Code -

```
#include <iostream>
#include <vector>
#include <cuda_runtime.h>
#include <cusolverDn.h>
#include <cassert>
#include <random>
#include <cmath>

using namespace std;

void convertToColumnMajor(const vector<vector<double>>& A, vector<double>& columnMajor) {
   int N = A.size();
   columnMajor.resize(N * N);
   for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j)
         columnMajor[j * N + i] = A[i][j];
}

vector<double> makeHilbertMatrix(int N) {
   vector<vector<double>> A(N, vector<double>(N, 0.0));
   for (int i = 0; i < N; ++i)
      for (int j = 0; j < N; ++j)
         A[i][j] = 1.0 / (i + j + 1);
   vector<double> columnMajor;
   convertToColumnMajor(A, columnMajor);
   return columnMajor;
}

vector<double> getResultVector(int N) {
   return vector<double>(N, 1.0);
}

vector<double> perturbVector(const vector<double>& vec) {
   vector<double> perturbed(vec);
   random_device rd;
   mt19937 gen(rd());
   uniform_real_distribution<> dis(0.01, 1.0);
   for (double& v : perturbed) {
      v += dis(gen);
   }
   return perturbed;
}
```

```
double computeNorm(const vector<double>& vec) {
   double norm = 0.0;
   for (double val : vec) norm += val * val;
   return sqrt(norm);
}

int main(int argc, char* argv[]) {
   int N = stoi(argv[1]);
   const int lda = N;
   const int ldb = N;
   const bool pivot_on = false;

   vector<double> A = makeHilbertMatrix(N);
   vector<double> B1 = getResultVector(N);
   vector<double> B2 = perturbVector(B1);
   vector<double> X1(N), X2(N);

   cusolverDnHandle_t cusolverH;
   cudaStream_t stream;
   cudaEvent_t start, stop;
   float elapsed_ms;

   double *d_A = nullptr, *d_B = nullptr, *d_work = nullptr;
   int *d_Ipiv = nullptr, *d_info = nullptr;
   int lwork = 0;

   cusolverDnCreate(&cusolverH);
   cudaStreamCreateWithFlags(&stream, cudaStreamNonBlocking);
   cusolverDnSetStream(cusolverH, stream);
   cudaEventCreate(&start);
   cudaEventCreate(&stop);

   cudaMalloc(&d_A, sizeof(double) * N * N);
   cudaMalloc(&d_B, sizeof(double) * N);
   cudaMalloc(&d_info, sizeof(int));
   if (pivot_on) cudaMalloc(&d_Ipiv, sizeof(int) * N);

   cudaMemcpy(d_A, A.data(), sizeof(double) * N * N, cudaMemcpyHostToDevice);
   cudaMemcpy(d_B, B1.data(), sizeof(double) * N, cudaMemcpyHostToDevice);

   cusolverDnDgetrf_bufferSize(cusolverH, N, N, d_A, lda, &lwork);
   cudaMalloc(&d_work, sizeof(double) * lwork);


   cudaEventRecord(start);
   cusolverDnDgetrf(cusolverH, N, N, d_A, lda, d_work, d_Ipiv, d_info);
   cusolverDnDgetrs(cusolverH, CUBLAS_OP_N, N, 1, d_A, lda, d_Ipiv, d_B, ldb, d_info);
   cudaEventRecord(stop);
```

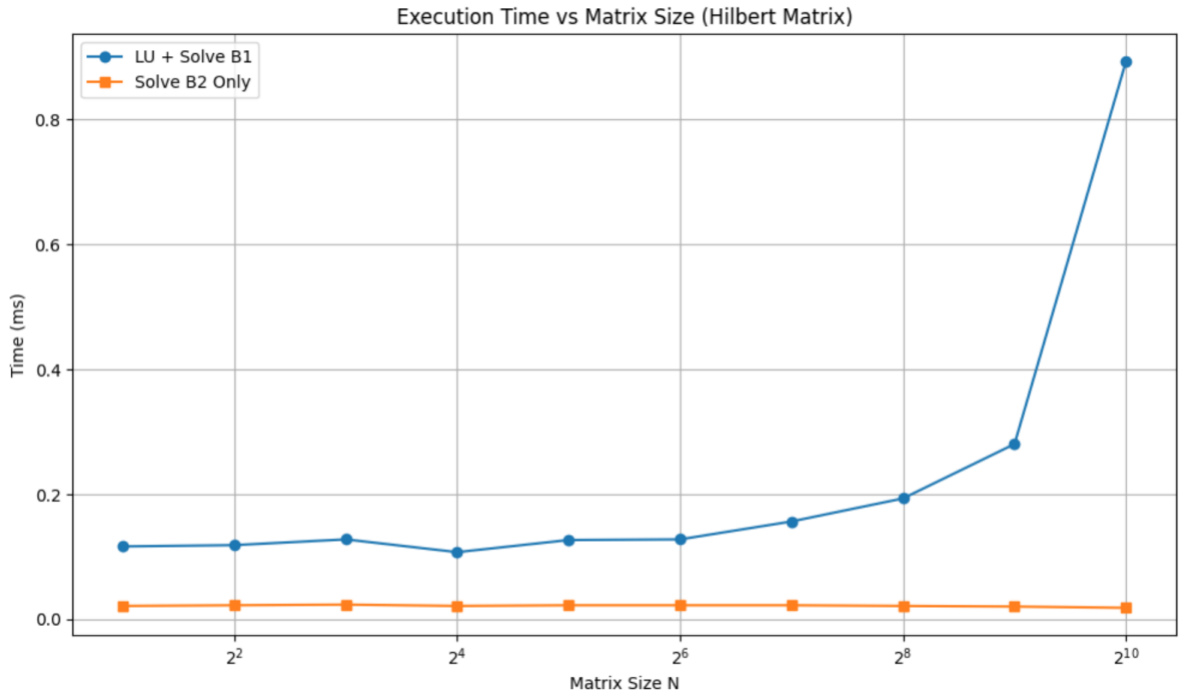
```
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsed_ms, start, stop);
    cout << "Time for LU + solve B1: " << elapsed_ms << " ms" << endl;

    cudaMemcpy(X1.data(), d_B, sizeof(double) * N, cudaMemcpyDeviceToHost);


    cudaMemcpy(d_B, B2.data(), sizeof(double) * N, cudaMemcpyHostToDevice);
    cudaEventRecord(start);
    cusolverDnDgetrs(cusolverH, CUBLAS_OP_N, N, 1, d_A, lda, d_Ipiv, d_B, ldb, d_info);
    cudaEventRecord(stop);
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&elapsed_ms, start, stop);
    cout << "Time for solve B2 only: " << elapsed_ms << " ms" << endl;

    cudaMemcpy(X2.data(), d_B, sizeof(double) * N, cudaMemcpyDeviceToHost);

    // check correctness results look skewed
    cout << "Norm of X1: " << computeNorm(X1) << endl;
    cout << "Norm of X2: " << computeNorm(X2) << endl;


    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_info);
    cudaFree(d_work);
    if (pivot_on) cudaFree(d_Ipiv);

    cusolverDnDestroy(cusolverH);
    cudaStreamDestroy(stream);
    cudaEventDestroy(start);
    cudaEventDestroy(stop);
    cudaDeviceReset();

    return 0;
}
```

Log -

```
N - 2
Time for LU + solve B1: 0.116736 ms
Time for solve B2 only: 0.021504 ms
Norm of X1: 6.32456
Norm of X2: 12.5279
```

N - 4
Time for LU + solve B1: 0.118784 ms
Time for solve B2 only: 0.022528 ms
Norm of X1: 235.83
Norm of X2: 5947.62
N - 8
Time for LU + solve B1: 0.128 ms
Time for solve B2 only: 0.023552 ms
Norm of X1: 314773
Norm of X2: 6.91749e+07
N - 16
Time for LU + solve B1: 0.10752 ms
Time for solve B2 only: 0.021504 ms
Norm of X1: 4.72964e+09
Norm of X2: 3.59851e+17
N - 32
Time for LU + solve B1: 0.126976 ms
Time for solve B2 only: 0.022528 ms
Norm of X1: 1.8454e+10
Norm of X2: 7.99514e+17
N - 64
Time for LU + solve B1: 0.128 ms
Time for solve B2 only: 0.022528 ms
Norm of X1: 3.59436e+10
Norm of X2: 2.04166e+18
N - 128
Time for LU + solve B1: 0.156672 ms
Time for solve B2 only: 0.022528 ms
Norm of X1: 9.4919e+09
Norm of X2: 2.54054e+18
N - 256
Time for LU + solve B1: 0.193536 ms
Time for solve B2 only: 0.021504 ms
Norm of X1: 6.56699e+10
Norm of X2: 1.49299e+19
N - 512
Time for LU + solve B1: 0.280576 ms
Time for solve B2 only: 0.02048 ms
Norm of X1: 3.13948e+11
Norm of X2: 4.46335e+19
N - 1024
Time for LU + solve B1: 0.892928 ms
Time for solve B2 only: 0.018432 ms
Norm of X1: 2.58884e+11
Norm of X2: 2.4986e+19

Timing Graph -



Discussion –

The results were very interesting for this assignment. While as expected, the first system with **LU** factorization (*cusolverDnDgetrf*) times increase progressively with the increase in matrix the timing for the second system remains almost the same as evidenced by the graph.

It made sense to me that since we already had the LU factors computed in the first step, solving the second system would be faster—but I didn't expect it to be practically instantaneous. That raised a flag: was the second solution X2 even accurate? To check this, I computed the norms of both X1 and X2 to get a sense of how sensitive the system is to changes in the right-hand side. As expected, for values of N=8,16,32,64,128,256, the norm of X2 exploded, suggesting that the Hilbert matrix is extremely ill-conditioned. What surprised me, though, was that for N=512 and 1024, the norm remained huge but relatively more stable again. This initially seemed inconsistent, but after looking into it further, I realized it aligns with known behavior—Hilbert matrices are notoriously unstable, and even tiny perturbations to the b vector can cause massive swings in the resulting x vector. These results confirm that instability firsthand.

Overall, this was an assignment which taught me to dive deep into the results.