PGI® COMPILERS &TOOLS

OPENACC INTRODUCTION AND APPLICATIONS UPDATE

PGI Compilers for Heterogeneous Supercomputing, September 2017

NVIDIA

PGI — THE NVIDIA HPC SDK

- Fortran, C & C++ Compilers
  - Optimizing, SIMD Vectorizing, OpenMP
- Accelerated Computing Features
  - OpenACC Directives, CUDA Fortran
- Multi-Platform Solution
  - X86-64 and OpenPOWER Multicore CPUs
  - NVIDIA Tesla GPUs
  - Supported on Linux, macOS, Windows
- MPI/OpenMP/OpenACC Tools
  - Debugger
  - Performance Profiler
  - Interoperable with DDT, TotalView

NVIDIA

PGI®

The Compilers & Tools for Supercomputing

THREE WAYS TO BUILD A FASTER COMPUTER

Faster Clock

More Work per Clock

Don't Stall

OPENACC FOR EVERYONE

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

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

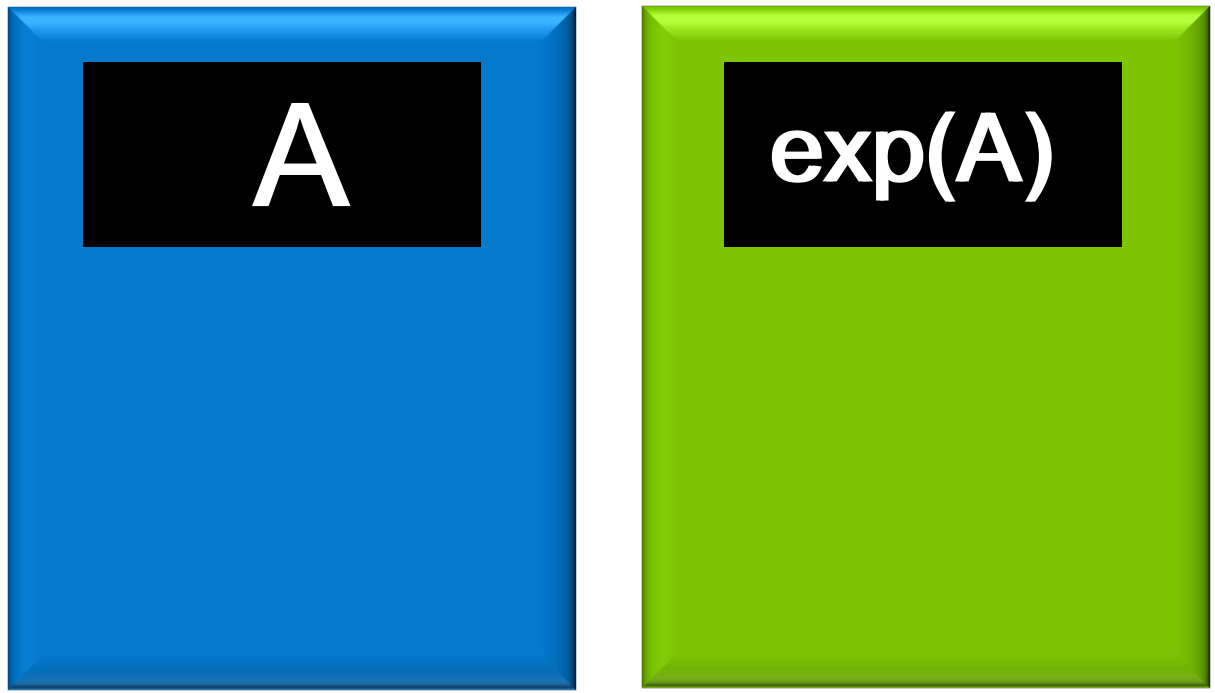
OpenACC
More Science, Less Programming

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

OPENACC KERNELS

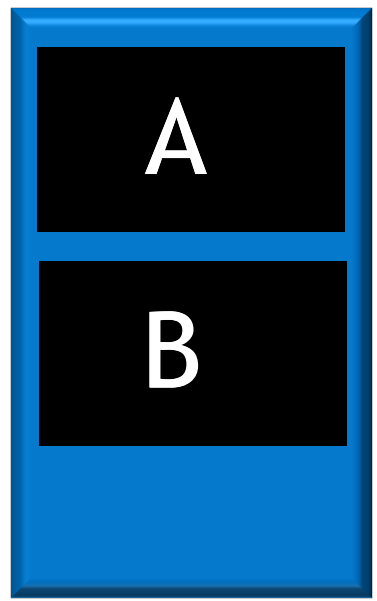
Easy On-ramp to GPU programming

```
. . .
< statement >
< statement >
!$acc kernels
  do i = 1, N
     A(i) = exp(A(i))
  enddo
!$acc end kernels
< statement >
. . .
```

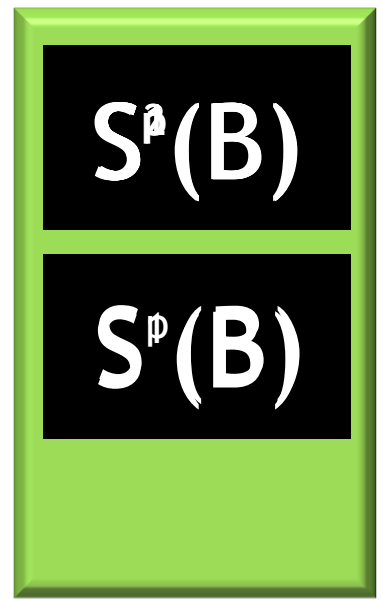


OPENACC DATA REGIONS

```
...
#pragma acc data copy(b[0:n][0:m]) \
                 create(a[0:n][0:m])
{
for (iter = 1; iter <= p; ++iter)
  #pragma acc kernels
  {
  for (i = 1; i < n-1; ++i){
    for (j = 1; j < m-1; ++j){
      a[i][j]=w0*b[i][j]+
              w1*(b[i-1][j]+b[i+1][j]+
                  b[i][j-1]+b[i][j+1])+
              w2*(b[i-1][j-1]+b[i-1][j+1]+
                  b[i+1][j-1]+b[i+1][j+1]);
  } }
  for( i = 1; i < n-1; ++i )
    for( j = 1; j < m-1; ++j )
      b[i][j] = a[i][j];
  }
}
}
...
```



Host Memory



Accelerator Memory

OPENACC IS FOR MULTICORE, MANYCORE & 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
```

CPU

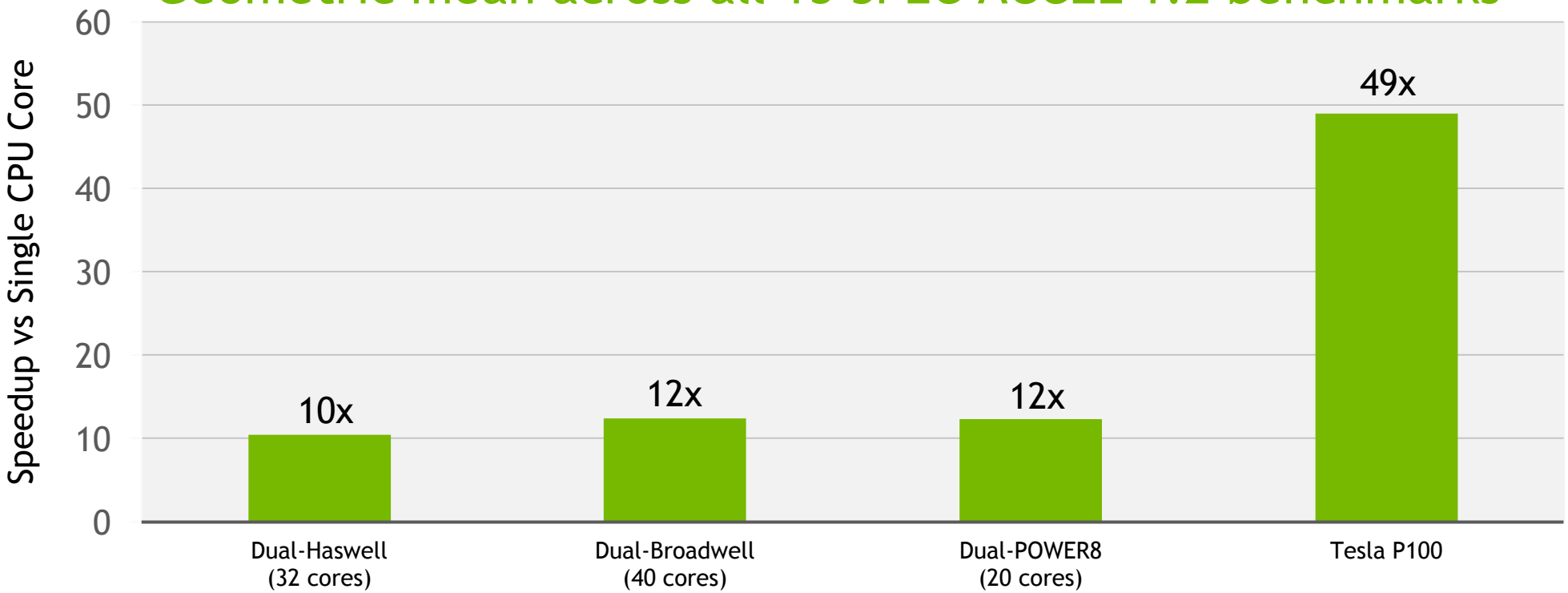
```
% pgfortran -ta=multicore –fast –Minfo=acc -c \
  update_tile_halo_kernel.f90
  . . .
  100, Loop is parallelizable
       Generating Multicore code
       100, !$acc loop gang
  102, Loop is parallelizable
```

GPU

```
% pgfortran -ta=tesla,cc35,cc60 –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 17.7 OPENACC PERFORMANCE

Geometric mean across all 15 SPEC ACCEL 1.2 benchmarks



Performance measured August, 2017 using PGI 17.7 compilers 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).

CLOVERLEAF V1.3

AWE Hydrodynamics mini-app

6500+ lines, !$acc kernels

OpenACC or OpenMP

Source on GitHub

http://uk-mac.github.io/CloverLeaf

OPENACC DIRECTIVES

http://uk-mac.github.io/CloverLeaf

```
 75 !$ACC KERNELS
 76 !$ACC LOOP INDEPENDENT
 77     DO k=y_min,y_max
 78 !$ACC LOOP INDEPENDENT PRIVATE(right_flux,left_flux,top_flux,bottom_flux,total_flux,
                                  min_cell_volume,energy_change,recip_volume)
 79       DO j=x_min,x_max
 80
 81         left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                &
 82                                     +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
 83         right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                &
 84                                     +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
 85         bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                &
 86                                     +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
 87         top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                &
 88                                     +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
 89         total_flux=right_flux-left_flux+top_flux-bottom_flux
 90
 91         volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)
 92
 93         min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
 94                            ,volume(j,k)+right_flux-left_flux                      &
 95                            ,volume(j,k)+top_flux-bottom_flux)
 97         recip_volume=1.0/volume(j,k)
 99         energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
101         energy1(j,k)=energy0(j,k)-energy_change
103         density1(j,k)=density0(j,k)*volume_change(j,k)
105       ENDDO
106     ENDDO
107 !$ACC END KERNELS
```

OPENACC DIRECTIVES FOR GPUS

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

```
75 !$ACC KERNELS
76 !$ACC LOOP INDEPENDENT
77     DO k=y_min,y_max
78 !$ACC LOOP INDEPENDENT PRIVATE(right_flux,left_flux,top_flux,bottom_flux,total_flux,
                                 min_cell_volume,energy_change,recip_volume)
79       DO j=x_min,x_max
80
81         left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                &
82                                     +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
83         right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                &
84                                     +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
85         bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                &
86                                     +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
87         top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                &
88                                     +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
89         total_flux=right_flux-left_flux+top_flux-bottom_flux
90
91         volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)
92
93         min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
94                            ,volume(j,k)+right_flux-left_flux                      &
95                            ,volume(j,k)+top_flux-bottom_flux)
97         recip_volume=1.0/volume(j,k)
99         energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
101        energy1(j,k)=energy0(j,k)-energy_change
103        density1(j,k)=density0(j,k)*volume_change(j,k)
105      ENDDO
106    ENDDO
107 !$ACC END KERNELS
```

OPENACC DIRECTIVES FOR MULTICORE CPUS

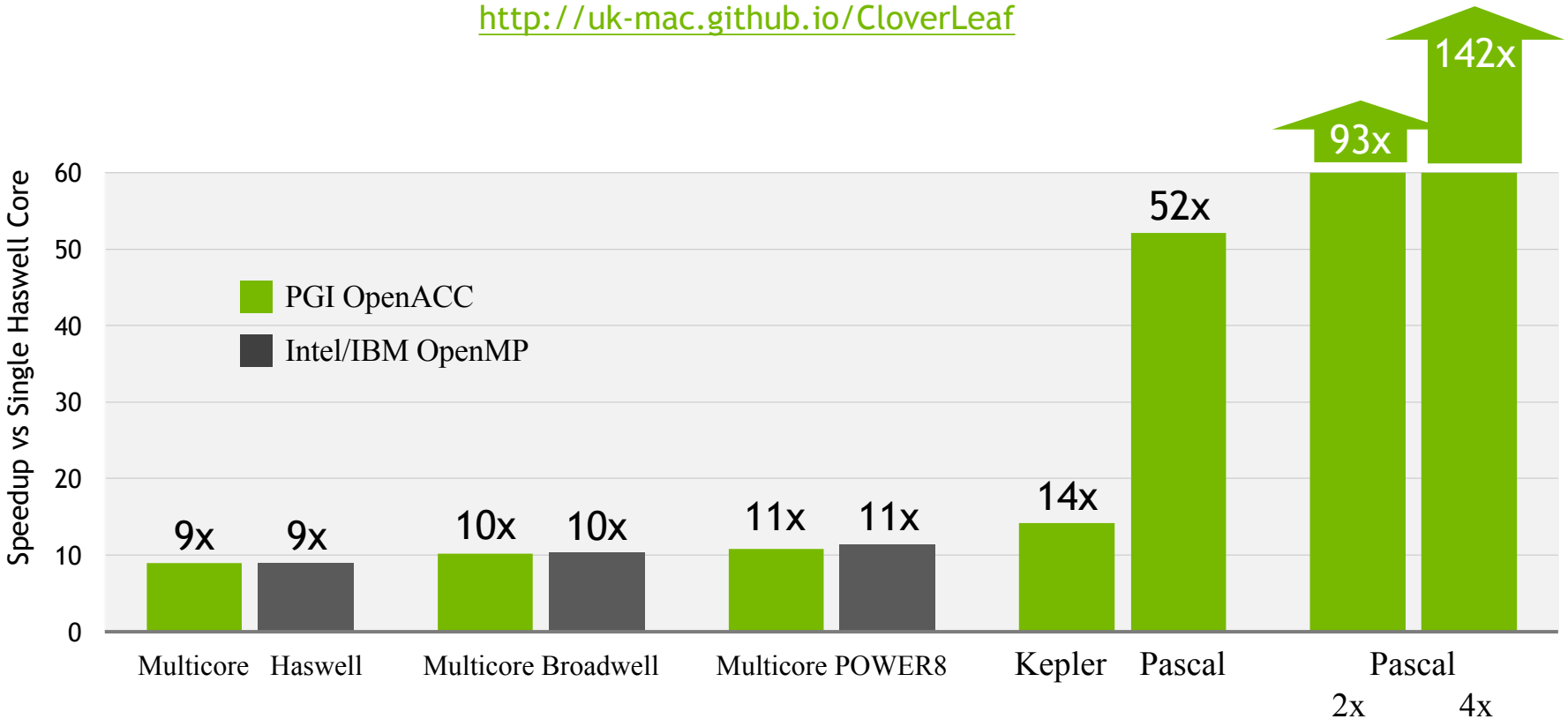
```
% pgfortran -fast -ta=multicore ... PdV_kernel.f90
pdv_kernel:
     ...
     77, Loop is parallelizable
         Generating Multicore code
         77, !$acc loop gang
     79, Loop is parallelizable
         3 loop-carried redundant expressions removed
           with 9 operations and 9 arrays
         Innermost loop distributed: 2 new loops
         Generated vector SIMD code for the loop
         Generated 2 prefetch instructions for the loop
         Generated 12 prefetch instructions for the loop
     ...
```

```
75 !$ACC KERNELS
76 !$ACC LOOP INDEPENDENT
77     DO k=y_min,y_max
78 !$ACC LOOP INDEPENDENT PRIVATE(right_flux,left_flux,top_flux,bottom_flux,total_flux,
                                 min_cell_volume,energy_change,recip_volume)
79       DO j=x_min,x_max
80
81         left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                &
82                                     +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
83         right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                &
84                                     +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
85         bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                &
86                                     +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
87         top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                &
88                                     +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
89         total_flux=right_flux-left_flux+top_flux-bottom_flux
90
91         volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)
92
93         min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
94                            ,volume(j,k)+right_flux-left_flux                      &
95                            ,volume(j,k)+top_flux-bottom_flux)
97         recip_volume=1.0/volume(j,k)
99         energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
101        energy1(j,k)=energy0(j,k)-energy_change
103        density1(j,k)=density0(j,k)*volume_change(j,k)
105      ENDDO
106    ENDDO
107 !$ACC END KERNELS
```

OPENACC PERFORMANCE PORTABILITY

AWE Hydrodynamics CloverLeaf mini-App, bm32 data set

http://uk-mac.github.io/CloverLeaf



Systems: Haswell: 2x16 core Haswell server, four K80s, CentOS 7.2 (perf-hsw10), Broadwell: 2x20 core Broadwell server, eight P100s (dgx1-prd-01), Minsky: POWER8+NVLINK, four P100s, RHEL 7.3 (gsn1.
Compilers: Intel 17.0.1, IBM XL 13.1.3, PGI 16.10.
Benchmark: CloverLeaf v1.3 downloaded from http://uk-mac.github.io/CloverLeaf the week of November 7 2016; CloverlLeaf_Serial; CloverLeaf_ref (MPI+OpenMP); CloverLeaf_OpenACC (MPI+OpenACC)
Data compiled by PGI November 2016.

PGI

GAUSSIAN 16

A Leading Computation Chemistry Code

Valinomycin
wB97xD/6-311+(2d,p) Freq
2.25X speedup

Hardware: HPE server with dual Intel Xeon E5-2698 v3 CPUs (2.30GHz ; 16 cores/chip), 256GB memory and 4 Tesla K80 dual GPU boards (boost clocks: MEM 2505 and SM 875). Gaussian source code compiled with PGI Accelerator Compilers (16.5) with OpenACC (2.5 standard).

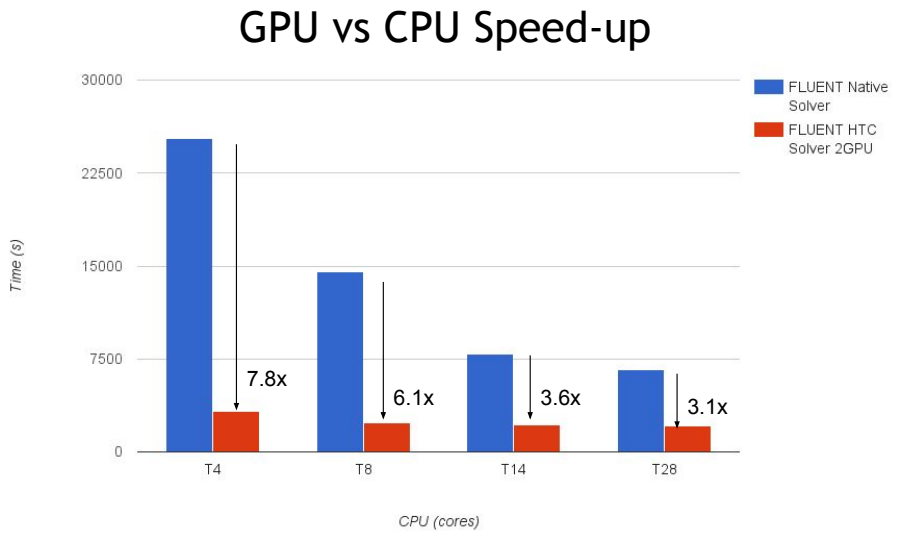
Mike Frisch, Ph.D.
President and CEO
Gaussian, Inc.

"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."

PGI

ANSYS FLUENT

Discrete Ordinate (DO) Radiation Solver



CPU Hardware:
(Haswell EP) Intel(R) Xeon(R) CPU E5-2695 v3 @ 2.30GHz, 2 socket x 14 = 28 cores
GPGPU Hardware:
Tesla K80 12+12 GB, Driver 346.46

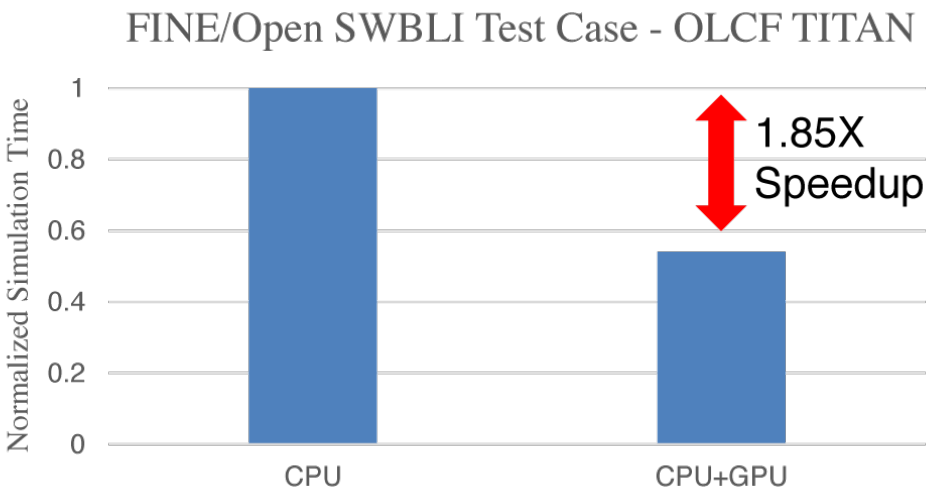
Sunil Sathe
Lead Software Developer
ANSYS Fluent

"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

Unstructured CDF Solver



System: Four nodes on the OLCF Titan supercomputer (16 cores and 1 GPU per node).

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.

PGI

MPAS ATMOSPHERE

Climate and Weather Modeling

A variable resolution MPAS Voronoi mesh

Source: mpas github
https://mpas-dev.github.io/atmosphere/atmosphere.html

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."

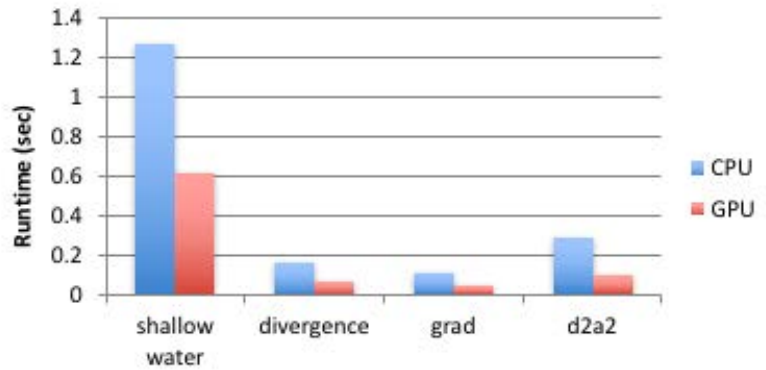
PGI

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

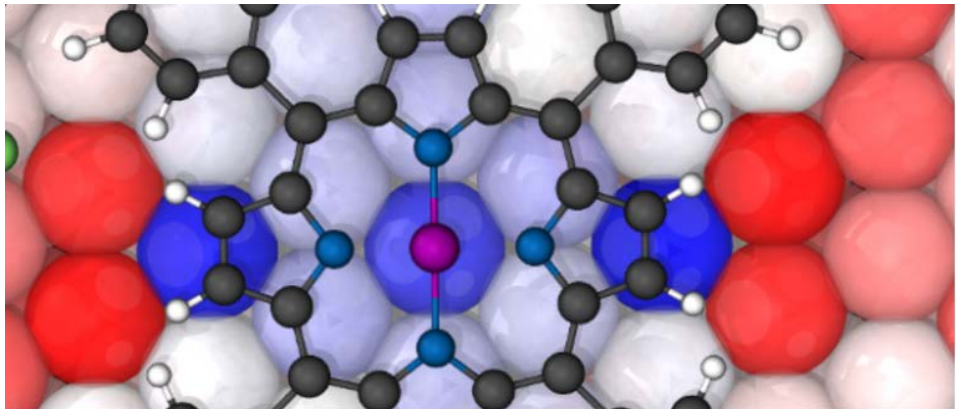
Mark Govett
Chief, HPC Section
NOAA

"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

QUANTUM ESPRESSO

Quantum Chemistry Suite



QUANTUMESPRESSO

www.quantum-espresso.org

Filippo Spiga
Head of Research Software Engineering
University of Cambridge

"CUDA Fortran gives us the full performance potential of the CUDA programming model and NVIDIA GPUs. !$CUF KERNELS directives give us productivity and source code maintainability. It's the best of both worlds."

PGI

COSMO

Regional Atmospheric Model

Image courtesy Meteoswiss.

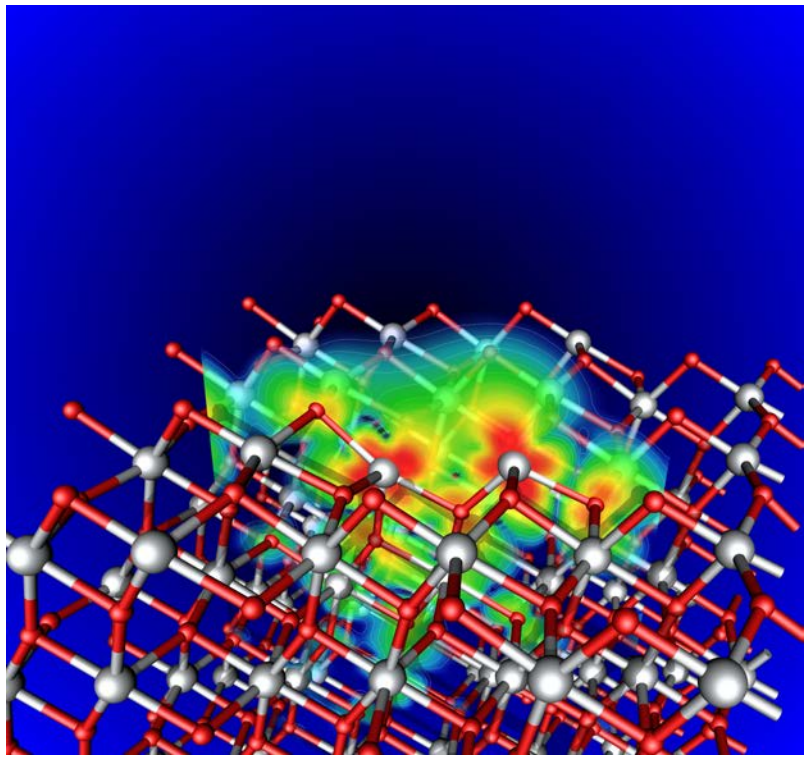
Dr. Oliver Fuhrer
Senior Scientist
Meteoswiss

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

PGI

VASP

The Vienna Ab Initio Simulation Package



Dr. Martijn Marsman
Computational Materials Physics
University of Vienna

“Early indications are that we can nearly match the performance of CUDA using OpenACC on GPUs. This will enable our domain scientists to work on a uniform GPU accelerated Fortran source code base.”

PGI

PGI FOR OpenPOWER+TESLA

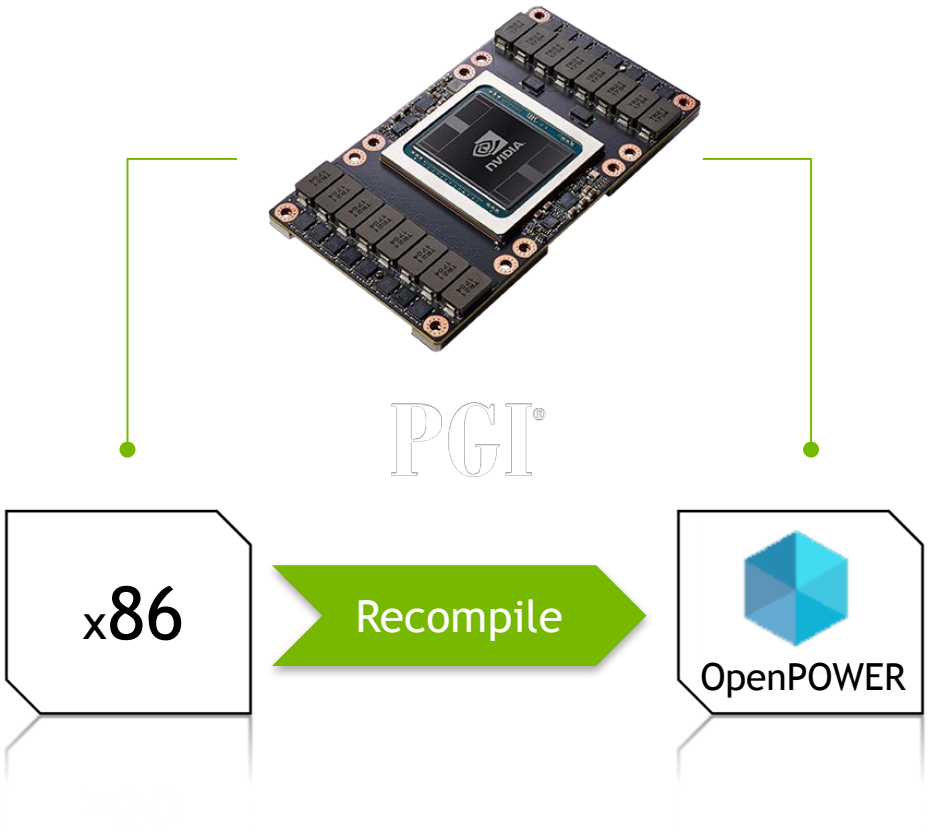
Fortran 2003, C11, C++14 compilers,
performance profiler

CUDA Fortran, OpenACC, OpenMP,
NVCC host compiler

Integrated IBM-optimized LLVM
OpenPOWER code generator

Available now at:

pgicompilers.com/openpower



PGI FOR OpenPOWER+TESLA

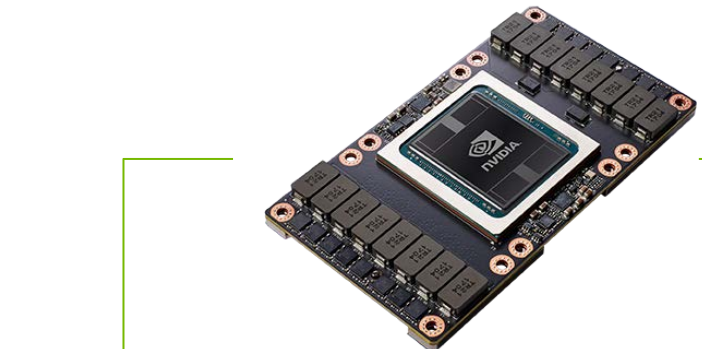
Fortran 2003, C11, C++14 compilers,
performance profiler

CUDA Fortran, OpenACC, OpenMP,
NVCC host compiler

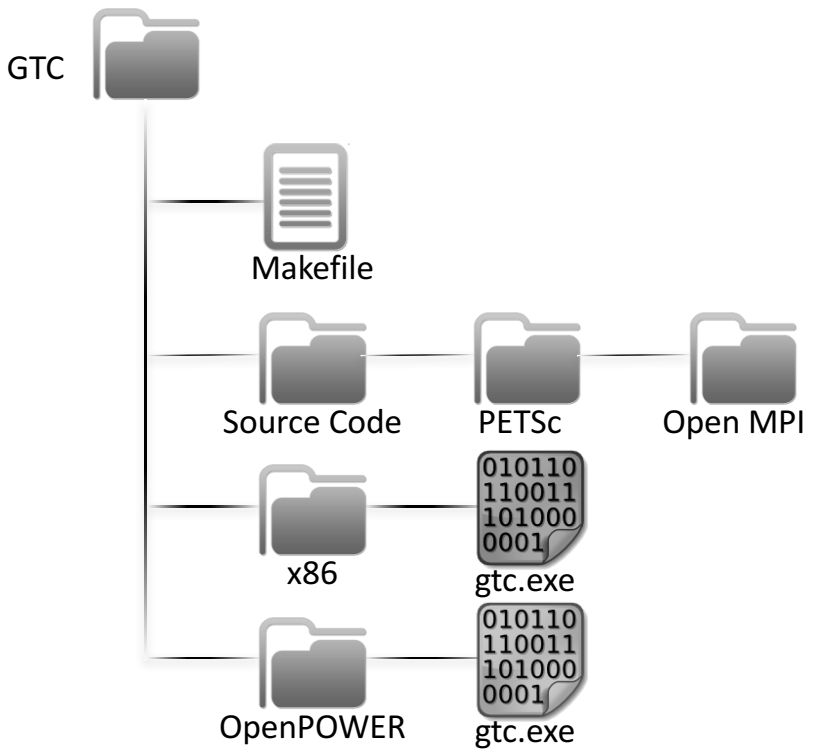
Integrated IBM-optimized LLVM
OpenPOWER code generator

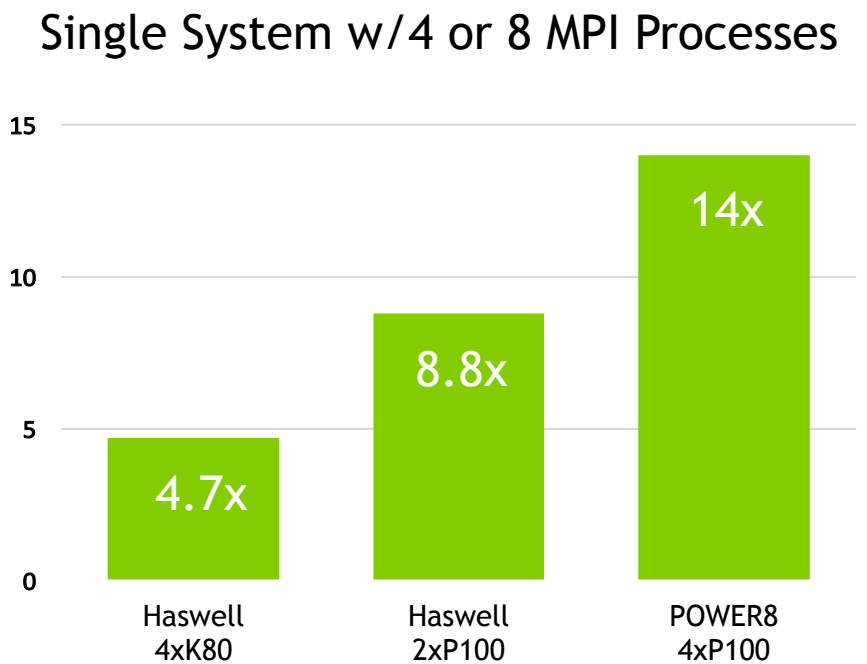
Available now at:

pgicompilers.com/openpower



Identical GTC Source Code / Build System on X86+TESLA and OpenPOWER+TESLA





X86 CPU: Intel Xeon E5-2698 v3, 32=cores
POWER CPU: IBM POWER8NVL, 40 cores

x86-64 TO OpenPOWER PORTING

C/C++ ABI DIFFERENCES

signed vs unsigned default char

long double

NUMERICAL DIFFERENCES

Intrinsics accuracy may differ across targets

FMA vs. no FMA

PLATFORM DIFFERENCES

Large memory model

C varargs

X86-SPECIFIC FEATURES

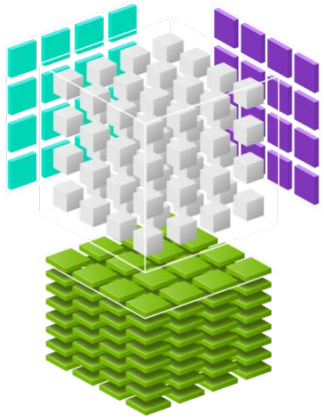
Inline *asm* statements

SSE/AVX intrinsics

VOLTA TO FUEL SUMMIT

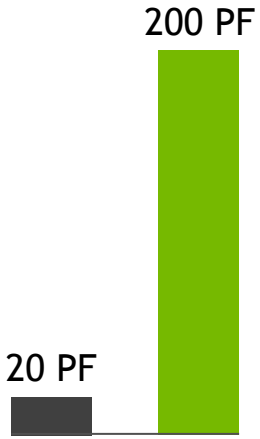
New Apex In AI Supercomputing

AI Exascale Today



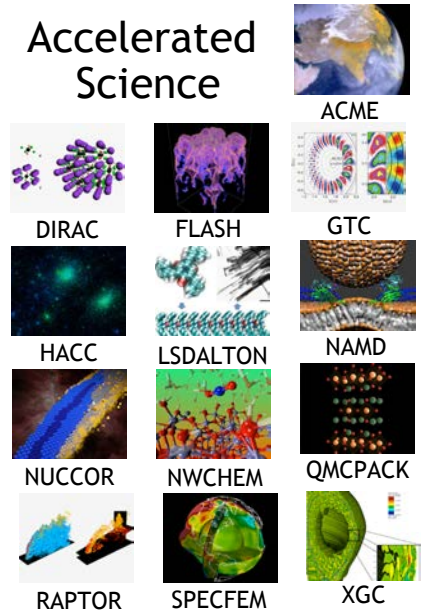
3+EFLOPS
Tensor Ops

Performance Leadership



10X
Perf Over Titan

Accelerated Science

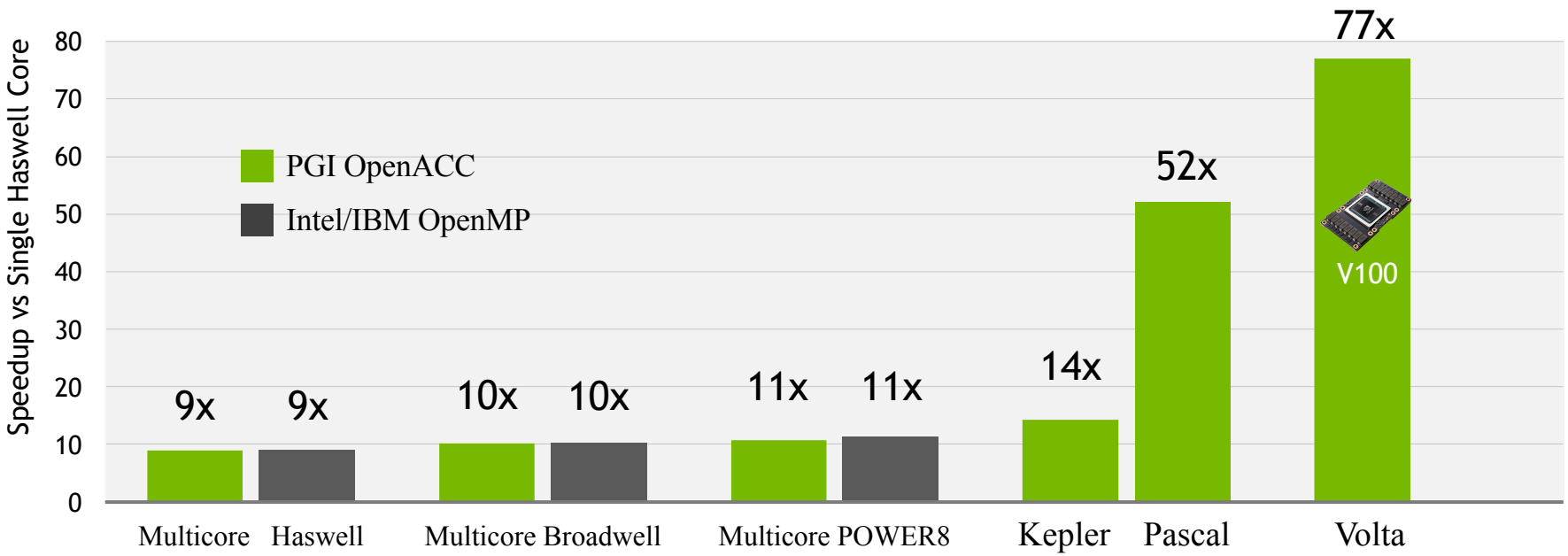


5-10X
Application Perf Over Titan

VOLTA V100 OPENACC PERFORMANCE

AWE Hydrodynamics CloverLeaf mini-App, bm32 data set

http://uk-mac.github.io/CloverLeaf



Systems: Haswell: 2x16 core Haswell server, four K80s, CentOS 7.2 (perf-hsw10), Broadwell: 2x20 core Broadwell server, eight P100s (dgx1-prd-01), Minsky: POWER8+NVLINK, four P100s, RHEL 7.3 (gsn1.
Compilers: Intel 17.0.1, IBM XL 13.1.3, PGI 16.10.
Benchmark: CloverLeaf v1.3 downloaded from http://uk-mac.github.io/CloverLeaf the week of November 7 2016; CloverlLeaf_Serial; CloverLeaf_ref (MPI+OpenMP); CloverLeaf_OpenACC (MPI+OpenACC)
Data compiled by PGI November 2016.

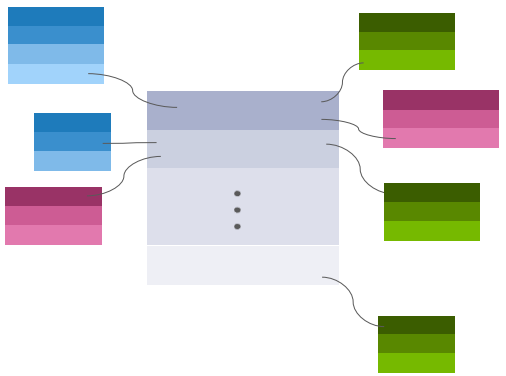
PGI

OPENACC UPTAKE IN HPC

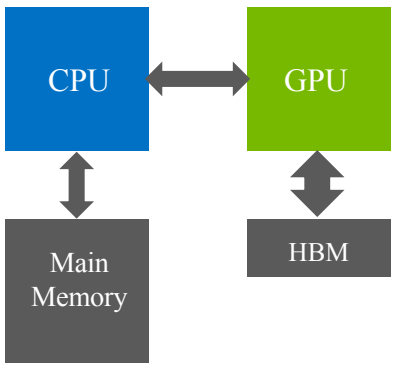
Applications	Hackathons	Workshops	Community
3 of Top10 Apps ANSYS Fluent, Gaussian, VASP* **5 ORNL CAAR Codes:** GTC, XGC, ACME, FLASH, LSDalton **Total tracked apps:** 104 *Started implementation	**6 ORNL Hackathons** 8-10 teams each (CU Boulder will be added) **60 applications accelerated since 2015** (both using OpenACC and CUDA during main hackathons only) **4 mini-Hackathons** CU Boulder, KAUST, BNL, CESGA (more to be scheduled) **Mentors** ~20 non-NVIDIA mentors All hackathons are initiated by users	**XSEDE** 2 a year ~200 attendees each **XSEDE** - NCSA Workshop 200 logged into quicklabs **Online Course** 2015 & 16: 2k reg, 1k attended each 2017 in planning Workshops WW Over 20 to date	**User Group F2F** 3 for 2017 with 50+ attendees each **Slack Channel** ~200 members 2x growth since Feb. 2017 **Downloads** 86% more monthly PGI CE download in 2017 Target to add 5 OpenACC members in 2017

OPENACC DATA MANAGEMENT

Modern Data Structures



Modern HPC Node Architectures



OPENACC 2.6 MANUAL DEEP COPY

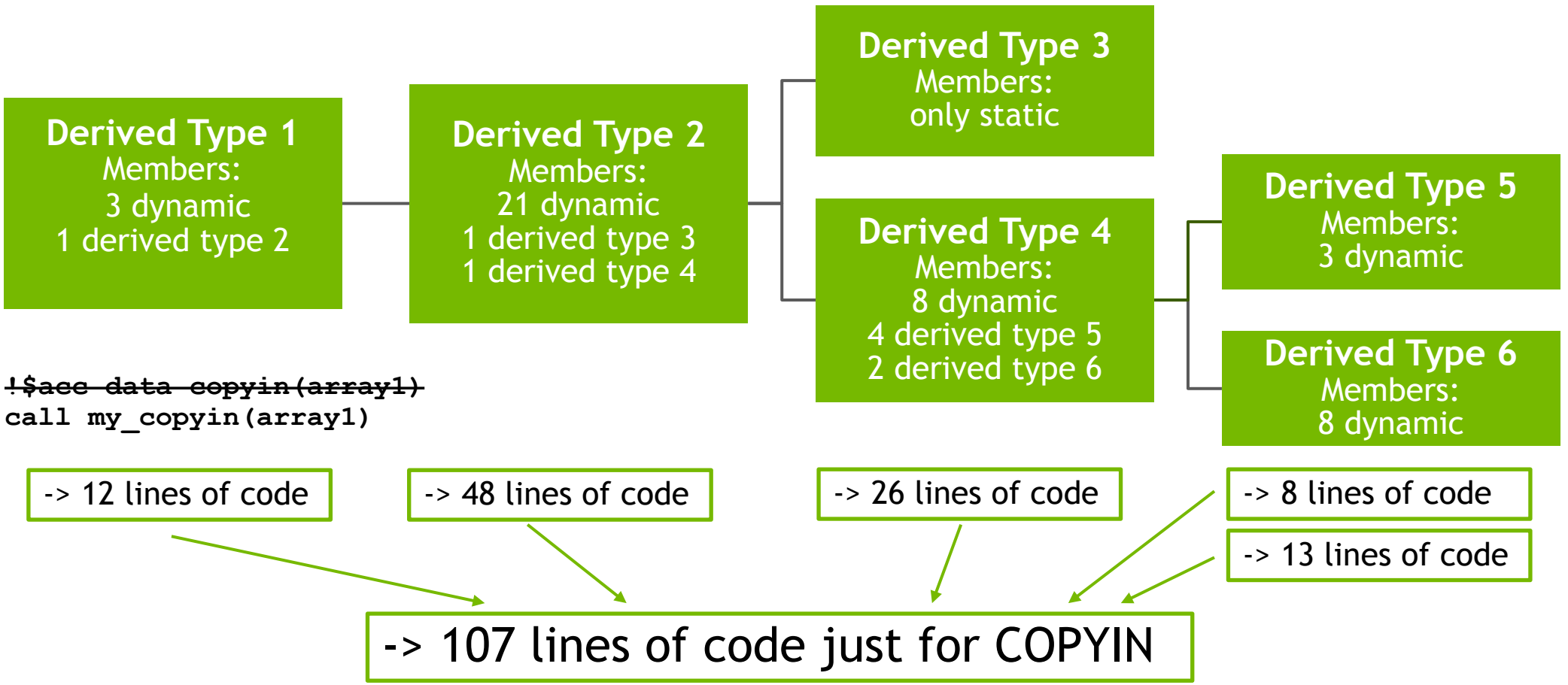
```
typedef struct points {
    float* x;  float* y;  float* z;
    int n;
    float coef, direction;
} points;


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];
            . . .
```

DRAFT OPENACC 3.0 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];
            . . .
```

OPENACC 2.6 MANUAL DEEP COPY

A real-world example: managing one aggregate data structure

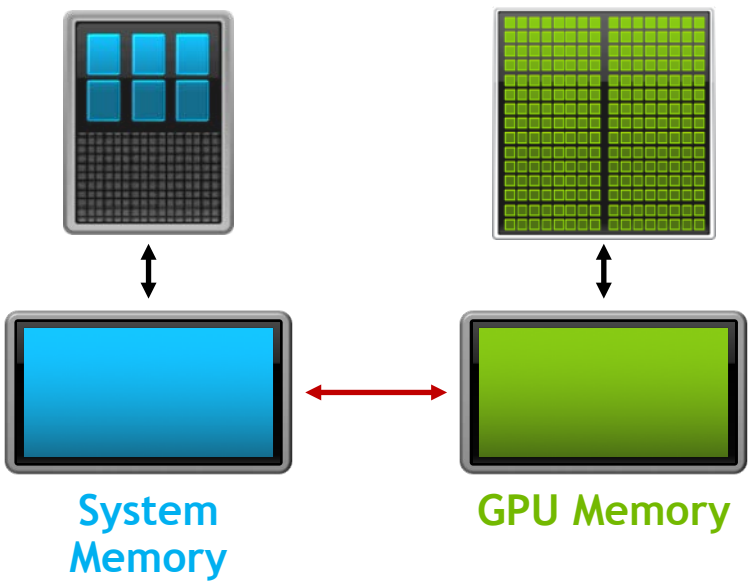


Plus additional lines of code for COPYOUT, CREATE, UPDATE

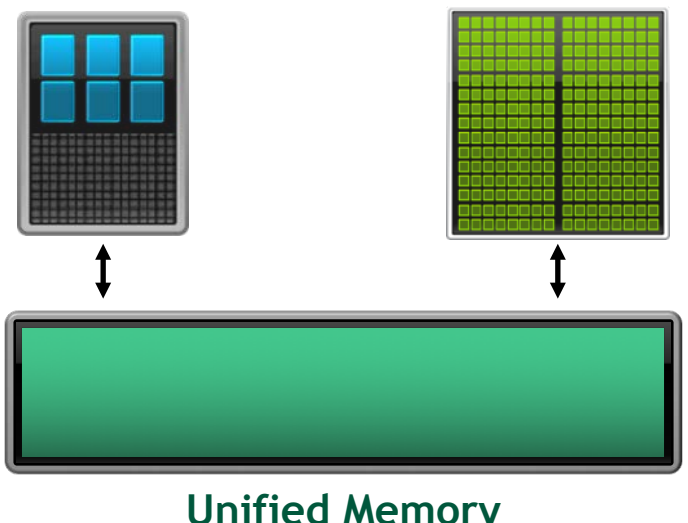
CUDA Unified Memory

Dramatically Lower Developer Effort

Developer View Today



Developer View With CUDA Unified Memory



CUDA UNIFIED MEMORY ON P100

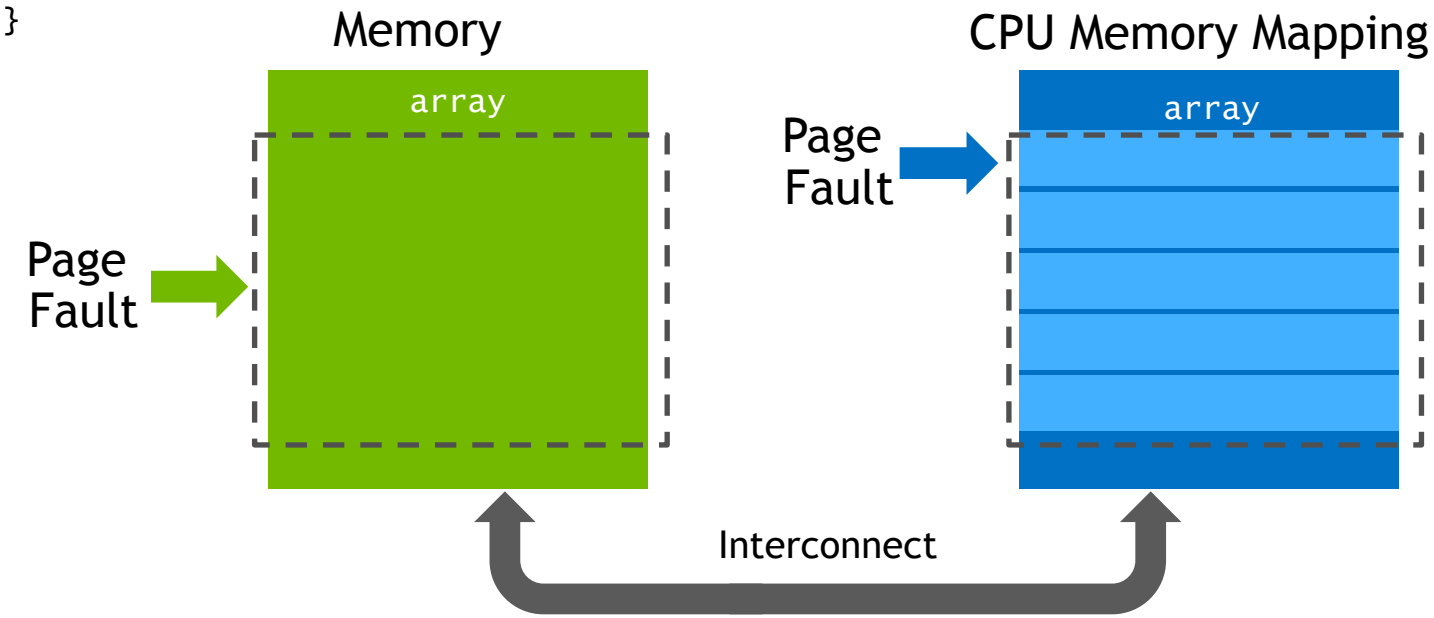
Servicing CPU *and* GPU Page Faults

GPU Code

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

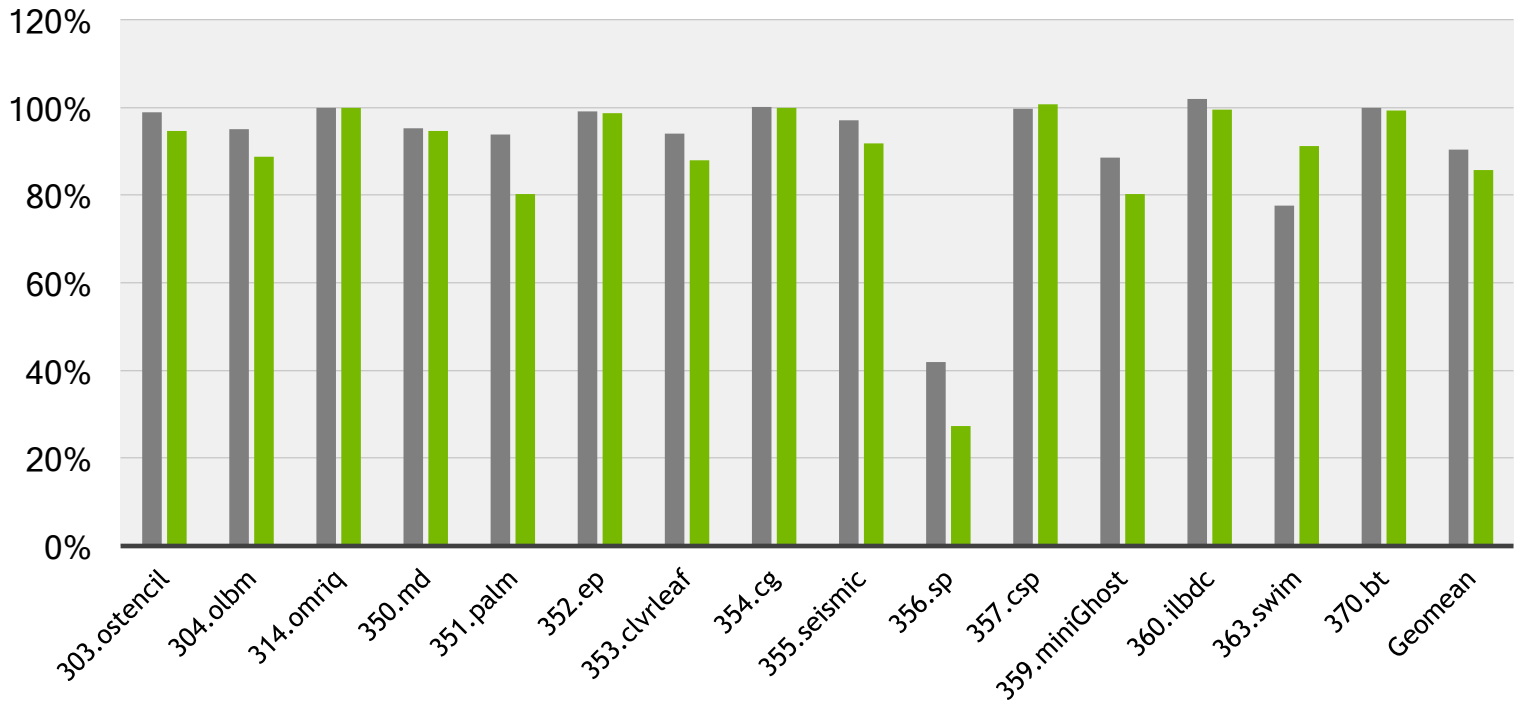
CPU Code

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



OPENACC WITH CUDA UNIFIED MEMORY

P100 Paging Engine Moves All Dynamically Allocated Data



100% = Pure Directive-based Data Movement

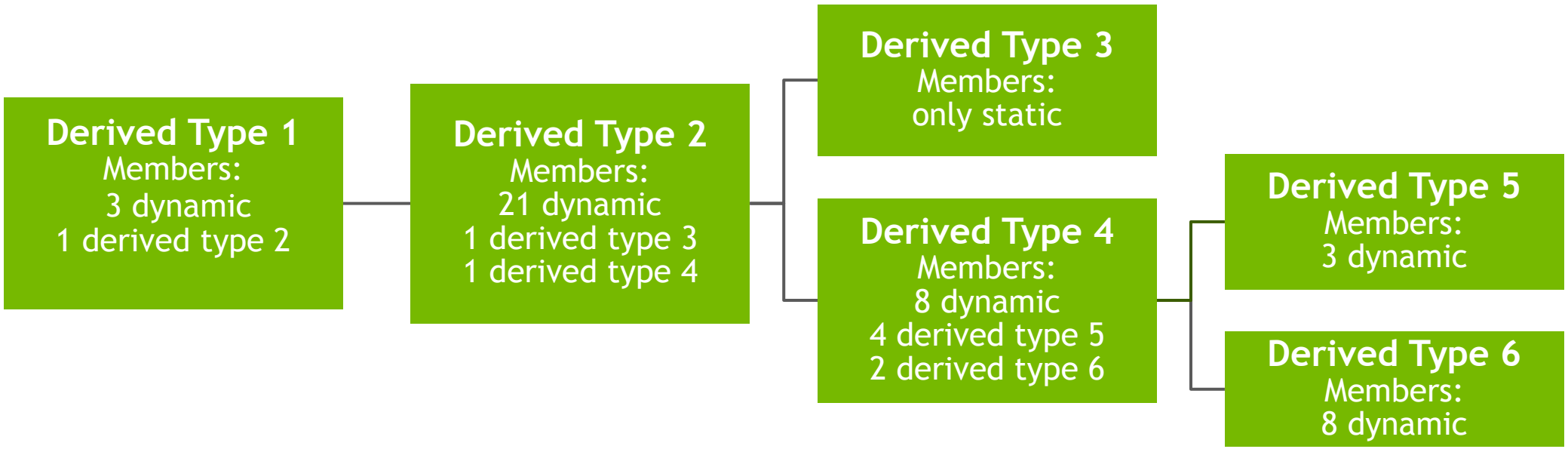
P8+P100 NVLink
HSW+P100 PCIe

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

PGI 17.5 Compilers OpenACC SPEC ACCEL™ 1.2 performance measured May, 2017 SPEC® and the benchmark name SPEC ACCEL™ are registered trademarks of the Standard Performance Evaluation Corporation.

OPENACC WITH UNIFIED MEMORY

A real-world example: managing one aggregate data structure



0 lines of code! It just works.

OPENACC WITH UNIFIED MEMORY

A real-world example: managing one aggregate data structure

With Manual Deep Copy

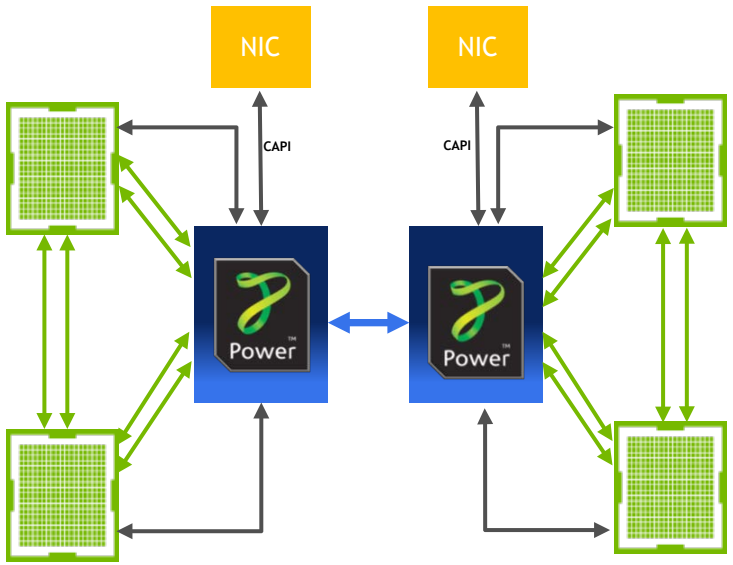
With Unified Memory

Total number of OpenACC directives required
for a real-world Solver port

IBM POWER NVLINK SYSTEMS

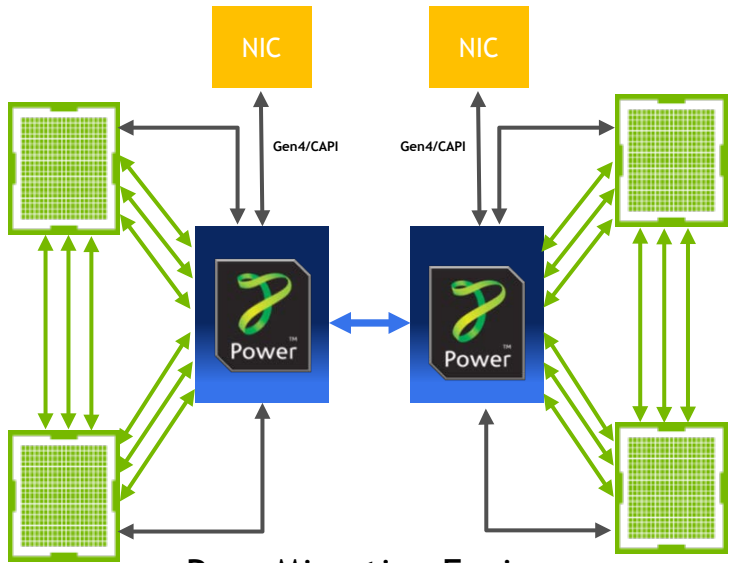
APPROVED DESIGNS FOR OPENPOWER ECOSYSTEM

4 GPU POWER8 FOR PASCAL



Page Migration Engine

4 GPU POWER9 FOR VOLTA



Page Migration Engine
Coherence, Atomics, ATS

P100 SXM2

40 GB/s full duplex each CPU:GPU and GPU:GPU link
IBM POWER8+ CPU

VOLTA SXM2

75 GB/s full duplex each CPU:GPU and GPU:GPU link
IBM POWER9 CPU

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

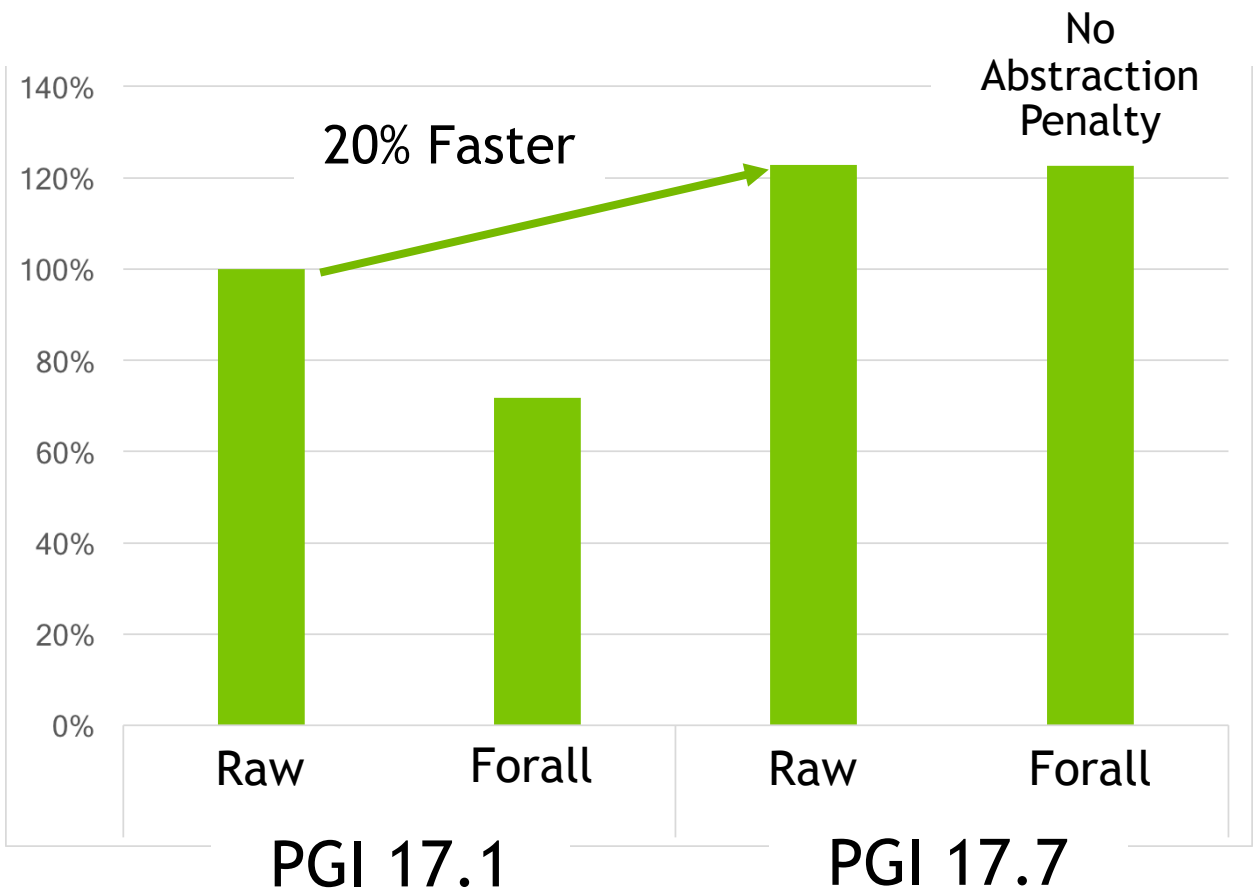
C++14 LAMBDAS WITH CAPTURE

Now supported in OpenACC Regions

```
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 );
  } elseif ( is_same<Executon_Policy, OpenACC> :: value ) {
  #pragma acc parallel loop
  for ( int i = s; i < e; ++i )
    body ( i );
  } return EndTimer ( );
}

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

double stime = bench_forall<Serial>(0, N, saxpy);
double time = bench_forall<OpenACC>(0, N, saxpy);
printf ( "OpenACC Speedup %f \n", stime / time );
}
```

C++ PERFORMANCE ENHANCEMENTS

LCALS Benchmark Performance



C++14 LAMBDAS WITH CAPTURE

Now supported in OpenACC Regions

```
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 );
  } elseif ( is_same<Executon_Policy, OpenACC> :: value ) {
  #pragma acc parallel loop
  for ( int i = s; i < e; ++i )
    body ( i );
  } return EndTimer ( );
}

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

double stime = bench_forall<Serial>(0, N, saxpy);
double time = bench_forall<OpenACC>(0, N, saxpy);
printf ( "OpenACC Speedup %f \n", stime / time );
}
```

OPENACC DIRECTIVES …

```
 75 !$ACC KERNELS
 76 !$ACC LOOP INDEPENDENT
 77     DO k=y_min,y_max
 78 !$ACC LOOP INDEPENDENT PRIVATE(right_flux,left_flux,top_flux,bottom_flux,total_flux,
                                 min_cell_volume,energy_change,recip_volume)
 79       DO j=x_min,x_max
 80
 81         left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                &
 82                                     +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
 83         right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                &
 84                                     +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
 85         bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                &
 86                                     +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
 87         top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                &
 88                                     +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
 89         total_flux=right_flux-left_flux+top_flux-bottom_flux
 90
 91         volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)
 92
 93         min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
 94                            ,volume(j,k)+right_flux-left_flux                      &
 95                            ,volume(j,k)+top_flux-bottom_flux)
 97         recip_volume=1.0/volume(j,k)
 99         energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
101         energy1(j,k)=energy0(j,k)-energy_change
103         density1(j,k)=density0(j,k)*volume_change(j,k)
105       ENDDO
106     ENDDO
107 !$ACC END KERNELS
```

FORTRAN 2015 PARALLEL LOOPS …

Fortran 2015 DO CONCURRENT

+ True Parallel Loops
+ Loop-scope shared/private data
– No support for reductions
– No support for data management

```
 77     DO CONCURRENT (k=y_min:y_max, j=x_min:x_max) &
 78        LOCAL (right_flux,left_flux,top_flux,bottom_flux,total_flux, &
                  min_cell_volume,energy_change,recip_volume)
 79
 80
 81        left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,k+1)                &
 82                                    +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
 83        right_flux= (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                &
 84                                    +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
 85        bottom_flux=(yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                &
 86                                    +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
 87        top_flux=   (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                &
 88                                    +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
 89        total_flux=right_flux-left_flux+top_flux-bottom_flux
 90
 91        volume_change(j,k)=volume(j,k)/(volume(j,k)+total_flux)
 92
 93        min_cell_volume=MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
 94                           ,volume(j,k)+right_flux-left_flux                      &
 95                           ,volume(j,k)+top_flux-bottom_flux)
 97        recip_volume=1.0/volume(j,k)
 99        energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
101        energy1(j,k)=energy0(j,k)-energy_change
103        density1(j,k)=density0(j,k)*volume_change(j,k)
105
106     ENDDO
```

LEARN MORE

Study and promote OpenACC application porting success stories

Binge watch and promote Michael Wolfe's OpenACC Programming how-to videos

Promote the PGI Community Edition – an easy on-ramp to OpenACC and accelerated computing

Follow PGI on Twitter and re-tweet relevant messages to your network

Promote upcoming OpenACC events to your contacts and colleagues

PGI® COMPILERS & TOOLS

NVIDIA®