Spatial data Parallel Processing on GPGPU

Driss En-Nejjary

École doctorale : Sciences pour l'ingénieur

THÈSE

Pour obtenir le grade de :

Docteur de l'Université Clermont Auvergne

Spécialité Doctorale en Informatique

Présentée et soutenue publiquement par

Driss EN-NEJJARY

Le 10 juin 2021

Des approches séquentielles et parallèles pour L'amélioration des performances des sélections et Des agrégations des données rasters

Membres du jury:

M. François Pinet	DR	INRAE	Directeur de thèse
Mme. Myoung-Ah KANG	MCF	Univ. Clermont Auvergne	Co-directeur de thèse
M. Alain Bouju	MCF- HDR	L3i, Univ. La Rochelle	Rapporteur
M. Sofian Maabout	MCF- HDR	LaBRI, Univ. Bordeaux	Rapporteur
M. Olivier Teste	Professeur	IRIT, Univ. Toulouse 2	Examinateur
M. Jean-Denis Mathias	DR	INRAE	Examinateur

Clermont Auvergne University

A thesis submitted in fulfillment of the requirements for the

Degree of Docteur de l'Université Clermont Auvergne

Discipline: Computer Science

Author: **EN-NEJJARY Driss**

Defended on: 10th June 2021

Spatial data Parallel Processing on GPGPU

Composition of the jury:

M. François Pinet	DR	INRAE	Thesis Director
Mrs. Myoung-Ah KANG	MCF	Clermont Auvergne University	Thesis Co-Director
M. Alain Bouju	MCF- HDR	L3i, La Rochelle University	Reviewer
M. Sofian Maabout	MCF- HDR	LaBRI, Bordeaux University	Reviewer
M. Olivier Teste	Professor	IRIT, Toulouse 2 University	Examiner
M. Jean-Denis Mathias	DR	INRAE	Examiner

Abstract

With the emergence and the production of a large volume of spatial data, supporting large scale and high-performance queries and analysis has become crucial and essential in several applications and fields. The tremendous advances in technology such as smartphones, internet of things, web, navigation systems and sensors, have led to the production of spatial datasets having large sizes. For example, climate and precision agriculture sector are ones of the fields affected by these advances in data acquisition technology where this kind of data is produced in high precision and large temporal sequences. Querying large-scale data allows extracting more valuable and meaningful information that is vital for decision-making, scientific advancement and scenario predictions. Unfortunately, most of existing methods and approaches are based on traditional computing framework (uniprocessors) which makes them not scalable and not adequate to deal with large-scale data.

In this work, we show that using the GPGPU can reduce the time of spatial data processing and save computations. In this regard, we have proposed to speed up three classical queries that have never been tackled before in the literature. First, we have proposed an optimized parallel method based on GPGPU to produce overlapping aggregated data summaries by the computation of the average temperature for all overlapped raster subsequences of a determined length for the studied region.

As a second contribution, we have tackled a raster selection query based on a threshold fixed by the user. In fact, in different analyses, users can be interested only in some rasters. Hence, we have implemented two solutions based on the GPGPU and the CPU that include a rejection procedure of rasters in the early stages of computations using on a sorting step.

Finally, we have proposed two high-performance methods for a selection query based on GPGPU and CPU for massive spatio-temporal data. The query consists on selecting fixed size disjoint raster subsequences based on their average satisfying a user threshold condition. The two methods include a rejection procedure of subsequences based on sorting.

Keywords: Spatial data science; Geographic Information System; Big data; General Purpose GPU; Parallel computing; Raster data; CUDA; CUB; Thrust.

Résumé

Ces dernières années, la production des données spatiales a connu un bond qualitatif et quantitative. En effet, Les énormes progrès technologiques, tels que les smartphones, l'Internet des objets, les systèmes de navigation et les capteurs ont conduit à la production des données spatiales de grande taille et à haute définition. Les capteurs, par exemple, sont maintenant plus précis, moins chers et plus performants générant des données haute précision et à grande fréquence. Proposer des requêtes et des opérations d'analyse puissantes et à grande échelle devient cruciale et essentielle dans plusieurs applications et domaines. Le secteur d'agriculture et environnemental est l'un des domaines touchés par ces progrès de la technologie d'acquisition de données. Interroger et analyser ce jeu de données permet d'extraire des informations vitales pour la prédiction, la prise de décision, et l'avancement scientifique. Malheureusement, la plupart des méthodes et approches existantes sont basées sur une approche traditionnelle à base de CPU (monoprocesseurs), ce qui les rend non évolutives, inadéquates pour traiter des données à grande échelle et prennent beaucoup de temps pour l'exécution.

Dans ce travail, nous montrons que l'utilisation du GPGPU peut réduire le temps de traitement des données spatiales et économiser les calculs. À cet égard, nous avons proposé d'accélérer une opération d'agrégation et deux requêtes classiques qui n'avaient jamais été abordées auparavant dans la littérature.

Tout d'abord, nous avons proposé une méthode parallèle optimisée en utilisant GPGPU pour produire des résumés de données basée sur l'agrégation chevauchées, plus précisément,

Le calcul de la température moyenne des séquences chevauchées des rasters de taille fixe.

Dans un second temps, nous avons abordé la requête de sélection des rasters basée sur un seuil fixé par l'utilisateur. En effet, dans différents scénarios, les utilisateurs ne peuvent s'intéresser qu'à certains rasters. Par conséquent, nous avons mis en place deux solutions robustes basées sur le GPGPU et le CPU qui incluent une procédure de rejet des rasters aussitôt que possible pour réduire le temps d'exécution en utilisant une étape de trie.

Finalement, nous avons proposé deux méthodes basées sur le GPGPU et le CPU pour une requête de sélection des séquences des rasters non chevauchés à base de leurs moyennes selon un seuil fixé par l'utilisateur.

Mots-clef: Calcule Parallèle; Données spatiales;Big data ; spatial data science; GPGPU; Système d’information Géographique, données rasters; CUDA; CUB; Thrust.

Acknowledgements

There are many people I would like to thank for helping me during my PhD.

First, I have to thank my supervisors: François Pinet research director (INRAE, TSCF) and the Prof. Myoung-Ah Kang (ISIMA, LIMOS). Through all of our exchanges, they taught me a lot about the problematic and their ideas made a big part of this work. They also gave me so many wonderful opportunities like the chance of visiting another lab abroad, being part of a summer school in Spain and attending conferences. they were not only advisers but real friends who always supported me and helped me out not only with my work but also at personal level. They were patient and encourage me a lot during my difficult situations especially the hard one where I was blocked in Morocco for over than 5 months because of the Covid19. Without their support and patience, I would not have finished my thesis especially with my difficult situation during these last months.

Regarding my work they improved the quality of our papers hugely and their suggestions had always a big impact. They would even take their personal time working on weekends to improve our work. I am very thankful to them because they believed in me, they gave me the opportunity to teach and supervise students' projects at ISIMA and they were always by my side.

I would like to express my sincere appreciation and gratitude to the Prof. Alain Bouju (Rochelle university, L3i) and Prof. Sofian Maabout (Bordeaux university, LaBRI) who honored me and agreed to review this manuscript. I also would like to thank Prof. Olivier Teste (Toulouse 2 university, IRIT) and the research director M. Jean-Denis Mathias (INRAE, LISC) for accepting to be part of my jury, for their valuable time and feedback.

Finally, I would like to thank my beautiful wife Sara NAKHLI, she was always there during this journey. I will be eternally grateful. She has always believed in me, a lot more than I do myself. This would not have been possible without her love and support.

This work was funded by grants from the French program « investissement d'avenir » managed by the Agence Nationale de la Recherche of the French government (ANR), the European Commission (Auvergne FEDER funds) and « Région Auvergne » under the LabEx IMobS 3 (ANR-10- LABX-16-01).

Contents

List of Figures

List of Tables

Part 1

This first part is organized in 3 chapters:

- Chapter 1: General introduction. In which we present the context and the motivation of our work, we outline the contributions and we present our methodology of research.
- Chapter 2: State of the art. In this chapter, we show the different methods used to process large-scale spatial data. We have classified them into three main categories: methods based on uniprocessor CPU, methods based on parallel processing using the GPGPU and finally the methods based on distributed systems.
- Chapter 3: Background. This chapter is dedicated to present the different concepts related to: Spatial data and their operations and parallel computation.

Chapter 1

Introduction

1.1 Context and motivation

The emergence and the tremendous advances in technology such as smartphones, internet of things (Palmaccio et al., 2020), networking capabilities unmanned technologies, navigation systems and sensors, have led to the production of large size of dataset especially the environmental data. Sensors, for instance, are now smaller, cheaper and even smart (Melesse et al., 2007). Sensors now are portable; they can be mounted on various platforms, such as microstates, drones, airships, vehicles, water-based vessels, and may even be carried or embedded in robots (Tao et al., 2007). In the other hand, Wireless Sensor Network (WSN) which is an important element in Internet of Things (IoT), allows monitoring environmental conditions, such as temperature, pressure, or humidity using sensors cooperatively (Madakam et al., 2015). These actual advances cited above, have led to an explosive volume of data more precisely spatial data. Hence, Spatial data are produced in a high precision, high wide coverage, and in a high temporal frequency (i.e., at each second) (Sawant et al., 2017) (Pinet, 2012). These spatial data are deployed for many applications, and received remarkable attention in many areas such as military, homeland security, healthcare environmental monitoring, precision agriculture and so on (Madakam et al. ,2015). This massive amount of spatio-temporal data can be used for addressing scientific challenges: such as climate changes and global warming etc. (Yang et al., 2019). Analyzing, this large set of data allows us to understand environmental phenomena better, make predictions that are more accurate, enhance surveillance and proactive decision-making in many applications.

The fields that interest us are the agriculture and the environment. Hence, transforming this large volume of data into actionable knowledge for better decision support is crucial and very challenging task for industry and research.

In the agriculture domain, especially the precision agriculture or the precision farming, geospatial data is collected, analyzed to maximize on yields (Grisso et al., 2004). This allows the farmer to gain more understanding on resources' optimization such as fertilizers, pesticides and herbicides, water in order to use them more efficiently. This will reduce extra expenses, increase productivity and as results maximize the profits (Ait Issad et al., 2019).

These data can also be utilized to analyze the links between different agricultural activities (livestock, crops, etc.) and the climate changes (Hamere, 2015), at a large spatial and temporal scale.

Many of these data take the form of raster sets. A raster is a geo-referenced 2-dimensional array in which each cell is associated with a value (Figure 1.1). The cells of a raster can be represented by pixels where the colors correspond to different values of a measure, such as temperature, vegetation density, CO2 measurements, etc. (Kang et al., 2015). Data availability and data storage are often no longer barriers, whereas the real bottleneck is, in many cases, the analysis of these spatial data that continue growing dramatically (Barbian and Assunção, 2017).

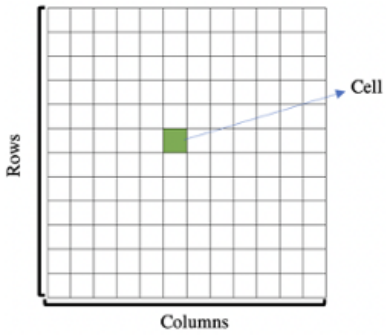

Figure 1.1. Raster is composed of rows and columns of cells.

In this work, we are interested in spatio-temporal rasters (Song et al., 2016) that can be viewed as a sequence of rasters for the same region and for a defined period of time. Each raster represents information related to the studied region at regular intervals of time (e.g. every second, minute, hour, etc.) – see Figure 1.2. Spatio-temporal rasters allow the analysis of the gradual evolution of temporal phenomena such as the detection of abnormal phenomenon evolution over time in the studied region.

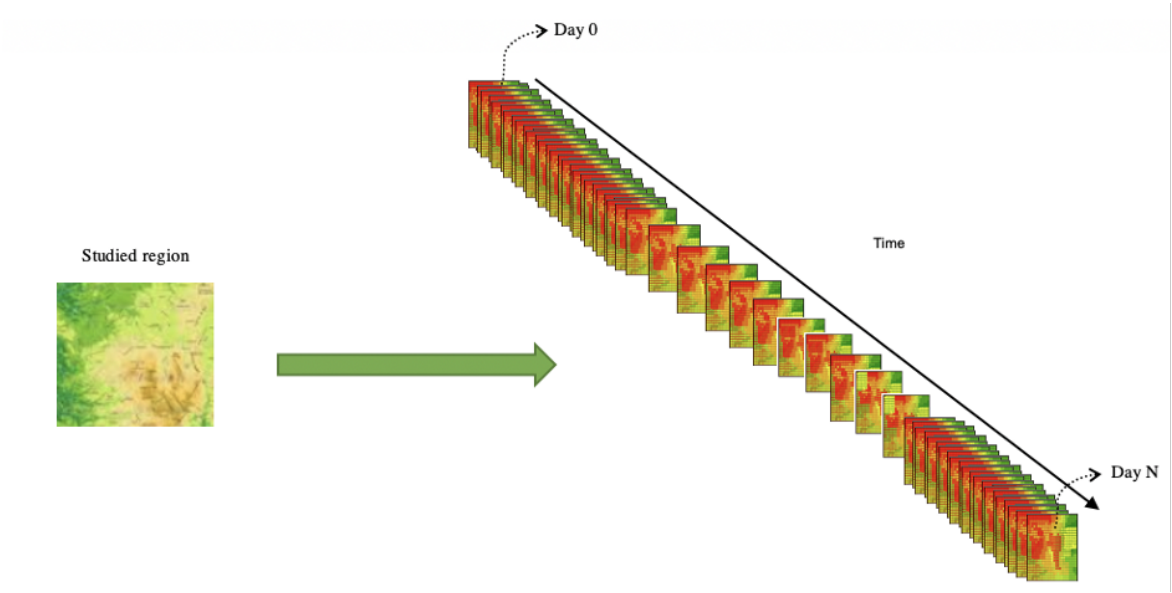

Figure 1.2. Spatio-temporal rasters representing the evolution of the temperature during N days for a studied region.

Large spatio-temporal rasters are deployed in many applications such as climate science to analyze atmospheric and oceanic conditions, which allow us to better understand Earth's system. These data are also useful in precision agriculture (Cisternas et al., 2020) to study the different factors affecting the crop yields in order to optimize the production cycle. Simultaneously, it also brings great challenges in management technology. In last decades, a large number of new approaches, parallel algorithms, processing tools, platforms, have been proposed and developed to improve the Deployment of these data, in order to extract the maximum knowledge that can help to get more accurate decisions and predictions.

The aim of our PhD, is to propose new methods based on sequential and parallel approaches. Our parallel approach is based on general-purpose processing on graphics processing units (GPGPU) (Harris, 2006), to process and analyze this large volume of spatio-temporal rasters efficiently in time and computation. Our purpose is to provide fast, efficient and scalable techniques for processing different massive computation queries over large-scale spatio-temporal data.

The majority of the existing methods are based on traditional approaches (CPU uniprocessors) since only small datasets were used; hence, optimization was not a priority. However, sequential methods are not anymore suitable for large datasets, which are growing exponentially. Hence

processing based on the GPGPU will be a good alternative to deal with the large-scale raster data. In fact, the GPGPU has been used in many applications such as image processing (Viola et al., 2003; Temizel et al., 2011), simulations (Walsh et al., 2009), and have shown good a performance of speed up. Thus, the interest of using such approach in our work.

Let D be a temporal sequence of rasters representing the temperature of the same geographical region, over a period of time, such that $D = (R_1, \ldots, R_n)$. Each raster R_i corresponds to a time i and each raster cell is associated with a temperature measurement value. We want to reduce the time execution of selection and aggregation queries over large raster data. Selection and aggregation operations are traditional queries useful in databases and data warehouse in our experiment; we use environmental and agricultural applications. However, it is possible to apply our approaches on other fields or data types.

1.2 Contributions

In our PhD, we tackle the problem of speeding up the processing of large spatio-temporal rasters for a studied region. As cited above, the majority of existing methods rely on classical approaches that are not suitable anymore, since we are dealing with large-scale spatio-temporal data. Our contributions based on GPGPU reduce query execution time and save computation time for the users in order to allow him/her to understand, react and make decisions quickly.

Let's consider a sequence of rasters representing the evolution of temperature over time for a specific region as shown in Figure 1.2. At each step of time (i.e. each hour), one raster is produced, and each raster cell stores a temperature value. Several contributions have been proposed:

- Our first contribution consists on proposing improved parallel methods based on GPGPU to produce overlapping aggregated data summaries by the calculation of the average values (e.g. temperature) for the studied region for all the possible overlapped raster subsequences of a determined length. The results show that our method can get a speed up of 60 compared to the classical approaches. This contribution has been the subject of a journal publication (EN-NEJJARY, 2019). The method is based on two fundamental steps: Computing the mean of each raster cells and after that computing the mean over each overlapped raster subsequences of order L. Such processing will give us the possibility to produce data summary as used in data warehouse, select specific data according to their average values or to

check data consistency with integrity constraints (for example, finding the periods where the average temperature is aberrant) (EN-NEJJARY, 2019).

- The second contribution has been the subject of the paper (EN-NEJJARY, 2018a). In this work, we have tackled raster selection queries based on a threshold fixed by the user. In fact, in different analyses, users can be interested only in some rasters (for example, days where the temperature were less than 15 C°). In that case, it is possible to reduce the processing time by implementing a rejection procedure of rasters based on the user's threshold in the early stages of the computation. To this end, we have added a sorting step to reject rasters that not satisfying the condition in an early time. We have proposed two methods based on the CPU, the first one is straightforward approach and the second one is based on a sorting step.
- In our third contribution, we have proposed the parallel version of the algorithm presented in the previous contribution. This work was materialized into a conference publication (EN-NEJJARY, 2018b).
- Our last contribution consists in a GPGPU-based method to implement the selection of spatio-temporal raster subsequences (a sequence of rasters for the same region and for a defined period of time) from a large spatio-temporal raster set, based on a user-defined condition (e.g., the average of raster cells must be less than a certain threshold. The results show that GPGPU-based methods reduce the execution time and enable us to get the query response 3 times faster than the traditional methods. Moreover, we propose a parallel sorting step using GPGPU to boost the response time of the query.

1.3 Methodology and publications

In our research, we have followed the hereunder steps:

- Providing a state of the art of spatial data and different techniques of processing. In this step we have studied spatial data, the map algebra (Tomlin, 1994), and the different proposed approaches to process large-scale raster data.
- Learning about the GPGPU programming and different approaches of optimization. In this context, I have participated in two summer schools dedicated to GPGPU parallel computing and different approaches of optimization.

- Studying different queries that have never been tackled before and which are very interesting for environmental and agriculture applications.
- Generating spatio-temporal raster datasets based on raw public dataset provided by the US National Oceanic and Atmospheric Administration: NOAA (Diamond et al., 2013).
- Formalizing the queries, proposing sequential and parallel resolutions, and comparing them. Making experiments and discussing the results.

The work performed during the PhD has leaded to the following publications:

1. EN-NEJJARY, D., PINET, F., KANG, M. -2019. Modeling and Computing Overlapping Aggregation of Large Data Sequences in Geographic Information Systems. International Journal of Information System Modeling and Design, vol.10(1), IGI Global USA, p. 20-41.

2. EN-NEJJARY, D., PINET, F., KANG, M. -2018. A Method to Improve the Performance of Raster Selection Based on a User-Defined Condition: An Example of Application for Agri-environmental Data. Advances in Intelligent Systems and Computing 893, 190-201., Springer.

3. EN-NEJJARY, D., PINET, F., KANG, M. -2018. Large-scale geo-spatial raster selection method based on a User-defined condition using GPGPU. 11th International Conference on Computer Science and Information Technology, Paris, France, 8 p.

4. Research poster for « 6 ème journée mobilité innovante - Robotique coopérative pour la transitique » in Aubiere, France.

1.4 Manuscript organization

This manuscript is divided into four main parts:

- The first part is organized in 3 chapters:
    - General introduction: in which we present the context and the motivation of our work, we outline the contributions and we present our methodology of research.
    - State of the art: In this chapter, we show the different methods used to process large-scale spatial data. We have classified them into three main categories: methods based on uniprocessor CPU, methods based on parallel processing using the GPGPU and finally the methods based on distributed systems.
    - Background: This chapter is dedicated to present the different concepts related to:
        - Spatial data, especially raster data.
        - Operations on raster data called map algebra
        - Parallel computing and GPU architecture
        - GPU-accelerated Libraries for Computing

Our aim is to make the readers comfortable while reading our manuscript and provide all the required knowledge to read and understand our methods without any difficulty.

- The second part concerns our contributions. It is organized as follow:
    - Chapter 4: This chapter concerns our first contribution related to the computing of overlapping aggregation of large raster sequences.
    - Chapter 5: This chapter provides a sequential method to improve the performance of raster selection based on a user-defined condition".
    - Chapter 6: This chapter proposes a GPGPU approach for the raster selection based on a user-defined
    - Chapter 7: It applies our user-defined selection approach on raster sequences (using GPGPU).

- The third part concerns the two last chapters 8 and 9:
    - Chapter 8: The application of our methods on real data of INRAE Montoldre. We present the data set of Montoldre, the acquisition, the type of data and the data characteristics.

- Chapter 9: concerns the Conclusion and perspectives. We provide a summary of our work, give outlines of our proposals and contributions, and finally suggest new research trends to extend our work.

Chapter 2

State of the Art of efficient raster processing

2.1 Introduction

Given its practical importance and the pressing need to process large-scale spatial data, several research works in the literature, have been proposed to speed up and optimize raster set processing. The proposed works tried to process more data with less computation time, which is highly required for aid decision support. Some works have tried to save computations and time processing by proposing new optimized algorithms based on the classical sequential CPU. The second category of works is based on new parallel architectures, for instance Graphics Processing Unit (GPU). The last one is based on distributed systems which are designed specifically for Big spatial data. In this chapter, we will present these methods, discussing some of the advantages and issues that arise.

2.2 CPU based methods

Driven by the idea of proposing a method that do not require implementing new functionalities or modifying existing ones of database management systems, the work presented in (Kang, 2015) proposes a method to improve query execution time to aggregate raster data stored in data warehouses (DW), by estimating the result instead of computing the exact result of the aggregation. The idea is about reducing the number of rasters required for the aggregation (using sum or average functions) and hence reducing the execution time of the query. This method includes a preprocessing step that groups the rasters, by similarity, into clusters. The similarity function depends on the nature of the data. Hence, each cluster contains rasters that are similar and each raster belongs to only one cluster. The proposed logical DW schema template for the raster can be seen in Figure 2.1

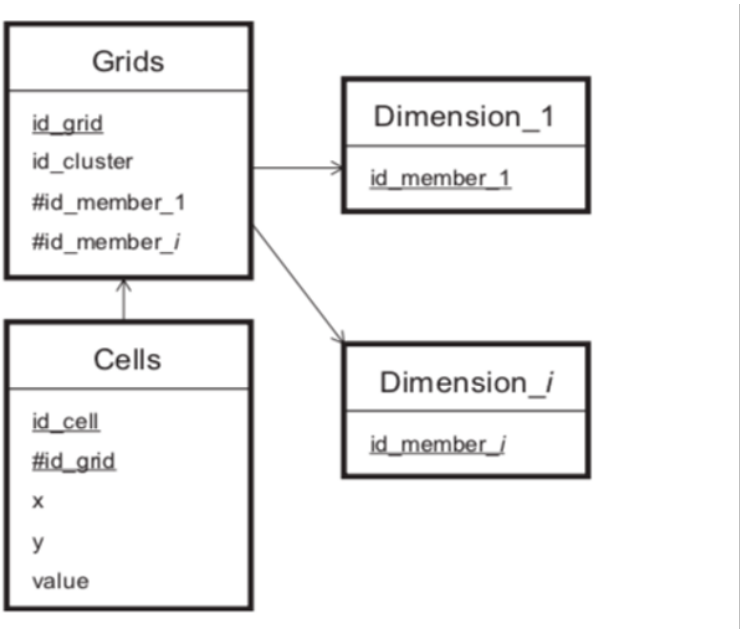

Figure 2.1. Logical DW schema template for grid storage (Kang et al., 2015).

The second step consists on querying the data by proposing to use another query model that calculates an estimation of the final result based on the initial query that compute the exact results. The new query model uses the clustering to reduce the number of arithmetic operations hence reducing the computation time. The rasters in the clusters are considered as equivalent. So, instead of taking all grids for the computation, only one grid by cluster is used. For example, to aggregate the 8 grids of Figure 2.2, only 3 grids are used. In the calculation, each one of these 3 grids is pondered by the size of its cluster. This technique used with large clusters allows an increase in time performance. Nevertheless, the more the cluster size is large, the more the quality of the result is poor.

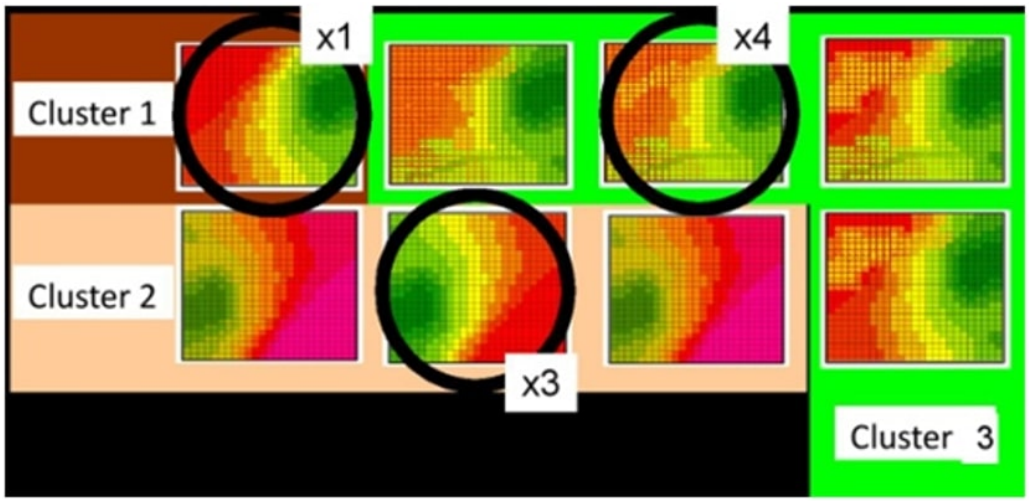

Figure 2.2. An example of aggregation of 8 grids.

So, the main advantages of this method are:

- Reducing the number of arithmetic operations
- Not requiring any modifications of data or the implementations of new functions in a database management system.

However, the fact that the results are not exact may have a drastic impact in some critical applications that need exact results. A preprocessing step is also needed to group the rasters into clusters.

Another method was proposed in (Baumann,2008) to speed up the computation of aggregation queries based on classical sequential CPU. The idea relays on the use of a pre-aggregated data step. Hence, the authors propose to use this step for aggregating query processing in multidimensional raster image databases. Furthermore, the authors propose a cost model to evaluate the efficiency of the pre-aggregated data step on the whole raster data analysis. This work focused on basic aggregations for instance: sum, average, count, maximum and minimum. The pre-aggregated data step is implemented as an optimization and evaluation extension for the query processing module of Rasdaman (Array DBMS) (Baumann et al., 1997). The proposed algorithm

starts by checking the existence of a perfect-matching between the input query and pre-aggregated queries. If it exists, then the result of pre-aggregated relation is returned. If not, a partial-matching is searched. If it is found, the result is returned from the pre-aggregated results or from raw data based on the minimum cost.

2.3 GPGPU based methods

Inspired by the power of the GPGPU to accelerate general purpose application (cited above), in the last years, different works have been proposed to speed up the processing and the analysis of raster data.

Hence, the work presented in (Zhang et al, 2010a) investigates the use of GPU to speed local operation of map algebra in particular the addition operator. Their goal is to reduce the computation of the addition of two giant rasters by mapping the traditional serial algorithms to GPU parallel processing architecture using CUDA (Compute Unified Device Architecture proposed by Nvidia) (Zeller, 2011). So, in their work, they suppose 2-layers data file of the same resolution (two rasters having the same resolution): file A and file B with the same size: m*n. These rasters are stored in float array: Array[2][m*n] . The Figure 2.3 describes the local sum operation between two rasters.

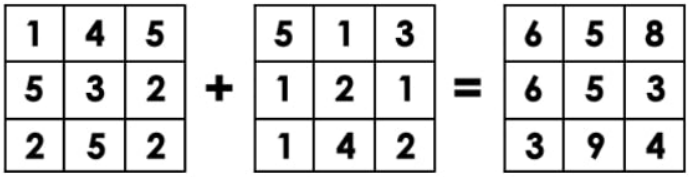

Figure 2.3. Addition operation between the two raster layers.

Concerning the CPU implementation of this operation; the authors propose, one loop to operate the arrays and save the results in Res array: Res[m*n], as follow:

```
Res[i]= Array[0][i]+ Array[1][i].
```

This method consists in a trivial algorithm to perform the sequential addition of the two rasters.

To implement this method on the GPU, the authors fixed 256 threads per block. Hence, the number of blocks is obtained as follow:

$$\frac{The\ size\ of\ the\ raster\ file}{\text{The thread number of per block}}$$

As mentioned before, each raster file is aligned and both of them are stored in 2D array arr_d such that the number of the rows is 2 and the size of number of the columns is the size of the raster files.

In order to get fast access to the rows and the best performance of cudaMemcpy2D(), the authors adopted a linear memory allocation using cudaMallocPitch() instead of allocating the 2D arrays with cudaMalloc. Hence, the access to the row i is realized by this instruction:

float*row=(float*)((char*)arr_d + pitch*i).

To get the data in this row: data =row [index of the element]

hence, the kernel is defined as follow:

```
__global__ static void SumKernel (float* arr_d, size_t pitch, float* arry, int Z) {
    float data, Data=0.0f;
    for(int i=0; i<Z; i++) {
        float*row=(float*)((char*)arr_d + pitch*i);
        data=row[blockIdx.x*blockDim.x+threadIdx.x];
        Data+=data;
    }
        arry[blockIdx.x*blockDim.x+threadIdx.x]=Data;
    }
```

As we can see in the kernel, each thread will execute the loop “For”. It means that each thread will access to its corresponding cell in raster A and B and sum the values. Finally, the result will be stored in the arry.

Note:

- **arr_d**: Corresponds to the input data **Array containing the two rasters.**
- **Arry**: Corresponds to result raster **Res**
- **Pitch**: Return value of the function **cudaMallocPitch()**
- **Z:** The number of rasters (2*)*

In most complex analysis, the processing of raster data is performed in many steps, one after another, using batch processing. The authors of (Steinbach et al., 2012) propose to speed up a batch processing of spatial raster analysis based on a specific use-case data flow using the computing power of the GPU on a map of 844 woodlands with a size of 2275 × 2263 (Figure 2.4). The data flow consists of two steps; each one requires many raster operations.

Figure 2.4. Dataflow Map (Steinbach et al.,2012)

The first step relay on a neighborhood analysis and consists in counting the cells in a specific wood neighborhood. The authors have chosen the wedge shape for neighborhood since it is a time-consuming task (Figure 2.5).

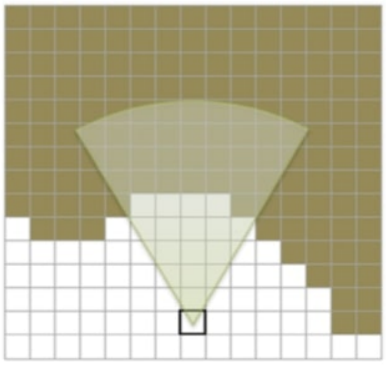

Figure 2.5. Wedge shape neighborhood (Steinbach et al.,2012).

These steps can be realized using raster data operations based on the sum operator for focal functions with different wedges. The second step concerns the mean value analysis which computes the center of each woodland. This analysis is done for each woodland individually. Hence, first, a single woodland is selected. Second, the Euclidean distance transformation (EDT) is applied to compute the shortest Euclidean distance from each cell to the nearest non empty cell. Finally, a zonal mean function is applied to compute the mean and hence get the center of each woodland.
These operations have been implemented as a plugin for the open-source software GRASS (Neteler et al., 2012), based on GPU, using OpenCL, which is an open standard that can be used to program CPUs, GPUs, and other devices. OpenCL programs can be run on Nvidia, AMD and others, which make them slower than CUDA concerning the performance.

Another work using the GPU to deal with raster analysis, is presented in (Wu et al.,2007). The authors have studied two types of raster analysis: raster local operation and neighborhood operation. In the case of raster local operation two subcases have been formalized as follow:

- Local operation based on a single raster where the cell values of the output raster are computed by: $Output_{i,j} = f(Input_{i,j})$.
- Local operation based on many rasters. In this case, several rasters are used as input. The output will be one raster computed, such that, $Output = f(Input^{0}_{i,j}, Input^{1}_{i,j}, ..., Input^{n}_{i,j})$ where n represents the number of raster cells. In this case, they propose a parallel implementation of weighted average local operation of many rasters.

The neighborhood operation is for a single raster using the following filter mask (Figure 2.6):

1	**1**	**1**
1	**1**	**1**
1	**1**	**1**

Figure 2.6. Filter mask.

To implement their algorithms on the GPU, the authors have used OpenGL 2 as graphic API and OpenGL Shading Language (GLSL) as shading language which is supported directly by OpenGL without extensions.

Unfortunately, in their test they did not use a very large data set. They also have used only 4 rasters in the case of local operation with multiple rasters, which is a very small number. Moreover, the choice of OpenGL is not common, since it is a library used primary for accelerating the Graphics Rendering on the GPUs and not widely used for general purpose computing.

Motivated by the need to provide large-scale raster geospatial data indexing, the work presented in (Zhang et al.,2010b) proposes a fast indexing of large-scale raster geospatial data to support Region-of-Interests (ROI) queries (or spatial range queries). This type of queries returns objects that satisfy one or more value range criteria, for instance temperature in the range [t_min,t_max]. To do that, the authors have designed a Cache Conscious Quadtree data structure (CCQ-Tree) based on the GPU, dedicated to fast indexing of large-scale raster data.

To construct the CCQ-Tree, the authors propose an implementation in parallel of 3 steps: The first one is the construction of a pyramid of matrices from the raw raster data. Second, the computation of the First-Child Node Positions, and finally generating the CCQ-Tree. The steps are executed sequentially in the CPU, however each step itself is executed in parallel. Once constructed, the index will speed up the range queries. The proposed algorithms were tested on real climate dataset published by WorldClim (Hijmans et al., 2005) for 5 × 11 rasters of size: 4096 × 4096.

Motivated by the power of the GPGPU devices, the work presented in (Zhang et al., 2013) proposes a parallel method to implement a quadtree for large-scale raster spatial indexing to speed up query processing and data analysis. The idea is mapping the geospatial processing into a set of data parallel primitives based on CUDA which are a highly-optimized functions. The design for the construction of a binned min–max quadtree (BMMQ-tree) (Zhang et al., 2010b) involves several steps (Figure 2.7):

1. First step consists on mapping a 2D input raster grid into a Z-ordered 1D array (Orenstein, 1986).
2. As a second step, recording the minimum and maximum values for every four consecutive Z-ordered raster cells.
3. The third step, derive higher levels of min–max tables from the lower-level ones, by using a procedure similar to Step 2.
4. The fourth step consists in two sub steps:
    a. Compute the positions of the first child nodes.
    b. Prune quadtree nodes that represent uniformly distributed quadrants.

As cited before, to implement the BMMQ-tree in parallel, the authors have used parallel primitives in order to perform each step in the quadtree construction. For instance,

- Scan parallel primitive;
- Transform;
- Scatter;

The use of these primitives allows the authors to focus on the design, rather than the implementation hardware specifications.

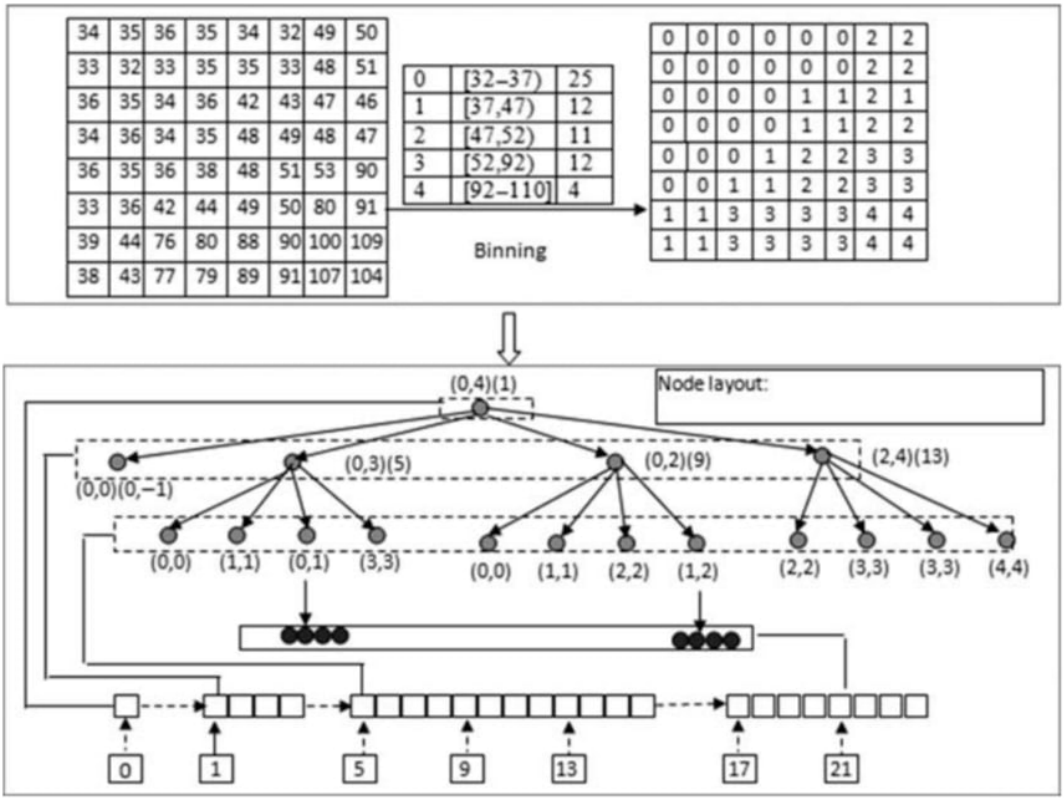

Figure 2.7. A BMMQ-tree (Zhang et al., 2010b).

The work presented in (Xia et al., 2011) tackles two main problems in geospatial data analysis:

- Interpolation using Inverse distance weighting (IDW) algorithm, that assigns geographical values to unknown spatial points using values from a usually scattered set of known points.
- Viewshed analysis of digital elevation model (DEM) (Walker et al., 1999) to determine visibility to or from a particular cell.

Since, these popular algorithms are very important in geospatial analysis and are highly computation-intensive, especial for large-scale spatial data, the authors propose to speed up these algorithms by using GPGPU and CUDA. To implement the IDW algorithm, first, a quadtree-based domain decomposition algorithm is applied for load balancing of the input data. Thus, the domain is decomposed into quads of different sizes (Figure 2.8). Finally, the IDW interpolation is executed on these quads.

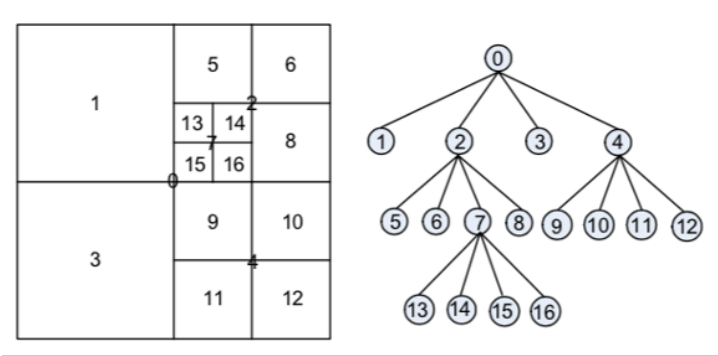

Figure 2.8. An example of the quadtree-based domain decomposition algorithm (Xia et al., 2011).

The parallel implementation of IDW, based on CUDA, consists in four steps (according to Clark parallel algorithm): spatial domain decomposition, interpolation, output data gathering, and visualization. Since the three steps, domain decomposition, data gathering, visualization are not computation-intensive, the authors decided to execute them on the CPU. However, the interpolation is executed on the GPU (Figure 2.9) because it is a massive computation task. Hence, each quad is assigned to a block of threads where its size is the maximum size of quad. Using IDW interpolation, each thread computes the interpolation value of one point based in the input neighborhood points.

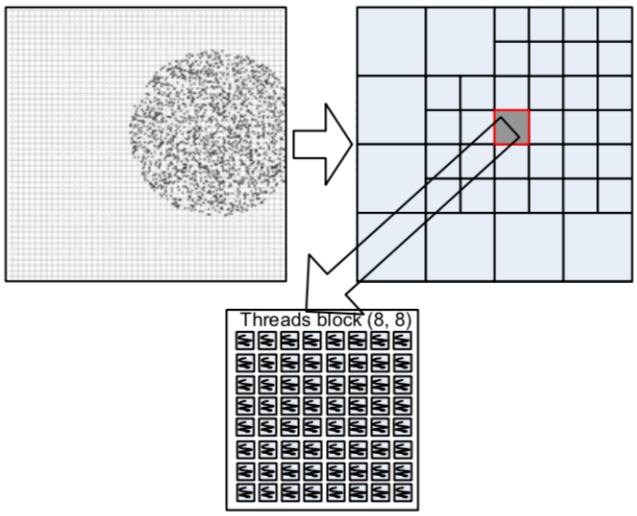

Figure 2.9. The quads assignment to the GPU threads blocks of the maximum size (Xia et al., 2011).

The Viewshed algorithm called line-of-sight analysis or intervisibility analysis consists in determining visibility to or from a particular cell (observer)(https://en.wikipedia.org/wiki/Viewshed_analysis). This algorithm
is based on two steps: (a) Finding all the rays from the observer; (b) Doing visibility analysis for all the cells passed by each ray (Figure 2.10)

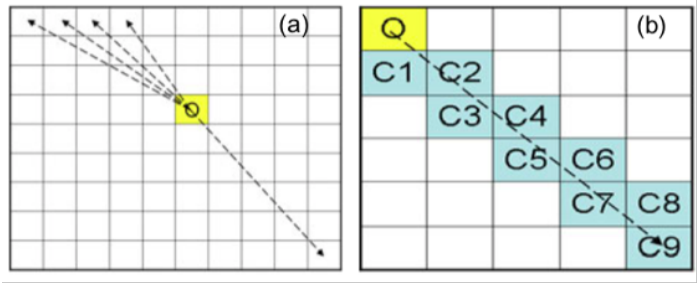

Figure 2.10. The two steps of Viewshed analysis on raster data (Xia et al., 2011).

As we can see in Figure 2.10.a, the input raster represents an elevation grid called DEM (digital elevation model). A line of sight is produced from the origin point (the observer) identified by 'O' to another destination cell. By scanning the grid cell, we can compute all the rays of sight.
In Figure 2.10.b, Moving along the line of sight from the observer O, all the pointed intersect with the line sight must be studied for visibility analysis. These two steps are implemented as two layered components: matrix traversal and ray traversal. The authors propose four combinations for viewshed implementation based on the CPU and the GPU. For instance, the method PMPR implements the two layered components completely in parallel (Figure 2.11), using threads matrix of GPU (Figure 2.12). The gray elements represent threads allocated for visibility computation.

PMPR

// Parallel, traverse all the cells passed by all the rays
Calculate the angles of all the cells passed by all the rays;

// Parallel, traverse all the rays
// For each ray, scan from the nearest to the furthest to observer
// For each ray, put the maximum angle ever scanned to an array
For all the rays, perform a max-scan on the calculated angles of the cells passed by each ray;

// Parallel, traverse all the rays
// For each ray, if angle_array[*i*]==max-scan_array[*i*], the cell is visible
For all the rays, compare each element in the same position of angle array and its max-scan array of each ray;

Three CUDA kernel procedures run on threads of GPU

Figure 2.11. (Xia et al., 2011)

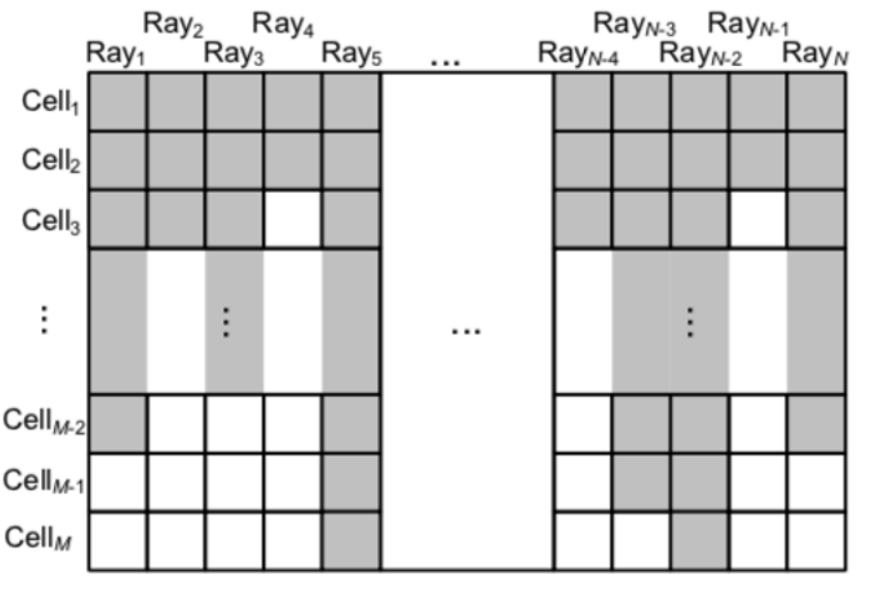

Figure 2.12. Threads matrix allocation on GPU for PMPR (Xia et al., 2011)

2.4 Distributed systems-based methods

Motivated by the success of the use of high performance parallel super computing techniques in particular grid computing and cloud computing in simulations, the work presented by (Gunawardena et al., 2016) propose new solutions for earthquake simulation based on distributed computing in order to analysis petabytes of data required for simulation. The authors suggest the use of MapReduce (Maitrey et al., 2015) system using Hadoop (http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009) and Mars (He et al., 2008) which is Hadoop on the GPU, to overcome the complexity of the dataset. Hence, in this work an octree for CVM (community velocity models, ground models) (Schlosser et al., 2008).

Another work trying to deal with large-scale geospatial data, the work presented in (Cary et al., 2009) experiments the use of MapReduce as a massively parallel computing tool in order to overcame non scalability of traditional approached based on single node. Hence the authors propose to solve two spatial data related problems using Hadoop on a Google & IBM cluster (Google and IBM Academic Cluster Computing Initiative): bulk-construction of R-Trees (Kriegel

et al., 1990) using vector data and aerial image quality computation involving raster data, the obtained results show an improvement of the completion time of two problems.

2.5 Synthesis and Strategy

In this chapter we have shown different methodologies that have been used for processing large-scale spatial data. We have classified these methods on three main categories according to the used technology: methods based on uniprocessor CPU, parallel methods based on the GPGPU and finally methods based on distributed systems. The main presented methods deal with raster data.

Research work	Studied Problem	Computation technic	Type of Result	Pre-processing
Performance optimization of grid aggregation in spatial data warehouses (Kang et al., 2015)	Raster aggregations	CPU	Estimated	No
Computing Aggregate Queries in Raster Image Databases Using Pre-Aggregated Data (Baumann, 2008)	Raster aggregations	CPU	Exact	Yes
GPU-Accelerated Parallel Algorithms for Map Algebra (Zhang et al., 2010a)	Addition of two rasters	GPGPU	Exact	No
Accelerating batch processing of spatial raster analysis using GPU (Steinbach et al., 2012)	Raster analysis	GPGPU	Exact	No
Improving the performance of spatial raster analysis in GIS using GPU (Wu et al., 2007)	Raster aggregations	GPGPU	Exact	No
Indexing Large-Scale Raster Geospatial Data Using	Indexing	GPGPU	Exact	No

Massively Parallel GPGPU Computing (Zhang et al., 2010b)				
High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives (Zhang et al., 2013)	Indexing	GPGPU	Exact	No
Accelerating geospatial analysis on GPUs using CUDA (Xia et al., 2011)	IDW interpolation and Viewshed analysis	GPGPU	Exact	No
Spatial Data Processing with MapReduce (Gunawardena et al., 2016)	Octree indexing for earthquake simulation	Hybrid Distributed systems	Exact	No
Experiences on processing spatial data with MapReduce (Cary et al., 2009)	R-Trees Indexing and Image quality computation	Distributed systems	Exact	No

Table 2.1. A review of methods for spatial data processing.

We summarize our state of the art in the table 2.1. The first notice which pops out in this review is that only few works have been proposed for the aggregation and no one for selection queries of raster data, which creates a need and opportunities to fulfil this lack. In fact, these types of queries are useful in data warehouses and many other applications.

The CPU-based raster aggregation methods rely on providing an approximative solutions (and not exact solutions), and pre-processing step which is often also an intensive computation task that need to be optimized (Daras et al., 2018). Pre-processing is not always possible and realistic, especially with large-scale spatial dataset and complex applications. The preprocessing can be also difficult to do when the aggregation is computed only on a user-defined raster subregion that can change over the queries - this aspect can make the aggregation inputs non-predictable.

The parallel based approaches (GPGPU and distributed systems) are an efficient solution to tackle the problem of large-scale spatial data which are data and computation intensive. The GPGPU devices are dotted with thousands of processing cores that are capable of launching thousands of threads simultaneously which make it ideal solution for massively raster parallel applications. Currently, they are used as accelerators of CPUs. Thanks to its high throughput (Garland and Kirk, 2010), the GPGPU are especially suitable for geospatial data processing due to the inherent parallelism of most of geospatial operations (Theobald, 2005). Furthermore, the GPGPU is based on the SIMD paradigm (single instruction multiple data) (Cardoso et al., 2017) which make it a convenient solution for our queries (raster cell aggregations based simple arithmetic operations such as addition). The GPGPU is low-cost powerful tool to speed up large-scale spatial data. It does not need complex infrastructure to manage the data. A single desktop or laptop computer having a GPU card can provide numerous parallel cores. In our opinion, GPGPU is a good candidate to implement raster processing like aggregation and selection.

Distributed systems (Firoj et al., 2015) constitute a powerful tool for Big Data that usually includes heterogenous datasets with sizes of terabytes and petabytes. Distributed systems are powerful but they usually need to build complex architectures and infrastructures, which are often costly and time-consuming in terms of maintenance. Usual raster data sets are dealing with gigabytes.

Chapter 3

Background

The goal of this chapter is to provide the readers with the necessary knowledge concerning spatial data, the Map Algebra and GPU parallel computing. The introduction of these conventions will allow the readers to be comfortable with rest of our manuscript in particular the contribution part.

3.1 Spatial Data

In general, Spatial data are divided into two main categories: raster and vector data (Agosto, 2013) (Figure 3.1).

3.1.1 Raster Data

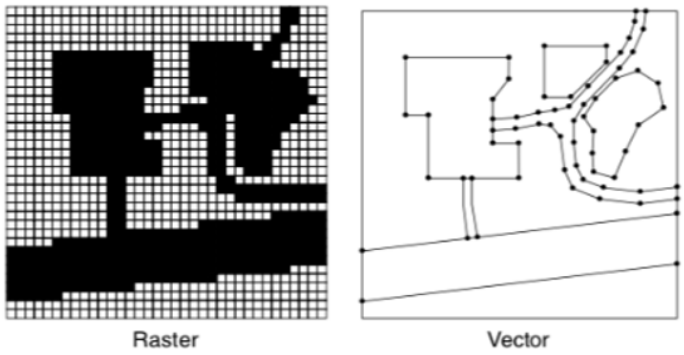

Figure 3.1. Raster and Vector data.

Raster data take the form of an 2D array or 2D-grid of cells. Each cell contains three information: the coordinates x and the y, and a value (Figure 3.2) that represent a measure of a characteristic in that geographic point such as: The pressure, temperature, elevation, soil pH, etc.

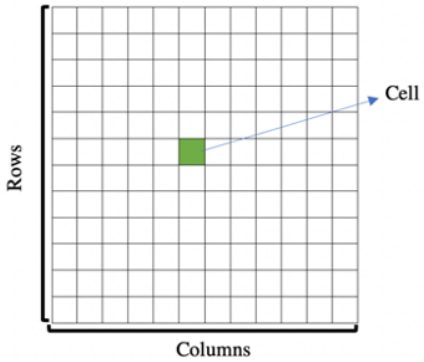

Figure 3.2. Raster is composed of rows and columns of cells.

The rasters can represent spatial objects for instance: points and arcs. Thus, A point can be represented by one cell, and the arc can be a sequence of cells. The second category of spatial data is the vector data which are spatial objects that are constructed using points and lines(edges) as primitives (Halpin et al., 2006).

Raster interpolation

Spatial interpolation (Greenberg et al.,2011) is a technique widely and commonly used in Geographic Information System (GIS), to create continuous raster data from a subsample of measurement point values such as soil properties, temperature, and precipitation in specific locations. The goal is to create surface data using mathematical function (interpolation function) to predict the unknow values for the missing location points (Figure 3.3).

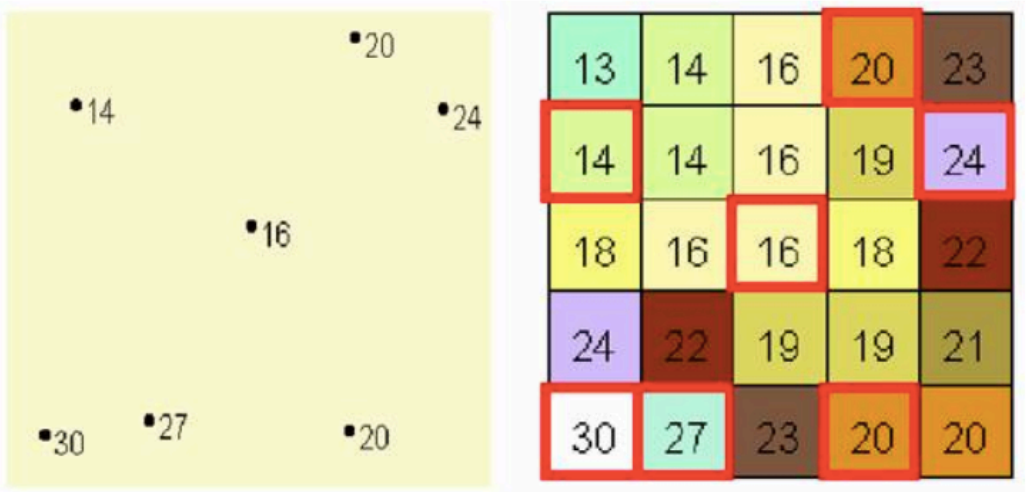

Figure 3.3. Raster interpolation(http://planet.botany.uwc.ac.za/nisl/gis/spatial/chap_1_11.htm).

Figure 3.3. Raster interpolation(http://planet.botany.uwc.ac.za/nisl/gis/spatial/chap_1_11.htm).

As we can see In the Figure 3.3. In the left we have the measurement points while in the right a raster is generated by the interpolation of these points. Thus, the missing values (unknown) are estimated based on the nearby measurement values. In the literature many methods have been proposed to perform spatial interpolation, for instance:

- Inverse distance weighting (IDW) (Singh and Verma, 2019).
- Least cost distance analysis (Greenberg et al.,2011).
- Kriging Interpolation (Singh and Verma, 2019).

Spatio-temporal data

Spatio-temporal data are data that are related and collected over time and space. In general, it describes a phenomenon in a certain location and time which allow to study its behavior on specific area overtime for instance climate, meteorology and biology in order to make predictions and precise description allowing reliable decision making. Nowadays, the volume of the produced spatio-temporal is growing dramatically especially in climate and environmental data. Hereunder an example of spatial-temporal data.

Spatio-temporal rasters, in particular, can be viewed as a sequence of rasters for the same region and for a defined period of time. Each raster represents information related to the studied region at regular intervals of time or frame time (temporal granularity) (Pozzani and Zimányi, 2012) (e.g. every second, minute, hour, etc.) – see Figure 3.4 and 3.5. Spatio-temporal rasters allow the analysis of the gradual evolution of temporal phenomena such as the detection of abnormal phenomenon evolution over time in a studied region.

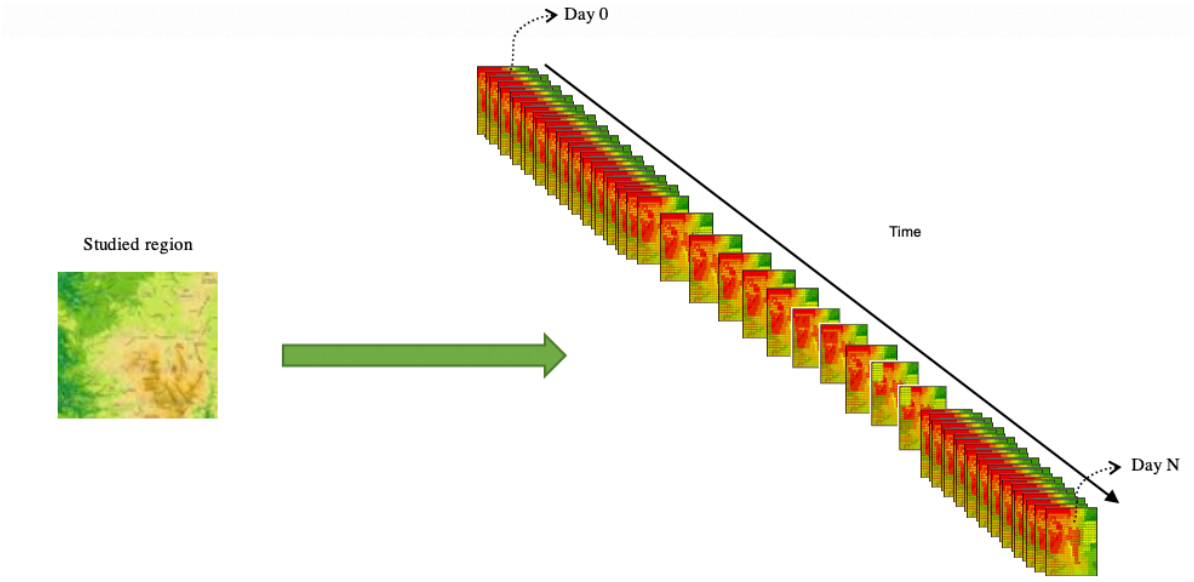

Figure 3.4. Spatio-temporal rasters representing the evolution of the temperature during N days for a studied region.

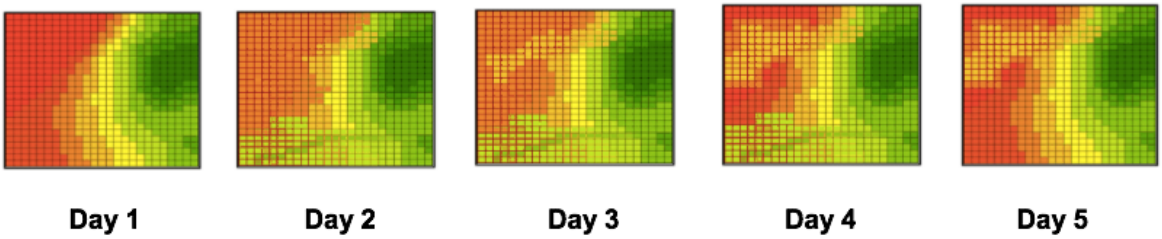

Figure 3.5. Example of a raster set.

3.1.2 Vector data

Vectors are constructed using discrete geometric locations (x, y values) (points or vertices) that define the shape of the spatial object (Figure 3.6). The vectors can be:

- Points: Points are defined using the coordinate x and y. Points can represent for example: the location of trees.
- Lines: Lines are defined using at least two connected points. Lines can represent for instance roads and streams.
- Polygons: Polygons defined by 3 or more connected and closed vertices. They represented for example lakes and oceans.

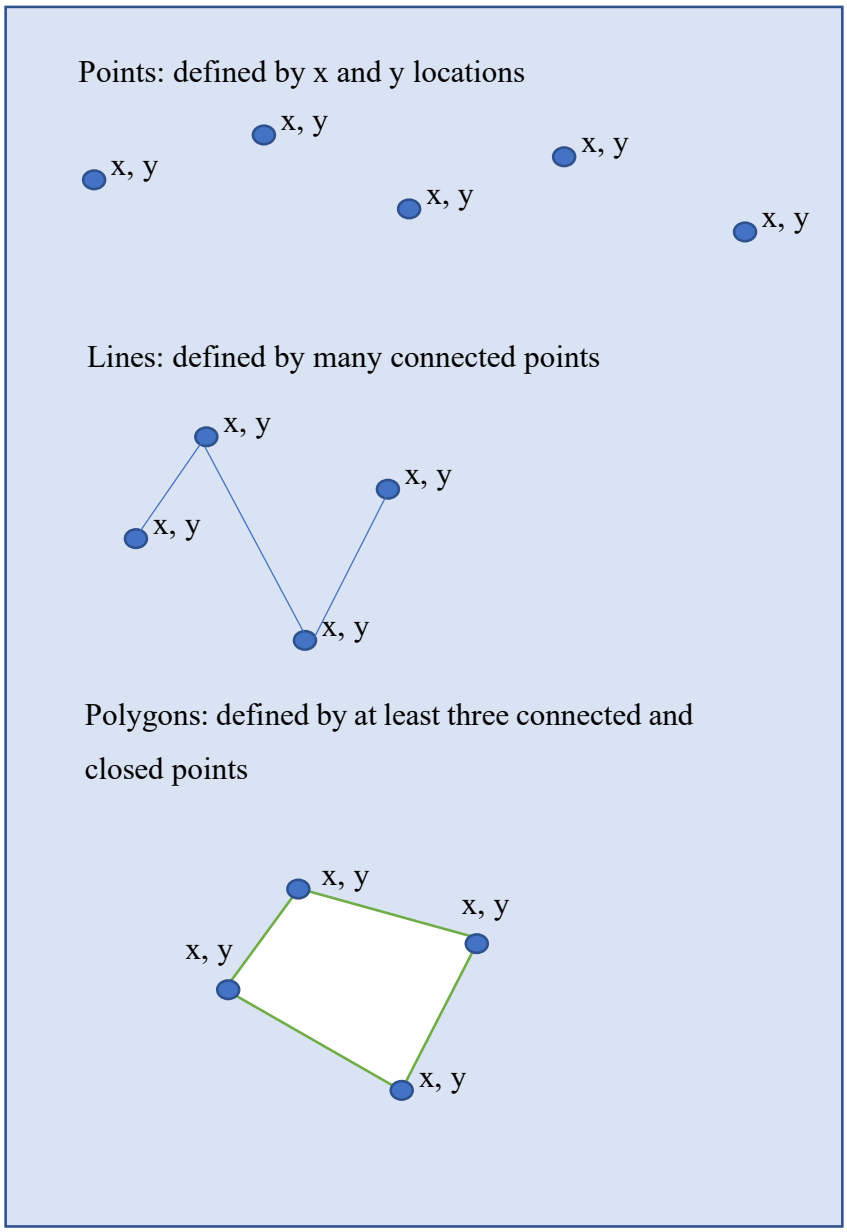

Figure 3.6 Categories of vector data (National Ecological Observatory Network (NEON))

3.2 Map Algebra

Map algebra (Pullar, 2001) or Raster Math is a set of conventions, capabilities and techniques adopted by GIS to visualize and process raster data proposed by Tomlin (Tomlin, 1994). There are many operations that can be performed using Map algebra. Some of them are simples and others are complexes:

- Arithmetic operations use basic mathematical basic operations for instance: Addition, Subtraction, Multiplication and Division.
- Statistical operations based on statistical operations such as: Minimum, Maximum, Average, and the Median.
- Relational operations used for cell comparisons using the following function: Greater than, Smaller than, and Equal to.
- Trigonometric operations using classical mathematics trigonometric functions such as: sine, cosine, tangent and arcsine.
- Exponential and logarithmic operations use the exponent and logarithm functions.
- Other more sophisticated operations.

The Map Algebra functions can be classified as follow:

- Local functions: they are based on cell-by-cell operations. For instance, suppose we have two rasters A and B, and we want to generate a third raster C which is the sum of cell by cell of the two rasters. Thus, we will add each cell in the same location of two rasters one by one (Figure 3.7) – this operation corresponds to a matrix sum.

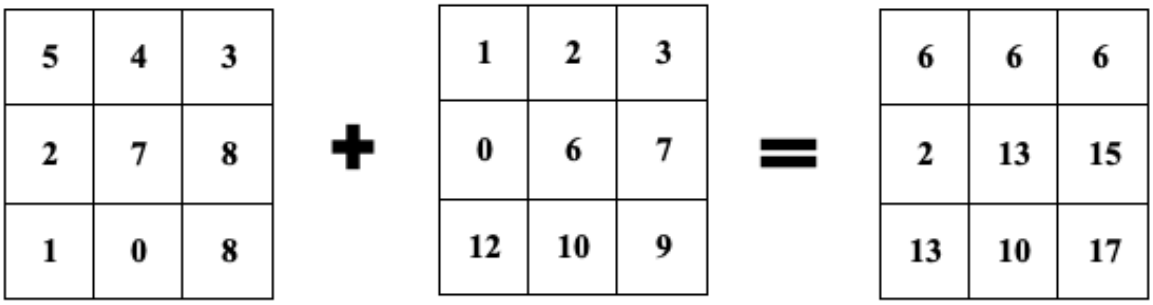

Figure 3.7. The sum of two rasters (cell by cell).

- Global functions: A global operation is a process or function that is performed on each output raster cell using all the cells of the input raster. As an example, the Euclidian distance is shown in (Figure 3.8)

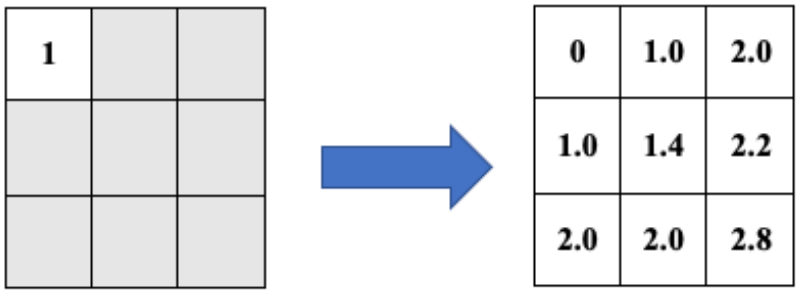

Figure 3.8. Global functions.

- Focal functions: Focal or neighborhood functions are operations that compute an output value of each cell or raster using neighborhood its values.
  Such operations are widely used in image processing (convolution, kernel, filtering) (Ak et al., 2012). An example of focal functions the figure bellow, in which the studied cell in the output is computed by summing up the cells falling on its window neighborhood of size 3 × 3.

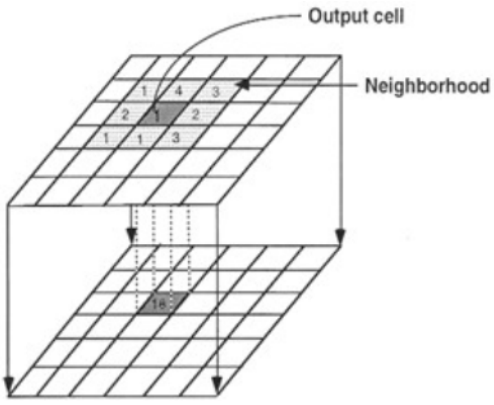

Figure 3.9. Focal functions.

- Zonal functions: A zonal operation is a spatial function that computes an output value of each cell using the zone containing that cell. An example of Zonal operations is the statistical zonal operations for instance: zonal mean where each output cell is obtained by computing the mean the zone containing the cell.

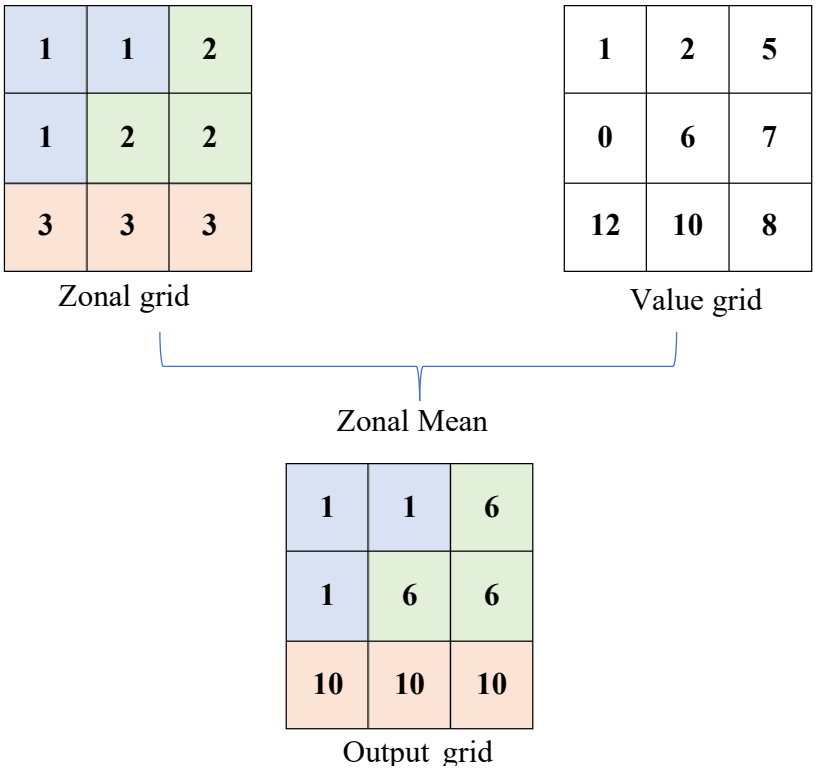

Figure 3.10. Zonal functions.

3.3 GPU parallel computing

The previous generations of computers contained only central processing units (CPUs) that are dedicated to performing general programming tasks. However, in the last decades, several computers with different architectures have emerged including other processing elements, for instance, GPUs (Graphics Processing Units). In the last decade, HPC (high-performance computing) (Assiroj et al., 2019) has known a significant evolution, because of the emergence of GPU-CPU heterogeneous architectures, which has led to a great revolution in parallel programming.

Driven by the success that have been achieved using the GPU devices for video rendering, in the last decade, many researchers have tried to use the GPGPU for speeding up many applications in several fields. For instance: simulation, image processing, machine learning, and GIS. For

example, the research presented in (Viola et al., 2003), tackled the problem of various non-linear filters for volume smoothing with edge preservation in image processing using the GPGPU. The authors of (Yang et al.,2008) propose the implementation of several image processing algorithms like histogram equalization, and edge detection (and others), based on the GPGPU using CUDA. In (Temizel et al.,2011), implementation using the GPGPU is proposed for image and video processing to tackle real-time issues and optimization.

GPUs (Graphics Processing Units) have had an incredible evolution. Driven by computer games, the performance and the capability of the GPUs have increased drastically in the past few years. After being a simple device for graphical tasks, GPUs have become devices with a highly parallel programmable processor which are capable of solving general problems (Kirk and Hwu, 2013). GPGPUs (General-Purpose Computing on Graphics Processing Units), unlike GPUs, are intended to deal with more general problems, such as simulations, optimizations and other complex problems in several application fields.

GPUs do not replace the CPU-based computing. In latency-oriented systems, CPUs are intended for several types of tasks and applications, especially those that involve intensive control task computing, branch prediction, large caches and data fetching (Kirk and Hwu, 2013). GPUs, which are throughput-oriented systems, are suitable for intensive parallel data computation tasks based on the SIMD paradigm (Single Instruction Multiple Data). Thus, thousands of efficient cores are used by the GPU to perform massive data parallel computing (Figure 3.11).

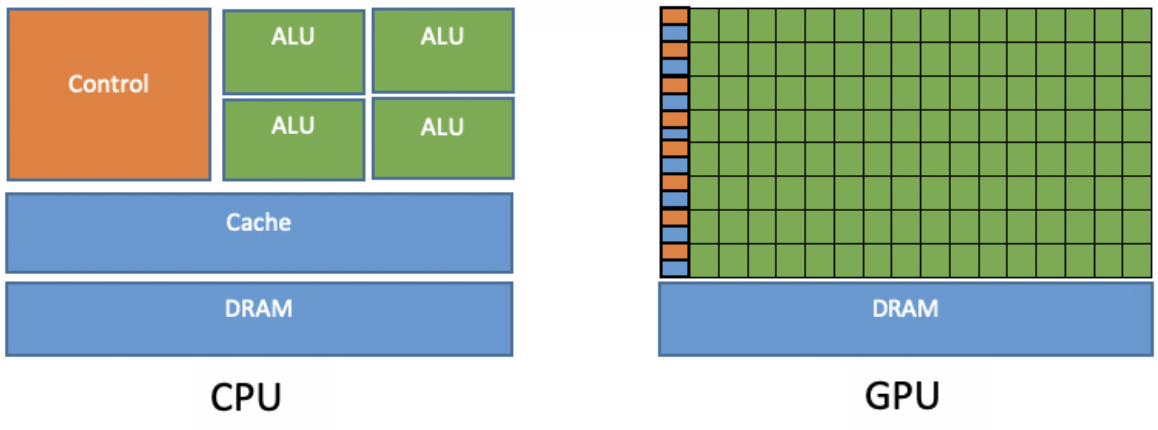

Figure 3.11. Difference between CPU and GPU architecture (Kirk and Hwu, 2013).

In the traditional architecture composed of a CPU and one or many core GPUs, the GPUs are used as co-processors to the CPU. The cooperation between the CPU (Host) and GPU (Device) is achieved through the PCI-express bus (Figure 3.12).

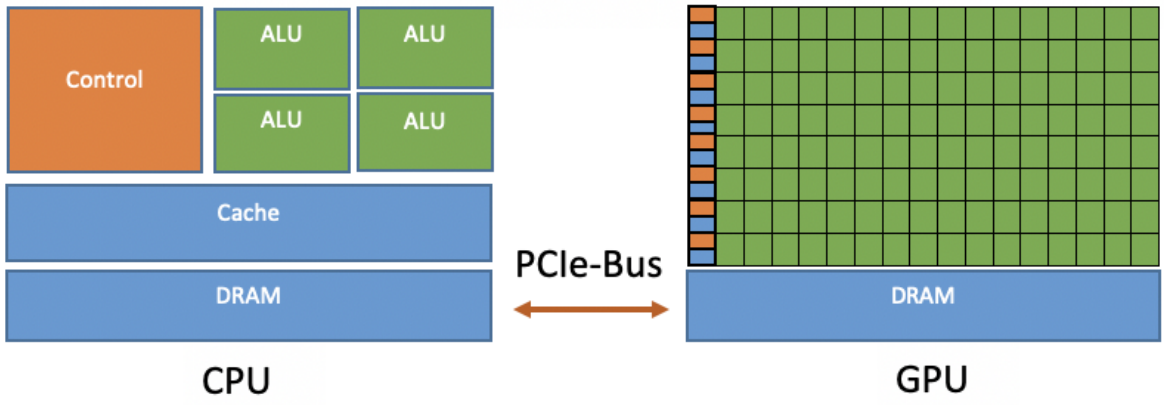

Figure 3.0.12. CPU-GPU Heterogeneous architecture (Kirk and Hwu, 2013).

The cooperation of the CPU and the GPU (Figure 3.13) led to a high-performance and powerful computing capabilities which make the heterogeneous architectures a suitable tool for HPC (High Performance Computing). The host code is run on the CPUs while the device code is run on GPUs. An application executing on a heterogeneous platform is in the first time initialized by the CPU. This one is responsible for managing the environment, code, and data for the device. In the end, parallel tasks are loaded on the device (GPU).

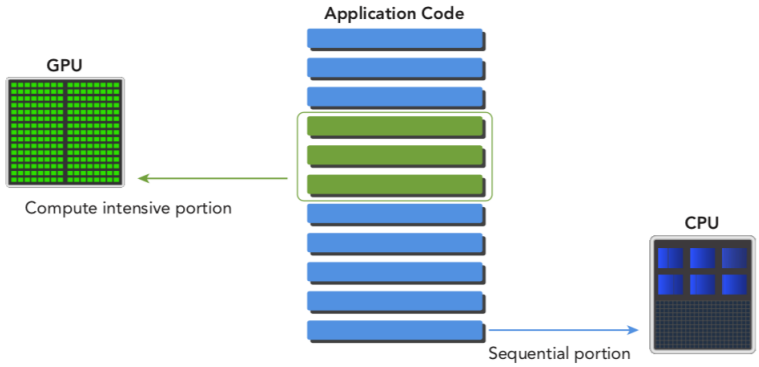

Figure 3.13. The cooperation of the CPU and the GPU.

The GPU device is composed of many Streaming Multiprocessors (SMs) which are responsible for running the parallel functions called kernels. The number of SMs can vary from one generation of CUDA GPUs to another (Kirk and Hwu, 2013). Each SM is composed of many computing units (cores), thousands of registers, several memory caches, warp schedulers, etc. (Figure 3.14).

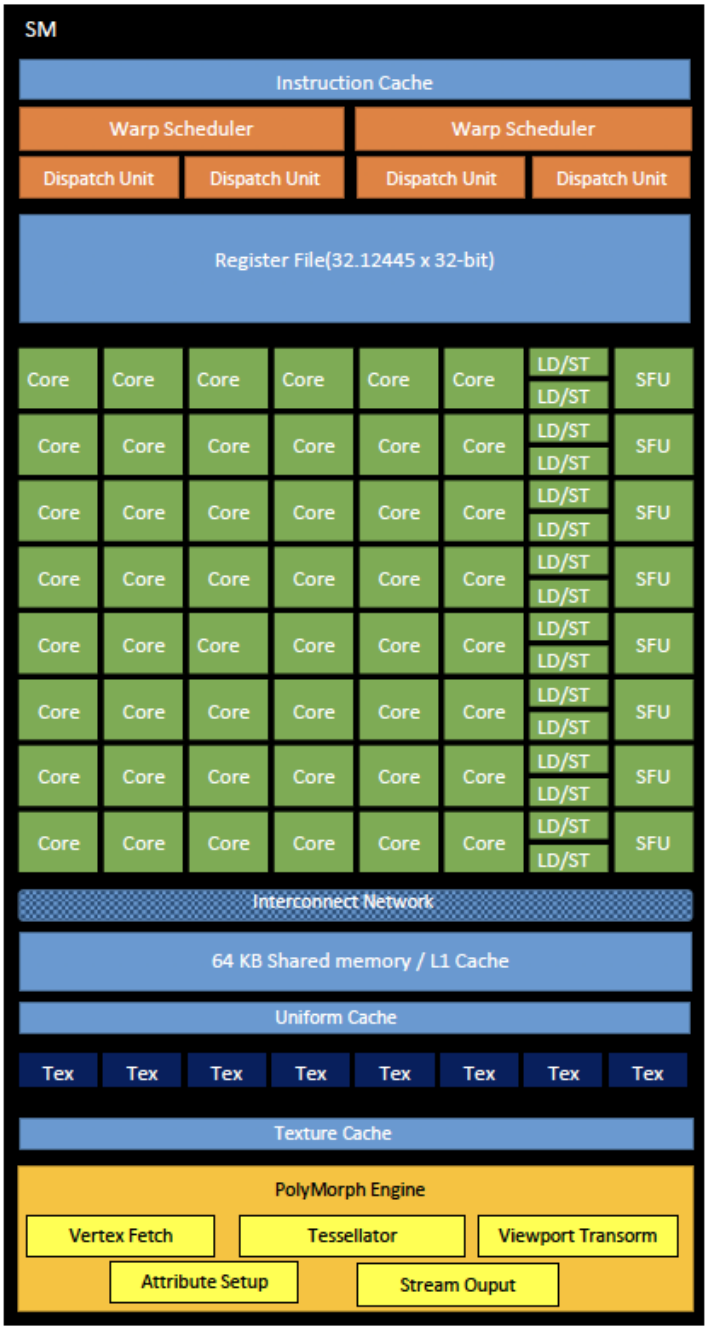

Figure 3.14. Example of an SM architecture (https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/architecture).

3.4 CUDA programming model

To exploit the power of this sophisticated architecture, new application programming interfaces (APIs) have been developed to use the GPU for non-graphics applications without using classical graphics-oriented APIs like OpenGL, by the use of extensions to high-level languages such as C, C++.

Hence, for instance, NVIDIA has created a parallel computing platform and programming model called CUDA (Cheng et al.,2014). Khronos Group has created the Open Computing Language (OpenCL), and Microsoft has created DirectCompute (Sanders et al., 2015). In our work, we have used CUDA since OpenCL (Kirk and Hwu, 2013) is slower than CUDA which is highly optimized by Nvidia in order to be used widely on the GPU.

While developing CUDA, NVIDIA took standard C and added a set of small number of keywords that are specific to CUDA. Thereby, using CUDA will be easier for the developers that are quite familiar with C. Thus, the programmer still code in C but in the same time, he incorporates the new CUDA keywords in his code to express the parallelism. In addition to C, developers can use CUDA with C++, Fortran and Python. Coding with CUDA requires a deep understanding of the different features of its architecture, especially the programming and the memory model.

CUDA execution programs consist of 3 steps:

- Initializing and copying the data from the CPU memory (i.e., the host) to memory to the GPU (i.e., the device).
- Invoking the kernel (parallel function executed on the Device (GPU) by many threads).
- In the end of the processing, transferring the results from the Device to the Host.

Kernels are functions that are executed on the GPU by many threads in parallel (Ruetsch and Oster, 2008). A deep understanding of how threads are organized in the device is mandatory to write kernels. CUDA offers to the programmer the possibility of organizing threads. CUDA provides a thread hierarchy abstraction. Threads are grouped in structures called blocks. The blocks are grouped in structures called grids. Launching kernels requires the definition of the size of the blocks and the size of the grids (Figure 3.15).

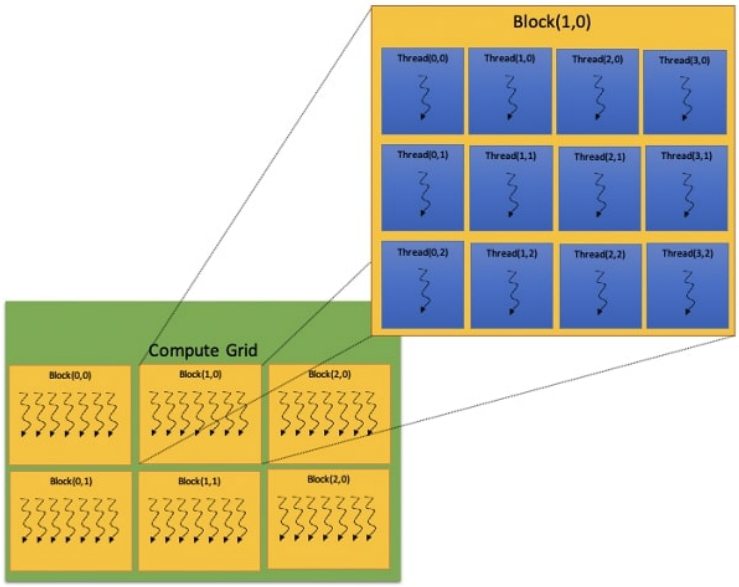

Figure 3.15. The CUDA concept of a grid of blocks (Cheng et al., 2014).

A grid is a set of many thread blocks, and thread blocks are made up of many threads that are cooperating with each other. Usually, the block is organized as a 3D array of threads and the grid as a 3D array of blocks. Each thread uses its block index combined with its own index to be identified in the global

GPU-accelerated Libraries for Computing

NVIDIA GPU-accelerated libraries provide highly-optimized functions that can help to write and optimally scale applications. Using them, allow to get highly efficient implementations of algorithms that are widely used as building blocks for many applications in several fields. Many kinds of libraries are available; there are libraries for linear algebra (Cublas and CUDA Math Library). For deep learning, there are libraries for parallel algorithms such as Thrust (used for parallel algorithms and data structures). The libraries that will be used in this thesis are Thrust and CUB since they provide a set of fundamental parallel algorithms that are implemented in an optimized and efficient way, such as reduction and sort which are used in this work.

Thrust

Developed by NVIDIA, Thrust is a high-level CUDA library that enables the programmers to get high performance and improve their productivity since it is based on STL (Standard Template

Library). While CUDA C/C++ offers a low-level control, which allow implementing high performance algorithms which require significant optimization and deep manipulation for mapping the algorithms onto the hardware. Thrust library is dedicated to problems which do not required low-level control to map the algorithms onto the hardware. Hence, the users only describe their algorithms in high-level, while the library takes over the decision of how the computation are implemented efficiently, for instance: the number of threads, the size of block and grids etc. Thus, the programmers do not worry about the hardware specification and are focused only on the design of the algorithms which will allow them to get more efficient implementations and be more productive. Thrust provides an efficient implementation of the fundamental and common parallel algorithms that constitutes building blocks for more complex algorithms, for instance: sort, reduction and scan algorithms. Moreover, the power of Thrust relies on its interoperability with other technologies, for example, C++, Open MP, etc. It is a part of the CUDA toolkit.

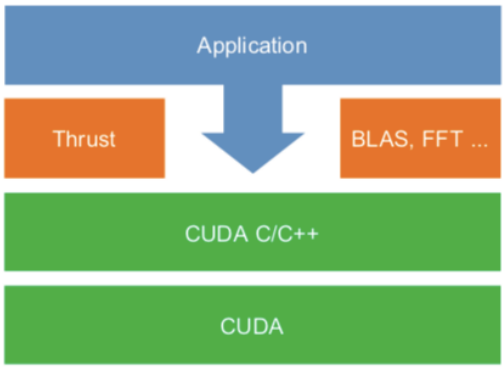

Figure 3.16. Thrust on the top CUDA C/C++.

Besides, programming with Thrust is not difficult and does not require extra knowledge since it is analogous to the use of the C++ STL with standard C code. As we can see in the example hereunder, with few lines, we can generate a vector of random numbers on the device (GPU) and sort them with the primitive sort.

```
#include <thrust/host_vector.h>
 #include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
```

```
#include <cstdlib>
int main(void) {
thrust::host vector<int> h vec(1 << 24);
thrust::generate(h vec.begin(), h vec.end(), rand);
  // transfer data to the device
thrust::device vector<int> d vec = h vec; // sort data on the device
thrust::sort(d vec.begin(), d vec.end());
  // transfer data back to host
thrust::copy(d vec.begin(), d vec.end(), h vec.begin());
 return 0;
}
```

Also, you can notice that we did not specify the number of threads, the size of the blocks as well as the size of the grid. As mentioned before, it is Thrust which will take in charge these specifications in order to provide high performance implementation.

CUB (CUDA UnBound)

CUB is a very fast library CUB library developed by Duane Merrill of NVIDIA Research (ref) and founded using the CUDA programming model. Unlike Thrust, CUB is deeply tied to CUDA, and can deal with low-level implementations such as thread-block and thread-warp levels.
It provides state-of-the-art, reusable software components for every layer of the CUDA programming model:

- Device-wide primitives.
- Block-wide "collective" primitives.
- Warp-wide "collective" primitives.

Like Thrust, it allows the programmer to get very high performance and efficiency and be more productive. Hereunder an example that illustrates the power of the CUB library and its interoperability with Thrust. First, in this example we have used Thrust library to define a device vector such that all the elements of the vector are equal to “1”. After that we apply the prefix-sum on this vector using the CUB primitive function: DeviceScan::ExclusiveSum . As We can see we can use easily Thrust and CUB without any problem of interoperability.

```
#include <thrust/reduce.h>
#include <thrust/device_vector.h>
#include <thrust/iterator/transform_iterator.h>
#include <thrust/iterator/counting_iterator.h>
#include <thrust/iterator/discard_iterator.h>
#include <thrust/copy.h>
#include <thrust/execution_policy.h>
#include <iostream>
#include <stdio.h>
#include <cub/cub.cuh>
#include "TimingGPU.cuh"
#include "Utilities.cuh"
typedef int mytype;
using namespace cub;
int main() {
 int num_items =120000 ; // number of images
// the array that will contain the result( CPU)
 float *h_result = (float*)malloc(num_items * sizeof(float));
// create a device array with  thrust which contain just ones
 thrust::device_vector<float> d_in(num_items, 1);
//  cast iterator to raw pointer which CUB uses
 float   *cub_d_in =thrust::raw_pointer_cast(&d_in[0]);
// create a device vector which will contain the result in the device
 thrust::device_vector<float> d_out(num_items);
//  cast iterator to raw pointer which CUB uses
 float   *cub_d_out =thrust::raw_pointer_cast(&d_out[0]);
 // Determine temporary device storage requirements
 void     *d_temp_storage = NULL;
 size_t   temp_storage_bytes = 0;
```

```
cub::DeviceScan::ExclusiveSum(d_temp_storage, temp_storage_bytes, cub_d_in, cub_d_out, num_items);
// Allocate temporary storage
cudaMalloc(&d_temp_storage, temp_storage_bytes);
timerGPU.StartCounter();
// run the PrefixSum
cub::DeviceScan::ExclusiveSum(d_temp_storage, temp_storage_bytes, cub_d_in, cub_d_out, num_items);
// copy just the first 10 elements of the result

cudaMemcpy(h_result, cub_d_out, 10*sizeof(float), cudaMemcpyDeviceToHost);

}
```

We have to highlight that CUB is the fasters library. The Figure 3.17 shows a comparison between CUB and Thrust for reduction.

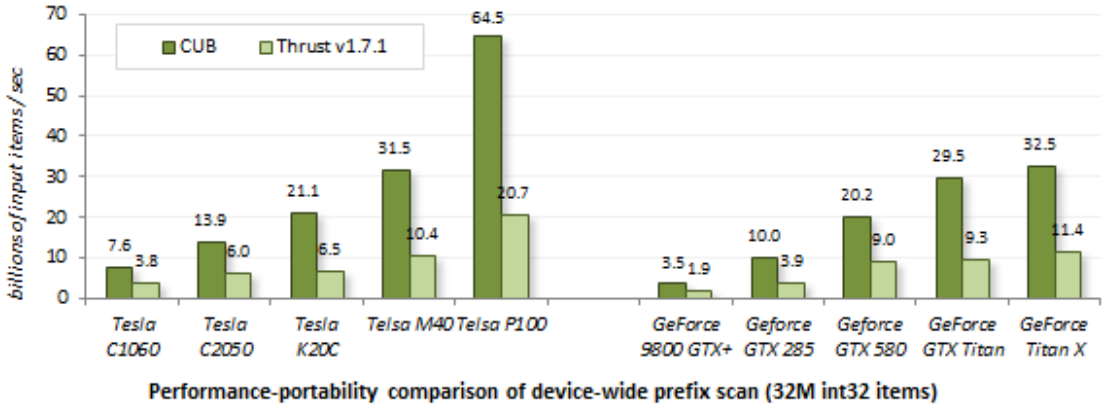

Figure 3.17. Performance comparison between Thrust and CUB (https://nvlabs.github.io/cub/)

Part 2

We dedicate this second part to present in details the different contributions realized during the PhD. Each chapter is related to a published paper contribution based on selections and aggregations of rasters. Thus, this section is organized as follow:

Chapter 4: This chapter concerns our first contribution on the computing of overlapping aggregation of large spatial data sequences.

Chapter 5: This chapter is related to a contribution to improve the performance of raster selection based on a user-defined condition using a sequential approach.

Chapter 6: This chapter is dedicated to a GPGPU-based approach for a raster selection based on a user-defined condition using GPGPU.

Chapter 7: This chapter proposes a disjoint raster subsequences selection method based on a user-defined condition using GPGPU. This approach combines selection and aggregation operations. Each chapter concludes with experiments on a simulated data set, initially presented in the publication associated to the chapter - its reference is indicated after the chapter title.

In our approach, we consider that we do not know in advance on which data the queries will be applied. The data set can change from a query to another. We take the point of view of a Database Management System (DBMS) designer and the goal is to implement generic DBMS operations. Our goal is not to improve the computation for a particular data set.

Chapter 4

Overlapping Aggregation of Raster Data Sequences using GPGPU

Results obtained in this chapter have been published at International Journal of Information System Modeling and Design, vol.10(1), IGI Global USA, p. 20-41.

4.1 Context and motivation

Using Data aggregations to produce data summaries is a classical approach that proved to be efficient for many applications. It can be used as a prior step for many interesting queries such as detecting abnormal phenomena or measurement errors provided by sensors. It is also central in data warehouse and On-Line Analytical Processing techniques. In large data set, the goal is to provide summaries to users in order to explore the data. In this chapter, overlapping aggregations is tackled. When aggregations are overlapping, more value variation can be detected.

Here is an illustration of a short example of the interest of the overlapping aggregation use: Figure.4.1 represents the evolution of the temperature during 11 days. Suppose one would like to detect value peaks in data, in our case we focus on sequences of 4 days where the average temperature over these sequences is greater than 25 C°. It is equivalent to the aggregation of the temperatures for 4 days using the mean operator over four days. Non-overlapping aggregations does not allow finding any sequence of four days satisfying the condition above, i.e., the aggregations of days 1-4, days 5-8, etc. do not allow detecting this data peak. Overlapping aggregations detects the sequence (days 3-6) which satisfies the condition.

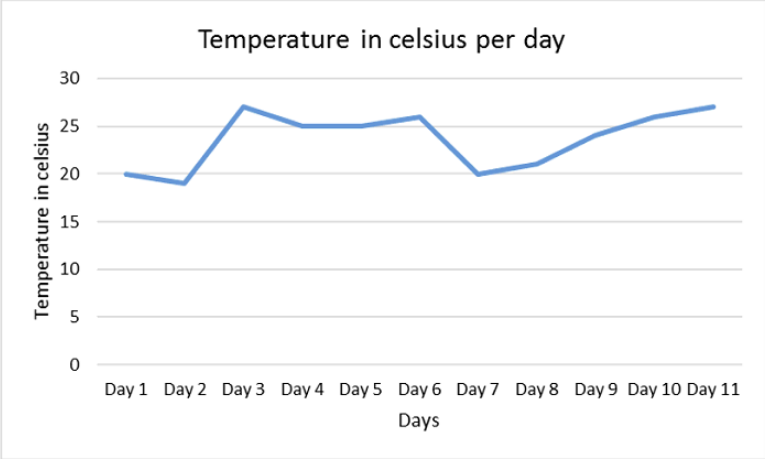

Figure 4.1. Evolution of the temperature over days.

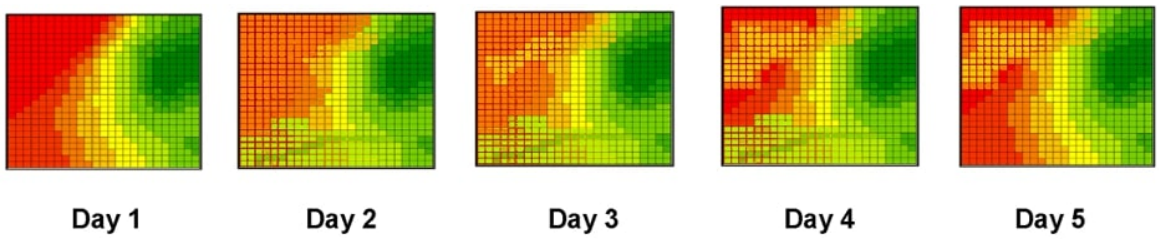

Figure 4.2. Example of a raster set.

The temperature value may come from raster cell aggregations. This paragraph provides details on the overlapping raster aggregations used in this chapter. Suppose the rasters represent the measurements of the temperature for the same region over many days. Figure 4.2 shows an example of this data set type. (1) As a first step, each cell will be aggregated (e.g., using the mean operator) for each raster r, in order to produce one average temperature value by raster. (2) In the second step, the average temperature will be aggregated over time. As an example, an overlapping temperature aggregation is considered in this chapter. This raster overlapping aggregations are more general than the non-overlapping case, because the overlapping processing allows us to compute more aggregations. More precisely, summarized data computed by disjoint aggregations are included in the summarized data computed by the overlapping aggregations. As a result, raster overlapping aggregations require intensive computations. Besides, the computation of summarized data using raster overlapping aggregations with many steps of interleaves, provides users with more data summaries. In this chapter, an interleave equal to 1 has been chosen for the test. The following

aggregations are computed for raster subsequences of length L: (R_1, …, R_L), (R_2, …, R_{L+1}), (R_3, …, R_{L+2}),..,(R_{N-L}, …, R_N).

This interleave choice require the heaviest and massively computations as more subsequence computations are needed. Nevertheless, our approach is generic and can be used for other interleave values. As indicated in the state of art, with the rise of big data, some researchers start to care about accelerating the processing of raster aggregation operations and apply the method to data warehouses. Hence, a recent work presented in (Kang et al., 2015) tackled disjoint raster aggregations. The authors tried to minimize the processing time by estimating the results instead of calculating exacts results that require heavy computations. No work has been proposed to improve overlapping aggregation processing time. In the era of big data, improving the processing time of such aggregations is crucial.

In this chapter, different GPGPU strategies for the implementation of the overlapping raster aggregation described above are provided and compared. It is shown that GPGPUs provide a very large improvement in terms of performance. Unlike the work proposed in (Kang et al., 2015), the methods proposed in our chapter produce exact aggregation results.

4.2 Formulation of the problem

This subsection presents the details of the problem formulation and the pseudo code for its resolution.

D = (R_1, …, R_N) is sequence of rasters R_i, and N is the number of rasters in D. All the rasters have the same size p × q in D.

$cell_{x,y}(R_i)$ is a cell in the raster R_i , and (x, y) are the coordinates of the cell in the raster.

Our problem is the calculation of the cell mean for each raster subsequence of size L in D.

The resolution can be presented in two steps.

Step1)

Calculate the cell mean for each $R_i \in D$:

$$\text{Mean}(R_i) = \frac{1}{p \times q} \sum_{x=1}^{p} \sum_{y=1}^{q} \text{cell}_{x,y}(R_i)$$

Once this first step is completed, the cell mean for each raster in D is obtained.

For instance, this result can correspond to the temperature mean of a studied region at each time t, as illustrated in Figure 4.3.

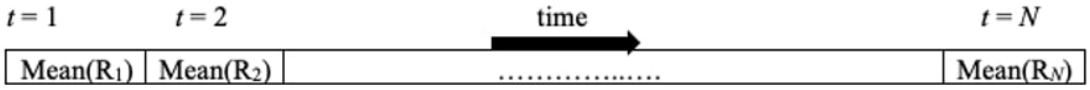

Figure 4.3. The cell means for each raster in D calculated by step 1.

Step 2)

In this second step, the average of these means is calculated for each subsequence s_u of size L in D:

$$\text{Mean}(s_u) = \frac{1}{L}\sum_{i=u}^{u+L-1} \text{Mean}(R_i)$$

Here is an illustrative example for these two steps. Let D be a dataset that contains 10000 rasters (N = 10000). The rasters represent the temperature of the same region over 10000 days. We will calculate all the average temperatures of this region for all raster subsequences of size 6 (L = 6). Our resolution consists of calculating the average temperature of the region for each period of 6 successive days.

Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10	…
26 28 27 26 27 27 25 28 28 28 26 28 27 28 27 27	24 26 27 26 25 26 25 25 26 25 26 27 26 25 25 26	27 28 27 27 26 28 26 27 27 27 28 28 28 27 26 27	25 27 27 27 26 26 26 26 27 25 26 27 28 27 26 27	25 27 26 25 26 27 27 26 27 26 27 26 26 27 26 27	25 26 27 26 25 26 24 25 26 25 25 27 25 27 25 25	25 28 27 26 27 24 25 28 28 24 27 26 26 28 29 27	25 25 26 26 27 26 25 27 26 26 26 25 26 26 25 25	26 25 27 26 29 27 28 28 28 29 26 25 25 28 26 27	29 28 29 31 29 30 28 30 30 29 29 30 28 28 29 29	
27.06	25.62	27.12	26.25	26.31	25.56	26.56	25.75	26.87	29.12	

…	Day 9991	Day 9992	Day 9993	Day 9994	Day 9995	Day 9996	Day 9997	Day 9998	Day 9999	Day 10000
…	34 33 34 32 35 32 33 31 32 33 32 34 34 33 34 33	32 32 32 30 32 33 31 32 32 33 30 33 31 32 32 34	33 32 32 32 33 29 32 34 32 34 33 32 31 32 33 31	30 29 23 30 29 31 30 32 34 33 32 31 30 31 32 29	30 31 31 33 30 29 34 32 31 30 32 34 33 31 32 32	33 30 32 32 32 33 32 35 34 32 30 33 33 33 30 34	31 31 31 30 30 31 31 32 31 31 32 34 30 30 32 31	30 30 33 31 32 31 32 31 35 32 33 33 33 32 32 32	33 30 32 32 32 33 31 35 31 32 30 21 32 33 32 20	33 32 32 32 32 30 32 30 34 32 34 32 33 33 30 30
	33.06	31.93	32.19	30.37	31.56	32.37	31.13	32	30.56	31.94

Figure 4.4. Example of raster set D.

For instance, some rasters are provided in Figure 4.4. In step 1, the mean of each raster R in D is computed. The result for each raster is shown below in Fig. 8. The results can be represented by one 1- dimensional array. In step 2, an average for each subsequence s_u of length 6 is calculated (see Figure 4.5).

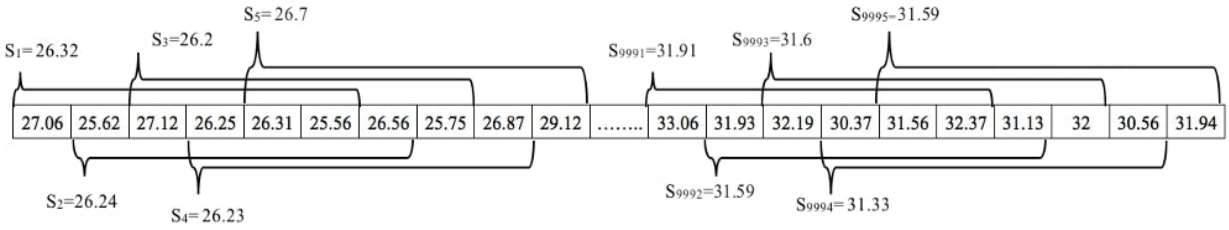

Figure 4.5. Example of some sequences s_u of length 6.

This chapter has chosen to split the resolution of the problem into two steps in order to make the application of a value selection between the first and the second steps possible. For example, users can decide to select only rasters having a mean greater than a defined threshold value after step 1. In this case, the aggregation of the second step will be performed only on these selected rasters mean values.

The sequential pseudo code for the two steps is as follow:

Step 1:

```
sum :=0
for j := 1 to  p {
        for k :=1 to q {
                sum:= sum+ cell_{j,k}(R_i)
      {
}
Mean_R_i := sum / (p*q)
```

Step 2 :

```
sum :=0
for j := u to u+L-1{
        sum := sum+A[j]
}
Mean_S_u :=sum/L
```

4.3 GPGPU-based method for the problem resolution

This section shows the different approaches tested to implement the two steps of the problem resolution on the GPGPU architecture.

4.3.1 Computing the mean of each raster (step 1)

The first step in our resolution is the computation of the mean of each raster in the dataset. This operation refers to the reduction of the raster, which is a 2D array, to a single value (the mean) using the average operator. In the literature, the reduction algorithms extract a single value from an array of values using a binary associative operator (Martin et al., 2012).

Given

- A binary associative operator with identity I.
- A set of n elements [a_0, a_1, ..., a_{n-1}].

Reduce ($\oplus$, s) returns a single value by computing the following: a_0, $\oplus$ a_1 $\oplus$,..., $\oplus$ a_{n-1}.

The single value could be the sum, the maximal value, the minimal value, etc. For example, take the following:

Reduce (+, [3 1 7 0 4 1 6 3]) = 25.

Reduction algorithms are one of the main parallel primitives. They are common in parallel processing and are used as building blocks for many algorithms. To compute the mean of each raster in the dataset, two approaches have been tested. The first one computes the mean of each raster in the dataset one by one using an unsegmented reduction. The second method computes the mean of each raster in the dataset at once. In this case, a segmented reduction-based approach is used.

To be more efficient, our algorithms have been implemented by adopting an existing parallel primitive approach since the main blocks of our method are already optimized and efficiently implemented by NVIDA's Libraries for instance Thrust (https://thrust.github.io) and CUB (https://nvlabs.github.io/cub).As detailed in the state of art, Thrust and CUB provides a set of fundamental parallel algorithms such as reduction and sort that are implemented in an optimized and efficient way. Hence, Thrust, for instance, provides to the developers the possibility to describe their computations in a high-level of abstraction which enhances the programmer productivity while enabling a high performance. Thrust is based on the Standard Template Library (STL) and

it provides a full interoperability with technologies such as: C++, CUDA, open MP, TPP and now it is a part of CUDA toolkit. The main characteristic of Thrust is that one can run the same code in a parallel or in a serial manner by just changing some few parameters in the code.
CUB is the fastest library since it is optimized only for CUDA. It provides also a collection of parallel primitive algorithms that are implemented in a very sophisticated way. Thrust's CUDA backend is built on top of CUB.

4.3.1.1 Unsegmented reduction-based approach

Technically speaking, the main step for implementing an algorithm on the GPGPU is to transfer data from the host (CPU) to the device (GPU). In our case, to compute the mean of each raster with a GPU, we have to transfer all the rasters of our dataset to the GPU. The use of the unsegmented reduction consists of transferring each raster of our dataset one by one from the CPU to the GPU and applying the reduction using the mean operator to the raster. At the end, the results are transferred back from the GPU to the CPU.
In the literature, many works have been proposed to improve the reduction algorithm processing time (Harris, 2007a). This interest comes from the fact that the reduction algorithms are used as one of the main components of many sophisticated programs. The fast reduction algorithm used in our work is presented in (Harris, 2007a). By viewing the raster as a vector in memory, the mean of this vector can be computed by applying the reduction algorithm with the sum operation.
In the algorithm, each thread will perform the sum operation of two interleaved pair values and store the result in the memory. At each pass, the threads use the intermediate results stored by the other threads. Since each thread takes two entries and produces one output, each step uses half the number of threads of the previous step. In the example presented in Figure. 4.6, the reduction with the sum operator is performed on the raster R_1 (day 1 in Figure. 4.4). To reduce R_1, it is needed to run 8 threads (Th_0,…,Th_1) for the first pass. Each thread processes the interleaved pairs of R1 and performs the reduction based on the sum operator.

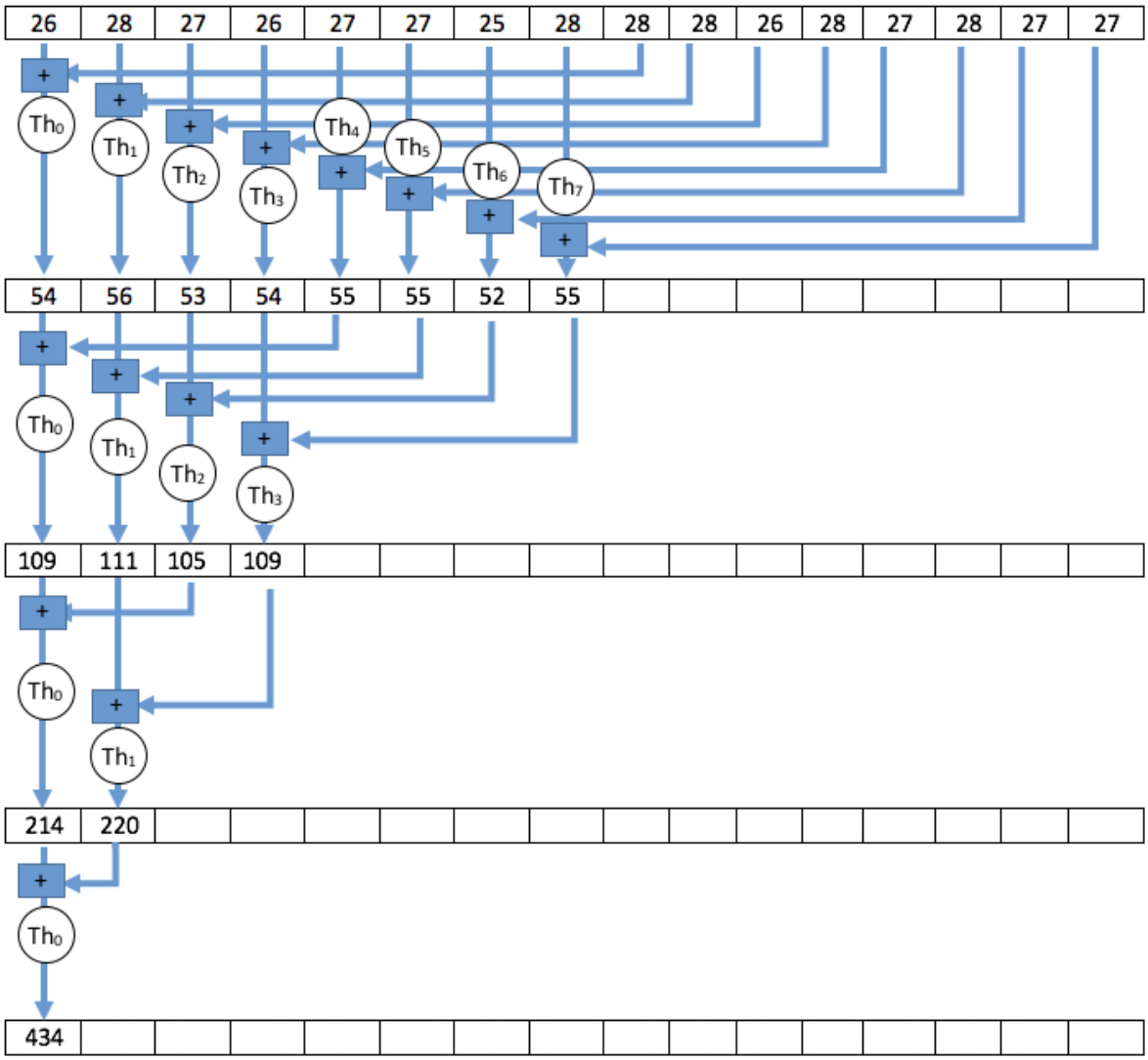

Figure 4.6. Illustrative example of the reduction algorithm (sum operator) used for raster R_1.

The unsegmented reduction algorithm is implemented and optimized in existing GPGPU parallel libraries. Thrust (https://thrust.github.io) and CUB (https://nvlabs.github.io/cub) provide high performance implementations of this algorithm. The use of these implementations has been tested. The unsegmented method can be time consuming in our case because, for each raster, one must wait for the completion of the reduction operation of the previous raster. Consequently, the mean is computed in parallel for all the cells of the same raster but is computed sequentially for the raster

dataset. In this approach, the rasters are transferred one by one by applying a time-consuming back and forth process between the host and device.

4.3.1.2 Segmentation-based approach

A segmentation-based method that overcomes the drawback of the unsegmented approach has been tested. The segmented reduction is also a building block for many algorithms. It consists of transferring the whole dataset from the CPU to the GPU in one single step. All the rasters are contained in one single array that concatenates all the vectors representing the rasters. Each raster forms a segment in this array. One unique key is assigned to each segment therefore to each raster. If there are *N* rasters, there will be *N* fixed size segments. This approach allows us to transfer the whole array to the GPU (if the GPU memory size is sufficient – otherwise several steps are required) and call the kernel just once, which performs the segmented reduction on the whole array based on the keys assigned to each segment. As an output, one 1D array containing all the mean values of the rasters is obtained. The reduction is performed on all the rasters at once. An illustrative Example is presented below.

Let R_1, R_2 and R_3 be 3 rasters of size 2×2 (Figure 4.7).

4	8
10	12

6	24
36	14

5	12
8	12

Figure 4.7. A set of three rasters of size 4.

Rasters are aligned and stored in one array as follow (Figure 4.8):

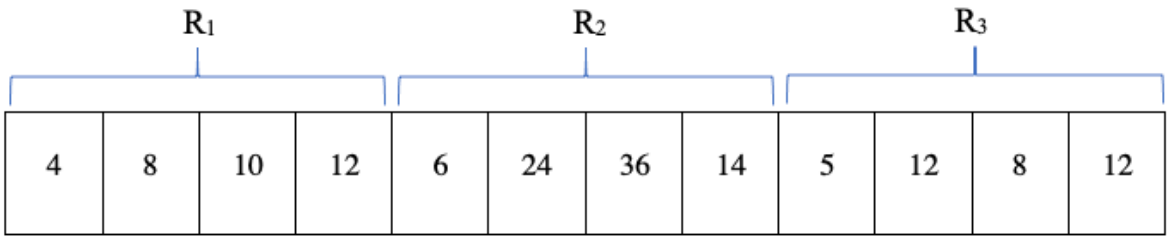

Figure 4.8. Aligning raster cells.

The system will assign to each data in the same raster the same key after that the mean of each raster is computed based on their keys (Figure 4.9).

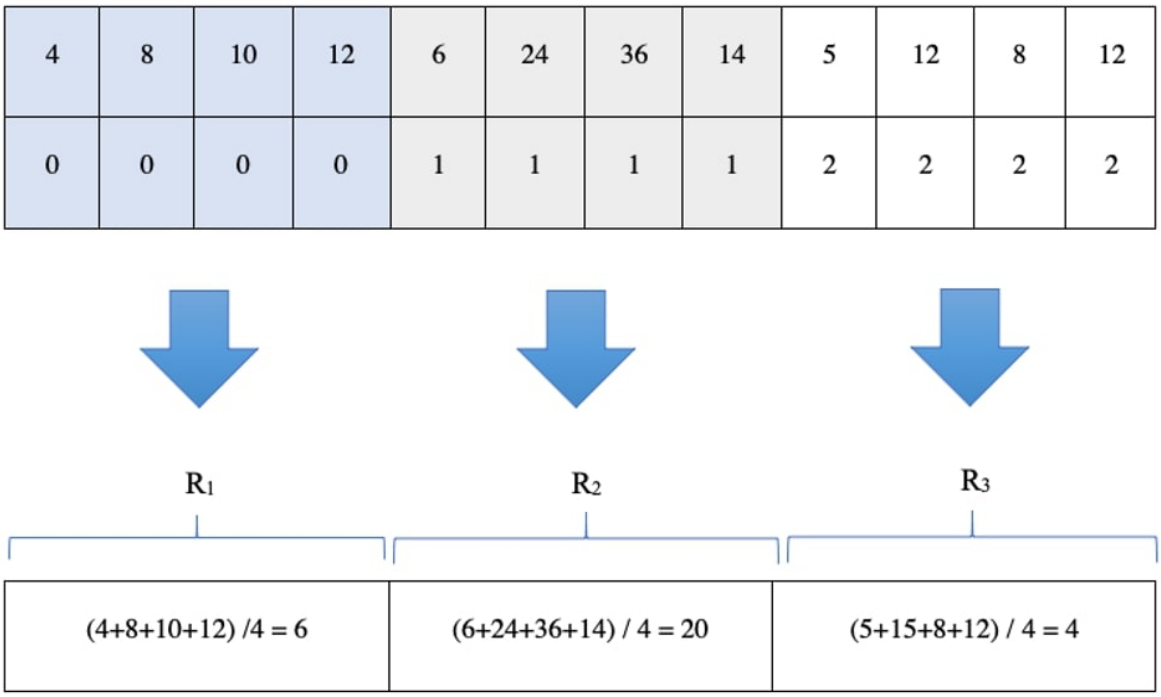

Figure 4.9. Segmented mean computations of rasters.

As for the unsegmented reduction, the algorithm is implemented and optimized by the Thrust and CUB libraries. In the segmented case, the segmented reduction function of CUB or Thrust is called just once in order to compute the mean of all the rasters in our dataset, unlike the unsegmented

approach in which the reduction function is called N times (N is the number of rasters in our dataset).

4.3.2 Computing the average of each sequence of size L (step 2)

After computing the mean of each raster in our dataset D, the next step consists of computing the mean of the subsequences s_u of length L. This step computes the mean over each subsequence s_u in D. This section presents a different approach that can be used.

4.3.2.1 Method 1: straightforward approach

In the first approach, each thread is assigned to one subsequence to compute the average. Each thread computes the average of the subsequence s_u (Figure 4.10).

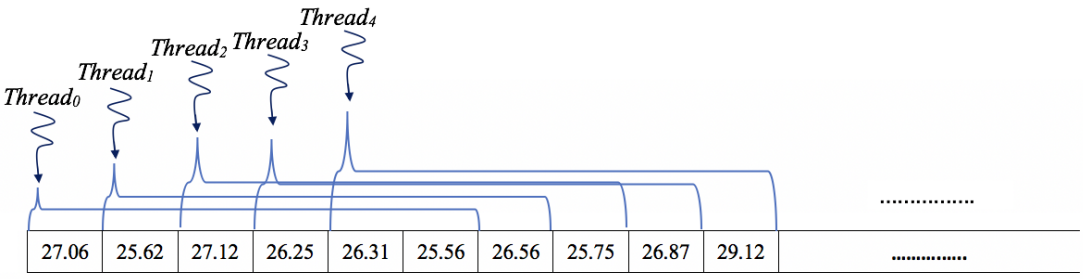

Figure 4.10. Single thread-based segmented reduction.

The limitations of this approach are twofold. First, all of the subsequences are processed in parallel, but the computation of the average is sequential in each subsequence since one thread computes sequentially all the operations to calculate the average of one sequence. It is possible to solve this problem by assigning a block of threads to each sequence and using the unsegmented reduction algorithm as described before. In this case, the average of subsequence s_u is computed by a block of threads that will cooperate together. Second, as the aggregations are overlapping, many threads compute the same calculation, which is redundant work. For instance, to compute the mean of s2, thread2 will compute the sum (A[2]+ A[3]+ A[4]+ A[5]+ A[6]+ A[7]), while the sum (A[1] + A[2] + A[3]+ A[4]+ A[5]+ A[6]) is computed by thread1 for the subsequence s1. Consequently, the elements in A will participate 6 times (the length of the subsequences) in the computations.

4.3.2.2 Method 2: Prefix Sum-based approach

To avoid the previous redundant computations, the Prefix Sum technique has been tested (Blelloch, 1997). Several algorithms have been proposed either in sequential or in parallel approaches for the implementation of this method. Our prefix sum is based on the efficient implementation presented in (Harris, 2007b).

Using this technique allows us to reuse the results of previous addition operations and as a result avoid extra redundant computations. There are two versions of Prefix Sum: the inclusive and the exclusive. The exclusive Prefix Sum operation takes a binary associative operator $\oplus$ and n elements $[a_1, a_2, ..., a_n]$ as parameters, and returns: $[I, a_1, (a_1 \oplus a_2), ..., (a_1 \oplus a_0 \oplus ... \oplus a_n)]$, such that I is the identity element of the associative operator

The inclusive Prefix-sum operation takes a binary associative operator $\oplus$ and n elements $[a_1, a_2, ..., a_n]$ as parameters, and returns: $[a_1, (a_1 \oplus a_2), ..., (a_1 \oplus a_0 \oplus ... \oplus a_n)]$. In our method, the exclusive Prefix Sum has been used. For example, if $\oplus$ is the addition, then the Exclusive Prefix Sum operation on the array [3 1 7 0 4 1 6 3] will return the array [0 3 4 11 11 15 16 22], which is denoted by *PS*. The average of the subsequences s_u in *S* is computed as follows:

$$\textbf{Mean}(\ S_u\ ) = (PS[u+L-1] - PS[u])/L$$

It is shown below how to use the Prefix Sum to compute the mean over each subsequence in the previous example. As input, the following array has been used.

27	25	27	26	26	25	26	25	26	29	…….

The prefix sum (i.e., the array denoted by *PS*) of the previous array is as follows.

0	27	52	79	105	131	156	182	207	233	…….

Consequently, now it is easy to get the mean over each subsequence of length *L* in the input, as shown in Figure 4.11. It simply needed to subtract the prefix sum from the shifted prefix sum with *L* elements to the right (in our example, *L*=6). Hence, the average of all subsequences s_u is obtained

in one pass. For the Prefix Sum implementation, we have used the efficient CUB and Thrust libraries.

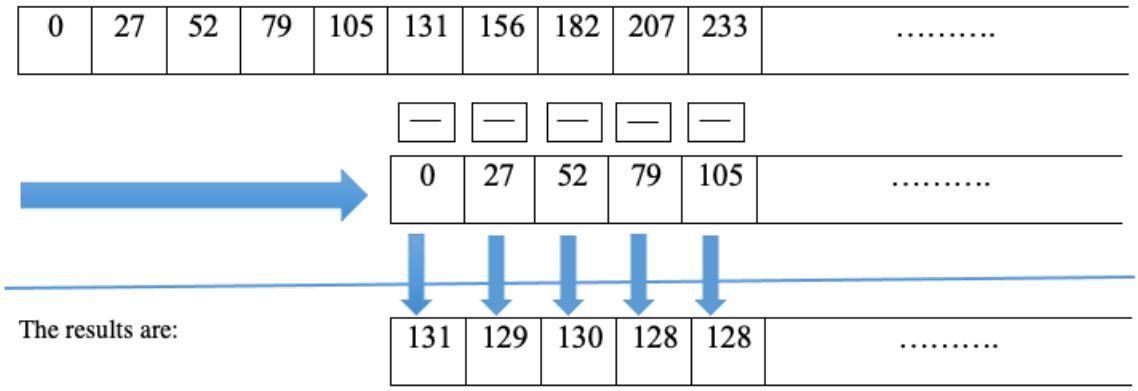

Figure 4.11. Illustrative example for Prefix Sum.

4.4 Experiments and results

To test the performance of the parallel implementation methods, experiments have been run on a Tesla K20C GPU card with 5 Gb of global memory and 2496 NVIDIA CUDA cores, which allowed us to test our method on large datasets and thus assess the stability of our method. The sequential version is run on the host CPU Intel(R) Core(TM) i7-2600K running at 3.40 GHz with 16 Gb of RAM. Concerning the data, the experiments are conducted using a temperature public dataset.

The experiments are designed in a way to test our method on different datasets with different structures and features, and also to study the stability of the performance of the proposed method with respect to the length of subsequences. To this end, we first created four datasets with the same size. In each dataset, the size of the generated rasters is different, starting with rasters with small size (32×32) to rasters with larger size (100×100). Furthermore, the number of generated rasters was also modified in each dataset. Thus, our method has been tested on different datasets, in order to study the impact of each of these parameters on the execution time and to have a clear idea about the performance. Second, our method was run multiple times with different length of subsequences.

4.4.1 Dataset

The public dataset provided by the US National Oceanic and Atmospheric Administration has been used (Diamond et al., 2013). This dataset provides a large amount of climate and historical weather data, including the following: air temperature, humidity, precipitation, etc. The data are available in various temporal acquisition rates: monthly, daily, hourly and sub-hourly (5-minute). This data covers many weather stations in the USA. In our work, hourly data for the Barrow station temperature have been used. At every 60 minutes, there is the min, max and the mean of the temperature of this station. Rasters have been simulated for the local studied region from this station. It has been assumed that the temperature of the region has a Gaussian distribution. The normal distribution is characterized by two parameters: the mean and the standard deviation. The mean is provided by our dataset, and the standard deviation ST has been estimated using the min and the max values provided by the dataset. The simple method to estimate the standard deviations ST is the range rule of thumb (Honzo et al., 2005). The approximation of ST is calculated as follows:

$$ST \approx \frac{max - min}{4}$$

In practice, the estimation of *ST* using the range rule of thumb is not sufficient when the *n* is extremely small or large (Honzo et al., 2005). This estimation was improved by (Honzo et al., 2005) to deal with this size problem:

$$ST \approx \begin{cases} \frac{1}{\sqrt{12}}\left[(max - min)^2 + \frac{(max - 2m + \min)^2}{4}\right]^{1/2} & n < 15 \\ \frac{max - min}{4} & 15 < n < 70 \\ \frac{max - min}{6} & n > 70 \end{cases}$$

In our case, large rasters have been generated. Consequently, the estimation of our standard deviation is calculated using the third case (n >70). Thus, the required parameters have been obtained in order to generate raster data from the raw data using the normal distribution.

Four datasets were created with the same size. In each dataset, the size of the generated rasters is different, starting with rasters with small size (32×32) to rasters with larger size (100×100). Furthermore, the number of generated rasters was also modified in each dataset. Thus, our method

has been tested on different datasets with different structures in order to show the impact of each of these parameters on the execution time.

4.4.2 Results

This subsection shows the results of our experiments based on the best approaches presented in our work. Both the GPU-based implementation (based on the CUB library) and the CPU-based implementation (based on the Thrust library) have been tested for Prefix Sum and reduction operations. The GPU-based methods on CUB and Thrust have been implemented, but since CUB is faster, only the results of our GPU-based implementation on CUB are presented in this chapter. Concerning the CPU-based implementations, two CPU versions have been implemented, the first one uses the pure C++ and the second one uses Thrust. Since a comparison of our GPU method with the fastest version of the CPU is needed, only the Thrust-based CPU version is presented. It is faster than our pure C++ based implementation. This can be done by changing the Thrust execution policy.

Different values have been tested for three main parameters: the size of the rasters, the number of rasters and the length of the subsequences. The goal was to identify their impacts on the computing time.

To test our work, large datasets have been generated to saturate our GPU memory (5 Gb). In the following, the first two subsections concern the experiments for each step of our method separately without including the time for data transfer between the CPU and the GPU. Hence, the results presented in Table 4.1 and Table 4.2, concern only the computation time of each step. Our two-fold objective in these subsections: first, to have a clear idea about the execution time required by each step in our method run on the GPU, and second to highlight the computation power of the GPU by excluding the data transfer cost. The last subsection concerns the experiments for the whole method (GPU version) including the time for the data transfer between the CPU and the GPU. Thus, Table 4.3 shows the performance of the whole method.

4.4.2.1 Mean computation for each raster (step 1)

In the experiments of this step, four different datasets have been generated with the same global Gb size by changing the raster size and the number of rasters. Table 1 compares the computing time for the raster mean calculation (step 1).

As shown in Table 1, the GPU is able to perform step 1 faster than the CPU for all the generated datasets. The results show clearly that our GPU implementation is 89 to 244 times faster than the CPU implementation. The GPU performs better when the amount of work is large, which is the case for the Dataset 1 in which the raster size is large.

However, the GPU acceleration decreases when the size of the rasters becomes small which is the case for the fourth dataset.

		Raster size	**Number of rasters**	**CPU (*ms*)**	**GPU (*ms*)**	**Acceleration**
Step 1	Dataset 1	100×100	120000	225160	922	244.20
	Dataset 2	96×96	130208	223316	965	231.41
	Dataset 3	64×64	292968	223102	874	258.51
	Dataset 4	32×32	1171875	224863	2499	89.98

Table 4.1. Computing time of the raster mean (step 1).

4.4.2.2 Computing the mean of subsequences s_u (step 2)

The test of the performance of our method is based on the same datasets used in the previous step. Table 4.2 shows that, with the method based on the Prefix Sum, the GPU processing time is 7 to 11 times faster than for the CPU version.

Table 4.2 also shows that the processing time increases when the number of rasters increases, even if the same amount of data is processed. The GPU always perform better when it has more work to do as in the case of Dataset 4.

		Raster size	Number of rasters	CPU (*ms*)	GPU (*ms*)	Acceleration
Step 2	Dataset 1	100×100	120000	44.66	5.8	7.7
	Dataset 2	96×96	130208	46	5.94	7.74
	Dataset 3	64×64	292968	104.16	10.43	9.98
	Dataset 4	32×32	1171875	414.81	34.59	11.99

Table 4.2. Computing the average of the subsequences s_u of length 100.

4.4.2.3 The execution time for the whole method including the data transfer

In this subsection, the results for the whole method (step1 and step 2) are presented for the same datasets used before. Furthermore, the time required for data transfer between the CPU and the GPU is included to have a complete evaluation of our method.

As we can see in the table below (Table 4.3), our method is faster than the CPU version and a very good speedup is obtained for all the generated datasets. Transferring data between the CPU and the GPU is time consuming; the main reason why we have adopted the segmented approach is to avoid multiple transfers between these devices.

	Raster size	Number of rasters	CPU (*ms*)	GPU (*ms*)	Acceleration
Dataset 1	100×100	120000	225204.66	3927.8	57.33
Dataset 2	96×96	130208	223362	3970.94	56.24
Dataset 3	64×64	292968	223206.16	3884.43	57.46
Dataset 4	32×32	1171875	225277.81	5533.59	40.71

Table 4.3. The execution time for the whole method including data transfer between the CPU and the GPU.

Furthermore, the effect of the length of the subsequences s_u on the runtime processing of the GPU has been analyzed. The performance has been tested for different values of s_u using dataset 1. Figure 4.12 shows that there is almost no impact of the length of subsequences s_u on the processing time of our method using the GPU. Thus, computations for subsequences of length 50 or 400 are

almost the same in terms of computing time. This result is an advantage of the Prefix Sum technique.

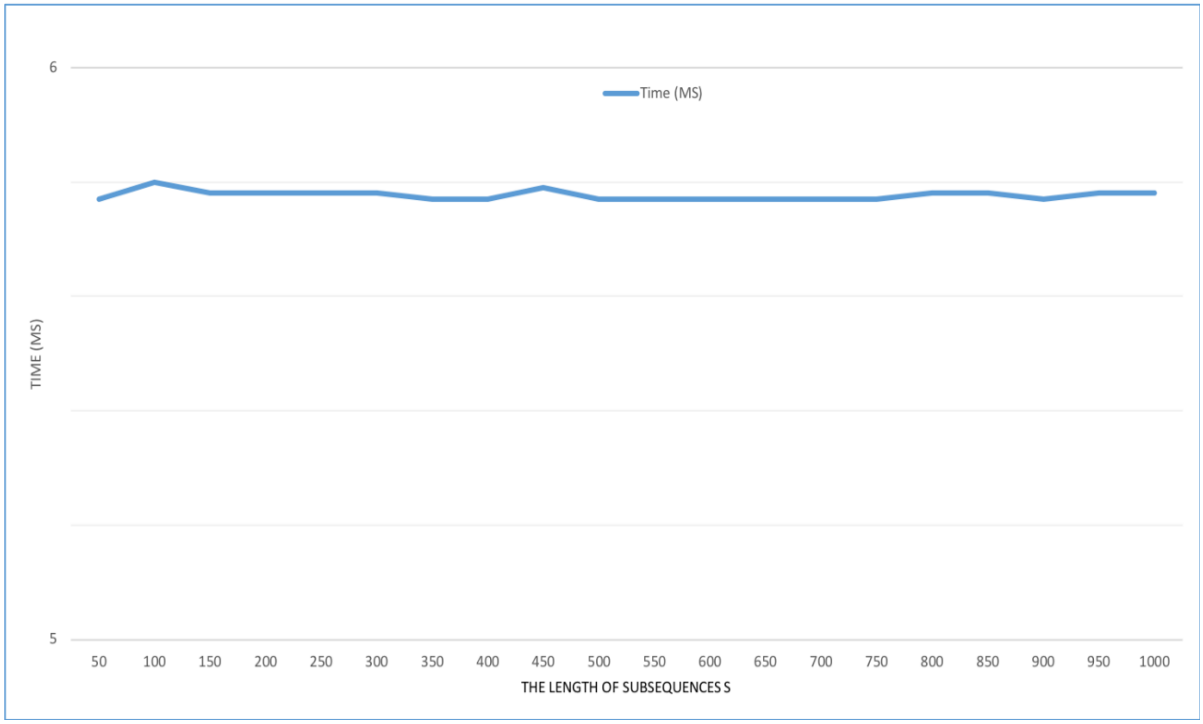

Figure 4.12. Impact of subsequence length on time processing.

4.5 Conclusions

In this work, a parallel approach has been presented for the overlapping aggregation process of raster sequences. This process is based on the map algebra principle. This type of overlapping aggregation can be found in many environmental data analyses. The result of this chapter highlights that the execution of such aggregations can significantly benefit from GPGPU processing. Our results show that it is possible to accelerate the execution more than 200 times the corresponding CPU execution, which clearly proves the potential for big data processing. Our methods are generic and can be used for more general cases and for other types of data, such as array and vector aggregation, since the cells coordinates are not required for computations and no assumptions are made on data value to perform the computations.

In the experiments that are illustrated in this chapter, the overlapping raster number between s_u and s_{u+1} is equal to L-1, and thus, all the subsequences are calculated. The GPGPU-based approach

tested in this chapter is general and can also be applied in the case of aggregations with a larger interleave between aggregated subsequences. For some applications in which all the subsequence aggregations are not needed, this parameter can be changed, for example, by calculating only the subsequences (R_1, …, R_L), (R_6, …, R_{L+5}), (R_{11}, …, R_{L+10}), etc., with L>5. In this type of case, the operational overlap is smaller than in the aggregations presented in this chapter.

As mentioned before, overlapping aggregations is a traditional process. It is used as a prior step for many spatial data queries and environmental data analyses. Hence, our method can be used as a parallel primitive for these applications to accelerate their processing.

Chapter 5

Selection of Rasters based on a User-Defined Condition: A Sequential Approach

Results obtained in this chapter have been published at Advances in Intelligent Systems and Computing 893, 190-201., Springer.

5.1 Context and motivation

This contribution proposes a new technique to improve the execution time of the selection of rasters in a raster sequence representing the evolution of a pheromone over time using only the CPU. The processing of the rasters consists in three main steps (shown in Figure 5.1). The different values of the raster cells are represented by colors. In the step (a), the user chooses a period of interest. More precisely, he/she selects a temporal raster (sub)sequence of interest in the large sequence of rasters. In the step (b), the user defines the geographical region to analyze in the sequence of rasters selected in step (a). This geographical region to analyze is the same for all these rasters. In the step (c), the system automatically selects every raster that satisfies a user-defined condition.

We illustrate this process on an example. A user would like to analyze a sequence of rasters representing the evolution of temperatures. For example, he/she wants to determine the set of rasters having low temperatures in order to:

- Study more precisely these cases and their possible local causes. It is a typical case of climate change analysis.
- Or analyze the impact of these temperatures on crops in agriculture in the context of farm decision support.

Thus, he/she manually chooses the period to be analyzed in the whole sequence (step (a)). Second, he/she manually chooses a geographical region of interest for his/her study (step (b)). Third, in the step (c), the user would like to automatically select every raster in which the average temperature of the region of interest is lower than a user-defined threshold (e.g., $\leq$ 10 C°). Consequently, the result is the set of the rasters that satisfy this condition.

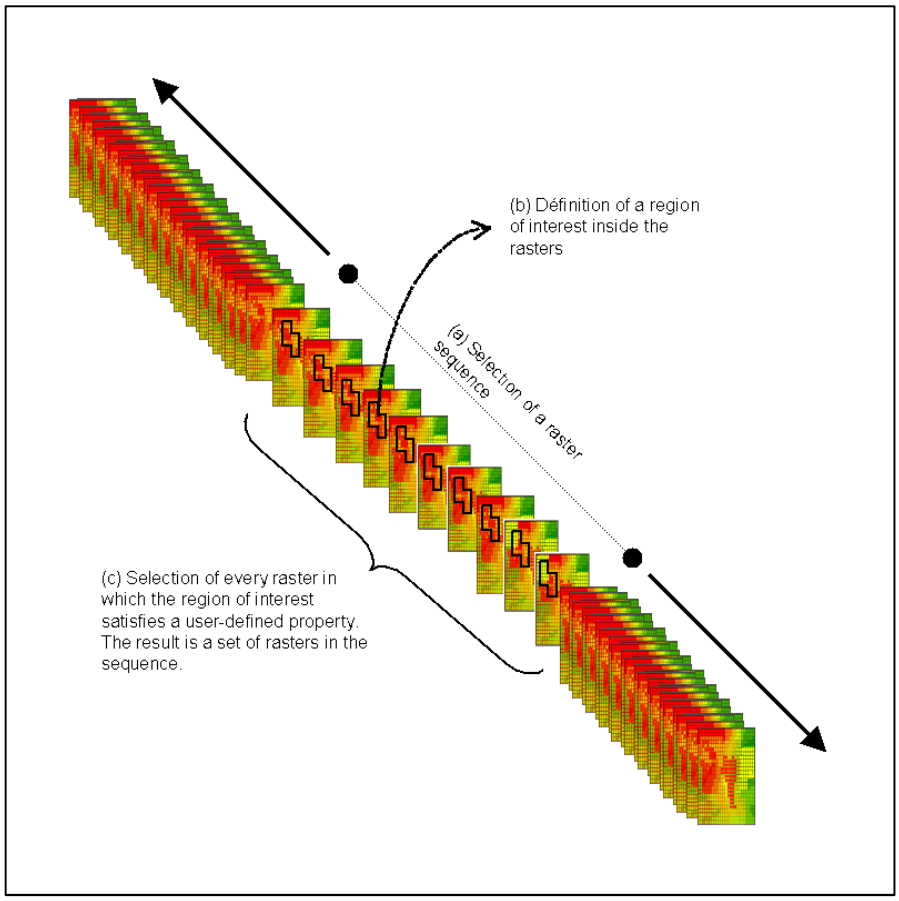

Figure 5.1. Description of the raster data process.

Our goal is to improve the execution time of the step (c) To illustrate, we use the following user-defined selection criterion on temperature rasters: the calculation of the average temperature for every raster and the selection of the temperature < 10C°. A naive algorithm for the step (c) is shown in Figure 5.2:

```
S := the whole sequence of rasters
A := the (sub)sequence of rasters chosen in S by the user
b := the region of interest chosen by the user
Result := { }

for every raster Ri in A
{
avg := the cell average for the region b in Ri
```

```
if avg < 10 then Result := Result ∪ { Ri }
}
```

Figure 5.2. Naive algorithm for step (c).

5.2 Raster data process improvement

Based on the Naïve approach showed in Figure 5.2, we propose an improvement of the step (c). The intuition behind this algorithm is to try to reject a raster that does not satisfy the user-condition (i.e., the cell value average must be lesser then the user's threshold) as soon as possible to avoid useless computation. The proposed improvement technique can improve the computation when the user's threshold is low (compared to the raster cell values). In this technique, the cell must contain only numerical positive value – consequently, a uniform translation or normalization must be used if the rasters do not comply with this constraint.

The calculation of the average is computed for each raster (in the region of interest). The average computation consists in calculating the sum of cell values for each raster. In our algorithm, we stop the sum computation as soon as possible, when we are sure that this sum becomes superior to the threshold value multiplied by the cell number of the region of interest.

We propose to sort the cell values in the region of interest in a descending order, for the average computation. In that case, the threshold is reached faster for the rasters that do not satisfy the condition. Unfortunately, the time complexity of a sort, i.e., O(n log n) for a quick sort, is higher than the sum computation, i.e., O(n). Consequently, we propose the following stages:

1. We propose to sort the value of the region of interest only for some rasters, e.g., compute a sort every 200 rasters, in sorting the cell values of the region of interest only for the rasters R_i, R_{i+200}, R_{i+400}, etc. Each one of these sorts produces a cell ordering.
2. We propose to use the cell ordering of the sorted rasters, for computing the sums for the other rasters. For example, the sort in R_i produces a cell ordering. This cell ordering will be used for computing the sum for each raster from R_i to R_{i+199}. The cell ordering determined by the sort of R_{i+200} will be used for each raster from R_{i+200} to R_{i+399}, etc.

The intuition behind this method is that in many phenomena the spatial distribution of values evaluates rather slowly over time. In the case of temperature rasters produced every 5 minutes, the highest values will often be on the same geographical part of the rasters for several tens of minutes or several hours. The frequency of the sort computation can be adapted to the nature of the data (e.g., sorting every 10 rasters, 50 rasters, 100 rasters, 200 rasters, etc.). This new version of the

algorithm is shown in Figure 5.2. In Figure 5.3, Ord is an array that corresponds to a mapping: Ord(1) is equal to the cell number (#) in b that has the highest value; Ord(m) is equal to the cell number (#) in b that has the lowest value. In Figure 5.3, the user-defined condition is "< threshold", but our approach can be used in the case of a condition "> threshold". In this latter case, the raster is accepted as soon as the threshold is reached (instead of being rejected when the threshold is reached as in Figure 5.3).

```
S := the whole sequence of rasters
A := the (sub)sequence of rasters chosen in S by the user
b := the region of interest chosen by the user
m := the number of cells in b
begin := the number (#) of the first raster in S
end := the number (#) of the last raster in S
it := the interleave between two consecutive rasters on which a sort is calculated
th := the user-defined threshold

Result := { }
maxsum := th * m
for i :=  begin to end step it
{
Sort the cell values of the region b in Ri in descending order
and produce the corresponding cell ordering Ord
for j := i to i+it
{
        if j > end  then { process completed ; stop } else
        {
                su := 0
                reject := false
                for k:= 1 to m {
                        v := the value in the cell Ord(k) in Rj
                        su := su + v
                        if su >= maxsum then
```

```
                        { reject := true ; break}
                }
                if reject is false then Result := Result ∪ { Rj }
        }
    }
}
```

Figure 5.3. Improved algorithm for step (c).

Several constraints must be satisfied in order to guarantee that this algorithm provides better performances in terms of execution time, for example, a low user-defined threshold or a spatial distribution of cell values sufficiently large in every raster to justify the interest of the sorting operation.

In practice, performance improvement depends on two things:

1) The size of the time window which must not be too large and also not too small. If it is too large the precision of sorting (reordering) the cells of the raster that falls in the same time window as the real sorted raster will drop. Hence the performance will drop also (hence the sort becomes useless). If the time window size is too small, the number of the sorted rasters will increase and since it is expensive, it will slow down the execution time).

2) The type of data (the temperature, pressure, etc.). Less the data is evolving the more the sorting is accurate.

5.3 Experiments and results

5.3.1 Dataset

We simulated a dataset related to temperatures using the same technique and the same source data as the one presented in Chapter 4. We have created 3 data sets having 3 different raster sizes; each data set contains 1420 rasters. We have avoided negative value as it a constraint of our approach. In our tests, the user-defined region of interest is the whole raster. In the produced data sets, we have one raster every day for four years. The tests have been applied on all these rasters – these rasters constitute the sequence A of rasters to analyze.

5.3.2 Results

In this subsection, we show the results of our experiments performed on the generated data. Different raster sizes have been tested. For each experiment, we test the naive algorithm and the improved version on the same data set. In our experiments, we also evaluate the impact of the main parameters on the execution time of our algorithm, for instance, the threshold and the interleave between the sorted rasters. To do this, we have chosen different thresholds and interleaves and run our algorithm using these different value parameters. Concerning the sort algorithm, we used a quick sort.

The impact of the threshold on the performance

Tables 5.1, 5.2 and 5.3 compare the computing time for the naive and the improved algorithm for the three data sets for different threshold values. Table 1 shows that the improved algorithm is faster than the naive one, especially when the threshold is not too low and not too high. The best performance is with th=40; our algorithm I is faster than the naive one with 3.07 seconds less for time execution. Whereas when the threshold is smaller, we obtain less performance (the case of th=30).

	Threshold *th*=30	**Threshold *th*=40**	**Threshold *th*=45**	**Threshold *th*=46**	**Threshold *th*=50**
Naive Algorithm	8.5(s)	13.17(s)	12.30(s)	14(s)	14.91(s)
Improved Algorithm	7.9(s)	10.1(s)	11.9(s)	12.9(s)	13(s)

Table 5.1. Dataset 1: Size of raster =100×100, contains 1420, Interleave =73.

In Table 5.2, our algorithm is faster than the naive one with 5 seconds less for execution time (*th*=42).

	Threshold *th*=39	**Threshold *th*=40**	**Threshold *th*=41**	**Threshold *th*=42**	**Threshold *th*=50**
Naive Algorithm	44.57(s)	45.30(s)	47.80(s)	49.4(s)	53.76(s)
Improved Algorithm	40.5(s)	42.28(s)	43.04(s)	44.47(s)	50.39(s)

Table 5.2. Dataset 2: Size of raster =200×200, contains 1420, Interleave =73.

	Threshold *th*=30	**Threshold *th*=40**	**Threshold *th*=42**	**Threshold *th*=50**	**Threshold *th*=70**
Naive Algorithm	36.81(s)	46.63(s)	48.53(s)	56.83(s)	67.03(s)
Improved Algorithm	32.57 (s)	44.57(s)	46.78(s)	53.55(s)	62.40(s)

Table 5.3. Dataset 3: Size of raster =240×240, contains 1420, Interleave =73.

As we can see in the Table 5.3, our algorithm is still faster than the naive one. More precisely, our algorithm is always faster than the naive one, whatever the value of the threshold. The user-defined threshold value has a direct impact on the performance of the improved algorithm.

The impact of the interleave size on the performance

The interleave value between the sorted rasters is important. It has also an impact on the performance of our algorithm. Choosing a low interleave implies sorting more rasters, which decreases the performance. In the other hand, choosing large interleave means sorting less rasters

which is good for the performance, but in the same time, many rasters that are in the same interleave will not follow the same behavior as the sorted raster.

In Table 5.4 we show how the interleave size influences the performance of our algorithm on the data set 1. As an example, we have tested three interleave sizes. As we can see in table 5.4, the best performance is obtained by choosing the size 73. The choice of the interleave value depends on the nature of data and the frequency of its production.

	Interleave =10	Interleave =20	Interleave =73
Naive Algorithm	13.17(s)	13.17(s)	13.17(s) (s)
Improved Algorithm	11.27(s)	10.53(s)	10.4

Table 5.4. The impact of the interleave on the performance (Data set 1), Threshold =40

Our algorithm shows interesting potential, it should be improved by using other faster sorting algorithms and also using raster data sets with significant variation of data in the same raster

Chapter 6

Selection of Rasters based on a User-Defined Condition: A GPGPU Approach

Results obtained in this chapter have been published at 11th International Conference on Computer Science and Information Technology, Paris, France, 8 p.

6.1 Context and motivation

In this chapter, we propose an improved scalable GPGPU based method, to handle the selection of large rasters from large temporal sequence dataset of rasters based on their average. This problem has been tackled using only the CPU computation in the previous chapter. To speed up the computation time, we decided to use the GPGPU since the underlined problem requires high massive computations since we need to compute the average of each large raster in a large dataset and check its average with a condition statement. In this work, we discuss in which cases our method can reach the best performance and achieving a good speedup.

In the rest of this chapter, we present only the parallel approach of the problem since the sequence versions (the naïve and the improved CPU version) have been presented in the previous chapter.

6.2 Raster Selection query: Data Parallel design

6.2.1 Naive approach

The naive approach consists on firstly aligning each raster in the dataset after that one thread is assigned to each raster; this later will be responsible for computing the raster average. As a result, the average of each raster in D will be computed in sequential (because each thread will go through each raster and computes the average of this later), but it will be performed in parallel for the dataset D.

The naïve approach will work efficiently if the size of the raster is small, but in our scenario, the rasters are acquired in a high precision and their size is very large. So, the average of each raster must be computed in a parallel by assigning a block of threads to each raster rather than one thread.

Hence, the average is computed in parallel for each raster and all over the dataset. Furthermore, we need to avoid the use of the “if" statement frequently, as there is a parallel computing. If we assign one thread for each raster, we need to use the “if” statement for each raster L × H times which is not efficient. Hence an improved approach is proposed to deal with these limitations.

6.2.2 Sorting-based method reminder and GPGPU-based approach

Our parallel improved approach consists on firstly aligning the rasters (we can use any alignment method, since we do not need the position of cells). Secondly, since we need to reject the raster if its average is greater than a certain threshold defined by the user, we decide to sort the rasters in the descending order (as shown in the previous chapter) in order to achieve the threshold as soon as possible and thus reject the raster in the early stages to avoid computing the average of the whole raster which is expensive in terms of time and computation. We need to outline that the sorting process is very expensive, thus sorting each raster is time consuming, which lead to a performance worse than computing the average of the whole raster, a case we try to avoid. To overcome this limitation, we propose to use the method shown in the previous chapter: instead of sorting all the rasters of the large dataset, we will sort only some of them. In our illustrative example, we divide our dataset D (sequence of rasters) into time windows (subsequence of rasters) of size T, e.g., T = 6 (Figure 1.6).

As indicated in chapter 5, we suppose that the rasters that are falling in a certain window of time have the same behavior in term of the values of cells. For example, if the rasters represent hourly temperature of a specific region, the temperature will not change or will make a slight change during a certain period of time (window) for instance 6 hours, so the rasters that fall in this time window have the same behavior. Thus, we need to sort only one raster from each window and reorder the others accordingly.

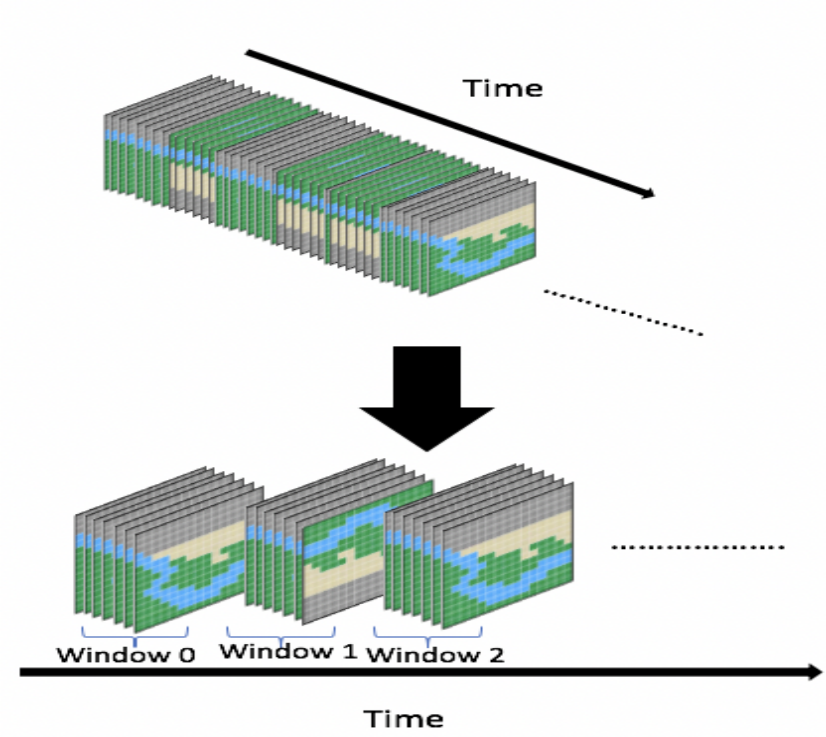

Figure 6.1. Dividing the dataset into time windows (subsequence of rasters).

The final step in our algorithms relay on dividing each raster to a set of tiles with fixed size (each tile has the same number of cells). The idea here consists in avoiding checking the average at each cell. As a result, the average is tested only at each tile. The example in the Figure 6.2 represents our method to compute the average of one raster. We need to outline that, instead of computing the average of the raster, we will only compute the sum of the raster and scaling the threshold by the size of the raster.

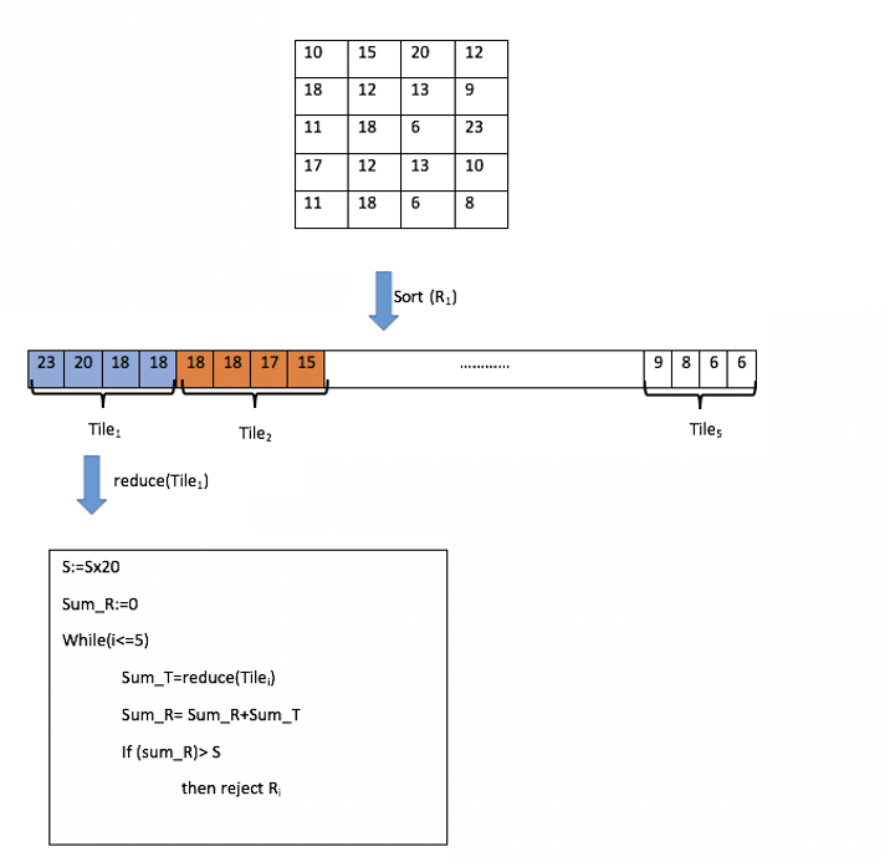

Figure 6.2. Illustration of our method for one raster(R_1).

We have reminded above the case of one raster. The same processing will be performed for each raster in parallel, which means that, for each raster, one thread will be assigned. This later will call the function which executes the previous processing. At the beginning, while we have developed our first algorithms on the GPU using pure CUDA implementations, we realized that our implementations cannot perform better than the parallel primitive implementations in term of productivity and efficiency. It is the reason why we decided to adopt a parallel primitive -based approach where it is possible to reduce implementation complexity and improve development productivity. Many of our algorithm's blocks are already optimized by NVIDIA's libraries such as Thrust and CUB. Our improved algorithm is based on two main blocks: the reduction and the sort, that are implemented efficiently in the libraries cited above. This allows us to have the best performance and hence achieving a higher speed up by using the improved implementations of the parallel primitives.

We summarize our method and provide an idea about its implementation on the GPU. Our method has two passes, hence two CUDA kernels are implemented: the first kernel consists on sorting the rasters of our dataset using the method cited above; our dataset D (sequence of rasters) is divided on a set of time windows (e.g., 6 days)

So, in each time window, only one raster is sorted by using the thrust primitive: **thrust::sort_by_key** or CUB primitive **cub::DeviceRadixSort::SortPairs**. Thus, we get the order of indices of the sorted raster. The other rasters that fall in the same windows of time will be reordered (sorted) according to the sorted raster using the gather transformation that we have implemented using the primitive: **thrust::gather**. The second kernel will be responsible of computing the average of rasters using our method described above. To do we can use the primitive: **thrust:: reduce** or **cub::DeviceReduce::Reduce**.

6.3 Experiments and results

To evaluate and test the performance of our algorithms, we ran experiments on a Tesla K20C GPU device with 5 Gb of global memory and 2496 NVIDIA CUDA cores. The sequential algorithm is run on the host CPU Intel(R) Core(TM) i7-2600K running at 3.40 GHz with 16 Gb of RAM. Concerning the data, we have used a temperature public dataset for our experiments.

6.3.1 Dataset

In our tests, we used the temperature data from public dataset provided by the US National Oceanic and Atmospheric Administration (Diamond et al., 2013); the same source as the one presented in the previous chapter. It provides several climate and weather data for many years which are produced from many weather stations. Furthermore, for this experiment, our studied region is composed from many stations that are near to each other for instance: Newton_5, Newton_8, Newton_11 and Watkinsville_5, etc. based on their geographical coordinates. Doing this allow us to have more variation of the values of data, hence the sorting step in our algorithm will have sense. If the values of the cells of the same rasters are closed to each other, sorting process does not have any sense.

To build our dataset, we have simulated temperature rasters for the local studied region using the same approach than the one presented in the previous chapters. Also, we have avoided negative value. For our experiments, we have generated two datasets, one dataset for a region

composed of 6 stations and another one composed of 3 stations. We have tested different values for the size of rasters, the number of rasters and also the threshold fixed by the user.

6.3.2 Results

In this subsection we report the results obtained by our experimentations on the simulated datasets. In the results shown in the tables below, we present the results only for the threshold equal to 30 and for a window time equal to 5 since they give us the best performance. As we can see in the table 1, we have tested the methods on different sizes of data. Table 6.1 shows that we obtained a significant speed up (speed up= 5 for rasters of size 1200000) compared to the sequential version and good speed up compared to the straightforward parallel approach (speed up =1.7 rasters of size 1200000) that does not use sorting phase. We have to mention that the straightforward parallel approach is not the naïve approach. In fact, it is the improved approach but without sorting phase.

	Raster Size ***36000***	**Raster Size** ***60000***	**Raster Size** ***120000***
Sequential Naïve Algorithm (ms)	504	841	1698
Improuved Sequential algorithm(ms)	460	737	1486
Straightforward parallel approach (ms)	241	421	571
Improuved parallel approach (ms)	158	235	339

Table 6.1. Dataset 1: 6 stations, 365 days, Windows time = 5 days, Threshold=30.

It is the same for Table 6.2 that shows the results for a sequence of rasters for 2 years (730 days). As we can observe, we have obtained a good speed up compared to all the methods but in the other hand we can see that we get less good speed up compared to the previous experiments on Dataset 1. This is due to two things: the first one is number of rasters which is greater than the previous one and hence we need to sort more rasters. The second one is the distribution of data.

In the Dataset 2, our studied region is formed from 3 stations which mean less variation of data inside our dataset and as a result sorting become less profitable compared to the Dataset 1.

	Raster Size ***18000***	**Raster Size** ***30000***	**Raster Size** ***60000***
Sequential Naïve Algorithm(ms)	541	852	1702
Improved Sequential algorithm(ms)	483	761	1529
Straightforward parallel approach(ms)	275	463	591
Impoved parallel approach (ms)	176	266	472

Table 6.2. Dataset 2: 3 stations, Windows time = 5, 730 days, Threshold=30.

6.4 Conclusion

To conclude, we have shown that we can speed up large scale geospatial queries by using the power of recent GPU cards. Using parallel primitives based on CUDA allowed us to further improvement our implementation and reducing coding complexity. Our experiments on our large-scale geospatial data have shown a good performance in term of time and we were able to obtain a significant speed up compared to CPU-based methods presented in the previous chapter.

Chapter 7

Selection of Raster Sequences based on a User-defined condition using GPGPU

Results obtained in this chapter are under submission.

7.1 Context and motivation

In previous chapters, we considered a selection criterion tested on each raster. This new chapter is dedicated to test a user-based condition on non-overlapping aggregated sequence of raster. This approach combines selection and aggregation operations. The workflow of the query is presented in Figure 7.1. In this later, we compute a spatiotemporal average of raster cells. A single numerical indicator is returned e.g., the average of the cells in all the raster in the studied temporal subsequence (a week). A user-based condition will be tested on the whole sequences of rasters.

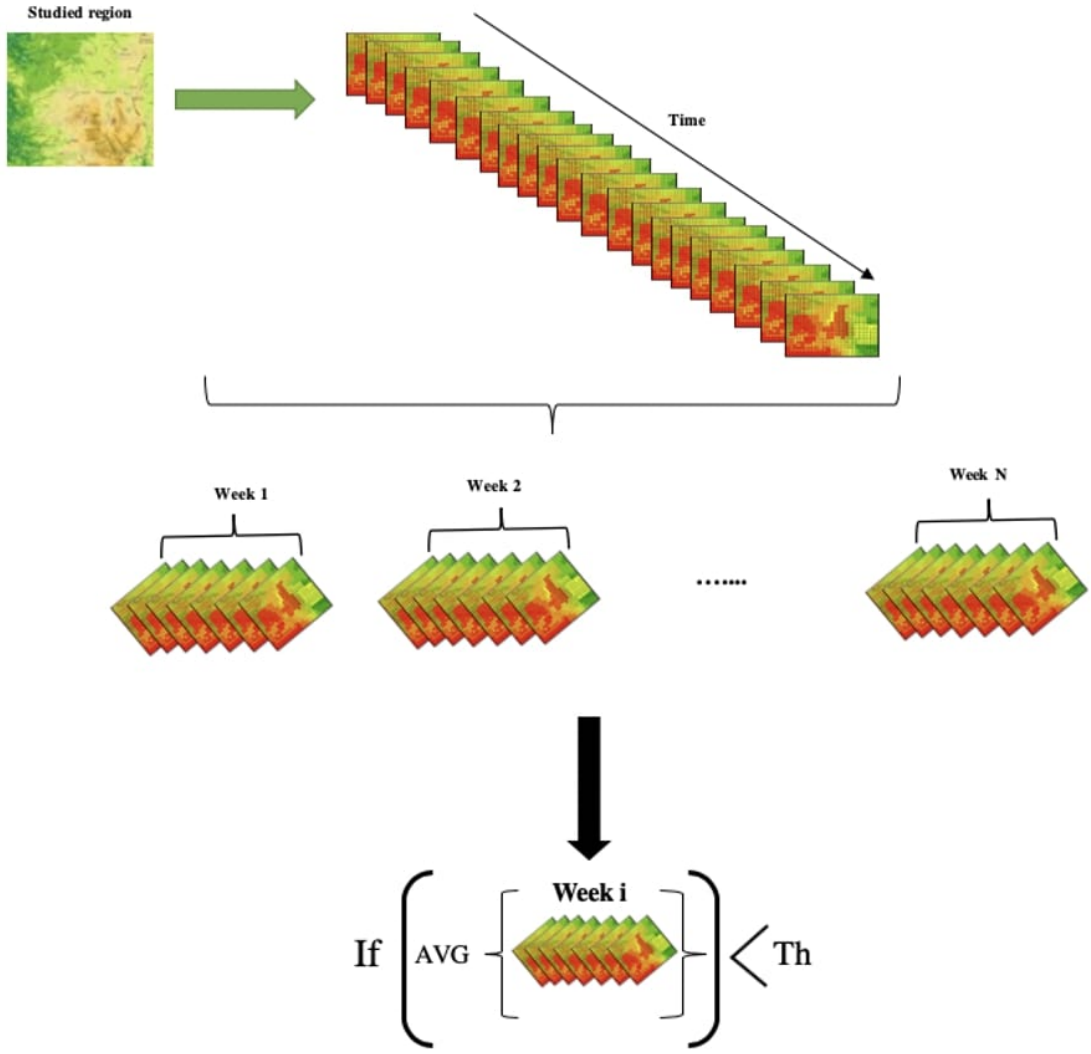

Figure 7.1. Overall framework for the query process.

Query definition and the sequential algorithm

Let D be the data set (R_1, …, R_N) of N rasters, where each R_i is 2D grid that has the size of p × q. All the rasters have the same size and correspond to the same geographical region.

Let $cell_{x,y}(R_i)$ be the cell in the raster R_i , and (x, y) be the coordinates of the cell in the raster.

The query relies on finding all the disjoint (i.e., non-overlapping) raster subsequences S_j in D of length L, such that the mean over S_j is less than T such that T is a threshold defined by the user.

```
The sequential straightforward method is as follow:
D := list of rasters
N:= the number of rasters in D
L:= the size of subsequences
T := the threshold chosen by the user
Result := { }
for every Sj in D {
sum_subseq:=0
for every raster Ri in Sj
{
avg_raster := the cell average for Ri
sum_subseq:= (sum _subseq +avg_raster)
}
avg_subseq:=(sum_ subseq)/L
if (avg < T)
then Result := Result ∪ { Sj }
}
```

7.3 Parallel methods for query processing

7.3.1 Straightforward parallel approach

This approach consists, as the first step, to compute the average of each subsequence in D and then check if it is satisfying the user condition. Concerning the first step, we have used the segmented reduction technique with the sum operator. The segmented reduction is a building block for many algorithms. In general, it relies on reducing data over many irregular-length segments. In our case, the segments have the same length which is the size of the subsequences L. In our method, rasters are aligned and stored in one array. The segments are composed of the cells of all the rasters

belonging to them. To perform the segmented reduction, one unique key is assigned to each segment therefore to each subsequence in our array. If there are N subsequences of size L, there will be N fixed-size segments.

This approach allows performing the reduction on all the subsequences only once (based on their keys). Hence the function (kernel) responsible for the reduction is called just once on the array containing the data. The output will be a single 1D array containing the means of all the subsequences S_j. An illustrative example is presented below.

Let's consider a set of 6 rasters of size 4 (Figure 7.2)

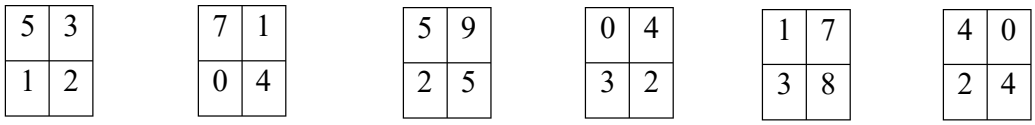

Figure 7.2. A set of 6 rasters of size 4.

Rasters are aligned and stored in one array as follow (Figure 7.3):

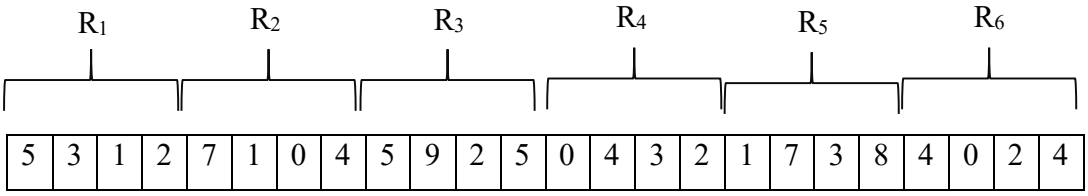

Figure 7.3. Alignment of the 6 rasters.

Our query allows selecting disjoint subsequences of size L such that the average over these subsequences satisfying the user's condition. In our example L= 3 (each subsequence contains 3 rasters see Figure 7.4).

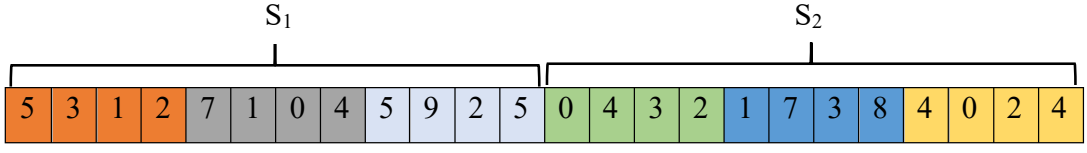

Figure 7.4. Example of two disjoint subsequences of size 3.

Now, the same key is assigned to each cell in the same subsequence. Hence, we get two segments and the sum of each subsequence is computed based on their keys (Figure 7.5)

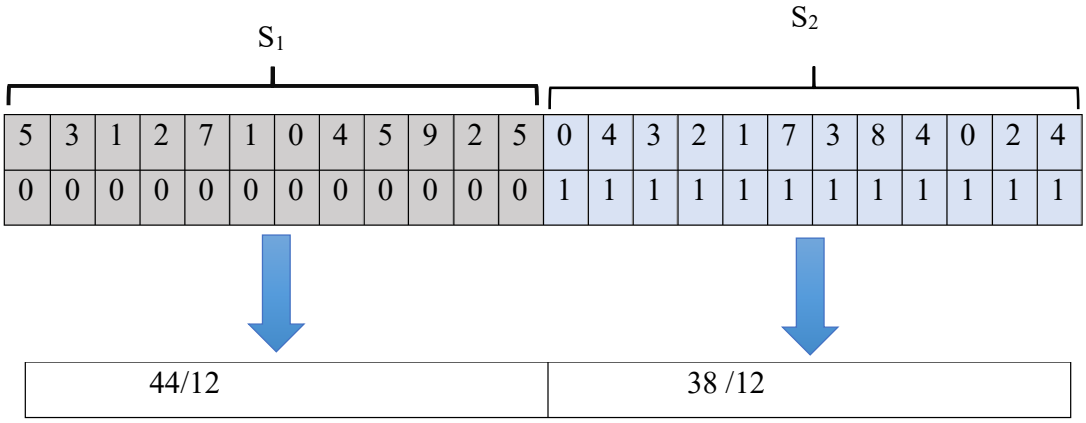

Figure 7.5. Example of two disjoint subsequences of size 3.

Concerning the second step which consists in testing if the average of each subsequence S_j is satisfying the user's condition, we assign one thread to each result in the output array which will be responsible for checking the condition.

The straightforward parallel approach is quite simple since the segmented reduction function is called only once. After that, threads are launched to check the user's threshold condition. However, the main limitation of this straightforward approach is that the reduction operation is very expensive in time and computation.

7.3.2 Improved parallel approach

7.3.2.1 Based on a sort

In this subsection, we overcome the limitation of the straightforward parallel approach by introducing a sorting step in the process. We have introduced this idea for the individual selection of rasters in the previous chapters. Here, the idea behind the sorting is to try to reject the subsequences not satisfying the query condition in early stages to avoid useless computations since

the goal of the query is not to compute the average of all subsequences. We do not have to complete the average computation for the subsequence that does not satisfy the condition. A subsequence average computation can be stopped as soon as we are sure that the user-defined condition will not be satisfied. As a first step, we propose to sort the cells of each raster in descending order. In that case, the threshold is reached faster for the subsequence S_j that does not satisfy the user-defined threshold.

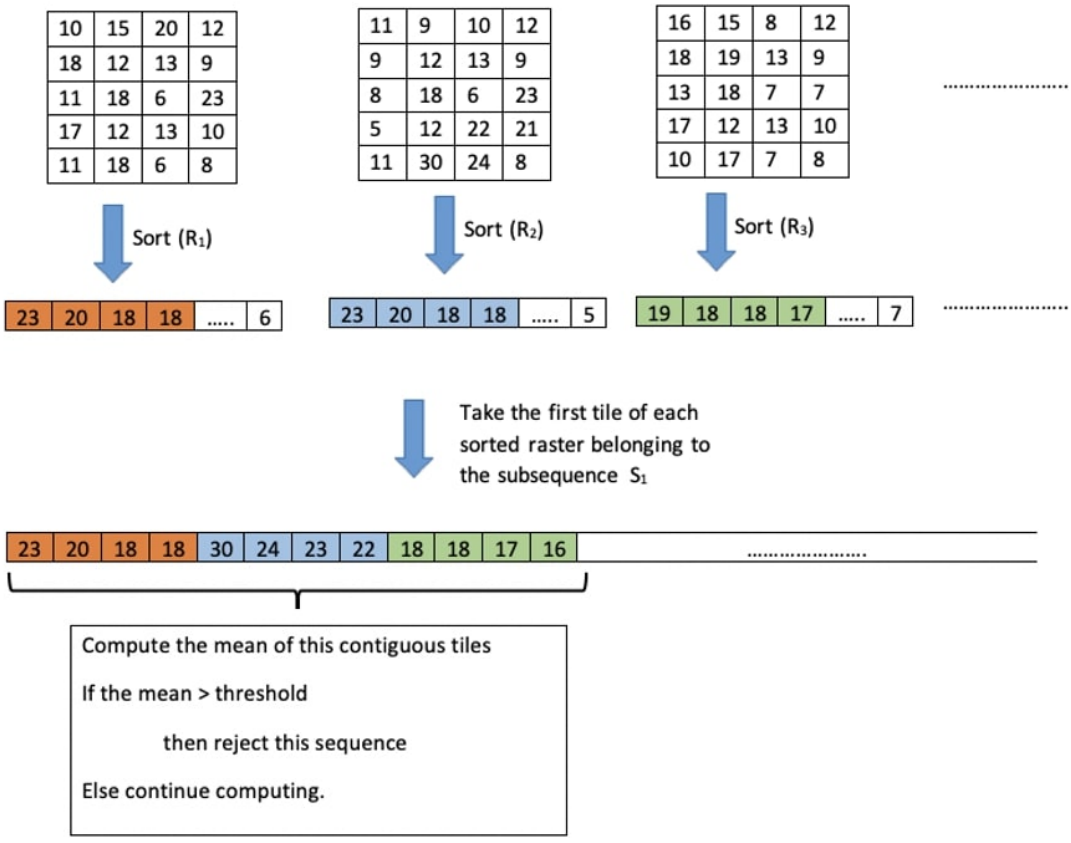

Figure 7.6. Illustration of our method for the first subsequence S_1.

Figure 7.6 shows the first step consisting in sorting all the rasters in parallel in descending order. To do that efficiently, we have adopted the segmented sort parallel technique to get all the rasters sorted in one shot. To achieve this objective, we have chosen a data parallel primitives approach, for instance, **cub::DeviceSegmentedRadixSort** parallel primitive function which allows performing a batched radix sort across multiple non-overlapping sequences. The reason why

we decide to use these parallel primitives is to reduce the implementation complexity and maximize the performance of our algorithms since these parallel primitives are highly improved.

Once the rasters are sorted, we move to the second step. It consists in splitting each sorted raster into equal segments (tiles). Thus, the first tile of each raster contains the cells of the largest numbers. We consider positive numbers for cell values. In the case of negative numbers in the datasets, shifting by a large number is needed (e.g. adding 100 to all the values).

In the first iteration, we will compute only the sum of the first tiles of each raster for a given subsequence S_j. If the result Res_1 does not satisfy the query condition, then the subsequence S_j is rejected. Otherwise, we repeat the same processing for the second tiles of each raster in S_j, add it to Res_1 (the previous results of the first segments) and check the results: if Res_2 does not satisfy the query condition then S_j is rejected. Otherwise, the process is repeated for the third segment of rasters in S_j and so on. This will be done in parallel at the same time for all the disjoint subsequences.

As indicated in the previous chapter, the main drawback of this approach is that the sorting process is expensive in time and computations, especially in the case of large rasters. We propose in the next subsection avoiding sorting all the rasters and settling for sorting only a few of them. The others will be somehow reordered according to the sorted rasters. This is the subject of the next subsection.

7.3.2.2 Based on the sophisticated sort

We have experimented the sort-based approach used in the previous chapter (Figure 7.7). To implement the parallel approach, first, we have sorted the indexes (keys) of rasters that must be sorted. We used the sorted keys to reorders the other rasters. To sort the raster and indexes, two solutions are available: the solution based on CUB with: **cub::DeviceRadixSort::SortPairs** and the solution based on Thrust with : **thrust::sort_by_key.**

The other rasters that fall in same windows of time will be reordered (sorted) according to the sorted raster using the gather transformation that can be implemented using the primitive: **thrust::gather**.

The rasters are sorted as described above. We need to implement the kernel responsible for computing the average of subsequences using our method described above. To do that, the primitives **thrust:: reduce** or the **cub::DeviceReduce::Reduce** functions can be used.

7.4

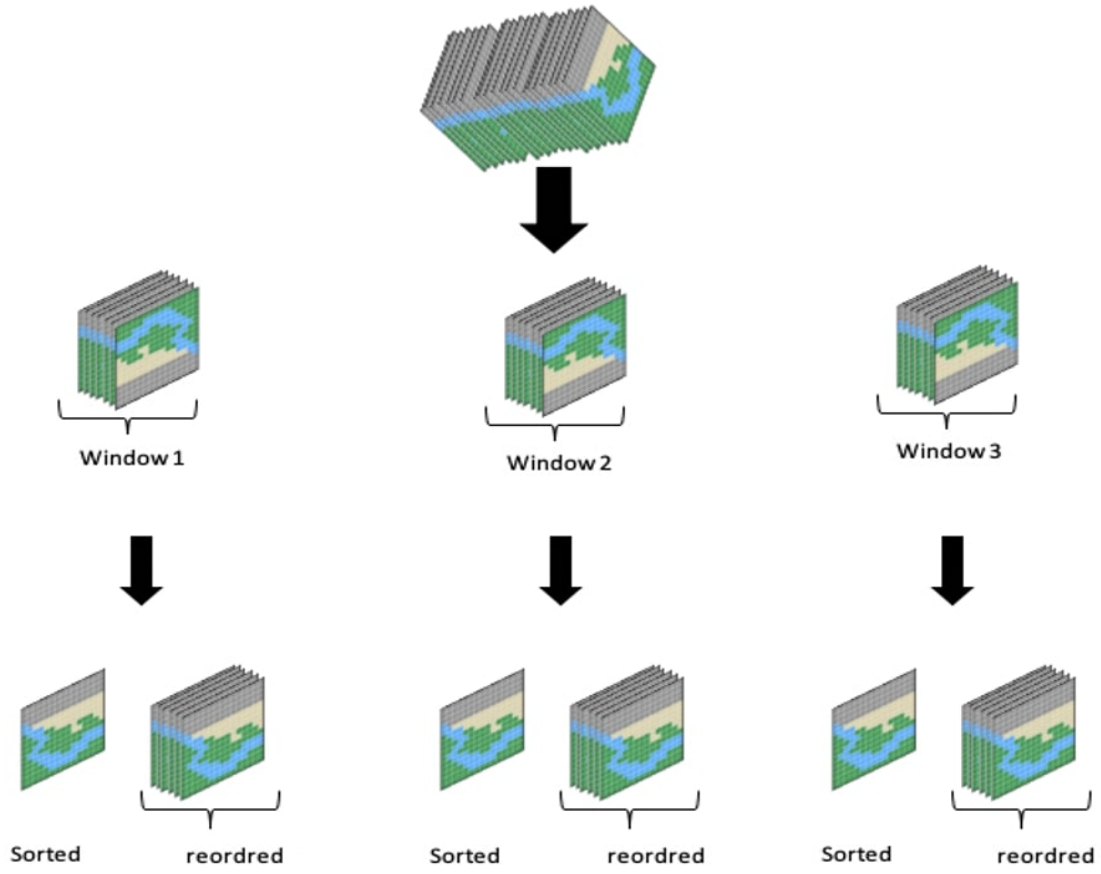

Figure 7.7. Strategy to avoid sorting all the rasters.

Experiments and results

Our experiments were performed on two platforms. Concerning the parallel methods, we have used a GPU-Based platform: Tesla K20 C. While the sequential approaches were performed on Intel(R) Core(TM). To implement our methods we have used C++, CUDA and the libraries Thrust and CUB.

Table 7.1 shows the details related to the used hardware and software configuration.

Platform	Hardware Configuration	Software Configuration
CPU	Intel(R) Core(TM) i7-2600K CPU @ 3.40 GHz Device global memory: 16 GB Cache size: 20,480 KB	Linux Ubuntu 19.04 C/C++ CUDA 8.0 Thrust v10.1.105 CUB v1.8.0
GPU	Tesla K20C CUDA Cores : 2496 Device global memory: 5 GB Memory Bandwidth: 208 GB/s	

Table 7.1. Configuration.

In the tests, we aimed to outline the power of using the GPGPU platform to speed up spatiotemporal raster queries over the classical approaches based on the CPU. We used the same method as the one presented in chapter 6 to generate the dataset.

7.4.1 Experiment Results and Analysis

7.4.1.1 The Impact of the type of data on the performance

As we can see in Table 7.2, we have fixed the size of the raster, the number of the rasters, threshold and also the size of the subsequences that we want select, since they do not have a huge impact on the performance.

Dataset	Standard deviation	Number of rasters	Size of rasters	CPU (ms)	GPU (ms)	Acceleration
Dataset 1	$\sigma = 7.21$	3650	1000×100	261200	200150	1.31
Dataset 2	$\sigma = 7.88$	3650	1000×100	273130	190700	1.43
Dataset 3	$\sigma = 8.82$	3650	1000×100	280810	140600	2.00
Dataset 4	$\sigma = 9.95$	3650	1000×100	259020	118720	2.18
Dataset 5	$\sigma = 11.22$	3650	1000×100	262500	95130	2.76
Dataset 6	$\sigma = 12.57$	3650	1000×100	257200	80147	3.21

Table 7.2. Threshold = 60, the size of subsequences = 10, the size of the windows = 10.

As shown in Table 7.2, the proposed method based on the GPGPU is better than the sequential method based on the CPU for all the generated datasets. Our method is sensitive to data distribution, that is why the results are better when the data has a large standard deviation due to an important variation inside the dataset. On the other hand, execution is slower when the standard deviation of the dataset is small. In fact, if the values in rasters are close to each other, the sort does not provide a great improvement, as the goal of the descendent sort is to sum the highest values first to reach the threshold as soon as possible. The evolution speed over time of the spatial distribution of cell values is a crucial factor in this approach. In the case of rapid spatial distribution value changes, our proposed heuristic will not increase the performance. Note that the sorting step costs about 40% of the computation cost. The fixed parameters of our datasets are not chosen arbitrary - they are the best values that give good performance.

7.4.1.2 The Impact of the time window size on the performance

Let's see how the time window size impacts the performance over "dataset 6" with the same fixed parameters cited in the previous subsection. As we can see in Figure 7.8, when the time window size is too small (size = 3), the number of time windows increases as well as the number of sorted rasters. Since the sorting is expensive, it will increase the time execution. On the contrary, when the time window size is too large (size= 15), it will impact the reordering precision of the rasters and hence the sorting becomes useless which will lead to performance drops. However, when the time window is not too small and not too large (size=10), we get good performance.

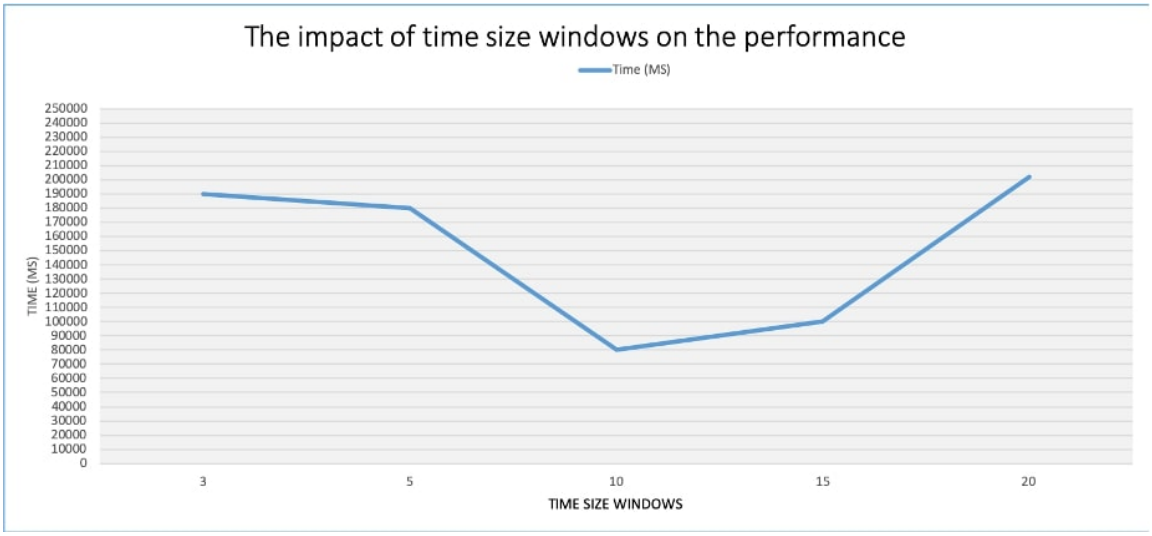

Figure 7.8. The time windows size's impact on the performance over the dataset 6.

7.4.1.3 The Impact of data size on the performance

In this subsection, we show the data size impact the performance of our method. To do that we have generated different datasets with different raster sizes. We have fixed the standard deviation which is equal to 12.57, the size of subsequences = 10 and the size of the time windows = 10.

Dataset	Number of rasters	Size of rasters	CPU (ms)	GPU (ms)	Acceleration
Dataset 1	3650	100×100	21350	6843	3.11
Dataset 2	3650	200×200	182070	57254	3.18
Dataset 3	3650	1000×100	257200	80147	3.21
Dataset 4	3650	500×500	415021	117040	3.54

Table 7.3.

As shown in Table 7.3, the proposed method is still better than the sequential over different dataset sizes.

7.5 Conclusion

Analyzing large-scale spatiotemporal data allows extracting more valuable and crucial information that is essential for many applications, for instance: supporting decision making, science discovery and prediction making. The main requirement for spatial data-intensive applications is the processing time and scalability. Unfortunately, most of the existing methods are based on traditional approaches and architecture which make them not appropriate to support querying massive spatial data efficiently.

In this work, we addressed the problem of speeding up the spatiotemporal rasters query consisting in selecting only disjoint raster subsequences of fixed size, such that the average of the cells over these subsequences is less than a user defined threshold. In our illustrations, we suppose that the region of interest of the user is the entire raster and not only the sub-region for simplification's purpose.

In our work, first, we have shown that we can improve the processing of such query by using the power of recent GPU cards. Furthermore, using parallel primitives based on CUDA allowed us to further optimize our implementation and reducing coding complexity. Secondly, we have designed and implement a new method including a rejection step based on sorting to reduce computations and hence to further accelerate the processing time of our query. The proposed

approaches have been tested on temperature data. Our experiments on our large-scale geospatial data have shown good performance in terms of time compared to the straightforward GPU method, and we were able to obtain a significant speedup compared to CPU based methods. We have to highlight that our methods based on sorting is sensitive to data distribution, as a result, the performance change depending on the distribution of the data set.

Part 3

As mentioned above in the introduction, this part concerns the two last chapters:

- Chapter 8: We dedicate this chapter for the application of our methods on real data of INRAE Montoldre. We present the data set of Montoldre, the acquisition, the type of data and the data characteristics. Finally, we show the results of our methods on this real data.
- Chapter 9: concerns the conclusion and perspectives. We provide a summary of our work, give outlines of our proposals and contributions, and finally suggest new research trends to extend our work.

Chapter 8

Experiments of the different approaches on real data sets of Montoldre

In this chapter, we extend the experimentations of our methods cited above to a real dataset provided by INRAE Montoldre site (Roussey et al.,2020). Our goals were to evaluate the performance of our methods on a real data set.

The chapter is organized as follow. First, we present the INRAE site located in Montoldre (Allier) and we describe the raw dataset produced by the Montoldre sensor network. We talk about the spatial interpolation used to produce rasters from these georeferenced sensor network data. We show results of our methods on this real dataset.

8.1 Sensor network in Montoldre

INRAE has a large experimental farm located in Montoldre (Figure 8.1). This later is dedicated to the development and the experimentation of agri-environmental techniques. Based on a sensor network composed by Live nodes (developed by LIMOS), this platform provides a real data that are used by researchers in their work related to environmental data. The measurements are air and soil humidity, temperature and light. They are measured using several sensors distributed over Montoldre site.

Figure 8.1. Montoldre INRAE experiment farm (Touseau and Le Sommer, 2019).

8.2 Raw Dataset Description

The Montoldre raw dataset produced by LiveNode is composed of two SQL tables: the network table and the sensors table.

- The network table concerns information about 10 sensor nodes.
- The sensors table contains columns such as: myNodeID, battery, temperature, humidity, light, etc., and 14970995 rows which correspond to the measurements of the different sensors during many months with different fine-grained frequency of acquisition.

The table below shows the description of some measures:

Measure	Meaning	Units
Battery	Battery state of node	mV
Temperature	Temperature measurement	C degree
Humidity	Air humidity measurement	Percent
Light	Light measurement	N/A
Watermark n	Measurement value of the n-th watermark device. Watermark is a soil humidity sensor. The Watermark sensors is in the soil at different soil depth.	Watermark's unit (range: 0 to 200)

Table 8.1. Description of the measures.

Note:

Watermark1: This sensor is deployed at 10 cm depth in the soil.

watermark2: This sensor is deployed at 20 cm depth in the soil.

watermark3: This sensor is deployed at 30 cm depth in the soil.

8.3 Spatial data interpolation

8.3.1 Spatial interpolation

Rasters are produced form the sensor network using spatial interpolation. Spatial Interpolation methods can be classified into three categories: geographical statistics methods for instance Kriging methods, non-geographical statistics (the main method in this category is the inverse distance weighted - IDW) and finally the hybrid approach. In this subsection we will present the IDW and kriging methods which are the eminent methods in geospatial interpolation methods.

Inverse Distance Weighted Interpolation Method

Based on the Tobler's first law of geography "*everything is related to everything else, but near things are more related than distant things*" (Tobler,1970), the Inverse Distance Weighted (IDW)(Li et al.,2018) is the most used interpolation method thanks to its simplicity and intuitive interpolation. The main advantage of the IDW is the easiness of implementation and the fact that

is keeping the measured value at sample location. Thus, the IDW is used by several fields and widely adopted by almost GIS. The idea behind the IDW is that the prediction of the values of unsampled point realized by computing the weighted average of the closest sampled points.
Hence the formula of the IDW is as follow (Li et al.,2018):

$$Z = \sum_{i=1}^{n} \frac{1}{(d_i)^p} Z_i \Big/ \sum_{i=1}^{n} \frac{1}{(d_i)^p}$$

Such that:

- Z: The unknown value for estimated value point,
- Z_i: The known value for exact value point,
- d_i: the distance between exact point and estimated point,
- p: A power parameter,
- n: The number of sample data points.

Kriging Interpolation Method

Kriging is a powerful geostatistical interpolation method is named after D.G. Krige from South Africa. The method is founded on the idea of estimating the unsampled points using sampled points and their spatial relationships (Singh and Verma, 2019) using Semivariogram (Tan and Xu, 2014) to assign optimal weights (kriging weights) to the sampled point values so as to compute the unsampled points. The Semivariogram can be a Gaussian model or others models. There are several variants of the kriging method, however the most used for spatial interpolation data is the ordinary version which assumes that the mean and variance of the values is constant across the spatial field. The optimal weights for each unsampled point computed in reference to all the sampled points using the following formula:

$$\hat{Z}(s_0) = \sum_{i=1}^{N} \lambda_i Z(s_i),$$

Where:

the value of the predicted point (z-hat, at location x-nought) is equal to the sum of the value of each sampled point (x, at location i) times that point's unique weight (lambda, for location i).

From a subpart of the Montoldre sensor dataset, 4980 rasters have been produced in 4 different resolutions (300x300, 350x350, 400x400, 450x450) for 3 types of measures (Temperature, humidity, Watermark 2, using the IDW interpolation methods – for a total of 4980*4*3=59760 rasters. The IDW interpolation method allows predicting the messing values of the unsampled locations. For each type of measurements and each resolution, one raster is generated by hour, with at least 3 available sensor measures – as all the sensors are not always active at the same time.

8.4 Experiments on Montoldre hourly dataset

In this subsection we report the results obtained by our experimentations on the Montoldre dataset using the overlapping aggregation of raster data sequences and the selection of raster sequences based on a User-defined condition. The process related to the selection of raster sequences based on a user-defined condition is a quite similar to the other raster selection process including aggregations.

Note that (as a precaution) we have modified the temperature values (by using a translation) to obtain only positive values.

Hereunder a table (Table 8.2) which shows the statistical description of the datasets such that:

- Min: The minimum value of the measure in the whole dataset.
- Max: The maximum value of the measure in the whole dataset.
- Mean: The mean value of the measure in the whole dataset.
- Standard deviation: The standard deviation of the measure in the whole dataset.
- Mean of rasters means: To compute this value, first we compute the mean of each raster used for our experiments then we compute the global mean which is the mean over the raster means).
- Mean of rasters standard deviation: To compute this value, first we compute the standard deviation of each raster used for our experiments then we compute the mean over all the standard deviations of the rasters).

Measure	Min	Max	Mean	Standard deviation	Mean of rasters means	Mean of raster standard deviation
Temperature	0	45.7	11.89	8.73	12	2.73
Humidity	9.9	100	78	21.97	78.15	15.20
Watermark 2	1	200	63.80	88.96	61.71	43.79

Table 8.2. Statistical Description of the measures.

Dataset	Raster size	CPU (ms)	GPU (ms)	Acceleration
Dataset1	450×450	247256	3988	62
Dataset2	400×400	203058	3501	58
Dataset3	350×350	187089	3171	59
Dataset4	300×300	165474	2853	58

Table 8.3. Experiments on temperature - Results for Overlapping Aggregation of Raster Data Sequences.

Dataset	Size of rasters	CPU (ms)	CPU With sorting(ms)	GPU (ms)	GPU With sorting(ms) (ms)
Dataset 1	100×100	25464	34180	21023	29117
Dataset 2	200×200	240255	380521	190762	316284
Dataset 3	1000×100	349654	471290	260273	397641
Dataset 4	500×500	587323	697138	418012	631540

Table 8.4. Experiments on temperature - Results for the Selection of Raster Sequences based on a User-defined condition.

Dataset	Raster size	CPU (ms)	GPU (ms)	Acceleration
Dataset1	450×450	247020	4117	60
Dataset2	400×400	195510	3430	57
Dataset3	350×350	187575	3075	61
Dataset4	300×300	162960	2910	56

Table 8.5. Experiments on air humidity – Results for Overlapping Aggregation of Raster Data Sequences.

Dataset	Size of rasters	CPU (ms)	GPU (ms)	GPU with sorting (ms)
Dataset 1	100×100	28231	17328	7133
Dataset 2	200×200	250165	131571	61604
Dataset 3	1000×100	369719	171803	83251
Dataset 4	500×500	608413	372631	127679

Table 8.6. Experiments on air humidity - Results for the selection of Raster Sequences based on a User-defined condition.

Dataset	Raster size	CPU (ms)	GPU (ms)	Acceleration
Dataset1	450×450	241227	3829	63
Dataset2	400×400	203547	3571	57
Dataset3	350×350	190806	3234	59
Dataset4	300×300	156520	2795	56

Table 8.7. Watermark experiments – Results for Overlapping Aggregation of Raster Data Sequences.

Dataset	Size of rasters	CPU (ms)	GPU (ms)	GPU with sorting (ms)
Dataset 1	100×100	27647,508	12981	7275,66
Dataset 2	200×200	266498,904	116730	63452,12
Dataset 3	1000×100	382122,09	162014	84916,02
Dataset 4	500×500	625116,384	302833	130232,58

Table 8.8. Watermark experiments - Results for the selection of Raster Sequences based on a User-defined condition.

8.5 Discussion

The overlapping aggregation of raster data sequence based on the GPU is still faster than the CPU version over the three measures: Temperature (Table 8.3), humidity (Table 8.5) and watermark (Table 8.7). As we can see in these tables, a very good speedup is obtained over all the generated datasets. These results were been absolutely expected since the performance of our method is independent to the data – the (GPGPU) parallel approach is better than the CPU "sequential" one. Besides, our method is generic and can be used for more general cases and for other types of data, such as array and vector aggregation.

Concerning the selection of raster disjoint subsequences based on a user-defined condition method, the results using the temperature dataset showed always good results for using the GPU over the CPU version. However, our optimized method based on rejection step based on sorting showed bad results for the CPU and the GPU for the temperature dataset. In fact, the time of execution is worse compared to methods without sorting. In fact, our optimized methods are based on a sorting step that must allow to reject the raster sequences earlier. However, the values of the temperature dataset are closer to each other, in rasters. The goal of the sorting step is to allow summing the highest values first. Consequently, when the values are too close in rasters, the sorting step is

useless to reach quickly the threshold and to reject raster in an earlier stage. In this case, the sorting step becomes a heavy step that burden off the shoulder of the method since sorting is expensive in term of computations, hence the execution time is higher than the methods without sorting.
Unlike the previous results on the temperature data. In the case of Humidity and Watermark datasets, our selection methods using the sort gets good results in the sequential and even better in the parallel version. This is due to the sorting step which allows rejecting raster sequences in earlier stages, because of the distribution of the humidity and watermark data values (Table 8.2).

Chapter 9

Conclusions and Perspectives

9.1 Summary of the work

The acquisition of environmental data has made a huge leap forward in term of technology and the price. Sensors are now, smaller, cheaper and even smarter, moreover, more and more georeferenced sensors are deployed for many applications such as environment monitoring, precision agriculture, positioning, this leads to the production of large spatial data. Data availability and data storage are often not anymore, a barrier, whereas the real bottleneck is, in many cases, the analysis of these spatial data that does not cease to grow dramatically.

Unfortunately, most of existing methods and approaches are based on traditional computing framework (uniprocessors) which makes them not scalable and not adequate to deal with large-scale spatial data. Processing large volume of data is both a challenge and a real opportunity. Querying large-scale spatial data allows extracting more valuable and meaningful information that is vital for decision making especially in precision agriculture. It can be used for recommendations on the use of agricultural inputs (water, phytosanitary treatments, etc.), to optimize production, for crop management in order to optimize and to reduce the use of agro-equipment and decision support systems for farmers. Spatial data is also important is scientific advancement and scenario predictions. The major requirements for the data-intensive spatial applications are the processing time and scalability. Spatial query processing must be fast and able to handle more large spatial data efficiently.

The work in our thesis focused on the acceleration of the processing of spatial data in order to support high-performance queries on this later. Our work is based on the use of the GPGPU device to achieve the expected results.

We started tackling some problems of processing spatial data in order to speed up and accelerate the time of processing.

First, we have worked on a classical problem of overlapping aggregations of large raster sequences, this later has never have been studied before, according to our survey. In this context, we have proposed many experiments and compared different GPGPU implementations strategies for this problem. We have used a public dataset NOAA that provides the temperatures max min and mean of a station at every 5 minutes for many years. The results show that our method is 60 times faster than the sequential version.

Secondly, we have tackled raster selection queries based on a threshold fixed by the user. In fact, in different analyses, users can be interested only in some rasters (for example, days where the temperature were greater than 15 C°). In that case, it is possible to reduce the processing time by implementing a rejection procedure of rasters based on the user's threshold in the early stages of the computation. To this end, we have added a sorting step to reject rasters that are not satisfying the condition in an early time. First, we have implemented a sequential method and in a second time, we have implemented a parallel-based method. Both methods have shown a good performance and the rejection step has improved the performance in both versions of the method. However, the GPGPU based method enhance allowed us to get more efficiency and time processing.

In addition, we have worked on a new query that consist on searching disjoint raster sequences data satisfying a user condition. We have implemented a parallel-based method to speed up the query by including a sorting step in the process. We obtained good results and the paper is almost done therefore almost ready for submission.

We have to highlight that our two last optimized methods based on sorting are sensitive to data distribution, as a result, the performance changes depending on the data set.

Finally, we have tested all our methods on the Montoldre dataset. In this regard, we have generated dataset based on the raw Montoldre dataset. Our methods were able to achieve good results, which confirms the potential of using the GPGPU to support large-scale dataset and the use of the rejection step to avoid useless additional computations in the case of suitable data.

9.2 Perspectives

The results of our work have clearly shown that the use of the GPGPU is very suitable to support high performance query on massive large-scale dataset. To this regard, interesting research

perspectives could be derived from our works on several sides and aspects. In our contributions, only simple map algebra operations were used for raster aggregations i.e., the sum. Hence, it will be interesting to investigate and propose methods based on the GPGPU that covers other map algebra operations such that, Max, min, etc. This will allow researchers to get a clear and complete idea about the power of the use of the GPGPU to process map algebra operations.

Concerning the first contribution, our proposed method is based on two main steps: computation of the mean of each raster then the computation of the mean over the overlapped sequences using the prefix-sum. An improvement of this method can be proposed using only one-pass step to compute the overlapped aggregations of rasters. In fact, the aggregation of the raster subsequences can be computed using only the prefix sum which will absolutely improve the results. Another interesting improvement can be done in the sorting step. In our works, the frequency of raster sorting was fixed based on an empirical method. In fact, we run our methods with many frequencies of sorting then we take the frequency that led to the best results. A good research contribution can be to propose a method to set a criterion to determine the best frequency of sorting based on the statistical description and the nature of data. Furthermore, in the case of the selection queries presented in our works it will be a good idea to check the correlation between many measures (for example temperature and humidity) if they have the same behavior during time. it will be possible to make sorting only for the first measure and reuse it for the other measures having the same behavior by reordering raster cells in order to answer the same selection queries which will avoid additional time processing.

Also, we believe that our proposed methods can be applied and cover other fields which have the same structure and behavior of our data. The investigation of the use of our methods (for instance the rejection process based on sorting) on other fields will be very useful in order to study the behavior of our methods on other types of data from other fields and applications. Finally, implementing our methods on other parallel solutions such as OpenCL will be very useful to make a complete comparison between all the solutions.

Bibliography

Agosto, E., 2013. Vector–raster server-side analysis: a PostGIS benchmark. Appl. Geomat. 5, 177–184.

Ait Issad, H., Aoudjit, R., & Rodrigues, J. J. P. C. (2019). A comprehensive review of Data Mining techniques in smart agriculture. Engineering in Agriculture, Environment and Food, 12(4), 511–525. https://doi.org/10.1016/j.eaef.2019.11.003

Ak, J. N., Liktor, G., & Dachsbacher, C. (2012). GPU Computing : Image Convolution.

Assiroj, P., Warnars, H. L. H. S., Kosala, R., Ranti, B., Supangat, S., Kistijantoro, A. I., & Abdurrachman, E. (2019). The Form of High-Performance Computing: A Survey. IOP Conference Series: Materials Science and Engineering, 662(5). https://doi.org/10.1088/1757-899X/662/5/052002

Barbian, M.H., Assunção, R.M., 2017. Spatial subsemble estimator for large geostatistical data. Spatial Statistics 22, 68–88. https://doi.org/10.1016/j.spasta.2017.08.004

Baumann, P. (2008). Computing Aggregate Queries in Raster Image Databases Using Pre-Aggregated Data. Lecture Notes in Engineering and Computer Science, 2173(1), 201–206.

Baumann, P., Furtado, P., Ritsch, R., Widmann, N., 1997. The RasDaMan approach to multidimensional database management. In: Proceedings of the 1997 ACM Symposium on Applied Computing ACM, San Jose, CA.

Blelloch, G., 1997. Prefix sums and their applications. Tech. Rep. CMUCS-90-190, School of Computer Science, Carnegie Mellon University.

Cary, A., Sun, Z., Hristidis, V., & Rishe, N. (2009). Experiences on processing spatial data with mapreduce. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial

Intelligence and Lecture Notes in Bioinformatics), 5566 LNCS, 302–319. https://doi.org/10.1007/978-3-642-02279-1_24

Cisternas, I., Velásquez, I., Caro, A., & Rodríguez, A. (2020). Systematic literature review of implementations of precision agriculture. Computers and Electronics in Agriculture, 176(May), 105626. https://doi.org/10.1016/j.compag.2020.105626

Cheng, J., Grossman, M., & McKercher, T. (2013). Professional CUDA C Programming. In Journal of Chemical Information and Modeling (Vol. 53, Issue 9).

Daras, G., Agard, B., & Penz, B. (2018). A spatial data pre-processing tool to improve the quality of the analysis and to reduce preparation duration. Computers and Industrial Engineering, 119(February 2017), 219–232. https://doi.org/10.1016/j.cie.2018.03.025

Diamond, H. J., T. R. Karl, M. A. Palecki, C. B. Baker, J. E. Bell, R. D. Leeper, D. R. Easterling, J. H. Lawrimore, T. P. Meyers, M. R. Helfert, G. Goodge, Thorne P. W., 2013: [dataset] U.S. Climate Reference Network after one decade of operations: status and assessment. Bull. Amer. Meteor. Soc., 94, 489-498. doi: 10.1175/BAMS-D-12-00170.1.

EN-NEJJARY, D., PINET, F., KANG, M. -2019. Modeling and Computing Overlapping Aggregation of Large Data Sequences in Geographic Information Systems. International Journal of Information System Modeling and Design, vol.10(1), IGI Global USA, p. 20-41.

EN-NEJJARY, D., PINET, F., KANG, M. -2018. A Method to Improve the Performance of Raster Selection Based on a User-Defined Condition: An Example of Application for Agri-environmental Data. Advances in Intelligent Systems and Computing 893, 190-201., Springer

EN-NEJJARY, D., PINET, F., KANG, M. -2018. Large-scale geo-spatial raster selection method based on a User-defined condition using GPGPU. 11th International Conference on Computer Science and Information Technology, Paris, France, 8 p

Garland, M. and Kirk, D.B., 2010. Understanding throughput-oriented architectures. Communications of the ACM, 53 (11), 58–66.

Greenberg, J. A., Rueda, C., Hestir, E. L., Santos, M. J., & Ustin, S. L. (2011). Least cost distance analysis for spatial interpolation. Computers and Geosciences, 37(2), 272–276. https://doi.org/10.1016/j.cageo.2010.05.012

Grisso, R., Alley, M., McCellan, P., Brann, D., & Donohue, S. (2004). Precision Farming: A comprehensive approach. Marketing Health Services, 24, 12–13.

Gunawardena, T., Vicari, A., & Mecca, G. (2016). Spatial data processing with MapReduce. 2015 IEEE 10th International Conference on Industrial and Information Systems, ICIIS 2015 - Conference Proceedings, 485–490. https://doi.org/10.1109/ICIINFS.2015.7399060

Halpin, T., Melton, J., Simon, A. R., & Chisholm, M. (2006). The Morgan Kaufmann Series in Data Management Systems. In Querying XML. https://doi.org/10.1016/b978-1-55860-711-8.50025-3

Harris, M. (n.d.). GPGPU Lessons Learned.

Harris M., Optimizing Parallel Reduction in CUDA, 2007a. http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf.

Harris M., Parallel Prefix Sum (Scan) with CUDA, 2007b. https://www.mimuw.edu.pl/~ps209291/kgkp/slides/scan.pdf.

He, B., Fang, W., Luo, Q., Govindaraju, N. K., & Wang, T. (2008). Mars: A MapReduce framework on graphics processors. Parallel Architectures and Compilation Techniques - Conference Proceedings, PACT, 260–269. https://doi.org/10.1145/1454115.1454152

Hozo, S.P., Djulbegovic, B., Hozo, I., 2005. Estimating the mean and variance from the median, range, and the size of a sample. BMC Medical Research Methodology 5, 13–13. https://doi.org/10.1186/1471-2288-5-13

http://hadoop.apache.org/common/docs/current/hdfs design.html, 2009

https://en.wikipedia.org/wiki/Viewshed_analysis.

http://planet.botany.uwc.ac.za/nisl/gis/spatial/chap_1_11.htm.

https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/architecture.

https://nvlabs.github.io/cub/

https://thrust.github.io) and CUB

H, Y. (2015). A Review on Relationship between Climate Change and Agriculture. Journal of Earth Science & Climatic Change, 07(02). https://doi.org/10.4172/2157-7617.1000335

J. A. Orenstein. Spatial query processing in an object- oriented database system. SIGMOD'86, 326-336, 1986.

Kang, M.-A., Zaamoune, M., Pinet, F., Bimonte, S., Beaune, P., 2015. Performance optimization of grid aggregation in spatial data warehouses. International Journal of Digital Earth 8, 970–988. https://doi.org/10.1080/17538947.2014.962999

Kirk, D. B., Hwu, W. W., 2013. Programming massively parallel processors: A hands-on approach. San Francisco, Calif: Morgan Kaufmann.

Kriegel, H. P., Seeger, B., Schneider, R., & Beckmann, N. (1990). The R-tree: an efficient and robust access method for points and rectangles. GIS for the 1990s. Proc. National Conference, Ottawa, 1990, 448–455. https://doi.org/10.1145/93597.98741

Li, Z., Wang, K., Ma, H., & Wu, Y. (2018). An Adjusted Inverse Distance Weighted Spatial Interpolation Method. 65(Cimns), 128–132. https://doi.org/10.2991/cimns-18.2018.29

Madakam, S., Ramaswamy, R., & Tripathi, S. (2015). Internet of Things (IoT): A Literature Review.Journal of Computer and Communications, 03(05), 164–173. https://doi.org/10.4236/jcc.2015.35021

Maitrey, S., & Jha, C. K. (2015). MapReduce: Simplified Data Analysis of Big Data. Procedia Computer Science, 57, 563–571. https://doi.org/10.1016/j.procs.2015.07.392

Martin, P., Ayuso, L., Torres, R., Gavilanes, A. 2012. Algorithmic Strategies for Optimizing the Parallel Reduction Primitive in CUDA. 2012 International Conference on High Performance Computing & Simulation (HPCS)

Md. Firoj Ali, & Rafiqul Zaman Khan. (2015). Distributed Computing: An Overview. International Journal of Advanced Networking and Applications (IJANA), 7(01), 2630–2635. http://www.ijana.in/papers/V7I-9.pdf

Melesse, A.M., Weng, Q., S.Thenkabail, P., Senay, G.B., 2007. Remote Sensing Sensors and Applications in Environmental Resources Mapping and Modelling. Sensors (Basel, Switzerland) 7, 3209–3241

Neteler, M., Bowman, M. H., Landa, M., & Metz, M. (2012). GRASS GIS: A multi-purpose open source GIS. Environmental Modelling and Software, 31, 124–130. https://doi.org/10.1016/j.envsoft.2011.11.014

National Ecological Observatory Network (NEON)

Palmaccio, M., Dicuonzo, G., & Belyaeva, Z. S. (2020). The internet of things and corporate business models: A systematic literature review. Journal of Business Research, May, 1–9. https://doi.org/10.1016/j.jbusres.2020.09.069

Pinet, F. (2012). Entity-relationship and object-oriented formalisms for modeling spatial environmental data. Environmental Modelling& Software 30 (80-91)

Pozzani, G., & Zimányi, E. (2012). Defining spatio-temporal granularities for raster data. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6121 LNCS (May 2014), 96–107. https://doi.org/10.1007/978-3-642-25704-9_10

Pullar, D., 2001. MapScript: A Map Algebra Programming Language Incorporating Neighborhood Analysis. GeoInformatica 5, 145–163. https://doi.org/10.1023/A:1011438215225

R. J. Hijmans, S. E. Cameron, J. L. Parra, P. G. Jones, and A. Jarvis. Very high-resolution interpolated climate surfaces for global land areas. International Journal of Climatology, 25(15):1965-1978, 2005.

Roussey, C., Bernard, S., André, G., & Boffety, D. (2020). Weather data publication on the LOD using SOSA/SSN ontology. Semantic Web, 11(4), 581–591. https://doi.org/10.3233/SW-200375

Ruetsch, G., & Oster, B. (2008). Getting started with cuda. Nvidia Corp, Aug, October. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Getting+Started+with+CUDA#0

Sanders, J., Kandrot, E., Dongarra, J. J., 2015. CUDA by example: An introduction to general-purpose GPU programming. Upper Saddle River: Addison-Wesley/Pearson Education

Sawant, S., Durbha, S.S., Jagarlapudi, A. (2017). Interoperable agro-meteorological observation and Analysis platform for precision agriculture : A case study in citrus crop water requirement estimation. Computers and Electronics in Agriculture 138 (175-187)

Schlosser, S. W., Ryan, M. P., Taborda, R., López, J., O'Hallaron, D. R., & Bielak, J. (2008). Materialized community ground models for large-scale earthquake simulation. 2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008. https://doi.org/10.1109/SC.2008.5215657

Singh, P., & Verma, P. (2019). A comparative study of spatial interpolation technique (IDW and Kriging) for determining groundwater quality. In GIS and Geostatistical Techniques for Groundwater Science. Elsevier Inc. https://doi.org/10.1016/B978-0-12-815413-7.00005-5

Song, M., Li, W., Zhou, B., & Lei, T. (2016). Spatiotemporal data representation and its effect on the performance of spatial analysis in a cyberinfrastructure environment - A case study with raster

zonal analysis. Computers and Geosciences, 87, 11–21. https://doi.org/10.1016/j.cageo.2015.11.005

Steinbach, M., & Hemmerling, R. (2012). Accelerating batch processing of spatial raster analysis using GPU. Computers & Geosciences, 45, 212–220. https://doi.org/10.1016/j.cageo.2011.11.012

Tan, Q., & Xu, X. (2014). Comparative analysis of spatial interpolation methods: An experimental study. Sensors and Transducers, 165(2), 155–163.

Temizel, A., Halici, T., Logoglu, B., Temizel, T.T., Omruuzun, F., Karaman, E., 2011. Experiences on image and video processing with CUDA and OpenCL. In: GPU Computing Gems. Morgan Kaufmann, Boston, pp. 547–567. (Chapter 34)

Theobald, D.M., 2005. GIS concepts and ArcGIS methods. 2nd ed. Fort Collins, CO: Conservation Planning Technologies, Inc.

Tomlin, C.D., 1994. Map algebra: one perspective. Landscape and Urban Planning 30, 3–12. https://doi.org/10.1016/0169-2046(94)90063-9

Touseau, L., & Le Sommer, N. L. (2019). Contribution of the web of things and of the opportunistic computing to the smart agriculture: A practical experiment. Future Internet, 11(2). https://doi.org/10.3390/fi11020033

Vincent Tao, C., & Li, J. (2007). Advances in mobile mapping technology. Advances in Mobile Mapping Technology,1–176.https://doi.org/10.4324/9780203961872

Viola, I., Kanitsar, A., Groller, M.E., 2003. Hardware-based nonlinear filtering and segmentation using high-level shading languages, in: IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control. Presented at the IEEE Visualization 2003, IEEE, Seattle, WA, USA, pp. 309–316. https://doi.org/10.1109/VISUAL.2003.1250387

Walker, J. P., & Willgoose, G. R. (1999). On the effect of digital elevation model accuracy on hydrology and geomorphology. Water Resources Research, 35(7), 2259–2268. https://doi.org/10.1029/1999WR900034

Walsh, S.D.C., Saar, M., Bailey, P., Lilja, D., 2009. Accelerating geoscience and engineering system simulations on graphics hardware, Computers & Geosciences 35(12):2353-2364

W. R. Tobler. "A computer movie simulating urban growth in the Detroit region," Econ Geogr, 1970, 46, 234-240.

Wu, Y., Ge, Y., Yan, W., & Li, X. (2007). Improving the performance of spatial raster analysis in GIS using GPU. Geoinformatics 2007: Geospatial Information Technology and Applications, 6754, 67540P. https://doi.org/10.1117/12.764613

Xia, Y. J., Kuang, L., & Li, X. M. (2011). Accelerating geospatial analysis on GPUs using CUDA. Journal of Zhejiang University: Science C, 12(12), 990–999. https://doi.org/10.1631/jzus.C1100051

Yang, C., Yu, M., Li, Y., Hu, F., Jiang, Y., Liu, Q., Sha, D., Xu, M., & Gu, J. (2019). Big Earth data analytics: a survey. Big Earth Data, 3(2), 83–107. https://doi.org/10.1080/20964471.2019.1611175

Yang, Z., Zhu, Y., Pu, Y., 2008. Parallel Image Processing Based on CUDA, in: 2008 International Conference on Computer Science and Software Engineering. Presented at the 2008 International Conference on Computer Science and Software Engineering, IEEE, Wuhan, China, pp. 198–201. https://doi.org/10.1109/CSSE.2008.1448

Zeller, C. (2011). CUDA C / C ++ Basics What is CUDA ? Slides.

Zhang, J., Yang, W., Sun, J., & Lv, Y. (2010). GPU-accelerated parallel algorithms for map algebra.2010 2nd Conference on Environmental Science and Information Application Technology, ESIAT 2010, 1, 882–885. https://doi.org/10.1109/ESIAT.2010.5567202

Zhang, J., & You, S. (2013). High-performance quadtree constructions on large-scale geospatial rasters using GPGPU parallel primitives. International Journal of Geographical Information Science, 27(11), 2207–2226. https://doi.org/10.1080/13658816.2013.828840

Zhang, J., You, S., & Gruenwald, L. (2010). Indexing large-scale raster geospatial data using massively parallel GPGPU computing. GIS: Proceedings of the ACM International Symposium on Advances in Geographic Information Systems, 450–453. https://doi.org/10.1145/1869790.1869859