OPENACC FOR SCIENCE

Jeff Larkin, NVIDIA

NERSC GPUs for Science Day 2019

OpenACC
More Science, Less Programming

OPENACC IS…

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

Add Simple Compiler Directive

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

OPENACC MEMBERS

OPENACC COMMUNITY MOMENTUM

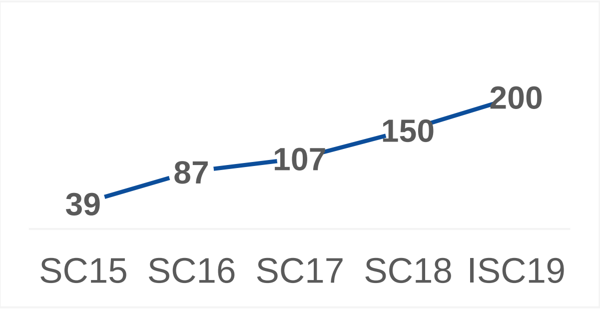
3 OF TOP 5 HPC APPS

Intersect360
RESEARCH

5 OF 13 CAAR CODES

summit

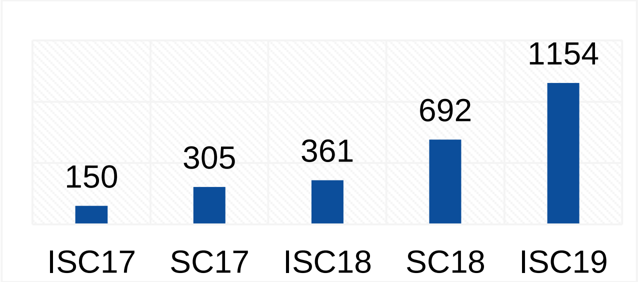
ACCELERATED APPS



SINCE GCC 5.0

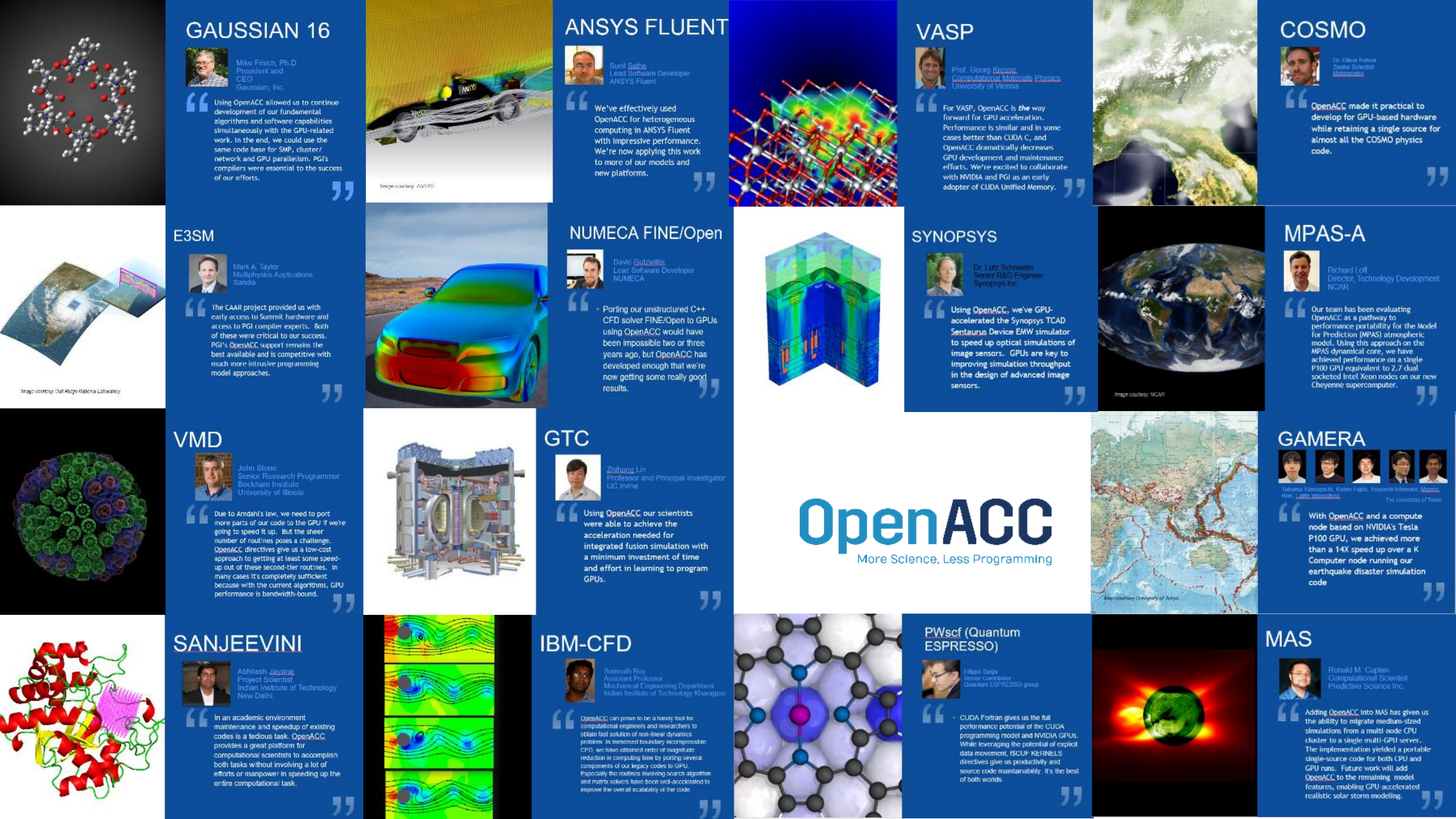
GCC

SLACK MEMBERS



180,000+ DOWNLOADS

PGI®
Community
EDITION



GAUSSIAN 16
Mike Frisch, Ph.D.
President and CEO
Gaussian, Inc.
Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/ network and GPU parallelism. PGI's compilers were essential to the success of our efforts.
Image courtesy: ANSYS
ANSYS FLUENT
Sunil Sathe
Lead Software Developer
ANSYS Fluent
We've effectively used OpenACC for heterogeneous computing in ANSYS Fluent with impressive performance. We're now applying this work to more of our models and new platforms.
VASP
Prof. Georg Kresse
Computational Materials Physics
University of Vienna
For VASP, OpenACC is *the* way forward for GPU acceleration. Performance is similar and in some cases better than CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts. We're excited to collaborate with NVIDIA and PGI as an early adopter of CUDA Unified Memory.
COSMO
Dr. Oliver Fuhrer
Senior Scientist
Meteoswiss
OpenACC made it practical to develop for GPU-based hardware while retaining a single source for almost all the COSMO physics code.
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.
Image courtesy: Oak Ridge National Laboratory
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.
Image courtesy: NCAR
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
Map courtesy University of Tokyo
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.

INTRODUCTION TO OPENACC

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 LOOP DIRECTIVE

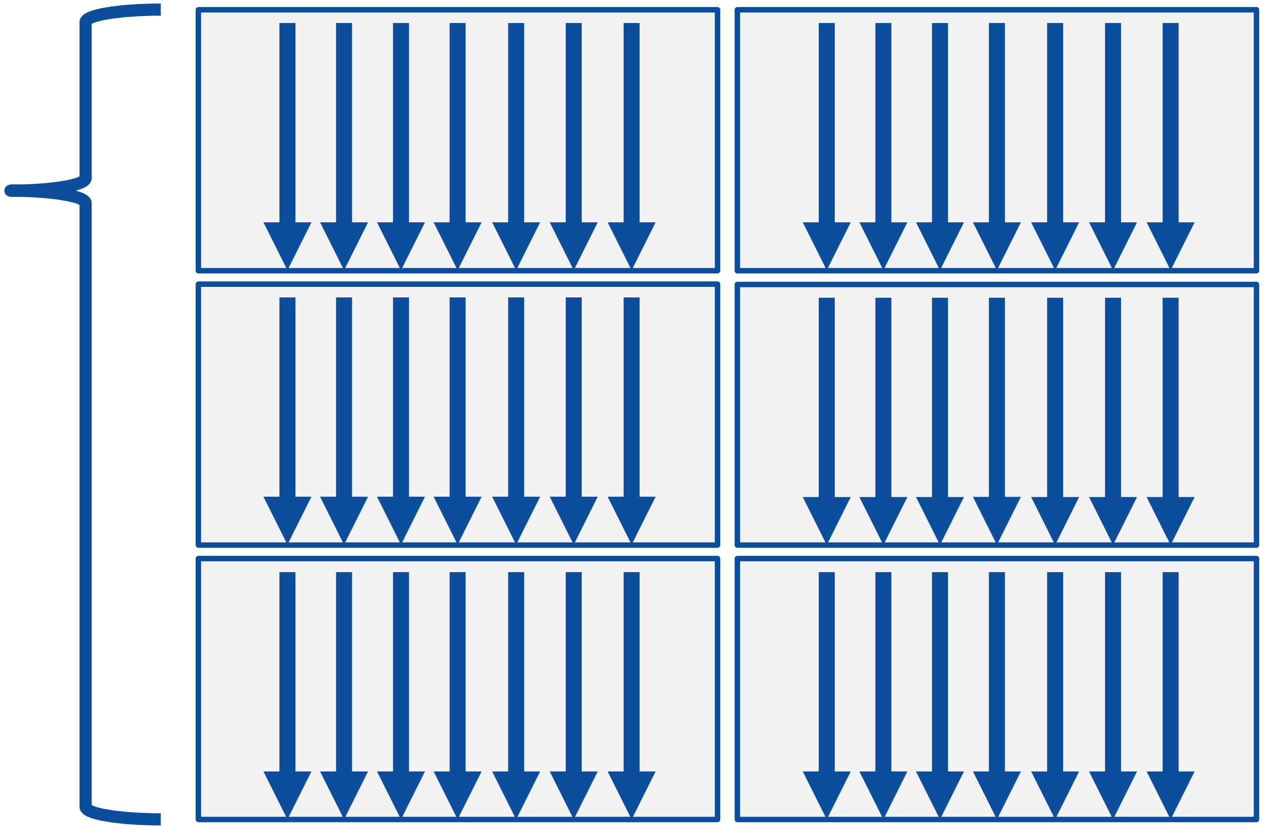
Expressing parallelism

```
#pragma acc parallel loop
{

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

}
```

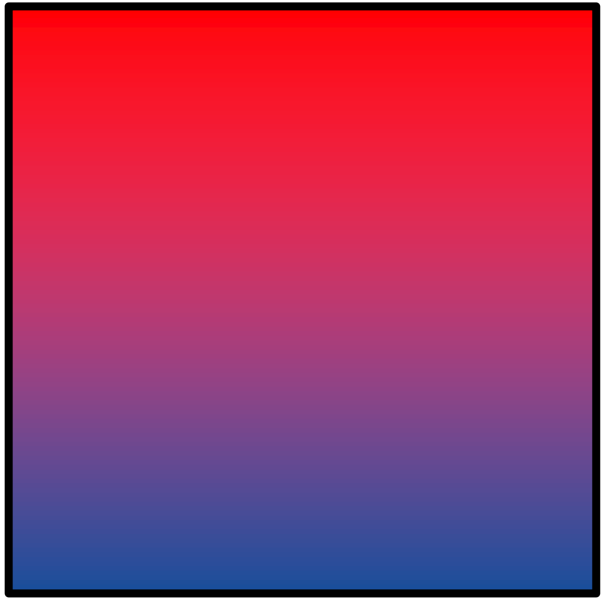
Generate parallelism and parallelize the next loop nest

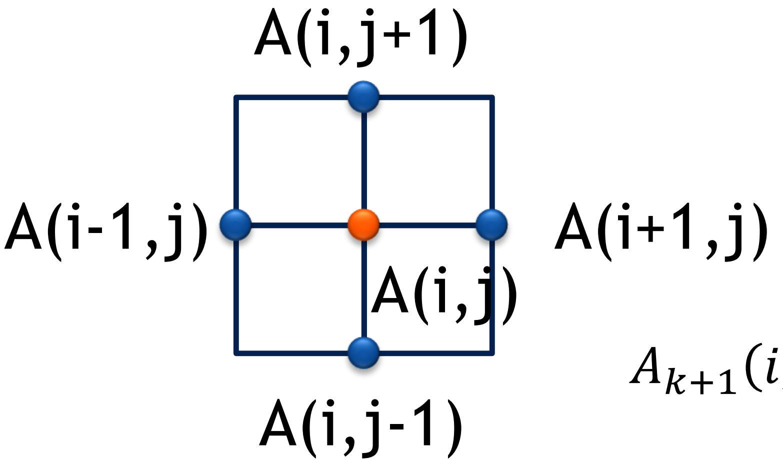


OPENACC EXAMPLE

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

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.

BUILDING THE CODE (GPU)

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

Instruct the compiler to build for an NVIDIA Tesla GPU using "CUDA Managed Memory"*

Print compiler feedback so we can see what it did.

* More on this in a moment

BUILDING THE CODE (GPU)

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

Affirms that a GPU kernel was generated.

Compiler detected possible need to move data and handled it for us.*

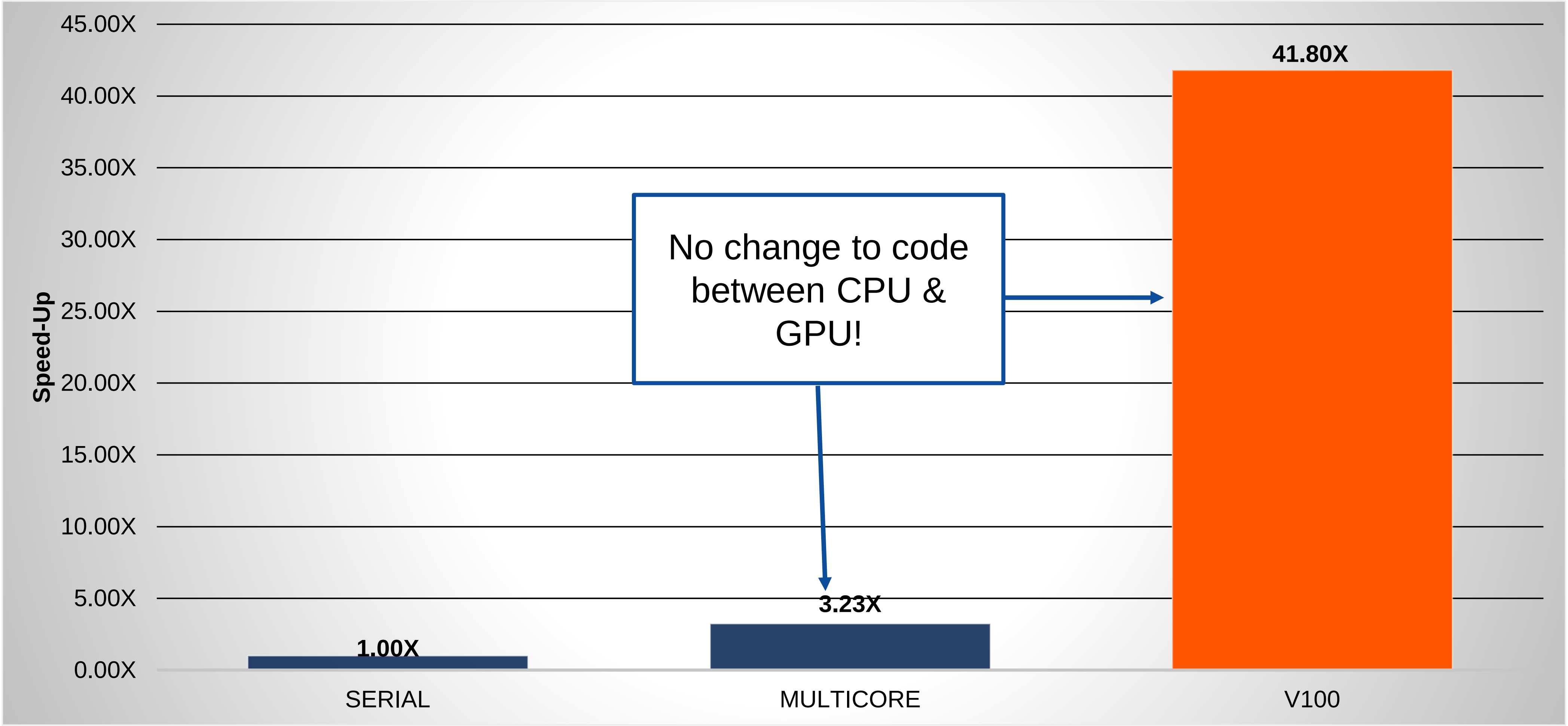
* More on this in a moment

BUILDING THE CODE (MULTICORE)

```
$ pgcc -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
     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
```

Building for a multicore CPU requires changing only a compiler flag.

OPENACC SPEED-UP

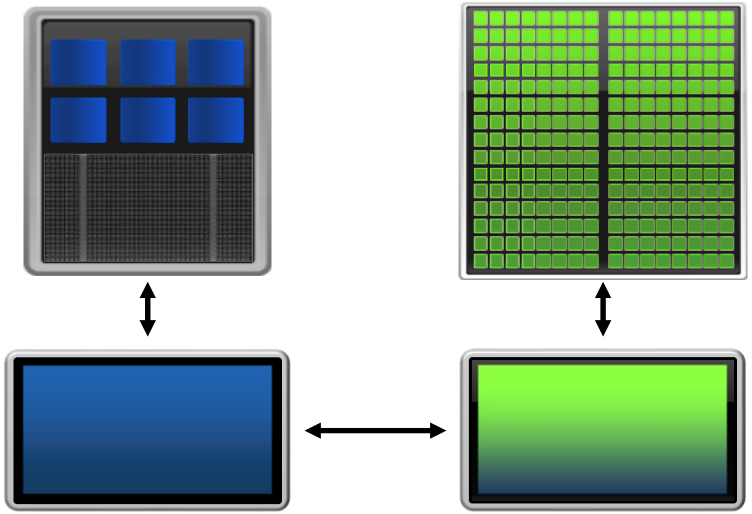


CUDA UNIFIED MEMORY

Simplified Developer Effort

Commonly referred to as
“managed memory.”

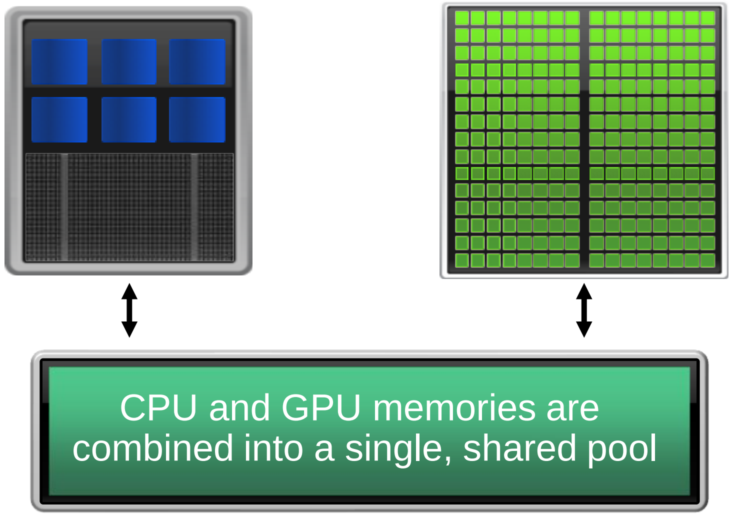
Without Managed Memory



System Memory

GPU Memory

With Managed Memory



Managed Memory

CUDA MANAGED MEMORY

- 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***
- But, the programmer can usually do better by explicitly managing the data movement.

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

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

BUILDING THE CODE (W/O MANAGED MEMORY)

```
$ pgcc -fast -ta=tesla -Minfo=accel laplace2d_uvm.c
PGC-S-0155-Compiler failed to translate accelerator region (see -Minfo messages):
Could not find allocated-variable index for symbol (laplace2d_uvm.c: 63)
PGC-S-0155-Compiler failed to translate accelerator region (see -Minfo messages):
Could not find allocated-variable index for symbol (laplace2d_uvm.c: 74)
main:
     63, Accelerator kernel generated
         Generating Tesla code
         63, Generating reduction(max:error)
         64, #pragma acc loop gang /* blockIdx.x */
         66, #pragma acc loop vector(128) /* threadIdx.x */
     64, Accelerator restriction: size of the GPU copy of Anew,A is unknown
     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 */
     75, Accelerator restriction: size of the GPU copy of Anew,A is unknown
     77, Loop is parallelizable
```

* “managed” keyword removed from tesla target, *fails to build*

OPENACC DATA MANAGEMENT

OPENACC DATA DIRECTIVE

Definition

- The data directive defines a lifetime for data on the device
- During the region data should be treated as owned by the accelerator
- Data clauses allow the programmer to control the allocation and movement of data
- When memory is shared, regions may be ignored

```
#pragma acc data clauses
{

  < Sequential and/or Parallel code >

}
```

```
!$acc data clauses

  < Sequential and/or Parallel code >

!$acc end data
```

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

OPTIMIZED DATA MOVEMENT

```
#pragma acc data copy(A[:n*m]) create(Anew[:n*m])
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++;
}
```

Copy A to/from the accelerator only when needed.

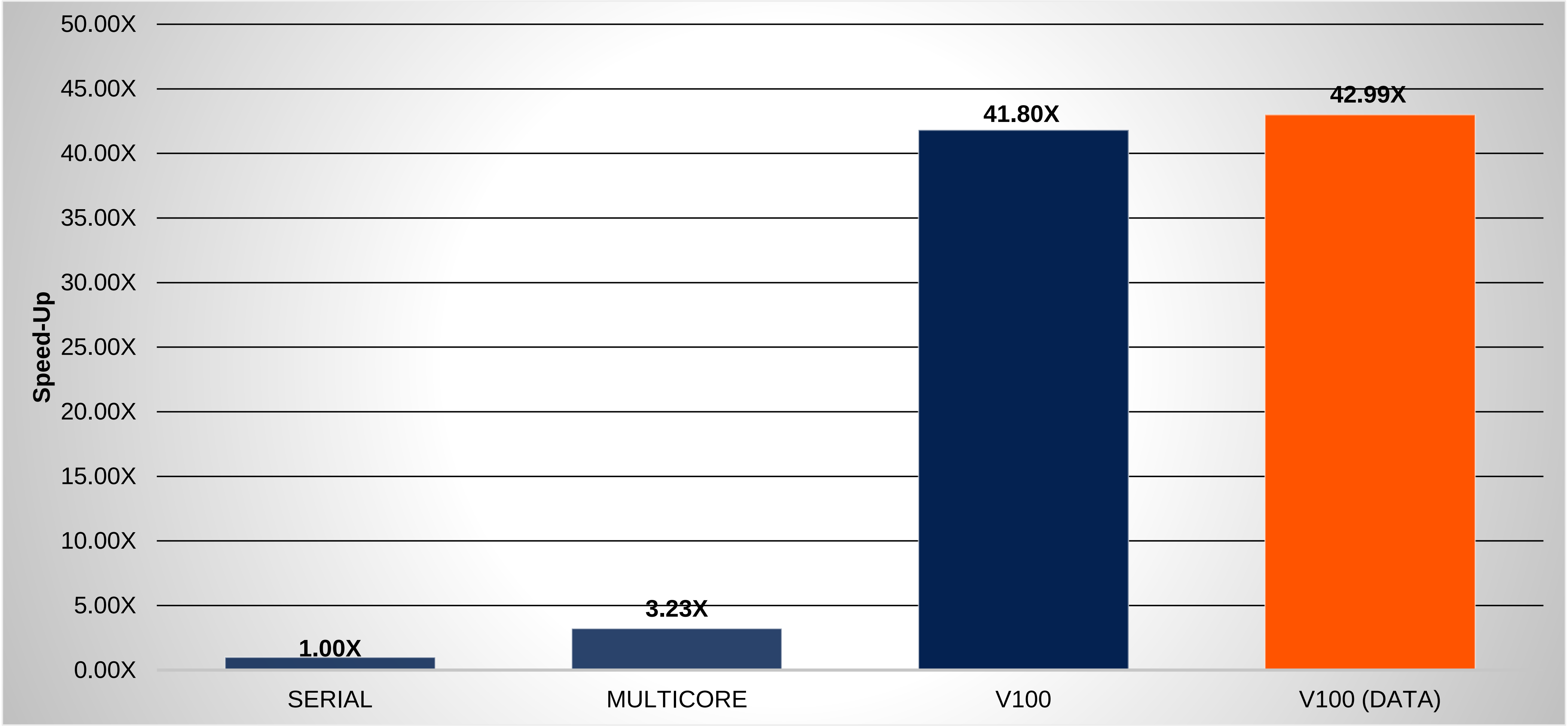
Create temporary space for Anew

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



OPENACC CASE STUDIES

OPENACC CASE STUDIES

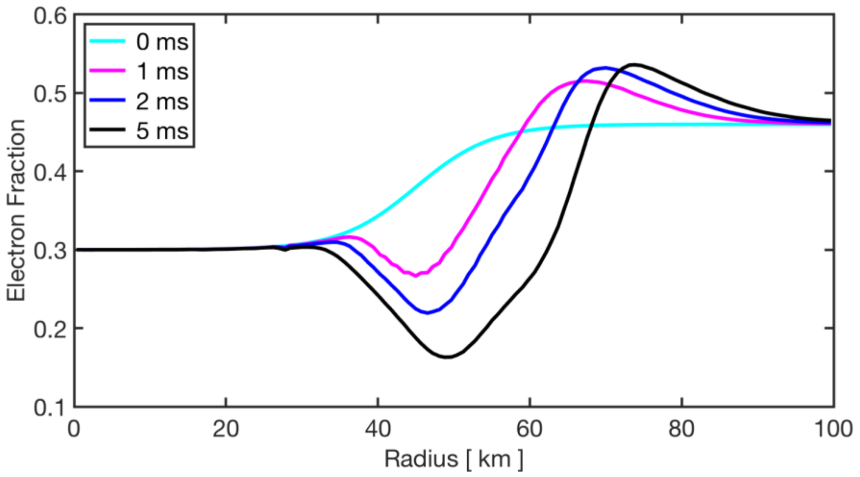
Real life lessons learned

- Thornado : OpenACC Interoperability with math libraries (Collaboration with Austin Harris, ORNL)
- E3SM : OpenACC with Unified Memory for Fortran Derived Types (Collaboration with Matt Norman, ORNL)

THORNADO

THORNADO – NEUTRINO TRANSPORT

- Neutrino transport problem mimicking core-collapse supernova
- DG-IMEX scheme
- Energy discretization: 32 points with $\varepsilon \in [0,300]$ MeV
- Spatial discretization: 12^3 points with $(x, y, z) \in [0,100]$
- Deleptonization Wave test
- Mock initial CCSN profile
- Initial neutrino spectrum from Fermi-Dirac distribution



Courtesy of Austin Harris, ORNL

GPU CODE TRANSFORMATION EXAMPLE

Original CPU code

```
DO i6=1,n5; …; DO i2=1,n2
 ! Calculate G(:,i2,i3,i4,i5,i6)            8 DGEMV
 DO i1=1,n1
  ! Calculate V(:,i1,i2,i3,i4,i5,i6)
  ! Calculate S(:,i1,i2,i3,i4,i5,i6)        8 DGEMV
  ! Calculate U(:,i1,i2,i3,i4,i5,i6)        12 DGEMV
 END DO
END DO; …; END DO
```

OpenACC code

```
! Calculate G                               8 DGEMM
! Calculate S                               2 DGEMM
!$ACC PARALLEL LOOP GANG VECTOR COLLAPSE(7)
DO i6=1,n6; …; DO i0=1,n0
 ! Calculate V(i0,i1,i2,i3,i4,i5,i6)
END DO; …; END DO
! Calculate U                               3 DGEMM
```

Courtesy of Austin Harris, ORNL

OPENACC/CUBLAS INTEROPERABILITY (DNRM2)

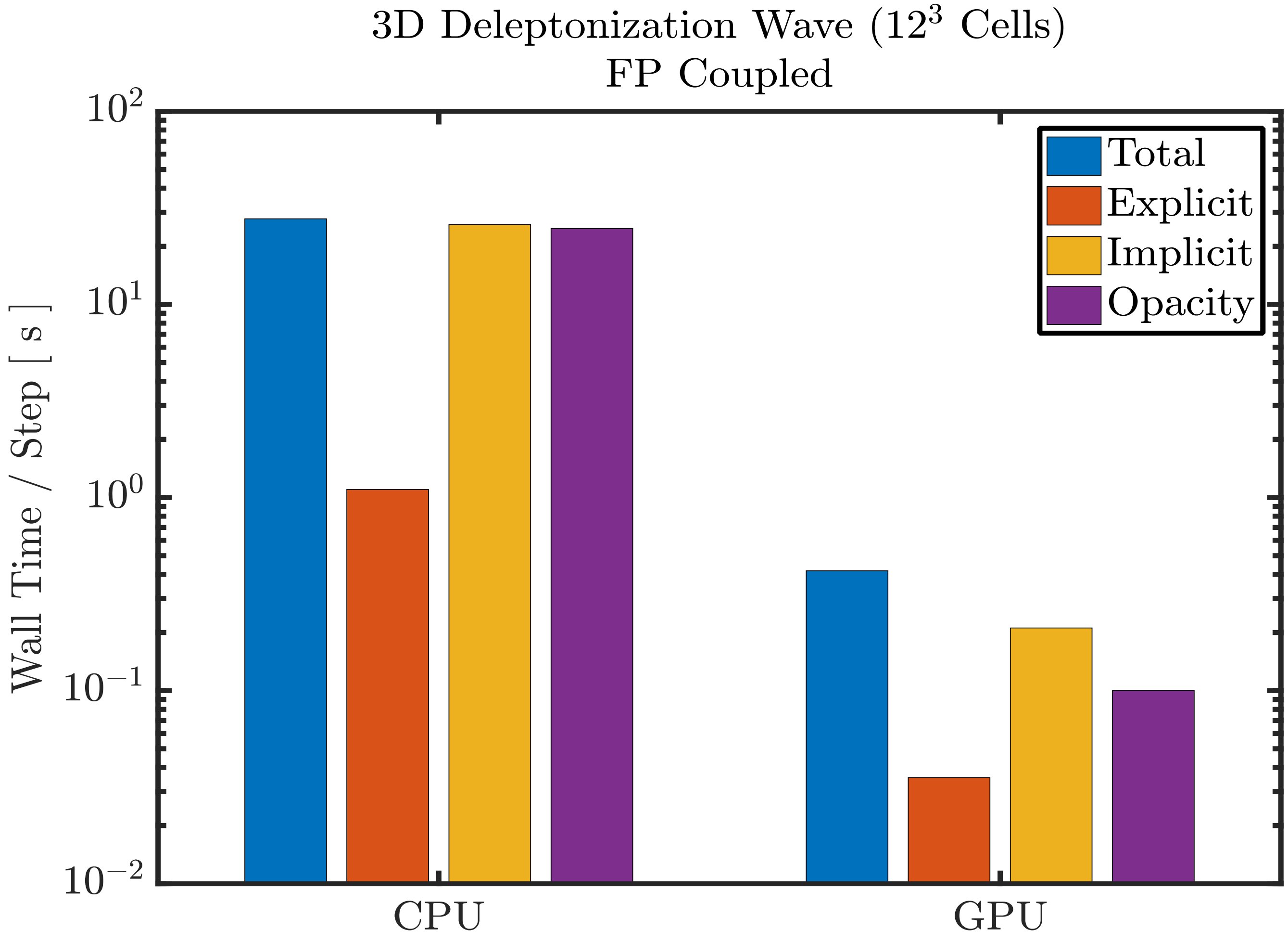
```
MODULE LinearAlgebraModule
USE DeviceModule
…
SUBROUTINE VectorNorm2( n, x, incx, xnorm )
REAL(8), DIMENSION(:), POINTER :: px
TYPE(C_PTR)               :: hx, dx
LOGICAL                   :: data_on_device
data_on_device = .false.
sizeof_x = n * c_sizeof(0.0d0)
px(1:n) => x(1:n)
hx = C_LOC( px )
data_on_device = device_is_present( hx, mydevice, sizeof_x )
IF ( data_on_device ) THEN
#if defined(THORNADO_OACC)
  !$ACC HOST_DATA USE_DEVICE( px )
#endif
  dx = C_LOC( px )
#if defined(THORNADO_OACC)
  !$ACC END HOST_DATA
#endif
  ierr = cublasDnrm2_v2( cublas_handle, n, dx, incx, xnorm )
ELSE
  xnorm = DNRM2( n, x, incx )
END IF
END SUBROUTINE
```

```
MODULE OpenACCModule
…
INTEGER(C_INT) FUNCTION acc_is_present(hostptr,bytes) &
  BIND(C,NAME="acc_is_present" )
USE, INTRINSIC :: iso_c_binding
TYPE(C_PTR), VALUE :: hostptr
INTEGER(C_SIZE_T), VALUE :: bytes
END FUNCTION
```

```
MODULE DeviceModule
USE OpenACCModule
LOGICAL FUNCTION device_is_present( hostptr, device, bytes )
TYPE(C_PTR), INTENT(in) :: hostptr
INTEGER, INTENT(in) :: device
INTEGER(C_SIZE_T), INTENT(in) :: bytes
#if defined(THORNADO_OACC)
device_is_present = ( acc_is_present( hostptr, bytes ) > 0 )
#else
device_is_present = .false.
#endif
END FUNCTION
```

Courtesy of Austin Harris, ORNL

GPU BENCHMARKS



Courtesy of Austin Harris, ORNL

PROGRAMMING MODEL COMPARISON

Compiler	Offload Model	T_{CPU}	T_{GPU}	Speedup
PGI v19.4	OpenACC v2.7	27.8 sec/step	0.42 sec/step	67X
XL v16.1.1	OpenMP v4.5	25.6 sec/step	0.99 sec/step	26X

Courtesy of Austin Harris, ORNL

E3SM

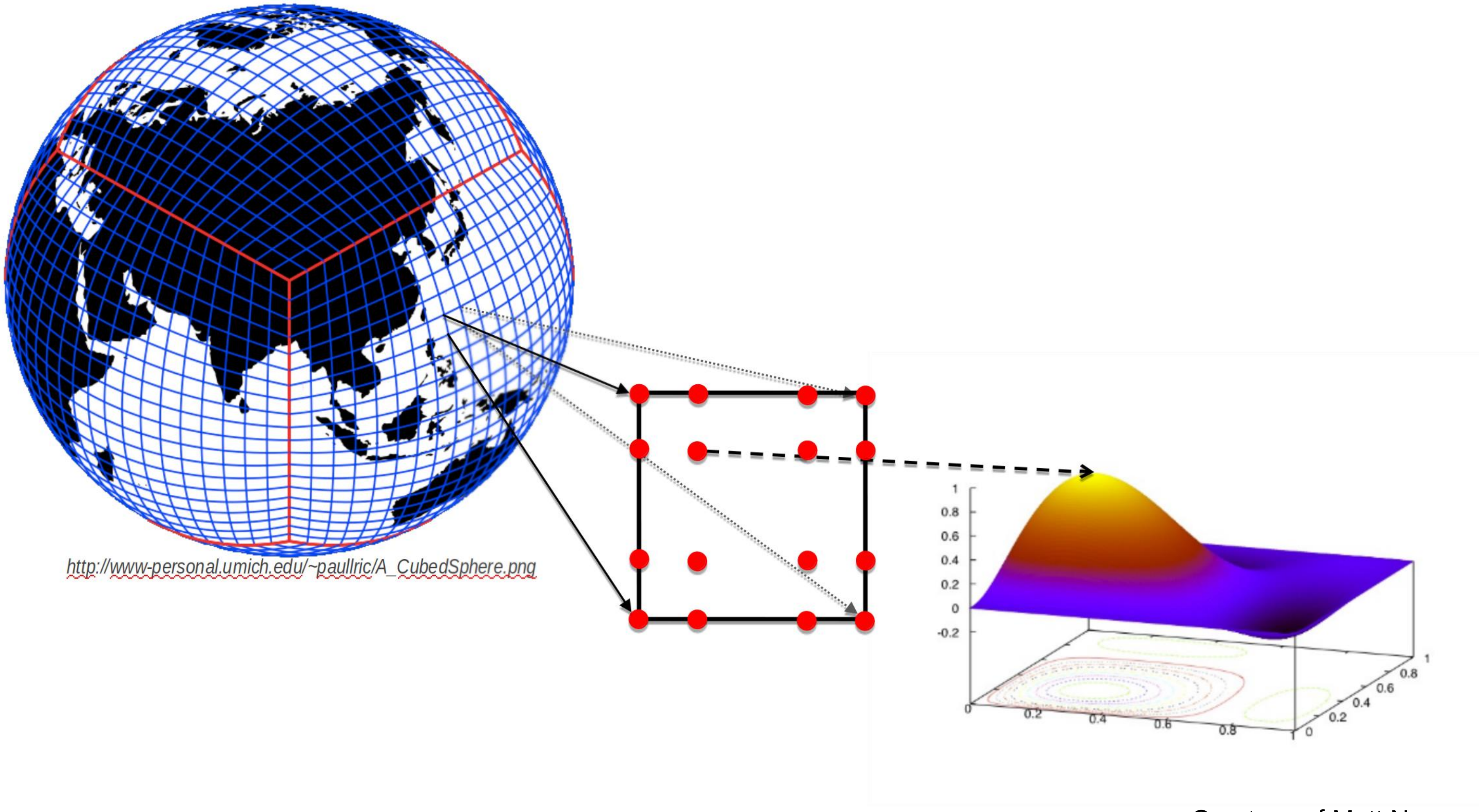
The Energy Exascale Earth System Model (E3SM)

- The U.S. DOE's high-resolution climate model
- Coupling of five components: (1) Atmosphere, (2) Ocean, (3) Land Surface, (4) Sea Ice, and (5) Land Ice
- Atmospheric model is most expensive component
  - "Cubed-sphere" non-orthogonal grid
  - Spectral Element method (continuous Galerkin, time-explicit)

- Because of throughput requirements, hi-res climate has very little work per node to accelerate

Courtesy of Matt Norman, ORNL

The Energy Exascale Earth System Model (E3SM)



http://www-personal.umich.edu/~paullric/A_CubedSphere.png

Courtesy of Matt Norman, ORNL

COMPLEX DATA TYPES

```
type crm_rad_type
   ! Radiative heating
   real(crm_rknd), pointer :: qrad(:,:,:,:)

   ! Quantities used by the radiation code. Note that these are strange in that they are
   ! time-averages, but spatially-resolved.
   real(crm_rknd), pointer :: temperature(:,:,:,:) ! rad temperature
   real(crm_rknd), pointer :: qv (:,:,:,:) ! rad vapor
   real(crm_rknd), pointer :: qc (:,:,:,:) ! rad cloud water
   real(crm_rknd), pointer :: qi (:,:,:,:) ! rad cloud ice
   real(crm_rknd), pointer :: cld(:,:,:,:) ! rad cloud fraction

   ! Only relevant when using 2-moment microphysics
   real(crm_rknd), pointer :: nc(:,:,:,:) ! rad cloud droplet number (#/kg)
   real(crm_rknd), pointer :: ni(:,:,:,:) ! rad cloud ice crystal number (#/kg)
   real(crm_rknd), pointer :: qs(:,:,:,:) ! rad cloud snow (kg/kg)
   real(crm_rknd), pointer :: ns(:,:,:,:) ! rad cloud snow crystal number (#/kg)
end type crm_rad_type
```

Using the managed memory option enabled this GPU port.

USING CUDA PREFETCH HINTS

Interface with CUDA prefetch API to improve performance

```
  subroutine memset_r8_flat(a,n,v,asyncid)
    implicit none
    real(8) :: a(n)
    real(8) :: v
    integer :: n, asyncid, i
#if defined(_OPENACC) && defined (_CUDA)
    !$acc host_data use_device(a)
    ierr = cudaMemsetAsync( a , v , n , acc_get_cuda_stream(asyncid) )
    !$acc end host_data
#else
    !$acc parallel loop async(asyncid)
    do i=1,n
      a(i) = v
    enddo
#endif
  end subroutine memset_r8_flat
```

PREFETCH WRAPPERS

Examples of multi-dimension pre-fectch

```
  subroutine prefetch_r8_3d(a)
    implicit none
    real(8) :: a(:,:,:)
    call prefetch_r8_flat(a,product(shape(a)))
  end subroutine prefetch_r8_3d
```

```
  subroutine prefetch_r8_flat(a,n)
    implicit none
    real(8) :: a(n)
    integer :: n
#if defined(_OPENACC) && defined (_CUDA)
    !$acc host_data use_device(a)
    ierr = cudaMemPrefetchAsync( a , n , acc_get_device_num(acc_device_nvidia) , acc_get_cuda_stream(asyncid_loc+1) )
    !$acc end host_data
#endif
  end s
```

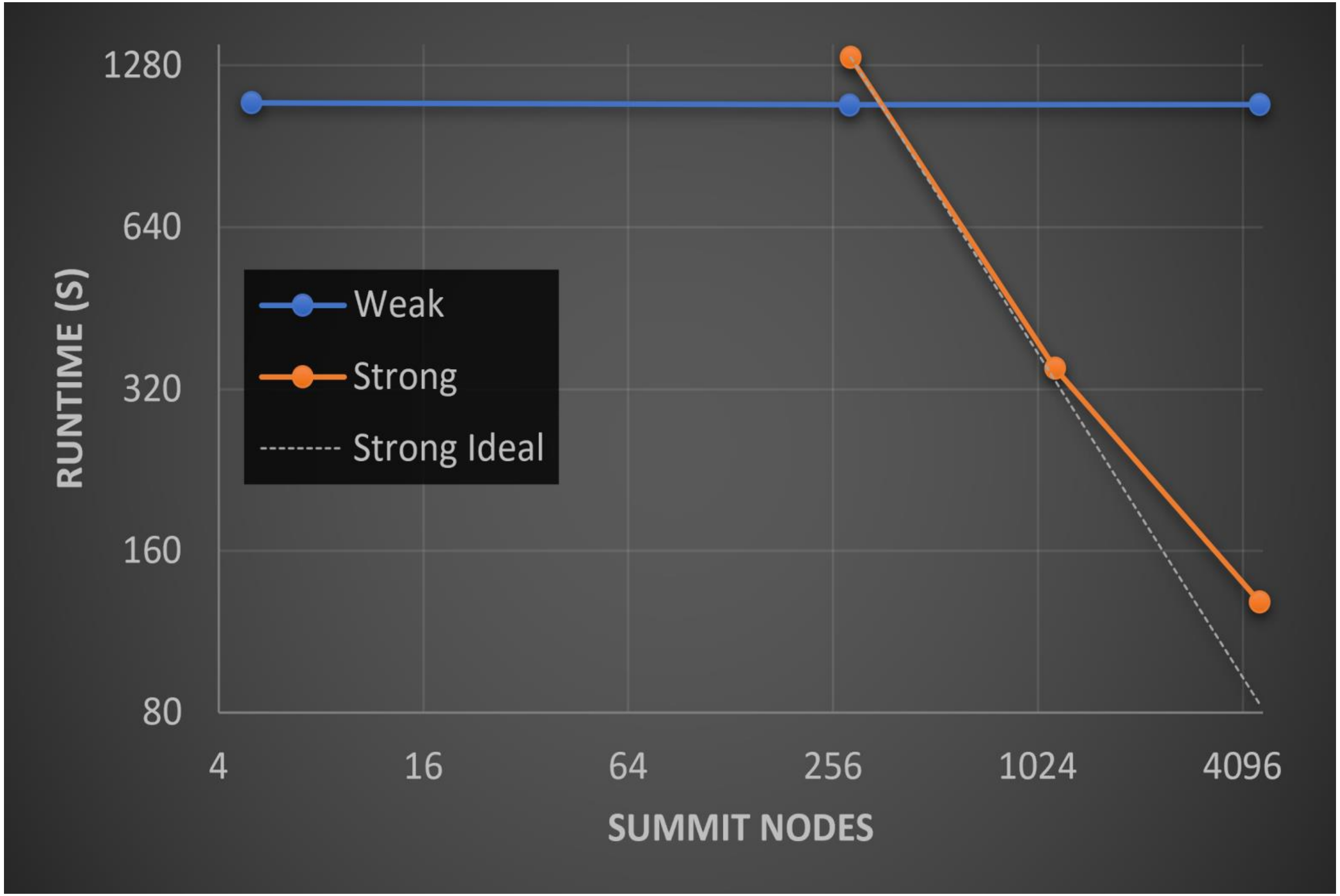
```
  subroutine memset_r8_3d(a,v,asyncid)
    implicit none
    real(8) :: a(:,:,:)
    real(8) :: v
    integer :: asyncid
    call memset_r8_flat(a,product(shape(a)),v,asyncid)
  end subroutine memset_r8_3d
```

Performance on OLCF Summit Supercomputer

Runtime for one model day

Weak: 28km GCM with 64x64 columns per CRM

Strong: 28km GCM with 16x16 columns per CRM



Courtesy of Matt Norman, ORNL

Performance on OLCF Summit Supercomputer

- Gordon Bell simulations: 3-D 500m global grid spacing at 2 SYPD
  - 28km GCM grid spacing, 64x64 CRM columns per GCM column
  - Using 4,600 nodes of Summit, we get 2.5% peak flop/s
  - 2.5% peak flops on Volta GPU = 33 flops per memory load / store
  - About 200 kernels in 30K LOCs using PGI OpenACC

- Current <u>production</u> simulations: 2-D CRM at 500m dx at 3 SYPD
  - 3 SYPD with 28km GCM grid spacing and 64x1 columns per CRM
  - Using 1,000 Summit nodes, also 2.5% peak flop/s
  - About 15x speed-up using 6 Voltas/node versus 2 Power9/node

Courtesy of Matt Norman, ORNL

CLOSING

CONCLUSION

OpenACC is a mature, directive-based programming model that is available for GPUs, multicore CPUs, and more and is in use by more than 200 scientific applications.

OPENACC RESOURCES

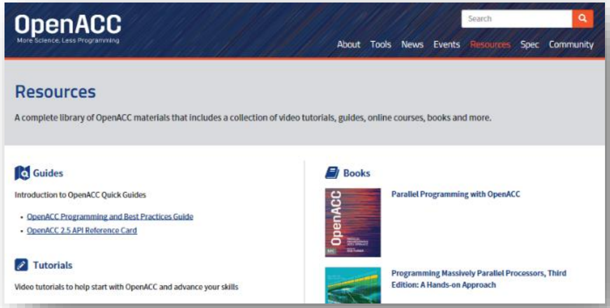
Guides ● Talks ● Tutorials ● Videos ● Books ● Spec ● Code Samples ● Teaching Materials ● Events ● Success Stories ● Courses ● Slack ● Stack Overflow

FREE Compilers

PGI Community EDITION

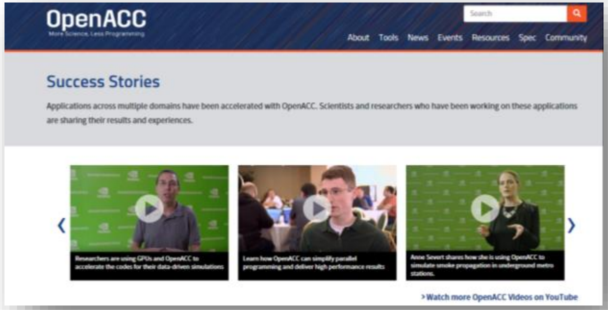
Resources

https://www.openacc.org/resources



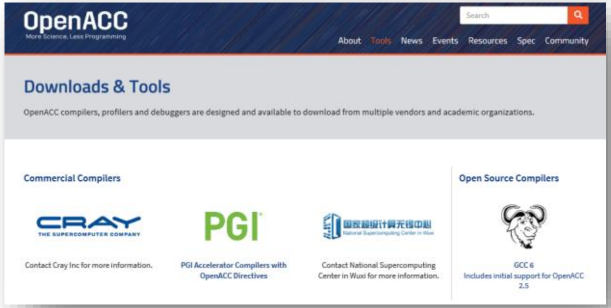
Success Stories

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



Compilers and Tools

https://www.openacc.org/tools



Events

https://www.openacc.org/events