# Final Project

Andres Calderon - SID:861243796

December 10, 2015

## 1 Motivation

1. What is the problem you would like to solve?

   Recently increase use of location-aware devices (such as GPS, Smart-phones and RFID tags) has allowed the collection of a vast amount of data with a spatial component linked to them. Different studies have focused in analyzing and mining this kind of collections [Leung, 2010][Miller and Han, 2001]. In this area, trajectory datasets have emerged as an interesting field where diverse kind of patterns can be identified [Zheng and Zhou, 2011][Vieira and Tsotras, 2013]. For instance, authors have proposed techniques to discover motion spatial patterns such as moving clusters[Kalnis et al., 2005], convoys[Jeung et al., 2008] and flocks [Benkert et al., 2006][Gudmundsson and van Kreveld, 2006]. In particular, [Vieira et al., 2009] and [Turdukulov et al., 2014] propose two novel algorithms to find moving flock patterns in very large spatio-temporal datasets.

2. Why is it important?

   A flock pattern is defined as a group of entities which move together for a defined lapse of time [Benkert et al., 2006]. Applications to this kind of patterns are diverse and range from surveillance to integrated transport systems. For example, [Turdukulov et al., 2014] explore the finding of this class of patterns to discover similarities between tropical cyclone paths. Also, [Calderon Romero, 2011] finds moving flock patterns in iceberg trajectories to understand their movement behavior and how they related to changes in ocean's currents.

3. What is your plan/outline of your solution?

   The algorithms proposed by [Vieira et al., 2009] and [Turdukulov et al., 2014] share the same initial strategy to detect flock patterns. In that, first they find clusters of points which could be close enough to initiate a flock for each time interval. This is a costly operation due to the large number of points and intervals to be analyzed. The technique uses a grid-based index and a stencil (see figure 1) to speed up the process but the complexity is still high. My plan is to allow individual threads to compute each of the stencils in the grid in parallel.

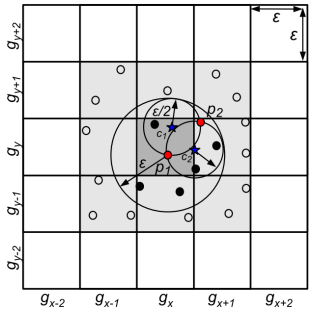
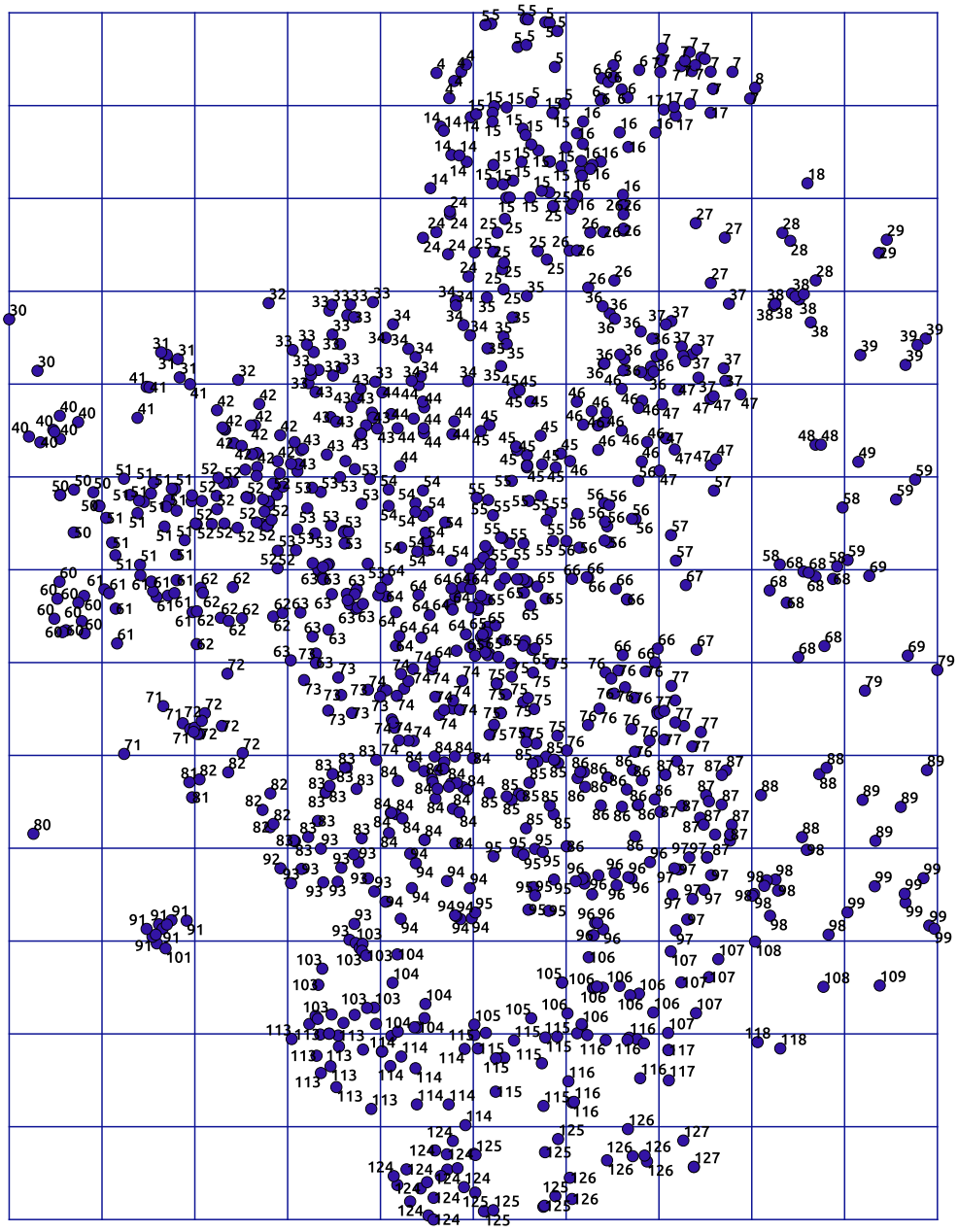



Figure 1: Grid-based index used in [Vieira et al., 2009].

## 2 Data



# 3 Code

Full code and other materials are available at Calderon Romero [2015].

## 3.1 bfe.cu

```
#include <stdio.h>
#include <stdlib.h>
#include <thrust/sort.h>
#include <thrust/functional.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>
#include "bfe.h"
#include "kernel.cu"

int main(int argc,char *argv[]){
  if(argc != 4){
    printf("Usage: %s TIMESTAMP EPSILON MU", argv[0]);
    return 1;
  }
  const int TIMESTAMP = atoi(argv[1]);
  const int EPSILON = atoi(argv[2]);
  const int E2 = EPSILON * EPSILON;
  const int MU = atoi(argv[3]);
  cudaError_t cuda_ret;

  FILE *in;
  FILE *out;
  in = fopen("oldenburg.csv", "r");
  out = fopen("output.csv", "w");
  fprintf(out, "oid;time;lat;lon;grid_id\n");
  char line[1024];
  int n = 0;
  short time;
  int lat; int lon;
  int max_lat = INT_MIN; int min_lat = INT_MAX;
  int max_lon = INT_MIN; int min_lon = INT_MAX;
  int M = 0;
  int N = 0;
  while (fgets(line, 1024, in)){
    atoi(strtok(line, ";"));
    if(atoi(strtok(NULL, ";\n")) > TIMESTAMP) continue;
    lat = atoi(strtok(NULL, ";\n"));
    if(lat > max_lat) max_lat = lat;
    if(lat < min_lat) min_lat = lat;
    lon = atoi(strtok(NULL, ";\n"));
    if(lon > max_lon) max_lon = lon;
    if(lon < min_lon) min_lon = lon;
    n++;
  }
  int *x;
  x = (int*) malloc( sizeof(int) * n);
  int *y;
  y = (int*) malloc( sizeof(int) * n);
  int *g;
  g = (int*) malloc( sizeof(int) * n);
  int *i;
  i = (int*) malloc( sizeof(int) * n);
  printf("Min and max latitude:\t(%d, %d)\n", min_lat, max_lat);
  printf("Min and max longitude:\t(%d, %d)\n", min_lon, max_lon);
  M = (max_lat - min_lat) / EPSILON + 1;
  N = (max_lon - min_lon) / EPSILON + 1;
  rewind(in);
  int j = 0;
  while (fgets(line, 1024, in)){
    atoi(strtok(line, ";"));
    time = atoi(strtok(NULL, ";\n"));
    if(time > TIMESTAMP) continue;
    lat = atoi(strtok(NULL, ";\n"));
    lon = atoi(strtok(NULL, ";\n"));
```

```
65        g[j] = M * ((N - 1) - ((lon - min_lon) / EPSILON)) + ((lat - min_lat) / EPSILON);
66        x[j] = lat;
67        y[j] = lon;
68        i[j] = j;
69             //printf("%d;%hi;%d;%d;%d\n", oid, time, lat, lon, g[j]);
70        j++;
71      }
72      printf("Number of points:\t%d\n", n);
73      printf("M x N : %d x %d\n", M, N);
74      int c = M * N;
75      //int r = createGrid("grid.shp", EPSILON, min_lat, max_lat, min_lon, max_lon);
76      printf("Sorting arrays...\n");
77      thrust::device_vector<int> d_x(x, x + n);
78      thrust::device_vector<int> d_y(y, y + n);
79      thrust::device_vector<int> d_g(g, g + n);
80      thrust::device_vector<int> d_i(i, i + n);
81      thrust::sort_by_key(d_g.begin(), d_g.end(), d_i.begin());
82      thrust::gather(d_i.begin(), d_i.end(), d_x.begin(), d_x.begin());
83      thrust::gather(d_i.begin(), d_i.end(), d_y.begin(), d_y.begin());
84      thrust::copy(d_g.begin(), d_g.end(), g);
85      thrust::copy(d_i.begin(), d_i.end(), i);
86      thrust::copy(d_x.begin(), d_x.end(), x);
87      thrust::copy(d_y.begin(), d_y.end(), y);
88
89      printf("Counting point indices...\n");
90      int *a;
91      a = (int*) malloc(sizeof(int) * c);
92      int *b;
93      b = (int*) malloc(sizeof(int) * c);
94      a[0] = g[0];
95      b[0] = 0;
96      int k = 0;
97      for(j = 0; j < n; j++){
98        if(g[j] != a[k]){
99          k++;
100         a[k] = g[j];
101         b[k] = j;
102       }
103     }
104     b[++k] = n;
105
106     int *x_d, *y_d, *g_d;
107     int *a_d, *b_d;
108     unsigned long *N_DISKS;
109     unsigned long *result;
110
111     result = (unsigned long*) malloc(sizeof(long) * c);
112
113     printf("cudaMalloc and cudaMemcpy stage...\n");
114     cuda_ret = cudaMalloc((void **) &x_d, sizeof(int) * n);
115     if(cuda_ret != cudaSuccess){
116       printf("\nChecking cudaMalloc for x ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
117       exit(EXIT_FAILURE);
118     }
119     cuda_ret = cudaMalloc((void **) &y_d, sizeof(int) * n);
120     if(cuda_ret != cudaSuccess){
121       printf("\nChecking cudaMalloc for y ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
122       exit(EXIT_FAILURE);
123     }
124     cuda_ret = cudaMalloc((void **) &g_d, sizeof(int) * n);
125     if(cuda_ret != cudaSuccess){
126       printf("\nChecking cudaMalloc for g ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
127       exit(EXIT_FAILURE);
128     }
129     cuda_ret = cudaMalloc((void **) &N_DISKS, sizeof(long) * c);
130     if(cuda_ret != cudaSuccess){
131       printf("\nChecking cudaMalloc for N_DISKS ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
```

```
132       exit(EXIT_FAILURE);
133     }
134     cuda_ret = cudaMalloc((void **) &a_d, sizeof(int) * c);
135     if(cuda_ret != cudaSuccess){
136       printf("\nChecking cudaMalloc for a ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
137       exit(EXIT_FAILURE);
138     }
139     cuda_ret = cudaMalloc((void **) &b_d, sizeof(int) * c);
140     if(cuda_ret != cudaSuccess){
141       printf("\nChecking cudaMalloc for b ...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
142       exit(EXIT_FAILURE);
143     }
144     cudaDeviceSynchronize();
145     cuda_ret = cudaMemcpy(x_d, x, sizeof(int) * n, cudaMemcpyHostToDevice);
146     if(cuda_ret != cudaSuccess){
147       printf("\nChecking cudaMemcpy for x_d...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
148       exit(EXIT_FAILURE);
149     }
150     cuda_ret = cudaMemcpy(y_d, y, sizeof(int) * n, cudaMemcpyHostToDevice);
151     if(cuda_ret != cudaSuccess){
152       printf("\nChecking cudaMemcpy for y_d...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
153       exit(EXIT_FAILURE);
154     }
155     cuda_ret = cudaMemcpy(g_d, g, sizeof(int) * n, cudaMemcpyHostToDevice);
156     if(cuda_ret != cudaSuccess){
157       printf("\nChecking cudaMemcpy for g_d...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
158       exit(EXIT_FAILURE);
159     }
160     cuda_ret = cudaMemcpy(a_d, a, sizeof(int) * c, cudaMemcpyHostToDevice);
161     if(cuda_ret != cudaSuccess){
162       printf("\nChecking cudaMemcpy for a_d...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
163       exit(EXIT_FAILURE);
164     }
165     cuda_ret = cudaMemcpy(b_d, b, sizeof(int) * c, cudaMemcpyHostToDevice);
166     if(cuda_ret != cudaSuccess){
167       printf("\nChecking cudaMemcpy for b_d...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
168       exit(EXIT_FAILURE);
169     }
170     cudaDeviceSynchronize();
171     const dim3 grid(1, 1, 1);
172     const dim3 block(k, 1, 1);
173
174     // Calling the kernel...
175     printf("Running the kernel...\nk=%d\n", k);
176     parallelBFE<<<grid, block>>>(x_d, y_d, g_d, a_d, b_d, n, k, M, N, E2, N_DISKS);
177
178     cuda_ret = cudaDeviceSynchronize();
179     if(cuda_ret != cudaSuccess){
180       printf("\nError lunching kernel...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret), __FILE__,
          ↪  __LINE__);
181       exit(EXIT_FAILURE);
182     }
183
184     cuda_ret = cudaMemcpy(result, N_DISKS, sizeof(long) * c, cudaMemcpyDeviceToHost);
185     if(cuda_ret != cudaSuccess){
186       printf("\nChecking cudaMemcpy for result...  %s in %s at line %d\n", cudaGetErrorString(cuda_ret),
          ↪  __FILE__, __LINE__);
187       exit(EXIT_FAILURE);
188     }
189     cudaDeviceSynchronize();
190
191     printf("\n");
192     for(int j = 0; j < k; j++){
```

```
    if(j % M == 0) printf("\n");
    if(result[j] >= MU){
      printf("%3d->%3li  ", a[j], result[j]);
    }
  }
  printf("\n");

  cudaFree(x_d);
  cudaFree(y_d);
  cudaFree(g_d);
  cudaFree(N_DISKS);

  free(x);
  free(y);
  free(g);
  free(i);
  free(result);

  return 0;
}
```

### 3.1.1 kernel.cu

```
#include <stdio.h>

__device__ int distance(int x1, int y1, int x2, int y2){
  int dx = x2 - x1;
  int dy = y2 - y1;
  return (dx * dx) + (dy * dy);
}

__device__ int findPosition(const int *a, int k, int b, int top){
  if(b < 0){
    return -1;
  }
  if(b > top){
    return -2;
  }
  for(int i = 0; i < k; i++){
    if(a[i] == b){
      return i;
    }
  }
  return -3;
}

__global__ void parallelBFE(const int *x, const int *y, int *g, const int *a, const int *b, int n, int k,
 ↪  int M, int N, int E, unsigned long *N_DISKS){
  int t = threadIdx.x;
  //int px[250];
  //int py[250];
  //int h;
  unsigned long j = 0;

  // Center-Medium
  int cm = a[t];
  for(int i = b[t]; i < b[t + 1]; i++){
    //px[j] = x[i];
    //py[j] = y[i];
    j++;
  }
  //h = j;

  // Left-Medium
  int lm;
  if(cm % M == 0){
    lm = -1;
  } else {
    lm = findPosition(a, k, cm - 1, M*N);
  }
```

```
47      if(lm >= 0){
48        for(int i = b[lm]; i < b[lm + 1]; i++){
49          //px[j] = x[i];
50          //py[j] = y[i];
51          j++;
52        }
53      }
54      // Right-Medium
55      int rm;
56      if(cm % M == M - 1){
57        rm = -1;
58      } else {
59        rm = findPosition(a, k, cm + 1, M*N);
60      }
61      if(rm >= 0){
62        for(int i = b[rm]; i < b[rm + 1]; i++){
63          //px[j] = x[i];
64          //py[j] = y[i];
65          j++;
66        }
67      }
68      // Center-Up
69      int cu = cm - M;
70      cu = findPosition(a, k, cu, M*N);
71      if(cu >= 0){
72        for(int i = b[cu]; i < b[cu + 1]; i++){
73          //px[j] = x[i];
74          //py[j] = y[i];
75          j++;
76        }
77      }
78      // Left-Up
79      int lu;
80      if(cm % M == 0){
81        lu = -1;
82      } else {
83        lu = findPosition(a, k, cm - M - 1, M*N);
84      }
85      if(lu >= 0){
86        for(int i = b[lu]; i < b[lu + 1]; i++){
87          //px[j] = x[i];
88          //py[j] = y[i];
89          j++;
90        }
91      }
92      // Right-Up
93      int ru;
94      if(cm % M == M - 1){
95        ru = -1;
96      } else {
97        ru = findPosition(a, k, cm - M + 1, M*N);
98      }
99      if(ru >= 0){
100       for(int i = b[ru]; i < b[ru + 1]; i++){
101         //px[j] = x[i];
102         //py[j] = y[i];
103         j++;
104       }
105     }
106     // Center-Down
107     int cd = cm + M;
108     cd = findPosition(a, k, cd, M*N);
109     if(cd >= 0){
110       for(int i = b[cd]; i < b[cd + 1]; i++){
111         //px[j] = x[i];
112         //py[j] = y[i];
113         j++;
114       }
115     }
116     // Left-Down
```

```
  int ld;
  if(cm % M == 0){
    ld = -1;
  } else {
    ld = findPosition(a, k, cm + M - 1, M*N);
  }
  if(ld >= 0){
    for(int i = b[ld]; i < b[ld + 1]; i++){
      //px[j] = x[i];
      //py[j] = y[i];
      j++;
    }
  }
  // Right-Down
  int rd;
  if(cm % M == M - 1){
    rd = -1;
  } else {
    rd = findPosition(a, k, cm + M + 1, M*N);
  }
  if(rd >= 0){
    for(int i = b[rd]; i < b[rd + 1]; i++){
      //px[j] = x[i];
      //py[j] = y[i];
      j++;
    }
  }
  N_DISKS[t] = j;
}
```

# 4 Output

```
================================================================================
=== Increasing number of points ================================================
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 0 2000 100
Min and max latitude:  (1314, 21542)
Min and max longitude:  (4391, 30096)
Number of points:  700
M x N : 11 x 13
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=100




 47->102
 57->109   58->118   59->122   60->105
 69->115   70->132   71->123
 80->101   81->134   82->133   83->103
 92->104   93->116   94->101



real  0m1.616s
user  0m1.375s
sys  0m0.115s
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 1 2000 100
Min and max latitude:  (1314, 21542)
Min and max longitude:  (4297, 30161)
Number of points:  1700
M x N : 11 x 13
Sorting arrays...
Counting point indices...
```

```
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=104


  5->127    6->139   15->112   16->178   17->187   18->119
 26->147   27->185   28->186   29->127   35->112   36->148   37->175
 38->175   39->174   40->135   45->160   46->238   47->251   48->239   49->204
 50->186   51->132   56->192   57->263   58->283   59->302   60->257   61->205
 62->107   67->166   68->224   69->267   70->310   71->288   72->218   73->104
 79->139   80->238   81->313   82->309   83->242   84->136
 90->107   91->190   92->243   93->266   94->231   95->167   96->107
101->105  102->158  103->200  104->207  105->194  106->163  107->102  113->105
114->147  115->156  116->149  117->128  125->142  126->154  127->120
137->110

real  0m0.145s
user  0m0.024s
sys  0m0.081s

================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 5 2000 100
Min and max latitude:  (858, 21542)
Min and max longitude:  (4297, 30800)
Number of points:  5652
M x N : 11 x 14
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=112


  5->138    6->233   15->198   16->377   17->473   18->316   19->137   26->300   27->565   28->655
    ↪  29->436
 30->183   36->169   37->395   38->588   39->622   40->469   41->234   42->127   45->179   46->319
 47->463   48->583   49->622   50->601   51->481   52->297   53->137   55->180   56->412   57->724
 58->842   59->862   60->713   61->652   62->488   63->312   64->140   66->243   67->528   68->874
  ↪  69->994
 70->1147   71->953   72->768   73->381   74->221   75->110   77->191   78->462   79->761   80->882
 81->1135   82->1011   83->863   84->377   85->217   86->101   89->255   90->453   91->714
 92->1057   93->1073   94->932   95->497   97->125  100->158  101->320  102->553  103->768
104->879  105->847  106->622  107->396  108->159  112->299  113->508  114->644
115->696  116->655  117->575  118->366  119->163  124->361  125->485  126->519  127->503
128->425  129->261  130->110  135->298  136->427  137->475  138->405  139->288  140->136  147->260
 ↪  148->327
149->283  150->174

real  0m0.133s
user  0m0.029s
sys  0m0.082s
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 10 2000 100
Min and max latitude:  (858, 21542)
Min and max longitude:  (4297, 30838)
Number of points:  10257
M x N : 11 x 14
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=112


  5->255    6->430   15->362   16->657   17->810   18->534   19->239   26->557   27->1012   28->1145
    ↪  29->764
 30->330   36->288   37->692   38->1063   39->1077   40->815   41->400   42->232   44->140   45->309
  ↪  46->555
 47->792   48->1048   49->1167   50->1118   51->874   52->521   53->250   55->290   56->673   57->1275
 58->1507   59->1641   60->1354   61->1218   62->857   63->551   64->255   66->384   67->866   68->1604
  ↪  69->1893
```

```
 70->2281  71->1833  72->1459  73->672  74->415  75->205  76->122  77->292  78->733  79->1382
  ↪  80->1718
 81->2306  82->1967  83->1607  84->638  85->384  86->183  87->123  88->157  89->418  90->815
  ↪  91->1370
 92->2072  93->2089  94->1742  95->888  97->215  98->106  99->138  100->272  101->547  102->1003
  ↪  103->1434
104->1663  105->1538  106->1109  107->699  108->267  109->131  110->120  111->161  112->537  113->919
  ↪  114->1174
115->1278  116->1190  117->1038  118->666  119->281  120->137  122->112  124->677  125->894  126->938
  ↪  127->887
128->745  129->474  130->184  135->563  136->766  137->835  138->697  139->513  140->240  147->445
  ↪  148->580
149->476  150->306

real  0m0.129s
user  0m0.040s
sys  0m0.068s
================================================================================
=== Decresing Epsilon ==========================================================
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 120 2000 600
Min and max latitude:  (292, 22279)
Min and max longitude:  (4191, 30838)
Number of points:  54636
M x N : 11 x 14
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=114


  5->812  6->1519  16->2051  17->3383  18->2539  19->1715  27->3859  28->5951  29->4225  30->3087
 31->1009  37->3615  38->5042  39->7109  40->4787  41->4166  42->1947  43->978  45->716
 46->1686  47->2853  48->6760  49->8164  50->9590  51->6328  52->4845  53->2096  54->1057
  ↪  56->1501
 57->4009  58->6181  59->10052  60->10847  61->10877  62->6956  63->4445  64->2029  65->1092
  ↪  66->624  67->1948
 68->5098  69->8395  70->13333  71->13782  72->11823  73->6131  74->3186  75->1516  76->946
  ↪  78->1830  79->4624
 80->8111  81->12517  82->12670  83->10082  84->4725  85->2434  86->1309  87->875  89->1326
  ↪  90->3001
 91->6641  92->10940  93->11795  94->9268  95->5152  96->3469  97->2137  98->943  100->1078
  ↪  101->2144
102->5218  103->7631  104->9223  105->7499  106->5986  107->4180  108->2522  109->983  111->723  112->1505
113->3897  114->5943  115->7895  116->6758  117->6013  118->4105  119->2558  120->945  123->828  124->2612
  ↪  125->3970
126->5608  127->4831  128->4050  129->2463  130->1197  135->1427  136->2431  137->3799  138->3648
  ↪  139->2784  140->1247
147->1195  148->1878  149->1785  150->1243

real  0m0.131s
user  0m0.044s
sys  0m0.070s
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 120 1000 600
Min and max latitude:  (292, 22279)
Min and max longitude:  (4191, 30838)
Number of points:  54636
M x N : 22 x 27
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=385


 33->668  34->694  35->709  36->851  37->802  54->728  55->968  56->885  57->1114  58->1361
  ↪  59->1250  60->644  76->924  77->1253  78->1160  79->1492
 80->1498  81->1165  98->1049  99->1476  100->1265  101->1744  102->1532  103->1117  120->1555
  ↪  121->2080  122->1653  123->2024  124->1542  125->1102
```

```
128->616  142->1901  143->2389  144->1923  145->1946  146->1462  149->840  150->906
163->729  164->2318  165->2647  166->2281  167->1787  168->1454  169->1480  170->851  171->1199  172->1307  173->763  184->817
185->962  186->2752  187->2948  188->2874  189->2178  190->1937  191->2024  192->991  193->1003  194->983  195->627  203->920  204->1217  205->1759  206->1789  207->2149
208->2989  209->3451  210->3223  211->3021  212->2288  213->1998  214->847  215->719  216->694  224->972  225->1528  226->2140  227->3008  228->2921  229->3008  230->2751
231->3753  232->3469  233->3535  234->2196  235->1647  246->833  247->1654  248->2703  249->3818  250->3250  251->3088  252->2902  253->4207  254->3923  255->3323
256->1785  257->1014  267->771  268->1180  269->1749  270->2533  271->3644  272->3404  273->3264  274->3881  275->4653  276->4080  277->2224  278->1152  279->720  280->624  282->695
289->617  290->749  291->1152  292->1792  293->2915  294->2794  295->2960  296->4659  297->5043  298->4345  299->1502  300->831  303->619  304->820  305->630
312->772  313->957  314->1231  315->2394  316->2719  317->2986  318->4283  319->4151  320->3610  321->1147  322->771
335->653  336->1075  337->1935  338->2305  339->2432  340->3057  341->3273  342->2894  343->1464  344->1004  345->730  357->813  358->1187  359->1690  360->2206  361->2726
362->2608  363->2586  364->2158  365->1844  366->1574  367->1420  368->1134  369->613  379->731  380->1054  381->1359  382->2234  383->3015  384->2904  385->2843  386->2184  387->2121  388->1503
389->1501  390->1545  391->1245  392->899  402->679  403->1034  404->1991  405->2765  406->2833  407->2467  408->2116  409->2043  410->1695  411->1552  412->1681  413->1489  414->1190
415->668  425->962  426->1941  427->2539  428->2371  429->1965  430->1800  431->1714  432->1374  433->1189  434->1294  435->1370  436->1185  437->731
448->1502  449->1825  450->1391  451->1233  452->1252  453->1393  454->1355  455->1065  456->920  457->843  458->699  469->943  470->1332  471->1673  472->1133  473->1144
474->1341  475->1644  476->1452  477->994  478->645  479->626  491->683  492->1148  493->1300  494->1098  495->994  496->1324  497->1532  498->1443  499->811
514->999  515->1278  516->1212  517->927  518->1253  519->1328  520->1316  521->668  536->614  537->884  538->1014  539->665  540->823  541->774  542->937  560->633
562->642  563->617  564->669

real  0m0.133s
user  0m0.044s
sys  0m0.074s
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 120 800 600
Min and max latitude:  (292, 22279)
Min and max longitude:  (4191, 30838)
Number of points:  54636
M x N : 28 x 34
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=580


 70->628   73->685   74->756   75->711   97->637   98->701  101->974  102->1096  103->880
125->752  126->725  127->646  128->838  129->1135  130->1027  131->724  152->602  153->762  154->621  155->795  156->1101  157->1184  158->897  180->778  181->1100  182->1054  183->1222
184->1414  185->1358  186->844  208->939  209->1325  210->1415  211->1196  212->1514  213->1406  214->937  237->1576  238->1813  239->1176
240->1231  241->1289  242->1041  246->706  247->623  264->712  265->1700  266->1864  267->1290  268->708  269->1036  270->1024  271->622  273->790
274->1021  275->865  293->1865  294->2038  295->1820  297->1212  298->1246  299->1039  301->821  302->798  303->701
318->670  319->918  320->948  321->2349  322->2444  323->2289  324->1303  325->1695  326->1717  327->1296  328->696  329->671  330->638
342->650  343->889  344->1058  345->1270  346->1386  347->1717  348->1565  349->2158  350->2329  351->2391  352->2173  353->1992  354->1668  355->1077
371->1022  372->1564  373->2199  374->2350  375->2473  376->1815  377->1821  378->2379  379->2471  380->2513  381->1630  382->1291  383->724  398->941  399->1468  400->2058  401->2656
402->2317  403->2162  404->1345  405->1822  406->2222  407->2687  408->2397  409->1496  410->879  425->872  426->1124  427->1578  428->2026  429->2678  430->2243  431->2054  432->1508  433->2694  434->3175
435->3131  436->2009  437->1071  438->761  453->653  454->853  455->1347  456->1628  457->2300  458->1948  459->1965  460->1687  461->3550  462->3476  463->3139  464->1248  465->852
483->725  484->1084  485->2029  486->2030  487->2030  488->1666  489->3609  490->3751  491->3136  492->908
512->645  513->1668  514->1793  515->1804  516->1200  517->3003  518->2884  519->2580  520->701
540->746  541->1467  542->1554  543->1504  544->1100  545->2170  546->2352  547->2153  548->1060  549->755  550->700  568->732  569->1184  570->1626  571->1583  572->1547  573->1526  574->1807
```

```
575->1533  576->1188  577->913  578->1015  579->851  595->675  596->796  597->1038  598->1397  599->1671
 ↪  600->2043  601->1609  602->1728  603->1265  604->1326  605->896  606->976  607->1117  608->1069
 ↪  609->807
624->616  625->749  626->1113  627->1909  628->2305  629->2055  630->1727  631->1653  632->1544  633->1192
 ↪  634->884  635->1085  636->1121  637->1119  638->801
653->723  654->1012  655->1859  656->2022  657->1928  658->1416  659->1539  660->1482  661->1215  662->778
 ↪  663->847  664->1079  665->1261  666->1026  667->723  681->609  682->1159  683->1647  684->1654
 ↪  685->1387  686->1173  687->1321
688->1239  689->1032  690->750  691->732  692->852  693->823  694->716  710->1197  711->1350  712->1242
 ↪  713->851  714->803  715->808  716->969  717->928  718->781  719->705  720->784  721->621  722->600
737->677  738->1036  739->1114  740->954  741->694  742->612  743->665  744->852  745->902  746->699
 ↪  747->641  765->681  766->928  767->1053  768->772  769->604  771->884
772->1179  773->1216  774->810  794->753  795->904  796->820  797->663  799->822  800->1009  801->1033
 ↪  802->760  823->963  824->927
825->737  827->849  828->1018  829->960  830->694  851->705  852->794  853->784  855->637


real  0m0.135s
user  0m0.049s
sys  0m0.075s
================================================================================
storm.ee.ucr.edu /home/tempmaj/classacc2391/PhD/Y1Q1/GPU/project $ time ./bfe 120 600 600
Min and max latitude:  (292, 22279)
Min and max longitude:  (4191, 30838)
Number of points:  54636
M x N : 37 x 45
Sorting arrays...
Counting point indices...
cudaMalloc and cudaMemcpy stage...
Running the kernel...
k=932


136->634  172->673
208->645  209->725  245->661  246->612  280->640  281->681
282->694  313->621  314->662  317->666  318->851  319->760  350->832  351->968  352->658  353->704
 ↪  354->677  355->959  356->841  357->666
387->970  388->1160  389->922  390->630  392->917  393->853  394->764  424->1061  425->1289  426->1274
 ↪  429->635  430->821  431->754
461->786  462->1193  463->1340  464->627  467->717  468->720  474->637
499->1348  500->1472  501->1044  504->763  505->805  506->622
536->1364  537->1562  538->1500  541->963  542->962  543->768  571->630  572->817  573->1620  574->1815
 ↪  575->1598  576->652  577->871  578->1328  579->1334
580->871  602->632  603->736  604->836  605->883  606->960  607->1047  608->1146  609->1172  610->1262
 ↪  611->1477  612->1507  613->1253  614->1276  615->1396  616->1193  617->761
639->789  640->975  641->1073  642->1283  643->1253  644->1337  645->1178  646->1214  648->1402  649->1511
 ↪  650->1571  651->1134  652->1045  653->895
675->768  676->821  677->1197  678->1470  679->1821  680->1503  681->1330  682->951  683->761  685->1185
 ↪  686->1611  687->1635  688->1015  689->700  712->682  713->930  714->1252  715->1505  716->1541
 ↪  717->1129  718->975
720->667  721->1019  722->1643  723->1864  724->1338  725->903  726->647  750->905  751->1323  752->1689
 ↪  753->1499  754->1076  755->786  756->695  757->934  758->1819  759->2171  760->2202  761->1236
 ↪  762->887  763->641
787->880  788->1154  789->1408  790->1338  791->1140  792->955  793->1021  794->1258  795->2417  796->2365
 ↪  797->2235  798->920  799->651
825->690  826->1123  827->1438  828->1350  829->1022  830->1030  831->1375  832->2658  833->2548  834->2111
 ↪  863->871
864->1320  865->1242  866->1105  867->915  868->1218  869->2409  870->2250  871->1895  900->698  901->1057
 ↪  902->967  903->887  905->942  906->1959  907->2014  908->1656
938->1032  939->979  940->941  942->672  943->1329  944->1598  945->1419  946->683
975->989  976->1026  977->982  978->613  979->727  980->990  981->1229  982->964  983->680  1012->801
 ↪  1013->1005  1014->946  1015->904  1016->745  1017->790  1018->972  1019->884  1020->805  1024->720
1050->824  1051->966  1052->1337  1053->1087  1054->859  1055->700  1056->771  1057->898  1061->709
 ↪  1062->775  1063->732
1087->752  1088->1223  1089->1586  1090->1391  1091->1017  1092->856  1093->1070  1094->1039  1095->840
 ↪  1098->612  1099->726  1100->777  1101->728  1102->609  1125->1289  1126->1504  1127->1405  1128->864
 ↪  1129->827
1130->1078  1131->1000  1132->851  1137->774  1138->861  1139->735  1161->713  1162->1228  1163->1260
 ↪  1164->1094  1165->705  1166->827  1167->980  1168->908  1169->790  1175->628
1176->641  1198->786  1199->969  1200->852  1201->721  1205->643  1206->613  1234->614
```

```
1235->876  1236->841  1237->683  1238->621  1243->618  1271->618  1272->822  1273->675  1280->601
1309->663  1346->608
1390->718  1391->711  1392->618
1427->654  1428->628  1459->625  1464->688


real  0m0.147s
user  0m0.045s
sys  0m0.084s
```

# 5 Profiler

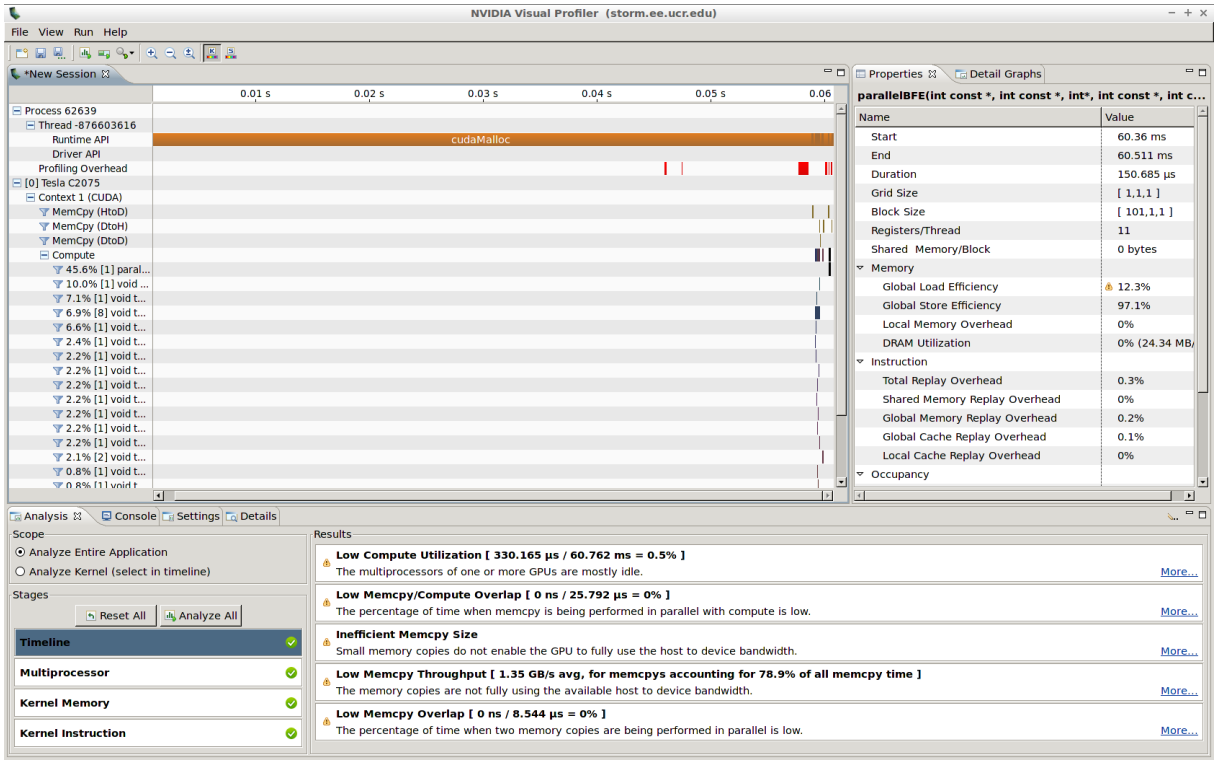



Figure 2: NVVP performance analysis for T1-E2K-M100.

# References


Marc Benkert, Joachim Gudmundsson, Florian Hübner, and Thomas Wolle. Reporting Flock Patterns. In Yossi Azar and Thomas Erlebach, editors, *Algorithms – ESA 2006*, number 4168 in Lecture Notes in Computer Science, pages 660–671. Springer Berlin Heidelberg, September 2006. ISBN 978-3-540-38875-3 978-3-540-38876-0. URL `http://link.springer.com/chapter/10.1007/11841036_59`. DOI: 10.1007/11841036_59.

Andres Oswaldo Calderon Romero. Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach. Master's thesis, University of Twente, 2011. URL `https://www.itc.nl/library/papers_2011/msc/gem/calderon.pdf`.

Andres Oswaldo Calderon Romero. Github personal repository, 2015. URL `https://github.com/aocalderon/PhD/tree/master/Y1Q1/GPU/project`.

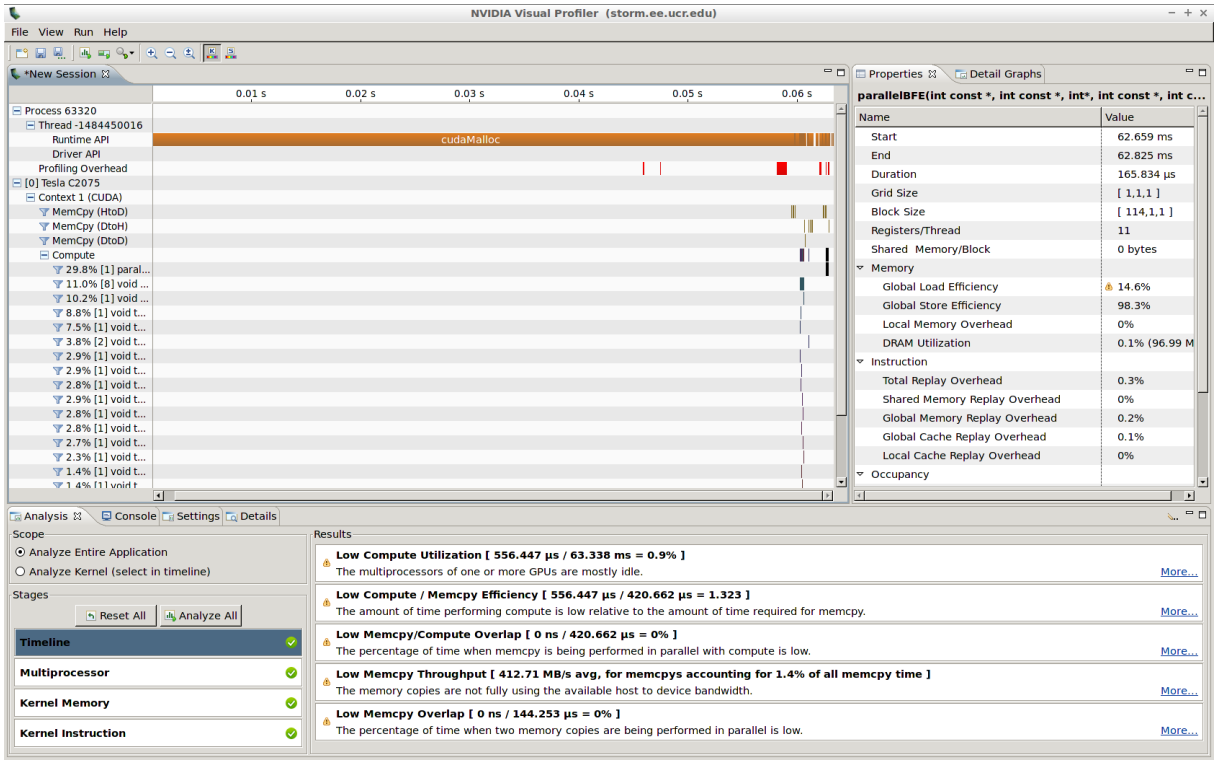



Figure 3: NVVP performance analysis for T120-E2K-M100.

Joachim Gudmundsson and Marc van Kreveld. Computing Longest Duration Flocks in Trajectory Data. In *Proceedings of the 14th Annual ACM International Symposium on Advances in Geographic Information Systems*, GIS '06, pages 35–42, New York, NY, USA, 2006. ACM. ISBN 1-59593-529-0. doi: 10.1145/1183471.1183479. URL `http://doi.acm.org/10.1145/1183471.1183479`.

Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of Convoys in Trajectory Databases. *Proc. VLDB Endow.*, 1(1):1068–1080, August 2008. ISSN 2150-8097. doi: 10.14778/1453856.1453971. URL `http://dx.doi.org/10.14778/1453856.1453971`.

Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On Discovering Moving Clusters in Spatio-temporal Data. In Claudia Bauzer Medeiros, Max J. Egenhofer, and Elisa Bertino, editors, *Advances in Spatial and Temporal Databases*, number 3633 in Lecture Notes in Computer Science, pages 364–381. Springer Berlin Heidelberg, August 2005. ISBN 978-3-540-28127-6 978-3-540-31904-7. URL `http://link.springer.com/chapter/10.1007/11535331_21`. DOI: 10.1007/11535331_21.

Yee Leung. *Knowledge Discovery in Spatial Data*. Springer Science & Business Media, March 2010. ISBN 978-3-642-02664-5.

Harvey J. Miller and Jiawei Han. *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, Inc., Bristol, PA, USA, 2001. ISBN 0415233690.

Ulanbek Turdukulov, Andres Oswaldo Calderon Romero, Otto Huisman, and Vasilios Retsios. Visual mining of moving flock patterns in large spatio-temporal data sets using a frequent pattern approach. *International Journal of Geographical Information Science*, 28(10):2013–2029, October 2014. ISSN 1365-8816. doi: 10.1080/13658816.2014.889834. URL `http://dx.doi.org/10.1080/13658816.2014.889834`.

Marcos R. Vieira and Vassilis Tsotras. *Spatio-Temporal Databases: Complex Motion Pattern Queries*. Springer Science & Business Media, October 2013. ISBN 978-3-319-02408-0.

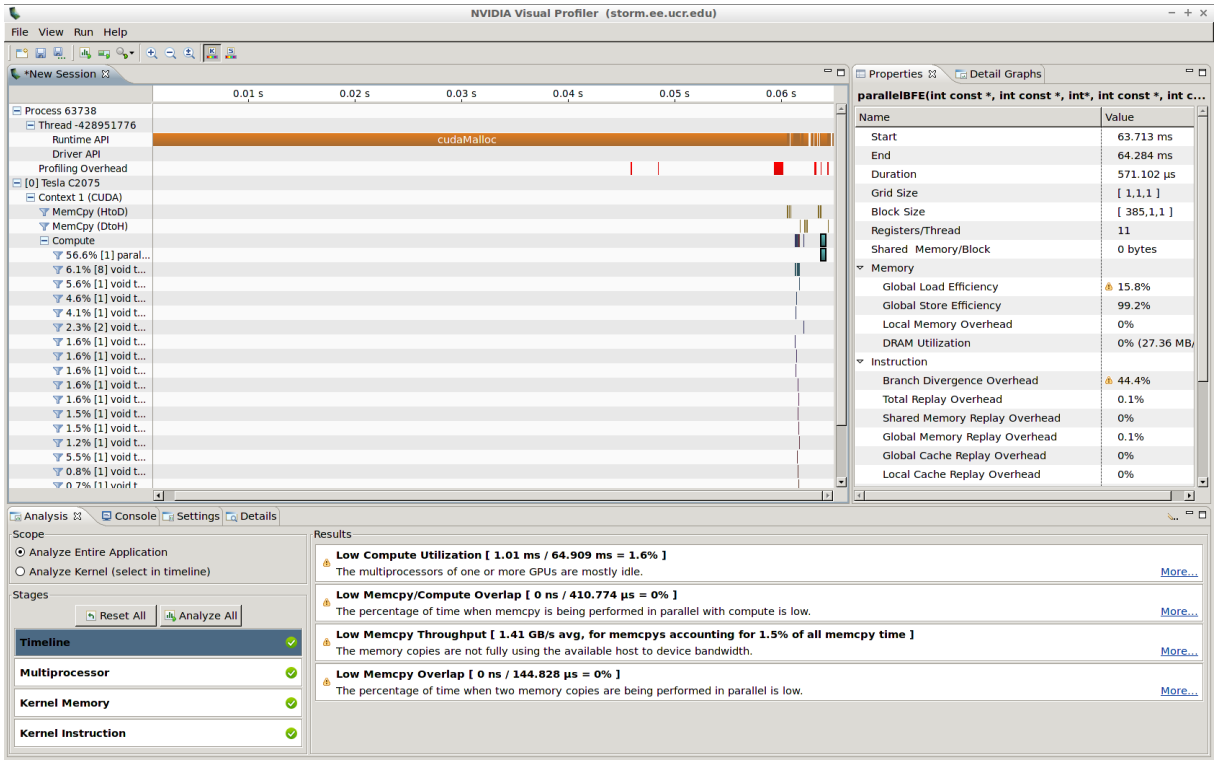



Figure 4: NVVP performance analysis for T120-E1K-M100.

Marcos R. Vieira, Petko Bakalov, and Vassilis J. Tsotras. On-line Discovery of Flock Patterns in Spatio-temporal Data. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, pages 286–295, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-649-6. doi: 10.1145/1653771.1653812. URL `http://doi.acm.org/10.1145/1653771.1653812`.

Yu Zheng and Xiaofang Zhou. *Computing with Spatial Trajectories*. Springer Science & Business Media, October 2011. ISBN 978-1-4614-1629-6.

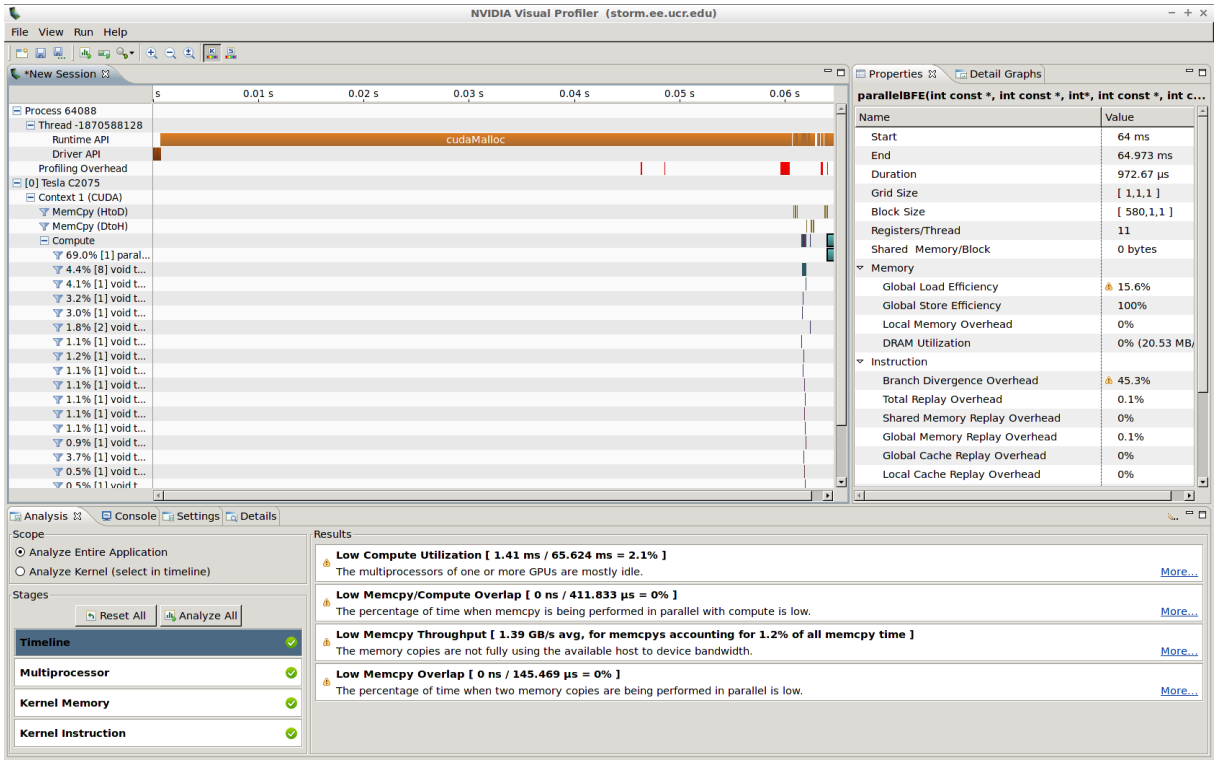



Figure 5: NVVP performance analysis for T120-E800-M100.

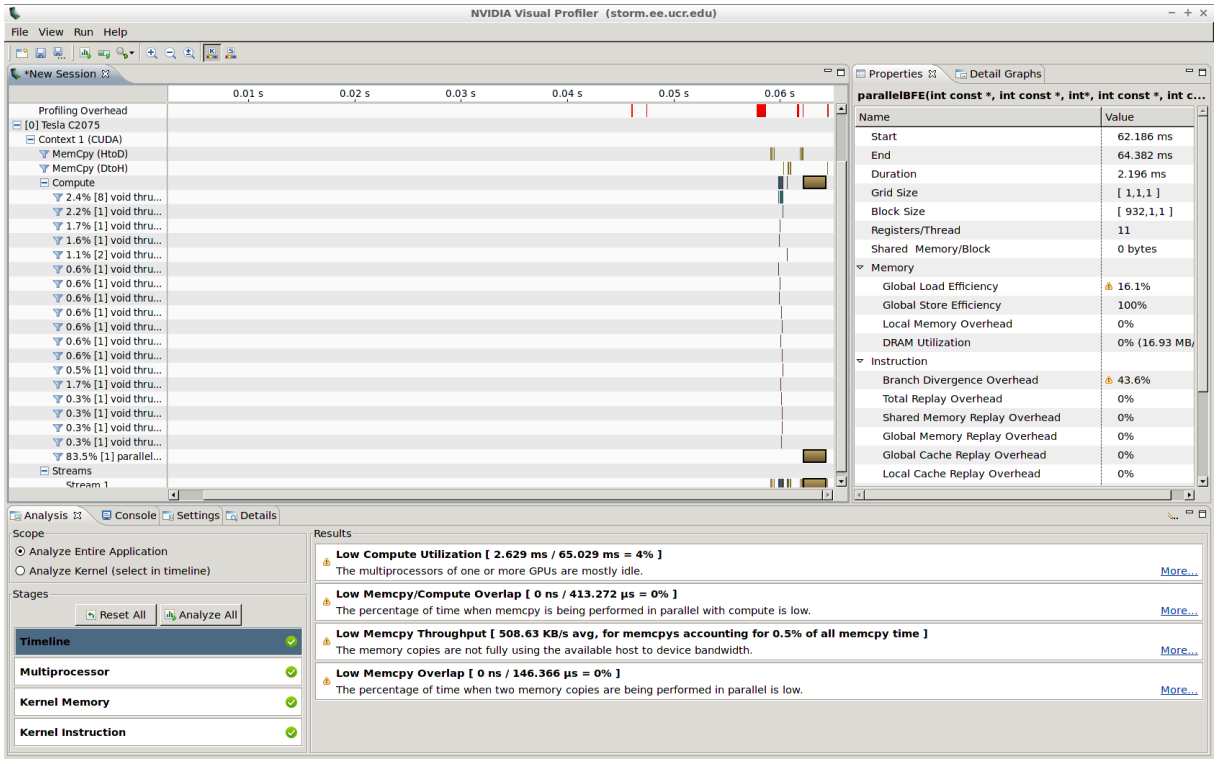



Figure 6: NVVP performance analysis for T120-E600-M100.