Optimizing research with GPUs on Hoffman2

Charles Peterson

Welcome Everyone!

🎯 Workshop Overview

🚀 Discover the power of GPU computing to accelerate your research on UCLA's Hoffman2 cluster! This beginner-friendly workshop will guide you through the basics of GPU utilization, enhancing your projects with cutting-edge computational efficiency. ⭐

👉 What you'll learn:

- 🧠 **Understanding GPU architecture** and its benefits
- 💻 **Hands-on access** to Hoffman2's advanced GPU resources
- 🐍 **Utilizing Python and R** for GPU computing
- **RPyLab** - A container with RStudio and Jupyter with GPU support (experimental)

For suggestions: cpeterson@oarc.ucla.edu

📖 Access the Workshop Files

This presentation and accompanying materials are available on 🔗 UCLA OARC GitHub Repository

You can view the slides in:

- 📄 PDF format - WS_HPC-GPU.pdf
- 🌐 HTML format: Workshop Slides
- 📹 Recordings can be found on our BOX account

> **Note:** 🛠️ This presentation was built using Quarto and RStudio.

Clone repository to access the workshop files:

```
git clone https://github.com/ucla-oarc-hpc/WS_HPC-GPU.git
```

GPU Basics

🤔 What are Graphic Processing Units?

- Initially developed for processing graphics and visual operations
  - CPUs were too slow for these tasks
  - Architecture of GPUs allows to handle massively parallel tasks efficiently
  - Found in everything from PCs, mobile phones, gaming consoles, and more

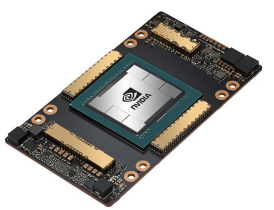
🚀 In the mid-2000s, GPUs began to be used for non-graphical computations. NVIDIA introduced CUDA, a programming language that allows for compiling non-graphic programs on GPUs, spearheading the era of General-Purpose GPU (GPGPU).

GeForce 256

- First ‘GPU’ in 1999
- 32 MB of memory
- 960 MFLOPS (FP32)

A100

- 80 GB of memory
- 19.5 TFLOPS (FP32)

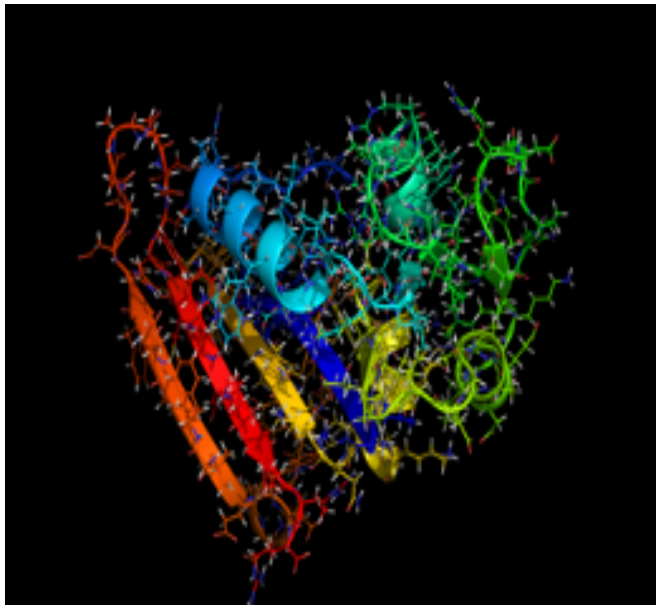


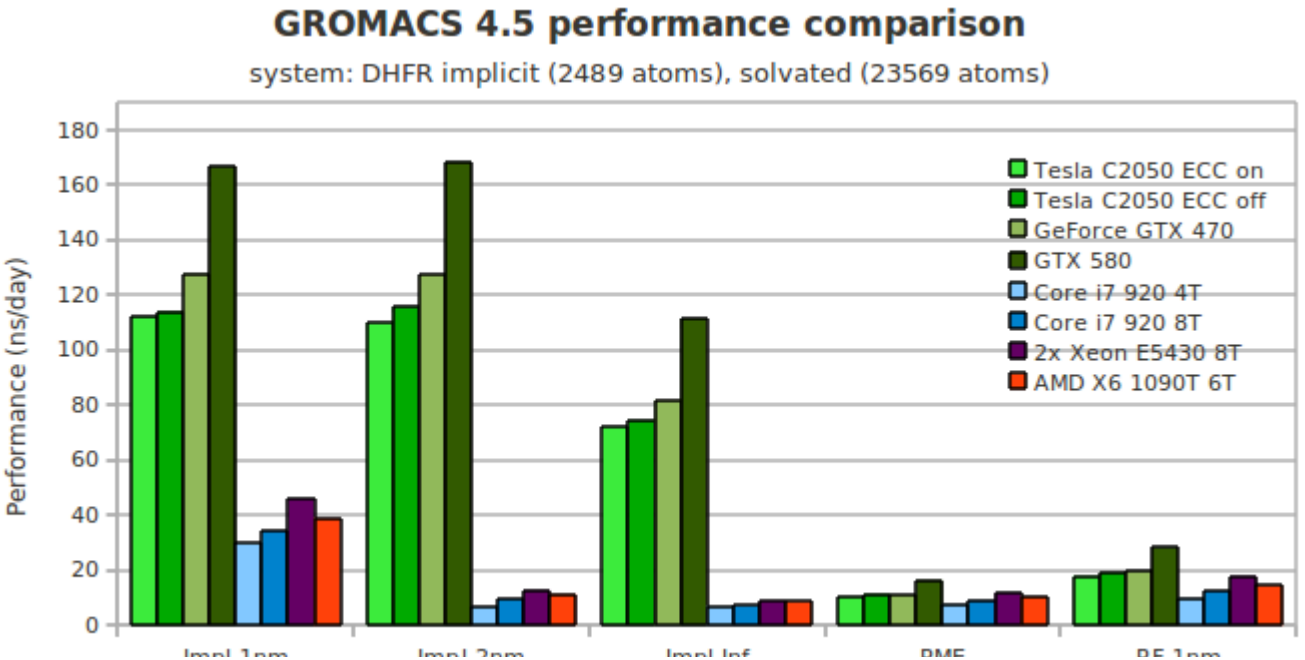
🌐 Applications of GPUs

GPUs are ubiquitous and found in devices ranging from PCs to mobile phones, and gaming consoles like Xbox and PlayStation.

Though initially designed for graphics, GPUs are now used in a wide range of applications.

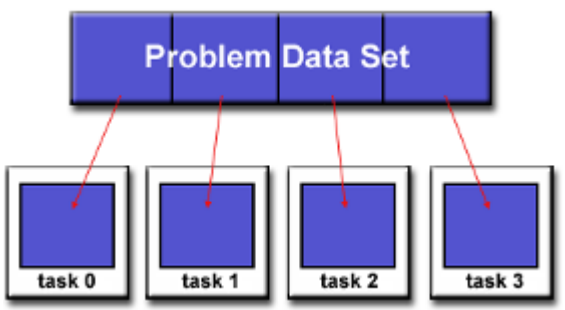
GPU Performance

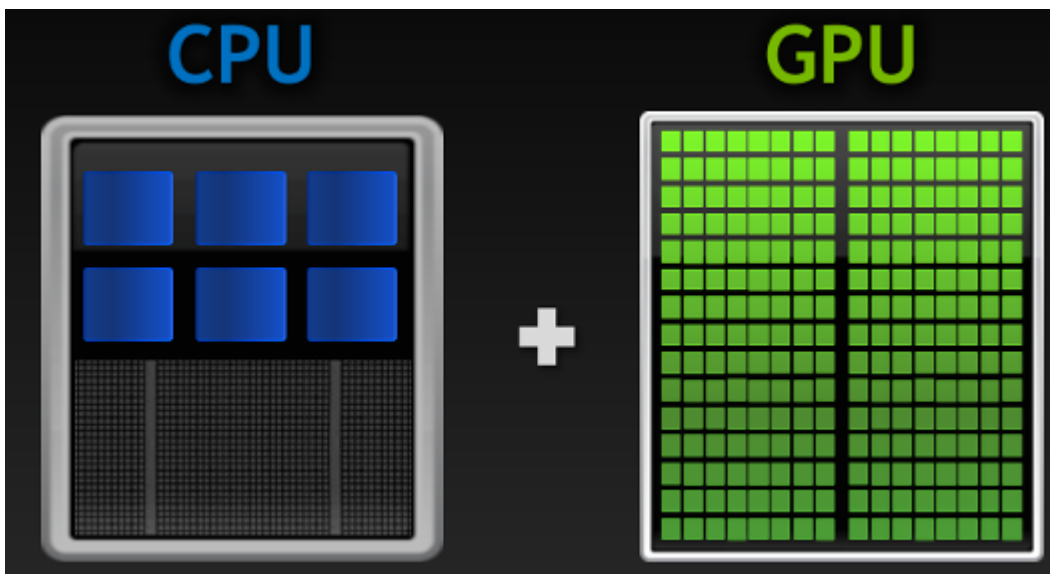




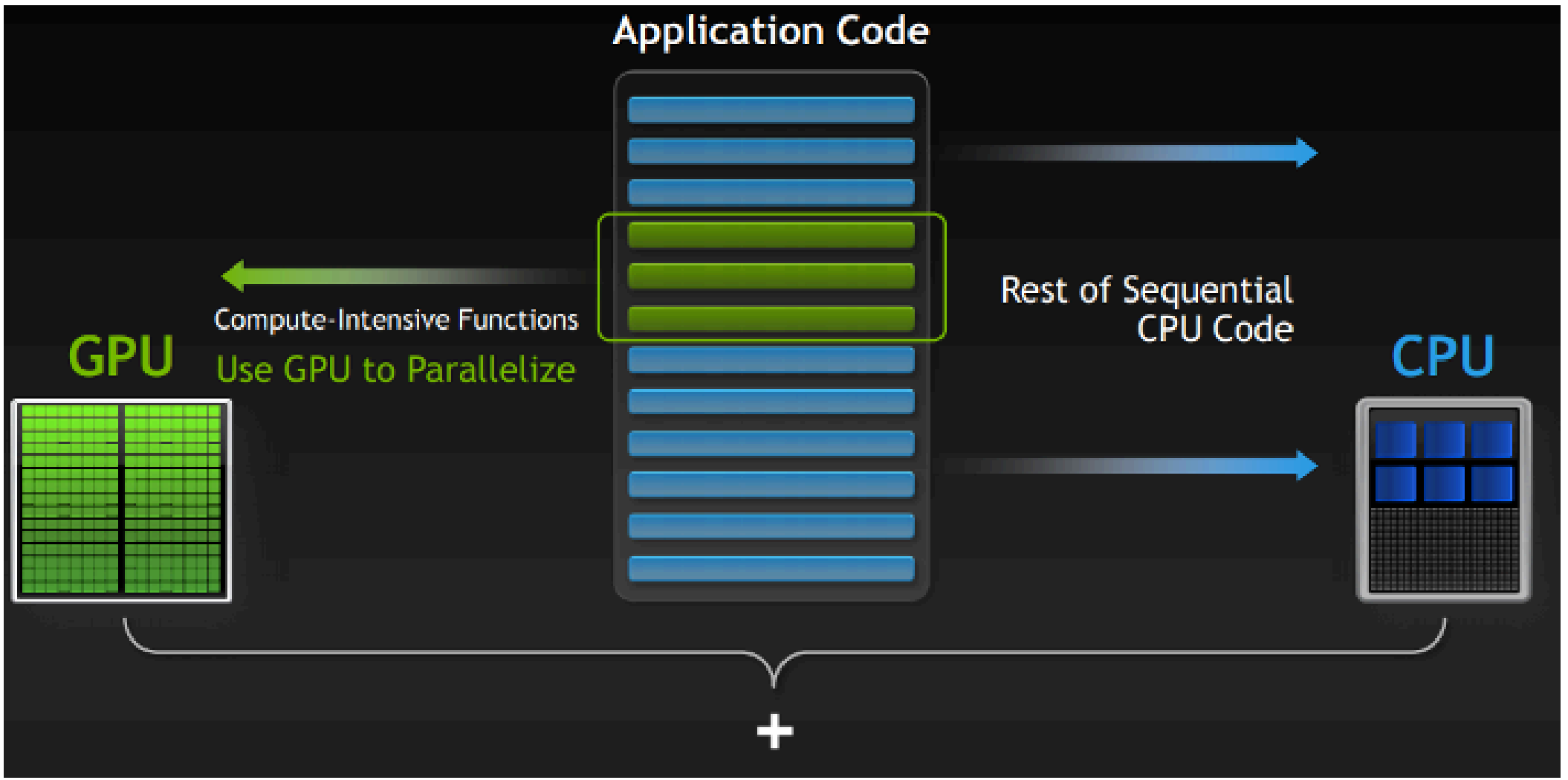
⚡ The Power of GPUs

The significant speedup offered by GPUs comes from their ability to parallelize operations over thousands of cores, unlike traditional CPUs.





GPU Workflow



👷 GPU considerations

- 🚧 **Code Optimization:** Some codes are not suitable for GPU.
- 👷 **GPU architecture:** Some codes can run more efficiently on some GPUs over others, or sometimes not at all.
- 🔄**Overhead:** Data transfer between CPU and GPU can be costly.
- 🧠 **Memory Management:** GPU memory is limited and can be a bottleneck.

📈 GPUs on Hoffman2

There are multiple GPU types available in the cluster. Each GPU has a different compute capability, memory size, and clock speed.

GPU type	# CUDA cores	VMem	SGE option
NVIDIA A100	6912	80 GB	-l gpu,A100,cuda=1
Tesla V100	5120	32 GB	-l gpu,V100,cuda=1
RTX 2080 Ti	4352	10 GB	-l gpu,RTX2080Ti,cuda=1
Tesla P4	2560	8 GB	-l gpu,P4,cuda=1

Interactive job

```
qrsh -l h_data=40G,h_rt=1:00:00,gpu,A100,cuda=1
```

Batch submission

```
#SBATCH -l gpu,A100,cuda=1
```

> **Note**
>
> If you would like to host GPU nodes on Hoffman2 or get `highp` access, please contact us!

GPU optimization

> ⚠ **Warning**
>
> When you using the `-l gpu` option, this only reserves the GPU for your job.
>
> You will still need to use GPU optimized software and libraries to take advantage of the GPU's parallel processing power.

The following sections will cover how to compile and run GPU optimized code on Hoffman2.

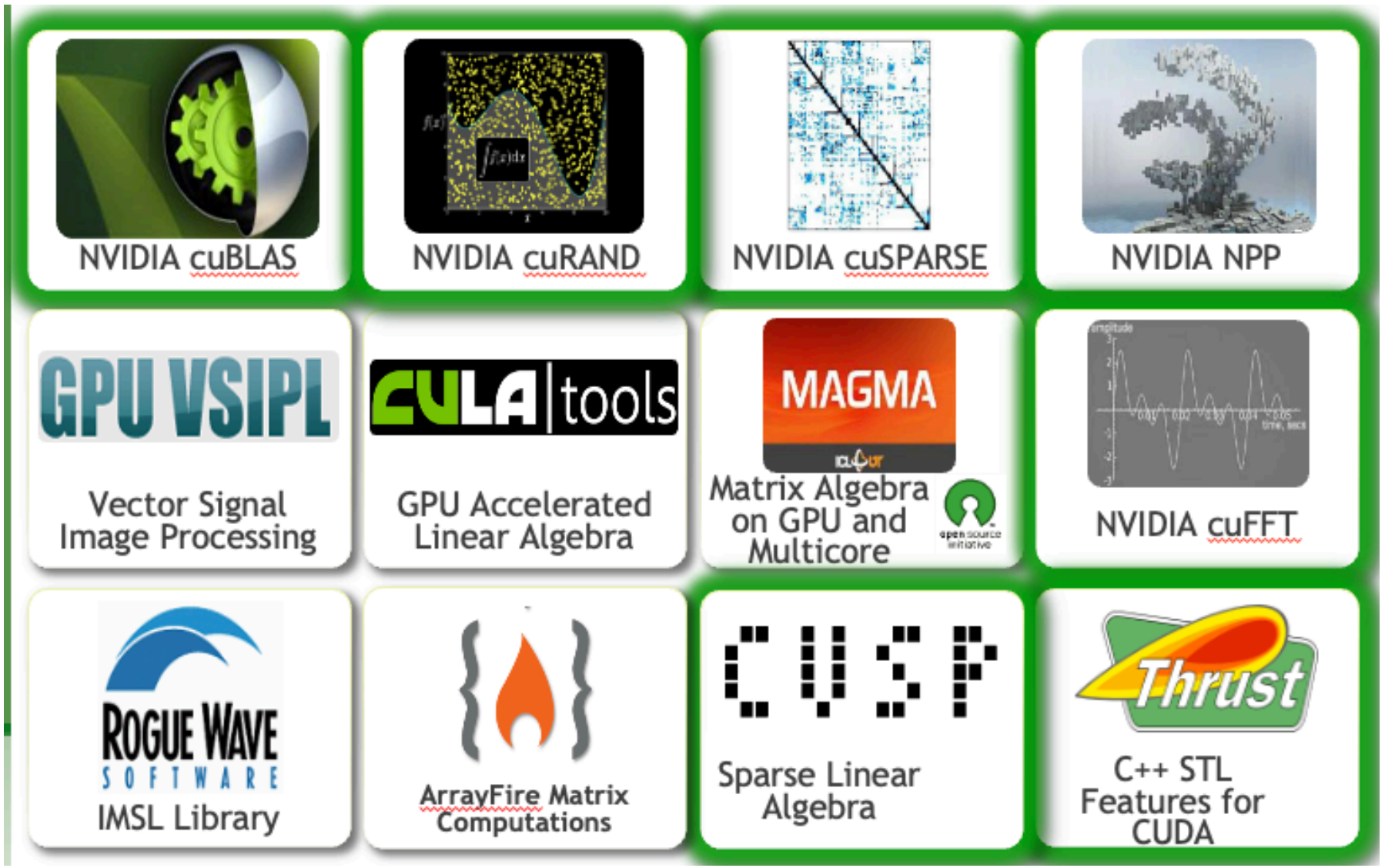
🔧 Compiling GPU Software

CUDA

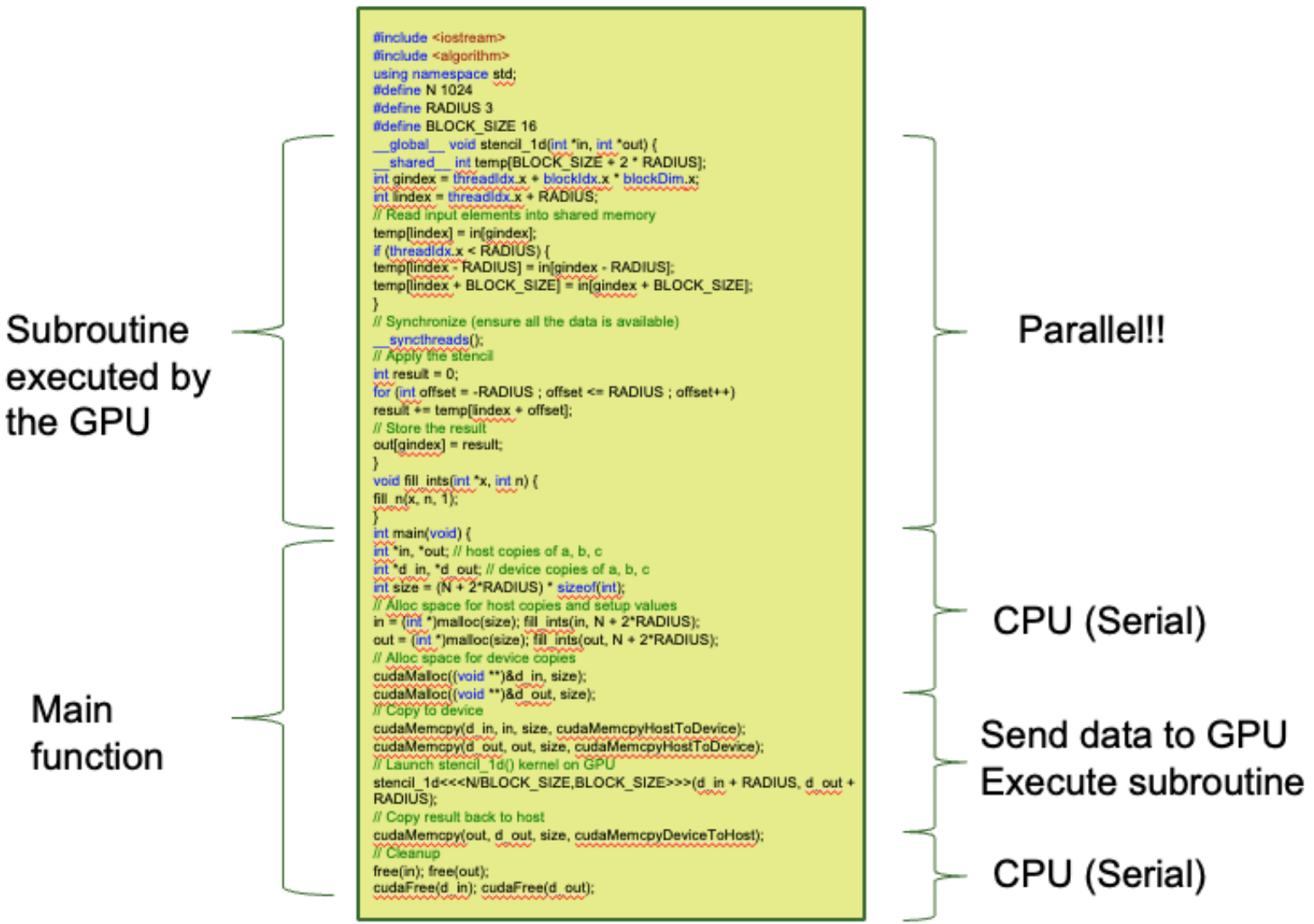
CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model from NVIDIA. It enables developers to write software that harnesses the power of GPUs for more than just graphics — expanding into high-performance computing and deep learning.

On Hoffman2, you can compile CUDA code by loading the `cuda` module. This prepares your environment with tools from the CUDA toolkit, which includes essential libraries and compilers for GPU code execution.

CUDA libraries



CUDA code example




```
#include <iostream>
#include <algorithm>
using namespace std;
#define N 1024
#define RADIUS 3
#define BLOCK_SIZE 16
__global__ void stencil_1d(int *in, int *out) {
__shared__ int temp[BLOCK_SIZE + 2 * RADIUS];
int gindex = threadIdx.x + blockIdx.x * blockDim.x;
int lindex = threadIdx.x + RADIUS;
// Read input elements into shared memory
temp[lindex] = in[gindex];
if (threadIdx.x < RADIUS) {
temp[lindex - RADIUS] = in[gindex - RADIUS];
temp[lindex + BLOCK_SIZE] = in[gindex + BLOCK_SIZE];
}
// Synchronize (ensure all the data is available)
__syncthreads();
// Apply the stencil
int result = 0;
for (int offset = -RADIUS ; offset <= RADIUS ; offset++)
result += temp[lindex + offset];
// Store the result
out[gindex] = result;
}
void fill_ints(int *x, int n) {
fill_n(x, n, 1);
}
int main(void) {
int *in, *out; // host copies of a, b, c
int *d_in, *d_out; // device copies of a, b, c
int size = (N + 2*RADIUS) * sizeof(int);
// Alloc space for host copies and setup values
in = (int *)malloc(size); fill_ints(in, N + 2*RADIUS);
out = (int *)malloc(size); fill_ints(out, N + 2*RADIUS);
// Alloc space for device copies
cudaMalloc((void **)&d_in, size);
cudaMalloc((void **)&d_out, size);
// Copy to device
cudaMemcpy(d_in, in, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_out, out, size, cudaMemcpyHostToDevice);
// Launch stencil_1d() kernel on GPU
stencil_1d<<<N/BLOCK_SIZE,BLOCK_SIZE>>>(d_in + RADIUS, d_out +
RADIUS);
// Copy result back to host
cudaMemcpy(out, d_out, size, cudaMemcpyDeviceToHost);
// Cleanup
free(in); free(out);
cudaFree(d_in); cudaFree(d_out);
```

🧪 CUDA code example

Here's a simple CUDA code example that performs matrix multiplication (1024x1024):

- Files are in the `MatrixMult` folder
  - `Matrix-cpu.cpp` contains CPU (serial) code
  - `Matrix-gpu.cu` contains the CUDA code
  - `MatrixMult.job` job submission file

Loading required modules

```
module load gcc/10.2.0
module load cuda/12.3
```

Compiling code

```
g++ -o Matrix-cpu Matrix-cpu.cpp
nvcc -o Matrix-gpu Matrix-gpu.cu
```

Submitting the job

```
qsub MatrixMult.job
```

GPU software

Be on the lookout for GPU optimized software for your research!

Other GPU platforms include:

- NVIDIA's HPC SDK (Software Developemnt Kit)
  - C/C++/Fortran compilers, Math libraries, and Open MPI

```
1  modules_lookup -m hpcsdk
```

- AMD ROCm (Radeon Open Compute)
  - For AMD GPUs

```
1  modules_lookup -m amd
```

Using Python/R for GPU Computing

GPUs for Python and R

There are several Python and R packages that use GPUs for varsious data-intensives tasks, like Machine Learning, Deep Learning, and large-scale data processing.

Python:

- TensorFlow: One of the most widely used libraries for machine learning and deep learning that supports GPUs for acceleration.
- PyTorch: A popular library for deep learning that features strong GPU acceleration and is favored for its flexibility and speed.
- cuPy: A library that provides GPU-accelerated equivalents to NumPy functions, facilitating easy transitions from CPU to GPU.
- RAPIDS: A suite of open-source software libraries built on CUDA-X AI, providing the ability to execute end-to-end data science and analytics pipelines entirely on GPUs.

R:

- gputools: Provides a variety of GPU-enabled functions, including matrix operations, solving linear equations, and hierarchical clustering.
- cudaBayesreg: Designed for Bayesian regression modeling on NVIDIA GPUs, using CUDA.
- gpuR: An R package that interfaces with both OpenCL and CUDA to allow R users to access GPU functions for accelerating matrix algebra and operations.
- Torch for R: An R machine learning framework based on PyTorch
- TensorFlow for R: An R interface to a Python build of TensorFlow

⭐ TensorFlow and PyTorch

Installing TensorFlow and PyTorch on Hoffman2 is straightforward using the Anaconda package manager. (Check out my Workshop on using Anaconda)

Create a new conda environmnet with CUDA tools.

```
mkdir -pv $SCRATCH/conda
module load anaconda3/2023.03
conda create -p $SCRATCH/conda/tf_torch_gpu python=3.10 s
conda activate $SCRATCH/conda/tf_torch_gpu
```

Install TensorFlow/PyTorch with GPU support and the NVIDIA libraries

```
pip3 install tensorrt-cu11 tensorrt-cu11-bindings tensorr
pip3 install tensorflow[and-cuda]==2.14
pip3 install torch torchvision torchaudio --index-url htt
```

Verify the TensorFlow installation. Will only work if you are on a GPU-enabled node.

```
# TensofFlow Test:
python -c "import tensorflow as tf; print('TensorFlow is

# PyTorch Test:
python -c "import torch; print('PyTorch is using:', ('GPU
```

👗 Fashion MNIST

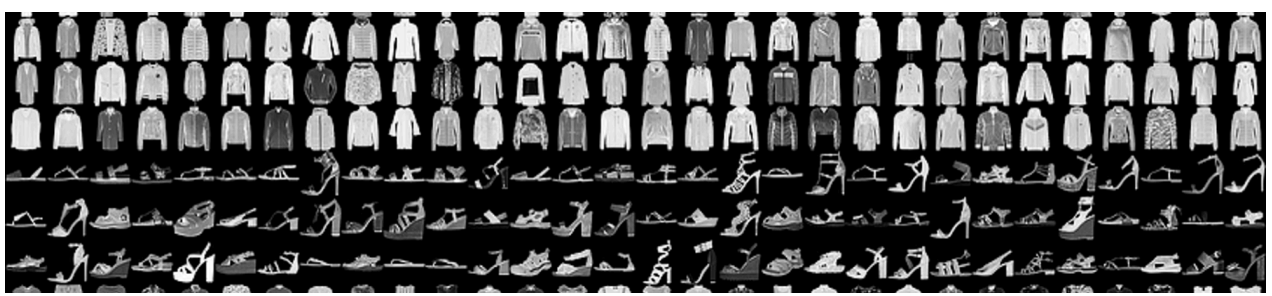
Explore machine learning with the "Fashion MNIST" dataset using TensorFlow:

Approach:

- We will use TensorFlow to train a Netural Net model for predicting fashion categories.

Dataset Overview:

- 📸 **Images:** 28x28 grayscale images of fashion products.
- 📊 **Categories:** 10, with 7,000 images per category.
- 🧮 **Total Images:** 70,000.



Runing Tensorflow

Now that we have TensorFlow installed, we can run some examples to test the GPU acceleration.

Files in the `TF-Torch` folder contain examples of using TensorFlow on Hoffman2.

Get a GPU node

```
qrsh -l h_data=40G,h_rt=1:00:00,gpu,A100,cuda=1
```

Set up your TensorFlow environment

```
module load anaconda3/2023.03
conda activate $SCRATCH/conda/tf_torch_gpu
```

Run CPU example

```
python minst-train-cpu.py
```

Run GPU example

```
python minst-train-gpu.py
```

This approach provides a hands-on way to see the difference in performance when using GPUs compared to CPUs for training machine learning models.

🧬 DNA classification with PyTorch

DNA Sequence Classification with PyTorch

- 🎯 **Objective:** Develop a model to classify DNA sequences.
- 🔬 **Gene Regions:** Segments of DNA containing codes for protein production.
- 🧪 **Dataset Creation:** Generate random DNA sequences labeled as ‘gene’ or ‘non-gene’.
- 🤖 **Model Development:** Use PyTorch to build a model predicting the presence of ‘gene’ regions.
- 🚀 **Leveraging GPUs:** Utilize the parallel processing power of GPUs for efficient training.

🏃 Running PyTorch

With PyTorch installed in the same Anaconda environment, we can now run the DNA classification example.

When running PyTorch on the GPU

```
1 device = torch.device('cuda' if torch.cuc
```

Force running PyTorch on the GPU

```
1 device = torch.device('cpu')
```

Run example

```
1 python dnatorch.py
```

Rapids for Genomic Data Analysis

We will use RAPIDS for genomic data analysis. RAPIDS is a popular platform to run data workflows, tasks, and manipulations, as well as, machine learning on GPUs.

We will

- Applying conditions to filter dataframes based on depth, quality, and allele frequency.
- Grouping data by chromosome and calculating mean statistics for depth, quality, and allele frequency.
- Speed comparison of these operations on GPU versus CPU.

🔨 Install Rapids

- RAPIDS: A suite of open-source software libraries and APIs built on CUDA to enable execution of end-to-end data science and analytics pipelines on GPUs.
- cuDF: Part of the RAPIDS ecosystem, cuDF is a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data.

Lets add Rapids to our environment

```
module load anaconda3/2023.03
conda create -p $SCRATCH/conda/myRapids -c rapidsai -c conda-forge -c nvidia  \
    rapids=24.04 python=3.10 cuda-version=11.8 -y
conda activate $SCRATCH/conda/myRapids
```

🔍 Running Rapids

Navigate GPU-accelerated data manipulation with cuDF:

Files in the `rapids` folder

- `rapids_analysis-gpu.py` - GPU version
- `rapids_analysis-cpu.py` - CPU version

The `rapid_analysis.job` will submit the job to the Hoffman2 cluster.

In this file, the line `#$ -l gpu,V100` will submit this job to the V100 GPU nodes.

Running Rapids

```
1  qsub rapids_analysis.job
```

💧 H2O.ai ML Example

Explore machine learning with H2O.ai using the Combined Cycle Power Plant dataset:

- H2O.ai is an open-source platform for machine learning and AI.
- We will work through an example from H2o-tutorials.
- ⭐ **Objective:** Predict the energy output of a Power Plant using temperature, pressure, humidity, and exhaust vacuum values.
- This example, we will use the R API, but H2O.ai has a Python API as well
- We will use XGBoost, a popular gradient boosting algorithm, to train the model.

🚀 Instaling H2O.ai

We will use R and install the H2O.ai package to run the example.

- Setting up the environment

```
module load cuda/11.8
module load gcc/10.2.0
module load R/4.3.0
```

- Installing H2O.ai in R

```
mkdir -pv $R_LIBS_USER
R -q -e 'install.packages(c("RCurl", "jsonlite"), repos = "https://cran.rstudio.com")'
R -q -e 'install.packages("h2o", type="source", repos=(c("http://h2o-release.s3.amazonaws
```

🏃 Running H2O.ai

In the `h2oai` folder, the `h2oaiXGBoost.R` script the code to run XGBoost on the Combined Cycle Power Plant dataset.

The `h2oML-gpu.job` file will submit the job to a **GPU** node.

```
1  qsub h2oML-gpu.job
```

The `h2oML-cpu.job` file will submit the job to a **CPU** node.

```
1  qsub h2oML-cpu.job
```

The H2O.ai functions will automatically detect the GPU and use it for training.

🎉 Wrap up

Hoffman2 has the resources and tools to help you leverage the power of GPUs for your research. ⭐

Main Takeaways:

- Use `-l gpu` option to reserve a GPU node
- Compile GPU optimize code with CUDA
- Understand how to use your software can efficiently use GPUs
- Use Python and R packages for GPU computing

👏 Thanks for Joining! ❤️

Questions? Comments?

- 📧 cpeterson@oarc.ucla.edu
- 🗓️ Look at for more Hoffman2 workshops

RPylab demo

RPylab

This is an experimental setup that I made that can run both RStudio and Jupyter on Hoffman2.

This environment has many loaded packages (mostly data science related)

A lot of these packages are optimized with Intel's OneAPI with MKL and GPU support

This is built using Docker and can be ran on any system with Apptainer

```
1  apptainer pull docker://ghcr.io/charliecpete
```

This is a pretty large container so it may some time to download (I already have it on Hoffman2). I'm working on some minimal versions without the many packages and well as non-GPU versions.

> ⚠ **Warning**

Running RStudio

This RStudio has TensorFlow and Torch for R installed with GPU support and MKL as well as many data science related R packages.

You can also run Python within this. Same Python as with Jupyter.

- Making tmp files for Rstudio

```
mkdir -pv $SCRATCH/rstudiotmp/var/lib
mkdir -pv $SCRATCH/rstudiotmp/var/run
mkdir -pv $SCRATCH/rstudiotmp/tmp
```

- Start RStudio

```
apptainer run --nv \
      -B $SCRATCH/rstudiotmp/var/lib:/var/lib/rstudio-server \
      -B $SCRATCH/rstudiotmp/var/run:/var/run/rstudio-server \
      -B $SCRATCH/rstudiotmp/tmp:/tmp \
         $H2_CONTAINER_LOC/rpylab_rpylab-R4.3.3-python-3.10.10-oneapi-gpu.sif rstudio
```

- Port forward

```
ssh -L 8787:COMPUTENODE:8787 USERNAME@hoffman2.idre.ucla.edu
```

- Open web browser

```
http://localhost:8787
```

Running Jupyter

This Jupyter also has TensorFlow and PyTorch installed with GPU support and MKL. There is also a R kernel in this Jupyter (same R from RStudio).

- Start Jupyter

```
apptainer run --nv \
          $H2_CONTAINER_LOC/rpylab_rpylab-R4.3.3-python-3.10.10-oneapi-gpu.sif jupyter
```

- Port forward

```
ssh -L 8888:COMPUTENODE:8888 USERNAME@hoffman2.idre.ucla.edu
```

- Open web browser

```
http://localhost:8888
```

Non-interactive R/Python

- R

```
apptainer run --nv \
        $H2_CONTAINER_LOC/rpylab_rpylab-R4.3.3-python-3.10.10-oneapi-gpu.sif Rscript myscript.
```

- Python

```
apptainer run --nv \
        $H2_CONTAINER_LOC/rpylab_rpylab-R4.3.3-python-3.10.10-oneapi-gpu.sif python myscript.p
```