PGI® COMPILERS & TOOLS

DIRECTIVE BASED GPU PROGRAMMING WORKSHOP @ CSCS

Markus Wetzstein, 31.05.2017

NVIDIA.

OPENACC ADOPTION

OPENACC ADOPTION

NEW PLATFORMS

OpenPOWER™

Sunway TaihuLight
#1 Top 500, Nov. 2016

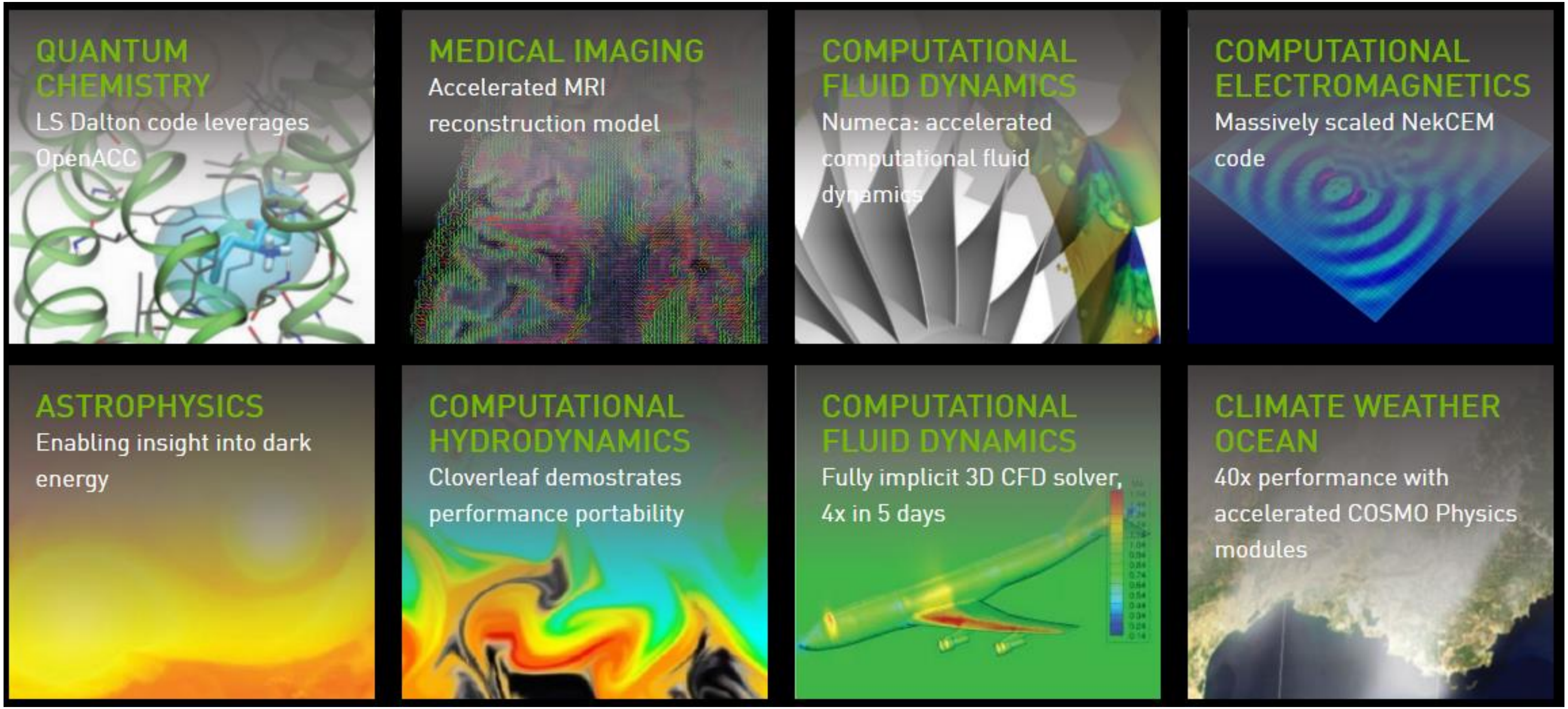
GROWING COMMUNITY

- 6,000+ enabled developers
- 4,500+ course registrants
- 250+ Hackathon attendees
- 150+ User Group members

PORTING SUCCESS

- Gaussian 16 ported to Tesla with OpenACC
- Five of 13 ORNL CAAR codes using OpenACC
- ANSYS Fluent R18 production release is GPU accelerated using OpenACC

OPENACC SUCCESS STORIES



developer.nvidia.com/openacc/success-stories

THE NVIDIA HPC SDK

PGI Fortran, C & C++ Compilers

- Optimizing, SIMD Vectorizing, OpenMP

Accelerated Computing Features

- OpenACC Directives
- CUDA Fortran

Multi-Platform Solution

- Multicore x86-64 and OpenPOWER CPUs, NVIDIA Tesla GPUs
- Supported on Linux, macOS, Windows

MPI/OpenMP/OpenACC Tools

- Debugger
- Performance Profiler
- Interoperable with DDT, TotalView

PGI®

The Compilers & Tools for Supercomputing

OPENACC FOR EVERYONE

New PGI Community Edition Now Available

	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

USING THE PGI COMPILER SUITE

INVOKING THE COMPILER

On Cray machines like Piz Daint @ CSCS:

```
module load PrgEnv-pgi
module avail pgi
module load pgi/…
module load craype-accel-nvidiaXX
```

The wrappers will then point to the PGI compilers: `ftn, cc, CC`

Without Cray wrappers: `pgfortran, pgcc, pgc++`

Useful basic options: `-fast` for optimizations, `-Minfo` for compiler feedback

If you want to see the possible arguments to a given option and check what the default is: `pgf90 -help [some other option]`

BUILDING ACCELERATOR PROGRAMS

```
pgfortran -acc a.f90
pgcc -acc a.c
pgc++ -acc a.cpp
```

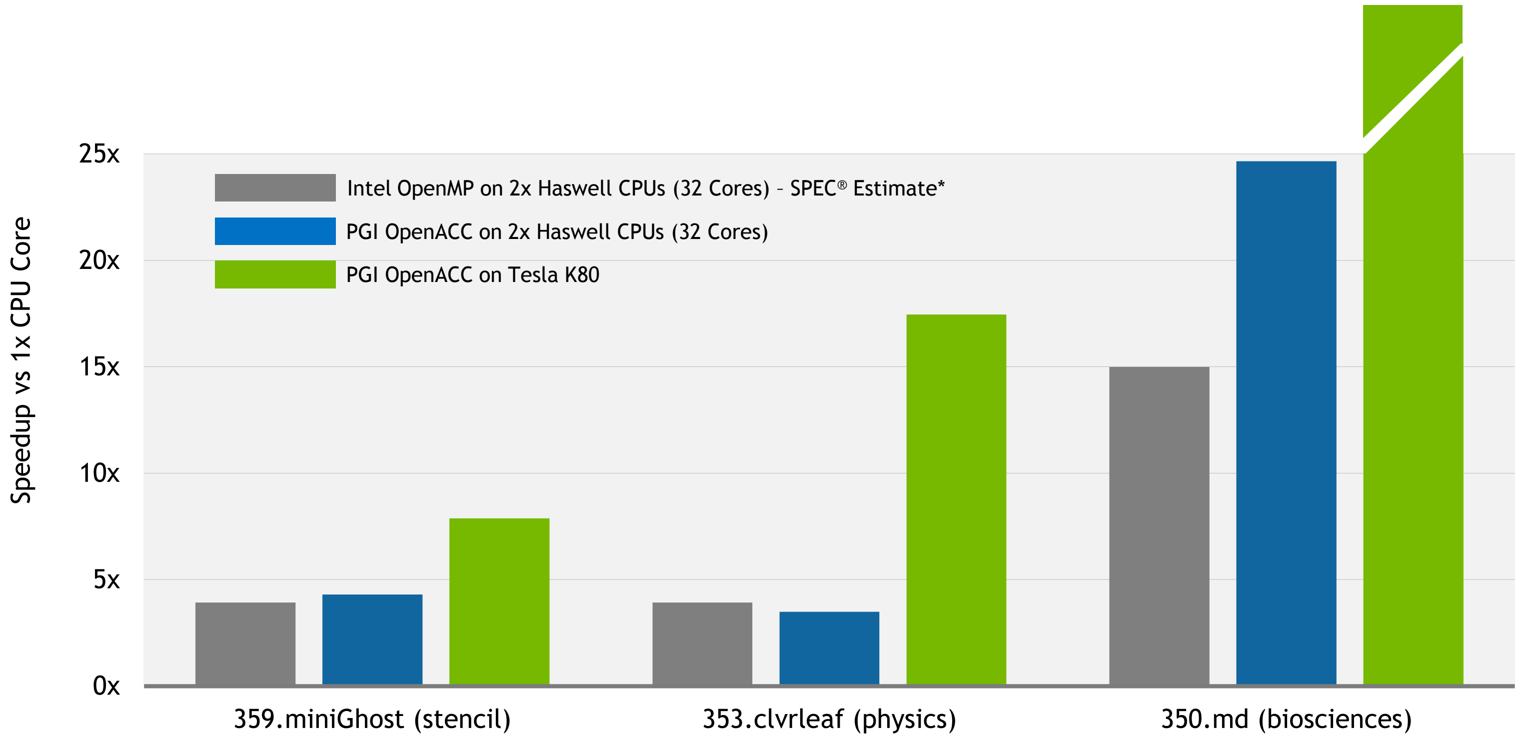
Other options:

```
-ta=tesla[:cc2x|cc3x|cc50|cc60]
-ta=tesla[:cuda7.0|cuda7.5|cuda8.0]
-ta=tesla,host  [default with -acc]
-ta=multicore
```

RECOMMENDED: Enable compiler feedback with `-Minfo` or `-Minfo=accel`

PGI OPENACC FOR MULTI-CORE X86 CPUS

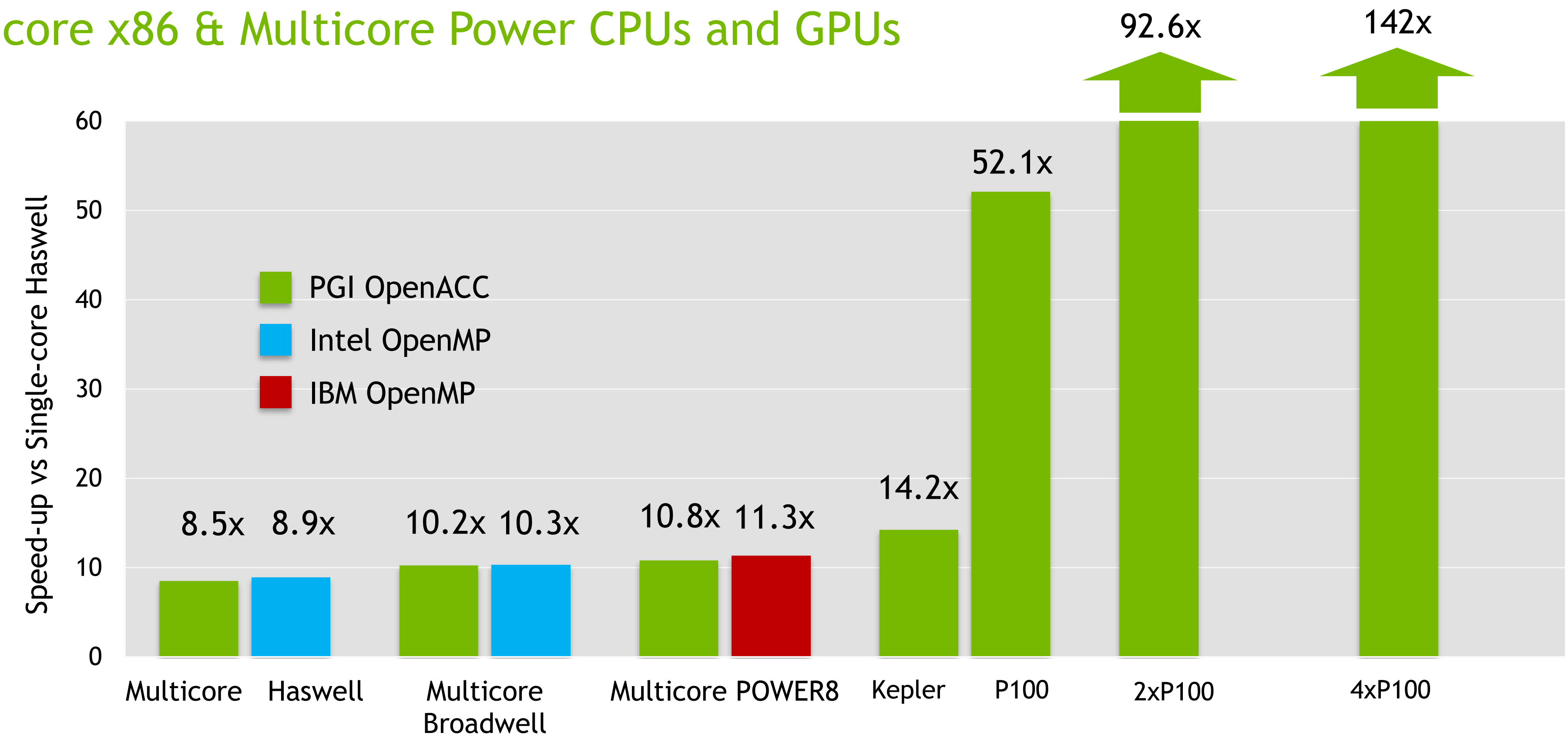
OpenACC Performance Portability Across Multicore x86 CPUs and GPUs



Supermicro SYS-2028GR-TRT, Intel Xeon E5-2698 v3, 32 cores, NVIDIA Tesla K80, 256GB of System Memory
PGI 15.7 Beta OpenACC Multicore and K80 results from SPEC ACCEL™ measured June 2015.
** Intel 15.0.90 OpenMP results use Cloverleaf reference application and SPEC OMP®2012 using workloads from SPEC ACCEL.*
SPEC® and the benchmark names SPEC ACCEL™ and SPEC OMP® are registered trademarks of the Standard Performance Evaluation Corporation.
More info: SPEC ACCEL www.spec.org/accel, OMP2012 www.spec.org/omp2012, CloverLeaf OpenMP uk-mac.github.io/CloverLeaf/, miniGhost ref 1.0.1 https://mantevo.org/download/

OPENACC PERFORMANCE PORTABILITY - CLOVERLEAF 1.3

Multicore x86 & Multicore Power CPUs and GPUs



AWE Hydrodynamics CloverLeaf mini-App, bm32 data set

PGI_ACC_TIME ENVIRONMENT VARIABLE

```
Accelerator Kernel Timing data
/proj/scratch/mwolfe/test/openacc/src/smooth4.c
  smooth_acc  NVIDIA  devicenum=0
    time(us): 317
    12: data region reached 5 times
        12: data copyin reached 10 times
             device time(us): total=121 max=19 min=11 avg=12
        23: data copyout reached 5 times
             device time(us): total=63 max=14 min=12 avg=12
    14: compute region reached 5 times
        17: kernel launched 5 times
            grid: [1x98]  block: [128]
             device time(us): total=133 max=90 min=9 avg=26
            elapsed time(us): total=176 max=99 min=17 avg=35
```

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
```

PGPROF COMMAND LINE PROFILER

`pgprof ./exe`

Report kernel and transfer times directly

Collect profiles for PGPROF GUI

```
%> pgprof  -o profile.out ./exe
```

Collect for MPI processes

```
%> mpirun –np 2 pgprof -o profile.%p.out ./exe
```

Collect profiles for complex process hierarchies

```
--profile-child-processes, --profile-all-processes
```

Collect key events and metrics

```
%> pgprof –events all flops_sp ./exe
```

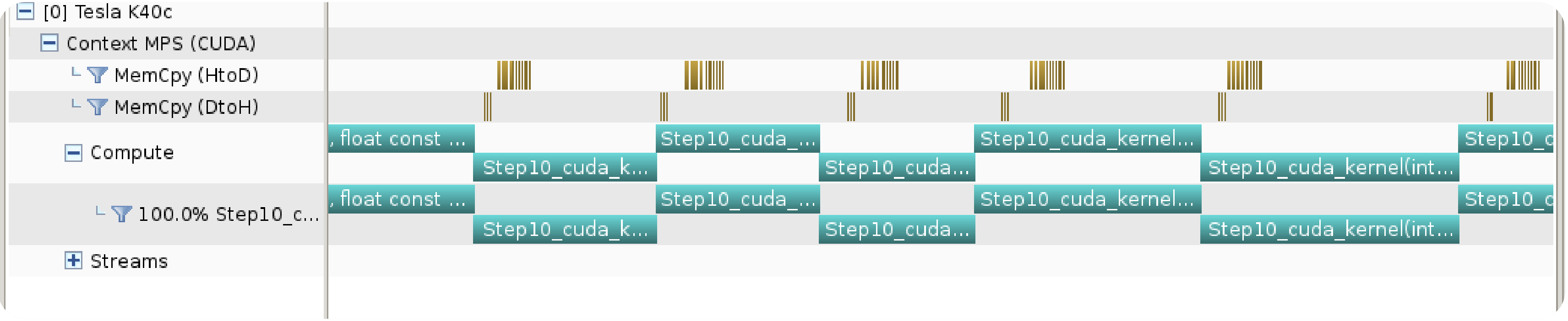
Trace stream usage

```
%> pgprof –print-gpu-trace ./exe
```

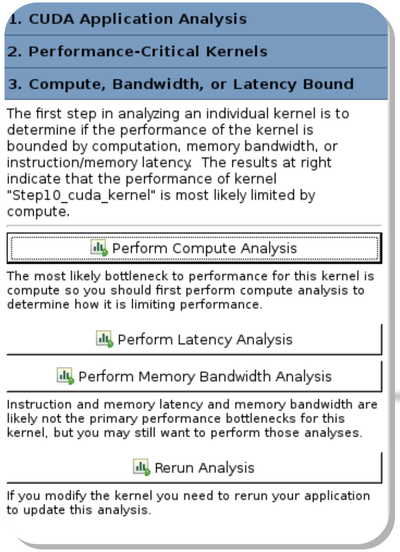
Full listing of options see: `pgprof --help`

PGPROF VISUAL PROFILER

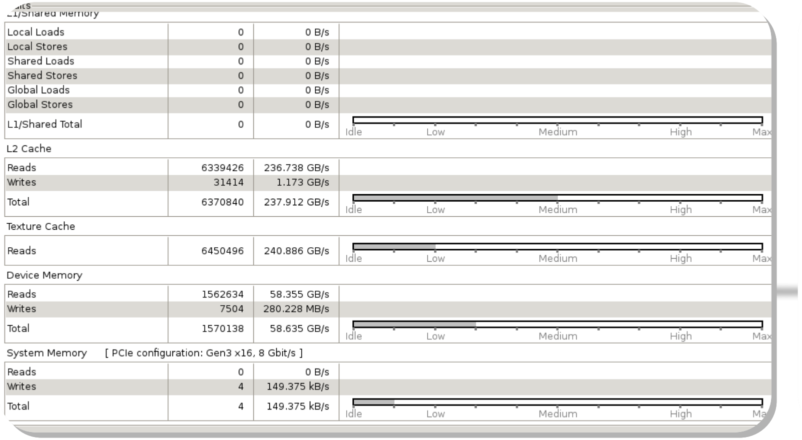
Timeline



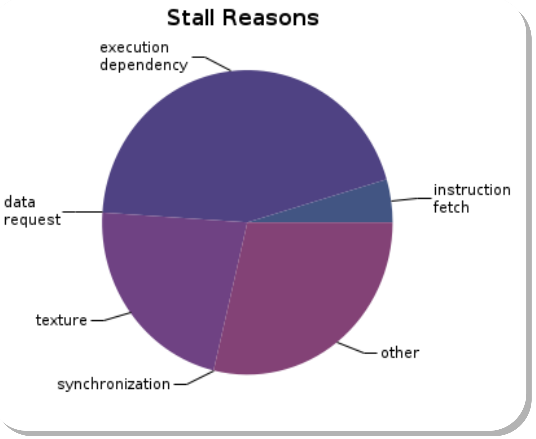
Guided System



Analysis



L1/Shared Memory			
Local Loads	0	0 B/s	
Local Stores	0	0 B/s	
Shared Loads	0	0 B/s	
Shared Stores	0	0 B/s	
Global Loads	0	0 B/s	
Global Stores	0	0 B/s	
L1/Shared Total	0	0 B/s	Idle Low Medium High Max
L2 Cache			
Reads	6339426	236.738 GB/s	
Writes	31414	1.173 GB/s	
Total	6370840	237.912 GB/s	Idle Low Medium High Max
Texture Cache			
Reads	6450496	240.886 GB/s	Idle Low Medium High Max
Device Memory			
Reads	1562634	58.355 GB/s	
Writes	7504	280.228 MB/s	
Total	1570138	58.635 GB/s	Idle Low Medium High Max
System Memory [PCIe configuration: Gen3 x16, 8 Gbit/s]			
Reads	0	0 B/s	
Writes	4	149.375 kB/s	
Total	4	149.375 kB/s	Idle Low Medium High Max



OPENACC DIRECTIVES: ROUTINE DECLARE ATOMIC

OPENACC routine DIRECTIVE

routine: Compile the following function for the device (allows a function call in device code)

Clauses: gang, worker, vector, seq

```
#pragma acc routine seq
void fun(…) {

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

```
#pragma acc routine vector
void fun(…) {
  #pragma acc loop vector
  for(int i=0;i<N;i++)
    ...
}
```

OPENACC routine DIRECTIVE

routine: Compile the following function for the device (allows a function call in device code)

Clauses: gang, worker, vector, seq

```
#pragma acc parallel loop gang \
            vector_length(VL)
   for(int i=0;i<N;i++)
    fun_vec(…);
}
```

```
#pragma acc routine vector
void fun_vec(…) {
  #pragma acc loop vector
  for(int i=0;i<N;i++)
    fun_seq(…);
}
```

```
#pragma acc routine seq
void fun_seq(…) {

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

OPENACC routine: FORTRAN

```
subroutine foo(v, i, n) {
  use …
  !$acc routine vector
  real :: v(:,:)
  integer, value :: i, n
  !$acc loop vector
  do j=1,n
    v(i,j) = 1.0/(i*j)
  enddo
end subroutine

!$acc parallel loop
do i=1,n
  call foo(v,i,n)
enddo
!$acc end parallel loop
```

The **routine** directive may appear in a Fortran function or subroutine definition, or in an interface block.

Nested acc routines require the routine directive within each nested routine.

The save attribute is not supported.

Note: Fortran, by default, passes all arguments by reference. Passing scalars by value will improve performance of GPU code.

SEQ ROUTINE AUTO-GENERATION

In a C++ program, many functions, certainly class member functions, appear as source code in header files included in the program. Oftentimes, the functions are defined in system headers or other application packages, and modifying those headers is either unwise or impossible.

The PGC++ compiler will take note of functions called in compute regions and implicitly add the pragma `acc routine seq` if there is no explicit `routine` directive.

GLOBAL DATA

```
float a[1000000];

extern void matvec(…);
…

for (i = 0; i < m; ++i) {
   matvec(v, x, i, n);
}
```

```
extern float a[];

void matvec (float* v, float* x, int
I, int n) {
   for (int j = 0; n < n; ++j)
      x[i] += a[i*n+j] * v[j];
}
```

DECLARE CREATE

```
float a[1000000];
#pragma acc declare create(a)

#pragma acc routine worker
extern void matvec(…);
…

#pragma acc parallel loop
for (i = 0; i < m; ++i) {
   matvec(v, x, i, n);
}
```

```
extern float a[];
#pragma acc declare create(a)

#pragma acc routine worker
void matvec (float* v, float* x, int
I, int n) {
   #pragma acc loop worker
   for (int j = 0; n < n; ++j)
      x[i] += a[i*n+j] * v[j];
}
```

GLOBAL DATA

Fortran:

```
module m
 real, allocatable :: x(:)
 !$acc declare create(x)
 ! x dynamically allocated on host+device
end module
```

Summary **`declare`** directive:

Used to generate a device declaration for global data.

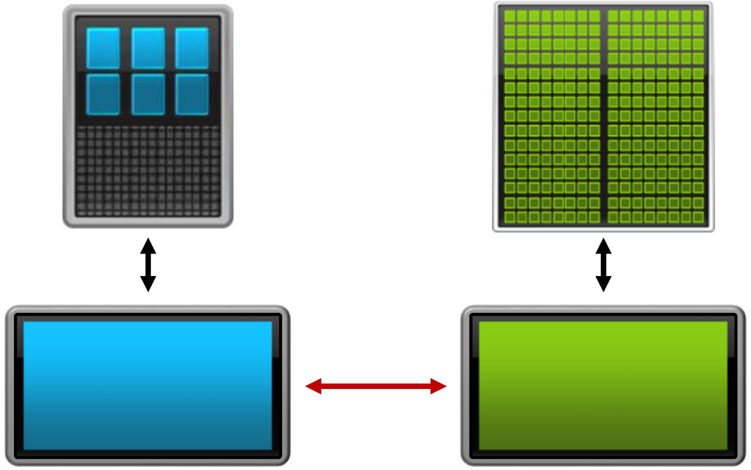
There are more complicated use cases and additional clauses to **`declare`**, but **`declare create`** is most likely what you will need for most scenarios.

NVIDIA UNIFIED MEMORY

UNIFIED MEMORY

Dramatically Lower Developer Effort

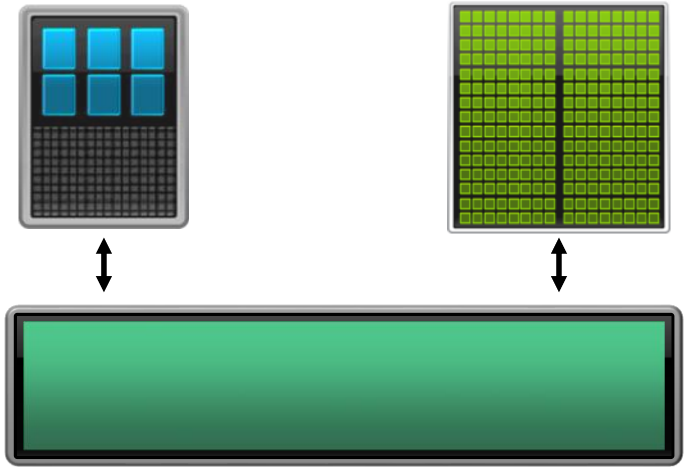
Developer View Today



System Memory

GPU Memory

Developer View With Unified Memory



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; ...
        ...
        }
...
```

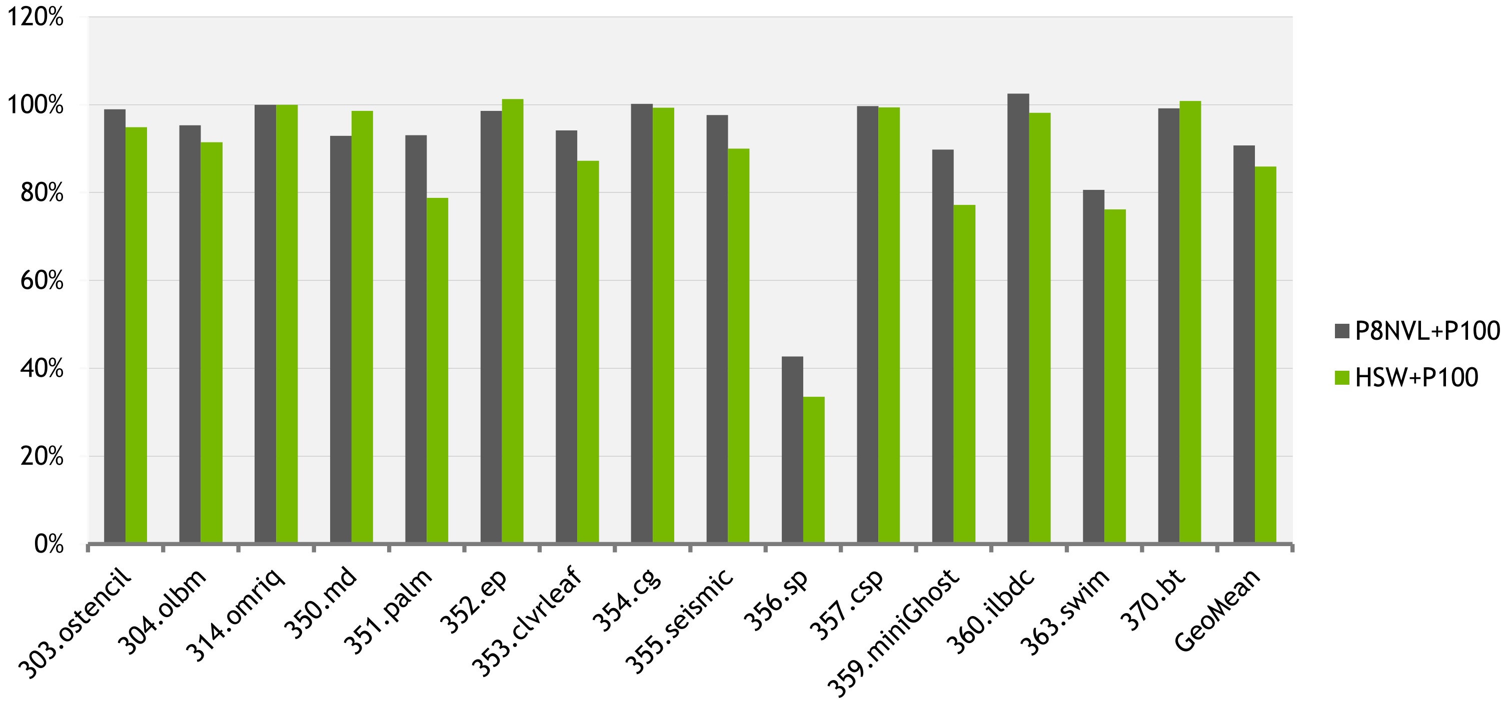
OPTIMIZE

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; ...
        ...
        }
...
```

CUDA UNIFIED MEMORY ON NVLINK VS PCIE

P100 Paging Engine Moves All Dynamically Allocated Data



100% = Directive-based Data Movement

OpenACC w/CUDA 8.0 UM using -ta=tesla:managed

CUDA UM performance is within 10% -15% of direct-based data movement

356.sp anomaly: Fortran automatics in a routine with accelerated loops

351.palm: lots of paging between host and device

PGI 17.1 Compilers OpenACC SPEC ACCEL™ 1.1 performance measured March, 20167 SPEC® and the benchmark name SPEC ACCEL™ are registered trademarks of the Standard Performance Evaluation Corporation.

OPENACC 2.5 SPECIFICATION AND OUTLOOK ON 2.6

OPENACC 2.5 FEATURES

(incomplete, see specification for full list)

- `num_gangs, num_workers, vector_length` are now allowed on `kernels` construct
- New directives: `init, shutdown, set`
- New `if_present` clause for `update` directive
- `acc routine` without `gang/worker/vector/seq` is now an error
- New `default(present)` clause was added for compute constructs
- New API routines `acc_get_default_async, acc_set_default_async`
- Asynchronous versions of data API routines added
- new OpenACC interface for profile and trace tools was added
- various clarifications added to the text

OPENACC 2.5 FEATURES (CONT.)

(incomplete, see specification for full list)

- Reference counting is now used to manage the correspondence and lifetime of device data
- Data clauses `copy[in/out]`, `create` changed to behave as `present_or_copy[in/out]`, `present_or_create`. Similar change for API routines `acc_copyin`, etc.
- The behavior of the `exit data` directive has changed to decrement the dynamic reference count. A new optional `finalize` clause was added to set the dynamic reference count to zero.

```
#pragma acc data copy(var)
{…
   #pragma acc data copy(var)
   {…}
}
```

```
Reference count for var: 0
1, copyin

2, no copyin!
1 after }, no copyout!
0, copyout
```

LIKELY OPENACC 2.6 FEATURES

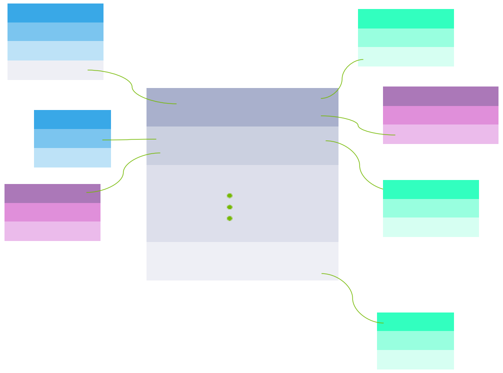
(2.6 specification not finalized yet!)

- manual deep copy in the spec (already supported by PGI)
- `no_create` clause (user request)
- `if` and `if_present` clauses on `host_data` directive (user request)
- standardize behavior of Fortran optional arguments in data clauses (user request)
- Fortran bindings for all API routines (user request)
- serial offload region (user request)
- device query routines
- method to catch errors and fail gracefully (user request; this is not error recovery, but would allow an MPI program to shut down)
- a number of small cleanup items

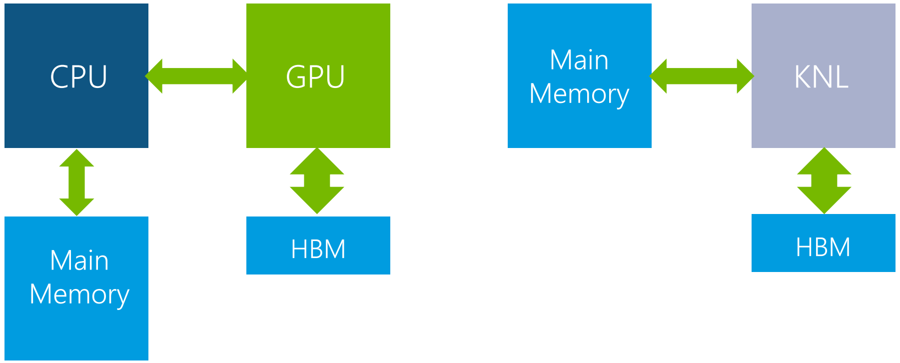
DEEP COPY

OPENACC 3.0 DEEP COPY

Modern Data Structures



Modern HPC Node Architectures



MANUAL DEEP COPY

```
typedef struct points {
    float* x;  float* y;  float* z;
    int n;
    float coef, direction;
} points;


void sub ( int n, float* y ) {
    points p;
    #pragma acc data create (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 update device (p.n)
        #pragma acc data copyin (p.x[0:n], p.y[0: n])
        {
            #pragma acc parallel loop
            for ( i =0; i<p.n; ++I ) p.x[i] += p.y[i];
            . . .
```

TRUE DEEP COPY

```
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];
            . . .
```

PGI

QUESTIONS ?

Check out www.openacc.org for online resources, news, events,etc.

Join the OpenACC usergroup, e.g. through
https://www.openacc.org/community