# Discrete Space Hartree-Fock GPU Acceleration

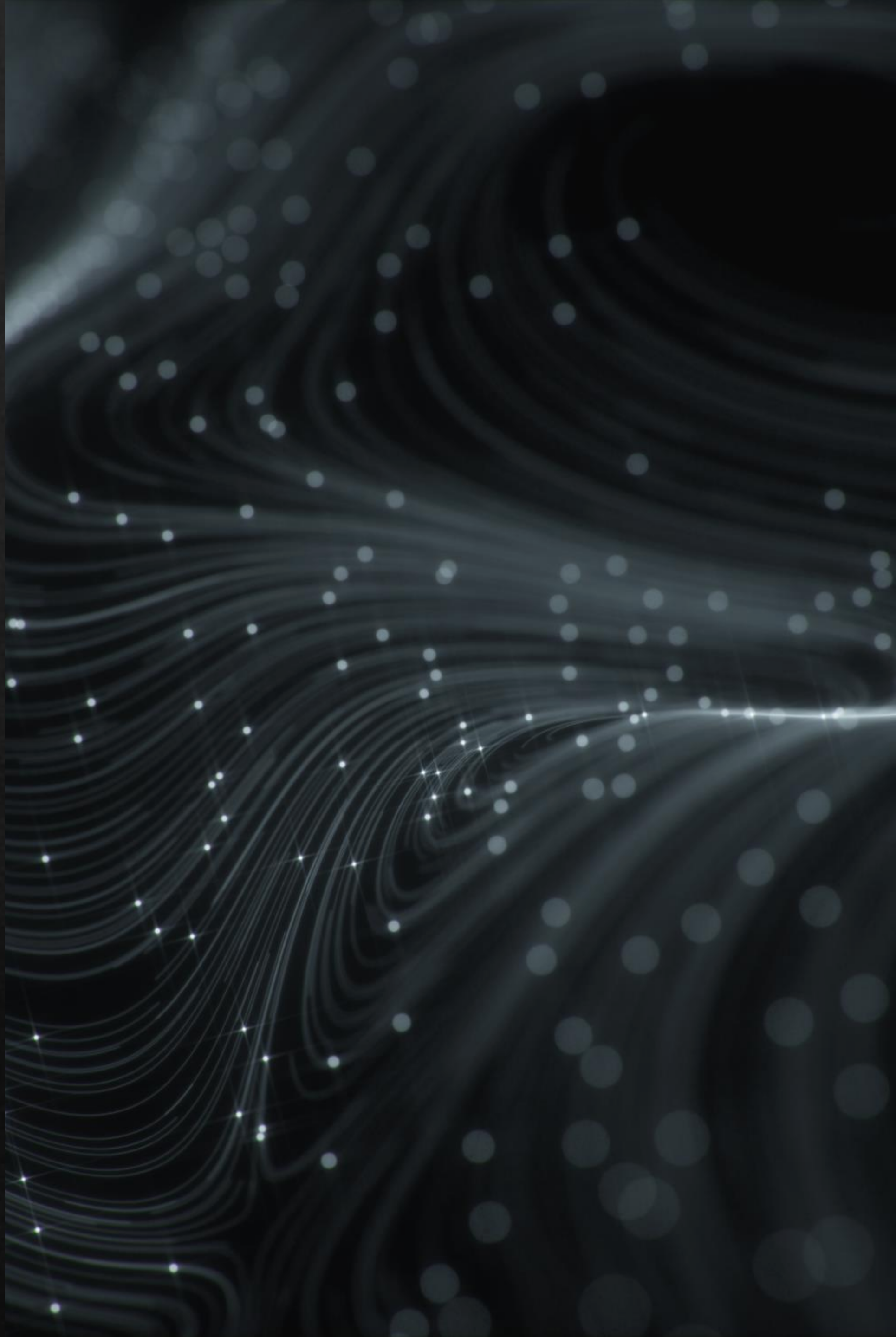
ELEC 542 Project Presentation

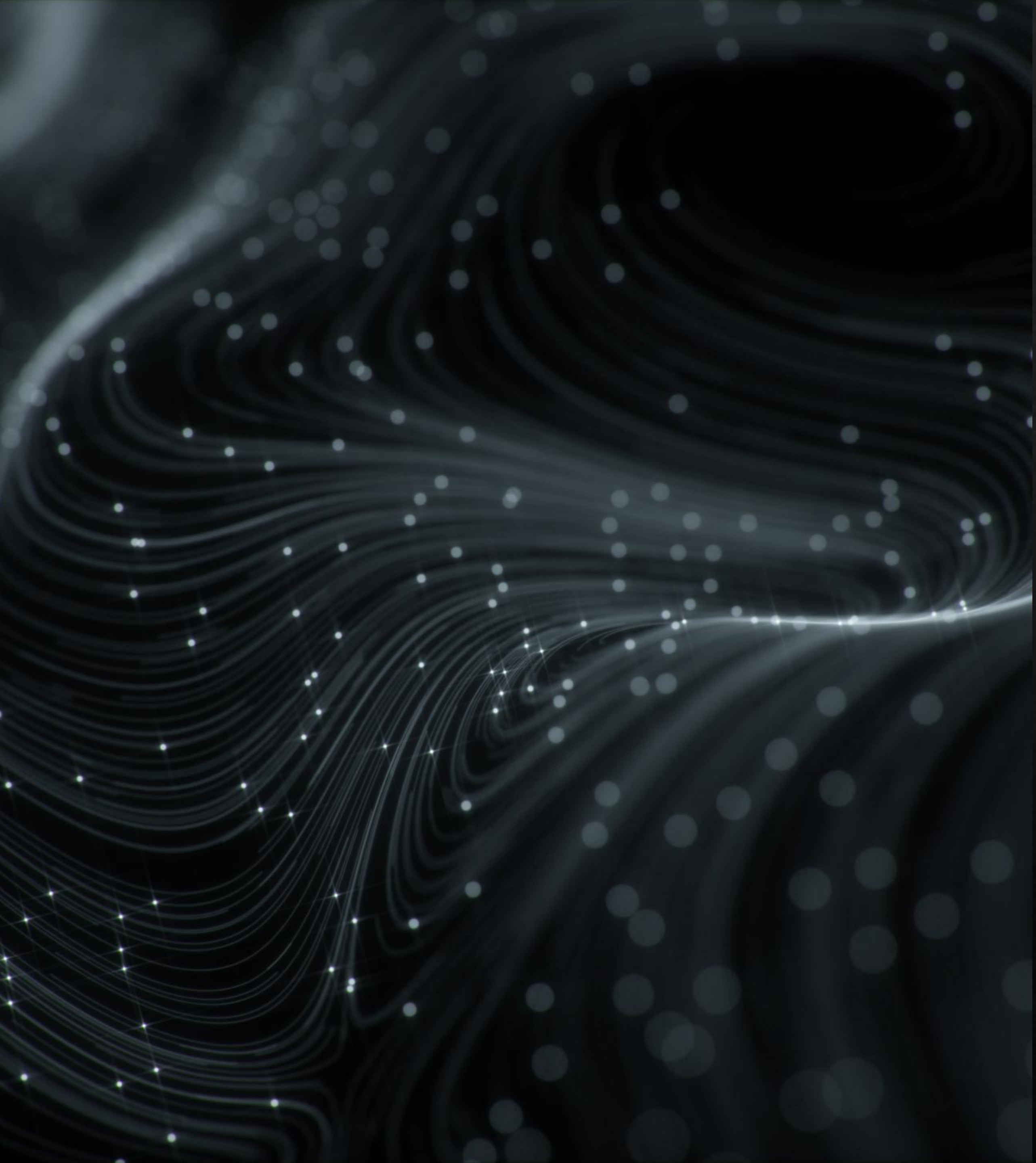
Michel Kakulphimp

# Introduction to GPUs

# Introduction to GPUs

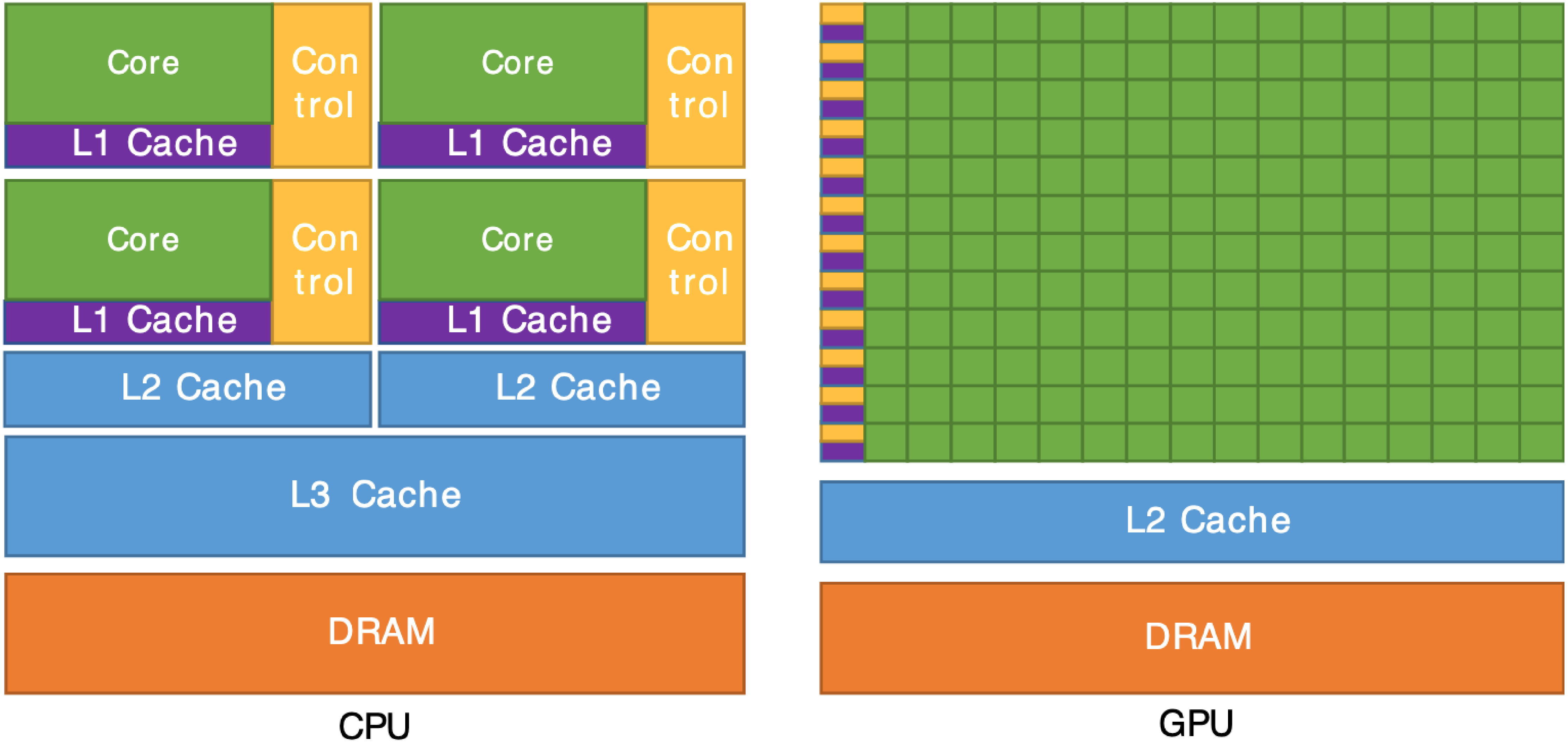
- Primarily designed for real-time computer graphics
- Well suited for highly parallelized computational workloads
- High memory bandwidth and many cores (RTX 3090 - 10496 cores!)
- Dependent on a host CPU, connected via high-speed high-bandwidth interconnect
- Different machine code than CPU, requires its own build tools
- Many GPU HW blocks are generic – just need to formulate problem differently
- Non-graphics workloads are a possibility and quite common
- Manufacturers provide APIs for development (NVIDIA CUDA used in this project)





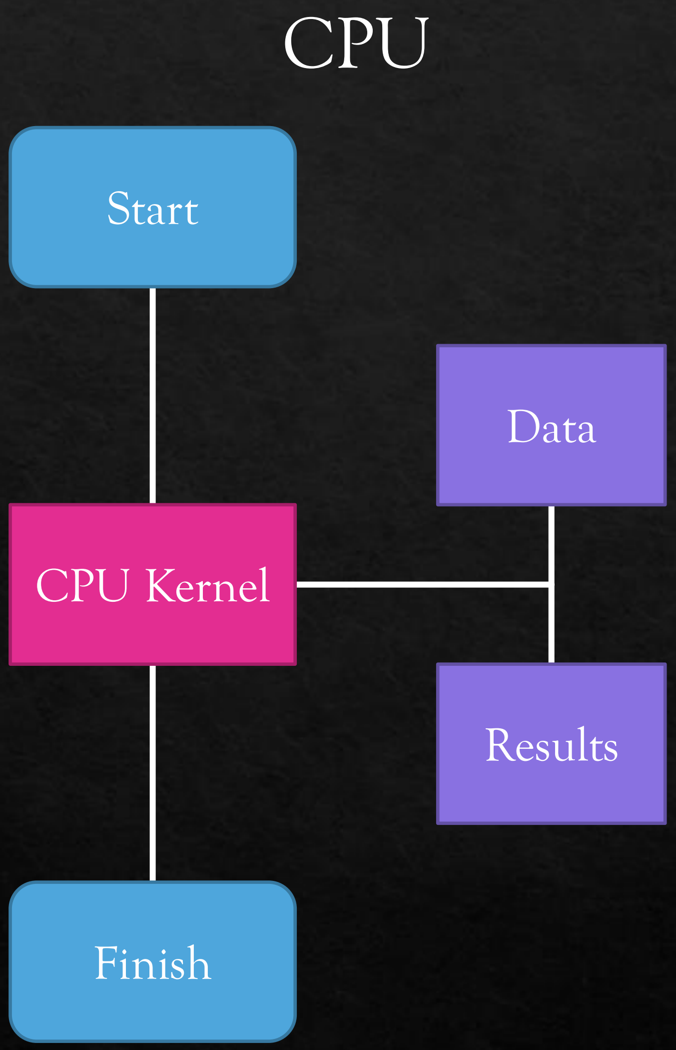
# GPU/CPU Comparison

- GPU is extremely parallel
  - Many more processing units (albeit weaker)
  - Optimized for SIMD instructions (i.e. one instruction – many data)
  - More transistors dedicated to ALU and FPUs
- CPU more versatile
  - Larger instruction set
  - Better serial execution
  - Lower latency
  - Larger/more caches

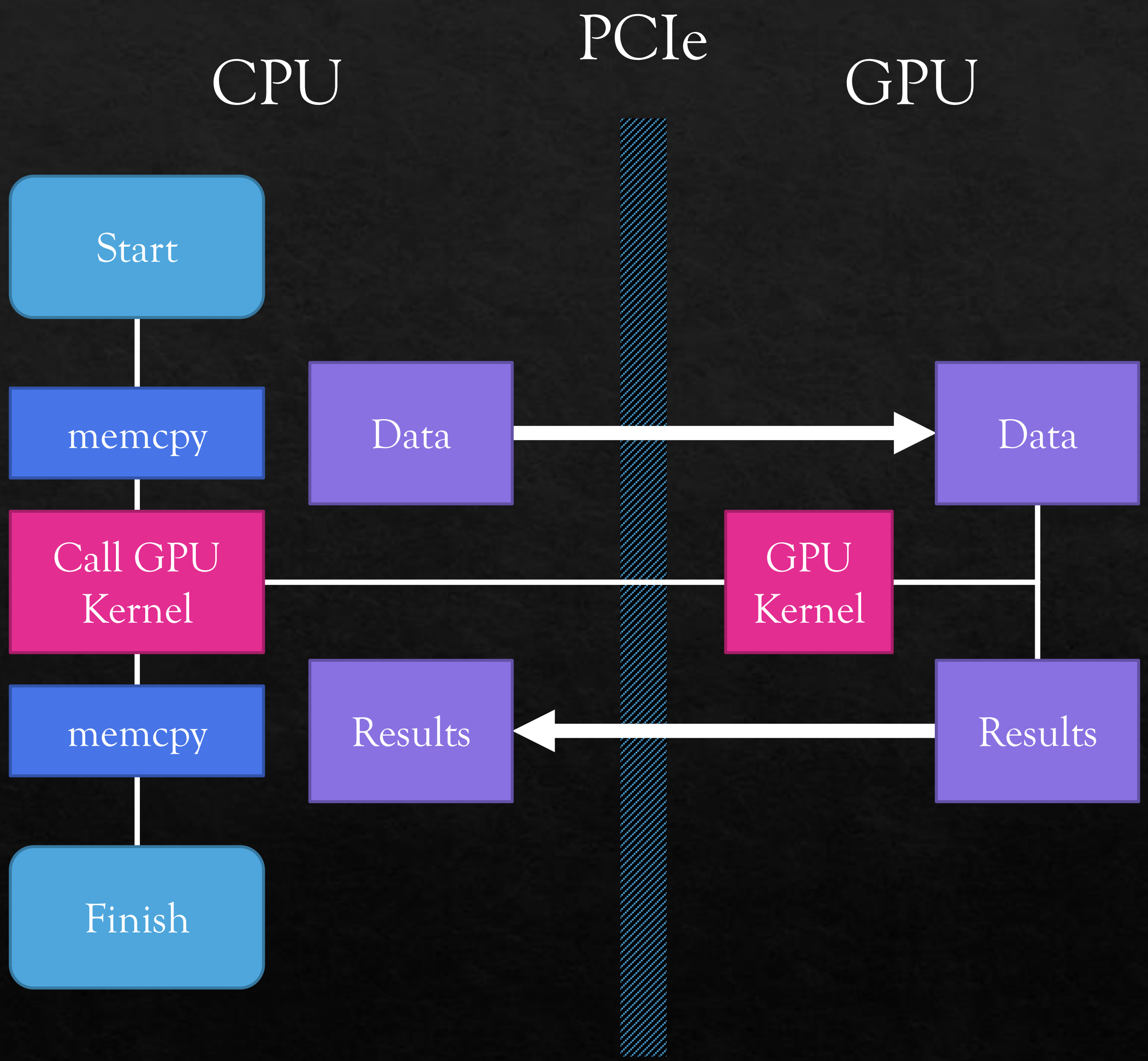

Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
Core
Con
trol
L1 Cache
L2 Cache
L2 Cache
L3 Cache
DRAM
CPU
L2 Cache
DRAM
GPU

# GPU Development Considerations

- Parallel Efficiency
  - Proper identification of parallelizable algorithmic components
  - Occupying as many GPU cores as possible
- Memory Throughput
  - Data must be well managed to ensure latency can be hidden
  - Moving data between CPU and GPU
- Instruction Throughput
  - Ensuring GPU cores are kept busy
  - Severe performance penalties from data dependencies (affects parallel efficiency as well)




CPU-only Execution




CPU-CPU Execution

# GPU Accelerating Hartree-Fock

$$\hat{F}(\vec{r})\psi_n(\vec{r}) = \epsilon_n\psi_n(\vec{r})$$

$$\hat{F}(\vec{r}) = \hat{H}_{core}(\vec{r}) + \sum_{n=1}^{N/2}\left[2J_n(\vec{r}) - K_n(\vec{r})\right]$$

$$\hat{F}(\vec{r})\psi_n(\vec{r}) = \epsilon_n \psi_n(\vec{r})$$

$$\hat{F}(\vec{r}) = \hat{H}_{\cancel{core}}(\vec{r}) + \sum_{n=1}^{N/2} \left[2J_n(\vec{r}) - K_n(\vec{r})\right]$$

$$\hat{F}(\vec{r})\psi_n(\vec{r}) = \epsilon_n \psi_n(\vec{r})$$

$$\hat{F}(\vec{r}) = \hat{H}_{\cancel{core}}(\vec{r}) + \underbrace{\sum_{n=1}^{N/2} \left[2J_n(\vec{r}) - K_n(\vec{r})\right]}_{\text{Numerical Integration}}$$

$$\underbrace{\hat{F}(\vec{r})\psi_n(\vec{r}) = \epsilon_n \psi_n(\vec{r})}_{\text{Eigensolver}}$$

$$\hat{F}(\vec{r}) = \cancel{\hat{H}_{core}(\vec{r})} + \underbrace{\sum_{n=1}^{N/2} \left[2J_n(\vec{r}) - K_n(\vec{r})\right]}_{\text{Numerical Integration}}$$

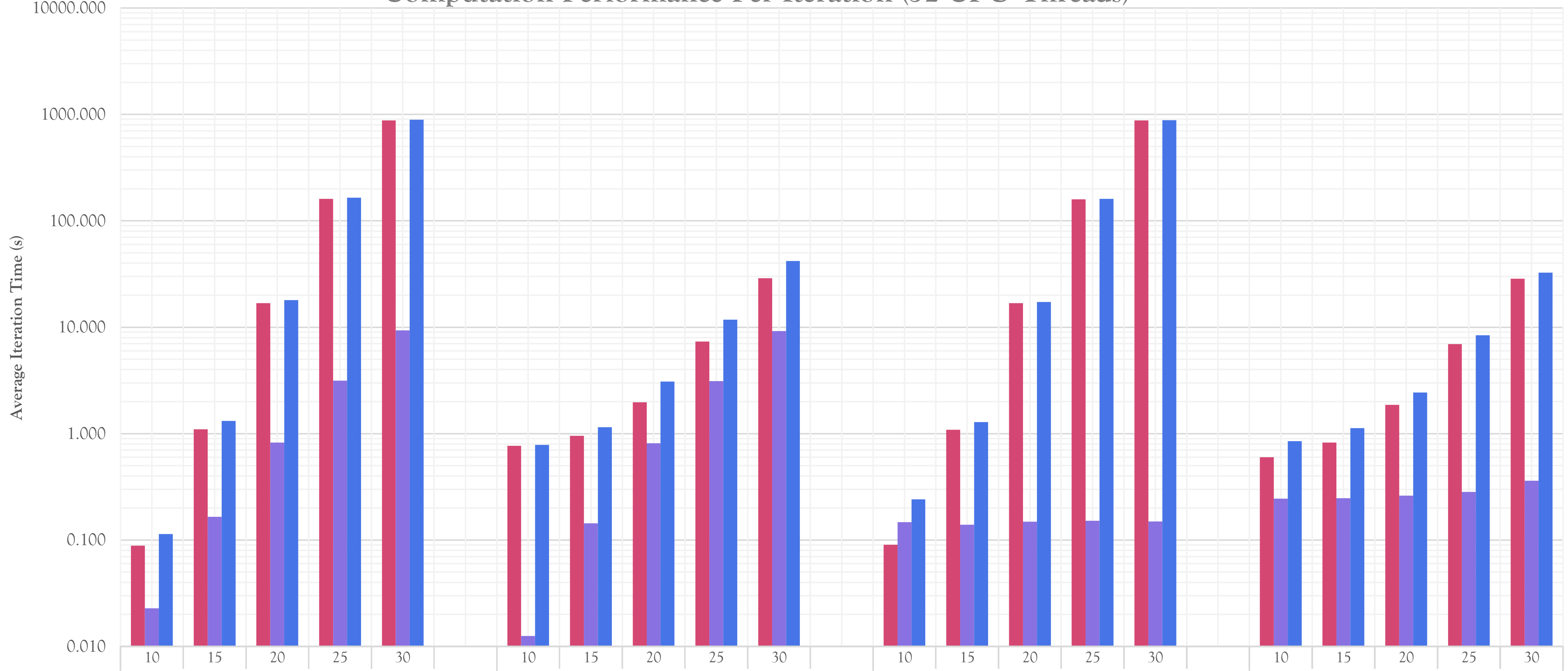
# GPU Accelerating Hartree-Fock

- Fixed original Python implementation
- Ported Python implementation to C++ (BLAS/LAPACK used for eigensolver)
- Profiled C++ application
- Identified CPU-intensive sections (compounded by iterations)
- Re-implemented CPU-intensive sections as offloaded GPU workload
- Re-profiled with accelerated sections

# GPU Accelerating Hartree-Fock

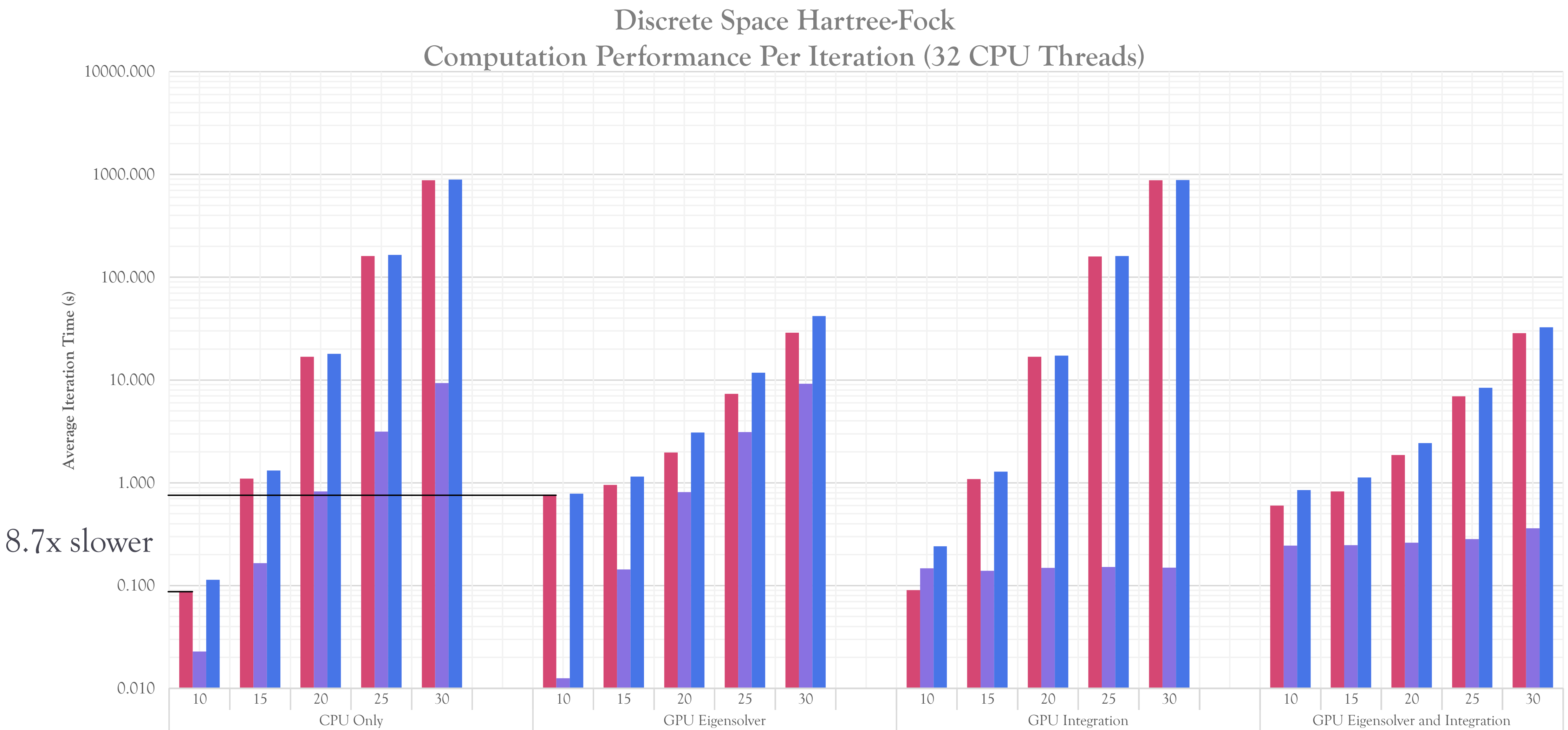
- Numerical Integration
  - Large solution spaces require numerous integrations performed along the diagonal
  - Leverage GPU’s massive parallelization capabilities
  - Ensure problem can be divided evenly into independent chunks (no memory dependencies)
- Diagonalization
  - Algorithm harder to perform parallel, but some speedups possible
  - Can leverage diagonalization routines provided by GPU vendor
  - CUDA cuSOLVER

# Discrete Space Hartree-Fock
# Computation Performance Per Iteration (32 CPU Threads)



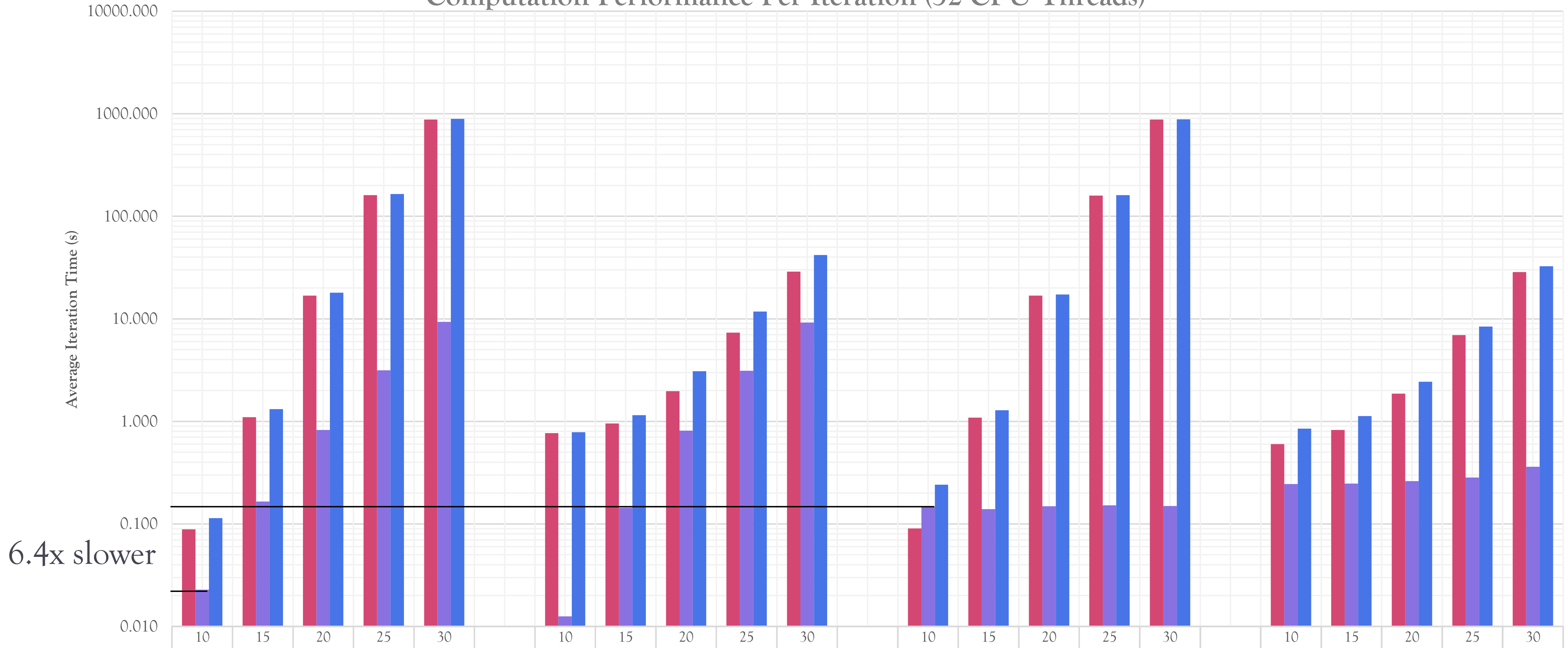

| | CPU Only | | | | | GPU Eigensolver | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 15 | 20 | 25 | 30 | 10 | 15 | 20 | 25 | 30 |
| Eigensolver Time (s) | 0.088 | 1.096 | 16.853 | 160.633 | 877.847 | 0.768 | 0.955 | 1.972 | 7.326 | 28.903 |
| Integration Time (s) | 0.023 | 0.165 | 0.823 | 3.149 | 9.362 | 0.013 | 0.143 | 0.810 | 3.123 | 9.194 |
| Total Time (s) | 0.114 | 1.317 | 17.977 | 164.988 | 891.473 | 0.783 | 1.151 | 3.088 | 11.789 | 41.990 |

| | GPU Integration | | | | | GPU Eigensolver and Integration | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 15 | 20 | 25 | 30 | 10 | 15 | 20 | 25 | 30 |
| Eigensolver Time (s) | 0.090 | 1.087 | 16.844 | 159.430 | 880.247 | 0.601 | 0.823 | 1.867 | 6.950 | 28.565 |
| Integration Time (s) | 0.147 | 0.140 | 0.149 | 0.151 | 0.149 | 0.244 | 0.248 | 0.262 | 0.284 | 0.361 |
| Total Time (s) | 0.241 | 1.282 | 17.295 | 160.707 | 883.721 | 0.850 | 1.124 | 2.440 | 8.422 | 32.540 |



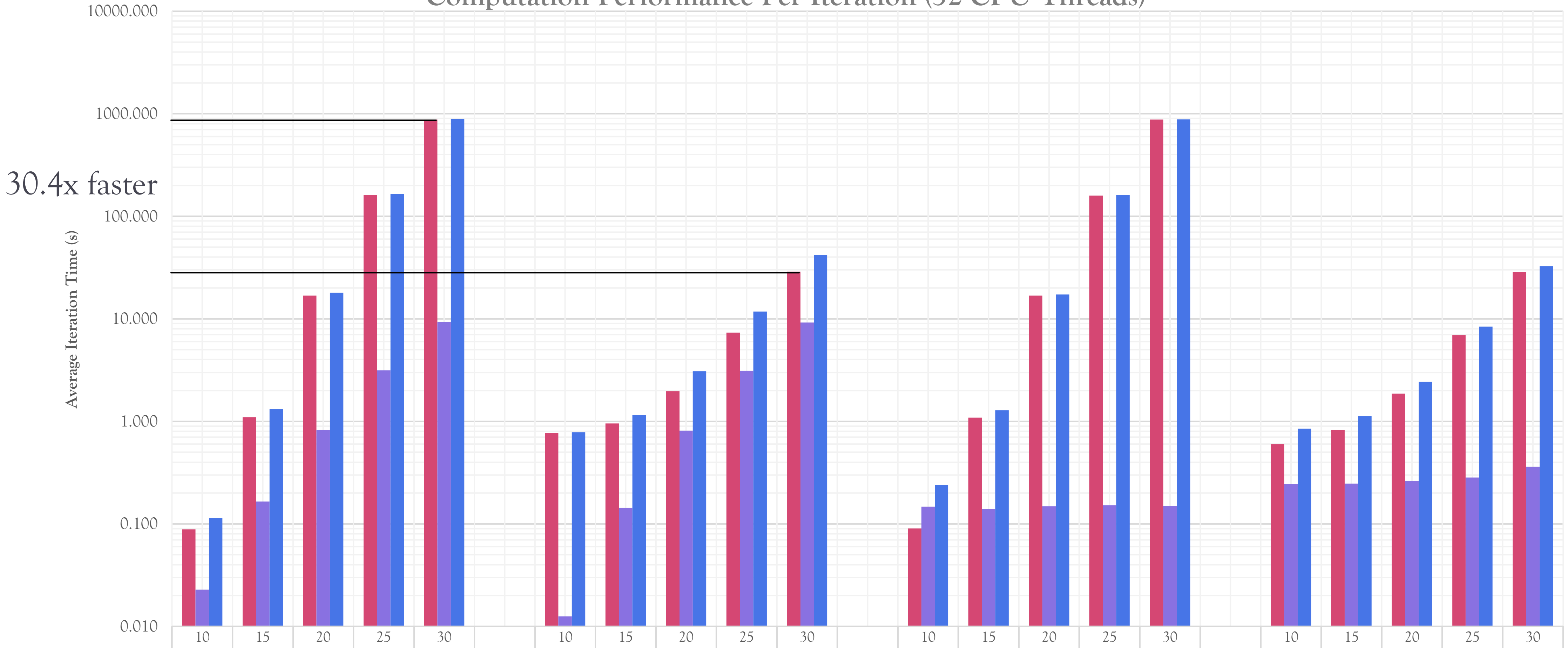

| | CPU Only 10 | CPU Only 15 | CPU Only 20 | CPU Only 25 | CPU Only 30 | GPU Eigensolver 10 | GPU Eigensolver 15 | GPU Eigensolver 20 | GPU Eigensolver 25 | GPU Eigensolver 30 | GPU Integration 10 | GPU Integration 15 | GPU Integration 20 | GPU Integration 25 | GPU Integration 30 | GPU Eigensolver and Integration 10 | GPU Eigensolver and Integration 15 | GPU Eigensolver and Integration 20 | GPU Eigensolver and Integration 25 | GPU Eigensolver and Integration 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.088 | 1.096 | 16.853 | 160.633 | 877.847 | 0.768 | 0.955 | 1.972 | 7.326 | 28.903 | 0.090 | 1.087 | 16.844 | 159.430 | 880.247 | 0.601 | 0.823 | 1.867 | 6.950 | 28.565 |
| Integration Time (s) | 0.023 | 0.165 | 0.823 | 3.149 | 9.362 | 0.013 | 0.143 | 0.810 | 3.123 | 9.194 | 0.147 | 0.140 | 0.149 | 0.151 | 0.149 | 0.244 | 0.248 | 0.262 | 0.284 | 0.361 |
| Total Time (s) | 0.114 | 1.317 | 17.977 | 164.988 | 891.473 | 0.783 | 1.151 | 3.088 | 11.789 | 41.990 | 0.241 | 1.282 | 17.295 | 160.707 | 883.721 | 0.850 | 1.124 | 2.440 | 8.422 | 32.540 |

## Discrete Space Hartree-Fock
## Computation Performance Per Iteration (32 CPU Threads)




| | CPU Only 10 | CPU Only 15 | CPU Only 20 | CPU Only 25 | CPU Only 30 | GPU Eigensolver 10 | GPU Eigensolver 15 | GPU Eigensolver 20 | GPU Eigensolver 25 | GPU Eigensolver 30 | GPU Integration 10 | GPU Integration 15 | GPU Integration 20 | GPU Integration 25 | GPU Integration 30 | GPU Eigensolver and Integration 10 | GPU Eigensolver and Integration 15 | GPU Eigensolver and Integration 20 | GPU Eigensolver and Integration 25 | GPU Eigensolver and Integration 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.088 | 1.096 | 16.853 | 160.633 | 877.847 | 0.768 | 0.955 | 1.972 | 7.326 | 28.903 | 0.090 | 1.087 | 16.844 | 159.430 | 880.247 | 0.601 | 0.823 | 1.867 | 6.950 | 28.565 |
| Integration Time (s) | 0.023 | 0.165 | 0.823 | 3.149 | 9.362 | 0.013 | 0.143 | 0.810 | 3.123 | 9.194 | 0.147 | 0.140 | 0.149 | 0.151 | 0.149 | 0.244 | 0.248 | 0.262 | 0.284 | 0.361 |
| Total Time (s) | 0.114 | 1.317 | 17.977 | 164.988 | 891.473 | 0.783 | 1.151 | 3.088 | 11.789 | 41.990 | 0.241 | 1.282 | 17.295 | 160.707 | 883.721 | 0.850 | 1.124 | 2.440 | 8.422 | 32.540 |

# Discrete Space Hartree-Fock
# Computation Performance Per Iteration (32 CPU Threads)



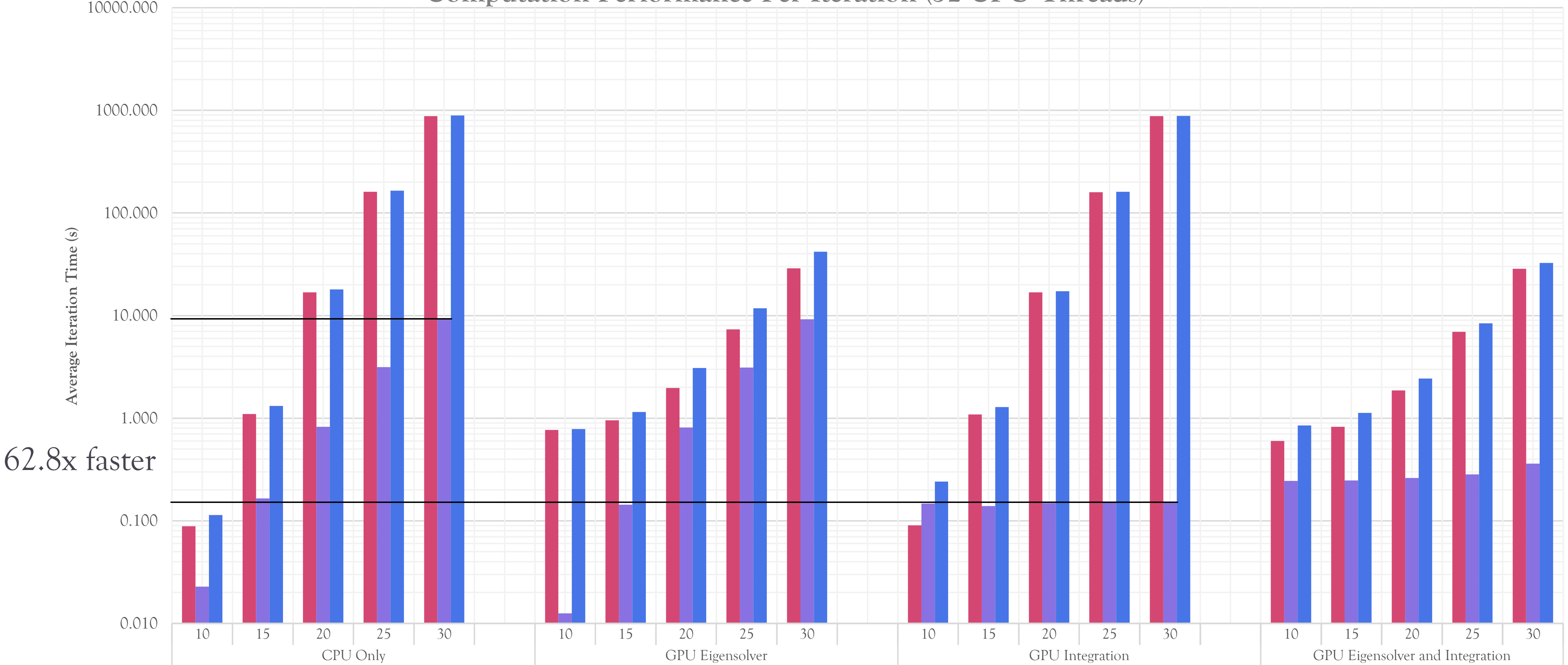

| CPU Only | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.088 | 1.096 | 16.853 | 160.633 | 877.847 |
| Integration Time (s) | 0.023 | 0.165 | 0.823 | 3.149 | 9.362 |
| Total Time (s) | 0.114 | 1.317 | 17.977 | 164.988 | 891.473 |

| GPU Eigensolver | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.768 | 0.955 | 1.972 | 7.326 | 28.903 |
| Integration Time (s) | 0.013 | 0.143 | 0.810 | 3.123 | 9.194 |
| Total Time (s) | 0.783 | 1.151 | 3.088 | 11.789 | 41.990 |

| GPU Integration | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.090 | 1.087 | 16.844 | 159.430 | 880.247 |
| Integration Time (s) | 0.147 | 0.140 | 0.149 | 0.151 | 0.149 |
| Total Time (s) | 0.241 | 1.282 | 17.295 | 160.707 | 883.721 |

| GPU Eigensolver and Integration | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|
| Eigensolver Time (s) | 0.601 | 0.823 | 1.867 | 6.950 | 28.565 |
| Integration Time (s) | 0.244 | 0.248 | 0.262 | 0.284 | 0.361 |
| Total Time (s) | 0.850 | 1.124 | 2.440 | 8.422 | 32.540 |

## Discrete Space Hartree-Fock
## Computation Performance Per Iteration (32 CPU Threads)




| | CPU Only | | | | | GPU Eigensolver | | | | | GPU Integration | | | | | GPU Eigensolver and Integration | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 15 | 20 | 25 | 30 | 10 | 15 | 20 | 25 | 30 | 10 | 15 | 20 | 25 | 30 | 10 | 15 | 20 | 25 | 30 |
| Eigensolver Time (s) | 0.088 | 1.096 | 16.853 | 160.633 | 877.847 | 0.768 | 0.955 | 1.972 | 7.326 | 28.903 | 0.090 | 1.087 | 16.844 | 159.430 | 880.247 | 0.601 | 0.823 | 1.867 | 6.950 | 28.565 |
| Integration Time (s) | 0.023 | 0.165 | 0.823 | 3.149 | 9.362 | 0.013 | 0.143 | 0.810 | 3.123 | 9.194 | 0.147 | 0.140 | 0.149 | 0.151 | 0.149 | 0.244 | 0.248 | 0.262 | 0.284 | 0.361 |
| Total Time (s) | 0.114 | 1.317 | 17.977 | 164.988 | 891.473 | 0.783 | 1.151 | 3.088 | 11.789 | 41.990 | 0.241 | 1.282 | 17.295 | 160.707 | 883.721 | 0.850 | 1.124 | 2.440 | 8.422 | 32.540 |

# Future Considerations

- Current implementation naïve
  - Off-the-shelf eigensolver used, could break the algorithm down further and hand optimize (ROI?)
  - Numerical integration only parallelized per diagonal entry, integration itself could be parallelized as well
- Apply CPU acceleration techniques to basis-function HF and DFT
  - Other opportunities and challenges exist with both
- GPU Acceleration not trivial – requires decent understanding of architecture and algorithm for best results.

# Questions?

# Thank You!

Have a great summer ☺