



## OPENACC AND UNIFIED MEMORY

Doug Miles, Cray User Group Meeting, 10 May 2017



# OPENACC DIRECTIVES

Data directives are designed to be optional

The diagram illustrates the mapping of OpenACC directives to specific code snippets. Three vertical columns of text on the left point via green arrows to specific parts of the code on the right.

- Manage Data Movement** points to the first line of code: `#pragma acc data copyin(a,b) copyout(c)`.
- Initiate Parallel Execution** points to the parallel loop structure:
  - Starts with `#pragma acc parallel`
  - Includes a brace `{`
  - Includes `#pragma acc loop gang vector`
  - Includes a for loop: `for (i = 0; i < n; ++i) {`
  - Includes assignment code: `c[i] = a[i] + b[i];`
  - Includes ellipsis: `...`
  - Includes a brace `}`
- Optimize Loop Mappings** points to the final brace and ellipsis at the bottom of the code block:
  - Includes a brace `}`
  - Includes ellipsis: `...`

```
#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

Data directives are designed to be optional

The diagram illustrates the optional nature of OpenACC directives by showing arrows pointing from descriptive text on the left to specific code elements in the center.

- An arrow labeled "Manage Data Movement" points to the first line of the code: `#pragma acc data copyin(a,b) copyout(c)`.
- An arrow labeled "Initiate Parallel Execution" points to the `#pragma acc parallel` directive.
- An arrow labeled "Optimize Loop Mappings" points to the `#pragma acc loop gang vector` directive and the nested `for` loop below it.

```
#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

Data directives are designed to be optional

Initiate  
Parallel  
Execution

Optimize  
Loop  
Mappings

```
...
#pragma acc parallel
{
#pragma acc loop gang vector
    for (i = 0; i < n; ++i) {
        c[i] = a[i] + b[i];
    ...
}
...
}
```

# OPENACC FOR MULTICORE CPUS & GPUS

```
98 !$ACC KERNELS
99 !$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 KERNELS
```

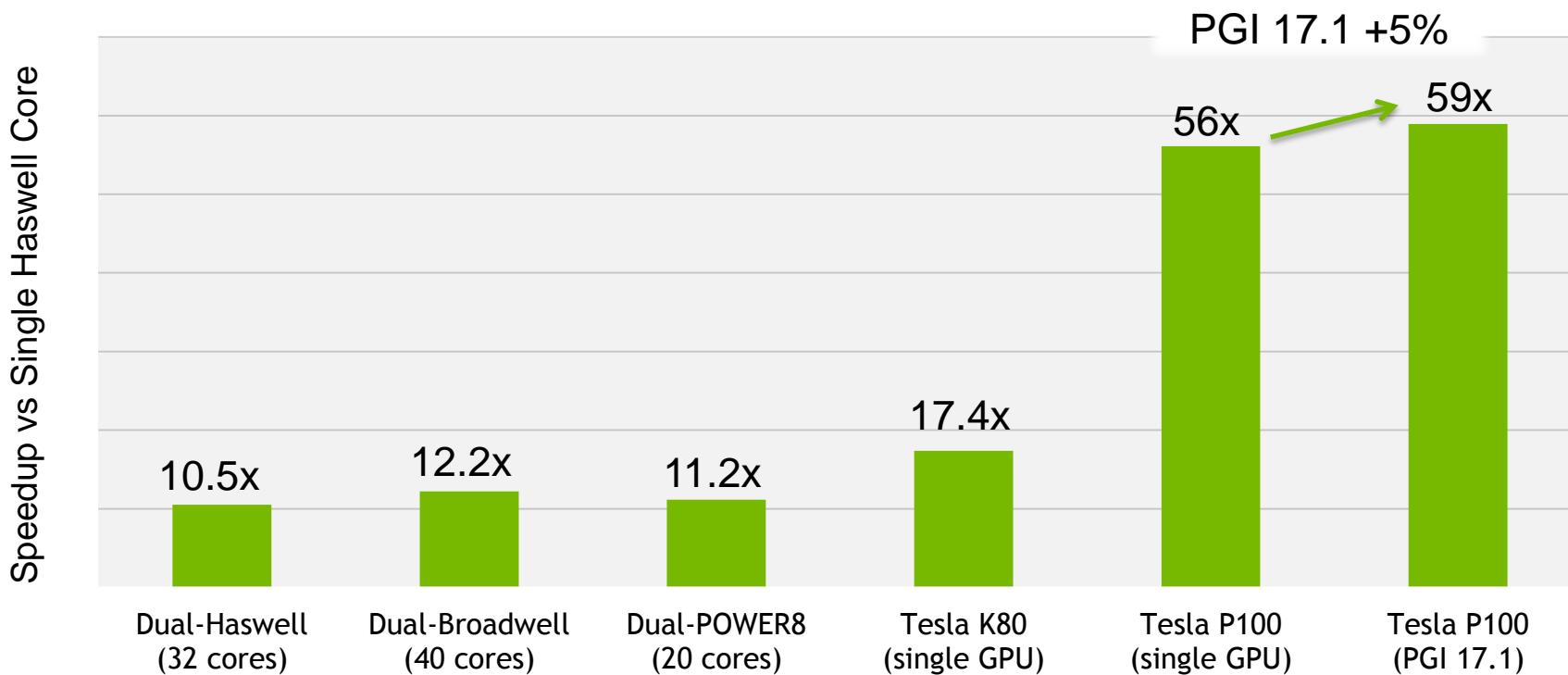


```
% pgfortran -ta=multicore -fast -Minfo=acc -c \
update_tile_halo_kernel.f90
...
100, Loop is parallelizable
Generating Multicore code
100, !$acc loop gang
102, Loop is parallelizable
```

```
% 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
```

# OpenACC SPEC ACCEL 1.1 Benchmarks

Geometric mean across all 15 benchmarks



Performance measured February and March, 2017 and are considered estimates per SPEC run and reporting rules. SPEC® and SPEC ACCEL® are registered trademarks of the Standard Performance Evaluation Corporation ([www.spec.org](http://www.spec.org)).

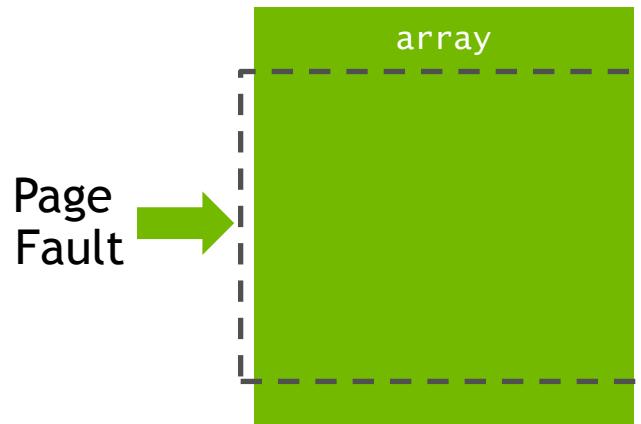
# CUDA UNIFIED MEMORY FOR TESLA

## Servicing CPU and GPU Page Faults

GPU Code

```
__global__
void setvalue(char *ptr, int index, char val)
{
    ptr[index] = val;
}
```

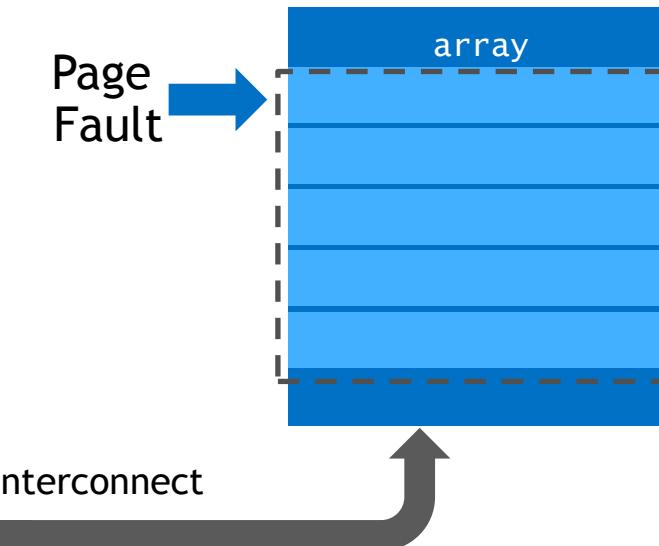
GPU Memory Mapping



CPU Code

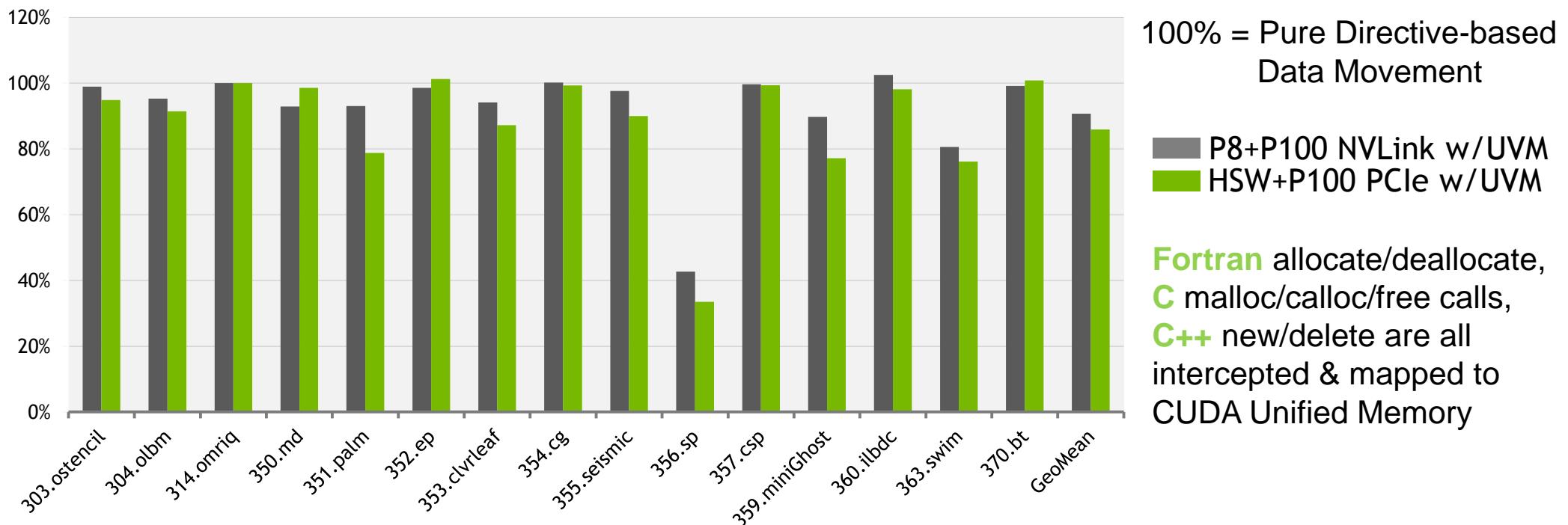
```
cudaMallocManaged(&array, size);
memset(array, size);
setvalue<<<...>>>(array, size/2, 5);
```

CPU Memory Mapping



# OPENACC WITH CUDA UNIFIED MEMORY

P100 Paging Engine Moves All Dynamically Allocated Data



100% = Pure Directive-based  
Data Movement

P8+P100 NVLink w/UVM  
HSW+P100 PCIe w/UVM

Fortran allocate/deallocate,  
C malloc/calloc/free calls,  
C++ new/delete are all  
intercepted & mapped to  
CUDA Unified Memory

# TESLA GPU PROGRAMMING IN 3 STEPS

## PARALLELIZE

Parallelize with OpenACC  
for multicore CPUs

```
% pgc++ -ta=multicore ...  
  
while ( error > tol && ...  
    error = 0.0;  
#pragma acc parallel loop ...  
    for( int j = 1; ...  
#pragma acc loop  
    for( int i = 1; ...  
    ...  
}  
...
```

## OFFLOAD

Port to Tesla using OpenACC  
with CUDA Unified Memory

## OPTIMIZE

Optimize and overlap data  
movement using OpenACC  
data directives

# TESLA GPU PROGRAMMING IN 3 STEPS

## PARALLELIZE

Parallelize with OpenACC  
for multicore CPUs

```
% pgc++ -ta=multicore ...  
  
while ( error > tol && ...  
    error = 0.0;  
#pragma acc parallel loop ...  
    for( int j = 1; ...  
#pragma acc loop  
        for( int i = 1; ...  
        ...  
    }  
...  
}
```

## OFFLOAD

Port to Tesla using OpenACC  
with CUDA Unified Memory

```
% pgc++ -ta=tesla:managed ...  
  
while ( error > tol && ...  
    error = 0.0;  
#pragma acc parallel loop ...  
    for( int j = 1; ...  
#pragma acc loop  
        for( int i = 1; ...  
        ...  
    }  
...
```

## OPTIMIZE

Optimize and overlap data  
movement using OpenACC  
data directives

# TESLA GPU PROGRAMMING IN 3 STEPS

## PARALLELIZE

Parallelize with OpenACC  
for multicore CPUs

```
% pgc++ -ta=multicore ...  
  
while ( error > tol && ...  
       error = 0.0;  
#pragma acc parallel loop ...  
       for( int j = 1; ...  
#pragma acc loop  
       for( int i = 1; ...  
       ...  
     }  
...  
}
```

## OFFLOAD

Port to Tesla using OpenACC  
with CUDA Unified Memory

```
% pgc++ -ta=tesla:managed ...  
  
while ( error > tol && ...  
       error = 0.0;  
#pragma acc parallel loop ...  
       for( int j = 1; ...  
#pragma acc loop  
       for( int i = 1; ...  
       ...  
     }  
...
```

## OPTIMIZE

Optimize and overlap data  
movement using OpenACC  
data directives

```
#pragma acc data create ...  
while ( error > tol && ...  
       error = 0.0;  
#pragma acc parallel loop ...  
       for( int j = 1; ...  
#pragma acc loop  
       for( int i = 1; ...  
       ...  
     }  
...
```



# PGI® COMPILERS & TOOLS

