INTRODUCTION TO OPENACC

João Paulo Navarro, Solutions Architect

OpenACC
More Science, Less Programming

LECTURE OUTLINE

Topics to be Covered

- What is OpenACC and Why Should You Care?
- Profile-driven Development
- First Steps with OpenACC
- Data Movement
- Hackathon

INTRODUCTION TO OPENACC

3 WAYS TO ACCELERATE APPLICATIONS

Applications

Libraries	Compiler Directives	Programming Languages
Easy to use Most Performance	Easy to use Portable code **OpenACC**	Most Performance Most Flexibility

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

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

OPENACC

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 reason to learn low-level details of the hardware.

OPENACC

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

Supported Platforms

POWER

Sunway

x86 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.

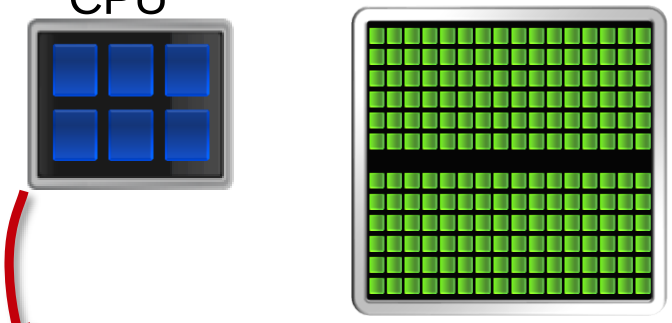
```
int main(){

...

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

}
```

OpenACC More Science, Less Programming NVIDIA aws Linux Academy

OPENACC

CPU

Parallel Hardware



```
int main(){

 <sequential code>

 #pragma acc kernels
 {
  <parallel code>
 }

}
```

Compiler Hint

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 reason to learn low-level details of the hardware.

OPENACC

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 reason to learn low-level details of the hardware.

DIRECTIVE-BASED HPC PROGRAMMING

Who’s Using OpenACC

3 OF TOP 5 HPC APPS

Intersect360 RESEARCH

5 OF 13 CAAR CODES

summit

2 OF LAST 9 FINALISTS

acm AWARDS Gordon Bell Prize

450 DOMAIN EXPERTS

OpenACC HACKATHON Advancing High Performance Computing

ACCELERATED APPS

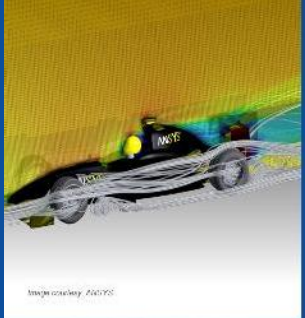
100,000 DOWNLOADS

PGI® Community EDITION

GAUSSIAN 16

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

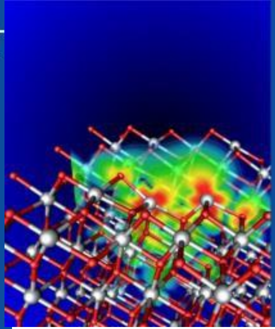
"Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/network and GPU parallelism. PGI's compilers were essential to the success of our efforts."



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."

E3SM

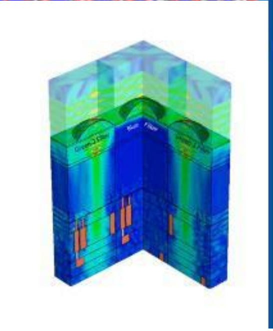
Mark A. Taylor
Multiphysics Applications
Sandia

"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

David 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

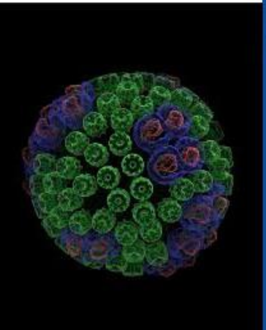
Dr. Lutz Schneider
Senior R&D Engineer
Synopsys Inc.

"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."

MPAS-A

Richard Loft
Director, Technology Development
NCAR

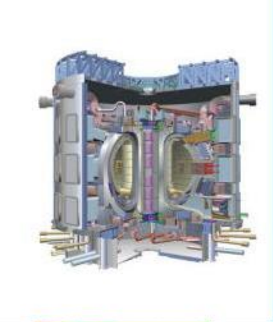
"Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer."



VMD

John Stone
Senior Research Programmer
Beckman 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

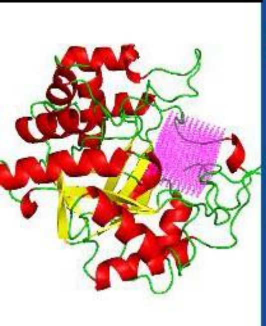
"Using OpenACC our scientists were able to achieve the acceleration needed for integrated fusion simulation with a minimum investment of time and effort in learning to program GPUs."

OpenACC
More Science, Less Programming

GAMERA

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

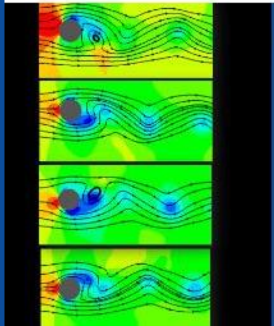
"With OpenACC and a compute node based on NVIDIA's Tesla P100 GPU, we achieved more than a 14X speed up over a K Computer node running our earthquake disaster simulation code"



SANJEEVINI

Abhilash Jayaraj
Project Scientist
Indian Institute of Technology
New Delhi

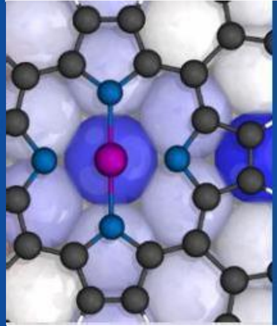
"In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task."



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

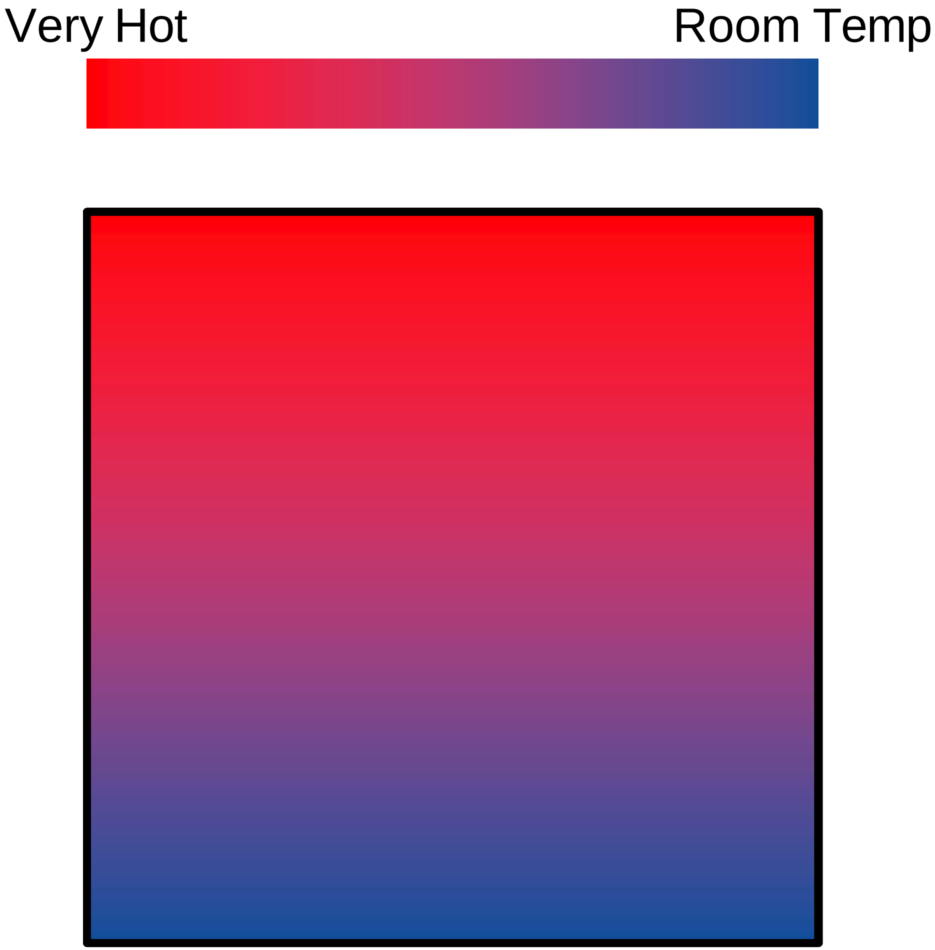
LAPLACE HEAT TRANSFER

Introduction to lab code - visual

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

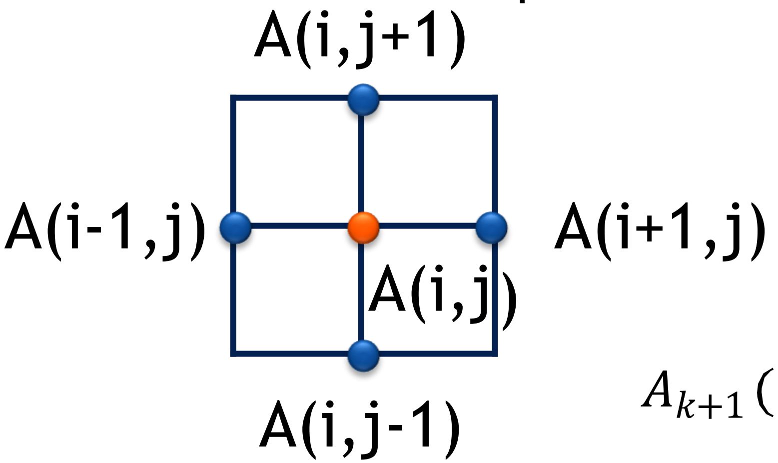
We will apply a consistent heat to the top of the plate.

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

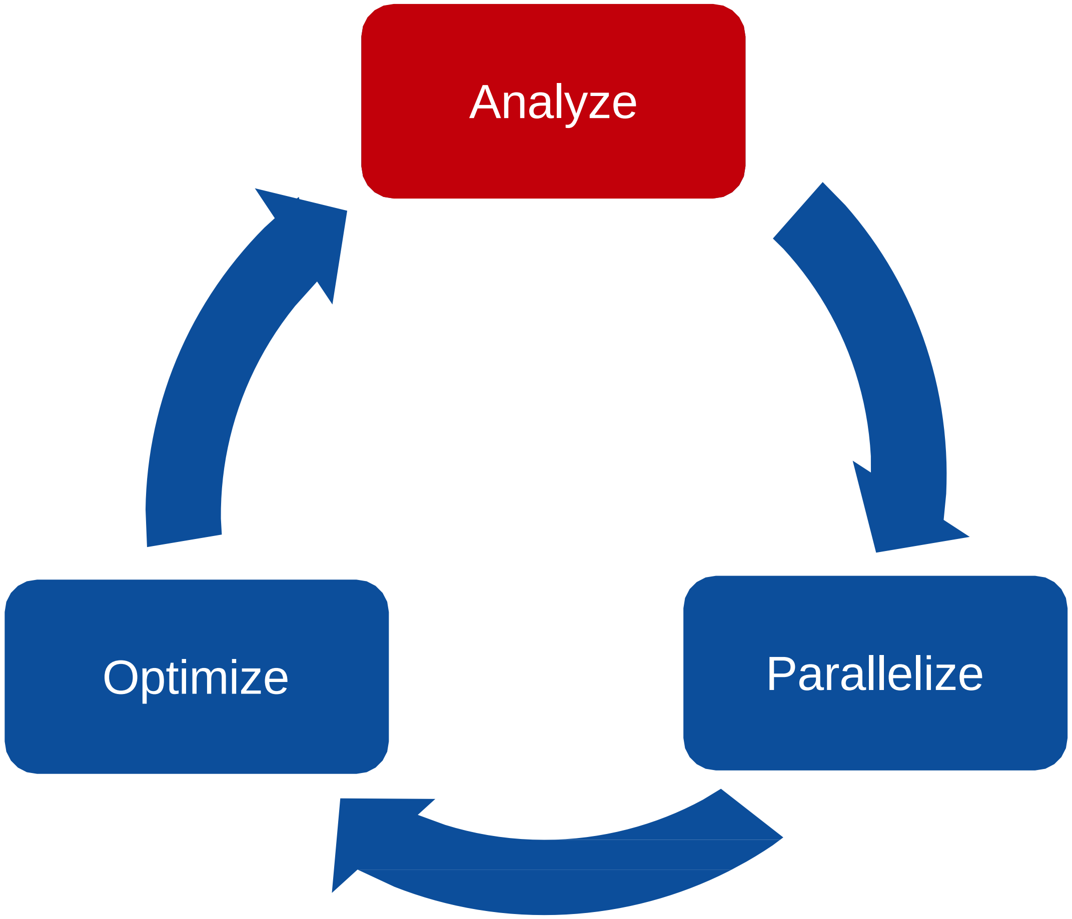
Compute max error for convergence

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.

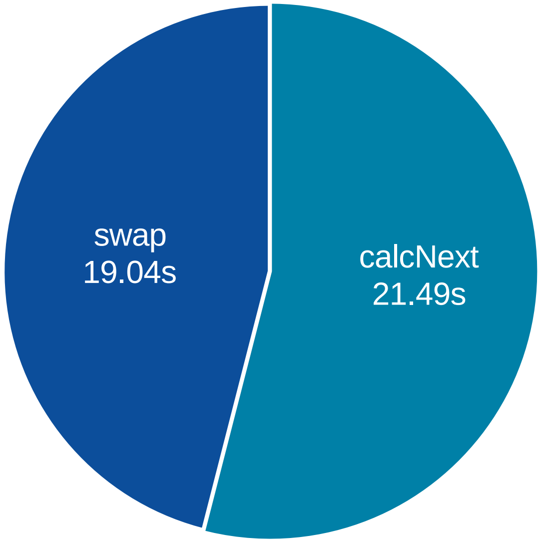
This can include information such as:

- 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

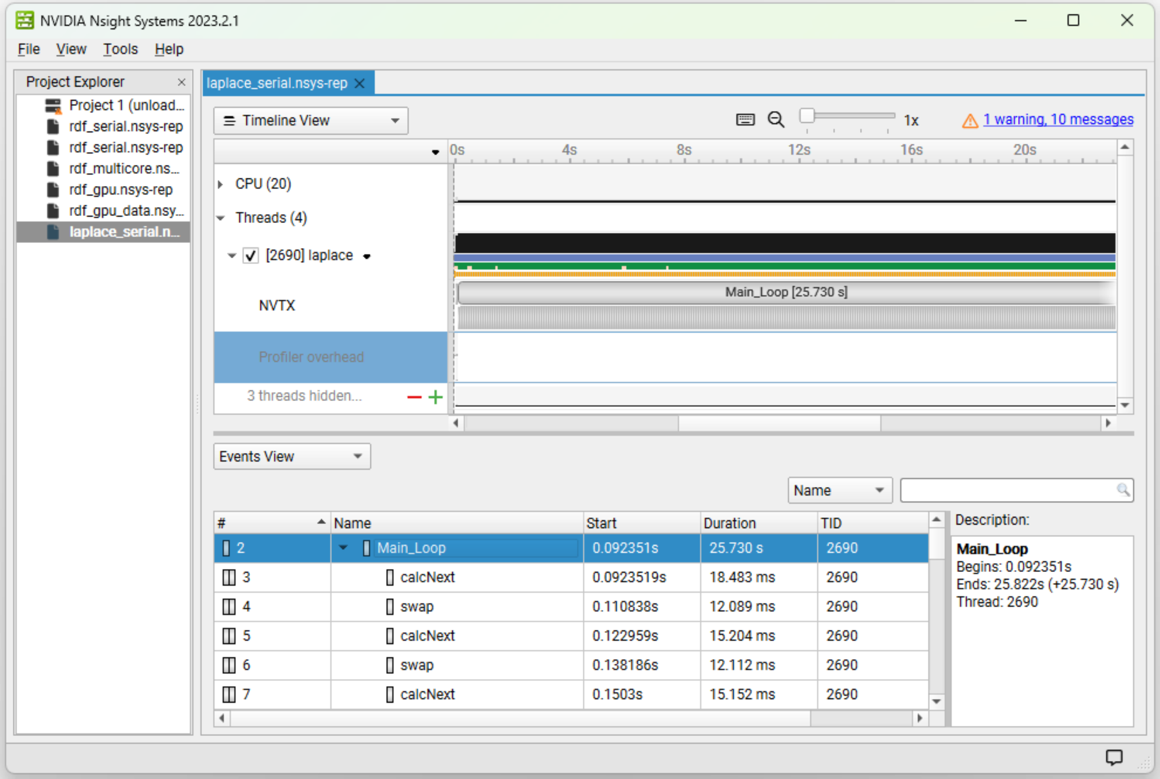
Total Runtime: 39.43 seconds



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 add NVTx markes in our code!

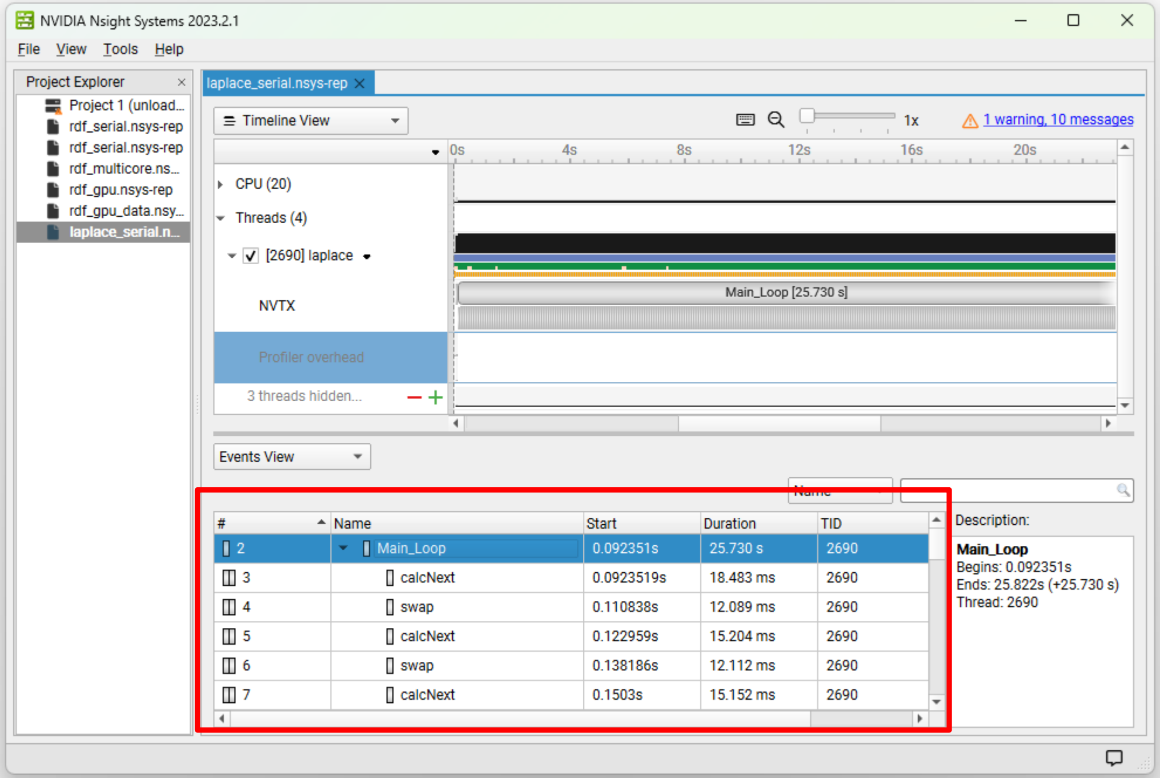


OpenACC More Science, Less Programming NVIDIA aws Linux Academy

PROFILING SEQUENTIAL CODE

CPU Details

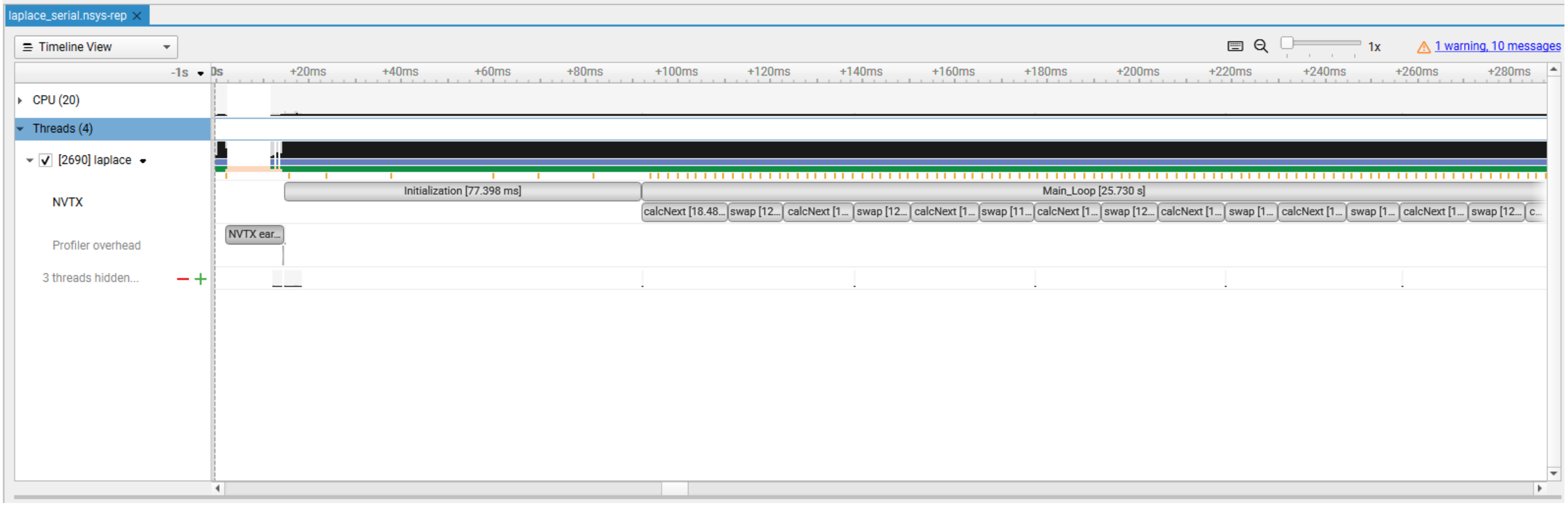
- On NVTX line we see the total time spent at each markup
- Double click in NVTX line, will open the event tab
- We will expand this information, and see more details about our code



PROFILING SEQUENTIAL CODE

CPU Details

- We can zoom in to get a close into the time spent in each NVTX marker.



OPENACC PARALLEL LOOP DIRECTIVE

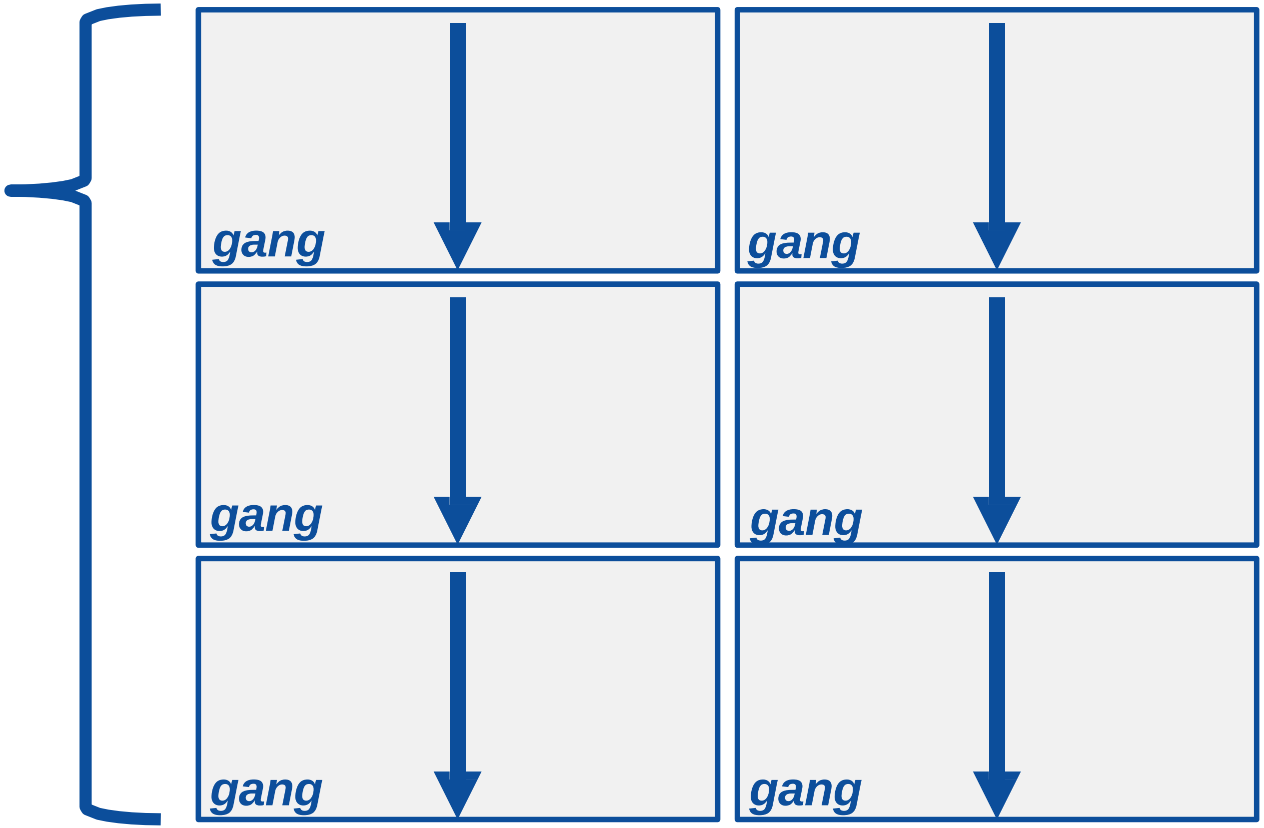
OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{
```

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

```
}
```



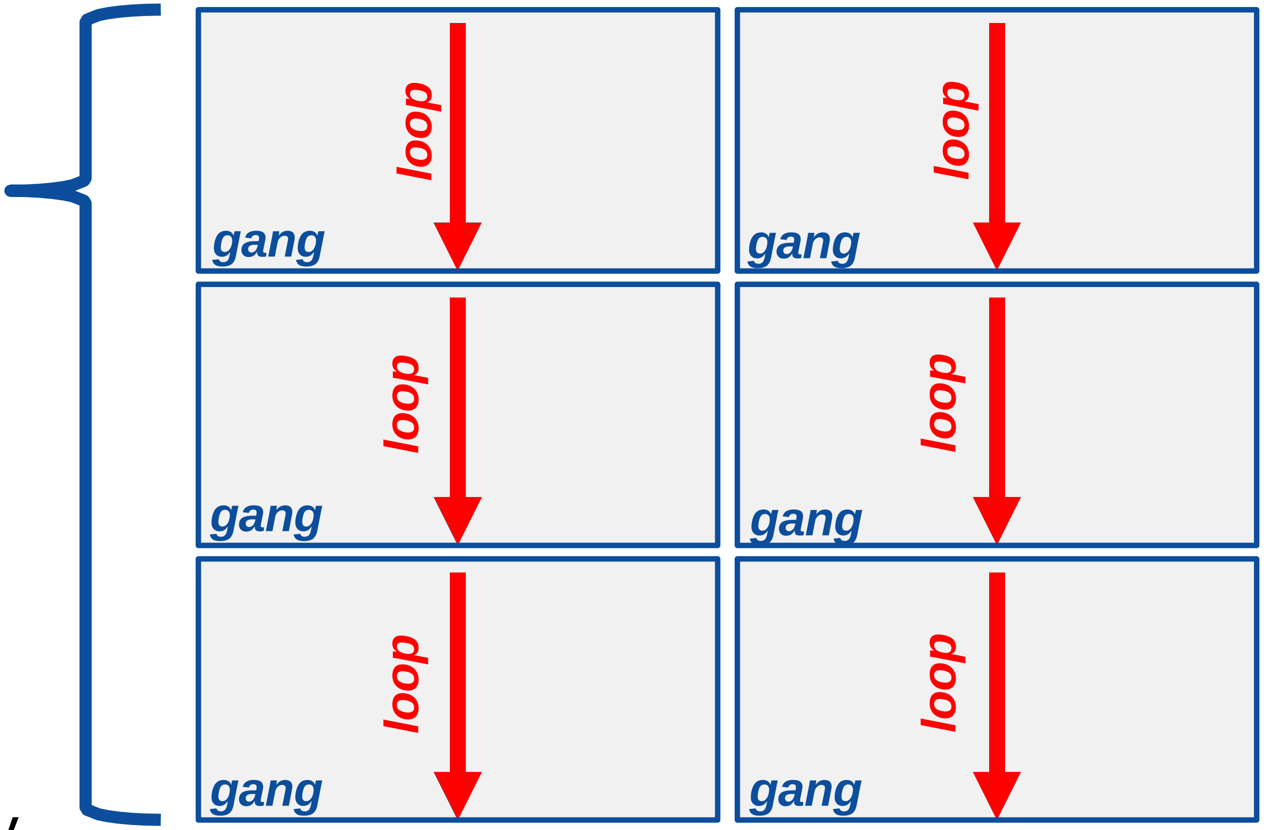
OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{

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

}
```

loop

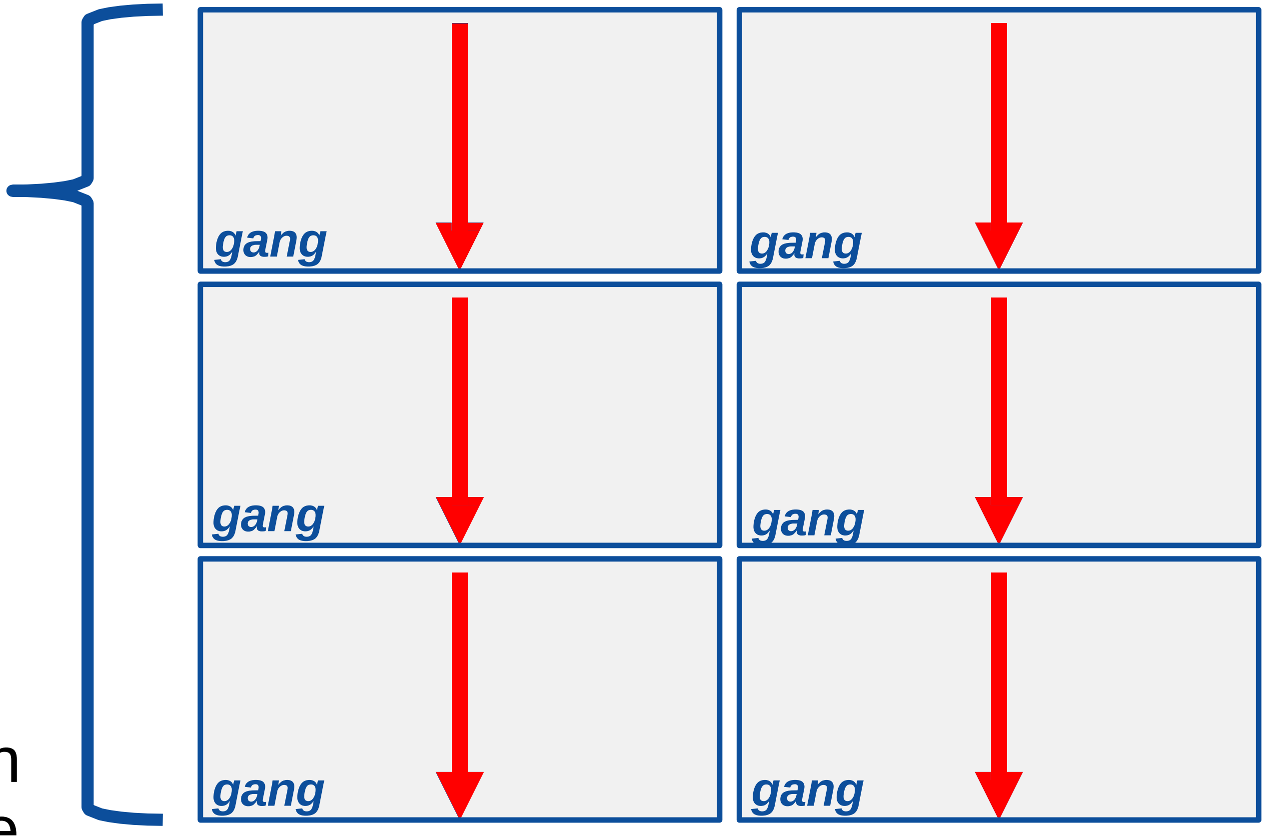
This loop will be executed redundantly on each gang



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 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
```

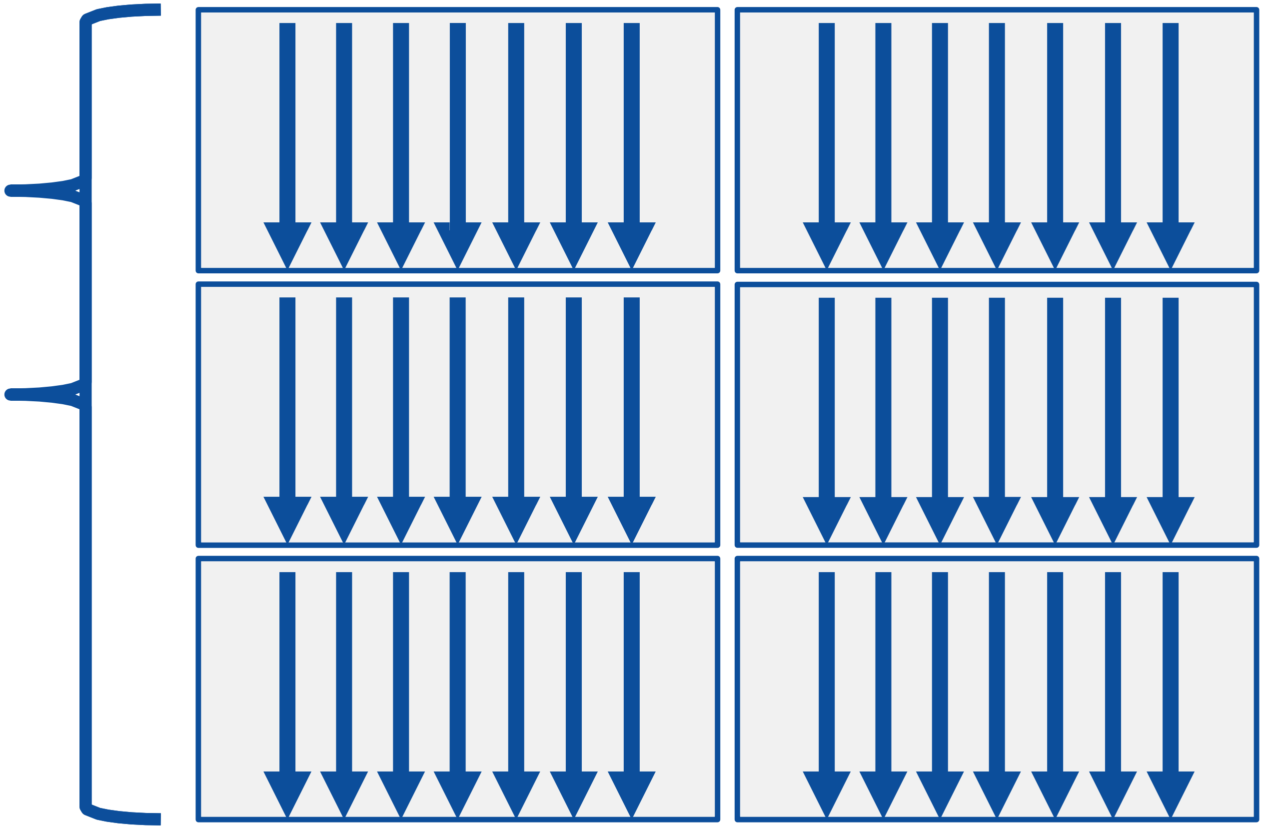
- This pattern is so common that you can do all of this in a single line of code
- In this example, the parallel loop directive applies to the next loop
- This directive both marks the region for parallel execution and distributes the iterations of the loop.
- When applied to a loop with a data dependency, 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 loops, each loop should be accompanied by a parallel directive
- Each parallel loop can have different loop boundaries and loop optimizations
- Each parallel loop can be parallelized in a different way
- This is the recommended way to parallelize multiple loops. Attempting to parallelize multiple loops 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.

REDUCTION CLAUSE

- The **reduction** clause takes many values and "reduces" them to a single value, such as in a sum or maximum
- Each thread calculates its part
- The compiler will perform a final reduction to produce a **single global result** 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 parallel acc 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

NVIDIA HPC SDK

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){ return y +
a*x; }
);
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}

#pragma omp target data map(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

PLATFORM SPECIALIZATION

CUDA

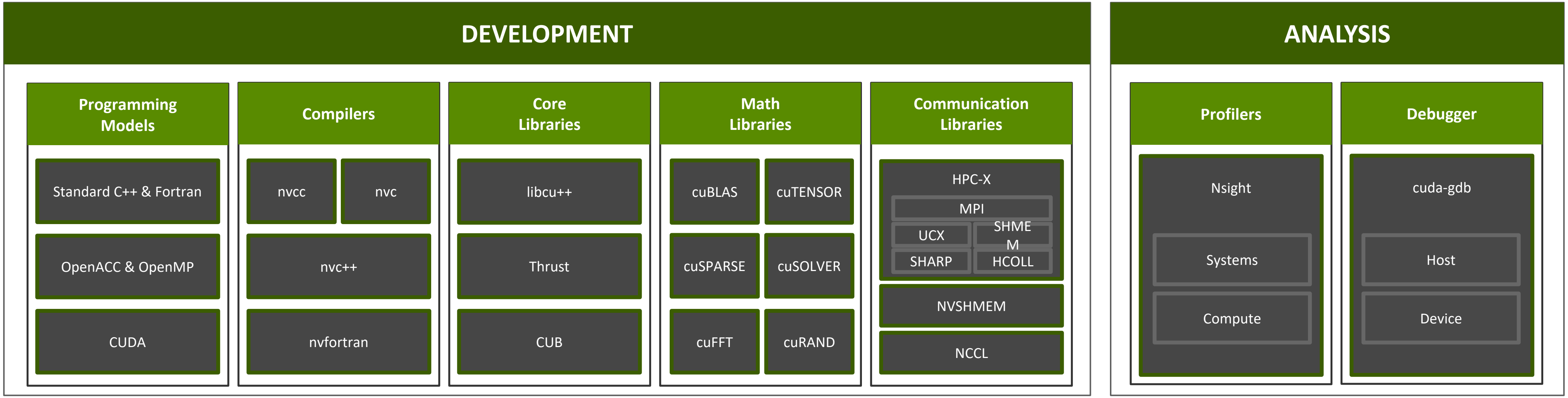
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

ACCELERATION LIBRARIES

Core	Math	Communication	Data Analytics	AI	Quantum

NVIDIA HPC SDK

Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud



Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

7-8 Releases Per Year | Freely Available

NVIDIA HPC SDK COMPILER BASICS

nvc, nvc++ and nvfortran

- The command to compile C code is ‘nvc’
- The command to compile C++ code is ‘nvc++’
- The command to compile Fortran code is ‘nvfortran’
- 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
```

NVIDIA HPC SDK 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

```
$ nvc -fast -Minfo=all main.c
$ nvc++ -fast -Minfo=all main.cpp
$ nvfortran -fast -Minfo=all main.f90
```

NVIDIA HPC SDK COMPILER BASICS

-acc flag

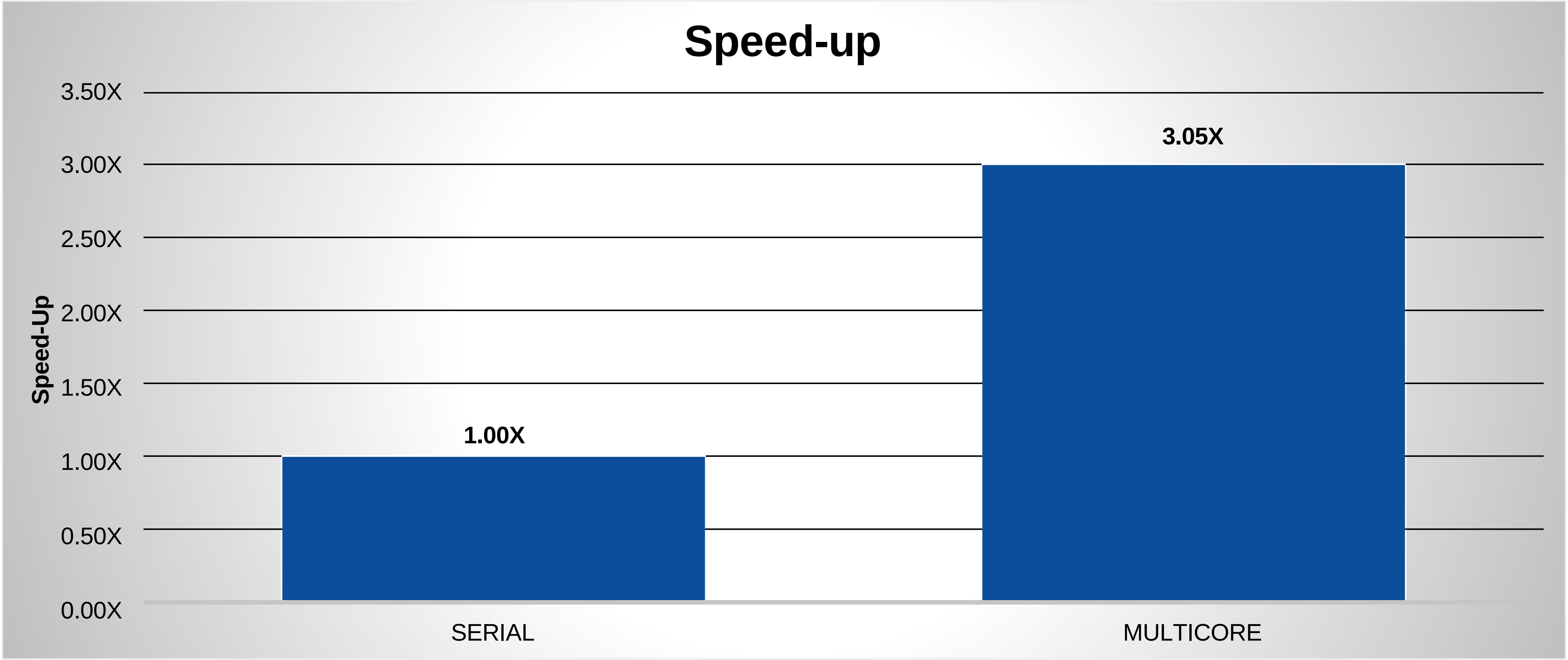
- The -acc flag enables building OpenACC code for an Accelerator (acc)
- -acc=multicore – Build the code to run across threads on a multicore CPU
- -acc=gpu -gpu:managed – Build the code for an NVIDIA GPU and manage the data movement for me (next lecture)

```
$ nvc -fast -Minfo=accel -acc=gpu -gpu:managed main.c
$ nvc++ -fast -Minfo=accel -acc=gpu -gpu:managed main.cpp
$ nvfortran -fast -Minfo=accel -acc=gpu -gpu:managed main.f90
```

BUILDING THE CODE (MULTICORE)

```
$ nvc -fast -acc=multicore -Minfo=accel laplace2d.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



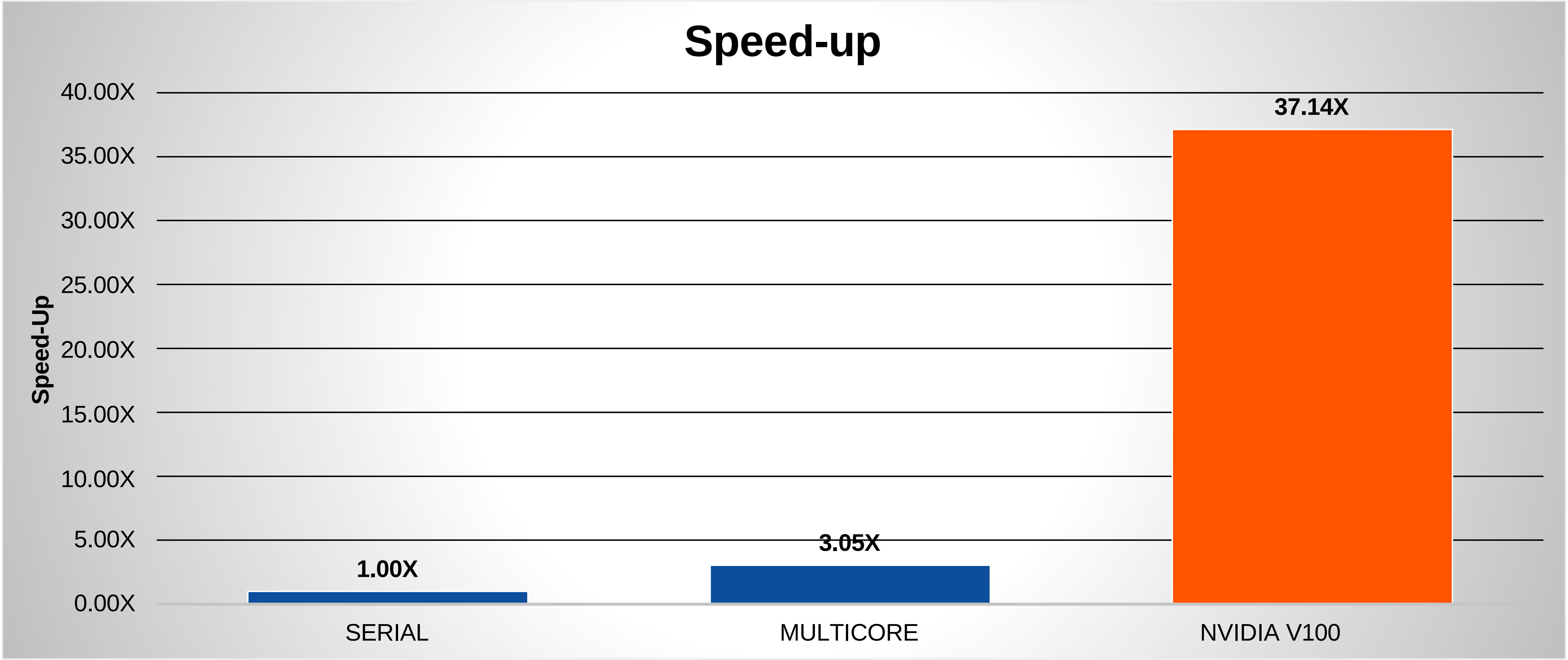
OpenACC More Science, Less Programming | NVIDIA | aws | Linux Academy

PGI 18.7, NVIDIA Tesla V100, Intel i9-7900X CPU @ 3.30GHz

BUILDING THE CODE (GPU)

```
$ nvc -fast -acc=gpu -gpu:managed -Minfo=accel laplace2d.c main:

     63, Accelerator kernel generated
         Generating NVIDIA GPU 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



PGI 18.7, NVIDIA V100, Intel i9-7900X CPU @ 3.30GHz

CLOSING REMARKS

KEY CONCEPTS

In this lecture 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 Lecture:

- Managing your data with OpenACC

OPENACC DATA MANAGEMENT

João Paulo Navarro, Solutions Architect

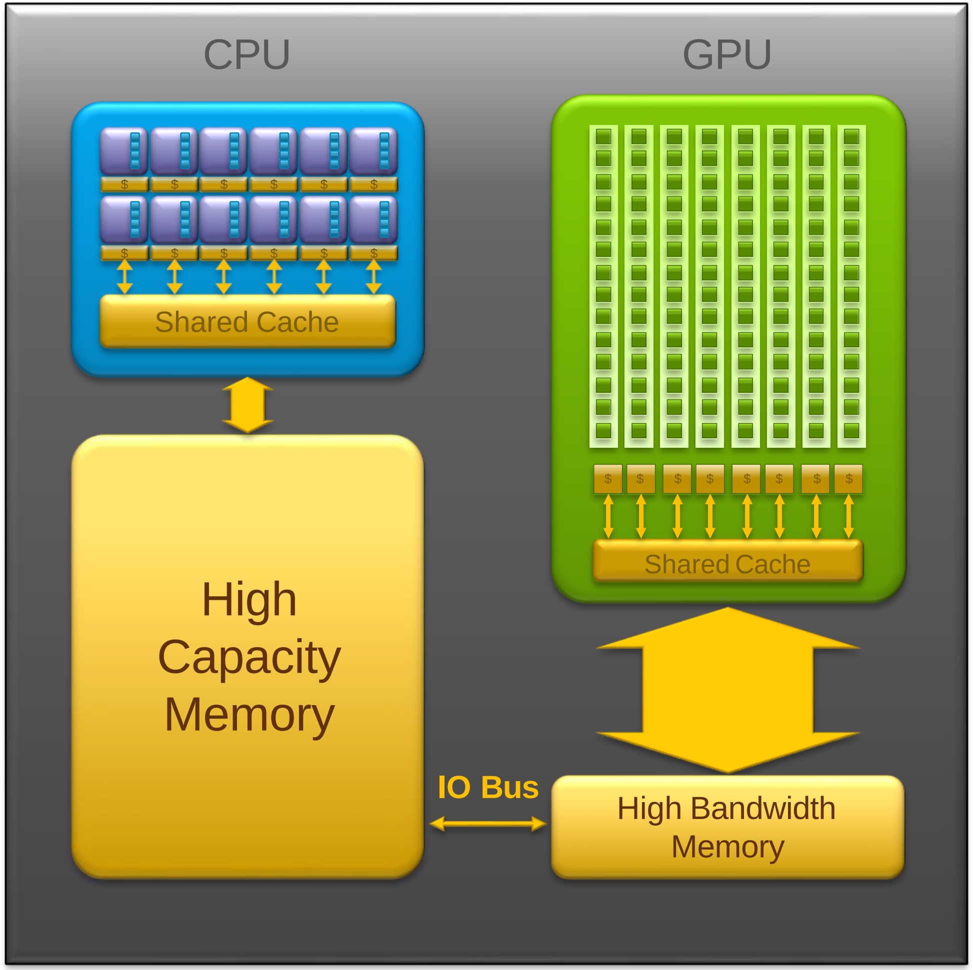
OpenACC
More Science, Less Programming

NVIDIA. aws Linux Academy

CPU AND GPU MEMORIES

CPU + GPU
Physical Diagram

- CPU memory is larger, GPU memory has more bandwidth
- CPU and GPU memory are usually separate, connected by an I/O bus (traditionally PCI-e)
- Any data transferred between the CPU and GPU will be handled by the I/O Bus
- The I/O Bus is relatively slow compared to memory bandwidth
- The GPU cannot perform computation until the data is within its memory

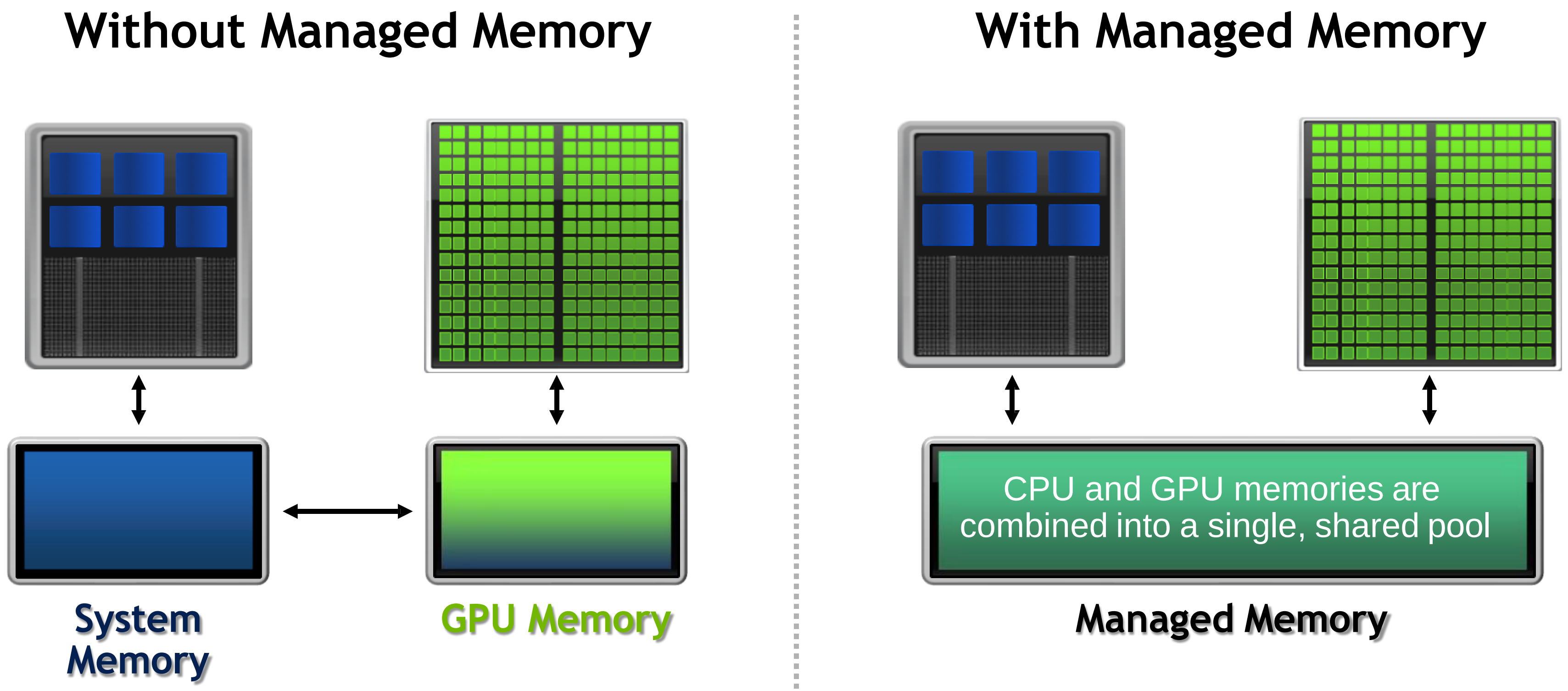


CUDA UNIFIED MEMORY

CUDA UNIFIED MEMORY

Simplified Developer Effort

Commonly referred to as *“managed memory.”*



CUDA MANAGED MEMORY

Usefulness

- Handling explicit data transfers between the host and device (CPU and GPU) can be difficult
- The PGI compiler can utilize CUDA Managed Memory to defer data management
- This allows the developer to concentrate on parallelism and think about data movement as an optimization

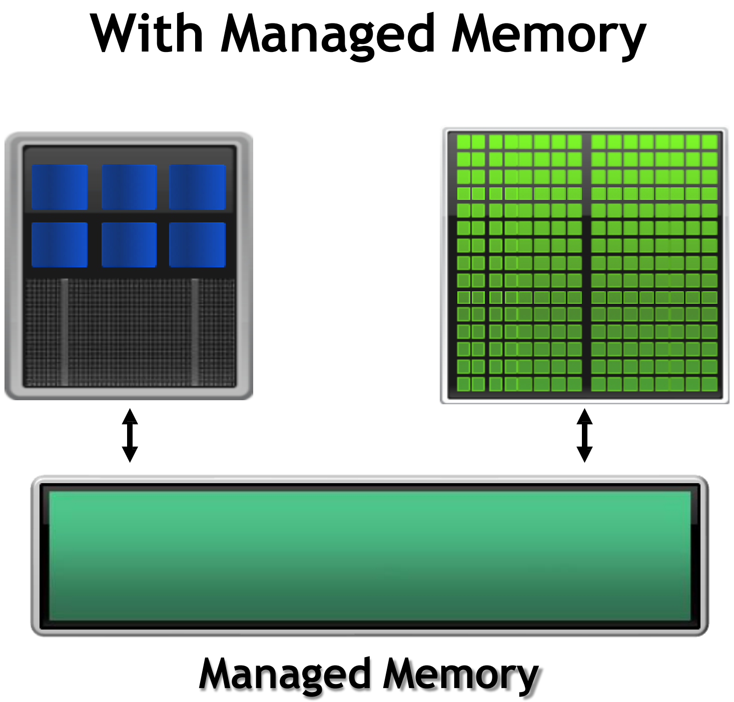
```
$ nvc –fast –acc=gpu -gpu:managed –Minfo=accel main.c
```

```
$ nvfortran –fast –acc=gpu -gpu:managed –Minfo=accel main.f90
```

MANAGED MEMORY

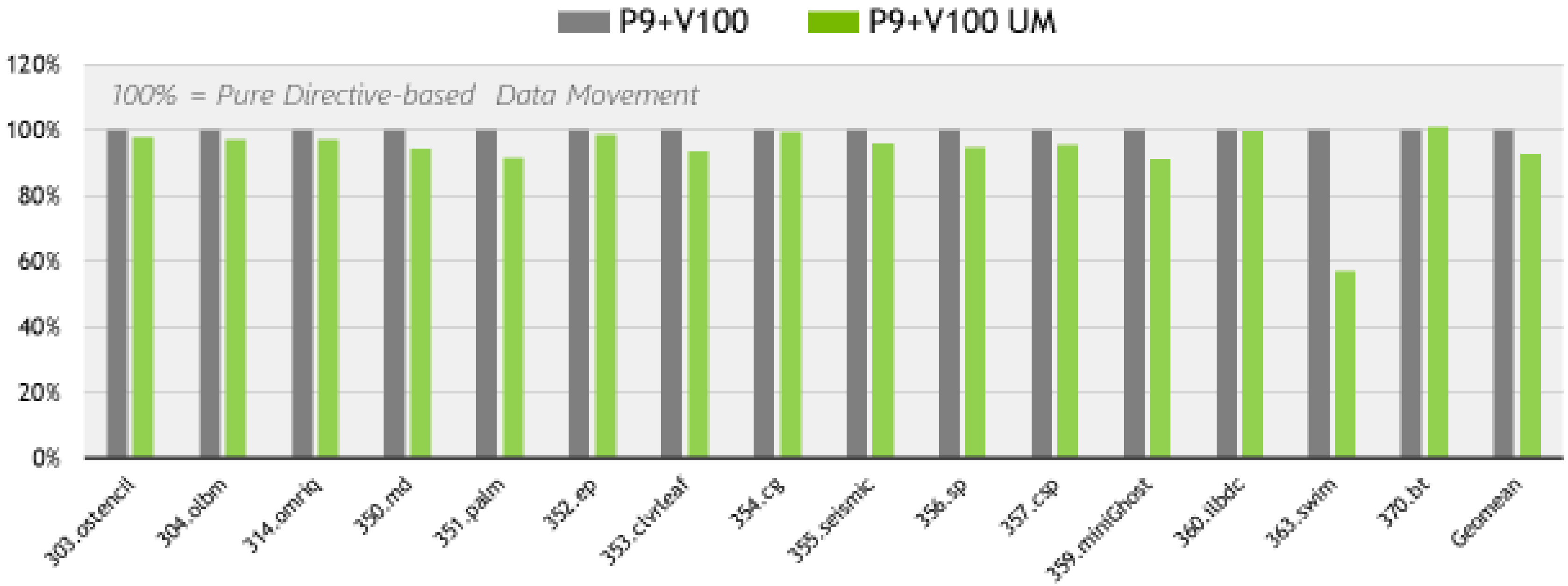
Limitations

- The programmer will almost always be able to get better performance by manually handling data transfers
- Memory allocation/deallocation takes longer with managed memory
- Cannot transfer data asynchronously
- Currently only available from PGI on NVIDIA GPUs.



SPEC ACCEL 1.2 OPENACC BENCHMARKS

OpenACC with Unified Memory vs OpenACC Data Directives



PGI 18.4 Compilers OpenACC SPEC ACCEL™ 1.2 performance measured June, 2018

SPEC® and the benchmark name SPEC ACCEL™ are registered trademarks of the Standard Performance Evaluation Corporation.

PGI

 NVIDIA

OpenACC More Science, Less Programming NVIDIA aws Linux Academy

* Slide Courtesy of PGI

LAST SESSION WE USED UNIFIED MEMORY

Now let's make our code run without.

Why?

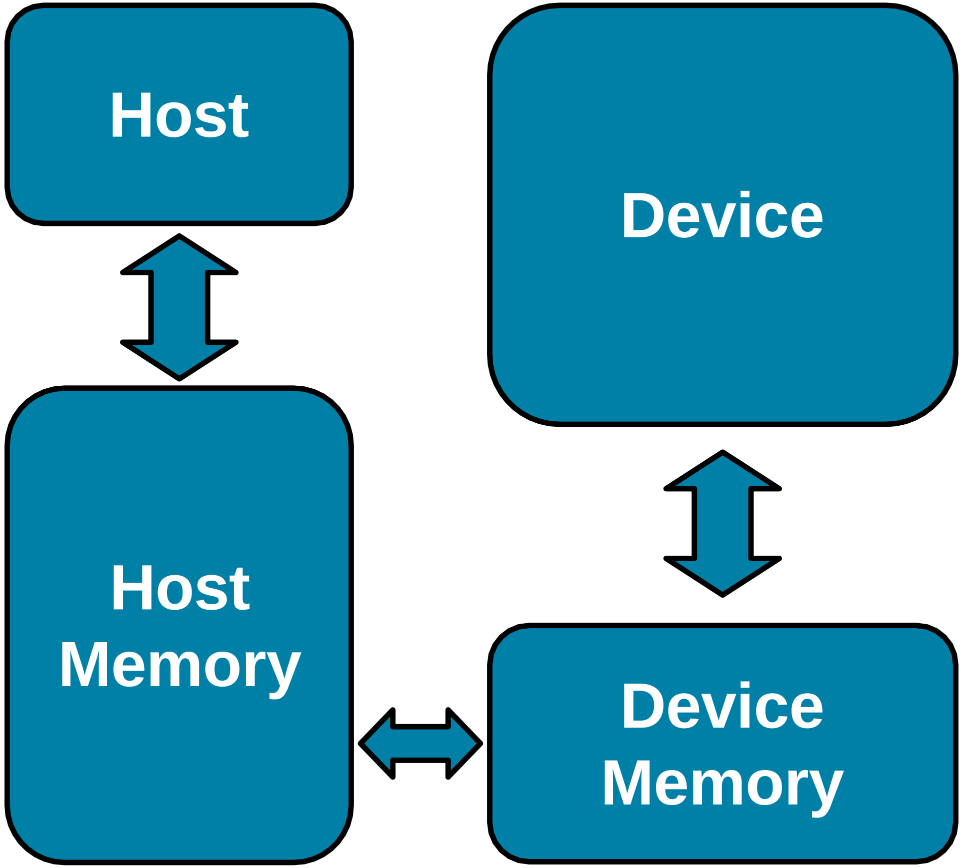
- Better data flow control and performance
- Currently the data always arrives "Just Too Late", let's do better

BASIC DATA MANAGEMENT

BASIC DATA MANAGEMENT

Between the host and device

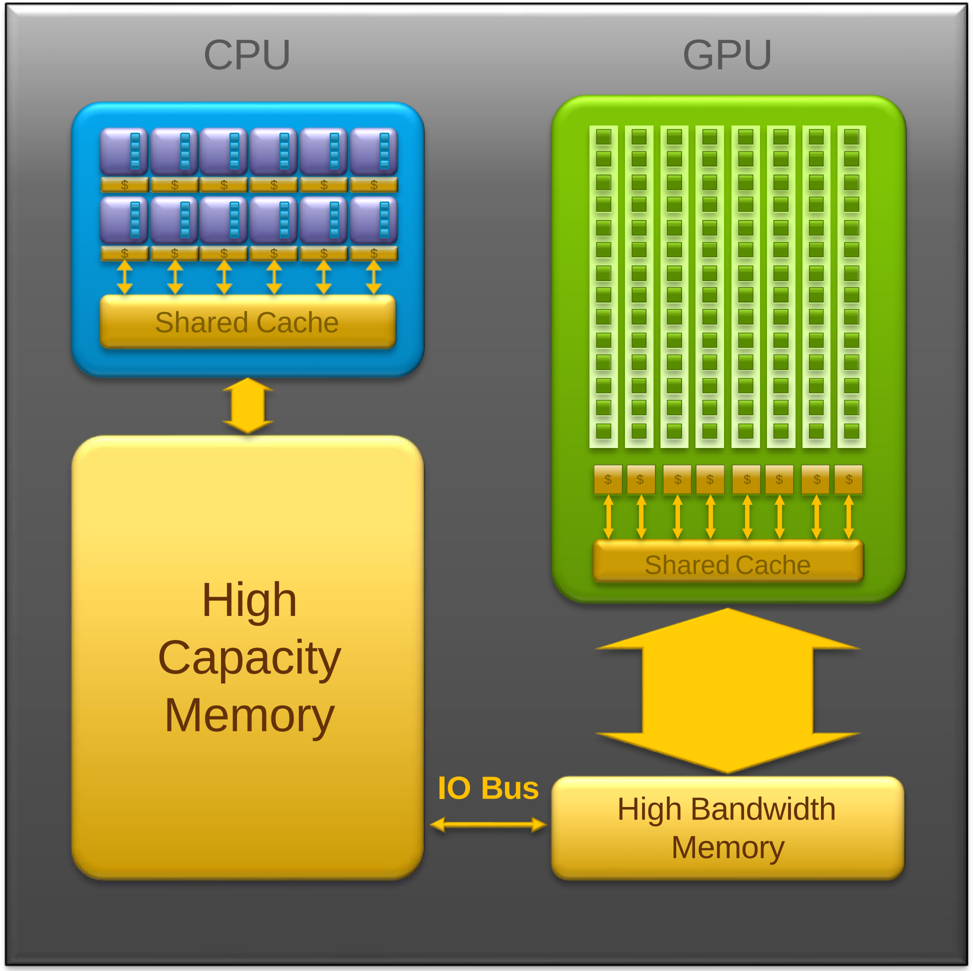
- The **host** is traditionally a CPU
- The **device** is some parallel accelerator
- When our target hardware is multicore, the host and device are the same, meaning that their memory is also the same
- There is no need to explicitly manage data when using a shared memory accelerator, such as the multicore target



BASIC DATA MANAGEMENT

Between the host and device

- When the target hardware is a GPU data will usually need to migrate between CPU and GPU memory
- Each array used on the GPU must be allocated on the GPU
- When data changes on the CPU or GPU the other must be updated



TRY TO BUILD WITHOUT "MANAGED"

Change the compiling line to remove "managed" part

```
nvc -acc=gpu -gpu:managed -Minfo=accel laplace2d.c jacobi.c
laplace2d.c:
PGC-S-0155-Compiler failed to translate accelerator region (see -Minfo
messages): Could not find allocated-variable index for symbol (laplace2d.c: 47)
calcNext:
     47, Accelerator kernel generated
         Generating Tesla code
         48, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         50, #pragma acc loop vector(128) /* threadIdx.x */
     48, Accelerator restriction: size of the GPU copy of Anew,A is unknown
     50, Loop is parallelizable
PGC-F-0704-Compilation aborted due to previous errors. (laplace2d.c)
PGC/x86-64 Linux 18.7-0: compilation aborted
jacobi.c:
```

DATA SHAPING

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.**

Principal use: For many important data structures in your code, this is a logical default to input, modify and return the data.

`copyin( list )` **Allocates memory on GPU and copies data from host to GPU when entering region.**

Principal use: Think of this like an array that you would use as just an input to a subroutine.

`copyout( list )` **Allocates memory on GPU and copies data to the host when exiting region.**

Principal use: A result that isn't overwriting the input data structure.

`create( list )` **Allocates memory on GPU but does not copy.**

Principal use: Temporary arrays.

ARRAY SHAPING

- Sometimes the compiler needs help understanding the *shape* of an array
- The first number is the start index of the array
- In C/C++, the second number is how much data is to be transferred
- In Fortran, the second number is the ending index

```
copy(array[starting_index:length])
```

C/C++

```
copy(array(starting_index:ending_index))
```

Fortran

ARRAY SHAPING (CONT.)

Multi-dimensional Array shaping

```
copy(array[0:N][0:M])
```

C/C++

Both of these examples copy a 2D array to the device

```
copy(array(1:N, 1:M))
```

Fortran

ARRAY SHAPING (CONT.)

Partial Arrays

```
copy(array[i*N/4:N/4])
```

C/C++

Both of these examples copy only ¼ of the full array

```
copy(array(i*N/4:i*N/4+N/4))
```

Fortran

OPTIMIZED DATA MOVEMENT

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

#pragma acc parallel loop reduction(max:err) copyin(A[0:n*m]) copy(Anew[0:n*m])
  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 copyin(Anew[0:n*m]) copyout(A[0:n*m])
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

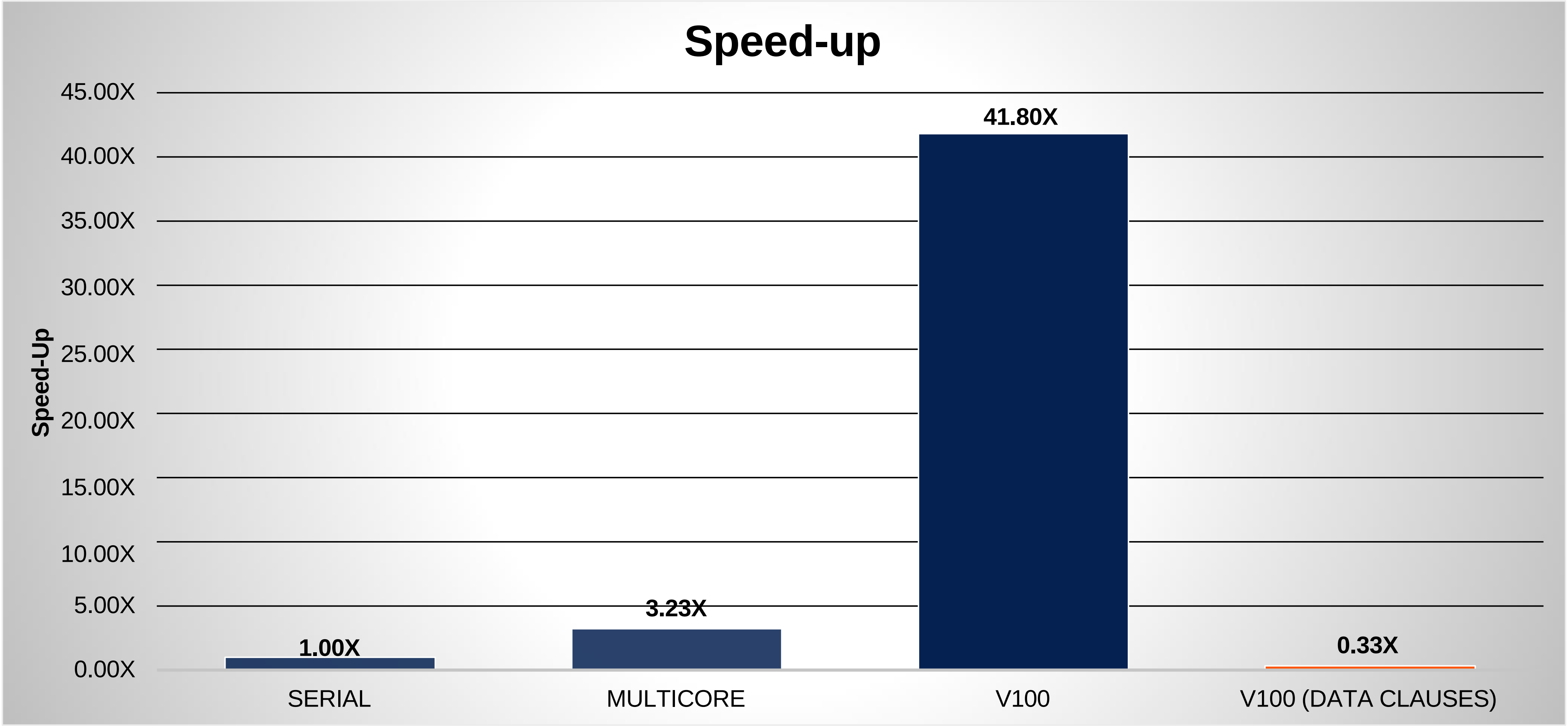
Data clauses provide necessary “shape” to the arrays.

TRY TO BUILD WITHOUT “MANAGED”

Change the compiling line to remove “managed” part

```
nvc -acc=gpu -Minfo=accel laplace2d.c jacobi.c
laplace2d.c:
calcNext:
     47, Generating copyin(A[:m*n]) Accelerator
         kernel generated Generating Tesla code
         48, #pragma acc loop gang /* blockIdx.x */
             Generating reduction(max:error)
         50, #pragma acc loop vector(128) /* threadIdx.x */
          47, Generating implicit copy(error)
              Generating copy(Anew[:m*n])
          50, Loop is parallelizable
     swap:
     62, Generating copyin(Anew[:m*n])
         Generating copyout(A[:m*n]) Accelerator
         kernel generated Generating Tesla code
         63, #pragma acc loop gang /* blockIdx.x */
         65, #pragma acc loop vector(128) /* threadIdx.x */ 65, Loop
     is parallelizable
jacobi.c:
```

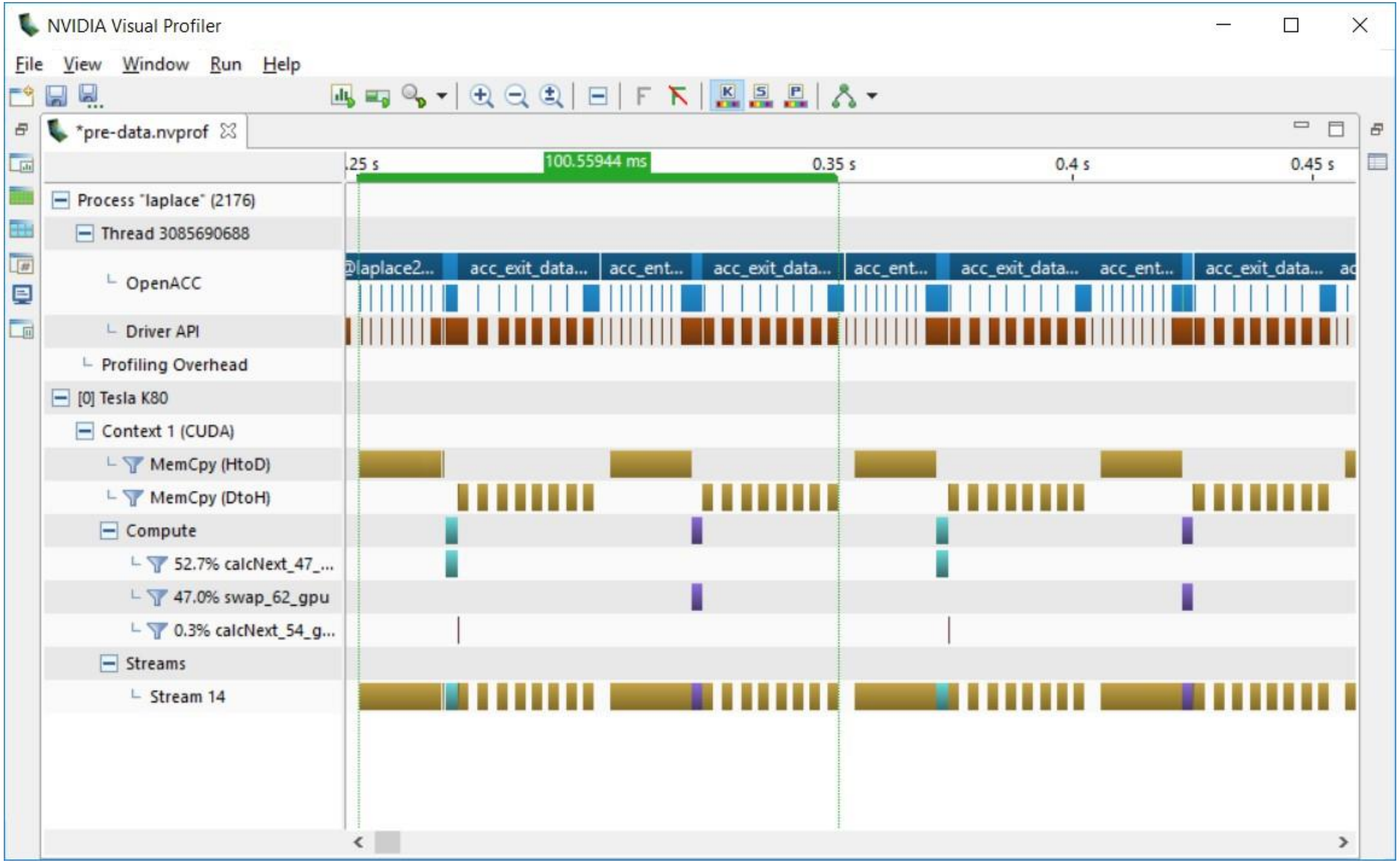
OPENACC ~~SPEED-UP~~ SLOWDOWN



WHAT WENT WRONG?

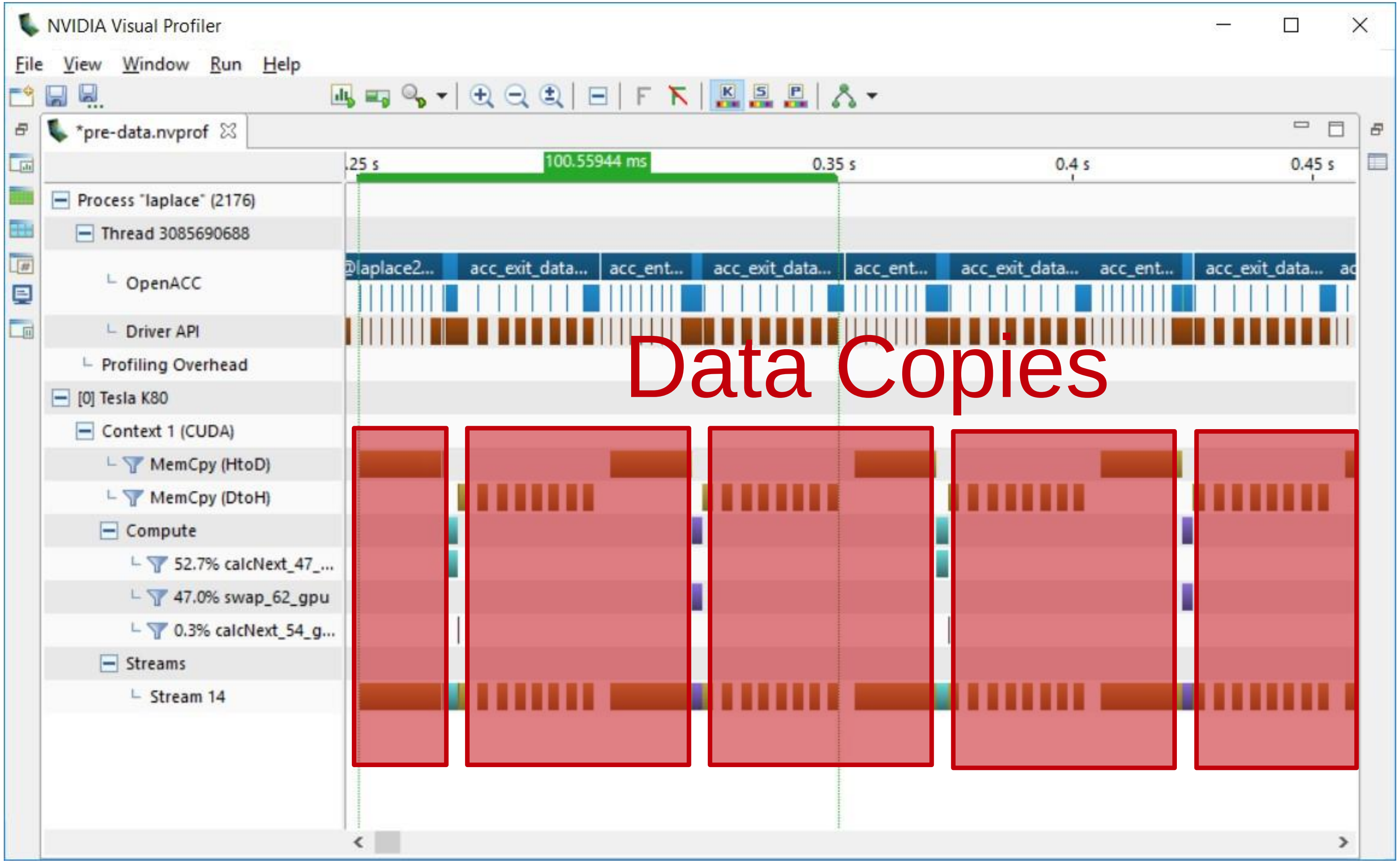
- The code now has all of the information necessary to build without managed memory, but it runs much slower.
- Profiling tools are here to help!

APPLICATION PROFILE (2 STEPS)



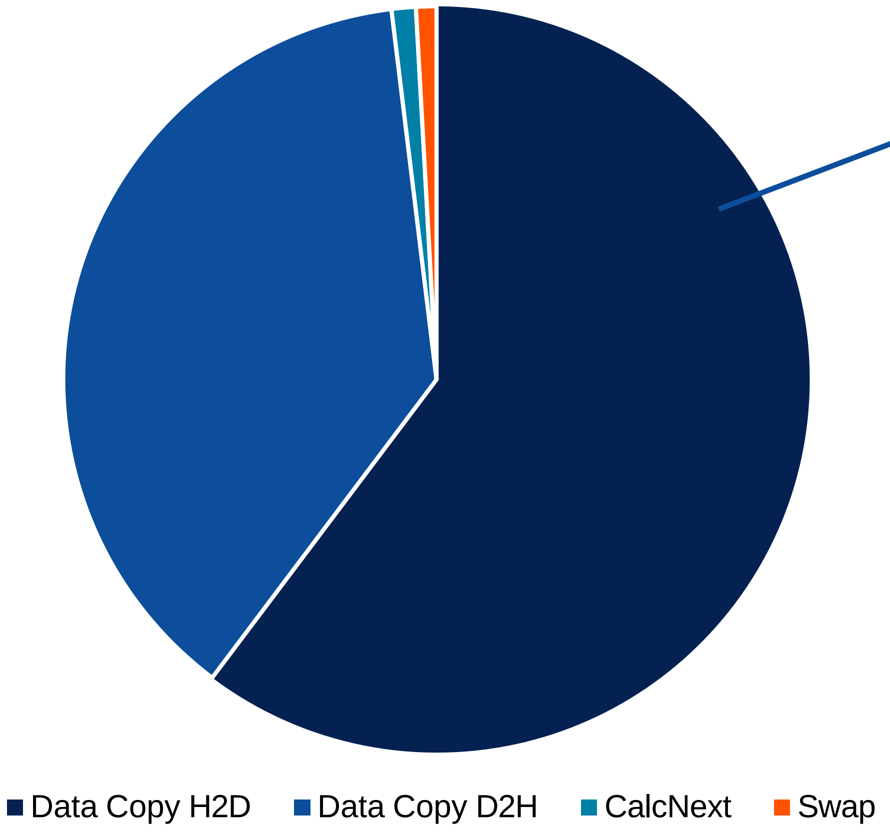
OpenACC
More Science, Less Programming

APPLICATION PROFILE (2 STEPS)



OpenACC
More Science, Less Programming

RUNTIME BREAKDOWN



Nearly all of our time is spent moving data to/from the GPU

OPTIMIZED DATA MOVEMENT

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

#pragma acc parallel loop reduction(max:err) copyin(A[0:n*m]) copy(Anew[0:n*m])
  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 copyin(Anew[0:n*m]) copyout(A[0:n*m])
  for( int j = 1; j < n-1; j++) {
    for( int i = 1; i < m-1; i++ ) {
      A[j][i] = Anew[j][i];
    }
  }
  iter++;
}
```

Currently we're copying to/from the GPU for each loop, can we reuse it?

OPTIMIZE DATA MOVEMENT

OPENACC DATA DIRECTIVE

Definition

- The data directive defines a lifetime for data on the device beyond individual loops
- During the region data is essentially “owned by” the accelerator
- Data clauses express shape and data movement for the region

```
#pragma acc data clauses
{

  < Sequential and/or Parallel code >

}
```

```
!$acc data clauses

  < Sequential and/or Parallel code >

!$acc end data
```

OPTIMIZED DATA MOVEMENT

```
#pragma acc data copy(A[:n*m]) copyin(Anew[:n*m])
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc parallel loop reduction(max:err) copyin(A[0:n*m])
  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 copyin(Anew[0:n*m]) copyout(A[0:n*m])
  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.

Copy initial condition of Anew, but not final value

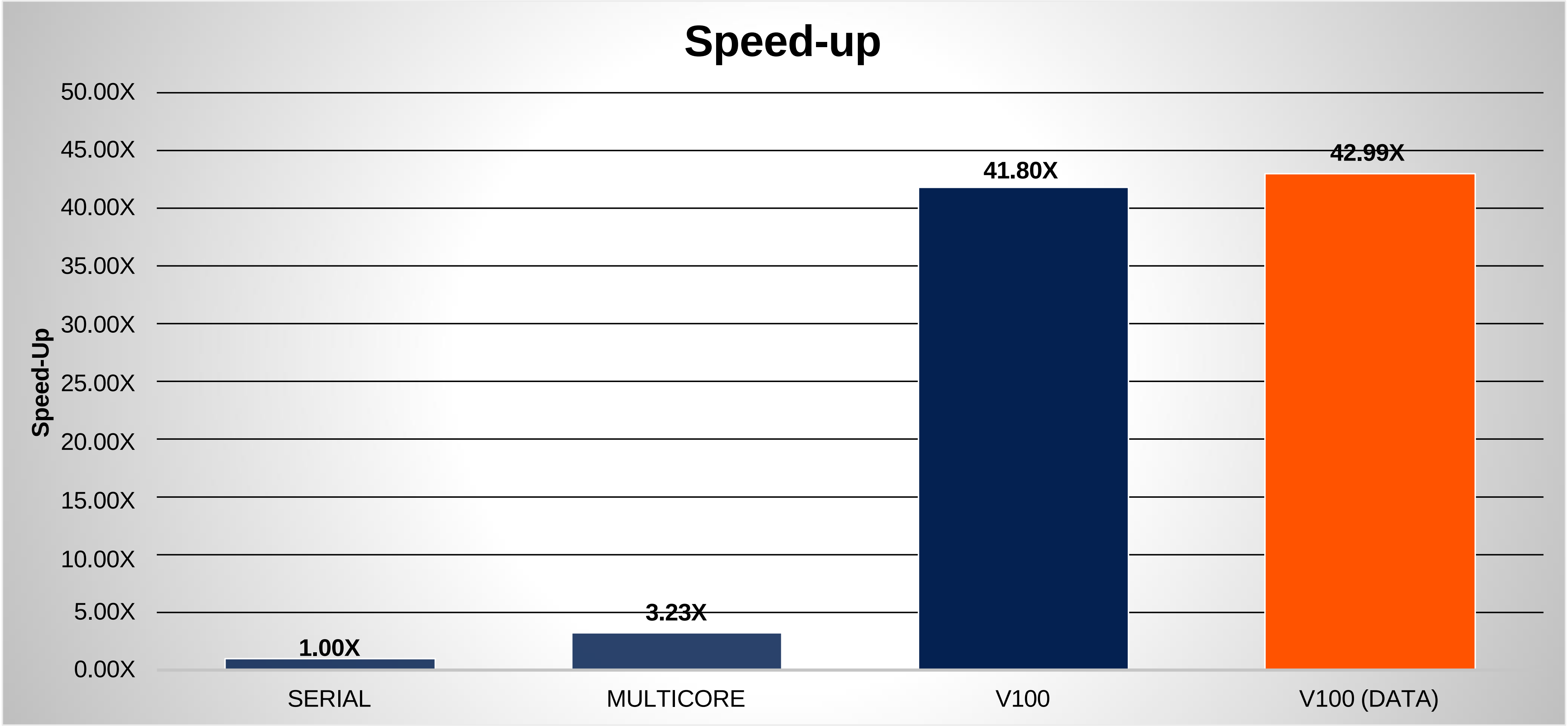
REBUILD THE CODE

```
pgcc -fast -ta=tesla -Minfo=accel laplace2d_uvm.c
main:
     60, Generating copy(A[:m*n])
         Generating copyin(Anew[:m*n])
     64, Accelerator kernel generated
         Generating Tesla code
         64, Generating reduction(max:error)
         65, #pragma acc loop gang /* blockIdx.x */
         67, #pragma acc loop vector(128) /* threadIdx.x */
     67, Loop is parallelizable
     75, Accelerator kernel generated
         Generating Tesla code
         76, #pragma acc loop gang /* blockIdx.x */
         78, #pragma acc loop vector(128) /* threadIdx.x */
     78, Loop is parallelizable
```

Now data movement only happens at our data region.

OpenACC More Science, Less Programming NVIDIA aws Linux Academy

OPENACC SPEED-UP



WHAT WE'VE LEARNED SO FAR

- CUDA Unified (Managed) Memory is a powerful porting tool
- GPU programming without managed memory often requires data shaping
- Moving data at each loop is often inefficient
- The OpenACC Data region can decouple data movement and computation

DATA SYNCHRONIZATION

OPENACC UPDATE DIRECTIVE

update: Explicitly transfers data between the host and the device

Useful when you want to synchronize data in the middle of a data region

Clauses:

self: makes host data agree with device data

device: makes device data agree with host data

```
#pragma acc update self(x[0:count])
#pragma acc update device(x[0:count])
```

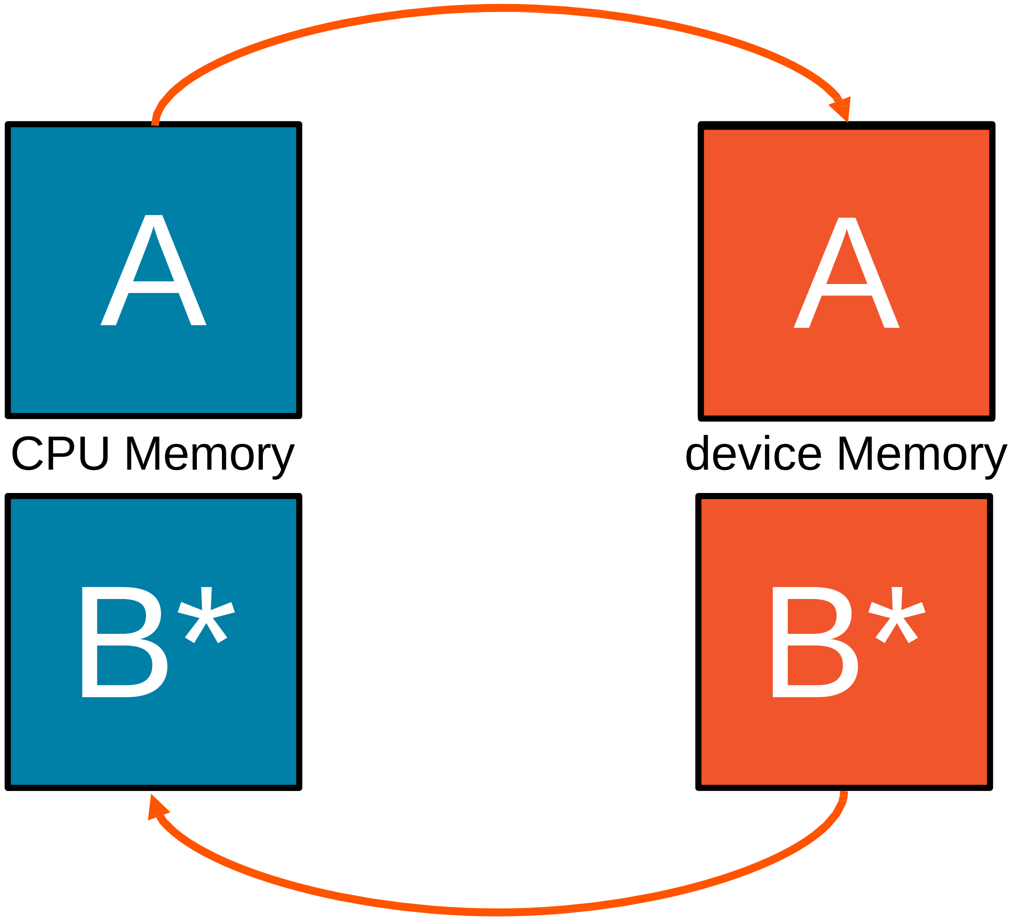
C/C++

```
!$acc update self(x(1:end_index))
!$acc update device(x(1:end_index))
```

Fortran

OPENACC UPDATE DIRECTIVE

`#pragma acc update device(A[0:N])`



The data must exist on both the CPU and device for the update directive to work.

`#pragma acc update self(A[0:N])`

SYNCHRONIZE DATA WITH UPDATE

```
int* A=(int*) malloc(N*sizeof(int)
#pragma acc data create(A[0:N])
while( timesteps++ < numSteps )
{
  #pragma acc parallel loop
  for(int i = 0; i < N; i++){
    a[i] *= 2;
  }

  if (timestep % 100 ) {
    #pragma acc update self(A[0:N])
    checkpointAToFile(A, N);
  }
}
```

- Sometimes data changes on the host or device inside a data region
- Ending the data region and starting a new one is expensive
- Instead, update the data so that the host and device data are the same
- Examples: File I/O, Communication, etc.

OpenACC More Science, Less Programming NVIDIA aws Linux Academy

CLOSING REMARKS

KEY CONCEPTS

In this lecture we discussed…

- Differences between CPU, GPU, and Unified Memories
- OpenACC Array Shaping
- OpenACC Data Clauses
- OpenACC Structured Data Region
- OpenACC Update Directive
- OpenACC Unstructured Data Directives

Next Week: Loop Optimizations