Scientific Applications in the NESAP Program

Muaaz Awan, Jaideep Pathak, Neil Mehta and Raphaël Prat

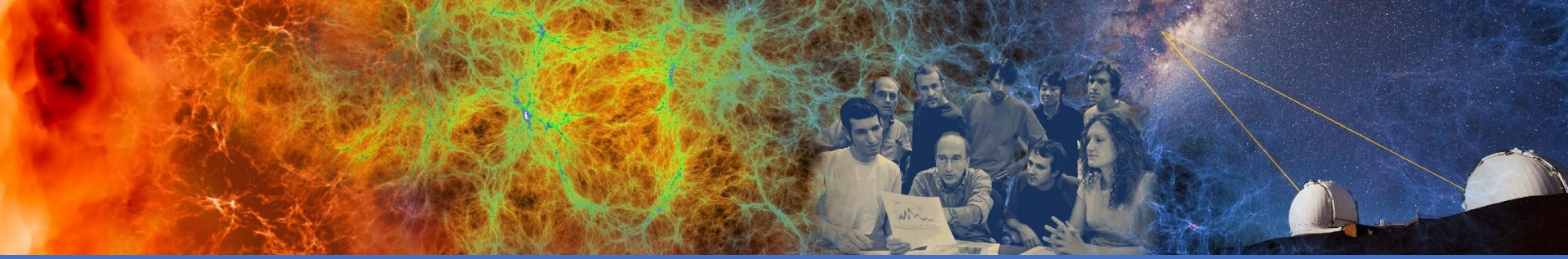
NESAP: Application Readiness

NESAP is NERSC's Application Readiness Program for preparing our workload for new systems.

Strategy: Partner with application development teams and vendors to port & optimize key applications of importance to the Office of Science. Share lessons learned with with NERSC community via documentation and training.

Resource Available to Teams: NERSC Staff technical liaisons, performance postdocs, access to vendor application engineers, hackathons, early access to hardware (GPU nodes on Cori and Perlmutter)

This talk: We will discuss insights from our collaboration with teams working with metagenome analysis, computational fluid dynamics, Machine learning assisted Molecular Dynamics and Climate Science



ExaBiom Project

Muaaz Awan

Microbiomes

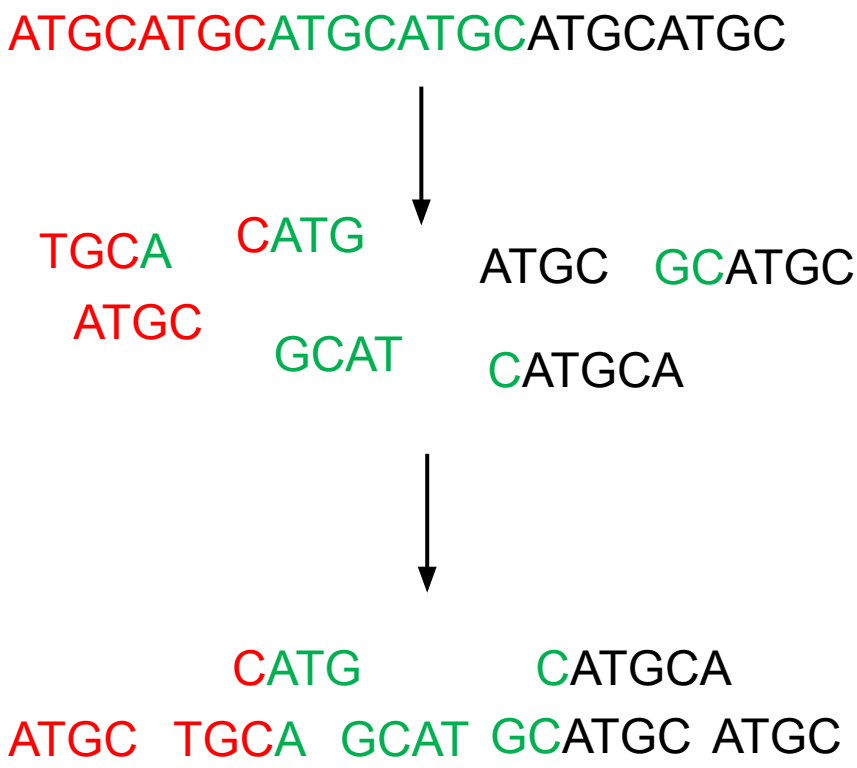
- **Microbes:** single cell organism, e.g. viruses, bacteria
- **Microbiomes:** communities of microbial species living in our environment
- **Metagenomics:** collective genome sequencing of these communities

Genome Assembly 101

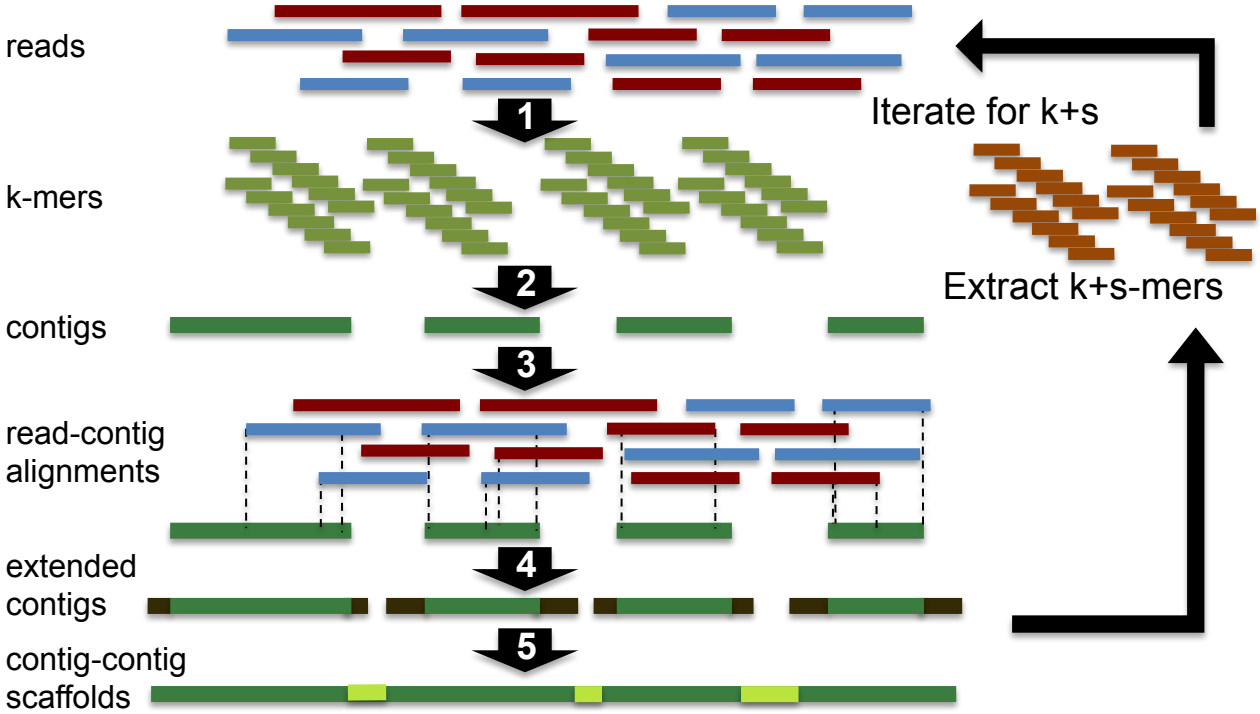
Genome of an organism is sampled using sequencing machines in the form of reads.

Objective is to assemble the pieces of genome back into their correct places *denovo.*

Like solving a puzzle, without a reference!



MetaHipMer assembly for short reads (UPC++)



1) K-mer Analysis
K-mer histogram

GPU

2) Contig Generation
Walk k-mer graph

3) Alignment
Align reads to contigs

GPU

4) Local Assembly
Extend ends of contigs

GPU

5) Scaffolding
Walk contig graph (iterate)

Actual pipeline is more complex, simplified for purpose of presentation

Algorithmic Motifs

Short Read Assembly:

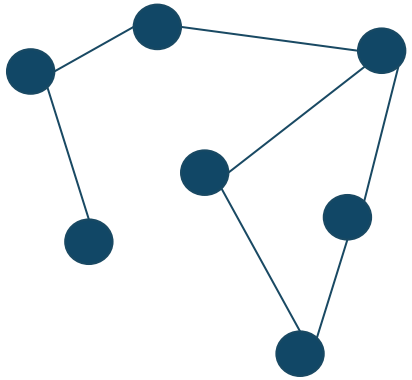
- Dynamic Programming Algorithms.
- Distributed and local Graph Traversals.
- Distributed and local hash tables.

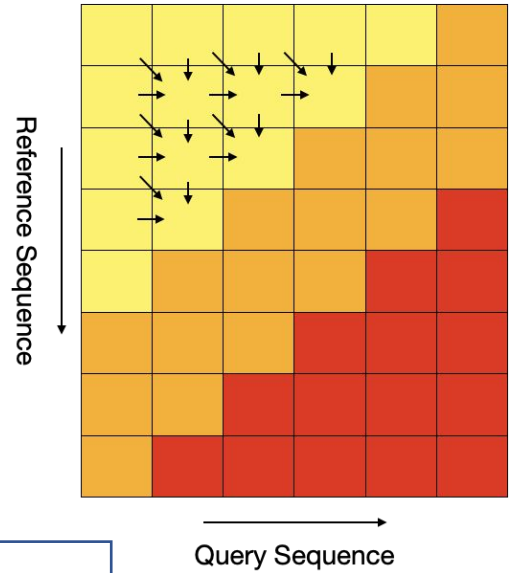
Long Read Assembly:

- Dynamic Programming Algorithms.
- Sparse matrix multiplication.
- Distributed hash tables.

Protein Similarity and Clustering:

- Dynamic Programming Algorithms.
- Sparse Matrix Multiplication.





ATGCATG

ATGCATG

ATGCATG

kmers	Ext
TGCA	T
GCAT	G
CATG	C

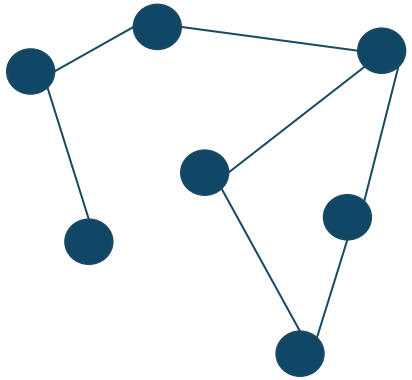
Challenges on GPUs

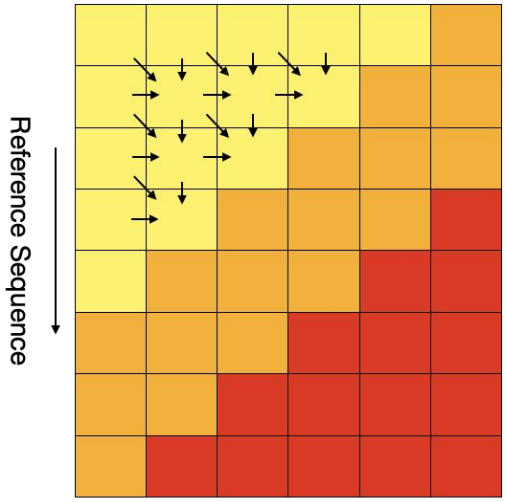
Ideal for GPUs:

- Localized and predictable memory access pattern.
- Alot of computations per each memory access.
- Equal amount of work can be distributed across threads.

What we have:

- Random or along diagonals memory access pattern.
- Integer only computations bound by memory bandwidth.
- Non deterministic amount of work.
- Varying or limited parallelism (DP and graph algorithms).





ATGCATG

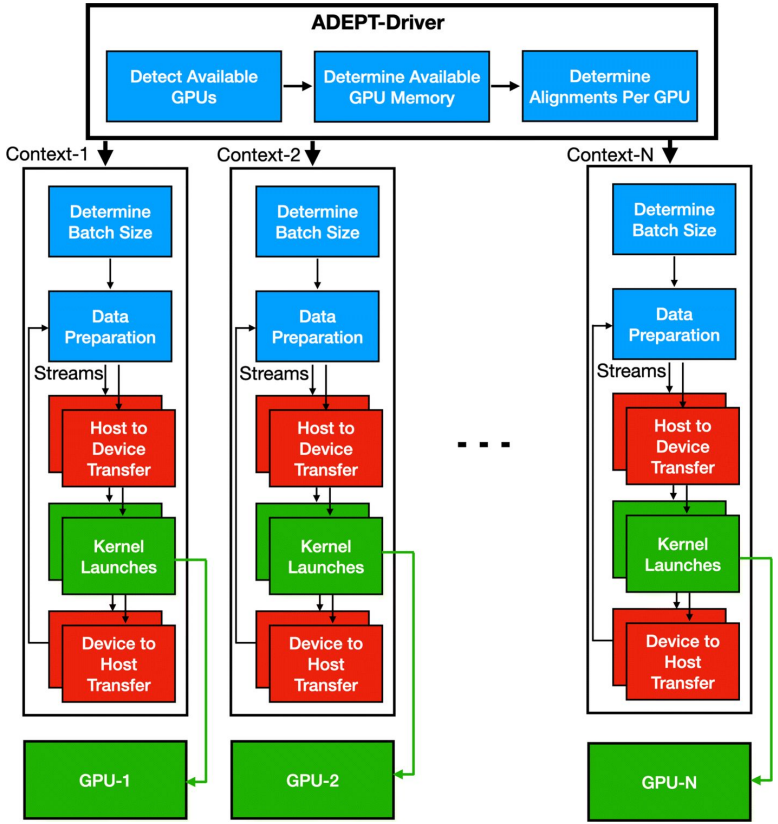
ATGCATG

ATGCATG

kmers	Ext
TGCA	T
GCAT	G
CATG	C

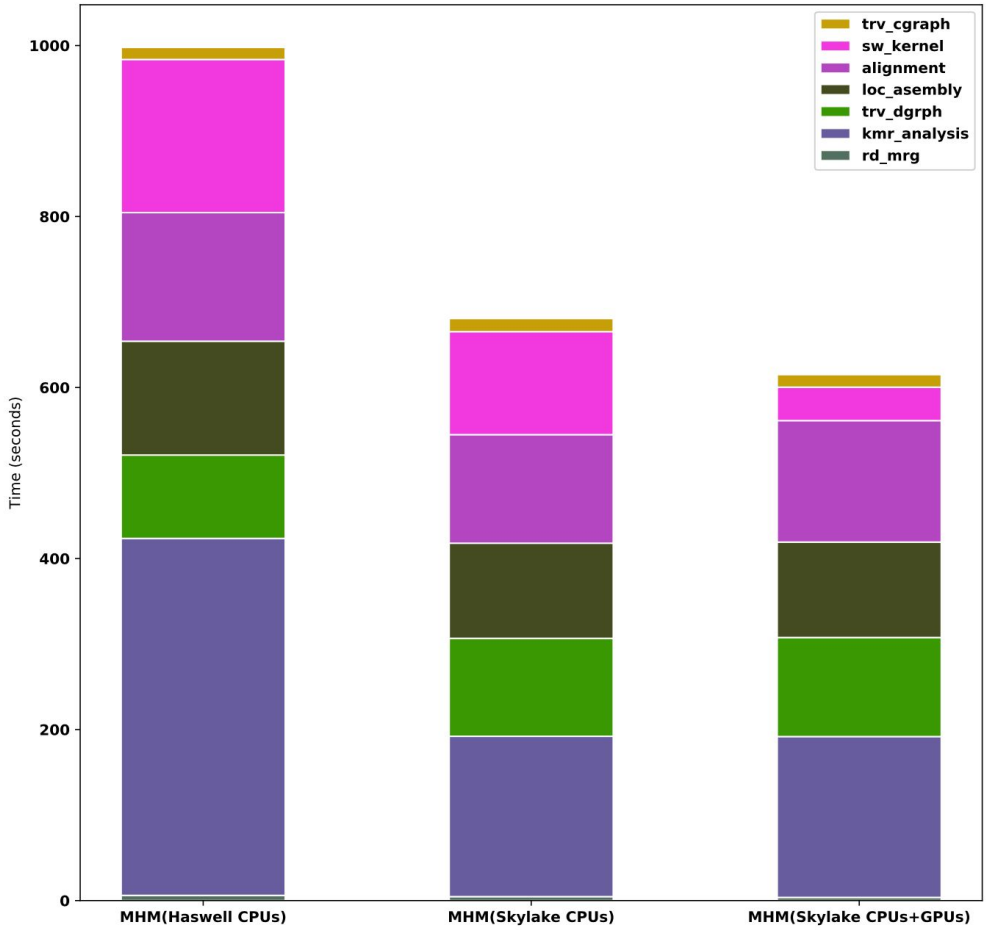
*ADEPT: A GPU accelerated library for sequence alignment

- ADEPT is domain independent i.e. performs equally well for protein and DNA sequences.
- Has been optimized for GPU architectures.
- Can scale across multiple GPUs.
- Easy to use, effectively drop in replacement for CPU libraries.

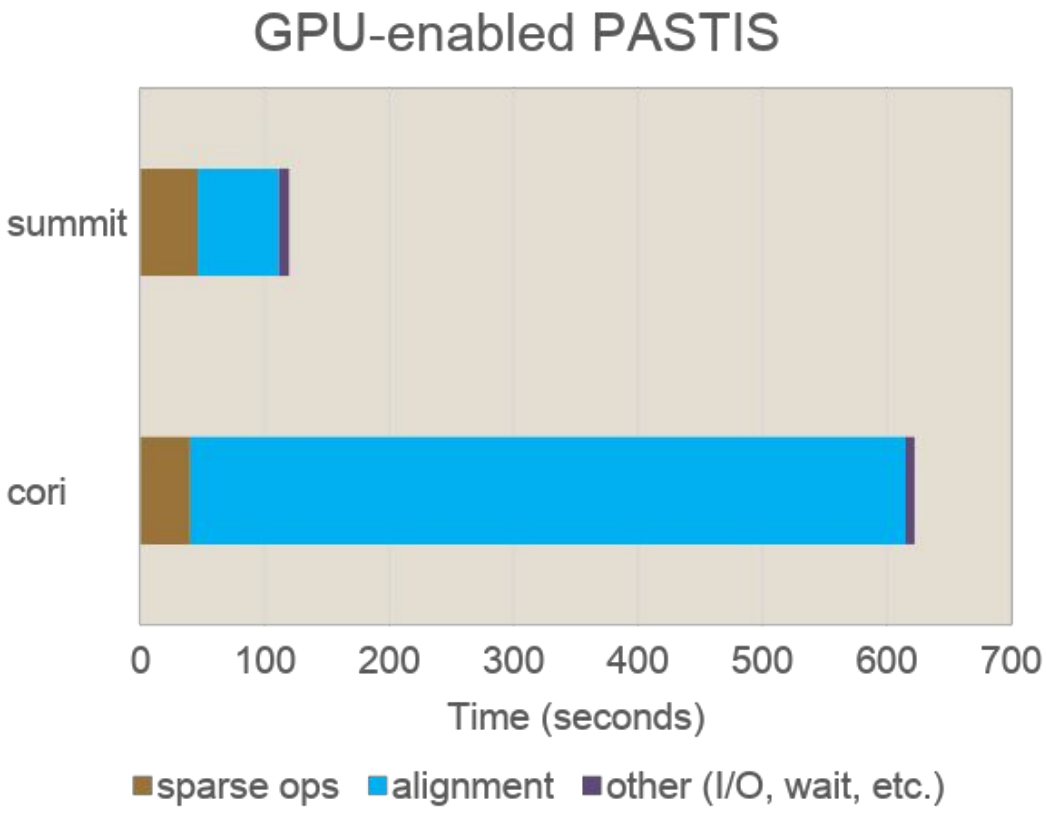


*Awan, Muaaz G., et al. "ADEPT: a domain independent sequence alignment strategy for gpu architectures." *BMC bioinformatics* 21.1 (2020): 1-29.

ADEPT in MetaHipMer



ADEPT in PASTIS



Dataset size: 5 million protein sequences

9.9 billion candidate alignments

~853 Million alignments

Protein similarity graph

- 5 Million nodes
- 64.6 million edges

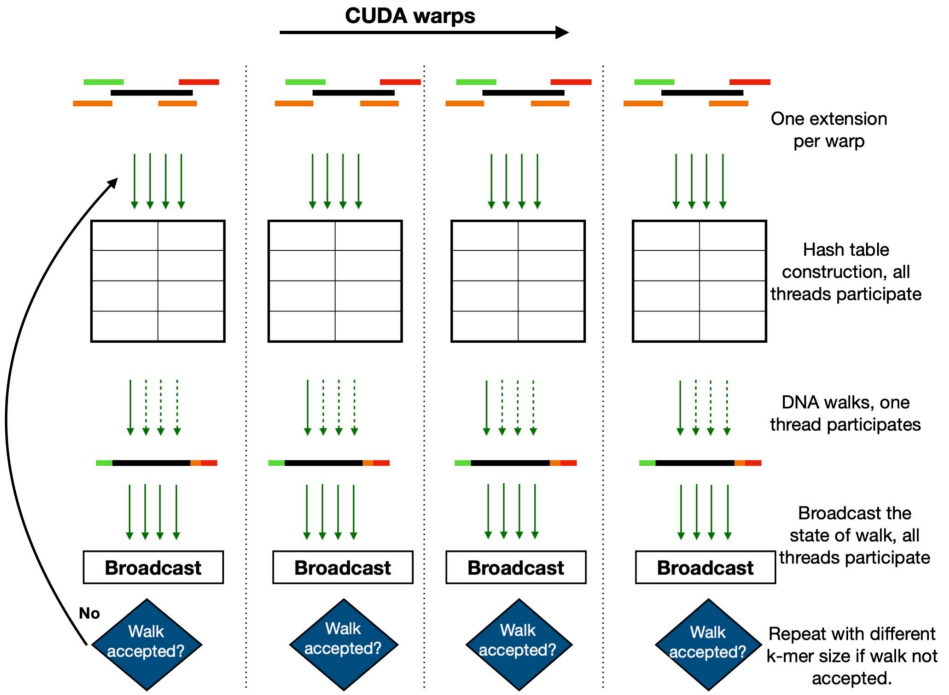
100 nodes of NERSC Cori and Summit

- ADEPT
- SeqAn
- **Alignment**: 5.2x speedup

Thanks to Oguz for this slide

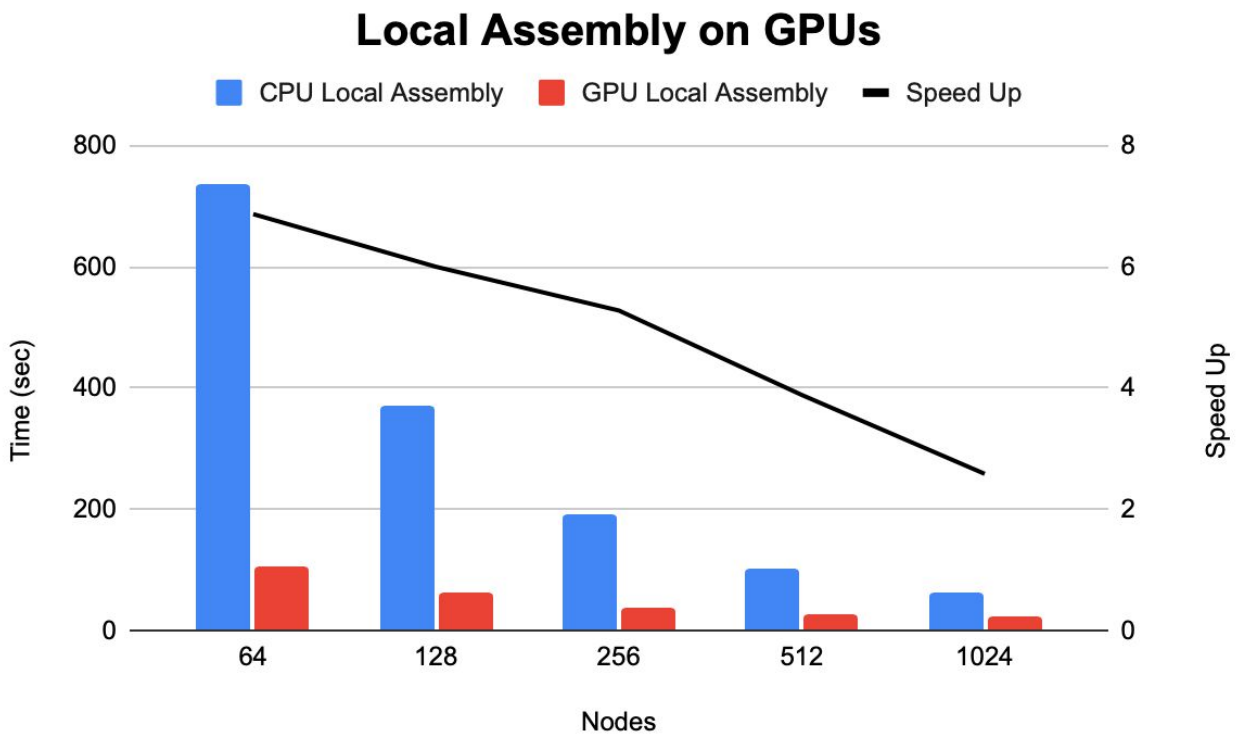
*GPU Accelerated Local Assembly

- Static hash tables were implemented on GPUs.
- Binning methods were used for achieving load balancing across GPU warps.
- Per warp hash tables were implemented to achieve faster performance.

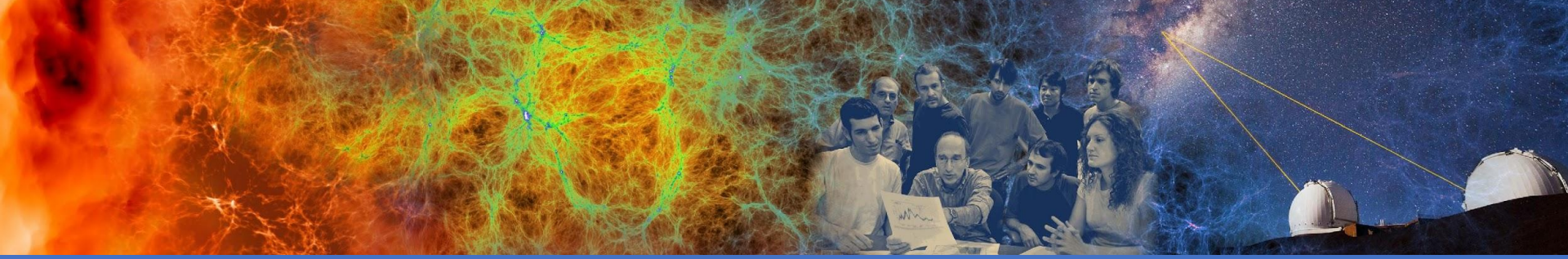


*Awan, Muaaz G., et al. "Accelerating large scale de novo metagenome assembly using GPUs." Appearing in SC21

GPU Accelerated Local Assembly



Thank you



CFD and Climate Science

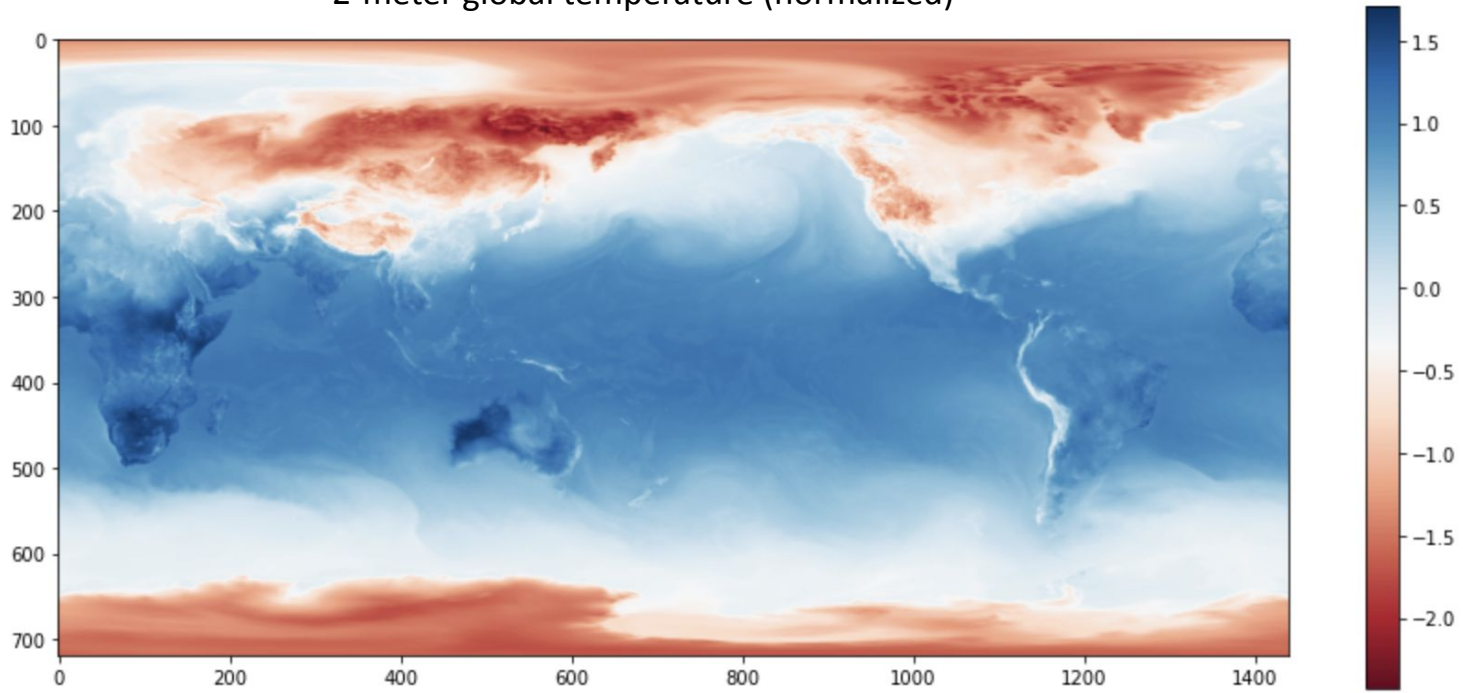
Jaideep Pathak

Introduction

Simulation of turbulent flows at high Reynolds number is a computationally challenging task relevant to a large number of engineering and scientific applications in diverse fields such as climate science, aerodynamics, and combustion.

Example Application: Numerical Weather Prediction

2-meter global temperature (normalized)



BERKELEY LAB

NERSC

Turbulent flows are typically modeled by the Navier-Stokes equations. Direct Numerical Simulation (DNS) of the Navier-Stokes equations with sufficient numerical resolution to capture all the relevant scales of the turbulent motions can incur prohibitive computational expense.

Consider a specific application: Short-term prediction of the state of a fluid dynamical system.

Prediction is difficult due to the mathematically chaotic nature of turbulent fluid flows which limits the predictability horizon.

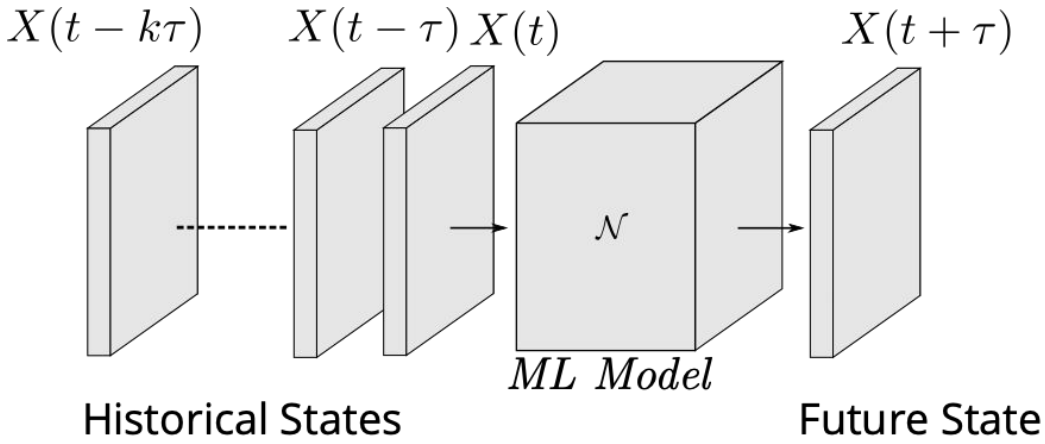
Nonetheless, short-term prediction of fluid flows is important for applications such as weather prediction.

Purely Data-Driven Forecasts?

Purely ML-based models attempt to construct a mapping between the history of the flow fields to the future.

The ML model has to learn a time-stepping function.

Such sequence ML models can be difficult to train at scale.



Purely Data-Driven Forecasts?

The computational cost of training pure ML fluid dynamics models remains prohibitively high for all but the simplest of problems.

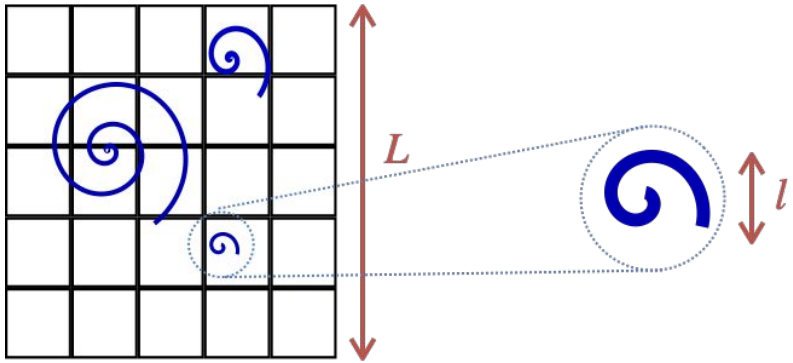
For instance, there have been a number of impressive advances in the development of data-driven models of the Earth's atmosphere but these models do not currently match the accuracy of state-of-the-art physics based General Circulation Models.

Hybrid ML-PDE approach

Data availability and computational resource constraints thus warrant the development of a hybrid approach where a physics-based numerical solver works in tandem with a coupled ML architecture, thus leveraging the strength of each approach while maintaining a reasonable computational cost.

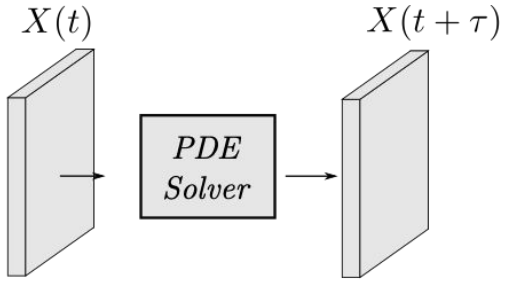
Resolution and Computational Cost

The computational cost of a simulation for a given problem is directly related to the resolution requirements, which are in turn determined on one side by the Kolmogorov length scale, and on the other side, by the size of the system under study.

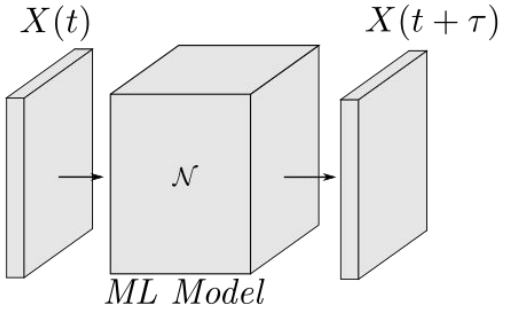


Hybrid ML-PDE approach

Purely Physics-Based Model



Purely Data-Driven Model



Is there a way to create a hybrid model?

Resolution and Computational Cost

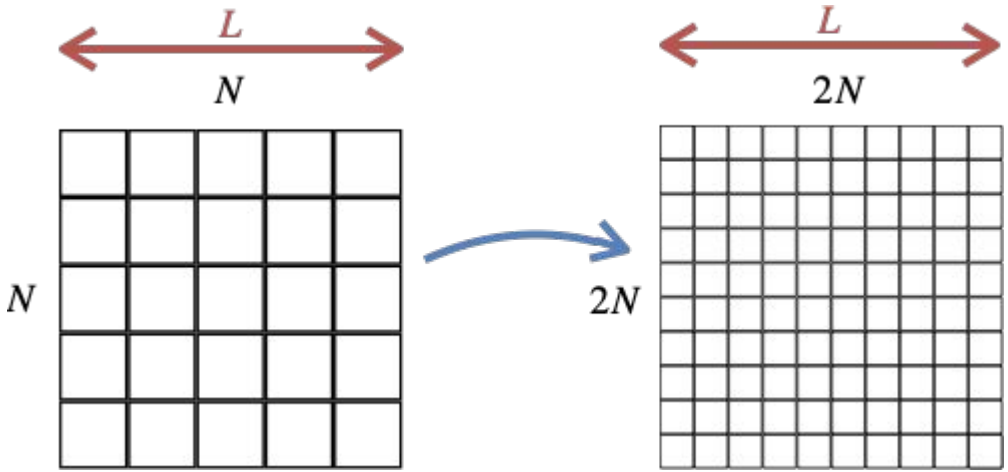
Let us look at some back-of-the-envelope simplified estimates of a the computational cost of a simulation with respect to the resolution.

Computational cost for a single PDE time step scales roughly as the number of grid cells.

Due to the Courant-Friedrichs-Lewy (CFL) condition, the largest possible time-step of a PDE solver roughly scales linearly as the size of the basic grid-cell.

Resolution and Computational Cost

Putting these constraints together, one can obtain a rough scaling estimate:



Doubling the resolution increases the overall cost of a simulation by a factor of 2^{d+1}

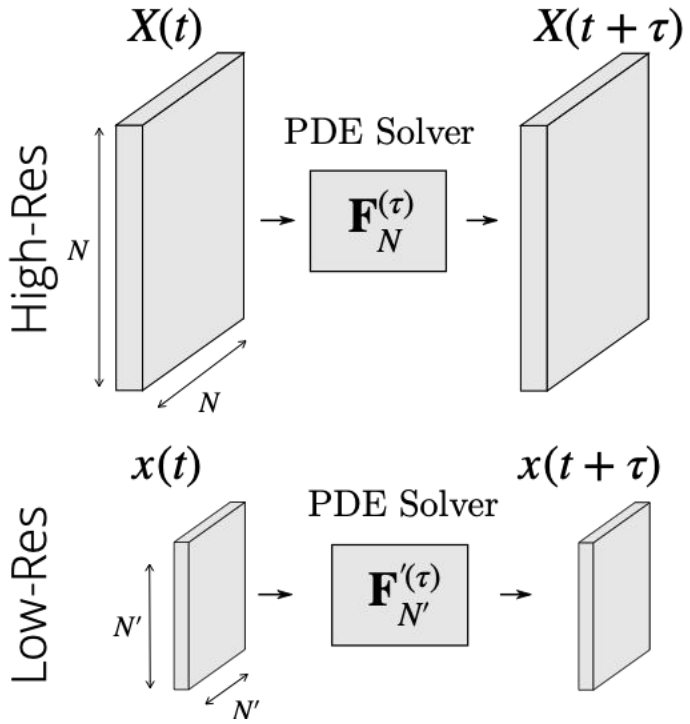
Resolution and Computational Cost

What we've discussed so far tells us that one way to accelerate a simulation is to run it on a coarse grid which does not resolve the fields fully down to the Kolmogorov scale.

However, this results in an imprecise/incomplete simulation.

The key idea we explore is to use ML to correct an under-resolved simulation.

ML-PDE



Consider a short interval of time τ

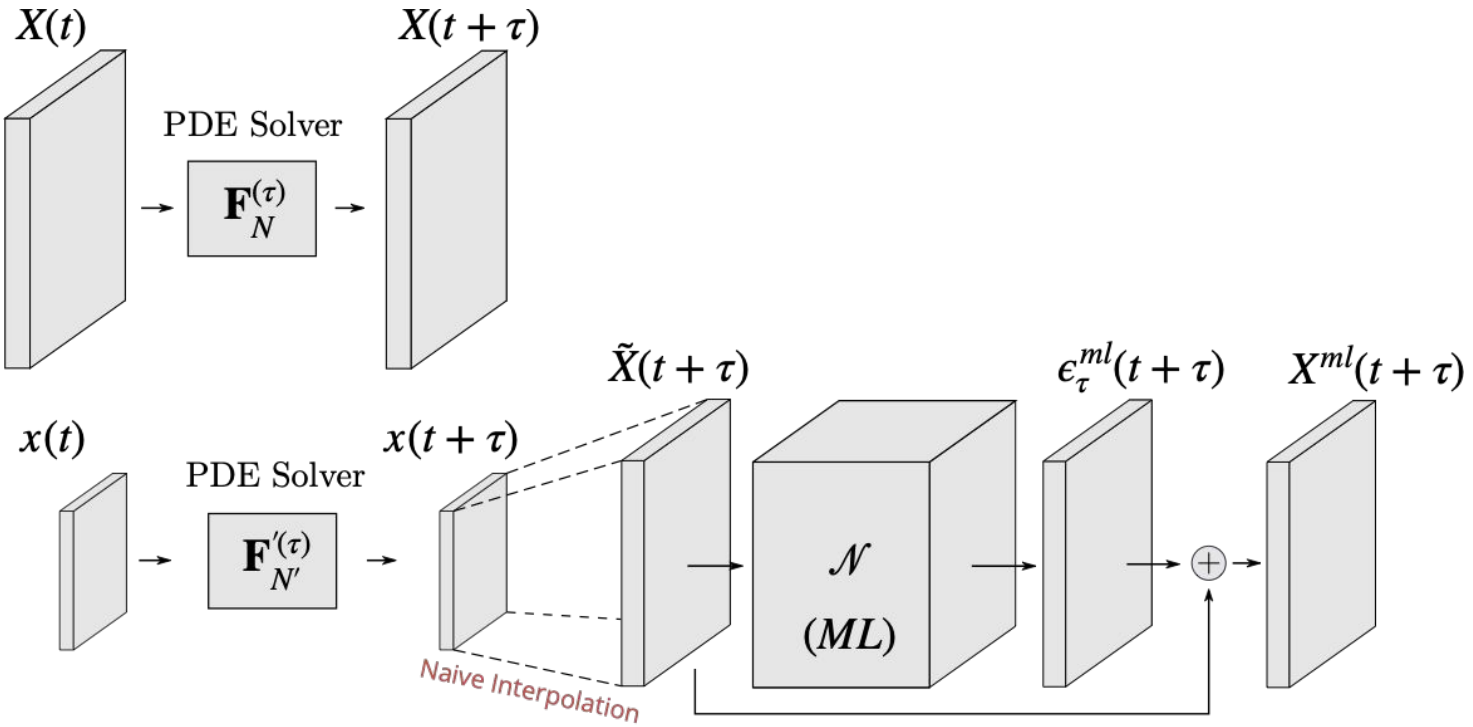
High-Resolution PDE time-stepping
Accurate, Expensive

Low-Resolution PDE time-stepping
Imprecise, Cheap

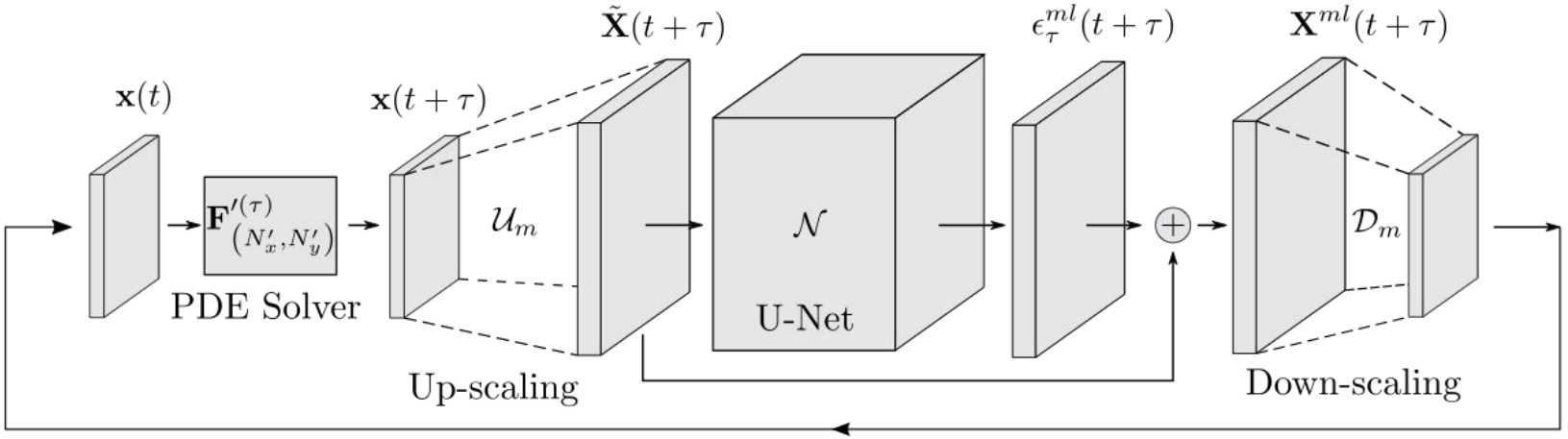
BERKELEY LAB

NERSC

ML-PDE


$X(t)$
$X(t+\tau)$
PDE Solver
$\mathbf{F}_N^{(\tau)}$
$\tilde{X}(t+\tau)$
$\epsilon_\tau^{ml}(t+\tau)$
$X^{ml}(t+\tau)$
$x(t)$
PDE Solver
$\mathbf{F}_{N'}^{\prime(\tau)}$
$x(t+\tau)$
$\mathcal{N}$
(ML)
Naive Interpolation

ML-PDE Hybrid Architecture

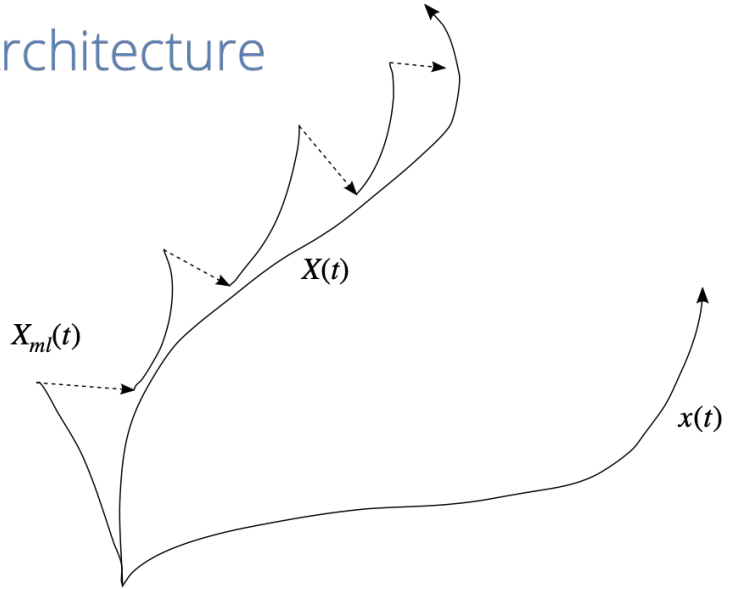


ML-PDE

ML-PDE Hybrid Architecture

Our architecture is more sophisticated than simply super-resolving a low resolution trajectory.

We nudge the low-resolution trajectory towards the correct high-resolution trajectory by correcting the low-resolution simulation at short intervals of time.

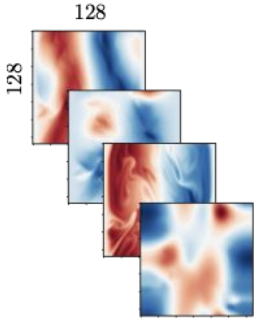


ML-PDE

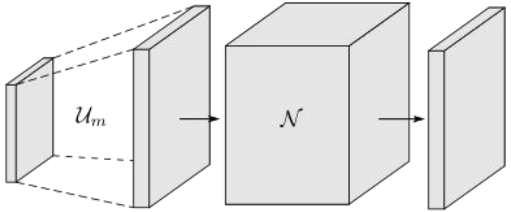
We built an initial prototype using Dedalus, a python based CFD solver and tested it on a Rayleigh-Benard Convection (RBC) system in the moderate-Reynolds number regime $(\textbf{Rayleigh Number} = 10^9)$.

Inputs: Low-Resolution fields

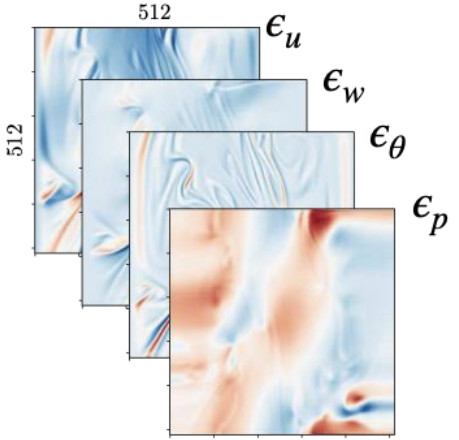
Horizontal velocity
Vertical velocity
Temperature
Pressure



$4 \times 128 \times 128$



Outputs: Model Error Corrections at high-resolution



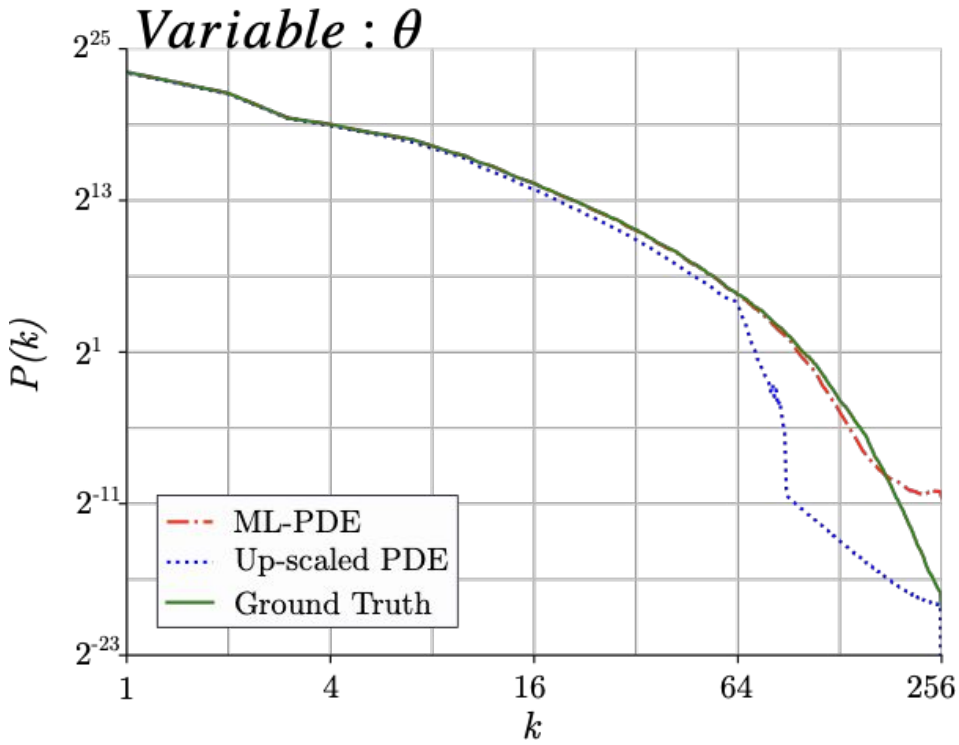
$4 \times 512 \times 512$

BERKELEY LAB

NERSC

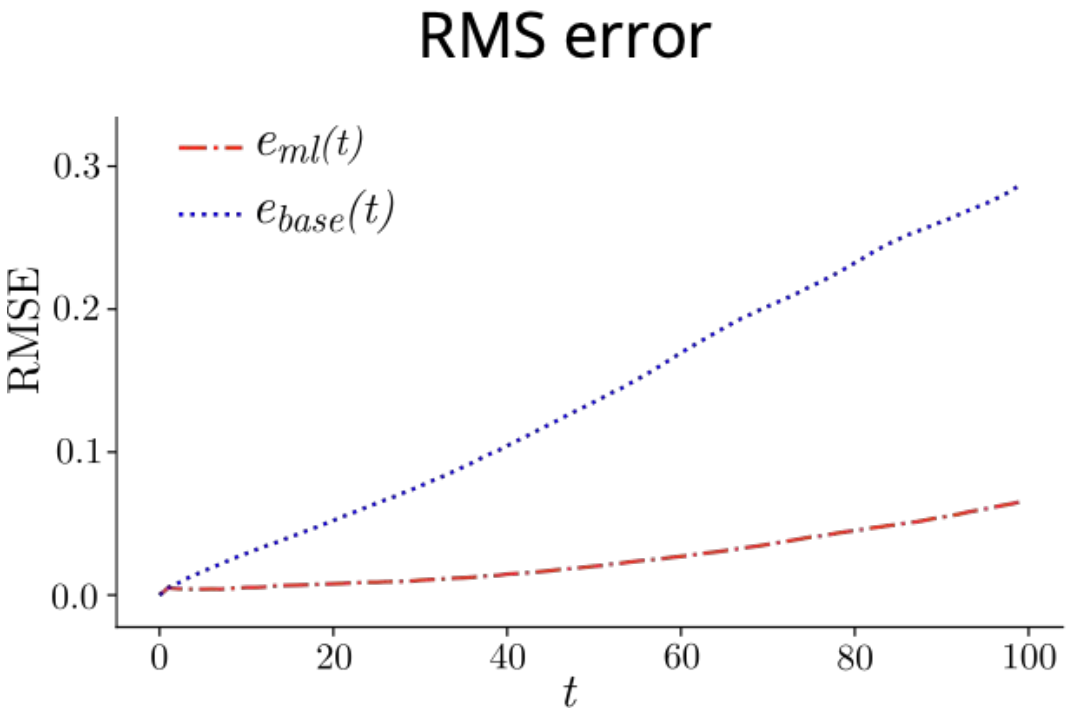
ML-PDE

After running the simulation for 100 steps ~2 eddy turnover times, we see that our architecture captures the high-res spectrum very well.



ML-PDE

We also compare the RMS error curves between the ML-PDE solution and the baseline Low-Res PDE solution (averaged over 20 initial conditions).

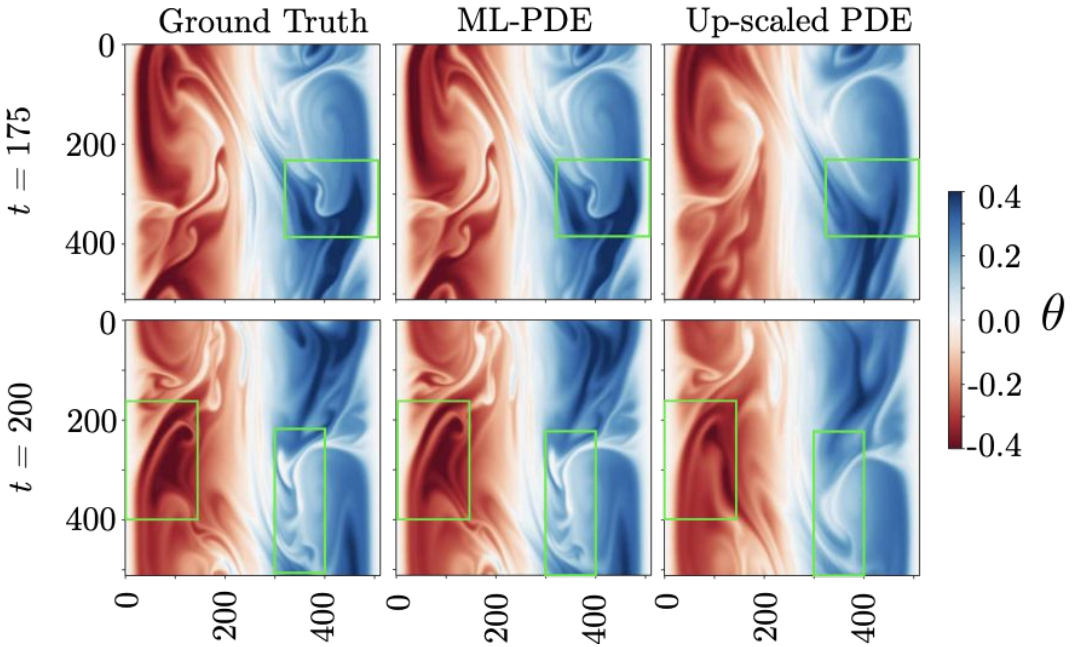


BERKELEY LAB

NERSC

ML-PDE

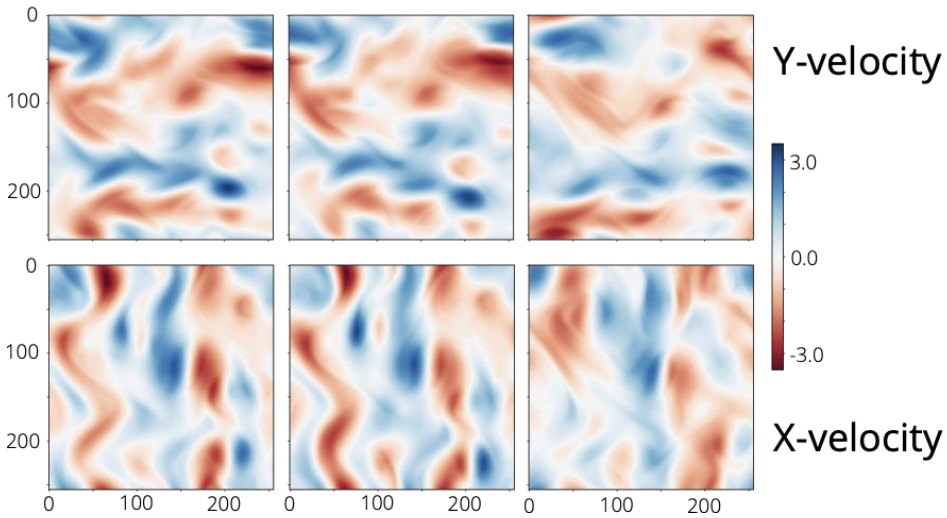
Temperature (θ) channel of the RBC simulation with 3 different simulators:

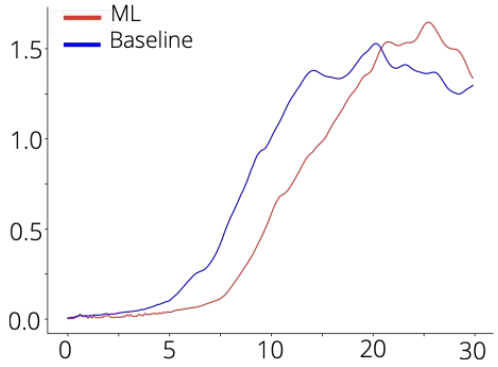


ML-PDE

2D-Kolmogorov Flow with forcing

$t = 10$ model time units





BERKELEY LAB

NERSC

Conclusions

The present work demonstrates an example flow simulation computed on a coarse mesh that is enhanced using a DL model to populate the finer scales that are normally available only by increasing resolution, and expense, of the simulation.

This technique also introduces a correction to the model errors resulting from the coarse mesh simulation.

In contrast to a post-facto Super-Resolution approach applied to artificially coarsened simulation data, we present a general technique to enhance PDE simulation data that is generated by the flow solver at low resolution.

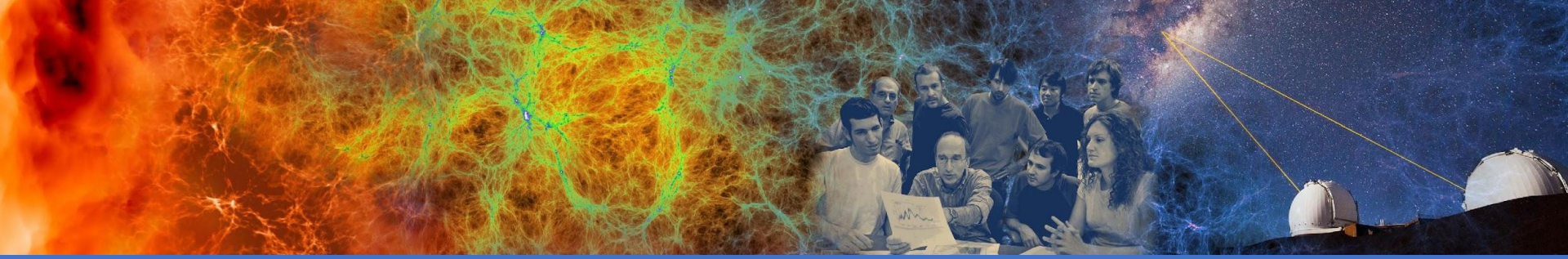
Conclusions

We envision many potential applications for this work. For Numerical Weather Prediction, one may be able to run a low resolution General Circulation Model and obtain high-resolution forecasts using our proposed ML architecture.

Large scale Direct Numerical Simulation of fluid flows for aerodynamics and combustion research might be possible.

ML will not replace physics-based CFD but will augment it and hopefully allow us to expand the horizons of numerical CFD.

ML-assisted DNS models can be a complementary approach to techniques like Large Eddy Simulation (LES) and Reynolds Averaged Navier-Stokes (RANS).

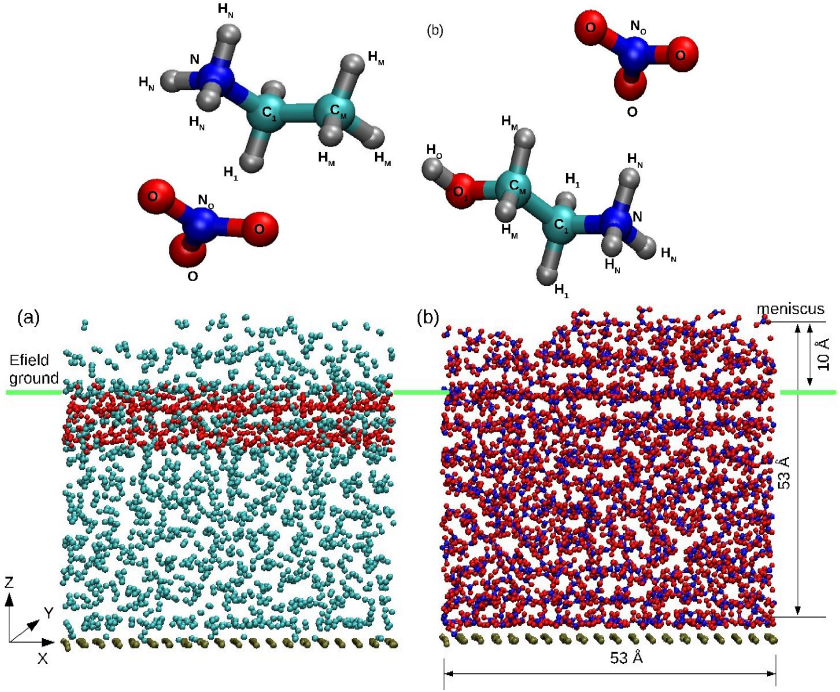


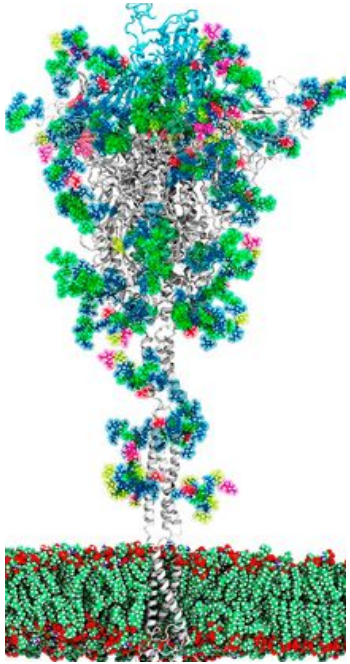
Molecular Dynamics

Neil Mehta

Introduction to Molecular Dynamics (MD)

Molecular Dynamics (MD) is a powerful tool to understand interactions of atoms and molecules.





ACS Cent. Sci. 2020, 6, 10

Applications:

- Material science
- Biomolecule research
- Aerospace engineering
- Ionic liquids
- Surface interactions
- Many more!

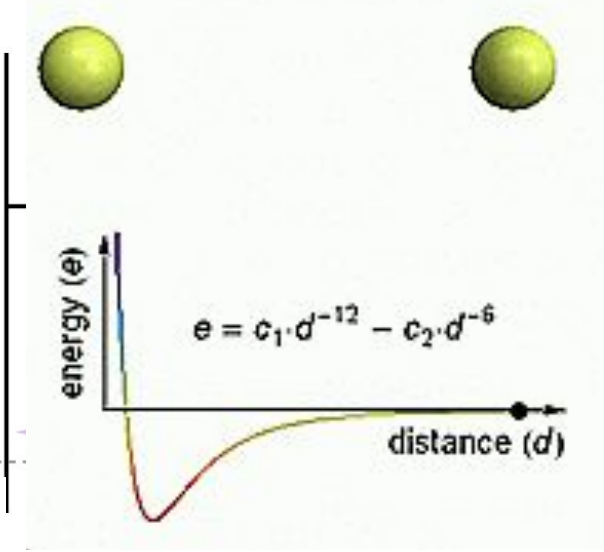
Summary of MD terminology

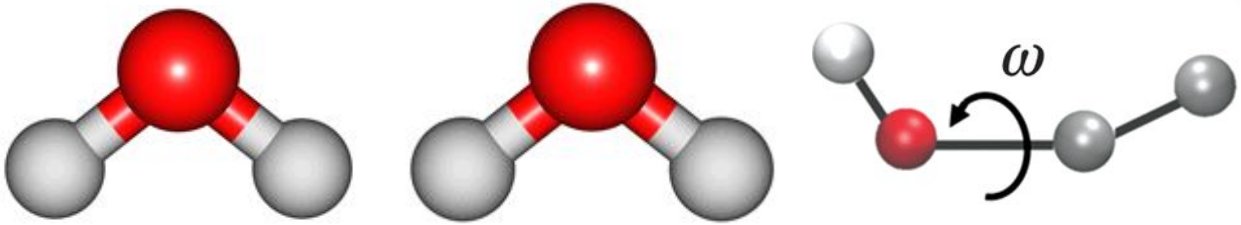
Interactions in **MD** are divided into **kinetic** and **potential** energy

$$E_{MD} = E_{kinetic} + E_{potential}$$

Potential energy interactions governed by an **equation** or **series of equations** known as **MD potentials**

$$E_{potential} = E_{Covalent\text{-}bond} + E_{Angle} + E_{Dihedral} + E_{vdW} + E$$





https://scilearn.sydney.edu.au/OrganicSpectroscopy/?type=Infrared
https://pubs.rsc.org/en/content/chapterhtml/2017/bk9781782627005-00001?isbn=978-1-78262-700-5
https://gifsdefisica.com/2018/12/18/potencial-de-lennard-jones/
http://ch301.cm.utexas.edu/imfs/mo/mo-theory-all.php

Motivation for efficient use of MD potentials

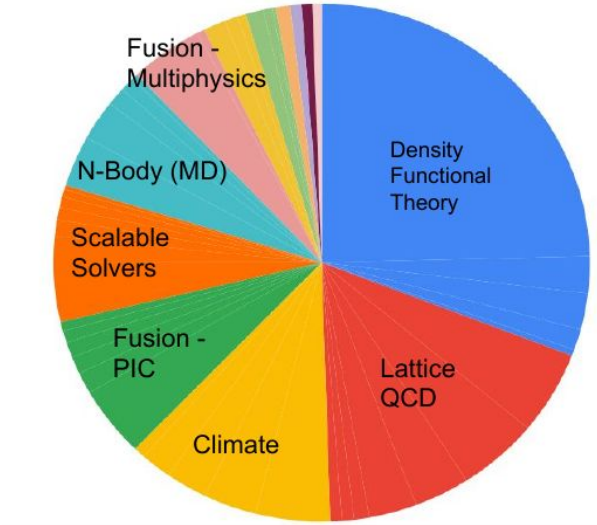
- **LAMMPS** developed under the auspices of **DOE** and multi-lab collaboration
- Beneficiary of **Exa-scale Computing Project (ECP).** Under ECP umbrella project **Exaalt**

For Users:

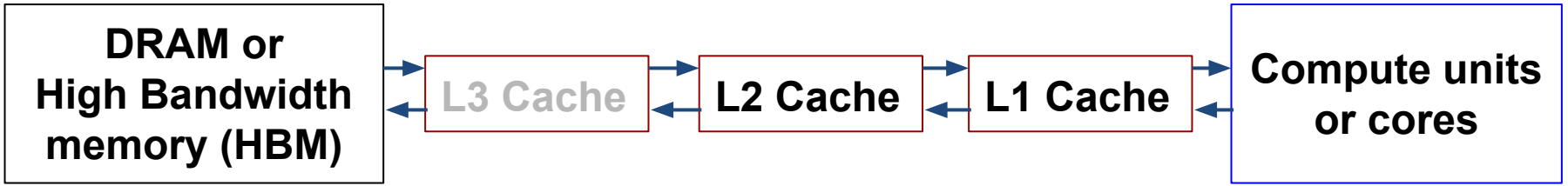
- **Efficiency matters,**
  less resources required
- Better selection of
  compute resources (GPU vs CPU)

For Developers:

- **Performance portability** independent of problem size
- Better **compiler** design **feedback**

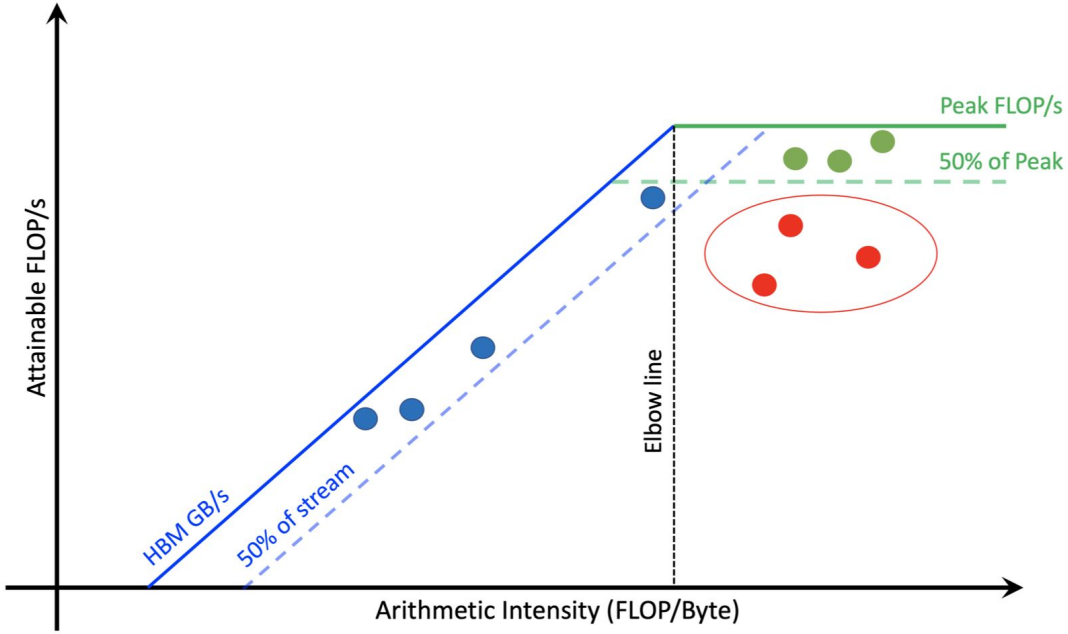


Typical memory layout and Hierarchy



- Data first moves from **DRAM to L3, L3 to L2, and L2 to L1**
- **After computing results,** data moved back across **memory hierarchy**

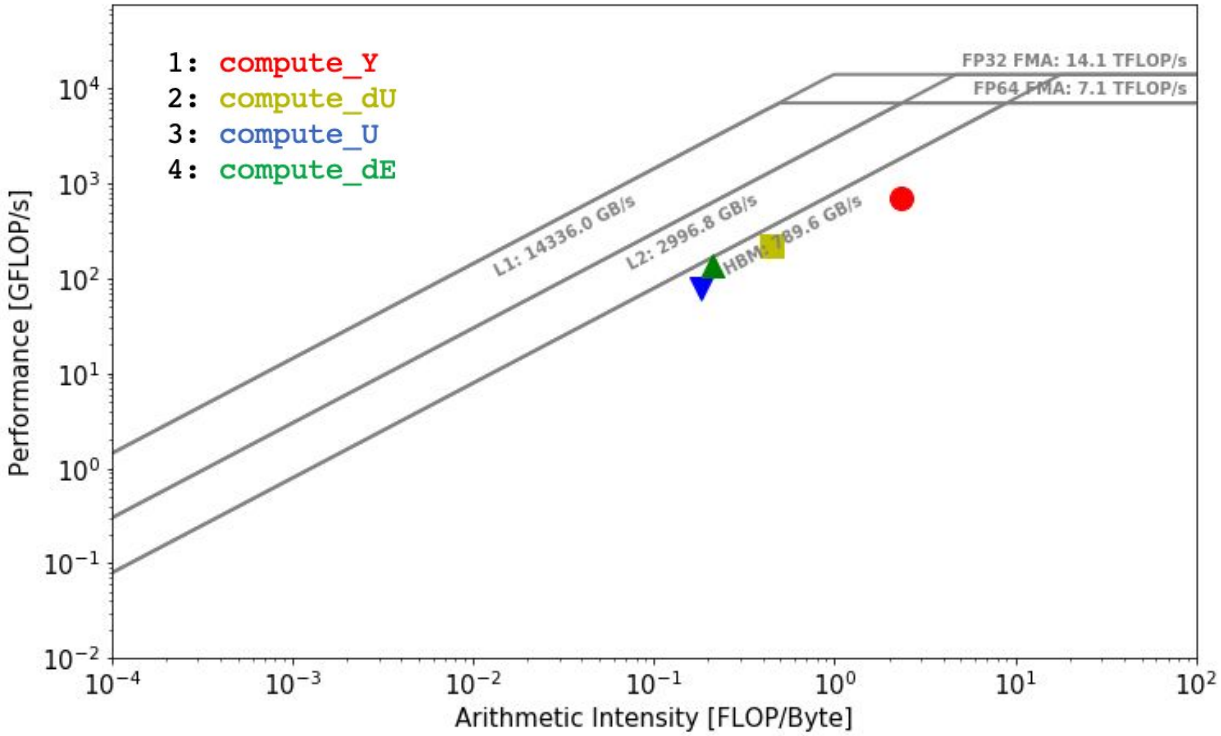
Brief summary of roofline analysis



- Region to the **left** of **'elbow line'** represents **memory bound**
- Region to the **right** represents **compute bound**
- **Kernels** that are **neither compute** or **memory bound** and show **potential for greater optimization**
- **Ideally** functions/kernels should shift **upwards and rightwards**

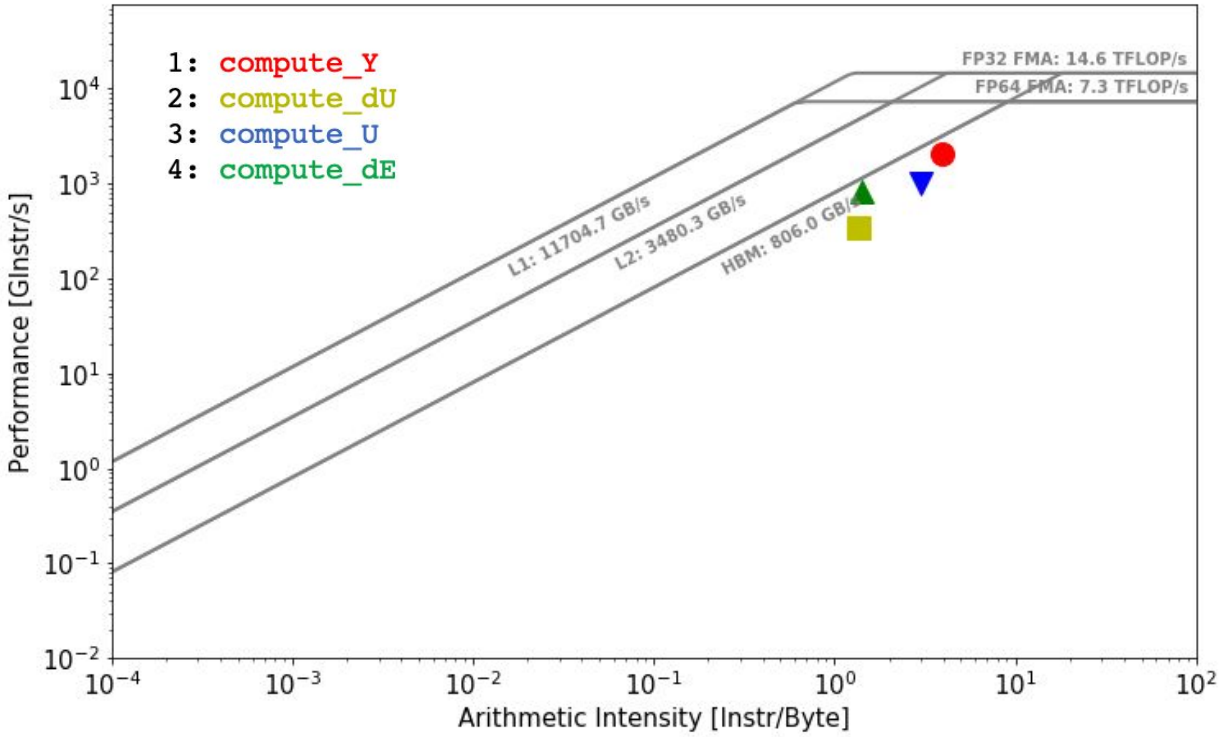
Roofline from NVIDIA V100

- **Memory bound result on V100**
- **3 of the 4 kernels have lower AI compared to other GPU**

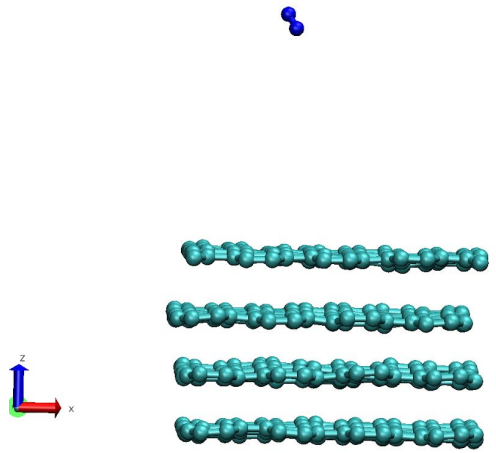


Roofline from AMD MI60

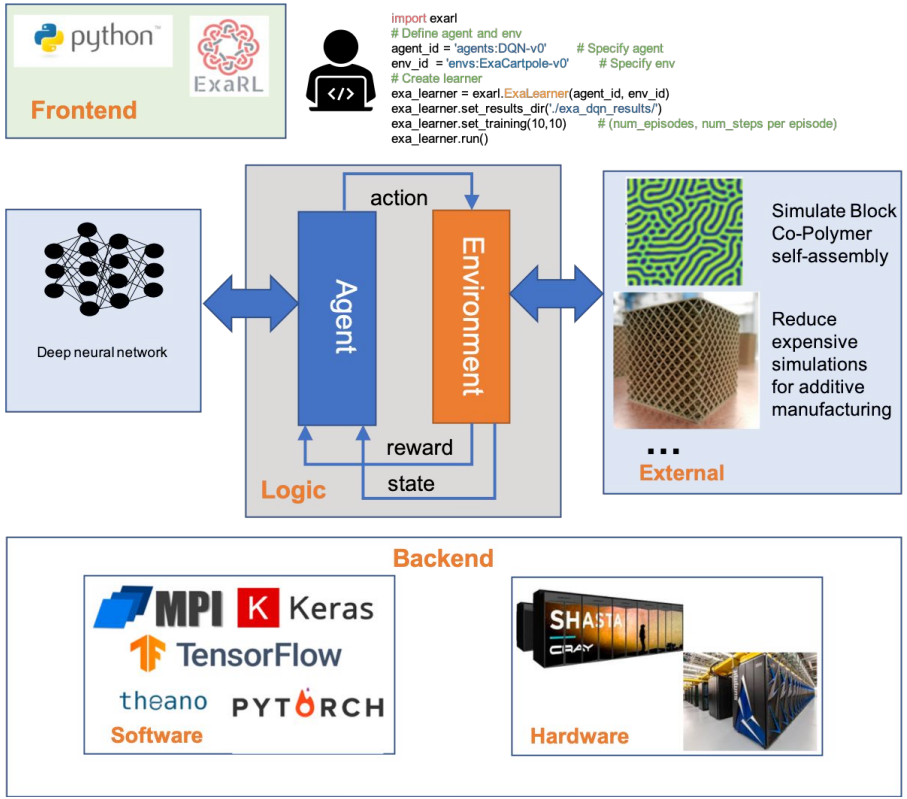
- **Kernels again close to the DRAM bandwidth line, indicating memory bound**
- **Kernels are closer to compute bound regime**
- **AI in the same order of magnitude**



Merging MD with Machine Learning



- New class of **MD simulations** will use **HPC** and **ML**
- Integrating advanced **AI framework** with **MD** is **challenging**



C++ application with Neural Networks

```
    compute_U();
    compute_bi();
    {
      compute_beta(beta,bispectrum,energy);
    }
    compute_Yi();
    compute_dU();
    compute_dE();
```

```
import torch
import numpy as np

def compute_beta(beta,bispectrum,energy):
    beta = autograd using bispectrum
    energy = byproduct of autograd
    update energy
    return energy,beta
```

- **TestSNAP** is a proxy app used for **optimizing LAMMPS SNAP** potential
- Require **data sharing** between **C++ and python** with as **little overhead** as possible

Example: Cuda to Pytorch

```
  cudaMalloc(&d_x, N*D_in*sizeof(double));

  for (int i = 0; i < N; i++) {
    for (int j = 0; j < D_in; j++)
      x[i*D_in+j] = i*j;
  }
  printf("Before: Value of X at 1,1 is %f \n",x[1*D_in+1]);

  cudaMemcpy(d_x, x, N*D_in*sizeof(double), cudaMemcpyHostToDevice);

// Calling simple NN implemented in Python
  pybind11::scoped_interpreter guard{};
  pybind11::module sys = pybind11::module::import("sys");
  sys.attr("path").attr("insert")(1, CUSTOM_SYS_PATH);
  pybind11::module py_simplenn = pybind11::module::import("py_simple");
  py::object ob1 = py_simplenn.attr("add_NN")(py::array_t<double, py::array::c_style | py::array::forcecast>(N*D_in,d_x,py::str{}),N,D_in);

  cudaDeviceSynchronize();
  cudaMemcpy(x,d_x, N*D_in*sizeof(double), cudaMemcpyDeviceToHost);
  printf("After: Value of X at 1,1 is %f \n",x[1*D_in+1]);

  cudaFree(d_x);
```

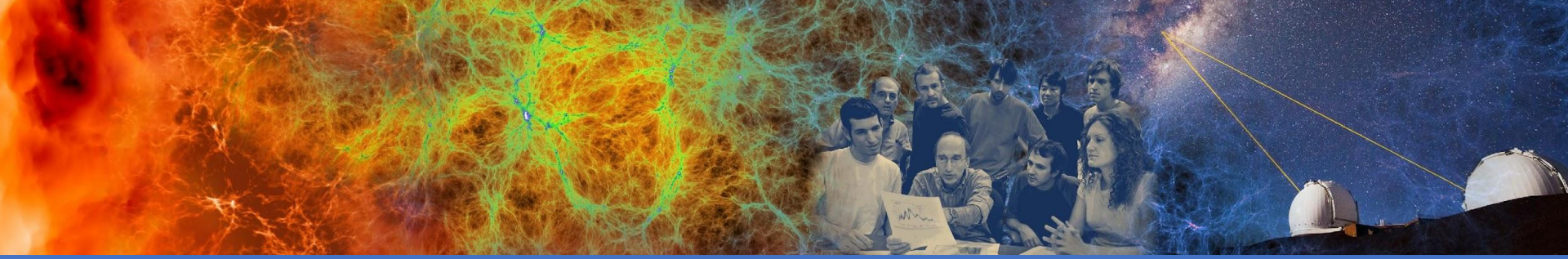
No-copy operation for sharing data between c++ and python!

Calling python functions from C++

```
pybind11::scoped_interpreter guard{};
pybind11::module sys = pybind11::module::import("sys");
sys.attr("path").attr("insert")(1, CUSTOM_SYS_PATH);
pybind11::module py_simplenn = pybind11::module::import("<py_file_name>");
py::object ob1 = py_simplenn.attr("<function_name>")(py::array_t<double, py
    ::array::c_style | py::array::forcecast>(ncol*nrows,foo1,py::str{}),
    ncols,nrows);
```

```
import numpy as np
import torch

class InterfaceHolder():
    def __init__(self,cuda_array_interface):
        self.__cuda_array_interface__ = cuda_array_interface

def add_NN(foo1, N, D_in):
    print("*********************************************")
    print("Start python")

    interface=foo1.__array_interface__
    t = InterfaceHolder(interface)
    x = torch.as_tensor(t, device=device)
    x = x.view(N,D_in)

    return None

    print("End python")
    print("*********************************************")
```

Thank You

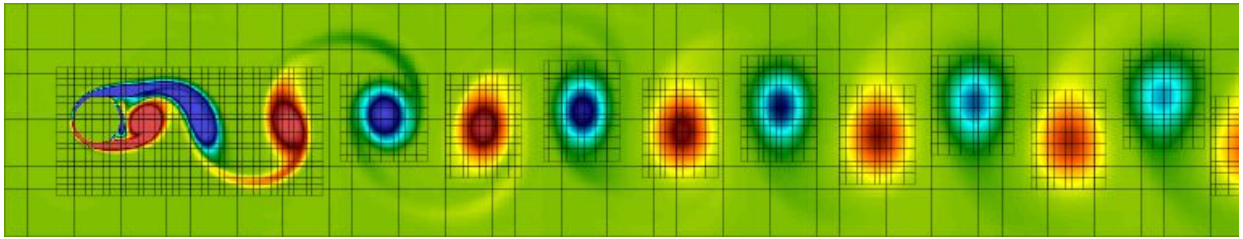


Computational Fluid Dynamics

Raphaël Prat

NERSC
BERKELEY LAB
U.S. DEPARTMENT OF ENERGY | Office of Science

Chombo4 + Proto



Chombo: Software for Adaptive Solutions of Partial Differential Equations

Chombo4 is the version using the middleware Proto for GPU portability

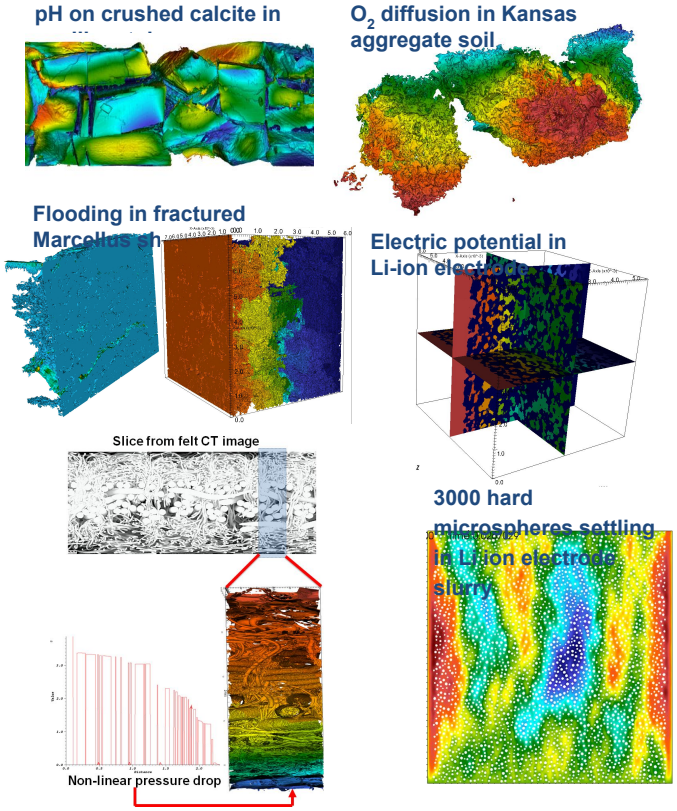
Proto: a flexible way to port stencil-based applications on CPU and GPU

Proto guidelines: User friendly, Performance and Portability

Applications

Chombo-Crunch

- High performance simulator of flow and transport in complex geometries
- Applied to subsurface science areas:
  - Carbon sequestration
  - Fracture evolution
  - Used fuel disposition
- Extended to engineering applications
  - Lithium ion battery electrodes
  - Roll-to-roll manufacturing
  - Paper manufacturing



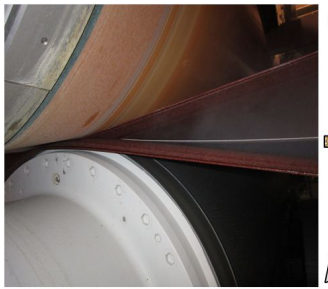
BERKELEY LAB

NERSC

Applications

Engineered pore scale heterogeneous materials: paper manufacturing

Paper machine press section



Close-up view of press nip

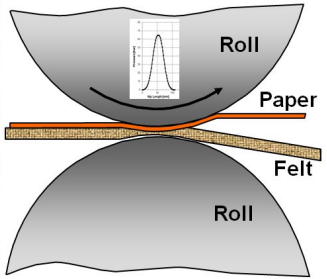
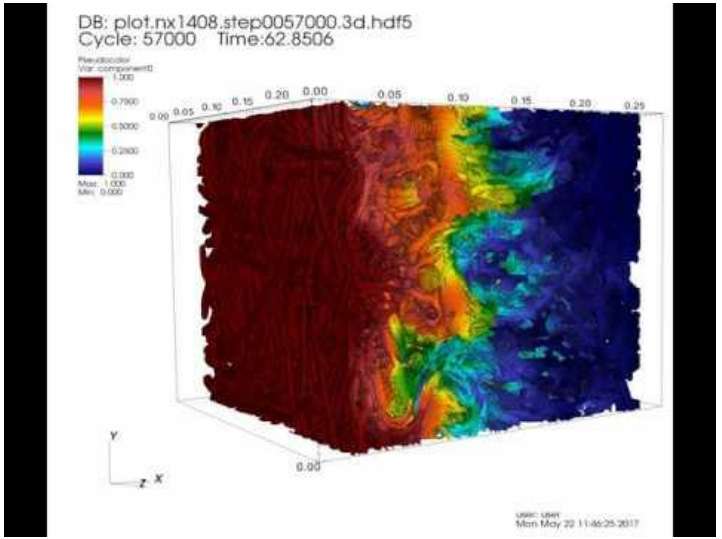


Illustration of press nip



2017 HPCWire Editor's Choice award winner for hpc in manufacturing

- ~50,000 processor cores at NERSC

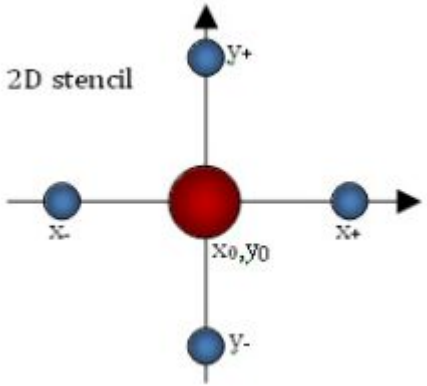
BERKELEY LAB

NERSC

Proto Features

Main features:

- Stencil
- ForAll
  - Pointwise operation
- MemType
  - CPU or GPU



$$u_{i,j}^{n+1} = \frac{1}{h^2}(-4u_{i,j}^n + u_{i-1,j}^n + u_{i+1,j}^n + u_{i,j-1}^n + u_{i,j+1}^n)$$

Laplacian in 2D

An extension of Proto is in development for irregular meshes

Challenges on GPU

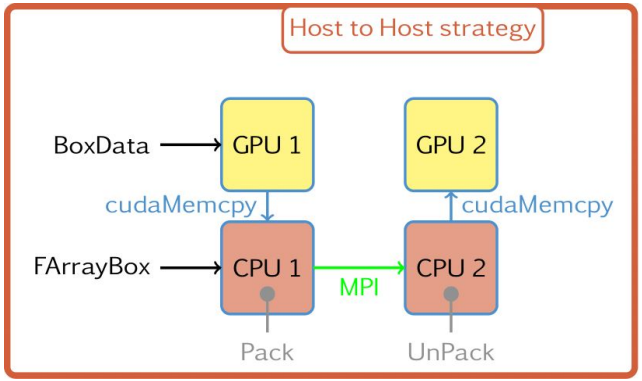
User friendly
- Remain the same design on CPU and GPU
- Limit the divergent code

Performance
- Adapt data
- The memory access pattern is predictable (for regular mesh)
- Bandwidth bound: maximize the data locality

Portability
- Port the code on NVidia, AMD or Intel GPU
- Propose a transparent solution
- Portability must not affect the performance

Exchange

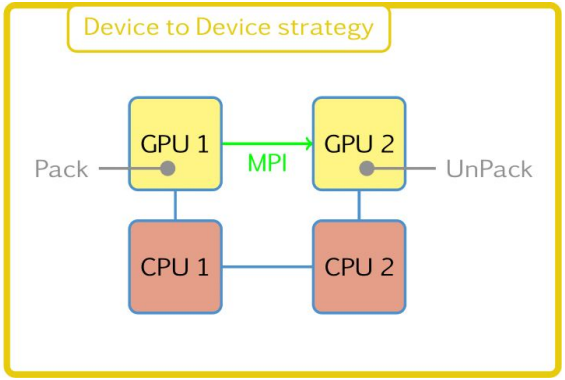
- The update of ghost areas take more than 70% of the total runtime
- This task is handle by the CPU
- cudaMemCopy are expensive
- Possibility of overlapping copies with kernels -> not enough gain



Version	GPUs	Total	Compute(s)	Compute(%)	Exchange(s)	Exchange(%)
Host to Host	6	1208.41	219.31	18.1 %	957.04	79.2 %
	48	1241.65	233.84	18.8 %	969.46	78.1 %
	384	1237.84	239.44	19.3 %	958.63	77.4 %
	3072	1304.48	237.53	18.2 %	957.86	73.4 %

Exchange

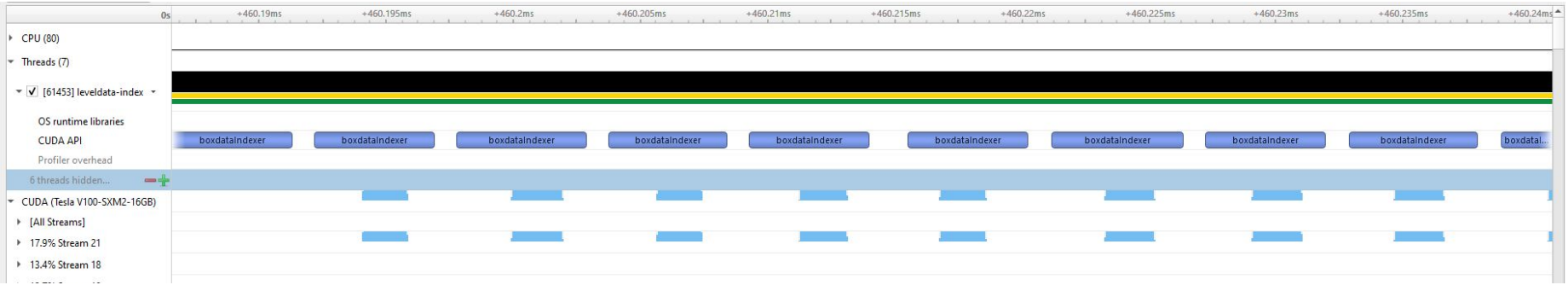
- Solution: use the CUDA-aware MPI feature
- Data are packed in buffer in the GPU memory
- Exchange is 13.6 time faster on 3,072 V100 GPUs



Version	GPUs	Total	Compute(s)	Compute(%)	Exchange(s)	Exchange(%)
Host to Host	6	1208.41	219.31	18.1 %	957.04	79.2 %
	48	1241.65	233.84	18.8 %	969.46	78.1 %
	384	1237.84	239.44	19.3 %	958.63	77.4 %
	3072	1304.48	237.53	18.2 %	957.86	73.4 %
Device to Device	6	115.07	55.59	48.3 %	29.51	25.6 %
	48	131.56	55.42	42.1 %	44.06	33.5 %
	384	159.94	54.83	34.3 %	69.53	43.5 %
	3072	180.34	56.73	31.5 %	70.34	39.0 %

Kernel Launch Time

- Legacy of the CPU design: we launch a lot of small kernels
- The kernel launch time is longer than the kernel execution time
  - CPU limitation
- Possibility to use cuda-graphs
  - creation is expensive and the gain is below what was expected
- Manual fusion of kernels
  - fusion of one kind of kernel
- Still in progress: manual fusion of different kernels



Results

Kernel	Before (GFLOPS)	After (GFLOPS)	Speed up (time)
stencilIndexer	154.73	266.79	2.27
boxDataIndexer	0.0	0.0	2.77

Table. Performance engineering of Proto. Kernel comparisons before and after fusion with nsight compute. (example : Proto/LevelData, 1GPU)

EBProto

- Proto achieves relevant performance with a portable design for different GPU types.
- We develop an extension of Proto for irregular meshes
- This extension allows dealing with complex geometries
- We have applied the previous optimizations to this package
  - Fusion
  - Functor
  - Device to Device exchange
- Pattern access is irregular
  - Performance drops
  - Adapt the memory accesses