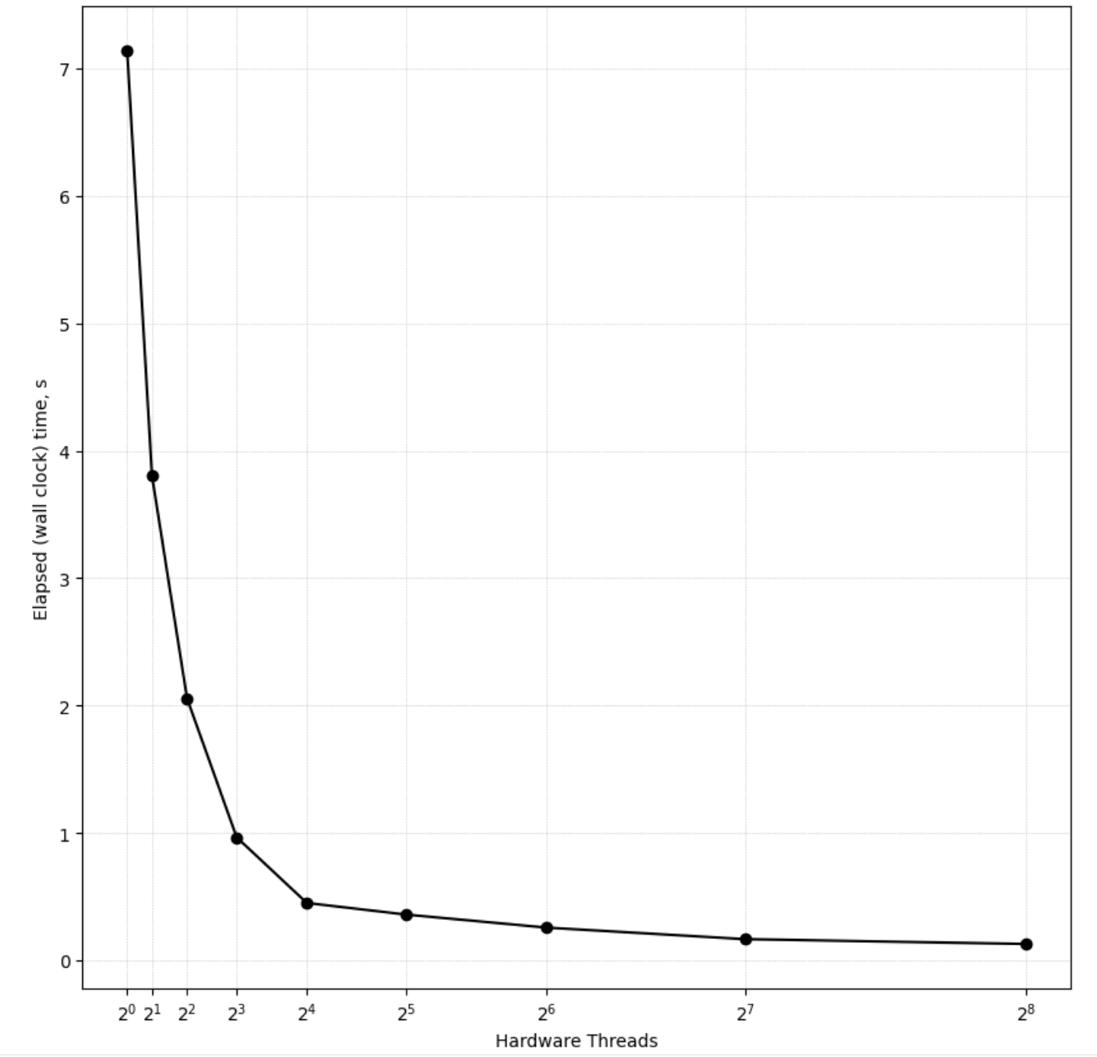
1. **Runtime Plot**



1. **Code Listing**

***Code written in C++***

#include <bits/stdc++.h>

#include <omp.h>

using namespace std;

#define ORDER 1000

#define AVAL 5.0

#define BVAL 7.0

int main(int argc, char\* argv[]){

int Pdim, Ndim, Mdim;

int i,j,k;

double tmpVal;

Ndim = Mdim = Pdim = ORDER;

vector<double> A(Ndim\*Pdim, AVAL);

vector<double> B(Pdim\*Mdim, BVAL);

vector<double> C(Mdim\*Ndim, 0);

int thread\_nums = atoi(argv[1]);

omp\_set\_num\_threads(thread\_nums);

double start = omp\_get\_wtime();

#pragma omp parallel for private(tmpVal, i, j, k)

for(int i = 0; i<Ndim; i++){

for(int j = 0; j<Mdim; j++){

tmpVal = 0.0;

for (k = 0; k < Pdim; k++) {

tmpVal += A[i \* Ndim + k] \* B[k \* Pdim + j];

}

C[i \* Ndim + j] = tmpVal;

}

}

double end = omp\_get\_wtime();

double run\_time = end - start;

printf("%.6f,", run\_time);

return 0;

}

1. **Program Output**
2. ***Makefile output –***

Running matrix\_mult with 1 threads...

Running matrix\_mult with 2 threads...

Running matrix\_mult with 4 threads...

Running matrix\_mult with 8 threads...

Running matrix\_mult with 16 threads...

Running matrix\_mult with 32 threads...

Running matrix\_mult with 64 threads...

Running matrix\_mult with 128 threads...

Running matrix\_mult with 256 threads...

1. ***Results File Output –***

# Threads Elapsed\_Time (s)

7.139942,3.803579,2.055581,0.965708,0.450673,0.359758,0.257488,0.166898,0.129000>

1. **MATLAB plot script**

import numpy as np

import matplotlib.pyplot as plt

hardware\_threads = np.array([2\*\*i for i in range(0, 9)]) # 2^1 to 2^7

elapsed\_time = [7.139942,3.803579,2.055581,0.965708,0.450673,0.359758,0.257488,0.166898,0.129000]

transformed\_x = np.sqrt(hardware\_threads)

plt.figure(figsize=(10, 10))

plt.plot(transformed\_x, elapsed\_time, marker='o', linestyle='-', color='black')

plt.xticks(transformed\_x, [f"$2^{i}$" for i in range(0, 9)])

plt.xlabel("Hardware Threads")

plt.ylabel("Elapsed (wall clock) time, s")

plt.grid(True, which="both", linestyle=":", linewidth=0.5)

1. **Results Analysis**

The time taken drastically reduces as number of threads increases from 1->2->4->8->16, reducing the time taken by almost an order of 2 for each iteration. However, as we go to higher number of hardware threads from 64->128->256 the decrease in time taken becomes more marginal as evidenced by the runtime plot plateauing out around 2^5 threads.