PGI® COMPILERS & TOOLS

DIRECTIVE BASED GPU PROGRAMMING

Markus Wetzstein

CSCS, May 2018

NVIDIA

ASYNCHRONOUS EXECUTION

SYNCHRONOUS VS ASYNCHRONOUS

Without additional clauses, all OpenACC directives are blocking, i.e. the CPU triggers the kernel launch or data transfer and waits until the completion of the operation.

All operations are inserted into a default activity queue. Items in the queue are executed in the order in which they are inserted.

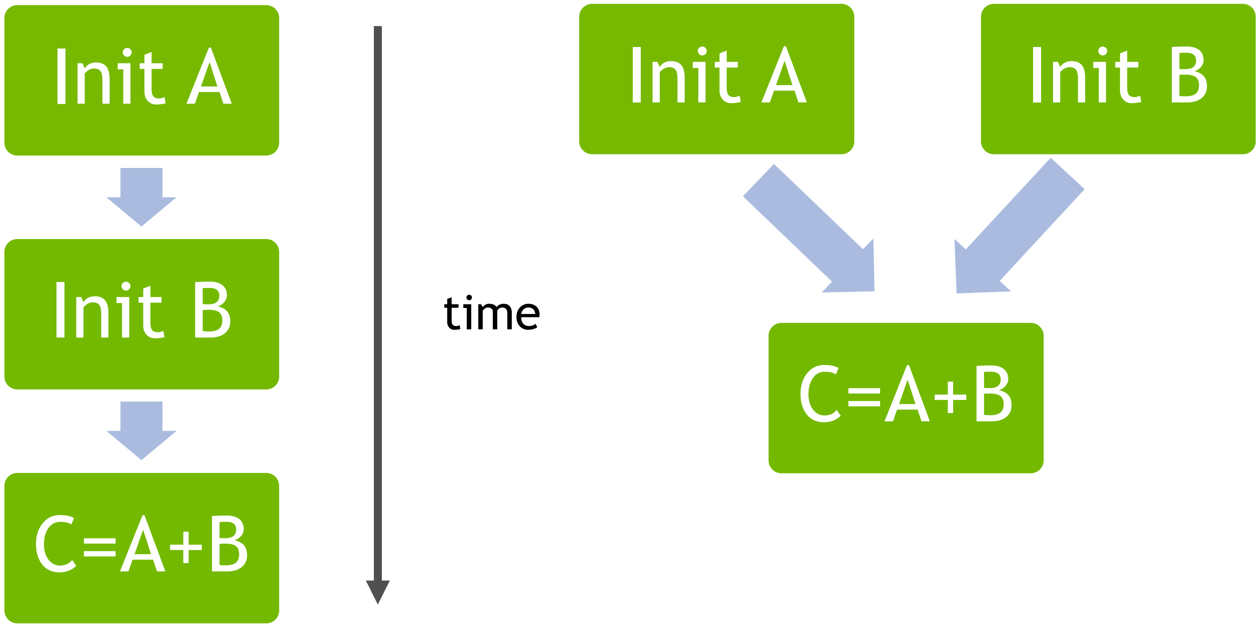
But:

Depending on our algorithm, we might have independent pieces of work, which could be executed in any order, or even simultaneously.

→ How could we exploit that?

EXAMPLE

Assume we want to initialize two arrays A and B with values, then sum them up into a third array.



Both initializations can be executed simultaneously, but need to be completed before the sum.

ASYNC QUEUES

OpenACC has multiple asynchronous activity queues. All kernel launches and data movements can be placed into different queues (or just the default queue).

Inside each activity queue, the operations are executed strictly in the order they were inserted.

But the queues are completely independent, and can overlap execution (resources permitting).

async and wait clauses / directives to distribute work and create synchronization points where needed.

On GPUs, the asynchronous activity queues correspond to CUDA streams.

OPENACC ASYNC AND WAIT

async(n): clause to launch work asynchronously in queue *n* (integer)

wait(n): directive to block host until all operations in queue *n* have completed. Without argument, all queues need to complete.

Can significantly reduce launch latency, enables pipelining and concurrent operations.

OPENACC ASYNC AND WAIT

async(n): clause to launch work asynchronously in queue *n*

wait(n): directive to block host until all operations in queue *n* have completed. Without argument, all queues need to complete.

Can significantly reduce launch latency, enables pipelining and concurrent operations

```
#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)
  ...
#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)
  ...
#pragma acc wait(1)
```

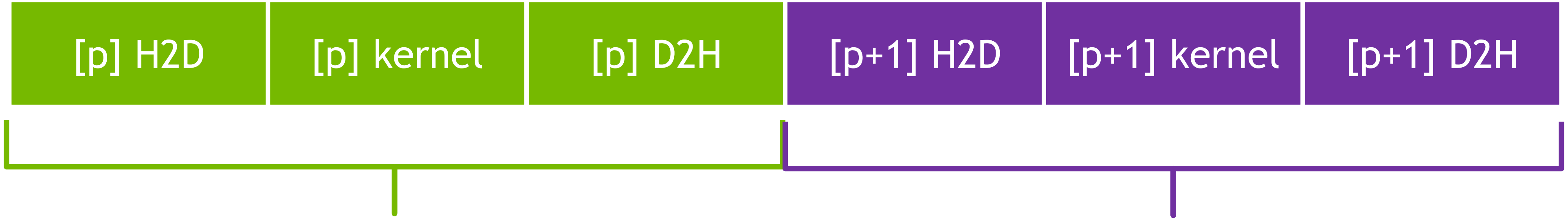
OPENACC PIPELINING

```
#pragma acc data …
for(int p = 0; p < nplanes; p++)
{
  #pragma acc update device(plane[p])
  #pragma acc parallel loop
  for (int i = 0; i < nwork; i++)
  {
    // Do work on plane[p]
  }
  #pragma acc update host(plane[p])
}
```

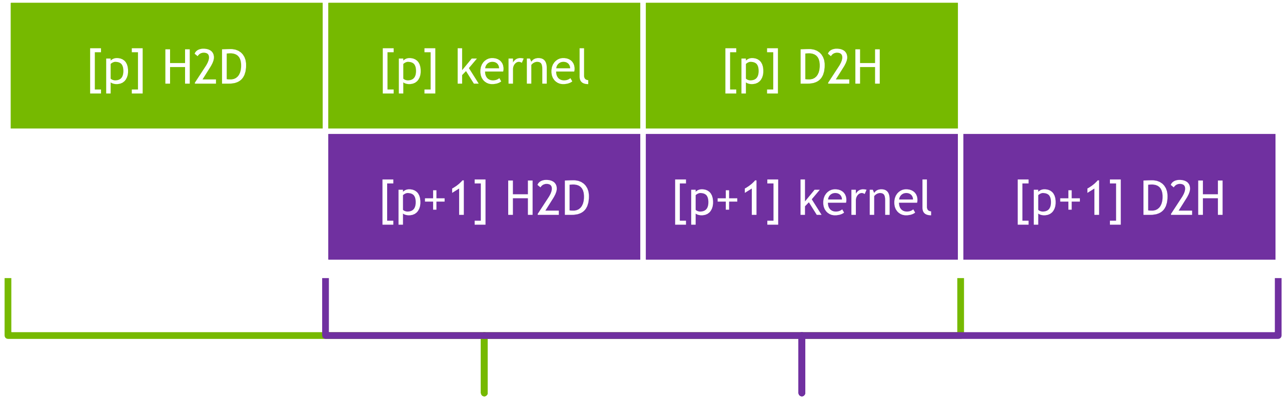
For this example, assume that each "plane" is completely independent and must be copied to/from the device.

As it is currently written, plane[p+1] will not begin copying to the GPU until plane[p] is copied from the GPU.

OPENACC PIPELINING — CONTINUED



P and P+1 Serialize



P and P+1 Overlap Data Movement

NOTE: In real applications, your boxes will not be so evenly sized.

OPENACC PIPELINING — CONTINUED

```
#pragma acc data …
for(int p = 0; p < nplanes; p++)
{
  #pragma acc update device(plane[p]) async(p)
  #pragma acc parallel loop async(p)
  for (int i = 0; i < nwork; i++)
  {
    // Do work on plane[p]
  }
  #pragma acc update host(plane[p]) async(p)
}
#pragma acc wait
```

Enqueue each plane in a queue to execute in order

Wait on all queues.

ASYNC ADDITIONS

```
#pragma acc parallel async(5) wait(4,3)
{
   #pragma acc loop collapse(2) gang vector
   for( j = 1; j < n-1; ++j )
      for( i = 1; i < m-1; ++i )
         x[i][j] = 0.25*( y[i-1][j] + y[i+1][j] +
                          y[i][j-1] + y[i][j+1]);

}
```

// this parallel region goes into queue 5

// this parallel region waits for existing work in queues 4,3 to finish before starting execution of the region

ASYNC ADDITIONS

```
#pragma acc wait(3)
```

// this host thread waits for queue 3 to complete

```
#pragma acc wait(3) async(5)
```

// queue 5 waits for all events on queue 3 to complete

DEEPCOPY - HANDLING USER DEFINED DATA STRUCTURES WITH OPENACC

DATA REGIONS IN OPENACC

intrinsic data types, static and dynamic

All static intrinsic data types of the programming language can appear in an OpenACC data directive, e.g. `real`, `complex`, `integer` scalar variables in Fortran.

Same for all fixed size arrays of intrinsic types, and dynamically allocated arrays of intrinsic type, e.g. `allocatable` and `pointer` variables in Fortran

The compiler will know the base address and size (arrays in C: size needs to be specified in directive).

… So what about **derived types**? Two variants:

```
type stat_def
    integer a,b
    real c
end type stat_def
type(stat_def)::var_stat
```

```
type dyn_def
    integer m
    real,allocatable,dimension(:) :: r
end type dyn_def
type(dyn_def)::var_dyn
```

DATA REGIONS IN OPENACC

derived data types

A static size derived data type can be handled by the compiler like a variable of intrinsic type: it knows the start address, size **and** the start address and size of each member.

But derived types with dynamic members are not handled automatically!

`var_dyn%r` was itself of intrinsic type, but in general it could be of another derived type!

```
type dyn_def
    integer m
    real,allocatable,dimension(:) :: r
end type dyn_def
type(dyn_def)::var_dyn
```

To treat the generic case, the compiler will need to generate code, such that the runtime system handles all possibilities. This is currently not supported in OpenACC 2.6 !

DATA REGIONS IN OPENACC

trees of derived data types

```
type dyn_def1
     integer m
     type(dyn_def2),pointer:: x
end type dyn_def1
type(stat_def)::var_dyn
```

```
type dyn_def2
     logical b
     real,pointer:: field(:,:)
end type dyn_def2
```

Programmer would like to write: `!$acc enter data copyin(var_dyn)` no matter what the composition of `dyn_def1` is! Including how deep the potential tree of derived types is, which dynamic members exist in which derived type, etc.

To treat the generic case, the runtime system would need to:

- allocate device memory for the parent type
- copy each static member
- allocate device memory for each dynamic member
- transfer the dynamic member
- attach the pointer in the parent to the member's device copy
- repeat until all leaves of the derived type tree are reached

PGI

NVIDIA

DEEPCOPY IN OPENACC

full vs manual

The generic case is a main goal for a future OpenACC 3.0 specification. This is often referred to as **full deep copy**.

Until then, writing a **manual deep copy** is the best way to handle derived types:

- Static members are handled by the compiler.
- The programmer manually copies every dynamic member.
- **AND** ensures correct pointer attachment/detachment in the parent!

OpenACC 2.6 provides all operations necessary to support manual deep copy.

DERIVED TYPE

manual copy

```
type dyn_def
    integer m
    real,allocatable,dimension(:) :: r
end type dyn_def
type(dyn_def)::var_dyn
…
allocate(var_dyn%r(some_size))
!$acc enter data copyin(var_dyn,var_dyn%r)
…
!$acc exit data copyout(var_dyn%r,var_dyn)
```

1. allocates device memory for `var_dyn`
2. copies `m` (H2D)
3. copies host pointer for `var_dyn%r`!
   -> device ptr defect

1. allocates device memory for `r`
2. copies `r` (H2D)
3. **attaches** the device copy's pointer `var_dyn%r` to the device copy of `r`

1. copies `r` (D2H)
2. deallocates device memory for `r`
3. **detaches** `var_dyn%r` on the device, i.e. overwrites `r` with its host value !
   -> device ptr defect

1. copies `m` (D2H)
2. copies `var_dyn%r` **-> host pointer intact !**
3. deallocates device memory for `var_dyn`

PGI

NVIDIA

DERIVED TYPE

manual copy

Important:

- the defect pointers must not be dereferenced!
- the compiler must know that `r` is the member `var_dyn%r` !
- otherwise, we must use extra API calls: acc_attach/acc_detach (OpenACC 2.6)

```
…
allocate(var_dyn%r(some_size))
!$acc enter data copyin(var_dyn)
call mycopyin(var%r)
call acc_attach(var_dyn%r)
…
call mycopyout(var%r)
call acc_detach(var_dyn%r)
!$acc exit data copyout(var_dyn)
```

```
subroutine mycopyin(r)
  real,allocatable,dimension(:)::r
!$acc enter data copyin(r)
end subroutine mycopyin

subroutine mycopyout(r)
  real,allocatable,dimension(:)::r
!$acc exit data copyout(r)
end subroutine mycopyout
```

MANUAL DEEPCOPY

Why the complication with the subroutine? -> Can help avoid (some) code replication when writing a generic manual deepcopy for a tree of derived types with several levels.

Typically, we need separate routines for `create`, `copyin`, `copyout`, `delete` directives

Important: use `update` directive only on members, never on a parent (it will overwrite the member pointers)!

PGI
ENVIRONMENT VARIABLES

PGI FEATURES

OpenACC environment variables

PGI_ACC_TIME (= 0 or 1)

Provides timing information for each kernel.

PGI_ACC_SYNCHRONOUS (= 0 or 1)

Enforces synchronous execution of all asynchronous operations (useful for debugging).

PGI_ACC_NOTIFY BIT MASK

1 – launch
```
launch CUDA kernel  file=smooth4.c function=smooth_acc line=17 device=0 num_gangs=98
num_workers=1 vector_length=128 grid=1x98 block=128
```
2 – data upload/download
```
upload CUDA data  file=smooth4.c function=smooth_acc line=12 device=0 variable=a
bytes=40000
download CUDA data  file=smooth4.c function=smooth_acc line=23 device=0 variable=a
bytes=40000
```
4 – wait (explicit or implicit) for device
```
Implicit wait  file=smooth4.c function=smooth_acc line=17 device=0
Implicit wait  file=smooth4.c function=smooth_acc line=23 device=0
```
8 – data/compute region enter/leave
```
Enter data region file=smooth4.c function=smooth_acc line=12 device=0
Enter compute region file=smooth4.c function=smooth_acc line=14 device=0
Leave compute region file=smooth4.c function=smooth_acc line=17 device=0
```
16 – data create/allocate/delete/free
```
create CUDA data  bytes=40000 file=smooth4.c function=smooth_acc line=12 device=0
alloc  CUDA data  bytes=40000 file=smooth4.c function=smooth_acc line=12 device=0
delete CUDA data  bytes=40448 file=smooth4.c function=smooth_acc line=23 device=0
```

AN OUTLLOK ON OPENACC

OPENACC STANDARD

- open consortium
- user driven standard
- commercial compilers: Cray, PGI
- open source compilers: gcc
- several academic research compilers

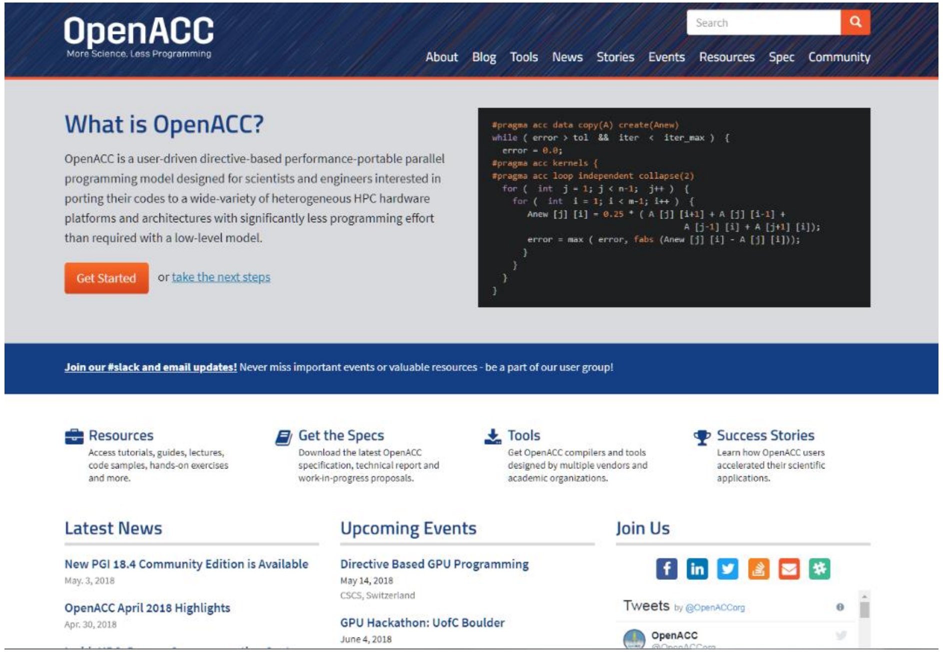
Current Members

CRAY THE SUPERCOMPUTER COMPANY
UNIVERSITY OF DELAWARE
東京工業大学 Tokyo Institute of Technology
KAUST King Abdullah University of Science and Technology
INDIANA UNIVERSITY
TOTAL
AMD
NVIDIA
国家超级计算无锡中心 National Supercomputing Center In Wuxi
ILLINOIS UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
NOAA NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION U.S. DEPARTMENT OF COMMERCE
OAK RIDGE National Laboratory
epcc
Mentor Graphics
BROOKHAVEN NATIONAL LABORATORY
Stony Brook University
HZDR HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF
VirginiaTech
SHANGHAI JIAO TONG UNIVERSITY 1896
MICHIGAN STATE UNIVERSITY

ONLINE RESOURCES

openacc.org

- OpenACC user group on slack
- OpenACC standard doc and quick guide
- Training materials
- Announcements for workshops, conferences, etc.



DRAFT OPENACC 3.0 TRUE DEEP COPY

Still in definition by the OpenACC Committee

```
typedef struct points {
    float* x;  float* y;  float* z;
    int n;
    float coef, direction;
    #pragma acc policy inout(x[0:n],y[0:n])
} points;

void sub ( int n, float* y ) {
    points p;

        p.n = n;
        p.x = ( float*) malloc ( sizeof ( float )*n );
        p.y = ( float*) malloc ( sizeof ( float )*n );
        p.z = ( float*) malloc ( sizeof ( float )*n );

        #pragma acc data copy (p)
        {
            #pragma acc parallel loop
            for ( i =0; i<p.n; ++I ) p.x[i] += p.y[i];
            . . .
```

OpenACC is for Multicore, Manycore & GPUs

```
 98 !$acc parallel
 99 !$acc loop independent
100      DO k=y_min-depth,y_max+depth
101 !$acc loop independent
102        DO j=1,depth
103          density0(x_min-j,k)=left_density0(left_xmax+1-j,k)
104        ENDDO
105      ENDDO
106 !$acc end parallel
```

Multicore CPU

```
% pgfortran -ta=multicore -fast -Minfo=acc -c \
  update_tile_halo_kernel.f90
  . . .
  100, Loop is parallelizable
       Generating Multicore code
       100, !$acc loop gang
  102, Loop is parallelizable
```

Tesla GPU

```
% pgfortran -ta=tesla -fast -Minfo=acc -c \
  update_tile_halo_kernel.f90
  . . .
  100, Loop is parallelizable
  102, Loop is parallelizable
       Accelerator kernel generated
       Generating Tesla code
       100, !$acc loop gang, vector(4) ! blockidx%y threadidx%y
       102, !$acc loop gang, vector(32) ! blockidx%x threadidx%x
```

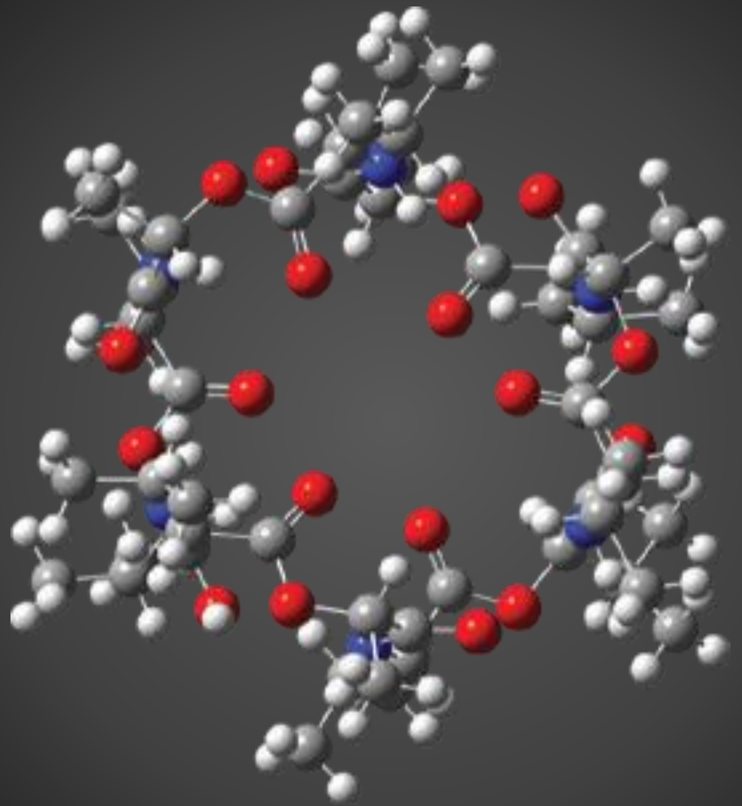
PGI COMPILERS FOR EVERYONE

The PGI 18.4 Community Edition

FREE

	PGI Community EDITION	PGI Professional EDITION	PGI Enterprise EDITION
PROGRAMMING MODELS OpenACC, CUDA Fortran, OpenMP, C/C++/Fortran Compilers and Tools	✓	✓	✓
PLATFORMS X86, OpenPOWER, NVIDIA GPU	✓	✓	✓
UPDATES	1-2 times a year	6-9 times a year	6-9 times a year
SUPPORT	User Forums	PGI Support	PGI Premier Services
LICENSE	Annual	Perpetual	Volume/Site

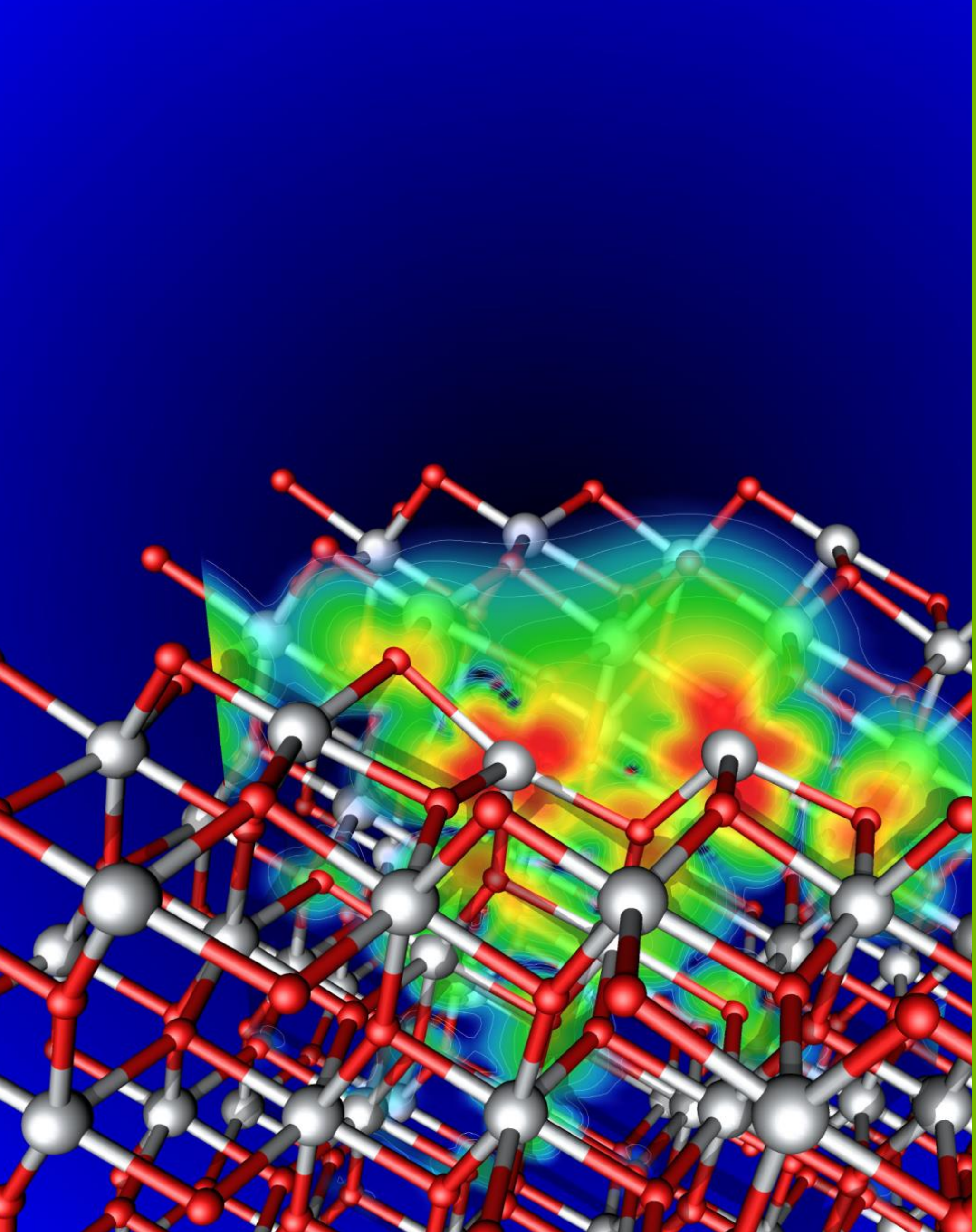
pgicompilers.com/community



GAUSSIAN 16

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

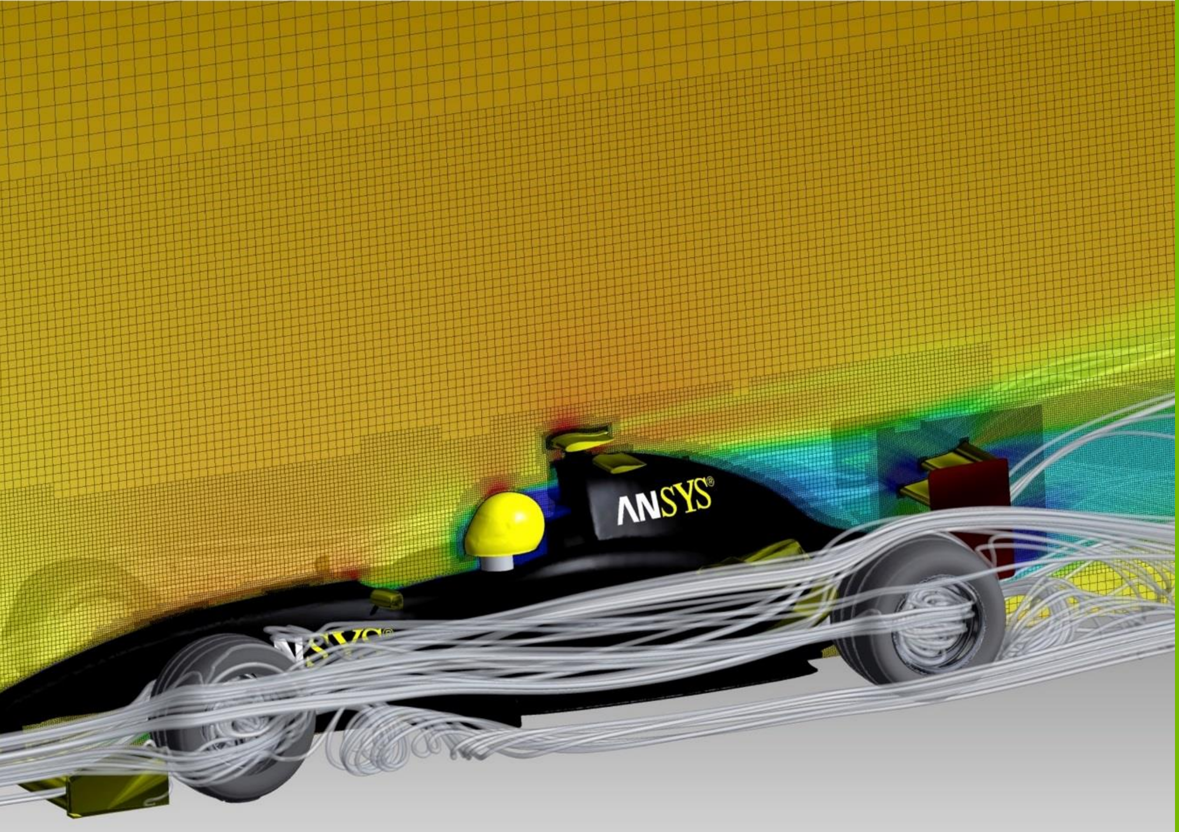
"Using OpenACC allowed us to continue development of our fundamental algorithms and software capabilities simultaneously with the GPU-related work. In the end, we could use the same code base for SMP, cluster/network and GPU parallelism. PGI's compilers were essential to the success of our efforts."



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."



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.”

PGI

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."

COSMO

Dr. Oliver Fuhrer
Senior Scientist
Meteoswiss

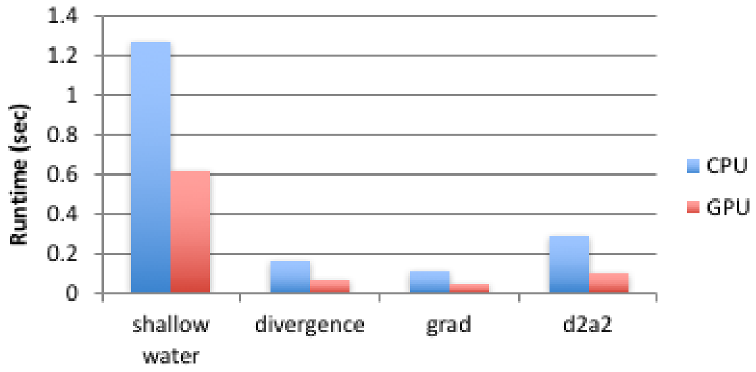
"OpenACC made it practical to develop for GPU-based hardware while retaining a single source for almost all the COSMO physics code."

FV3 WEATHER MODEL

Global Weather Forecast Model

OpenACC Performance

- PGI compiler V16.10
- 2X faster performance on GPU
  - Dual-socket Haswell CPU and NVIDIA Pascal (P100) GPU



Slide courtesy of Mark Govett,
NOAA / Earth System Research Laboratory

Mark Govett
Chief, HPC Section
NOAA

"Lessons learned in the development of NIM and F2C-ACC have proven invaluable in our current efforts to create a single, performance portable version of the FV3 weather model using OpenACC."

PGI

Image courtesy: NCAR

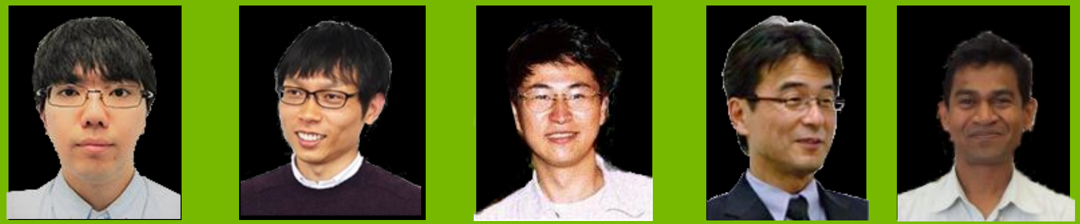
MPAS-A

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."

Map courtesy University of Tokyo

GAMERA FOR GPU



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