# OPENACC DATA MANAGEMENT

Bharat Kumar, Andreas Herten

OpenACC
More Science, Less Programming

# LECTURE 2 OUTLINE

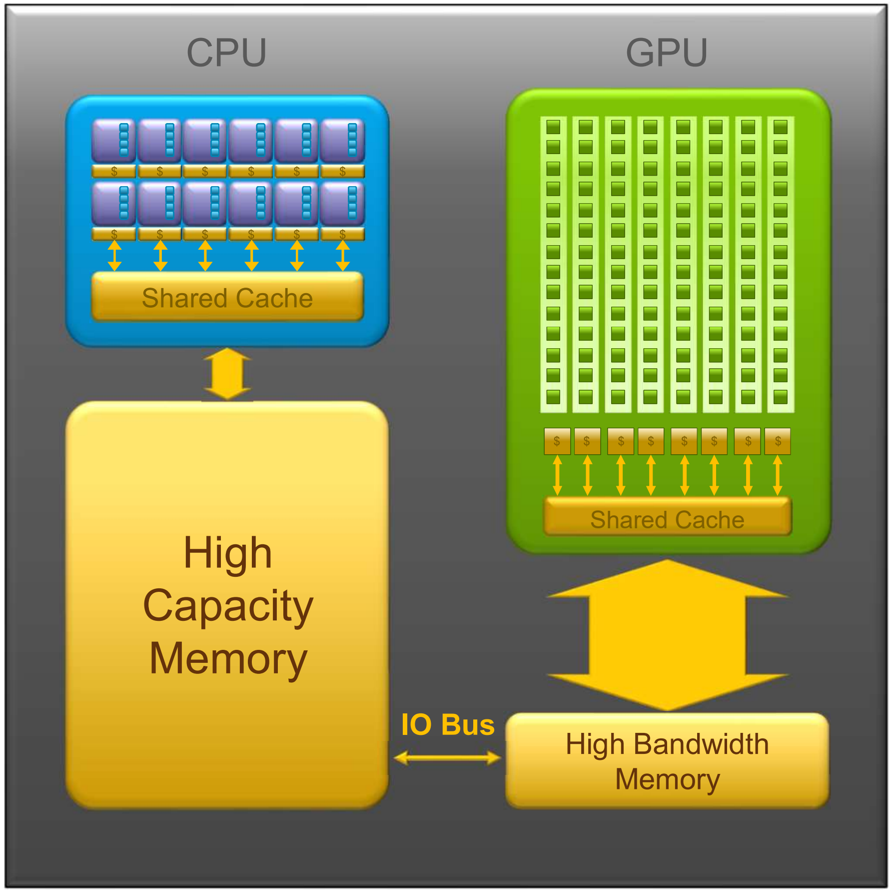
## Topics to be covered

- CPU and GPU Memories
- CUDA Unified (Managed) Memory
- OpenACC Data Management
- Lab 2

# 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

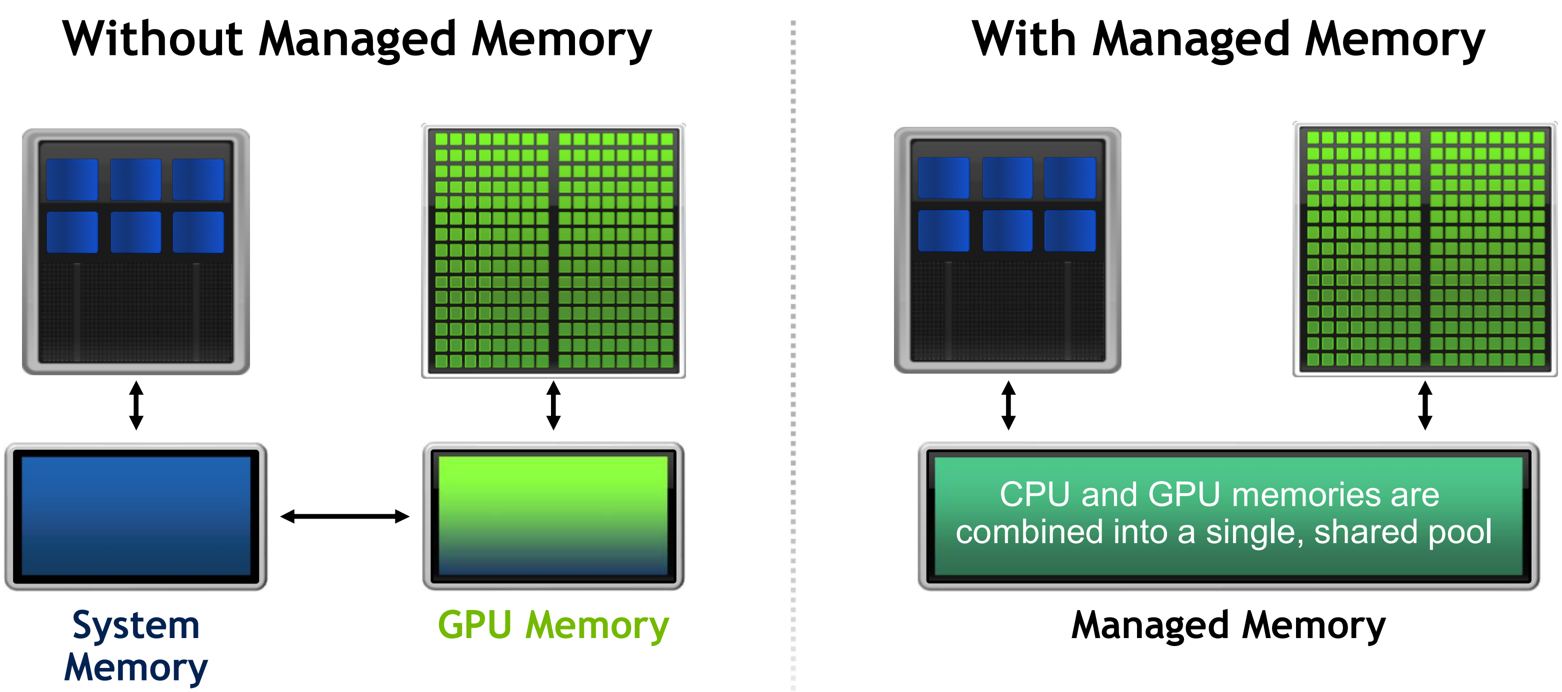


# CUDA UNIFIED MEMORY

# 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

```
$ pgcc -fast -acc -ta=tesla:managed -Minfo=accel main.c
```

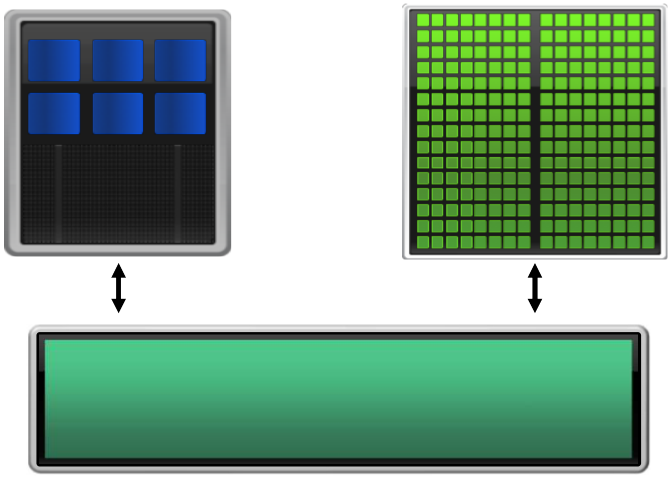
```
$ pgfortran -fast -acc -ta=tesla:managed -Minfo=accel main.f90
```

# 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.

**With Managed Memory**



# LAST TASK USED UNIFIED MEMORY

## Now let's make our code run without.

Why?

- Removes reliance on PGI and NVIDIA GPUs
- Currently the data always arrives "Just Too Late", let's do better

# TRY TO BUILD WITHOUT "MANAGED"

## Change –ta=tesla:managed to remove "managed"

```
pgcc -ta=tesla -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

# 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))
```

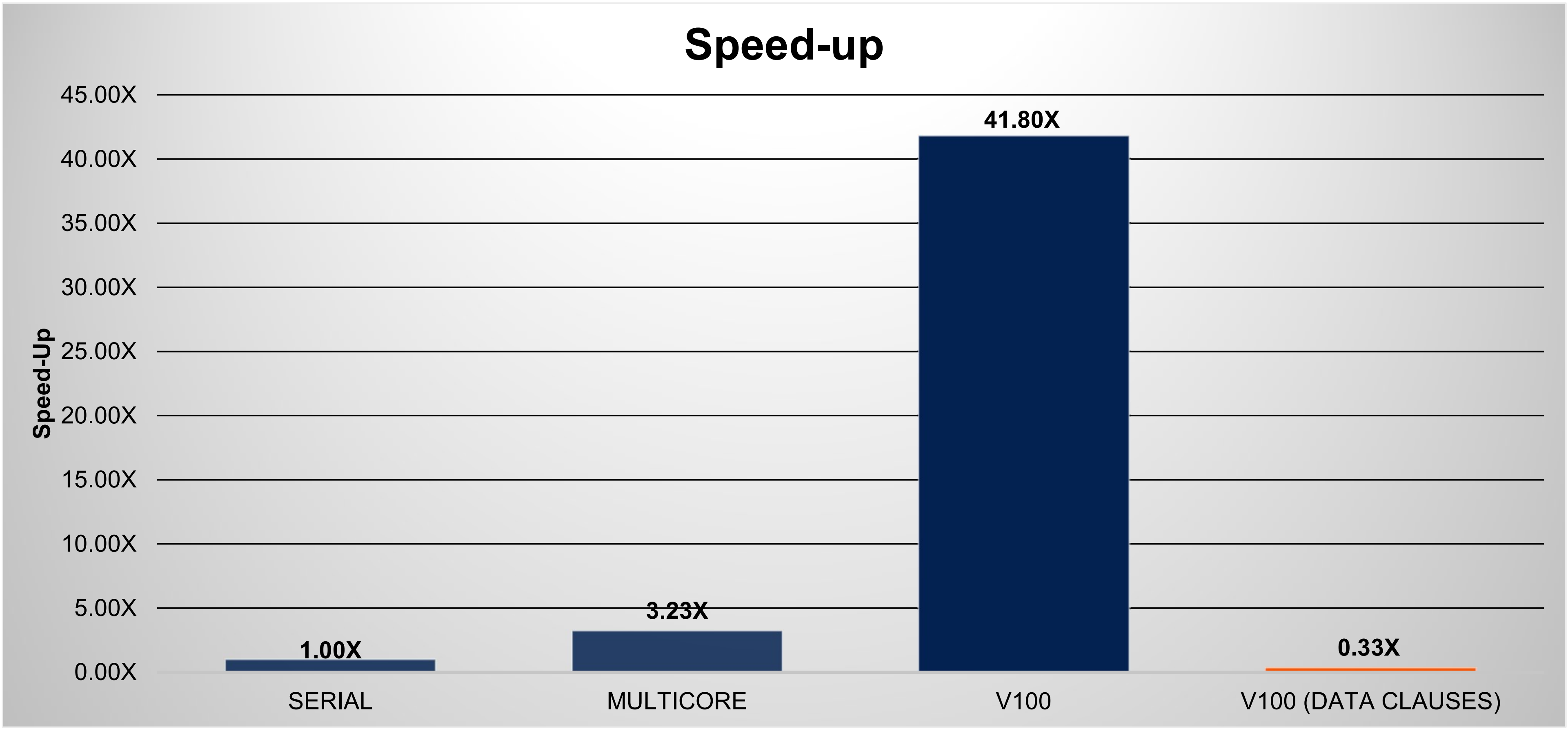
Fortran

# 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.

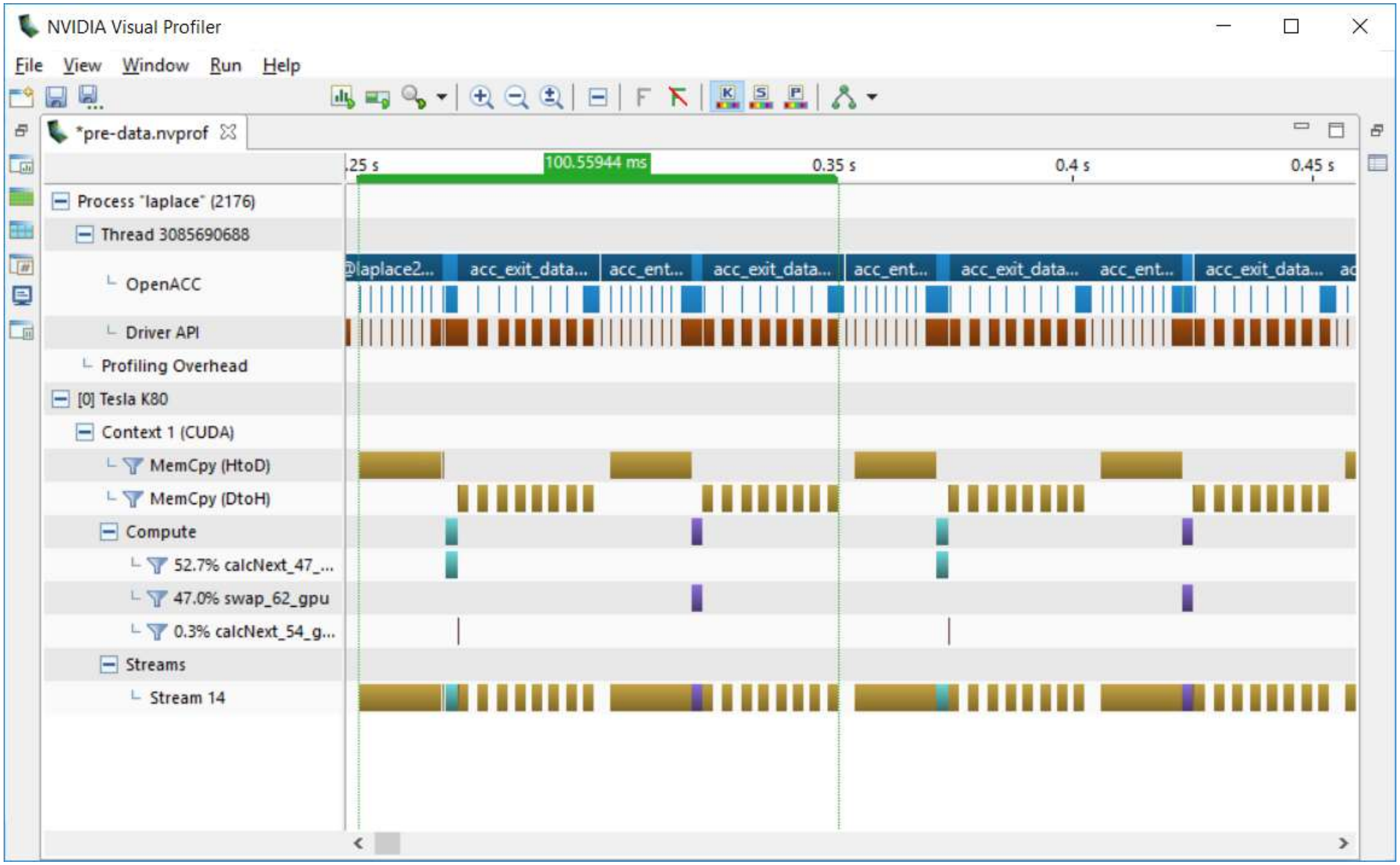
# OPENACC ~~SPEED-UP~~ SLOWDOWN



# WHAT WENT WRONG?

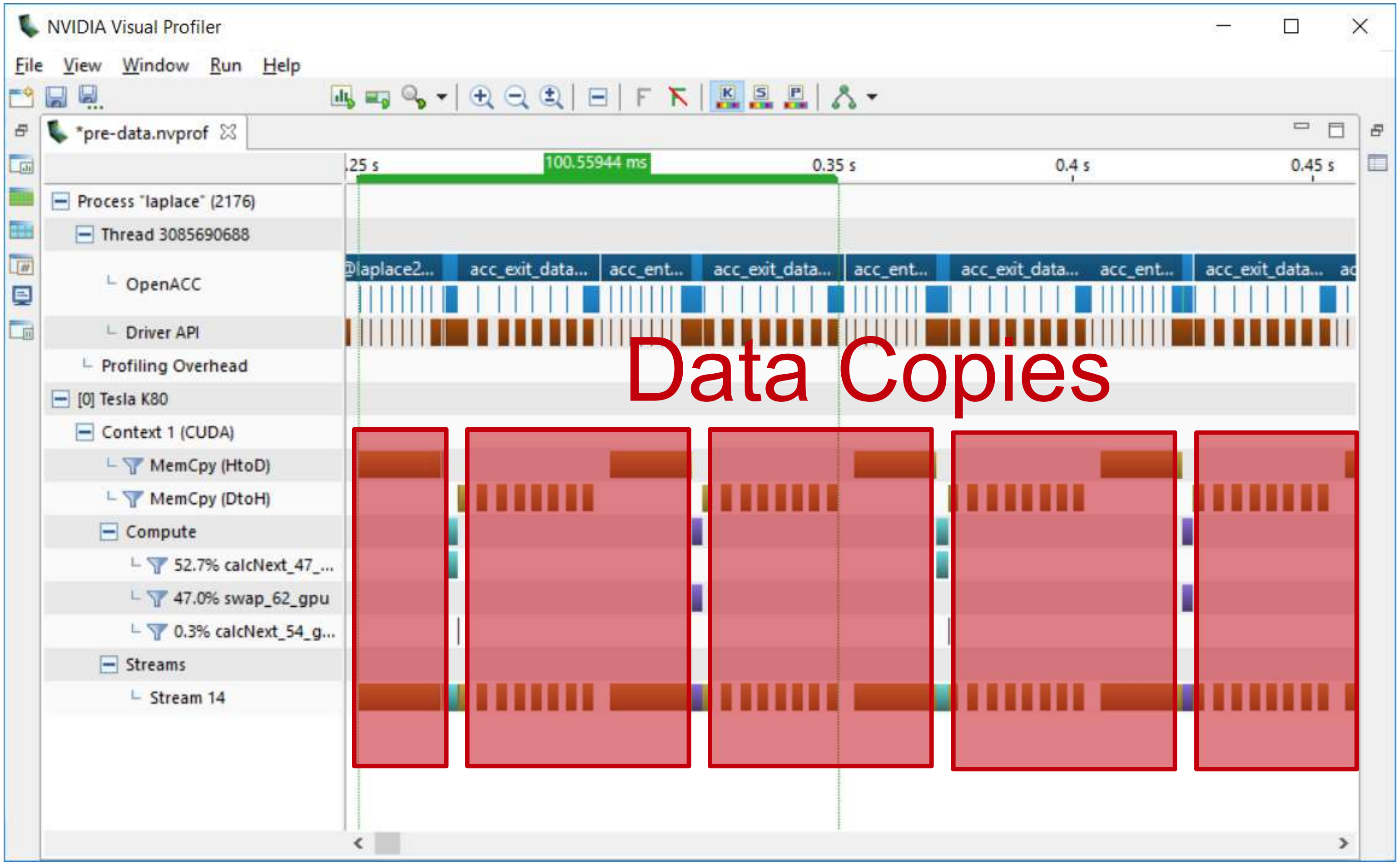
- The code now has all of the information necessary to build without managed memory, but it runs much slower.
- Profiling tools are here to help!

# APPLICATION PROFILE (2 STEPS)



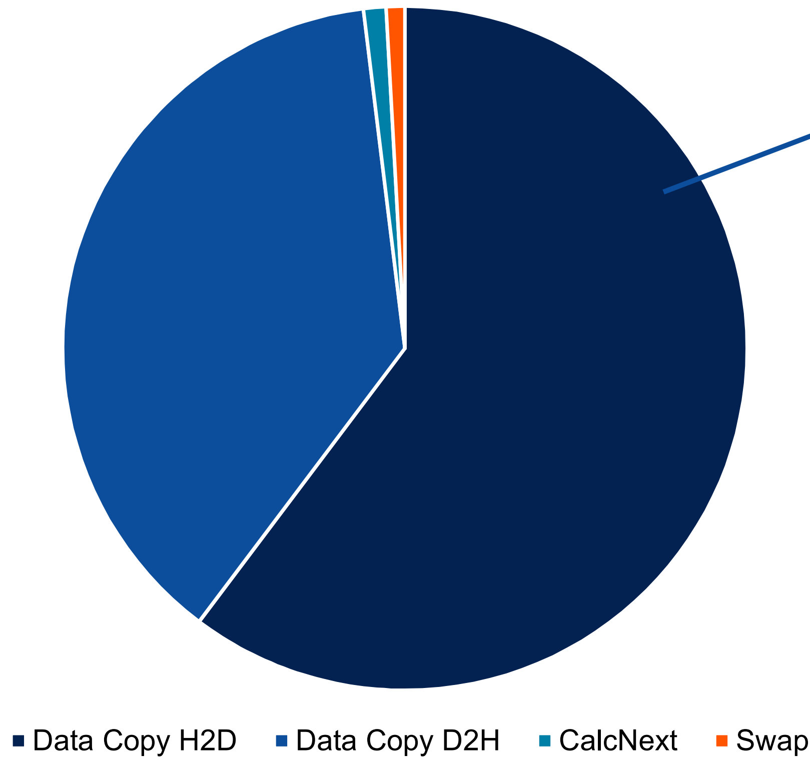

OpenACC

# APPLICATION PROFILE (2 STEPS)




OpenACC

# 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.**

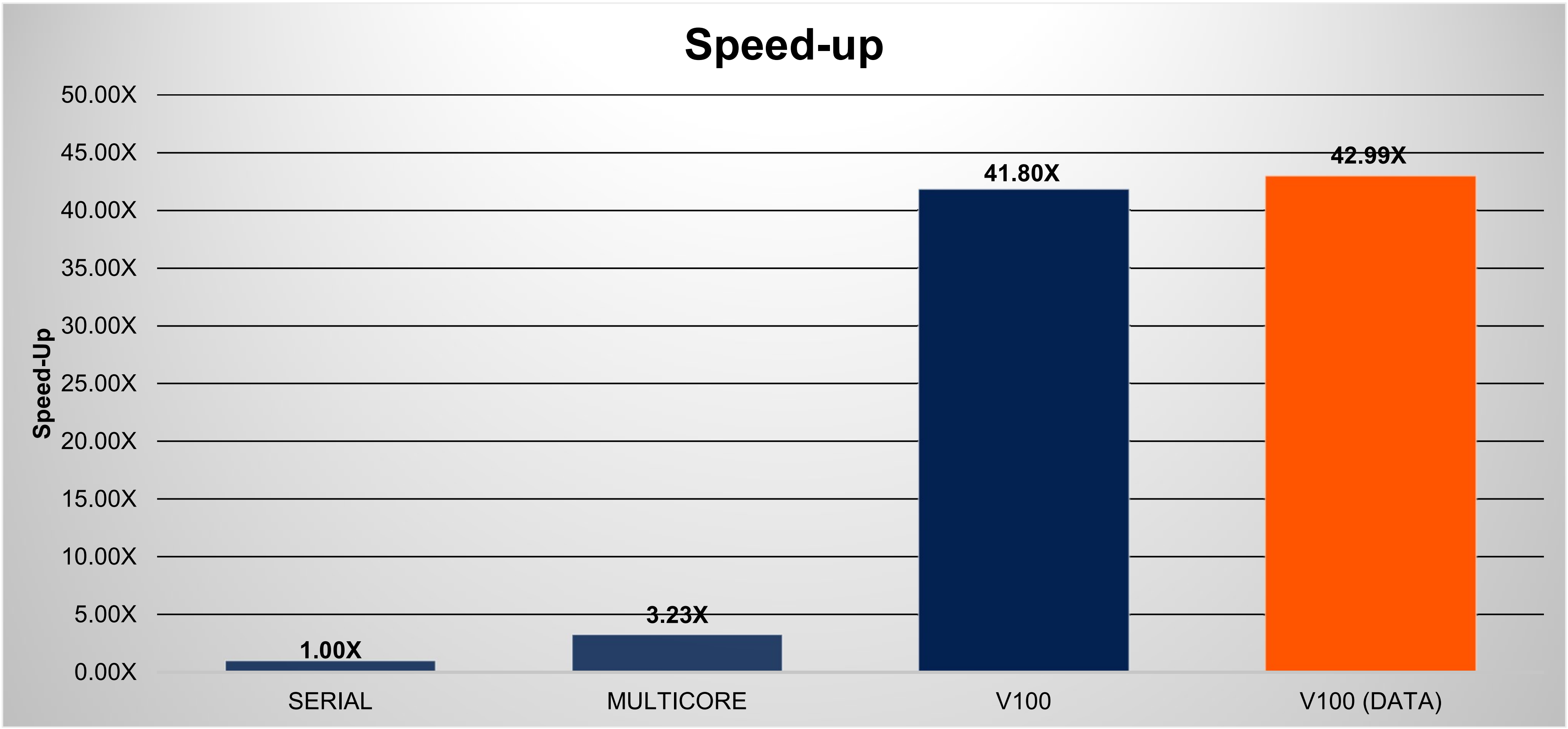
**Copy initial condition of Anew, but not final value**

# 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 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

# UNSTRUCTURED DATA DIRECTIVES

# UNSTRUCTURED DATA DIRECTIVES

## Enter Data Directive

- Data lifetimes aren't always neatly structured.
- The **enter data** directive handles device memory **allocation**
- You may use either the **create** or the **copyin** clause for memory allocation
- The enter data directive is **not** the start of a data region, because you may have multiple enter data directives

```
#pragma acc enter data clauses

  < Sequential and/or Parallel code >

#pragma acc exit data clauses
```

```
!$acc enter data clauses

  < Sequential and/or Parallel code >

!$acc exit data clauses
```

# UNSTRUCTURED DATA DIRECTIVES

## Exit Data Directive

- The **exit data** directive handles device memory **deallocation**
- You may use either the **delete** or the **copyout** clause for memory deallocation
- You should have as many **exit data** for a given array as **enter data**
- These can exist in different functions

```
#pragma acc enter data clauses

  < Sequential and/or Parallel code >

#pragma acc exit data clauses
```

```
!$acc enter data clauses

  < Sequential and/or Parallel code >

!$acc exit data clauses
```

# UNSTRUCTURED VS STRUCTURED

## With a simple code

### Unstructured

- Can have multiple starting/ending points
- Can branch across multiple functions
- Memory exists until explicitly deallocated

```
#pragma acc enter data copyin(a[0:N],b[0:N]) \
  create(c[0:N])

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

#pragma acc exit data copyout(c[0:N]) \
  delete(a,b)
```

### Structured

- Must have explicit start/end points
- Must be within a single function
- Memory only exists within the data region

```
#pragma acc data copyin(a[0:N],b[0:N]) \
  copyout(c[0:N])
{
  #pragma acc parallel loop
  for(int i = 0; i < N; i++){
    c[i] = a[i] + b[i];
  }

}
```

# UNSTRUCTURED DATA DIRECTIVES

## Branching across multiple functions

```
int* allocate_array(int N){
   int* ptr = (int *) malloc(N * sizeof(int));
   #pragma acc enter data create(ptr[0:N])
   return ptr;
}

void deallocate_array(int* ptr){
   #pragma acc exit data delete(ptr)
   free(ptr);
}

int main(){
   int* a = allocate_array(100);
   #pragma acc kernels
   {
      a[0] = 0;
   }
   deallocate_array(a);
}
```

- In this example enter data and exit data are in different functions
- This allows the programmer to put device allocation/deallocation with the matching host versions
- This pattern is particularly useful in C++, where structured scopes may not be possible.

# 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

# OPENACC RESOURCES

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

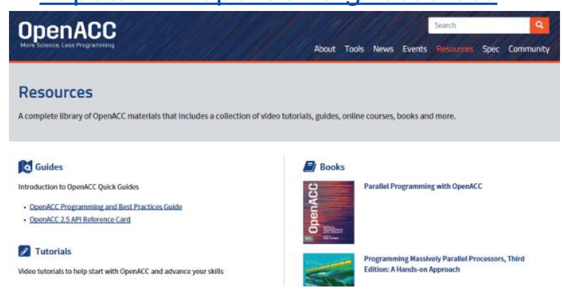
**FREE Compilers**

PGI Community EDITION

OpenACC NVIDIA aws Linux Academy

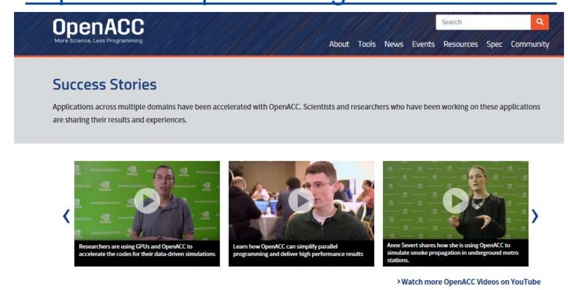
## Resources

https://www.openacc.org/resources



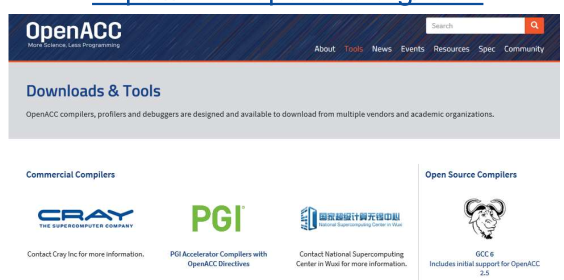

## Success Stories

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




## Compilers and Tools

https://www.openacc.org/tools




## Events

https://www.openacc.org/events