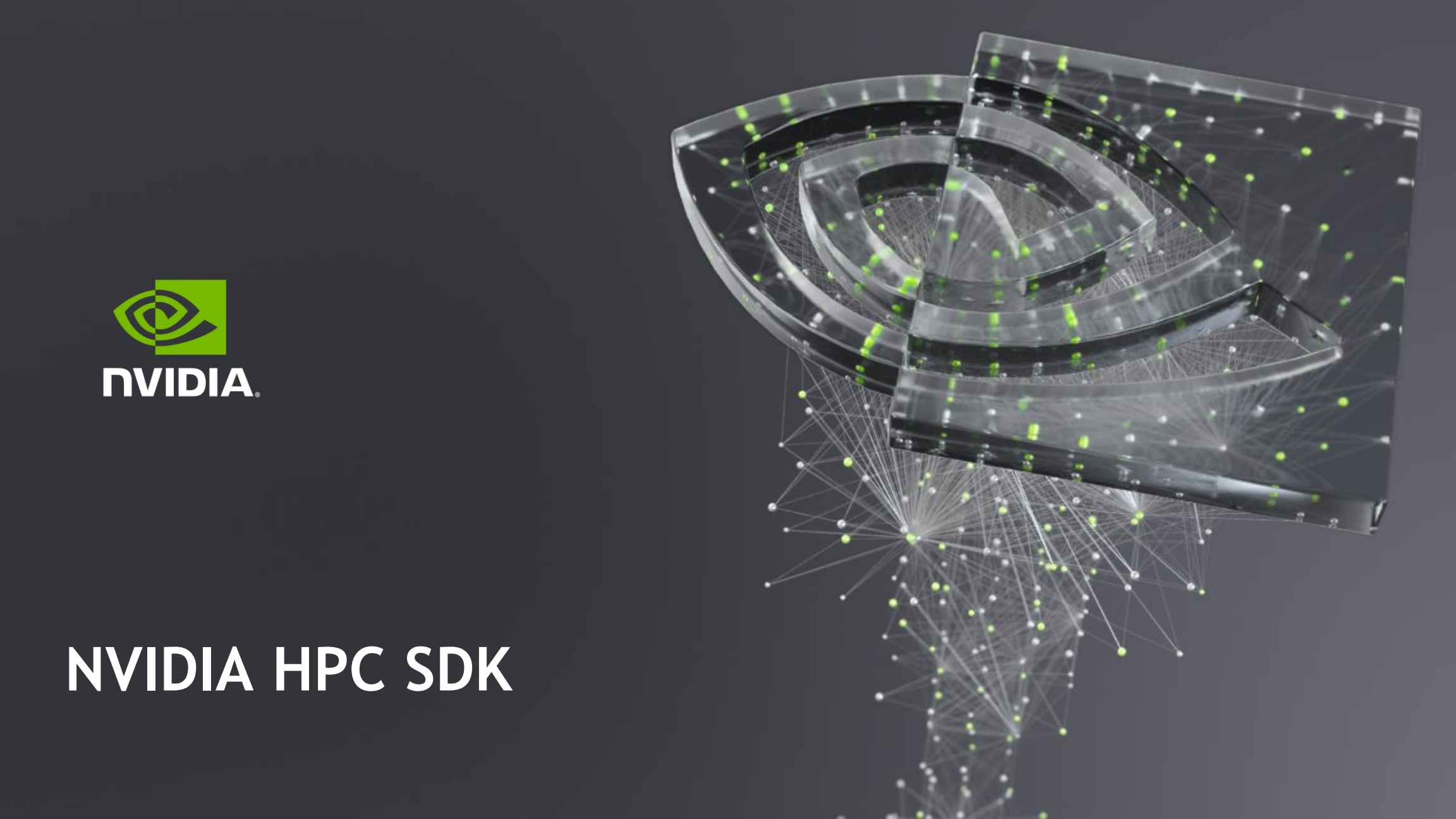


NVIDIA HPC SDK

ACCELERATED PROGRAMMING IN 2020 AND BEYOND

Libraries | 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
```

GPU Accelerated
C++ and Fortran

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

...

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

...

}
```

Incremental Performance
Optimization with Directives

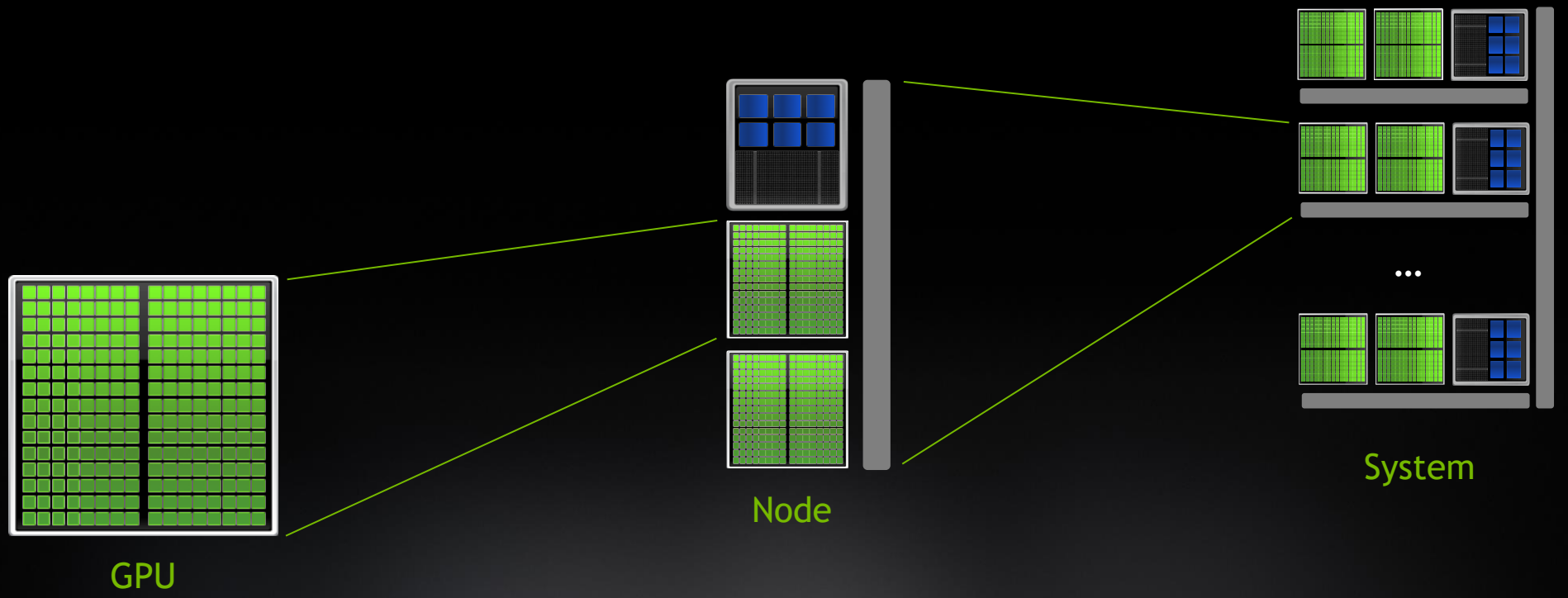
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

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

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

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Libraries

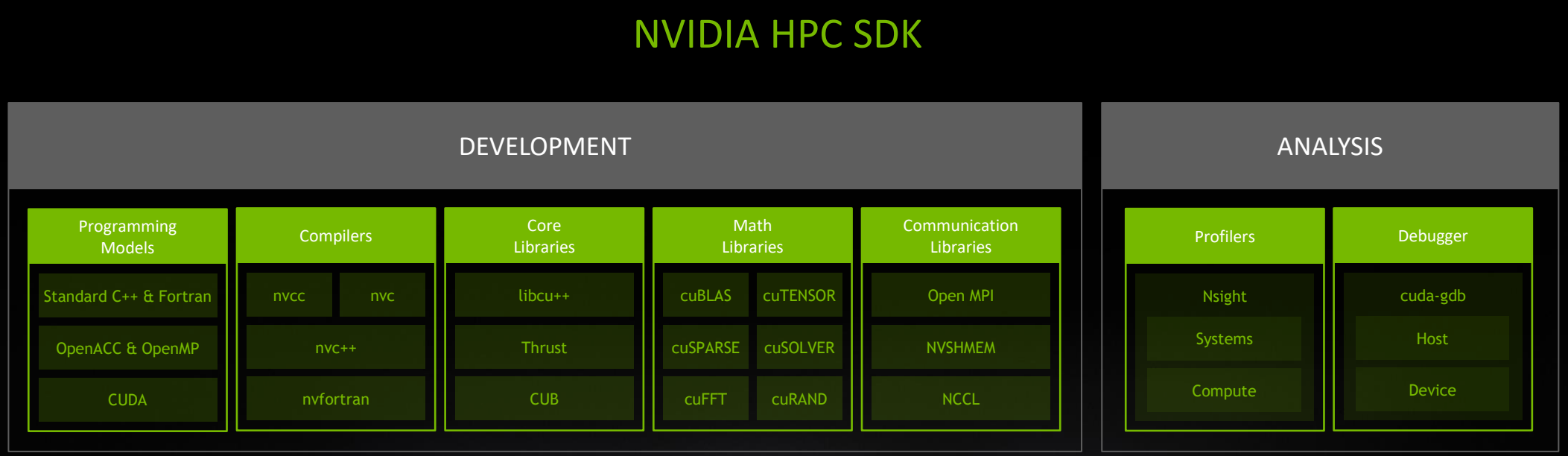
PROGRAMMING GPU-ACCELERATED HPC SYSTEMS

GPU | CPU | Interconnect



AVAILABLE NOW: THE NVIDIA HPC SDK

Available at developer.nvidia.com/hpc-sdk, on NGC, and in the Cloud



Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

7-8 Releases Per Year | Freely Available

ACCELERATED PROGRAMMING IN 2020 AND BEYOND

Libraries | 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
```

GPU Accelerated
C++ and Fortran

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

...

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

...

}
```

Incremental Performance
Optimization with Directives

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

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Libraries

PARALLEL PROGRAMMING IN ISO C++

C++ Parallel Algorithms

```
std::sort(std::execution::par, c.begin(), c.end());

std::unique(std::execution::par, c.begin(), c.end());
```

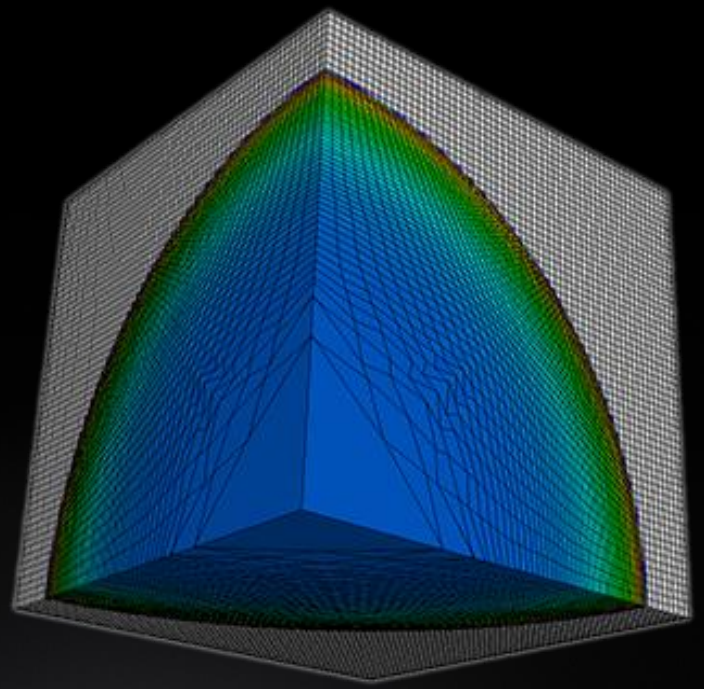
- Introduced in C++17
- Parallel and vector concurrency via execution policies

  `std::execution::par, std::execution::par_unseq, std::execution::seq`
- Several new algorithms in C++17 including
  - `std::for_each_n(POLICY, first, size, func)`
- Insert `std::execution::par` as first parameter when calling algorithms
- NVC++: automatic GPU acceleration

C++17 PARALLEL ALGORITHMS

Lulesh Hydrodynamics Mini-app

- ~9000 lines of C++
- Parallel versions in MPI, OpenMP, OpenACC, CUDA, RAJA, Kokkos, ISO C++…
- Designed to stress compiler vectorization, parallel overheads, on-node parallelism



codesign.llnl.gov/lulesh

PARALLEL C++

- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …

```cpp
static inline
void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                  Index_t *regElemlist, Real_t dvovmax, Real_t& dthydro)
{
#if _OPENMP
  const Index_t threads = omp_get_max_threads();
  Index_t hydro_elem_per_thread[threads];
  Real_t dthydro_per_thread[threads];
#else
  Index_t threads = 1;
  Index_t hydro_elem_per_thread[1];
  Real_t dthydro_per_thread[1];
#endif
#pragma omp parallel firstprivate(length, dvovmax)
  {
    Real_t dthydro_tmp = dthydro ;
    Index_t hydro_elem = -1 ;
#if _OPENMP
    Index_t thread_num = omp_get_thread_num();
#else
    Index_t thread_num = 0;
#endif
#pragma omp for
    for (Index_t i = 0 ; i < length ; ++i) {
      Index_t indx = regElemlist[i] ;

      if (domain.vdov(indx) != Real_t(0.)) {
        Real_t dtdvov = dvovmax / (FABS(domain.vdov(indx))+Real_t(1.e-20)) ;

        if ( dthydro_tmp > dtdvov ) {
          dthydro_tmp = dtdvov ;
          hydro_elem = indx ;
        }
      }
    }
    dthydro_per_thread[thread_num] = dthydro_tmp ;
    hydro_elem_per_thread[thread_num] = hydro_elem ;
  }
  for (Index_t i = 1; i < threads; ++i) {
    if(dthydro_per_thread[i] < dthydro_per_thread[0]) {
      dthydro_per_thread[0] = dthydro_per_thread[i];
      hydro_elem_per_thread[0] = hydro_elem_per_thread[i];
    }
  }
  if (hydro_elem_per_thread[0] != -1) {
    dthydro = dthydro_per_thread[0] ;
  }
  return ;
}
```

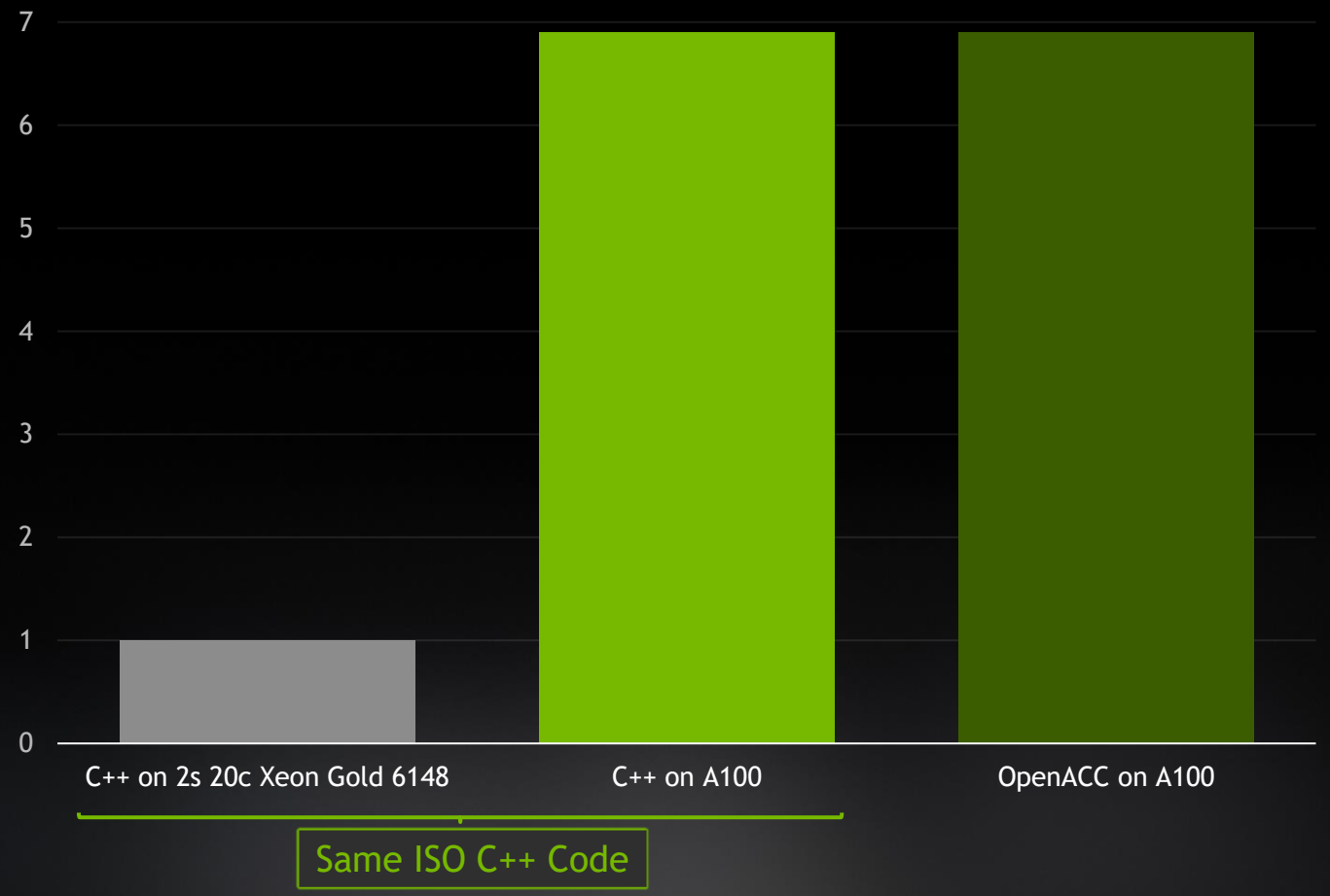
C++ with OpenMP

```cpp
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                                               Index_t *regElemlist,
                                               Real_t dvovmax,
                                               Real_t &dthydro)
{
  dthydro = std::transform_reduce(
    std::execution::par, counting_iterator(0), counting_iterator(length),
    dthydro, [](Real_t a, Real_t b) { return a < b ? a : b; },
    [=, &domain](Index_t i)
  {
      Index_t indx = regElemlist[i];
      if (domain.vdov(indx) == Real_t(0.0)) {
        return std::numeric_limits<Real_t>::max();
      } else {
        return dvovmax / (std::abs(domain.vdov(indx)) + Real_t(1.e-20));
      }
  });
}
```

Parallel C++17

LULESH PERFORMANCE

Speedup – Higher is Better



STLBM

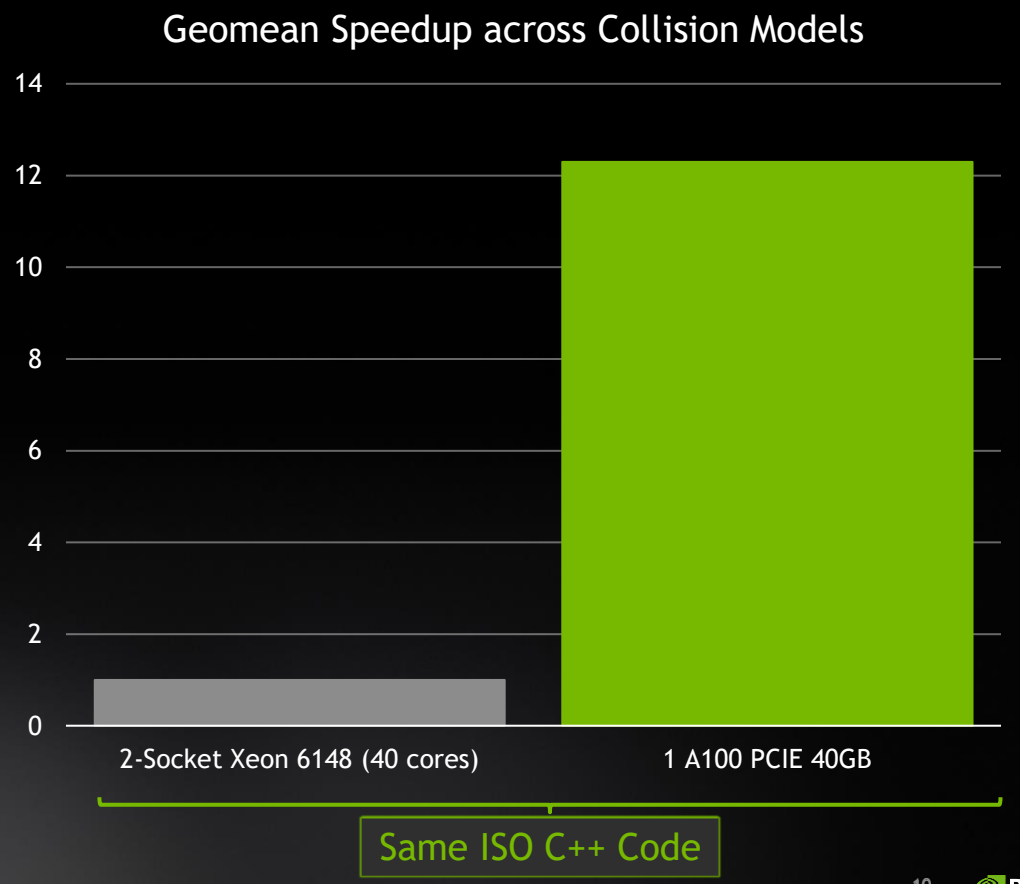
Many-core Lattice Boltzmann with C++ Parallel Algorithms

- Framework for parallel lattice-Boltzmann simulations on multiple platforms, including many-core CPUs and GPUs
- Implemented with C++17 standard (Parallel Algorithms) to achieve parallel efficiency
- No language extensions, external libraries, vendor-specific code annotations, or pre-compilation steps

"We have with delight discovered the NVidia "stdpar" implementation of C++17 Parallel Algorithms. … We believe that the result produces state-of-the-art performance, is highly didactical, and introduces ***a paradigm shift in cross-platform CPU/GPU programming*** *in the community."*

-- Professor Jonas Latt, University of Geneva

https://gitlab.com/unigehpfs/stlbm



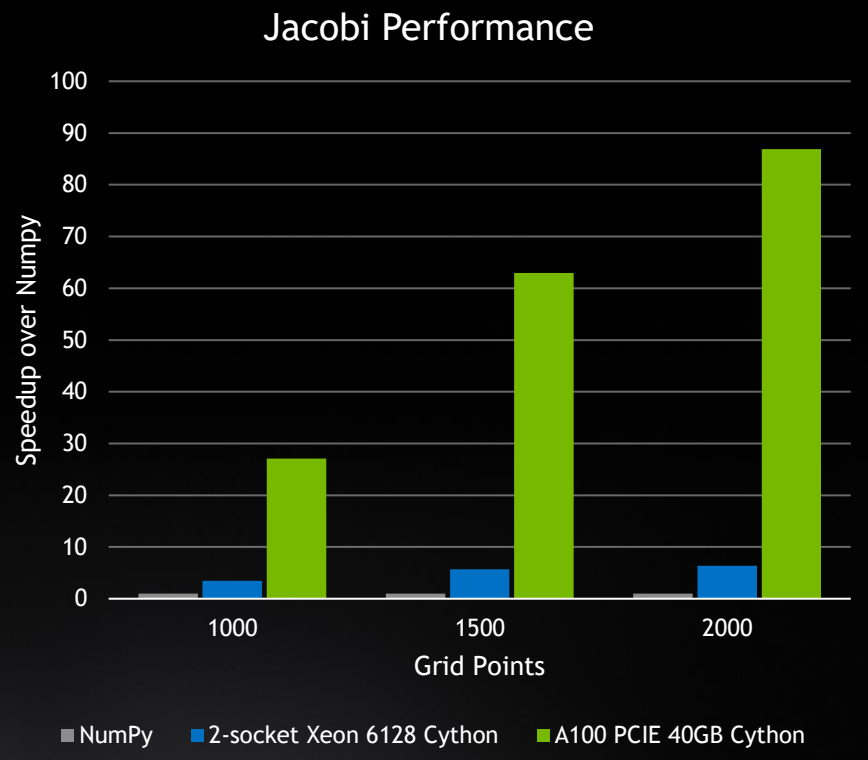
PARALLEL C++ & CYTHON

Using NVC++ and CYTHON to Accelerate Python

A100 Performance for Python

- Access to C++ performance with Cython
- A100 Acceleration with NVC++ stdpar in a Jupyter Notebook
- Up to 87X Speed-up over Numpy

```
def jacobi_solver(float[:, :] data, float max_diff, int max_iter=10_000):
    M, N  = data.shape[0], data.shape[1]
...
    keep_going = True
    while keep_going and iterations < max_iter:
        iterations += 1
        for_each(par, indices.begin(), indices.end(), avg(T1, T2, M, N))
        keep_going = any_of(par, indices.begin(),
                                 indices.end(), converged(T1, T2, max_diff))
        swap(T1, T2)
...
    return iterations
```



HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

C++17

Parallel Algorithms

- In NVC++
- Parallel and vector concurrency

Forward Progress Guarantees

- Extend the C++ execution model for accelerators

Memory Model Clarifications

- Extend the C++ memory model for accelerators

C++20

Scalable Synchronization Library

- Express thread synchronization that is portable and scalable across CPUs and accelerators
- In libcu++:
  - `std::atomic<T>`
  - `std::barrier`
  - `std::counting_semaphore`
  - `std::atomic<T>::wait/notify_*`
  - `std::atomic_ref<T>`

Coming to NVC++ Soon

C++23 and Beyond

Executors

- Simplify launching and managing parallel work across CPUs and accelerators

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Linear Algebra

- C++ standard algorithms API to linear algebra
- Maps to vendor optimized BLAS libraries

Extended Floating Point Types

- First-class support for formats new and old: `std::float16_t/float64_t`

C++ Linear Algebra

```
// Matrix
double* A = new double[nRows*nCols];
// Vectors
double* y = new double[nRows];
double* x = new double[nCols];

// y = 1.0*A*x;
dgemv('N',nRows,nCols,1.0,A,nRows,x,1,0.0,y,1)
```

```
// Matrix
mdspan<const double,dynamic_extent,dynamic_extent> A(A_ptr,nRows,nCols);
// Vectors
mdspan<const double,dynamic_extent> x(x_ptr,nCols);
mdspan<double,dynamic_extent> y(y_ptr,nRows);

// y = 1.0*A*x;
matrix_vector_product(y,A,x);
```

BLAS

- 11 arguments to do Ax=b
- Scalar types in function name
- No mixed precision support
- Fortran data layout (or transpose tricks)

Standard C++ BLAS

- Only the 3 expected arguments to do Ax=b
- Function name doesn't encode scalar type
- Naturally handle mixed precision
- Includes equivalent of every BLAS function
- Will map to cuBLAS – tensor cores in ISO C++!

Executors

```
// Existing code.
void compute(int resource, …) {
  switch(resource) {
    case GPU:
      kernel<<<…>>>(…);
      …
    case MULTI_GPU:
      cudaSetDevice(0);
      kernel<<<…>>>(…);
      cudaSetDevice(1);
      kernel<<<…>>>(…);
      …
    case SIMD:
      #pragma simd
      …
    case OPENMP:
      #pragma omp parallel for
      …
  }
}
```

```
Void compute(Executor ex, …)
{
  // use ex uniformly
  ex.execute(…);
}
```

Simplify Work Across CPUs and Accelerators

- Uniform abstraction between code and diverse resources
- ISO standard
- Standardize Kernel Launch
- **Write once, run everywhere**

Organize CUDA Code

- Regularize kernel launches
- Streamline recurring programming tasks
- Safen error-prone launches
- Collect CUDA-specific features

HPC PROGRAMMING IN ISO FORTRAN

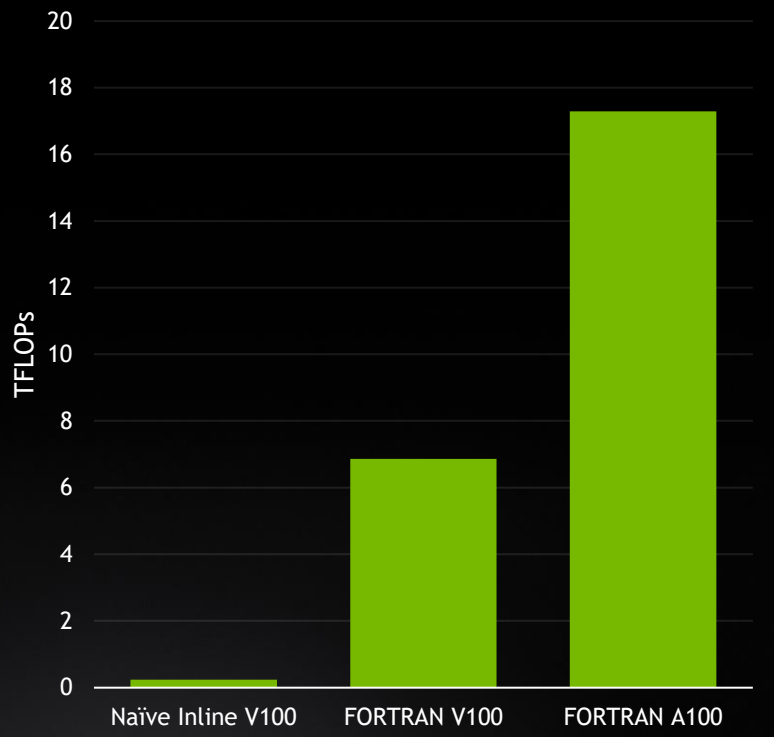
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c
...
!$acc enter data copyin(a,b,c) create(d)

do nt = 1, ntimes
  !$acc kernels
  do j = 1, nj
    do i = 1, ni
      d(i,j) = c(i,j)
      do k = 1, nk
        d(i,j) = d(i,j) + a(i,k) * b(k,j)
      end do
    end do
  end do
    !$acc end kernels
end do

!$acc exit data copyout(d)
```

Inline FP64 matrix multiply

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c

...

do nt = 1, ntimes
  d = c + matmul(a,b)
end do
```

MATMUL FP64 matrix multiply



HPC PROGRAMMING IN ISO FORTRAN

Examples of Patterns Accelerated in NVFORTRAN

```
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(transpose(b))
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(b)
d = reshape(a,shape=[ni,nj,nk])
d = reshape(a,shape=[ni,nk,nj])
d = 2.5 * sqrt(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = alpha * conjg(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = reshape(a,shape=[ni,nk,nj],order=[1,3,2])
d = reshape(a,shape=[nk,ni,nj],order=[2,3,1])
d = reshape(a,shape=[ni*nj,nk])
d = reshape(a,shape=[nk,ni*nj],order=[2,1])
d = reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1])
d = abs(reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1]))
c = matmul(a,b)
c = matmul(transpose(a),b)
c = matmul(reshape(a,shape=[m,k],order=[2,1]),b)
c = matmul(a,transpose(b))
c = matmul(a,reshape(b,shape=[k,n],order=[2,1]))
```

```
c = matmul(transpose(a),transpose(b))
c = matmul(transpose(a),reshape(b,shape=[k,n],order=[2,1]))
d = spread(a,dim=3,ncopies=nk)
d = spread(a,dim=1,ncopies=ni)
d = spread(a,dim=2,ncopies=nx)
d = alpha * abs(spread(a,dim=2,ncopies=nx))
d = alpha * spread(a,dim=2,ncopies=nx)
d = abs(spread(a,dim=2,ncopies=nx))
d = transpose(a)
d = alpha * transpose(a)
d = alpha * ceil(transpose(a))
d = alpha * conjg(transpose(a))
c = c + matmul(a,b)
c = c - matmul(a,b)
c = c + alpha * matmul(a,b)
d = alpha * matmul(a,b) + c
d = alpha * matmul(a,b) + beta * c
```

HPC PROGRAMMING IN ISO FORTRAN

DO CONCURRENT

DO CONCURRENT in NVFORTRAN

- Available in NVFORTRAN 20.11
- Automatic GPU acceleration & multi-core support
- Syntax for nested parallelism / loop collapse; expose more parallelism to the compiler

```
subroutine smooth( a, b, w0, w1, w2, n, m, niters )
 real, dimension(:,:) :: a,b
 real :: w0, w1, w2
 integer :: n, m, niters
 integer :: i, j, iter
  do iter = 1,niters
   do concurrent(i=2 : n-1, j=2 : m-1)
     a(i,j) = w0 * b(i,j) + &
              w1 * (b(i-1,j) + b(i,j-1) + b(i+1,j) + 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))
   enddo
   do concurrent(i=2 : n-1, j=2 : m-1)
     b(i,j) = w0 * a(i,j) + &
              w1 * (a(i-1,j) + a(i,j-1) + a(i+1,j) + a(i,j+1)) + &
              w2 * (a(i-1,j-1) + a(i-1,j+1) + a(i+1,j-1) + a(i+1,j+1))
   enddo
  enddo
```

HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Array Syntax and Intrinsics
- NVFORTRAN 20.5
- Accelerated matmul, reshape, spread, …

DO CONCURRENT
- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays
- Coming Soon
- Accelerated co-array images

Coming to NVFORTRAN Soon

Fortran 202x

DO CONCURRENT Reductions
- `REDUCE` subclause added
- Support for `+`, `*`, `MIN`, `MAX`, `IAND`, `IOR`, `IEOR`.
- Support for `.AND.`, `.OR.`, `.EQV.`, `.NEQV` on `LOGICAL` values
- Atomics

ACCELERATED PROGRAMMING IN 2020 AND BEYOND

Libraries | 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
```

GPU Accelerated
C++ and Fortran

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

...

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

...

}
```

Incremental Performance
Optimization with Directives

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

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Libraries

OpenACC DIRECTIVES

Manage Data Movement

Initiate Parallel Execution

Optimize Loop Mappings

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

OpenACC
Directives for Accelerators

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

PERFORMANCE PORTABLE

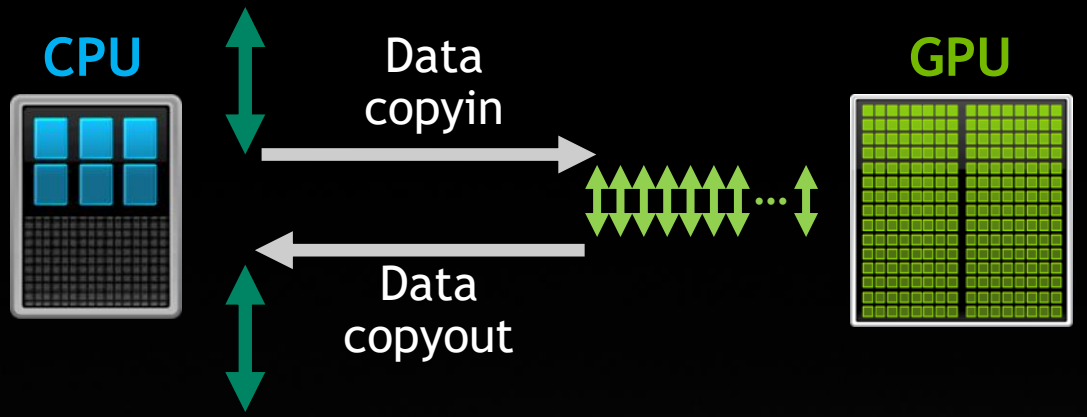
```
 98 !$acc parallel
 99 !$acc loop independent
100       DO k=y_min-depth,y_max+depth
101 !$acc loop independent
102         DO j=1,depth
103           density0(x_min-j,k)=left_density0(left_xmax+1-j,k)
104         ENDDO
105       ENDDO
106 !$acc end parallel
```


```
% nvfortran -acc=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
```

```
% nvfortran -acc=gpu –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 AUTO-COMPARE

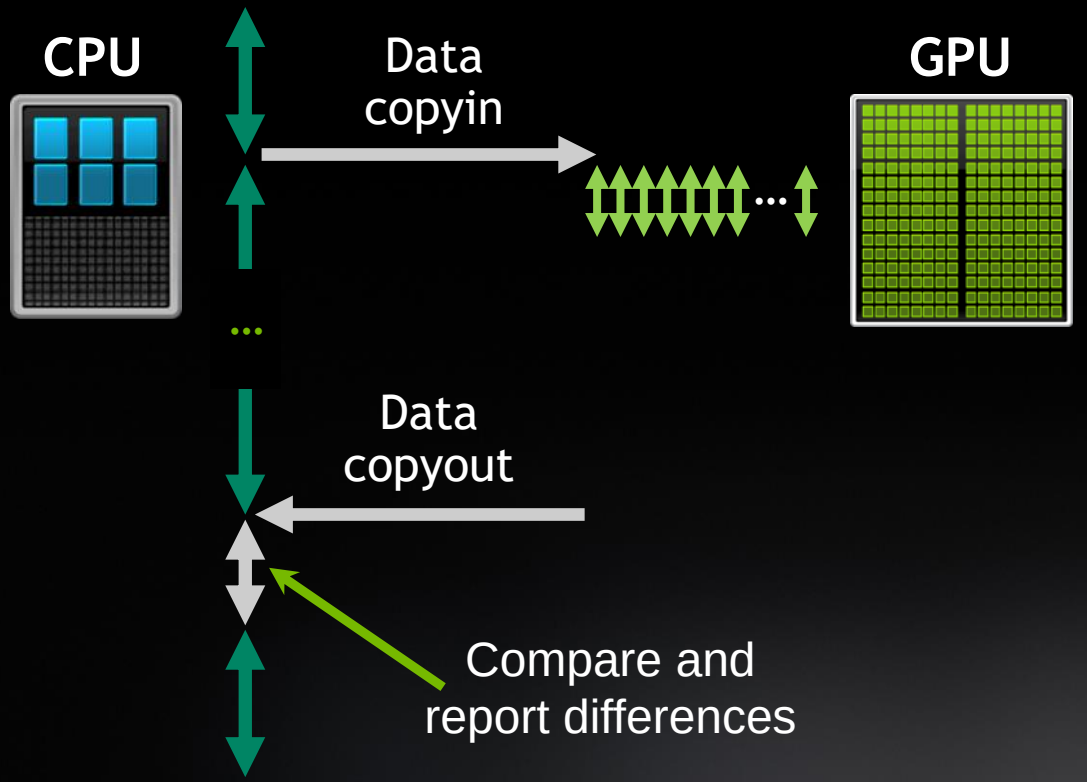
Find where CPU and GPU numerical results diverge



Normal OpenACC execution mode, no auto-compare

OpenACC AUTO-COMPARE

Find where CPU and GPU numerical results diverge



`-acc -gpu=autocompare`

Compute regions run redundantly on CPU and GPU

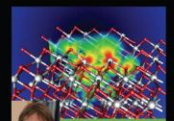
Results compared when data copied from GPU to CPU

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

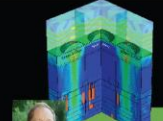
"For VASP, OpenACC is the way forward for GPU acceleration. Performance is similar to CUDA C, and OpenACC dramatically decreases GPU development and maintenance efforts."

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

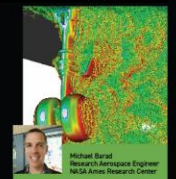
"The OMAR project provided us with early access to Summit hardware and access to PGI compiler experts. Both of these were critical to our success. PGI's OpenACC support remains the best available and is competitive with much more intrusive programming model approaches."

VMD

John Stone
Senior Research Programmer
Beckman 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 accelerated "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

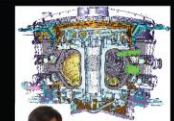
"Our team has been evaluating OpenACC as a pathway to performance portability for the Model for Prediction (MPAS) atmospheric model. Using this approach on the MPAS dynamical core, we have achieved performance on a single P100 GPU equivalent to 2.7 dual socketed Intel Xeon nodes on our new Cheyenne supercomputer."

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



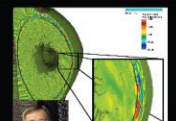
Zhihong Lin
Professor
and Principal Investigator
UC Irvine

"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."

OpenACC

More Science, Less Programming

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 1.1x speed-ups compared to CPU-only execution when running at scale on 2668 nodes of ORNL's new Summit supercomputer."

HIPSTAR

Richard Sandberg
Investigator
University of Melbourne

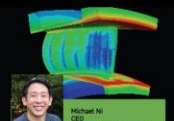
"For a relatively small production case of about 80 million grid points, our OpenACC-enabled version running on two CPU nodes with 4 P100s each was approximately 20.5x faster than two nodes with 24 Haswell cores each. We're looking forward to running a larger simulation on Summit early in 2019."

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.

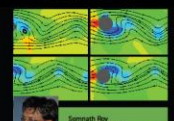
"Using OpenACC on a single Tesla V100 in the Amazon Cloud, our GPU-accelerated Code Leo flow solver runs 14 times faster at 30% less cost compared to runs using all 72 vCPU cores of a Xeon Platinum c5.18xlarge instance. We feel this will revolutionize the aerospace design cycle, enabling the delivery 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 2.1x 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 fast air 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

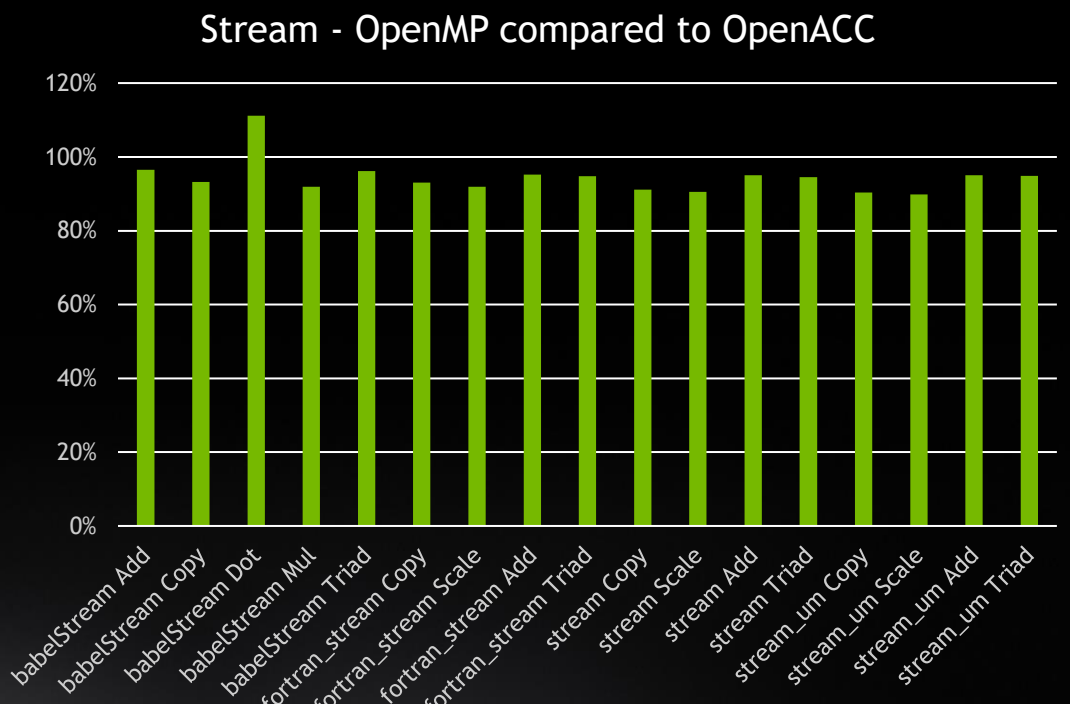
Igor Sfiligoi
HPC Software Developer
General Atomics

"We planned to spend a month porting our OpenACC code from X86 + GPUs to POWER9 + GPUs. Using the PGI compilers and our standard build environment, we were running in an afternoon. It just worked!"

OpenMP Target Offload

Open Beta Available in HPC SDK

- Available in NVC++ and NVFORTRAN 20.11
- Supporting a subset of the OpenMP 5.0 specification
  - Determined by application priorities
  - Details provided in product documentation
- Interoperable – support for CUDA streams



ACCELERATED PROGRAMMING IN 2020 AND BEYOND

Libraries | 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
```

GPU Accelerated
C++ and Fortran

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

...

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

...

}
```

Incremental Performance
Optimization with Directives

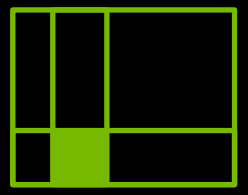
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

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

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

Maximize GPU Performance with
CUDA C++/Fortran

GPU Accelerated Libraries

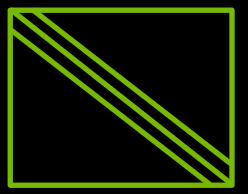
A100 FEATURES IN MATH LIBRARIES

Automatic Acceleration of Critical Routines in HPC and AI



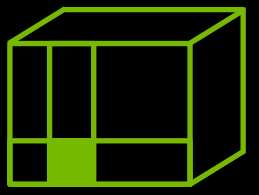
cuBLAS

BF16, TF32 and FP64 Tensor Cores



cuSPARSE

Sparse MMA Tensor Core, Increased memory BW, Shared Memory and L2

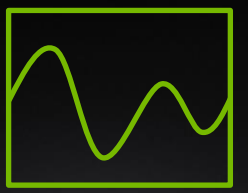


cuTENSOR

BF16, TF32 and FP64 Tensor Cores

cuSOLVER

BF16, TF32 and FP64 Tensor Cores



cuFFT

Increased memory BW, Shared Memory and L2

CUDA Math API

BF16 Support

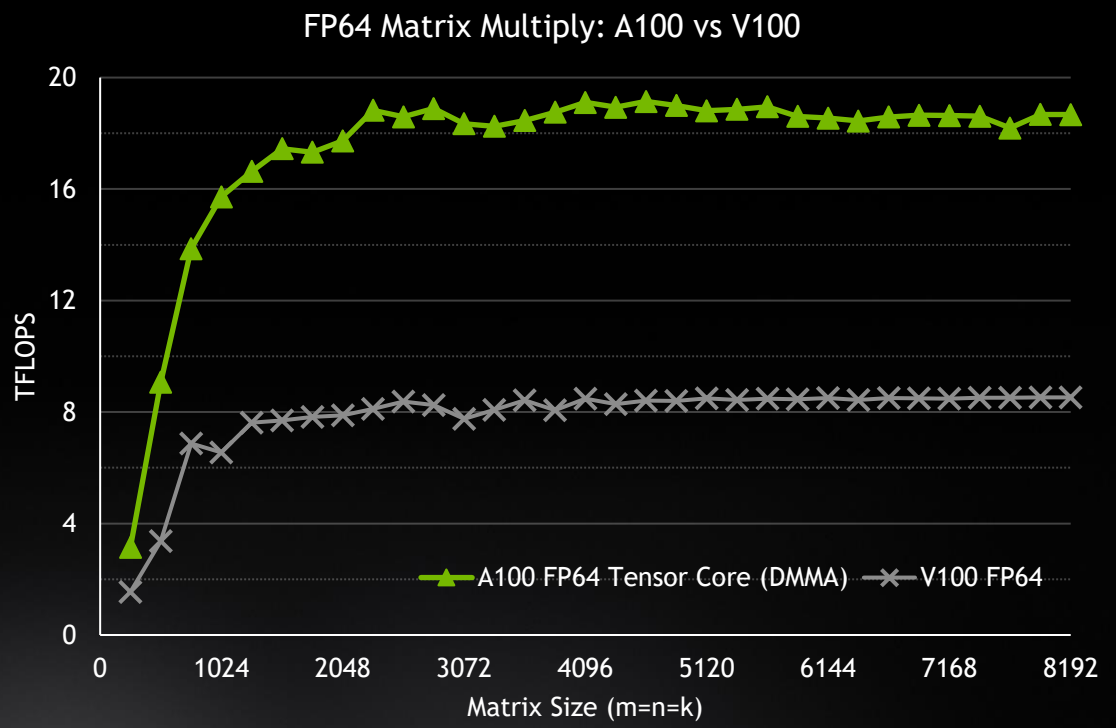
A100 TENSOR CORES IN LIBRARIES

cuBLAS

- Automatic Tensor Core acceleration
- Removed matrix size restrictions for Tensor Core acceleration

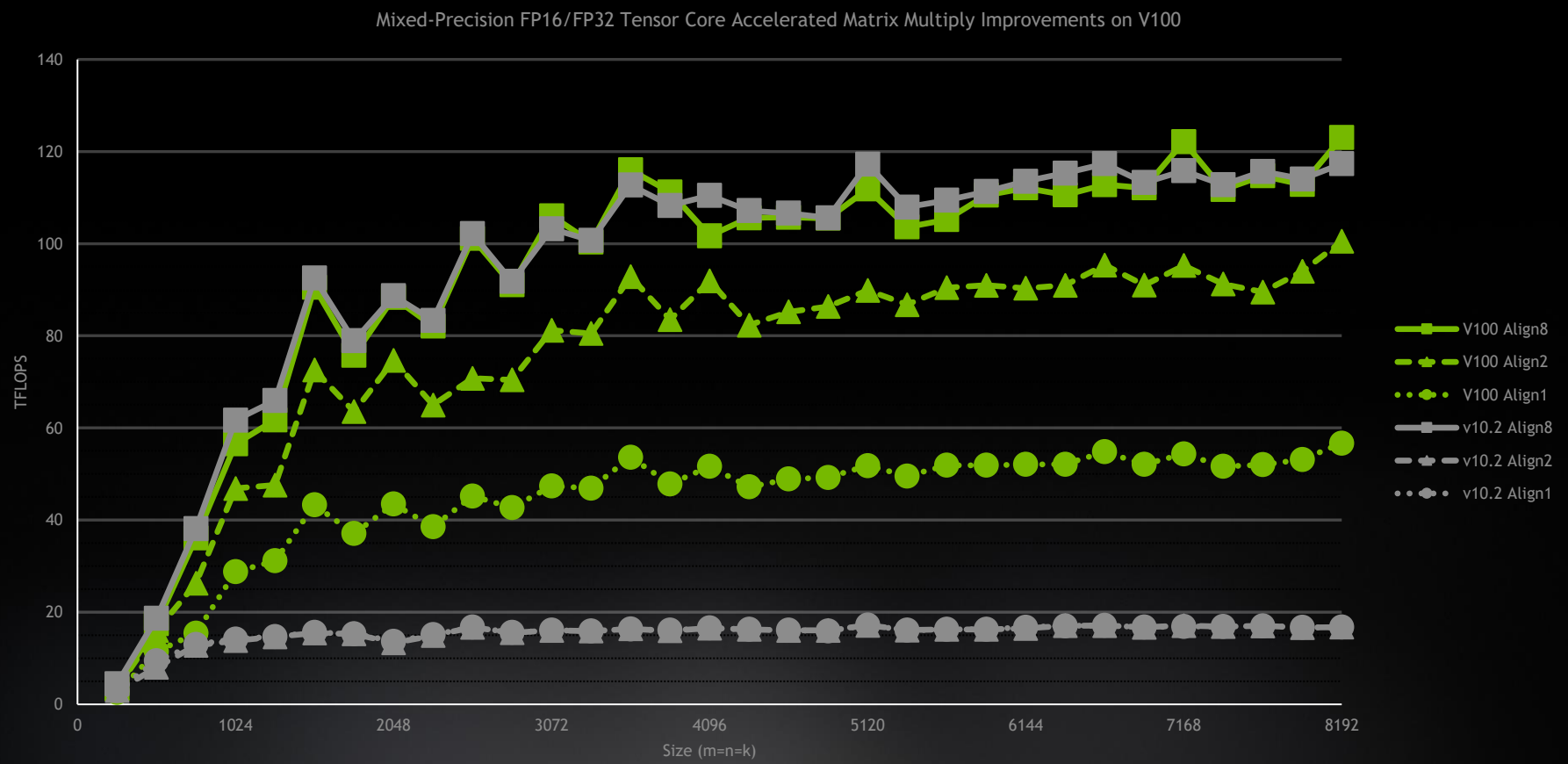
DGEMM on A100

- Up to 19 TFLOPs, 2.4x V100



TENSOR CORES IN LIBRARIES

Eliminating Alignment Requirements To Activate Tensor Cores for MMA



AlignN means alignment to 16-bit multiplies of N. For example, align8 are problems aligned to 128bits or 16 bytes.

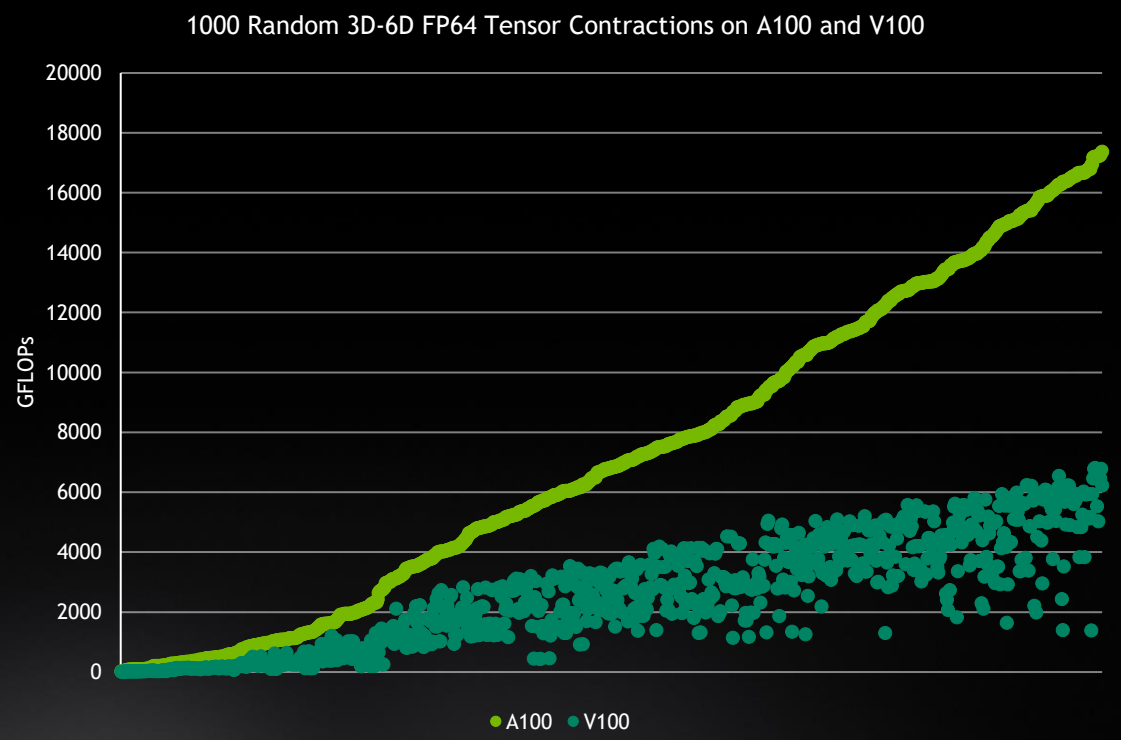
A100 TENSOR CORES IN LIBRARIES

cuTENSOR Tensor Contraction

- Tensor Contractions and Reductions
- Elementwise Operations
- Mixed Precision Support
- Elementwise Fusion
- Automatic Tensor Core Acceleration

A100 vs V100

- Up to 13.1X Speedup
- Average 3.4X Speedup



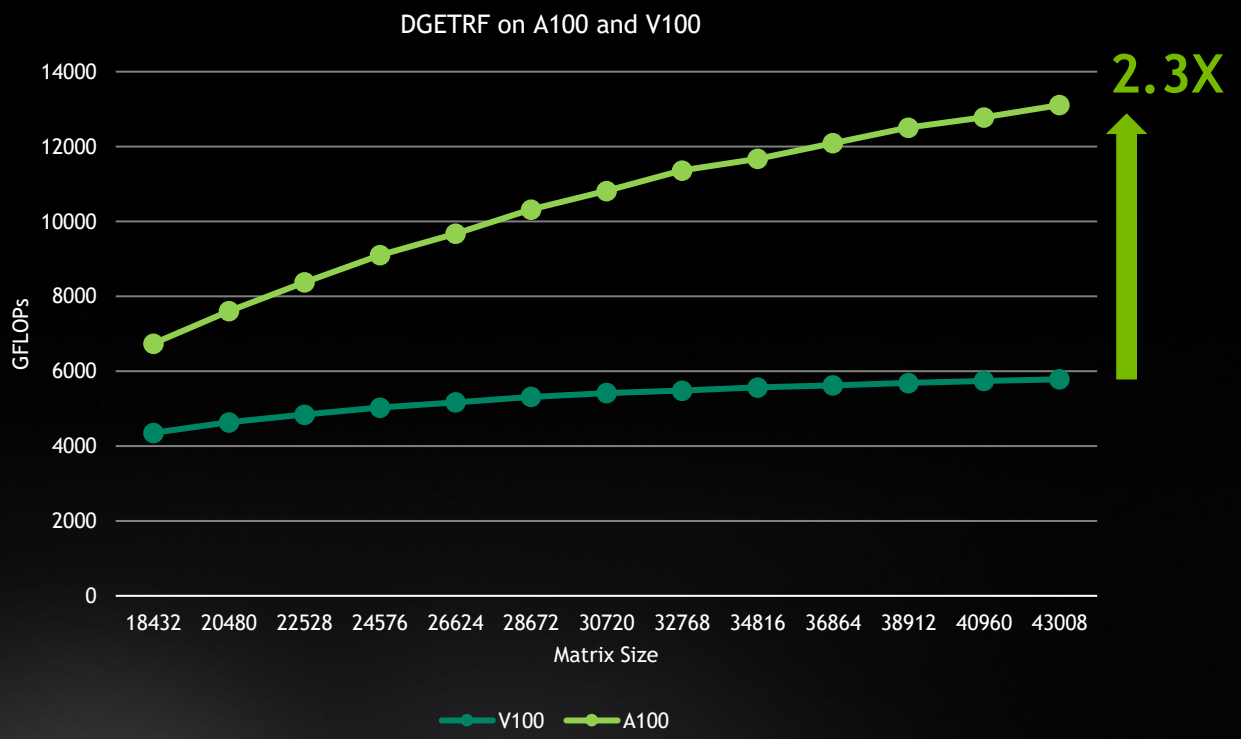
A100 TENSOR CORES IN LIBRARIES

cuSOLVER Linear Solvers

- Automatic DMMA acceleration for factorizations and linear solvers
- Real and Complex LU, Cholesky, QR

A100 vs V100

- Up to 2.8X Speedup



A100 data collected with pre-production hardware and software

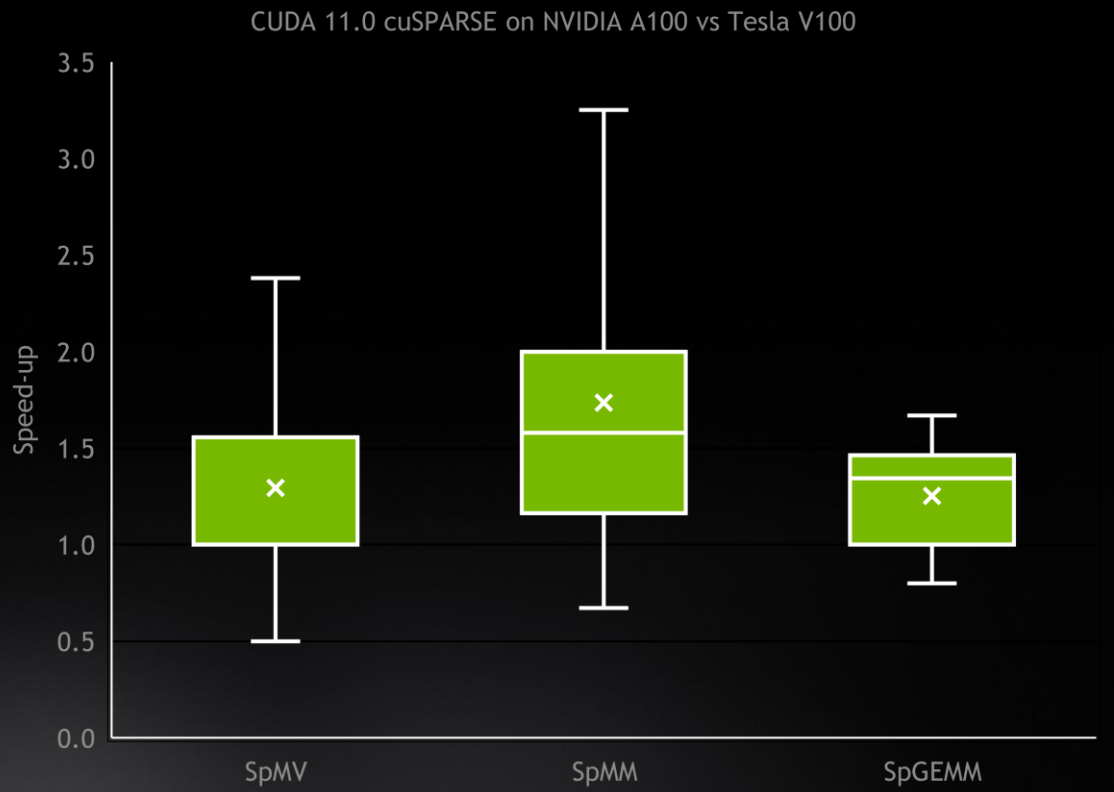
cuSPARSE

Generic APIs and A100 Support

New generic APIs with improved performance

- SpVV - Sparse Vector Dense Vector Multiplication
- SpMV – Sparse Matrix Dense Vector Multiplication
- SpMM – Sparse Matrix Dense Matrix Multiplication
- SpGEMM – Sparse Matrix Sparse Matrix Multiplication
  - 4.6 GEOMEAN speedup over legacy APIs

```
cusparseStatus_t
cusparseSpMM(cusparseHandle_t              handle,
             cusparseOperation_t           transA,
             cusparseOperation_t           transB,
             const void*                   alpha,
             const cusparseSpMatDescr_t    matA,
             const cusparseDenseMatDescr_t matB,
             const void*                   beta,
             cusparseDenseMatDescr_t       matC,
             cudaDataType                  computeType,
             cusparseSpMMAlg_t             alg,
             void*                         externalBuffer)
```

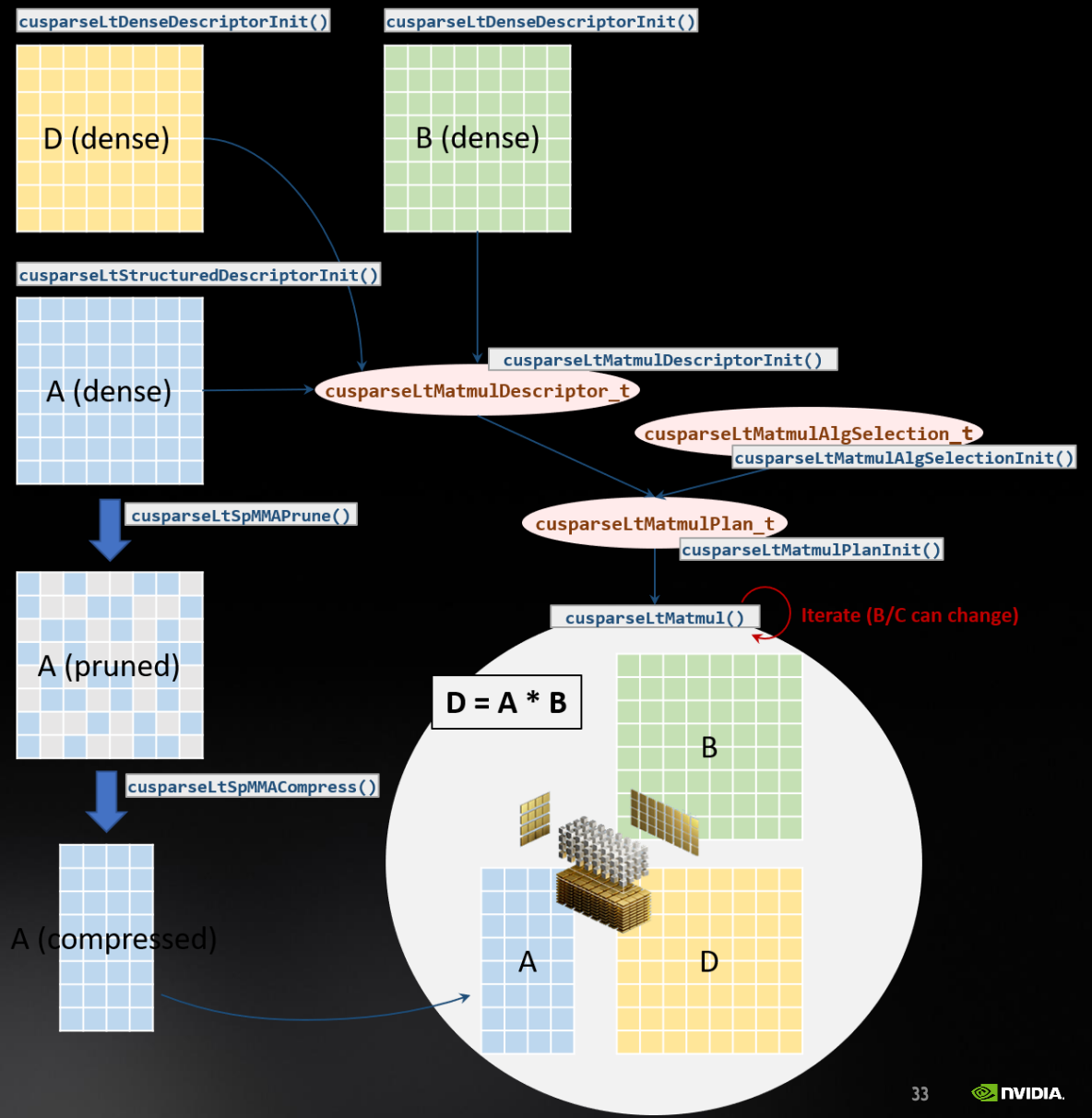


cuSPARSELt

Extension Library with Sparse Matmul

- High-performance library for general matrix-matrix operations in which at least one operand is a sparse matrix
- Ampere Sparse MMA tensor core support
- Mixed-precision support
- Matrix pruning and compression functionalities
- Auto-tuning functionality

```
cusparseStatus_t
cusparseLtMatmul(cusparseHandle_t*          handle,
                 cusparseMatmulPlan_t*      transA,
                 const void*                alpha,
                 const void*                d_A,
                 const void*                d_B,
                 const void*                beta,
                 const void*                d_C,
                 void*                      d_D,
                 void*                      workspace,
                 cudaStream_t*              streams,
                 int32_t                    numStreams)
```



MULTI GPU SUPPORT IN LIBRARIES

Linear Algebra and FFT

cuFFT

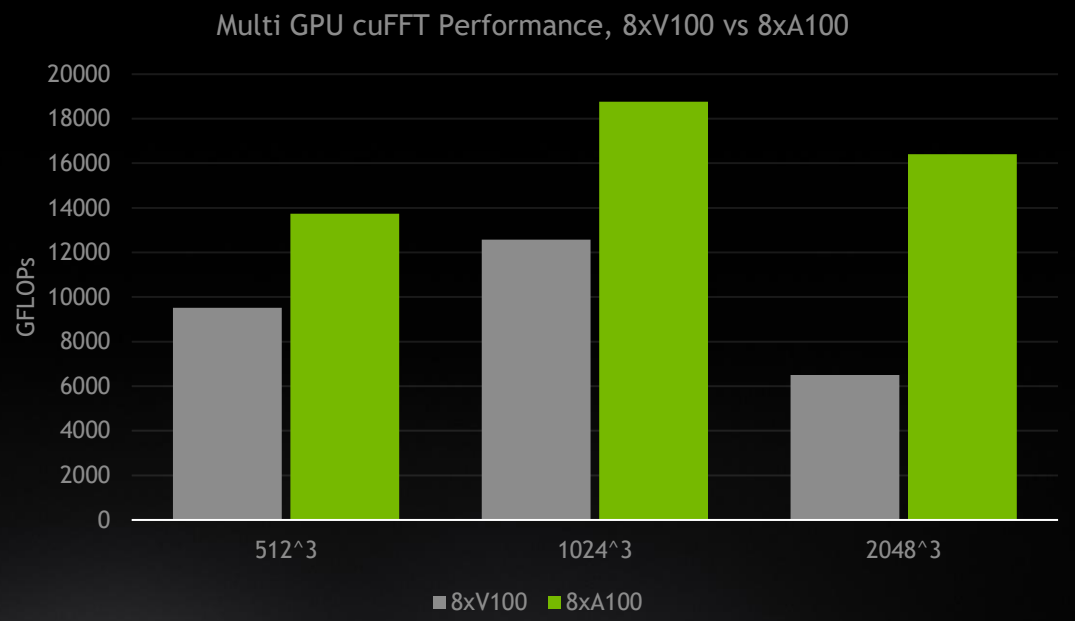
- Single Process Multi-GPU FFT
- **Multi Node Multi-GPU FFT Coming Soon**

cuSOLVER

- Single Process Multi-GPU Eigensolver
- Single Process Multi-GPU LU
- Single Process Multi-GPU Cholesky
- **Multi Node Multi-GPU LU Coming Soon**

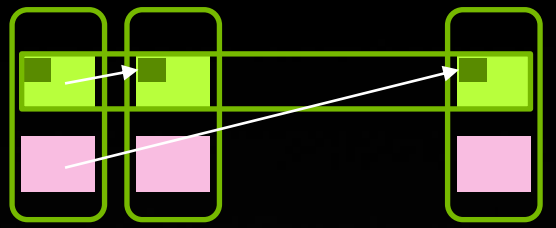
cuBLAS

- Improved Single Process Multi-GPU GEMM

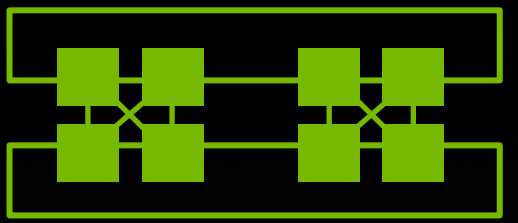


Communication **Libraries**

Single GPU, Multi GPU, and Multi Node



NVSHMEM



NCCL

+ CUDA-Aware OpenMPI

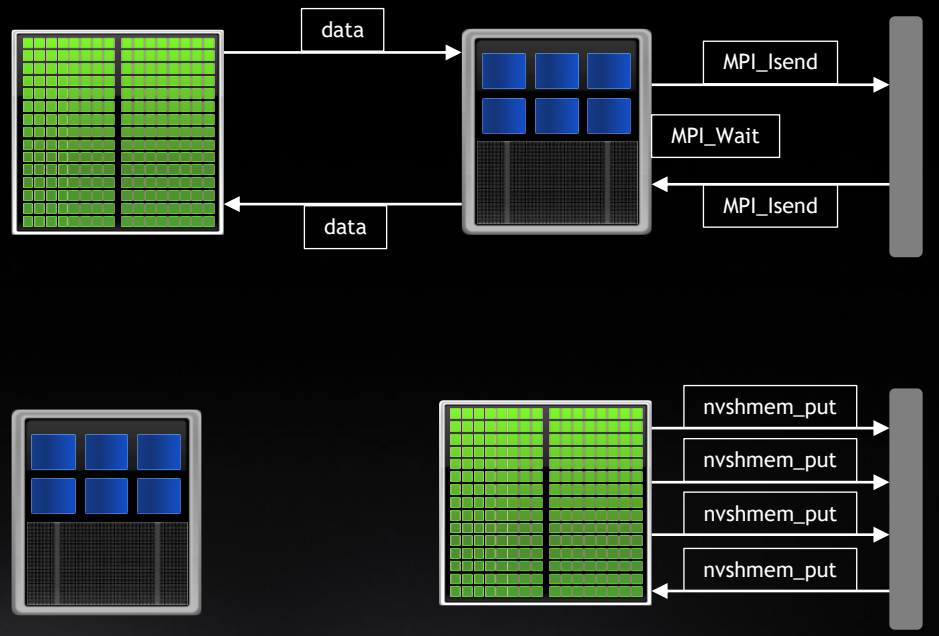
INTRODUCING NVSHMEM

GPU Optimized SHMEM

- Initiate from CPU or GPU
- Initiate from within CUDA kernel
- Issue onto a CUDA stream
- Interoperable with MPI & OpenSHMEM

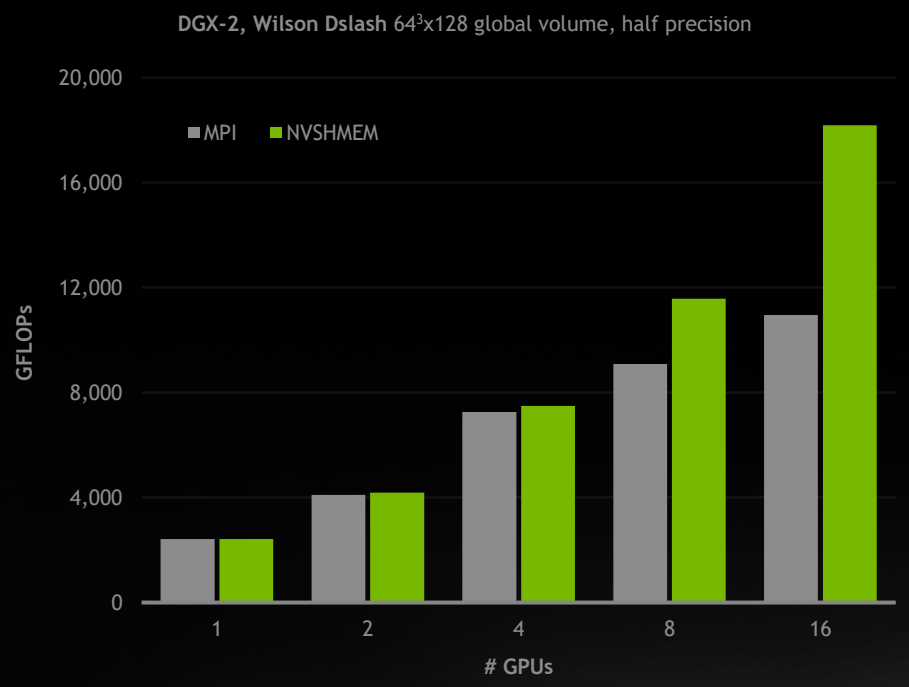
Pre-release Impact

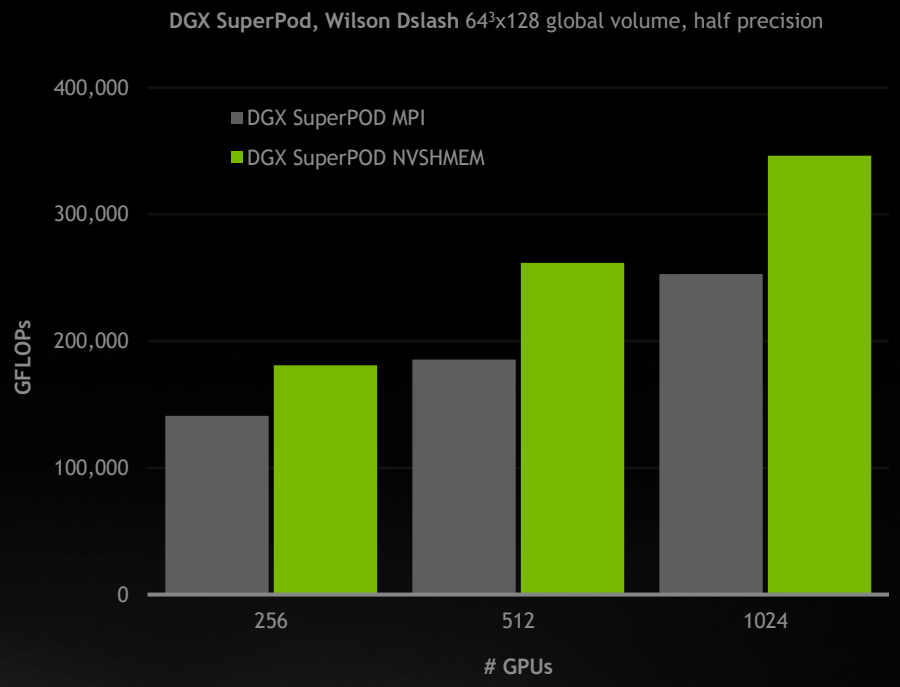
- LBANN, Kokkos/CGSolve, QUDA



INTRODUCING NVSHMEM

Impact in HPC Applications





QUDA: Quantum Chromodynamics on CUDA

- Up to 1.7X Single Node Speedup
- Up to 1.4X Multi Node Speedup

NCCL

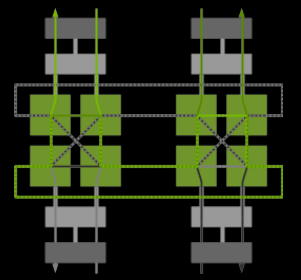
GPU-Optimized Collectives

- Multi-GPU and Multi-Node Collectives Optimized for NVIDIA GPUs
- Automatic Topology Detection
- Easy to integrate | MPI Compatible
- Minimize latency | Maximize bandwidth

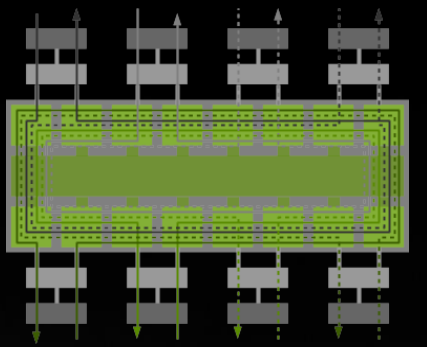
NCCL 2.8

- Scalable algorithms, tested on more than 24,000 GPUs
- Improved AllReduce on NVIDIA Ampere GPUs
- Supports send/receive operations to cover all communication needs
- Improved All2All performance and usability

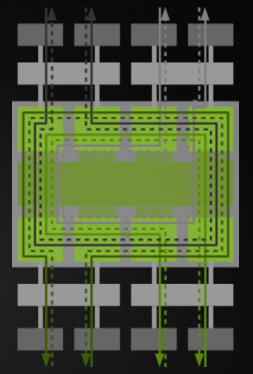
DGX-1
48 GB/s



DGX-2
85 GB/s



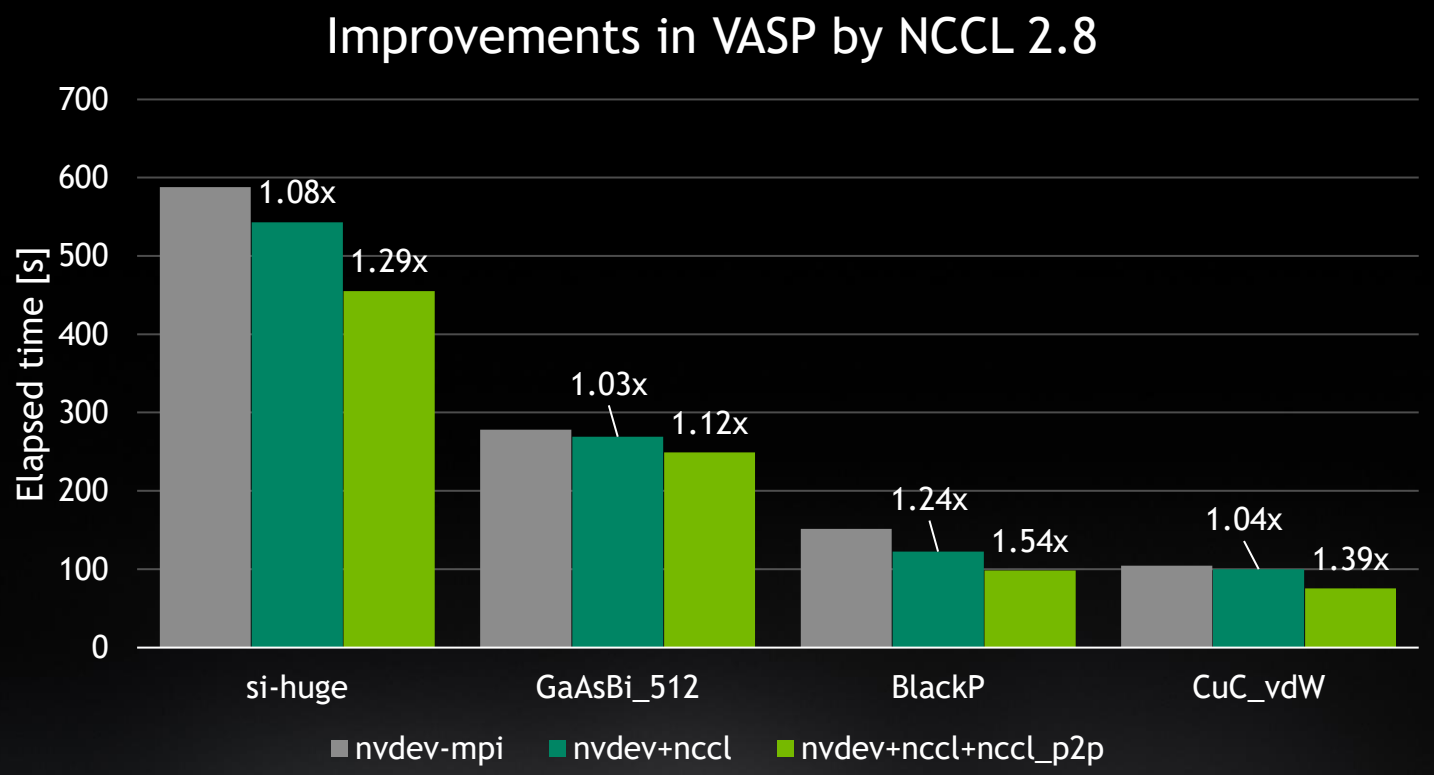
DGX A100
192 GB/s



AllReduce Perf

AT-SCALE PERFORMANCE WITH P2P COLLECTIVES

VASP molecular dynamics is a top-5 HPC app; in OpenACC, via HPC SDK



Courtesy of Alexey Romanenko and Stefan Maintz; 1 DGX1 with 8 V100s, 2S E5-2698 @ 2.2 GHz
Based on an internal development version of VASP, P2P targeted for release after VASP 6.1.2

MULTI GPU WITH THE NVIDIA HPC SDK

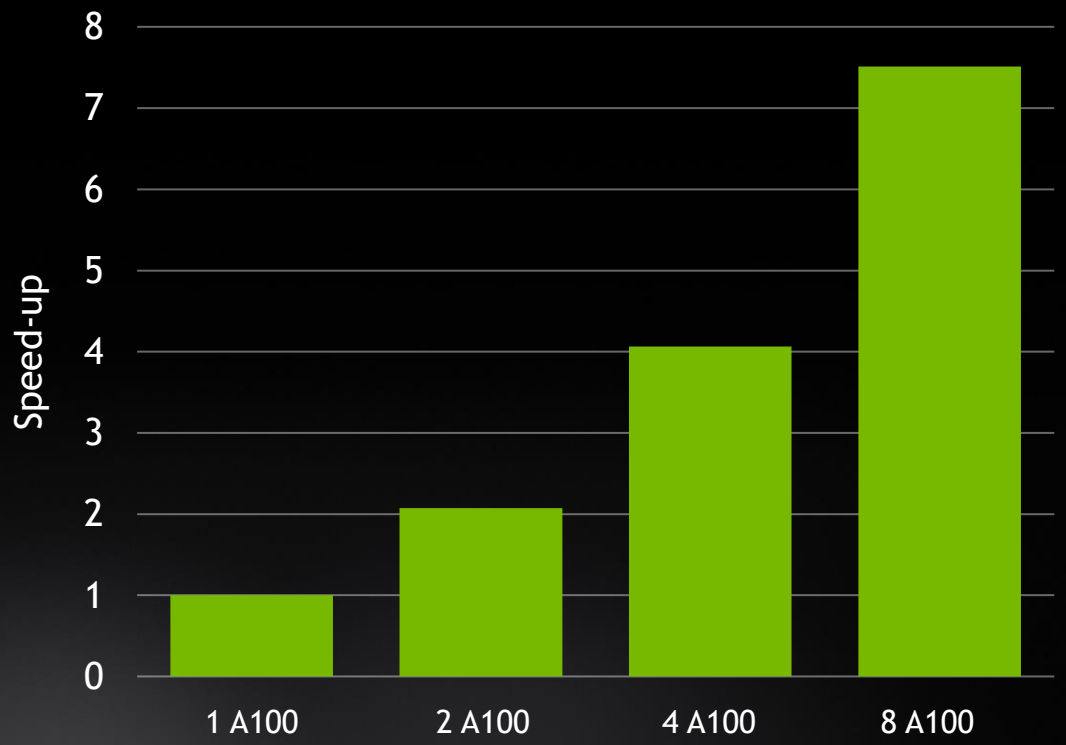
Cloverleaf Hydrodynamics Mini-App

Full Integration provided by HPC SDK

- Fortran + OpenACC + Open MPI

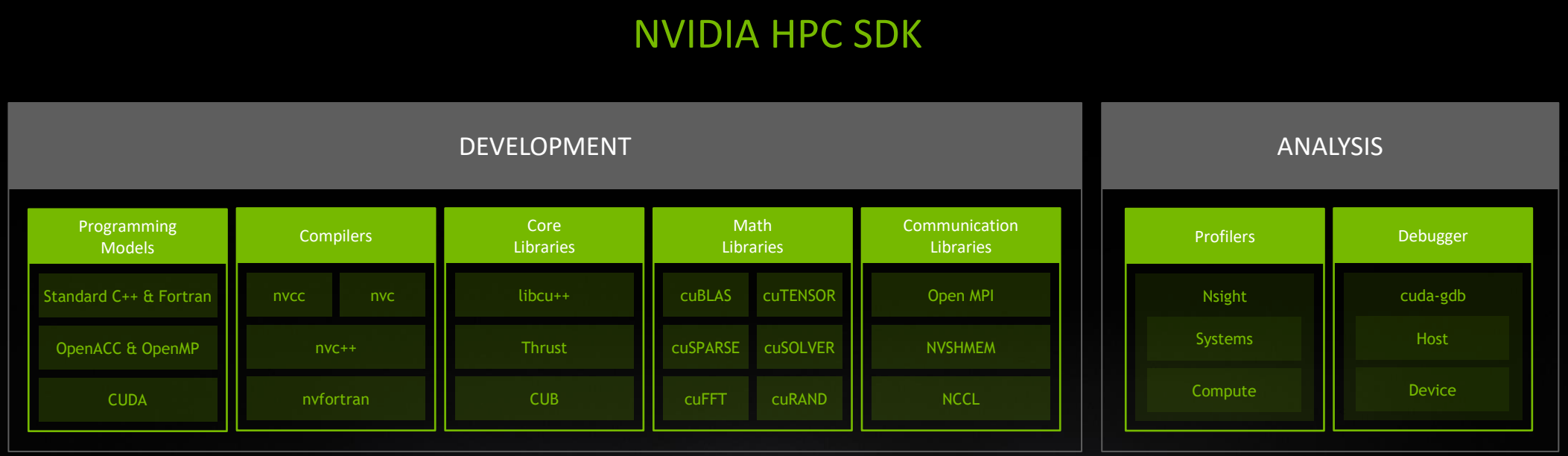
Strong Scaling - Cloverleaf BM128

- Perfect scaling to 4 A100 GPUs
- 7.5X speed-up on 8 A100 GPUs



AVAILABLE NOW: THE NVIDIA HPC SDK

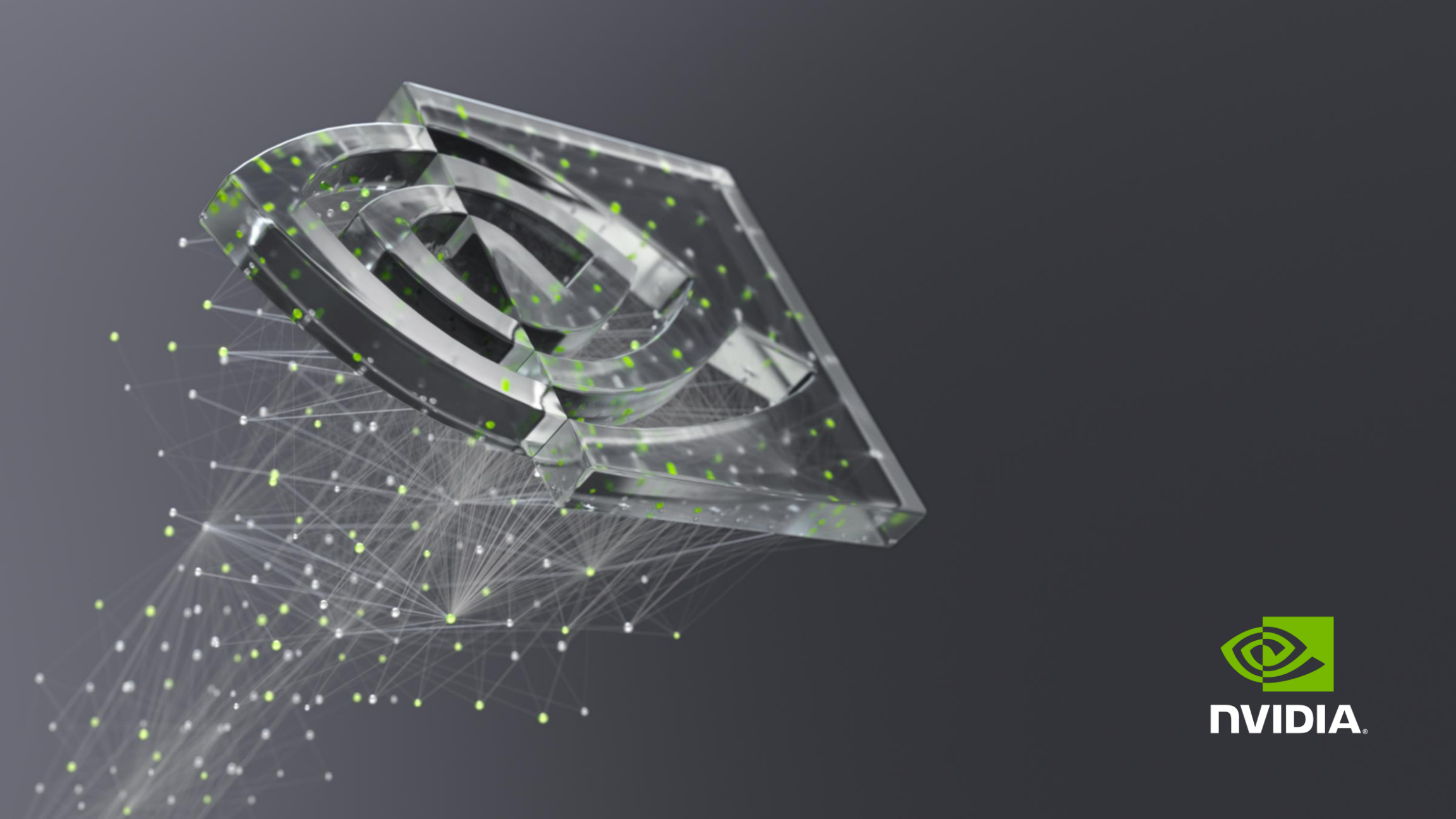
Available at developer.nvidia.com/hpc-sdk, on NGC, and in the Cloud



Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

7-8 Releases Per Year | Freely Available


NVIDIA