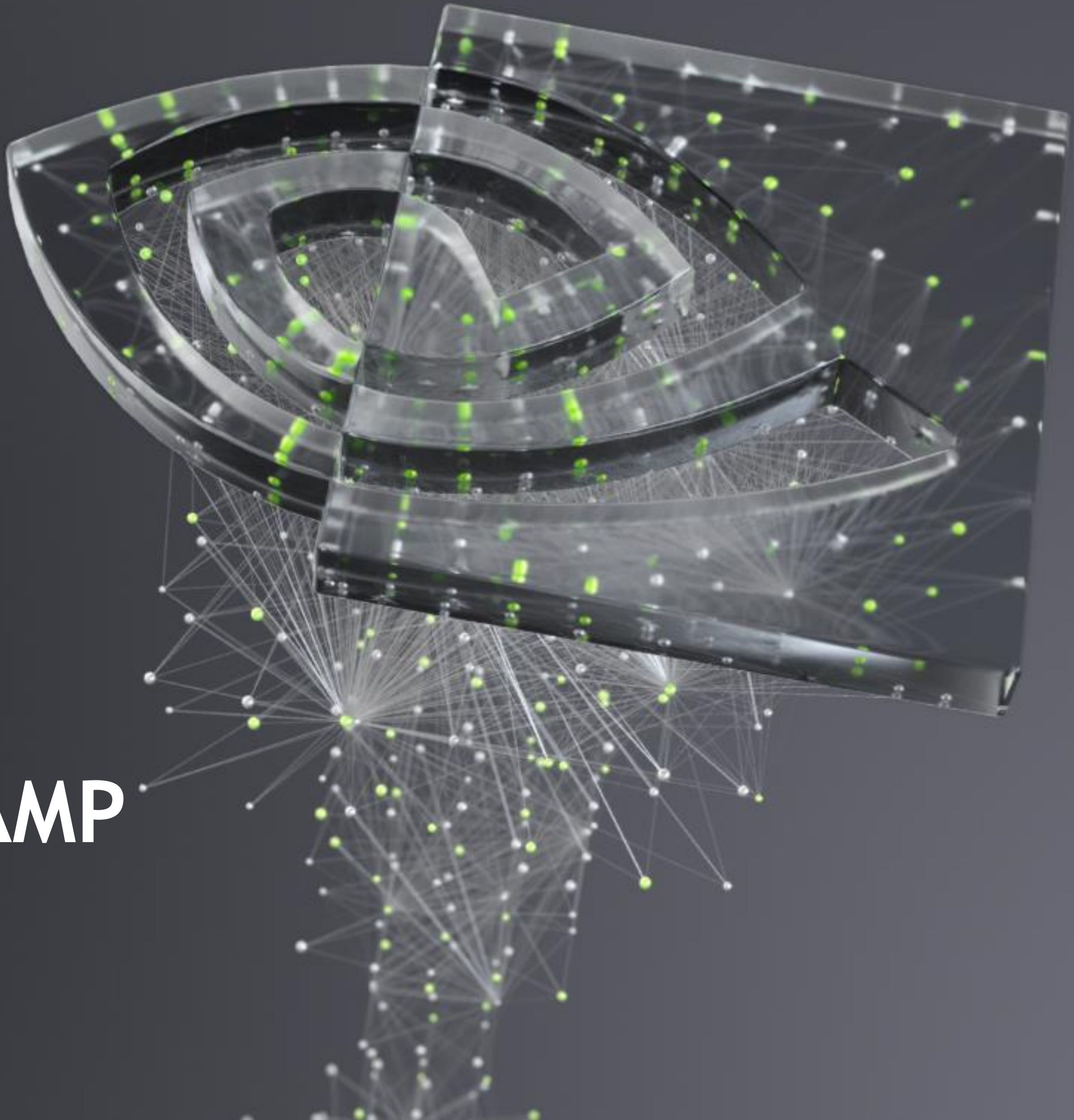
N-WAYS GPU BOOTCAMP

OPENACC



Agenda

OpenMP CPU

OpenACC GPU → LAB

OpenMP GPU → LAB

OPENMP

What to expect?

- OpenMP basic
- OpenMP target offload constructs for accelerated computing
- Portability between multicore and GPU

OPENMP

A Brief History

- 1996 - Architecture Review Board (ARB) formed by several vendors implementing their own directives for Shared Memory Parallelism (SMP).
- 1997 - 1.0 was released for C/C++ and Fortran with support for parallelizing loops across threads.
- 2000, 2002 – Version 2.0 of Fortran, C/C++ specifications released.
- 2005 – Version 2.5 released, combining both specs into one.
- 2008 – Version 3.0 released, added support for tasking
- 2011 – Version 3.1 release, improved support for tasking
- 2013 – Version 4.0 released, added support for offloading (and more)
- 2015 – Version 4.5 released, improved support for offloading targets (and more)

OPENMP ON CPU

OPENMP

Syntax

```
#pragma omp directive
```

```
!$ omp directive
```

- #pragma in C/C++ is what's known as a "compiler hint."
- omp is an addition to our pragma, it is known as the “sentinel”. It specifies that this is an OpenMP pragma. Any non-OpenMP compiler will ignore this pragma.
- directives are commands in OpenMP that will tell the compiler to do some action. For now, we will only use directives that allow the compiler to parallelize our code

OPENMP

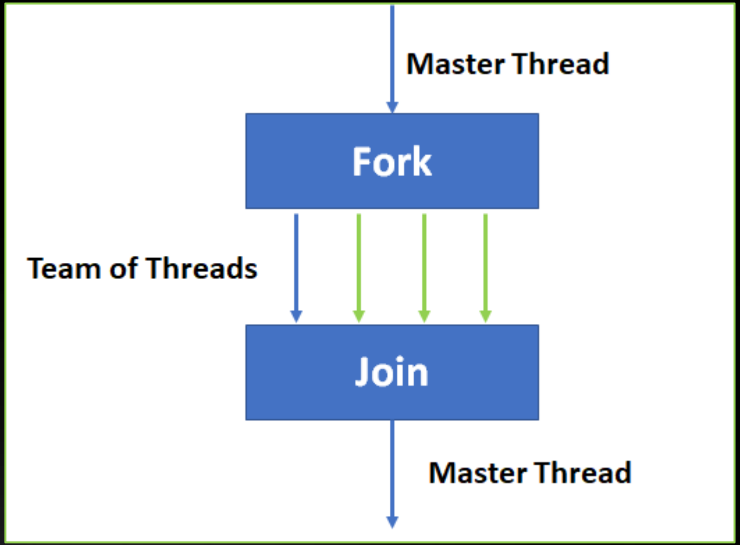
Fork Join Model

Fork Join Model

- OpenMP uses the fork-join model of parallel execution. All OpenMP programs begin as a single process: the master thread. The master thread executes sequentially until the first parallel region construct is encountered.

- FORK: the master thread then creates a team of parallel threads.The statements in the program that are enclosed by the parallel region construct are then executed in parallel among the various team threads.

- JOIN: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread.

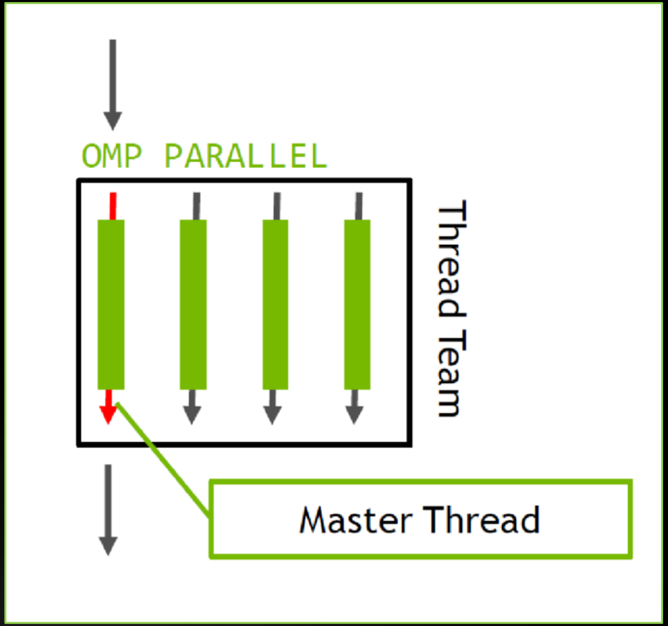


OPENMP

Parallel Region

PARALLEL Directive

- Spawns a team of threads
- Execution continues redundantly on all threads of the team.
- All threads join at the end and the master thread continues execution.



OpenMP Parallel Region

C - Syntax

```
//Include the header file
#include <omp.h>

 main(int argc, char *argv[]) {

 int nthreads;

 /* Fork a team of threads*/
 #pragma omp parallel
   {

   /* Obtain and print thread id */
   printf("Hello World from thread = %d\n", omp_get_thread_num());

   /* Only master thread does this */
   if (omp_get_thread_num() == 0)
     {
     nthreads = omp_get_num_threads();
     printf("Number of threads = %d\n", nthreads);
     }

   }  /* All threads join master thread and terminate */

 }
```

Include Header File

- Spawns parallel region

- Get Thread Id

OpenMP Parallel Region

Fortran - Syntax

```
program hello
  integer :: omp_rank

  !$omp parallel private(omp_rank)

  omp_rank = omp_get_thread_num()
  print *, 'Hello world! by thread ', omp_rank

  !$omp end parallel

end program hello
```

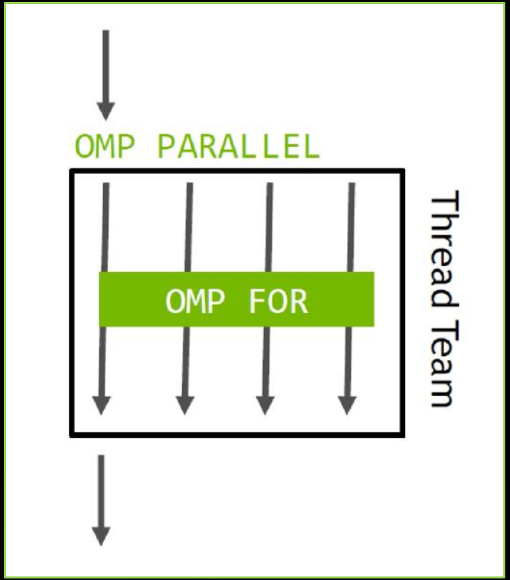
- Spawns parallel region
- Get Thread Id

OPENMP

Worksharing

FOR/DO (Loop) Directive

- Divides ("workshares") the iterations of the next loop across the threads in the team
- How the iterations are divided is determined by a schedule.



C/C++

```
//Create a team of threads
#pragma omp parallel
{
//workshare this loop across those threads.
   #pragma omp for
   for (i=0; i < N; i++)
     c[i] = a[i] + b[i];

   }   /* end of parallel region */
```

Fortran

```
!Create a team of threads
!$omp parallel
!workshare this loop across those threads.
   !$omp for
    do i=1,N
        < loop code >
    end do
!$omp end parallel
```

Example codes

Pi

SAXPY code

https://github.com/yhgon/cuda/blob/master/05.%20OpenAcc/saxpy.cc

Mat mul

https://github.com/yhgon/cuda/blob/master/05.%20OpenAcc/mat_sum_mp.cc

OPENACC

What to expect?

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

OpenACC is...

a directives-based

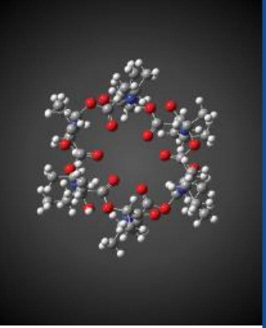
parallel programming model

designed for

performance and **portability.**

Add Simple Compiler Directive

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



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.”

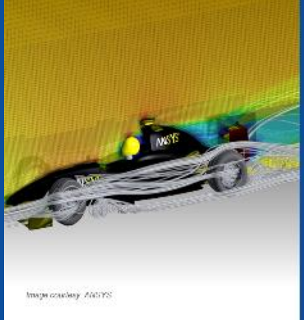
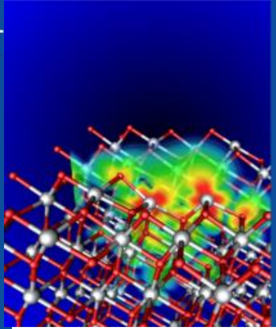


Image courtesy: ANSYS

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.”

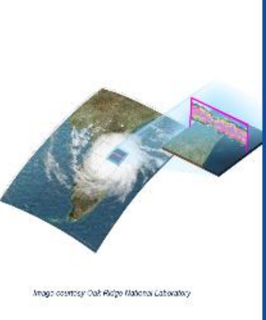


Image courtesy: Oak Ridge National Laboratory

E3SM

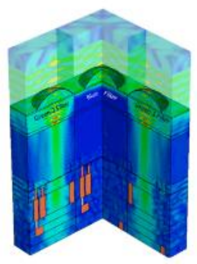
Mark A. Taylor
Multiphysics Applications
Sandia

> “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

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.

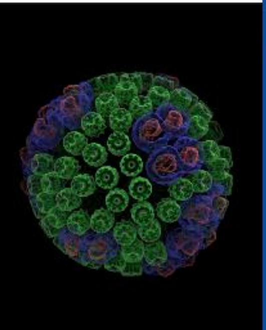
> “Using OpenACC, we've GPU-accelerated the Synopsys TCAD Sentaurus Device EMW simulator to speed up optical simulations of image sensors. GPUs are key to improving simulation throughput in the design of advanced image sensors.”

Image courtesy: NCAR

MPAS-A

Richard Loft
Director, Technology Development
NCAR

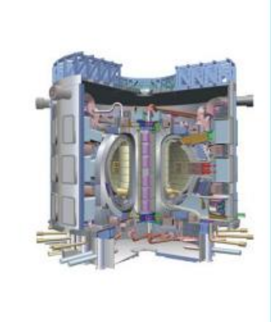
> “Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne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

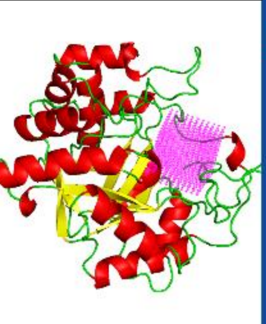
OpenACC
More Science, Less Programming

Map courtesy University of Tokyo

GAMERA

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

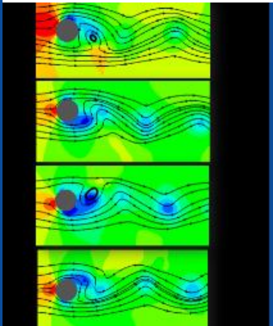
> “With OpenACC and a compute node based on NVIDIA's Tesla P100 GPU, we achieved more than a 14X speed up over a K Computer node running our earthquake disaster simulation code”



SANJEEVINI

Abhilash Jayaraj
Project Scientist
Indian Institute of Technology
New Delhi

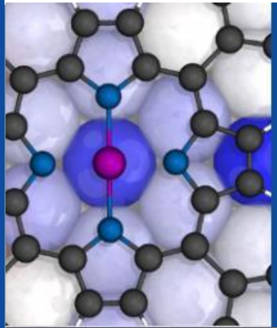
> “In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task.”



IBM-CFD

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

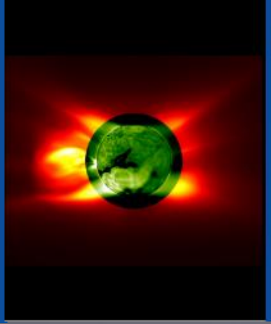
> “OpenACC can prove to be a handy tool for computational engineers and researchers to obtain fast solution of non-linear dynamics problem. In immersed boundary incompressible CFD, we have obtained order of magnitude reduction in computing time by porting several components of our legacy codes to GPU.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

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.

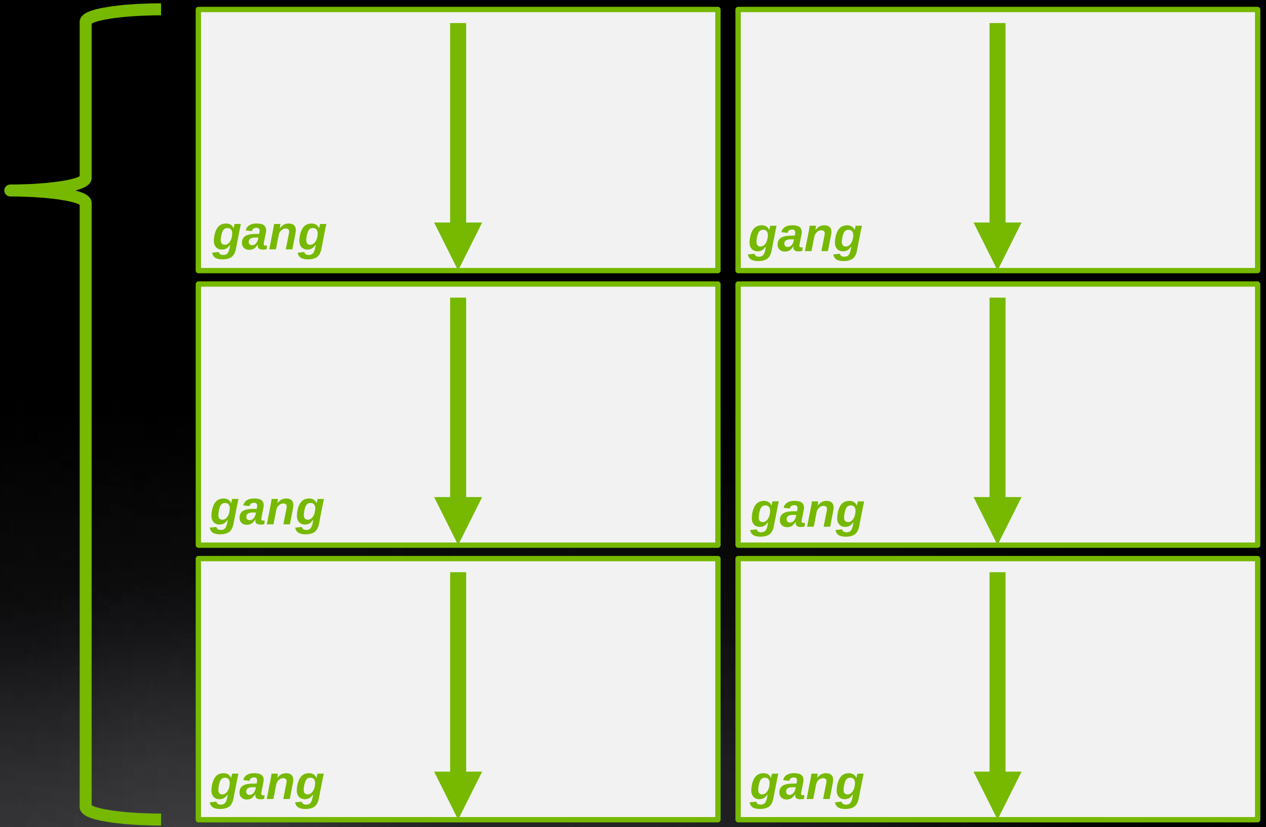
OPENACC PARALLEL DIRECTIVE

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
{
```

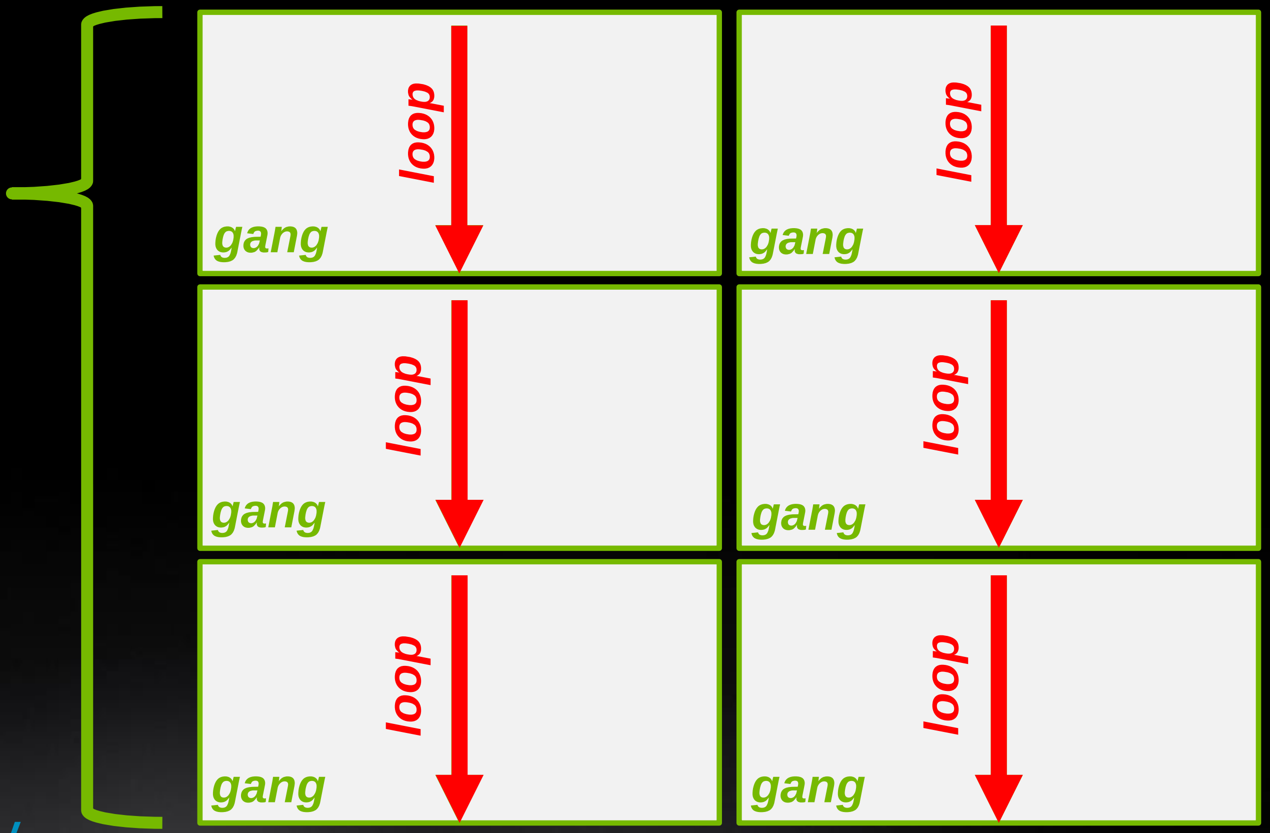
#pragma acc parallel

loop

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

}
```

This loop will be executed redundantly on each gang



OPENACC PARALLEL DIRECTIVE

Expressing parallelism

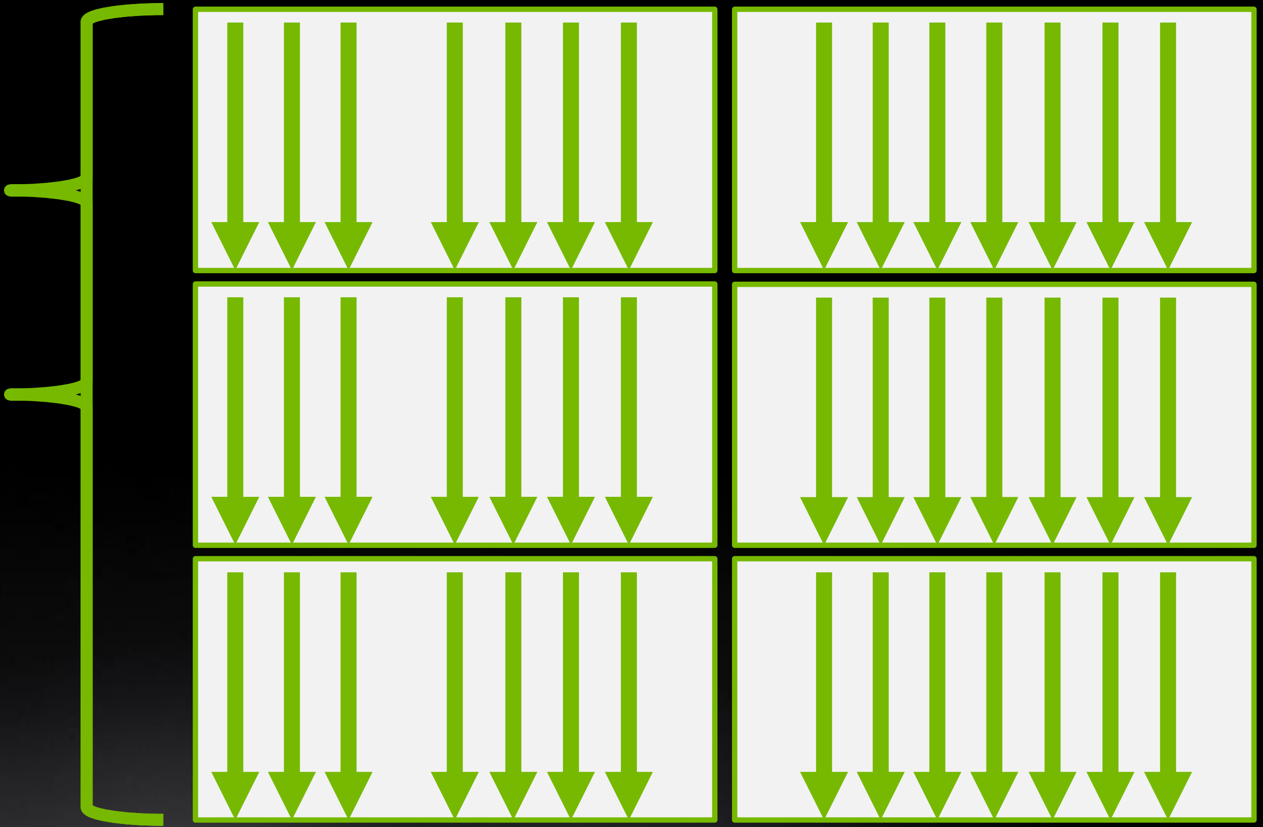
```
#pragma acc parallel
{


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



}
```

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



OPENACC PARALLEL 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

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
```

```
nvfortran –fast –Minfo=accel –ta=tesla:managed main.f90
```

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
```

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

RDF

Pseudo Code -C

```
#pragma acc parallel loop
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);
                 #pragma acc atomic
                 d_g2[ig2] = d_g2[ig2] +1 ;
                 }
        }
    }
}
```

- Parallel Loop construct

- Atomic Construct

RDF

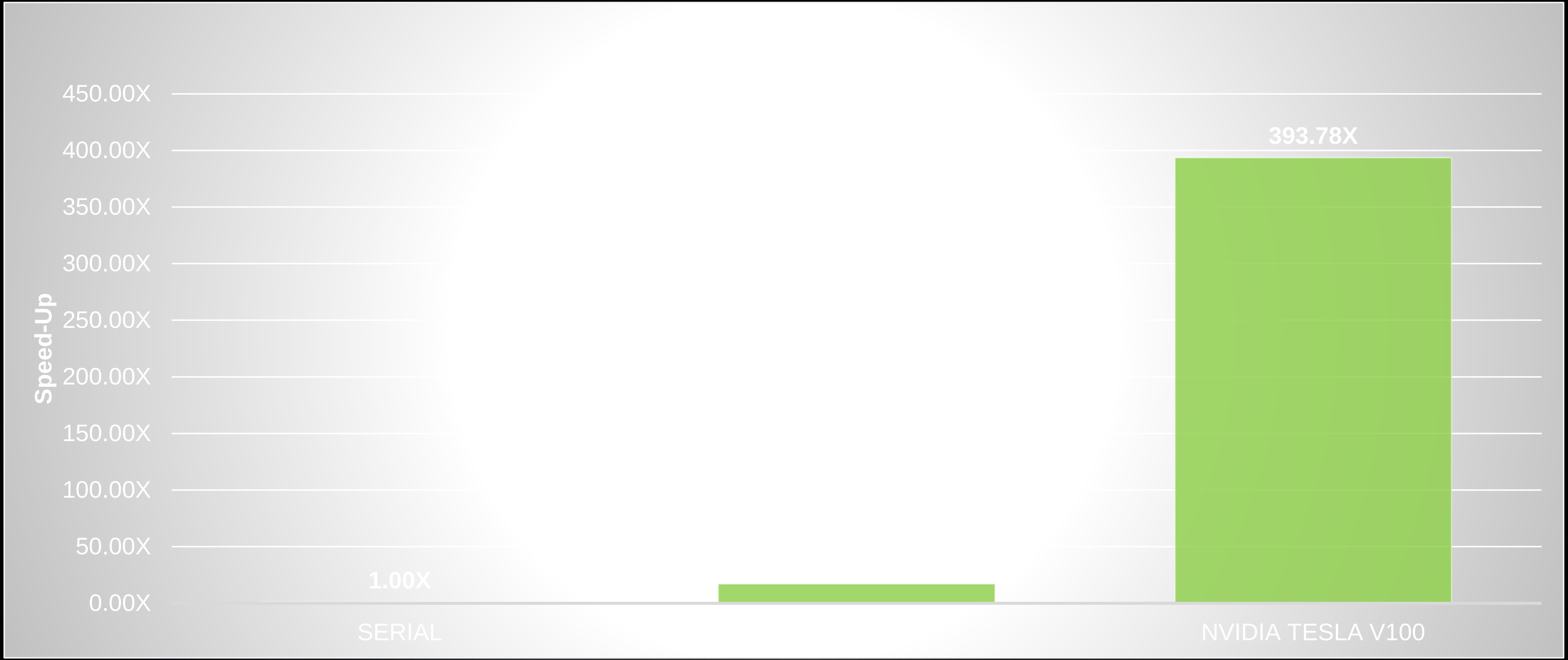
Pseudo Code - Fortran

```
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

OPENACC SPEEDUP

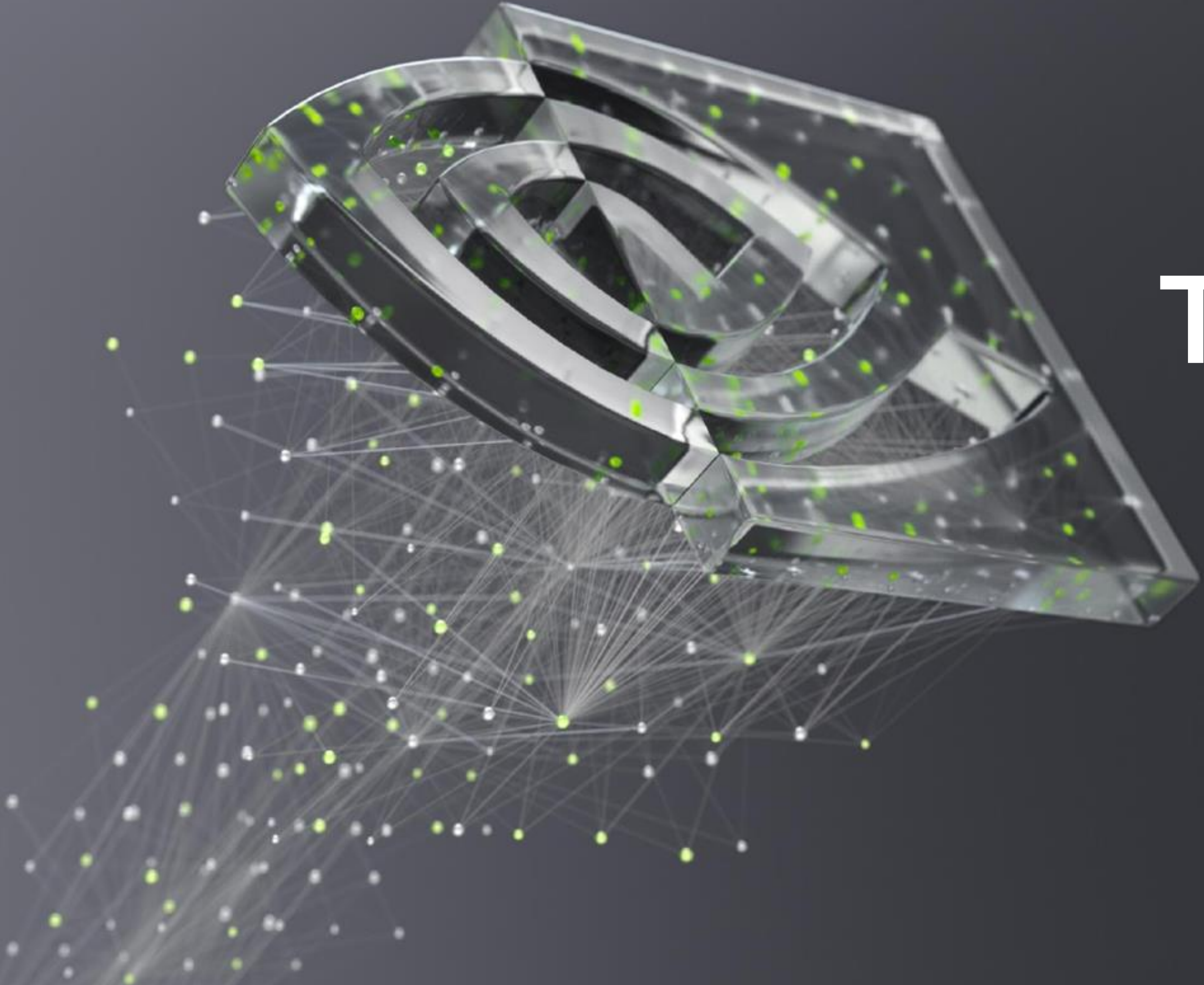


HPC SDK 20.11, NVIDIA Tesla V100, DGX1

REFERENCES

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

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

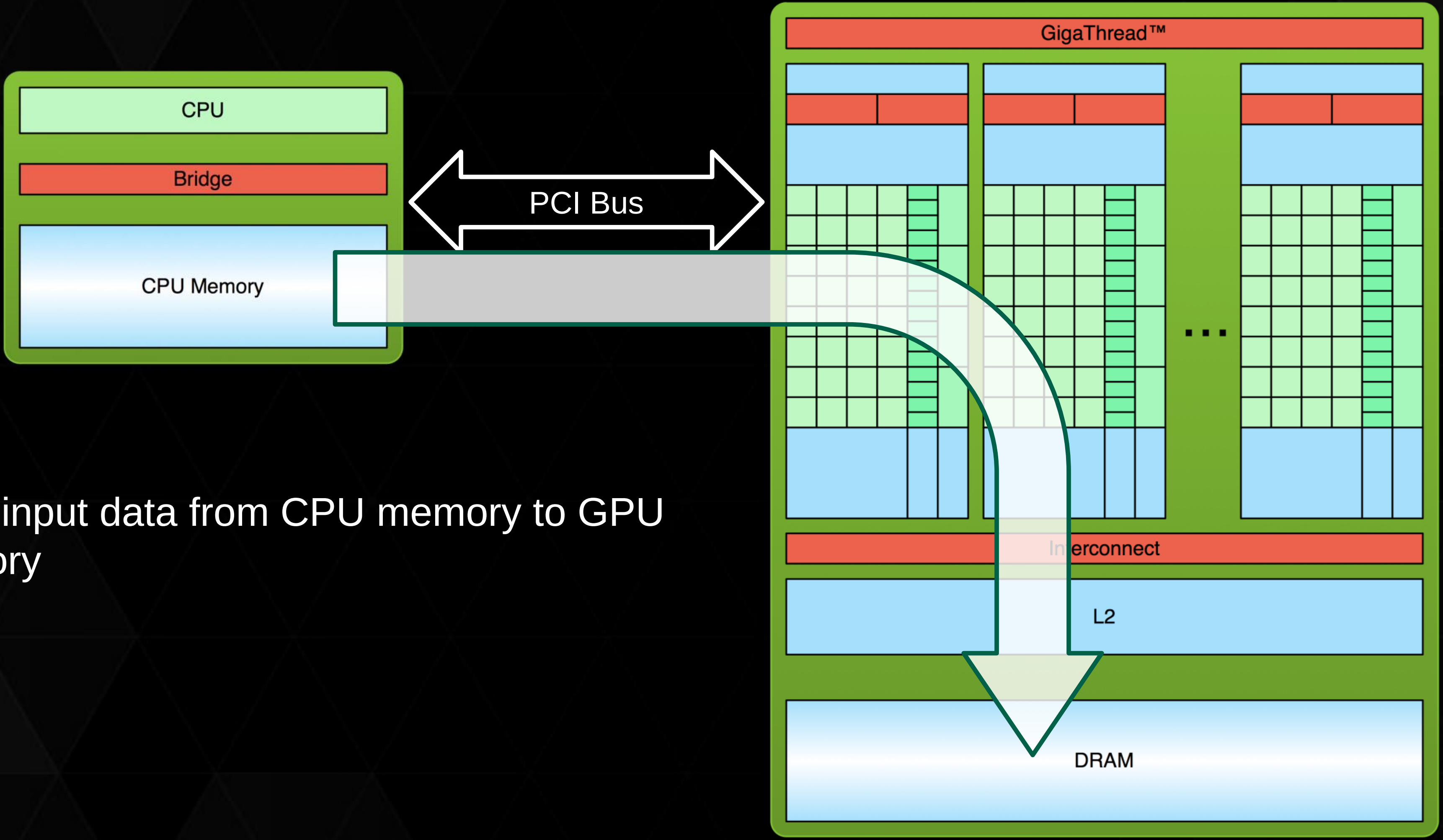


THANK YOU

NVIDIA

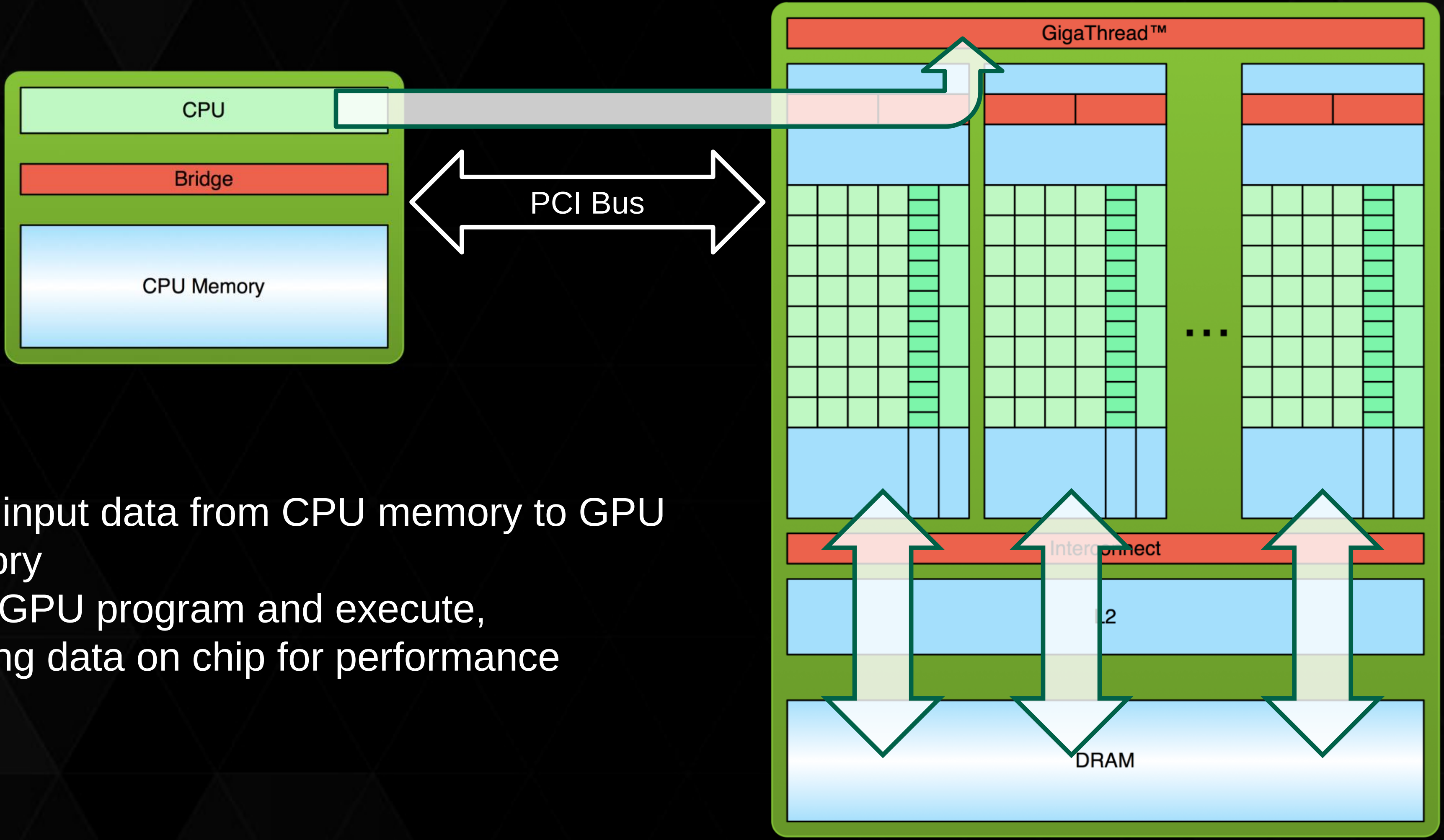
ADDITIONAL EXERCISE CONTENT

PROCESSING FLOW – STEP 1



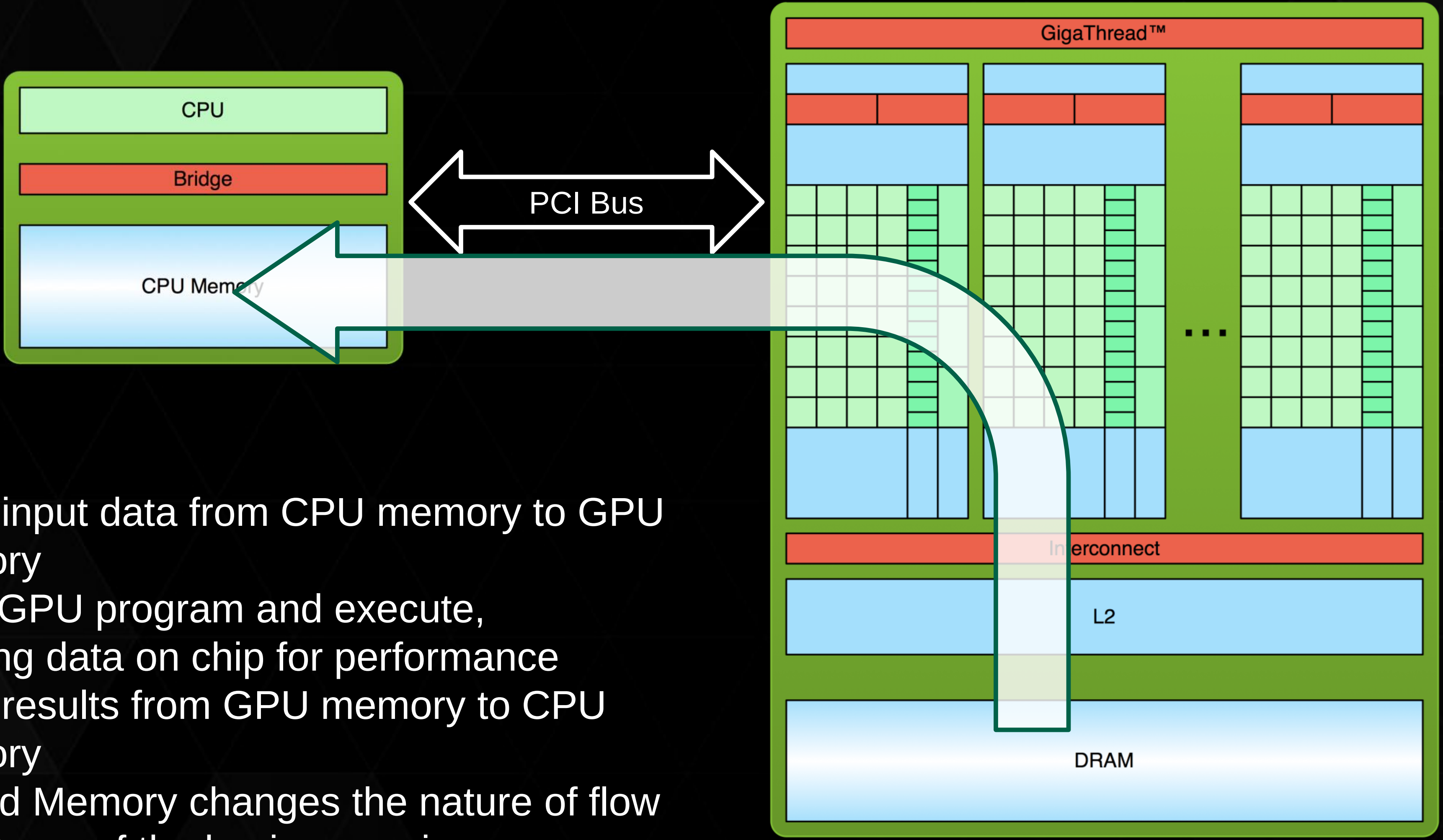
1. Copy input data from CPU memory to GPU memory

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

PROCESSING FLOW – STEP 3

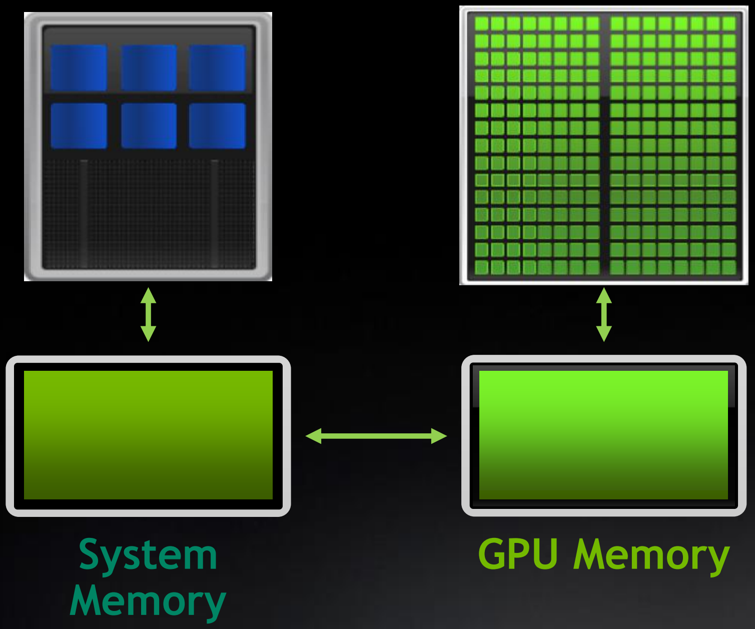


1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory
4. Unified Memory changes the nature of flow
   - Some of the basics remains same

CUDA UNIFIED MEMORY

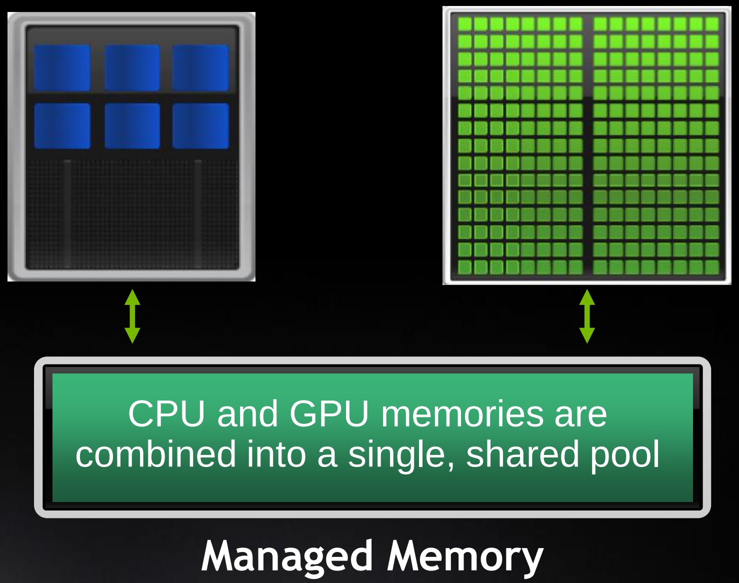
Simplified Developer Effort

Commonly referred to as *"managed memory."*



System Memory

GPU Memory



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

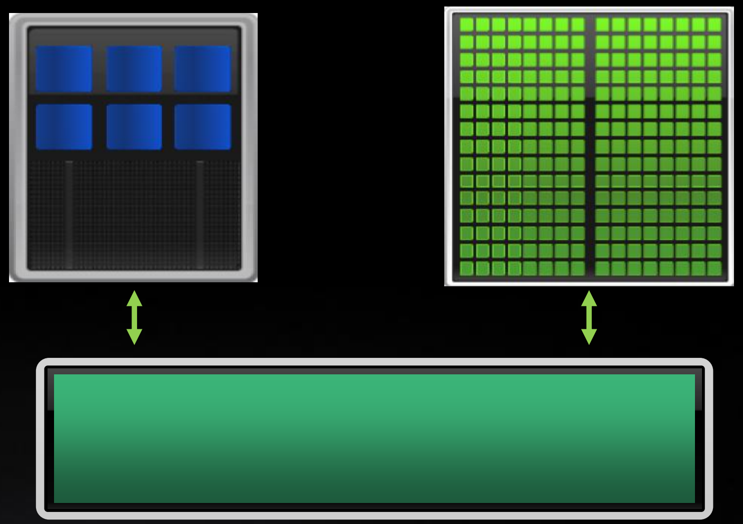
Managed Memory

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



Managed Memory

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

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
```

STRUCTURED DATA DIRECTIVE

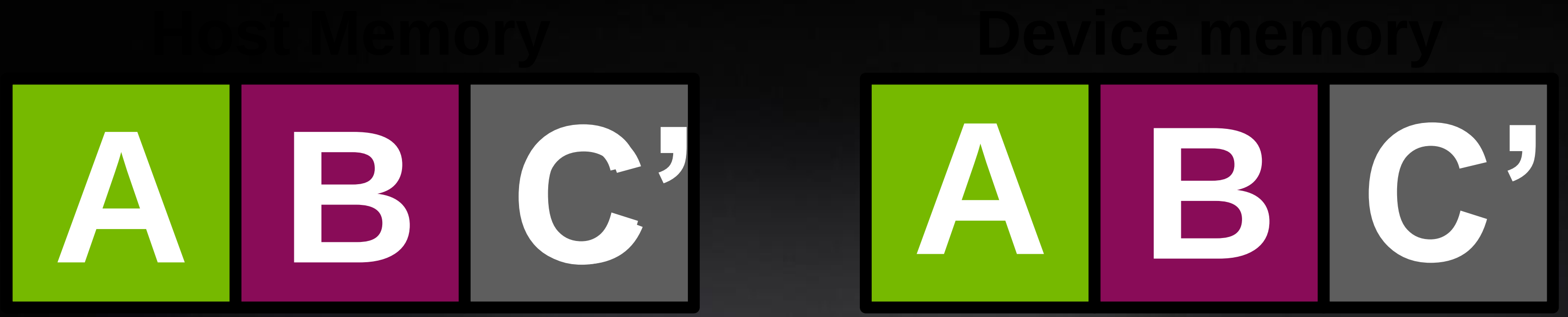
Example

```
#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]



TARGETING THE GPU

openMP GPU directive

OpenMP	OpenACC
omp target teams, distribute, collapse	acc kernels, loop, gang, vector
target data, map	acc data copy
loop	
If	

#pragma parallel for reduction(max:error)

OPENMP

Target Offloading

TARGET Directive

- Offloads execution and associated data from the CPU to the GPU
- The target device owns the data, accesses by the CPU during the execution of the target region are forbidden.
- Data used within the region may be implicitly or explicitly mapped to the device.
- All of OpenMP is allowed within target regions, but only a subset will run well on GPUs.

C/C++

```
#pragma omp target
{
                #pragma omp parallel for reduction(max:error)
                for( int j = 1; j < n-1; j++) {
                }
 }}
```

Fortran

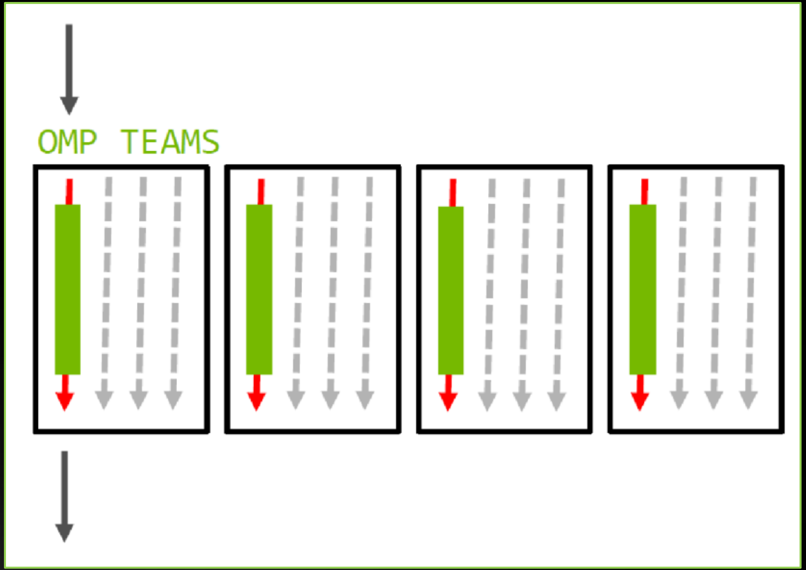
```
!Moves this region of code to the GPU and implicitly maps data.
  !$omp target
     !$omp parallel for
     do i=1,N
        ANew(j) = A (j-1) + A(j+1)
     end do
  !$omp end target
```

OPENMP

Teams

Teams Directive

- To better utilize the GPU resources, use many thread teams via the TEAMS directive.
- Spawns 1 or more thread teams with the same number of threads
- Execution continues on the master threads of each team (redundantly)
- No synchronization between teams

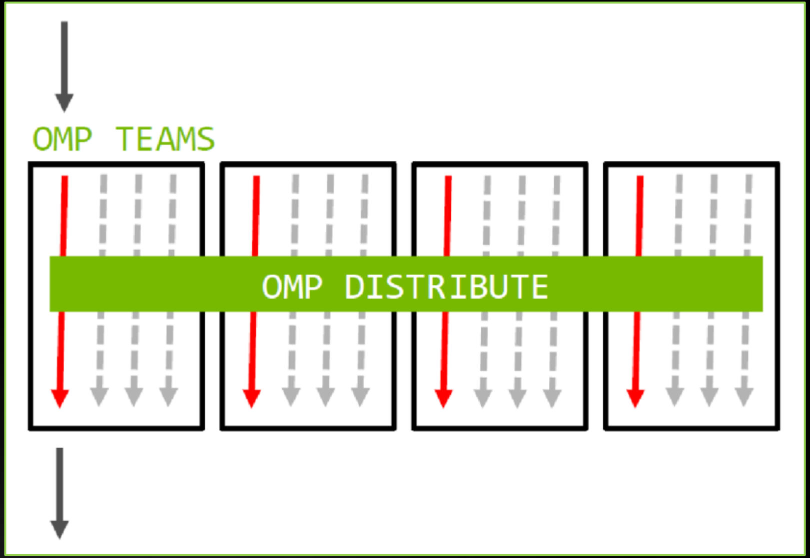


OPENMP

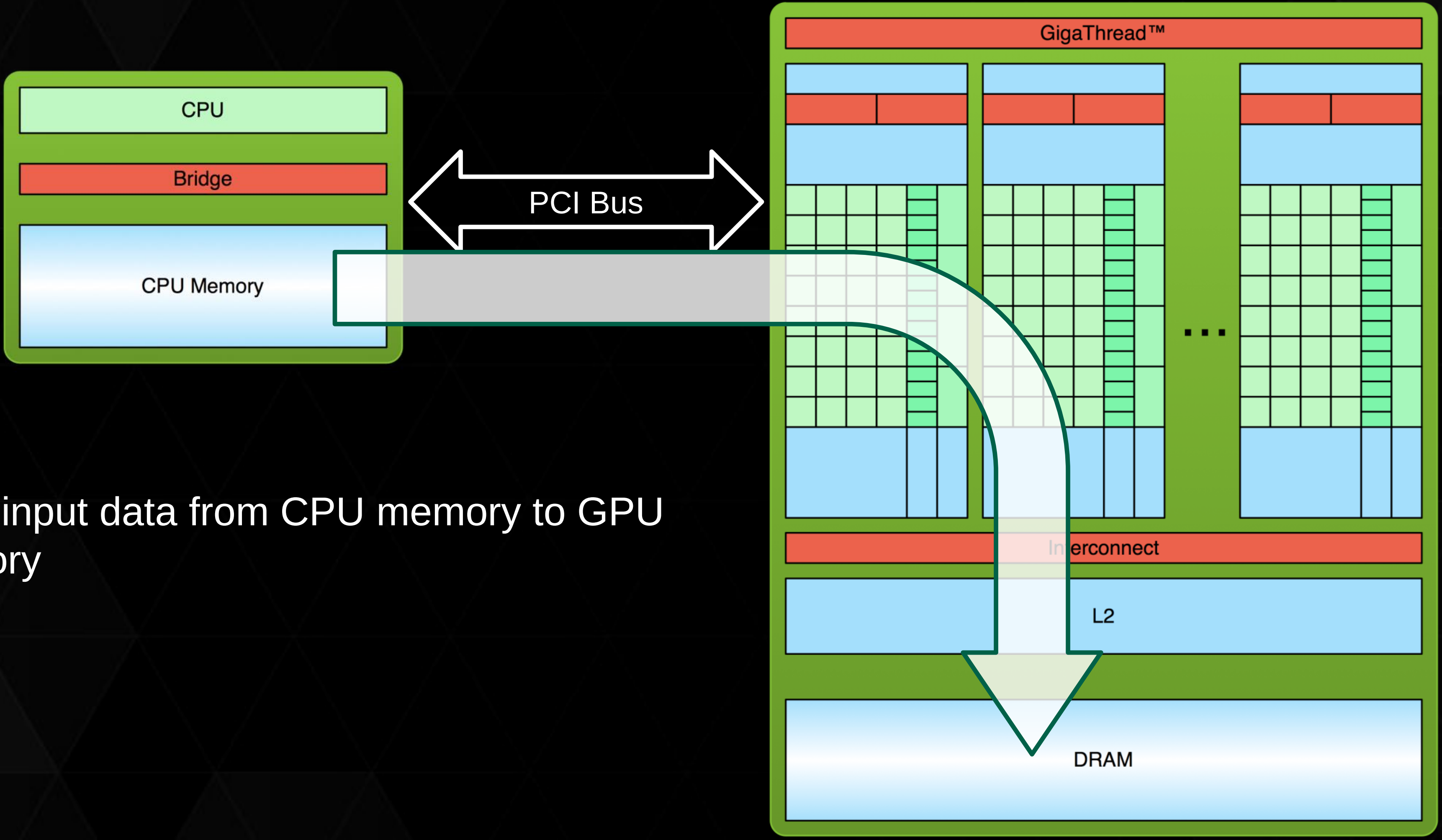
Teams

Distribute Directive

- Distributes the iterations of the next loop to the master threads of the teams.
- Iterations are distributed statically.
- There’s no guarantees about the order teams will execute.
- No guarantee that all teams will execute simultaneously
- Does not generate parallelism/worksharing within the thread teams.

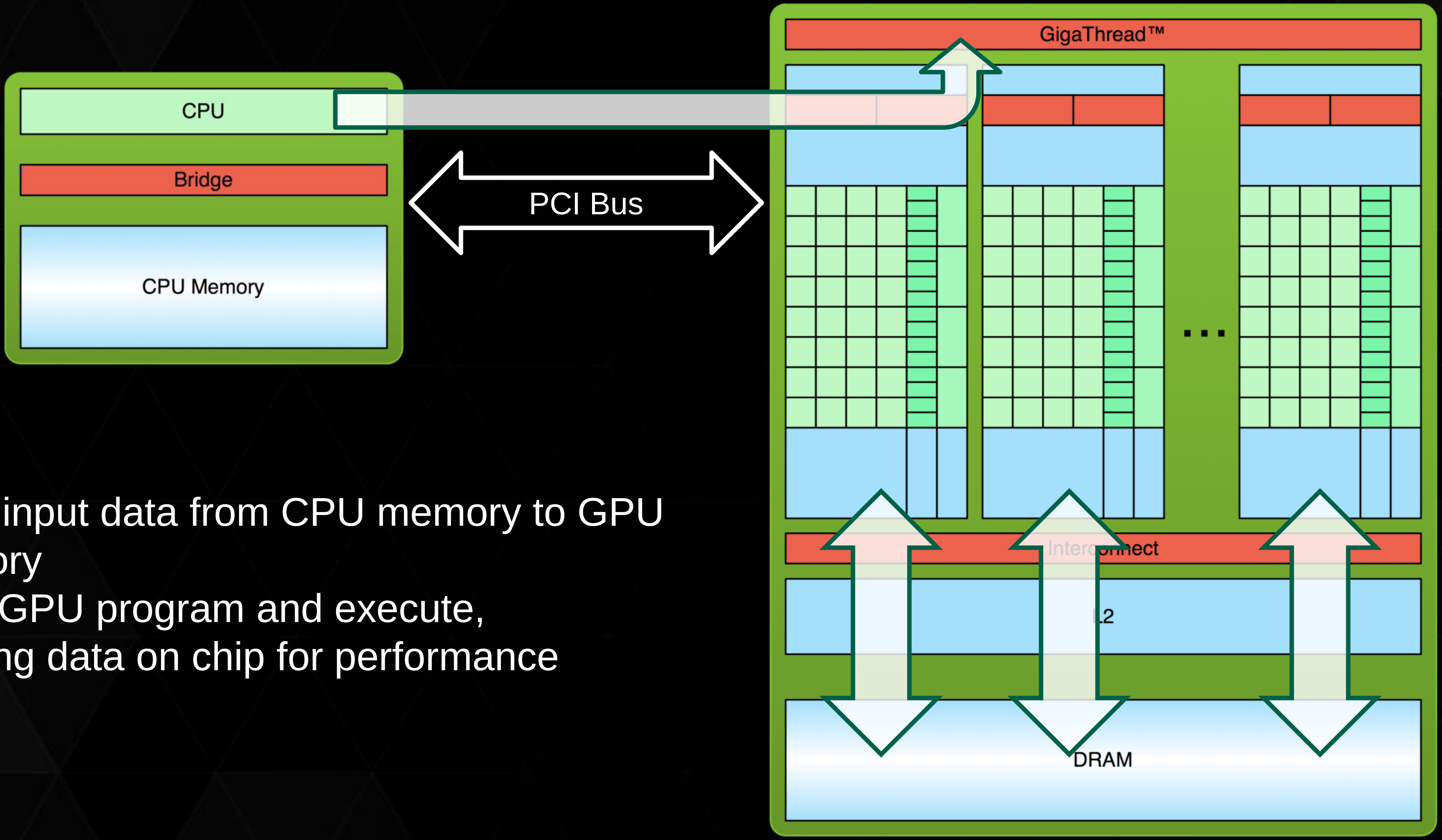


PROCESSING FLOW – STEP 1



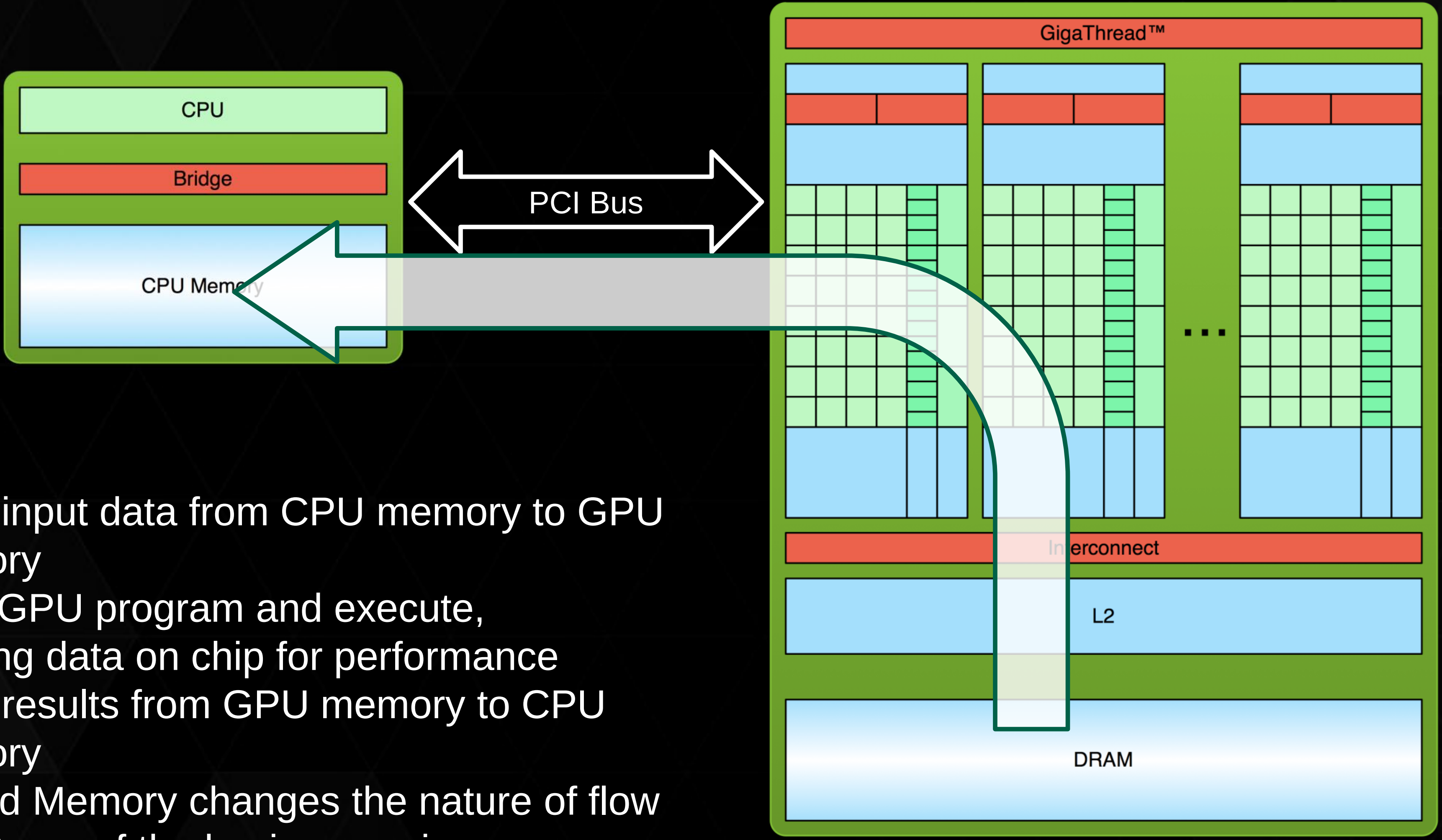
1. Copy input data from CPU memory to GPU memory

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

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

OPENMP

Data Offloading

TARGET Data Directive

- Offloads data from the CPU to the GPU, but not execution
- The target device owns the data, accesses by the CPU during the execution of contained target regions are forbidden.
- Useful for sharing data between TARGET regions

```
#pragma omp target data map(to:A[:n]) map(from:ANew[:n])
  {
    #pragma omp parallel for
    for( int j = 1; j < n-1; j++) {
      ANew[j] = A [j-1] + A[j+1];
    }
  }
```

```
!$omp target data map(to:A(:)) map(from:ANew(:))
    !$omp parallel for
    do i=1,N
      ANew(j) = A (j-1) + A(j+1)
    end do
!$omp end target data
```

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 OpenMP Target Offload onto GPU
  - The command to compile C code is ‘nvc’
  - The command to compile C++ code is ‘nvc++’
  - The command to compile fortran code is ‘nvfortran’

NVIDIA HPC SDK

- The NVIDIA HPC SDK includes the new NVIDIA HPC compiler supporting OpenMP C and Fortran
  - -mp: compiler switch to enable processing of OpenMP directives and pragmas
  - gpu: OpenMP directives are compiled for GPU execution plus multicore CPU fallback; this Beta feature is supported on Linux/x86 for NVIDIA V100 or later GPUs.
  - multicore: OpenMP directives are compiled for multicore CPU execution only; this sub-option is the default.

```
nvc –mp=gpu main.c
```

```
nvfortran –Minfo=mp –mp=gpu main.f90
```

BUILDING THE CODE

-Minfo shows more details

```
Use of loop in Fortran:

!$omp target teams loop
do n1loc_blk = 1, n1loc_blksize
  do igp = 1, ngpown
    do ig_blk = 1, ig_blksize
      do ig = ig_blk, ncouls, ig_blksize
        do n1_loc = n1loc_blk, ntband_dist, n1loc_blksize
          !expensive computation codes
        enddo
      enddo
    enddo
  enddo
enddo
```

```
$ nvfortran test.f90 -mp=gpu -Minfo=mp
42, !$omp target teams loop
  42, Generating "nvkernel_MAIN__F1L42_1" GPU kernel
      Generating Tesla code
    43, Loop parallelized across teams ! blockidx%x
    44, Loop run sequentially
    45, Loop run sequentially
    46, Loop run sequentially
    47, Loop parallelized across threads(128) !
threadidx%x
  42, Generating Multicore code
    43, Loop parallelized across threads
```

RDF

Pseudo Code - C

```
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

RDF

Pseudo Code -C

```
#pragma omp target data map(d_x[0:nconf*numatm],...)
for (int frame=0;frame<nconf;frame++){
  #pragma omp target teams distribute parallel for
  for(int id1=0;id1<numatm;id1++)     {
    for(int id2=0;id2<numatm;id2++)   {
        …
        r=sqrtf(dx*dx+dy*dy+dz*dz);
        if (r<cut) {
          ig2=(int)(r/del);
          #pragma omp atomic
          d_g2[ig2] = d_g2[ig2] +1 ;
          }
        }
  }
}
```

- Target Offload construct
- Map data to GPU
- Distribute Inner Loop

- Atomic Construct

RDF

Pseudo Code – Fortran

```
!$omp target data map(x(:,:), y (:,:), z (:,:), g (:))
do iconf=1,nframes
      if (mod(iconf,1).eq.0) print*,iconf
      !$omp target teams distribute parallel do
private(dx,dy,dz,r,ind)
      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
            !$omp atomic
            g(ind)=g(ind)+1.0d0
          endif
        enddo
      enddo
    enddo
```

- Map data to GPU

- Target Offload construct
- Distribute Inner Loop

- Atomic Construct

PRIVATE CLAUSE

In the C/C++ language it is possible to declare variables inside a lexical scope ; roughly: inside curly braces.

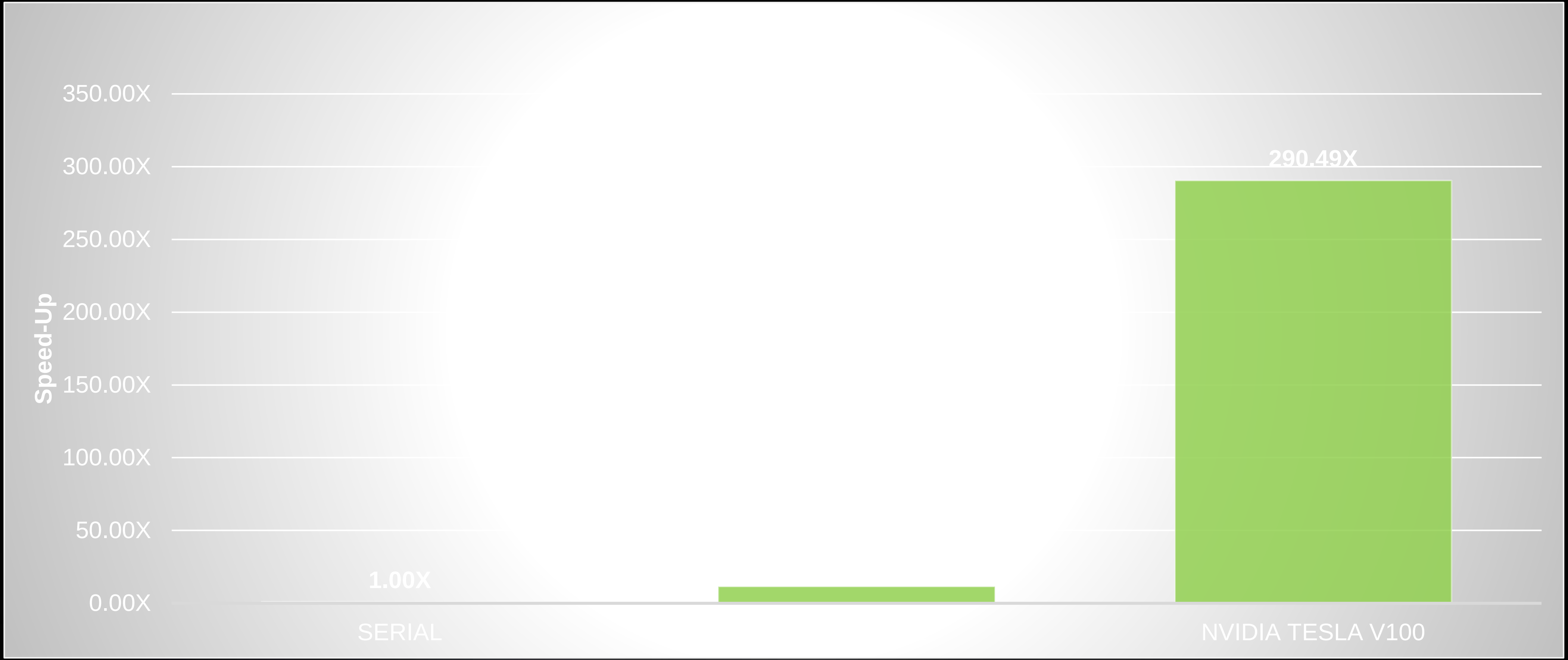
```
int x = 5;
#pragma omp parallel
  {
    int x; x = 3;
    printf("local: x is %d\n", x);
  }
```

This concept extends to OpenMP parallel regions and directives: any variable declared in a block following an OpenMP directive will be local to the executing thread

```
int x = 5;

#pragma omp parallel private(x)
{
    x = x+1; // dangerous
    printf("private: x is %d\n",x);
}
  printf("after: x is %d\n",x); // also
dangerous
```

OPENMP SPEEDUP



HPC SDK 20.11, NVIDIA Tesla V100, DGX1

KNOWN LIMITATIONS

HPC SDK LIMITATION

- Not all functionality associated with loop is supported in the Beta release of OpenMP target offload.
- The compilers support loop regions containing procedure calls as long as the callee does not contain OpenMP directives.

REFERENCES

https://on-demand.gputechconf.com/gtc/2016/presentation/s6510-jeff-larkin-targeting-gpus-openmp.pdf

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