Project Objectives: Parallelizing two APSP (All Pairs shortest path) Algorithm with GPU

Selected Algorithms: Floyd-Warshall, Dijkstra for all nodes ( Johson’s Algorithm)

Note: All of the code written in this project is parallelized by CUDA and written in C.

**Floyd-Warshall**

**Functions**

void floyd\_warshall\_serial (int\*\* *graph*, int\*\* *dist*, int *N*)

This function is serial implementation of floyd warshall algorithm which gets graph adjacency matrix , distance matrix and number of nodes as input.

void makeAdjacency(int *n*)

Gets and integer as an input and makes a random matrix in A\_Matrix.

\_\_global\_\_ void floyd\_warshall\_parallel\_kernel(int\* *dev\_dist*, int *N*, int *k*)

CUDA kernel which is parallel implementation of the main loop of serial function.

    int gridx = pow(2, N - 4), gridy = pow(2, N - 4); *//Dimensions of grid*

    int blockx = pow(2, 4), blocky = pow(2, 4);

    dim3 dimGrid(gridx, gridy);

    dim3 dimBlock(blockx, blocky);

Grid and block dimensions. Each thread is responsible to calculate one element of distance matrix using this simple formula for thread ID:

int tid = (*blockIdx*.y \* *gridDim*.x + *blockIdx*.x) \* *blockDim*.x + *threadIdx*.x;

**runtime comparison**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **10** | **100** | **1000** | **10000** |
| **CPU** | **0.000002** | **0.000010** | **0.00569** | **5.168711** | **10+ min** |
| **GPU** | **0.000012** | **0.000073** | **0.000078** | **0.000641** | **0.006310** |

**Dijkstra for all vertexes**

**Functions**

int minDistance(int\* *dist*, int\* *stpSet*, int *n*)

A utility function to find the vertex with minimum distance value, from the set of vertices not yet included in the shortest path tree.

void dijkstra\_all(int\*\* *graph*, int\*\* *dist*, int\*\* *stpSet*, int *n*)

A function which runs Dijkstra for all vertices by executing the loop below:

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

dijkstra(graph, dist[i],stpSet[i],n,i);

}

void dijkstra(int\*\* *graph*,int\* *dist*,int\* *stpSet*, int *n*, int *src*)

Serial Implementation of Dijkstra algorithm.

\_\_global\_\_ void dijkstra\_all(int\*\* *graph*, int\*\* *dist*, int\*\* *stpSet*, int *n*)

CUDA kernel of the to run Dijkstra algorithm for each vertex. Due to the serial nature of Dijkstra algorithm each thread runs the algorithm for a single node. When working with very large datasets which is the main focus of high performance computing by GPUs we will get a better speed up.

    int gridx = pow(2, N - 4), gridy = pow(2, N - 4); *//Dimensions of grid*

    int blockx = pow(2, 4), blocky = pow(2, 4);

    dim3 dimGrid(gridx, gridy);

    dim3 dimBlock(blockx, blocky);

Grid and block dimensions.

int tid = (*blockIdx*.y \* *gridDim*.x + *blockIdx*.x) \* *blockDim*.x + *threadIdx*.x;

**runtime comparison**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **1** | **10** | **100** | **1000** | **10000** |
| **GPU** |  |  |  |  |  |
| **CPU** | **0.000002** | **0.000024** | **0.011901** | **9.340781** | **15+ min** |