OPENACC is...

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

Add Simple Compiler Directive

```c
main()
{
    <serial code>
    #pragma acc kernels
    {
        <parallel code>
    }
}
```
OPENACC COMMUNITY MOMENTUM

3 OF TOP 5 HPC APPS

5 OF 13 CAAR CODES

ACCELERATED APPS

180,000+ DOWNLOADS

3 OF TOP 5 HPC APPS

5 OF 13 CAAR CODES

ACCELERATED APPS

180,000+ DOWNLOADS
Gaussian 16

Mike Frisch, PhD
Professor of Chemistry
University of California, Berkeley

Using Gaussian 16, researchers continue to push the boundaries of computational chemistry. This software has been extensively used in numerous applications, from drug discovery to material science, making it a cornerstone of our research.

ANSYS Fluent

Kurt Umbrello
Sr. Product Manager
ANSYS

We are effectively using ANSYS Fluent for high-fidelity computer simulations, which improves performance and reduces costs. This software is essential in the development of many of our new platforms.

VASP

Prof. Uwe века
University of Innsbruck

For VASP, OpenACC is the key to making our software more efficient. Performance is critical in such fields as quantum, and we have been working closely with NVIDIA to ensure that VASP development and maintenance are seamlessly integrated. We are also collaborating with NVIDIA and AMD on an early version of CUDA, which is now available.

Cosmo

Dr. Christopher Joustra
Director of Technology Development

OpenACC has helped us develop a high-performance application portfolio, and we see it as a critical tool for the future of our research.

E3SM

Paul A. Levy
E3SM Applications Lead

Our simulations are pushing the boundaries of what we can achieve in climate science. OpenACC has allowed us to harness the power of GPUs, making our simulations more efficient and scalable.

NUMECA FINE/Open

David Goodwin
Chief Technology Officer
NUMECA

Putting our advanced CFD solver FINE/Open to GPUs using OpenACC has been a game-changer. We have now successfully developed simulations that are running more efficiently than ever before.

SYNOPSIS

Dr. Luis Oberoi
Scientific Advisory Board
NUMECA

Using OpenACC, we are working to improve the performance of our simulation code. Our goal is to leverage the full potential of GPU computing in the design of advanced simulation software.

MPAS-A

Michael P. Farrell
Director of Technology Development

OpenACC has enabled us to achieve significant performance improvements in our simulations, allowing us to tackle larger and more complex problems.

GAMER

Dr. Yvon Madelin
Senior Researcher

With OpenACC and a computer based on NVIDIA’s Tesla P100 GPU, we achieved more than a 10x speedup over a K Computer node running our earthquake disaster simulation code.

SANJEEVINI

Anil K. Joshi
Project Coordinator
Indian Institute of Technology New Delhi

We have developed a high-performance simulation package using OpenACC, which has enabled us to achieve significant performance improvements in our simulations.

IBM-CFD

Kavita Kulkarni
IBM

Our simulations are critical for understanding complex physical phenomena. OpenACC has enabled us to accelerate our simulations, making them more efficient and scalable.

PWscf (Quantum ESPRESSO)

Kees van Duijneveldt
NIST Center for Neutron Research

OpenACC has been instrumental in our efforts to simulate complex materials. It has enabled us to achieve significant performance improvements in our simulations.

MAS

Dr. Michael J. Capelli
Director of Technology Development

OpenACC has been a key component of our research efforts, enabling us to achieve significant performance improvements in our simulations.
INTRODUCTION TO OPENACC
OpenACC Directives

- Manage Data Movement
  
  ```
  #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];
        ...
      }
    }
  }
  ...
  ```

- Initiate Parallel Execution

- Optimize Loop Mappings

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, Manycore
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 its 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: $\nabla^2 f(x, y) = 0$

$$A_{k+1}(i, j) = \frac{A_k(i-1, j) + A_k(i+1, j) + A_k(i, j-1) + A_k(i, j+1)}{4}$$
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++ ) {


      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++;
}
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++;
}
Building the Code (GPU)

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

Print compiler feedback so we can see what it did.

$ 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

* 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=m 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
65, 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

No change to code between CPU & GPU!
CUDA UNIFIED MEMORY
Simplified Developer Effort

Without Managed Memory

[Diagram showing system memory and GPU memory without managed memory]

With Managed Memory

[Diagram showing CPU and GPU memories combined into a single, shared pool as managed memory]

Commonly referred to as “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
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.

Definition

```
#pragma acc data clauses
{
    < Sequential and/or Parallel code >
}
```

```
!$acc data clauses

    < Sequential and/or Parallel code >

!$acc end data
```
<table>
<thead>
<tr>
<th>Declaration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy(list)</code></td>
<td>Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.</td>
</tr>
<tr>
<td><strong>Principal use:</strong></td>
<td>For many important data structures in your code, this is a logical default to input, modify and return the data.</td>
</tr>
<tr>
<td><code>copyin(list)</code></td>
<td>Allocates memory on GPU and copies data from host to GPU when entering region.</td>
</tr>
<tr>
<td><strong>Principal use:</strong></td>
<td>Think of this like an array that you would use as just an input to a subroutine.</td>
</tr>
<tr>
<td><code>copyout(list)</code></td>
<td>Allocates memory on GPU and copies data to the host when exiting region.</td>
</tr>
<tr>
<td><strong>Principal use:</strong></td>
<td>A result that isn’t overwriting the input data structure.</td>
</tr>
<tr>
<td><code>create(list)</code></td>
<td>Allocates memory on GPU but does not copy.</td>
</tr>
<tr>
<td><strong>Principal use:</strong></td>
<td>Temporary arrays.</td>
</tr>
</tbody>
</table>
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

```c/c++
copy(array[starting_index:length])
```

```fortran
copy(array(starting_index:ending_index))
```
#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++;
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

<table>
<thead>
<tr>
<th>Speed-Up</th>
<th>SERIAL</th>
<th>MULTICORE</th>
<th>V100</th>
<th>V100 (DATA)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1.00X</td>
<td>3.23X</td>
<td>41.80X</td>
<td>42.99X</td>
</tr>
</tbody>
</table>
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 $(x, y, z) \in [0,100]$ km
- 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,n6; ...; DO i2=1,n2
  ! Calculate G(:,i2,i3,i4,i5,i6)
  DGEMV
  DO i1=1,n1
    ! Calculate V(:,i1,i2,i3,i4,i5,i6)
    ! Calculate S(:,i1,i2,i3,i4,i5,i6)
    ! Calculate U(:,i1,i2,i3,i4,i5,i6)
    DGEMV
  END DO
END DO; ...; END DO
```

**OpenACC code**

```
! Calculate G
  DGEMM
! Calculate S
  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)
    DGEMM
  END DO; ...; END DO
  DGEMM
  END DO; ...; END DO
```

*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 * 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 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
!important defined(THORNADO_OACC)
device_is_present = ( acc_is_present( hostptr, bytes ) > 0 )
#else
device_is_present = .false.
!important
END FUNCTION

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

Courtesy of Austin Harris, ORNL
GPU BENCHMARKS

3D Deleptonization Wave (12^3 Cells)
FP Coupled

Wall Time / Step [s]

CPU
GPU

Total
Explicit
Implicit
Opacity

Courtesy of Austin Harris, ORNL
## PROGRAMMING MODEL COMPARISON

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Offload Model</th>
<th>$T_{CPU}$</th>
<th>$T_{GPU}$</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>PGI v19.4</td>
<td>OpenACC v2.7</td>
<td>27.8 sec/step</td>
<td>0.42 sec/step</td>
<td>67X</td>
</tr>
<tr>
<td>XL v16.1.1</td>
<td>OpenMP v4.5</td>
<td>25.6 sec/step</td>
<td>0.99 sec/step</td>
<td>26X</td>
</tr>
</tbody>
</table>

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)

Courtesy of Matt Norman, ORNL
COMPLEX DATA TYPES

```fortran
17  type crm_rad_type
18    ! Radiative heating
19    real(crm_rknd), pointer :: qrad(:,,:),:
20
21    ! Quantities used by the radiation code. Note that these are strange in that they are
22    ! time-averages, but spatially-resolved.
23    real(crm_rknd), pointer :: temperature(:,,:),:
24    ! rad temperature
25    real(crm_rknd), pointer :: qv (,:,:,:),:
26    ! rad vapor
27    real(crm_rknd), pointer :: qc (,:,:,:),:
28    ! rad cloud water
29    real(crm_rknd), pointer :: qi (,:,:,:),:
30    ! rad cloud ice
31    real(crm_rknd), pointer :: cld (,:,:,:),:
32    ! rad cloud fraction
33
34    ! Only relevant when using 2-moment microphysics
35    real(crm_rknd), pointer :: nc (,:,:,:),:
36    ! rad cloud droplet number (#/kg)
37    real(crm_rknd), pointer :: ni (,:,:,:),:
38    ! rad cloud ice crystal number (#/kg)
39    real(crm_rknd), pointer :: qs (,:,:,:),:
40    ! rad cloud snow (kg/kg)
41    real(crm_rknd), pointer :: ns (,:,:,:),:
42    ! rad cloud snow crystal number (#/kg)
43  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

```fortran
119  subroutine memset_r8_flat(a,n,v,asyncid)
120     implicit none
121     real(8) :: a(n)
122     real(8) :: v
123     integer :: n, asyncid, i
124   ifdef (_OPENACC) && defined (_CUDA)
125     !$acc host_data use_device(a)
126     ierr = cudaMemcpyAsync( a , v , n , acc_get_cuda_stream(asyncid) )
127     !$acc end host_data
128   else
129     !$acc parallel loop async(asyncid)
130     do i=1,n
131         a(i) = v
132     enddo
133 endif
134 end subroutine memset_r8_flat
```
PREFETCH WRAPPERS

Examples of multi-dimension pre-fetchch

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)
  !$acc host_data use_device(a)
  ierr = cudaMemcpyAsync( a, n, acc_get_device_num(acc_device_nvidia), acc_get_cuda_stream(asyncid_loc+1) )
  !$acc end host_data
  #endif
end subroutine prefetch_r8_flat

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/stores
  - About 200 kernels in 30K LOCs using PGI OpenACC

- Current production 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

Compilers and Tools
https://www.openacc.org/tools

Resources
https://www.openacc.org/resources

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

Events
https://www.openacc.org/events

Free Compilers