PGI® COMPILERS &TOOLS

PGI UPDATES FOR 2019

Michael Wolfe

GTC, March 2019

NVIDIA

AGENDA

PGI Introduction

Platforms, Programming models, Performance, Interoperability

What's new in PGI 2019

Features, Targets, CUDA Fortran, OpenACC, Containers and Clouds

PGI Directions

OpenACC, OpenMP, Languages

PGI — THE NVIDIA HPC SDK

Fortran/C/C++ Compilers and Tools

Multi-Platform Solution

- X86-64 and OpenPOWER Multicore CPUs
- NVIDIA Tesla GPUs
- Supported on Linux, macOS, Windows

High Performance Parallelism

- Optimization, SIMD Vectorization
- OpenMP and OpenACC on CPU
- OpenACC and CUDA Fortran on GPU

Interoperability

- GNU compatible C++
- Rogue Wave Totalview, Arm Allinea DDT, nvprof
- OpenACC, CUDA Fortran, CUDA C++

NVIDIA

PGI®

The Compilers & Tools for Supercomputing

19.4 Coming Soon!

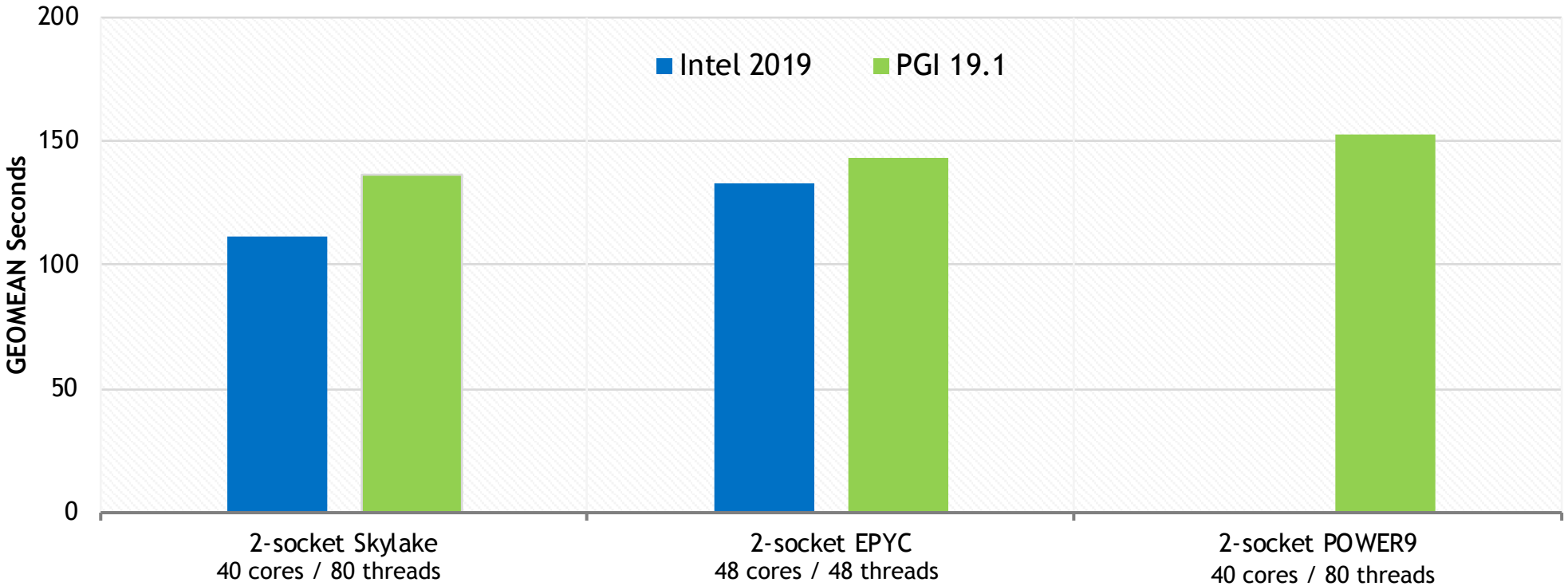
PGI COMPILERS FOR EVERYONE

The PGI 18.10 Community Edition

	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

SPEC CPU 2017 FP SPEED BENCHMARKS

OpenMP 3.1 Performance on Multicore CPUs – smaller is better



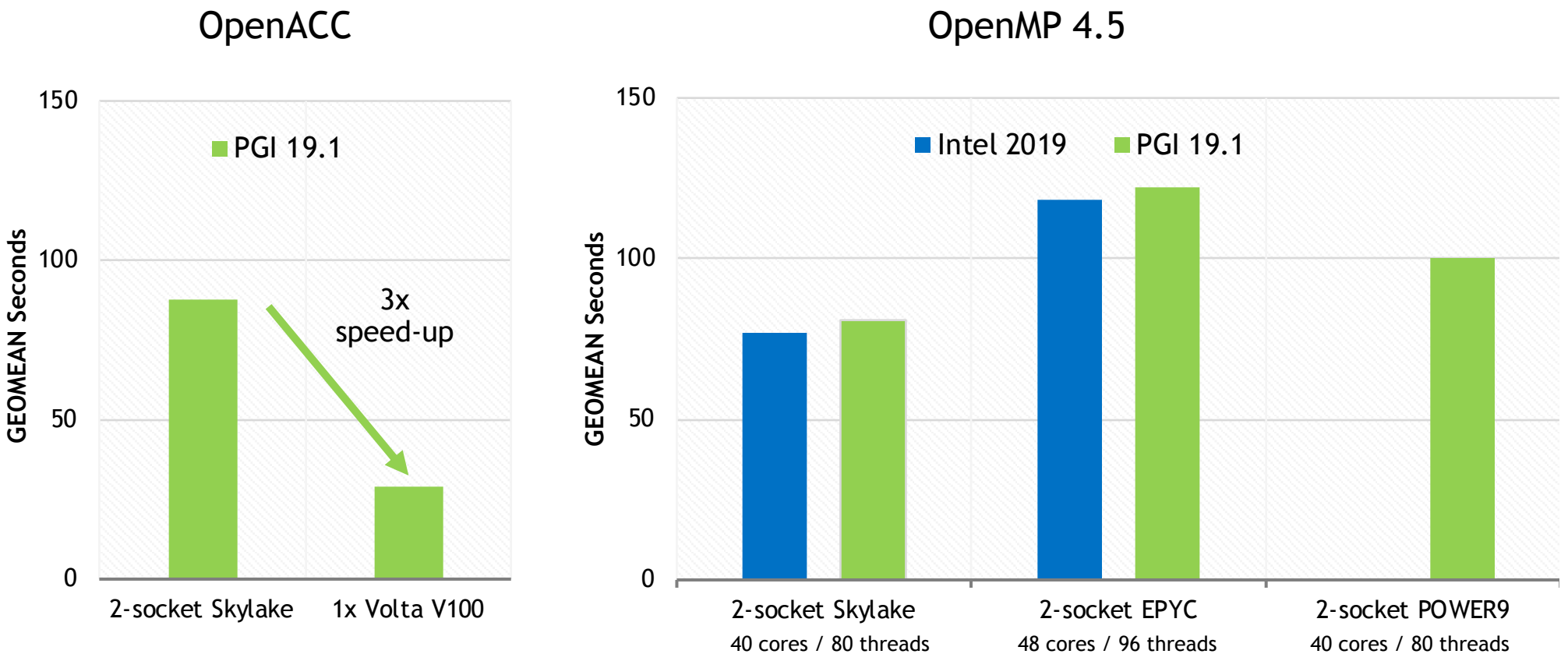
Performance measured January, 2019. Skylake: Two 20 core Intel Xeon Gold 6148 CPUs @ 2.4GHz w/ 376GB memory, hyperthreading enabled. EPYC: Two 24 core AMD EPYC 7451 CPUs @ 2.2GHz w/ 256GB memory. POWER 9: Two 20 core IBM POWER9 D2.2 @ 3.4GHz w 128GB memory.

Compilers: PGI 19.1, Intel 2019.

SPEC® is a registered trademark of the Standard Performance Evaluation Corporation (www.spec.org).

SPEC ACCEL 1.2 BENCHMARKS

OpenACC vs OpenMP Performance on CPUs and GPUs



Performance measured January, 2019. Skylake: Two 20 core Intel Xeon Gold 6148 CPUs @ 2.4GHz w/ 376GB memory, hyperthreading enabled with two NVIDIA Tesla V100-PCIe-16GB GPU @ 1.53GHz. EPYC: Two 24 core AMD EPYC 7451 CPUs @ 2.2GHz w/ 256GB memory. POWER9: Two 20 core IBM POWER9 DD2.2 @ 3.4GHz w 128GB memory.

PGI | 19.1

Fortran, C and C++ for the Tesla Platform

Full C++17 language

OpenMP 4.5 for CPUs

CUDA 10.x support

CUDA Fortran Tensor Core Support

OpenACC 2.6

OpenACC + printf, deep copy, C++ lambda

PCAST

PGI in the Cloud

pgicompilers.com/whats-new

CUDA FORTRAN

CUDA FORTRAN

```
real, device, allocatable, dimension(:) :: Xd, Yd, Rd
real, pinned, allocatable, dimension(:) :: X, Y
real :: R

. . .

allocate (Xd(N), Yd(N), Rd((N+127)/128))
Xd = X(:)
Yd = Y(:)

call dsaxpy1<<<(N+127)/128, 128>>>(Yd,A,Xd,N)

call dsum1<<<(N+127)/128, 128>>>(Yd,Rd,N)

call dsum1<<<1, 128>>>(Rd,Rd,(N+127)/128)

Y(:) = Yd
R = Rd(1)
deallocate (Xd, Yd, Rd)

. . .
```

CPU Code

```
attributes(global) subroutine daxpy1(Y,A,X,N)
real :: Y(*), X(*)
integer, value :: N
real, value :: A
i = (blockidx%x-1) * blockdim%x + threadidx%x
if (i <= N) Y(i) = A * X(i) + Y(i)
end subroutine daxpy1

attributes(global) subroutine dsum1(Y,R,N)
real :: Y(*), R(*)
integer, value :: N
real, shared :: part(128)
s = 0
j = (blockidx%x-1) * blockdim%x + threadidx%x
do i = j, N, blockdim%x * griddim%x
  s = s + Y(i)
enddo
j = threadidx%x ; k = blockdim%x
part(j) = s
do while (k > 1)
  k = k / 2
  if (j <= k) part(j) = part(j) + part(j+k)
  call syncthreads()
enddo
R(blockidx%x) = part(1)
end subroutine dsum1
```

Tesla Code

PGI

CUDA FORTRAN DIRECTIVES

```
real, device, allocatable, dimension(:) :: Xd, Yd, Rd
real, pinned, allocatable, dimension(:) :: X, Y
real :: R

. . .

allocate (Xd(N), Yd(N), Rd((N+127)/128))
Xd = X(:)
Yd = Y(:)
R = 0.0
!$cuf kernel do(1) <<<*,*>>>
do i = 1, n
 Yd(i) = A * Xd(i) + Yd(i)
 R = R + Yd(i)
enddo

Y(:) = Yd

deallocate (Xd, Yd)

. . .
```

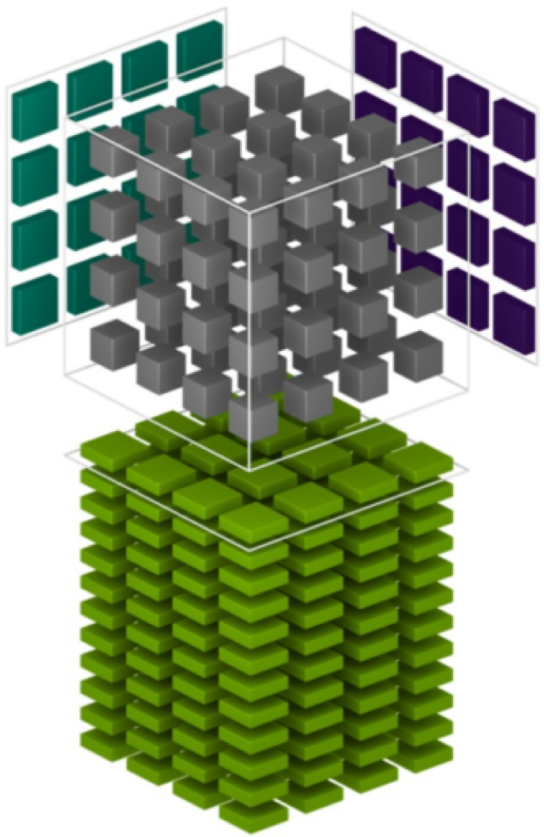
CPU Code

Compiler Generated!

Tesla Code

V100 TENSOR CORES

Fast Matrix Multiply Using CUDA Fortran



Fast FP16 matrix multiply and accumulation into FP16 or FP32

8x to 16x faster than pure FP32 or FP64 in the same power envelope

Up to 125 TFlops FP16 performance

CUDA FORTRAN TENSOR CORES

```
attributes(global) subroutine ww(a, b, c, m, k, n)
 use wmma
 integer, value :: m, k, n
 CUFReal2, device :: a(m, k)
 CUFReal2, device :: b(k, n)
 CUFReal2, device :: c(m, n)
 WMMASubMatrix(WMMAMatrixA, 32, 8, 16, Real, WMMAColMajor):: sa
 WMMASubMatrix(WMMAMatrixB, 32, 8, 16, Real, WMMAColMajor):: sb
 WMMASubMatrix(WMMAMatrixC, 32, 8, 16, Real, WMMAColMajor):: sc

 call wmmaLoadMatrix(sa, a(1,1), m)
 call wmmaLoadMatrix(sb, b(1,1), k)
 call wmmaMatmul(sc, sa, sb, sc)
 call wmmaStoreMatrix(c(1,1), sc, m)
end subroutine
```

OPENACC

OPENACC EXAMPLE

```
for (iter = 0; iter < 100; ++iter) {

  for (i = 1; i < n-1; ++i) {

    for (j = 1; j < n-1; ++j) {
       NH[j+i*n] = 0.25 * (OH[(j-1)+i*n] + OH[[(j+1)+i*n] +
                           OH[j+(i-1)*n] + OH[j+(i+1)*n]);
    }
  }
  tmp = OH; OH = NH; NH = tmp;
}
```

SUPPORT FOR PRINTF() IN OPENACC

Debugging / tracing GPU-accelerated code regions

```
#pragma acc kernels copy(x[0:256])
{
  for (int i = 0; i < 256; i++) {
    if (x[i] < small) {
      printf("x at loc %03d is small: %d\n",i,x[i]);
      x[i] = small;
    } else if (x[i] > large) {
      printf("x at loc %03d is large: %d\n",i,x[i]);
      x[i] = large;
    }
 }
```

Support for formatted output using printf() statements in OpenACC compute regions

Debug and trace OpenACC programs during development and tuning

Works with both CPU and GPU

OPENACC TRUE DEEP COPY DIRECTIVES

```
typedef struct points {
    float* x;  float* y;  float* z;
    int n;
    float coef, direction;
    #pragma acc shape(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];
        . . .
```

Still in definition by the OpenACC Committee

Beta Implementation since PGI 18.7

OPENACC TRUE DEEP COPY DIRECTIVES

```
typedef struct points {
    float* x;  float* y;  float* z;
    int n;
    float coef, direction;
    #pragma acc policy<dpmove> copy(x[0:n]) copyin(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<dpmove>)
    {
        #pragma acc parallel loop
        for ( i =0; i<p.n; ++i ) p.x[i] += p.y[i];
        . . .
```

Still in definition by the OpenACC Committee

Beta Implementation since PGI 18.7

C++14 LAMBDAS AND OPENACC

Lambdas with Capture by Value 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<Execution_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 );
}
```

DIRECTIVE-BASED HPC PROGRAMMING

Who's Using OpenACC?

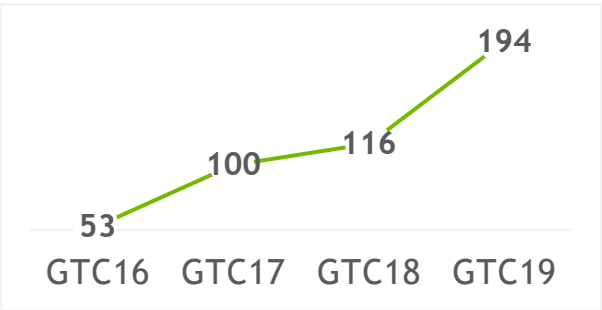
3 OF TOP 5 HPC APPS

Intersect360 RESEARCH

5 OF 13 CAAR CODES

summit

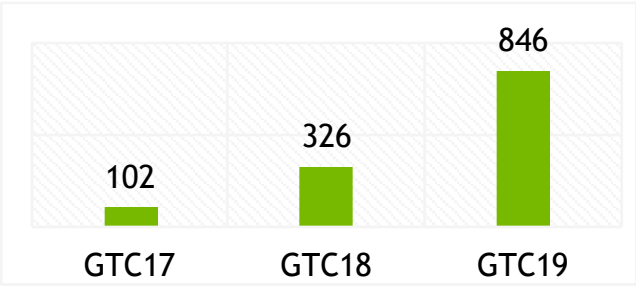
ACCELERATED APPS



725 TRAINED EXPERTS

OpenACC HACKATHON
Advancing High Performance Computing

SLACK MEMBERS



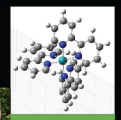
160,000+ DOWNLOADS

PGI®
Community EDITION

OpenACC

More Science, Less Programming

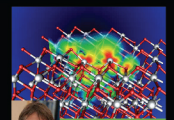
GAUSSIAN



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."

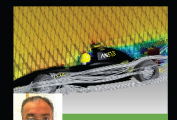
VASP



Prof. Georg Kresse
Computational Materials Physics
University of Vienna

"For VASP, OpenACC is the way forward for GPU acceleration. Performance is similar to CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts."

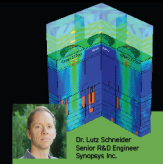
ANSYS Fluent



Sunil Sathe
Lead Software Developer
ANSYS Fluent

"We recently started evaluating OpenACC for parallelization on multicore CPUs. Based on some early work, one of our OpenACC-based solvers is as fast as the OpenMP version on multicore CPUs and delivers speed-ups of 4-5x on a Tesla P100 compared to all the cores of a dual-socket server."

SYNOPSYS



Dr. Lutz Schneider
Senior R&D Engineer
Synopsys Inc.

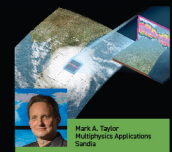
"Using OpenACC, we've accelerated the Synopsys TCAD Sentaurus Device EMW simulator to speed up FDTD simulations of image sensors by a factor of 15 on a single V100 GPU compared to a dual-socket Broadwell server. GPUs are key to improving simulation throughput in the design of advanced image sensors."

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."

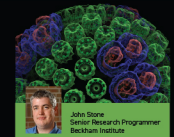
E3SM



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."

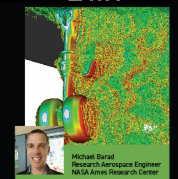
VMD



John Stone
Senior Research Programmer
Beckham Institute

"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."

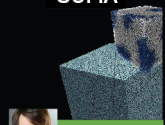
LAVA



Michael Barad
Research Aerospace Engineer
NASA Ames Research Center

"We used OpenACC to port our LAVA Lattice-Boltzman mini-app to GPU. Running a single block grid at 256^3, one Volta V100 was 5x faster than two 40-core Skylake CPUs."

SOMA



Ludwig Schneider
PhD Student
Georg-August-Universität Göttingen

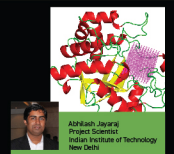
"OpenACC enables us to compile a single code base for multiple architectures. This keeps the code maintainable and flexible even for future accelerators. For our OpenACC accelarated "SCMP" algorithm, a single V100 outperforms 24 CPU cores by roughly a factor of 10."

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."

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 effort or manpower in speeding up the entire computational task."

MPAS-A

Richard Loft
Director, Technology Development, NCAR

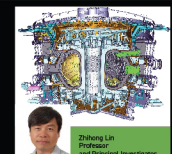
"Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer."

GAMERA FOR GPU

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."

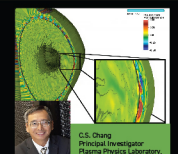
GTC



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."

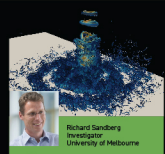
XGC



C.S. Chang
Principal Investigator
Plasma Physics Laboratory,
Princeton University

"Using a combination of CUDA and OpenACC for our most compute-intensive kernels, the GPU-accelerated version of XGC delivers over 11x speed-ups compared to CPU-only execution when running at scale on 2048 nodes of ORNL's new Summit supercomputer."

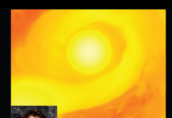
HIPSTAR



Richard Sandberg
Investigator
University of Melbourne

"For a relatively small production case of about 80 million grid points, our OpenACC-enabled version running on two GPU nodes with 4 P100s each was approximately 20.5x faster than two nodes with 24 Haswell cores each. We're looking forward to running a larger simulation on Summit early in 2019."

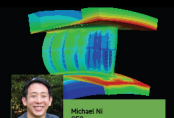
CASTRO/MAESTRO



Adam Jacobs
Ph.D. candidate
Stony Brook University

"For scientific applications that run on several different super-computing architectures and need to be usable for many generations of architecture, the cons of something like CUDA outweigh the pros. That's why we prefer OpenACC."

ADS CFD



Michael Ni
CEO
AeroDynamic Solutions, Inc.

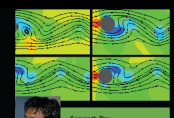
"Using OpenACC on a single Tesla V100 in the Amazon Cloud, our GPU-accelerated Code Leo flow solver runs 1.4 times faster at 30% less cost compared to runs using all 72 vCPU cores of a Xeon Platinum c5.18xlarge instance. We feel this will revolutionize the aerospace design cycle, enabling the delivery of more durable, more reliable and higher performing designs at reduced development cost."

FLASH

Bronson Messer
Senior Scientist
Oak Ridge National Laboratory

"We're using OpenACC on Summit to accelerate our most compute-intensive kernels. We love OpenACC interoperability and how this allows us to use multiple methods to perform memory placement and movement. CPU-GPU performance of a 288 species network on Summit, something impossible to do on Titan, is 2.9x faster than GPU only."

IBM-CFD



Somnath Roy
Assistant Professor
Mechanical Engineering Dept.
Indian Institute of Technology
Kharagpur

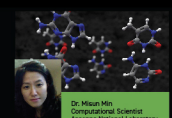
"Using OpenACC to accelerate our immersed boundary incompressible CFD code, we're seeing an order of magnitude reduction in computing time running on GPUs. Routines involving our search algorithm and matrix solvers perform especially well with OpenACC, and improve the overall scalability of the code."

HIFUN

Murali Krishna Nagaram
Chief Technology Officer
S&J Engineering Solutions
Pvt. Ltd.

"OpenACC allowed us to port our legacy CFD solvers to hybrid CPU+GPU platforms in a form that is readable and maintainable, and enabled us to exploit GPUs in our HiFUN MPI CFD solver in very little time."

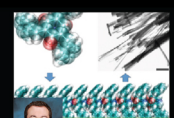
NEKCEM



Dr. Misun Min
Computational Scientist
Argonne National Laboratory

"The most significant result from our performance studies is faster computation with less energy consumption compared with our CPU-only runs. The GPU required only 39 percent of the energy needed for 16 CPUs to do the same computation. That OpenACC is an open standard was an important factor in our decision to use it for our research."

LSDALTON



Dmytro Bykov
Computational Scientist
Oak Ridge National
Laboratory

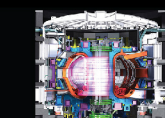
"Using OpenACC, we see large performance gains with very little effort. GPU acceleration varies over the course of our simulations due to differing fragment sizes, but is typically 3x–5x. On Summit we can now do simulations of several thousand atoms, compared to maybe 800 on Titan."

FINE/Open

David Gutzwiller
Lead Software Developer
NUMECA

"Porting our unstructured C++ CDF 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."

CGYRO



Igor Sfiligoi
HPC Software Developer
General Atomics

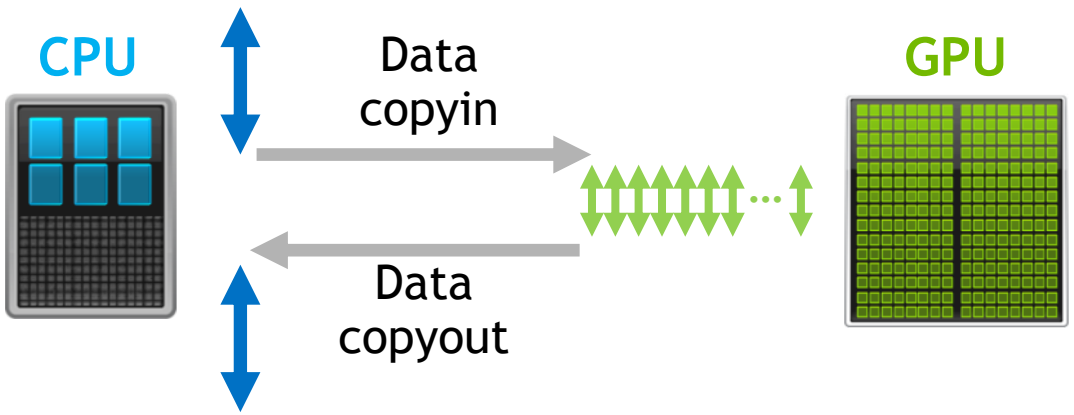
"We planned to spend a month porting our OpenACC code from XSB + GPUs to POWER9 + GPUs. Using the PGI compilers and our standard build environment, we were running in an afternoon. It just worked!"

PCAST

PGI COMPILER-ASSISTED SOFTWARE TESTING

OPENACC AUTO-COMPARE

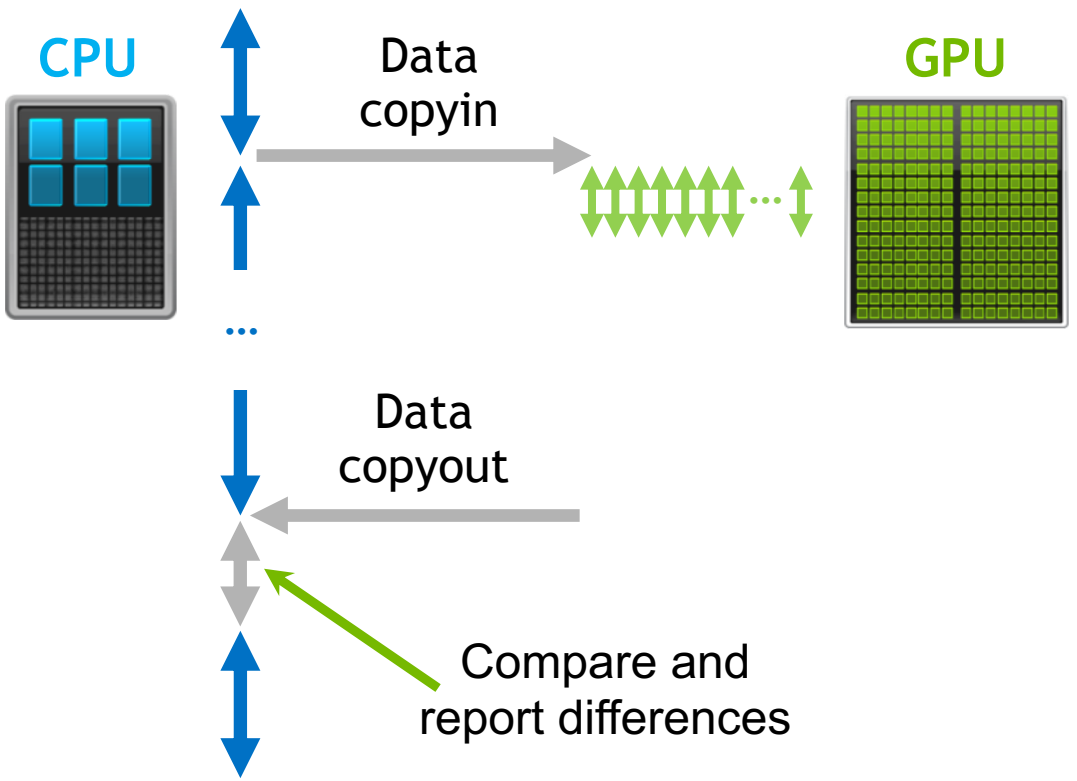
Find where CPU and GPU numerical results diverge



Normal OpenACC execution

OPENACC AUTO-COMPARE

Find where CPU and GPU numerical results diverge



-ta=tesla:autocompare

Compute regions run redundantly on CPU and GPU

Results compared when data copied from GPU to CPU

pgicompilers.com/pcast

AUTO-COMPARE OVERVIEW

PGI Compiler-Assisted Software Testing

```
$ pgcc -ta=tesla:autocompare -o a.out example.c


$ PGI_COMPARE=summary,compare,abs=1 ./a.out
PCAST a1 comparison-label:0 Float
        idx: 0 FAIL ABS  act: 8.40187728e-01
exp: 1.00000000e+00 tol: 1.00000001e-01
        idx: 1 FAIL ABS  act: 3.94382924e-01
exp: 1.00000000e+00 tol: 1.00000001e-01
        idx: 2 FAIL ABS  act: 7.83099234e-01
exp: 1.00000000e+00 tol: 1.00000001e-01
        idx: 3 FAIL ABS  act: 7.98440039e-01
exp: 1.00000000e+00 tol: 1.00000001e-01
```

Test for program correctness, and determine points of divergence

Detect diverging results between CPU & GPU or multiple CPU code versions

Flexible calling and instrumentation options

pgicompilers.com/pcast

PCAST SUMMARY

OpenACC, GPU to CPU comparison

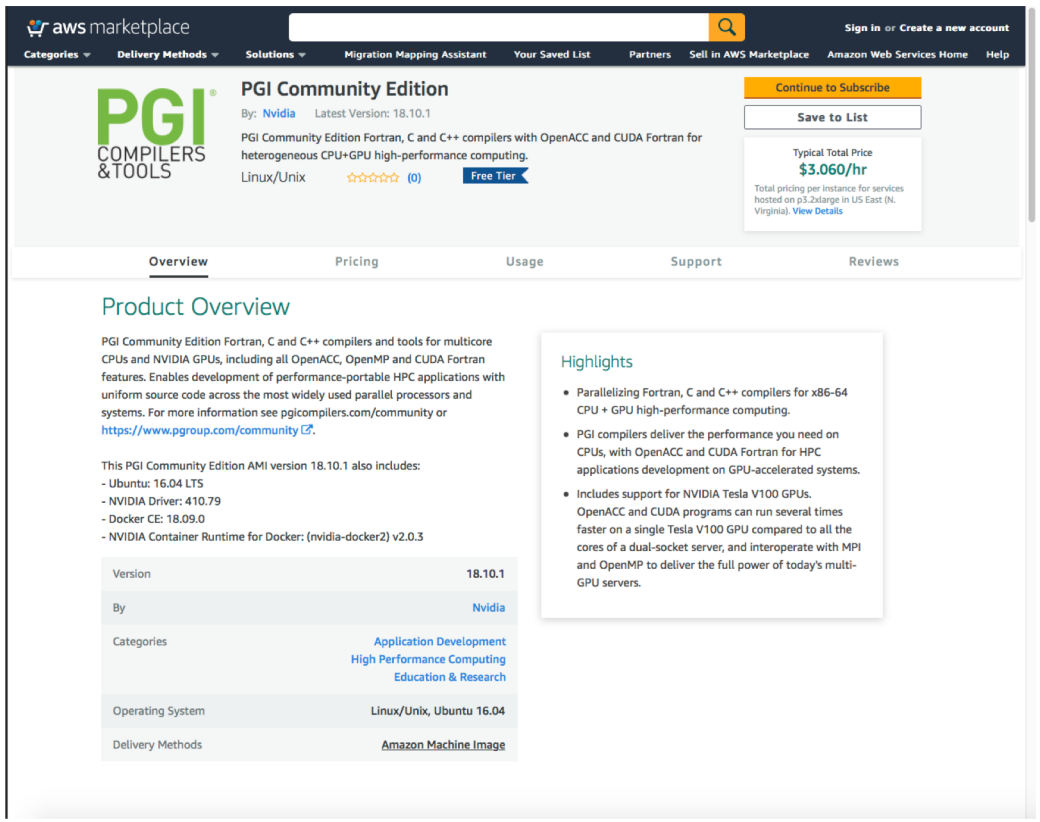
- `-ta=tesla,autocompare`, compare at `copyout` or `update host`
- `-ta=tesla,redundant`, redundant execution
- `acc_compare()` to dynamically compare device-to-host data

CPU to CPU comparison

- `pgi_compare()` to save data to file, then to compare to saved data


PGI in the Cloud
Develop | Test | Benchmark | Deploy

PGI ON AMAZON WEB SERVICES



Easy access to the latest NVIDIA GPU hardware and PGI Community Edition Fortran, C and C++ compilers

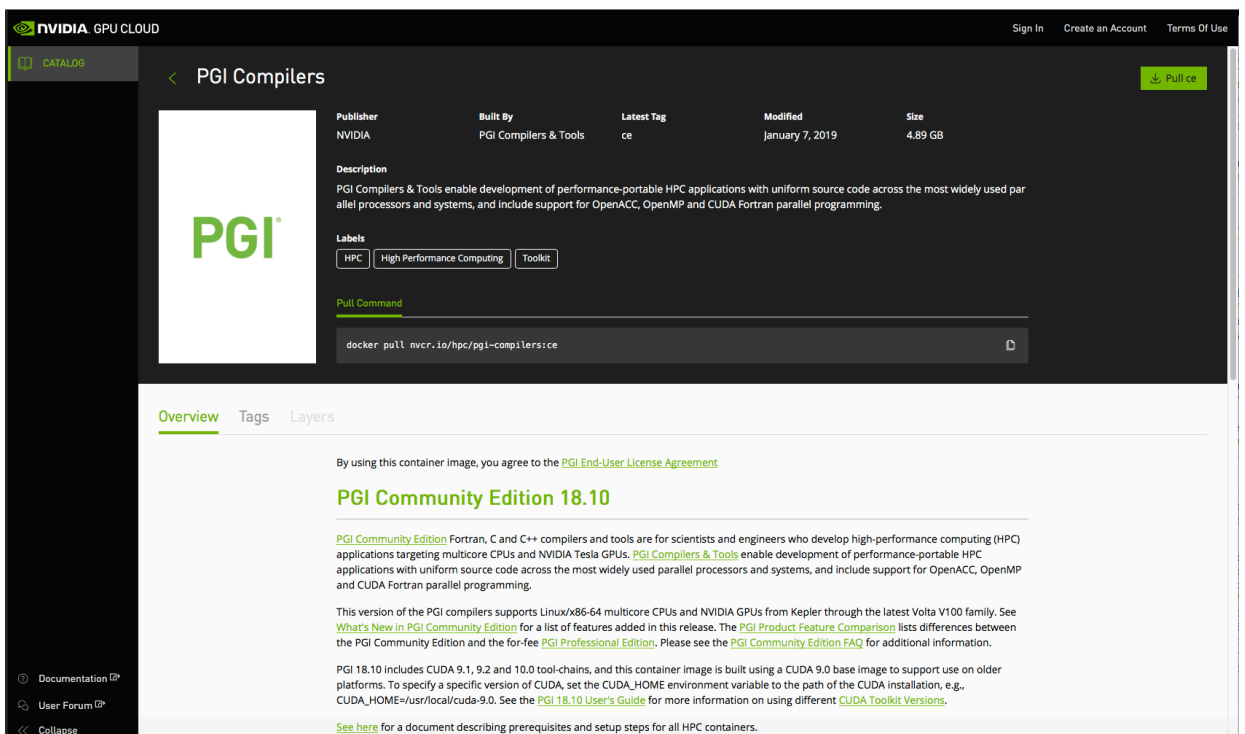
pgicompilers.com/cloud

PGI DOCKER IMAGE ON NVIDIA GPU CLOUD

Run PGI compilers on any Docker-enabled system

Use Containers to reliably run software when moving from one computing environment to another.

ngc.nvidia.com



Future Directions

NEW FEATURES IN OPENACC 2.7

Spec Finalized Nov 2018

Multicore CPU targets

Partially-shared memory

Array / struct reductions

Multi-GPU programming

UPCOMING FEATURES IN OPENACC 3.0

True Deep Copy directives, using feedback for PGI initial implementation

Support for C++ Lambdas, directives within/above Lambdas

Better support for multiple devices (device-to-device copy, ...)

Error handler

Better support for Fortran pointers, Fortran interfaces to all API routines

Update base languages, Fortran 2018, C++17, C18

Many smaller fixes

PCAST IMPROVEMENTS

From User Feedback

More control over compare frequency

Select function, file, variable names

Options to test single file or function

```
#pragma acc compare(a[0:n])

#pragma pgi compare(b[0:n])
```

REAL(2) IN CUDA FORTRAN

```
real(2), allocatable, device :: x(:)
real(2), allocatable :: hx(:)

allocate(x(n), hx(n))

call fill<<<n/128,128>>>(x)

hx(:) = x(:)
print *, hx(1)
```

SCALABLE PARALLELISM WITH OPENACC…

```
% 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
```

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

SCALABLE PARALLELISM WITH OPENMP…

```
% pgfortran -fast -mp=tesla -Minfo -c PdV_kernel.f90
pdv_kernel:
     ...
     77, Tesla device code generated
         77, OMP teams, thread, simd parallelism
         79, collapsed
     ...
```

```
75 !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(2)
76 !$OMP& PRIVATE(right_flux, left_flux, top_flux, bottom_flux, total_flux,
77 !$OMP&         min_cell_volume, energy, change, recip_volume)
78     DO k=y_min,y_max
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
```

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

PARALLEL FEATURES IN FORTRAN AND C++

Fortran 2018

Array syntax (F90)

FORALL (F95)

Co-arrays (F08, F18)

DO CONCURRENT (F08, F18)

C++17

Threads (C++11)

pSTL Parallel Algorithms (C++17)

CLOVERLEAF V1.3

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

AWE Hydrodynamics mini-app

6500+ lines, !$acc kernels

OpenACC or OpenMP

Source on GitHub

FORTRAN WITH OPENACC DIRECTIVES

```
% pgfortran -fast -ta=tesla:managed -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
```

FORTRAN 2018 DO CONCURRENT

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

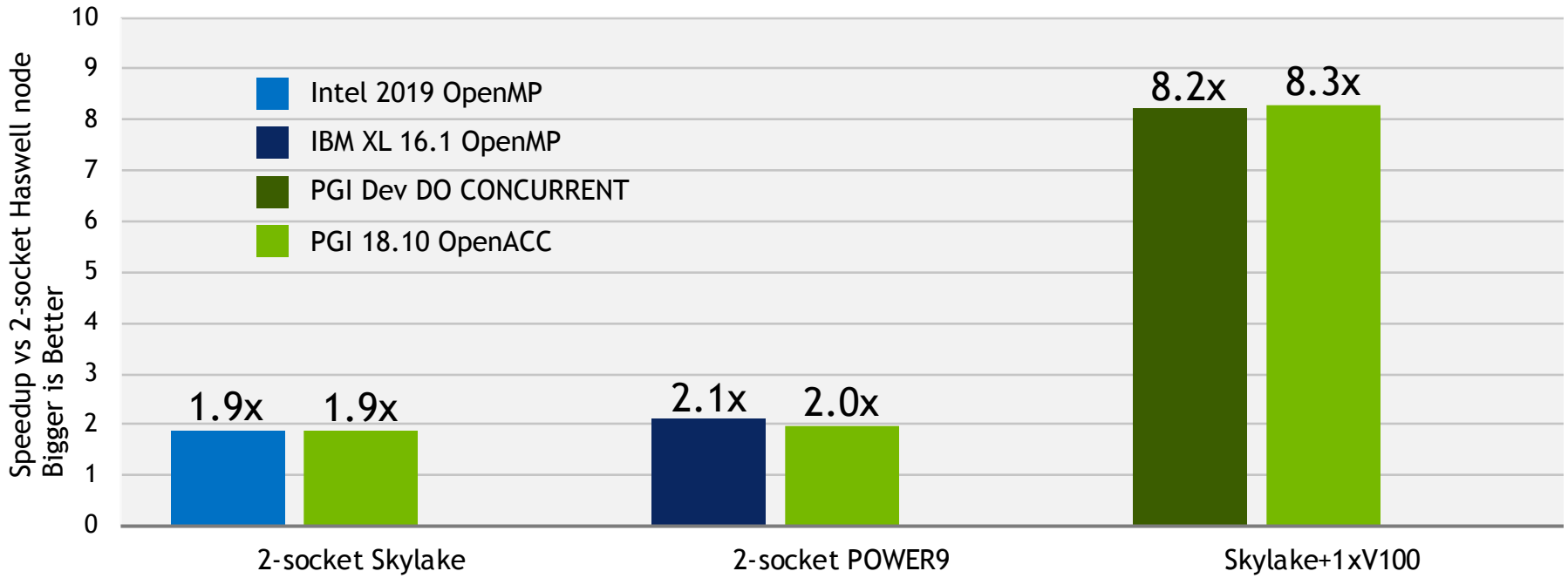
```
75
76
77      DO CONCURRENT (k=y_min:y_max, j=x_min:x_max) &
78                          LOCAL (right_flux,left_flux,top_flux,bottom_flux,total_flux, &
                                   min_cell_volume,energy_change,recip_volume)
79
80
81          left_flux=  (xarea(j  ,k  )*(xvel0(j  ,k  )+xvel0(j  ,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
107
```

https://github.com/UoB-HPC/CloverLeaf_doconcurrent

PGI

FORTRAN 2018 DO CONCURRENT FOR V100

CloverLeaf AWE Hydrodynamics mini-App, bm32 data set



Systems: Skylake 2x20 core Xeon Gold server (sky-4), POWER9 2x20 core POWER9 DD2.1 server (wsn1)
Compilers: Intel 2019.0.045, PGI 18.10, IBM XL 16.1
Benchmark: CloverLeaf v1.3 OpenACC, OpenMP and DoConcurrent versions downloaded from https://github.com/UoB-HPC the week of October 29, 2018

https://github.com/UoB-HPC/CloverLeaf_doconcurrent

PGI

C++17 PARALLEL ALGORITHMS (PSTL)

Parallel and Vector Concurrency for the Standard Algorithms

Execution Policies for most STL algorithms

std::execution::seq, par, par_unseq

A few new parallel algorithms

std::transform_reduce

std::for_each_n

…

C++17 Parallel Algorithms and Beyond, Bryce Lelbach, CppCon 2016

C++17 PARALLEL ALGORITHM: FOR_EACH_N()

```
% pgc++ -fast -ta=tesla:managed,pstl -Minfo -c lulesh.cc
  ...
```

- No compiler feedback
- The parallel STL is a library, not a language
- Library calls are opaque to the compiler
- See S9770 David Olsen Talk C++17 Parallel Algorithms for NVIDIA GPUs with PGI C++

```
2100 static inline
2101 void CalcVelocityForNodes(Domain &domain, const Real_t dt,
2102                           const Real_t u_cut, Index_t numNode)
2103 {
2104
2105
2106    std::for_each_n (POLICY_PAR, idx(0), numNode, [=](Index_t i)
2107    {
2108      Real_t xdtmp, ydtmp, zdtmp ;
2109
2110      xdtmp = domain.xd(i) + domain.xdd(i) * dt ;
2111      if( FABS(xdtmp) < u_cut ) xdtmp = Real_t(0.0);
2112      domain.xd(i) = xdtmp ;
2113
2114      ydtmp = domain.yd(i) + domain.ydd(i) * dt ;
2115      if( FABS(ydtmp) < u_cut ) ydtmp = Real_t(0.0);
2116      domain.yd(i) = ydtmp ;
2117
2118      zdtmp = domain.zd(i) + domain.zdd(i) * dt ;
2119      if( FABS(zdtmp) < u_cut ) zdtmp = Real_t(0.0);
2120      domain.zd(i) = zdtmp ;
2121    });
2122 }
```

THE FUTURE OF GPU PROGRAMMING

Standard Languages | Directives | CUDA

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
```

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

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

...

std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});

...

}
```

```
__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, ...);
```

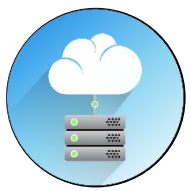
GPU Accelerated C++17 and Fortran 2018

Incremental Performance Optimization with OpenACC

Maximize GPU Performance with CUDA C++/Fortran

PGI C++ AND FORTRAN COMPILERS & TOOLS

A cross-platform HPC SDK and Easy On-ramp to GPU Computing



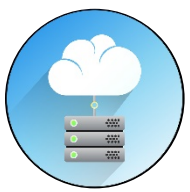
Cloud

HPC Systems

intel AMD OpenPOWER

Workstations

Enabling
Parallel Computing



Cloud

HPC Systems

GPU-Powered
NVIDIA

Workstations

World-class HPC compilers for multicore CPUs

From Sequential, to Parallel, to Accelerated Computing

GPU Acceleration with CUDA, Directives, native C++17/F18