5 Ways to Accelerate with GPUs

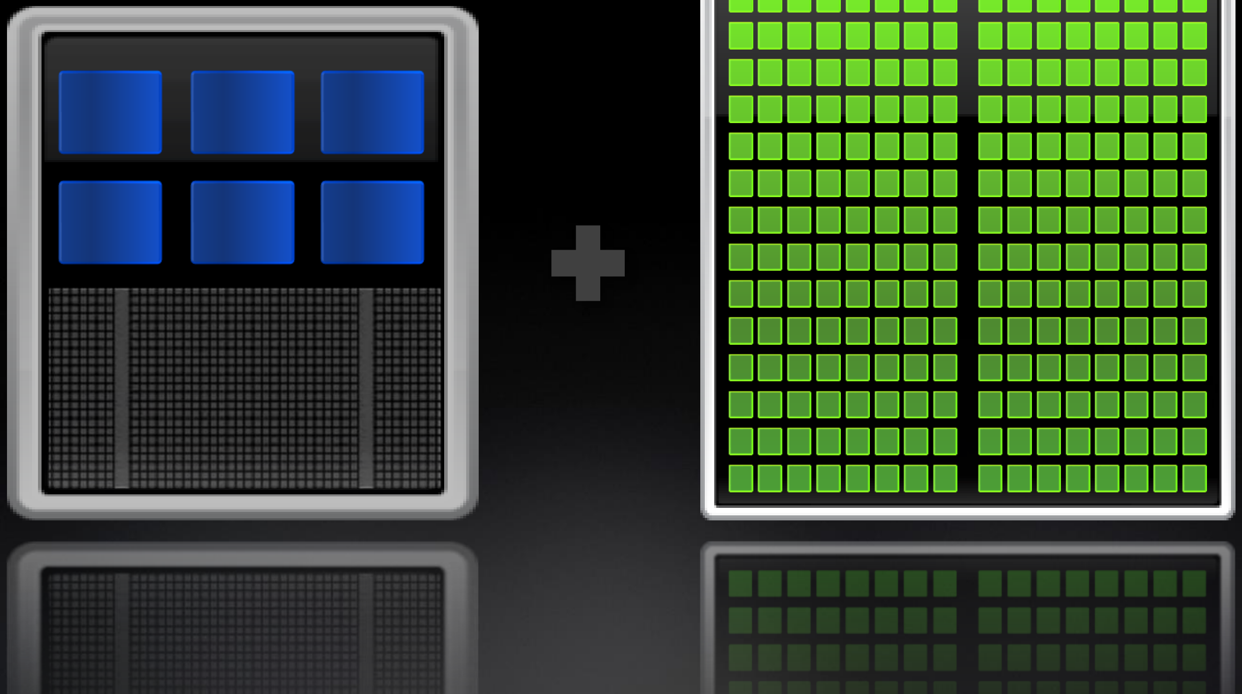
Brad Palmer, Senior Solutions Architect

First, some GPU basics

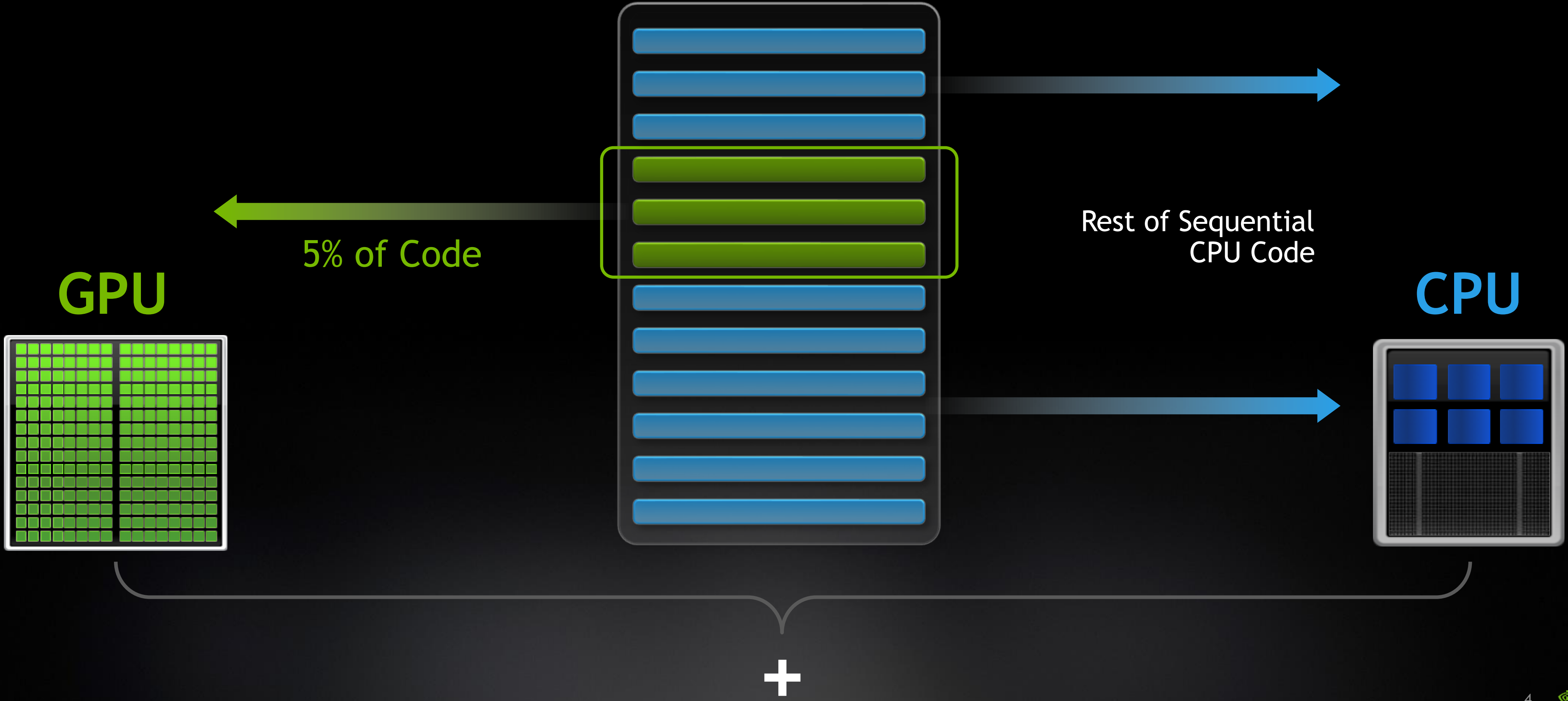
ACCELERATED COMPUTING

CPU
Optimized for
Serial Tasks

GPU Accelerator
Optimized for
Parallel Tasks

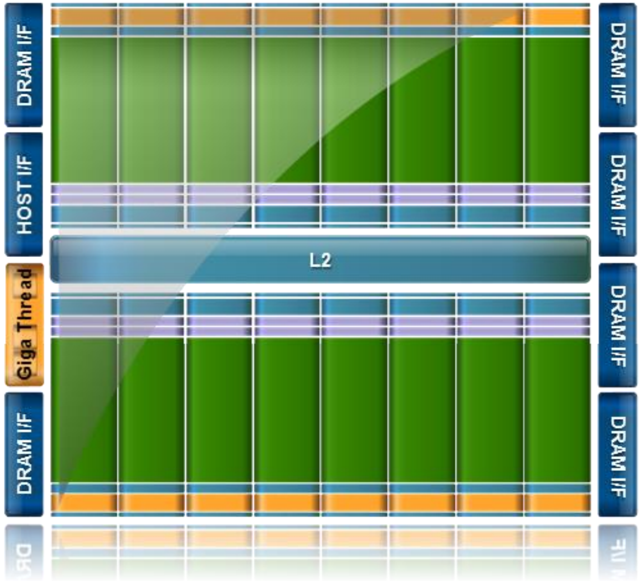


HOW GPU ACCELERATION WORKS



GPU ARCHITECTURE

Two Main Components



Global memory

Analogous to RAM in a CPU server

Accessible by both GPU and CPU

H100 has ***80 GB***

Streaming Multiprocessors (SM)

Perform the actual computation

Each SM has its own: Control units, registers, execution pipelines, caches

H100 has ***114 SMs***

GPU ARCHITECTURE

Streaming Multiprocessor (SM)

Many CUDA Cores per SM

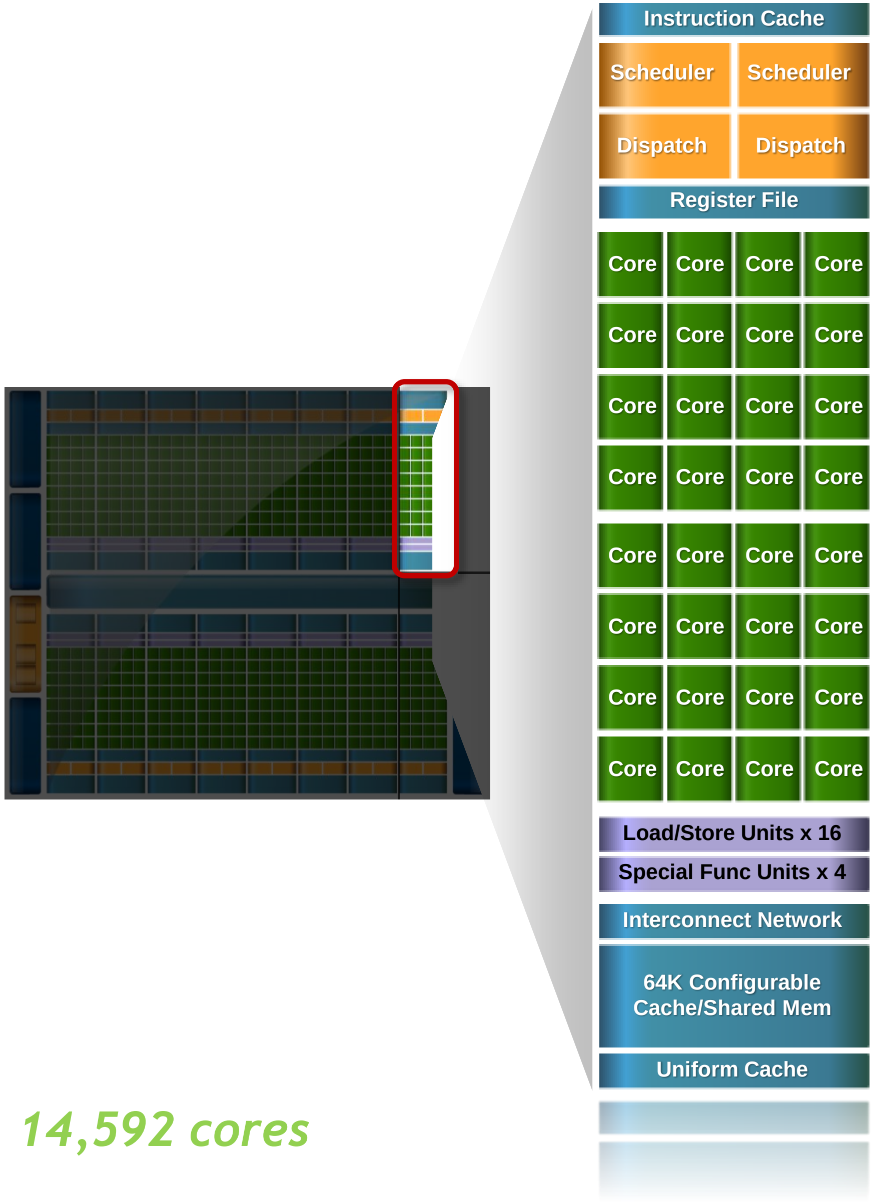
- Architecture dependent
- *H100 SM has 128 cores*

Special-function units

- cos/sin/tan, etc.

Shared mem + L1 cache

Thousands of 32-bit registers



Instruction Cache

Scheduler | Scheduler

Dispatch | Dispatch

Register File

Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core
Core Core Core Core

Load/Store Units x 16

Special Func Units x 4

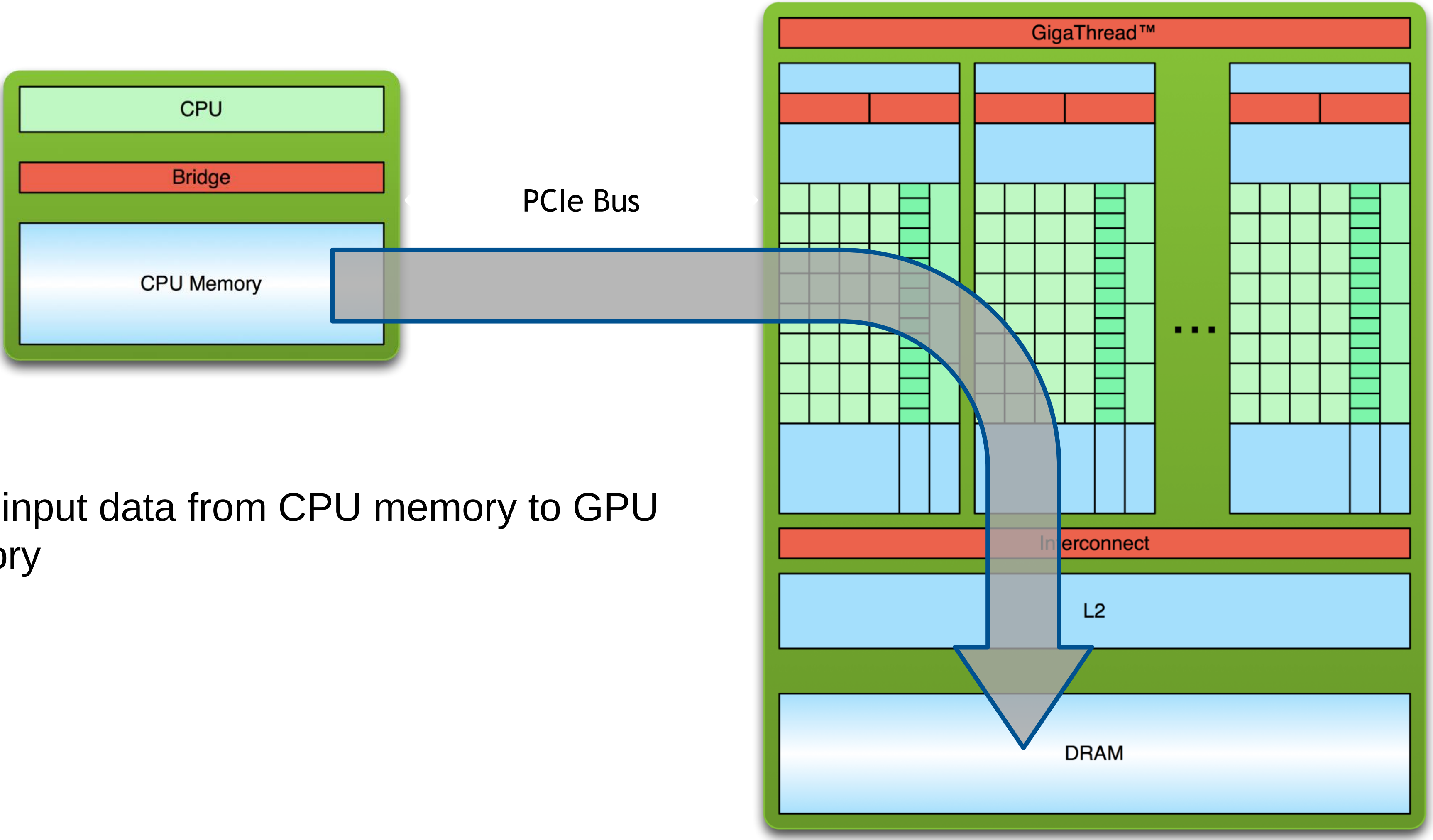
Interconnect Network

64K Configurable Cache/Shared Mem

Uniform Cache

H100 PCIe has a total of 14,592 cores

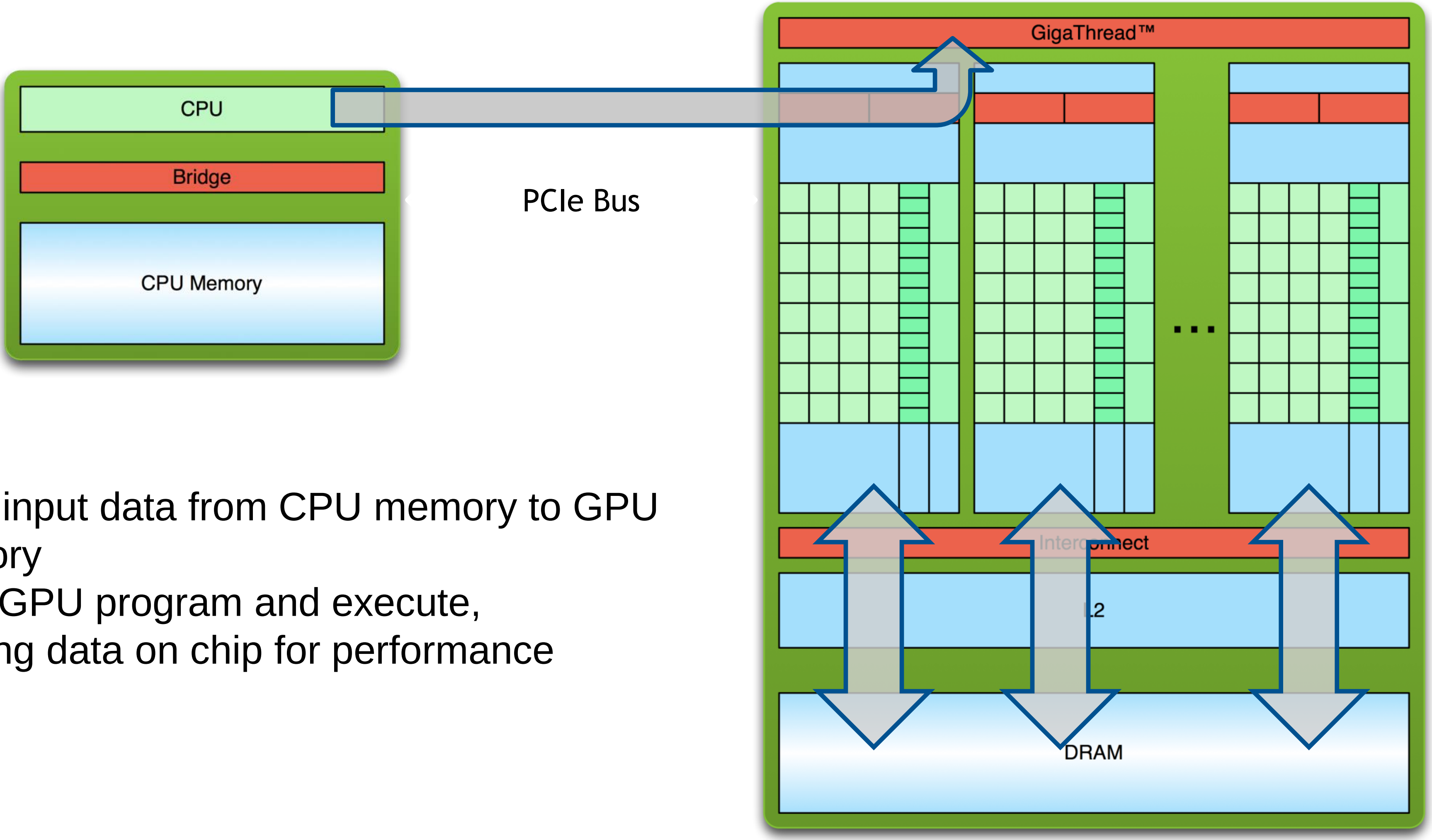
PROCESSING FLOW



1. Copy input data from CPU memory to GPU memory

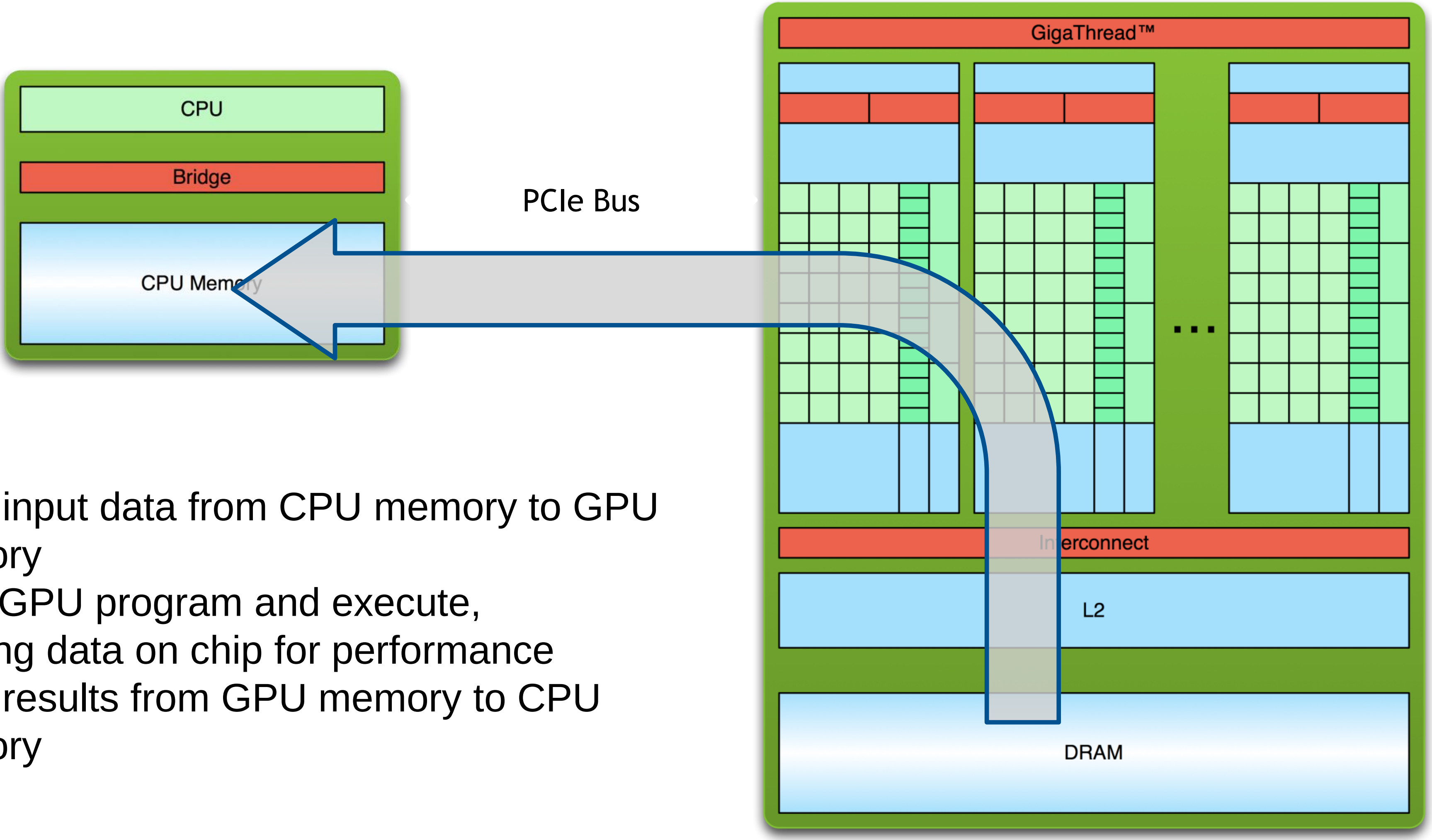
A100 memory bandwidth is 25x PCIe gen4

PROCESSING FLOW



1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance

PROCESSING FLOW



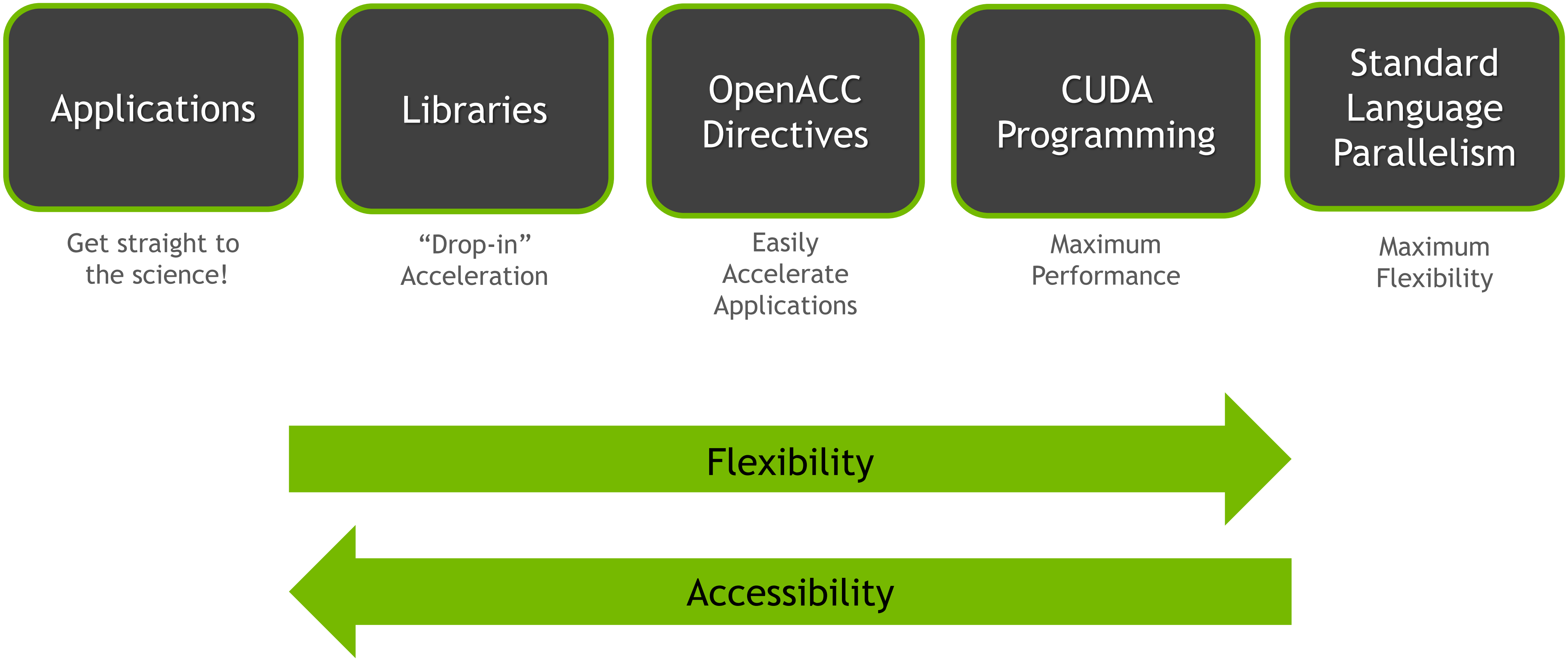
1. Copy input data from CPU memory to GPU memory
2. Load GPU program and execute, caching data on chip for performance
3. Copy results from GPU memory to CPU memory

NVIDIA CUDA

- A parallel computing platform and application programming interface (API) model created by NVIDIA
- Allows software developers and software engineers to use a CUDA-enabled GPUs for general purpose processing
- Backwards compatible
- The name CUDA was originally an acronym for Compute Unified Device Architecture

The Five Ways to Accelerate with GPUs

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

"Drop-in" Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

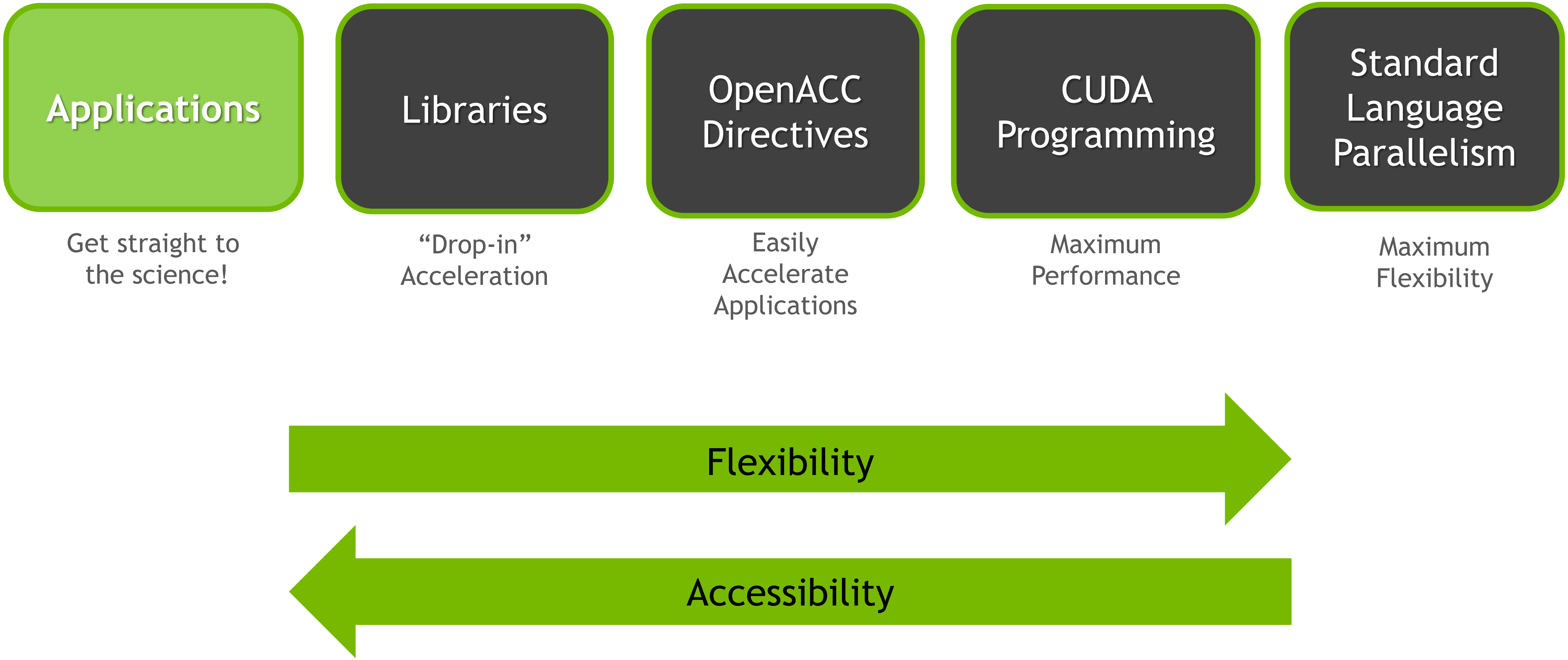
Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

"Drop-in" Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

THOUSANDS OF GPU-ACCELERATED APPLICATIONS

Transforming Every Industry

ARTIFICIAL INTELLIGENCE

- PyTorch
- MXNet
- TensorFlow

•••

CLIMATE & WEATHER

- Cosmos
- Gales
- WRF

•••

COMPUTATIONAL FINANCE

- O-Quant Options Pricing
- MUREX
- MISYS

•••

DATA SCIENCE & ANALYTICS

- Anaconda
- H20
- OmniSci

•••

FEDERAL DEFENSE & OTHER

- ArcGIS Pro
- EVNI
- SocetGXP
- Cyllance
- FaceControl

•••

LIFE SCIENCES

- Amber
- LAMMPS
- GROMACS
- NAMD
- Relion
- VASP

•••

MANUFACTURING, CAD, & CAE

- Ansys Fluent
- Abaqus SIMULIA
- AutoCAD
- CST Studio Suite

•••

MEDIA & ENTERTAINMENT

- DaVinci Resolve
- Premiere Pro CC
- Redshift Renderer

•••

MEDICAL IMAGING

- aidoc
- PowerGrid
- RadiAnt

•••

OIL & GAS

- Echelon
- RTM
- SPECFEM3D

•••

RETAIL

- Everseen
- Deep North
- Third Eye Labs
- AWM
- Malong
- Clarifai
- Antuit

•••

SUPERCOMPUTING & HER

- Chroma
- GTC
- MILC
- QUDA
- XGC

•••

For a comprehensive list of all apps, please refer to GPU application catalog: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/gpu-applications-catalog.pdf

Sample GPU Accelerated Applications

See https://www.nvidia.com/en-us/gpu-accelerated-applications/

- Amber
- GROMACS
- LAMMPS
- NAMD
- Relion
- Chroma
- GTC
- MILC
- SPECFEM3D
- FUN3D

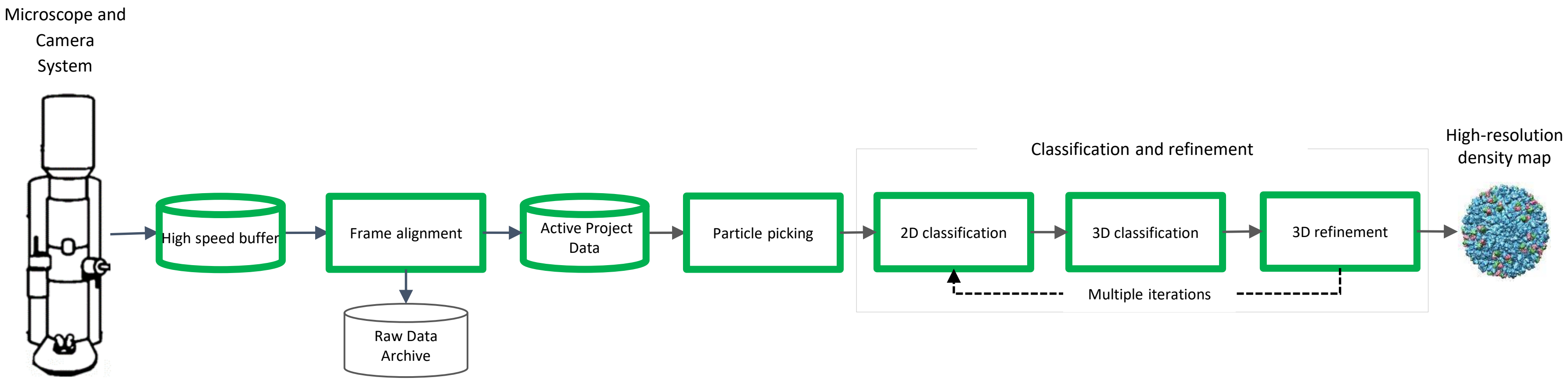
Standard Benchmark speedup on single A100 vs dual CPU

https://developer.nvidia.com/hpc-application-performance

Amber	13x – 39x
GROMACS	6x – 9x
LAMMPS	5x – 18x
NAMD	6x – 8x
Relion	4x – 5x
Chroma	32x
GTC	14x
MILC	32x
SPECFEM3D	29x
FUN3D	13x

Single particle electron microscopy

SINGLE PARTICLE ANALYSIS

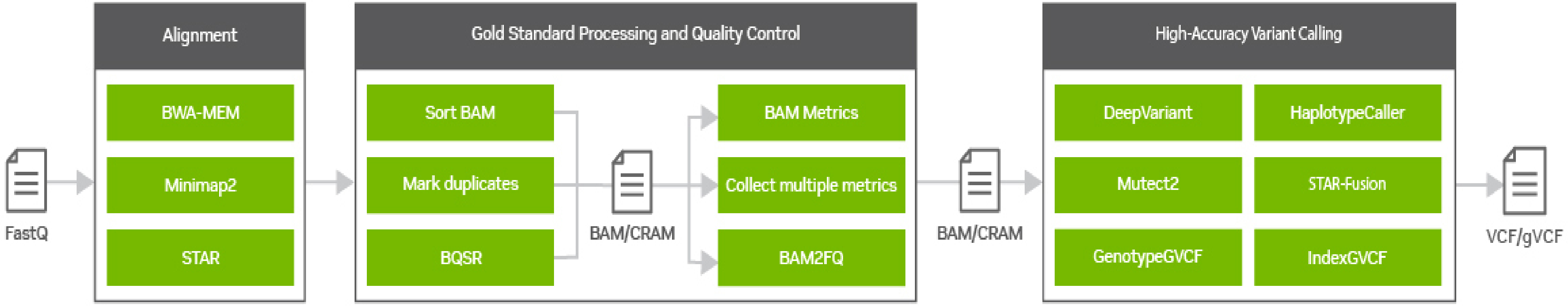


GPU-accelerated apps (above)

- AMIRA
- BioEM
- cryoSPARC
- cyYOLO
- Dynamo
- EMAN2
- emClarity
- GCTF
- IMOD
- MotionCor2
- RELION
- Tomviz
- Topaz
- VMD
- Warp

NVIDIA Parabricks for Alignment & Variant Calling

Speed, Scale, Accuracy



Universal Analysis

Industry-standard tools for all major sequencers, ported to GPU

Up to 100x Acceleration

Up to 100x faster for WGS compared to CPU-only

Up to 50% Lower Cost

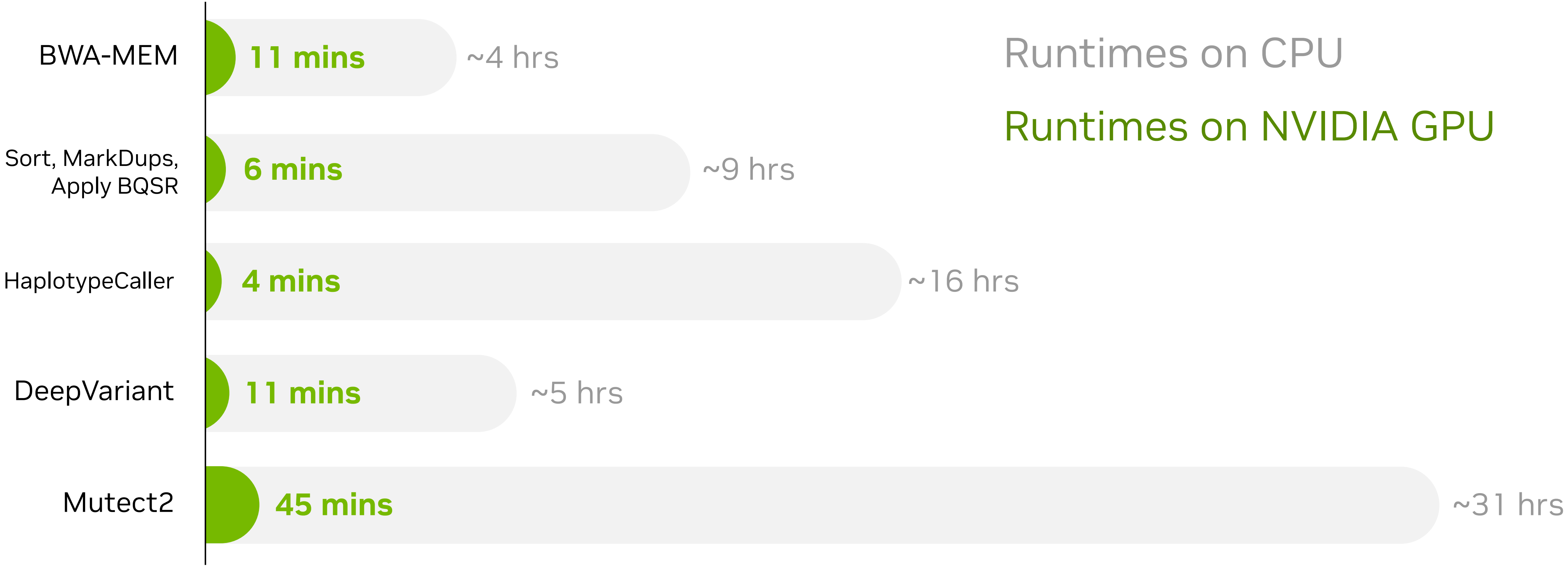
Up to 50% lower compute cost for WGS compared to CPU-only

Higher Accuracy with AI

The power of deep learning for customized high accuracy analysis

Up to 80x Acceleration

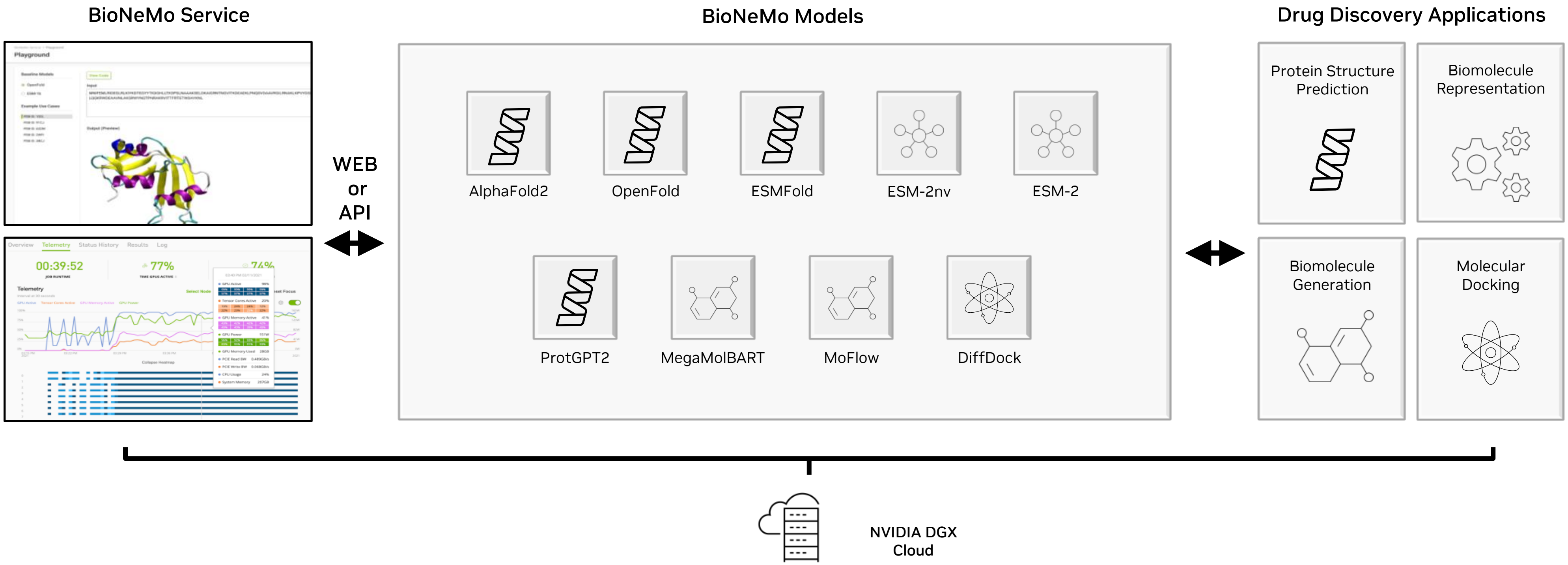
Industry-standard results, faster



NVIDIA Parabricks v4.0 Benchmarks
Dataset: HG002 30x WGS, except Mutect2 on SEQC2 50x WGS
CPU: m5.24xlarge; GPU: 8xA100, except DeepVariant & Mutect2 on 8xV100

NVIDIA BioNeMo

Cloud Service for Customizing and Running Generative AI in Drug Discovery



BioNeMo Service

WEB or API

BioNeMo Models

AlphaFold2

OpenFold

ESMFold

ESM-2nv

ESM-2

ProtGPT2

MegaMolBART

MoFlow

DiffDock

Drug Discovery Applications

Protein Structure Prediction

Biomolecule Representation

Biomolecule Generation

Molecular Docking

NVIDIA DGX Cloud

Customizable SOTA Generative AI

Innovate faster and more competitively using proprietary data sets to train and fine-tune drug discovery workflows

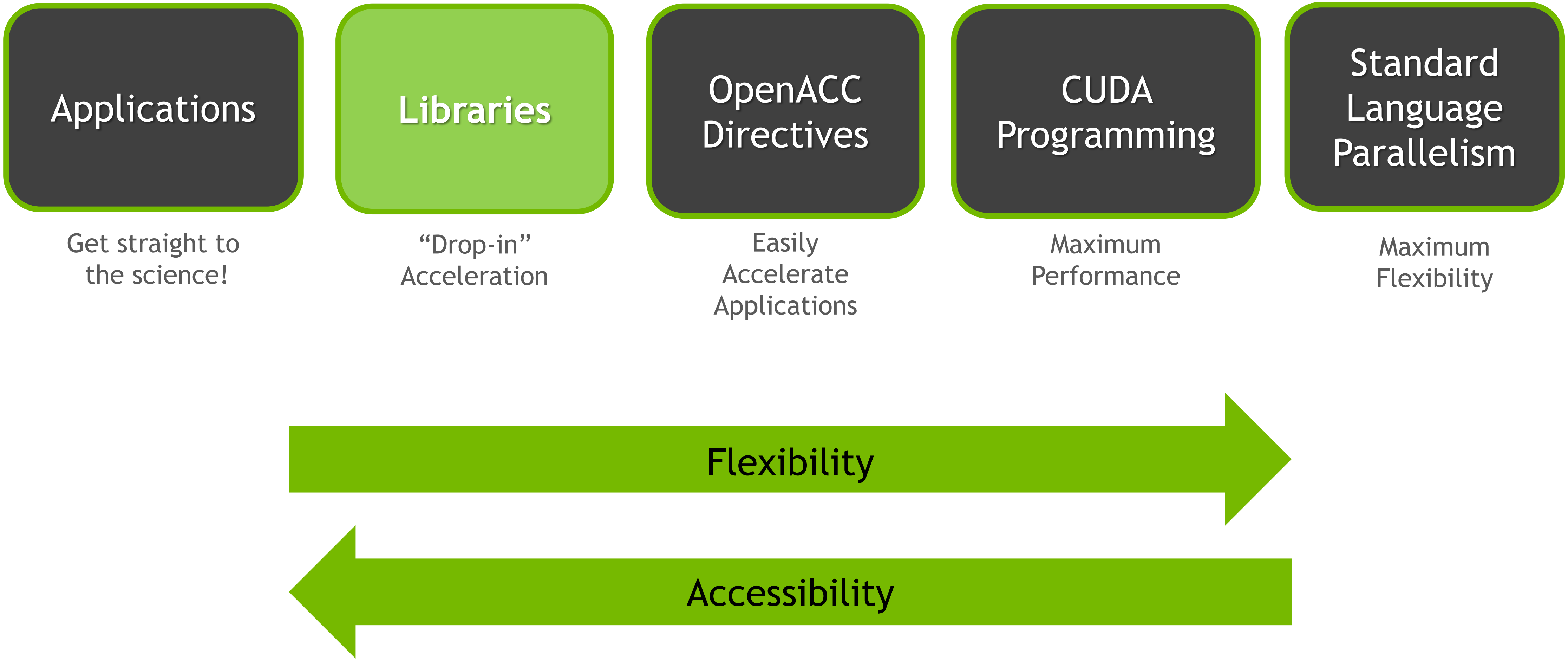
Easy and Instant Access to Optimized AI

Eliminate the need for building IT infrastructure, managing open-source software, optimize for throughput

Seamless and Scalable AI Service

Ultimate flexibility in experimenting and building enterprise grade generative AI workflows with GUI & API Endpoints on scalable managed infrastructure

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

"Drop-in" Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

LIBRARIES: EASY, HIGH-QUALITY ACCELERATION

EASE OF USE Using libraries enables GPU acceleration without in-depth knowledge of GPU programming

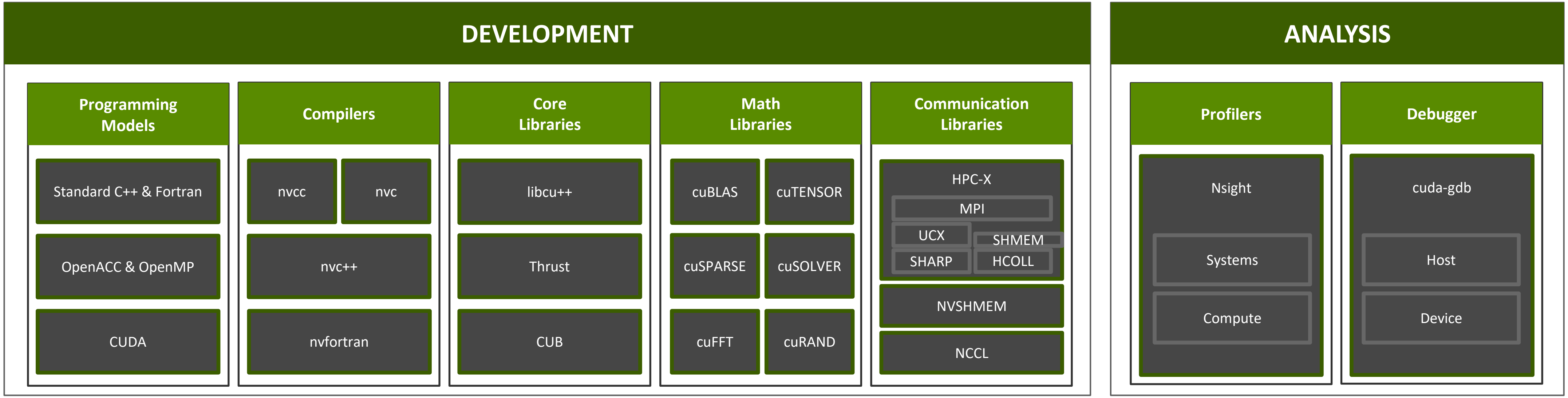
“DROP-IN” Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes

QUALITY Libraries offer high-quality implementations of functions encountered in a broad range of applications

PERFORMANCE NVIDIA libraries are tuned by experts

NVIDIA HPC SDK

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



DEVELOPMENT

Programming Models
- Standard C++ & Fortran
- OpenACC & OpenMP
- CUDA

Compilers
- nvcc
- nvc
- nvc++
- nvfortran

Core Libraries
- libcu++
- Thrust
- CUB

Math Libraries
- cuBLAS
- cuTENSOR
- cuSPARSE
- cuSOLVER
- cuFFT
- cuRAND

Communication Libraries
- HPC-X
  - MPI
  - UCX
  - SHMEM
  - SHARP
  - HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
  - Systems
  - Compute

Debugger
- cuda-gdb
  - Host
  - Device

Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
7-8 Releases Per Year | Freely Available

3 STEPS TO CUDA-ACCELERATED APPLICATION

Step 1: Substitute library calls with equivalent CUDA library calls

`saxpy ( … )` ► `cublasSaxpy ( … )`

Step 2: Manage data locality

- with CUDA: `cudaMalloc(), cudaMemcpy(), etc.`
- with CUBLAS: `cublasAlloc(), cublasSetVector(), etc.`

Step 3: Rebuild and link the CUDA-accelerated library

```
gcc myobj.o –l cublas
```

SAXPY is “Single-Precision A times X Plus Y”

GPU Accelerated Libraries (some examples)

https://developer.nvidia.com/how-to-cuda-libraries

CUBLAS – an implementation of BLAS (Basic Linear Algebra Subprograms).

CUFFT – a Fast Fourier Transform library with support for the FFTW API.

CURAND – provides facilities that focus on the simple and efficient generation of high-quality pseudorandom and quasi-random numbers.

CUSPARSE – contains a set of basic linear algebra subroutines used for handling sparse matrices.

cuSOLVER – GPU-accelerated dense and sparse direct solvers (LAPACK-like features)

CUDA Math Library – GPU-accelerated standard mathematical function library (Available to any CUDA C or CUDA C++ application simply by adding “#include math.h” in your source code)

Thrust – GPU-accelerated library of C++ parallel algorithms and data structures

nvJPEG – High performance GPU-accelerated library for JPEG decoding

ArrayFire – open source library for matrix, signal, and image processing

MAGMA – linear algebra routines for heterogeneous architectures

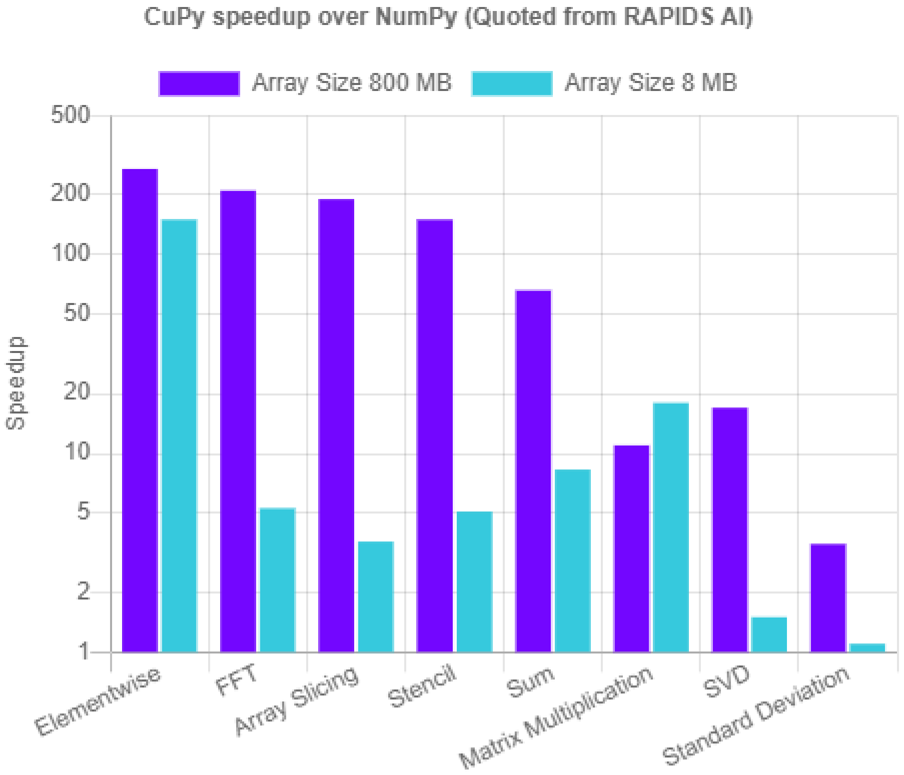
CHOLMOD – functions for sparse direct solvers

https://github.com/nvidia/cudalibrarysamples

CuPy

https://cupy.dev/

- Open-source array library for GPU-accelerated computing
- Interface is highly compatible with NumPy and SciPy
- Can be used as a drop-in replacement in most cases
- Just replace `numpy` and `scipy` with `cupy` and `cupyx.scipy`
- Speeds up some operations more than 100X



RAPIDS

https://rapids.ai/

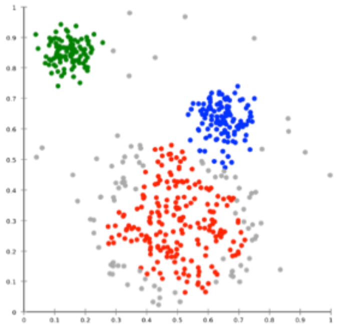
RAPIDS: a suite of open source software libraries and APIs gives you the ability to execute end-to-end data science and analytics pipelines entirely on GPUs. Licensed under Apache 2.0

Popular Libraries:

- **cuDF** – a pandas-like dataframe manipulation library
- **cuML** – GPU versions of algorithms in scikit-learn
- **cuSignal** – signal processing library based on SciPy Signal
- **cuGraph** – Network-X-like accelerated graph analytics library
- **cuSpatial** – GPU-accelerated GIS and spatiotemporal algorithms

ALGORITHMS

GPU-accelerated Scikit-Learn



Cross Validation

Hyper-parameter Tuning

More to come!

Classification / Regression

Decision Trees / Random Forests
Linear/Lasso/Ridge/ElasticNet Regression
Logistic Regression
K-Nearest Neighbors
Support Vector Machine Classification and Regression
Naive Bayes

Inference

Random Forest / GBDT Inference (FIL)

Preprocessing

Text vectorization (TF-IDF / Count)
Target Encoding
Cross-validation / splitting

Clustering Decomposition & Dimensionality Reduction

K-Means
DBSCAN
Spectral Clustering
Principal Components
Singular Value Decomposition
UMAP
Spectral Embedding
T-SNE

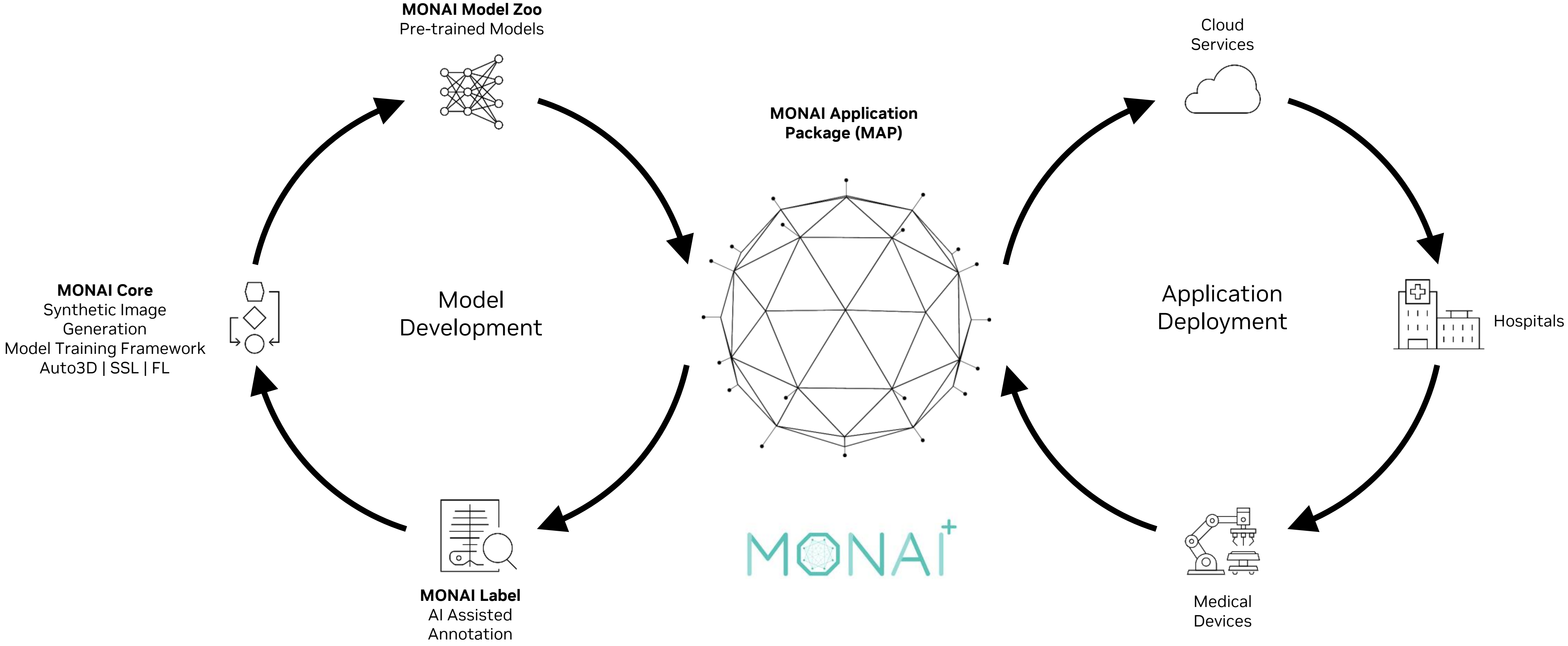
Time Series

Holt-Winters
Seasonal ARIMA / Auto ARIMA

https://github.com/rapidsai/cuml#supported-algorithms

MONAI

A comprehensive Medical AI framework built by experts, accelerated by NVIDIA.



MONAI Model Zoo
Pre-trained Models

MONAI Core
Synthetic Image Generation
Model Training Framework
Auto3D | SSL | FL

Model Development

MONAI Label
AI Assisted Annotation

MONAI Application Package (MAP)

Cloud Services

Application Deployment

Hospitals

Medical Devices

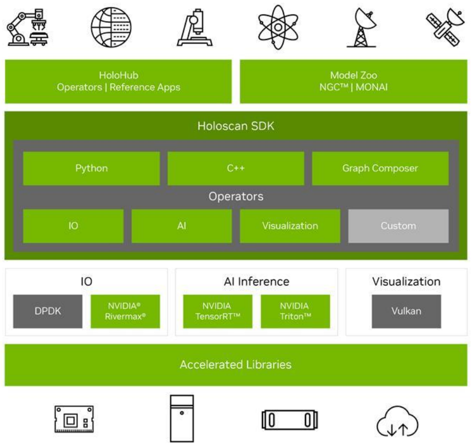
https://monai.io/

NVIDIA Holoscan

The sensor processing platform for building real-time AI

Clara Holoscan SDK

Build



HoloHub
Operators | Reference Apps

Model Zoo
NGC™ | MONAI

Holoscan SDK

Python | C++ | Graph Composer

Operators

IO | AI | Visualization | Custom

IO: DPDK | NVIDIA® Rivermax®

AI Inference: NVIDIA TensorRT™ | NVIDIA Triton™

Visualization: Vulkan

Accelerated Libraries

Developer Kits

Validate

NVIDIA IGX Orin DevKit (EA)
Orin, RTX A6000, ConnectX-7
Available Now

NVIDIA IGX

Deploy

Developer Productivity

Optimized for High-performance

Sensor Partner Ecosystem

C++, Python

.

Secure

Secure by design

Remote provisioning and management

.

Medical Grade

Ready for certification (IEC 60601, 62304).

.

Production Ready

Customizable white-label platform

Long term support

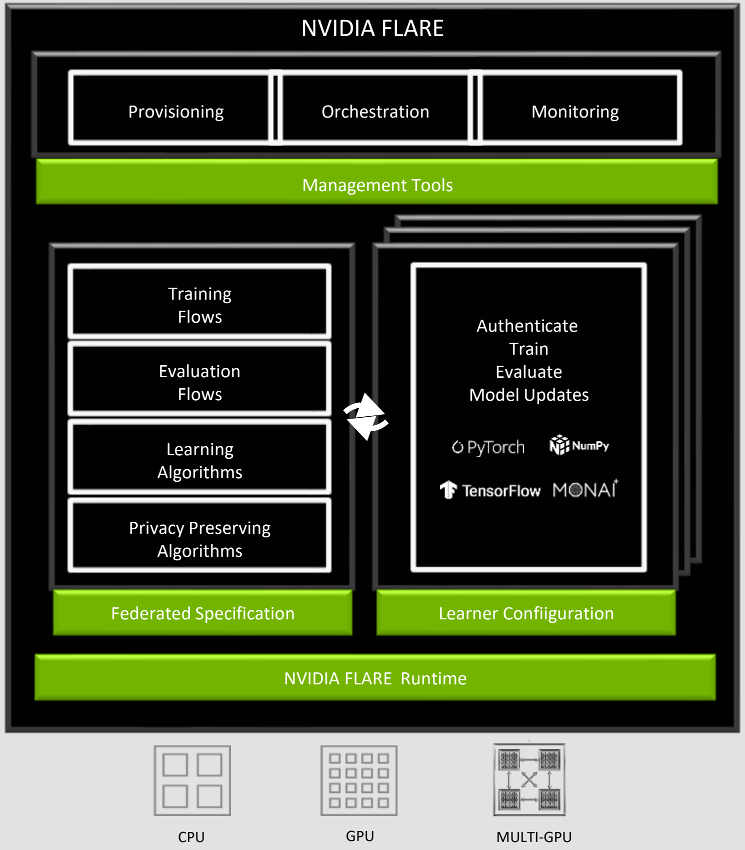
.

NVIDIA FLARE

Open-Source SDK for Federated Learning

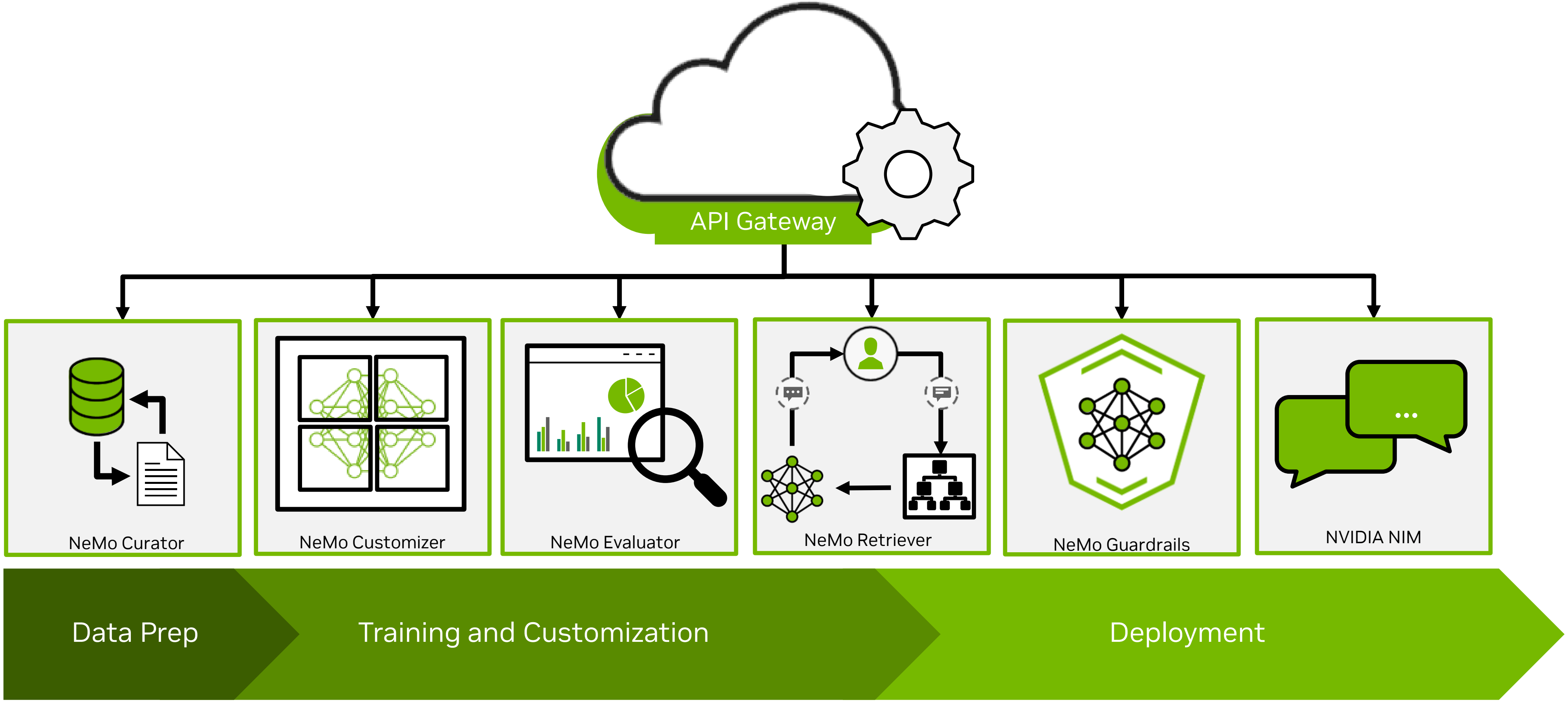
- Apache License 2.0 to catalyze FL research & development
- Enables Distributed, Multi-Party Collaborative Learning
- Production Scalability with high availability and multi-task execution
- Adapt existing ML/DL workflows to a Federated paradigm
- Privacy Preserving Algorithms
    - Homomorphic Encryption & Differential Privacy
- Secure Provisioning, Orchestration & Monitoring
- Programmable APIs for Extensibility

Available on Github: https://github.com/nvidia/nvFlare



Building Generative AI Applications for the Enterprise

Build, customize, and deploy generative AI models with NVIDIA NeMo.



API Gateway

NeMo Curator

NeMo Customizer

NeMo Evaluator

NeMo Retriever

NeMo Guardrails

NVIDIA NIM

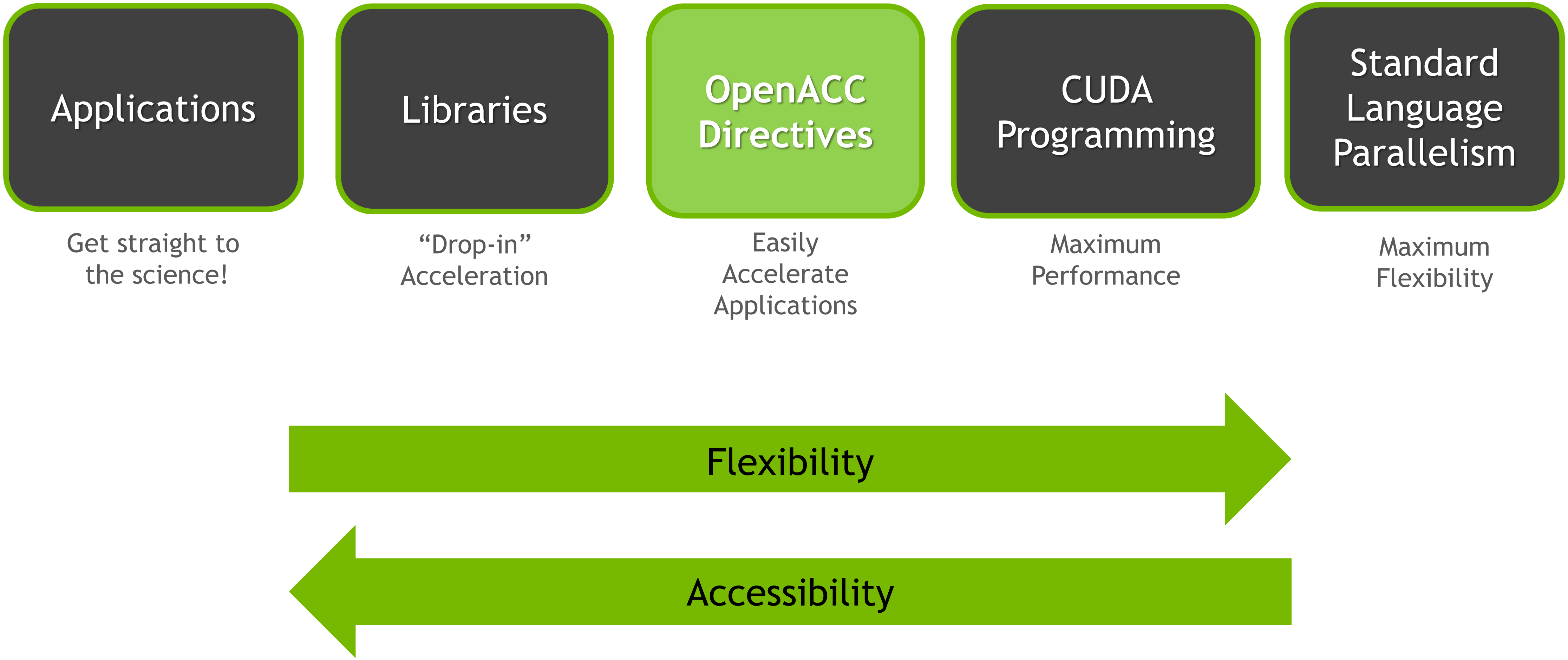
Data Prep

Training and Customization

Deployment

Microsoft Azure · aws · Google Cloud · ORACLE · NVIDIA DGX Cloud · DELL Technologies · Hewlett Packard Enterprise · Lenovo · SUPERMICRO · NVIDIA

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

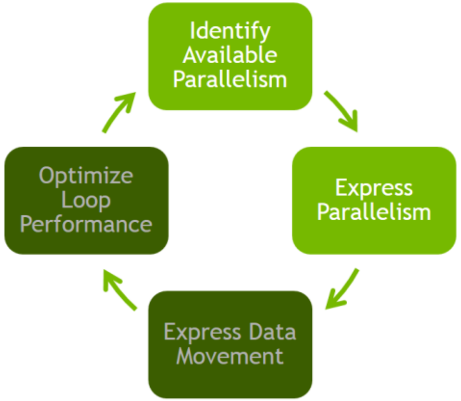
Flexibility

Accessibility

OpenACC Directives

https://www.openacc.org/

OpenACC is a user-driven directive-based performance-portable parallel programming model. It is designed for scientists and engineers interested in porting their codes to a wide-variety of heterogeneous HPC hardware platforms and architectures with significantly less programming effort than required with a low-level model.



C

```
#pragma acc directive [clause [,] clause] …]
```

Often followed by a structured code block

Fortran

```
!$acc directive [clause [,] clause] …]
```

Often paired with a matching end directive surrounding a structured code block

```
!$acc end directive
```

- Simple Compiler hints
- Compiler Parallelizes code
- Works on many-core GPUs & multicore CPUs

https://www.gpuhackathons.org/

A VERY SIMPLE EXERCISE: SAXPY

SAXPY in C

```
void saxpy(int n,
           float a,
           float *x,
           float *restrict y)
{
#pragma acc kernels
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}


...
// Perform SAXPY on 1M elements
saxpy(1<<20, 2.0, x, y);
...
```

SAXPY in Fortran

```
subroutine saxpy(n, a, x, y)
  real :: x(:), y(:), a
  integer :: n, i
$!acc kernels
  do i=1,n
    y(i) = a*x(i)+y(i)
  enddo
$!acc end kernels
end subroutine saxpy


...
$ Perform SAXPY on 1M elements
call saxpy(2**20, 2.0, x_d, y_d)
...
```

TOP HPC APPS ADOPTING OPENACC

OpenACC - Performance Portability And Ease of Programming

ANSYS Fluent Gaussian
VASP

3 of Top 10 Apps

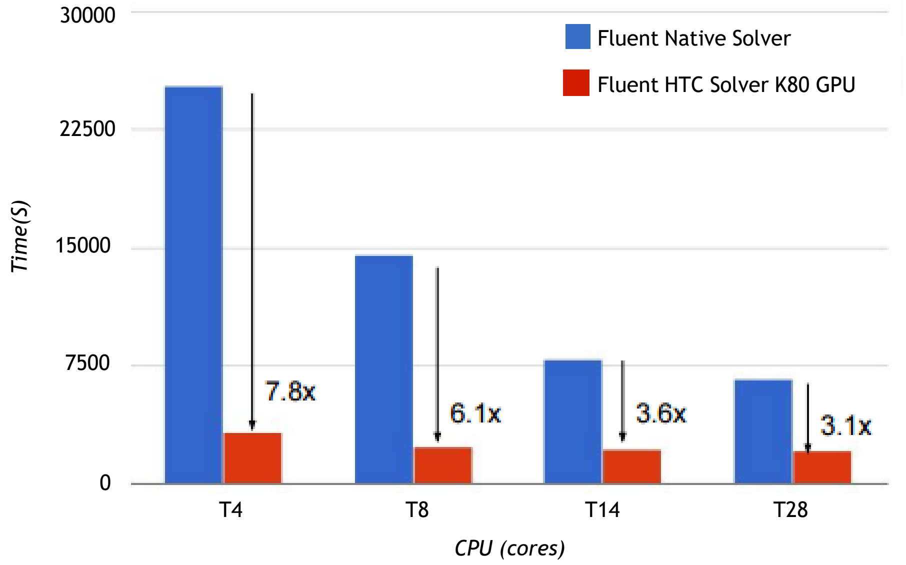
GTC
XGC
ACME
FLASH
LSDalton

5 ORNL CAAR Codes

COSMO
ELEPHANT
RAMSES
ICON
ORB5

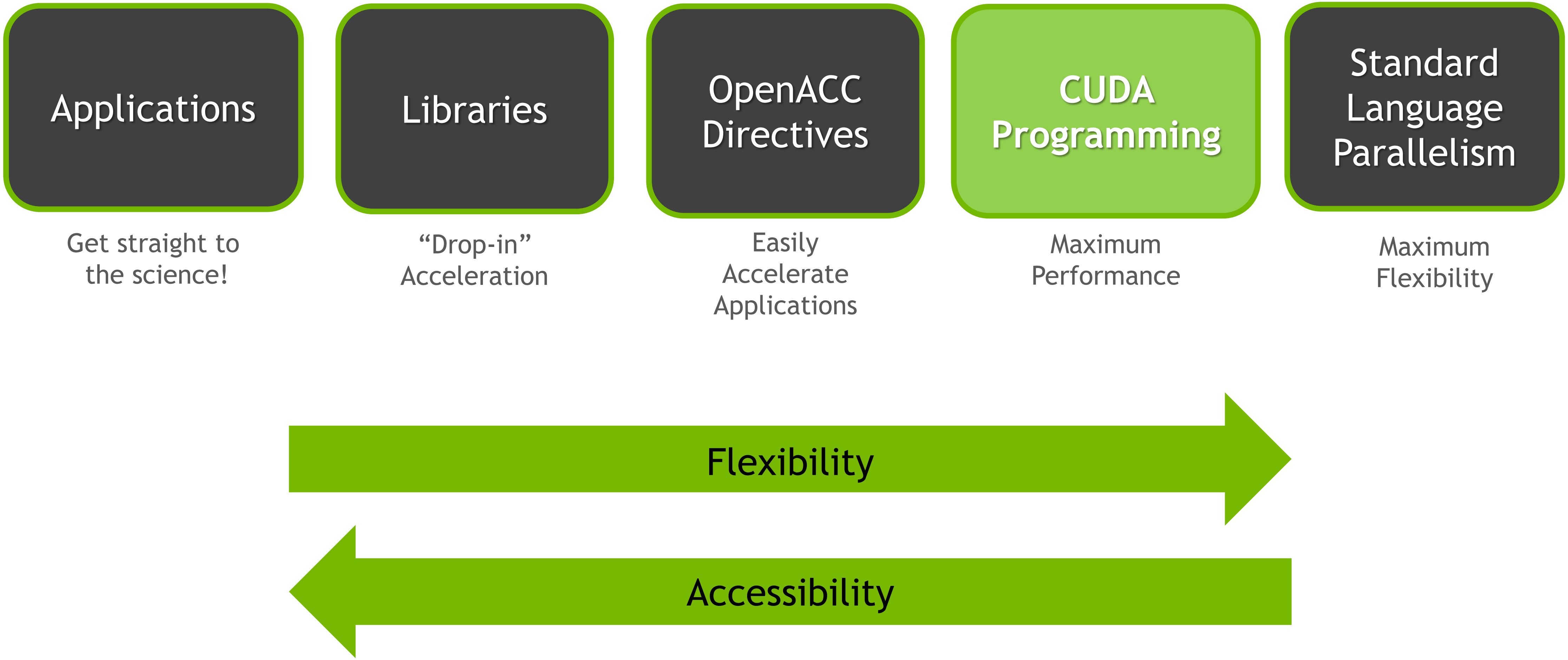
5 CSCS Codes

ANSYS Fluent R18.0 Radiation Solver



CPU: (Haswell EP) Intel(R) Xeon(R) CPU E5-2695 v3 @2.30GHz, 2 sockets, 28 cores
GPU: Tesla K80 12+12 GB, Driver 346.46

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

CUDA Programming (ultimate control)

https://developer.nvidia.com/blog/even-easier-introduction-cuda/

CUDA gives you fine-level control over

- thread execution
- use of GPU memory hierarchy

Tune your code for optimal performance

Scale your parallel execution to multiple GPUs and multiple hosts using NCCL and MPI

CUDA API – C, C++, Fortran, Julia, Python

CUDA aware MPI (OpenMPI, MVAPICH, Spectrum MPI, and more)

CUDA C

```
void saxpy_serial(int n,
                  float a,
                  float *x,
                  float *y)
{

  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

// Perform SAXPY on 1M elements
saxpy_serial(4096*256, 2.0, x, y);
```

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

// Perform SAXPY on 1M elements
saxpy_parallel<<<4096,256>>>(n,2.0,x,y);
```

http://developer.nvidia.com/cuda-toolkit

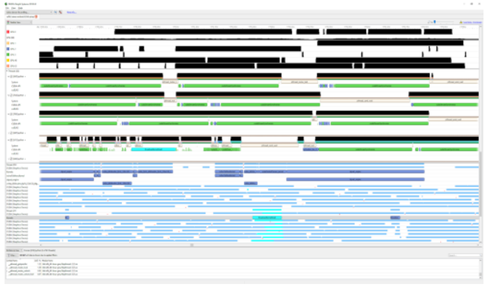
RAPID PARALLEL C++ DEVELOPMENT

- Resembles C++ STL
- High-level interface
  - Enhances developer productivity
  - Enables performance portability between GPUs and multicore CPUs
- Flexible
  - CUDA, OpenMP, and TBB backends
  - Extensible and customizable
  - Integrates with existing software
- Open source

```
// generate 32M random numbers on host
thrust::host_vector<int> h_vec(32 << 20);
thrust::generate(h_vec.begin(),
                 h_vec.end(),
           `      rand);
// transfer data to device (GPU)
thrust::device_vector<int> d_vec = h_vec;
// sort data on device
thrust::sort(d_vec.begin(), d_vec.end());
// transfer data back to host
thrust::copy(d_vec.begin(),
             d_vec.end(),
             h_vec.begin());
```

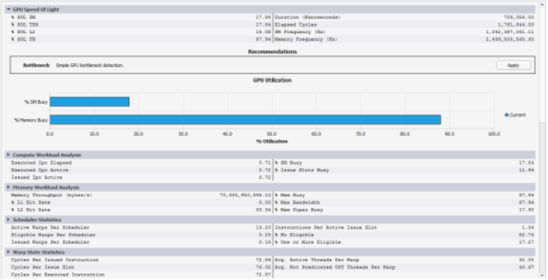
http://developer.nvidia.com/thrust   or http://thrust.googlecode.com

COMPUTE DEVELOPER TOOLS



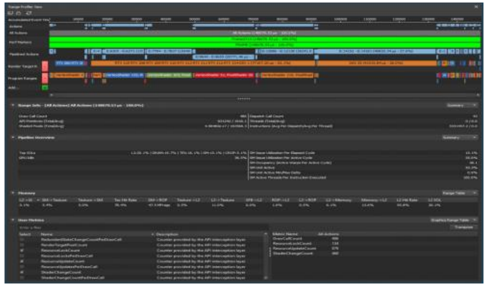
Nsight Systems

System-wide application algorithm tuning



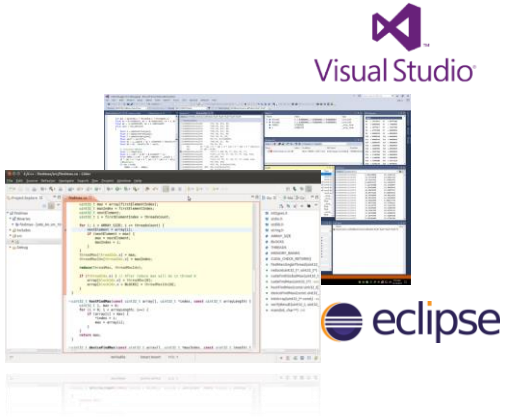
Nsight Compute

CUDA Kernel Profiling and Debugging



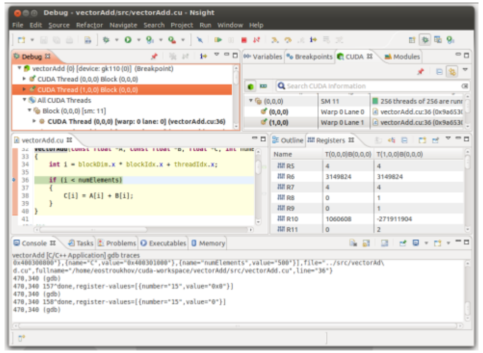
Nsight Graphics

Graphics Shader Profiling and Debugging



IDE Plugins

Nsight Eclipse Edition/Visual Studio (Editor, Debugger)



Debug - vectorAdd/src/vectorAdd.cu - Nsight

```
vectorAdd (0) [device: gk110 (0)] (Breakpoint)
  CUDA Thread (0,0,0) Block (0,0,0)
  CUDA Thread (1,0,0) Block (0,0,0)
  All CUDA Threads
    Block (0,0,0) [sm: 11]
      CUDA Thread (0,0,0) [warp: 0 lane: 0] (vectorAdd.cu:36)
```

```
(0,0,0)      SM 11            256 threads of 256 are runr
  (0,0,0)    Warp 0 Lane 0    vectorAdd.cu:36 (0x9a653(
  (1,0,0)    Warp 0 Lane 1    vectorAdd.cu:36 (0x9a653(
```

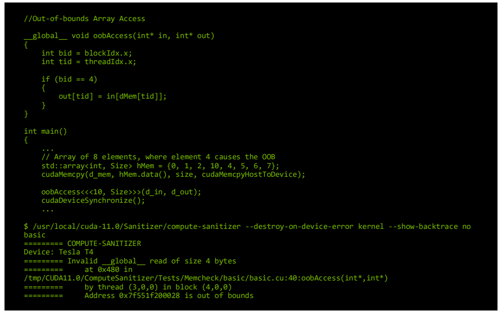
```
vectorAdd(const float *A, const float *B, float *C, int nume
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements)
    {
        C[i] = A[i] + B[i];
    }
}
```

Name	T(0,0,0)B(0,0,0)	T(1,0,0)B(0,0,0)
R5	4	4
R6	3149824	3149824
R7	4	4
R8	0	1
R9	0	1
R10	1060608	-271911904
R11	0	2

```
vectorAdd [C/C++ Application] gdb traces
0x400300000"},{name="C",value="0x400301000"},{name="numElements",value="500"}],file="../src/vectorAd\
d.cu",fullname="/home/eostroukhov/cuda-workspace/vectorAdd/src/vectorAdd.cu",line="36"}
470,340 (gdb)
470,340 157^done,register-values=[{number="15",value="0x0"}]
470,340 (gdb)
470,340 158^done,register-values=[{number="15",value="0"}]
470,340 (gdb)
```

cuda-gdb

CUDA Kernel Debugging

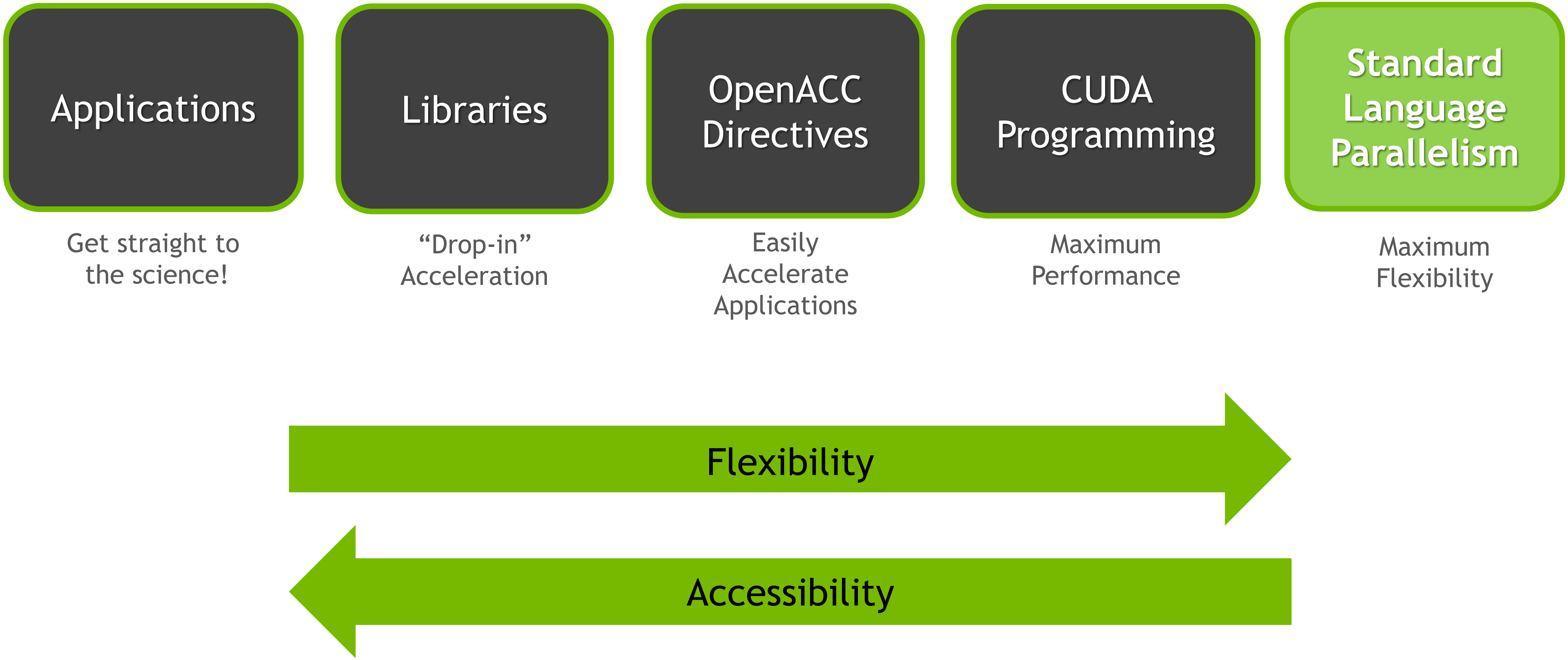


```
//Out-of-bounds Array Access

__global__ void oobAccess(int* in, int* out)
{
    int bid = blockIdx.x;
    int tid = threadIdx.x;

    if (bid == 4)
    {
        out[tid] = in[dMem[tid]];
    }
}

int main()
{
    ...
    // Array of 8 elements, where element 4 causes the OOB
    std::array<int, Size> hMem = {0, 1, 2, 10, 4, 5, 6, 7};
    cudaMemcpy(d_mem, hMem.data(), size, cudaMemcpyHostToDevice);

    oobAccess<<<10, Size>>>(d_in, d_out);
    cudaDeviceSynchronize();
    ...

$ /usr/local/cuda-11.0/Sanitizer/compute-sanitizer --destroy-on-device-error kernel --show-backtrace no
basic
========= COMPUTE-SANITIZER
Device: Tesla T4
========= Invalid __global__ read of size 4 bytes
=========     at 0x480 in
/tmp/CUDA11.0/ComputeSanitizer/Tests/Memcheck/basic/basic.cu:40:oobAccess(int*,int*)
=========     by thread (3,0,0) in block (4,0,0)
=========     Address 0x7f551f200028 is out of bounds
```

Compute Sanitizer

Memory, Race Checking

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

STANDARD LANGUAGE PROGRAMMING

ACCELERATED STANDARD LANGUAGES

ISO C++

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

```
matrix_product(par, mA, mB,
mC);
```

ISO Fortran

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

```
C = matmul(A, B)
```

Python

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

```
c = np.matmul(a, b)
```

PLATFORM SPECIALIZATION

CUDA

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

https://developer.nvidia.com/blog/accelerating-standard-c-with-gpus-using-stdpar/
https://developer.nvidia.com/blog/accelerating-fortran-do-concurrent-with-gpus-and-the-nvidia-hpc-sdk/
https://developer.nvidia.com/cunumeric

HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

Preview support coming to NVC++

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>`

C++23 and Beyond

Executors / Senders-Recievers

- Simplify launching and managing parallel work across CPUs and accelerators

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Range-Based Parallel Algorithms

- Improved multi-dimensional loops

Linear Algebra

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

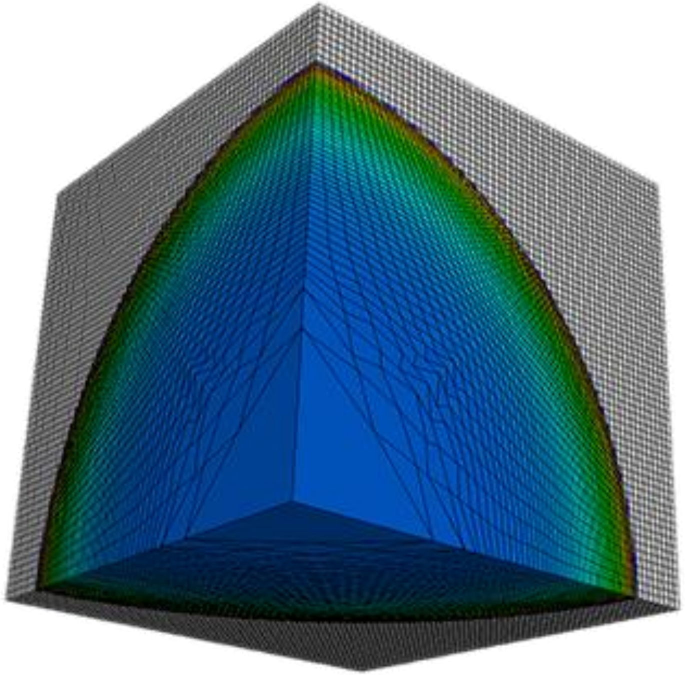
Extended Floating Point Types

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

C++17 PARALLEL ALGORITHMS

Lulesh Hydrodynamics Mini-app

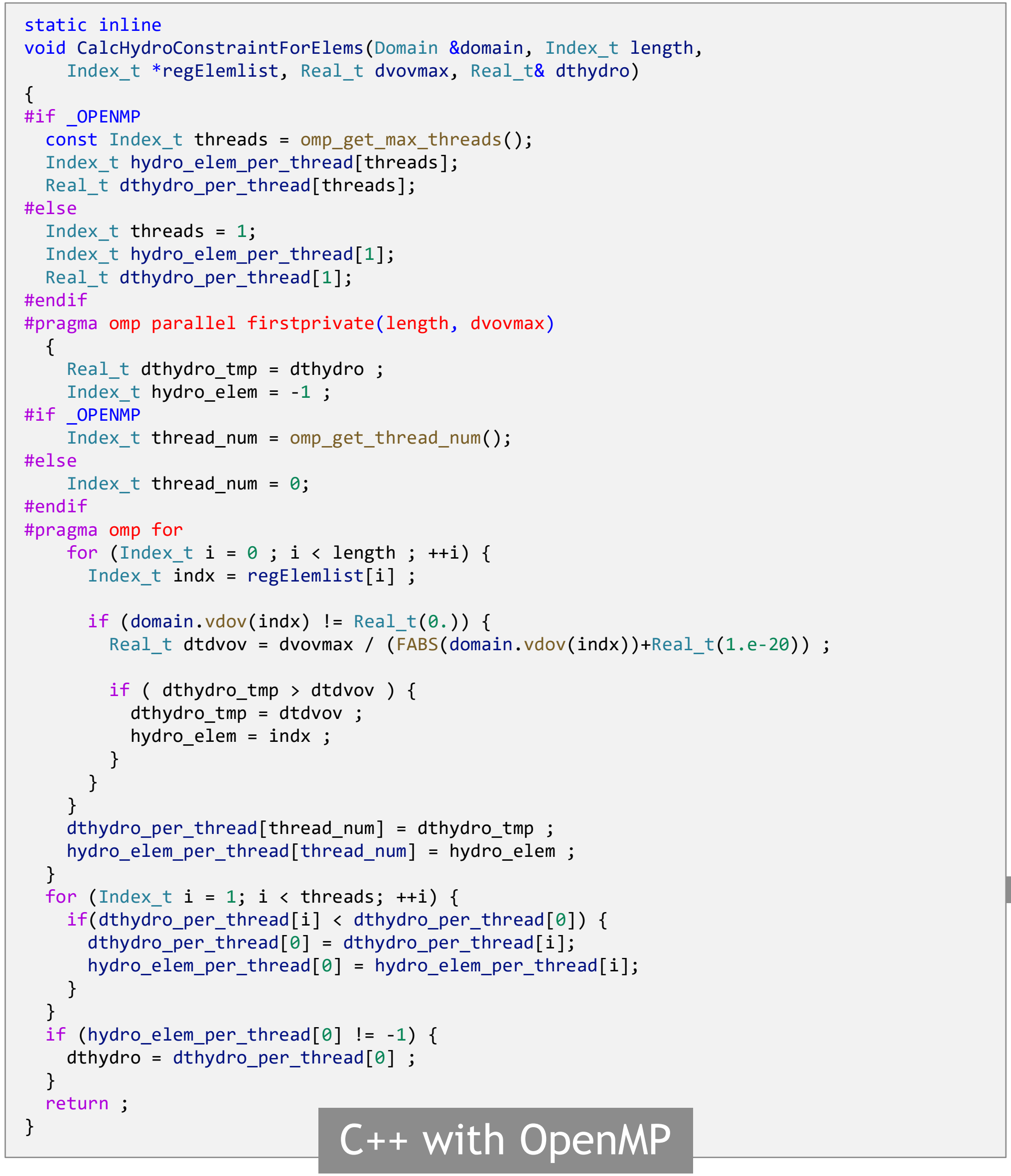
- ~9000 lines of C++
- Parallel versions in MPI, OpenMP, OpenACC, CUDA, RAJA, Kokkos, ISO C++…
- Designed to stress compiler vectorization, parallel overheads, on-node parallelism



codesign.llnl.gov/lulesh

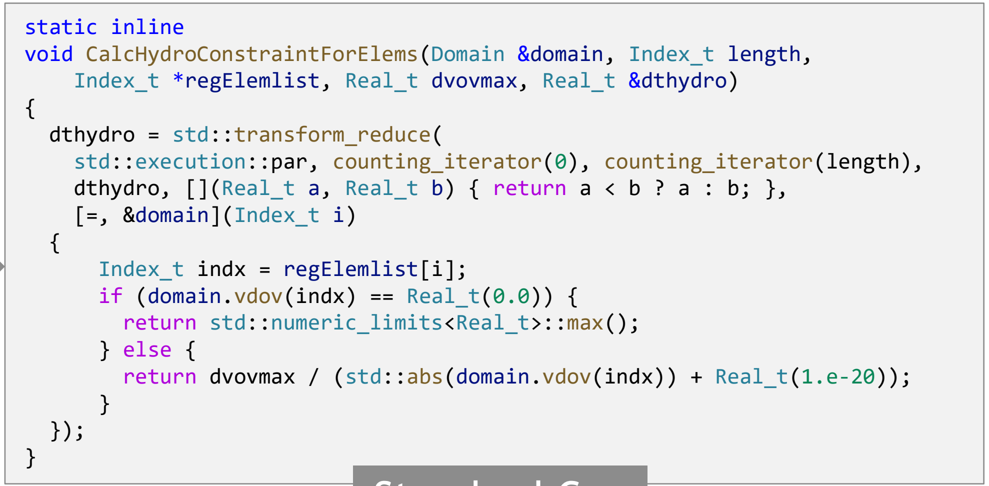
STANDARD C++

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



```
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 ;
}
```

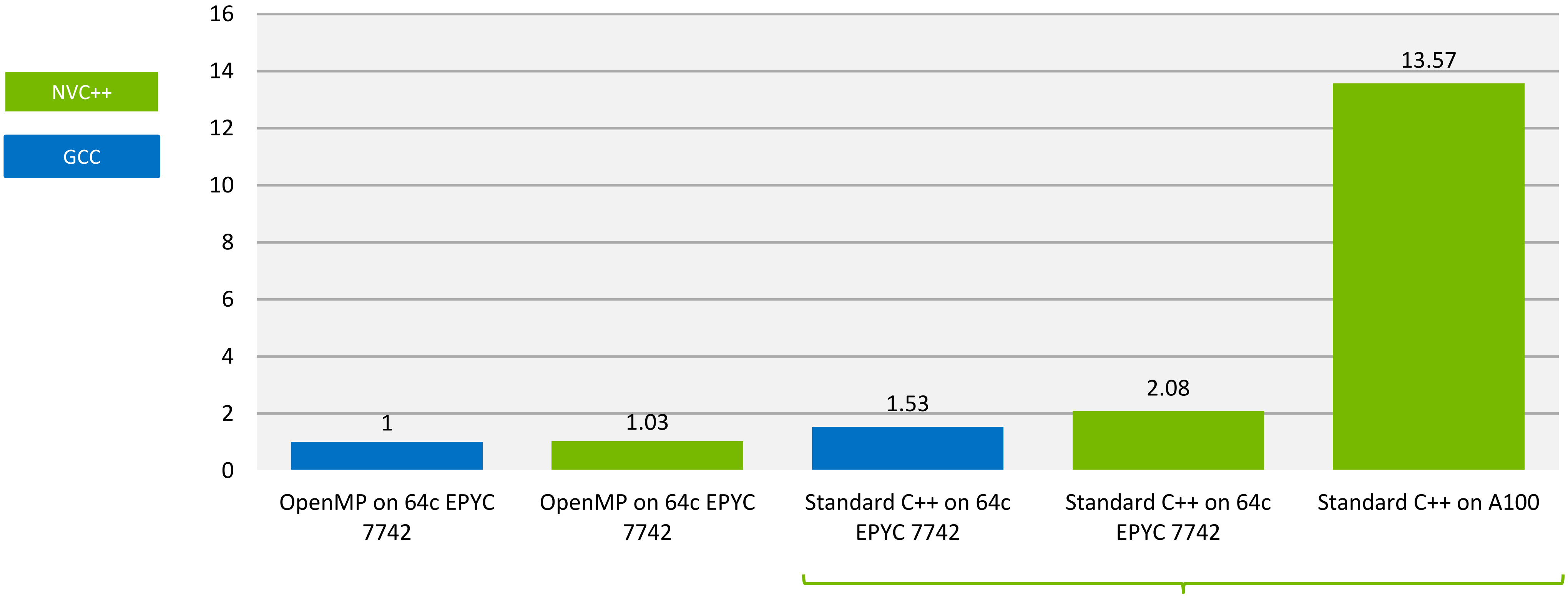
C++ with OpenMP



```
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));
      }
  });
}
```

Standard C++

C++ STANDARD PARALLELISM



Same ISO C++ Code

ACCELERATED STANDARD LANGUAGES

Parallel performance for wherever your code runs

ISO C++

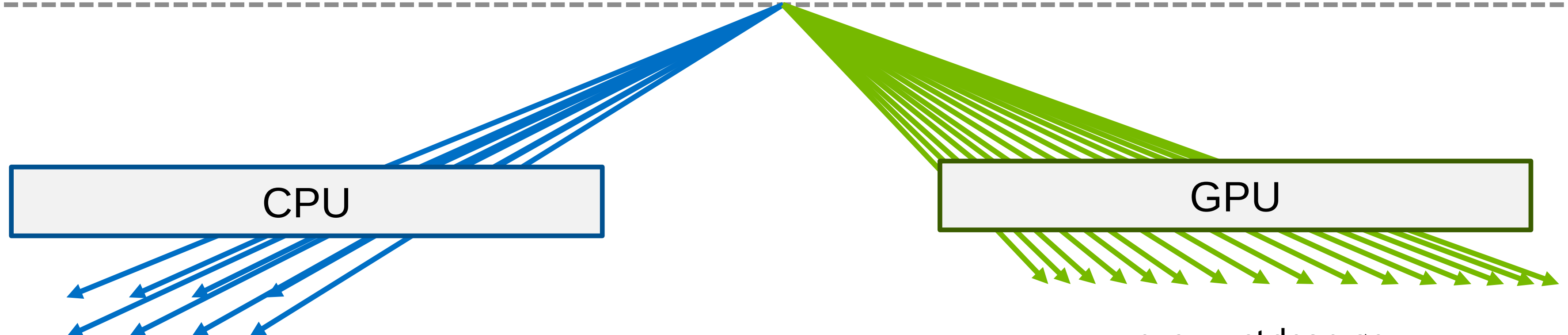
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

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

Python

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```



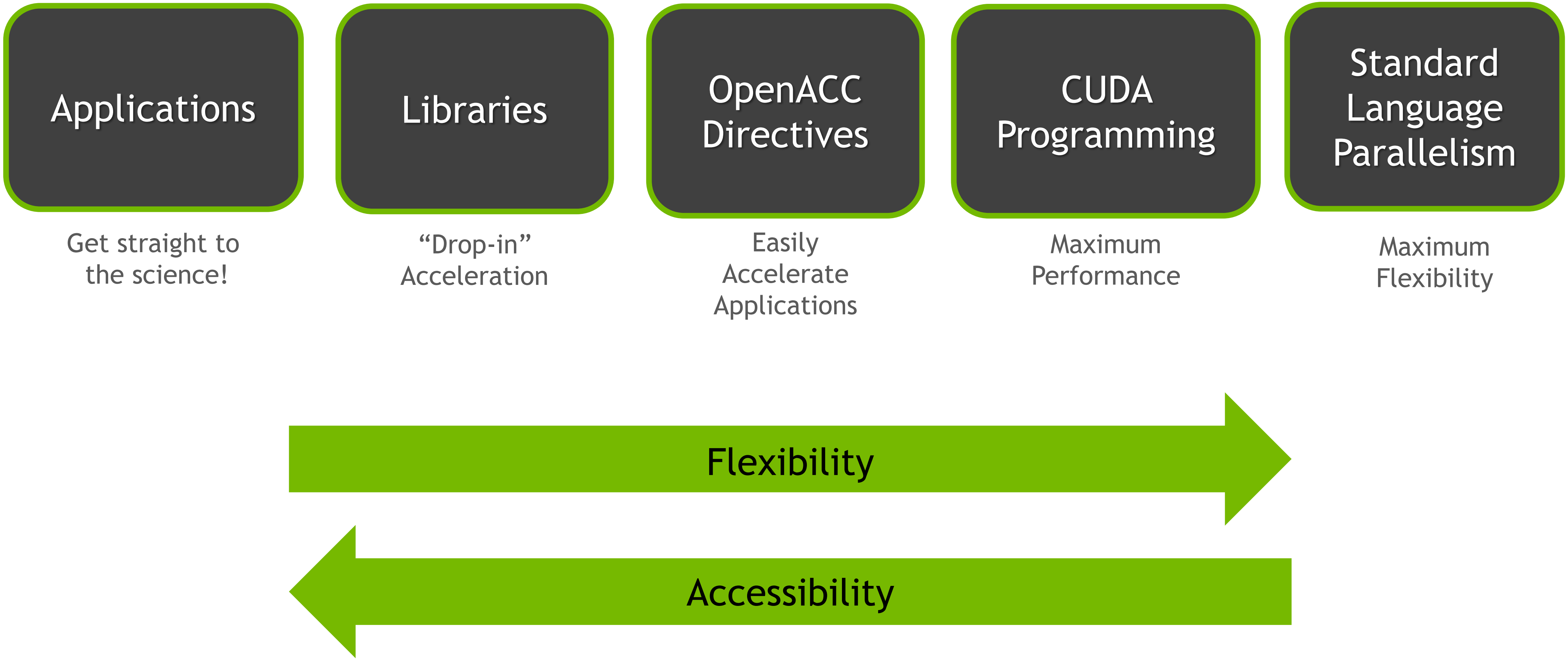
CPU

GPU

```
nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
legate -cpus 16 saxpy.py
```

```
nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
legate -gpus 1 saxpy.py
```

5 WAYS TO ACCELERATE WITH GPUS



Applications

Get straight to the science!

Libraries

“Drop-in” Acceleration

OpenACC Directives

Easily Accelerate Applications

CUDA Programming

Maximum Performance

Standard Language Parallelism

Maximum Flexibility

Flexibility

Accessibility

NVIDIA®