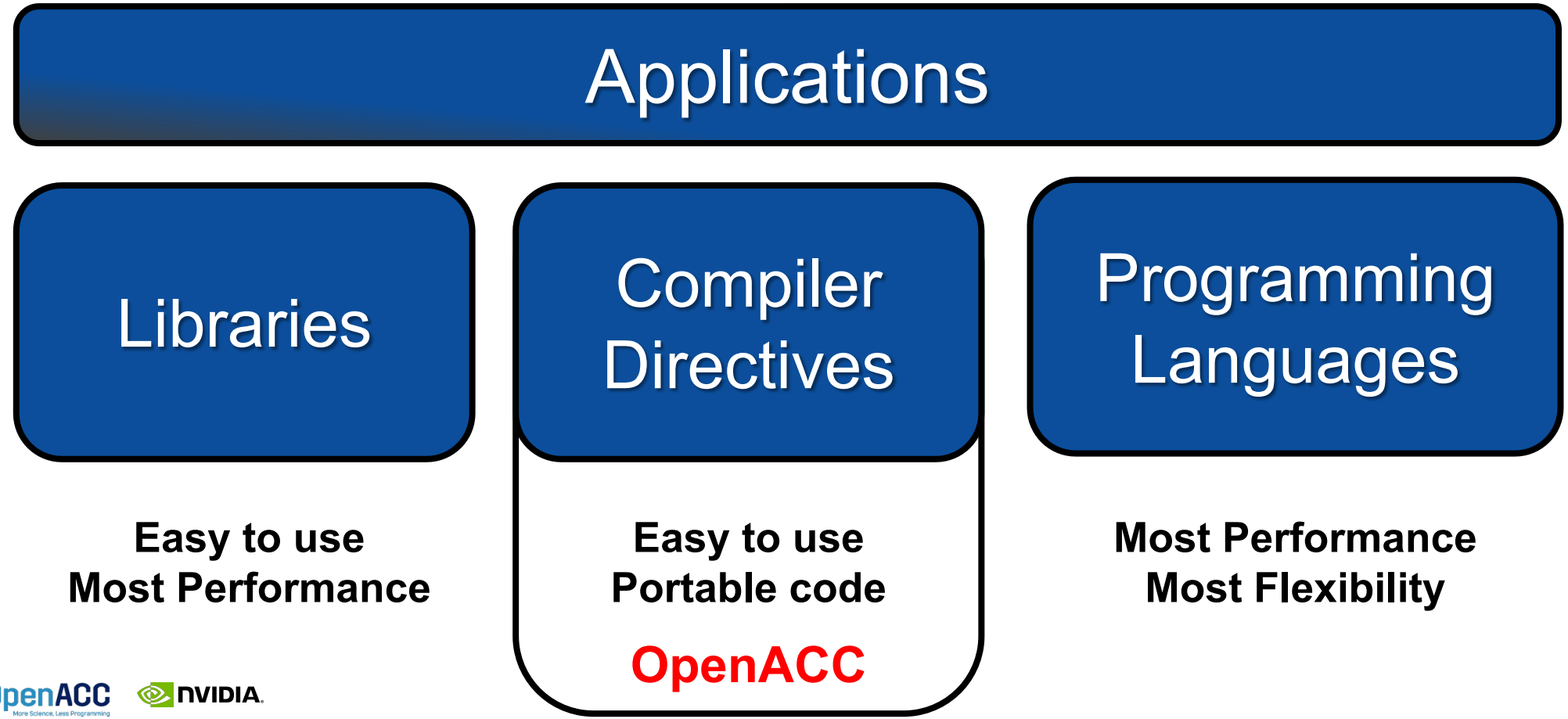
MODULE 1 OUTLINE

Topics to be covered

- What is OpenACC and Why Should You Care?
- Profile-driven Development
- First Steps with OpenACC
- Lab 1
- Where to Get Help

INTRODUCTION TO OPENACC

3 WAYS TO ACCELERATE APPLICATIONS



OpenACC More Science, Less Programming

NVIDIA

OPENACC IS…

a directives-based **parallel programming model** designed for **performance** and **portability**.

Add Simple Compiler Directive

```
main()
 {
  <serial code>
  #pragma acc kernels
  {
    <parallel code>
  }
}
```

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
      for (i = 0; i < n; ++i) {
  #pragma acc loop vector
          for (j = 0; j < n; ++j) {
            c[i][j] = a[i][j] + b[i][j];
            ...
          }
      }
  }
  ...
}
```

OpenACC
Directives for Accelerators

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

OPENACC STRENGTHS

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Single Source

- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

Low Learning Curve

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No requirement to learn low-level details of the hardware.

OPENACC: INCREMENTAL

Incremental

- Maintain existing sequential code
- Add annotations to expose parallelism
- After verifying correctness, annotate more of the code

Enhance Sequential Code

```
#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
    < loop code >
}

#pragma acc parallel loop
for( i = 0; i < N; i++ )
{
    < loop code >
}
```

Begin with a working sequential code.

Parallelize it with OpenACC.

Rerun the code to verify correctness and performance

OPENACC: SINGLE SOURCE

Supported Platforms

POWER

Sunway

x86 CPU

ARM CPU

AMD GPU

NVIDIA GPU

PEZY-SC

Single Source

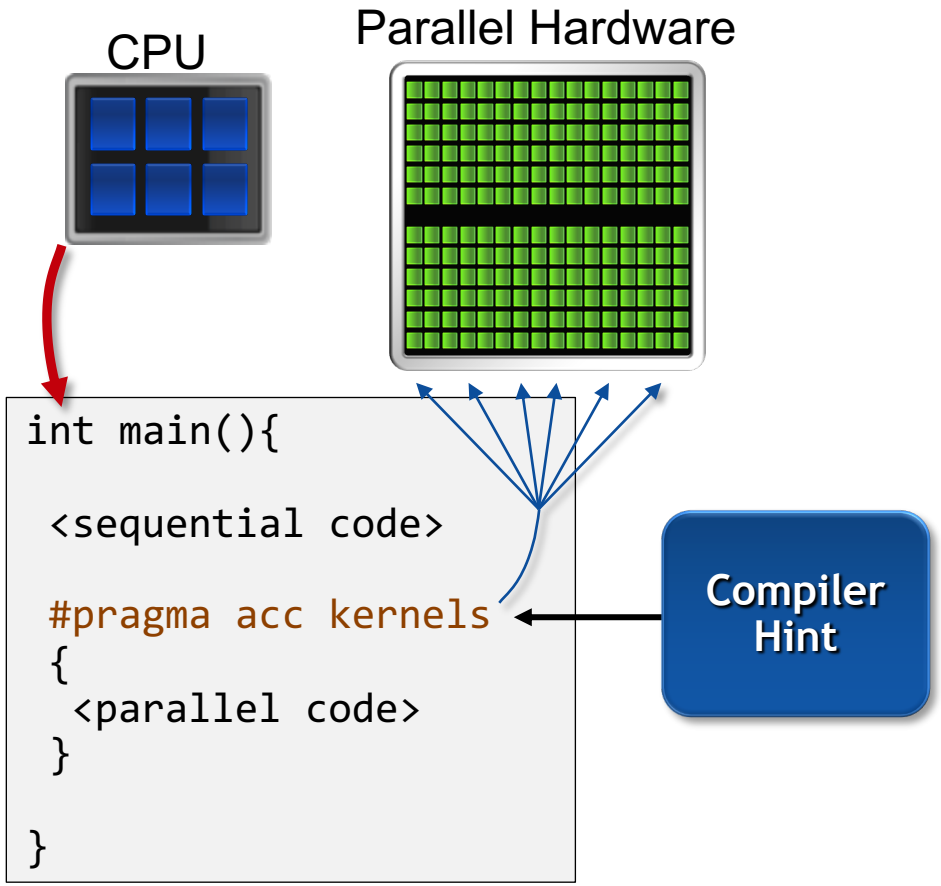
- Rebuild the same code on multiple architectures
- Compiler determines how to parallelize for the desired machine
- Sequential code is maintained

The compiler can **ignore** your OpenACC code additions, so the same code can be used for **parallel** or **sequential** execution.

```
int main(){

...

   #pragma acc parallel loop
   for(int i = 0; i < N; i++)
      < loop code >

}
```

OpenACC More Science, Less Programming NVIDIA

OPENACC: PROGRAMMABILITY



The programmer will give hints to the compiler.

The compiler parallelizes the code.

Low Learning Curve

- OpenACC is meant to be easy to use, and easy to learn
- Programmer remains in familiar C, C++, or Fortran
- No requirement to learn low-level details of the hardware.

OpenACC More Science, Less Programming NVIDIA

DIRECTIVE-BASED HPC PROGRAMMING

Who's Using OpenACC?

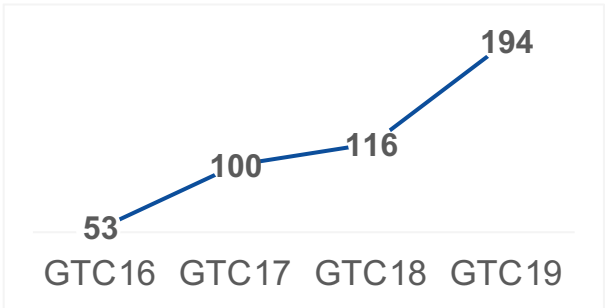
3 OF TOP 5 HPC APPS

Intersect360 RESEARCH

5 OF 13 CAAR CODES

summit

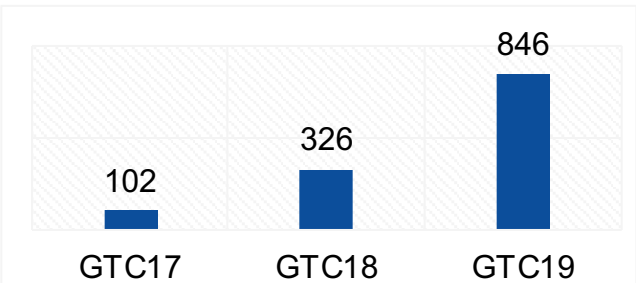
ACCELERATED APPS



725 TRAINED EXPERTS

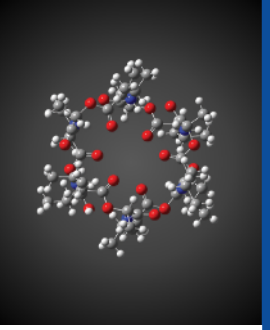
OpenACC HACKATHON
Advancing High Performance Computing

SLACK MEMBERS



160,000+ DOWNLOADS

PGI®
Community EDITION



GAUSSIAN 16

Mike Frisch, Ph.D.
President and
CEO
Gaussian, Inc.

"Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/network and GPU parallelism. PGI's compilers were essential to the success of our efforts."

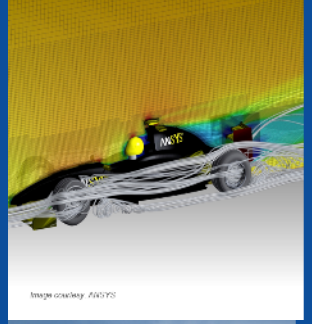
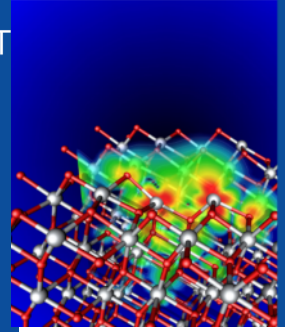


Image courtesy: ANSYS

ANSYS FLUENT

Sunil Sathe
Lead Software Developer
ANSYS Fluent

"We've effectively used OpenACC for heterogeneous computing in ANSYS Fluent with impressive performance. We're now applying this work to more of our models and new platforms."



VASP

Prof. Georg Kresse
Computational Materials Physics
University of Vienna

"For VASP, OpenACC is *the* way forward for GPU acceleration. Performance is similar and in some cases better than CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts. We're excited to collaborate with NVIDIA and PGI as an early adopter of CUDA Unified Memory."

COSMO

Dr. Oliver Fuhrer
Senior Scientist
Meteoswiss

"OpenACC made it practical to develop for GPU-based hardware while retaining a single source for almost all the COSMO physics code."

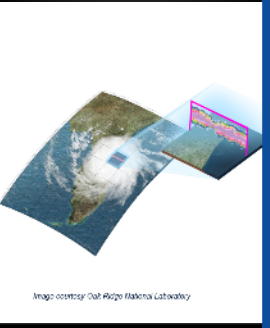
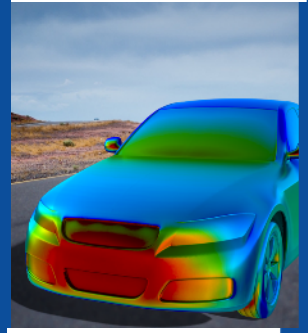


Image courtesy: Oak Ridge National Laboratory

E3SM

Mark A. Taylor
Multiphysics Applications
Sandia

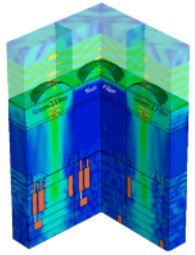
"The CAAR project provided us with early access to Summit hardware and access to PGI compiler experts. Both of these were critical to our success. PGI's OpenACC support remains the best available and is competitive with much more intrusive programming model approaches."



NUMECA FINE/Open

Davic Gutzwiller
Lead Software Developer
NUMECA

"Porting our unstructured C++ CFD solver FINE/Open to GPUs using OpenACC would have been impossible two or three years ago, but OpenACC has developed enough that we're now getting some really good results."



SYNOPSYS

Dr. Lutz Schneider
Senior R&D Engineer
Synopsys Inc.

"Using OpenACC, we've GPU-accelerated the Synopsys TCAD Sentaurus Device EMW simulator to speed up optical simulations of image sensors. GPUs are key to improving simulation throughput in the design of advanced image sensors."

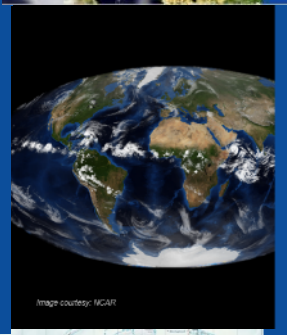
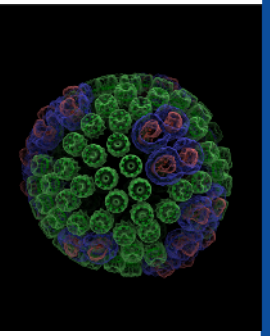


Image courtesy: NCAR

MPAS-A

Richard Loft
Director, Technology Development
NCAR

"Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 7.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer."



VMD

John Stone
Senior Research Programmer
Beckham Institute
University of Illinois

"Due to Amdahl's law, we need to port more parts of our code to the GPU if we're going to speed it up. But the sheer number of routines poses a challenge. OpenACC directives give us a low-cost approach to getting at least some speed-up out of these second-tier routines. In many cases it's completely sufficient because with the current algorithms, GPU performance is bandwidth-bound."

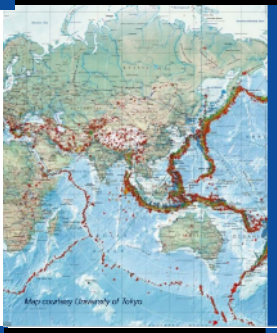
GTC

Zhihong Lin
Professor and Principal Investigator
UC Irvine

"Using OpenACC our scientists were able to achieve the acceleration needed for integrated fusion simulation with a minimum investment of time and effort in learning to program GPUs."

OpenACC

More Science, Less Programming



Map courtesy University of Tokyo

GAMERA

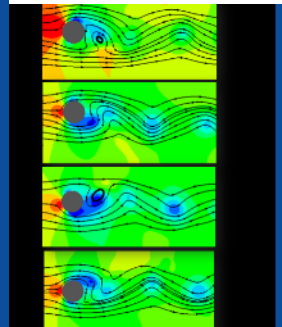
Takuma Yamaguchi, Kohei Fujita, Tsuyoshi Ichimura, Muneo Hori, Lalith Wijerathne
The University of Tokyo

"With OpenACC and a compute node based on NVIDIA's Tesla P100 GPU, we achieved more than a 14X speed up over a K Computer node running our earthquake disaster simulation code"

SANJEEVINI

Abhilash Jayaraj
Project Scientist
Indian Institute of Technology
New Delhi

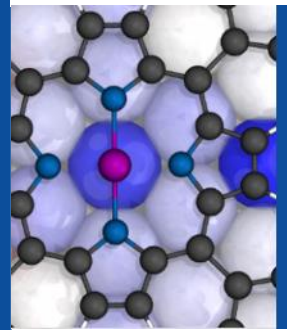
"In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task."



IBM-CFD

Somnath Roy
Assistant Professor
Mechanical Engineering Department
Indian Institute of Technology Kharagpur

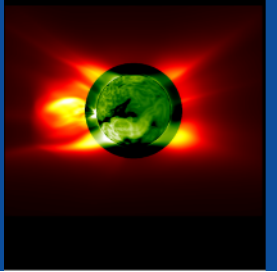
"OpenACC can prove to be a handy tool for computational engineers and researchers to obtain fast solution of non-linear dynamics problem. In immersed boundary incompressible CFD, we have obtained order of magnitude reduction in computing time by porting several components of our legacy codes to GPU. Especially the routines involving search algorithm and matrix solvers have been well-accelerated to improve the overall scalability of the code."



PWscf (Quantum ESPRESSO)

Filippo Spiga
Senior Contributor
Quantum ESPRESSO group

"CUDA Fortran gives us the full performance potential of the CUDA programming model and NVIDIA GPUs. While leveraging the potential of explicit data movement, !$CUF KERNELS directives give us productivity and source code maintainability. It's the best of both worlds."



MAS

Ronald M. Caplan
Computational Scientist
Predictive Science Inc.

"Adding OpenACC into MAS has given us the ability to migrate medium-sized simulations from a multi node CPU cluster to a single multi-GPU server. The implementation yielded a portable single-source code for both CPU and GPU runs. Future work will add OpenACC to the remaining model features, enabling GPU-accelerated realistic solar storm modeling."

OPENACC SYNTAX

OPENACC SYNTAX

Syntax for using OpenACC directives in code

C/C++

```
#pragma acc directive clauses
<code>
```

Fortran

```
!$acc directive clauses
<code>
```

- A ***pragma*** in C/C++ gives instructions to the compiler on how to compile the code. Compilers that do not understand a particular pragma can freely ignore it.
- A ***directive*** in Fortran is a specially formatted comment that likewise instructions the compiler in it compilation of the code and can be freely ignored.
- "***acc"*** informs the compiler that what will come is an OpenACC directive
- ***Directives*** are commands in OpenACC for altering our code.
- ***Clauses*** are specifiers or additions to directives.

EXAMPLE CODE

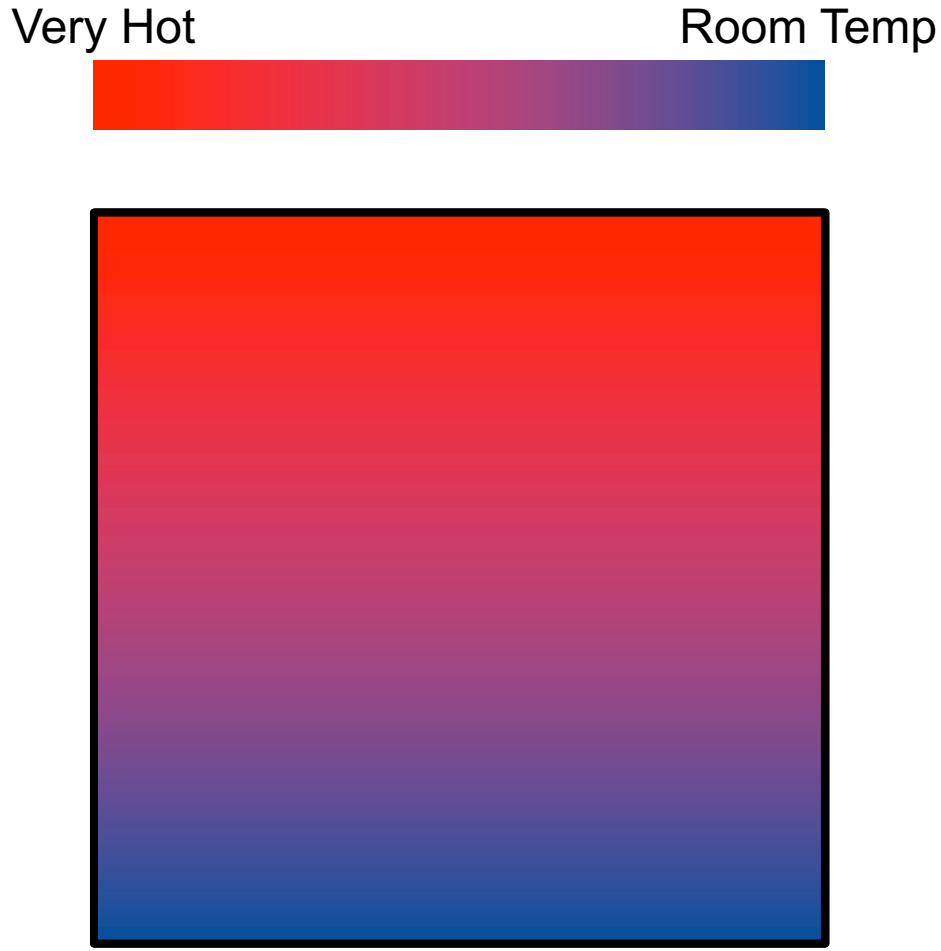
LAPLACE HEAT TRANSFER

Introduction to lab code - visual

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

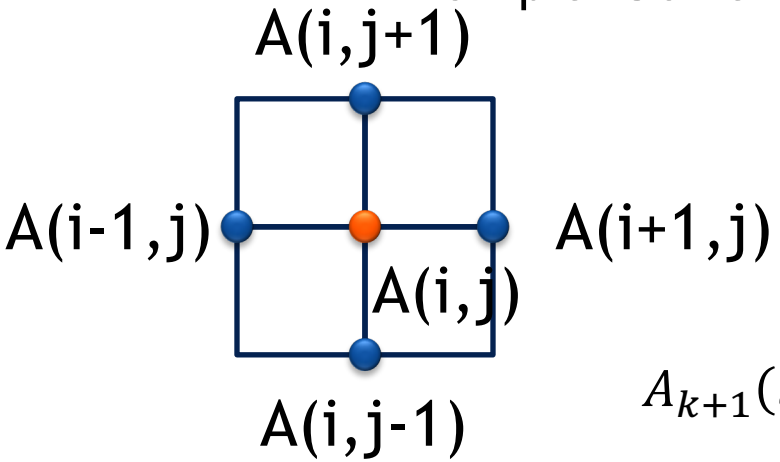
We will apply a consistent heat to the top of the plate.

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



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: $\nabla^2 f(x, y) = 0$



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

JACOBI ITERATION: C CODE

```
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

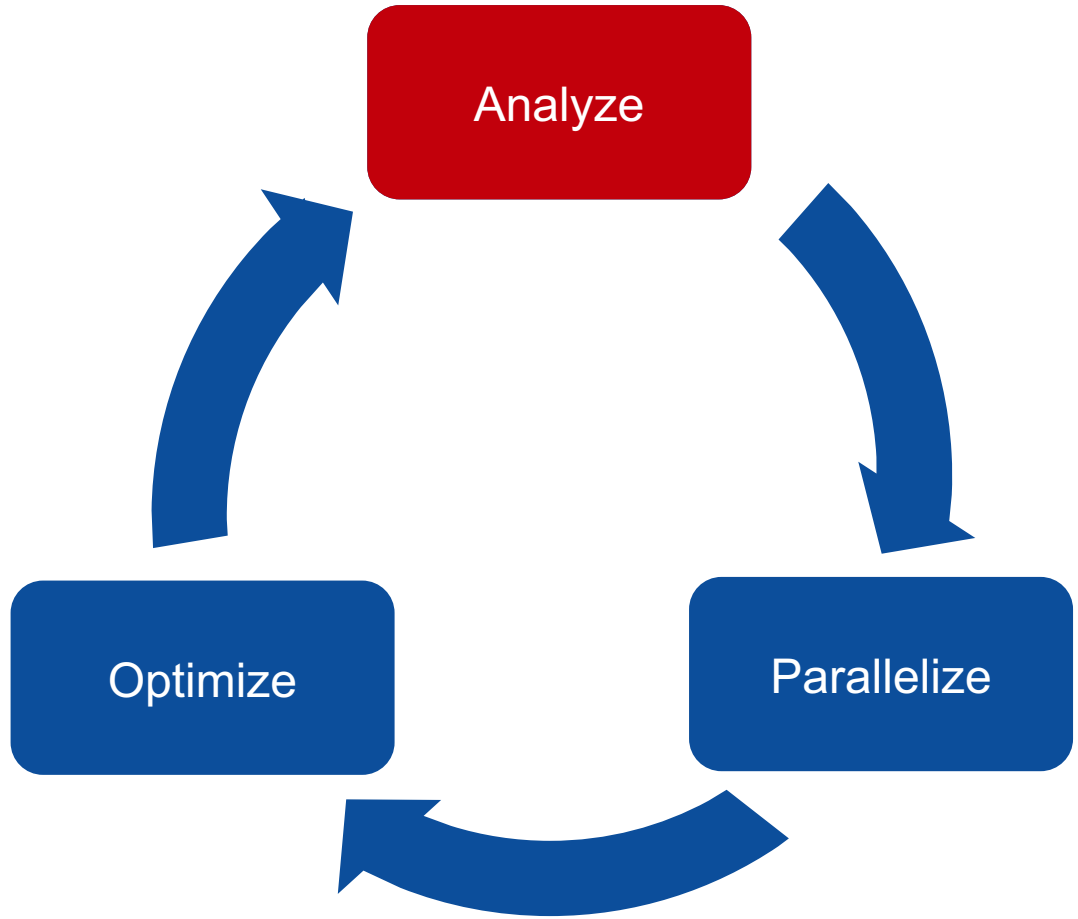
Compute max error for convergence

Swap input/output arrays

PROFILE-DRIVEN DEVELOPMENT

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 and check for correctness.
- **Optimize** your code to improve observed speed-up from parallelization.



PROFILING SEQUENTIAL CODE

Profile Your Code

Obtain detailed information about how the code ran.

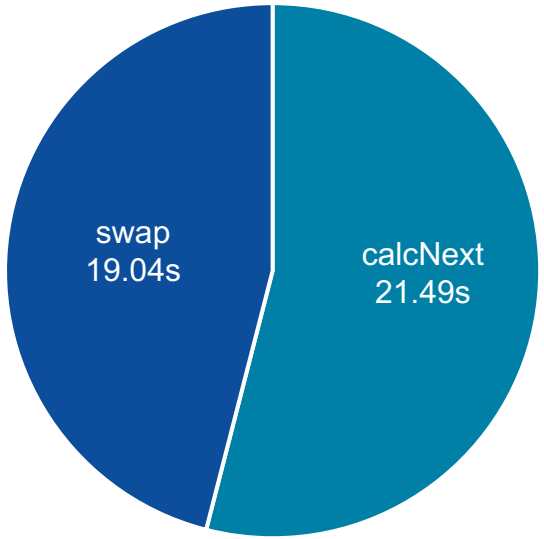
This can include information such as:

- 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

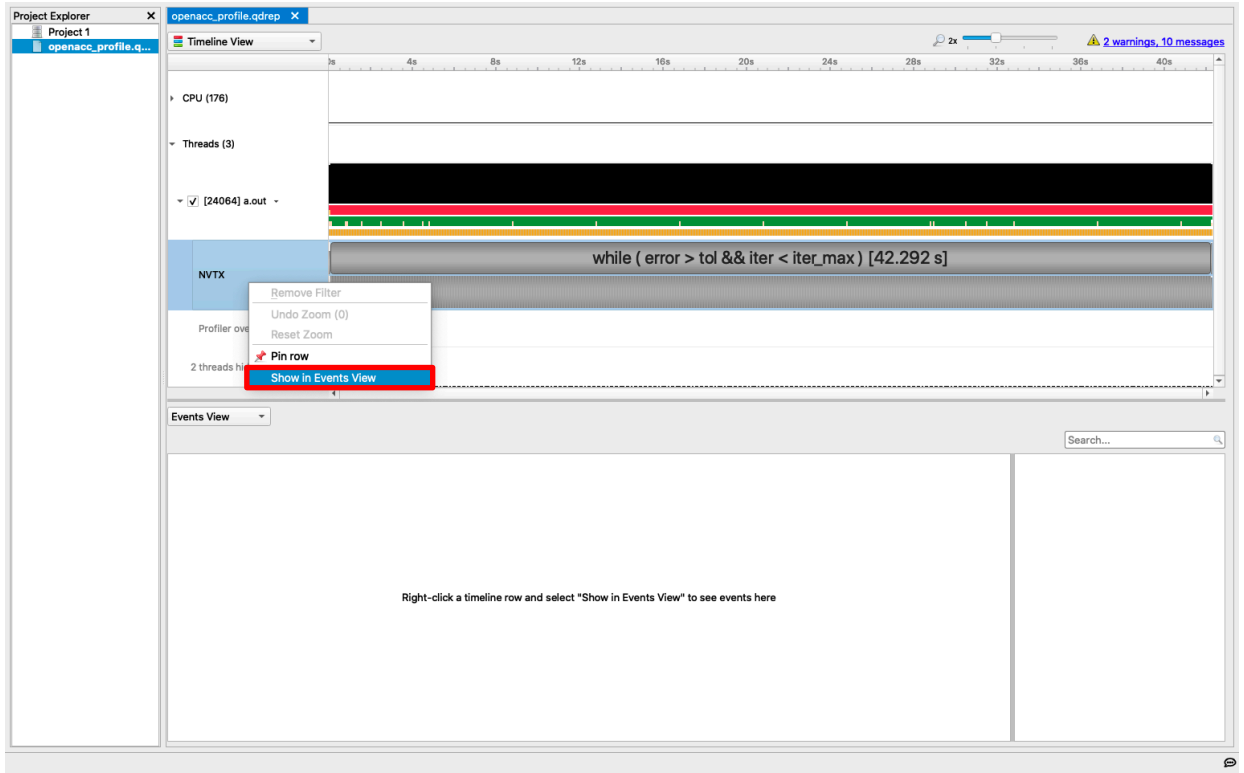
Total Runtime: 39.43 seconds



PROFILING SEQUENTIAL CODE

First sight when using NSight Systems

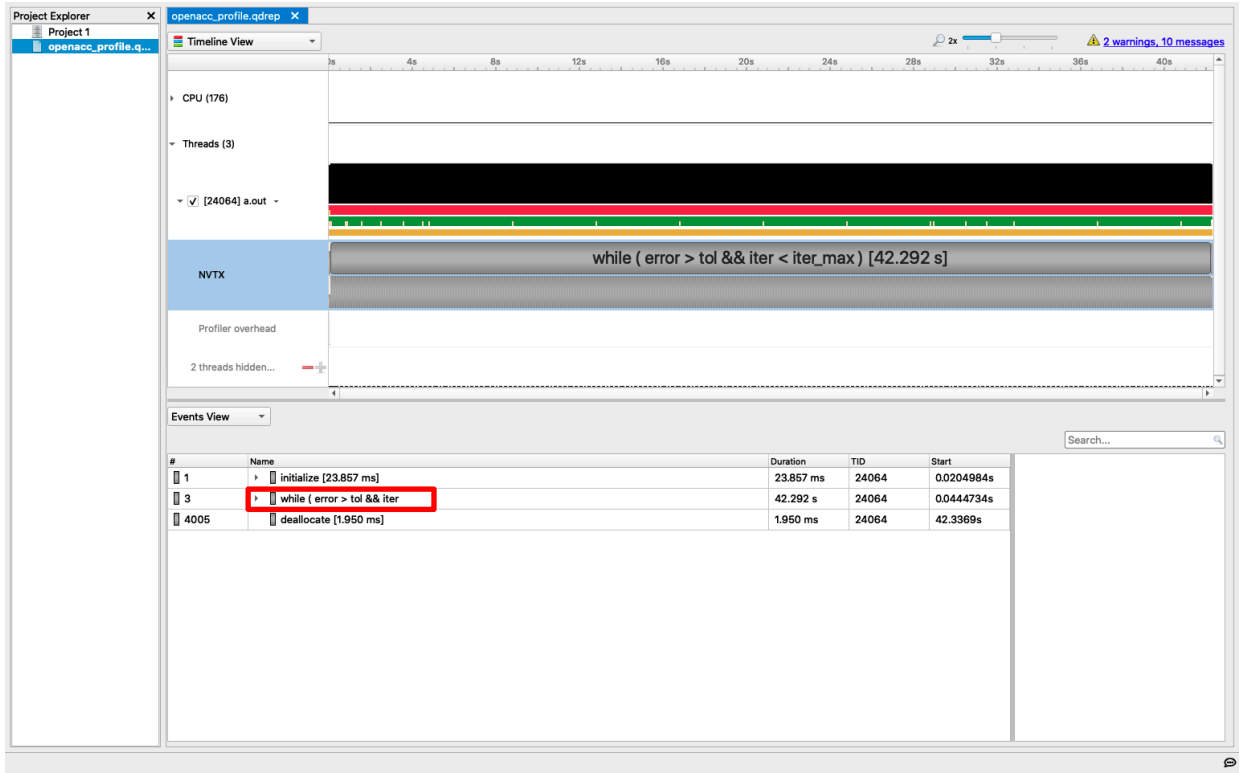
- Profiling a simple, sequential code
- Our sequential program will on run on the CPU.
- To view information about how our code ran, we should right click the “NVTX” row and select “Show in Events View”.



PROFILING SEQUENTIAL CODE

CPU Details

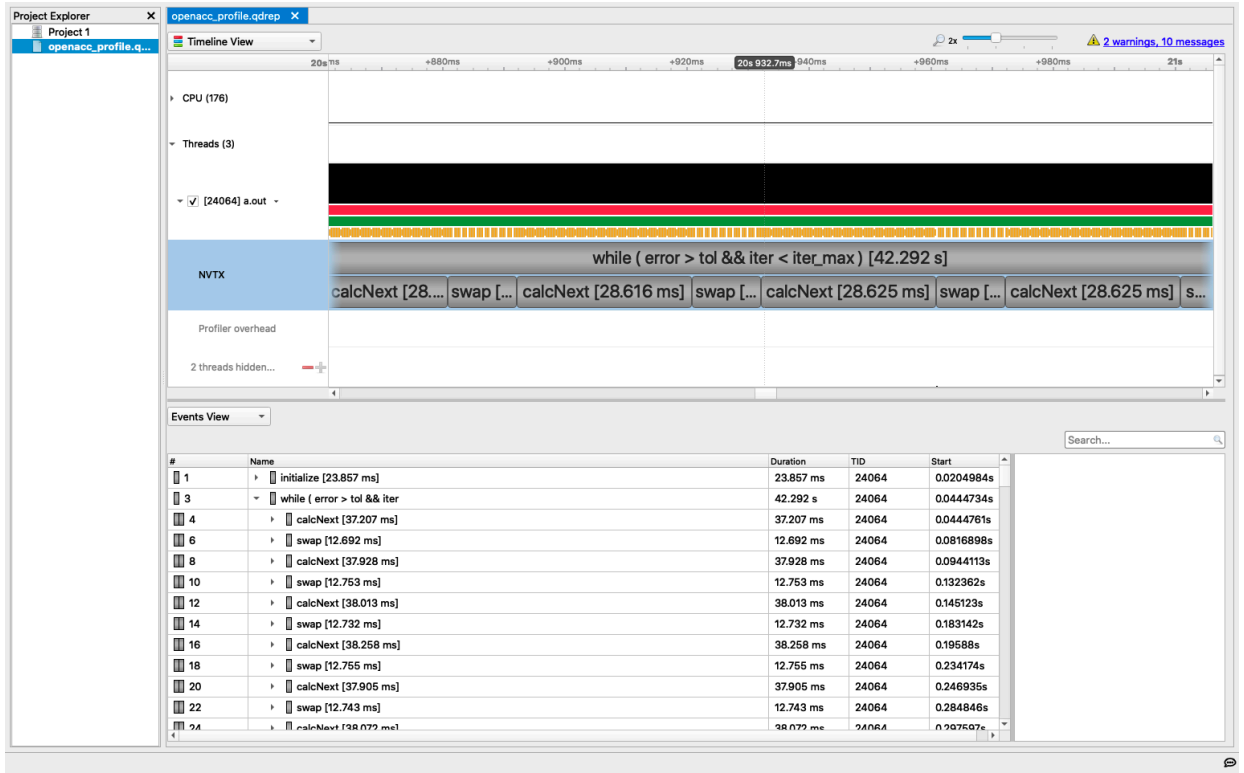
- Within the “Events View” tab, we can see the various parts of our code, and how long they took to run.
- We see the “while” loop taking up the majority of our runtime (42.2 s).
- We can click the arrow to examine the calls made within this loop.



PROFILING SEQUENTIAL CODE

CPU Details

- We see repeated calls to “calcNext” and “swap” within our while loop. We will focus on these!
- We also can see this on our zoomed in timeline.
- NVTX lets us push ranges onto a stack. A new row within the NVTX section will be created for each nested range.



PROFILING SEQUENTIAL CODE

CPU Details

- Nsight's --stats flag also prints out runtime statistics
- We see that initialize and deallocate take almost no runtime
- calcNext accounts for 36.2% of total runtime
- swap accounts for 13.8% of total runtime

Time(%)	Total Time	Instances	Average	Minimum	Maximum	Range
50.0	42293194393	1	42293194393.0	42293194393	42293194393	while (error > tol && iter < iter_max)
36.2	30601618780	1000	30601618.8	28602296	40445804	calcNext
13.8	11673978245	1000	11673978.2	11322706	13462161	swap
0.0	23810757	1	23810757.0	23810757	23810757.	initialize
0.0	1973444	1	1973444.0	1973444	1973444	deallocate

OPENACC PARALLEL LOOP DIRECTIVE

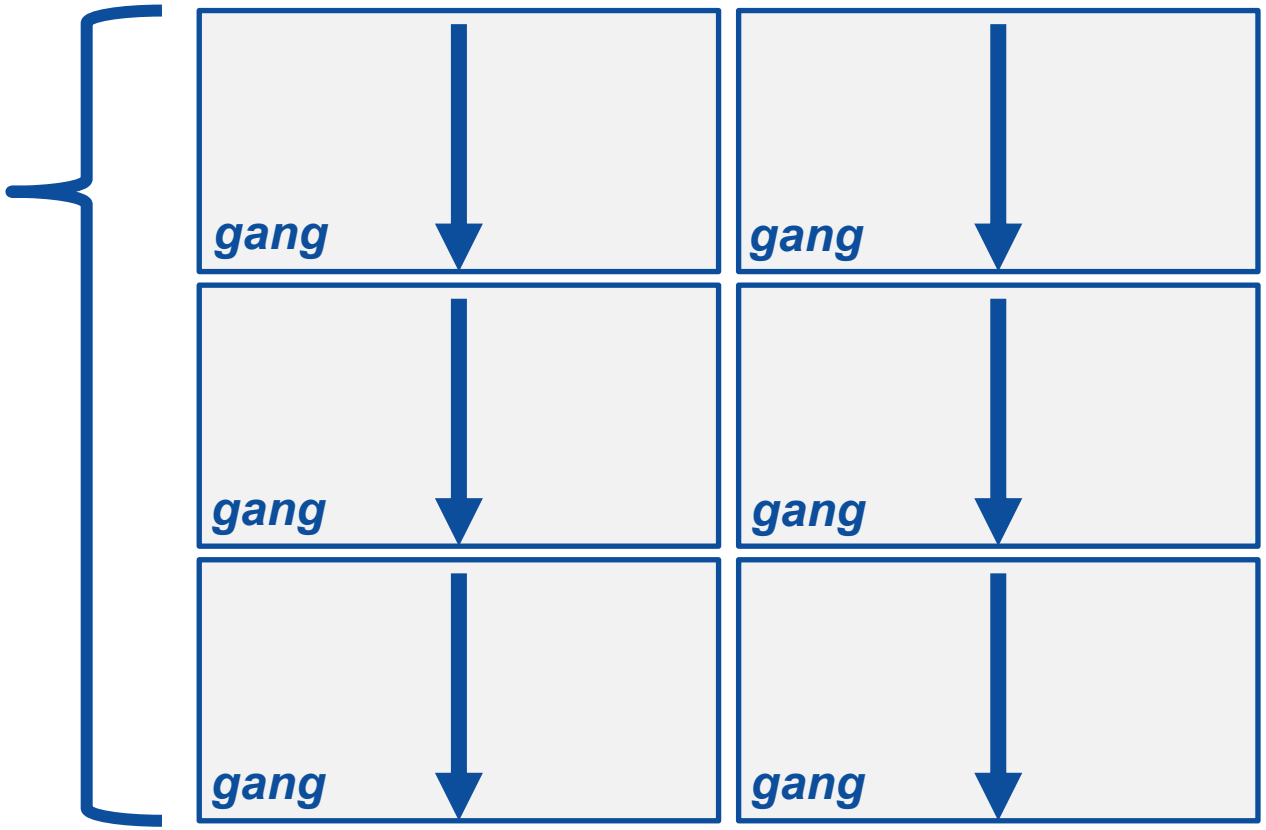
OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{
```

When encountering the ***parallel*** directive, the compiler will generate *1 or more parallel* ***gangs***, which execute redundantly.

```
}
```



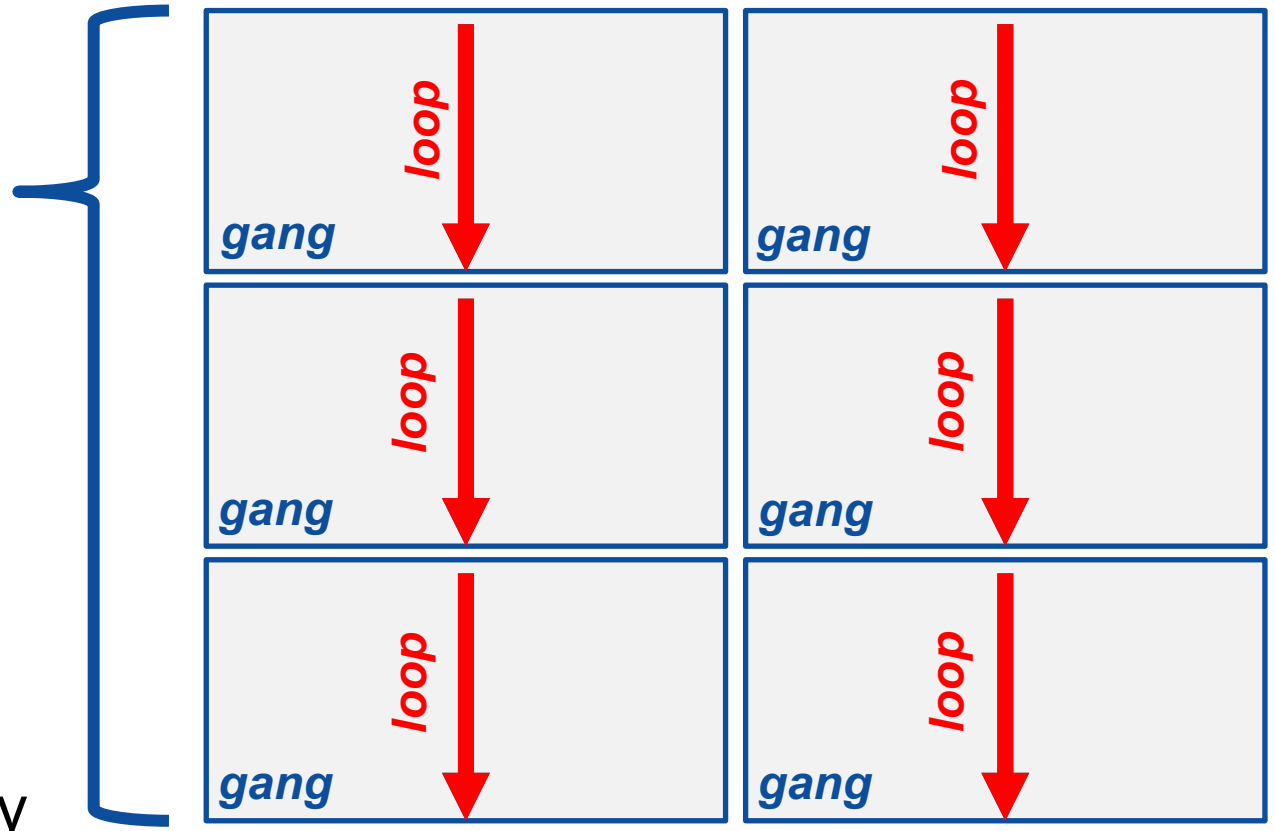
OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{

    for(int i = 0; i < N; i++)
    {
        // Do Something
    }

}
```

loop

This loop will be executed redundantly on each ***gang***

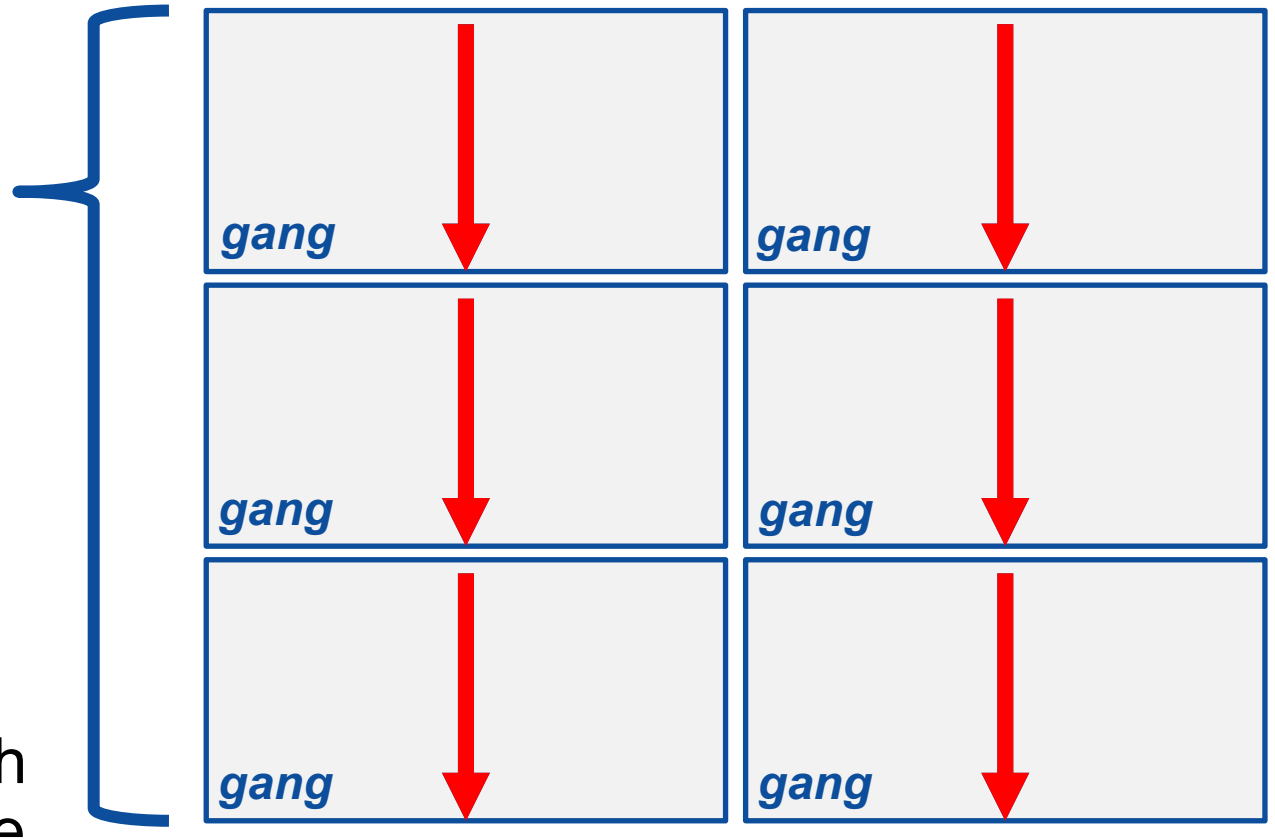


OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{

    for(int i = 0; i < N; i++)
    {
        // Do Something
    }

}
```

This means that each ***gang*** will execute the entire loop



OPENACC PARALLEL DIRECTIVE

Parallelizing a single loop

C/C++

```
#pragma acc parallel
{
  #pragma acc loop
  for(int i = 0; j < N; i++)
    a[i] = 0;
}
```

Fortran

```
!$acc parallel
  !$acc loop
  do i = 1, N
    a(i) = 0
  end do
!$acc end parallel
```

- Use a **parallel** directive to mark a region of code where you want parallel execution to occur
- This parallel region is marked by curly braces in C/C++ or a start and end directive in Fortran
- The **loop** directive is used to instruct the compiler to parallelize the iterations of the next loop to run across the parallel gangs

OPENACC PARALLEL LOOP DIRECTIVE

Parallelizing a single loop

C/C++

```
#pragma acc parallel loop
for(int i = 0; j < N; i++)
  a[i] = 0;
```

Fortran

```
!$acc parallel loop
do i = 1, N
  a(i) = 0
end do
```

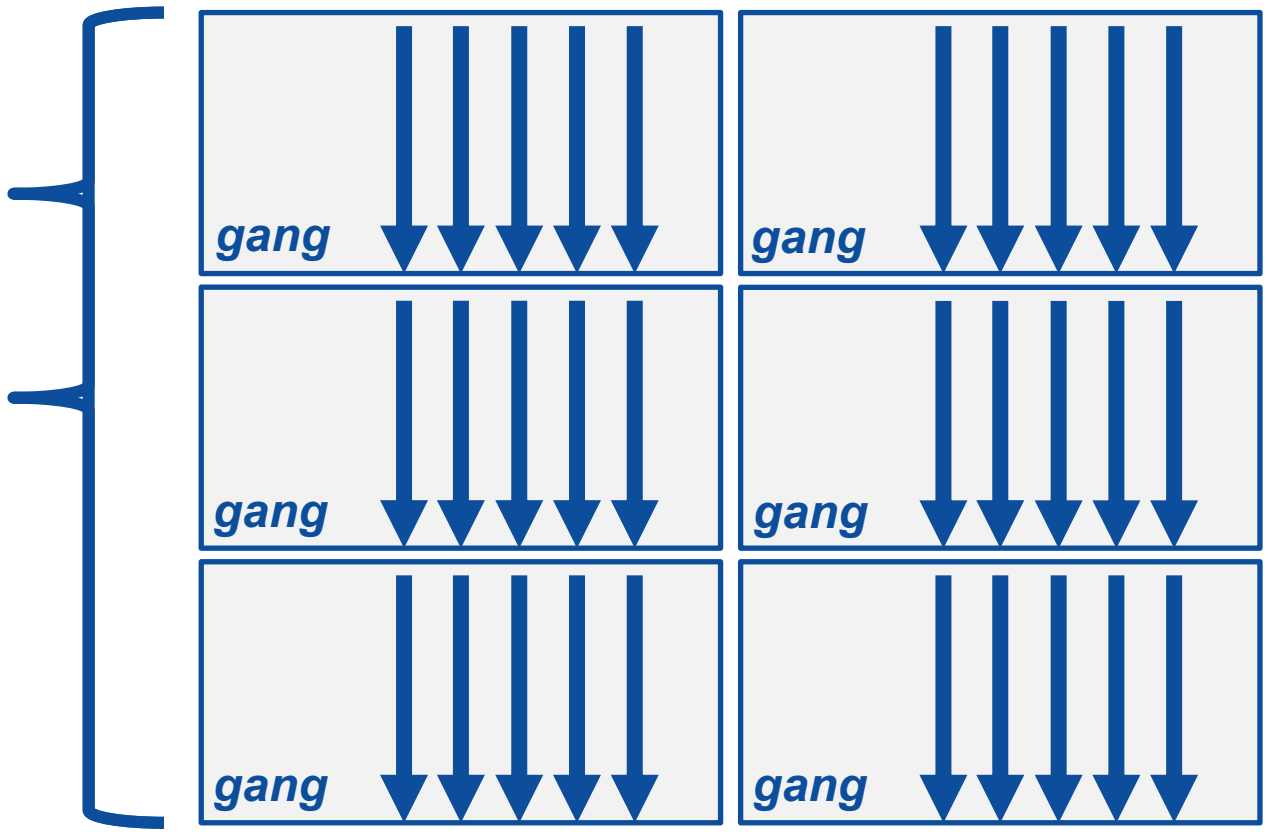
- This pattern is so common that you can do all of this in a single line of code
- In this example, the parallel loop directive applies to the next loop
- This directive both marks the region for parallel execution and distributes the iterations of the loop.
- When applied to a loop with a data dependency, parallel loop may produce incorrect results

OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{


    #pragma acc loop
    for(int i = 0; i < N; i++)
    {
        // Do Something
    }

}
```

The ***loop*** directive informs the compiler which loops to parallelize.



OPENACC PARALLEL LOOP DIRECTIVE

Parallelizing many loops

```
#pragma acc parallel loop
for(int i = 0; i < N; i++)
  a[i] = 0;

#pragma acc parallel loop
for(int j = 0; j < M; j++)
  b[j] = 0;
```

- To parallelize multiple loop nests, each should be accompanied by a parallel directive
- Each parallel loop nest can have different loop boundaries and loop optimizations
- Each parallel loop nest can be parallelized in a different way
- This is the recommended way to parallelize multiple loop nests. Attempting to parallelize multiple loop nests within the same parallel region may give performance issues or unexpected results.

PARALLELIZE WITH OPENACC PARALLEL LOOP

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

#pragma acc parallel loop reduction(max:err)
  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 parallel loop
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

Parallelize first loop nest, max *reduction* required.

Parallelize second loop.

We didn't detail *how* to parallelize the loops, just *which* loops to parallelize.

OpenACC More Science, Less Programming NVIDIA

REDUCTION CLAUSE

- The **reduction** clause takes many values and "reduces" them to a single value, such as in a sum or maximum
- Each partial result is calculated in parallel
- A **single result** is created by combining the partial results using the specified operation

```
for( i = 0; i < size; i++ )
  for( j = 0; j < size; j++ )
    for( k = 0; k < size; k++ )
      c[i][j] += a[i][k] * b[k][j];
```

```
for( i = 0; i < size; i++ )
  for( j = 0; j < size; j++ )
    double tmp = 0.0f;
    #pragma acc parallel loop \
      reduction(+:tmp)
    for( k = 0; k < size; k++ )
      tmp += a[i][k] * b[k][j];
    c[i][j] = tmp;
```

REDUCTION CLAUSE OPERATORS

Operator	Description	Example
`+`	Addition/Summation	`reduction(+:sum)`
`*`	Multiplication/Product	`reduction(*:product)`
`max`	Maximum value	`reduction(max:maximum)`
`min`	Minimum value	`reduction(min:minimum)`
`&`	Bitwise and	`reduction(&:val)`
`|`	Bitwise or	`reduction(|:val)`
`&&`	Logical and	`reduction(&&:val)`
`||`	Logical or	`reduction(||:val)`

BUILD AND RUN THE 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
```

PGI COMPILER BASICS

-ta flag

- The -ta flag enables building OpenACC code for a “Target Accelerator” (TA)
- -ta=multicore – Build the code to run across threads on a multicore CPU
- -ta=tesla:managed – Build the code for an NVIDIA (Tesla) GPU and manage the data movement automatically (more next module)

```
$ pgcc –fast –Minfo=accel –ta=tesla:managed main.c
$ pgc++ -fast -Minfo=accel –ta=tesla:managed main.cpp
$ pgfortran –fast –Minfo=accel –ta=tesla:managed main.f90
```

PGI COMPILER BASICS

-Mcuda flag

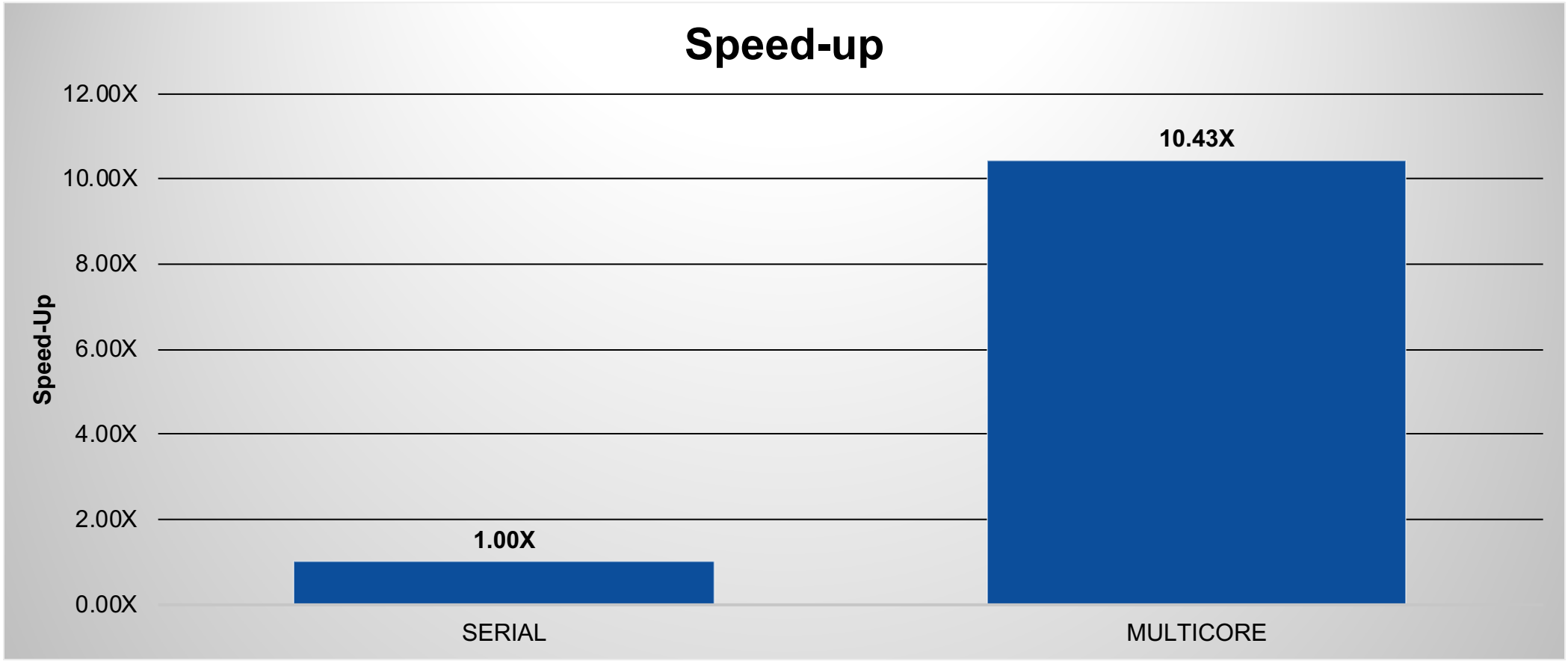
- The -Mcuda flag is needed when using NVTX regions in our code
- -lnvToolsExt – link the NVTX API
- This allows us to use NVTX regions in our code for both CPU and GPU profiling

```
$ pgcc -fast -Minfo=accel -ta=tesla:managed -Mcuda -lnvToolsExt main.c
$ pgc++ -fast -Minfo=accel -ta=tesla:managed -Mcuda -lnvToolsExt main.cpp
$ pgfortran -fast -Minfo=accel -ta=tesla:managed -Mcuda -lnvToolsExt main.f90
```

BUILDING THE CODE (MULTICORE)

```
$ pgcc -fast -ta=multicore -Minfo=accel -Mcuda -lnvToolsExt laplace2d_uvm.c
main:
     63, Generating Multicore code
         64, #pragma acc loop gang
     64, Accelerator restriction: size of the GPU copy of Anew,A is unknown
         Generating reduction(max:error)
     66, Loop is parallelizable
     74, Generating Multicore code
         75, #pragma acc loop gang
     75, Accelerator restriction: size of the GPU copy of Anew,A is unknown
     77, Loop is parallelizable
```

OPENACC SPEED-UP

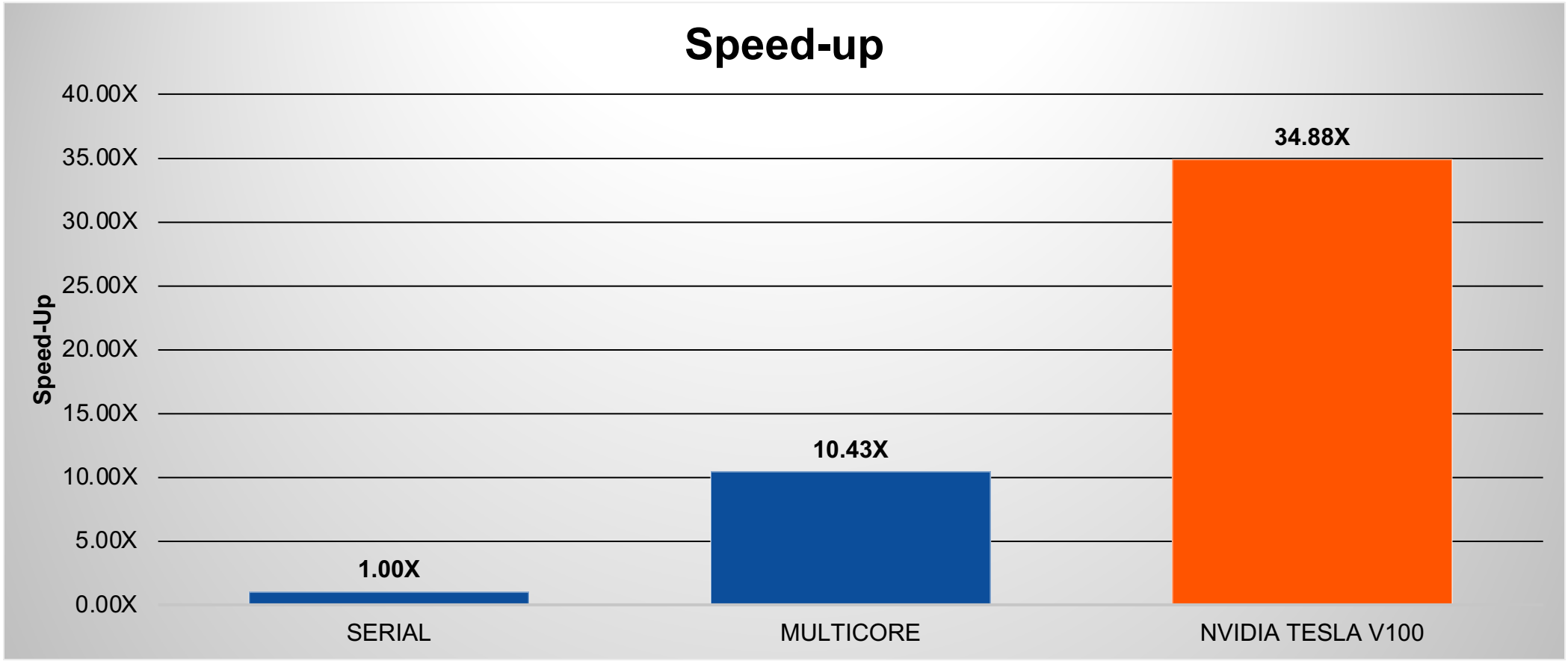


PGI 19.10, NVIDIA Tesla V100, IBM POWER9 22-core CPU @ 3.07GHz

BUILDING THE CODE (GPU)

```
$ pgcc -fast -ta=tesla:managed -Minfo=accel -Mcuda -lnvToolsExt laplace2d_uvm.c
main:
     63, Accelerator kernel generated
         Generating Tesla code
         64, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         66, #pragma acc loop vector(128) /* threadIdx.x */
     63, Generating implicit copyin(A[:])
         Generating implicit copyout(Anew[:])
         Generating implicit copy(error)
     66, Loop is parallelizable
     74, Accelerator kernel generated
         Generating Tesla code
         75, #pragma acc loop gang /* blockIdx.x */
         77, #pragma acc loop vector(128) /* threadIdx.x */
     74, Generating implicit copyin(Anew[:])
         Generating implicit copyout(A[:])
     77, Loop is parallelizable
```

OPENACC SPEED-UP



OpenACC More Science, Less Programming

NVIDIA

PGI 19.10, NVIDIA Tesla V100, IBM POWER9 22-core CPU @ 3.07GHz

CLOSING REMARKS

KEY CONCEPTS

This module we discussed…

- What is OpenACC
- How profile-driven programming helps you write better code
- How to parallelize loops using OpenACC's **parallel loop** directive to improve time to solution

Next module:

- Managing your data with OpenACC

OPENACC RESOURCES

Guides • Talks • Tutorials • Videos • Books • Spec • Code Samples • Teaching Materials • Events • Success Stories • Courses • Slack • Stack Overflow

FREE Compilers

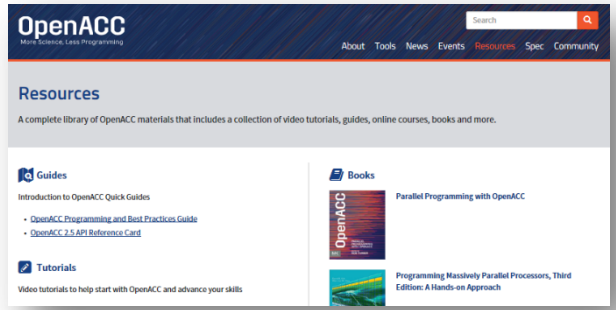
PGI Community Edition

slack

https://www.openacc.org/community#slack

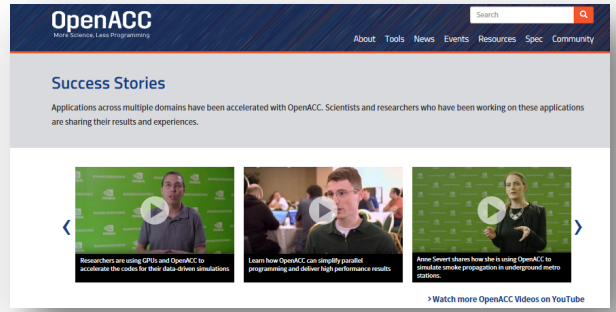
Resources

https://www.openacc.org/resources



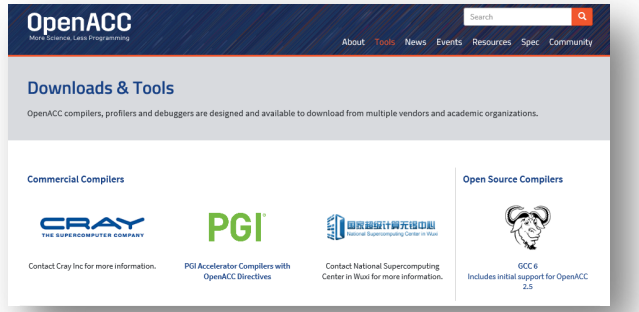
Success Stories

https://www.openacc.org/success-stories



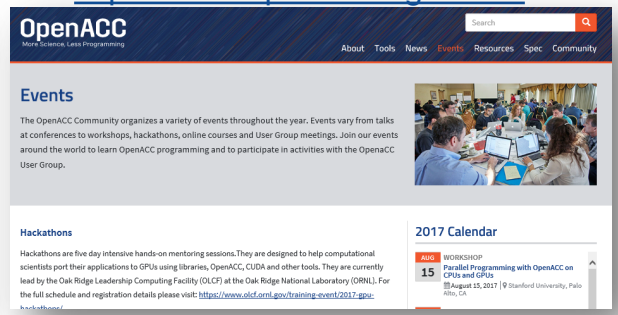
Compilers and Tools

https://www.openacc.org/tools



Events

https://www.openacc.org/events



OpenACC More Science, Less Programming NVIDIA