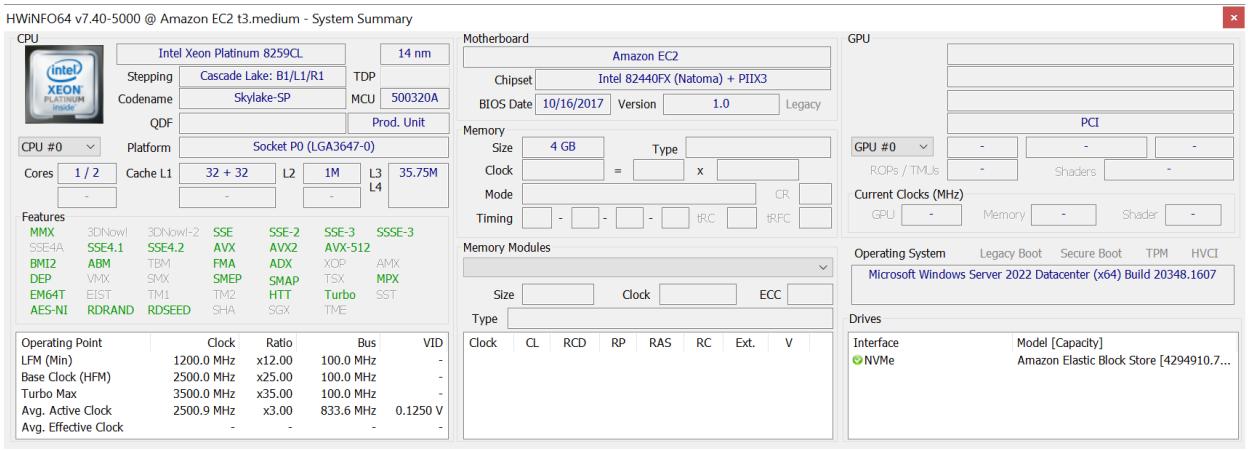
Bellman Ford Algorithm code optimization using SIMD vectorization and Tiling

Name: Maksuda Rabeya ID: 912154290
Name: Ramesh Adhikari ID: 912147172

Environment: Windows OS
IDE: Visual Studio community version
Compiler: MSVC
Service: Amazon cloud service
System Configuration:



Data Set: soc-sign-bitcoinotc.csv
Data link: https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html

Simple Bellman Ford Algorithm execution time : 1.139.

```
PS C:\code> cl bellmanford.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.35.32216.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

bellmanford.c
bellmanford.c(18): warning C4047: '=': 'int' differs in levels of indirection from 'void *'
bellmanford.c(53): warning C4098: 'bellmanFord': 'void' function returning a value
Microsoft (R) Incremental Linker Version 14.35.32216.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:bellmanford.exe
bellmanford.obj
PS C:\code> ./bellmanford.exe
Simple bellmanford compute time 1.139
PS C:\code>
```

Code segment for Bellman Ford algorithm:

```
void bellmanFord(int start, int* edge_from, int* edge_to, int* edge_weight, int n, int e, int* predecessor, int* dist) {

    // int dist[n];
    for (int i = 0; i < n + 1; i++) {
        if (i == start) {
            dist[i] = 0;
        }
        else {
            dist[i] = INF;
        }
        predecessor[i] = NULL;

    }


    // relax edges n-1 times
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < e; j++) {

            int u = edge_from[j];
            int v = edge_to[j];
            int  w = edge_weight[j];

            // exit(0);
            if (dist[u] + w < dist[v]) {

                dist[v] = dist[u] + w;
                predecessor[v] = u;

            }
        }

    }



    // check for negative-weight cycles
    for (int i = 0; i < e; i++) {
        int u = edge_from[i], v = edge_to[i], w = edge_weight[i];
        if (dist[u] + w < dist[v]) {
            printf("Graph contains negative-weight cycle\n");
            return;
        }
    }
```

SIMD Vectorization Bellman Ford Algorithm execution time: 0.968

```
PS C:\code> cl bellmanford_simd.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.35.32216.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

bellmanford_simd.c
bellmanford_simd.c(18): warning C4047: '=': 'int' differs in levels of indirection from 'void *'
bellmanford_simd.c(53): warning C4098: 'bellmanFord': 'void' function returning a value
Microsoft (R) Incremental Linker Version 14.35.32216.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:bellmanford_simd.exe
bellmanford_simd.obj
PS C:\code> ./bellmanford_simd.exe
Simple bellmanford compute time 1.218
PS C:\code> ./bellmanford_simd.exe
Simple bellmanford compute time 0.968
PS C:\code> ./bellmanford_simd.exe
Simple bellmanford compute time 1.025
PS C:\code>
```

SIMD Vectorization Bellman Ford Code Segment:

```
int SIMD_vectorization(int* node, int* edge_u, int* edge_v, int source, int n, int e, int* distance, int* predecessor)
{
    int* negative_loop;
    int i, j, e1;
    e1 = e - e % 4;
    for (i = 1; i < n + 1; i++) {
        if (i == source) distance[i] = 0;
        else distance[i] = MAX1;
        predecessor[i] = -1;
    }

    int u, v;
    __m128i vset1 = _mm_set_epi32(1, 1, 1, 1);
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < e1; j += 4) {

            __m128i vector_source = _mm_loadu_epi16(edge_u + j);
            __m128i vector_dest = _mm_loadu_epi16(edge_v + j);
            __m128i vector_src_dist = _mm_i32gather_epi32(distance, vector_source, 4);
            __m128i vector_dest_dist = _mm_i32gather_epi32(distance, vector_dest, 4);
            __m128i vector_src_dist1 = _mm_add_epi32(vector_src_dist, vset1);
            __m128i mask = _mm_cmplt_epi32(vector_src_dist1, vector_dest_dist);

            __m128i vector_src_mask = _mm_and_si128(mask, vector_src_dist1);
            __m128i vector_dest_mask = _mm_andnot_si128(mask, vector_dest_dist);
            __m128i vector_updated_dist = _mm_or_si128(vector_src_mask, vector_dest_mask);
            _mm_i32scatter_epi32(distance, vector_dest, vector_updated_dist, 4);

            __m128i vmaskpreu = _mm_and_si128(mask, vector_source);
            __m128i vmaskprev = _mm_andnot_si128(mask, _mm_i32gather_epi32(predecessor, vector_dest, 4));
            __m128i vnewpre = _mm_or_si128(vmaskpreu, vmaskprev);
            _mm_i32scatter_epi32(predecessor, vector_dest, vnewpre, 4);


        }
        for (j = e1; j < e; j++) {
            u = edge_u[j];
            v = edge_v[j];
            if (distance[u] + 1 < distance[v]) {
                distance[v] = distance[u] + 1;
                predecessor[v] = u;
            }
        }
    }
```

SIMD Vector optimization using Tiling of Bellman Ford Algorithm execution time: 0.001

```
PS C:\code> cl .\bellmanford_simd_optimization.c
Microsoft (R) C/C++ Optimizing Compiler Version 19.35.32216.1 for x86
Copyright (C) Microsoft Corporation.  All rights reserved.

bellmanford_simd_optimization.c
.\bellmanford_simd_optimization.c(21): warning C4047: '=': 'int' differs in levels of indirection from 'void *'
.\bellmanford_simd_optimization.c(56): warning C4098: 'bellmanFord': 'void' function returning a value
.\bellmanford_simd_optimization.c(264): warning C4047: 'function': 'int *' differs in levels of indirection from 'int'
.\bellmanford_simd_optimization.c(264): warning C4024: 'SIMD_vectorization_tiling': different types for formal and actual parameter 1
.\bellmanford_simd_optimization.c(264): warning C4047: 'function': 'int' differs in levels of indirection from 'int *'
.\bellmanford_simd_optimization.c(264): warning C4024: 'SIMD_vectorization_tiling': different types for formal and actual parameter 4
Microsoft (R) Incremental Linker Version 14.35.32216.1
Copyright (C) Microsoft Corporation.  All rights reserved.

/out:bellmanford_simd_optimization.exe
bellmanford_simd_optimization.obj
PS C:\code> .\bellmanford_simd_optimization.exe
Simple bellmanford compute time 0.001
PS C:\code>
```

Code Segment with tiling:

```
int  t = 4; // tile size 4
for (ib = 0;ib < n; ib += t) {
    for (jb = 0;jb < e1; jb += t) {
        for (i = ib; i < (ib - 1 + t); i++) {
            for (j = jb; j < (jb + t); j += 4) {

                __m128i vector_source = _mm_loadu_epi16(edge_u + j);
                __m128i vector_dest = _mm_loadu_epi16(edge_v + j);
                __m128i vector_src_dist = _mm_i32gather_epi32(distance, vector_source, 4);
                __m128i vector_dest_dist = _mm_i32gather_epi32(distance, vector_dest, 4);
                __m128i vector_src_dist1 = _mm_add_epi32(vector_src_dist, vset1);
                __m128i mask = _mm_cmplt_epi32(vector_src_dist1, vector_dest_dist);

                __m128i vector_src_mask = _mm_and_si128(mask, vector_src_dist1);
                __m128i vector_dest_mask = _mm_andnot_si128(mask, vector_dest_dist);
                __m128i vector_updated_dist = _mm_or_si128(vector_src_mask, vector_dest_mask);
                _mm_i32scatter_epi32(distance, vector_dest, vector_updated_dist, 4);

                __m128i vmaskpreu = _mm_and_si128(mask, vector_source);
                __m128i vmaskprev = _mm_andnot_si128(mask, _mm_i32gather_epi32(predecessor, vector_dest, 4));
                __m128i vnewpre = _mm_or_si128(vmaskpreu, vmaskprev);
                _mm_i32scatter_epi32(predecessor, vector_dest, vnewpre, 4);


            }
            for (j = e1; j < e; j++) {
                u = edge_u[j];
                v = edge_v[j];
                if (distance[u] + 1 < distance[v]) {
                    distance[v] = distance[u] + 1;
                    predecessor[v] = u;
                }
            }
        }

    }
```

Summary:

From this project we learn that we can achieve better performance using SIMD and we can achieve better performance using tiling with proper tile size.