# To OpenACC 3.0, and Beyond!

*Jeff Larkin <jlarkin@nvidia.com>,*

*Sr. DevTech SW Engineer, NVIDIA*

OpenACC

More Science, Less Programming

# To OpenACC 3.0, and Beyond!

*Jeff Larkin <jlarkin@nvidia.com>,*
*OpenACC Technical Committee Chair*

OpenACC
More Science, Less Programming

# About OpenACC


**More Science, Less Programming**

**OpenACC is a directives-based programming model designed to deliver performance and portability for modern parallel programs.**

# 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];
          ...
      }
  }
  ...
}
```

OpenACC
Directives for Accelerators

- Incremental
- Single source
- Interoperable
- Performance portable
- CPU, GPU, Manycore

# OPENACC DIRECTIVES

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

### 3 OF TOP 5 HPC

Intersect360 RESEARCH

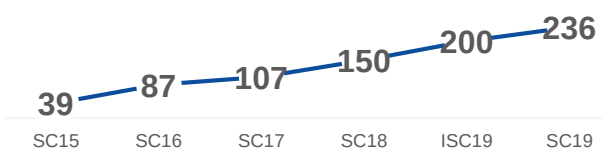
### 18% OF INCITE ON SUMMIT

summit

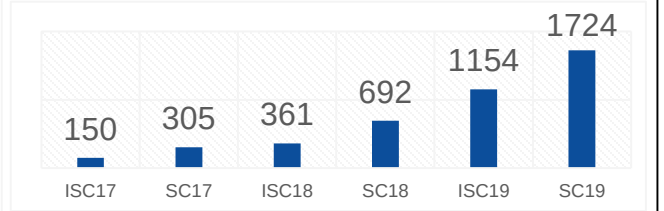
### PLATFORMS SUPPORTED

NVIDIA GPU
X86 CPU
POWER CPU
Sunway
ARM CPU
AMD GPU

### OPENACC APPS

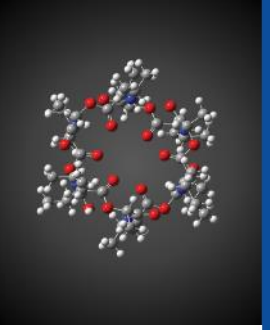



### OPENACC SLACK MEMBERS




### >200K DOWNLOADS

PGI Community EDITION



## GAUSSIAN 16

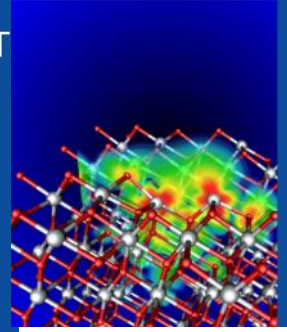
Mike Frisch, Ph.D.
President and
CEO
Gaussian, Inc.

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.

## ANSYS FLUENT

Sunil Saha
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.

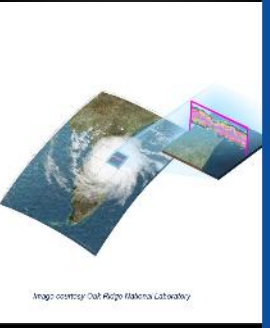


Image courtesy Oak Ridge National Laboratory

## E3SM

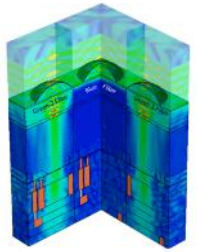
Mark A. Taylor
Multiphysics Applications
Sandia

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.

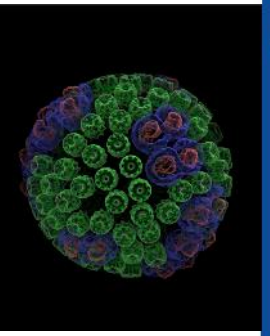
Using OpenACC, we've GPU-accelerated the Synopsys TCAD Sentaurus Device EMW simulator to speed up optical simulations of image sensors. GPUs are key to improving simulation throughput in the design of advanced image sensors.

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 7.7 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.

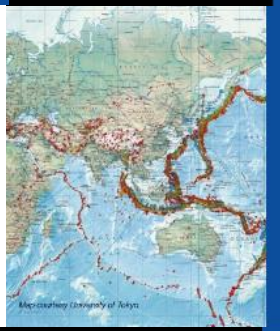
## GTC

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

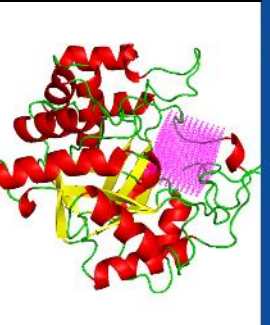


Map courtesy University of Tokyo

## GAMERA

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

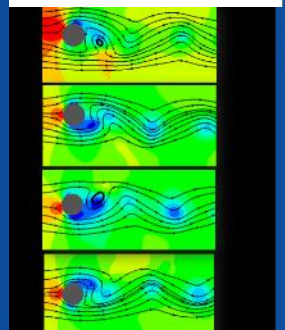
With OpenACC and a compute node based on NVIDIA's Tesla P100 GPU, we achieved more than a 14X speed up over a K Computer node running our earthquake disaster simulation code



## SANJEEVINI

Abhilash Jayaraj
Project Scientist
Indian Institute of Technology
New Delhi

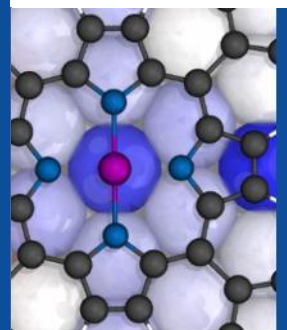
In an academic environment maintenance and speedup of existing codes is a tedious task. OpenACC provides a great platform for computational scientists to accomplish both tasks without involving a lot of efforts or manpower in speeding up the entire computational task.



## IBM-CFD

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

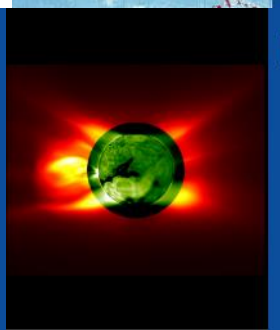
OpenACC can prove to be a handy tool for computational engineers and researchers to obtain fast solution of non-linear dynamics problem. In immersed boundary incompressible CFD, we have obtained order of magnitude reduction in computing time by porting several components of our legacy codes to GPU. Especially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

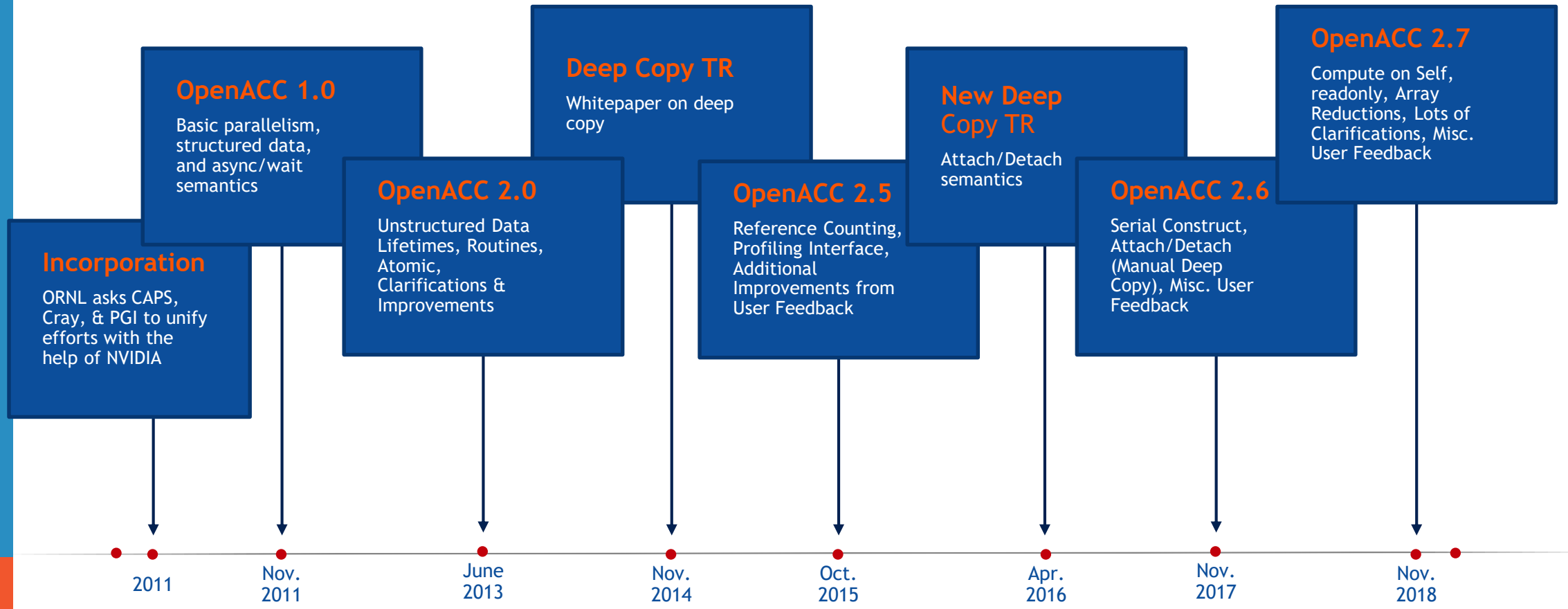


## MAS

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.

# A Brief History



# Announcing OpenACC 3.0

Added C18, C++17, Fortran 2018 as supported base languages

Support for C++ lambdas

Improved multi-device support through direct memory copies and synchronization

Added zero-on-create to data clauses

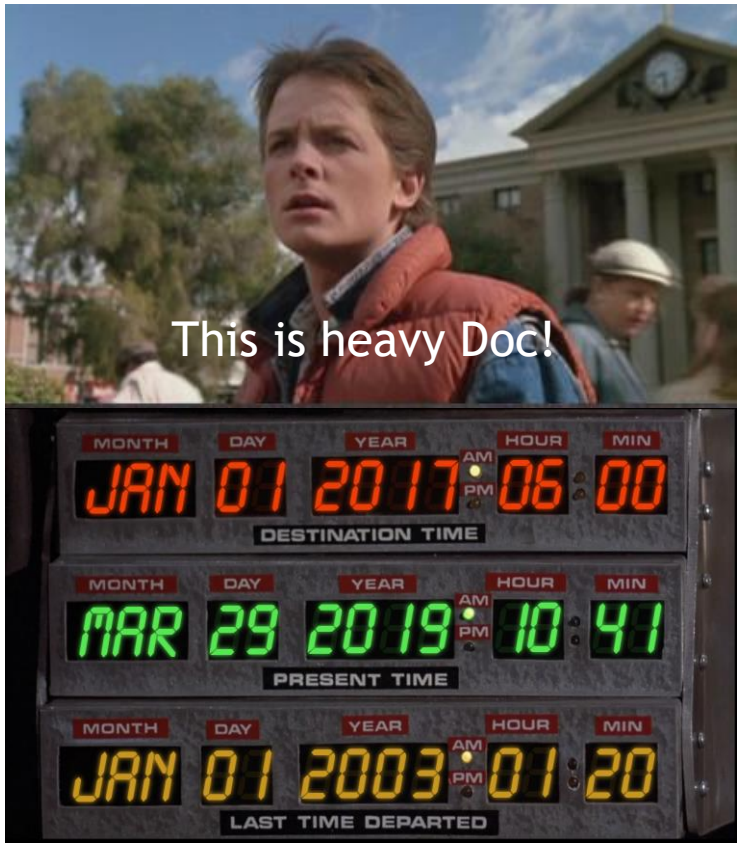
Expanded list of directives that support the “if” clause

Lots of clarifications and clean-up

# Update Base Languages

## Welcome to the new Millennium

- OpenACC 2.7 supports C99, C++98, and Fortran 2003, it's time to catch up
- C, C++, & Fortran all now have native support for parallelism, where do directives fit with this?
- Updating the base languages was a necessary step in being able to interoperate with native parallelism



# C++ Lambdas

## Motivation

- Lambdas provide a simple way to declare an anonymous function close to where it will be used.
- Lambdas may be inline or captured for reuse
- An increasingly common C++ pattern is to create execution policies according to how the code should execute
- The lambda is reused according to the execution policy

```
template <typename Execution_Policy, typename BODY>
double bench_forall ( int s, int e, BODY body ) {
 StartTimer ();
 if ( is_same<Execution_Policy, Serial> :: value ) {
   for ( int i = s; i < e; ++i )
     body ( i );
 } else if ( is_same<Execution_Policy, OpenACC> :: value ) {
   #pragma acc parallel loop
   for ( int i = s; i < e; ++i )
     body ( i );
 } else if ( is_same<Execution_Policy, OpenACC_multicore> ::
value ) {
   #pragma acc parallel loop self
   for ( int i = s; i < e; ++i )
     body ( i );
 }
 return EndTimer ( );
}

using T = double;
void do_bench_daxpy ( int N, T *a, T *b, Tx) {
 auto daxpy = [=]( int i ) /* Capture-by-Value */
   { b[i] += a[i] * x; };

double time = bench_forall<Serial>(0, N, daxpy);
double cputime = bench_forall<OpenACC_multicore>(0, N, daxpy);
double gputime = bench_forall<OpenACC>(0, N, daxpy);
printf ( "OpenACC Multicore Speedup %f \n", time / cputime );
printf ( "OpenACC GPU Speedup %f \n", time / gputime );
}
```

# Lambda expression

## Possible to use lambda in OpenACC region?

**Input code with lambda**

```
void saxpy(int N, float * a, float * b, float x) {
  /* Create a lambda object with Capture-by-Value */
  auto lsaxpy = [=](int i) { b[i] += a[i] * x; };

  /* Use it as loop body */
  #pragma acc parallel loop
  for (int i = 0; i < N; ++i)
     lsaxpy(i);
}
```

**Conceptually generated code**

```
void saxpy(int N, float * a, float * b, float x) {
  class __lambda_7_17 {
  private:
    float *b, *a, x;
  public:
    inline /*constexpr */ void operator()(int i) const {
          b[i] += a[i] * x;
    }
    __lambda_7_17(float * _b, float * _a, float _x)
                  : b{ _b } , a{ _a } , x{ _x } {}
  };

  /* Create lambda object */
  __lambda_7_17 lsaxpy = __lambda_7_17{ b, a, x };

  for (int i = 0; i < N; ++i)
    lsaxpy.operator()(i);
}
```

# lambdas in openacc

## Two major challenges for OpenACC

1. The *operator()* function
   - Fact: There is no associated symbol for *operator()* function.
   - Problem: User cannot put ***acc routine***
   - ***Solution: Lambdas, including operator, are implicitly declared routine seq***
2. Variables that are used in the lambda body
   - Fact: Captured variables become struct member (struct members must be attached)
   - Problem: User cannot attach them explicitly since lambda members are not visible
   - ***Solution: Captured variables given implicit data clauses according to type***

# Multi-device Improvements

## Synchronous & Asynchronous D2D copies without updating host

- OpenACC 3.0 enables direct device-to-device copies

**OpenACC 2.7**

```
// Indirect Copy A from Device 0 to Device 1

// Set the current device to 0
#pragma acc set device(0)
// Update the host copy of A
#pragma acc update self(A[:N])
// Set the current device to 1
#pragma acc set device(1)
// Update the device 1 copy of A
#pragma acc update device(A[:N])
```

**This results in 2 PCIe transfers, blocking the host twice.**

*directives-based version still being designed

**OpenACC 3.0**

```
// Direct Copy A from Device 0 to Device 1

// Set the current device to 0
#pragma acc set device(0)
// Get device 0's pointer
float *srcA = acc_deviceptr(A);
// Set the current device to 1
#pragma acc set device(1)
// Get device 1's pointer
float *dstA = acc_deviceptr(A);
// Update the device 1 copy of A
acc_memcpy_d2d(dstA, srcA, numBytes, 1, 0);
```

**This results in 1 PCIe/NVLINK transfer, blocking the host once (maybe).**

# Multi-device Improvements

## Direct synchronization between devices

- OpenACC 3.0 enables direct device-to-device synchronization

**OpenACC 2.7**

```
// Host-based wait on devices 0 & 1

// Set the current device to 0
#pragma acc set device(0)
// Wait on device 0's queue 0
#pragma acc wait(0)
// Set the current device to 1
#pragma acc set device(1)
// Wait on device 1's queue 100
#pragma acc wait(100)
```

**This results in blocking the host twice.**

**OpenACC 3.0**

```
// Direct waiting across devices w/o blocking

// Set the current device to 1
#pragma acc set device(1)
// Asynchronously have queue 100 on device 1
// wait on queue 0 of device 0
#pragma acc wait(devnum:0,queues:0) async(100)
```

**Work in queue 100 of device 1 will depend on queue 0 on device 0 without blocking the host.**

# Additional Uses for If

## One motivating example

- Motivating example came from a sweep algorithm

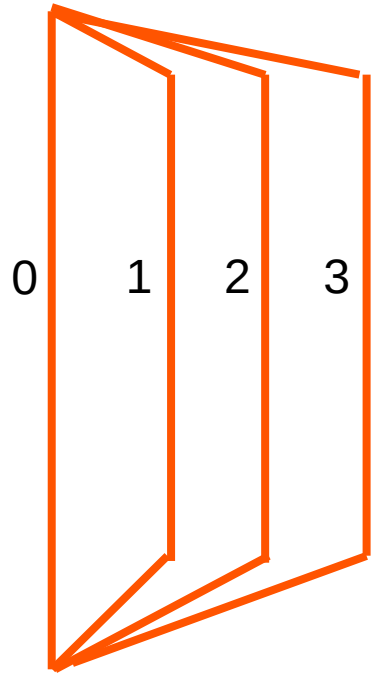
**Creates a single point of entry so all async queues fork from 0.**

**OpenACC 2.7**

```
if (wavefront == 0 && octant>0)
{
// Create single entry to graph on 0
#pragma acc wait(0) async(octant)
  ;
}
```

**OpenACC 3.0**

```
// No Need for additional empty if
#pragma acc wait(0) async(octant) if(wavefront==0)
```



# Zero Modifier

## Zero device arrays upon creation

- Developers can specify arrays to be zeroed on creation without extra kernels or memory transfers.

**OpenACC 2.7**

```
!$acc data create(a)
!$acc kernels
A(:) = 0
... Do stuff
!$acc end kernels
!$acc end data
```

```
float* a = calloc(N, sizeof(float));
#pragma acc data copyin(a[:N])
{
... Do stuff
}
```

**Both cases result in either an extra kernel or data transfer**

**OpenACC 3.0**

```
float* a = calloc(N, sizeof(float));
#pragma acc data create(zero:a[:N])
{
... Do stuff
}
```

**The memory is initialized to zero by whatever means the device supports**

# The Future of Parallel Programming

## Standard Languages | Directives | Specialized Languages

```
std::for_each_n(POL, idx(0), n,
                [&](Index_t i){
    y[i] += a*x[i];
});
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
#pragma acc data copy(x,y) {

...

std::for_each_n(POL, idx(0), n,
                [&](Index_t i){
    y[i] += a*x[i];
});

...

}
```

```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

**Drive Base Languages to Better Support Parallelism**

**Augment Base Languages with Directives**

**Maximize Performance with Specialized Languages & Intrinsics**

# OpenACC's Future in a Parallel World

## Do directives still matter?

- We cannot assume that legacy codes will rush to new language features.
- New developments may need new features, but it's OK if they don't.
- Interoperability and composability are critical
- The base languages provide a portable parallelism, but directives may still provide device-specific optimization paths.

# 3 Swim Lanes

## Different Users with Different Needs

```
#pragma acc data copy(x,y) {

...

#pragma acc parallel loop
for(int i=0; i<N; ++i ) {
    y[i] += a*x[i];
});

...

}
```

Traditional/Legacy OpenACC

Directives for Parallelism & Data Movement, including Deep Copy

```
// #pragma acc data copy(x,y) {

...

#pragma acc parallel loop
for(int i=0; i<N; ++i ) {
    y[i] += a*x[i];
});

...

// }
```

Unified Memory OpenACC

Directives for parallelism, Unified Memory for Data, Locality Rules

```
#pragma acc data copy(x,y) {

...

std::for_each_n(POL, idx(0), n,
                [&](Index_t i){
    y[i] += a*x[i];
});

...

}
```

Parallel Languages & OpenACC

Base languages provide parallelism, OpenACC handles data movement, Locality still Rules.

# My OpenACC Wishlist

- Defined behavior with Fortran `do concurrent`, `block`, and co-arrays.
- Defined behavior with C++ parallelism
- Modernized C++ interface (attributes, namespace, function overloading)
- Directives to help with data layout transformations