Team 11

Khasan Akhmadiev
Aikun Bexultanova

# High-Performance Thermodynamic Property Computation using Parallelization and Optimization Techniques

Skoltech

## Introduction

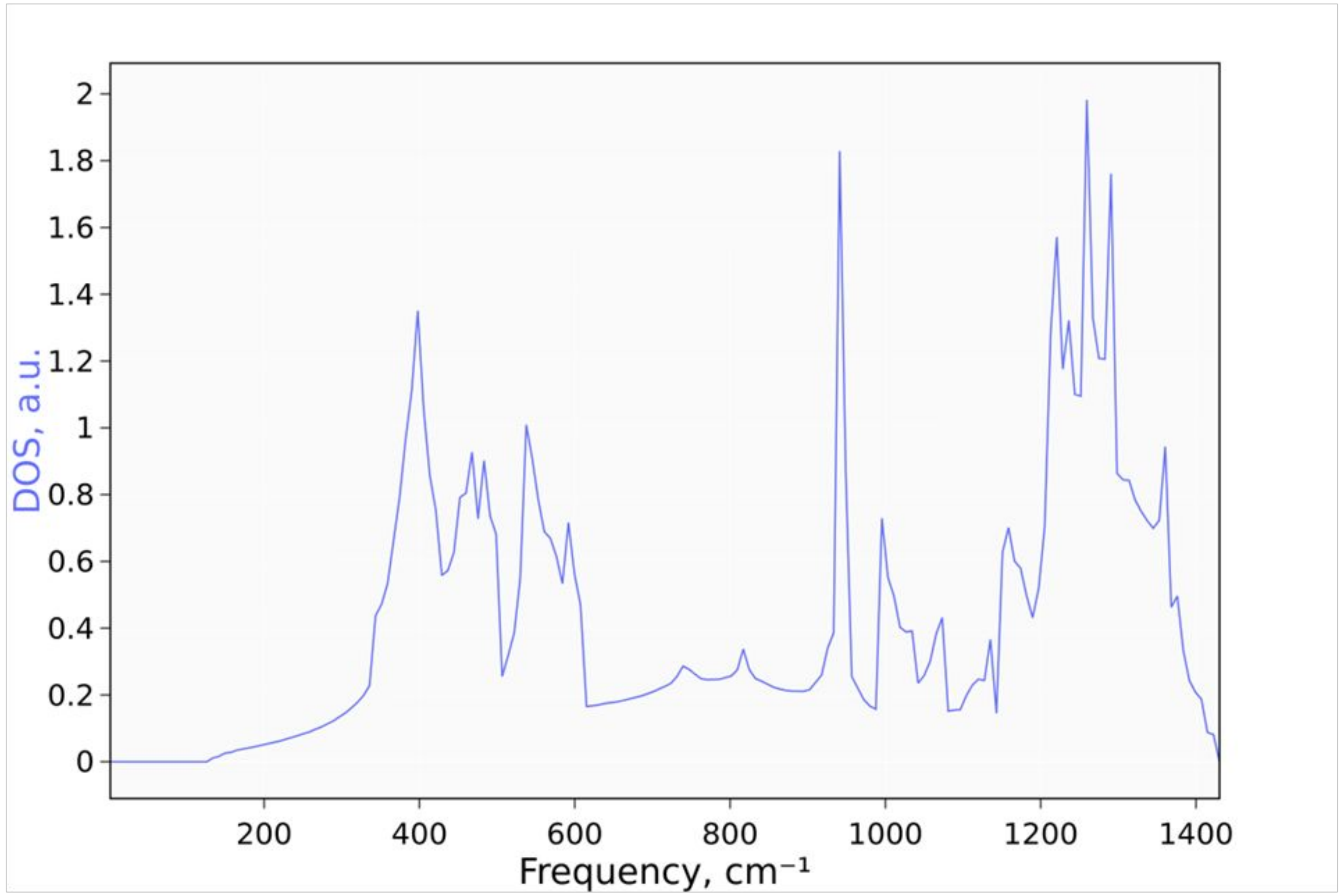
**Purpose of the Project:**

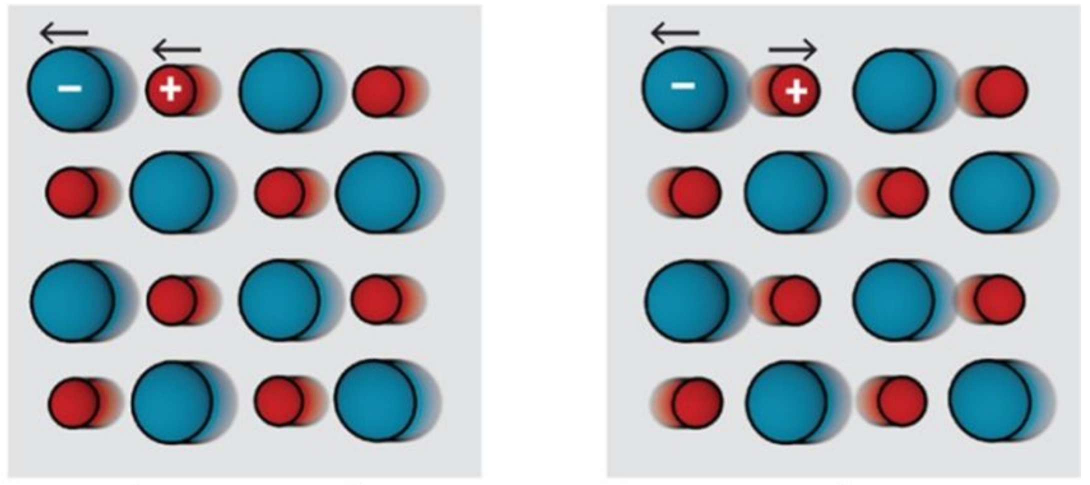
- To explore and implement **high-performance Python methods** for computational tasks
- Chosen Task: **Solving Thermodynamic Equations** efficiently

**Why High Performance?**

- Standard Python can be slow for computationally heavy tasks
- Utilizing optimized methods can save **time** and **resources**

## Why we need phonon dispersion





## Thermodynamical properties

### Free Helmholtz Energy

$$F(T) = k_B T \int \ln\left(1 - e^{-\frac{\hbar\omega}{k_B T}}\right) g(\omega) d\omega$$

### Heat capacity

$$C_v(T) = k_B \int \left(\frac{\hbar\omega}{k_B T}\right)^2 \frac{e^{\frac{\hbar\omega}{k_B T}}}{\left(e^{\frac{\hbar\omega}{k_B T}} - 1\right)^2} g(\omega) d\omega$$

### Enthropy

$$S(T) = -k_B \int \left[\frac{e^{\frac{\hbar\omega}{k_B T}}}{\left(e^{\frac{\hbar\omega}{k_B T}} - 1\right)} - \ln\left(1 - e^{-\frac{\hbar\omega}{k_B T}}\right)\right] g(\omega) d\omega$$

### Gibbs Free Energy

$$G(T) = F(T) + \int_0^T S(T')\, dT'$$

Skoltech

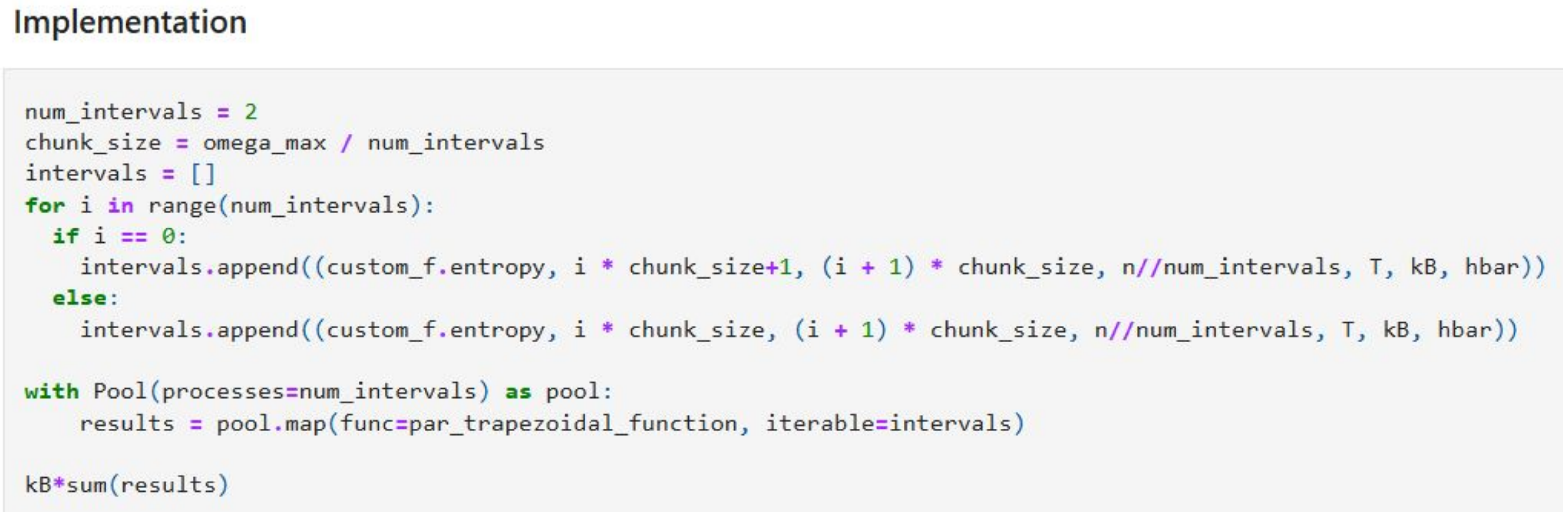
## Tools and Libraries Used

- **NumPy:** Optimized numerical computations using vectorized operations.
- **Numba:** JIT (Just-In-Time) compiler for Python functions.
- **CuPy:** library for GPU-accelerated computing with Python.
- **Multiprocessing:** Parallel execution of tasks.
- **MPI:** portable message-passing standard designed to function on parallel computing architectures.

**Benchmarking:** compare execution time for each task

# Enthropy. Multiprocessing

$$S(T) = k_B \int_0^\infty \left( \frac{\hbar\omega}{k_B T} \coth\left(\frac{\hbar\omega}{2k_B T}\right) - \ln\left(2\sinh\left(\frac{\hbar\omega}{2k_B T}\right)\right)\right) g(\omega) d\omega$$



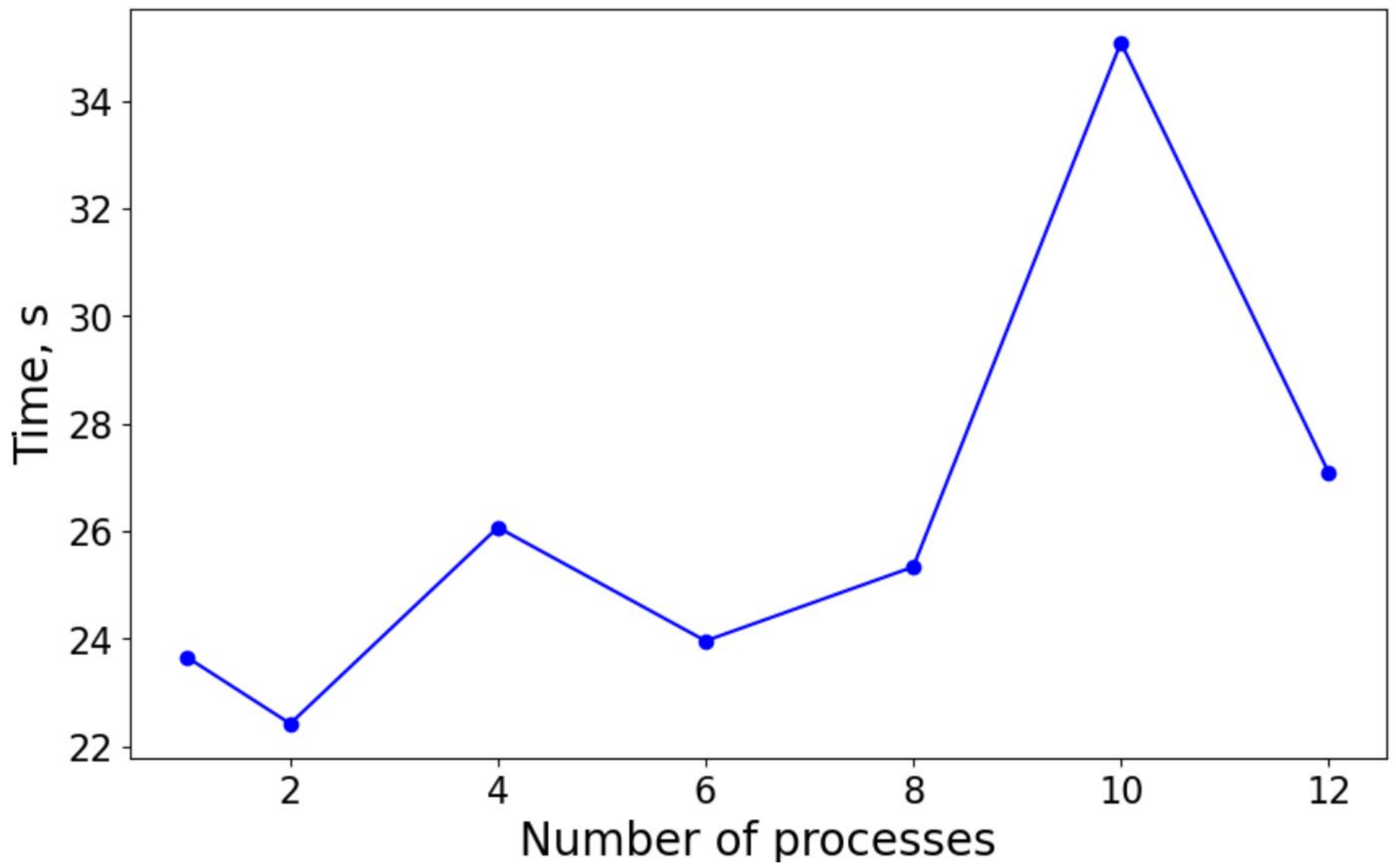
## Implementation

```
num_intervals = 2
chunk_size = omega_max / num_intervals
intervals = []
for i in range(num_intervals):
  if i == 0:
    intervals.append((custom_f.entropy, i * chunk_size+1, (i + 1) * chunk_size, n//num_intervals, T, kB, hbar))
  else:
    intervals.append((custom_f.entropy, i * chunk_size, (i + 1) * chunk_size, n//num_intervals, T, kB, hbar))

with Pool(processes=num_intervals) as pool:
    results = pool.map(func=par_trapezoidal_function, iterable=intervals)

kB*sum(results)
```

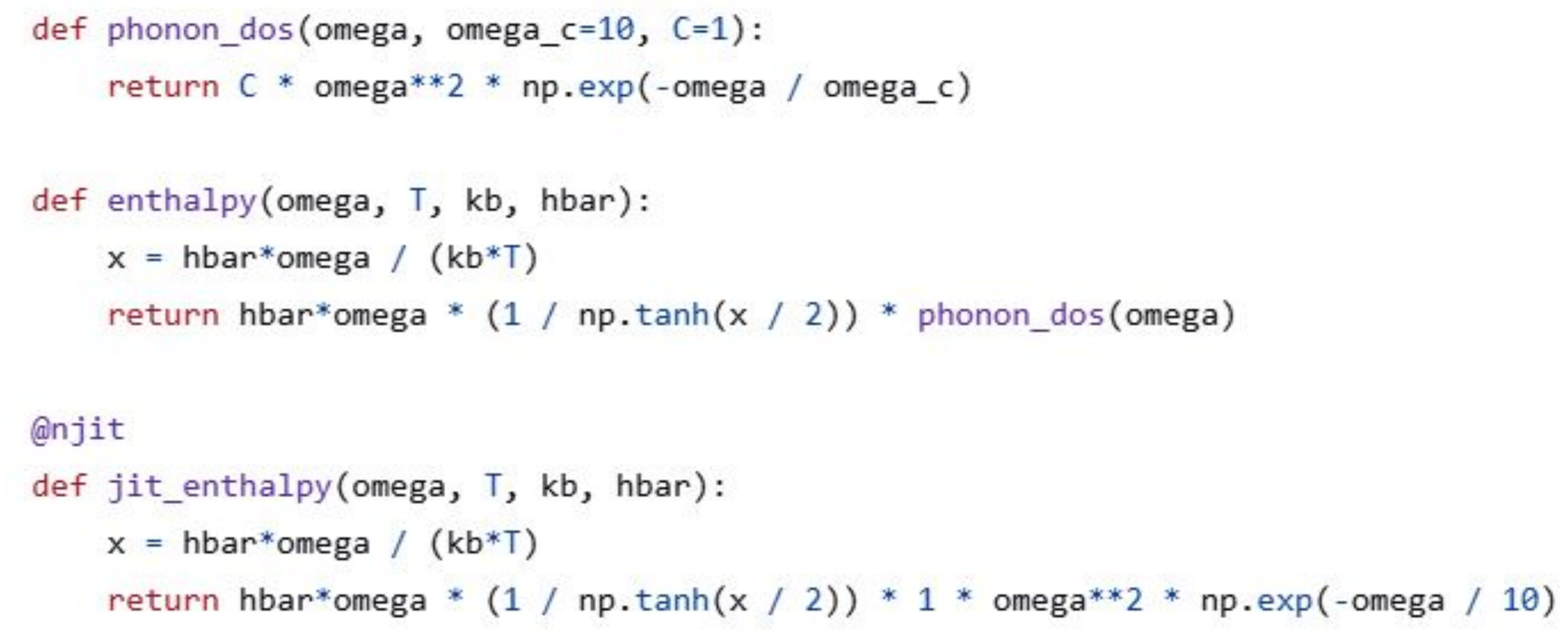
# Entropy evaluation



## Enthalpy. Numba

$$H(T) = \int_0^\infty \hbar\omega \coth\left(\frac{\hbar\omega}{2k_BT}\right) g(\omega) d\omega$$

Regular & numba implementation



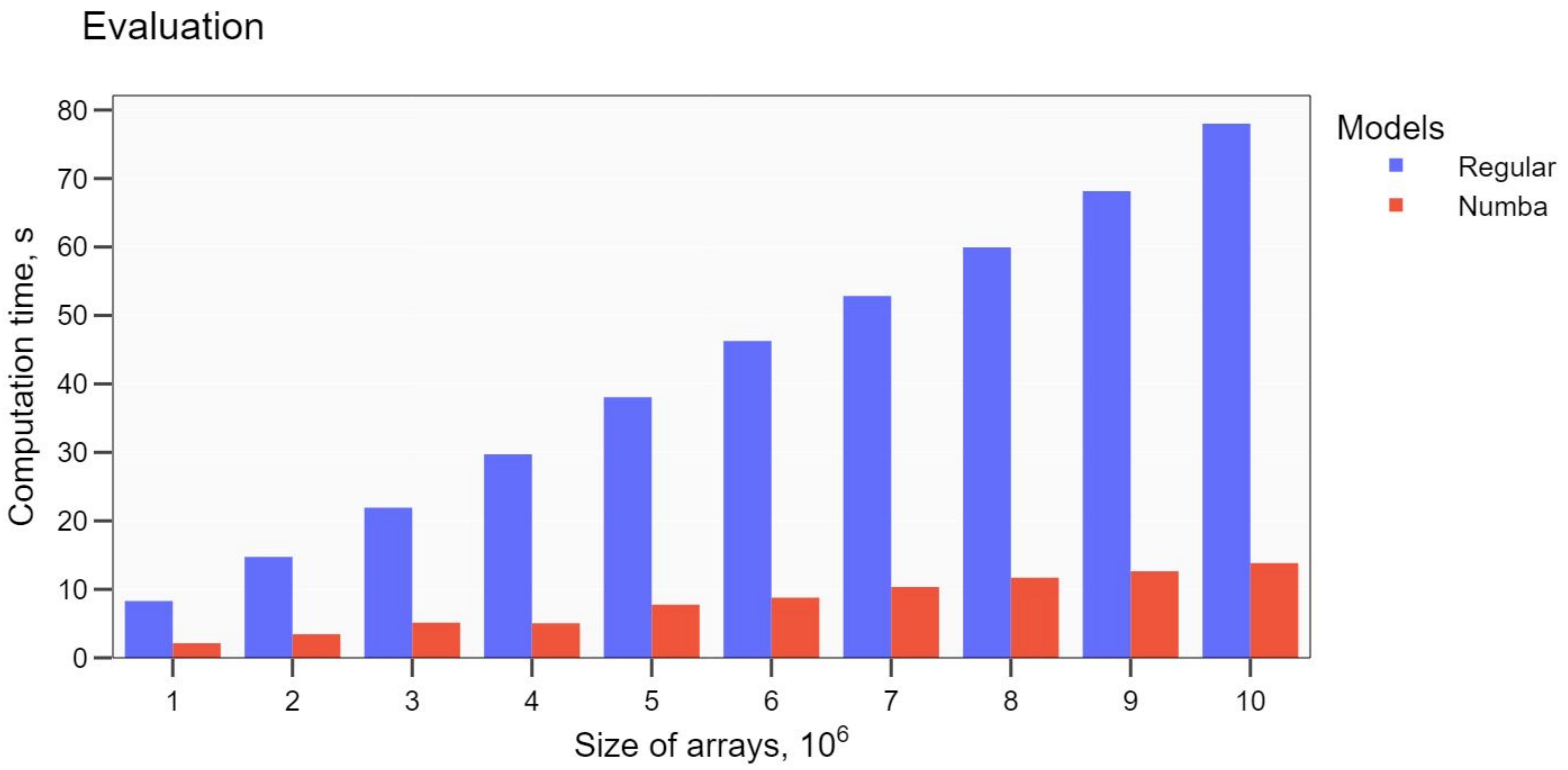
```
def phonon_dos(omega, omega_c=10, C=1):
    return C * omega**2 * np.exp(-omega / omega_c)

def enthalpy(omega, T, kb, hbar):
    x = hbar*omega / (kb*T)
    return hbar*omega * (1 / np.tanh(x / 2)) * phonon_dos(omega)

@njit
def jit_enthalpy(omega, T, kb, hbar):
    x = hbar*omega / (kb*T)
    return hbar*omega * (1 / np.tanh(x / 2)) * 1 * omega**2 * np.exp(-omega / 10)
```

Skoltech

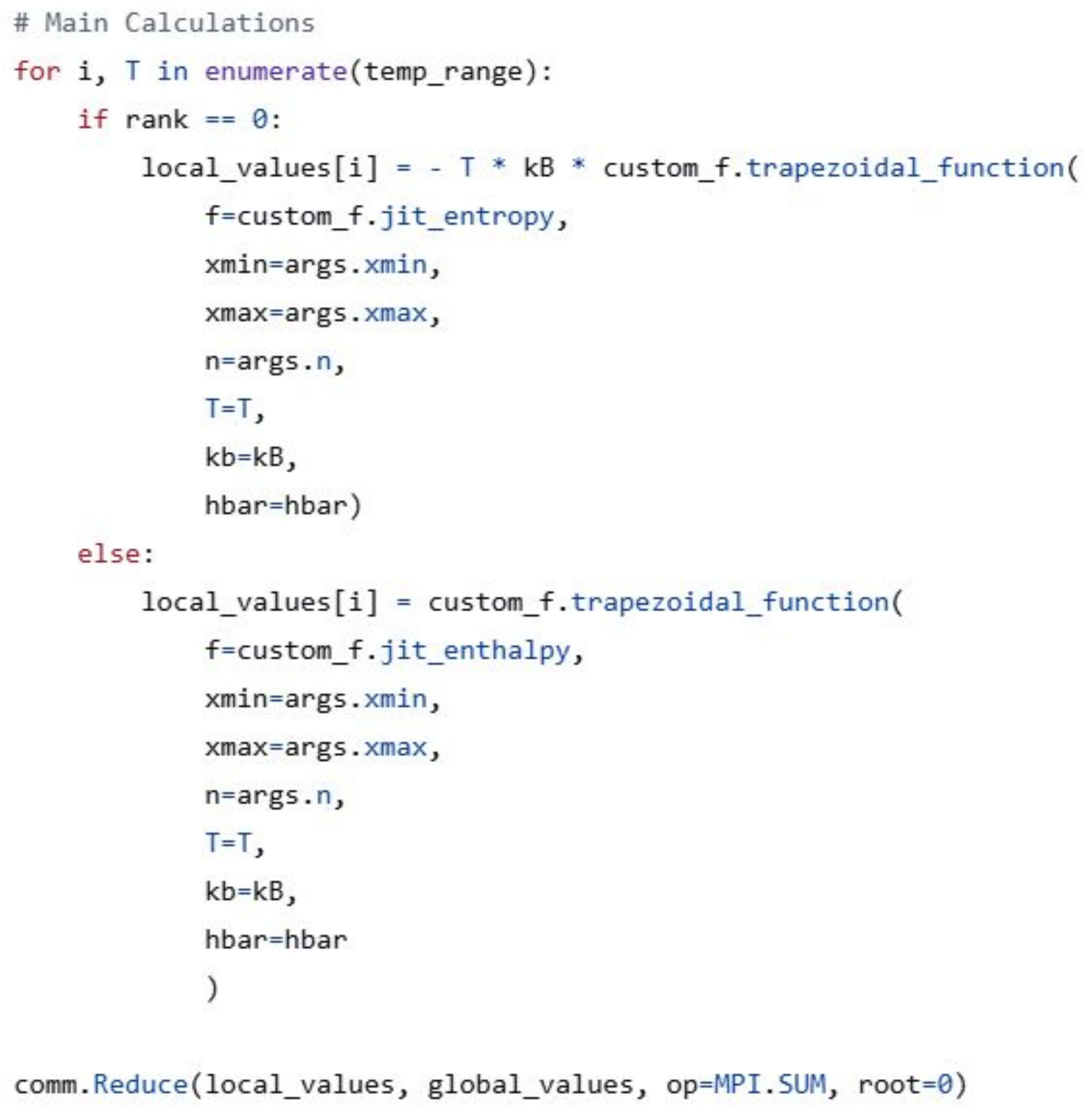
# Enthalpy evaluation



Skoltech

## Gibbs energy. MPI

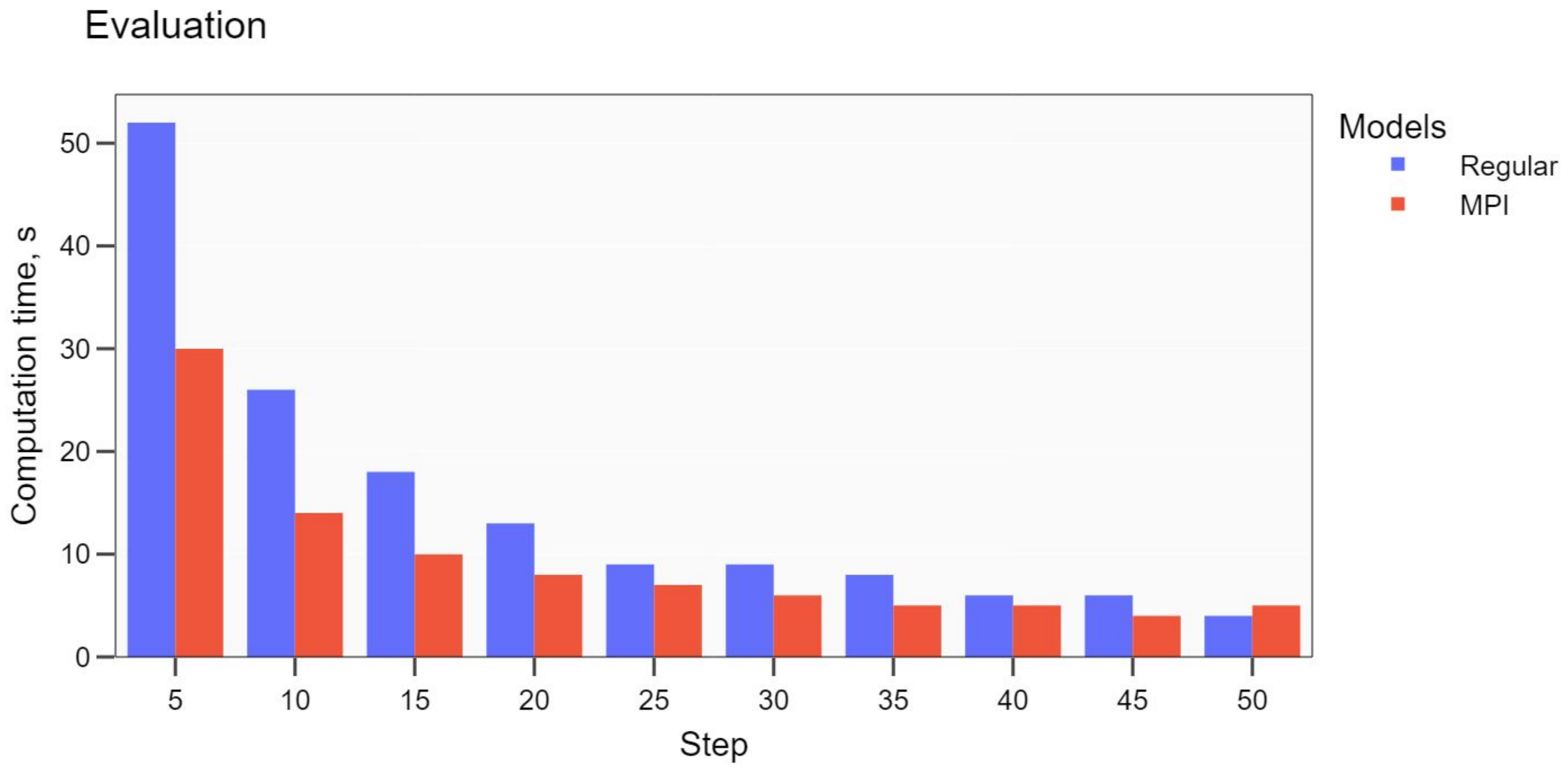
$$G(T) = H(T) - T \cdot S(T)$$



```
# Main Calculations
for i, T in enumerate(temp_range):
    if rank == 0:
        local_values[i] = - T * kB * custom_f.trapezoidal_function(
            f=custom_f.jit_entropy,
            xmin=args.xmin,
            xmax=args.xmax,
            n=args.n,
            T=T,
            kb=kB,
            hbar=hbar)
    else:
        local_values[i] = custom_f.trapezoidal_function(
            f=custom_f.jit_enthalpy,
            xmin=args.xmin,
            xmax=args.xmax,
            n=args.n,
            T=T,
            kb=kB,
            hbar=hbar
            )

comm.Reduce(local_values, global_values, op=MPI.SUM, root=0)
```

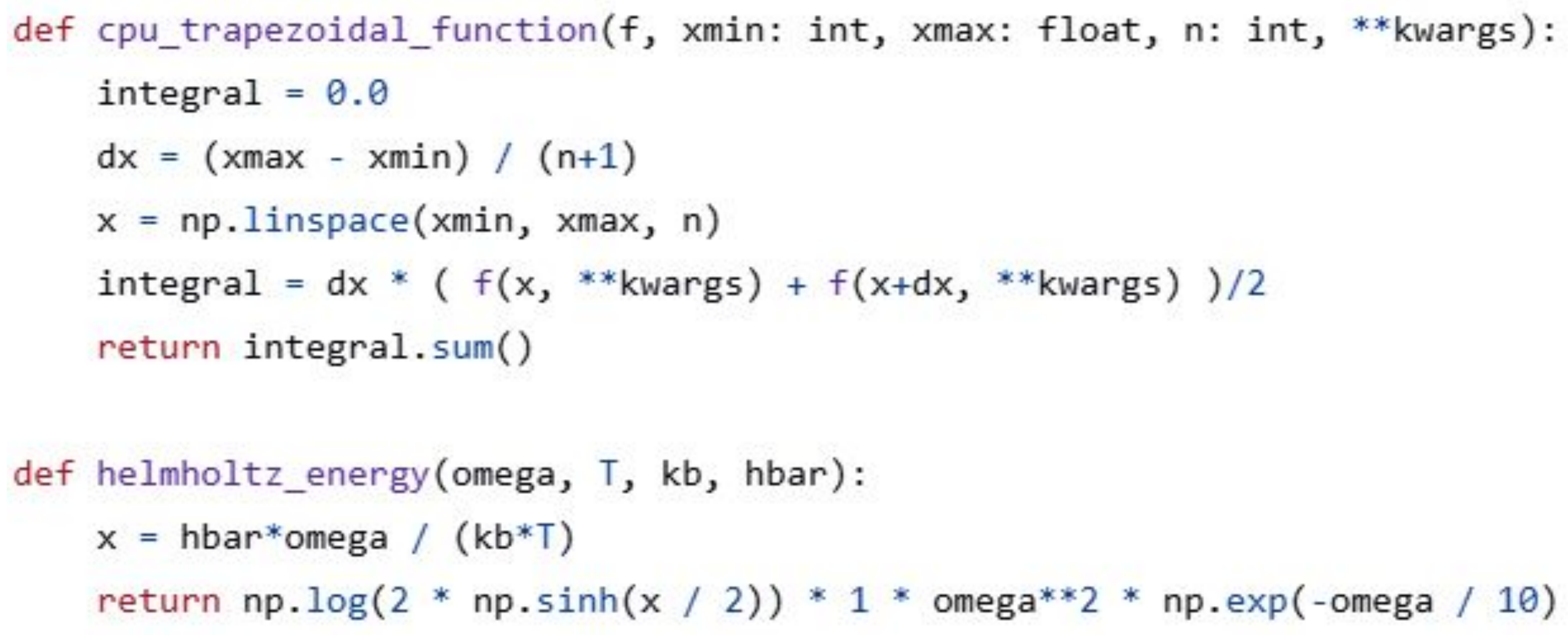
## Gibbs energy evaluation



Skoltech

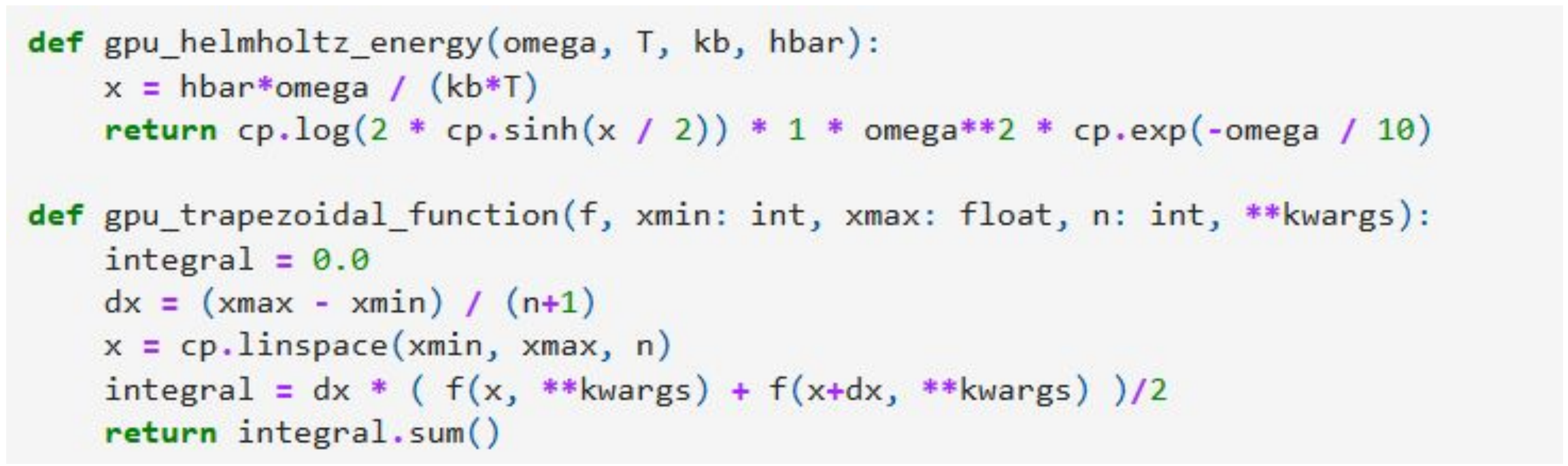
## Helmholtz Free energy. GPU

$$F(T) = k_B T \int_0^{\infty} \ln\left(2\sinh\left(\frac{\hbar\omega}{2k_B T}\right)\right) g(\omega)d\omega$$



```
def cpu_trapezoidal_function(f, xmin: int, xmax: float, n: int, **kwargs):
    integral = 0.0
    dx = (xmax - xmin) / (n+1)
    x = np.linspace(xmin, xmax, n)
    integral = dx * ( f(x, **kwargs) + f(x+dx, **kwargs) )/2
    return integral.sum()


def helmholtz_energy(omega, T, kb, hbar):
    x = hbar*omega / (kb*T)
    return np.log(2 * np.sinh(x / 2)) * 1 * omega**2 * np.exp(-omega / 10)
```

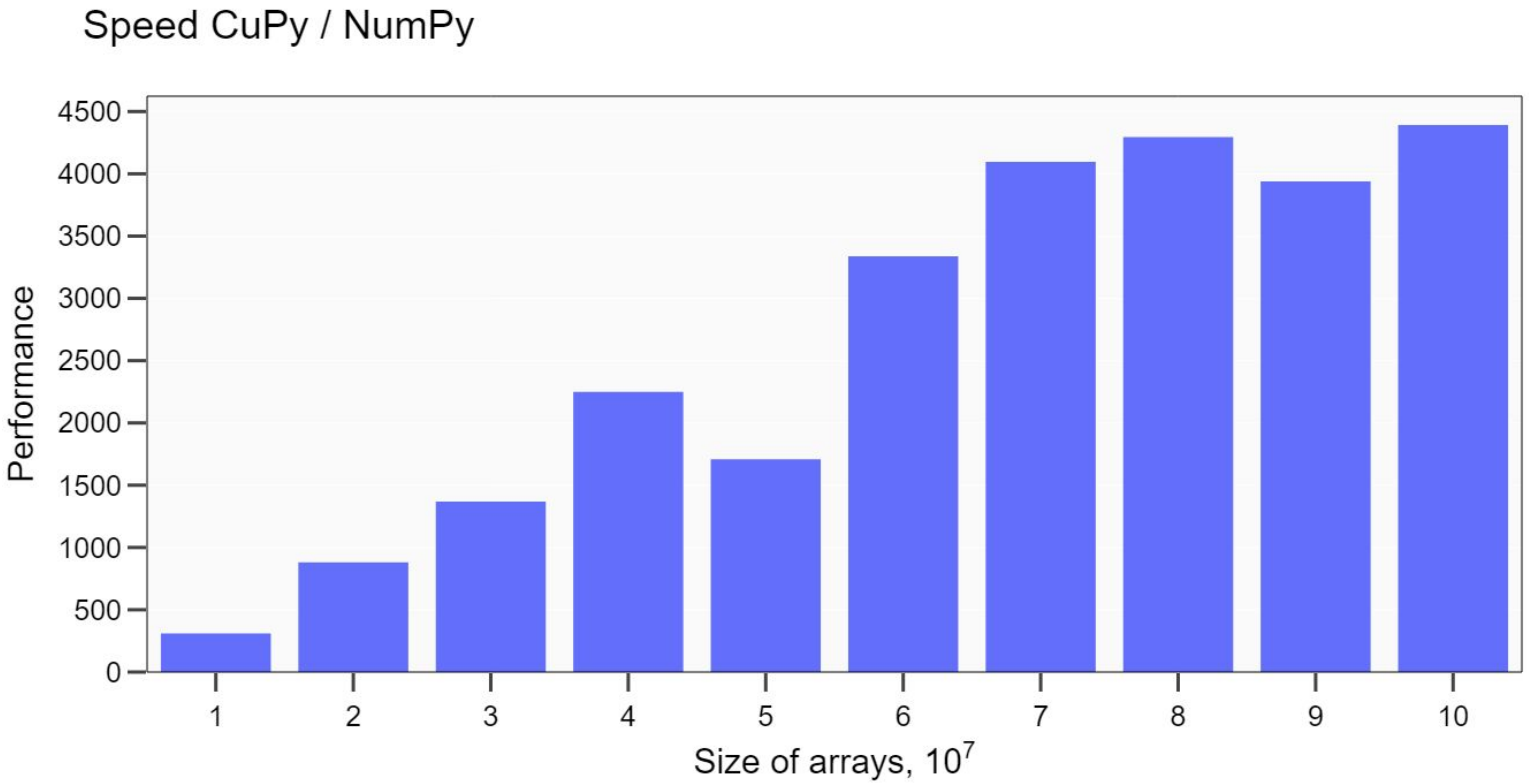


```
def gpu_helmholtz_energy(omega, T, kb, hbar):
    x = hbar*omega / (kb*T)
    return cp.log(2 * cp.sinh(x / 2)) * 1 * omega**2 * cp.exp(-omega / 10)

def gpu_trapezoidal_function(f, xmin: int, xmax: float, n: int, **kwargs):
    integral = 0.0
    dx = (xmax - xmin) / (n+1)
    x = cp.linspace(xmin, xmax, n)
    integral = dx * ( f(x, **kwargs) + f(x+dx, **kwargs) )/2
    return integral.sum()
```

Skoltech

## Evaluation



Skoltech

# Thx

Skoltech