Accelerating Biomedical Data Science with GPUs: Practical Approaches and Tools

Minerva Scientific Computing Environment

https://labs.icahn.mssm.edu/minervalab

S M Shamimul Hasan, Ph.D.
The Minerva HPC Team

October 23, 2024

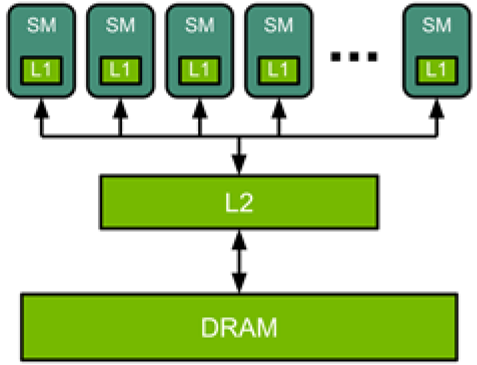
Icahn School of Medicine at **Mount Sinai**

Outline

- GPU Architecture Fundamentals
- Ways to Accelerate with GPUs
  - Application-Based Solutions
  - GPU-Optimized Libraries
  - OpenACC Directives
  - CUDA Programming
  - Standard Language Parallelism

GPU Architecture Fundamentals

- GPUs are equipped with thousands of smaller, efficient cores that can perform simple tasks in parallel
- Key Architectural Components:
  - **Streaming Multiprocessors (SMs)**
    - The core computational units of a GPU
    - Each SM contains multiple CUDA cores, responsible for parallel data processing
    - SMs have their own L1 cache to store frequently accessed data and shared memory for fast data sharing between threads
    - Warp scheduling: SMs execute instructions in parallel, typically in groups of 32 threads (warps), which helps maximize throughput
  - **L2 Cache**
    - Shared by all SMs, which improves data access efficiency when multiple SMs need the same data
  - **High-Bandwidth DRAM**
    - Used for storing data
    - Data is fetched from DRAM to SMs via the L2 and L1 caches to optimize memory bandwidth usage



https://docs.nvidia.com/deeplearning/performance/dl-performance-gpu-background/index.html

GPU Architecture Fundamentals

- Multiply-Add Operations:
  - One of the most frequent operations in neural networks is **multiply-add**, used to compute dot products in fully-connected and convolutional layers
  - GPUs are optimized for these operations, with each multiply-add operation counting as two floating-point operations (FLOPs). Modern GPUs can process millions to billions of these operations per second, making them ideal for AI and machine learning applications that require high computational throughput
- Tensor Cores and CUDA Cores:
  - **Tensor Cores** (introduced in Volta architecture) are specialized units for accelerating matrix multiplications, critical for machine learning
  - **CUDA Cores** handle general-purpose computing tasks when operations do not fit the matrix multiplication model, such as element-wise operations

Ways to Accelerate with GPUs

- Application-Based Solutions
  - Directly leverage pre-built applications for immediate results
- GPU-Optimized Libraries
  - Utilize high-performance libraries for seamless acceleration
- OpenACC Directives
  - Simplify code modifications to accelerate existing applications easily
- CUDA Programming
  - Gain maximum performance through custom GPU code development
- Standard Language Parallelism
  - Flexibly integrate GPU acceleration using standard parallelism techniques

Ways to Accelerate with GPUs: Application-Based Solutions

Key Applications Across Industries

Industry	Popular GPU-Accelerated Applications
Artificial Intelligence	PyTorch, MXNet, TensorFlow, Caffe, Keras, Scikit-learn, ONNX, DeepStream
Climate & Weather	Cosmos, Gales, WRF, MPAS, NEMS, RegCM, GEM, ICON
Computational Finance	O-Quant Options Pricing, Murex, MISYS, Numerix, GPUdb, RiskVal, CuQuant
Data Science & Analytics	Anaconda, H2O, OmniSci, RAPIDS, Dask, XGBoost, TensorRT, cuML
Federal Defense & Security	ArcGIS Pro, EVNI, SocetGXP, Cylance, FaceControl, Raytheon, Harris Geospatial, TensorVision
Life Sciences	Amber, LAMMPS, GROMACS, NAMD, Relion, VASP, AlphaFold, SCHRODINGER
Manufacturing & Engineering	Ansys Fluent, Abaqus SIMULIA, AutoCAD, CST Studio Suite, Altair, Simcenter, OpenFOAM, NASTRAN
Media & Entertainment	DaVinci Resolve, Premiere Pro CC, Redshift Renderer, Autodesk Maya, Blender, Nuke, Unreal Engine, 3ds Max
Medical Imaging	Aidoc, PowerGrid, RadiAnt, NVIDIA Clara, Arterys, iCAD, Visage, Philips IntelliSpace
Oil & Gas	Echelon, RTM, SPECfem3D, Paradigm, Schlumberger Eclipse, PetroMod, JewelSuite, GeoTeric
Retail	Everseen, Deep North, Third Eye Labs, AWM, Malong, Clarifai, Antuit, Google Cloud AI
Supercomputing & HPC	Chroma, GTC, MILC, QUDA, XGC, HPL, NWChem, VMD, BerkeleyGW

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

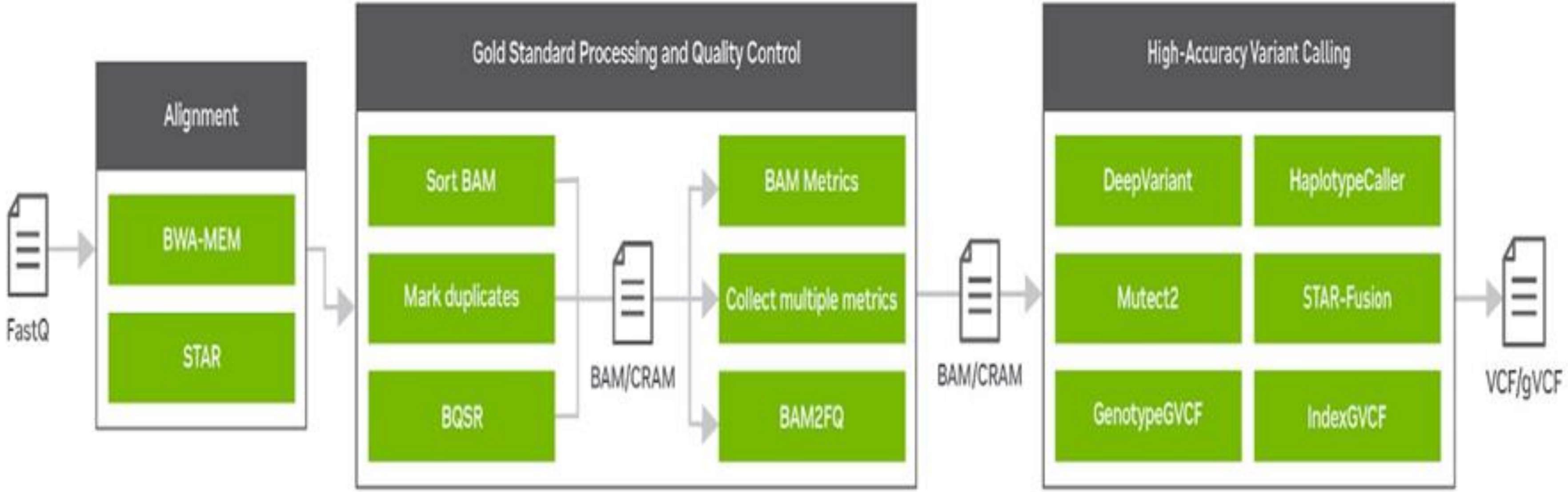
Performance Gains of Standard Benchmarks: A100 vs Dual CPU

Application	Speedup on A100 vs Dual CPU
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

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/05/FiveWays-HealthCare-April2024.pdf

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

NVIDIA Parabricks for Alignment & Variant Calling



https://developer.nvidia.com/blog/new-research-highlights-speed-and-cost-savings-of-clara-parabricks-for-genomic-analyses/

NVIDIA Parabricks for Alignment & Variant Calling

- **Alignment (BWA-MEM, Minimap2, STAR)**
  - **GPU:** 11 minutes | **CPU:** ~4 hours
  - Input: FastQ files
- **Gold Standard Processing & Quality Control (Sort BAM, Mark Duplicates, BQSR)**
  - **GPU:** 6 minutes | **CPU:** ~9 hours
  - Metrics: BAM Metrics, Collect Multiple Metrics
  - Input/Output: BAM/CRAM
- **High-Accuracy Variant Calling (DeepVariant, HaplotypeCaller, Mutect2)**
  - **GPU:** 4-45 minutes | **CPU:** ~16-31 hours
  - Output: VCF/gVCF files

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/05/FiveWays-HealthCare-April2024.pdf

NVIDIA BioNeMo

- NVIDIA BioNeMo is a generative AI platform for drug discovery that simplifies and accelerates the training of models on proprietary data, ensuring easy, scalable model deployment for drug discovery applications
- **Key Features**:
  - **LLM for Proteins & Molecules**: BioNeMo leverages transformer-based LLMs for biological and chemical data, including proteins, DNA, and small molecules
  - **Pretrained Models**: Offers access to pre-trained AI models optimized for tasks such as molecular property prediction, sequence generation, and structure-based drug design
  - **Custom Model Training**: Supports fine-tuning of models on proprietary datasets to meet specific research goals
  - **Integration**: Easily integrates with NVIDIA's GPU-accelerated platforms like Clara and AI frameworks, offering massive parallel processing capabilities

https://www.nvidia.com/en-us/clara/bionemo/

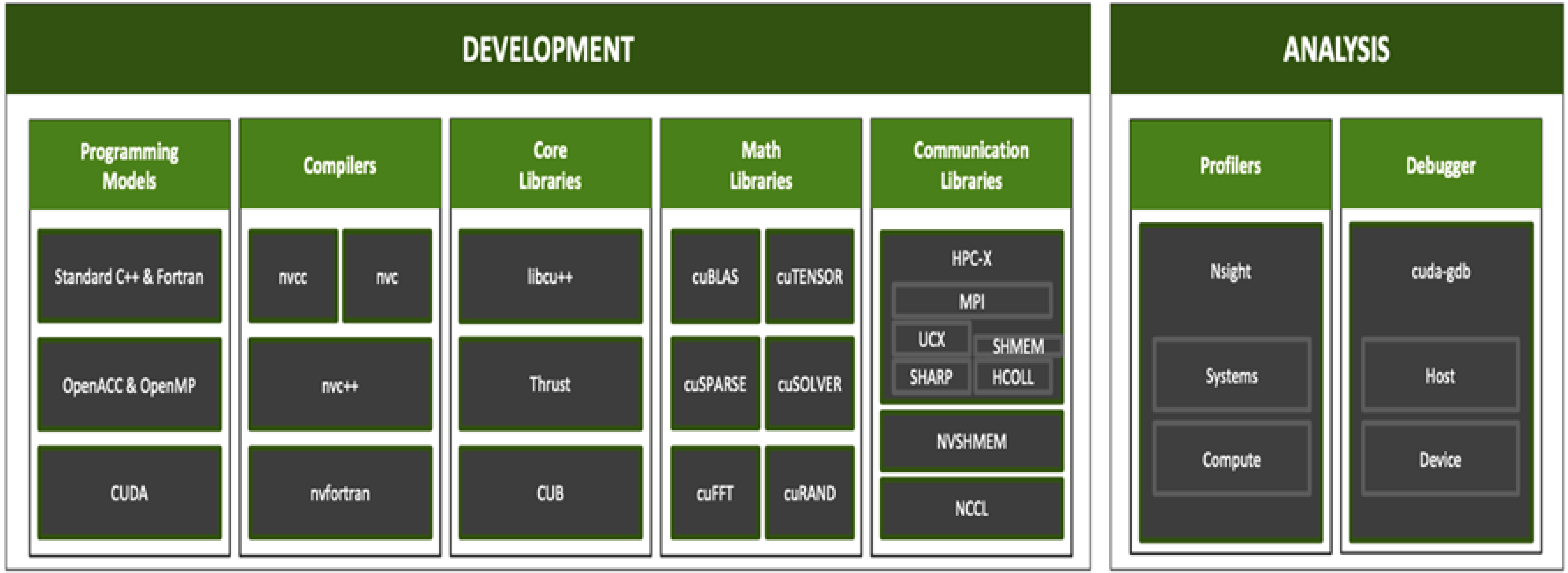
NVIDIA BioNeMo

- **MegaMolBART:** A model for generating and learning representations of small molecules, useful in drug discovery and chemistry
- **ESM-2nv 3B:** A large protein model that predicts protein properties and aids in structure prediction and functional annotation
- **EquiDock DB5 Model:** Predicts protein-protein interactions, essential for understanding biological processes and drug design
- **DiffDock Score Model:** Generates ligand poses for drug-protein interactions, improving drug discovery efforts
- **Geneformer:** Analyzes single-cell gene expression, advancing research in personalized medicine and developmental biology

https://docs.nvidia.com/bionemo-framework/latest/

Ways to Accelerate with GPUs: GPU-Optimized Libraries

NVIDIA HPC Software Development Kit (SDK)



https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/05/FiveWays-HealthCare-April2024.pdf

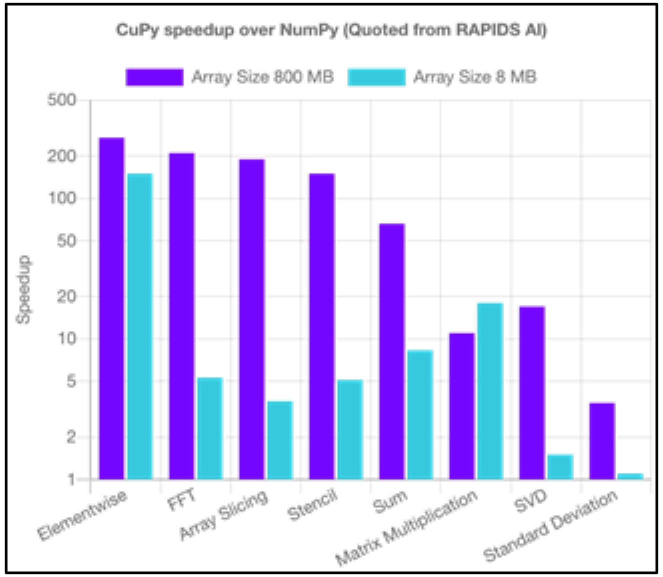
GPU-Accelerated Libraries

- **Linear Algebra Libraries**
  - *cuBLAS*: Basic Linear Algebra Subroutines
  - *cuBLASLt, cuBLASMp, cuBLASDx*: Lightweight, multi-process, and device-side BLAS extensions
  - *cuTENSOR, cuTENSORMg*: Tensor linear algebra, including multi-GPU support
- **Linear Solvers & Sparse Matrix Operations**
  - *cuSOLVER, cuSOLVERMp*: Dense and sparse direct solvers
  - *cuSPARSE, cuSPARSELt*: Sparse matrix BLAS and lightweight variants
- **Fourier Transform & Random Numbers**
  - *cuFFT, cuFFTMp, cuFFTDx*: Fast Fourier Transform variants
  - *cuRAND*: Random number generation
- **Image, Video, and Compression**
  - *NPP, NPP+*: Image, video, and signal processing
  - *nvJPEG, nvJPEG2000, nvTIFF*: JPEG and TIFF encode/decode
  - *nvCOMP*: Data compression/decompression

https://github.com/nvidia/cudalibrarysamples

CuPy: GPU-Accelerated Python Library

- **Overview**: Open-source library that accelerates Python computations using NVIDIA CUDA for high performance on GPUs
- **Performance**: Achieves up to 100x speedups in tasks like linear algebra, deep learning, and random number generation
- **NumPy/SciPy Compatible**: Functions as a drop-in replacement for NumPy and SciPy with minimal code changes
- **Custom Kernels**: Allows easy creation and compilation of custom CUDA kernels for optimized operations
- **Applications**: Ideal for data science, machine learning, and scientific computing



https://cupy.dev/

CuPy: GPU-Accelerated Python Library

NumPy

```
import numpy as np
size = 10000
A = np.random.rand(size, size)
B = np.random.rand(size, size)
C = np.dot(A, B)
```

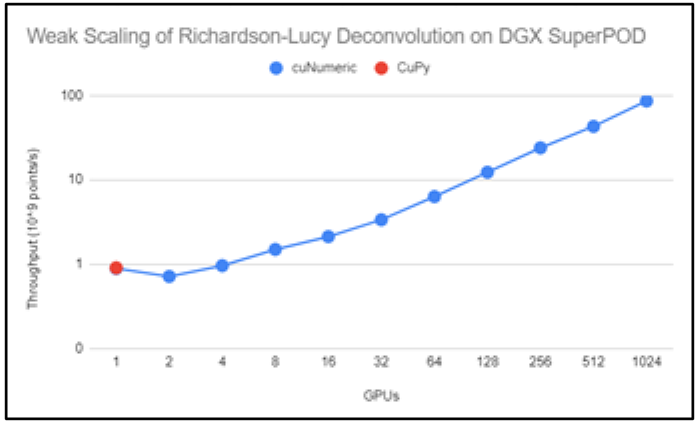
CuPy

```
import cupy as cp
size = 10000
A = cp.random.rand(size, size)
B = cp.random.rand(size, size)
C = cp.dot(A, B)
```

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/04/MountSinai_accelerated_general_data_science-compressed.pdf

cuNumeric: GPU-Accelerated NumPy Replacement

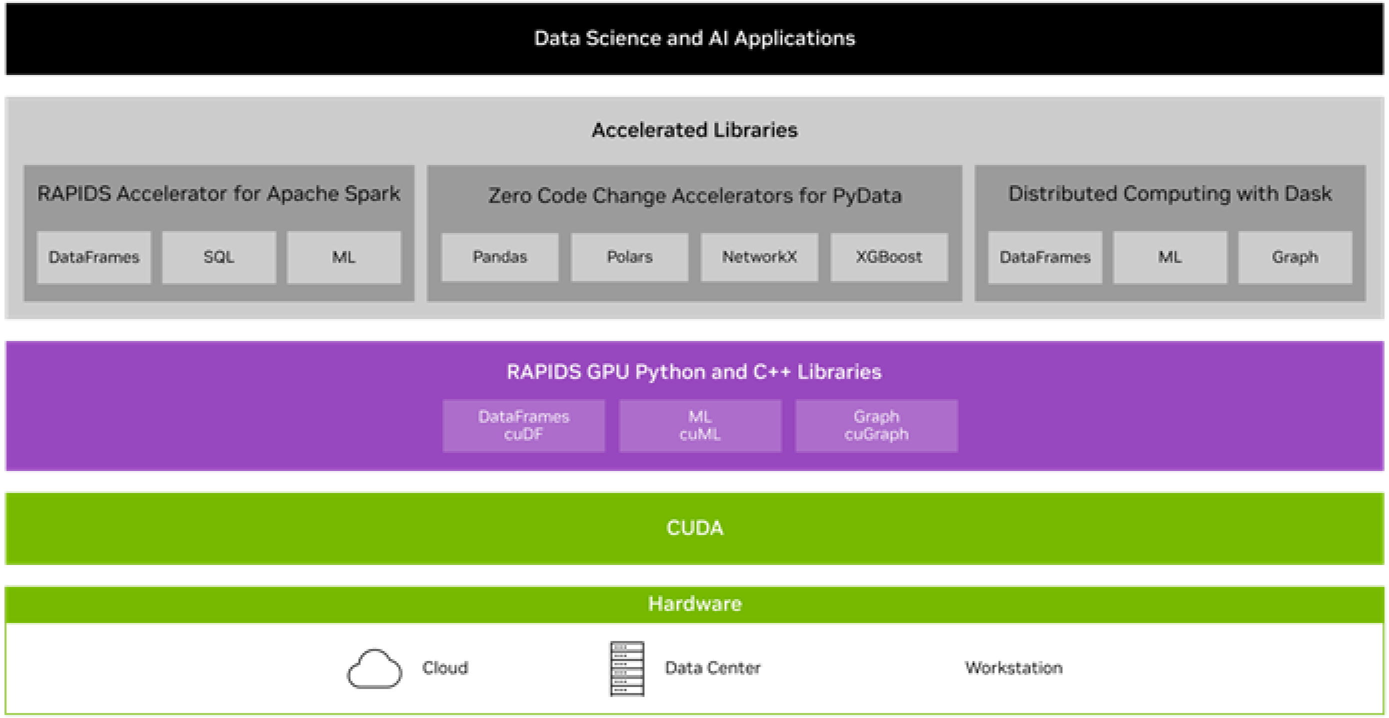
- **Overview**: cuNumeric is a drop-in replacement for NumPy, designed to scale computations across multiple GPUs and nodes without code changes.
- **Key Features**
  - Full NumPy functionality with GPU acceleration
  - Seamless integration with existing Python workflows
  - Leverages the Legate framework for distributed computing
- **Use Case**: Ideal for large-scale data processing tasks in scientific computing, machine learning, and AI
- **Benefit:** Significant performance gains in handling complex numerical computations on large clusters



https://developer.nvidia.com/cunumeric

RAPIDS

- RAPIDS is a collection of open-source software libraries and APIs that enables you to run complete data science and analytics pipelines entirely on GPUs


Data Science and AI Applications
Accelerated Libraries
RAPIDS Accelerator for Apache Spark
DataFrames
SQL
ML
Zero Code Change Accelerators for PyData
Pandas
Polars
NetworkX
XGBoost
Distributed Computing with Dask
DataFrames
ML
Graph
RAPIDS GPU Python and C++ Libraries
DataFrames cuDF
ML cuML
Graph cuGraph
CUDA
Hardware
Cloud
Data Center
Workstation

https://developer.nvidia.com/rapids

RAPIDS

Category	CPU Libraries	GPU Libraries (RAPIDS)
Data Processing	Pandas	cuDF
Machine Learning	scikit-learn	cuML
Graph Processing	NetworkX	cuGraph
Geospatial Data	GeoPandas, SciPy	cuSpatial
Signal Processing	SciPy.signal	cuSignal
Image Processing	scikit-image	cuCIM

cuDF

- A GPU-accelerated DataFrame library
- Similar to pandas, but utilizes the power of GPUs for enhanced performance
- Part of the RAPIDS AI framework developed by NVIDIA
- Designed for large-scale data processing and analytics
- Installable via Conda or Pip

```
import cudf as pd
import numpy as np

# Load a CSV file as a cuDF DataFrame
data_gpu = pd.read_csv("data/sample_data.csv")

# Generate some statistics
mean_values = data_gpu.mean()
print("Mean of each column:\n", mean_values)

# Filtering rows based on a condition
filtered_data =
data_gpu[data_gpu['column_name'] > 50]

# Display filtered data
filtered_data.head(5)
```

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/04/MountSinai_accelerated_general_data_science-compressed.pdf

CPU vs. GPU ETL Workflows

- **Time Consuming ETL (Extract, Transform, Load) Steps (CPU-Powered)**
  - **Configure ETL**: Requires extensive configuration and manual work.
  - **Data Download & Preparation**: Long hours waiting for data downloads
  - **Frequent Restarts**: Restart workflows due to errors or missed steps
  - **Training Delays**: Minimal time left for model training, mostly focusing on data preparation
- **Accelerated ETL and Training (GPU-Powered)**:
  - **Fast Configuration**: Rapid setup of ETL pipelines
  - **Optimized Data Handling**: Handles datasets with increased speed
  - **Integrated Validation and Training**: Time saved for comprehensive model training and validation
  - **Reduced Rework**: Minimized workflow restarts, leading to more efficient work cycles
- **Takeaway**: With GPU acceleration, data scientists spend significantly less time on repetitive ETL tasks, shifting their focus to training, testing, and optimizing machine learning models

cuML - GPU Machine Learning Algorithms

Category	Algorithm
Clustering	Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
	Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)
	K-Means
	Single-Linkage Agglomerative Clustering
Dimensionality Reduction	Principal Components Analysis (PCA)
	Incremental PCA
	Truncated Singular Value Decomposition (tSVD)
	Uniform Manifold Approximation and Projection (UMAP)
	Random Projection

https://github.com/rapidsai/cuml

cuML - GPU Machine Learning Algorithms

Category	Algorithm
Linear Models for Regression or Classification	Linear Regression
	Linear Regression with Lasso or Ridge Regularization
	ElasticNet Regression
	LARS Regression
	Logistic Regression
	Naive Bayes
	Stochastic Gradient Descent (SGD), Coordinate Descent (CD), and Quasi-Newton (QN) (including L-BFGS and OWL-QN) solvers for linear models

https://github.com/rapidsai/cuml

cuML - GPU Machine Learning Algorithms

Category	Algorithm
Nonlinear Models for Regression or Classification	Random Forest (RF) Classification
	Random Forest (RF) Regression
	Inference for decision tree-based models
	K-Nearest Neighbors (KNN) Classification
	K-Nearest Neighbors (KNN) Regression
	Support Vector Machine Classifier (SVC)
	Epsilon-Support Vector Regression (SVR)

https://github.com/rapidsai/cuml

cuML - GPU Machine Learning Algorithms

Category	Algorithm
Preprocessing	Standardization, or mean removal and variance scaling / Normalization / Encoding categorical features / Discretization / Imputation of missing values / Polynomial features generation / and coming soon custom transformers and non-linear transformation
Time Series	Holt-Winters Exponential Smoothing
	Auto-regressive Integrated Moving Average (ARIMA)
Model Explanation	SHAP Kernel Explainer
	SHAP Permutation Explainer

https://github.com/rapidsai/cuml

Numba

- Numba is a just-in-time (JIT) compiler that translates Python code to machine code at runtime, significantly improving performance for numerical computations
- Key Features
  - **JIT Compilation:** Numba compiles Python functions at runtime for fast performance
  - **Easy Integration:** Works seamlessly with NumPy and pandas
  - **Decorator-Based:** Use the @jit decorator to accelerate Python functions without code changes
  - **Support for GPUs:** Numba can target NVIDIA GPUs for parallel computation (@cuda.jit)
- Benefits
  - **Speed:** Offers speed-ups comparable to compiled languages like C
  - **Ease of Use:** No need to rewrite Python code in C or other languages to get better performance
  - **Parallelization:** Enables easy parallel programming with features like GPU support

Numba

GPU

CPU

```
from numba import jit

@jit
def vector_add(a, b):
    n = len(a)
    result = np.zeros(n)
    for i in range(n):
        result[i] = a[i] + b[i]
    return result

# Vector addition on the CPU
result = vector_add(a, b)
```

```
from numba import cuda

@cuda.jit
def vector_add_gpu(a, b, result):
    i = cuda.grid(1)
    if i < len(a):
        result[i] = a[i] + b[i]

# Define grid and block size
threads_per_block = 256
blocks_per_grid = (len(a) + (threads_per_block - 1))
// threads_per_block

# Launch kernel
vector_add_gpu[blocks_per_grid,
threads_per_block](a, b, result)
```

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/04/MountSinai_accelerated_general_data_science-compressed.pdf

Medical Open Network for Artificial Intelligence (MONAI)

- Initiative by NVIDIA and King’s College London
- Built to create an inclusive AI research community for healthcare imaging
- Collaboration includes academic and industry leaders
- Provides open-source PyTorch-based frameworks for:
  - Annotation, model building, training, deployment, and optimization
- Focus on reproducibility and collaboration
- Key components:
  - **MONAI Core:** Training AI models in healthcare imaging
  - **MONAI Label:** Smart image annotation
  - **MONAI Deploy SDK:** Convert models into deployable AI applications
  - **MONAI Model Zoo:** Pre-built medical imaging models

https://monai.io/

https://github.com/Project-MONAI

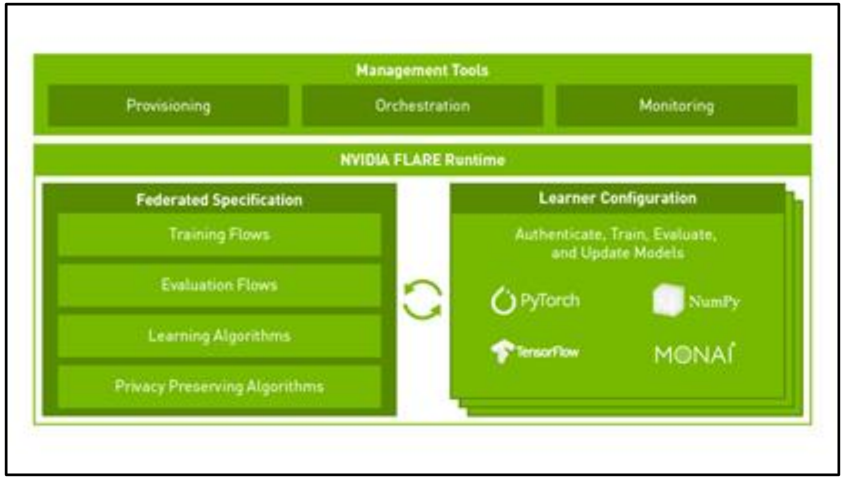
NVIDIA Holoscan

- NVIDIA Holoscan is the sensor processing platform that streamlines the development and deployment of AI and high-performance computing (HPC) applications for real-time insights
- **Key Benefits**
  - **Sensor Processing**: Supports video capture, ultrasound research, and legacy medical devices
  - **Low Latency**: Holoscan SDK helps measure end-to-end latency for video processing
  - **AI Pipelines:** Access AI reference pipelines for radar, high-energy light sources, endoscopy, ultrasound, and other streaming video applications
- **Use Cases**
  - **Medical Devices**: Real-time AI for surgery, helping clinical teams with patient-specific decisions
  - **Edge Computing**: Scalable AI solutions from surgery to satellites

https://www.nvidia.com/en-us/clara/holoscan/

NVIDIA FLARE

- NVIDIA FLARE (NVIDIA Federated Learning Application Runtime Environment) is a domain-agnostic, open-source, and extensible SDK for Federated Learning
- **Key Features**:
  - **Privacy-Preserving Algorithms**: Protects data privacy with algorithms that prevent reverse engineering of model updates
  - **Distributed Multi-Party Collaboration**: Enables AI model development across diverse data sources without sharing data
  - **Supports Popular ML/DL Frameworks**: Integrates seamlessly with frameworks like PyTorch, TensorFlow, and more
  - **Extensible Management Tools**: Offers SSL certifications, admin console, and TensorBoard for experiment monitoring



https://developer.nvidia.com/flare

Ways to Accelerate with GPUs: OpenACC Directives

OpenACC Directives

- OpenACC is a directive-based programming model for parallel computing, designed to make performance-portable code accessible to scientists and engineers across various HPC hardware platforms. It enables efficient parallelization without the complexities of low-level programming
- Key Benefits:
  - **Simplified Code Parallelism:** Use directives to easily identify parallel regions
  - **Accelerator Ready:** Ideal for many-core GPUs and multicore CPUs
  - **Less Effort:** Reduces development time and complexity compared to CUDA or OpenCL

C

```
#pragma acc directive [clause [,] clause] ...
{
// Code to be executed in parallel
}
```

Fortran

```
!$acc directive [clause [,] clause] ...
! Code to be executed in parallel
!$acc end directive
```

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/05/FiveWays-HealthCare-April2024.pdf

SAXPY Example: SAXPY is "Single-Precision A times X Plus Y"

SAXPY in C

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

...

// Perform SAXPY on 1M elements with an
offset of 1000
saxpy_with_offset(1 << 20, 2.5, x, y,
1000);
...
```

SAXPY in Fortran

```
subroutine saxpy_with_offset(n, a, x, y,
offset)
    real :: x(:), y(:), a
    integer :: n, i, offset

    !$acc kernels
    do i = offset + 1, n + offset
        y(i) = a * x(i - offset) + y(i)
    end do
    !$acc end kernels

end subroutine saxpy_with_offset

...

! Perform SAXPY on 1M elements with an offset
of 1000
call saxpy_with_offset(2**20, 2.5, x, y, 1000)
...
```

Ways to Accelerate with GPUs: CUDA Programming

CUDA Programming

- What is CUDA?
  - CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA
  - It enables developers to utilize the power of NVIDIA GPUs for general-purpose computing
- Key Features:
  - **Parallel Computing**: CUDA allows thousands of threads to execute concurrently, maximizing the utilization of the GPU.
  - **Heterogeneous Programming**: Code can run on both the CPU (host) and GPU (device), allowing for efficient division of tasks.
  - **Flexible Memory Management**: CUDA provides various types of memory (global, shared, constant) that can be optimized for different tasks.
- CUDA provides APIs for C/C++, Fortran, Python, Julia
- CUDA-aware MPI implementations include OpenMPI, MVAPICH, Spectrum MPI, and others

CUDA C

```
void saxpy_offset_serial(int n, float a,
float *x, float *y, int offset)
{
    for (int i = offset; i < n + offset;
++i)
    {
        y[i] = a * x[i - offset] + y[i];
    }
}

// Perform SAXPY on 1M elements with an
offset of 1000
saxpy_offset_serial(4096 * 256, 2.0, x, y,
1000);
```

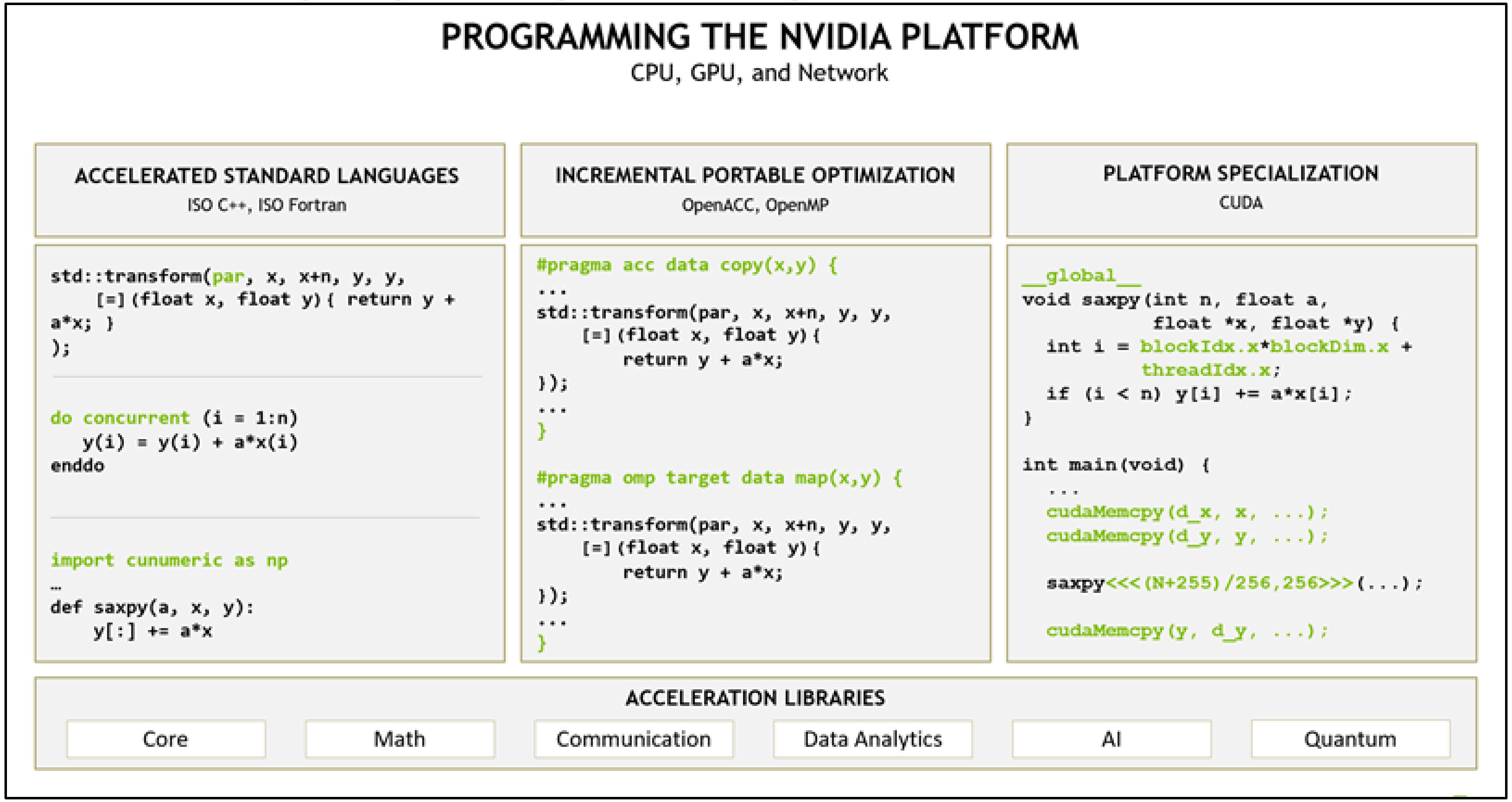
```
__global__
void saxpy_offset_parallel(int n, float a,
float *x, float *y, int offset)
{
    int i = blockIdx.x * blockDim.x +
threadIdx.x + offset;
    if (i < n + offset) {
        y[i] = a * x[i - offset] + y[i];
    }
}

// Perform SAXPY on 1M elements with an
offset of 1000
saxpy_offset_parallel<<<4096, 256>>>(n,
2.0, x, y, 1000);
```

https://labs.icahn.mssm.edu/minervalab/wp-content/uploads/sites/342/2024/05/FiveWays-HealthCare-April2024.pdf

Ways to Accelerate with GPUs: Standard Language Parallelism

Standard Language Programming



PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

```
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
```

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

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

```
#pragma omp target data map(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

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, ...);
```

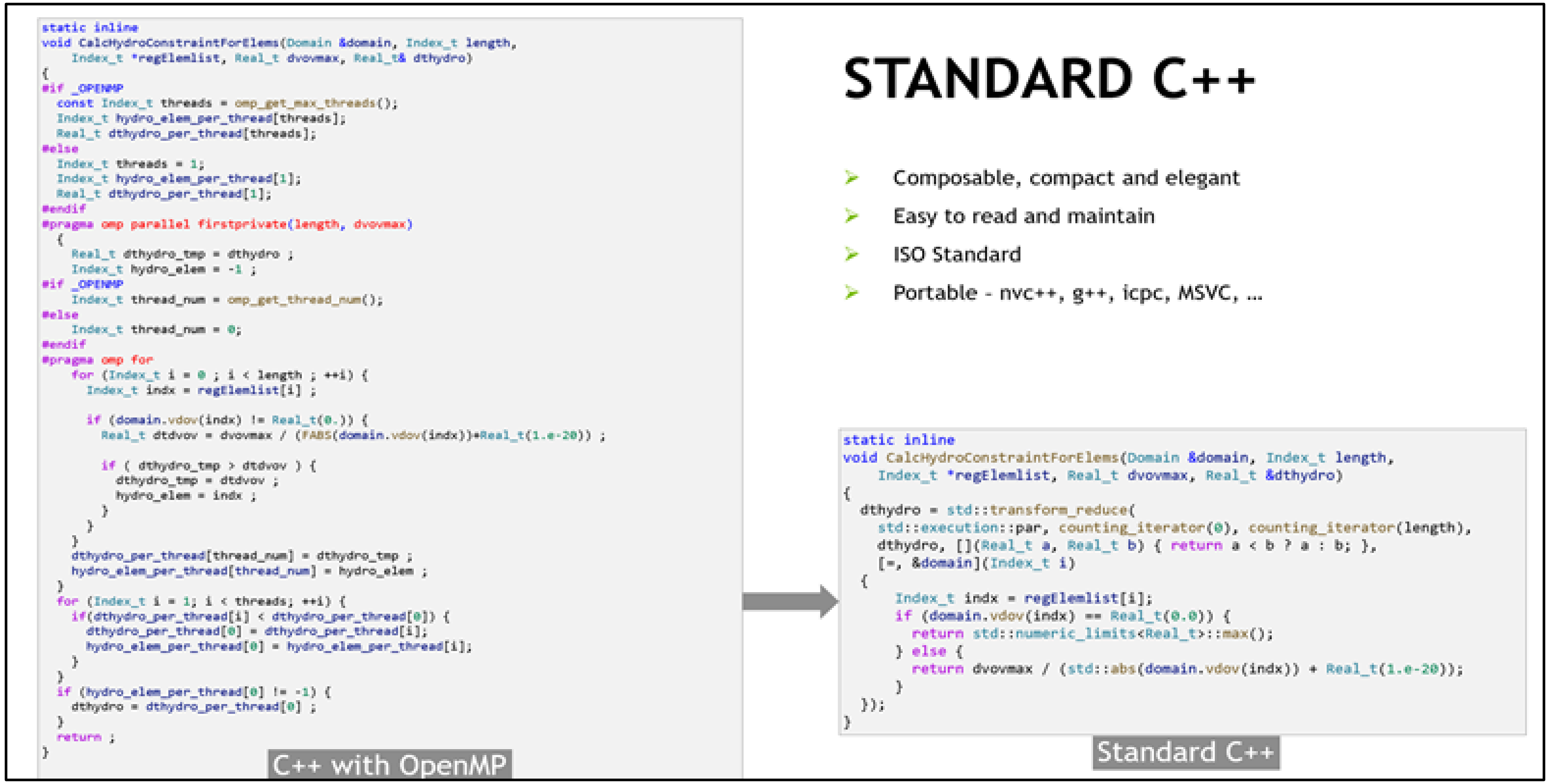
ACCELERATION LIBRARIES

Core	Math	Communication	Data Analytics	AI	Quantum

https://developer-blogs.nvidia.com/wp-content/uploads/2022/01/Fig-1-Programming-the-Nvidia-Platform-2.png

Standard Language Programming

Lulesh Hydronynamics Mini-app



```
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

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)
{
  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++

https://developer-blogs.nvidia.com/wp-content/uploads/2022/01/Fig-2-Standard-C-1.png

User GPU Software Environment - Major packages

OS: **Rocky 9.4** with **glibc-2.34(GNU C library) available**

- Packages with GPU support:
  - Schrödinger Suite, Amber tools, NAMD, Gromacs, Alpha Fold2, etc.
- AI tools with python/3.12.5
  - CuPy, cuDF, cuML, Numba, scikit-learn, Scanpy, Squidpy, etc.
  - Minerva Python instruction
- AI tools with conda
  - MONAI, Rapids, NVFlare, tensorflow, pytorch, etc.
  - Minerva conda instruction
- AI tools with singularity
  - Holoscan, BioNeMo, Parabricks, DeepVariant, etc.
  - Minerva singularity instruction
  - Minerva Singularity training
- Cuda toolkit versions up to 12.4.0
- Nsight Systems

Important Reminder

- Need assistance? Feel free to contact us at:

hpchelp@hpc.mssm.edu

Acknowledgements

- Supported by the Clinical and Translational Science Awards (CTSA) grant UL1TR004419 from the National Center for Advancing Translational Sciences, National Institutes of Health.

CTSA Clinical & Translational Science Awards®