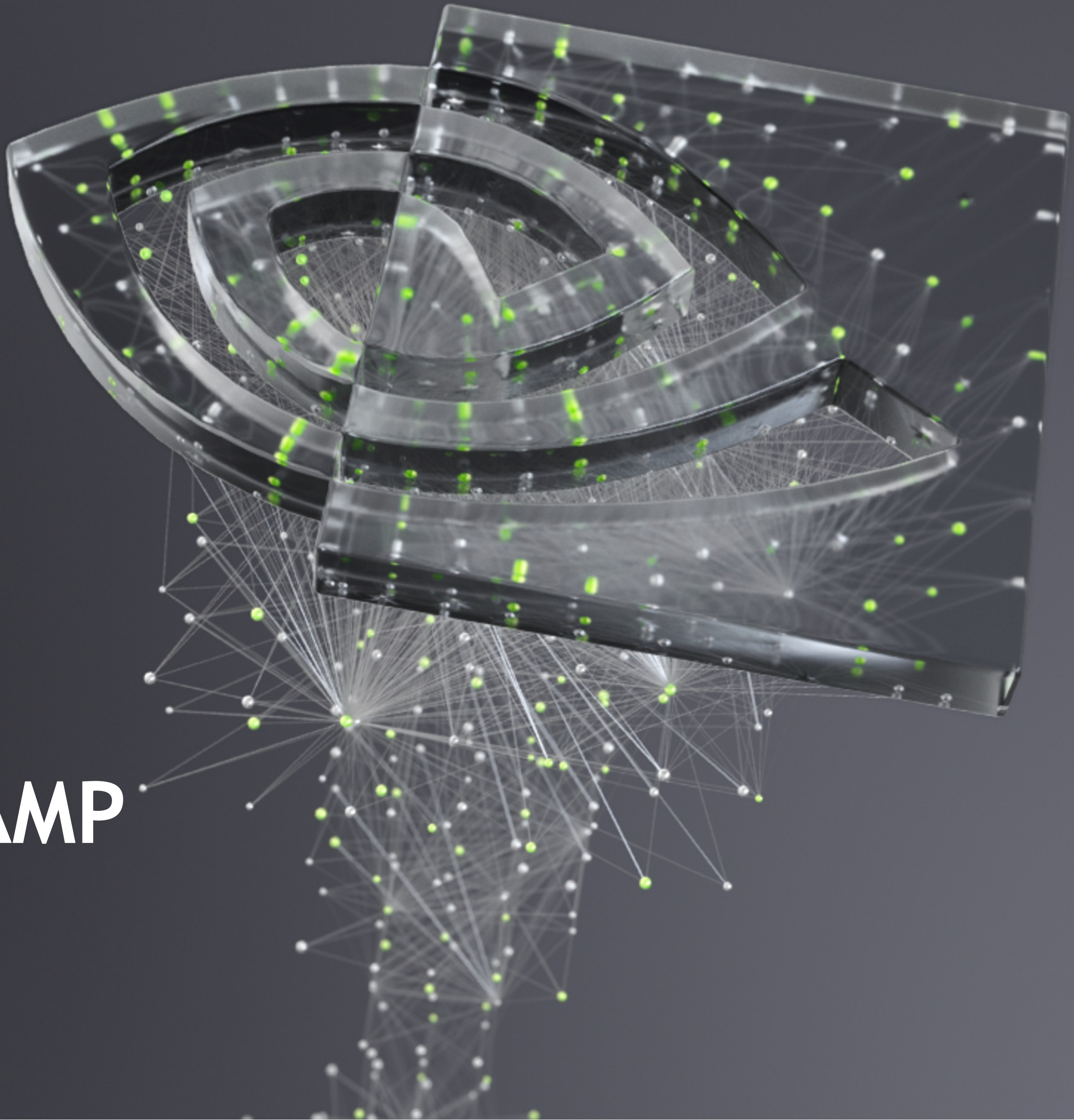
NVIDIA



N-WAYS GPU BOOTCAMP

OPENACC

OPENACC

What to expect?

- Basic introduction to OpenACC directives
- HPC SDK Usage
- Portability across Multicore and GPU

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

OpenACC is...

a directives-based

parallel programming model

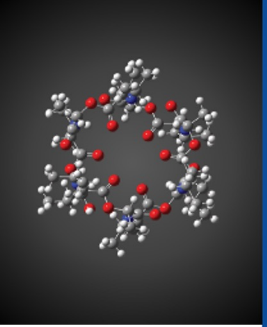
designed for

performance and **portability.**

Add Simple Compiler Directive

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

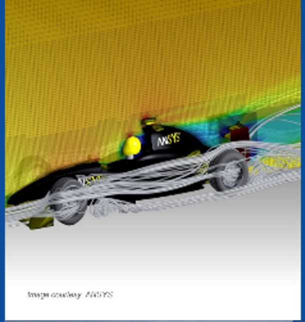
This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)



GAUSSIAN 16

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

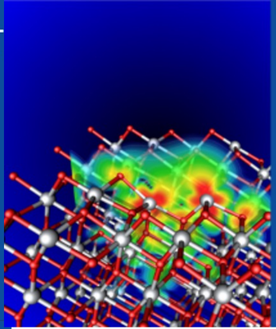
“Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/ network and GPU parallelism. PGI's compilers were essential to the success of our efforts.”



ANSYS FLUENT

Sunil Sathe
Lead Software Developer
ANSYS Fluent

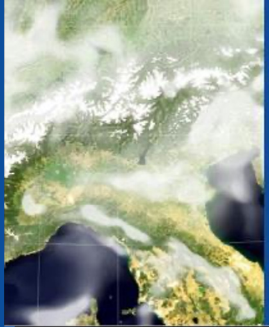
“We've effectively used OpenACC for heterogeneous computing in ANSYS Fluent with impressive performance. We're now applying this work to more of our models and new platforms.”



VASP

Prof. Georg Kresse
Computational Materials Physics
University of Vienna

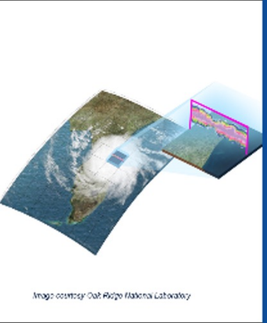
“For VASP, OpenACC is *the* way forward for GPU acceleration. Performance is similar and in some cases better than CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts. We're excited to collaborate with NVIDIA and PGI as an early adopter of CUDA Unified Memory.”



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

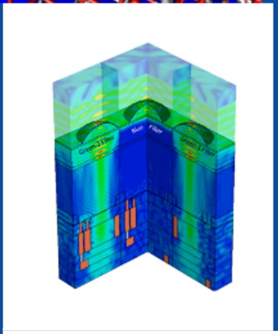
Mark A. Taylor
Multiphysics Applications
Sandia

“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

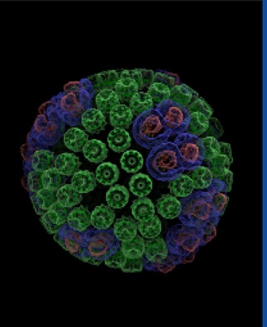
Dr. Lutz Schneider
Senior R&D Engineer
Synopsys Inc

“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

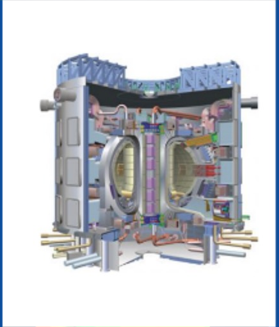
“Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer.”



VMD

John Stone
Senior Research Programmer
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

GAMERA

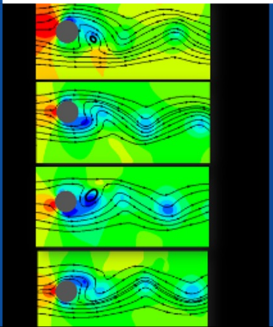
Takuma Yamaguchi, Kohei Fujita, Tsuyoshi Ichimura, Muneo Hori, Lalith Wijerathne
The University of Tokyo

“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

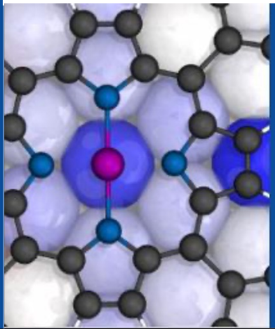
“In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task.”



IBM-CFD

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

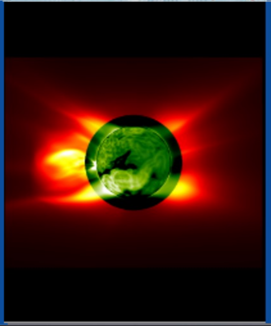
“OpenACC can prove to be a handy tool for computational engineers and researchers to obtain fast solution of non-linear dynamics problem. In immersed boundary incompressible CFD, we have obtained order of magnitude reduction in computing time by porting several components of our legacy codes to GPU. 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 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

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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.

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

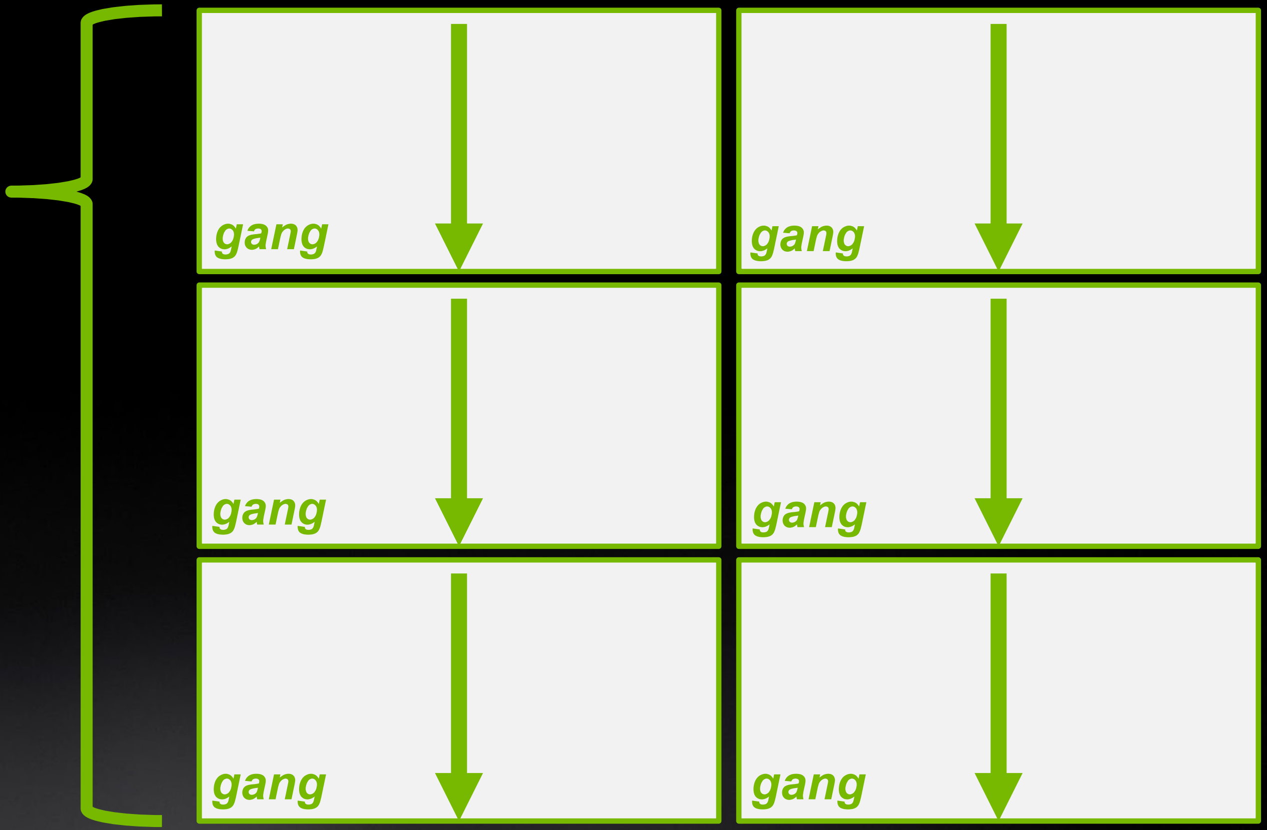
OPENACC PARALLEL DIRECTIVE

Expressing parallelism

```
#pragma acc parallel
{
```

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

```
}
```



This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

OPENACC PARALLEL DIRECTIVE

Expressing parallelism

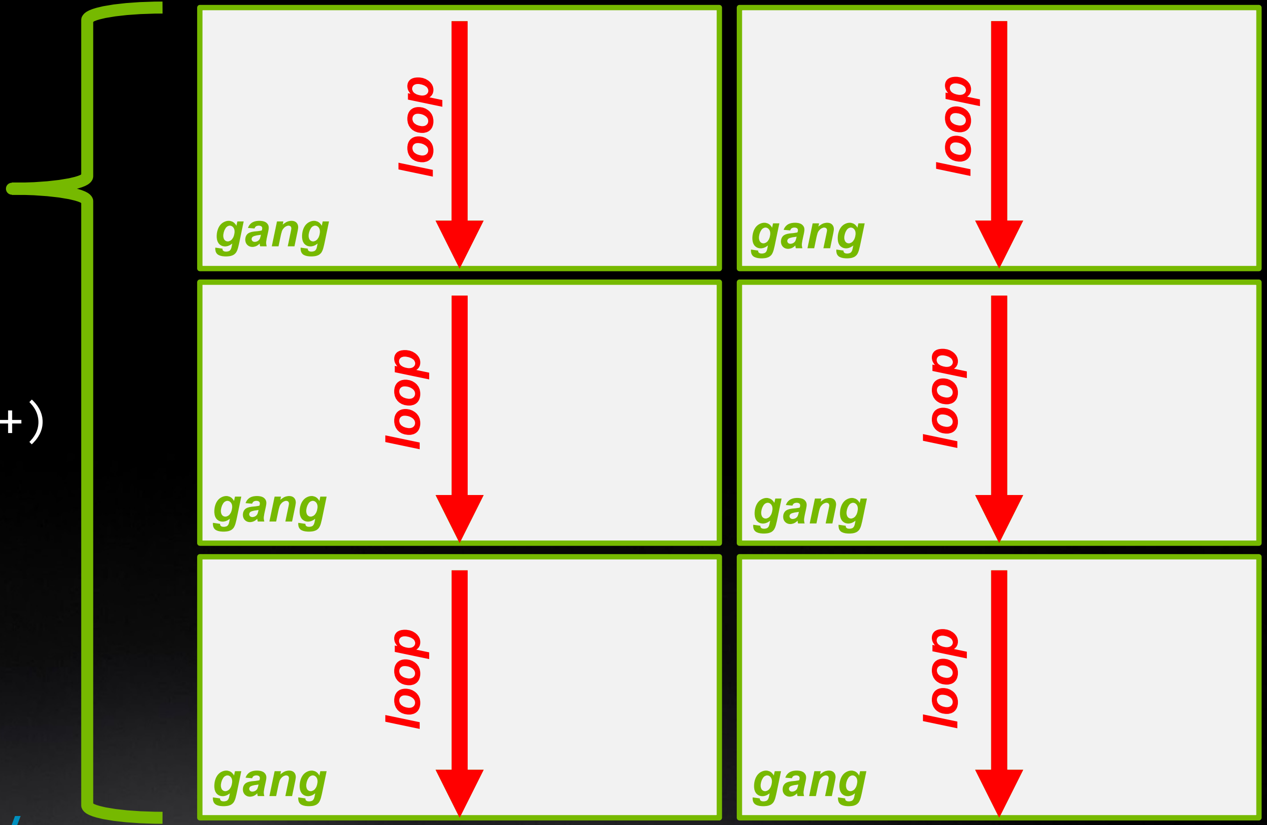
```
#pragma acc parallel
{
```

loop

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

}
```

This loop will be executed redundantly on each gang



This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

OPENACC PARALLEL DIRECTIVE

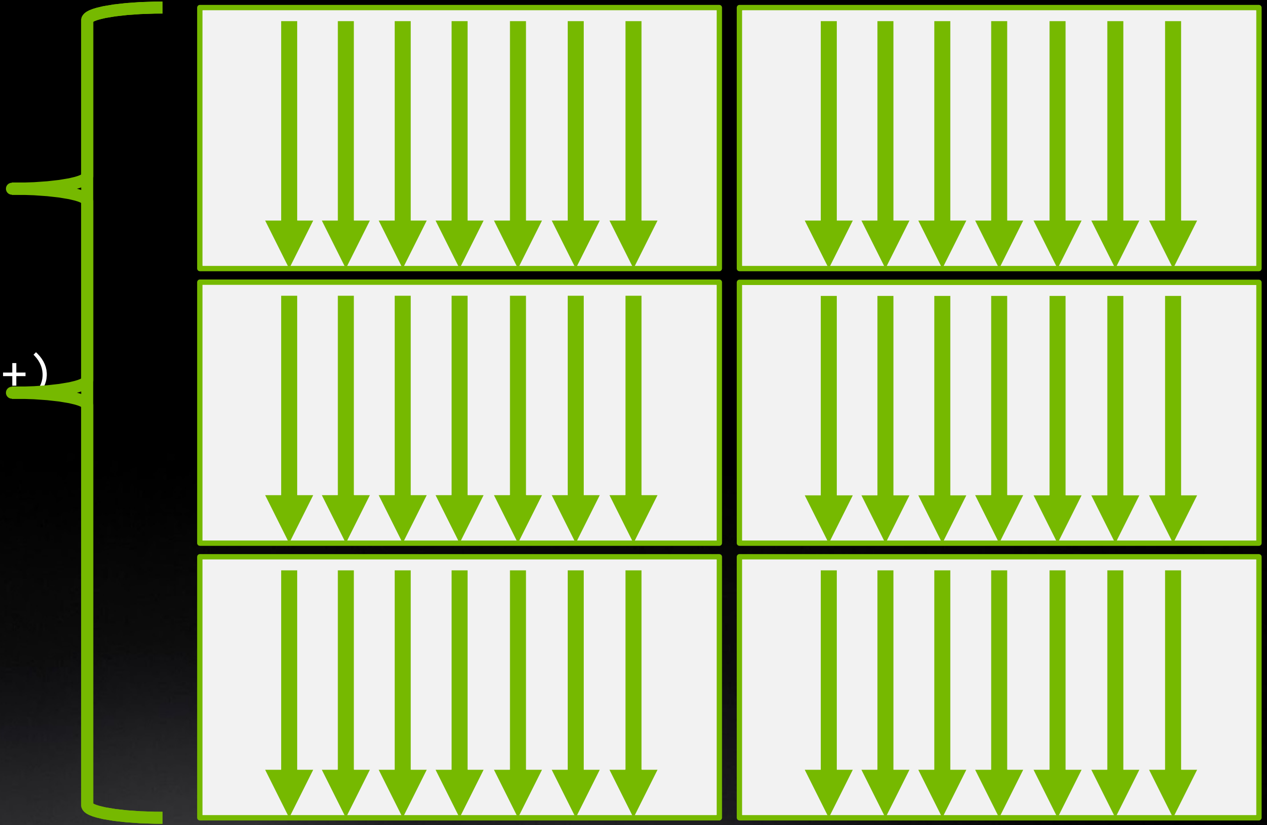
Expressing parallelism

```
#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.



This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

BUILD AND RUN THE CODE

NVIDIA HPC SDK

- Comprehensive suite of compilers, libraries, and tools used to GPU accelerate HPC modeling and simulation application
- The NVIDIA HPC SDK includes the new NVIDIA HPC compiler supporting OpenACC C and Fortran
  - The command to compile C code is ‘nvc’
  - The command to compile C++ code is ‘nvc++’
  - The command to compile Fortran code is ‘nvfortran’

```
nvc –fast –Minfo=accel –ta=tesla:managed main.c
or
nvc -fast -Minfo=accel -acc=gpu -gpu=managed main.c
```

```
nvfortran –fast –Minfo=accel –ta=tesla:managed main.f90
or
nvfortran -fast -Minfo=accel -acc=gpu -gpu=managed main.f90
```

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

BUILDING THE CODE

-Minfo shows more details

```
$ nvc -fast -ta=multicore -Minfo=accel 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
```

```
$ nvc -fast -ta=tesla:managed -Minfo=accel rdf.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 copy(error)
     66, Loop is parallelizable
```

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

RDF

Pseudo Code

```
for (int frame=0;frame<nconf;frame++){

  for(int id1=0;id1<numatm;id1++)
  {
    for(int id2=0;id2<numatm;id2++)
    {
      dx=d_x[]-d_x[];
      dy=d_y[]-d_y[];
      dz=d_z[]-d_z[];

      r=sqrtf(dx*dx+dy*dy+dz*dz);

      if (r<cut) {
        ig2=(int)(r/del);
        d_g2[ig2] = d_g2[ig2] +1 ;
      }
    }
  }
}
```

- Across Frames
- Find Distance
- Reduction

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

RDF
Pseudo Code -C

```
for (int frame=0;frame<nconf;frame++) {

  #pragma acc parallel loop
  for(int id1=0;id1<numatm;id1++) {
    for(int id2=0;id2<numatm;id2++) {
      dx=d_x[]-d_x[];
      dy=d_y[]-d_y[];
      dz=d_z[]-d_z[];

      r=sqrtf(dx*dx+dy*dy+dz*dz);

      if (r<cut) {
        ig2=(int)(r/del);
        #pragma acc atomic
        d_g2[ig2] = d_g2[ig2] +1 ;
      }
    }
  }
}
```

- Parallel Loop construct

- Atomic Construct

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

RDF

Pseudo Code - Fortran

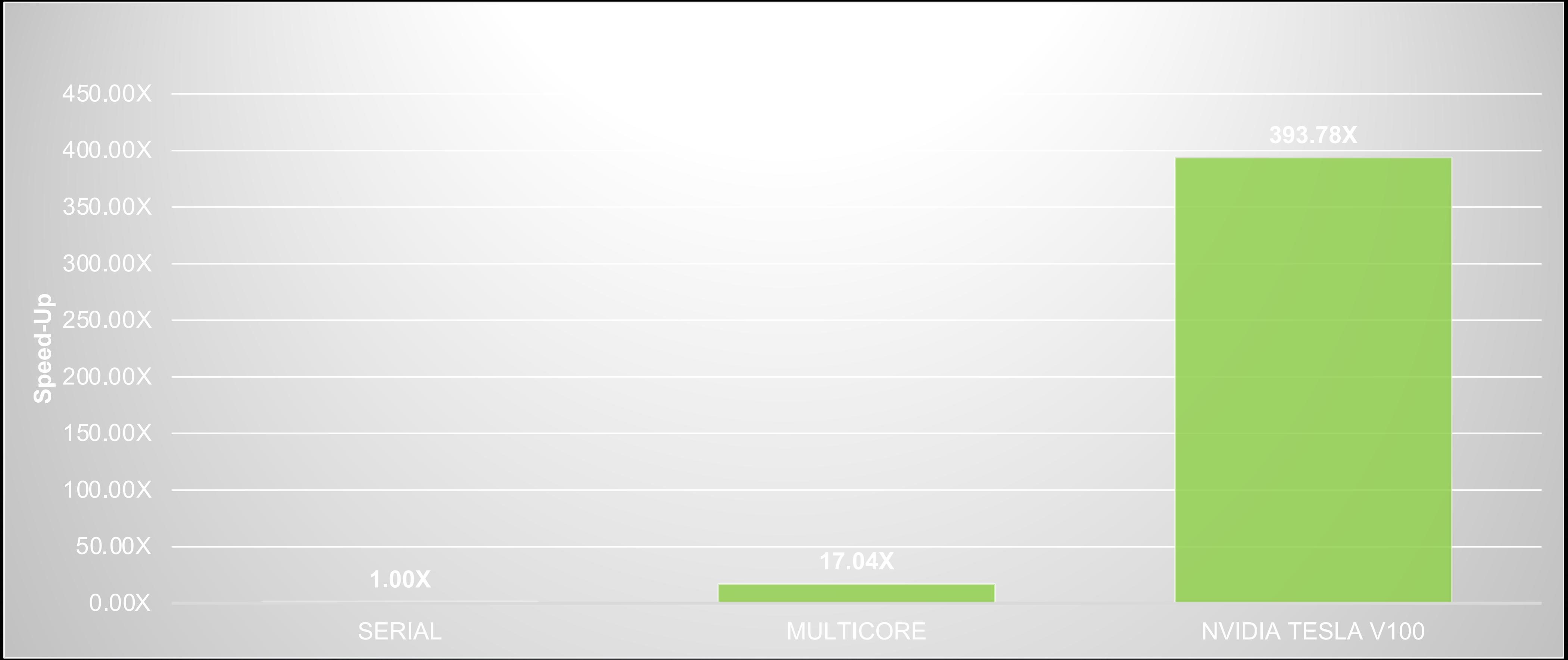
```
do iconf=1,nframes
     if (mod(iconf,1).eq.0) print*,iconf
     !$acc parallel loop
     do i=1,natoms
       do j=1,natoms
         dx=x(iconf,i)-x(iconf,j)
         dy=y(iconf,i)-y(iconf,j)
         dz=z(iconf,i)-z(iconf,j)
                     ...
         if(r<cut)then
           !$acc atomic
           g(ind)=g(ind)+1.0d0
         endif
       enddo
     enddo
   enddo
```

- Parallel Loop construct

- Atomic Construct

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

OPENACC SPEEDUP



HPC SDK 20.11, NVIDIA Tesla V100, DGX1

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

REFERENCES

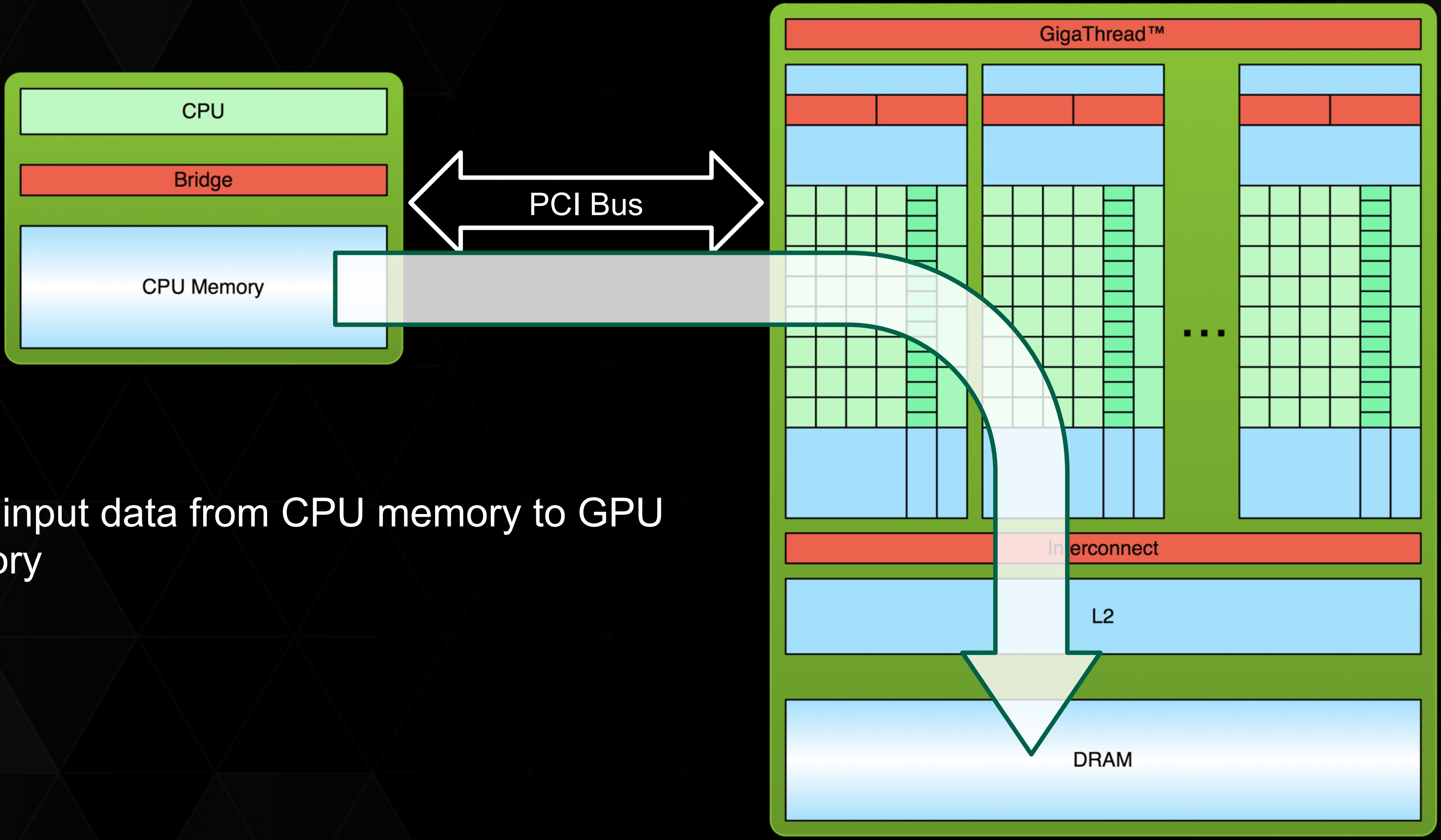
https://www.openacc.org/get-started

https://developer.nvidia.com/hpc-sdk

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

ADDITIONAL EXERCISE CONTENT

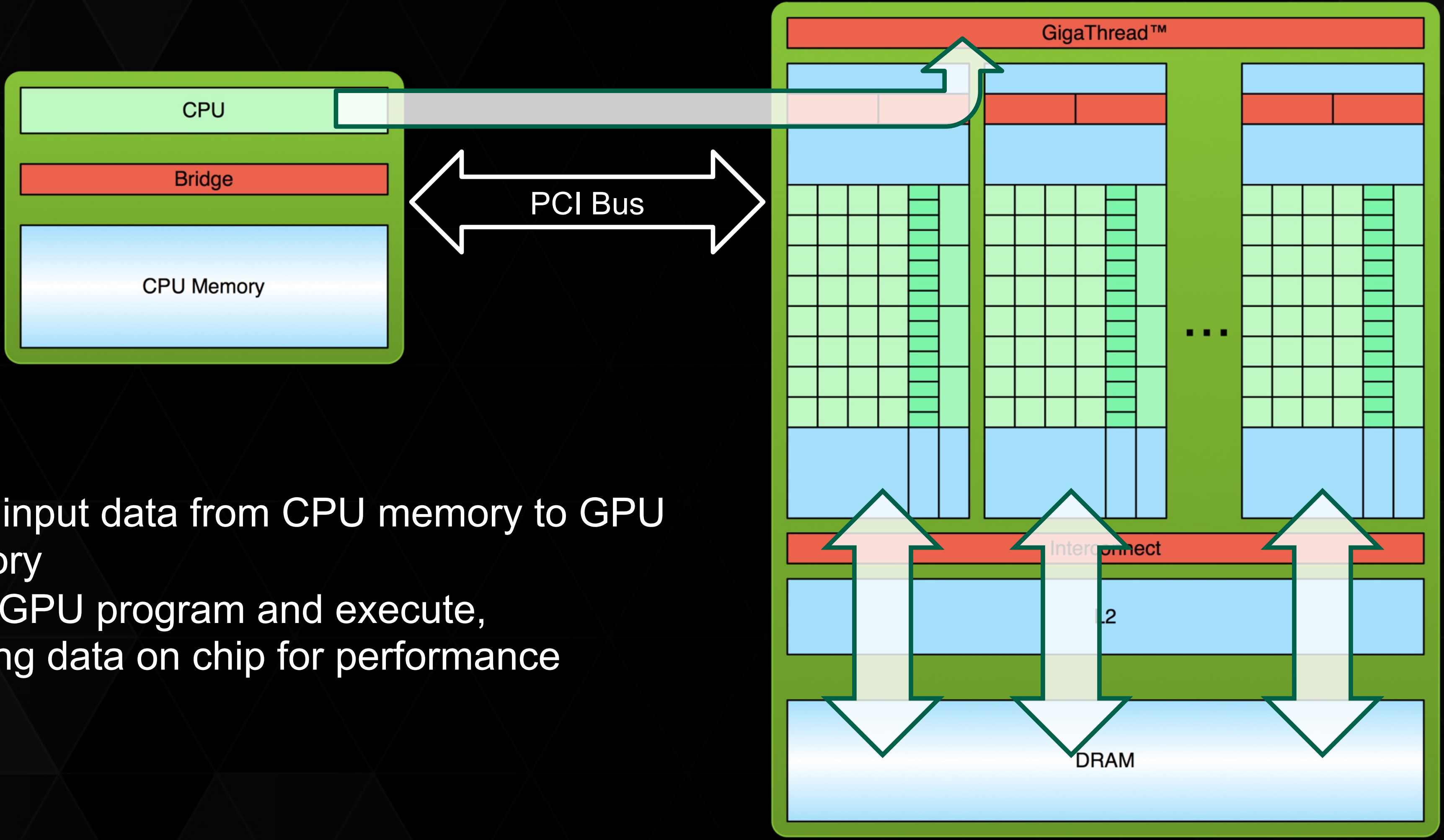
PROCESSING FLOW – STEP 1



1. Copy input data from CPU memory to GPU memory

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

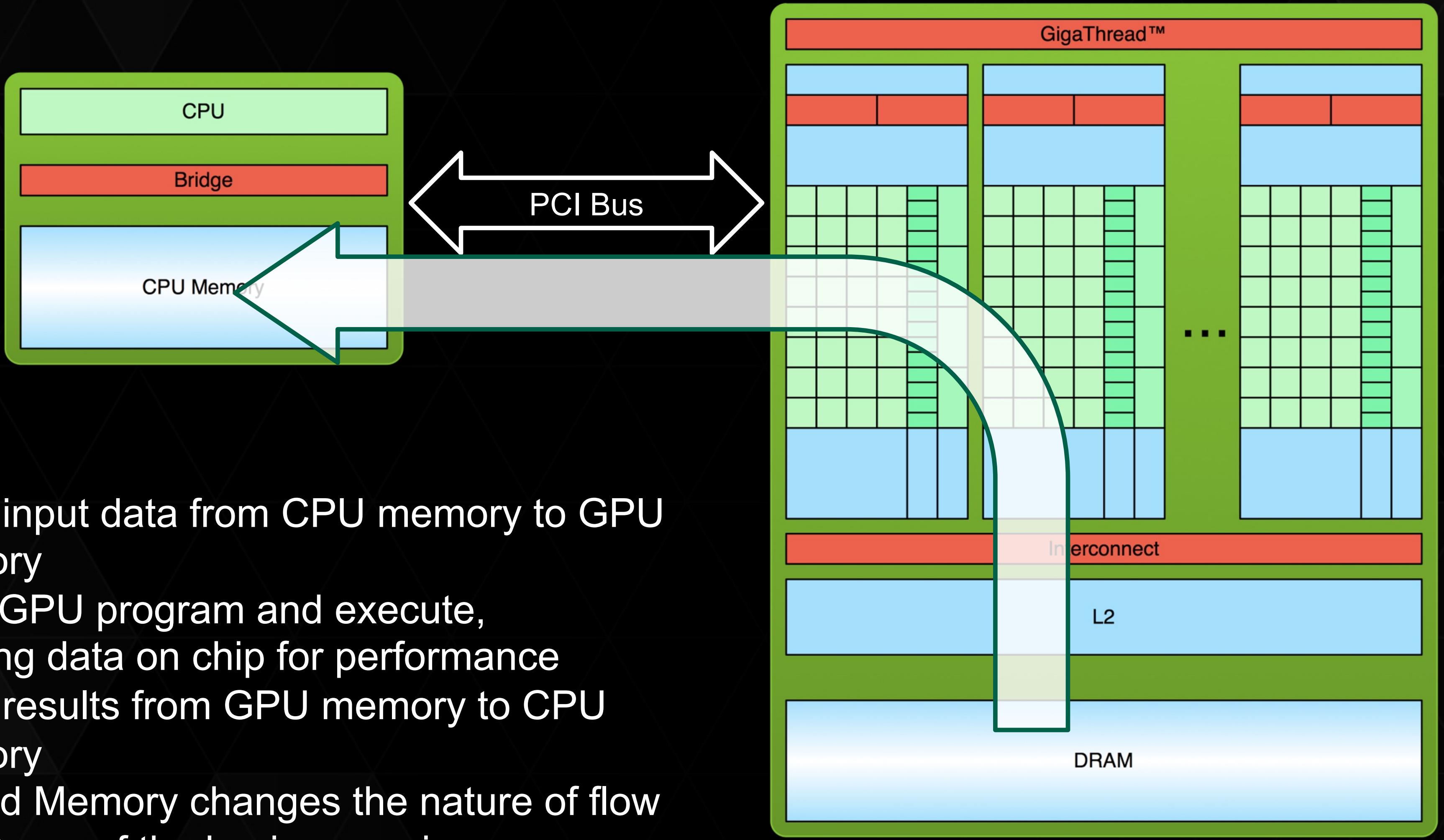
PROCESSING FLOW – STEP 2



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

PROCESSING FLOW – STEP 3



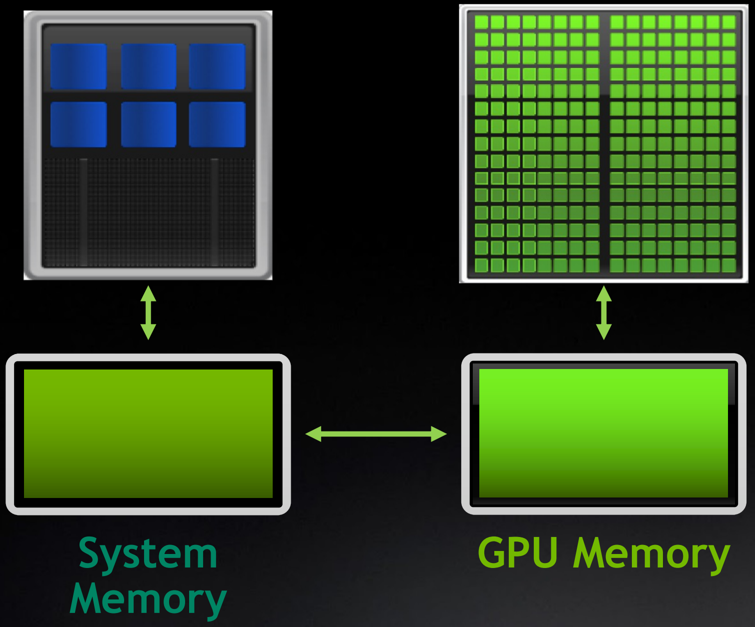
1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory
4. Unified Memory changes the nature of flow
   - Some of the basics remains same

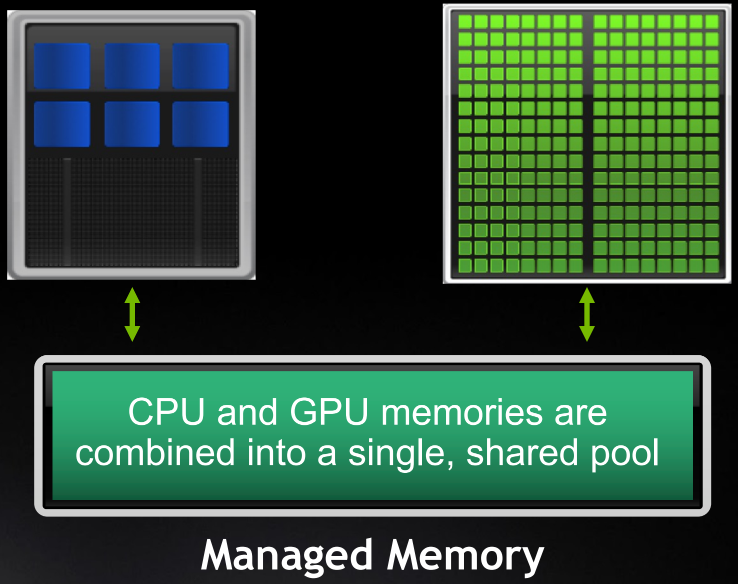
This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

CUDA UNIFIED MEMORY

Simplified Developer Effort

Commonly referred to as *"managed memory."*




CPU and GPU memories are combined into a single, shared pool
Managed Memory

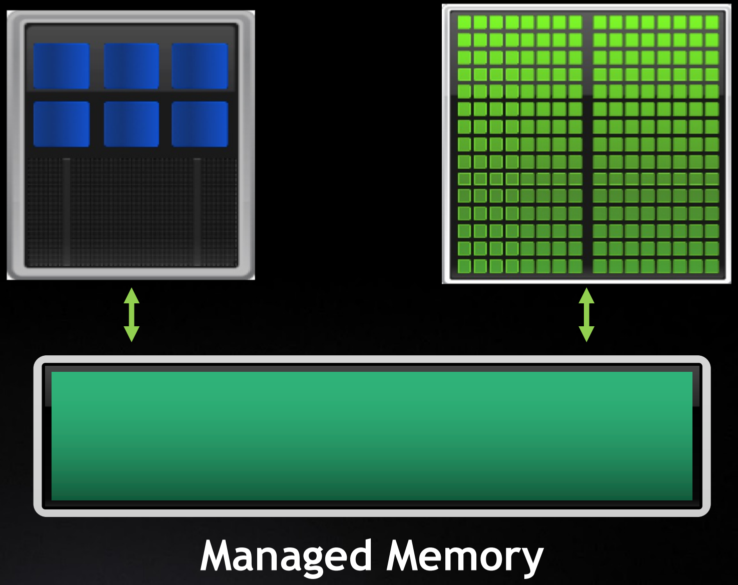
This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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 NVIDIA Compiler on NVIDIA GPUs.

With Managed Memory



This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

CUDA MANAGED MEMORY

- OpenACC:

```
nvc++ -fast -ta=tesla:managed -Minfo=accel main.c
                     or
nvc++ -fast -acc=gpu -gpu=managed -Minfo=accel main.c
```

  - Enabled using -ta=tesla:managed or -gpu=managed

- std::par

```
nvc++ -stdpar=gpu program.cpp -o program
```

  - All allocations uses managed memory

- OpenMP:
  - Current Beta release does not support Unified memory. Need explicitly map target directive to copy data

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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
```

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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.

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

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

This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)

STRUCTURED DATA DIRECTIVE

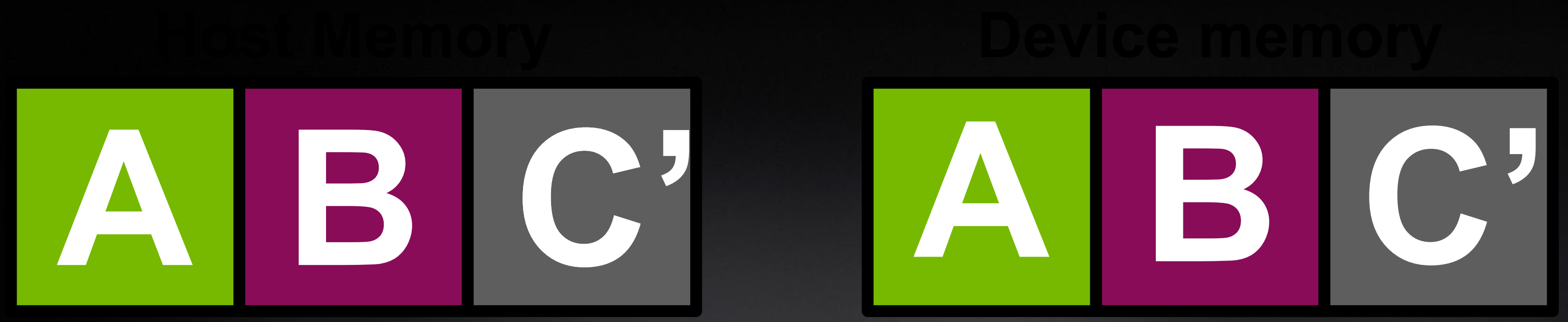
Example

```
#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];
      }
}
```

Action

Allocate A on device

[illegible]



NVIDIA.
This material is released by NVIDIA Corporation under the Creative Commons Attribution 4.0 International (CC BY 4.0)