HPC Lab2

Konstantinos Konsoulas
Ioannis Roumpos

1 November 2022

Contents

1 Introduction

In this assignment we were given a sequential code of K-Means algorithm,we profiled it using Vtune and we used OpenMP API in order to take advantage of the multicore processors of our machines running the code over multiple threads.

2 Technical Information

Compiler and flags:
-icc -fast -qopenmp
fast is used for maximum optimised help from the compiler.
qopenmp is used for the pragma directives.

3 Implementation

3.1 Computational Loops

The block which is critical in our implementation is the red one.According to Vtune, this block requires most of the CPU time.Taking into account the pro-

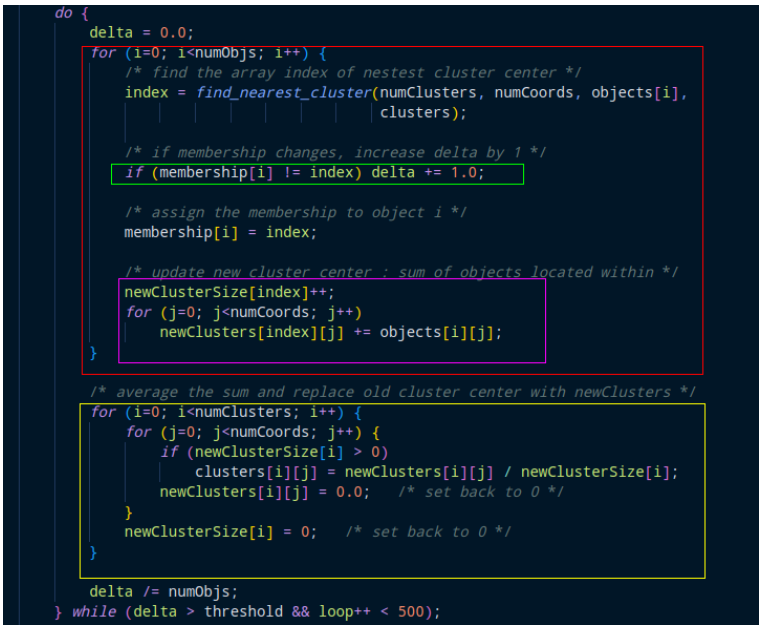

Figure 1: Blocks of sequential code

filing, we examined if there were any data dependecies in the code in order to parallelise the workflow.
Firstly, the find nearest cluster function is data dependency free so is parallelisable. The index was required to be private to each thread to make sure that the function result is not lost.
Next, we observed that the operation *delta += 1.0*,in the green block, needed to be *atomic* to avoid race conditions.Afterwards,in the magenta block,we noticed that the block could result in a data dependency as multiple threads can have the same *index* value and that lead us to make the block of code critical.
As a result,j was not required to be private and that may have led to a small overhead in the final results.Also, i variable is not necessary to be private since by default the *pragma for* directives take it into account.At the end of the red block there is a barrier implied since the *new cluster* array values are needed in the code below.
The second computation intensive loop is the yellow one.We can easily observe that there is no data dependency therefore parallelisation is applicable and the j variable is private.
After testing different scheduling policies we reached the conclusion that guided scheduling is the one with the best results.

```
108     do {
109         delta = 0.0;
110
111         #pragma omp parallel for schedule(guided) private (index,j,i)
112         for (i=0; i<numObjs; i++) {
113                 /* find the array index of nestest cluster center */
114
115             index = find_nearest_cluster(numClusters, numCoords, objects[i],
116                                          clusters);
117
118
119             /* if membership changes, increase delta by 1 */
120             if (membership[i] != index){
121                 #pragma omp atomic
122                 delta += 1.0;
123             }
124
125             /* assign the membership to object i */
126
127             membership[i] = index;
128
129             /* update new cluster center : sum of objects located within */
130             #pragma omp critical
131             {
132             newClusterSize[index]++;
133             for (j=0; j<numCoords; j++)
134                 newClusters[index][j] += objects[i][j];
135             }
136         }
137
138
139         /* average the sum and replace old cluster center with newClusters */
140         #pragma omp parallel for schedule(guided) private(j)
141         for (i=0; i<numClusters; i++) {
142             for (j=0; j<numCoords; j++) {
143                 if (newClusterSize[i] > 0)
144                     clusters[i][j] = newClusters[i][j] / newClusterSize[i];
145                 newClusters[i][j] = 0.0;   /* set back to 0 */
146             }
147             newClusterSize[i] = 0;   /* set back to 0 */
148         }
149         delta /= numObjs;
150     } while (delta > threshold && loop++ < 500);
151
```

Figure 2: Final version of the code

```
/* initialize membership[] */
#pragma omp parallel
{
#pragma omp for nowait
for (i=0; i<numObjs; i++) membership[i] = -1;

#pragma omp single
{
/* need to initialize newClusterSize and newClusters[0] to all 0 */
newClusterSize = (int*) calloc(numClusters, sizeof(int));
assert(newClusterSize != NULL);

newClusters    = (float**) malloc(numClusters *            sizeof(float*));
assert(newClusters != NULL);
newClusters[0] = (float*)  calloc(numClusters * numCoords, sizeof(float));
assert(newClusters[0] != NULL);
}
#pragma omp for
for (i=1; i<numClusters; i++)
    // newClusters[i] = newClusters[i-1] + numCoords;
    newClusters[i] = newClusters[0] + i*numCoords;
}
```

Figure 3: Initialiased Code

3.2 Initialisation Loops

We parallelised the initialisation section of the code which gave a not measurable decrease in execution time.In more detail,in the initialisation of the *membership* array we used *pragma nowait* as it is not needed in the rest of the block.We continue by using the *pragma single* directive for the memory initialisation of the matrixes since we need only one thread to execute those operations.Concluding,we wanted to include the last for loop in the parallel section but we observed that it contains a data dependency.So, we changed the code in order to be free of data dependency.The final version of the code is shown in the figure 3 above.

4 Measurements

To measure our changes in our code we experimented with two different implementations asides the original sequential.The first one is with parallel static version of the code ,without scheduling policy, and the other one is with guided scheduling policy.We run the code with different numbers of threads (1,4,8,16,32,64). The figures shown the decrease of the CPU time as we increase the number of threads with the exception in the case of 64 threads were we observe that the computational time in increased compared to the time of the case we are using 32 threads due to overhead.
Also, we need to clarify that we run the sequential code with only 1 thread and since there will be no different in CPU time with more threads.
To conlude, as we see at the figures below the parallel guided version of our code is the optimal one even if the difference between the static is small but yet measurable.

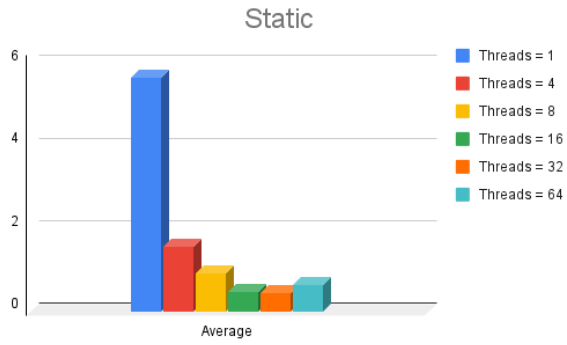

Figure 4: Parallel Static Code

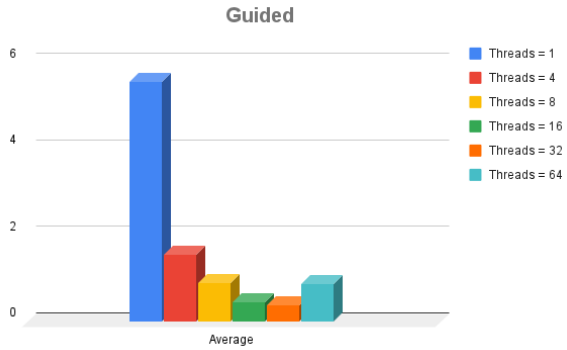

Figure 5: Parallel Guided Code

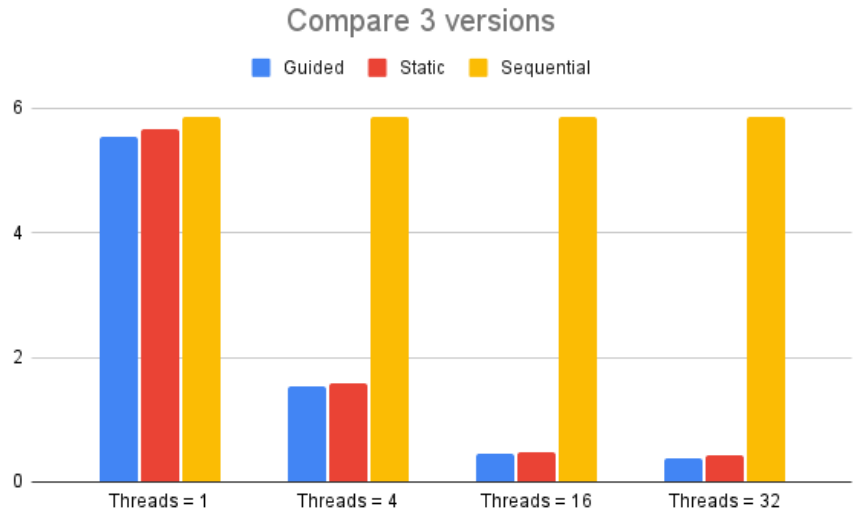

Figure 6: Final version of the code

5 Contact

Konstantinos Konsoulas(2975): kkonsoulas@uth.gr
Ioannis Roumpos (2980) : iroumpos@uth.gr