Solving common influence region queries with the GPU

Marta Fort and J.Antoni Sellarès*

Departament Informàtica, Matemàtica Aplicada i Estadística. Universitat de Girona.

Abstract

In this paper we propose and solve *common influence region queries*. We present GPU parallel algorithms, designed under CUDA architecture, for approximately solving the studied queries. We also provide and discuss experimental results showing the efficiency of our approach.

Introduction

Common influence region queries are related to the capacity of two sets of facilities of different type, competitive and collaborative, of attracting customers. Solutions to common influence region queries help decision makers to develop competitive-collaborative opportunities.

Papers [2, 3, 1] deal with a related problem which tries to maximize the area of the Voronoi region of a new non-weighted facility. Fort and Sellarès [4] present an algorithm for optimizing k-nearest/farthest influence regions of a set of non-weighted facilities.

The advancement in GPUs (Graphics Processing Units) hardware design, together with CUDA (Compute Unified Device Architecture), make them attractive to solve problems which can be treated in parallel as an alternative to CPUs. General-Purpose computing on GPUs (GPGPU) is playing an increasing role in scientific computing applications, which range from numeric computing operations to data mining or geometric processing, where the potential of the GPU for delivering real performance gains on computationally complex, large problems is demonstrated.

By working towards practical solutions, since exact algorithms to solve the common influence region queries are hard to implement and quite slow in practice, we explore a GPU parallel approach, designed under CUDA architecture, for solving the queries approximately. We also provide and discuss experimental results obtained with the implementation of our algorithms that show the efficiency and scalability of our approach.

*Email:(mfort,sellares)@imae.udg.edu.

Authors research supported by TIN2010-20590-C02-02.

1 Preliminaries

1.1 The weighted area of a region

Let $\mathcal{R} = \{R_1, \cdots, R_m\}$ be a partition of the domain D. We associate with each region R_i a non-negative number w_i, called the weight of R_i.

We define the weighted area of the region $R \subset D$, denoted $w(R)$, as:

$$w(R) = \sum_{j=1,\cdots,m} w_j \ \mu(R \cap R_j)\,,$$

where $\mu(R \cap R_j)$ denotes the area of the subregion $R \cap R_j$.

1.2 CUDA architecture

CUDA is a parallel computing architecture that makes GPUs accessible for computation like CPUs. The CUDA processors, which can be executed in parallel, are referred as threads, and each thread executes the instructions contained in the so called kernels in parallel. Each thread computation is independent from the others, however, there exist some read-modify-write atomic operations called atomic functions. They read and return the value stored in a memory position, operate on it and store the result without allowing, during the whole process, any other access to that memory position. These operations allow the users to obtain global results when several threads access to the same memory position. For instance we can obtain a global sum, maximum or minimum in a specific position by using them.

Several types of memories can be used, data stored in global memory are accessible by every thread and are visible from the CPU, global memory is where more data can be allocated, but is the slowest access time memory. Shared memory and registers are the fastest memory, shared memory can be accessed by all the threads of the same block, and registers store the local variables of the threads. The number of accesses to memory are reflected in the execution times of the algorithms. Thus they are provided in the complexity analysis, ν read or written values to global memory are represented by r^g_ν and w^g_ν, and to shared memory they are denoted by r^s_ν and w^s_ν.

2 Influence regions

Let S be a set of n points included in a bounded domain D of the Euclidean plane. Each point $s \in S$ is associated with a positive real weight $w_s > 0$. The weighted distance $d_s(q)$ from an arbitrary point $q \in D$ to the point $s \in S$ is defined as $d_s(q) = (1/w_s)\, d(s,q)$, where $d(p,q)$ denotes the Euclidean distance between points p, q.

The k-influence region of a point $s \in S$, denoted $I_k(s,S)$, is the set of points of D having s among their k-nearest points in S. For any point $q \in D$, denote by $n_k(q,S)$ the weighted distance from q to its k-th nearest point in S, i.e. the weighted distance to the point of S that ranks number k in the ordering of the points by increasing weighted distance from q. We have:

$$I_k(s,S) = \{q \in D \,|\, d_s(q) \leq n_k(q,S)\}\,.$$

From the definition follows that the 1-influence region of a site s is its 1-Voronoi region, and that $I_k(s,S) \subset I_{k+1}(s,S)$.

A k-influence region is bounded by bisectors of points in S. In general, k-influence regions need not to be convex, nor simply connected, nor even connected. Two k-influence regions may share disconnected edges (see Figures 1 a) and b)).

2.1 Common influence regions

Let P and Q be finite disjoint sets of n and m weighted points, respectively, within the bounded domain D of the Euclidean plane.

The (k,k')-Common Influence Region of $p \in P$, $q \in Q$, denoted $C_{k,k'}(p,q,P,Q)$, is the set of points of D having p among their k-nearest points in P and at the same time having q among their k'-nearest points in Q:

$$C_{k,k'}(p,q,P,Q) = I_k(p,P) \cap I_{k'}(q,Q)\,.$$

In Figure 1 c) we can see an example of a $(5,2)$-common influence region painted in green.

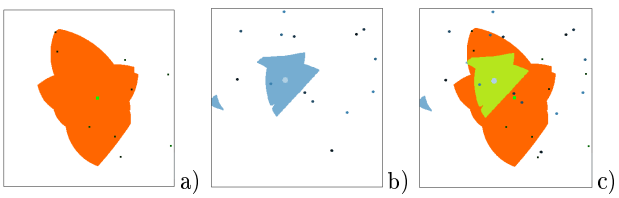


Figure 1: a) 5-influence region, b) 2-influence region, c) $(5,2)$-common influence region.

2.2 Common influence region queries

Let P and Q be finite disjoint sets of n and m weighted points, respectively, within the bounded domain D of the Euclidean plane, and $\mathcal{R}$ be a weighted partition of D. We study the following queries:

Feasible pairs query. Given fixed non-empty sets $P' \subseteq P$ and $Q' \subseteq Q$ and a real positive number W_0, find all pairs (p,q), $p \in P'$ and $q \in Q'$, such that the weighted area $w(C_{k,k'}(p,q,P,Q)) \geq W_0$.

Feasible partners query. Given fixed non-empty sets $P' \subseteq P$ and $Q' \subseteq Q$ and a real positive number W_0, find all points $p \in P'$, such that for each $q \in Q'$ the weighted area $w(C_{k,k'}(p,q,P,Q)) \geq W_0$.

The parameters k, k', W_0, I_0 should be chosen by an expert, according to the localization of p with respect the points of Q' and the cardinality of Q', during an iterative what-if analysis process.

To obtain $w(C_{k,k'}(p,q,P,Q)) = w\big(I_k(p,P) \cap I_{k'}(q,Q)\big)$, we can first compute the overlay intersection between $I_k(p,P)$, $I_{k'}(q,Q)$ and the regions of the weighted partition $\mathcal{R}$. Since each maximal connected region of the resulting intersection is contained in a region of $\mathcal{R}$, it has univocally associated a weight. Consequently, the weighted area $w(C_{k,k'}(p,q,P,Q))$ can be written as a sum of functions that depends on the location of p and q. Since computing the maximal connected regions together with their weighted area is difficult, it is also difficult computing the weighted area of a common influence region and solving common influence region queries. This has motivated us to explore an alternative GPU parallel approach, designed under CUDA architecture, for approximately solving common influence region queries.

3 Solving common influence region queries using CUDA

The weighted areas of the common influence regions are estimated by using a discretization of the domain D. An axis-parallel rectangular grid of size $H \times W$ is superimposed on the domain D defining a set S of r points corresponding to the geometric center of the grid cells which are weighted according to the domain partition $\mathcal{P}$. As it is obvious, the bigger the size of the used grid the more accurate the obtained common influence region. The weighted area is estimated by summing up the weights of all the points of S contained in the common influence region. Notice that, depending on the domain weighted partition, small changes in the common influence region can produce big changes in its weighted area.

Solving the defined queries requires an initial step consisting in computing the k-nearest neighbor distance from each point of S to the points in P and the k' nearest neighbor to the points in Q. Notice that even though the queries subsets P' and Q' are used,

the k and k'-neighbor distances are always referred to P and Q.

3.1 k-nearest neighbor distance computation

We start by computing the weighted distance of each point $s \in S$ to its k-th nearest neighbor in P, considering all the points in parallel. With this aim, we have extended the CudaKNN algorithm described by Liang et al. [5] to handle the weighted case. The algorithm computes for each $p \in P$ the weighted distances to all the points in S using shared memory and making the threads in a block cooperate to determine, after exploring all the point in P, the k-nearest neighbors of s. We transfer the points of S and P from the CPU to the GPU, then we compute the k-weighted nearest distance of each point in S to P and store all them in global memory in a 1D array n_{P_k} of size r.

In the same way, we also transfer the points of S and Q from the CPU to the GPU, compute the k'-weighted nearest distance of each point in S to Q and store them in a 1D array $n_{Q_{k'}}$ of size r.

3.2 Influence region queries resolution

Given $P' \subseteq P$ and $Q' \subseteq Q$ of size n' and m' respectively, and assuming that the k and k'-nearest neighbor distances from each point of S to P and Q have been computed, we explain how the proposed queries can be solved.

Our approach for solving the feasible pairs and the possible partners queries follow a very similar procedure.

Feasible pairs query

The solution of the feasible pair query is reported giving the number of feasible pairs and the list containing them. In order to solve it we consider $n'm'$ threads, an integer to store the number of feasible pairs and an 1D-array of size $n'm'$ to store the feasible pairs. Each thread estimates the weighted area $w_{p,q} = w(C_{k,k'}(p,q,P,Q))$ of one pair (p,q). In the case that $w_{p,q}$ gets the minimum required value W_0, the number of feasible pairs is incremented by one, and the pair is stored in the first empty position of the feasible pairs array.

The weighted area $w_{p,q}$ is estimated by considering all the points $s \in S$, computing the distances $d_p(s)$ and $d_q(s)$ and comparing them with $n_k(s,P)$ and $n_{k'}(s,Q)$ which are the ones stored in n_{P_k} and $n_{Q_{k'}}$. Thus $s \in C_{k,k'}(p,q,P,Q)$ whenever $d_p(s) \leq n_{P_k}(s)$ and $d_q(s) \leq n_{Q_{k'}}(s)$. In such a scenario, where all threads check all the points in S, shared memory should be used. The B threads in a block cooperate loading the space points and their neighbor distances from global to shared memory. This requires using three shared memory arrays of size B per block: S^s which stores B weighted space points, and $n^s_{P_k}$, and $n^s_{Q_{k'}}$ storing their corresponding k and k'-neighbor distances to P and Q, respectively. The i-th thread of the block reads from global memory the point $s_i \in S$ and its corresponding k and k' nearest neighbor distances, and stores them in the i-th position of S^s and $n^s_{P_k}$ and $n^s_{Q_{k'}}$, respectively. When all the threads in the block have finished loading this information, the first B points of S can be analyzed. Then each thread determines which of these points are contained in its corresponding common region $C_{k,k'}(p,q,P,Q)$ and accordingly accumulates their weights in a register. Once all the threads of the block have checked all the already loaded points, the next B points of S and their distances are loaded to shared memory and analyzed. We keep on proceeding similarly until all the points in S have been handled, and consequently w_{pq} has been estimated.

In order to obtain correct results it is important that the threads in a block wait each other in some critical points. In fact, two synchronization points are needed, one after transferring the corresponding weighted point of S with its two associated distances and the other when all the already stored points have been checked, just before starting transferring new information to shared memory. Otherwise there is no guarantee that all the threads in the block have finished with their tasks. The implementation has to be carefully done when r, the number of points in S, is not a multiple of B, the number of threads in a block, to avoid accesses to non-existent memory positions, but guaranteing that all the points of S are loaded to shared memory.

Feasible partners query

To solve this query we use an array f_p, of size n', which is used as a boolean array. At the end of the process, positions containing a 1 correspond to the points of P' having all the points of Q' as feasible partners.

After initializing the array f_p with ones, we proceed as before estimating the weighted common influence areas of all the $m'n'$ pairs in $P' \times Q'$. When a thread finishes estimating its corresponding weight w_{pq}, it checks wether $w_{pq} < W_0$, and in such a case, it sets $f_p[p]$ to 0. Thus, only for those points p achieving the minimum influence value for all the points $q \in Q'$ we will have $f_p[p] = 1$ at the end of the process.

The number, n'_f, and the list, f, of feasible partners can be obtained by performing a prefix sum on the array f_p, which is stored in p_s. After allocating an array of size n'_f, n' threads are considered, each thread checks wether its point of P' is a feasible partner of Q', by looking at its corresponding position of f_p. If it is so, the point of P' is stored in f in the position given by p_s.

3.3 Complexity analysis

Computing simultaneously for the r points of S the k-neighbor distances with respect to the set of n points by using the CudaKNN algorithm [5] requires $O(rnB)$ total work, where B is the number of threads per block. The computation is done in three different steps, that have $O(r)$, $O(B)$ and $O(k + n/B)$ operations per thread, respectively. Thus, the initial step in our case requires $O(r(n + m)B)$ total work.

Solving the feasible pairs query requires $n'm'$ threads performing $O(r)$ operations each, leading to $O(rn'm')$ total work. Concerning memory accesses, in the worst case $O(r^g_{n'm'r/B}+w^s_{n'm'r/B}+r^s_{n'm'r}+w^g_{n'm'})$ accesses are required. Finally, the global memory needed is $3r + n' + m' + n'm' + 1$, while $3B$ floats per block are required in shared memory.

The differences between solving the feasible pairs query and the feasible partners one, before finding the list of feasible partners, do not change the work per thread neither the global work nor the memory accesses. However, the global memory requirements are reduced to $3r + 2n' + m'$. Finding the list f only increases the global memory requirements in $n' + n'_f$.

4 Experimental Results

In this section, we present several experimental results obtained with the implementation of our algorithms. In all the experiments the points of P and Q are randomly distributed in the domain and their weights are randomly obtained integers between 1 and 15. They are based on the weighted domain partition presented in Figure 2. The points of the domain are painted in a purple color gradation, the darker the color the smaller the density value.

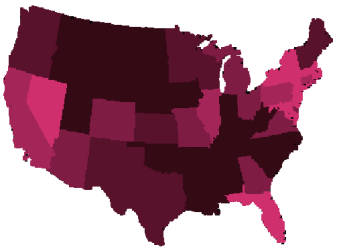
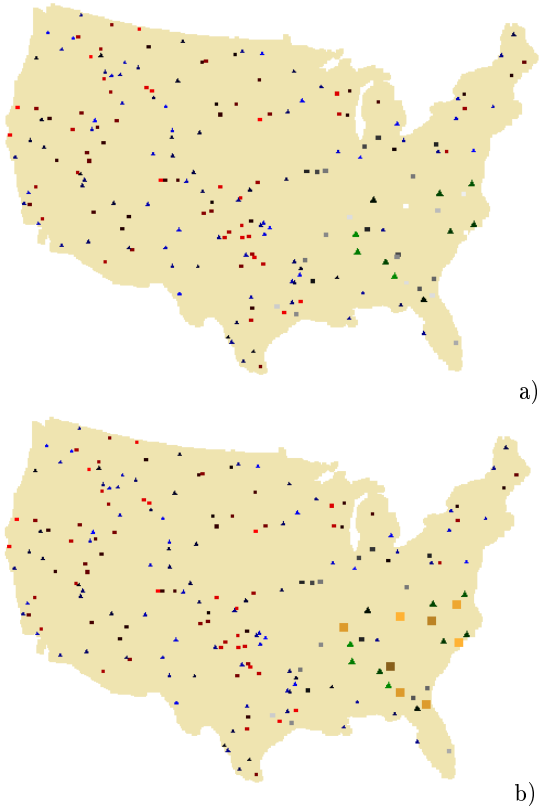


Figure 2: Domain weighted partition according to the population density.

In Figure 3, we provide the solution of a feasible partners query. Figure 3 a) shows the sets P, Q, P' and Q', that have 100, 100, 25 and 10 points, respectively. Points of sets P' and Q' are represented by grey squares and green triangles, and the points of P and Q not in P' and Q', by red squares and blue triangles, respectively. The orange squares of Figure 3 b) are the points of P' conforming the solution of the feasible partners query defined by P, Q, P', Q' for $k = k' = 15$, when the minimum required influence value is 20.



a)

b)

Figure 3: a) Points of the sets P, Q, P', Q'. b) Solutions of the feasible partners query.

For a grid of size 400×400, $n = 500$, $n' = 50$, $m = 200$, $m' = 25$, $k = 20$ and $k' = 20$, the running times needed to solve a feasible pairs query and a feasible partners query are 12 and 16 milliseconds. Using the CPU sequential version of the algorithms the times become 347 and 351 miliseconds, respectively. In the CPU version, the feasible partners query can be solved in 62 miliseconds if point $p \in P'$ is set as a not feasible partner at the first $q \in Q'$ for which $w(C_{k,k'}(p, q, P, Q)) < W_0$. This can not be done in the GPU, because threads cooperate transferring information to shared memory.

References

[1] O. Cheong, A. Efrat, and S. Har-Peled, *Finding a guard that sees most and a shop that sells most*, Discrete Comput. Geom. **37(4)** (2007), 545–563.

[2] M. Denny, *Solving geometric optimization problems using graphics hardware*, Comput. Graph. Forum **22** (2003), no. 3, 441–452.

[3] F. K. H. A. Dehne, R. Klein, and R. Seidel, *Maximizing a Voronoi region: the convex case*, Int. J. Comput. Geometry Appl. 15(5) (2005), 463–476.

[4] M. Fort and J.A. Sellarès, *GPU-Based Influence Regions Optimization*, ICCSA (2012), 253–266.

[5] S. Liang, Y. Liu, C. Wang, L. Jian, *A CUDA-based parallel implementation of k-nearest neighbor algorithm*, IEEE'09 (2009), 291–296.