UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ADIEL SEFFRIN SARATES JR.

Optimizing Two-dimesional Shallow Water Based Flood Hydrological Model with Stream Architectures

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Philippe O. A. Navaux

Porto Alegre
2015

CIP – CATALOGING-IN-PUBLICATION

Sarates Jr., Adiel Seffrin

Optimizing Two-dimesional Shallow Water Based Flood Hydrological Model with Stream Architectures / Adiel Seffrin Sarates Jr.. – Porto Alegre: PPGC da UFRGS,

.

79 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS,

. Advisor: Philippe O. A. Navaux.

1. Hydrological Model. 2. Stream Architecthures. 3. GPGPU. 4. LISMIN. I. Navaux, Philippe O. A.. II. Optimizing Two-dimesional Shallow Water Based Flood Hydrological Model with Stream Architectures.

“Die Realität ist für diejenigen,
die ihre Träume nicht aushalten”

— Spielzeitspruch Saison 2013/2014 Residenz Theather München

— Mote da temporada 2013/14 do Residenz Theather de Munique

AGRADECIMENTOS

Agradeço a todos que de alguma forma contribuíram para a realização deste trabalho. Desde o apoio e palavras de incentivo dos familiares durantes estes dois anos de pesquisa e aprendizado.

Aos amigos e colegas que ocuparam seu tempo com a leitura, correções e sugestões para aprimorar os resultados obtidos e a uma melhor clareza na exposição das ideias. Em especial, agradeço à Francis Birck Moreira pela correção do inglês nos capítulos os quais pôde e ao Rodrigo Kassick pelas dicas de programação necessárias. Agradeço também aos demais integrantes do GPPD por todo suporte e conversas, onde foi possível ver pontos diferentes para tornar este trabalho mais abrangente. Agradeço ao professor Philippe Navaux pela oportunidade de desenvolver este trabalho.

Aproveito para agradecer aos professores Walter Collischonn e Rodrigo Paiva, do Instituto de Pesquisas Hidráulicas - IPH/UFRGS, pelo auxílio nos temas relacionados ao seu campo, bem como ao professor Jeffrey Neal da universidade de Bristol, que forneceu uma versão de seu modelo, a qual serviu como base e ponto de partida para este trabalho.

A todos amigos e também àqueles não citados, um grande abraço e obrigado por tudo.

ABSTRACT

This study aims to explore the difficulties and the benefits of using Streams architectures for the simulation of hydrological events based on shallow water equations. For this purpose, is created foundation on hydrological modeling and some classes of existing models, heterogeneous architectures, and more specifically the two-dimensional model based on the equations used Saint-Venan. A timeline is constructed relating the applied optimizations beginning from the first serial model optimized for a GPU version showing each step taken in the form of an algorithm to reach the best performance. With these optimizations a speedup about 4 times was obtained for small areas and 10 times with a middle level of detailing for a large area with a high level of detailing. These results were produced comparing the GPU performance with a CPU and 24 threads version.

Keywords: Hydrological Model, Stream Architecthures, GPGPU, LISMIN.

Otimizando Modelos de Inundação Bidimensionais baseados em Águas Rasas com arquiteturas Stream.

RESUMO

O presente trabalho tem como objetivo explorar as dificuldades bem como os benefícios da utilização de arquiteturas Streams para a simulação de eventos hidrológicos baseados nas equações de águas rasas. Pra tal, é criado embasamento sobre modelagem hidrológica e os algumas classes de modelos existentes, arquiteturas heterogêneas e mais especificamente do modelo bidimensional usado baseado nas equações de Saint-Venan. Com isso é construida a linha de tempo referente às otimizações aplicadas ao modelo inicialmente serial até sua versão otimizada para GPUs, exibindo cada passo tomado em forma de algoritmo para chegar ao objetivo. Com estas otimizações foi obtido um speedup de quatro vezes para pequenas áreas e de 10 vezes com uma resolução média para uma grande área com um alto nível de detalhamento, quando comparado com uma versão de 24 threads.

Palavras-chave: Modelagem Hidrológica, Arquiteturas Stream, GPGPU, LISMIN.

LIST OF FIGURES

2.1 Time-series plots of 30-day annual-maxima series in Thames River at Kingston. . . . 26
2.2 Classification of hydrologic models based on the way they represent variations of space, time and uncertainty of hydrologic systems. . . . 28
2.3 Three approaches of hydrological models. From left to right: lumped, semi-distributed and distributed model approach. . . . 28

3.1 One single-chip CBEA with a CPU core and eight SIMD accelerator cores. . . . 35
3.2 A diagram of the FPGA architecture. . . . 35
3.3 A diagram of the GPU architecture. . . . 36
3.4 SISD, SIMD, and stream processing compared 37
3.5 Comparative of theoretical peak power of single and double precision for CPU and GPU 39

4.1 Spatial representation of the finite difference using three points to increase the model stability 46

5.1 Flowchart of model execution steps. . . . 50
5.2 Steps for generation of matrix H, representing the water depth of each area in analyzed map area. . . . 52
5.3 Visual model elements responsible for generation of simulation's matrices. . . . 53
5.4 Steps for flow computing. . . . 53
5.5 Event hydrograph simulated in test case. . . . 55
5.6 Grid of Thread Blocks 59
5.7 Pageable Data Transfer vs Pinned Data Transfer 61
5.8 Developer view of memory hierarchy with and without unified memory 63

6.1 Map areas used in this work. . . . 66
6.2 Execution time and speedup for serial and parallel version of Glasgow simulation in CPU. . . . 67
6.3 Execution time and speedup for parallel version of Glasgow simulation in CPU with 24 threads and GPU optimization. . . . 68
6.4 Plots of simulated water depth over he time for OpenMp and CUDA versions with no difference in results. . . . 69
6.5 Execution time and speedup for serial and parallel version of Glasgow simulation in CPU and GPU. . . . 70

6.6 Hydrogram related to equation 6.1. 71
6.7 Execution time parallel versions of Itajaí-Açu River simulation in CPU and GPU. 71

LIST OF TABLES

6.1 Measured improvement over time for technique on the line over column and the average time for each technique. 69

LIST OF ALGORITHMS

1 Model Pseudocode 51
2 Serial Implementation (Timestep Computation) 53
3 Serial Implementation (Flow Discharge Computation) 54
4 Serial Implementation (New Water Level Computation) 54
5 Serial Implementation (Boundary Conditions) 55
6 OpenMp Implementation 56
7 GPU Implementation (Allocation and Copy for matrix H) 57
8 GPU Implementation (Cuda kernels) 58
9 GPU Implementation (Thread Control) 59
10 GPU Implementation (memory allocation) 61
11 GPU Implementation (Streams) 62
12 GPU Implementation (Unified Memory) 63

LIST OF ABBREVIATIONS AND ACRONYMS

API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
CBEA	Cell Broadband Engine Architecture
CPU	Central Processing Unit
cuBLAS	CUDA Basic Linear Algebra Subroutines
CUDA	Compute Unified Device Architecture
DEM	Digital Elevation Model
DMA	Direct memory access
FLOPS	Floating-point Operations Per Second
FPGA	Field Programmable Gate Array
FPGA	Field Programmable Gate Array
GDDR5	Graphics Double Data Rate Type Five
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processor Unit
GR4J	Modèle du Génie Rural à 4 paramètres Journalier
GSSHA	Gridded Surface Subsurface Hydrologic Analysis
HBV	Hydrologiska Byråns Vattenbalansavdelning
HDL	Hardware Description Language
HPC	High Performance Computing
HRU	Hydrological Response Units
I/O	Input and Ouput
LiDAR	Light Detection And Ranging
LISMIN	LISFLOOD Minimal version
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MOHYSE	Modèle Hydrologique Simplifié à l'Extrême

MPMD	Multiple Program Multiple Data
PC	Personal Computer
PCI-E	Peripheral Component Interconnect Express
PDE	Partial Differential Equation
RAM	Random Access Memory
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SMX	Streaming Multiprocessor
SPMD	Multiple Program Multiple Data
TOPLATS	TOPMODEL based land atmosphere transfer scheme

CONTENTS

1 INTRODUCTION 21
1.1 Problems 21
1.2 Motivation 22
1.3 Subject 23
1.4 Text Organization 23

2 HYDROLOGICAL MODELLING 25
2.1 Flood Types 25
2.2 Hydrological Models 26
2.3 Model's evolution 26
2.4 Model Type 27
2.5 Lumped Models 29
2.6 Quasi-Distributed Models 30
2.7 Distributed Models 30
2.8 Stochastic Models 32
2.9 Modeling Conclusion 32

3 HETEROGENEOUS ARCHITECTURE 33
3.1 Architectural characteristics 34
3.2 GPU Computing 36
3.3 Conclusion of Heterogeneous Architectures 38

4 TWO-DIMENSIONAL FLOOD INUNDATION MODELING 41
4.1 Initial formulation 41
4.2 Inertial formulation of the shallow water equations 44
4.3 Improving the stability of inertial formulation 45
4.4 Conclusion of two-dimensional flood inundation modeling 47

5 PROPOSAL OF OPTIMIZATIONS TO LISMIN MODEL FOR GPU ENVIRONMENT 49
5.1 Base model and pseudo-code 49
5.2 Serial Code 51
5.3 CPU Parallel Code 56
5.4 GPU Parallel Code 56
5.4.1 GPU code recode 57
5.4.2 GPU block management 58
5.4.3 Gpu code with pinned memory 60
5.4.4 GPU code with streams 61

5.4.5 GPU code with Unified Memory 62

6 RESULTS 65
6.1 CPU results 65
6.2 GPU results 66
6.2.1 Summarized results 66
6.2.2 Resources 68
6.3 Scalability results 69

7 ANALYSIS AND CONCLUSIONS 73

REFERENCES 75

1 INTRODUCTION

Flood events in urban areas are becoming more frequent as a consequence of several factors including population growth, increasing pressure on communities to develop on flood prone areas, climate change which has magnified the intensity of rainfall, sea level rise which threatens coastal areas, and decaying or poor infrastructure. The effects of inundations vary from wetlands to extreme flood events that result in entire cities with underwater houses. These events generate economic and social impacts, high cost aid programs in addition to the infrastructural damages. Warning systems and action plans in case of flooding are already part of the daily life of dozens of countries. Unfortunately, to be prepared for all kinds of flood events, a high investment in several areas is necessary, which makes it impossible for many societies. To overcome this issue, it is necessary to understand what are the principal flood events that put in risk determined location. The most familiar type of flood event is the pluvial flooding which occurs when intense rainfall's runoff cannot drain away the water quickly enough. Understanding this and knowing which are the flood events that have more probability to happen, the entities responsible for warnings and safety can focus on these problems.

In order to predict this range of events, flood models need to support emergency management directly and for urban planning. Flood models can help to identify the most effective risk reduction measures through comparative analysis of the socio-economic and environmental consequences of each alternative found.

Future flood impacts can be limited through risk reduction measures such as changes in land use and building codes, selective relocation of vulnerable assets, improvement of flood defenses, emergency awareness and insurance. Moreover, flood inundation models play a central role in the evaluation, selection and in some cases the implementation of these measures.

1.1 Problems

As hydrological model simulations need to have a short enough time to achieve a final state and produce useful results, it is necessary to develop models that compute this

simulation in the fastest possible way.

The major problems to obtain this efficiency is due to the tenporal data dependencies related to environmental models, which evolve through time, making it impossible to compute a *future step* without knowledge of current simulation state. Moreover the size of analyzed area can difficult simulation as we need to manage this data to compute in correct order. However, although each simulation step is necessary to compute the next one, computing different environmental effects of the same step that can be considered independent to each other (e.g. the flow between two different cells) can be parallelized. Working with large area models allows us to use a parallel approach in the analyzed area and improve the computational time to find the next simulation state.

In order to speed up such models, the utilization of hardware that uses Stream architectures introduces extra work due to reported difficulty to recode these models when compared with other methods that do not need external hardware to run (NEAL et al., 2010). Stream architectures explore the concurrency in processing applications where the applications are considered as a sequence of computation instructions (RIXNER, 2001).

In this way, the main theme of this dissertation is related to the impact and influence of stream architectures in the behavior and performance of such models.

1.2 Motivation

With the evolution and advances of processing resources, several applications could take advantage of these new technologies. The possibility to run portions of your application in different processing elements (i.e. many processors in the same system, different cores in the same processor) enables the application to achieve its final stage faster given a good parallelization. This opportunity to speed up applications allows scientific and business communities to scale out their applications, expanding the horizons of research and efficiency. Execution time has always been a concern for both communities.

An example of this can be seen in the gaming industry. It leveraged the development of graphic cards to accelerate and improve the quality of created games, and as a collateral effect also enabled a new way to compute non-game applications. The high performance achievable with graphic cards is possible due to the type of processing available, where we apply the same operation to different data in parallel.

As the High Performance Computing (HPC) community reaches to scientific applications, many solutions were developed to improve the performance of scientific simulations through the use of new architectures. Once the gaming industry has shown the great advance in applying small instructions to several data in games using its Graphics Processor Unit (GPU), it became obvious that a large portion of scientific simulations could also take advantage of GPUs. This work aims to explore the benefits of this architecture when using hydrological models focused in flood events.

1.3 Subject

Given the performance problems of the models, we defined the main subjects of this work as listed below. Analysis of potential bottlenecks in hydrological simulations: we search the critical areas of a given model applying some known resolution method and consider manners to solve or minimize them. Analysis of potential independent portions of model: we look for improvements in the instructions call aiming for a better simulation performance for all related architectures used. Evaluation of difficulty to *re-code* a hydrological model: In order to be able to execute simulations in a stream architecture, we evaluate the difficulty to do this translation of the original model's code. Evaluation of global performance: we analyze the losses or gains in performance using one of these architectures when compared with the original model architectures.

1.4 Text Organization

In Chapter 2 we describe the state-of-the-art of hydrological modeling. The differences among some flood types are introduced as well as a classification of hydrological models based on the way they represent variations of space and time giving examples of each model. In Chapter 3 we introduce the state-of-the-art about heterogeneous architectures which provides tools to speed up several applications. A review of Flynn's taxonomy is made and features some of the components used to create heterogeneous architectures. In Chapter 4, we make a study of the model that was used to evaluate this work, showing the main equations of shallow water models as well as the latest mathematical and computational improvements, and why they are made. In Chapter 5 we detail which optimizations can be made in codes based in stream architectures to take advantage in simulation of this specific hardware. In Chapter 6 , the results of optimization cited in the last chapter are shown along with their respective analysis. And finally, the last Chapter is reserved for conclusions of this work.

2 HYDROLOGICAL MODELLING

In this chapter, we define basic concepts, such as the types and differences among flood types. With these concepts we can move on to the definition and evolution of hydrological models and then to a detailed explanation of the main models used today.

2.1 Flood Types

Flooding can take many forms. Between the most familiar pluvial flood and the probably least familiar, the groundwater flooding, there are several types of floods. The pluvial flooding occurs when the urban drains cannot drain the water from a intense rainfall quickly enough. When this flooding event merges the rainfall water with another terrain source, like small Urbain watercourses, this event is called a surface-water flooding. In some cases where there are combined sewer and storm draining systems, storm water can cause sewage to flow into streets and rivers, contributing to surface water flooding. There are cases in which the rainfall is heavier than usual causing a very quick inundation. This event is called flash flooding. The Bobscastle flood in 2004 is a well known example (GOLDING; CLARK; MAY, 2005).

For communities that live near rivers, the most common flood event is called fluvial flooding. It occurs when there is too much water in a river and it spills on to adjacent flood plains. This flood event has a big impact for families that lives in the shore of these rivers. Besides this, weather systems interact with the oceans creating storm surges and large waves. When combined with high tides or earthquakes, these can overtop and undermine sea defenses and lead to coastal erosion and flooding. That is the flood event that hit India in 2004 and 2007 (GOSLING et al., 2011).

Finally, maybe the least familiar flooding comes from a rising water table. When the water table reaches the surface and more rain falls, groundwater flooding can happen. (HARDAKER; COLLIER, 2013)

2.2 Hydrological Models

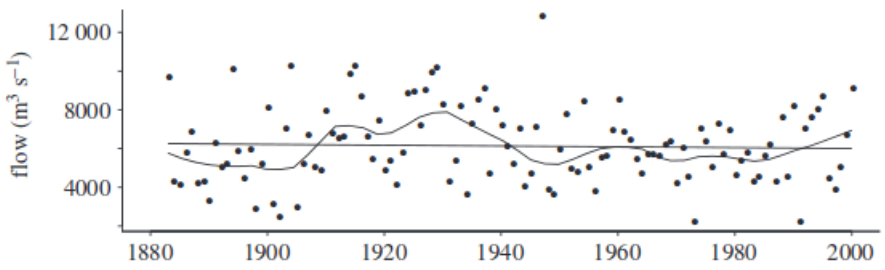
Hydrological modeling has become an indispensable tool for studying flood-related processes and water management in catchments. Extreme flood events are becoming common and faster warnings are necessary so that appropriate actions can be taken. Figure 2.1 shows the distribution of the water flow in River Thames at Kingston. In the last 100 years, the river has had a huge water flow several times (ROBSON, 2002).

Computer based hydrological models have been developed and applied at an ever increasing rate during the past four decades, due to the improvement of models and methodologies that are continuously emerging from the research community. The demand for improved tools increases with the increasing pressure on water resources. These models help to forecast events simulating the duration of inundation, flood depth, velocity and associated water forces. Monetary losses are primarily scaled by flood depth and duration, while flood velocity and associated force is important for structural damage assessment. High velocity flows occur on steep topography, such as alluvial fans found in arid regions, with dam-break and dike-break flooding, and with coastal flooding from tsunami and storm surge (SCHUBERT; SANDERS, 2012). According to *Nicandrou et. al.* (NICANDROU, 2011), the hydrological models may vary in terms of how processes are represented, in time and in space, the adopted scales and methods of solution to applied equations. More than that, all hydrological models can be classified according to its space dimensions, time and randomness.

2.3 Model's evolution

Hydrological modeling is more than a hundred years old, with the firsts insight of mathematical models for relating storm runoff to rainfall intensity. Over this period, several concepts and models have been developed. The 30's $\sim$ 40's was a *'theoretical period'*, where researchers like Sherman , Horton and Keulegan among others developed important theories. One of these theories is know as Horton's law, which constitutes the

Figure 2.1: Time-series plots of 30-day annual-maxima series in Thames River at Kingston.



Source: (ROBSON, 2002)

foundation of quantitative geomorphology. The next years could be called as *'development of concepts period'*. Until the middle of the 1960s, hydrologic modeling primarily involved the development of concepts, theories and models of individual components of the hydrologic cycle, such as overland flow, channel flow, infiltration, depression storage, evaporation, interception, subsurface flow, and base flow (SINGH; WOOLHISER, 2002).

In the end of 20's century, between the year of 1970 and 2000, several papers with a now solid ground were published. The hydrological models had been divided in groups such as physically based, conceptual, process-oriented, lumped, distributed and more. With its evolution and the advance of tools to get data, new models issues have emerged. Parameters estimative (SCHUMANN, 1993), parametrization, calibration and validation (REFSGAARD, 1997) (SAHOO; RAY; CARLO, 2006), how to scale the surface to be analyzed (BLÖSCHL; SIVAPALAN, 1995) are now a intrinsic part of hydrologic research.

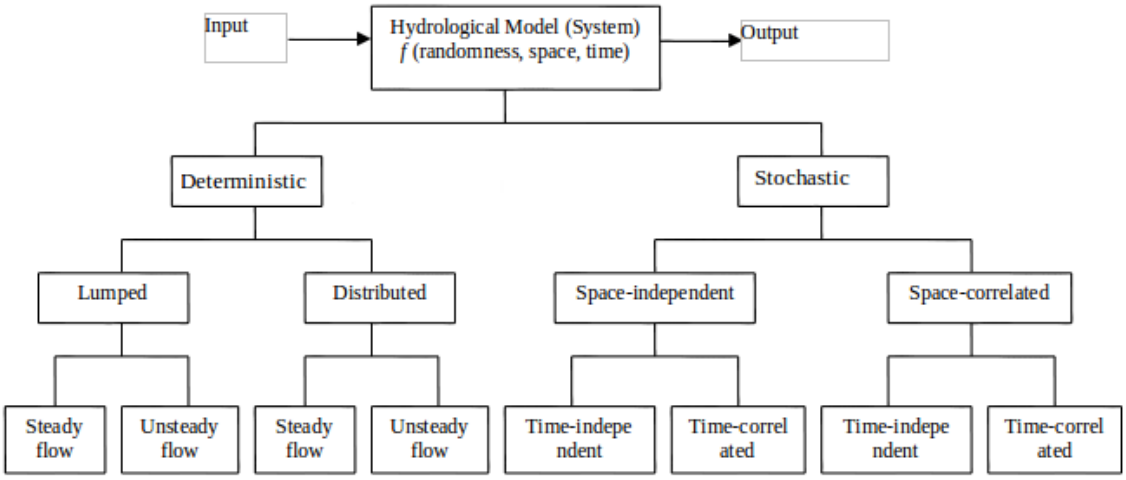
Nowadays, to optimize these models, in addition to new approaches, new resources to compute them are necessary. Models that execute computations in a high precision for big scales demand powerful units of processing. In this way, the older CPUs could not compete with these models in a satisfactory time. With the advance of computational architectures, new model implementations are emerging, decreasing the execution time without losing the required precision.

2.4 Model Type

In *Bates et.al.* (BATES, 2012) a quick overview about the flood research was made. In the paper he argues that in the last 10 years, flood inundation research had rapidly moved from being a 'data-poor' to a 'data-rich' science, with promising possibilities for model development and process insight. Until the late 1990s, the only data available to build, parameterize, calibrate and validate hydraulic models were from limited ground topographic surveys and sparse ground gauging stations with spacings in the ranges between 10 and 60 km. Occasionally, air photos of flooding were available, but these were not used to test model performance in a systematic way because the available terrain data were insufficiently detailed to provide confidence in distributed model predictions. The errors in the then-available data and their limited information content meant that it was very difficult to discriminate between different model physics and parameters, such that many different models could fit the available validation data equally well yet lead to different future predictions or process inferences. Nowadays, these parameters are derived directly from digital representations of the topography being easier to retrieve, refine and analyze these data.

Citting Nicandrou et. al. (NICANDROU, 2011), "The different types of models are separated according to the type of their processes, i.e. whether they are lumped or distributed (see Figure 2.2). The system (hydrological model) can be classified into two

Figure 2.2: Classification of hydrologic models based on the way they represent variations of space, time and uncertainty of hydrologic systems.

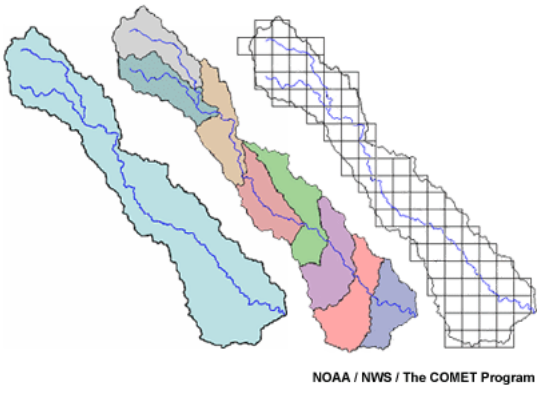


Source: (CHOW et al., 1988)

main categories, deterministic (no randomness) and the stochastic (randomness). These categories are then broken down into further classifications. The deterministic model is classified as lumped (processes are assumed to be spatially uniform) or distributed (processes vary in space). Each type is further classified as steady or unsteady depending on variations with time."

In a majority of cases the system (catchment) characteristics, many of the processes, the input, and even some of the boundary conditions are all lumped, but some of the processes that are directly linked to the output are distributed - for example, the rainfall-runoff process. These models are not fully distributed; rather they are quasi-distributed at best. The figure 2.3 introduce these subdivisions and the next three subsections will detail each of them.

Figure 2.3: Three approaches of hydrological models. From left to right: lumped, semi-distributed and distributed model approach.



Source: (RESEARCH, 2006).

2.5 Lumped Models

A lumped model is generally described as the model which is expressed by ordinary differential equations taking no account of spatial variability of processes, input, boundary conditions and the system's (catchments) geometric characteristics, in other words, the lumped models treat the complete watershed as a single homogeneous element and develop a single outflow hydrograph.

Lumped models make significant simplifications about the real world, therefore these models require a fewer number of parameters or data to be calibrated for their operation. Even with these simplifications, lumped models have proven to be quite successful in simulating an observed flow hydrograph (PAUDEL; NELSON; DOWNER, 2009). One of these simplifications is that rainfall is uniformly distributed over a watershed both spatially and temporally over a given time period. We know that this rainfall distribution does not occur in the real world watershed, although there might be a limited number of cases where this might become a closer approximation (REED et al., 2004). Lumped models assume uniform soil types, vegetation types, and land use practices over a watershed. This is a significant assumption as the infiltration properties that are often governed by the soil and land use widely vary in a watershed. Mean aerial runoff for the drainage basin is computed by making abstractions from the mean aerial precipitation (PAUDEL et al., 2010).

In 1932, Sherman introduced the concept of the unit hydrograph. The unit hydrograph is the runoff that results at the downstream outlet of a drainage basin from a unit depth (i.e. 1 inch or 1 mm) of excess rainfall for a storm of uniform intensity for a specified duration over an entire watershed drainage (MAIDMENT; MAYS, 1988). Traditional lumped models use this unit concept to transform this runoff to determine the total stream flow at the basin outlet (SHERMAN, 1932).

The major benefit of using such simplified models is the ease in their calibration as smaller number of control parameters is used and it is easy to establish some pattern in their variation to produce desired watershed response. In general, their applicability is limited to gauged watersheds where the expected conditions are within the historic data used for calibration and no significant change in catchment conditions has occurred (REED et al., 2004).

Some lumped models are listed in the current bibliography (SEILLER; ANCTIL; PERRIN, 2011). One of them is the french model called *Modèle Hydrologique Simplifié à l'Extrême* (**MOHYSE**, Hydrological Model Simplified at Extreme) (FORTIN; TURCOTTE, 2006). The MOHYSE is a lumped conceptual model of surface flow processes using four reservoirs which include the main processes found in more complex models (snow accumulation and melt, evaporation and transpiration, surface runoff, infiltration, subsurface flow, groundwater recharge, base flow and river routing). For each

of these processes a simple, and in most cases linear, representation of the process is used (LAROCQUE et al., 2010). Another recent work is the *modèle du Génie Rural à 4 paramètres Journalier* (**GR4J**, Model of Rural Engineering with 4 daily parameter). This is a conceptual lumped four-parameter rainfall-runoff model, either lumped or gridded application, daily rainfall-runoff model. The strongest point of this model is that it gives good performance in a range of catchment contexts with only four calibration parameters, besides being well established and widely used. However, it has some weaknesses. GR4J is unable to account for partial area contributions to stream flow and has no water quality modeling capability (PERRIN; MICHEL; ANDRÉASSIAN, 2003).

2.6 Quasi-Distributed Models

Between the lumped models and the distributed models, some authors have defined alternative models, called quasi-distributed or semi-distributed models. These models discretize the watershed into homogeneous sub-areas or sub-basins based on the topography or drainage area, having a less complex spatial representation. This group of models simulates all hydrological process within spatially non-explicit Hydrological Response Units (HRU). Results for each HRU can be lumped within sub catchments and routed downstream. HRUs can be defined based on soil units, land use or a combination of both. Although the impact of environmental change is not simulated with the same spatial resolution as in the spatially explicit approach, these semi-distributed models still require a considerable amount of parameters that might be difficult to obtain. A further simplification is achieved when hydrological fluxes are simulated with the sub catchment scale as the smallest spatial unit (BORMANN et al., 2009).

The *Hydrologiska Byråns Vattenbalansavdelning* model (**HBV**, Hydrological Agency's Water Balance Department model) is an example of this group. **HBV** model is classified as a semi-distributed conceptual model. It consists of three main components: (1) subroutines for snow accumulation and melt; (2) subroutines for soil moisture accounting; (3) response and river routing subroutines. Data requirements and corrections are outlined. Input data are precipitation and, in areas with snow, air temperature. The soil moisture accounting procedure requires data on potential evapotranspiration (BERGSTRÖM; SINGH et al., 1995).

2.7 Distributed Models

Unlike lumped models, spatially explicit models can be conceptual or physically based, and can simulate both spatial and temporal variability of the watershed characteristics. Besides, these models can be parameterized and validated to spatially distributed observations (limited by the constraints of the scale triplet: scaling, support and

extent) (BORMANN et al., 2009).

The physically based fully distributed models divide the entire basin into elementary unit areas such as grid cells and flows are passed/routed from one grid cell to another as water drains through the basin. These models take an explicit account of spatial variability of processes, input, boundary conditions, and/or system (catchment) characteristics (NICANDROU, 2011).

This allows the heterogeneity of the watershed to be simulated at each of the grid cells. Grid resolution is generally chosen in such a way that it is small enough to represent the spatial variation of major runoff processes such as rainfall, infiltration, transformation, etc., but large enough to be computationally practical (PAUDEL et al., 2010). The relationship between rainfall and runoff is a complicated process which is defined by numerous parameters, each with inherent uncertainties. To better incorporate the variations and uncertainties involved in defining a watershed response to the rainfall, a physically based distributed model is always preferable. These models, because of the larger computational requirements, add a massive computational burden. Historically, these computational resources were unavailable to general practitioners. With the advent of powerful computers, numerous quasi-distributed and distributed hydrologic models are emerging and many claim to be the best model, or at least capable of solving a wide variety of problems. Between several models, some models are listed below:

The Gridded Surface Subsurface Hydrologic Analysis (**GSSHA**) is a physically based, distributed-parameter hydrologic model intended to identify runoff mechanisms and simulate surface water flows in watersheds. The GSSHA model is capable of simulating stream flow generated from Hortonian runoff, saturated source areas, exfiltration, and groundwater discharge to streams. The model employs mass-conserving solutions of partial differential equations (PDEs) and closely links the hydrologic compartments to assure an overall mass balance and correct feedback. (PAUDEL; NELSON; DOWNER, 2009)

The TOPMODEL based land atmosphere transfer scheme (**TOPLATS**) is a spatially explicit, grid based, time continuous, and multi-scale model (PETERS-LIDARD; ZION; WOOD, 1997). It combines soil-atmosphere-transfer-scheme, calculating the local scale vertical water fluxes, with the TOPMODEL approach which laterally redistributes the water in the catchment. While the processes within the lower atmosphere are described in a physically based way, the soil water flow is simulated using an approximation of the Richards' equation for a limited number of soil layers. Inter-flow as well as a routing routine is not integrated into the model. Base flow is calculated using a recession function based approach (BORMANN et al., 2009) (BEVEN; KIRKBY, 1979).

2.8 Stochastic Models

During the last few decades, several types of stochastic models have been developed and proposed for modeling hydrological time series and generating synthetic stream flows. Some of such stochastic models are autoregressive (AR), Moving Average, Autoregressive Moving Average, and Autoregressive Integrated Moving Average (BHAT; MILLER, 1972) (HABERLANDT; RADTKE, 2014) . These models are called system theoretic transfer function models because they attempt to establish a linkage between several phenomena without internal description of the physical processes involved. Broadly, the stochastic models are classified as Autoregressive Moving Average models, disaggregation models, and models based on the concept of pattern recognition (LOHANI; KUMAR; SINGH, 2012). Stochastic models explicitly consider the probabilistic nature of model inputs and parameters. Results take the form of probability distributions or processes rather than single numbers. These models have the advantage of accounting for the stochasticity inherent in real systems. However, it has some limitations, as probability distributions must be estimated, synthetic time series generated, the presentation of results is more difficult, and there are difficulties reproducing persistence (Hurst phenomenon) and non-stationarity of time series (HAROU et al., 2009).

2.9 Modeling Conclusion

With these concepts we are able to define the main subject of this work with more details. The model used as test case is defined as a distributed model, however it is a reduced version of a bigger model. This reduction was made to analyze only the computational part of this model in some heterogeneous architecture, which will be studied and detailed in the following chapter.

3 HETEROGENEOUS ARCHITECTURE

The history of the development and practical application of hydrology and other environmental models has been closely aligned with the development of computer hardware and software. While there has always been a focus on the scientific aspects of model improvements (BEVEN, 2001), there have also been constraints on what can be achieved given the available computing power.

Along the decades of 1960 and 1970, according with Tristam (TRISTRAM; HUGHES; BRADSHAW, 2014) when the computational models started to be developed, access to mainframe digital computers was largely restricted to universities or large state agencies, and even those computers were relatively slow and lacked memory in comparison to modern computers. Consequently the development made within research institutions were generally unavailable to practicing scientists and engineers.

With the increased availability and use of desktop computers (PCs) through the 80's, it was possible for the environmental models to become a part of the everyday toolbox of scientific and engineering practitioners. Until the 90's, many that proposed scientific developments remained constrained by computing limitations and were still not generally available for practical use.

However, it was only around 2005 that it became practical to run complex, spatially detailed models on PCs. Computer processors are continuously getting faster and more complex. As well as the drive for scientific advancement, the demand for faster processors can be attributed to the ever increasing computation requirements of the research, industrial, business, and entertainment sectors. In the past, computer users simply waited for reliable increases in processor speeds to handle computation problems that were not really feasible at the time. However, it is believed that the single-core processors have hit the power wall (MEENDERINCK; JUURLINK, 2009), meaning that the single-core frequency improvements can no longer be easily made because of the power and heat constraints (ROSS, 2008). CPU manufacturers have then changed their focus from single-core processors to multicore processors.

With the growth in computing power and data, also grows the willingness to run the hydro-meteorological simulations at greater model resolution as well as the use of new

archiving and backup services for historical analysis available in remote data centers in the cloud fashion. Moreover, major hardware vendors promote highly parallel, many-core and power-efficient computing devices, focusing on low power computing cores rather than increasing their complexity and clock frequency, as well as new computing approaches such as General Purpose Graphics Processing Unit (GPGPU) based computing, Cell B.E. or Field Programmable Gate Array (FPGA). These new architectures bring the current tendency to increase performance by parallelism instead of clock frequency. This means that increased performance must come from multi-chip, multi-core or multi-context parallelism. Flynn's taxonomy defines four levels of parallelism in hardware (BRODTKORB et al., 2010):

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

In addition, two subdivisions of MIMD are defined:

1. Single Program Multiple Data (SPMD)
2. Multiple Program Multiple Data (MPMD)

However, to benefit from modern computing architectures, applications and data structures have to be adapted properly. In other words, legacy applications simply cannot take full advantage of the new computing hardware as they have to be often rewritten or implemented from scratch in a multi-threaded or multi-process manner to take full advantage of the hardware. (KUROWSKI; KULCZEWSKI; DOBSKI, 2011)

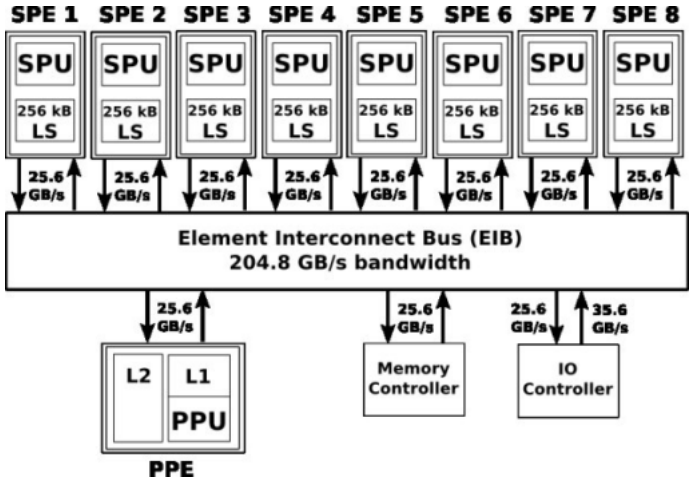
3.1 Architectural characteristics

In this section, some features of Cell BE, FPGA and GPU architectures are detailed.

The single Cell Broadband Engine Architecture (CBEA) chip, illustrated in Figure 3.1, is a multi-core microprocessor, it means that is a micro-architecture that combines a general-purpose Power Architecture core of modest performance (also called Power Processing Element - PPE in figure 3.1) with streamlined co-processing elements (Specialized Processing Element - SPE). It consists of a traditional CPU core and eight SIMD accelerator cores. It is a very flexible architecture, where each core can run separated programs in a MPMD fashion and communicate through a fast on-chip bus (EIB). Its main design criteria has been made to maximize performance while consuming a minimum amount of power. It greatly accelerates multimedia and vector processing applications, as well as many other forms of dedicated computation.

A field-programmable gate array (FPGA), illustrated in Figure 3.2, is an integrated

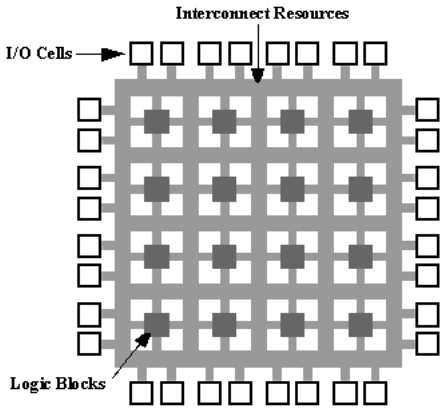
Figure 3.1: One single-chip CBEA with a CPU core and eight SIMD accelerator cores.



Source: (SCHARFE; PIELOT; SCHREIBER, 2010).

circuit designed to be configured by a customer or a designer after manufacturing—hence "field-programmable". Consisting of an array of logic blocks in combination with a standard multi-core CPU. FPGAs can also incorporate regular CPU cores on-chip, making it a heterogeneous chip by itself. The FPGA configuration is generally specified using a hardware description language (HDL), similar to the one used for an application-specific integrated circuit (ASIC). They offer fully deterministic performance and are designed for high throughput, for example, in telecommunication applications.

Figure 3.2: A diagram of the FPGA architecture.



Source: cursos.olimex.cl/fpga/

In contrast to modern CPUs, which follow the multiple instruction, multiple data architecture, graphics processing units (GPUs) are built as massively parallel processors using a single instruction, multiple data (SIMD) architecture. Driven primarily by the computer gaming industry, commodity GPUs have become powerful and highly parallel computation devices. In recent years, their increased programmability has made them

attractive for general purpose on computation, and GPU manufacturers have been improving the ability of GPUs to perform such computations efficiently. Arguably, a solution to existing efficiency problems in uncertainty environmental modeling is the use of powerful GPUs instead of effective accelerators for the problems that map well to a SIMD architecture (TRISTRAM; HUGHES; BRADSHAW, 2014). The architecture of the NVIDIA GK100 is illustrated in figure 3.3. This device consists in 13 multiprocessor that shares an L2 cache and has a L1 cache integrated to share data among all the proccesing elements contained within them. It also contains a global DDR5 memory.

Figure 3.3: A diagram of the GPU architecture.

Source: http://techreport.com/review/22989/a-brief-look-at-nvidia-gk110-graphics-chip

3.2 GPU Computing

The first GPUs were designed as graphics accelerators cards, supporting only specific fixed-function pipelines. Starting in the late 1990s, the hardware became increasingly programmable. In the next years, artists and game developers weren't the only ones doing ground-breaking work with the technology: Researchers were tapping on its excellent floating point performance. The General Purpose GPU (GPGPU) movement had dawned.

But GPGPU was far from easy back then, even for those who knew graphics programming languages such as OpenGL. Developers had to map scientific calculations onto problems that could be represented by triangles and polygons. This start to change when a group of Stanford University researchers set out to re-imagine the GPU as a "streaming processor."

In 2003, a team of researchers led by Ian Buck unveiled Brook, the first widely adopted programming model to extend C with data-parallel constructs. Using concepts such as streams, kernels and reduction operators, the Brook compiler and runtime system exposed the GPU as a general-purpose processor in a high-level language. Most impor-

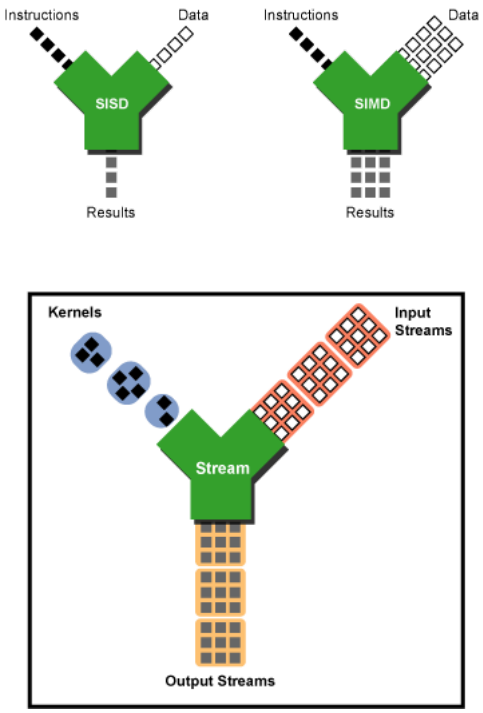
tantly, Brook programs were not only easier to write than hand-tuned GPU code, they were seven times faster than similar existing code. (NVIDIA, 2013)

Stream processing is a computer programming paradigm, related to SIMD, that allows some applications to more easily exploit a limited form of parallel processing. Such applications can use multiple computational units without explicitly managing allocation,synchronization, or communication among those units.

The stream processing paradigm simplifies parallel software and hardware by restricting the parallel computation that can be performed. Given a set of data (a stream), a series of operations (kernel functions) is applied to each element in the stream. Commonly one kernel function is applied to all elements in the stream as showed in figure 3.4.

Kernel functions are usually pipelined, and local on-chip memory is reused to minimize external memory bandwidth. Since the kernel and stream abstractions expose data dependencies, compiler tools can fully automate and optimize on-chip management tasks. Stream processing hardware can use score boarding, for example, to launch a direct memory access (DMA) at runtime, when dependencies become known. The elimination of manual DMA management reduces software complexity, and the elimination of hardware caches reduces the amount of the area not dedicated to computational units such as Arithmetic Logic Units. However with the advance of technology, the size of processing elements are reduces and the cache are present in the newer devices.

Figure 3.4: SISD, SIMD, and stream processing compared



Source: dmi.unict.it/ bilotta/gpgpu/notes/05-gpgpu-history.html

GPU devices are now attracting more attention because they can accelerate digital

terrain analysis in a more efficient and economical way than multi-core CPUs in single PCs or than clusters and were used to compare performance with traditional CPU parallelism (QIN; ZHAN, 2012) (LIMA, 2014). Modern GPU computing enables application programmers to exploit parallelism using new parallel programming languages such as CUDA and OpenCL and a growing set of familiar programming tools, leveraging the substantial investment in parallelism, which high-resolution real-time graphics require. For a better performance in NVIDIA GPU's, it is necessary to rewrite the model's code to Compute Unified Device Architecture (CUDA).

CUDA is a parallel computing platform and programming model that enables dramatic increases in computing performance by harnessing the power of the GPU. CUDA gives program developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the GPUs can be used for general purpose processing, the GPGPU approach. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread quickly.

The GPU architecture provides large memory bandwidth and floating point operations per seconds (FLOPS) when compared to conventional CPU. Using high-level languages, GPU-accelerated applications run the sequential part of their workload on the CPU – which is optimized for single-threaded performance – while accelerating parallel processing on the GPU.

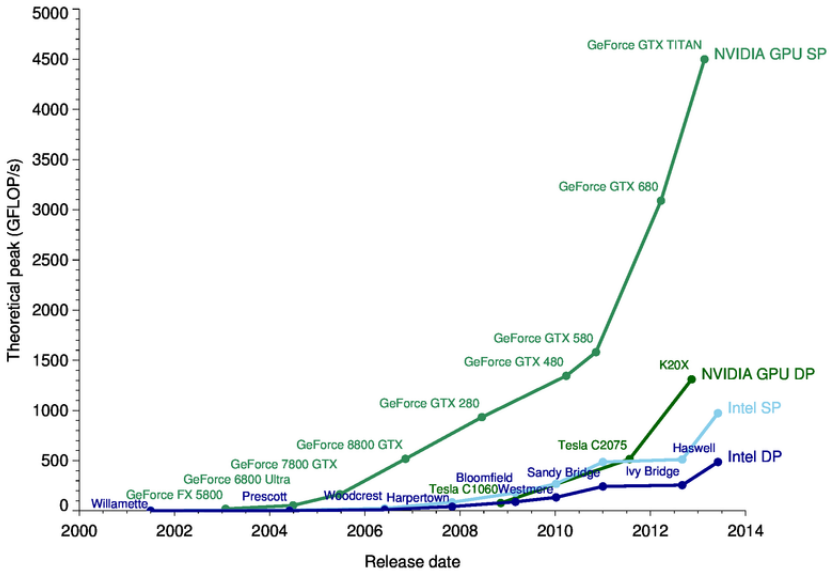
For example, NVIDIA Tesla GK110B achieves 288 GB/s of memory bandwidth compared to a 3.9 GHz Haswell family, Intel Core i7-4770K unit whose theoretical peak memory bandwidth is 25.6 GB/s. NVIDIA Tesla GK110B can provide 1.43 TFlop of double precision throughput against 40.11 GFlop of Intel Core i7-4770.

The figure 3.5 shows a comparative of theoretical peak performance of GPU's and CPU's expliciting the huge advantage that modern GPU obtained opposite to CPU's even when comparing double precision performance from GPU with single precision from CPU.

3.3 Conclusion of Heterogeneous Architectures

Among the studied architectures in this chapter, the GPU architecture was chosen due it great performance in large data set. In the next chapter will be studied the model in a mathematical way to understand the behavior of used set of equations.

Figure 3.5: Comparative of theoretical peak power of single and double precision for CPU and GPU



Source: michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html

4 TWO-DIMENSIONAL FLOOD INUNDATION MODELING

Since the methods to predict floodplain inundations proposed by (ZANOBETTI et al., 1970), such methods have become really popular. Along all these years, several improvements were made in these methods. The most recently was made by (BATES; HORRITT; FEWTRELL, 2010) giving a more efficient formulation of the shallow water for fllod inundation modeling. Following will be showed an overview of these equations and its evolution as well as the concepts used in the LISFLOOD model, which was used to validate this work. LISFLOOD is a two-dimensional hydrodynamic model specifically designed to simulate floodplain inundation in a computationally efficient manner over complex topography developed in University of Bristol.

4.1 Initial formulation

Initially, the methods to predict floodplain inundation discretized floodplains into irregular polygonal units representing surface areas between 1 and $10\ km^2$ of natural storage cell and calculated the fluxes of water between these according to some uniform flow formulas such as Manning's equations.

For many models, flows are calculated using some form of the one dimensional Saint-Venant equations, and when bankfull flow is exceeded, water is routed into and between the floodplain storage units. More recently the availability of powerful computing and detailed floodplain topography available through remote sensing (e.g. LiDAR data) (MARKS; BATES, 2000) has allowed to move from large, irregular storage units of the floodplain discretization as a fine spatial resolution regular grid with cell areas between 10^{-3} and $10^{-2}km^2$.

In this formulation each cell inside the grid is a storage area, for which the mass balance is updated at every time step according to the fluxes of water into and out of each cell. Similar to polygonal storage cell models, the flux is calculated analytically using uniform flow formula, but with the advantage of higher resolution predictions and removal of the need for the modeler to make explicit decisions about the location of storage compartments and the linkages between these.

Such models solves a continuity equation relating flow into a cell and its change in volume:

$$\frac{\Delta h}{\Delta t} = \frac{\Delta Q}{\Delta x \Delta y} \tag{4.1}$$

where h is the water free surface height [m], t is the time [s], Q is the volumetric flow rate and Δx and Δy are the cell dimensions. These models also solves a flux equation for each direction where flow between cells is calculated according to Manning's law:

$$Q_x^{i,j} = \frac{h_{flow}^{5/3}}{n} \left(\frac{h^{i-1,j} - h^{i,j}}{\Delta x} \right)^{1/2} \Delta y \tag{4.2a}$$

$$Q_y^{i,j} = \frac{h_{flow}^{5/3}}{n} \left(\frac{h^{i,j-1} - h^{i,j}}{\Delta y} \right)^{1/2} \Delta x \tag{4.2b}$$

where h i,j is now the water free surface height [m] at the node (i, j), n is the Manning's friction coefficient [$m^{-1/3}s$], and Q_x and Q_y describe the volumetric flow rates between floodplain cells [$m^3 s^{-1}$]. The flow depth, h_{flow}, represents the depth through which water can flow between two adjacent cells and is defined as the difference between the highest water free surface in these cells and the highest bed elevation of them. These equations are solved explicitly using a finite difference discretization of the time derivative term:

$$\frac{h_{t+\Delta t}^{i,j} - h_t^{i,j}}{\Delta t} = \frac{Q_{x,t}^{i-1,j} - Q_{x,t}^{i,j} + Q_{y,t}^{i,j-1} - Q_{y,t}^{i,j}}{\Delta x \Delta y} \tag{4.3}$$

where h_t and $Q_{*,t}$ represent depth and volumetric flow rate at time t respectively for x and y direction, and Δt is the model time step which is held constant throughout the simulation.

Instead of computing the numerical solution of the full shallow water equations, the storage cell formulation have the advantage that fluxes are calculated analytically having the computational costs per time step potentially much lower. Such methods also interface readily with newly available remotely sensed terrain data which typically arrives in the form of a regular grid and for this reason the number of research codes based on these techniques has proliferated over the last decade (HUNTER et al., 2007).

Although this method can only be applied to gradually varied flows and does not include inertia or the ability to capture supercritical effects, for many floodplain inundation problems the representation is appropriate. Such models were originally conceived for application at coarse grid resolutions (25–100 m) and early applications showed that at these scales there was a distinct computational advantage over full solutions of the 2D Saint–Venant equations (HORRITT; BATES, 2001).

However, some concerns appear. Unless the constant time step used to solve Eq. 4.3 was small, simulations with storage cell models quickly developed 'chequerboard' type

instabilities as all the water in a particular cell drained into the adjacent ones in a single and large time step and at the next time step, this situation would reverse and all the water would flow back. To solve this, many modelers include a 'flow limiter' to prevent too much water leaving a given cell in one time step. Unfortunately flow-limited storage cell models often showed very little sensitivity to floodplain friction and their results were strongly dependent on the grid size and time step selected.

A solution to this problem was provided by (HUNTER et al., 2005) based on adaptive time-stepping. This approach aims to remove the need to invoke the flow limiter by finding the optimum time step i.e. large enough for computational efficiency and small enough for stability at each iteration. This optimum time step is obtained using an analysis of the governing equations and their analogy to a diffusion system which gives the following expression for time variation:

$$\Delta t = \frac{\Delta x^2}{4} min\left(\frac{2n}{h_{flow}^{5/3}}\left|\frac{\partial h}{\partial x}\right|^{1/2}, \frac{2n}{h_{flow}^{5/3}}\left|\frac{\partial h}{\partial y}\right|^{1/2}\right) \tag{4.4}$$

To implement this scheme is necessary to search the entire domain for the minimum time step value and using this value to update h in Eq. 4.3. The time step will be adaptive and change during the course of a simulation, but still uniform in space at each time step. The adaptive time step model showed a better absolute performance (HUNTER et al., 2006) than the classical fixed time-step version at this spatial resolution, but at approximately six times the computational cost. In particular the adaptive model appeared able to simulate floodplain wetting and drying more realistically.

Even with the success of this new set of equations, the timestep found with the Eq. 4.4 has a fundamental problem when decreasing the grid size, the timestep reduces quadratically. For applications with grid sizes in the range 25 to 100 m, this cause a 2 - 10 times increase in simulation time which could be offset through advances in processor speed (BATES; ROO, 2000). However, applications of hydraulic models to simulate urban areas usually uses finer resolution grids (less than 10 m) increases costs by several orders of magnitude such that at these scales adaptive time step storage cell codes actually proved slower than full 2D solutions of the shallow water equations (FEWTRELL et al., 2008).

Adaptive time step storage cell codes are therefore incompatible with the fine spatial resolution grids increasingly required for urban flood modeling. The best solution in the last years was to invoke a flow limiter, but this leads to a poor representation of flow dynamics. Once for fine grids full 2D models gives shorter simulation times, for practical applications they were only able to treat small (less than $1km^2$) areas at the required detail level. To allow wide area urban flood modeling at fine spatial resolution a new hydraulic model formulation is required. With this objective, a new set of equations for adaptive time step storage cell were formulated being possible to overcome the quadratic dependency solving analytically the Eq. 4.2 with approximately the same computational

cost. The new scheme therefore retains all the computational advantages of storage cell models over full 2D codes whose equations require expensive numerical solution, yet with none of the previous listed disadvantages (BATES; HORRITT; FEWTRELL, 2010).

4.2 Inertial formulation of the shallow water equations

In the urban model benchmarking study of (HUNTER et al., 2008) became clear that the lack of mass and inertia in Eq. 4.2 was the main reason why storage cell models required the strict time step control imposed by Eq. 4.4. In this equation flux is a function of gravity and friction and this flux is overestimated specially in areas of deep water where there is only a small free surface gradient. However, in a gradually varying shallow water flows the effect of inertia is to reduce fluxes between cells, avoiding the use of flow limiters. In this study, it was proposed that the solution was to modify explicit storage cell codes to include inertial terms that allow the use of a larger stable time step, being the simulation with quicker run times.

The starting point for derivation of new equation is the momentum equation from the quasi-linearized one-dimensional Saint–Venant or Shallow Water equations:

$$\underbrace{\frac{\partial Q}{\partial t}}_{acceleration} + \underbrace{\frac{\partial}{\partial x}\left[\frac{Q^2}{A}\right]}_{advection} + \underbrace{\frac{gA\partial(h+z)}{\partial x}}_{waterslope} + \underbrace{\frac{gn^2Q^2}{R^{4/3}A}}_{frictionslope} \tag{4.5}$$

where Q $[m^3s^{-1}]$ is the discharge, A is the flow cross section area $[m^2]$, z is the bed elevation $[m]$, R is the hydraulic radius $[m]$, g is the acceleration due to gravity $[ms^{-2}]$ and all other terms are defined as above.

For many floodplains flows advection is relatively unimportant (HUNTER et al., 2007) so this term can be neglect. Assuming a rectangular channel, dividing through by a constant flow width, w $[m]$ and approximate the hydraulic radius (R) with the flow depth (h) is possible to discretize Eq. 4.5 to obtain the equation in terms of time step:

$$\left(\frac{q_{t+\Delta t} - q_t}{\Delta t}\right) + \frac{gh_t\partial(h_t+z)}{\partial x} + \frac{gn^2q_t^2}{h_t^{7/3}} = 0 \tag{4.6a}$$

$$q_{t+\Delta t} = q_t - gh_t\Delta t\left[\frac{\partial(h_t+z)}{\partial x} + \frac{n^2q_t^2}{h_t^{10/3}}\right] \tag{4.6b}$$

Equations 4.6 exhibit the same equation. Eq. 4.6a show the discretized version while Eq. 4.6b shows the explicit discretized Saint-Venant equation with respect to the time step.

To improve the stability of equation, since instabilities may still arise at shallow depths when the friction term becomes large, replaces a q_t in the friction term by a $q_{t+\Delta t}$ leads the equation to a linear equation in the unknown $q_{t+\Delta t}$ wich has some of the improved convergence properties. With some arrangements detailed in (BATES; HORRITT; FEWTRELL,

2010) we have the final equation to compute the flux:

$$q_{t+\Delta t} = \frac{q_t - g h_t \Delta t \frac{\partial (h_t + z)}{\partial x}}{1 + g \Delta t n^2 q_t / h_t^{7/3}} \tag{4.7}$$

The advantage of this formulation is that since the acceleration term is included, the water is modeled with some mass and it is therefore less likely to generate the rapid reversals in flow which lead to a chequerboard oscillation. Shallow water wave propagation will also be represented, rather than the diffusive behavior typical of previous storage cell models. The enhanced stability of Eq. 4.7 stems from the increase in the denominator as the friction term increases, forcing the flow to zero, as would be expected for shallow depths.

Although, Eq. 4.7 still subject to the Courant–Friedrichs–Levy condition:

$$C_r = \frac{V \Delta t}{\Delta x} \tag{4.8}$$

where the non-dimensional Courant number, C_r , needs to be less than 1 for stability and V is a characteristic velocity [ms^{-1}]. In the case of a shallow water flow where advection is ignored this characteristic velocity is equals to celerity of a long wave with small gravity amplitude:

$$\sqrt{gh} \tag{4.9}$$

Eq.4.8 gives a necessary but not sufficient condition for model stability, and is used to estimate a suitable model time step at $t + \Delta t$:

$$\Delta t_{max} = \alpha \frac{\Delta x}{\sqrt{g h_t}} \tag{4.10}$$

where α is a coefficient in the range 0.2–0.7 used to produce a stable simulation for most floodplain flow situations calculated with experimental tests. This time step is typically 1–3 orders of magnitude larger than the stable time step for the purely diffusive scheme of Eq. 4.4. Moreover, within this range, proportionally larger time step differences become apparent as the grid size decreases, as for Eq. 4.10 time step scales with $1/\Delta x$ rather than $(1/\Delta x)^2$.

4.3 Improving the stability of inertial formulation

With this set of new equations, several models adopted then resulting in a higher use and analysis of it. One of the mayor point was a remaining unsolved stability problems. (BATES; HORRITT; FEWTRELL, 2010) when published the set of equations related that considerable numerical instabilities arise at low friction scenarios. These instabilities represented a huge obstacle to application that simulates flood in urban areas, where

relatively smooth surfaces are found.

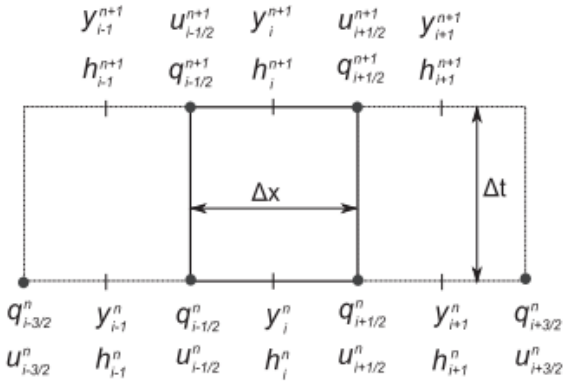
In 2012, (ALMEIDA et al., 2012) proposed a modification in equation of the model. Instead of using the previous flow and the information of water depth of the neighbors of border cell (where is computed the flow), they propose to use a centered finite difference method taking both water depth cells data besides the 3 related flow borders.

The mathematical difference is how to compute de partial derivative $\frac{\partial q}{\partial t}$. The first term on the left side on eqution 4.6a shows the previous scheme to compute the partial difference, where only time changes. For simplicity, we can rewrite equation 4.6a as:

$$\frac{\partial q}{\partial t}|^{i-1/2} = \frac{q_{t+\Delta t}^{i-1/2} - q_t^{i-1/2}}{\Delta t} \tag{4.11}$$

introducing the superscript notation to denote the spatial location from previous computed flux. Figure 4.1 shows how the model space are now considered, where $y^n = h_t + z$, $y^{n+1} = h_{t+\Delta t} + z$ and y_i^{n+1} denotes the next iteration on position i.

Figure 4.1: Spatial representation of the finite difference using three points to increase the model stability



To introduce the centered finite difference, $q_{t+\Delta t}^{i-1/2}$ will be estimated as a weighted average of $q_t^{i-1/2}$ and q_t values in the two neighboring interfaces. With this changes, equation 4.12 will be replaced by:

$$\frac{\partial q}{\partial t}|^{i-1/2} = \frac{q_{t+\Delta t}^{i-1/2} - \left[\theta q_t^{i-1/2} + \frac{(1-\theta)}{2}(q_t^{i-3/2} - q_t^{i+1/2})\right]}{\Delta t} \tag{4.12}$$

which also replace equation 4.7 for:

$$q_{t+\Delta t}^{i-1/2} = \frac{\left[\theta q_t^{i-1/2} + \frac{(1-\theta)}{2}(q_t^{i-3/2} - q_t^{i+1/2})\right] - g h_t \Delta t \frac{\partial (h_t+z)}{\partial x}}{1 + g\Delta t n^2 q_t^{i-1/2}/h_t^{7/3}} \tag{4.13}$$

With this modifications, it was expected and related an increase in computational cost, however was noticed a considerable increase in model stability, allowing the computation with larger timestep, which means even with a bigger cost, it is possible to achieve the

final state of simulation faster.

4.4 Conclusion of two-dimensional flood inundation modeling

The modifications applied in the model equations are helpful to understand all possible bottlenecks in serial and parallel executions. The equation used to produce the stable model have more parameters consequently have a higher computational cost. All the conclusions made along this chapter allow us to move on and study all optimization techniques that can be applied to accelerate this model.

Some techniques will be detailed in the next chapter to make possible the comparison between CPU and GPU versions of this model.

5 PROPOSAL OF OPTIMIZATIONS TO LISMIN MODEL FOR GPU ENVIRONMENT

With the advance of stream architectures, several new features are developed with the purpose of accelerate the computational time. Previous results show the great potential of GPUs to accelerate flood models (KALYANAPU et al., 2011), however one of the counter point to migrate an application to a different architecture or even update the version of one GPU, is that sometimes, different versions of running framework made the model code incompatible.

This chapter will address the details of the migration of a reduced version of LISFLOOD, called LISMIN, from a serial code to a parallel code able to run in GPUs that support the NVIDIA CUDA framework version 5.5 and one final optimization that requires the version 6.5 of the same framework.

5.1 Base model and pseudo-code

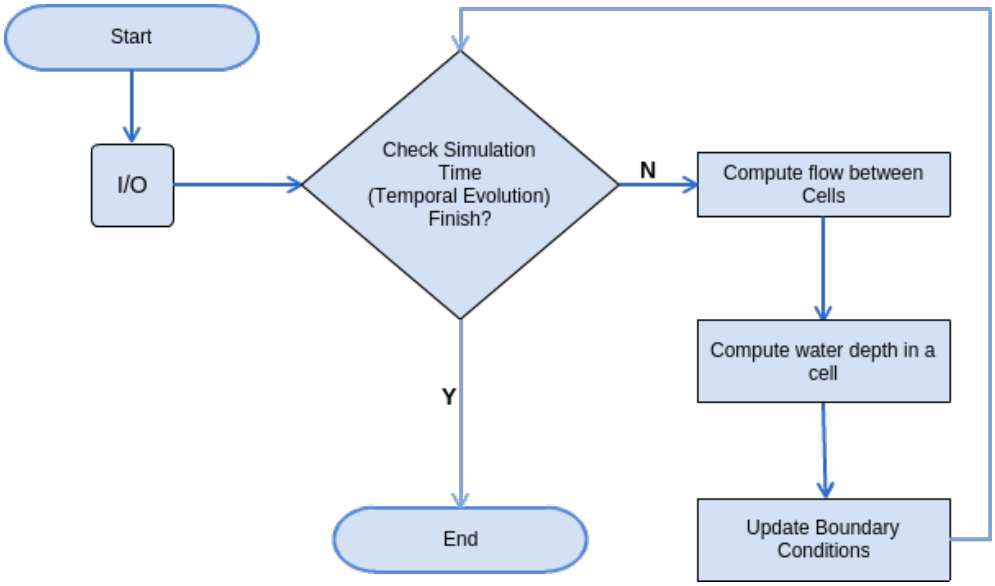
The starting point of the optimizations is to analyze the models pseudocode to understand its behavior. The pseudocode is showed in Algorithm 1. The model can be classified in five phases.

1. I/O
2. Temporal Evolution
3. Computation of the flow between cells
4. Computation of the total amount of water in a cell
5. Update boundary conditions

Algorithm 1 and Figure 5.1 give us a high level of the model behavior expliciting the five steps listed above.

The I/O phase in models that run only in CPU consist basically in read setup and initial state files as well as write the intermediate state of simulation for further analyze. However in a stream based architecture, the I/O is present in every data transfer between the CPU and in this case the GPU.

Figure 5.1: Flowchart of model execution steps.



Source: The Author

Common simulations have thousand iterations, being the I/O a constant part and sometimes one of the main areas of bottlenecks in simulation performance. The main files read by this model are related to water depth (referenced as H in codes), terrain topology (referenced as Z in codes) and the flow in x and y-direction (referenced as Qx and Qy in codes). Each file will generate an independent matrix where the dependency is managed by the model. Figure 5.2 shows how the water depth file is interpreted in the model.

The relation among physical variables, like water depth and flow are related as following. Looking at figure 5.3 it is possible to differentiate 3 main elements. *(a)* the green columns (vertical lines); *(b)* the yellow rows (horizontal lines); and *(c)* the blue dots. Each of these elements will generate a matrix.

For convenience and easiness, let m the number of rows in the grid and n the number of columns. In case *(a)*, each segment of a column that are a common border for 2 cell, will generate an element of the matrix related to the flow in y-direction. So each green column will contribute with m elements in matrix Qy. An analogous reasoning can be applied to generate the matrix Qx, where each line will contribute with n elements. The blue dots are an average of the water level inside a cell and each one will be interpreted as a value in the matrix H and the terrain height where each one will be interpreted as a value in the matrix Z. With these 4 matrices, the model manages the water flow along the simulation, reading the initial data and write files whenever be necessary for a further visual reproduction.

With the use of GPU to compute, every time that a data need to be updated in a different area of the actual computation a data transfer is necessary, so techniques to hide the transfer time or to reduce this time are useful optimizations.

Algorithm 1 Model Pseudocode

```
1:  procedure MAIN
2:      Read data files and setup model
3:      while execution time < total time do
4:          Compute new time step
5:          Compute the flow in x-direction
6:          Compute the flow in y-direction
7:          Update cells water depth
8:          Apply boundary condition
9:          Increment execution time
10:         if Condition to output = true then
11:             Write out updated data files
12: End simulation
```

Source: The Author

The temporal evolution is a common part of all environmental model. This control is usually simple, however the process to compute the time step to be taken can vary in complexity and affect the total time. Usually this control is observed in a loop function incremented at the end of water movement phase, restricted to a simulation time condition. When a setted time is reached, the simulation is complete and the program ends.

The next two phases are strongly dependent of the resolution method used. In this work and model, the domain is splited in a grid and the flow between two cell is computed for all cells in domain. In this model all flow are computed in two steps, one for the flow in x direction and another for y direction once the Saint-Venant equations are unidimensional. To solve this, it is checked if there is a flow between two adjacent cells (figure 5.3a) and in positive case, the flow is computed for x (figure 5.3b) and further for y direction (figure 5.3c). When all cells have their flow updated, a matrix with these flows is stored to update the water depth in next phase and for the flow computation in next iteration.

After that, the next kind of phase begin where all the flow computed in the previous phase are gathered, adding both incoming and outcome water to compute the new depth of the water in every cell.

And finally the last kind of phase is when the boundary conditions are applied to model. In this test case the water is added in one single cell and spreads over the ground. Being a single cell update, a better performance is expected running this step in CPU instead in GPU even with transfer time.

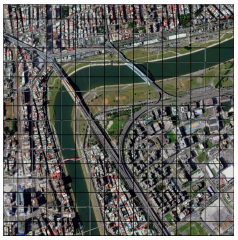
5.2 Serial Code

The first version of the model was the serial version, that was also used as baseline for performance to compare with the parallel versions. The serial code was developed in

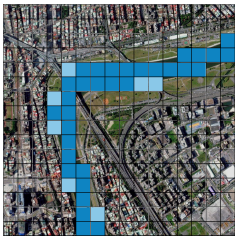
Figure 5.2: Steps for generation of matrix H, representing the water depth of each area in analyzed map area.

(a) It is defined a terrain to analyze

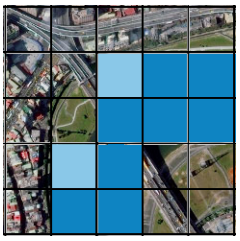
(b) The selected domain are split in several cell according to required resolution



(c) The water is detected and the depthness is computed



(d) Each area with water are aggregate to determine the average of depthness for this region



(e) A matrix is generated where each cell has a value corresponding to related depthness of an area in the map.

0	0	0	0	0
0	0	1	2	2
0	0	2	2	2
0	1	2	0	0
0	2	2	0	0

Source: The Author

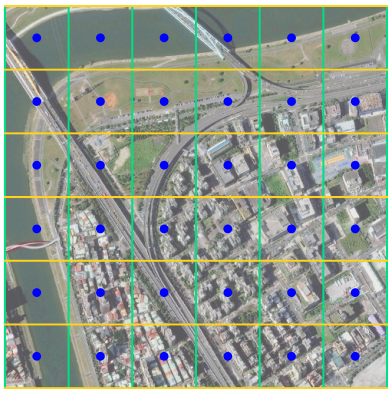
C++ due the easiness to parallelize with OpenMP and further to recode in CUDA.

In the serial version, the input data phase is a single phase at the beginning of simulation. In this phase the digital elevation model (DEM) file is loaded as well as the initial water state. Once all simulation was computed in CPU, there is no need to transfer data between CPU and an external device.

All versions implement the same inertial solver, using the equation 4.10. In this version, this equation was implemented as follow:

Being H the matrix related to the water depth, the first step is to find the maximum value in this matrix. To avoid a division by zero, when all domain is dry, it was defined a minimum water depth, that was used when the maximum level of water is lower than this constant (here, we used 0.001). After the stable time step was found, to avoid lose this stability, some checks are made to guarantee this. This stable time step is included in equation 4.10 and are limited by the relation 4.8. Finally, it is checked if when the new time step is added, the simulation time will not exceed the total defined time and if be overtaken, the time step will set as the difference between the final time and the actual time. An explained algorithm is showed in Algorithm 2.

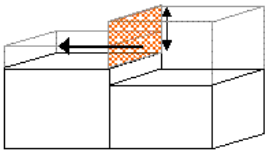
Figure 5.3: Visual model elements responsible for generation of simulation's matrices.



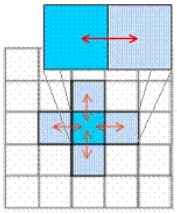
Source: The Author

Figure 5.4: Steps for flow computing.

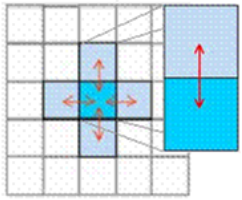
(a) Check if there is flow between two cells



(b) Compute the flow in x-direction



(c) Compute the flow in y-direction



Source: bristol.ac.uk/geography/research/hydrology/models/lisflood/

Algorithm 2 Serial Implementation (Timestep Computation)

1: **procedure** COMPUTE TIMESTEP
2: $maxH = max\_element(H, H + (rows * cols))$; ▷ find the max value in H
3: $maxH = max(maxH, 0.001)$; ▷ Avoiding null water depth
4: $localTimeStep = alpha * dx/sqrt(g * maxH)$; ▷ Compute new time step
5: $Tstep = min(Tstep, localTimeStep)$; ▷ Keeping the already know stable time step
6: $Tstep = min(maxDtFlood, Tstep)$; ▷ time step must be lower than a defined upperbound
7: $Tstep = min(Tstep, totalTime - executionTime)$; ▷ time step do not overcome total time

Source: The Author

To compute the flow between two adjacent cells, all domain is analyzed. Having one cell as base, the relative water depth, i.e. the water depth added the terrain height, is compared with the adjacent cell. If there is a difference, it means that a flow between these cells will occur and this flow is calculated based on Equation 4.7. Algorithm 3 shows how to compute the flow in x-direction and for y-direction the code is analogous.

In this version, all discharges in x-direction are calculated and when it is complete are calculated the y-direction flow. The order was set arbitrarily, however as the matrices that stores the flow states are independent for both direction, there is no difference computing x-direction before y-direction or vice versa.

Algorithm 3 Serial Implementation (Flow Discharge Computation)

```
1: procedure COMPUTE FLOW DISCHARGE
2:     for i = 1 to i ≤ rows do
3:         for j = 1 to j ≤ cols do
4:             flow = max(z0 + h0, z1 + h1) − max(z0, z1); ▷ z is the terrain height and h is the water depth
5:             if flow > 0 then
6:                 Sf = −(z0 + h0 − z1 − h1)/dx; ▷ 'Sf' represent the friction slope coefficient
7:                 QoldPrev = Qx[prevCell]/dx; ▷ Take previous flow at position i-1
8:                 Qold = Qx[baseCell]/dx; ▷ Take previous flow at position i
9:                 QoldNext = Qx[nextCell]/dx; ▷ Take previous flow at position i +1
10:                Q = ((theta ∗ Qold + ((1 − theta)/2) ∗ (QoldPrev + QoldNext)) − g ∗ flow ∗ Tstep ∗ Sf)/(1 + g ∗ Tstep ∗ FPn^2 ∗ |Qold|/hflow^(7/3)) ∗ dx; ▷ Equation 4.7
11:                Qx[baseCell] = Q; ▷ Store new flow
```

Source: The Author

After compute the flow in both directions for all cells in domain an extra functions is executed gathering this information to update the water depthness, also for all cells. This new depth depends of all input and output of water that cross cell borders as well as the time step, that represents the amount of time that the related flow cross the cell area. The Algorithm 4 shows how it is updated. The acronyms rb, lb, tb and bb means, respectively, right border, left border, top border and bottom border.

Algorithm 4 Serial Implementation (New Water Level Computation)

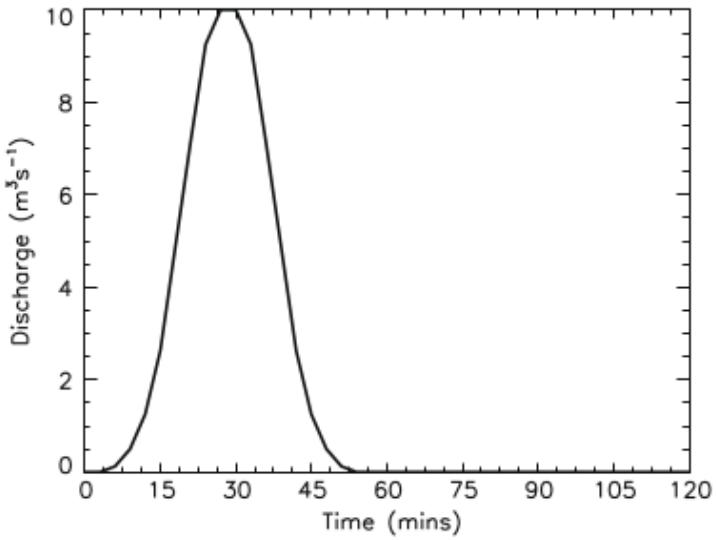
```
1: procedure UPDATE WATER LEVEL
2:     for i = 1 to i ≤ rows do
3:         for j = 1 to j ≤ cols do
4:             vol = Tstep ∗ (Qx[rb] − Qx[lb] + Qy[tb] − Qy[bb]) ▷ Compute flow of the 4 borders
```

Source: The Author

Finishing the water movement, boundary conditions are applied to model and will be part of the model in the next iteration. In this work, a variable amount of water are added in a single point of domain following the experiment related in (BATES; HORRITT; FEWTRELL, 2010). Figure 5.5 shows the hydrograph for the simulated event.

This behavior was generated using a linear interpolation of a list of discharges in fixed given times. Algorithm 5 shows how this condition is applied.

Figure 5.5: Event hydrograph simulated in test case.



.

Source: (BATES; HORRITT; FEWTRELL, 2010)

Algorithm 5 Serial Implementation (Boundary Conditions)

1: **procedure** APPLY BOUNDARY CONDITION
2: $boundary\_i = boundary\_i\_old$;
3: **while** $boundary\_i \leq nbdy$ **do** ▷ $nbdy$ is the number or data in boundary file
4: **if** $boundaryList[boundary\_i] < executionTime$ **then**
5: $deltaBoundary = boundaryList[boundary\_i] - boundaryList[boundary\_i - 1]$;
6: $a1 = (executionTime - boundaryList[boundary\_i - 1])/deltaBoundary$;
7: $a2 = 1 - a1$;
8: $H[location] = Tstep * (a1 * boundaryWaterAmount[boundary\_i] + a2 * boundaryWaterAmount[boundary\_i - 1])/dx$;
9: $boundary\_i_old = boundary\_i$;
10: $boundary\_i = nbdy$;
11: $boundary\_i++$

Source: The Author

After all procedures finished, the execution time is updated with the actual time step and a check is made. If 1% more of total simulation was achieved since the last record of output files, a new set of files are stored in disk saving the state of simulation, else there is no output until the next per cent be achieved and simulation goes on with the data only in memory.

5.3 CPU Parallel Code

In order to optimize the execution time of this simulation, one of the easiest way to parallelize code is the OpenMP API. OpenMP is a set of compiler directives and callable runtime library routines that extend Fortran, C and C++ to express shared-memory parallelism (DAGUM; MENON, 1998).

To compare the performance achieved with the use of GPUs, it was also optimized the model to run with the openMP API, to do a 'fair' comparison of execution times. Another point to use this method of parallelism is due it easiness when compared with GPU to program the model. With use of compiler directives is possible to run the model with a large number of process aiming a better result.

In this work, it was possible to parallelize with this technique the computation of flows in both direction as well as the new water level for all cells in domain. For the flows computation was used a directive in external *for* loop to maintain the same thread with the maximum number of elements near together to maximize the cache hits. Algorithm 6 shows an generic call from the used directive. The difference between the call from flows is only the related matrix to read values, while for the water level update a new set of variables is necessary to compute it.

Algorithm 6 OpenMp Implementation

```
1: procedure PRAGMA DIRECTIVE
2:     #pragma omp parallel for private(private vars) shared(shared vars)
3:     for i = 1 to i ≤ rows do
4:         for j = 1 to j ≤ cols do
5:             ...
```

Source: The Author

The temporal control and the application of boundary conditions, in this work are not parallelized, once a single variable is necessary to do this control and the value is shared by all process, however no parallel process change the value, i.e. the time step value. For the boundary condition, once only one point is changed, there is no reason to do this in parallel, however for other cases that are no concurrency to update several places, is indicated to use this technique too.

5.4 GPU Parallel Code

This section describes the five optimizations made in this work with utilization of a GPU card. They are:

1. GPU code recode
2. Block and Grid size analysis

3. Utilization of pinned memory
4. Utilization of stream for concurrent processing
5. Use of Unified memory

5.4.1 GPU code recode

The first technique is not a real optimization because for some programs, when executed inside a GPU card their performance is worst than when are executed only in CPU. However, this model has the main characteristic that incentive the use of GPU, the large repetition of a same function, not only a single instruction, in multiple data.

This recode brings the first high impact in development time once it is more difficult to parallelize a code using GPUs with CUDA than using CPUs with openMP. The first difference is the need to allocate memory also for GPU card and not only for CPU. More than that, it is necessary to manage where the data need to be to correctly compute the simulation. The data need to be transfered from CPU to GPU and vice versa whenever is necessary.

The algorithm 7 gives an example of these processes for the matrix H, however for all matrices that need to be in CPU and GPU need to have these functions called.

Algorithm 7 GPU Implementation (Allocation and Copy for matrix H)

```
1:  procedure CPU MALLOC
2:      H = new double[cols * rows]();
3:      ...
4:  procedure GPU MALLOC
5:      cudaMalloc((void**)&d_H, cols * rows * sizeof(double));
6:      ...
7:  procedure CPU TO GPU MEMORY COPY
8:      cudaMemcpy(d_H, H, [size], cudaMemcpyHostToDevice);
9:      ...
10: procedure GPU TO CPU MEMORY COPY
11:     cudaMemcpy(H, d_H, [size], cudaMemcpyDeviceToHost);
12:     ...
```

Source: The Author

The new version of code changes the method to manage the temporal evolution and execute the computation of flow discharge and update the water level in cells. For temporal evolution it was used the cuBLAS library to find the maximum elements inside the matrix H. The NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) library is a GPU-accelerated version of the complete standard BLAS library that delivers a better performance than the older libraries. It has a complete support for all 152 standard BLAS routines for single, double, complex, and double complex data types and other features (NVIDIA, 2014).

However, the temporal evolution management is still on CPU and the updated matrix H present in GPU memory will deliver his maximum value in GPU, hence is necessary copy this value back to CPU to calculate new time step. Once the routine to find the maximum element is optimized and the data to be transfered is only one element (with the size of a double), we already have an improvement in this function call. The remaining steps for temporal evolution stay the same from previous listed versions.

All function used to compute flow discharge and update the water level are converted to GPU kernels. Kernels are defined as C functions that, when called, are executed N times in parallel by N different CUDA threads, as opposed to only once like regular C functions. Each kernel receives the amount of threads that its allowed to use. This value is specified using a $<<<$ ... $>>>$ syntax between the function call and the parameters pass. The kernel calling in this model can be viewed in algorithm 8 and used as example to illustrate that. Variables *dimGrid* and *dimBlock* can have one, two, or three dimensions and are represented by structures that contains the values of block dimension and grid dimension of threads respectively.

Algorithm 8 GPU Implementation (Cuda kernels)

1: **procedure** KERNEL CALL
2: $calculateQx <<< dimGrid, dimBlock >>> (parameters)$
3: $calculateQy <<< dimGrid, dimBlock >>> (parameters)$
4: $updateWaterLevel <<< dimGrid, dimBlock >>> (parameters)$

Source: The Author

An illustration of the grid of thread blocks behavior is given in figure 5.6. This structure allow the programmer to have a visual idea about what is happening inside the GPU.

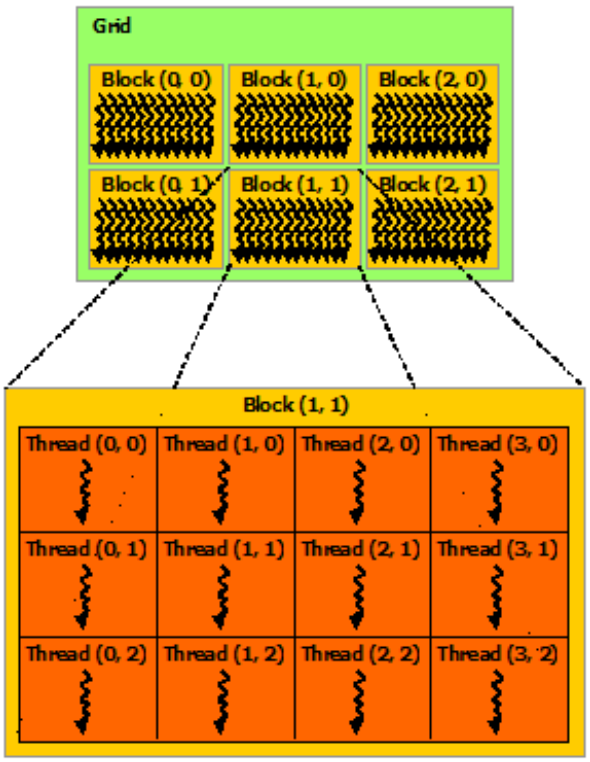
Inside these kernels, some changes were made in relation to standard C functions from previous versions. The nested *for* loops were replaced by a *check function* of the total threads reaching the kernel. This check function is an if clause that allow threads with a characteristic to start its computing. Having more than $rows * columns$ threads to execute this kernel, is necessary only to guarantee that only the right number of threads will work or, in case were exists less threads than $rows * columns$ is necessary share the work among the active and current threads. The control of which threads are allowed to execute the calculus is given by algorithm 9.

To apply the boundary condition in matrix H, it is necessary copy the matrix to CPU, apply the condition and then copy back to GPU for the next iteration.

5.4.2 GPU block management

In order to tune our model when running in GPU, one of the firsts optimization that will be shared with the next techniques, is to achieve the highest occupancy in a GPU multiprocessor (SMX). To do this, is necessary to study how many threads are available in

Figure 5.6: Grid of Thread Blocks



Source: arc.vt.edu/resources/software/cuda/

Algorithm 9 GPU Implementation (Thread Control)

```
1: procedure THREAD CONTROL INSIDE A KERNEL
2:     y = blockIdx.x * blockDim.x + threadIdx.x;      ▷ Column id
3:     x = blockIdx.y * blockDim.y + threadIdx.y;      ▷ Row id
4:     tid = x * blockDim.x * gridDim.x + y;           ▷ Thread id
5:     if tid < columns * rows then
6:         ...
```

Source: The Author

each block and how many blocks can be executed in GPU. In this study we use a NVIDIA TESLA K20, that allow address 2048 threads for each multiprocessor with no more than 1024 threads in each block and can run 16 thread blocks in a SMX. The cardinalities of the three different dimensions of each block must be defined on each CUDA kernel launch and for a given problem encoding, the different threadblock sizes and shapes can significantly affect the overall code performance (TORRES; GONZALEZ-ESCRIBANO; LLANOS, 2011). With this information is possible to analyze the best shape and size for our application and optimize them.

To avoid to spent unnecessary resource, three strategies as described in (TORRES; GONZALEZ-ESCRIBANO; LLANOS, 2011):

1. The number of threads per block must be a divisor of the number of maximum

threads per multiprocessor

2. The number of threads per block must be a multiple of the number of threads per warp, to fill them
3. The number of threads per block must be large enough to not generate more block per multiprocessor than the total allowed

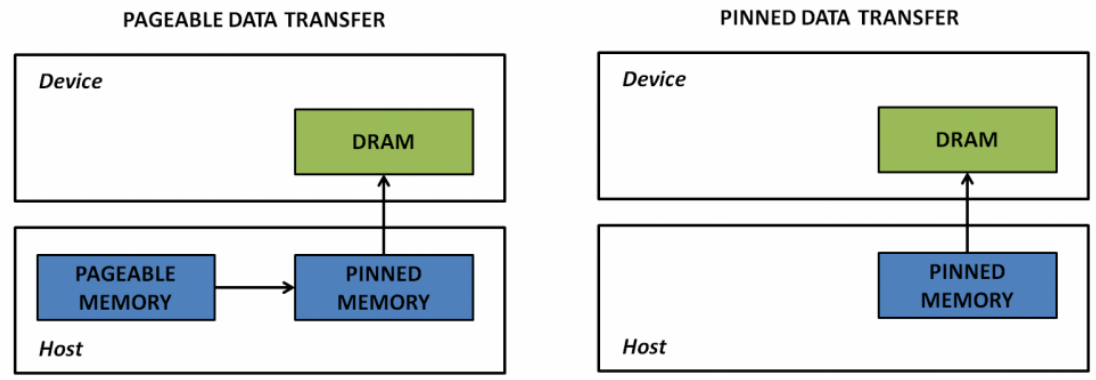
To meet these requirements we need to consider that the maximum threads per SMX in K20 is 2048. For the number of threads per block be a divisor, this number must be a power of 2, hence the number of threads possible considering item 1 are {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024}. 2048 is not an option due the fact that one block can have only 1024 threads inside of it. The item 2 reduces the number of possibilities dropping off the blocks with few threads, letting as possible option {32, 64, 128, 256, 512, 1024} for the number of threads per block. The third item limit the number of blocks inside a SMX. Once we have 16 block as the maximum number of blocks, it implies in each block need to have at least 128 threads, what generate the 16 blocks. After these three consideration we have the final possible option of the number of threads per block as {64, 128, 256, 512, 1024}.

5.4.3 Gpu code with pinned memory

One of the most knew bottleneck of GPU compute is the large amount of data that need to be transfered from CPU to GPU and vice versa. To allow programmers to use a larger virtual address space than is actually available in the RAM, CPUs implement a virtual memory system called non-locked memory, in which a physical memory page can be swapped out to disk. When the CPU needs that page, it loads it back in from the disk. Unfortunately this cause a delay in memory transaction letting the bandwidth of the PCI-E bus to connect CPU and GPU not fully exploited. This method stores non-locked memory not only in memory once it can be swapped, therefore the drive needs to access all pages of the non-locked memory, copy it into pinned buffer and send it to the Direct Memory Access (DMA) and this process is synchronous, i.e. it is a page-by-page copy. In practice, all PCI-E transfers are made using DMA-based transfers, once the driver does this in background when you do not use a page-locked memory directly. To do this, the drive need to allocate a block of paged-locked memory, do a CPU copy from regular memory to the page-locked memory, initiate the transfers, wait the transfer be completed and then free the page-locked memory (COOK, 2013). Figure 5.7 exhibits how the memory flux occurs in pageable and pinned memory.

This was important when the memory capacity is limited, however, with todays memories, being easy and cheap to expand this limit, the use of virtual memory is no longer necessary for many applications which will fit within the CPU memory space. Therefore, for a better performance in memory transfers, lock the CPU memory to avoid that

Figure 5.7: Pageable Data Transfer vs Pinned Data Transfer



Source: devblogs.nvidia.com/parallelforall/how-optimize-data-transfers-cuda-cc/

be swapped and enable the DMA between CPU and GPU memory rather than using a staging buffer, can result in faster transfers.

To enable the pinned memory allocation, it is necessary to change the normal *malloc* (or *new* command in this work) in C code for the *cudaMallocHost* as showed in algorithm 10.

Algorithm 10 GPU Implementation (memory allocation)

```
1: procedure PAGEABLE MEMORY ALLOCATION
2:     H = new double(size)                    ▷ or H = malloc(size);
3: procedure PINNED MEMORY ALLOCATION
4:     cudaMallocHost((void**)&H, size )
```

Source: The Author

The counterpoint is that once pinned, this portion of memory is unavailable to other processes, including the operational system. However, with the easiness of upgrade the amount of total memory, this can be overpassed with few efforts.

5.4.4 GPU code with streams

One of the main reasons to use GPU is the great parallel performance. The ability to perform an operation in multiple data simultaneously enable great speedups in several programs. With the advance of these technologies, not only data could be parallelized. In newest GPU cards multiple operation can also be executed simultaneously. To do this, processing streams were created. A stream is defined as a sequence of operations that execute in issue-order on the GPU. It means, in NVIDIA cards that support CUDA, that CUDA operations in different streams may run concurrently and operations from different streams may be interleaved.

To use this method is necessary a page-locked memory, and for concurrent streams, also is needed that CUDA operation must be in different, non-0, streams. The default

stream is also knew as stream 0 and can run concurrently with other streams.

The difference among previous GPU codes is in kernel calling. The syntax <<< ... >>> need to be used with the four expected components, i.e., number of blocks, number of threads, amount of allocate dynamically memory per block and associated stream The last argument will set in with stream the kernel will be executed. Algorithm 11 shows the creation of two streams and the kernel to compute the flow in x and y-direction concurrently.

Algorithm 11 GPU Implementation (Streams)

```
1: procedure STREAMS CREATION
2:     cudaStream_t stream1, stream2;
3:     cudaStreamCreate ( &stream1) ;
4:     cudaStreamCreate ( &stream2) ;
5: procedure KERNEL CALL OVER A STREAM
6:     calculateQx <<< dimGrid, dimBlock, 0, stream1>>>(parameters);
7:     calculateQy <<< dimGrid, dimBlock, 0, stream2>>>(parameters);
```

Source: The Author

Once the computation of flow in both direction is calculated separately and each one has it own matrix, there are no issues in execute in parallel and concurrently.

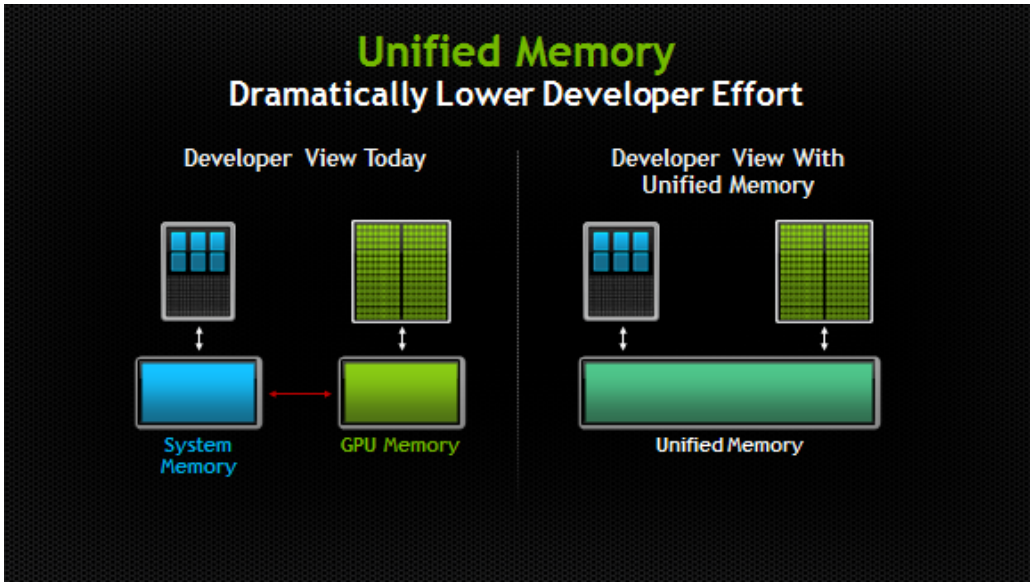
5.4.5 GPU code with Unified Memory

With the release of CUDA 6, one of the biggest barriers to consider the GPU programming in programs development, the difficulty to program, starts to left behind. Until CUDA 6 the view for programmer when related to memory was exactly the real world. The memories of the CPU and GPU are physically distinct and separated by the PCI-Express bus and data that is shared between them must be allocated in both memories and explicitly copied.

Unified Memory creates a pool of managed memory that is shared between the CPU and GPU and this memory is accessible to both the CPU and GPU using a single pointer. The key is that the system automatically migrates data allocated in Unified Memory between host and device so that it looks like CPU memory to code running on the CPU, and like GPU memory to code running on the GPU (HARRIS, 2013). Figure 5.8 illustrate how it works.

The two main benefits published by this new feature are the simpler programming and memory model and performance through data locality. Unified Memory reduces the effort to programming in CUDA environment by making device memory management an optimization, rather than a requirement. More, by migrating data on demand between the CPU and GPU, Unified Memory can offer the performance of local data on the GPU, while providing the ease of use of globally shared data.

Figure 5.8: Developer view of memory hierarchy with and without unified memory



Source: devblogs.nvidia.com/parallelforall/powerful-new-features-cuda-6/

Unified memory do not replace all previous optimizations, it only offer more resources to programmer. So programmers still having access to explicit device memory allocation and asynchronous memory copies to optimize data management and CPU-GPU concurrency.

The difference in code can be viewed in algorithm 12. All memory allocations (CPU and GPU) are replaced by the new *cudaMallocManaged()*. More, all copies are no more necessary, and kernel calls do not need receive pointers of GPU allocated variables and structures. Now they receive the same CPU pointer that points to common memory area.

Algorithm 12 GPU Implementation (Unified Memory)

1: **procedure** UNIFIED MEMORY ALLOCATION
2: $cudaMallocManaged((void**)\&H, cols * rows * sizeof(double))$; ▷ H is the CPU pointer for matrix H
3: ...
4: **procedure** KERNEL CALL WITH UNIFIED MEMORY
5: calculateQx <<< dimGrid, dimBlock, 0, stream1>>>(**CPU parameters**);
6: ...

Source: The Author

6 RESULTS

With the utilization of GPGPU's and remodeling both hydrological and environmental models great results were produced. Natural phenomenas and flood situations could be modeled and computed with an interesting speedup. Besides the hardware capabilities, the area of flood or flow to be analyzed is also a major factor of impact in the execution time, depending on the detail level and area resolution. Each optimization achieve interesting results and will be discussed in this chapter.

All test were realized in a system running Ubuntu 12.04.5 version with 3.13.0-37-generic kernel. The machine Dell PowerEdge R720 with 2 Xeon E5-2630 processor with 6 cores each and 2 threads per core running at 2,3GHz totaling 24 threads and 32GB of memory. For GPU tests was used a NVIDIA TESLA K20 with 2496 CUDA cores, 5GB GDDR5 memory and a memory bandwidth up to 208GB/sec.

To validate the obtained results was simulated a flood event in Greenfields area in Glasgow, UK (figure 6.0a), comprising an area of 0.7 x 0.4 km with a grid resolution about 2 x 2m as described in Test 4 of (BATES; ROO, 2000) and profile code of (NEAL et al., 2010) resulting in a computational grid about 350 x 200 cells.

Then to evaluate the model scalability, was created a synthetic scenario simulating a 2 hours rain over the Itajaí-açú river in Santa Catarina, BR (figure 6.0b), comprising an area of approximately 5 x 4.6 km with a grid resolution about 1.0353 x 1.0353 m resulting in a grid of computation about 4871 x 4414 cells.

6.1 CPU results

The serial version of the model that simulate 2 hours of a flood in an urban area of Greenfields in Glasgow (UK), spent 152.91 seconds running, on average. To do a fair comparison between CPU and GPU, were generated parallel versions that uses openMP with 6, 12 and 24 threads to compare the best of then with GPU results, once is expected for this scenario a better performance of GPU version. The best speed up is achieved with the 24 threads, when the total time to simulate was 47.64 seconds and the average time for 12 threads was 49.26 seconds.

Figure 6.1: Map areas used in this work.

(a) Greenfields area in Glasgow(UK) used for validation tests

(b) Itajai area in Santa Catarina (BR) used for scalability test

Source: The Author

The overall performance and speedup for these instances can be viewed in figure 6.2.

6.2 GPU results

In GPU were made 5 progressive optimizations, where in each of then a new feature was added to previous model. After that, the results will be listed and discussed.

6.2.1 Summarized results

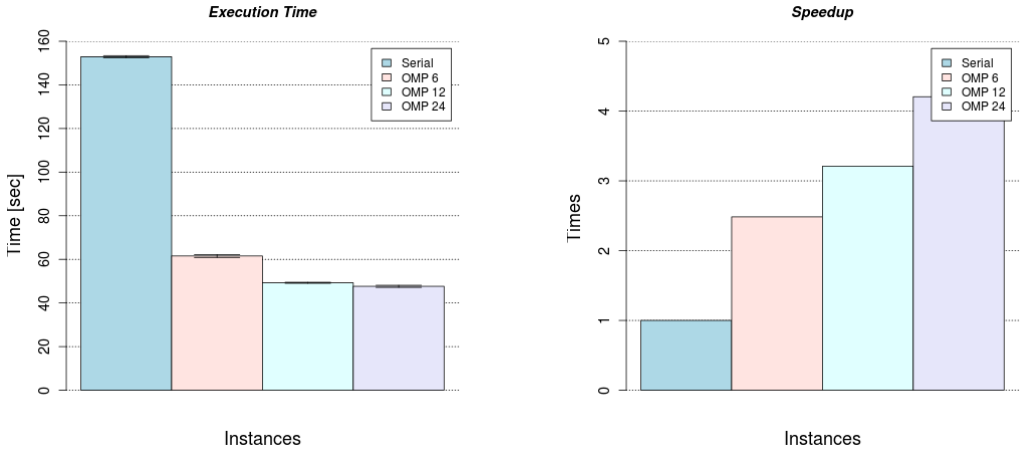
Recode: The first technique was responsible to recode the model to be able to run in GPU. This process allowed the simulation of Glasgow area to gain more than 23% in performance, reducing the best average time in CPU (47.64 seconds) to 36.33 seconds in GPU. This initial achieve was possible due the high data parallelism in this model, being possible to increase the performance when compared with a few CPU threads.

Block and Size: The second optimization, focused in optimize CUDA blocks size to fill the SMX, has a small impact in performance overall compared with the recoded model, being statistically the same, running in 36.36 seconds. For this result, were used

Figure 6.2: Execution time and speedup for serial and parallel version of Glasgow simulation in CPU.

(a) Execution time for serial and parallel versions of the model.

(b) Speedup achieved for serial and parallel versions of the model.



Source: The Author

128 threads inside each of 600 blocks. Increasing the number of threads, in this example, had no improvement in performance. With the related work, is possible to expect a better improvement in larger models that keep the multiprocessors filled.

In both previous techniques, the occupation was approximately of 70% of total processing. A bigger occupancy can be achieved if more single precision data was used.

Next optimizations explore the benefits of different kind of memory allocation.

Pinned Memory: The second great gain in performance were achieved when we began to use the pinned memory instead the pageable memory. The improvement in performance achieved was about 42% compared with the last optimization, accumulating an decrease of 56.29% in execution time against the CPU better time. This result is important to show that even with *fast computational resources*, the memory still being a bottleneck in performance that needs to be evaluated.

Streams: Like the adjustment of size of blocks, the introduction of streams to execute some kernels in parallel also produces no effects (or very low) in performance in this scenario and this can be caused due the small amount of data and the explicit need to synchronize those streams. For large problems, this technique could be useful if there are *space* inside the SMX. If we fill up the SMX, the streamed kernels also need wait to be executed.

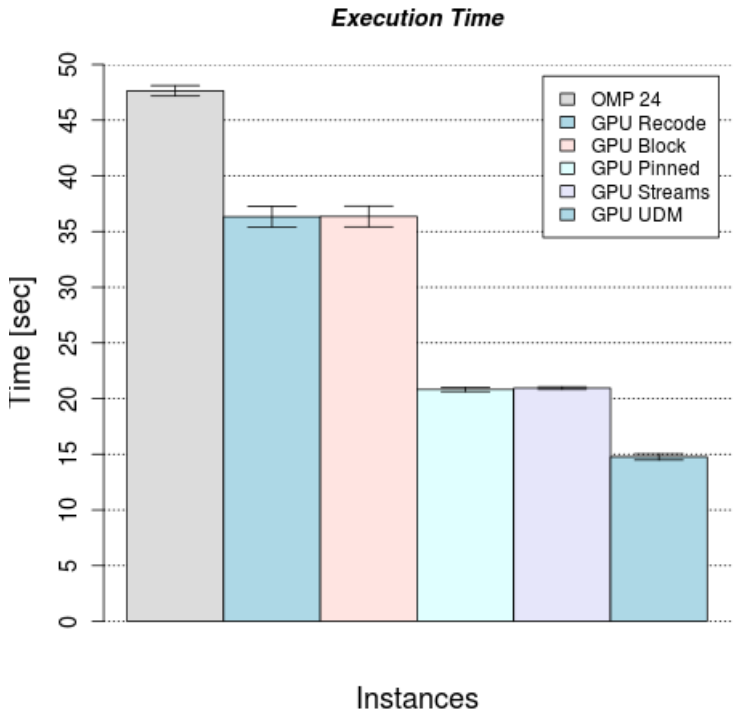
Unified Memory: The last optimization, that requires at least a CUDA 6 version to be implemented also have a great achievement in performance, more 29% in execution time were reduced in relation to Stream technique. This technique allow the CUDA framework to manage all copies, avoiding unnecessary idles due the explicit copy.

With all these optimizations, to simulate 2 hours of flood in this scenario was necessary 14.76 seconds, being this time 69.01% lower than the CPU version with 24 threads or 3.22 times faster. If we compare with the initial serial version, the time to solution was 10.35 times faster.

6.2.2 Resources

All optimization made using GPU were tested 20 times and used the average time. At end of each optimization scenario, more two runnings are made generating 3 output set of files that were compared with the serial model output, which was validated in previous papers. In this comparison, the error was lower than 10^{-6} in all scenarios. For an overview of performance results when compared with the best version used OpenMp, figure 6.3 exhibit all times for GPU instances and the performance of 24 threads using OpenMp version.

Figure 6.3: Execution time and speedup for parallel version of Glasgow simulation in CPU with 24 threads and GPU optimization.

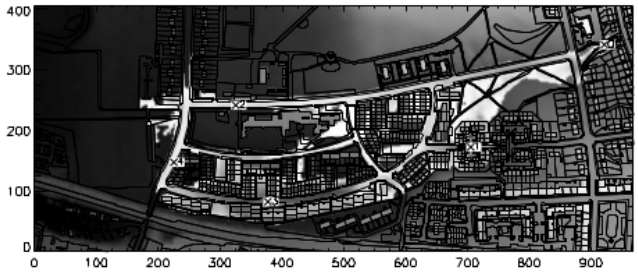


Source: The Author

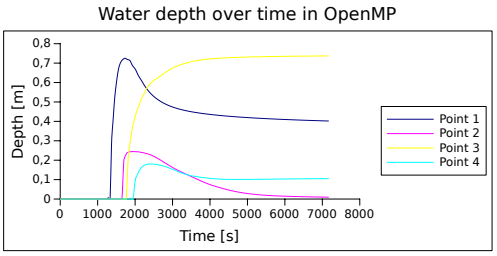
After all optimization was also tested the water depth in 4 control points as showed in figure 6.3a. Figure 6.3b show the water depth over the time when the simulation was executed with OpenMp while figure 6.3c shows the same test when used GPU. Data are collected at every 0.5% of running time generating 200 plotable points. Note that the values are the same, implying in no difference between two versions.

Figure 6.4: Plots of simulated water depth over he time for OpenMp and CUDA versions with no difference in results.

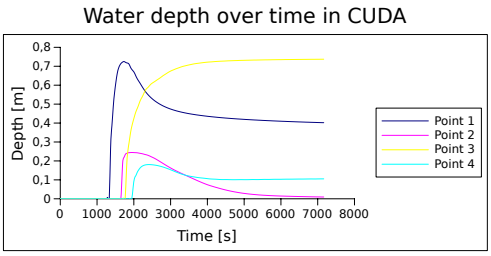
(a) Control points to measure water depth in Greenfield area tests



(b) Water depth plotted over the time at the four control points for LISMIN-OMP version with 24 threads



(c) Water depth plotted over the time at the four control points for LISMIN-GPU version



Source: The Author

A complete comparison between CPU and GPU execution times are also showed in figure 6.5 where is possible to see the final performance being more than 10 times faster than the initial serial version. A detailed view can be obtained in table 6.1.

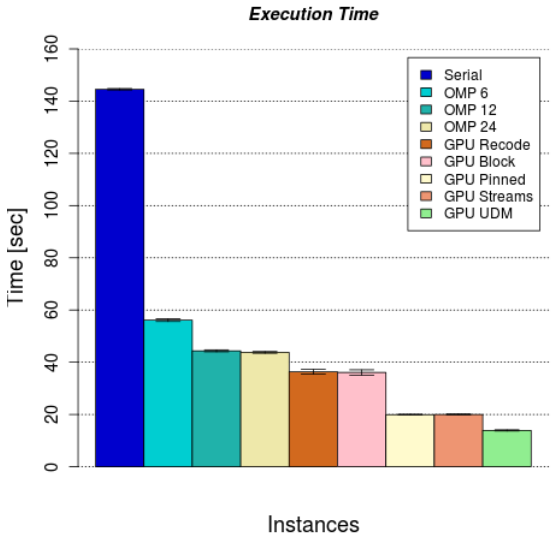
Table 6.1: Measured improvement over time for technique on the line over column and the average time for each technique.

	CPU	Recode	Block	Pinned	Stream	Umd	Avg. Time
CPU	-	23.74%	23.68%	58.19%	58.06%	70.84%	47.64
Recode		-	-0.08%	45.17%	45.00%	61.77%	36.33
Block			-	45.21%	45.05%	61.80%	36.36
Pinned				-	-0.30%	30.27%	19.92
Stream					-	30.48%	19.98
Umd						-	13.89

6.3 Scalability results

The last test to compare performance of CPU and GPU is the scalability of the model simulation. The scenario used to test the scalability is 280 times larger than the used to

Figure 6.5: Execution time and speedup for serial and parallel version of Glasgow simulation in CPU and GPU.



Source: The Author

validate the model. Now is simulates a huge in-flow in Itajai-Açu river and analyzed the time to simulate all flow.

To compare the CPU and GPU versions, only the CPU version with 24 threads are executed and the most optimized version in GPU. To simulate the in-flow in the river was used a synthetic hystrogram proposed in (BRITAIN, 1975) and cited by (PONTES, 2011). This hydrogram can be defined as follow and represented by figure 6.6:

$$Q(t) = Q_{base} + (Q_{peak} - Q_{base})[\frac{t}{T_p} exp(1 - \frac{t}{T_p})]^{\beta} \tag{6.1}$$

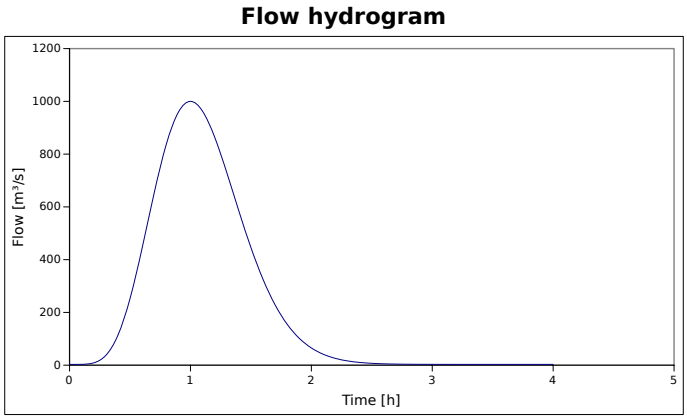
Where:

- Q_{base} is the baseline of flow and in this scenario defined as $3m^3/s$
- Q_{peak} is the peak achieved set as $1000m^3/s$
- T_p is the time when the peak was achieved, here set as 1 hour
- β is the curvature parameter, maying vary between 2 and 20. Here was used a value of 4.

Exploiting equation 6.1 it is possible to see that was necessary more than 3 hours and less than 4 to the input flow return to base value. Therefore in this scenario was simulated 4 hours of flood instead 2 hours like previous test cases.

For the case where the stability equations are not included, the timestep was lower the when the equation is applied. In this case, the GPU simulation of this 4 hours, takes 5.136 hours while in CPU the time was 47.7 hours. It gives a improvement of 9.28 times in processing time. Using the same constant that bounds the timestep, but with the stable

Figure 6.6: Hydrogram related to equation 6.1.

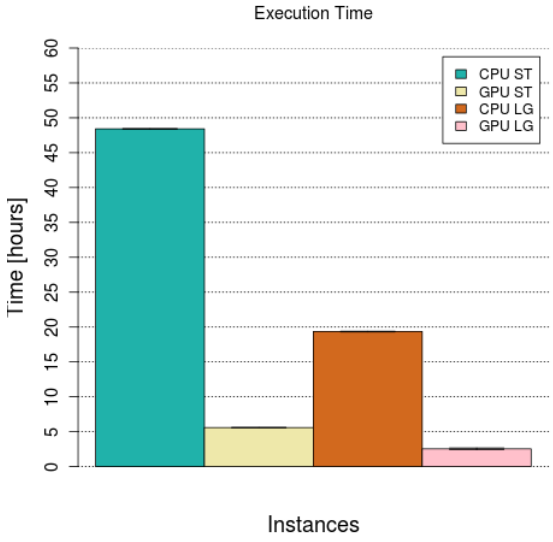


Source: The Author

equation, these times rise to 5.588 hours in GPU and 48.417 in CPU, giving a performance of 8.66 times better.

Increasing the constant and consequently the timestep, simulations times were reduced to 2.533 hours in GPU and 19.363 hours in CPU, giving a performance of 7.642 times better. The execution time of the implemented scalability test is exhibited in figure 6.7

Figure 6.7: Execution time parallel versions of Itajaí-Açu River simulation in CPU and GPU.



Source: The Author

7 ANALYSIS AND CONCLUSIONS

The study of environmental models, including the hydrological models, is recent if compared with other sciences. Computer based hydrological models have been developed and applied at an ever increasing rate during the past four decades, due to the improvement of models and methodologies. Several concepts and classes of models were established, each one with specified features. For more simple implementation, the lumped models can be very helpful while for more detailed results, the distributed models are also more complex to calibrate and spend more time giving better results.

Innumerably models were formulated and validated, covering several characteristics of natural phenomenas, enabling a better prediction of climatic events, natural disasters or accidents. Dam break flood and others natural or not events, can be handled quickly to alert all possible affected. This agility to manage flood events is very important for the economy, since structural damages are avoided and less resources in reconstruction programs need to be invested.

The increased availability of desktops computers and the advance of computational architectures contributed to accelerate these models in the last years enabling the acceleration of the models execution time. Selecting appropriated hardware resource, faster models can be developed without hard work to recode them. With the parallel hardwares and technologies, several different resources could be used to achieve a good speedup for environmental models. MIMD or stream applications explores the parallelism that can be extract from models.

Here was analyzed the performance that can be obtained from GPU cards including its evolution. The performance obtained shows that from small or large areas, stream architectures can avoid great losses. For small analyzed areas, predict the event in approximately 15 seconds can be crucial to handle situations inside a specific area in some cities or neighborhoods, while with all this potential applied in large areas, is possible to reduce the impact of natural disaster, like the flood event that hits Santa Catarina, Brazil in 2008 where more than 1.5 million people are affected. This works also analyzed a simulation of an river in this area, obtained a complete simulation of water behavior in less than 3 hours against almost 20 hours without the use stream architectures. For more

detailed areas, with high water slopes, e.g. rivers that bypasses some hills, the processing time in CPU is about 45 hours being beat from GPU, that simulate the same problem in less than 6 hours.

The great performance achieved in this work is primary due the chosen model behavior. To obtain the maximum of GPU performance, the application need to fit in the goal of this architecture. GPU as other stream architectures have their peak performance when applying an instruction or a set of instructions in several data. Once the environmental model needs to compute the physical changes along the simulation in multiple locations, the improvement in performance as well explained.

Besides the scientific community, open access models are developed diffusing the information about flood events to everyone who wants. Cloud and mobile applications are now more popular than before but this resources need to evolve more to aggregate information from all the globe. In a non far future, when the exascale era begins, the hydrological models may be executed in real-time for every location, helping the governments take more effective actions, save money and most important, helping the population not to lose their belongings.

This work open some new goals to future works as the analysis of different models with different concepts as well as the study of others heterogeneous architectures to compare the benefits with the result obtained in this work. In this context, hybrid models that combine rainfall models and other kinds of models are a relevant field as well as the implementation of these techniques in complete version of LISFLOOD and the analyze of this work in different devices, like multi GPUs as NVIDIA K10 or Xeon Phi.

REFERENCES

ALMEIDA, G. A. et al. Improving the stability of a simple formulation of the shallow water equations for 2-d flood modeling. **Water resources research**, Wiley Online Library, v. 48, n. 5, p. 5528, 2012.

BATES, P.; ROO, A. D. A simple raster-based model for flood inundation simulation. **Journal of hydrology**, Elsevier, v. 236, n. 1, p. 54–77, 2000.

BATES, P. D. Integrating remote sensing data with flood inundation models: how far have we got? **Hydrological Processes**, Wiley Online Library, v. 26, n. 16, p. 2515–2521, 2012.

BATES, P. D.; HORRITT, M. S.; FEWTRELL, T. J. A simple inertial formulation of the shallow water equations for efficient two-dimensional flood inundation modelling. **Journal of Hydrology**, Elsevier, v. 387, n. 1, p. 33–45, 2010.

BERGSTRÖM, S.; SINGH, V. et al. The hbv model. **Computer models of watershed hydrology.**, Water Resources Publications, p. 443–476, 1995.

BEVEN, K.; KIRKBY, M. A physically based, variable contributing area model of basin hydrology/un modèle à base physique de zone d'appel variable de l'hydrologie du bassin versant. **Hydrological Sciences Journal**, Taylor & Francis, v. 24, n. 1, p. 43–69, 1979.

BEVEN, K. J. **Rainfall-runoff modelling: the primer**. [S.l.]: Wiley Chichester, 2001.

BHAT, U. N.; MILLER, G. K. **Elements of applied stochastic processes**. [S.l.]: J. Wiley, 1972.

BLÖSCHL, G.; SIVAPALAN, M. Scale issues in hydrological modelling: a review. **Hydrological processes**, Wiley Online Library, v. 9, n. 3-4, p. 251–290, 1995.

BORMANN, H. et al. Spatially explicit versus lumped models in catchment hydrology–experiences from two case studies. In: **Uncertainties in environmental modelling and consequences for policy making**. [S.l.]: Springer, 2009. p. 3–26.

BRITAIN, N. E. R. C. G. **Flood studies report**. [S.l.]: The Council, 1975.

BRODTKORB, A. R. et al. State-of-the-art in heterogeneous computing. **Scientific Programming**, IOS Press, v. 18, n. 1, p. 1–33, 2010.

CHOW, V. T. et al. **Applied hydrology**. [S.l.: s.n.], 1988.

COOK, S. **CUDA programming: a developer's guide to parallel computing with GPUs**. [S.l.]: Newnes, 2013.

DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. **Computational Science & Engineering, IEEE**, IEEE, v. 5, n. 1, p. 46–55, 1998.

FEWTRELL, T. et al. Evaluating the effect of scale in flood inundation modelling in urban environments. **Hydrological Processes**, Wiley Online Library, v. 22, n. 26, p. 5107–5118, 2008.

FORTIN, V.; TURCOTTE, R. Le modèle hydrologique mohyse. **Note de cours pour SCA7420, Département des Sciences de la Terre et de l'Atmosphère, Université du Québec à Montréal**, v. 23, 2006.

GOLDING, B.; CLARK, P.; MAY, B. The boscastle flood: Meteorological analysis of the conditions leading to flooding on 16 august 2004. **Weather**, Wiley Online Library, v. 60, n. 8, p. 230–235, 2005.

GOSLING, S. N. et al. Climate: Observations, projections and impacts - bangladesh. **Climate: Observations, projections and impacts**, Met Office, 2011.

HABERLANDT, U.; RADTKE, I. Hydrological model calibration for derived flood frequency analysis using stochastic rainfall and probability distributions of peak flows. **Hydrology and Earth System Sciences**, Copernicus GmbH, v. 18, n. 1, p. 353–365, 2014.

HARDAKER, P.; COLLIER, C. Flood risk from extreme events (free)—a national environment research council directed programme. **Quarterly Journal of the Royal Meteorological Society**, John Wiley & Sons, Ltd., v. 139, n. 671, p. 281–281, 2013. ISSN 1477-870X. Available from Internet: <http://dx.doi.org/10.1002/qj.2129>.

HAROU, J. J. et al. Hydro-economic models: Concepts, design, applications, and future prospects. **Journal of Hydrology**, Elsevier, v. 375, n. 3, p. 627–643, 2009.

HARRIS, M. **Unified Memory in CUDA 6**. 2013. <http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/>. [Online; accessed 13-july-2014].

HORRITT, M.; BATES, P. Predicting floodplain inundation: raster-based modelling versus the finite-element approach. **Hydrological processes**, Wiley Online Library, v. 15, n. 5, p. 825–842, 2001.

HUNTER, N. et al. Improved simulation of flood flows using storage cell models. **Proceedings of the ICE-Water Management**, Thomas Telford, v. 159, n. 1, p. 9–18, 2006.

HUNTER, N. et al. Benchmarking 2d hydraulic models for urban flooding. **Proceedings of the ICE-Water Management**, Thomas Telford, v. 161, n. 1, p. 13–30, 2008.

HUNTER, N. M. et al. Simple spatially-distributed models for predicting flood inundation: a review. **Geomorphology**, Elsevier, v. 90, n. 3, p. 208–225, 2007.

HUNTER, N. M. et al. An adaptive time step solution for raster-based storage cell modelling of floodplain inundation. **Advances in Water Resources**, Elsevier, v. 28, n. 9, p. 975–991, 2005.

KALYANAPU, A. J. et al. Assessment of gpu computational enhancement to a 2d flood model. **Environmental Modelling & Software**, Elsevier, v. 26, n. 8, p. 1009–1016, 2011.

KUROWSKI, K.; KULCZEWSKI, M.; DOBSKI, M. Parallel and gpu based strategies for selected cfd and climate modeling models. In: **Information Technologies in Environmental Engineering**. [S.l.]: Springer, 2011. p. 735–747.

LAROCQUE, M. et al. Groundwater contribution to river flows–using hydrograph separation, hydrological and hydrogeological models in a southern quebec aquifer. **Hydrology and Earth System Sciences Discussions**, Copernicus GmbH, v. 7, n. 5, p. 7809–7838, 2010.

LIMA, J. V. F. **A runtime system for data-flow task programming on multicore architectures with accelerators**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, 2014.

LOHANI, A.; KUMAR, R.; SINGH, R. Hydrological time series modeling: A comparison between adaptive neuro-fuzzy, neural network and autoregressive techniques. **Journal of Hydrology**, Elsevier, v. 442, p. 23–35, 2012.

MAIDMENT, D.; MAYS, L. **Applied Hydrology**. Tata McGraw-Hill Education, 1988. (McGraw-Hill series in water resources and environmental engineering). ISBN 9780070702424. Available from Internet: <http://books.google.com.br/books?id=RRwidSsBJrEC>.

MARKS, K.; BATES, P. Integration of high-resolution topographic data with floodplain flow models. **Hydrological Processes**, v. 14, n. 11-12, p. 2109–2122, 2000.

MEENDERINCK, C.; JUURLINK, B. (when) will cmps hit the power wall? In: EURO-PAR 2008 WORKSHOPS-PARALLEL PROCESSING, 2009. **Proceeding...** [S.l.]: Springer, 2009. p. 184–193.

NEAL, J. C. et al. A comparison of three parallelisation methods for 2d flood inundation models. **Environmental Modelling & Software**, Elsevier, v. 25, n. 4, p. 398–411, 2010.

NICANDROU, A. Hydrological assessment and modelling of the river fani catchment, albania. University of Glamorgan, 2011.

NVIDIA. **HISTORY OF GPU COMPUTING**. 2013. <http://www.nvidia.com.br/object/cuda_home_new.html>. [Online; accessed 13-july-2014].

NVIDIA. **NVIDIA cuBLAS library**. 2014. <https://developer.nvidia.com/cublas>. [Online; accessed 28-September-2014].

PAUDEL, M.; NELSON, E. J.; DOWNER, C. W. Assessment of lumped, quasi-distributed and distributed hydrologic models of the us army corps of engineers. In: WORLD ENVIRONMENTAL AND WATER RESOURCES CONGRESS 2009, 2009. **Proceeding...** Great Rivers: ASCE, 2009. p. 5932–5942.

PAUDEL, M. et al. An examination of distributed hydrologic modeling methods as compared with traditional lumped parameter approaches. Department of Civil and Environmental Engineering Brigham Young University, 2010.

PERRIN, C.; MICHEL, C.; ANDRÉASSIAN, V. Improvement of a parsimonious model for streamflow simulation. **Journal of Hydrology**, Elsevier, v. 279, n. 1, p. 275–289, 2003.

PETERS-LIDARD, C.; ZION, M.; WOOD, E. A soil-vegetation-atmosphere transfer scheme for modeling spatially variable water and energy balance processes. **Journal of Geophysical Research: Atmospheres (1984–2012)**, Wiley Online Library, v. 102, n. D4, p. 4303–4324, 1997.

PONTES, P. R. M. **Comparação de modelos hidrodinâmicos simplificados de propagação de vazão em rios e canais**. Dissertation (Master) — Universidade Federal do Rio Grande do Sul, 2011.

QIN, C.-Z.; ZHAN, L. Parallelizing flow-accumulation calculations on graphics processing units—from iterative dem preprocessing algorithm to recursive multiple-flow-direction algorithm. **Computers & Geosciences**, Elsevier, v. 43, p. 7–16, 2012.

REED, S. et al. Overall distributed model intercomparison project results. **Journal of Hydrology**, Elsevier, v. 298, n. 1, p. 27–60, 2004.

REFSGAARD, J. C. Parameterisation, calibration and validation of distributed hydrological models. **Journal of Hydrology**, Elsevier, v. 198, n. 1-4, p. 69–97, 1997.

RESEARCH, T. U. C. for A. **Basic Hydrologic Science Course Runoff Processes - Section Five: Runoff Modeling Concepts**. 2006. <http://wegc203116.uni-graz.at/meted/hydro/basic/Runoff/print_version/05-runoffmodeling.htm#14>. [Online; accessed 10-april-2014].

RIXNER, S. **Stream processor architecture**. [S.l.]: Springer Science & Business Media, 2001.

ROBSON, A. J. Evidence for trends in uk flooding. **Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences**, The Royal Society, v. 360, n. 1796, p. 1327–1343, 2002.

ROSS, P. E. Why cpu frequency stalled. **Spectrum, IEEE**, IEEE, v. 45, n. 4, p. 72–72, 2008.

SAHOO, G.; RAY, C.; CARLO, E. D. Calibration and validation of a physically distributed hydrological model, mike she, to predict streamflow at high frequency in a flashy mountainous hawaii stream. **Journal of Hydrology**, Elsevier, v. 327, n. 1, p. 94–109, 2006.

SCHARFE, M.; PIELOT, R.; SCHREIBER, F. Fast multi-core based multimodal registration of 2d cross-sections and 3d datasets. **BMC bioinformatics**, BioMed Central Ltd, v. 11, n. 1, p. 20, 2010.

SCHUBERT, J. E.; SANDERS, B. F. Building treatments for urban flood inundation models and implications for predictive skill and modeling efficiency. **Advances in Water Resources**, Elsevier, v. 41, p. 49–64, 2012.

SCHUMANN, A. Development of conceptual semi-distributed hydrological models and estimation of their parameters with the aid of gis. **Hydrological sciences journal**, Taylor & Francis, v. 38, n. 6, p. 519–528, 1993.

SEILLER, G.; ANCTIL, F.; PERRIN, C. Multimodel evaluation of twenty lumped hydrological models under contrasted climate conditions. **Hydrology and Earth System Sciences Discussions**, v. 8, p. 10895–10933, 2011.

SHERMAN, L. K. Streamflow from rainfall by the unit-graph method. **Eng. News Record**, v. 108, p. 501–505, 1932.

SINGH, V. P.; WOOLHISER, D. A. Mathematical modeling of watershed hydrology. **Journal of hydrologic engineering**, American Society of Civil Engineers, v. 7, n. 4, p. 270–292, 2002.

TORRES, Y.; GONZALEZ-ESCRIBANO, A.; LLANOS, D. R. Understanding the impact of cuda tuning techniques for fermi. In: HIGH PERFORMANCE COMPUTING AND SIMULATION (HPCS), 2011 INTERNATIONAL CONFERENCE ON. **Proceeding...** [S.l.]: IEEE, 2011. p. 631–639.

TRISTRAM, D.; HUGHES, D.; BRADSHAW, K. Accelerating a hydrological uncertainty ensemble model using graphics processing units (gpus). **Computers & Geosciences**, Elsevier, v. 62, p. 178–186, 2014.

ZANOBETTI, D. et al. Mekong delta mathematical model program construction. **Journal of the Waterways, Harbors and Coastal Engineering Division**, ASCE, v. 96, n. 2, p. 181–199, 1970.