Georgia Tech College of Computing
School of Computer Science

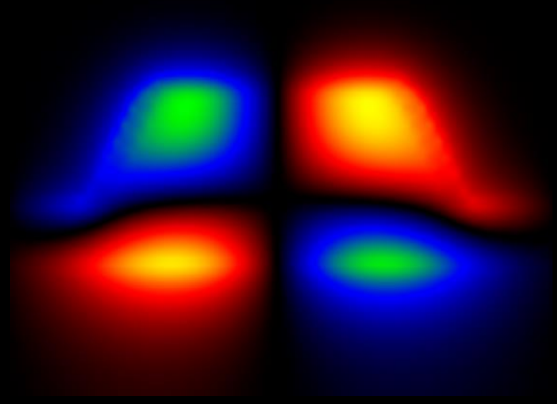
Georgia Tech College of Engineering
School of Electrical and Computer Engineering

GPU-Accelerated Electromagnetic Mode Solvers for High Performance Computing Applications (GAEMSHiPCA)

John Moxley

Joel Slaby

Conner Yurkon



March 16, 2023

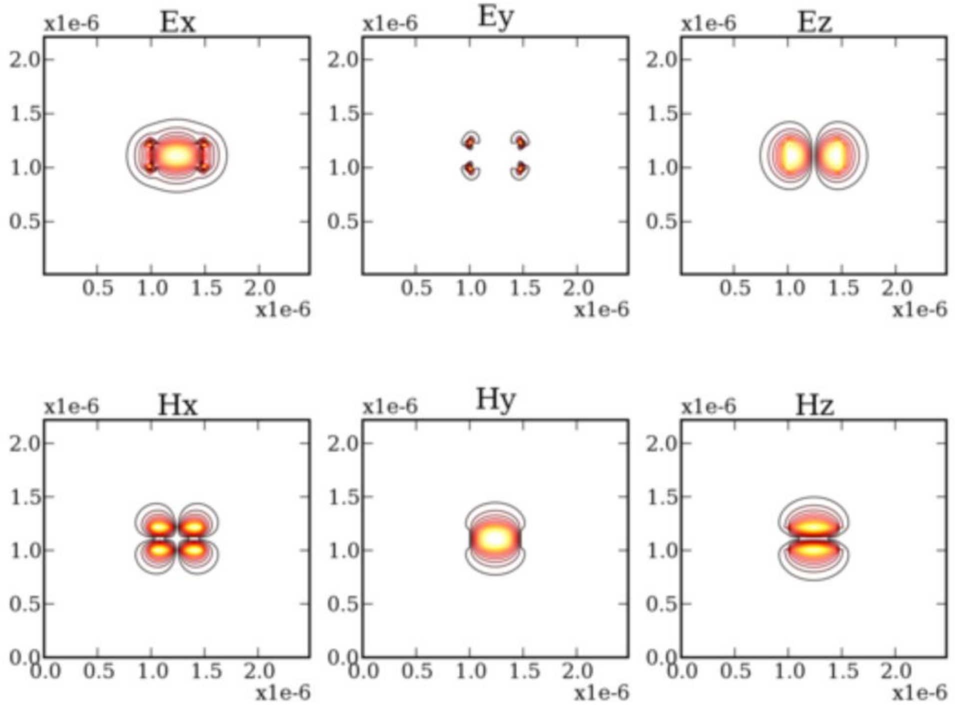
Dielectric Waveguide Mode Solver

- **Goal:** Find solution to maxwell's equations for a given geometry
  - Iterative eigenvalue problem with sparse matrices

$$(\mu^{-1}\nabla\times\epsilon^{-1}\nabla\times)\mathbf{H}=\omega^{2}\mathbf{H}$$

- **Current challenges:**
  - Solving for eigenvalues is an expensive calculation
  - Limited to smaller structures at lower resolutions
  - Speed-up can enable:
    - faster device design and optimizations
    - higher resolution calculations (more accurate)

- **Existing work:**
  - A GPU Solver for Sparse Generalized Eigenvalue Problems With Symmetric Complex-Valued Matrices Obtained Using Higher-Order FEM: https://ieeexplore.ieee.org/iel7/6287639/8274985/08468163.pdf
  - This work uses the finite-element method (FEM), which isn't scalable to larger geometries. We will use the finite-difference method (FD)



https://ieeexplore.ieee.org/iel7/6287639/8274985/08468163.pdf

Understanding Light Propagation

- Light is described with both an electric field, E, and a magnetic field, H
  - Each field has three components (x, y, z)
  - Total of 6 field components $(E_x, E_y, E_z, H_x, H_y, H_z)$
  - Each field is a function of *position* and *time*
- How are the fields connected? - **Maxwell equations**
- Time varying magnetic fields induce spatial varying electric fields (and vice versa)
  - Time varying fields can also be considered in frequency domain using a fourier transform

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0}$$

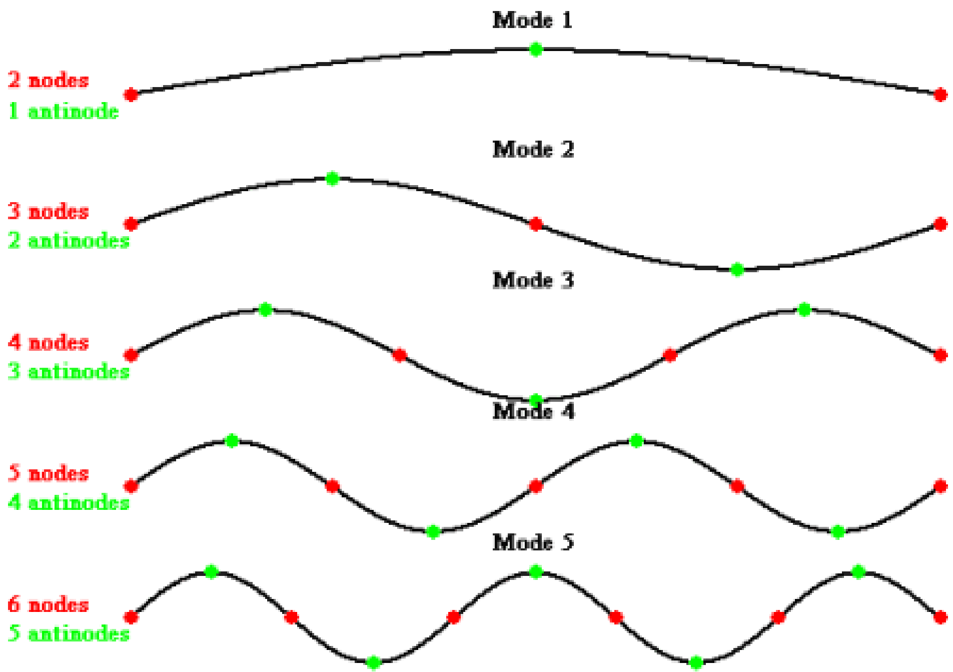
$$\nabla \cdot \mathbf{B} = 0$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t}$$

Understanding Waveguide Modes

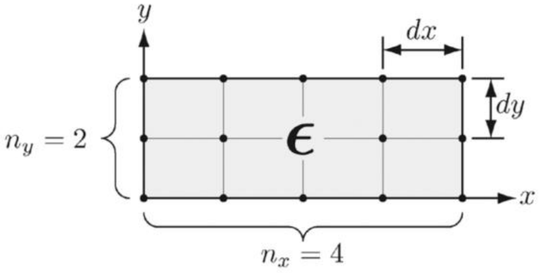
- Traditional wave equation is of the form: $\frac{d^2u}{dt^2} = c^2\frac{d^2u}{dx^2}$
- Solution $u = A\sin(kx + \omega t)$
  - ω is time dependence
  - k is the spatial dependence (wave vector)
- A **mode** is one of these solutions to the wave equation
  - It is a spatially stable solution
  - Each mode is defined by its wave vector k
- Light is a wave! $\frac{d^2E_x}{dz^2} = \mu\epsilon\frac{d^2E_x}{dt^2}$
- We can combine maxwell's equations to get a similar form for the E and H fields
- Thus, for a given waveguide geometry (ϵ profile) there exists a stable solution to maxwell equations following the form of the wave equation



https://blog.soton.ac.uk/soundwaves/standing-waves/2-string-modes/ isvr

Mode Solver Problem Breakdown

(1) Define your geometry

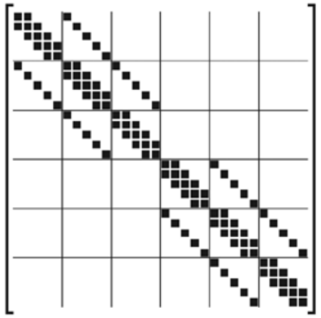


(2) Create the maxwell operator matrix $\boldsymbol{A}$

sparse matrix

$$\underbrace{(\mu^{-1}\nabla\times\epsilon^{-1}\nabla\times)}\mathbf{H}=\omega^2\mathbf{H}$$

$$\boldsymbol{A} =$$

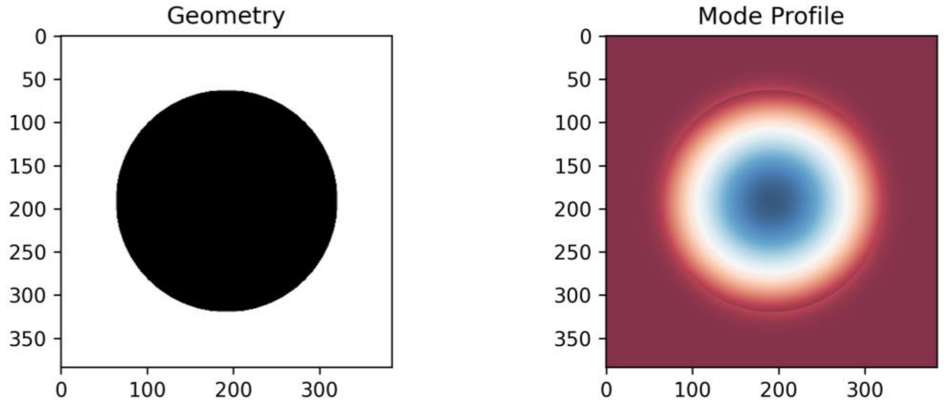


(3) Calculate the eigenvalues

This is where HPC comes into play

$$\begin{bmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \end{bmatrix} = \beta^2 \begin{bmatrix} H_x \\ H_y \end{bmatrix}$$

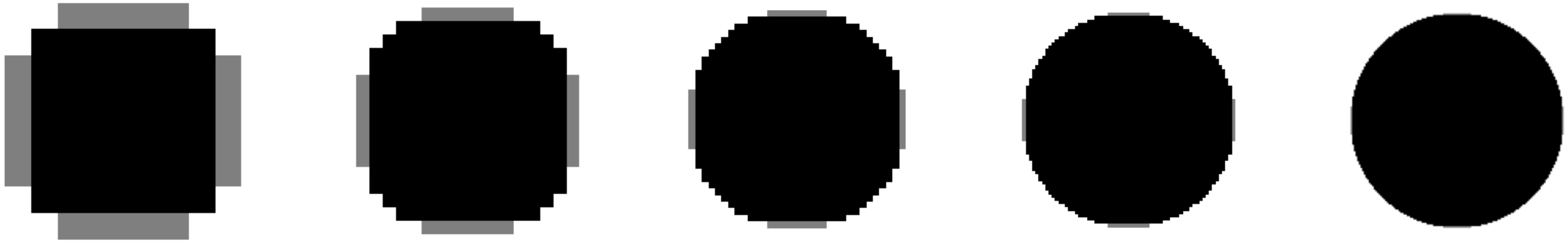
(4) Compute the mode fields (eigenvectors)



A. B. Fallahkhair, K. S. Li and T. E. Murphy, "Vector Finite Difference Modesolver for Anisotropic Dielectric Waveguides", J. Lightwave Technol. 26(11), 1423-1431, (2008).

Performance Metrics

- **Speed vs Resolution**
  - For a given resolution, how long does the algorithm take to complete?
  - Focusing on GPU speedup—fix the number of processors, memory, etc.
  - Compare the accelerated version to the baseline open-source versions
- **Performance of the Resolution (convergence)**
  - At what rate does the resolution increase as we allow for more execution time?
  - This needs to be at least O(Δn^2)
  - With the speedup, we'll be able to more accurately calculate the convergence rate, because we'll be able to simulate at higher resolutions



Baseline Metrics

- Current state-of-the-art mode solving represented by many open-source repositories

- **Mpb**
  - https://github.com/NanoComp/mpb
  - Implemented in C
  - Includes implementation with distributed memory using MPI
- **Empy**
  - Implemented in python
  - Fully vectorial finite difference method
  - https://github.com/lbolla/EMpy
- **Modesolver**
  - Implemented in python
  - https://github.com/jtambasco/modesolverpy

EMpy

Electromagnetic Python

Solution: GPU Accelerated Mode Solver

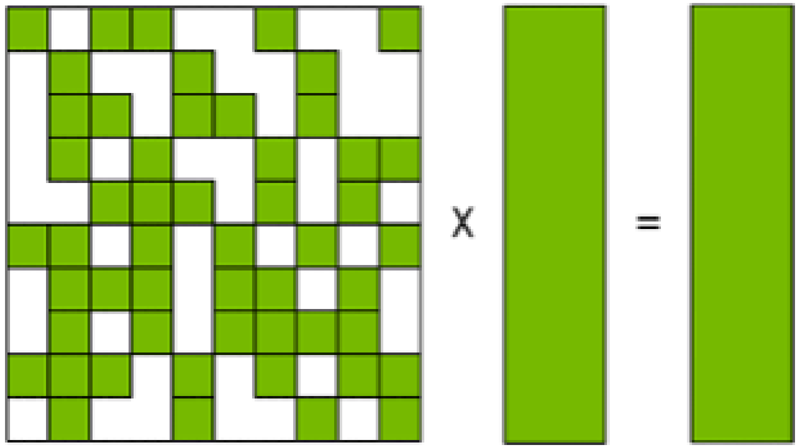
- **Goal:** Accelerate a presently implemented mode solver on a GPU

- **Design:**
  - Start with existing solution (serial), open-source such as *empy*
  - Convert the eigenmode problem to be GPU compatible
    - Will require parallelizable pre-conditioner
    - Possible algorithms include locally block preconditioned conjugate gradient (LOBPCG)
  - Validate accuracy and determine GPU speed-up

- **Challenges:**
  - Solving an eigenmode problem requires an iterative solution
  - Will have to take advantage of matrix sparsity to reduce gpu memory requirements

Taking Advantage of Matrix Sparsity on GPU

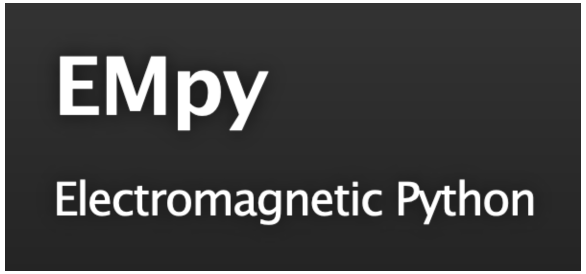
- We can use the fact that matrices we will be working with are *sparse*, i.e., very few coefficients are nonzero values
- Memory consumption can be reduced (and performance increased) by using special representations of these matrices, storing only the nonzero coefficients
- For portions of the algorithm that involve multiplying matrices, we can take advantage of Sparse-Matrix Dense-Matrix Multiplication (SpMM) on CUDA
  - The cuSPARSE library provides `cusparseSpMM` for this purpose

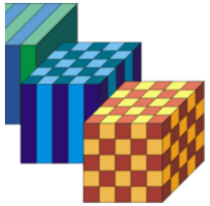


https://developer.nvidia.com/blog/accelerating-matrix-multiplication-with-block-sparse-format-and-nvidia-tensor-cores/

Validation

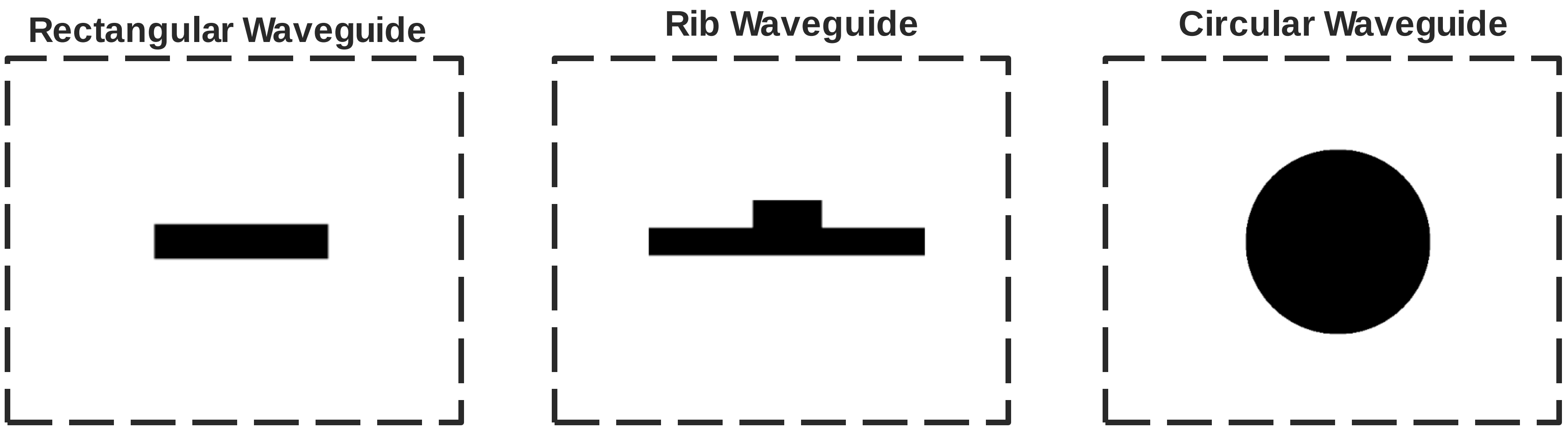
- Results will be compared against current open-source mode solvers
- Multiple mode solvers will be used in tandem since accuracy also depends on simulation resolution





Planned Experiments

- **Datasets:** Select three geometries (primarily isotropic but with consider anisotropic dielectrics)
  - Rectangular waveguide
  - Rib waveguide
  - Circular waveguide



- **Testbed:**
  - PACE Cluster (coc-ice, coc-ice-multi, coc-ice-gpu)
- **Potential plots:**
  - Speed vs Resolution
  - Performance vs Resolution (convergence)

CSE 6230 Project Proposal:

Performance Analysis of Proxy-apps for Computational Chemistry Methods

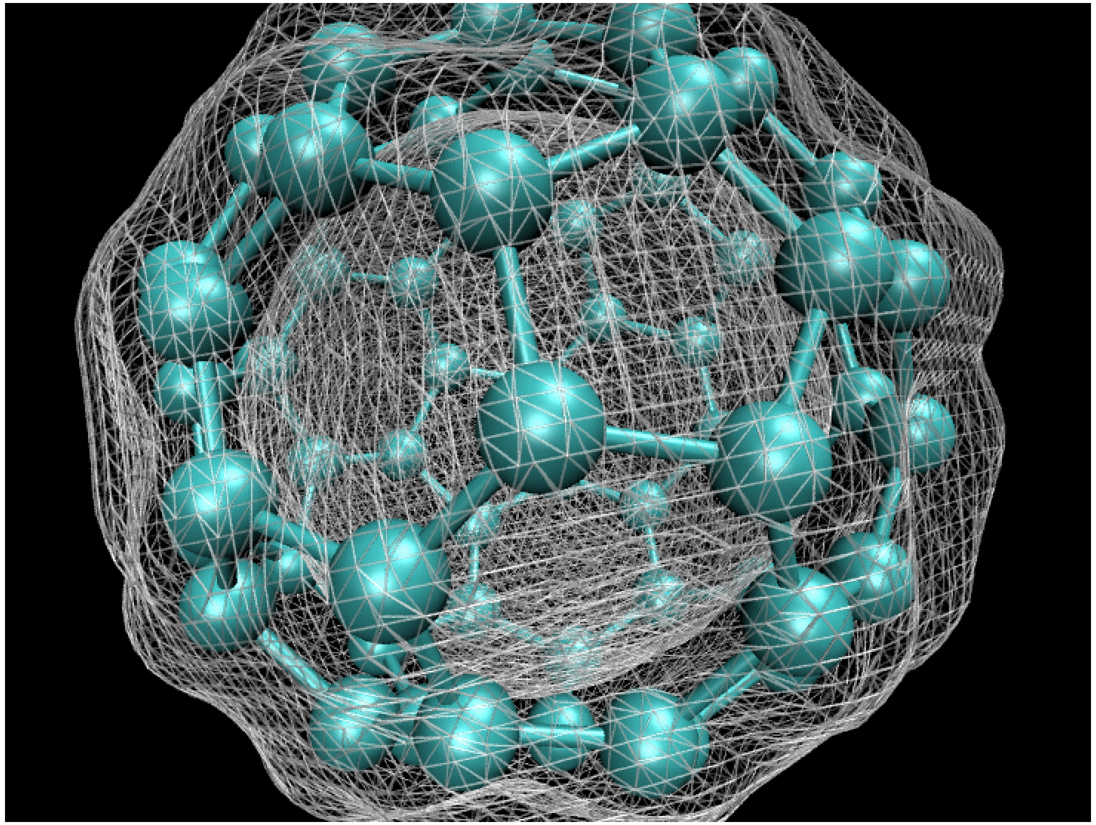
Shehan Parmar, Austin Wallace, Blair Johnson

Spring 2023

Georgia Tech

Project Category & Problem Definition

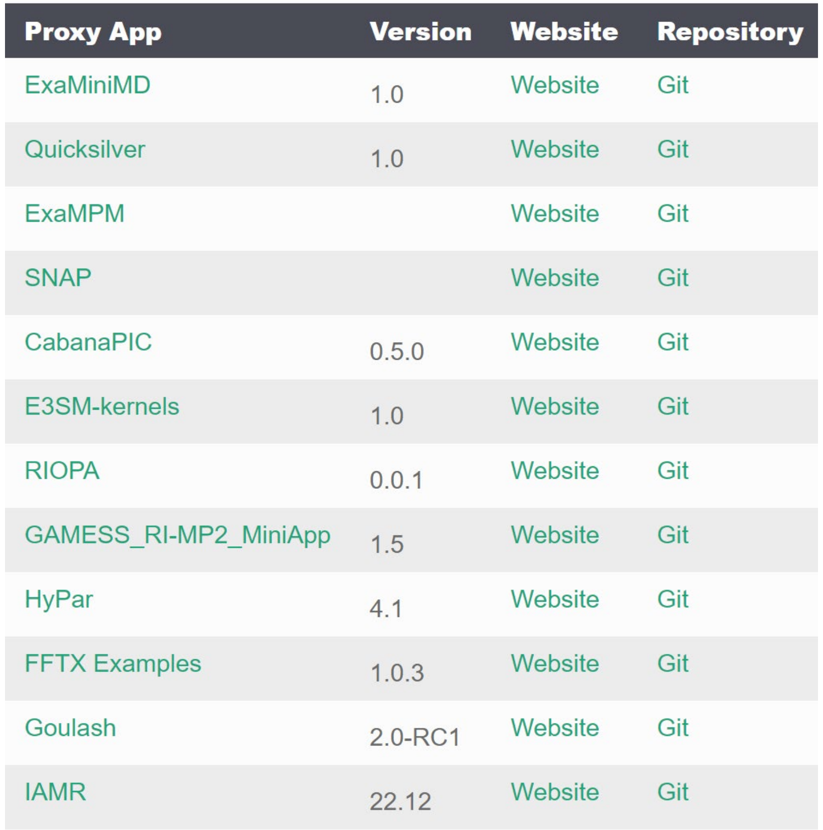
- **Category**: Scientific Application & Reproducibility
- **Problem Definition**:
  - Density Functional Theory (DFT) is used to investigate properties of molecular systems.
  - DFT relies on solving the Kohn-Sham equations, which require computationally expensive orthogonalization of large matrices.
  - Proxy apps can be used to reduce development workload and yet draw conclusions on code performance on heterogeneous architectures.



https://en.wikipedia.org/wiki/Density_functional_theory#/media/File:C60_isosurface.png

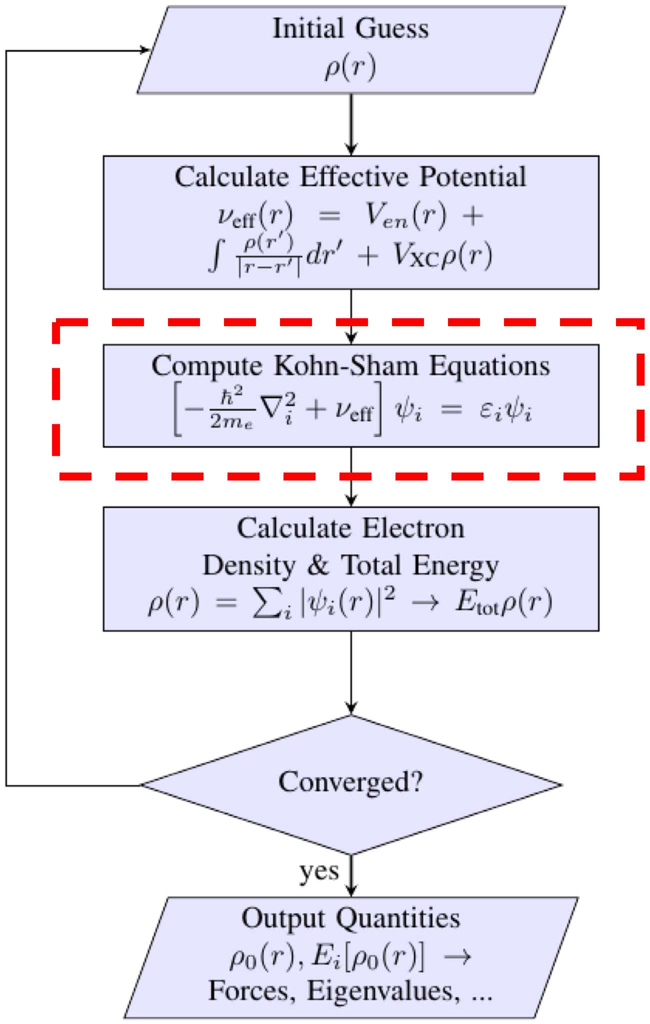
What are proxy apps?

- Proxy apps reduce the problem to essential components to understand performance-critical aspects of an algorithm.
- In this work, we will employ the **Löwdin orthonormalization of a tall-skinny matrix** as a proxy app for solving the KS equations.
- Other possible proxy-apps: https://proxyapps.exascaleproject.org/app/



Proxy App	Version	Website	Repository
ExaMiniMD	1.0	Website	Git
Quicksilver	1.0	Website	Git
ExaMPM		Website	Git
SNAP		Website	Git
CabanaPIC	0.5.0	Website	Git
E3SM-kernels	1.0	Website	Git
RIOPA	0.0.1	Website	Git
GAMESS_RI-MP2_MiniApp	1.5	Website	Git
HyPar	4.1	Website	Git
FFTX Examples	1.0.3	Website	Git
Goulash	2.0-RC1	Website	Git
IAMR	22.12	Website	Git

Procedure for Solving Kohn-Sham Equations



Initial Guess
$\rho(r)$

Calculate Effective Potential
$\nu_{\text{eff}}(r) = V_{en}(r) + \int \frac{\rho(r')}{|r-r'|}dr' + V_{XC}\rho(r)$

Compute Kohn-Sham Equations
$\left[-\frac{\hbar^2}{2m_e}\nabla_i^2 + \nu_{\text{eff}}\right]\psi_i = \varepsilon_i\psi_i$

Calculate Electron Density & Total Energy
$\rho(r) = \sum_i |\psi_i(r)|^2 \rightarrow E_{\text{tot}}\rho(r)$

Converged?

yes

Output Quantities
$\rho_0(r), E_i[\rho_0(r)] \rightarrow$
Forces, Eigenvalues, ...

1. Compute the Gram matrix $S = A^T A$
2. Compute $C = S^{1/2}$
3. Update A: $A_{new} = AC$

Majid, M.F.; Mohd Zaid, H.F.; Kait, C.F.; Ahmad, A.; Jumbri, K. Ionic Liquid@Metal-Organic Framework as a Solid Electrolyte in a Lithium-Ion Battery: Current Performance and Perspective at Molecular Level. *Nanomaterials* **2022**, *12*, 1076. https://doi.org/10.3390/nano12071076

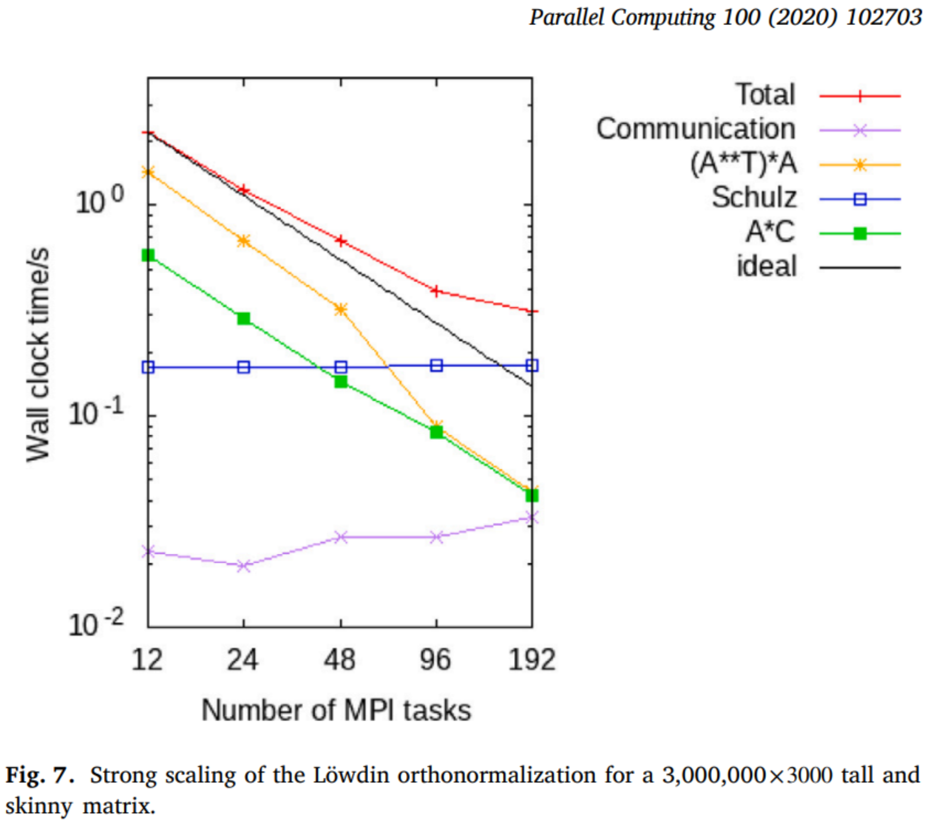
Goals + Performance Metrics

Goals:

- Reproduce and benchmark method from [1] on GT clusters
- Introduce and benchmark mixed precision scheme

Performance Metrics

- Compute & communication time
- Iterations to convergence / departure from orthogonality



Parallel Computing 100 (2020) 102703

Fig. 7. Strong scaling of the Löwdin orthonormalization for a 3,000,000×3000 tall and skinny matrix.

Strong scaling benchmark method [1]

[1] M. Lupo Pasini, B. Turcksin, W. Ge, and J.-L. Fattebert, “A parallel strategy for density functional theory computations on accelerated nodes,” *Parallel Computing*, vol. 100, p. 102703, Dec. 2020, doi: 10.1016/j.parco.2020.102703.

Baselines

- Cholesky factorization (CholeskyQR)
- Lowdin Orthonormalization from [1]

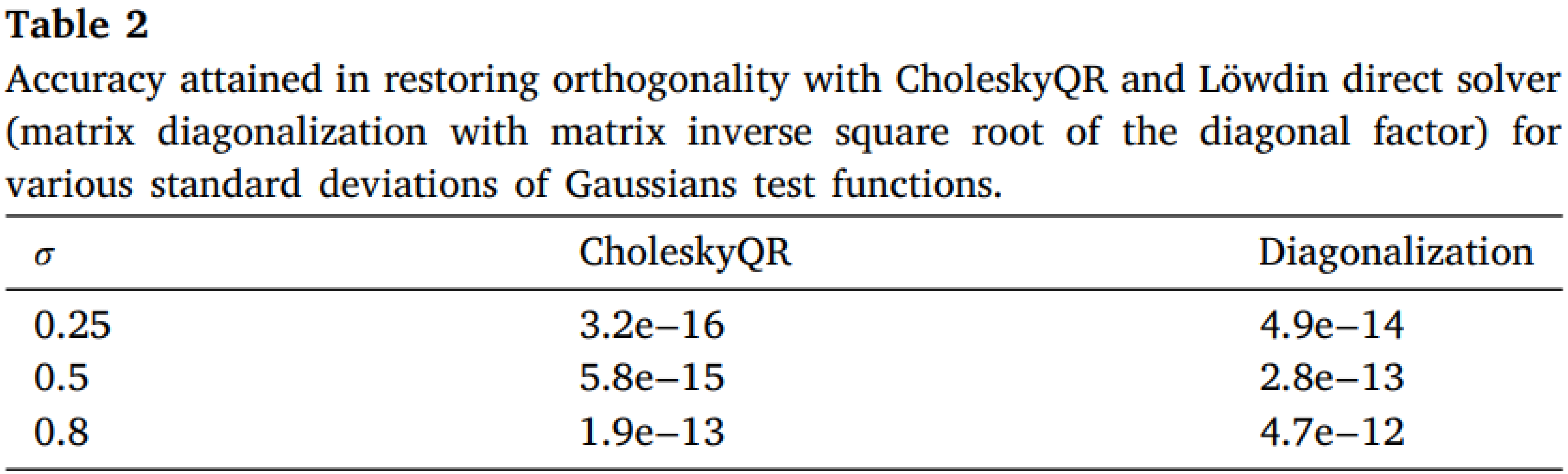


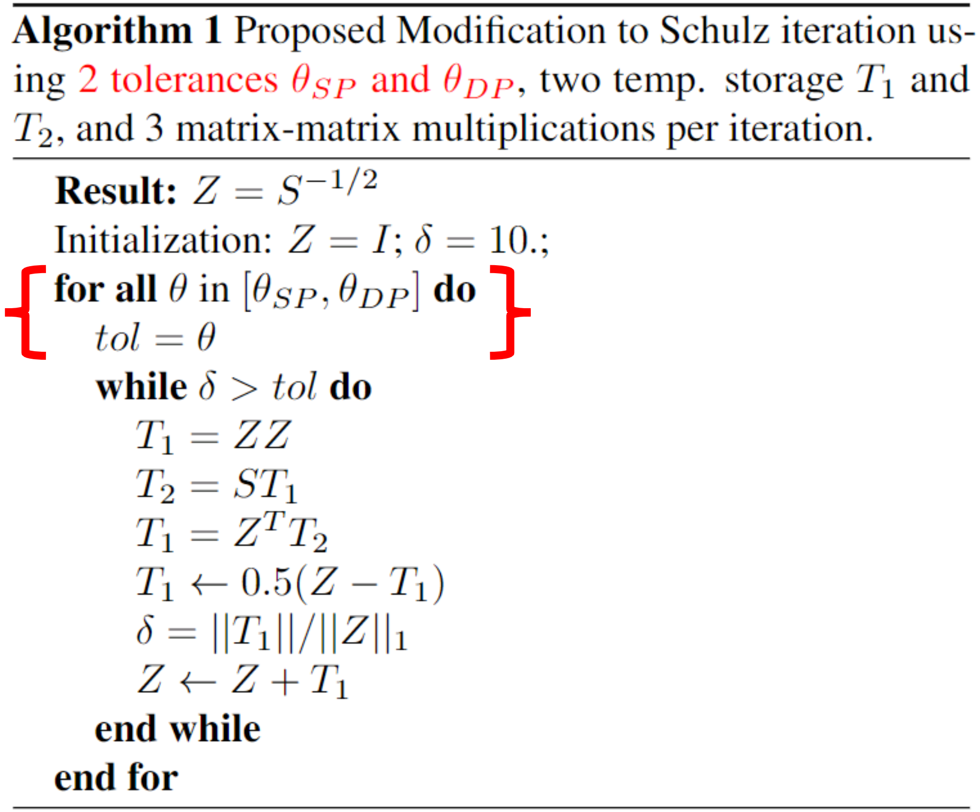
Table 2

Accuracy attained in restoring orthogonality with CholeskyQR and Löwdin direct solver (matrix diagonalization with matrix inverse square root of the diagonal factor) for various standard deviations of Gaussians test functions.

σ	CholeskyQR	Diagonalization
0.25	3.2e−16	4.9e−14
0.5	5.8e−15	2.8e−13
0.8	1.9e−13	4.7e−12

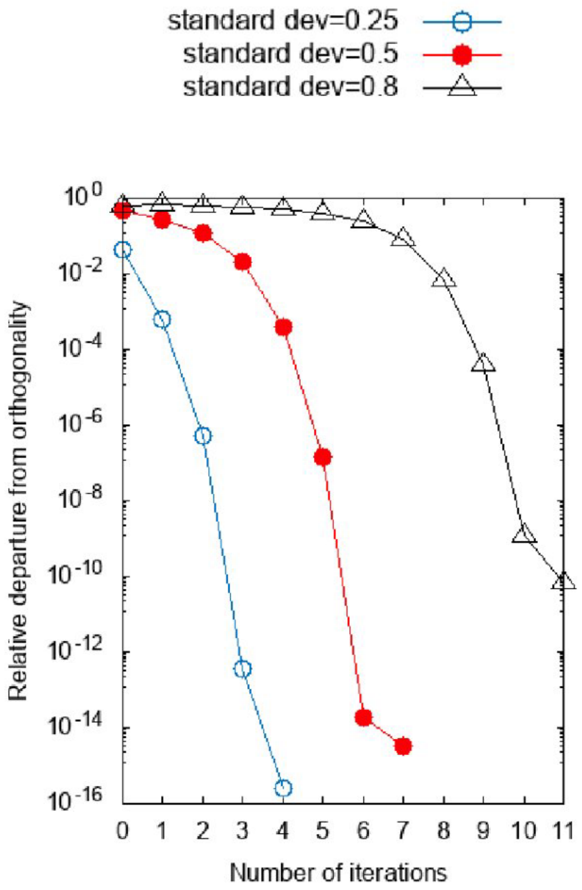
Proposed Solutions

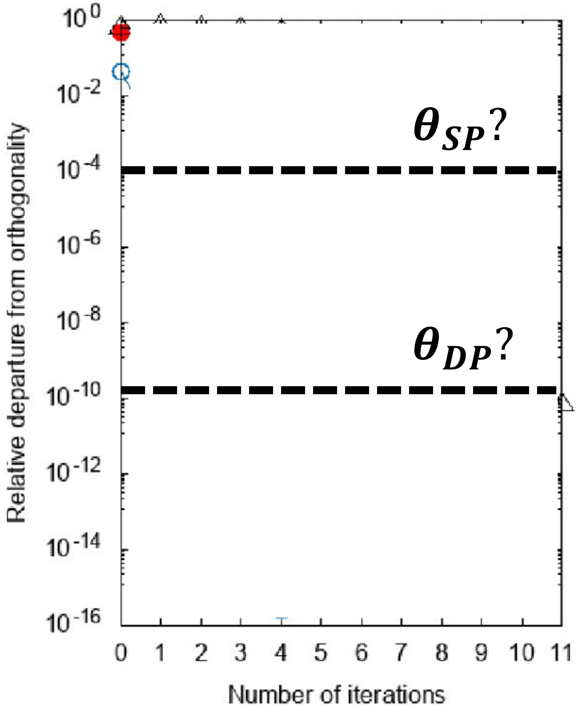
- **Validation**:
  - Pasini et al. Paper
  - BLAS/Sequential version of orthonormalization procedure
- **Dataset**: Use test cases from Pasini et al.
- **Algorithm:**

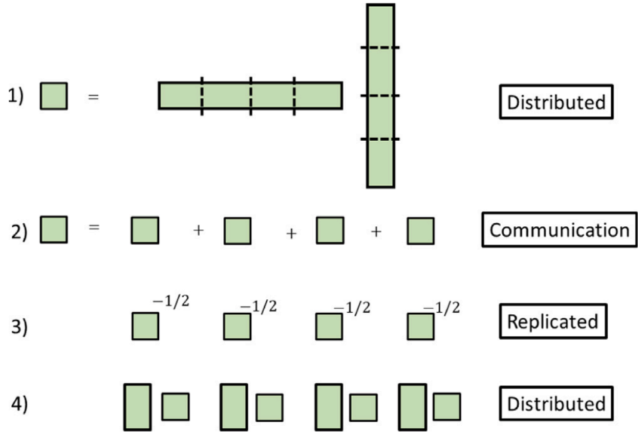


Algorithm 1 Proposed Modification to Schulz iteration using 2 tolerances θ_{SP} and θ_{DP}, two temp. storage T_1 and T_2, and 3 matrix-matrix multiplications per iteration.

```
Result: $Z = S^{-1/2}$
Initialization: $Z = I$; $\delta = 10.$;
for all $\theta$ in $[\theta_{SP}, \theta_{DP}]$ do
    $tol = \theta$
    while $\delta > tol$ do
        $T_1 = ZZ$
        $T_2 = ST_1$
        $T_1 = Z^T T_2$
        $T_1 \leftarrow 0.5(Z - T_1)$
        $\delta = ||T_1||/||Z||_1$
        $Z \leftarrow Z + T_1$
    end while
end for
```



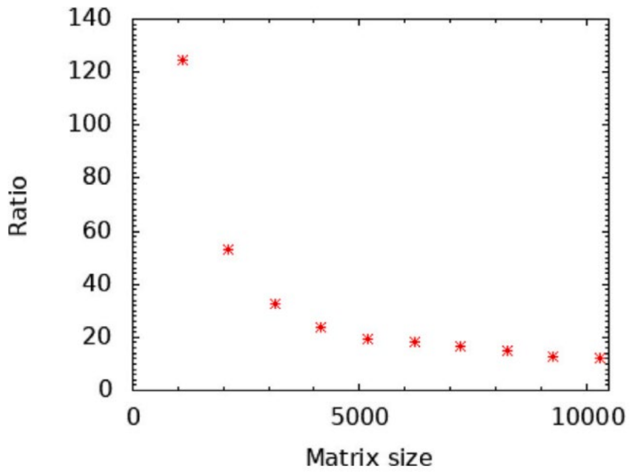




Experiments

- **Potential Test Beds**:
  - COC-ICE
  - HIVE (4 V100s/node, 16 nodes)
  - ICEHAMMER
- **Other Potential Plots**:
  - Wall clock time vs processes
  - Communications time vs processes
  - Compute time vs processes
  - Strong and weak scaling
  - Convergence iterations vs departure from orthogonality vs mixed-precision schedule

Runtime ratio between method in [1] and cuBLAS DGEMM as a function of matrix size



	MPI Tasks	Matrix Size	Alternate Parallel Strategy?
Benchmark Test			
Mixed Precision Test			

[1] M. Lupo Pasini, B. Turcksin, W. Ge, and J.-L. Fattebert, “A parallel strategy for density functional theory computations on accelerated nodes,” *Parallel Computing*, vol. 100, p. 102703, Dec. 2020, doi: 10.1016/j.parco.2020.102703.

Accelerating Discrete Wavelet Transform

Devashish Gupta, Parima Mehta, Rakesh Mugaludi

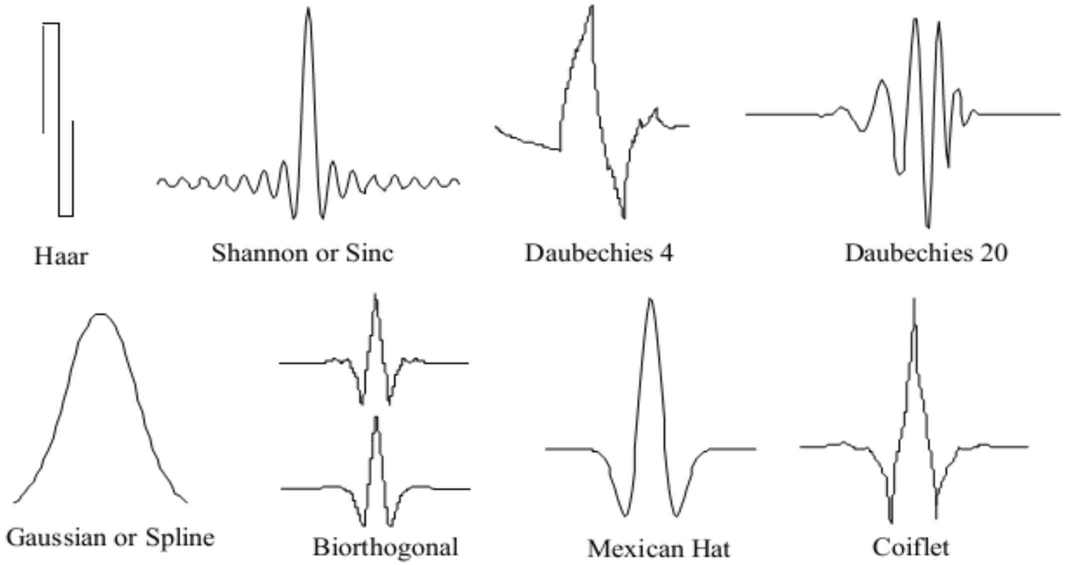
Project Category: Application

CSE 6230: High Performance Parallel Computing

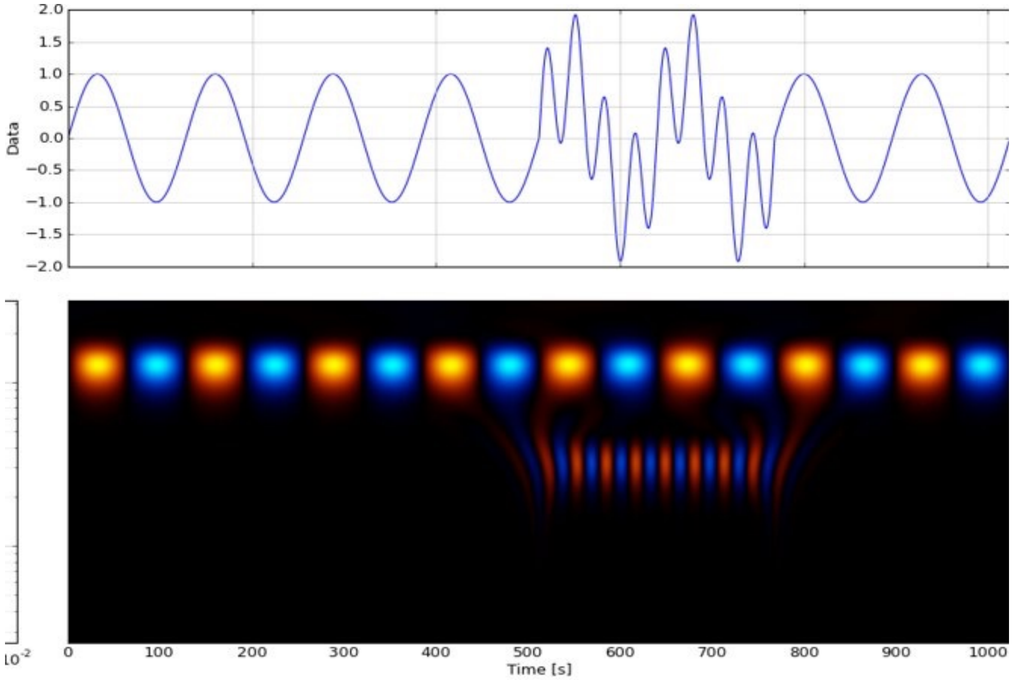
Georgia Tech

Introduction

- Discrete Wavelet Transform (DWT) is a powerful mathematical tool in signal processing and multiresolution analysis
- It is a generalization of the Fast Fourier Transform that allows capturing global and local features of the input data



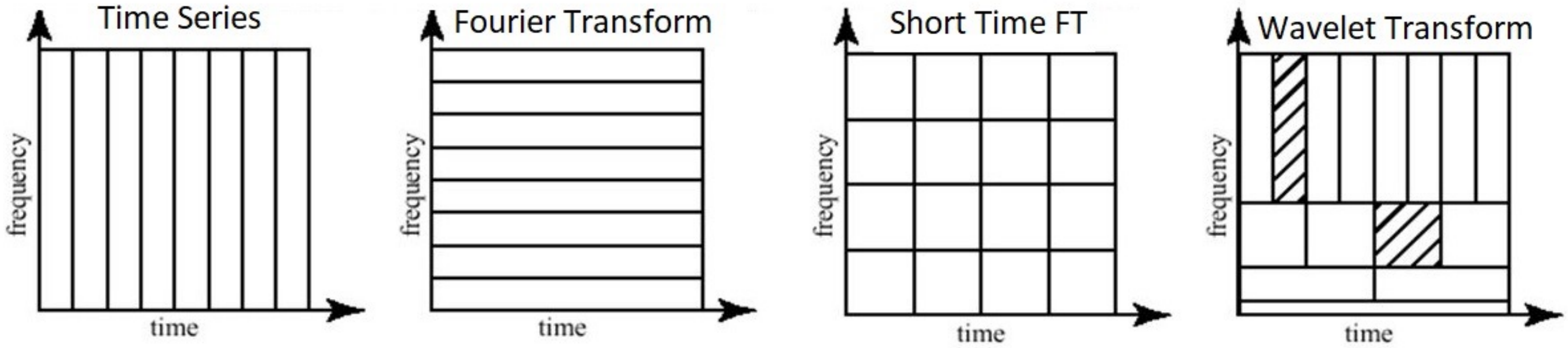
Widely used 1D mother wavelets



Multiresolution analysis

Problem

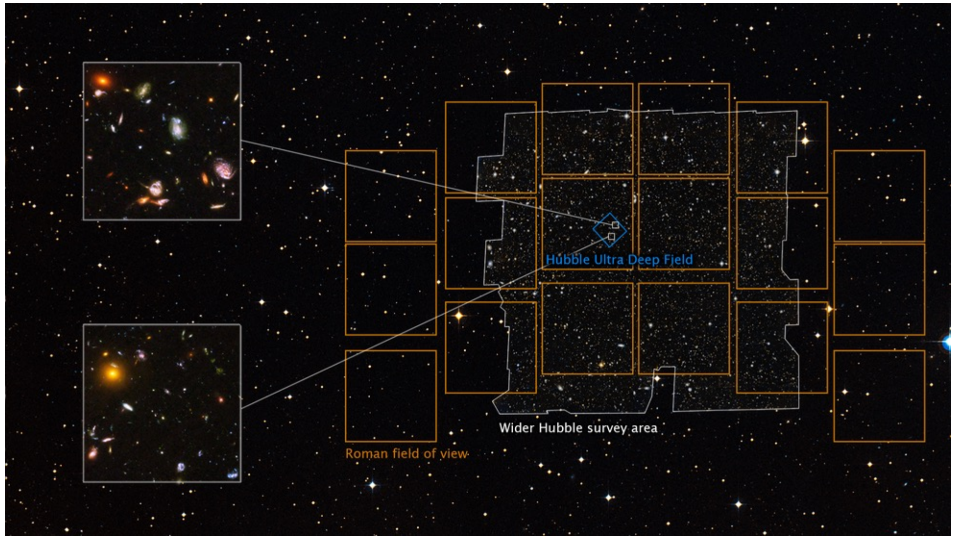
- DWT hierarchically decomposes the input into a basis of wavelets derived from the 'mother wavelet'
- It involves repeated correlations of the input with the scaled and shifted mother wavelets
- Complexity scales exponentially with increasing number of levels



Time-frequency resolution allocation for time domain data, FT, STFT and DWT.

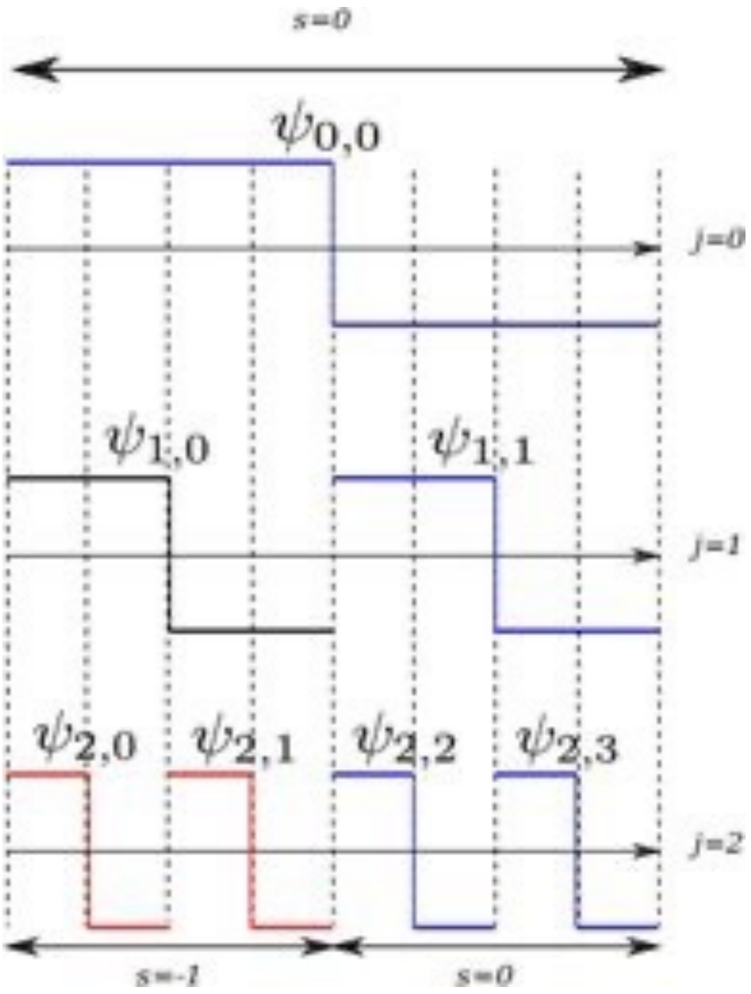
Motivation

- Diverse applications:
  - Image compression and denoising
  - Gravitational wave detection and analysis
  - EEG & ECG signal analysis
  - Feature extraction for ML
  - Multiresolution analysis of financial signals
  - Seismic data analysis, earthquake prediction



- Improvements in time to solution would benefit processing large images. Ex. James Webb Space Telescope generates 12.6GB of raw image data/hour.

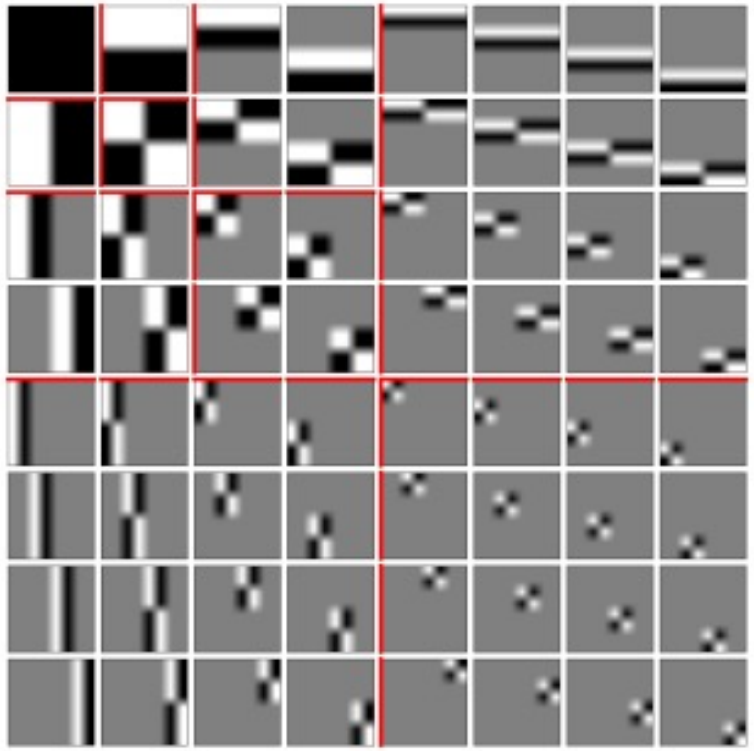
Solution



Hierarchical basis for 1D Haar transform

- DWT, especially Haar wavelet transform is highly parallelizable as:
    - It involves repeated correlations of the input with the scaled and shifted mother wavelets
    - Computation has temporal independence within a level and spatial independence across levels.
    - High scope for data and computation reuse with minimal communication.

Solution



Hierarchical basis for 2D Haar transform

- Challenges to tackle:
  - Each level requires different amount of computation
  - Ensuring coalesced memory access
  - Optimal domain decomposition for balanced workload.

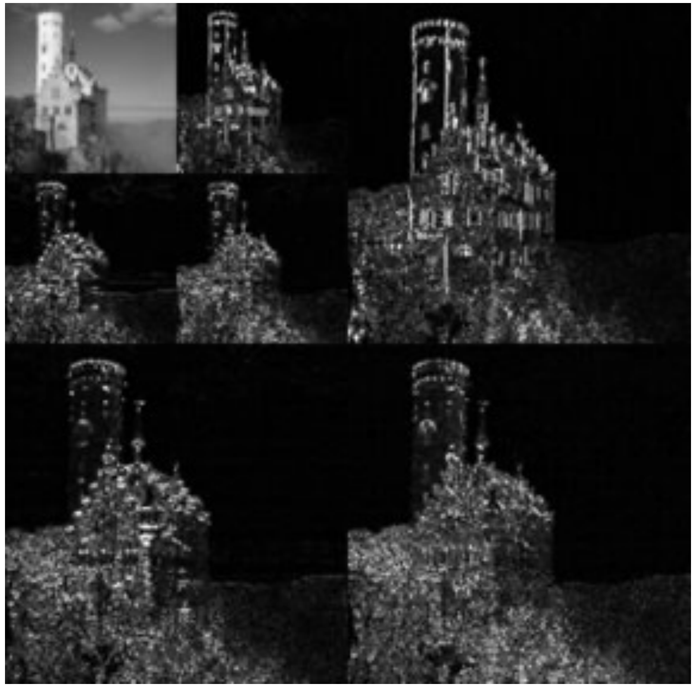
Goal: *Improve time to solution for 1D & 2D Discrete Wavelet Transform*

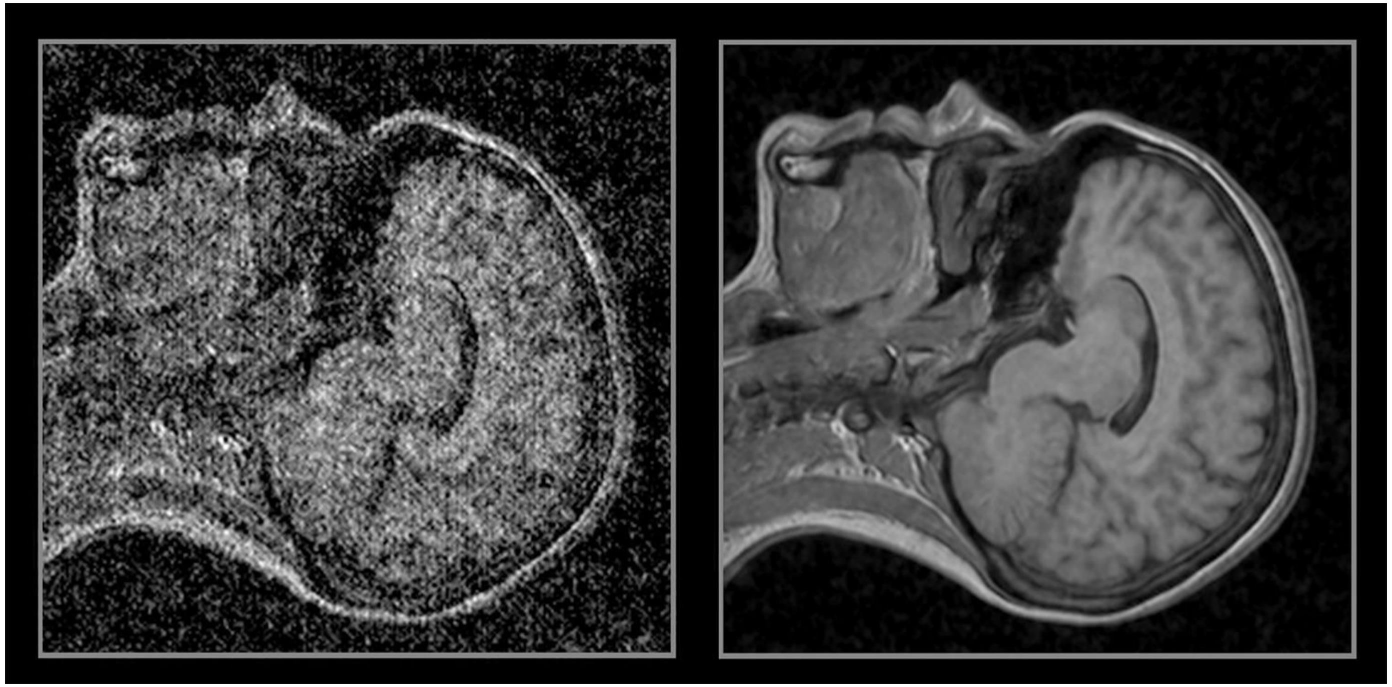
Validation

- Wavelet transforms having several components, the ability to compare and validate each of these components would be important for realizing a functionally equivalent parallel implementation
- We aim to validate our final output against existing implementations such as in **Matlab's Wavelet toolbox** and **GNU Scientific Library (GSL)**
- These libraries support multiple wavelet transforms allowing us to extend and compare against a wider set if time permits

Validation

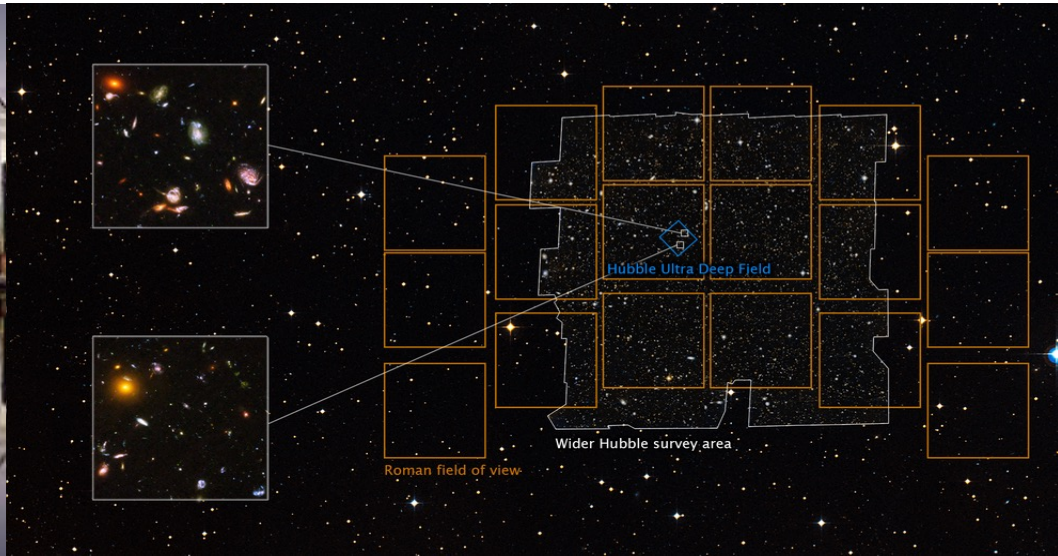
- We further aim to focus on validating our implementation on tasks such as signal decomposition (image compression) and signal denoising





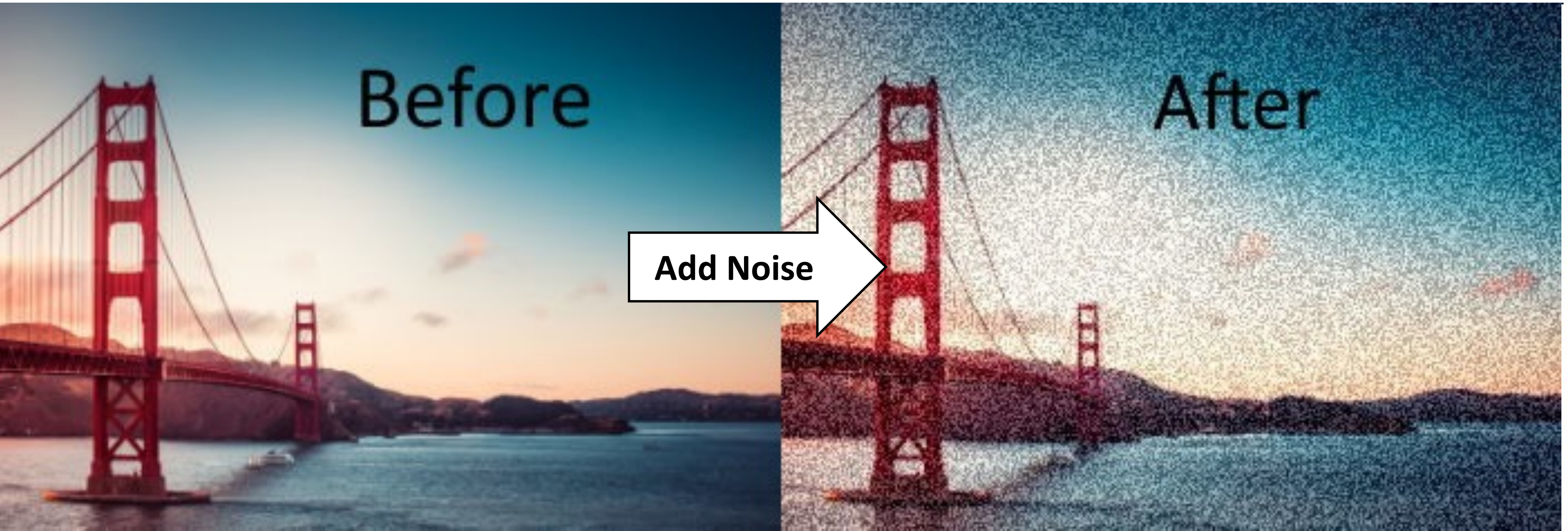
Datasets and Testbed

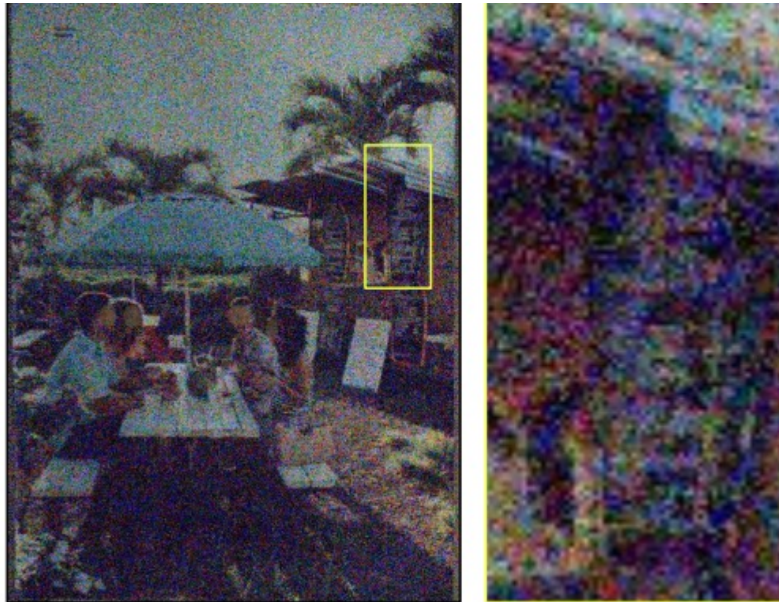
- All available license free signals (images) on the internet can be used for both signal-decomposition and signal-denoising task. This gives us access to a diverse signal-set with varying resolution and composition
- Gigapixel, Panoramic, and Deep Field Astronomy images would act as large workloads to push our implementation to its limits



Datasets and Testbed

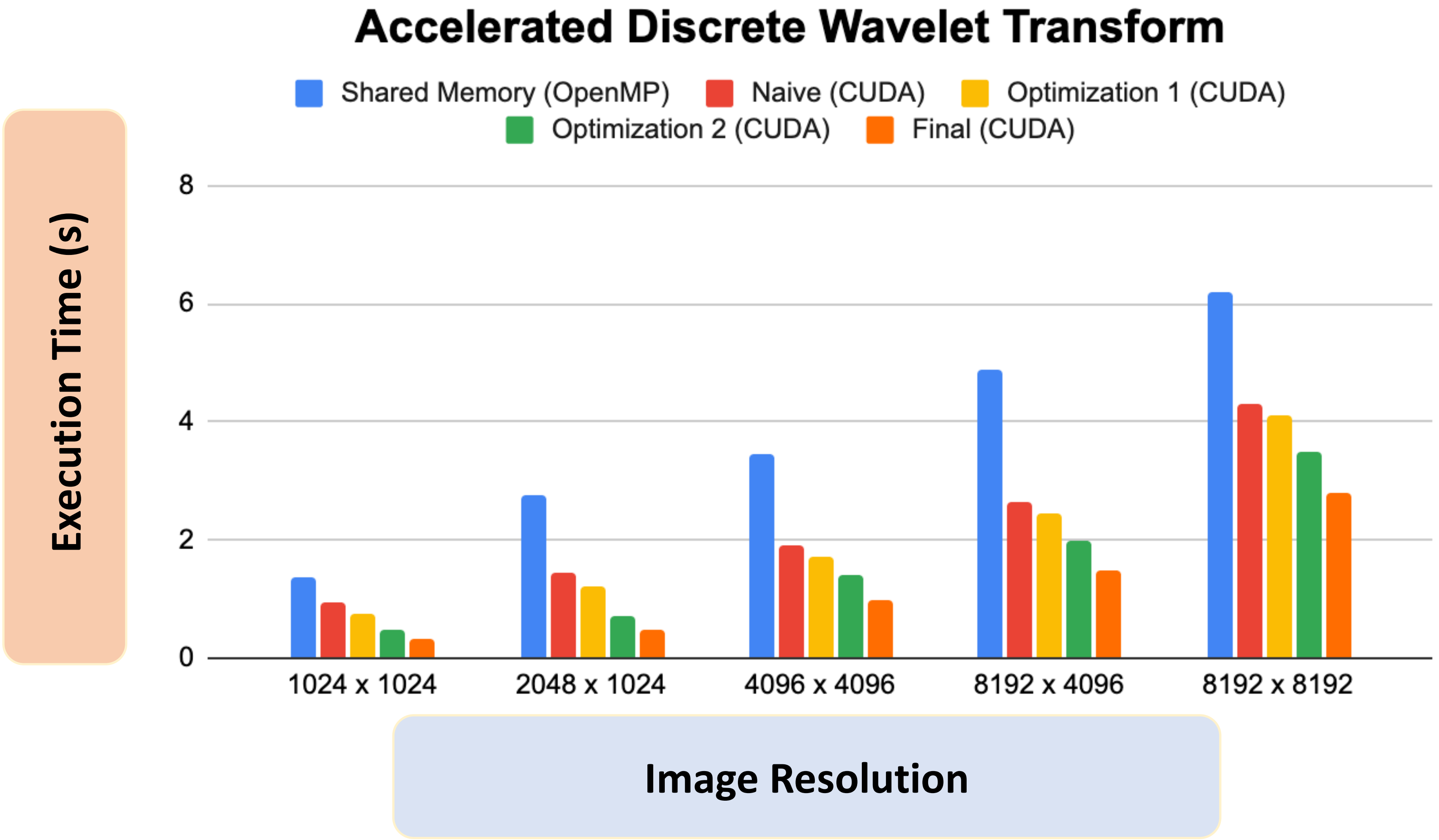
- Denoising literature and Kaggle have a large corpus of genuine noisy image datasets such as for low-light photography and raw sensor output. One can also generate noisy images synthetically
- We aim to test our implementation on NVIDIA V100 GPUs on the PACE cluster





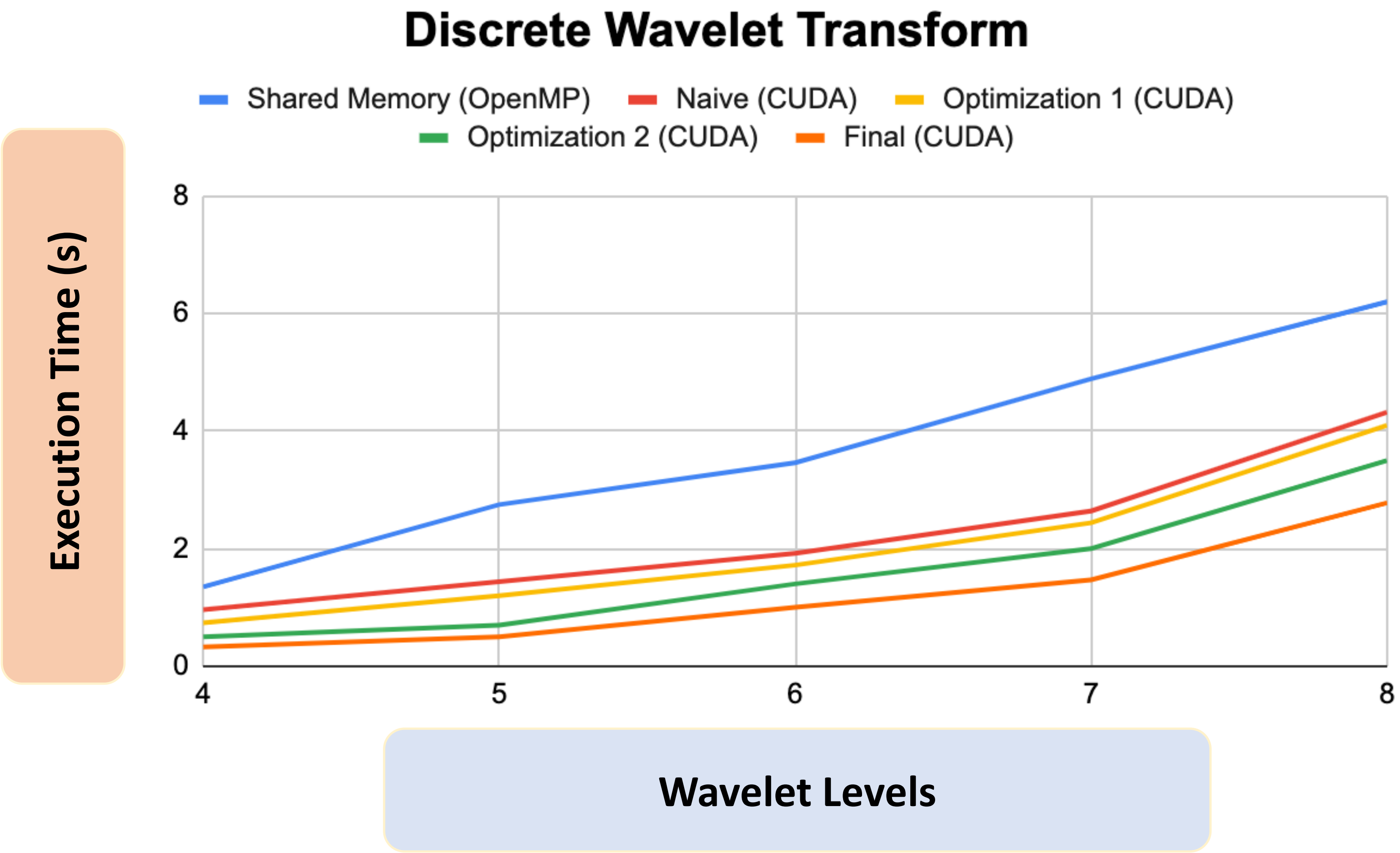
Performance Evaluation

- **Baseline**: Shared memory implementation using OpenMP



Performance Evaluation

- **Baseline**: Shared memory implementation using OpenMP



Thank you

CSE 6230 Project Proposal

Parallel Framework for Particle Dynamics Simulation

Category: Application

Yu Du
yudu@gatech.edu

Changhai Man
cman8@gatech.edu

Qiang Wu
qwu350@gatech.edu

Georgia Institute of Technology

March 2023

Motivation

Lots of applications calls for simulation frameworks about large-scale particle systems!

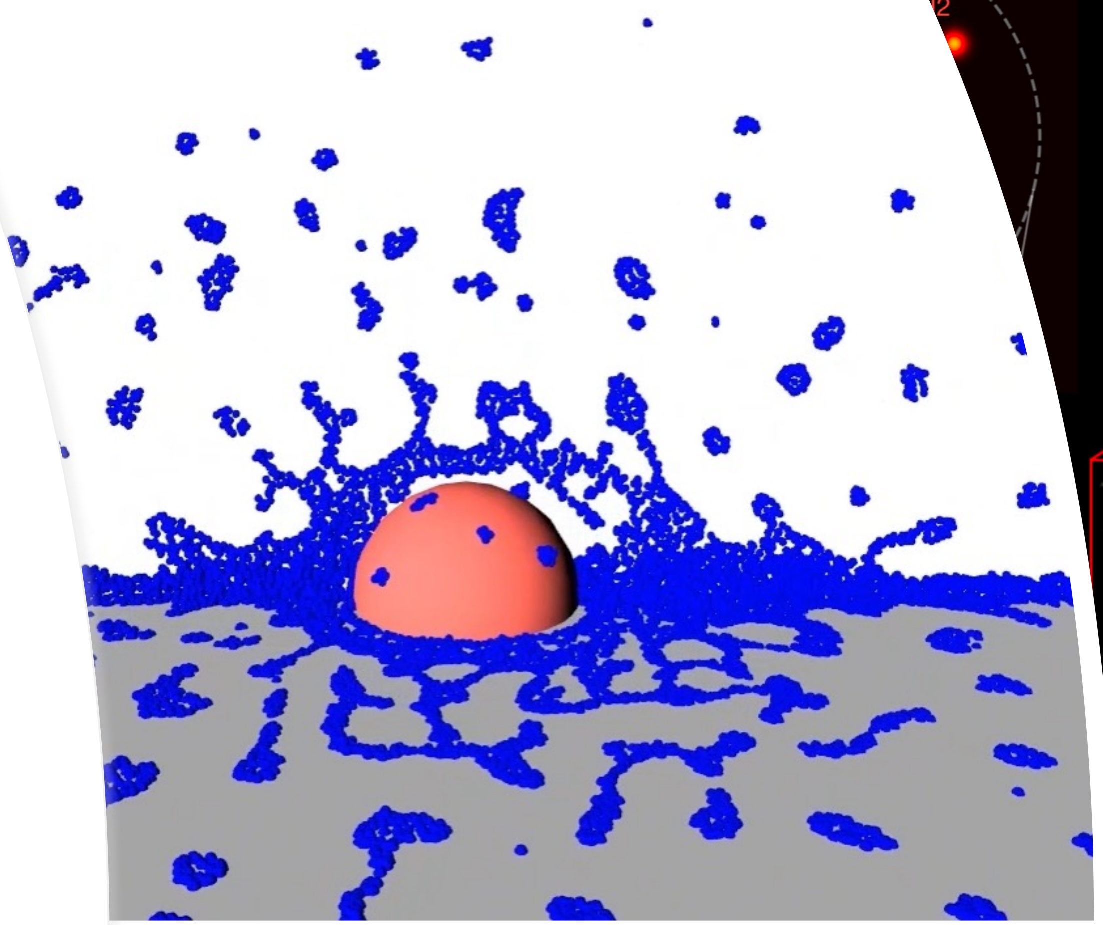
- Astronomy:

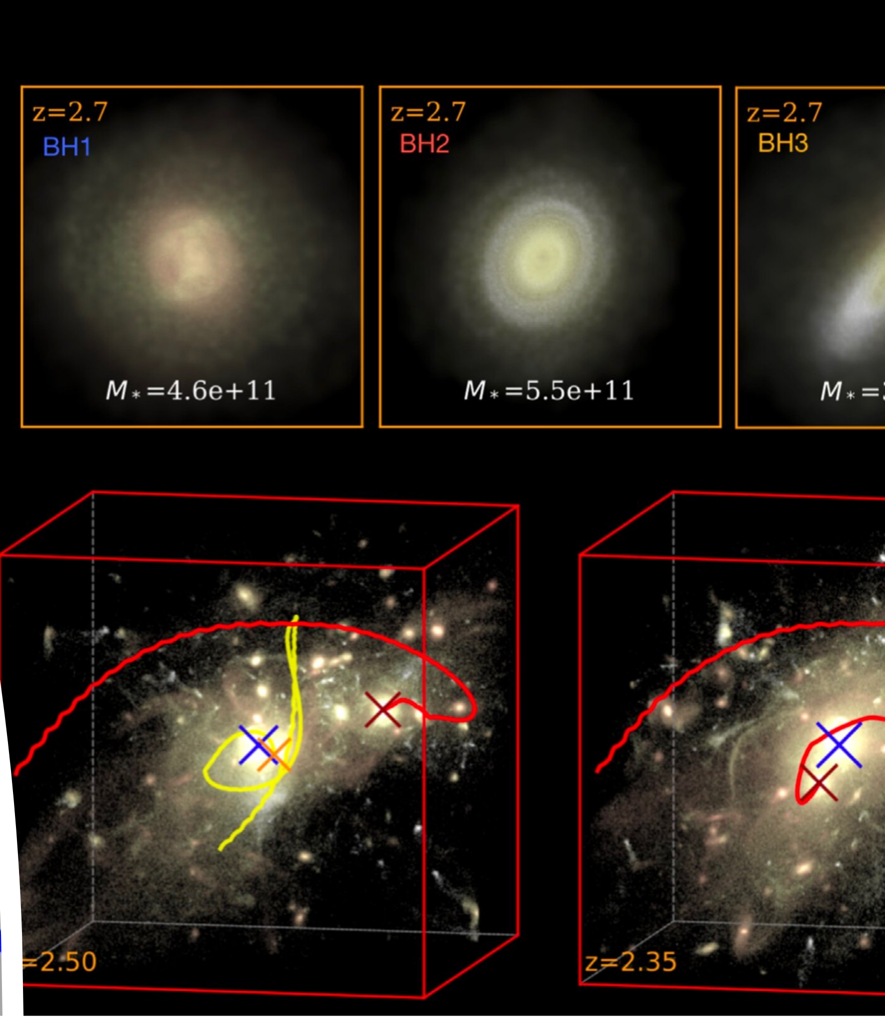
  Gravity simulations for galaxy systems.
- Chemistry & Biochemistry:

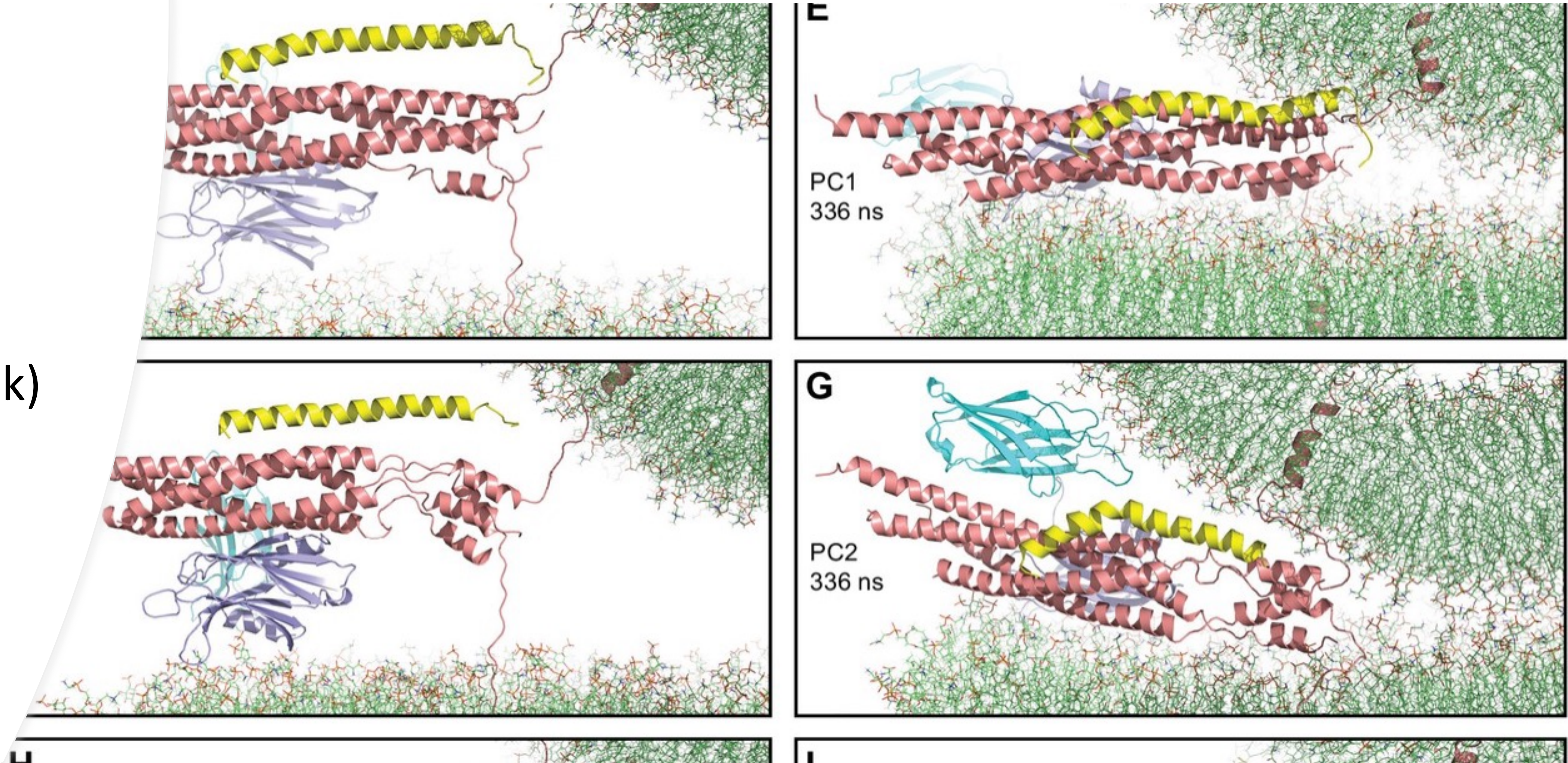
  Molecular dynamics
- Electro-Dynamics:

  Multiple particles moving under electromagnetic fields (Accelerator, Tokamak)
- Fluid-Dynamics:

  Smoothed Particle Hydrodynamics
- Many others...







Problem Abstraction

For particle dynamic systems, we have:

A set of particles $\mathbf{p} \in \boldsymbol{\Omega}$.

For each particle $\boldsymbol{p}$, it has the position(assume in 3d) $\boldsymbol{r}$, velocity $\boldsymbol{v}$, and some other attributes $\boldsymbol{s}$ (for example, massive, charge, temperature, box volume, etc.)

$$\boldsymbol{p} = (\boldsymbol{r}, \boldsymbol{v}, \boldsymbol{s}), \boldsymbol{r} = (x, y, z), \boldsymbol{v} = \left(v_x, v_y, v_z\right), \boldsymbol{s} = (m, q, \dots)$$

Between each particles, there is some interactions, for example, gravity, Coulomb, Van der waals force.

$$\begin{bmatrix} f(\boldsymbol{p}_1, \boldsymbol{p}_2) \cdots f(\boldsymbol{p}_1, \boldsymbol{p}_k) \\ \vdots \qquad \cdots \qquad \vdots \\ f(\boldsymbol{p}_k, \boldsymbol{p}_1) \cdots f(\boldsymbol{p}_k, \boldsymbol{p}_k) \end{bmatrix}$$

These interactions to one particle can be merged:

$$f(\boldsymbol{p_k}, \boldsymbol{\Omega}) = \sum_i f(\boldsymbol{p}_k, \boldsymbol{p}_i)$$

And the interactions will affect the position of each particles along the time under certain timestep Δt:

$$\boldsymbol{v_k}(t + \Delta t) = \boldsymbol{v}_k(t) + \frac{f(\boldsymbol{p}_k, \boldsymbol{\Omega})}{m} \Delta t$$

$$\boldsymbol{r_k}(t + \Delta t) = 2\boldsymbol{r}_k(t) - \boldsymbol{r}_k(t - \Delta t) + \frac{f(\boldsymbol{p}_k, \boldsymbol{\Omega})}{m} \Delta t^2$$

Proposed Solution

All particles inside a box with/without Periodic Boundary Conditions (PBC)

Initial velocities and positions in shared memory; each GPU thread assess a local set of them

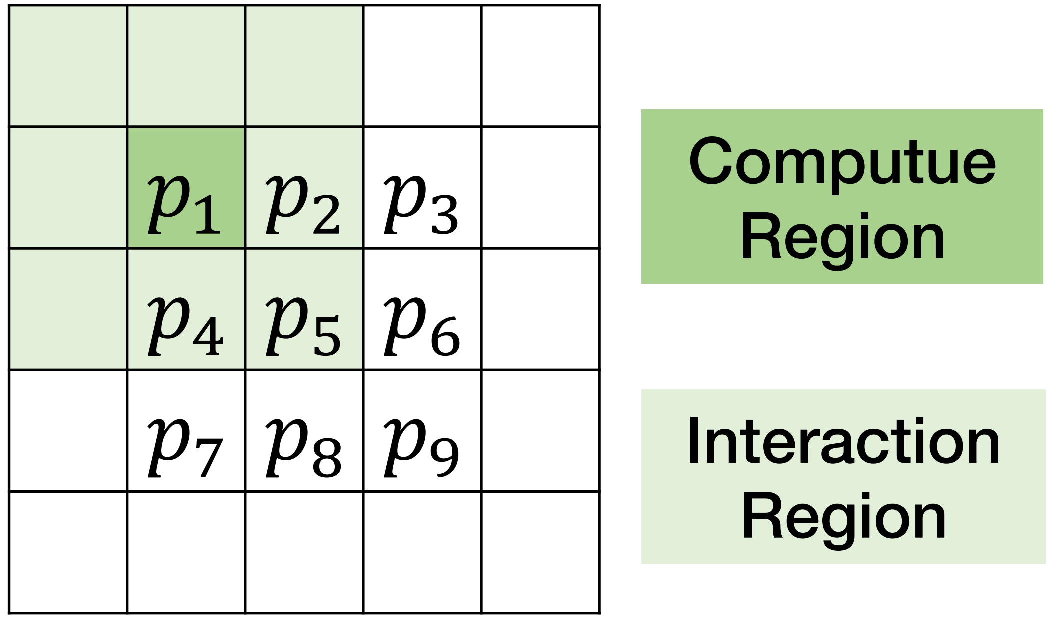
Simplification:

1. Use cutoff, outside which the forces can be neglected
2. Verlet Integrator on position, Leap-Frog Integrator on velocity

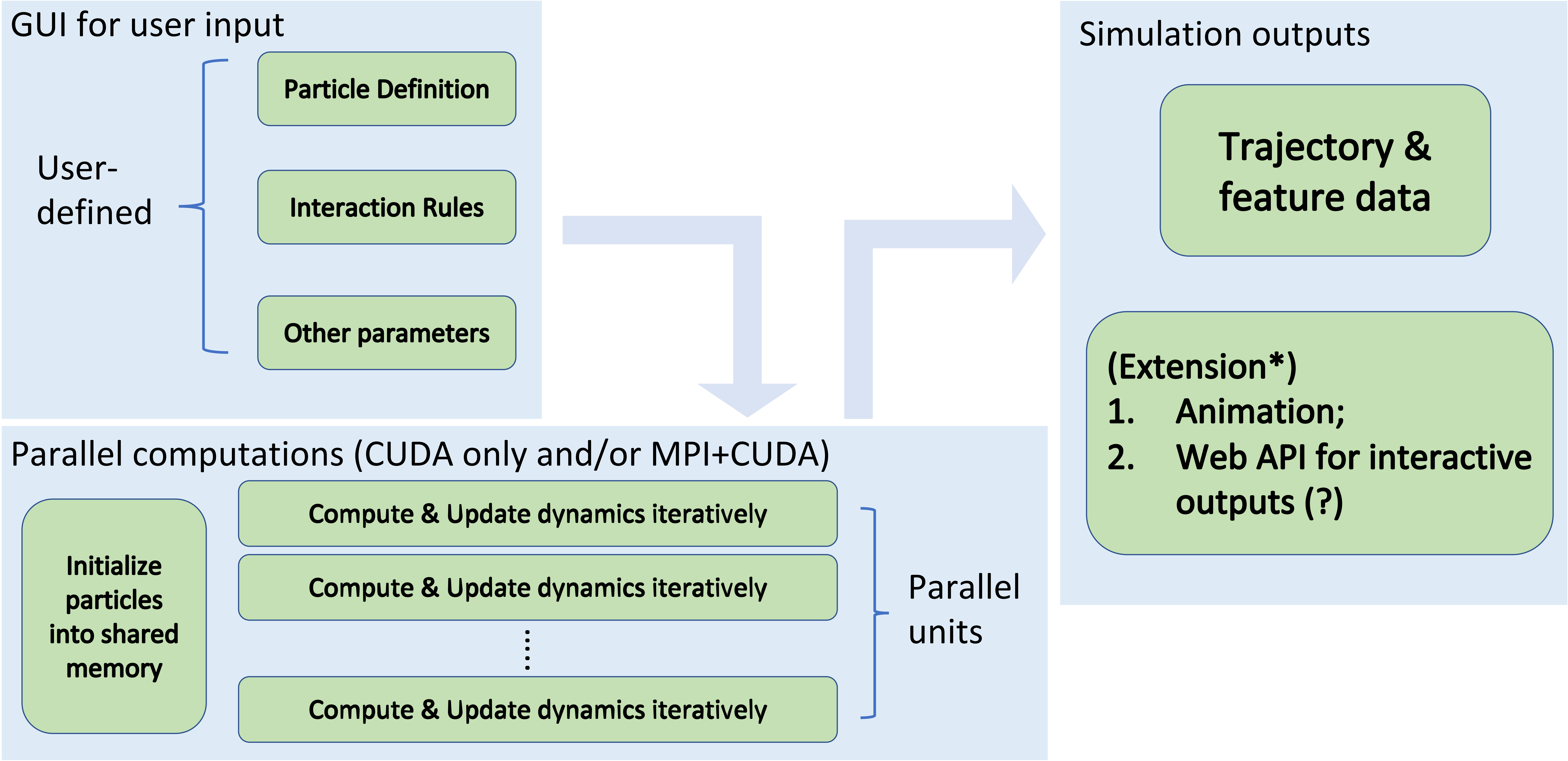
Which GPU block assess which subset of particles?

I. (CUDA-aware) MPI: Those particles having interaction should be on the same GPU (cut the computation by half)

II. Each block in charge of a predetermined partition of the box (less communication)



Framework Overview



GUI for user input

User-defined

Particle Definition

Interaction Rules

Other parameters

Parallel computations (CUDA only and/or MPI+CUDA)

Initialize particles into shared memory

Compute & Update dynamics iteratively

Compute & Update dynamics iteratively

⋮

Compute & Update dynamics iteratively

Parallel units

Simulation outputs

Trajectory & feature data

(Extension*)

1. Animation;
2. Web API for interactive outputs (?)

Experiment Setup

- Testbed: PACE
- Validation: 1. Compare with the sequential version of our algorithm

  2. Check if results comply with Van der waals equation $\left(P+\frac{n^2a}{V^2}\right)(V-nb)=nRT$

  3. Output data alignment with respect to cutoffs
- Baseline (benchmark): To be determined, will be from one or more of the three categories

  1. Public academic / business repositories (e.g., this simulation demo)

  2. Published simulation software (e.g., PDPS)

  (Both of which can find sequential / parallel realizations)

  3. Reproduce paper results for comparison
- Datasets: Randomly Initialized (will refer to public datasets like PubChem)
- Metrics:

  1. Speedup=$\frac{T(n,1)}{T(n,p)}$ (compare our computation time with other solutions)

  2. Roofline Model

  3. Produce strong scaling plot

Summary

- Category: Application
- Problem: Large-scale Particle Dynamics Simulation
- Performance Metric: Speedup + Roofline Model
- Baseline: Our sequential version
- Solution:

  1) MPI+CUDA. 2) Partition with cut-off range.

  3) Periodic re-partition for tradeoff of acc & perf.
- Validation:

  1) Try to find some benchmark.

  2) If no benchmark, try to simulate some real-world phenomenon.

  3) Find some constant variable in system, try to verify it so not change during simulation.
- Test bed: PACE COC-ICE, AWS if possible (might need some sponsorship LOL)
- Potential plots:

  1) Speedup and Strong Scaling. 2) Performance breakup

  3) Simulation results rendering.

Q&A

Yu Du
yudu@gatech.edu

Changhai Man
cman8@gatech.edu

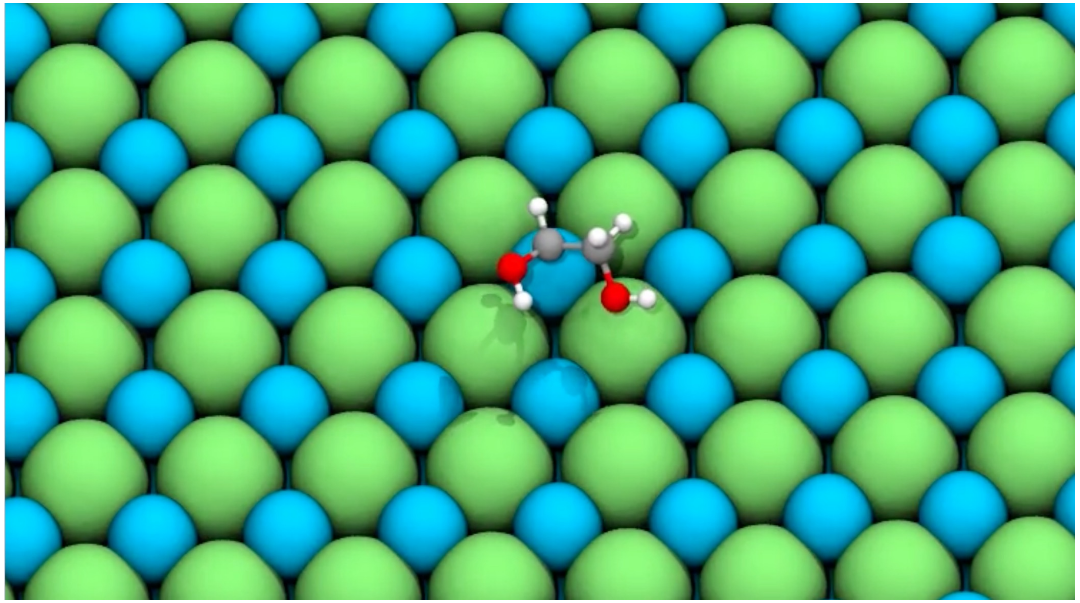
Qiang Wu
qwu350@gatech.edu

Georgia Institute of Technology

March 2023

CSE6320: High Performance Parallel Computing

Distributed Hyperparameter Tuning for Machine Learning Potentials in Catalysis

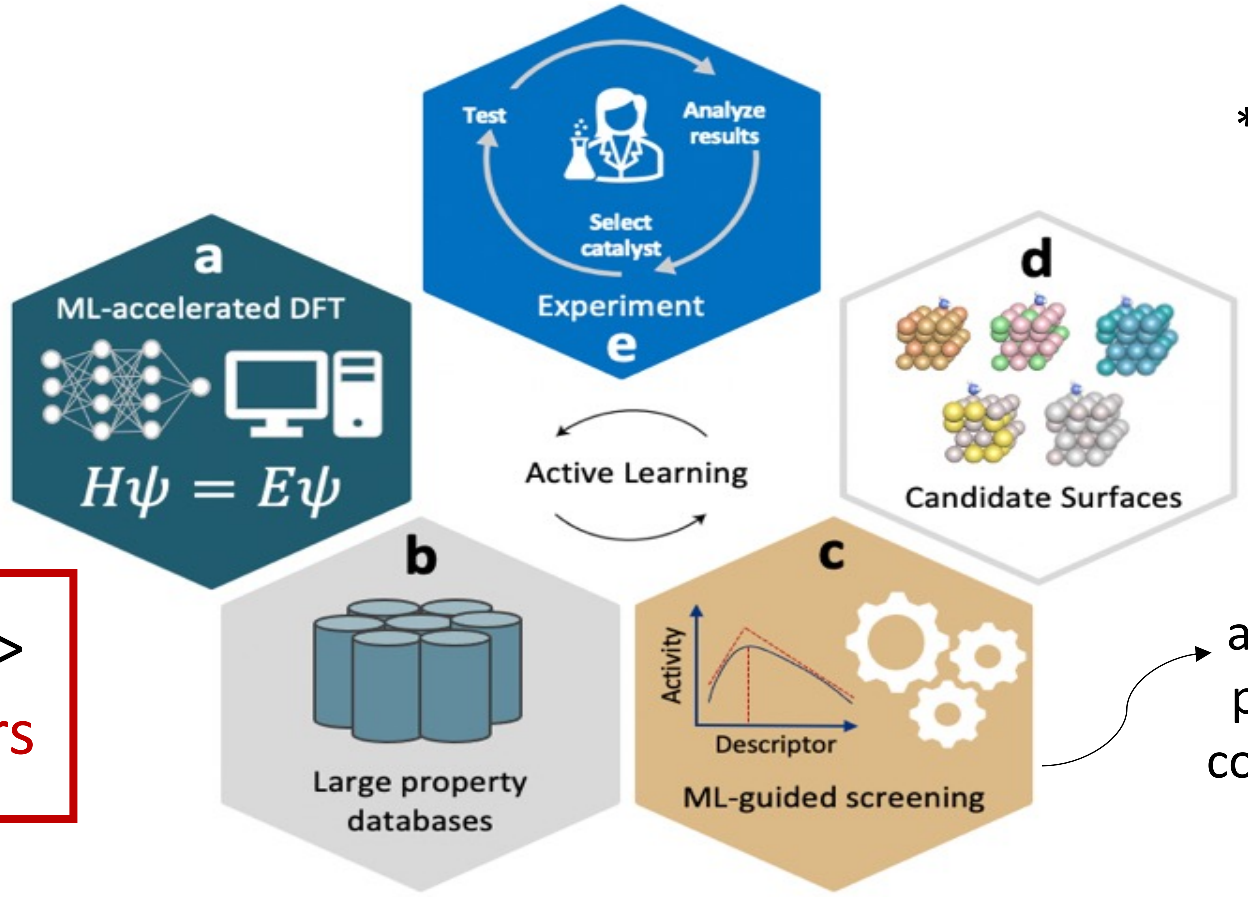


Project Proposal
March 14, 2023

Omar Jiménez

Motivation: Data-Driven Catalyst Discovery Workflow

*many variations, this is just one example

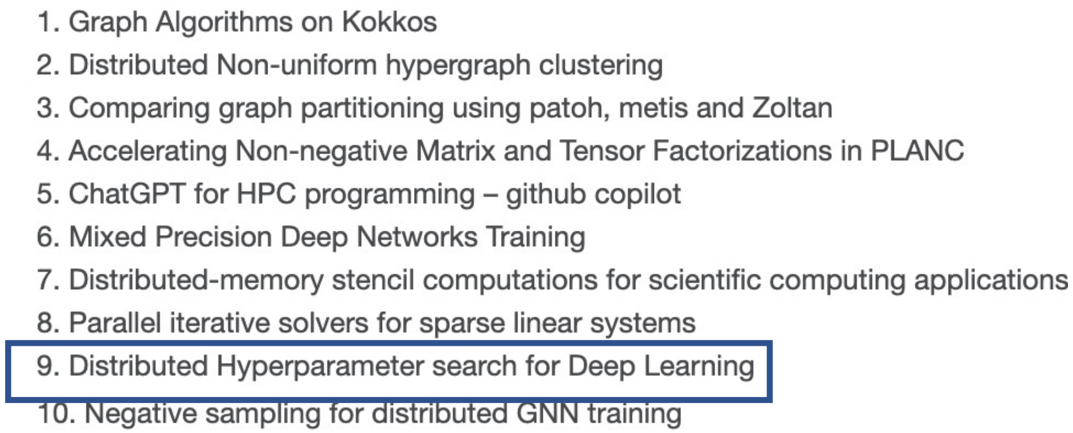


ML -> Training -> Hyperparameters

also involves ML, but not planning to focus on this component for this project

Project

Original idea from Piazza post on 03/08/23



1. Graph Algorithms on Kokkos
2. Distributed Non-uniform hypergraph clustering
3. Comparing graph partitioning using patoh, metis and Zoltan
4. Accelerating Non-negative Matrix and Tensor Factorizations in PLANC
5. ChatGPT for HPC programming – github copilot
6. Mixed Precision Deep Networks Training
7. Distributed-memory stencil computations for scientific computing applications
8. Parallel iterative solvers for sparse linear systems
9. Distributed Hyperparameter search for Deep Learning
10. Negative sampling for distributed GNN training

Project Type: ***Application.*** I will integrate distributed hyperparameter search algorithms into training of ML potentials relevant to data-driven catalyst discovery workflows

Problem Statement

Problem: Depending on the search space, search algorithms for hyperparameter tuning can be *computationally expensive* (from hours to days). State-of-the-art packages for training of ML potentials (e.g., SchNetPack, AMPTorch, etc.) do not support distributed hyperparameter tuning.

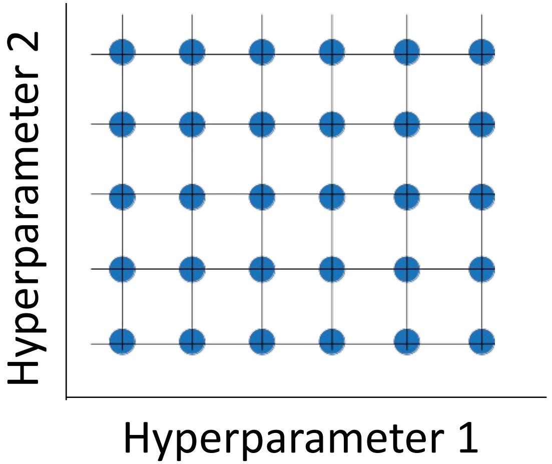
Solution: *Distributed* Hyperparameter Search

Performance Category: Time to solution

Solution

Solution: *Distributed* Hyperparameter Search. Different algorithms for distributed search might be considered.

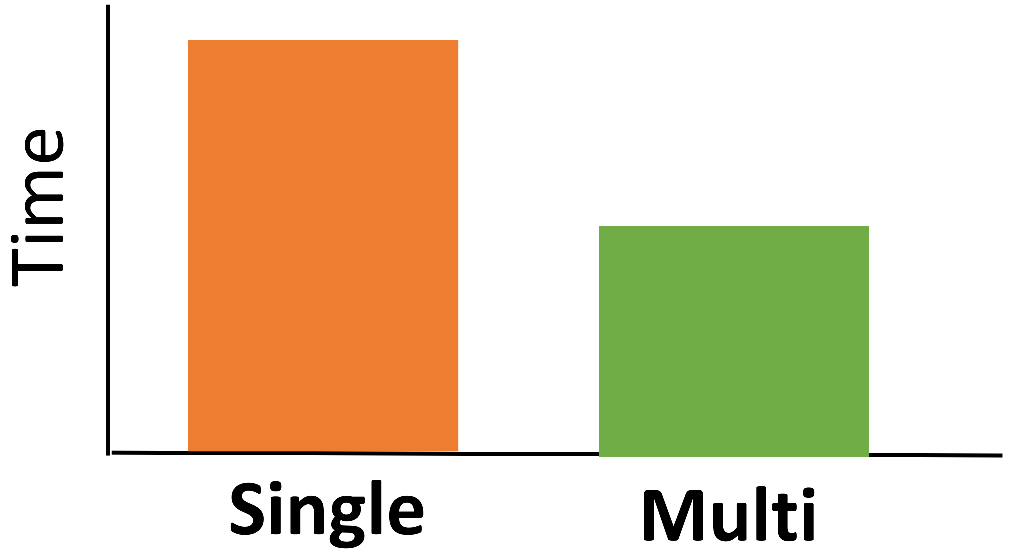
Important: Parallelization "friendliness" (i.e., parallel algorithms already exist or easily adapted – inventing algorithms is outside scope of project).



Grid Search classical example and very easy to distribute (independency of computations). Other algorithms might be considered.

Baselines

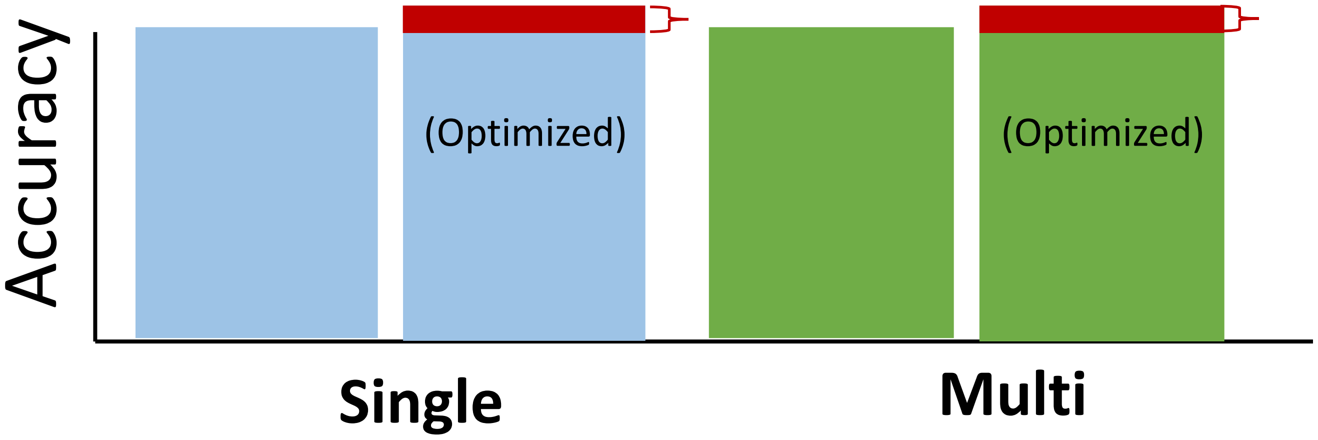
Performance Category: Time to solution



Validation

Important: Some Hyperparameter Search Algorithms are *not deterministic* and may not return the same “optimal hyperparameters” between trials.

Validation: Relative model accuracy improvement w.r.t. sequential output

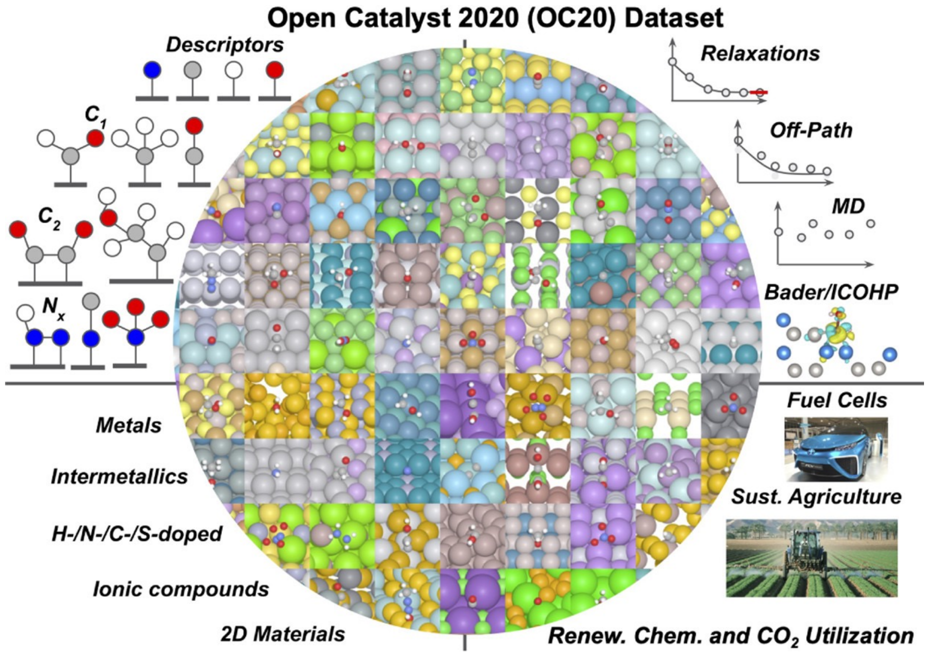


*If “deterministic” (e.g., Grid Search), direct validation could be conducted in theory. However, even for same set of hyperparameters, this is still subject to a small amount of variance in training (depending on ML algorithms).

Experiments

Dataset

- Open Catalyst 2020 (OC20) Dataset
- Dataset of DFT calculations for catalysis systems with high chemical diversity
- **Small chunk** of it. Full dataset has *millions* of data points which is not feasible given resource constraints



Experiments

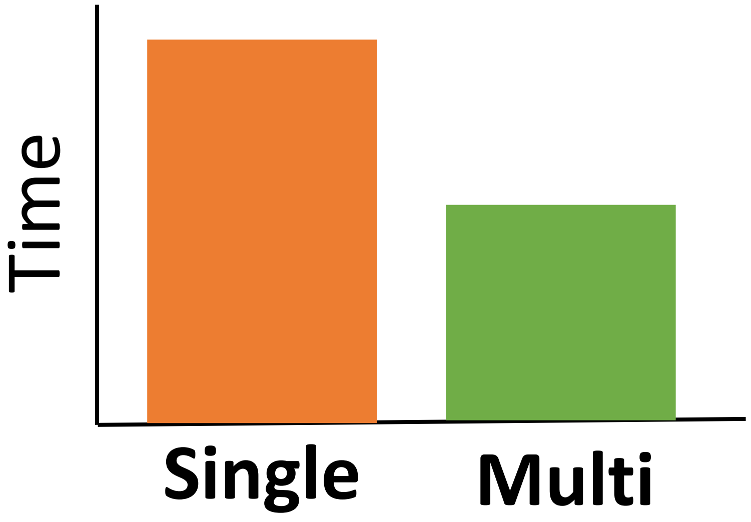
Testbed

- Either PACE or cloud services (AWS/Azure/GCP)
- PACE usage ultimately depends on whether I can install all necessary dependencies to run the codes
- PACE should be possible since I have successfully installed some of the packages already (e.g., Atomic Simulation Environment library)

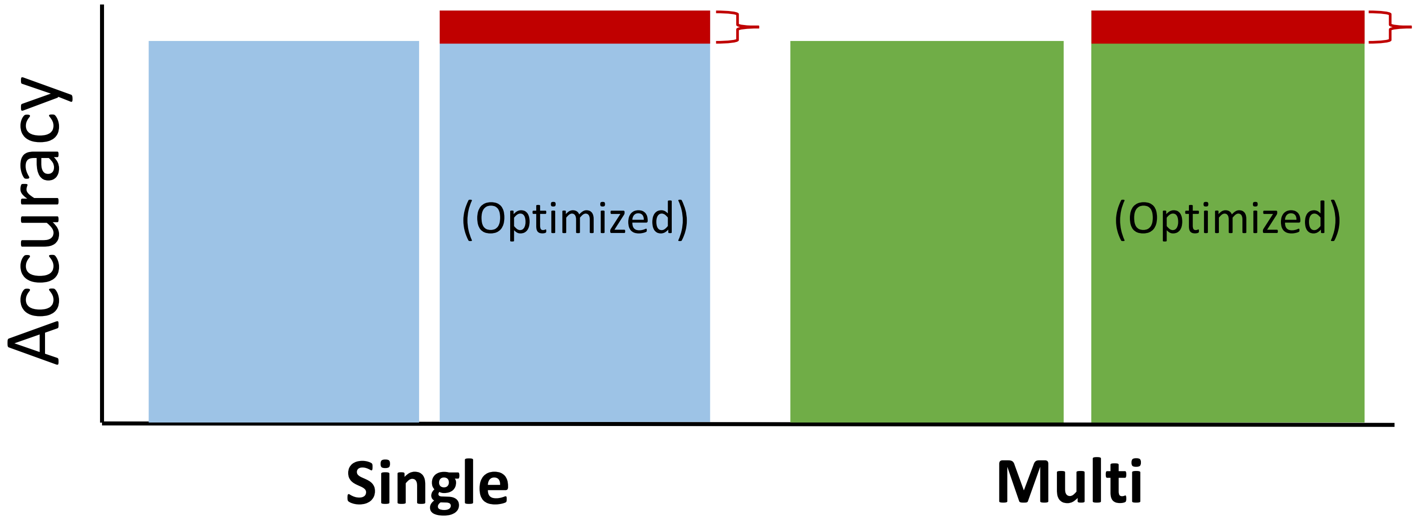
Experiments

Results Validation

1. Faster?



2. Model accuracies comparable?



Accelerate Tensor Computation Leveraging TVM and SIMD on ARM CPU

Fan Qu, Peidi Song

Georgia Tech

CONTENTS

1 Category

2 Problem

3 Performance Metric

4 Baselines

5 Solution

6 Validation

7 Datasets

8 Platforms

9 Potential Plots

10 References

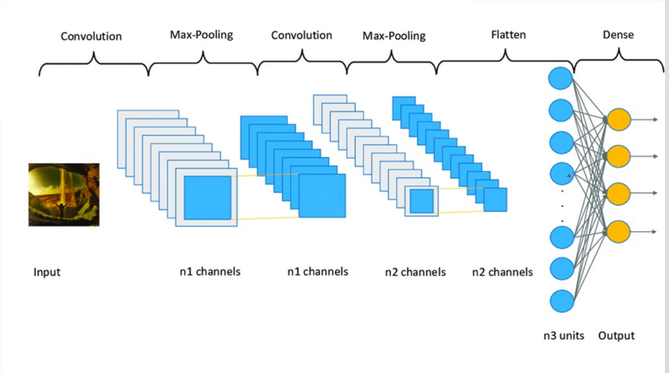
Category

- Application
  - In detail, our focus is on developing a practical solution to accelerate tensor computation

Problem

- Tensor computation
  - Widely used in neural networks
  - E.g., GEMM, Convolution, Nomalization
  - Computationally expensive and time-consuming, particularly on ARM CPUs

- Goal
  - Accelerate tensor computation on ARM CPUs
  - Leverages TVM and SIMD technology



Performance Metrics

- Runtime for data of different of size
- Evaluate different tensor operations

Baselines

- PyTorch
- TensorFlow
- AutoTVM
- Ansor
- …

Solution

TVM

- A Python interface end-to-end compiler framework for CPU, GPU, and accelerators
- Seperate computation and optimization
  - Define computation
    - C = sum(A[i,k]*B[k,j], reduce=k)
  - Define schedule primitive
    - io, ii = split(i, factor=8)
    - reorder(io, jo, ko, ki, ii, ji)

Solution

NEON SIMD Instructions

arm Developer

- NEON is a technology that enables parallel processing on ARM CPUs.
- Vectorization: processing multiple elements of the tensors at the same time
- Low-Level Optimization: loop unrolling and memory alignment to maximize the performance

Solution

- Optimization steps
  - Tensor computation definition from mathematical formula
  - Handwritten NEON SIMD vectorized kernel
  - TVM schedule primitives for blocking (tiling)
  - Integrate them and generate optimized codes

Validation

We will use the calculation result of PyTorch as the ground truth.

Datasets

- Different operators
  - GEMM
  - Convolution 2D
  - BatchNorm
  - ...
- Different tensor sizes
- Generate tensors randomly

Platforms

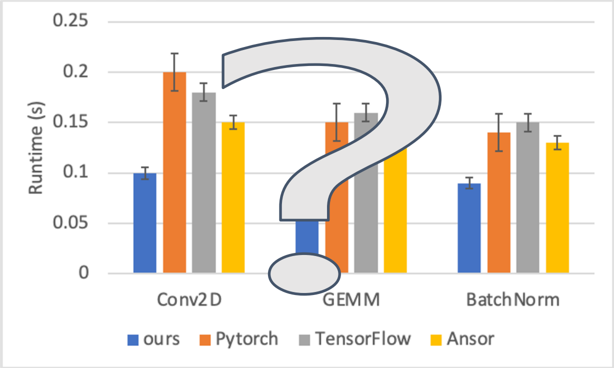
We plan to optimize common CPUs for consumers and servers.

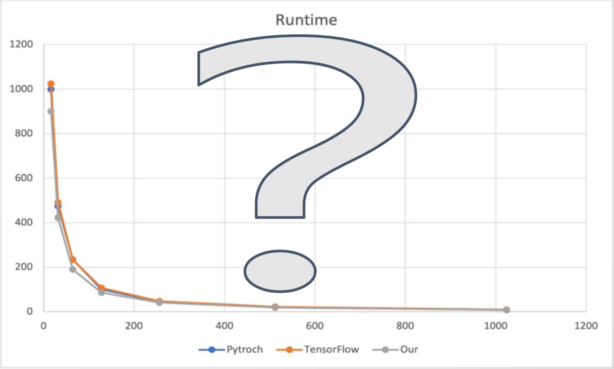
- Apple Silicon
- AWS Graviton processors

Potential Plots

- Performance Comparison Bar Charts
  - x-axis: operator × size × method
  - y-axis: runtime

- Performance Comparison Line Charts
  - x-axis: size
  - y-axis: runtime
  - lines: different methods





References

- Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." Advances in neural information processing systems 32 (2019).
- Abadi, Martín, et al. "Tensorflow: a system for large-scale machine learning." Osdi. Vol. 16. No. 2016. 2016.
- Chen, Tianqi, et al. "TVM: An automated end-to-end optimizing compiler for deep learning." arXiv preprint arXiv:1802.04799 (2018).
- Chen, Tianqi, et al. "Learning to optimize tensor programs." Advances in Neural Information Processing Systems 31 (2018).
- Zheng, Lianmin, et al. "Ansor: Generating high-performance tensor programs for deep learning." Proceedings of the 14th USENIX Conference on Operating Systems Design and Implementation. 2020.

Thanks