

論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	Tree-Based Mesh-Refined GPU Accelerated Tsunami Simulator for Real Time Operation
著者(和文)	アルセアクニヤ マルロン
Author(English)	MARLON RODOLFO ARCE ACUNA
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第11017号, 授与年月日:2018年12月31日, 学位の種別:課程博士, 審査員:青木 尊之,肖 鋒,木倉 宏成,筒井 広明,加藤 之貴
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第11017号, Conferred date:2018/12/31, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

TOKYO INSTITUTE OF TECHNOLOGY

DOCTORAL THESIS

**Tree-Based Mesh-Refined GPU Accelerated Tsunami Simulator
for Real Time Operation**

Author:

Marlon R. Arce Acuña

Supervisor:

Prof. Takayuki Aoki

A thesis submitted in fulfilment of the requirements

for the degree of Doctor of Engineering

in the

Graduate School of Science and Technology

Department of Nuclear Engineering

September 2017

Abstract

The turn of the 21st century showed us vividly like never before the vast devastation and human loss that tsunamis can cause with the events in Indonesia in 2004 and Japan 2011. This increased the need to develop better simulation tools.

Tsunami simulations traditionally have been developed for CPU computation. With the introduction of General Purpose GPU computing (GPGPU) the possibility to exploit the unparallel speed-up provided by GPU computation was opened. CUDA made GPUs programmable for scientific applications. Taking advantage of this technology we present a program that, while providing high accuracy and reliability, does not sacrifice speed or require excessive simplification. The Shallow Water Equations (SWE) in Cartesian and Spherical coordinates are used to simulate completely on GPU the generation, propagation and inundation of tsunamis on the Indian Ocean. The method of characteristics with a cubic-interpolation is implemented to compute the wave propagation.

Moreover a tree-based refinement is customized to generate a block mesh domain, blocks with higher resolution are obtained near the coastline while coarser resolution remain the open ocean. In order to use resources more efficiently a second refinement, by focal area is introduced. By permitting higher refinement levels only inside designated areas the total number of blocks is drastically reduced while the accuracy on those areas remain high. 7 Levels are generated, the highest resolution is 50m. 4 focal areas are used located in Mozambique, Sri Lanka, Seychelles and Comoros.

The resulting model, named TRITON-G is GPU optimized and extended to multi-GPU, where Hilbert's space filling curve was utilized to partition the domain and maintain load balance. A 10-hour simulation of the Indonesian 2004 tsunami was finished in 40 minutes using 3 Tesla K40 cards. Hindcast of this event's gauges show agreement with the wave arrival times recorded and the main event's peaks. Inundation maps are generated for the focal areas; comparison with existing surveys show correct prediction of the run-up heights.

Acknowledgements

First and foremost I want to thank Prof Aoki for sharing his extraordinary knowledge and professional experience with me in order to complete this work. A deep thank you for all help given during my path as his student. His invaluable advice, patience, motivation, and support have had great significance in my growing as a professional.

It has been an honor in my life to have the opportunity to develop my research with a World leading figure in CFD like him. Words are not enough so let every page of this thesis be a thank you to him.

Also I'd like to thank the research group at RIMES Thailand, for their fine collaboration, for their kind hospitality during my visits to their headquarter, and especially for their invaluable feedback during the development and completion of this project.

A deep thank you to my family, my mother (Damaris) and sister (Angie), for their prayers and for always supporting me in all ways possible and always believing in me.

Finally I'd like to thank the Ministry of Education, Culture, Sports, Science and Technology MEXT for the scholarship to support my initial studies in Japan and once again to Prof. Aoki for selflessly giving me the means to complete my degree.

Contents

List of Figures	ix
List of Tables	xv
Acronyms	xvii
Symbols	i
Chapter 1. Introduction	1
1.1 General Introduction	1
1.2 Tsunami-genesis and History	5
1.3 Operational Tsunami Simulators Survey	11
1.4 Motivation, Innovation and Objectives	14
1.5 Work Outline	15
Chapter 2. Governing Equations.....	17
2.1 Conservation Principles	17
2.2 Free Surface.....	18
2.3 Cartesian Coordinates	20
2.4 Spherical Coordinates	25
Chapter 3. Numerical Methods and Boundary Conditions	29
3.1 Semi-Lagrangian Method	30
3.2 Propagation	32
3.2.1 Hyperbolic system	32
3.2.2 Method of characteristics	34
3.3 Source terms	39
3.4 Non-linear SSWE validation.....	40
3.5 Inundation	44
3.5.1 SWE Benchmark: Dambreak.....	46
3.6 Tsunami Source Model.....	48
3.7 Boundary Conditions	49

Chapter 4. Tree-based Refinement and Bathymetry	51
4.1 Mesh Generation	51
4.1.1 Computational Domain	51
4.1.2 Tree-based refinement.....	52
4.1.3 Refinement by distance.....	56
4.1.4 Refinement by Focal Area	64
4.1.5 Dry area removal.....	67
4.2 Block Halo Update.....	70
4.3 Topography and Bathymetry	72
Chapter 5. GPU Computing	75
5.1 Introduction.....	75
5.2 CUDA	75
5.2.1 Memory Model and Hierarchy.....	76
5.2.2 Programming Model	78
5.2.3 Data Handling.....	79
5.2.4 Compilation for GPU Computing.....	80
5.3 SSWE GPU Kernels.....	81
5.4 GPU Halo swap	83
5.5 Kernel Types	84
5.6 Multi-GPU.....	87
5.6.1 Domain Partition: Space Filling Curve	87
5.6.2 Communication and Buffers.....	91
5.7 TRITON-G Output	95
5.7.1 Type of Output	95
5.7.2 Optimizations	97
5.7.3 Post-processing	100
5.8 TRITON-G Performance and Optimizations	101
5.8.1 Kernel Optimizations.....	103
5.8.2 Sub-cycling.....	106
5.8.3 Performance Measurements	109

Chapter 6. Numerical Simulations: Application.....	113
6.1 Application: Hindcast Indonesia Tsunami December 2004	113
6.1.1 Tide gauge comparison	121
6.1.2 Inundation Run-up: Focal Areas, Phuket.....	128
6.2 Concluding remarks.....	136
Chapter 7. Conclusions and Future Work	137
7.1 Conclusions.....	137
7.2 Future Work	139
Appendix A: Previous Work	143
Appendix B. GPU Boost.....	147
Appendix C Additional TRITON-G Tests	151
Bibliography.....	161

List of Figures

<i>Fig. 1.1 Tohoku Tsunami 2011, Japan. Left: Before and after at the Fukushima Daiichi NPP; Right: Nuclear disaster summarized (Credit left: Earth Imaging Journal; Credit right: AFP)</i>	2
<i>Fig. 1.2 Elevations and locations of structures and components at the Fukushima Dainichi NPP [2]</i>	4
<i>Fig. 1.3 Elevations and locations of structures and components at the Fukushima Dainichi NPP [2]</i>	4
<i>Fig. 1.4 Floating-Point operations per second for CPU and GPU [3]</i>	5
<i>Fig. 1.5 Tsunami Generation by marine earthquake (©NOAA)</i>	6
<i>Fig. 1.6 Classical representation of the 1896 Sanriku Tsunami, Japan</i>	8
<i>Fig. 1.7 Before (a) and after (b) satellite images of the effects of the Dec 26th 2004 Tsunami Banda Aceh Shore, Indonesia. (Credit: DigitalGlobe)</i>	10
<i>Fig. 1.8 Tohoku Tsunami wave overflows barrier in Miyako, Iwate Prefecture, Japan 2011 (Credit: Reuters)</i>	11
<i>Fig. 2.1 Free Surface model. h: water depth; z: bathymetry; H: constant level</i>	19
<i>Fig. 2.2 Spherical Coordinates, θ latitude, λ longitude</i>	25
<i>Fig. 3.1 Time-space Grid for the one-dimensional Advection Equation (2.1). The slope is a characteristic curve along which $u(x,t)=u(0,0)$ and the blue point shows the interpolated result for the semi-Lagrangian scheme. The green region shows the domain of dependence of P</i>	31
<i>Fig. 3.2 Space-time diagram showing the characteristics C^\pm. Black dots represent the grid points while the dotted points represent the values to be interpolated Γ^\pm, u^\pm used to find Γ^{n+1}, u^{n+1} (Diagram based on [66])</i>	36
<i>Fig. 3.3 Stencil used to create cubic polynomial interpolation</i>	37
<i>Fig. 3.4 Far-field propagation benchmark: Idealized Gaussian source on northern hemisphere. Each slide represents 1000 seconds.</i>	42

<i>Fig. 3.5 Gaussian Initial condition benchmark at 5000s. Left image shows the original result from [69], imagine on the right shows the result for TRITON-G.</i>	43
<i>Fig. 3.6 Sketch of the Inundation and run-up heights on a shore slope</i>	44
<i>Fig. 3.7 Reconstructed water depth $h_{L,R}$ for inundation [73]</i>	45
<i>Fig. 3.8 Parabolic bowl problem. Left: water depth error for different values of thin-film ϵ;</i>	46
<i>Fig. 3.9 Dam break computation at 0.1s Case A</i>	47
<i>Fig. 3.10 Dam break computation at 0.1s Case B</i>	47
<i>Fig. 3.11 Tsunami Fault Source: Manila Trench, zoomed at right</i>	48
<i>Fig. 3.12 Tsunami Fault Source: Java Trench, zoomed at right</i>	49
<i>Fig. 3.13 Wall Boundary stencil with reflection at point i. Orange dots represent land and blue dots water</i>	50
<i>Fig. 4.1 Study case. Indian Ocean domain</i>	52
<i>Fig. 4.2 Block-base refinement. Left: Block with neighbors. Right: Parent-children refinement</i>	53
<i>Fig. 4.3 Quadtree structure construction for a domain with a mooned-shape focus</i>	54
<i>Fig. 4.4 Indian Ocean Domain</i>	56
<i>Fig. 4.5 Samples to generate distance function on objects with irregular shapes. (Credits: Batman logo is a trademark of Warner Bros. Tsubame emblem is a trademark of Tokyo Institute of Technology)</i>	57
<i>Fig. 4.6 Signed Distance Function. Positive values represent distances on land, negative values distance on water</i>	58
<i>Fig. 4.7 Refinement stripe representation for L2, L3 and L4</i>	59
<i>Fig. 4.8 Distance Refining Process Indian Ocean Domain using refinement stripe. Total 7 levels, highest resolution 50 m.</i>	63
<i>Fig. 4.9 SAT. (A): Two non intersecting convex polygons; (b) Projection of the non intersecting polygons [86].</i>	65
<i>Fig. 4.10 Focal refinement proof of concept. (a): 5 level refinement with no FA; (b): 5 level refinement with FA represented by a circular shape.</i>	66

<i>Fig. 4.11 Focal areas used in this work. a) Mozambique; b) Sri Lanka; c) Comoros; d) Seychelles</i>	67
<i>Fig. 4.12 Inland dry block removal cases allowed and not allowed</i>	68
<i>Fig. 4.13 Mesh Refinement for Indian Ocean Domain with 4 Focal Areas: Mozambique, Comoros, Seychelles and Sri Lanka.</i>	69
<i>Fig. 4.14 Cell coarsening, averaging down</i>	70
<i>Fig. 4.15 Halo update for neighboring blocks; blue block at level l, orange block at level $l+1$.</i>	70
<i>Fig. 4.16 Halo interpolation stencil for the four edges: north, east (a) and west, south (b)</i>	71
<i>Fig. 4.17 GEBCO bathymetry and topography for the Indian Ocean domain</i>	73
<i>Fig. 4.18 Additional bathymetry databases for replacement with higher accuracy by RIMES. Light-gray shaded areas represent 150m resolution, dark shaded areas represent 50m resolution. (a) Mozambique, (b) Comoros, (c) Seychelles, (d) Sri Lanka</i>	73
<i>Fig. 4.19 Single-point peak bathymetry example</i>	74
<i>Fig. 5.1 nVIDIA Tesla K40C GPU used in this research, 2880 CUDA cores</i>	76
<i>Fig. 5.2 CUDA Memory Model</i>	77
<i>Fig. 5.3 Programming Model. Grid, Blocks, Threads hierarchy representation</i>	78
<i>Fig. 5.4 CUDA blocks and threads diagram for the SSWE Kernel. Top: threads configuration per block; Bottom left: X and Y block configuration; Bottom right: Z block configuration</i>	82
<i>Fig. 5.5 Mesh blocks colored by kernel type. Red: Wet; Green: Wall; Blue: Inundation. Top: zoom over Sri Lanka FA</i>	85
<i>Fig. 5.6 Number of blocks per type</i>	86
<i>Fig. 5.7 Application of the Hilbert orientation tables to obtain the SFC after refinement</i> ..	88
<i>Fig. 5.8 Hilbert Space Filling Curve tests on domains with different geometry, refined levels and large number of blocks. It can be seen that the SFC (line in blue) traces all blocks exactly once</i>	89
<i>Fig. 5.9 Hilbert Space Filling Curve for Indian Ocean Domain</i>	90
<i>Fig. 5.10 Indian Ocean Domain Load Balance on 3 GPUs, each GPU represented by a different color</i>	91

<i>Fig. 5.11 Buffer packaging based on UDP structure</i>	93
<i>Fig. 5.12 GPU buffer. Data collected and packed for a single communication</i>	94
<i>Fig. 5.13 TRITON-G computational flow</i>	96
<i>Fig. 5.14 Optimization by grouping the output blocks (L7) together</i>	97
<i>Fig. 5.15 Output overlap and optimization using Pipes</i>	98
<i>Fig. 5.16 Concept of the Pipe Asynchronous processing by using shared memory</i>	99
<i>Fig. 5.17 TRITON-G Framework</i>	100
<i>Fig. 5.18 FA Images generation process</i>	100
<i>Fig. 5.19 TRITON-G Optimization, a total of 59.6% speed up was achieved</i>	104
<i>Fig. 5.20 Illustration of the sub-cycling process</i>	107
<i>Fig. 5.21 Load Balance example due to the effect of sub-cycling</i>	108
<i>Fig. 5.22 Number of blocks per level before with (orange) an without (blue) sub-cycling</i>	109
<i>Fig. 5.23 Computing breakdown shown in percentage</i>	110
<i>Fig. 5.24 A 10-hour Simulation Runtimes</i>	111
<i>Fig. 5.25 Computing time required to obtain the first results during simulation</i>	111
<i>Fig. 5.26 A 10 hour simulation runtime comparison with 3 different GPUs</i>	112
<i>Fig. 6.1 Hourly snapshots of the Indonesian 2004 tsunami propagation after the earthquake simulated by TRITON-G.</i>	121
<i>Fig. 6.2 Gauge locations in the Indian Ocean: Male, Gale, Diego Garcia, Colombo and Point Le Rue.</i>	122
<i>Fig. 6.3 Comparison of arrival times Diego Garcia tide gauge vs TRITON-G</i>	123
<i>Fig. 6.4 Comparison of arrival times Male tide gauge vs TRITON-G</i>	124
<i>Fig. 6.5 Comparison of arrival times Gan tide gauge vs TRITON-G</i>	125
<i>Fig. 6.6 Comparison of arrival times Colombo for tide gauge, TRITON-G, RIMES</i>	125
<i>Fig. 6.7 Comparison of arrival times Point La Rue tide gauge vs TRITON-G</i>	126
<i>Fig. 6.8 Hambantota Inundation Map, Sri Lanka FA</i>	129
<i>Fig. 6.9 Inundation comparison Hambantota, Sri Lanka; RIMES (left) vs TRITON-G (right)</i>	130

<i>Fig. 6.10 Hambantota, Sri Lanka; left: maximum wave height; right: maximum wave velocity</i>	131
<i>Fig. 6.11 Seychelles FA, left: maximum arrivale wave; right: inundation map</i>	132
<i>Fig. 6.12 Comoros FA, left: maximum arrivale wave; right: inundation map</i>	133
<i>Fig. 6.13 Phuket Inundation: left bathymetry databases used; right: zoom on Kamala and Patong</i>	134
<i>Fig. 6.14 Kamala and Patong maximum inundation map TRITON-G</i>	134
<i>Fig. 6.15 Kamala inundation map comparison. Left: Suppasri et.al. [101]; right: TRITON-G</i>	135
<i>Fig. A.1 Tsunami Simulation running on Terrain II. Increments of aprox.1min shown</i>	144
<i>Fig. A.2 Japan Tohoku Region study case, SRTM and ETOPO merged Bathymetry [103]</i>	145

List of Tables

Table 5.1 Ordering and orientation tables for the Hilbert SFC in two dimensions.....	88
Table 5.2 Left: Number of lines and length sent by boundary type. Right: Amount of data exchanged between processors in the Indian Ocean case (in kB).....	93
Table 5.3 RIMES machine for TRITON-G.....	101
Table 5.4 Tesla K40C Main Specifications.....	103
Table 5.5 Maximum dt per level and the resulting sub-cycling number.....	107
Table A.1 Asynchronous Scalability for 2, 4 and 8 GPUs; Tsubame 1.5 Tesla S1070	145

Acronyms

AMR	Adaptive Mesh Refinement
BE	Boussinesq Equations
BOSZ	Boussinesq Model For Ocean And Surf Zones
CUDA	The Compute Unified Device Architecture
DART	Deep-Ocean Assessment And Reporting Of Tsunamis
FA	Focal Area
FDM	Finite Difference Methods
FLOPS	Floating-point operations per second
GEBCO	General Bathymetric Chart Of The Oceans
GPGPU	Graphic Cards For General Purpose
MOC	Method Of Characteristics
MOST	Method Of Splitting Tsunami
NOAA	National Oceanic And Atmospheric Administration
NPP	Nuclear Power Plant
NRA	Nuclear Regulatory Agency
NS	Navier-Stokes Equations
RIFT	Real-Time Inundation Forecasting Of Tsunamis
RIMES	Regional Integrated Multi-Hazard Early Warning System For Africa And Asia
SAT	Separating Axis Theorem
SELFE	Semi-Implicit Eulerian-Lagrangian Finite Elements

SFC	Space Filling Curve
SGM	Surface Gradient Method
SM	Streaming Multiprocessors
SSWE	Spherical Shallow Water Equations
SWE	Shallow Water Equations
TRITON-G	Tsunami Refinement and Inundation Real Time Operational Numerical Model for GPU
TUNAMI	Tohoku University's Numerical Analysis Model For Investigation
UDP	User Datagram Protocol

Symbols

ρ	mass
Π	viscous stress tensor
λ	longitude
θ	latitude
Ω	rotation of the Earth
ε	thin film of water
τ	bed shear stress/friction
a	Earth's radius
f	Coriolis
g	gravity
p	pressure
p_{atm}	atmospheric pressure

To my mother, whose teachings and support have always been the most wonderful gifts I have ever gotten, to the memory of my father and to my sister for her unconditional support.

Nemo vir est qui mundum non reddat meliorem

Chapter 1. Introduction

七転び八起き

Japanese Proverb

1.1 General Introduction

Natural forces in the Earth make it a place in constant change. The face of the Earth is continuously being shaped by the natural elements it experiences. These forces however, can represent a potential threat to living due to the extraordinary phenomena they represent. These natural disasters such as landslides, hurricanes, volcanic eruptions, earthquakes and tsunamis are factors inherent to Earth and appropriate study and research must be done in order to understand them, prepare and when possible, forecast.

Earthquakes represent one of the most common natural disasters that countries experience. Particularly, countries that lie along the Earth's tectonic plates interfaces and faults, experience constant earthquakes due to their natural movement. The earthquakes produced by these faults vary in magnitude, from non-noticeable to large scale and powerful ones. The Pacific Rim along the Pacific Ocean, where Japan lies, is one of the most active and destructive tectonic plates in the World. The Java trench, in the Indian Ocean represent another very active and highly dangerous fault.

When the epicenter of the earthquakes are submarine there is a potential threat of generating a tsunami. The sudden displacement of large amounts of water due to the fault

fracture or subduction can trigger a wave carrying this energy along the ocean. When this wave reaches the coast its amplitude increases and produces an inundation in land. The long-wave nature of tsunami waves make them particularly difficult to track; tsunami can be wavelengths in excess of hundreds of kilometers. Such long-wave makes the amplitude of the tsunami barely noticeable at sight when traveling in the open ocean. The magnitude a tsunami hit a coast with depends on various factors, the type and magnitude of the earthquake and fault characteristics. Also the coastal shapes may amplify or diminish the effect of the arrival wave.

In December 2004, in the coastal city of Aceh Indonesia a powerful earthquake of magnitude 9.0 stroke the country with great destruction and was followed by a destructive tsunami that produced damaged not only locally in Indonesia but affected neighbors countries as well as countries as far as Seychelles.

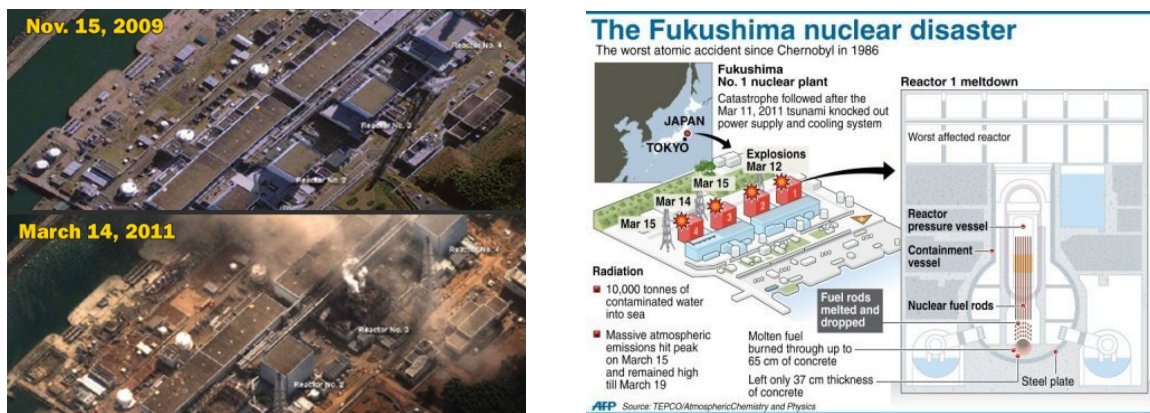


Fig. 1.1 Tohoku Tsunami 2011, Japan. Left: Before and after at the Fukushima Daiichi NPP; Right: Nuclear disaster summarized (Credit left: Earth Imaging Journal; Credit right: AFP)

Then, in 2011 Japan itself experienced the effects of a devastating earthquake followed by a no less powerful tsunami. In March 2011, 400 km off the Tohoku coast in Japan, a large earthquake magnitude 8.9 caused devastation in the region. Infrastructure was damaged or destroyed, tens of thousands of human lives were lost within minutes. However, this tragedy

was followed a powerful tsunami generated by the fault rupture. A nearly 400km-long piece of the fault broke and triggered a tsunami that spread along the Pacific Ocean. Due to the closeness of Japan to the epicenter the first waves took less than an hour to hit the coasts. Wave height of above 30 meters were reported to hit coasts along the Tohoku region. Particularly around half an hour after the earthquake main event, the tsunami wave reached the coasts of Fukushima. Along the destruction on port areas, the wave hit nuclear power plants (NPP) located there. Fukushima Daiichi NPP run by Tokyo Electric Company, TEPCO was hit with waves of over 30 meter high. This tsunami inundated the coast and flooded the NPP producing a temporary loss of electric power which disabled the reactor's coolant (see Fig. 1.1). The effect of this was an overheating of the nuclear fuel, which produce partial and total meltdown in the reactor's cores. The nuclear accident that followed produced the evacuation of thousands of people living in a radius of 40-km around the NPP. Even today, more than half a decade later the consequences of this accident are still present and affecting the lives of people.

The silver lighting behind this tragedy is the understanding and extreme care that should be taken when designing a NPP. Safety must be the first concern and preparation to natural disasters such as tsunamis is the first priority. In 2013 the Japanese Nuclear Regulatory Agency (NRA) introduced new stricter regulation on the design of NPP, specifically they announced their objective to [1]:

“(ii) Significantly enhance design basis and strengthen protective measures against natural phenomena which may lead to common cause failure.

Strict evaluation of earthquakes, tsunamis, volcanic eruptions, tornadoes and forest fires: countermeasures against tsunami inundation and due consideration to ensure diversity and independence.”

Therefore, today more than never the existence of an accurate and reliable way to forecast and simulate the effects of tsunamis has become of extremely importance. As it can be seen

in Fig. 1.2 the design of the Fukushima NPP included potential inundation that underestimated the real threat of a major tsunami inundation.

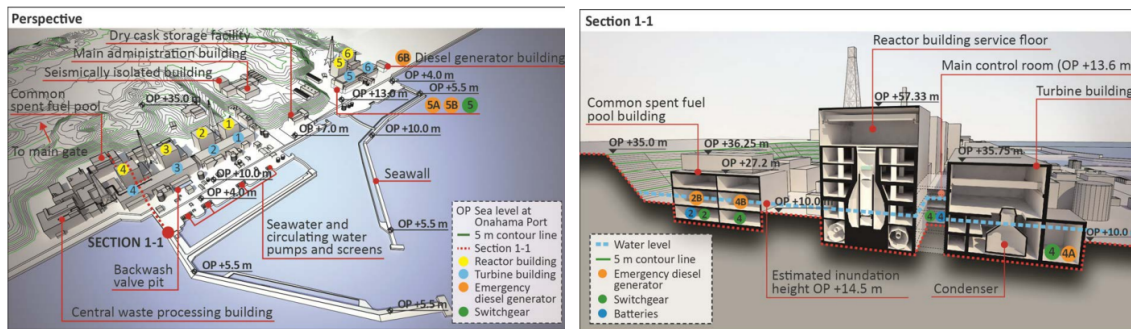


Fig. 1.3 Elevations and locations of structures and components at the Fukushima Daiichi NPP [2]

However this kind of complex modeling is very computational demanding and take long time to produce results using traditional programming. Until recently the creation of this kind of tool required to implement simplifications in model to allow for a faster computation. Or, on the other hand, utilize more complex models at the expense of large computing times. Using traditional CPU programming represented a limit in the possibilities for forecasting. However this changed when in the 2000s nVIDIA introduced their CUDA, language extension, to program their graphic cards for scientific purposes. This disruptive technology of GPGPU computing has taken off in the last years and proven to be a game changer not just in the field of CFD but also in finance, artificial intelligence, data mining, deep learning and more. Due to the nature of parallel computing for graphics, GPUs evolved with hundreds and thousands of cores more than CPUs. These dedicated cores to exclusively compute, is what put GPGPU in a different level, providing outstanding performance and speed. Fig. 1.4 shows an enlightening chart where the advantages of using GPU are appreciated by the much higher floating-point performance than that of traditional CPUs. Currently nVIDIA's Tesla P100 card represents the latest and most powerful of this technology: Pascal; it achieves a staggering peak performance of 9.3 TeraFLOPS, contains 3584 cores and 16 GB of memory. This single card can turn a simple machine into a small supercomputer.

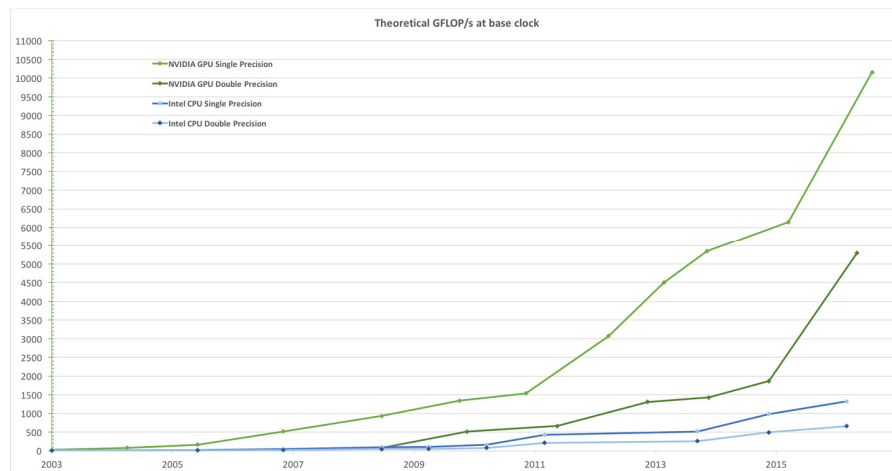


Fig. 1.4 Floating-Point operations per second for CPU and GPU [3]

By applying this GPGPU technology to the tsunami simulation model, a forecasting can be produced that, not only provides high accuracy by solving non-linear models but also delivers high speed. Just like the Japanese proverb that opens this work reads, *Fall seven times, stand up eight*, the tragedies generated by the destructive tsunamis should not offer discourage and despair but instead should offer an opportunity to stand up again, learn the lessons and improve every time with the latest knowledge and technology available.

1.2 Tsunami-genesis and History

Tsunamis can be generated by landslide, meteorites or earthquakes. The latter case being the most common one. As in seen in Fig. 1.5 when an earthquake occurs on the marine platform a water displacement can be cause in the ocean surface that travels as a long wave spreading in all directions until reaches a coast or the energy dissipates.

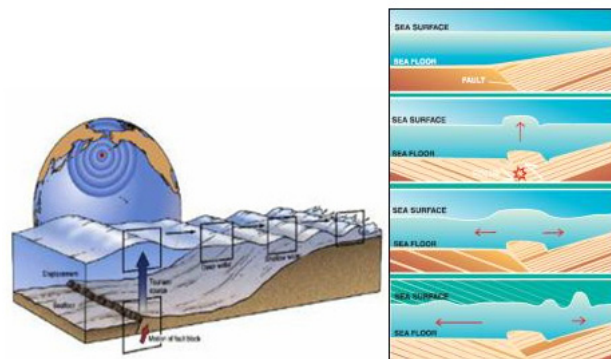


Fig. 1.5 Tsunami Generation by marine earthquake (©NOAA)

Since the energy from the earthquake is transmitted in the tsunami as this moving wave, it actually can travel far distances without much dissipation. A strong earthquake could have the effect of producing trans-oceanic tsunamis. The speed at which the tsunami travels is proportional to the water depth, the deeper is faster it moves; the average traveling speed is about 800 km/h, which is comparable to that of a commercial jet airplane.

In order to understand the existence of tsunami damage in human civilization and the effects they have, a brief survey of historical tsunamis is presented with some details about them; the event are ordered chronologically and represent another proof of the importance of understanding and prediction such events.

Tsunami from the Storegga landslide

Tsunami deposits found in Scotland, Iceland and Norway have been inferred to be from an event called the Storegga slide ([4], [5]). Considered the largest submarine landslide to be mapped, and found in the Norwegian Sea, it has been observed to have taken place three times: at 30-50 ka, and 6-8 ka twice, at a total volume of $5.6 \times 10^3 \text{ km}^3$ [6].

Hokkaido tsunami (17th century)

Prehistoric sand sheets on the Pacific coast of Hokkaido extending to as much as 3km inland show that the tsunamis produced in the Southern Kuril trench were larger than those recorded in the last few centuries. Best explained by earthquakes rupturing at multiple segments of the Kuril subduction zone, these tsunamis had large inundation area and long recurrence interval averaging about 500 years; the most recent one occurring in the 17th century. Some of the recent tsunamis recorded there, like the ones from the 1952-2003 Tokachi-oki or the 1894-1973 Nemuro-oki earthquakes ($M_w \sim 8$) only penetrated less than 1km from the coast [7].

Cascadia tsunami (1700)

Along the Cascadia subduction zone, geological and historical studies have unearthed evidence of a prehistoric earthquake [8]. The size and date of this earthquake can be inferred to be $M_w = 8.7-9.2$ and January 26, 1700, based on historical records in Japan that show a tsunami that struck the Pacific coasts with heights of 1-5m, as well as tsunami deposits found around the Pacific Northwest and geological and tree-ring evidence that show that a coseismic subsidence occurred some 300 years ago and generated a strong tsunami ([9], [10]).

1883 Krakatoa tsunami

The largest volcanic eruption recorded in human history was the 1883 eruption in Krakatoa, Indonesia. The northern part of the island was replaced by a 270m deep caldera; the sound waves from the eruption were heard more than 4000km across the Indian Ocean. The tsunamis generated took over 34,000 lives and decimated coastal villages ([11], [12]). Along the Sunda Strait, the maximum tsunami height was measured at about 15m, and tide gauges were disturbed as far as India, New Zealand, and San Francisco; the farther locations interpreted as a combination of sea waves and air waves from the eruption ([13], [14]).

1896 Sanriku tsunami



Fig. 1.6 Classical representation of the 1896 Sanriku Tsunami, Japan

The tsunami in 1896 (*Fig. 1.6*) that devastated the Sanriku coast of Japan and was unusual in the way it was generated. In what is known as a "tsunami earthquake", the Sanriku earthquake was not great in size ($M_s = 7.2$) and was weak, only 2-3 in the Japanese intensity scale (IV-V on modified Mercalli scale), but resulted in 22,000 casualties, the worst tsunami hazard in Japan in the last few centuries. The run up height was $>10\text{m}$ for 200km of Sanriku coast ([15], [16]).

1946 Aleutian tsunami

On the morning of April 1, 1946, an earthquake with a surface wave magnitude M_s of 7.4 generated massive tsunamis that hit the Aleutians and traveled south through the Pacific ocean to the Hawaiian Islands [15]. It took 159 lives and demolished the Scotch Cap lighthouse, which was situated on a 10m cliff in Unimak Island, 100 km from the epicenter. A Pacific tsunami warning system was introduced shortly thereafter [17].

1958 Lituya Bay tsunami

A strike-slip earthquake ($M_w=7.9$) on July 10 1958 in Lituya Bay, Alaska, triggered a rock slide (700-900m wide, 1000m long, 90m thick, or the total volume of $3 \times 10^7 \text{m}^3$) [17] that generated water waves that surged up the opposite slope and stripped trees as high up as 520m in altitude [18]. This event is the largest water run-up recorded, and is not considered a typical tsunami because the waves were contained within the bay.

1960 Chilean tsunami

On May 22, 1960, the largest ($M_w=9.5$) earthquake of the 20th century occurred off the southern shore of Chile [19], the resulting tsunami destroying the Chilean coast and killing more than 1000 people before propagating across the Pacific. In 15 hours, it hit the Hawaiian Islands, taking 61 lives, and reached Japan after about 23 hours, causing 142 more deaths. The bathymetry in the Pacific Ocean and the sphericity of the Earth had a focusing and resonance effect, where the energy was focused towards Japan, producing a large tsunami (~5m). This led to the forming of an international tsunami warning system in the Pacific [20].

1993 Hokkaido tsunami

The Southwest Hokkaido earthquake on July 12 1993 ($M_w=7.8$) caused a tsunami to hit Okushima Island in the Japan Sea. Around the island, the tsunami reached heights of 5-10m, resulting in more than 200 casualties, while near the valley; the maximum run-up was more than 30m. To reproduce the measured heights of the tsunami, recorded waveforms on tide gauges, and seismic and geodetic data, numerical computations of the tsunamis were made ([21], [22]).

1998 Papua New Guinea tsunami

Tsunami heights from an earthquake ($M_w=7.1$) along the coast of New Guinea Island on July 17, 1998 were as high as 15m around Sissano Lagoon, near the epicenter, causing a reported death toll of more than 2000. Marine surveys conducted after the tsunami showed bathymetry features that focused the tsunami's energy towards the lagoon and possible sources of submarine landslide. A local effect of the this tsunami is supported by the fact that the large tsunami was limited to a small region (~40km) and tide gauges around Japan reported amplitudes of $< 10\text{cm}$ [23]. To reproduce the heights from this tsunami, numerical simulations reveal that another source, possibly from submarine landslides, is needed in addition to the earthquake fault motion ([24], [25]).

2000s Tsunamis

The turn of the 21st century showed us the reality of the terrible and devastating damage and death that Tsunamis can cause as never before. In 2004 a massive earthquake of magnitude 9.0 on the Richter scale [26], off Sumatra Island triggered a tsunami with deadly consequences (*Fig. 1.7*).

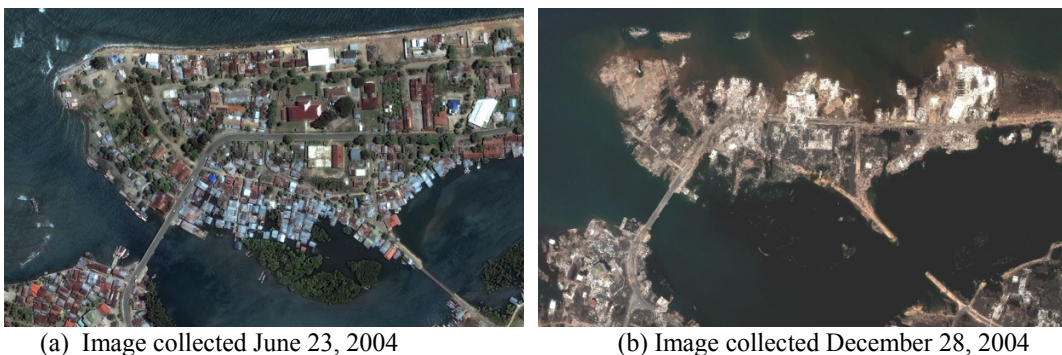


Fig. 1.7 Before (a) and after (b) satellite images of the effects of the Dec 26th 2004 Tsunami Banda Aceh Shore, Indonesia. (Credit: DigitalGlobe)

According to the World Health Organization the death toll for these events exceeds 200,000 people [27] and includes several countries spread along the Indian Ocean.

Not much later in 2011 a tsunami triggered by a M9.0 earthquake [28] on the east coast of Japan produced in the Tohoku region yet another disaster (*Fig. 1.8*). This time over 15,000 people died from these events and the destruction generated was massive in port and city infrastructure, housing, tele-communications [29] and the subsequent nuclear crisis due to the tsunami-induced damage of several reactors in a power plant [30].



Fig. 1.8 Tohoku Tsunami wave overflows barrier in Miyako, Iwate Prefecture, Japan 2011 (Credit: Reuters)

1.3 Operational Tsunami Simulators Survey

In order to understand the efforts made to produce tsunami models and their current state, a survey is presented with the most relevant operational programs.

For several decades efforts have been made to develop such models for tsunami generation and propagation using different governing equations, numerical methods, mesh grids as well as wet-dry algorithms to estimate the run-up. In general these efforts can be grouped in depth-average, hydrostatic or non-hydrostatic long wave equations.

Models for the shallow water equations (SWE) started by solving their linear form based on finite difference methods (FDM) taking after the work of Hansen [31] and Fischer [32] in the 1950s. A more recent review of these methods can be found in the publication by Kowalik [33] and Imamura [34]. From these first steps came the tsunami propagation model known as TUNAMI (Tohoku University's Numerical Analysis Model for Investigation) [35] which solved the shallow water equations in a non-linear form instead, formulated in a flux-conservative way for mass conservation and also introduced a discharge computation [34] for the elevation near the shoreline. In a very similar manner the ALASKA-tectonic and Landslide models (GI'-T) (GI'-L) were introduced, these also solved the non-linear shallow water and used leapfrog FDM as TUNAMI [36]. Continuing the approach of solving the non-linear SWE came MOST (Method of Splitting Tsunami) [37], an extensively used model for tsunami simulation, it tried to incorporate the effect of dispersion on the simulation [38], also it was original by introducing a function to add points in the shoreline to keep better track. A more recent tsunami model in the group of non-linear SWE is GeoClaw, it implemented a unique approach to deal with the issue of transferring fluid kinematic throughout nested grids by refining specified cells during the simulation thus solving better in those areas [39]. More recent models incorporate a real-time application such as RIFT (Real-Time Inundation Forecasting of Tsunamis) [40]. Like several of the previous models a Leap-Frog scheme is also used for these real-time models, also a linear SWE is implemented to save computing time. Another recent real-time model is EasyWave [41] which also employs linear approximations for speed up and leap-frog scheme as its numerical scheme. Additionally the latest version of EasyWave introduced GPU acceleration for solving parts of the existing CPU code and on single precision.

These mentioned models can be classified as hydrostatic, however assuming a hydrostatic pressure implies also nondispersive models which can limit the outcome and modeling in certain cases such as tsunamis generated from an underwater landslide or small faults, which generate smaller wave lengths. In order to include this effect since the 1990s some models took the direction of solving the depth-integrated Boussinesq equations (BE) instead of the SWE for tsunami propagation. The initial efforts considered a weak nonlinear model [42]

however, models for the nonlinear equations were also developed not long after, for instance Nwogu et. al. [43], Lynett et. al. [44]. Solving the Boussinesq equation is, in general, more computationally demanding than solving the SWE hence in order to reduce the computational time some techniques have been used such using parallel clusters or introducing nested-grids. An example of this implementation is FUNWAVE-TVD [45] which is an extended version of FUNWAVE, a run-up and propagation model based on fully nonlinear and dispersive Boussinesq equations [46]. FUNWAVE also introduced a nested grid method for solving the equations and its later version has been fully parallelized using MPI-FORTRAN. Another example of BE models is BOSZ (Boussinesq Model for Ocean and Surf Zones [47]), this model combines the dispersive effect from the BE with the shock capturing ability of the nonlinear SWE; BOSZ is mainly used for near-shore simulation since is based on Cartesian coordinates and thus not suited for large areas, also it does not implement nested grids.

A variant on how to approach the tsunami models has been to solve the problem in three dimensions. Along these lines SELFE (Semi-Implicit Eulerian-Lagrangian Finite Elements) [48] is a well-known model which solves the 3D Navier-Stokes equations (NS) using an unstructured grid, however when used for tsunami simulation it is usually configured as a 2D solver for faster computation. Other full 3D models can be found in Abadie et. al. [49] and Horrillo et. al. [50]. Although these models tend to capture difficult coastlines very well and can include multiple fluids or even materials, the computation cost is still so great that make it possible only to apply them effectively in small areas, as opposed to transoceanic propagations. An approach to deal with the high computational cost has been to implement a hybrid system, incorporating a 3D NV model with hydrostatic or non-hydrostatic models, an example of this hybrid modeling is THETIS [51] which uses the 3D NV to solve the complex source area or landslide and then interpolates the propagation into a 2D non-linear SWE or BE model for better efficiency.

1.4 Motivation, Innovation and Objectives

The past tsunami events and specially the recent ones in Japan and Indonesia are the main motivation to find a fresh and new approach for tsunami modeling, that, while it retains a high degree of the complexities of the physics involved, it still delivers a fast and accurate simulation for early warning forecasting. Also, the existence of GPU cards to drastically speed-up simulations motivates us to develop this model as a full-GPU program to take full advantage of the outstanding performance of GPGPU.

Furthermore, the Indian 2004 tsunami disaster created not only worldwide awareness about this natural threat but also helped to develop and improve disaster prevention efforts. A great example of this is the *Regional Integrated Multi-Hazard Early Warning System for Africa and Asia* RIMES [52], established in 2009 and formed by several country members in that region and based in Thailand. Following their aim to “*provide regional early warning services and build capacity [...] in the end-to-end early warning of tsunami [...] hazards*”, RIMES has a constant interest in improving their tsunami forecasting tools. Moreover, we are motivated by the opportunity to collaborate with RIMES in their latest project to develop this work for their specific needs and resources, and produce an operational tool suitable to replace their previous tsunami forecasting program.

The main innovation of this work is to push the envelope of speed by implementing a full operational tsunami model on multi-GPU for fast computation, with high accuracy in a large domain. Also, the customization of tree-based refinement to this specific work to generate a mesh that can identify and track complex coast shapes, generate high resolution, and still use computational resources efficiently.

Hence, our main objective is to produce a robust program that delivers accurate results and computational speed and that can be used as operational model. The major purpose is to apply this simulator to tsunamis for wide area and long distance propagation from the focal spot. In this kind of long distance simulations, like the tsunami event of Tohoku 2011 or

Indonesia 2004, the effect of the accuracy of the initial fault become weak; due to the collaboration with RIMES we chose the latter as a study case for this work.

Furthermore, in order to achieve high performance and short computational time, this program is developed for full GPU implementation and expanded to Multi-GPU for further acceleration.

A second objective is to implement the Method of Characteristics and a 3rd Order Semi-Lagrangian numerical scheme to solve the non-linear shallow water equations, allowing high accuracy. With this also unwanted effects of numerical dispersion and diffusion are minimized. A corollary to this objective is the smooth and correct integration of Cartesian and Spherical versions of the shallow water equations.

Another important objective is to generate a domain mesh that utilizes memory and computational resources efficiently. A customized refinement is needed that can track complex coast shapes and focuses on specific areas of interest. Also the customization of the refinement is necessary to fine-tune the generated mesh to the specific needs and computational resources of RIMES.

Finally there is the goal to develop a program that while robust is also flexible for different settings and parameters. More importantly that this flexibility also provides reliability for operation use. For reference herein after this program is named TRITON-G, acronym for Tsunami Refinement and Inundation Operational Numerical Model for GPU.

1.5 Work Outline

The thesis is organized as follows: a review of the governing equations is given in chapter 2. A deduction of the shallow water equations is outlined and then the extension to spherical coordinates is presented.

The numerical method and boundaries are explained in section 3. The method of characteristics is used to solve hyperbolic systems, first it is demonstrated that the shallow water equations are hyperbolic and then the method of characteristics applied to solve them is explained. The cubic interpolation used in the characteristics is also presented.

In section 4 a description of the domain mesh refinement is presented. The tree-based refinement briefly explain and then its customizations for our specific program are explained. Details about the domain topography and bathymetry used are also described.

GPU and parallel computing is covered in section 5. The way to implement the numerical simulation is explained. The type of kernels used are also detailed. Multi-GPU and its load balance is outlined and the chapter finishes by presenting results of performance.

In chapter 6 several results are presented including model validation with existing tsunami propagation data and run-up measurements. Hindcast for the Indonesia 2004 Tsunami is studied.

The thesis finishes with Chapter 7 which summarizes the research done and presents its conclusions.

Chapter 2. Governing Equations

In order to model the tsunami propagation and run-up the Non-Linear Shallow Water Equations are used. Since the inundation areas of interest are considered small scale consisting of a few kilometers, the SWE in Cartesian coordinates are chosen. On the contrary, for the rest of the domain the SWE in Spherical Coordinates (SSWE) are used instead to take into account the spherical shape of the Earth and to include effects such as Coriolis forces. Here we present the derivation of the SWE starting from the conservation principles and then extend to the SSWE.

2.1 Conservation Principles

For a compressive material the conservation laws in a differential conservation law form for mass and momentum are:

$$\rho_t + \nabla \cdot (\rho V) = 0 \quad (2.1)$$

$$\frac{\partial}{\partial t} (\rho V) + \nabla \cdot [\rho V \otimes V + pl - \Pi] = \rho g \quad (2.2)$$

where the mass is in ρ and the momentum ρV . The independent variables are: time t and x, y, z for the space. The dependant variables are ρ for density, velocity $V = (u, v, w)$, pressure p , g is the gravity and the tensors:

$$V \otimes V = \begin{pmatrix} v^2 & uv & vw \\ vu & v^2 & vw \\ wv & vw & w^2 \end{pmatrix} \quad (2.3)$$

$$\Pi = \begin{pmatrix} \tau^{xx} & \tau^{xy} & \tau^{xz} \\ \tau^{yx} & \tau^{yy} & \tau^{yz} \\ \tau^{zx} & \tau^{zy} & \tau^{zz} \end{pmatrix} \quad (2.4)$$

the Π stands for the viscous stress tensor. The conservation equations are written in their components as follows:

$$\begin{aligned} \rho_t + u\rho_x + v\rho_y + w\rho_z + \rho(u_x + v_y + w_z) &= 0 \\ u_t + uu_x + vu_y + wu_z + \frac{1}{\rho} p_x &= g_1 \\ v_t + uv_x + vv_y + wv_z + \frac{1}{\rho} p_y &= g_2 \\ w_t + uw_x + vw_y + ww_z + \frac{1}{\rho} p_z &= g_3 \end{aligned} \quad (2.5)$$

2.2 Free Surface

A surface in a three-dimensional domain is assumed to be under the effect of gravity. The x - y coordinates defines the horizontal plane and z the vertical direction or the associated elevation of the free surface. A sketch of this description can be seen in *Fig. 2.1*

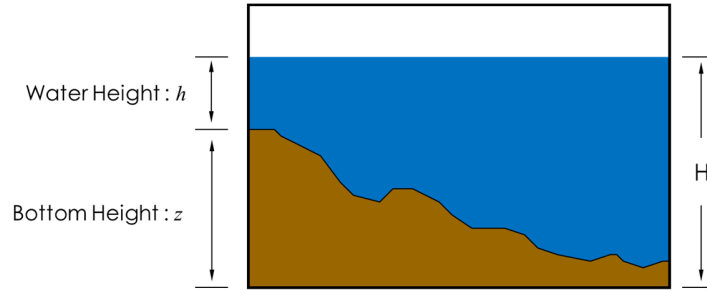


Fig. 2.1 Free Surface model. h : water depth; z : bathymetry; H : constant level

There are some boundaries to consider. First a function $b(x,y)$ is defined as the bottom boundary, referred also as *bed*, as follows:

$$z = b(x, y) \quad (2.6)$$

Then, the free surface is defined by:

$$z = s(x, y, t) \equiv b(x, y) + h(x, y, t) \quad (2.7)$$

where $h(x,y,t)$ is the water height, defined as the distance between the water surface and the bed.

If a constant density is assumed and the gravity g is taken as $\mathbf{g} = (0, 0, -g)$ with a constant value of $g=9.8m/s^2$, then the conservation equations (2.5) can be simplified to:

$$u_x + v_y + w_z = 0 \quad (2.8)$$

$$u_t + uu_x + vv_y + ww_z = -\frac{1}{\rho} p_x \quad (2.9)$$

$$v_t + uv_x + vv_y + wv_z = -\frac{1}{\rho} p_y \quad (2.10)$$

$$w_t + uw_x + vw_y + ww_z = -\frac{1}{\rho} p_z - g \quad (2.11)$$

With given initial and boundary conditions this set of equations can be solved for the four unknowns: p , u , v , w . However, solve it directly this way is considerable computationally demanding. A way to determine the boundary condition to find the solution is to assume the depth of the water to be small with respect to the wave length. This give the advantage to lead to nonlinear initial value problems analogous to those associated with wave propagation in compressible materials [53].

To obtain the shallow water equations first the boundary conditions for the general problem are studied. For the surface assume that the condition is given by:

$$\psi(x, y, z, t) = 0 \quad (2.12)$$

For the free surface then:

$$\psi(x, y, z, t) \equiv z - s(x, y, t) = 0 \quad (2.13)$$

and for the bottom boundary:

$$\psi(x, y, z, t) \equiv z - b(x, y) - 0 \quad (2.14)$$

Finally a kinematic and dynamical condition are assumed on the surface, respectively:

$$\frac{D}{Dt}\psi(x, y, t) = 0 \quad (2.15)$$

$$p(x, y, z, t) \Big|_{z=s(x,y)} = p_{atm} = 0 \quad (2.16)$$

where p_{atm} is the atmospheric pressure, taken for convenience as zero.

2.3 Cartesian Coordinates

The SWE can be derived from the previous assumptions; this derivations follows the detailed explanation also found in Toro E. [54]. The SWE in Cartesian coordinates can be obtained following these steps. First, the vertical component of the acceleration is assumed

negligible, which is a reasonable assumption considering that waves moves along the surface mainly horizontal for our purposes. Inserting this condition, $dw/dt=0$, in equation (2.11) gives:

$$p_z = -\rho g \quad (2.17)$$

$$f \quad (2.18)$$

By using the dynamical condition (2.16) we obtain:

$$p = \rho g(s - z) \quad (2.19)$$

Taking the derivate of this expression with respect to x and y gives:

$$p_x = \rho g s_x, \quad p_y = \rho g s_y \quad (2.20)$$

since both are independent of z then the x and y components of the acceleration of the water particles du/dt and dv/dt are also independent of z . Next, these two expressions are replaced in (2.9) and (2.10) to obtain:

$$u_t + uu_x + vv_y + ww_z = -gs_x \quad (2.21)$$

$$v_t + uv_x + vv_y + ww_z = -gs_y \quad (2.22)$$

The next step requires integrating the continuity equation (2.8) along z , from the bottom to the free surface, as follows:

$$\int_b^s (u_x + v_y + w_z) dz = 0 \quad (2.23)$$

$$w|_{z=s} - w|_{z=b} + \int_b^s u_x dz + \int_b^s v_y dz = 0 \quad (2.24)$$

In order to find the first two terms of this last equations the boundary conditions assumed in the previous section are used. By applying condition (2.15) to (2.13) and (2.14) w can be determined, thus:

$$(s_t + us_x + vs_y - w)|_{z=s} = 0 \Rightarrow w|_{z=s} = (s_t + us_x + vs_y)|_{z=s} \quad (2.25)$$

$$(ub_x + vb_y - w)|_{z=b} = 0 \Rightarrow w|_{z=b} = (ub_x + vb_y)|_{z=b} \quad (2.26)$$

In turn, these values for w can be inserted back in (2.24) which gives:

$$(s_t + us_x + vs_y)|_{z=s} - (ub_x + vb_y)|_{z=b} + \int_b^s u_x dz + \int_b^s v_y dz = 0 \quad (2.27)$$

To solve the last two terms of this equations, the Leibniz's formula:

$$\frac{d}{dx} \int_{a(x)}^{b(x)} f(x, y) dy = \int_{a(x)}^{b(x)} \frac{\partial f}{\partial x} dy + f(x, b(x)) \frac{db(x)}{dx} - f(x, a(x)) \frac{da(x)}{dx} \quad (2.28)$$

is applied to obtain:

$$\int_b^s u_x dz = \frac{\partial}{\partial x} \int_b^s u dz - u|_{z=s} \cdot s_x + u|_{z=b} \cdot b_x \quad (2.29)$$

$$\int_b^s v_y dz = \frac{\partial}{\partial y} \int_b^s v dz - v|_{z=s} \cdot s_y + v|_{z=b} \cdot b_y \quad (2.30)$$

These two equations are now used in (2.27) and after simplifications the following expression is found:

$$s_t + \frac{\partial}{\partial x} \int_b^s u dz + \frac{\partial}{\partial y} \int_b^s v dz = 0 \quad (2.31)$$

According to the definitions given in the previous we know that u and v do not depend of z ; also we know that $s=b+h$ and $b_t=0$. By using this in equation (2.31) can be finally written as:

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x} [u(s-b)] + \frac{\partial}{\partial y} [v(s-b)] = 0 \quad (2.32)$$

$$\Rightarrow h_t + (hu)_x + (hv)_y = 0 \quad (2.33)$$

This equation is the *law of conservation of mass* in its differential conservation law form. To obtain similar equations for the momentum the next procedure is followed. First, equation (2.33) is multiplied by u and (2.21) by h , then added, and after simplification we get:

$$(hu)_t + (huv)_y + u^2 h_x + 2uhu_x + gh h_x = -ghz_x \quad (2.34)$$

Finally, assuming h is differentiable such that $h \frac{\partial h}{\partial x} = \frac{\partial}{\partial x} \left(\frac{1}{2} h^2 \right)$ then the last equation simplifies to:

$$(hu)_t + \left(hu^2 + \frac{1}{2} gh^2 \right)_x + (huv)_y = -ghz_x \quad (2.35)$$

Similarly for the momentum in y :

$$(hv)_t + \left(hv^2 + \frac{1}{2} gh^2 \right)_y + (huv)_x = -ghz_y \quad (2.36)$$

These last two equations together with (2.33) are known as the non-linear SWE. Furthermore, these equations can be written in conservative form as:

$$U_t + F(U)_x + G(U)_y = S(U) \quad (2.37)$$

with:

$$U = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix} \quad F(U) = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2} gh^2 \\ huv \end{bmatrix} \quad G(U) = \begin{bmatrix} hv \\ hvu \\ hv^2 + \frac{1}{2} gh^2 \end{bmatrix} \quad S(U) = \begin{bmatrix} 0 \\ -gh \frac{\partial z}{\partial x} + \tau_x \\ -gh \frac{\partial z}{\partial y} + \tau_y \end{bmatrix}$$

where \mathbf{U} represents the vector of conserved variables, h is the water depth, hu and hv represent the momentum, u and v are the fluid velocities in the x and y direction respectively, \mathbf{F} and \mathbf{G} represent fluxes and \mathbf{S} represents a source vector that can include terms like the effect of bed slope (bathymetry z), bed shear stress τ (friction).

A numerical scheme is regarded as well-balanced or satisfying the C-property [55] if it preserves steady states at rest, for instance, the undisturbed surface of lake. When the fluid is at rest $u(x,t)=0$ then $H(x,t)=h(x,t)+z(x)$ represents a steady state that should hold in time and not produce spurious oscillations [56]. When source terms that depend on geometry are introduced in the equations, the gradient in the conserved variables incorporates a geometry term that does not vanish under steady-state conditions. In order to guarantee a well-balanced scheme an additional change in equations (2.37) is introduced by expressing them in terms of a constant water level H ; this way it is possible to compute accurately steady state. The result of substituting H for the one dimensional x -direction case of h and hu is:

$$\begin{aligned} \frac{\partial h}{\partial t} + \frac{\partial hu}{\partial x} &= 0 \\ \Downarrow \\ \frac{\partial H}{\partial t} + \frac{\partial Hu}{\partial x} &= \frac{\partial zu}{\partial x} \end{aligned} \tag{2.38}$$

and

$$\begin{aligned} \frac{\partial hu}{\partial t} + \frac{\partial}{\partial x} \left(hu^2 + \frac{1}{2} gh^2 \right) &= -gh \frac{\partial z}{\partial x} \\ \Downarrow \\ \frac{\partial Hu}{\partial t} + \frac{\partial}{\partial x} \left(Hu^2 + \frac{1}{2} gH^2 \right) &= u \frac{\partial zu}{\partial x} \end{aligned} \tag{2.39}$$

respectively. The equivalent symmetrical result for y -direction can be readily found in the same fashion.

2.4 Spherical Coordinates

In order to approximate the Earth's surface the non-linear shallow water equations are solved on the sphere, referred to as Spherical Shallow Water Equations (SSWE). *Fig. 2.2* shows the definition of longitude λ and latitude θ on the sphere. By using this coordinate system not only the effect of the Earth's curvature on the propagation is included but also forces generated by its rotation i.e. Coriolis. Also the non-linear equations are more adequate to describe the flow motion in coastal areas where the wave length of the incident tsunami becomes shorter and the amplitude becomes larger as the leading wave of the tsunami propagates into shallow waters.

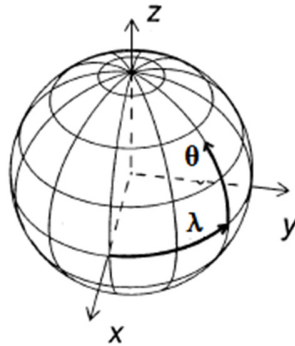


Fig. 2.2 Spherical Coordinates, θ latitude, λ longitude

If the earth's radius is a , then the linear dimension of a region of the sphere are given by:

$$\Delta x_\lambda = a \cos \theta \Delta \lambda \quad \Delta x_\theta = a \Delta \theta \quad (2.40)$$

Hence the non-linear SWE ([57], [58]) on the sphere can be written as the following set of equations:

$$\begin{aligned}
 \frac{\partial h}{\partial t} + V \cdot \nabla h + \frac{h}{a \cos \theta} \left(\frac{\partial u}{\partial \lambda} + \frac{\partial v \cos \theta}{\partial \theta} \right) &= 0 \\
 \frac{\partial u}{\partial t} + V \cdot \nabla u - \left(f + \frac{u}{a} \tan \theta \right) v + \frac{g}{a \cos \theta} \frac{\partial z}{\partial \lambda} &= 0 \\
 \frac{\partial v}{\partial t} + V \cdot \nabla v + \left(f + \frac{u}{a} \tan \theta \right) u + \frac{g}{a} \frac{\partial z}{\partial \theta} &= 0
 \end{aligned} \tag{2.41}$$

Moreover, the set of equations (2.41) can be re-written for the vector of conserved variables \mathbf{U} in (2.37) as:

$$\begin{aligned}
 \frac{\partial h}{\partial t} + \frac{1}{a \cos \theta} \frac{\partial}{\partial \lambda} (hu) + \frac{1}{a} \frac{\partial}{\partial \theta} (hv) - \frac{hv}{a} \tan \theta &= 0, \\
 \frac{\partial hu}{\partial t} + \frac{1}{a \cos \theta} \frac{\partial}{\partial \lambda} \left(hu^2 + \frac{g}{2} h^2 \right) + \frac{1}{a} \frac{\partial huv}{\partial \theta} - \frac{huv}{a} \tan \theta \\
 - \left(f + \frac{u}{a} \tan \theta \right) hv + \frac{gh}{a \cos \theta} \frac{\partial z}{\partial \lambda} + \tau_\lambda &= 0, \\
 \frac{\partial hv}{\partial t} + \frac{1}{a \cos \theta} \frac{\partial huv}{\partial \lambda} + \frac{1}{a} \frac{\partial}{\partial \theta} \left(hv^2 + \frac{g}{2} h^2 \right) - \frac{hv^2}{a} \tan \theta \\
 + \left(f + \frac{u}{a} \tan \theta \right) hu + \frac{gh}{a} \frac{\partial z}{\partial \theta} + \tau_\theta &= 0
 \end{aligned} \tag{2.42}$$

where f stands for the Coriolis force, defined as:

$$f = 2\Omega \sin \theta \tag{2.43}$$

and Ω is the rotation of the Earth. The parameters used in this work are:

$$a=6.37122 \times 10^6 [m]$$

$$\Omega=7.292 \times 10^{-5} [s^{-1}] \quad (2.44)$$

$$g=9.80616 [ms^{-1}]$$

It is worth noting that besides the additional new terms due to the spherical setting that correspond to the source term in equations (2.42), there is still a high similarity with the Cartesian SWE multiplied by a factor either $1/\cos \theta$ or $1/a$. Also this quasi symmetry between latitude and longitude equations will prove useful in the kernel optimizations.

Chapter 3. Numerical Methods and Boundary Conditions

Computational Fluid Dynamics or *CFD* is a branch of science that by using the aid of computers, studies the behavior of fluid-flow problems based on the physical governing equations. The solution is usually focus on variables such as heights, velocity or pressure in particular parts of the domain, or in some cases in the overall behavior.

Due to the complexity of the equations involved in these kind of problems it is not often that an analytic solution can be found, perhaps only for the simplified cases. Hence numerical methods have made great progress over the past decades to help find solutions to these complex problems. For example, finite difference, finite element, finite volume, spectral methods just to mention the most familiar ones.

When numerical methods are used an accurate solution is sought with minimum error possible. Because of sources or error during the simulation and in the method, it is important to benchmark results and study stability and accuracy. These errors may come from discretization, since the domain is not really the real one but a representation of it in spaced points; input data errors, due to the fact that it is nearly impossible to work with real geometries and some simplifications must be made; initial and boundary condition errors, similar to the previous one because it is difficult to model the real conditions.

CFD provide many advantages for simulation development. It can be produced quickly and inexpensively due to the decrease of the prices in computing. The results produced are detailed and comprehensive, readily accessible, something difficult to do when dealing with real life experiments. Also changing of parameters depending on the user specifications can be done relatively easy, therefore different cases can be studied with the same framework or code.

CFD has also the ability to simulate real conditions very accurately. More importantly, it permits to study unnatural or hazardous events, such as natural disasters (Tsunami,

Hurricanes), conflagrations, explosions or nuclear plant failures, without having to risk human lives or investing huge budgets to set an experiment.

3.1 Semi-Lagrangian Method

Although it is possible to carry out, in theory, a prediction in a Lagrangian framework by following a set of marked parcels, in practice this is not a reasonable alternative because shear and stretching deformations has the effect to concentrate these parcels in a few regions therefore it is difficult to keep an uniform resolution along the domain. It is possible to take advantage of the conservative properties of Lagrangian schemes, while maintaining uniform resolution by using a semi-Lagrangian method [59]. This form allows relatively long time steps while retaining numerical stability and high accuracy. In a very simple and introductory approach this procedure can be illustrated with the one dimensional advection equation. A more robust and complete method will be applied to the Shallow Water Equations in a next chapter. Let's assume we have the one-dimensional advection equation:

$$\frac{\partial f}{\partial t} + u \frac{\partial f}{\partial x} = 0 \quad (3.1)$$

According to this equation 2.1 the field f is conserved following the x direction at speed u . Hence for any grid point we have $x_i = i\Delta x$ and time $t_s = s\Delta t$:

$$f(x_i, t_s + \Delta t) = f(\tilde{x}_i^s, t_s) \quad (3.2)$$

where \tilde{x}_i^s is the location at time t_s for the point located at the point x_i at the time $t_s + \Delta t$. However this point's position generally speaking does not lie on a grid point exactly, because of this reason to evaluate the right side of (2.2) requires interpolation from the grid point values at a time t . When $u > 0$ the position \tilde{x}_i^s would be located between the points x_{i-p} and

x_{i-p-1} where p is the integer part of the measure of the number of grid points traversed in a timestep: $u\Delta t / \Delta x$. For the sake of this first survey simplicity let's assume linear interpolation is used, then:

$$f(\tilde{x}_i^s, t_s) = \alpha f(x_{i-p-1}, t_s) + (1-\alpha)f(x_{i-p}, t_s) \quad (3.3)$$

with $\alpha = (x_{i-p} - \tilde{x}_i^s) / \Delta x$. Thus as shown in Figure 2.1 $p=1$ and to predict u at a point P then the interpolation must happen between $i=1$ and $i=2$, the result is marked in blue.

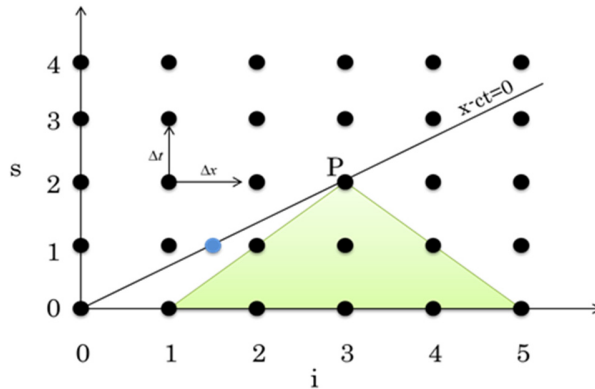


Fig. 3.1 Time-space Grid for the one-dimensional Advection Equation (2.1). The slope is a characteristic curve along which $u(x,t)=u(0,0)$ and the blue point shows the interpolated result for the semi-Lagrangian scheme. The green region shows the domain of dependence of P

In a more realistic model the velocity field is also calculated rather than give as we assumed here. Thus, for a two-dimensional field:

$$f(x, y, t + \Delta t) = f(x - u\Delta t, y - v\Delta t, t) \quad (3.4)$$

where the velocity components at time t can be used to estimate the fields at $t + \Delta t$. If these are found then they can be used to provide a more accurate approximation to (2.4). The right side of (2.4) is again solved by interpolation, in this case in a two dimensional interpolation.

It can be appreciated in the grid *Fig. 3.1* that the semi-Lagrangian scheme guarantees that the domain of influence in the numerical solution corresponds to that of the physical problem. Hence this scheme is computationally stable for time steps much larger than possible with an explicit Eulerian scheme. Finally the semi-Lagrangian scheme also preserves the values of conservative properties quite accurately, a very desirable characteristic since we will be working a conservative scheme.

3.2 Propagation

3.2.1 Hyperbolic system

Let's express the SWE in the state of conservation laws as:

$$\begin{aligned}
 h_t + uh_x + hu_x + vh_x + hv_x &= 0, \\
 u_t + uu_x + gh_x + vu_y &= -gb_x, \\
 v_t + uv_x + gh_y + vy_y &= -gb_y
 \end{aligned}
 \tag{3.5}$$

These equations can be written in a quasi-linear form such as:

$$W_t + A(W)W_x + B(W)W_y = S
 \tag{3.6}$$

with:

$$W = \begin{bmatrix} h \\ u \\ v \end{bmatrix}, \quad A(W) = \begin{bmatrix} u & h & 0 \\ g & u & 0 \\ 0 & 0 & u \end{bmatrix}$$

$$B(W) = \begin{bmatrix} v & 0 & h \\ 0 & v & 0 \\ g & 0 & v \end{bmatrix}, \quad S = \begin{bmatrix} 0 \\ -gb_x \\ -gb_y \end{bmatrix}$$

A system of m conservation laws with Jacobian matrices $\mathbf{A}(\mathbf{U})$ and $\mathbf{B}(\mathbf{U})$ is said to be hyperbolic if the matrix \mathbf{C} formed by the linear combination of the Jacobian matrices $\mathbf{A}(\mathbf{U})$ and $\mathbf{B}(\mathbf{U})$:

$$C = \omega_1 A + \omega_2 B \quad (3.7)$$

has m real eigenvalues for any vector \mathbf{U} of conserved variables and any vector $\omega = [\omega_1, \omega_2]$, such that $\omega \neq 0$. To find these eigenvalues first we construct the \mathbf{C} matrix using equation (3.7):

$$C = \begin{bmatrix} 0 & w_1 & w_2 \\ (a^2 - u^2)w_1 - uvw_2 & 2uw_1 + vw_2 & uw_2 \\ -uvw_1 + (a^2 - v^2)w_2 & vw_1 & uw_1 + 2vw_2 \end{bmatrix} \quad (3.8)$$

where $a = \sqrt{gh}$. Then from solving $\det(C - \lambda I)$, where I is the three-dimensional identity matrix, we find the eigenvalues λ_i of \mathbf{C} to be:

$$\begin{aligned} \lambda_1 &= uw_1 + vw_2 - a|w| \\ \lambda_2 &= uw_1 + vw_2 \\ \lambda_3 &= uw_1 + vw_2 + a|w| \end{aligned} \quad (3.9)$$

Since ω_1 and ω_2 are defined as real parameters such that:

$$|w| = \sqrt{w_1^2 + w_2^2} > 0 \quad (3.10)$$

then it is proven that the eigenvalues (3.9) are all real and hence the system is hyperbolic.

3.2.2 Method of characteristics

In order to compute the propagation of the tsunami, the SSWE (6) are solved using the Method of Characteristics (MOC). This is a commonly used method in gas dynamics ([60], [61]) developed in the 1960s, explained in detail by Rusanov [62]. MOC can be applied to reduce hyperbolic partial differential equations, such as the SSWE, to a family of ordinary differential equations which in turn can be integrated from a chosen initial value. In general, it uses the Riemann invariants of a system to find the solution of the set of equations. A typical approach when using MOC is to introduce a dimensional splitting in the multi-dimensional equations, in the case of the Cartesian SWE equations (2.38) and (2.39) represent the directional splitting for the x -direction [63]. To apply the Method of Characteristics to the SSWE (2.42) first let's express them in vector form as:

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial \lambda} + B \frac{\partial U}{\partial \theta} + S = 0 \quad (3.11)$$

with

$$A = \frac{1}{a \cos \theta} \begin{bmatrix} 0 & 1 & 0 \\ \Gamma^2 - u^2 & 2u & 0 \\ -uv & v & u \end{bmatrix},$$

$$B = \frac{1}{a} \begin{bmatrix} 0 & 0 & 1 \\ -uv & v & u \\ \Gamma^2 - v^2 & 0 & 2v \end{bmatrix},$$

$$S = \begin{bmatrix} \frac{-hv \tan \theta}{a} \\ -(f + \frac{u}{a} \tan \theta)hv - \frac{huv}{a} \tan \theta + \frac{gh}{a \cos \theta} \frac{\partial z}{\partial \lambda} \\ (f + \frac{u}{a} \tan \theta)hu - \frac{hv^2}{a} \tan \theta + \frac{gh}{a} \frac{\partial z}{\partial \theta} \end{bmatrix}$$

where $\Gamma \equiv \sqrt{gh}$. As mentioned before, following the directional splitting technique, equation (3.11) can be expressed as:

$$\frac{\partial U}{\partial t} + S = 0, \quad (3.12)$$

$$\frac{\partial U}{\partial t} + A \frac{\partial U}{\partial \lambda} = 0, \quad (3.13)$$

$$\frac{\partial U}{\partial t} + B \frac{\partial U}{\partial \theta} = 0 \quad (3.14)$$

From these three equations it can be seen that equation (3.12) simply represents the source term and more importantly that equations (3.13) and (3.14) are in advection form. To find the solution of these two equations an additional step of diagonalizing the matrix is necessary in order to find the invariants and characteristics of the equation, a detailed description of this procedure can be found in Ogata et.al. [64], Stoker [65]. Thus the equations for the Riemann invariants and their eigenvalues (Λ) for λ are:

$$\Lambda_{\pm}^{\lambda} = \frac{1}{a \cos \theta} (u \pm \Gamma), \quad \Lambda_3^{\lambda} = \frac{1}{a \cos \theta} u \quad (3.15)$$

$$\frac{D^{\pm}}{Dt} \left(\Gamma \pm \frac{u}{2} \right) = 0 \quad (3.16)$$

Similarly for θ :

$$\Lambda_{\pm}^{\theta} = \frac{1}{a} (v \pm \Gamma), \quad \Lambda_3^{\theta} = \frac{1}{a} v \quad (3.17)$$

$$\frac{D^{\pm}}{Dt} \left(\Gamma \pm \frac{v}{2} \right) = 0 \quad (3.18)$$

The following description shows the results for λ however the solution for θ can be found analogously. Equation (3.16) means that the solution at a given grid point i , is determined

from two characteristics along C^+ and C^- (Fig. 3.2). Thus the result at a time $n+1$ can be found from:

$$\Gamma_i^{n+1} = \frac{1}{2} \left\{ \Gamma^+ + \Gamma^- + \frac{1}{2}(u^+ - u^-) \right\} \quad (3.19)$$

$$u_i^{n+1} = \frac{1}{2} \left\{ u^+ + u^- + 2(\Gamma^+ - \Gamma^-) \right\} \quad (3.20)$$

where Γ^\pm and u^\pm are the variables value at a time n , however they might not necessarily lie on a grid point.

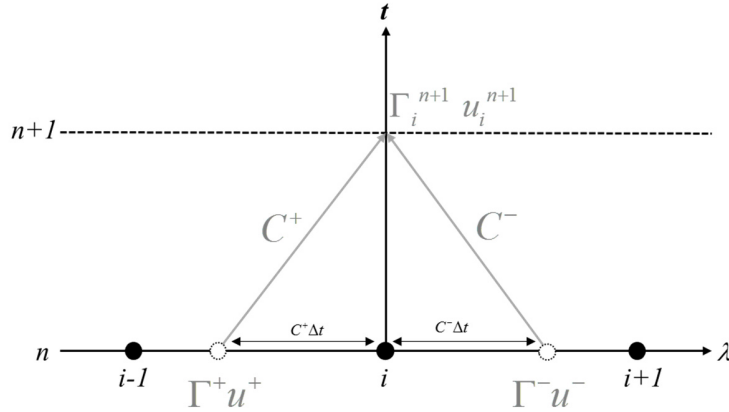


Fig. 3.2 Space-time diagram showing the characteristics C^\pm . Black dots represent the grid points while the dotted points represent the values to be interpolated Γ^\pm, u^\pm used to find Γ^{n+1}, u^{n+1} (Diagram based on [66])

Hence we must implement an interpolation in order to find their value and solve (3.19), (3.20). Following a similar procedure as T. Yabe et.al ([66], [67], [68]) we utilize a cubic-polynomial approximation $F(\lambda)$ on the grid profile to find the interpolated values and to allow for a large time step Δt . The stencil used to create the polynomial is depicted in Fig.

3.3. Let define $F(\lambda)$ as:

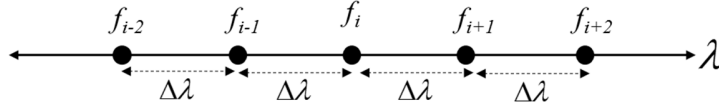


Fig. 3.3 Stencil used to create cubic polynomial interpolation

$$F(\lambda) = \tilde{a}\lambda^3 + b\lambda^2 + c\lambda + d \quad (3.21)$$

thus, four conditions are required to determine the polynomial. It should be noted the use of the special character \tilde{a} to avoid confusion with the radius of the Earth a . Then, the chosen conditions are:

$$\begin{aligned} f_{i+1} &= F(\Delta\lambda) \\ f_i &= F(0) \\ f_{i-1} &= F(-\Delta\lambda) \\ f_{i-2} &= F(-2\Delta\lambda) \end{aligned} \quad (3.22)$$

Using these conditions a system of four equation with four unknowns can be created and the solution for the coefficients determined. This system can be readily solved by using math software like Maple® to obtain for $u^+_{\lambda/\theta} \Delta t \geq 0$:

$$\begin{aligned} \tilde{a} &= \frac{f_{i+1} - 3f_i + 3f_{i-1} - f_{i-2}}{6\Delta\lambda^3} \\ b &= \frac{f_{i+1} - 2f_i + f_{i-1}}{2\Delta\lambda^2} \\ c &= \frac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta\lambda} \\ d &= f_i \end{aligned} \quad (3.23)$$

with $u^+_{\lambda} = \frac{1}{a \cos \theta} (u + \Gamma)$ for longitude; $u^+_{\theta} = \frac{1}{a} (v + \Gamma)$ and $\Delta\theta$ instead of $\Delta\lambda$ for latitude.

Similarly for $u^-_{\lambda/\theta} \Delta t < 0$:

$$\begin{aligned}
 \tilde{a} &= \frac{f_{i+2} - 3f_{i+1} + 3f_i - f_{i-1}}{6\Delta\lambda^3} \\
 b &= \frac{f_{i+1} - 2f_i + f_{i-1}}{2\Delta\lambda^2} \\
 c &= \frac{-f_{i+2} + 6f_{i+1} - 3f_i - 2f_{i-1}}{6\Delta\lambda} \\
 d &= f_i
 \end{aligned} \tag{3.24}$$

with $u^-_{\lambda} = \frac{1}{a \cos \theta} (u - \Gamma)$ for longitude and $u^-_{\theta} = \frac{1}{a} (v - \Gamma)$, $\Delta\theta$ for latitude.

By looking at the similarities between expressions (3.23) and (3.24) it becomes clear that is possible to create a set of coefficients that include both cases in one. In order to achieve this, first let's define the following expressions:

$$\begin{aligned}
 sp &= \frac{\text{SIGN}(u^{\pm}_{\lambda/\theta} \Delta t) + 1}{2} \\
 sm &= 1 - sp
 \end{aligned} \tag{3.25}$$

where SIGN is defined as the sign of the respective term $u^{\pm}_{\lambda/\theta} \Delta t$, 1 for positive and -1 for negative. Using these expressions let's also define the following variables:

$$\begin{aligned}
 A_{P2} &= 0 \cdot sp + 1 \cdot sm & C_{P2} &= 0 \cdot sp - 1 \cdot sm, \\
 A_{P1} &= 1 \cdot sp - 3 \cdot sm & C_{P1} &= 2 \cdot sp + 6 \cdot sm, \\
 A_{C0} &= -3 \cdot sp + 3 \cdot sm & C_{C0} &= 3 \cdot sp - 3 \cdot sm, \\
 A_{M1} &= 3 \cdot sp - 1 \cdot sm & C_{M1} &= -6 \cdot sp - 2 \cdot sm, \\
 A_{M2} &= -1 \cdot sp + 0 \cdot sm & C_{M2} &= 1 \cdot sp + 0 \cdot sm
 \end{aligned} \tag{3.26}$$

Therefore, finally single expressions for the coefficients \tilde{a} , b and c can be written as:

$$\begin{aligned}
 \tilde{a} &= \frac{A_{P2}f_{i+2} + A_{P1}f_{i+2} + A_{C0}f_i + A_{M1}f_{i-1} + A_{M2}f_{i-2}}{6\Delta\lambda^3} \\
 b &= \frac{f_{i+1} - 2f_i + f_{i-1}}{2\Delta\lambda^2} \\
 c &= \frac{C_{P2}f_{i+2} + C_{P1}f_i + C_{C0}f_i + C_{M1}f_{i-1} + C_{M2}f_{i+2}}{6\Delta\lambda}
 \end{aligned} \tag{3.27}$$

Although not clear at first sight, the objective of creating single expressions for \tilde{a} , b and c lies in the programming implementation. Using expressions (3.23) and (3.24) create the need of divergent computing branches on two separate paths, whereas expression (3.27) creates a single path. Branch divergence is a condition that can penalize performance in GPU computing hence the importance of creating expressions that avoid this situation when possible.

3.3 Source terms

For the source term, the bottom friction is discretized and computed as in (3.37). Central finite differences are used to solve the bed slope:

$$\frac{gh}{a \cos \theta} \frac{\partial z}{\partial \lambda} = \frac{gh_i}{a \cos \theta_i} \frac{z_{i+1} - z_{i-1}}{2\Delta\lambda} \tag{3.28}$$

The terms involving trigonometrical expressions such as cosine, tangent can be solved analytically at each grid point since the variables are known, i.e.:

$$\begin{aligned}
 \frac{hv \tan \theta}{a} &= \frac{(hv)_i}{a} \tan \theta_i \\
 \frac{huv}{a} \tan \theta &= u_i \frac{(hv)_i}{a} \tan \theta_i
 \end{aligned} \tag{3.29}$$

Same approach goes for the terms including the Coriolis expression f :

$$\left(f + \frac{u}{a} \tan \theta\right) h v = \left(2\Omega \cos \theta_i + \frac{u_i}{a} \tan \theta_i\right) (h v)_i \quad (3.30)$$

Finally, the use of real bathymetry introduces the challenge of handling unusually large and sharp gradients during computation. In order to avoid divergence or spurious oscillations an artificial viscosity is introduced in the equations. This is represented by the diffusion equation, discretized as:

$$f^{n+1} = f^n + \mu(f_{i+1} - 2f_i + f_{i-1}) \quad (3.31)$$

with the coefficient $\mu = \frac{\kappa \Delta t}{\Delta \lambda^2}$ on an uniform mesh. Moreover a normalization is applied to this equation in order to guarantee that the effect of this term is equal on grids with different resolution, such as in the case mesh refinement. For this purpose let's define a new coefficient $\hat{\kappa}$ defined as:

$$\hat{\kappa} \equiv \kappa \Delta \lambda \quad (3.32)$$

By doing this a new coefficient μ is obtain that is now independent of the grid solution, thus:

$$\mu = \frac{\hat{\kappa}}{\Delta \lambda} \quad (3.33)$$

The appropriate value of $\hat{\kappa}$ is chosen to be small enough not to affect the wave propagation yet of significant where large gradients are present to improve stability.

3.4 Non-linear SSWE validation

The first step necessary to confirm the correct implementation of the numerical method to solve the SSWE is to benchmark the results with an appropriate test. While there are many analytical and laboratory test cases for near-field tsunamis, it is surprising the lack of these tests for long range propagation and even less in spherical coordinates. From the scare options

to benchmark we present a test case from Kirby et.al. [69] to model equations and sensitivity to dispersion and Coriolis effects. This test can be used to confirm the correct propagation of the waves in spherical coordinates for far-field simulations.

This benchmark case utilize an idealized source and ocean, and assumes a flat bathymetry over the entire ocean. The water depth is constant over the whole domain $h=3000\text{m}$. The grid resolution is 0.75 arc-min. The domain is in the northern hemisphere and covers a region from 15° to 40° latitudinal and from -10° to 15° longitudinal. An initial idealized Gaussian elevation:

$$H(\lambda, \theta) = A \exp \left[-\frac{1}{W^2} \left((\lambda - \lambda_c)^2 + (\theta - \theta_c)^2 \right) \right] \quad (3.34)$$

centered at $(\lambda_c, \theta_c) = (0^\circ, 30^\circ)$ is used as source. The source width W is 0.25° and the amplitude A is 1 m.

Fig. 3.4 show our results of the wave propagation for this test case. Whereas the initial source is a circular Gaussian-shape wave, it is noticeable the deformation the wave experiences due to the curvature of the sphere as it evolves and spread over the domain.

Moreover, a direct comparison is presented in *Fig. 3.5*, the image on the left is taken from the original benchmark report. It represents a portion of the wave at 5000 seconds; since the wave is symmetrical and the bottom is flat, there is no loss of generality reporting just this portion of the domain. The image on the right represent the result obtained by TRITON-G.

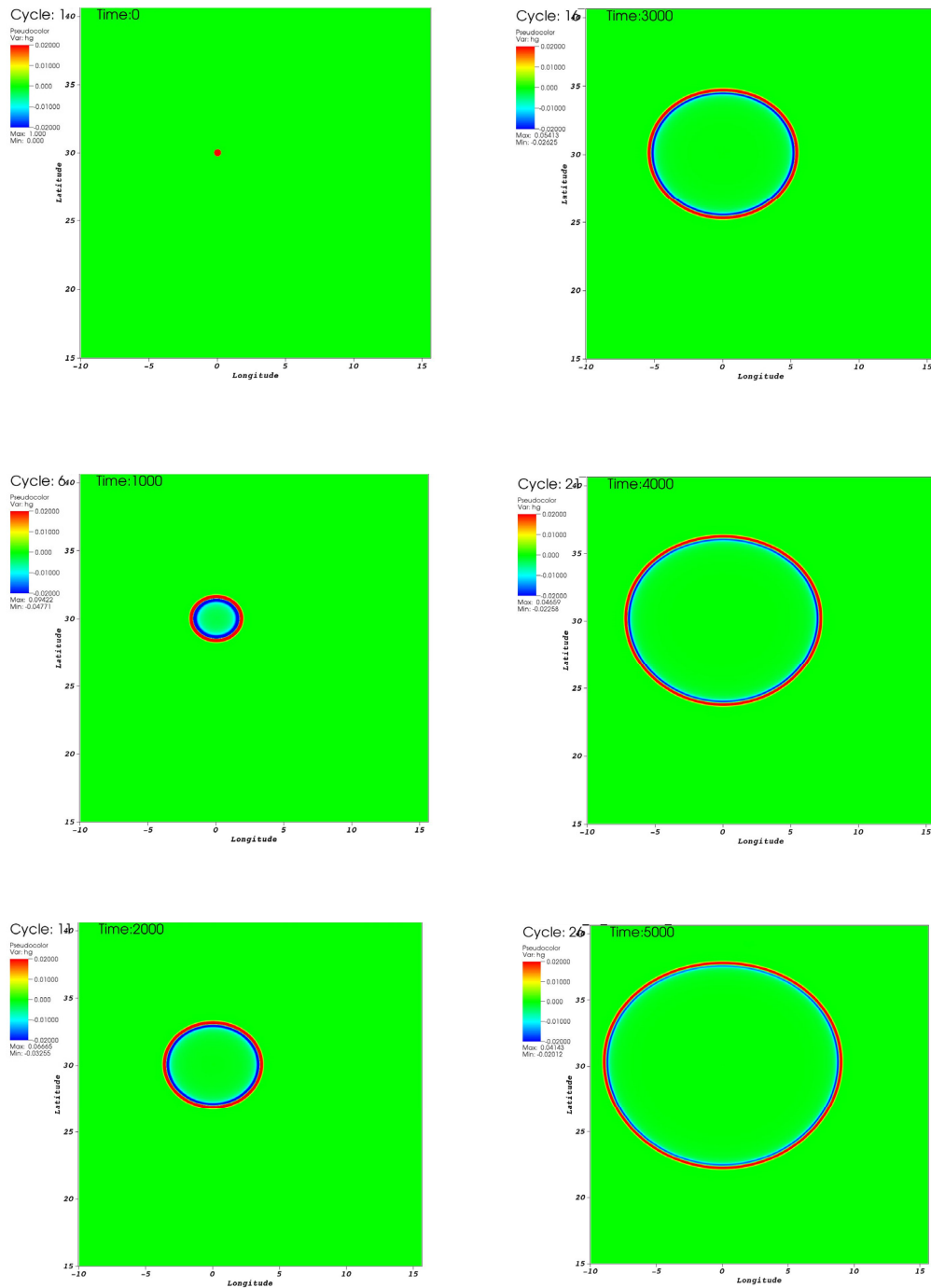


Fig. 3.4 Far-field propagation benchmark: Idealized Gaussian source on northern hemisphere. Each slide represents 1000 seconds.

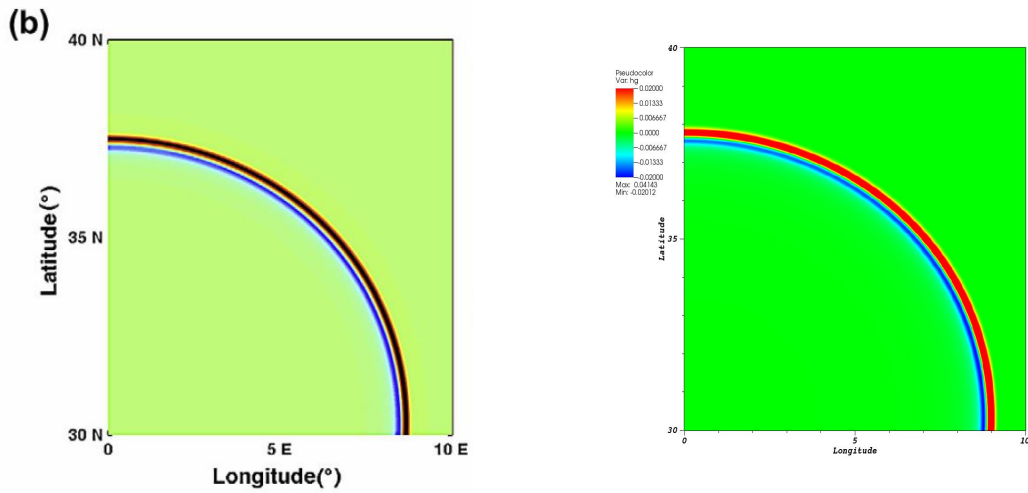


Fig. 3.5 Gaussian Initial condition benchmark at 5000s. Left image shows the original result from [69], imagine on the right shows the result for TRITON-G.

Overlapping of the results proved excellent agreement between waves; using this as a guide it can be concluded that the numerical methods reproduce the propagation wave correctly thus validating them.

In conclusion, it was shown that the spherical implementation was correctly used; it should be mentioned that for this work the domain's latitudes are relatively close to the equator ($\pm 20^\circ$), hence the effect of Coriolis is barely noticeable during the propagation; moreover earlier investigations has found that this effect is of minimal importance for tsunami wave fronts ([70], [71]).

3.5 Inundation

As mentioned before in section 2, since the inundation areas of interest correspond to cases with lengths of just a few kilometers, the SWE in Cartesian coordinates (2.37) are suitable to determine the tsunami run-up. For this purpose we use the implementation developed by Sugiyama et.al ([72], [73]) which is briefly explained here. Fig. 3.6 depicts a sketch of the definition of inundation height and run-up height.

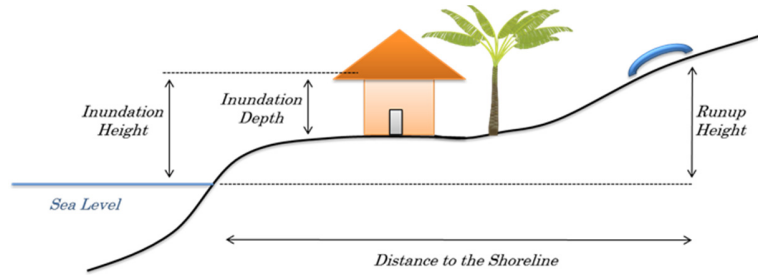


Fig. 3.6 Sketch of the Inundation and run-up heights on a shore slope

This implementation employs the Surface Gradient method (SGM) [74] to solve the SWE. It uses the data at cell center to determine the fluxes \mathbf{F} and \mathbf{G} . In general depth gradient methods cannot determine the water depth value at cell interface accurately when including effects of the bed slope as well as variations in the free surface since they cannot be determined there accurately. These inaccuracies are spread during the computation resulting in an incorrect simulation of the inundation. In order to overcome this, the SGM uses a constant water level H instead of the variable h (see section 2.1). Fig. 3.7 depicts the stencil for the water depth reconstruction, it shows that by using H instead, the water depth at the cell interface ($i+0.5$) can be accurately determined. Thus to reconstruct the water depth we have:

$$h_{L,R\ i+0.5} = \max(H_{L,R\ i+0.5} - \bar{z}_{i+0.5}, 0) \quad (3.35)$$

where \bar{z} is:

$$\bar{z}_{i+0.5} = (z_i + z_{i+1}) / 2 \quad (3.36)$$

A MUSCL scheme is used to find the flux value while Local-Lax-Friedrichs [75] is used to solve the bed slope source term. For the time integration a 3rd Order TVD Runge-Kutta scheme was used. Lastly, the bottom friction is computed using Manning's formula:

$$\begin{aligned} \tau_x &= \frac{gn^2}{h_i^{7/3}} hu_i \sqrt{(hu_i)^2 + (hv_i)^2} \\ \tau_y &= \frac{gn^2}{h_i^{7/3}} hv_i \sqrt{(hu_i)^2 + (hv_i)^2} \end{aligned} \quad (3.37)$$

where n is the Manning's roughness coefficient obtained from a look-up table or parameter file, a default value of 0.025 is used on all the domain except where a database with specific values for a region is provided.

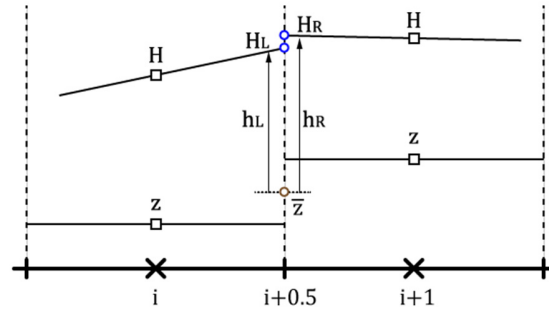


Fig. 3.7 Reconstructed water depth $h_{L,R}$ for inundation [73]

The particularity of this run-up implementation lies on the assumption of a thin film of water ε on the land. This parameter, which is set to be small compared to the wave height, allows to compute the wave propagation over land keeping it stable. On the run-up, if after computing the water level following the description in the previous paragraphs its value is found to be less than that of ε (i.e. $h < \varepsilon$) then the water level is set equal to ε while the

momentum set at rest ($hu=hv=0$) on a grid point. This implementation has proven to be robust and stable to compute tsunami run-up under different benchmarks and simulations.

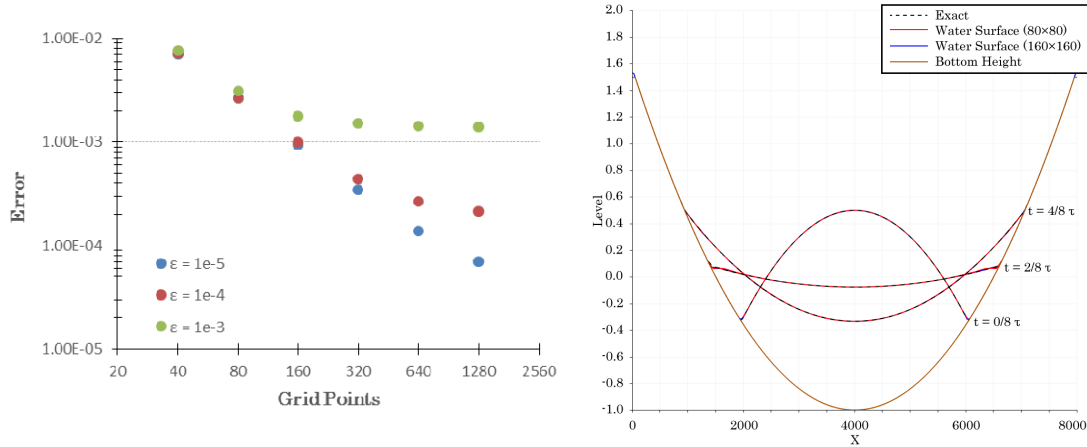


Fig. 3.8 Parabolic bowl problem. Left: water depth error for different values of thin-film ϵ ;

Right: Cross section at $t = \frac{\tau}{2}$ and $\epsilon = 10^{-4}$ [73].

As an example, Fig. 3.8 shows one of the results for the Parabolic Bowl problem [76], as it can be seen, this technique is able to track almost identically the analytical solution of the parabolic bowl on different grid sizes as it evolves in time with minimum error.

3.5.1 SWE Benchmark: Dambreak

In order to test the accuracy of the implementation to solve the SWE the dam break benchmark is used [77]. This one dimensional problem is important for verification since analytical solution is available. A horizontal and frictionless channel of length 1 m is considered, and it is broken instantaneously at time $t=0$. The initial state of the water depth is given by:

$$h_0 = \begin{cases} 1.00 & (x < 0.5) \\ h_{low} & (x \geq 0.5) \end{cases} \quad (3.38)$$

$$u_0 = 0$$

where two cases are considered: Case A with $h_{low} = 0.10$ and Case B with $h_{low} = 0.01$. The CFL number is 0.1 and two grid sizes nx are used, $nx = 100$ and $nx = 800$.

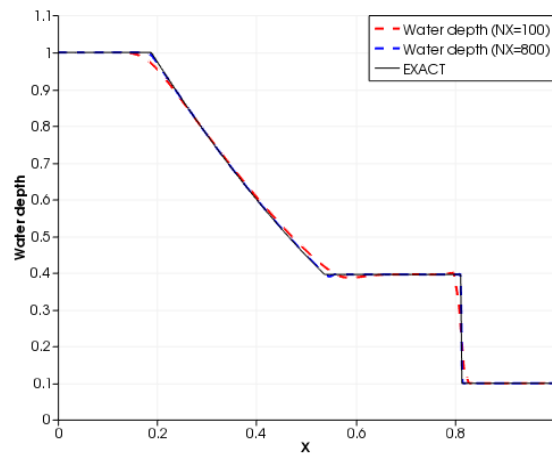


Fig. 3.9 Dam break computation at 0.1s Case A

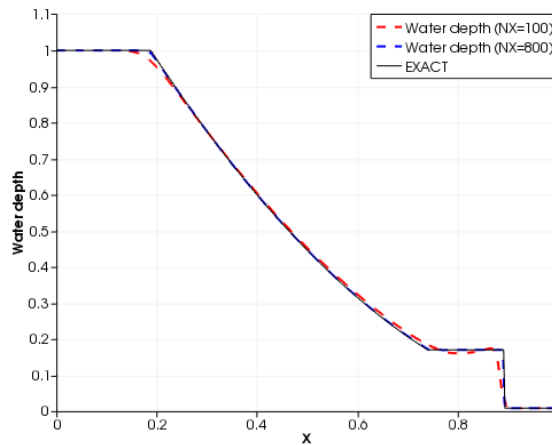


Fig. 3.10 Dam break computation at 0.1s Case B

Results at time 0.1 are shown in *Fig. 3.9* and *Fig. 3.10* for the two depth ratios. The results show excellent agreement with the analytical solution. Both shock wave and rarefaction wave are predicted without any numerical oscillation.

3.6 Tsunami Source Model

TRITON follows the usual three-step simulation style: *Generation-Propagation-Inundation*. The models used for propagation and inundation were explained in sections 3.1 and 3.2 respectively. In the case of *Generation*, it depends on knowing the exact parameters of the source to create an initial condition. This has proven to be essential in order to obtain an accurate simulation, however due to the complex nature of the source dynamics during an earthquake and the difficulty to track it in real time, currently is beyond our grasp to obtain these parameter precisely and instantly. Therefore since the goals of this work do not fall on the study of sources, instead of implementing a dynamic source generation we opted for a coseismic deformation. This deformation is calculated from the theory of displacement fields proposed by Mansinha et. al. [78].

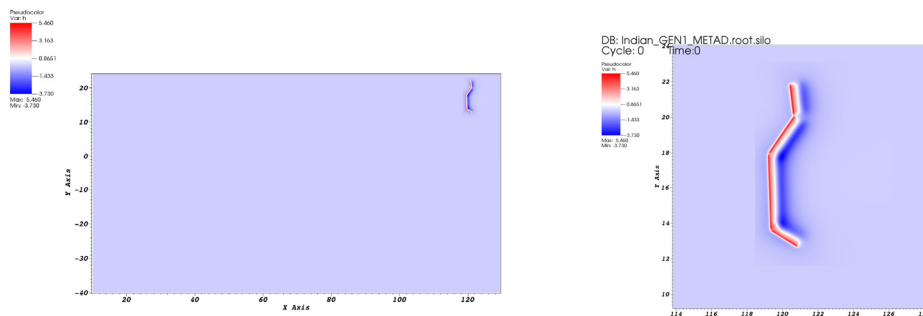


Fig. 3.11 Tsunami Fault Source: Manila Trench, zoomed at right

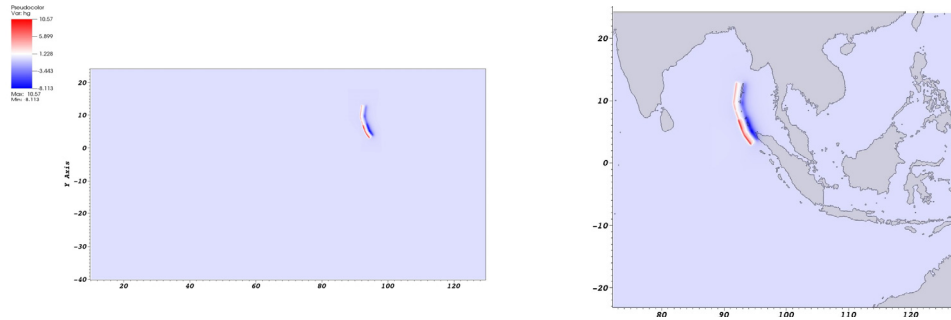


Fig. 3.12 Tsunami Fault Source: Java Trench, zoomed at right

Their objective was to provide a closed analytical expression that “facilitates the interpretation of near-fault measurements”. The expressions provided, valid at depth and surface, consist solely on algebraic and trigonometric functions that can be readily evaluated numerically based on a few source parameters like: dip, strike, slip and length. These parameters are provided to TRITON-G through a parameter file at the start of the simulation. Moreover these source parameters can be supplied from an online database to reflect the latest information available. The original code, kindly provided by RIMES was written in C language, therefore this source generation code was ported to GPU using CUDA. Due to the pure algebraic nature of the equations and the lack of access to neighbor point’s information the speed up obtained is over 100 times. Two examples of fault sources generated are depicted in Fig. 3.11 and Fig. 3.12 for the Manila and Java trench respectively.

3.7 Boundary Conditions

The domain used can contain two kinds of boundaries: *open*, which let the wave out or *wall*, which reflects the wave in. A wall boundary condition creates a total reflection when a wave hits a dry point. Fig. 3.13 depicts the stencil used for this case, the orange dots represent *dry* points ($z_i \geq 0$) while the blue ones represent *wet* points ($z_i < 0$). For instance, when a wave traveling left-right reaches the dry point at f_i , it is reflected in the opposite direction by substituting the values for water depth and momentum on the mirrored dry points as:

$$h \begin{cases} h_{i+2} = h_{i-2} \\ h_{i+1} = h_{i-1} \end{cases} \quad hu \begin{cases} hu_{i+2} = -hu_{i-2} \\ hu_{i+1} = -hu_{i-1} \end{cases} \quad (3.39)$$

Furthermore, an open boundary condition could be set in the case of a domain edge with all stencil points as wet points. Leaving the sign of the momentum unchanged on (21) while computing the Riemann invariant just in one direction, creates the effect of an open boundary, effectively letting the running wave vanish outward through that edge.

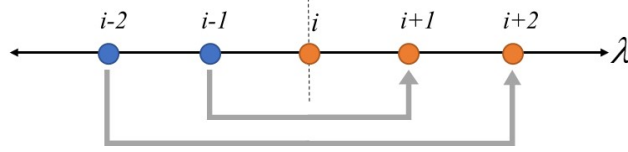


Fig. 3.13 Wall Boundary stencil with reflection at point i . Orange dots represent land and blue dots water

As it can be noted in equations (2.42), the term $\cos \theta$ in their denominators produces a discontinuity at the poles of the spherical coordinate system. Hence when working on a complete sphere, special techniques and treatment have to be used to compute over the poles without divergence. However in our case the domain chosen represents a portion of the Earth centered in the Indian Ocean and doesn't extend near the poles in any circumstance, which allows us not to be affected by the poles discontinuity. Taking all this into account, for our domain (Fig. 4.4) the boundaries used are: open boundary condition at the South and East edges, and wall boundary condition at the North and West edges. Also, all shorelines have wall boundary condition except for the special cases where particular areas set as *inundation* are defined; in those cases the proper run-up on the shore is computed using the methods described in previous sections.

Chapter 4. Tree-based Refinement and Bathymetry

Adaptive Mesh Refinement was initially introduced by Berger et.al. ([79], [80]) in the 1980s as a method to solve PDEs on an automatically changing hierarchical grid, thus solving for a set accuracy on certain areas of the interest instead of unnecessarily overly refining on the whole domain. Here we present the implementation of the tree-based refinement and customizations for this work's specific objectives: obtain a mesh with 7 Levels that track the coastline shape; this mesh should provide high resolution (50m) in particular areas of interest while using memory and computational resources efficiently.

4.1 Mesh Generation

The objective of refining the domain lies in producing a mesh that can provide high accuracy just in areas of interest. Refining the whole domain up to a high resolution would produce an unmanageable large mesh. By focusing on the requirements of RIMES operation, special customization can be introduced to fine-tune the refinement. Hence all the advantages of the adaptive refinement are conserve while a novel implementation is developed to fit our specific resources needs and focus only on regions of interest.

4.1.1 Computational Domain

Since RIMES was established as a warning-system organization in the Indian Ocean, it is reasonable that the domain chosen to perform this work is a large portion of this ocean. This domain is shown in *Fig. 4.1*; the base level uniform mesh has a size of 3584 by 1920, the geophysical extension is: Latitude: $[-35^\circ, 30^\circ]$, Longitude: $[20^\circ, 140^\circ]$.

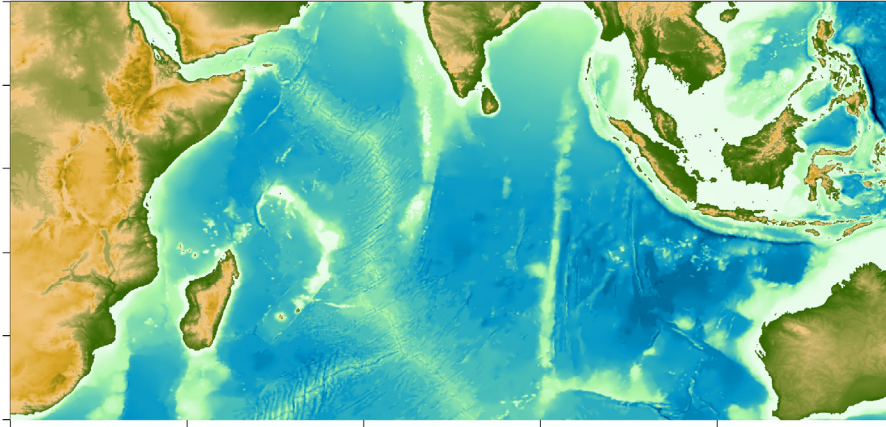


Fig. 4.1 Study case. Indian Ocean domain

This domain is initially divided into blocks which will serve as the base for the refinement intended. This process is described in the following sections.

4.1.2 Tree-based refinement

The basic principle of tree-based refinement is to start with a uniform mesh covering the computational domain, and in regions that require higher resolution, a finer subgrid is added. If more resolution is need, an even finer mesh is added. This process can be repeated recursively. This process has significant effect on reducing the mesh size only if the areas of interest are relatively small compared to the domain.

There are several ways in which the mesh can be organized to generate the subgrids. Patch-based, cell-based and block-based are three of the most common ways to arrange the mesh. Patch-based requires introducing rules and algorithms to determine ways to cluster points, this can prove to be not so efficient and programmatically complex. The cell-based arrangement offers the advantage of a very detailed way to refine since points are treated individually, however the mesh generated can be considerably large and the additional task of keeping track of all cells connectivity is a large computational overhead.

For these reason we implement block-based for refinement. Since all points are arranged in blocks of equal dimensions there is no need of complicated rules for clustering. And by treating the points as a region subgrid can be created where needed without producing an un-manageable large number.

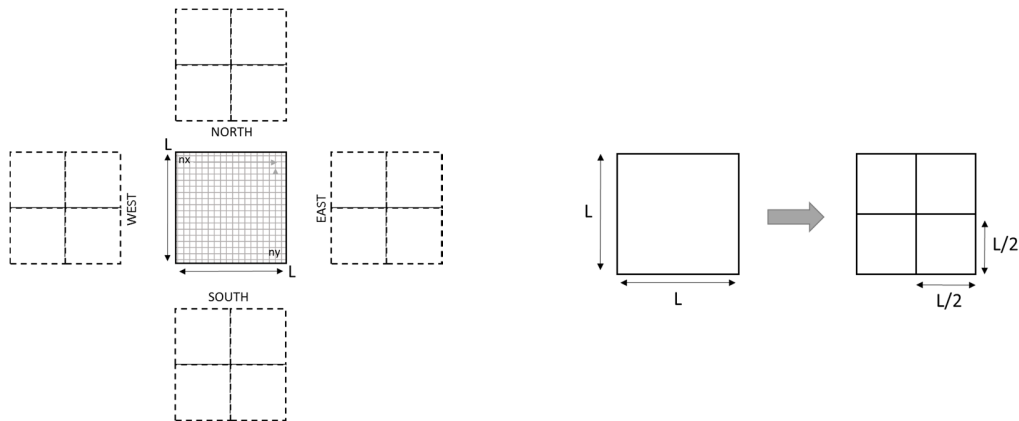


Fig. 4.2 Block-base refinement. Left: Block with neighbors. Right: Parent-children refinement

A block is composed of cells organized two dimensionally in symmetrical rows and columns. A representation of this block is shown in the left in Fig. 4.2. The edges of a block will be referred to as North, South, West and East. Each block has neighbor on its edges with whom they share an overlapping point at the edges. All blocks are interconnected by a logical tree that keeps the *pedigree* of every member in a branch. Thus it is possible to find readily which block is the immediate neighbor or parent by simple looking at the tree branch.

When a block requires refinement it produces four children that splits the parent branch into four new branches as shown in the right of *Fig. 4.2*, this is known as a quadtree. Each of these children has the same number of cell points as the father however the physical length is half hence doubling the parent's resolution. This process can be repeated recursively for as many times as the application needs it. As more levels are requested, each time a block gets refined its parent branch is split in four more, creating deeper trees, as shown in Level 4 of *Fig. 4.3*. the advantage of preserving this connectivity is that is easy to trace up several generations the original block or neighbors; also this tree structure permits to determine the surrounding neighbors by looking at the brothers in a branch.

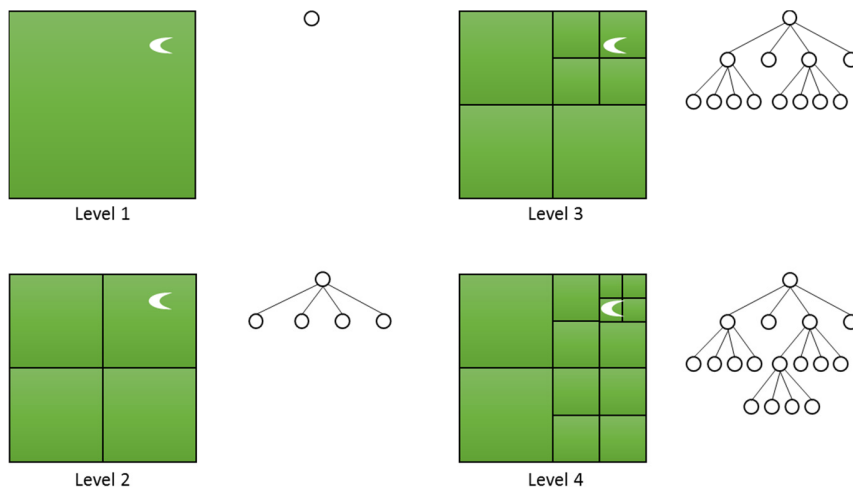


Fig. 4.3 Quadtree structure construction for a domain with a mooned-shape focus

Then, in order to generate the refined mesh, we start with a coarse uniform domain divided into blocks. A block can be flagged for refinement recursively until it reaches a set threshold, effectively creating several hierarchical levels. A quadtree data structure is used to keep track of the blocks connectivity. *Fig. 4.3* show this quadtree structure in a graphical representation; the connectivity between blocks store information about the block and its

neighbors in a simple way. This structure is vital in order to update neighboring values at the blocks's edges.

If we denote a level by l , the difference on spatial resolution between two adjacent levels is the refinement ratio r , and should be a positive integer as:

$$r = \frac{\Delta x_{l+1}}{\Delta x_l} \quad (4.1)$$

The value of this ratio is a free parameter that is problem dependent. However using large integers introduce issues in the computation, the existence of an abrupt change from one level to the next requires special treatment, especially when complex bathymetry or topography is involved. In order to avoid this and to create a smooth transition between levels a refinement ratio of two is chosen for this study.

Although we follow the refinement procedure used in tree-based refinement, a couple of customizations are introduced to tailor-adapt it to our specific interests. Also, since the domain represent bathymetry and does not change in time, the mesh can be generated at the beginning a single time. This permits keeping the advantages of tree-based refining while removing the further overhead intrinsically associated with re-meshing during the simulation.

Moreover, in general, tree-based refinement employs an error estimate as a rule to determine if a block should be flagged for refinement, however in our implementation the refinement rule depends not on an error threshold but on a *target* resolution combined with two factors:

- a) The block's distance from the coastline
- b) The presence of a focal area.

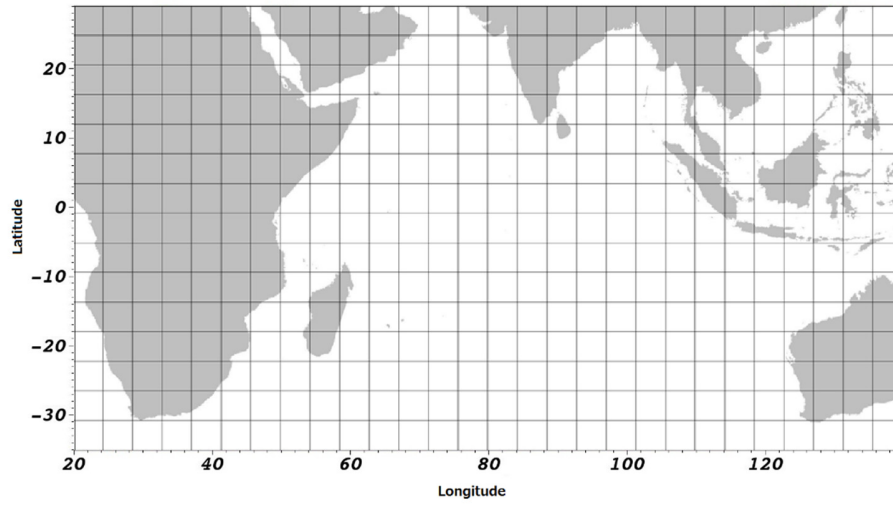


Fig. 4.4 Indian Ocean Domain

4.1.3 Refinement by distance

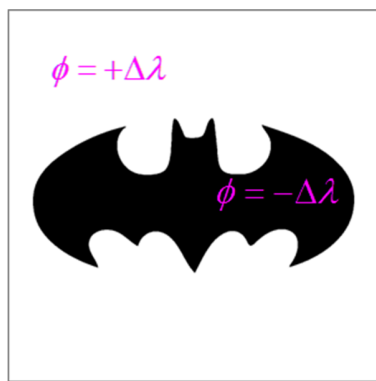
The refinement rule's first factor depends on the distance of the block to the shoreline, the objective is to recursively refine blocks close to the coast until reach the target high-resolution threshold, while blocks far in the ocean can remain with a coarser resolution. This process involves two steps:

- a) Determining the block's distance from the coast
- b) Checking if its distance is within refinement.

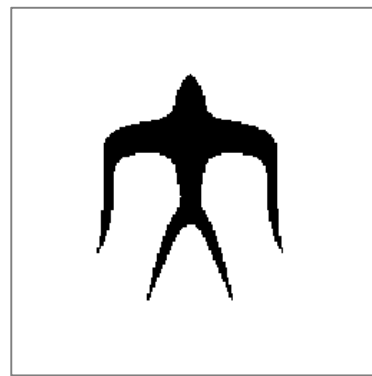
To accurately estimate the geo-distance between two points can be a complex task since the surface of the Earth is not a perfect sphere. However, for the purpose of refining this mesh, is enough to use a rough estimate of the distance between the shoreline and the blocks. This is achieved by creating a signed distance function based on the Level-set method [81]. Based on this theory, in order to generate the distance function ϕ the following equation has to be solved:

$$\frac{\partial \phi}{\partial t} + \text{sig}(\phi)(|\nabla \phi| - 1) = 0 \quad (4.2)$$

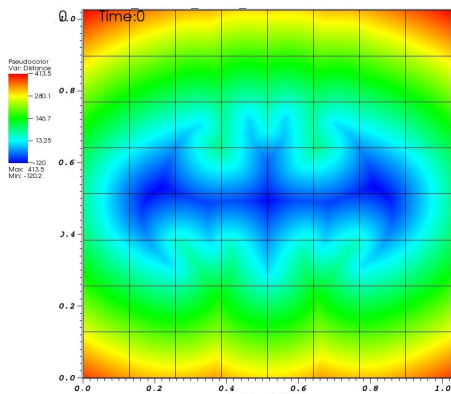
The initial condition for this function is generated by defining positive values in the outside area of a surface and negative values in the inside, while setting as 0 the interface between them, this is known as the *zero level* of the function. This initialization is shown in Fig. 4.5 (a), where the black points represent the inside of the surface. Having this initial condition then equation (4.2) is numerically integrated to obtain the correct distance.



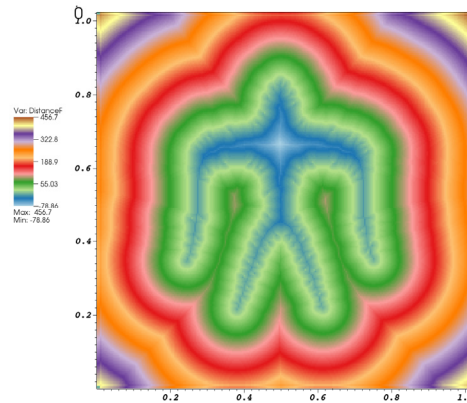
(a)



(c)



(b)



(d)

Fig. 4.5 Samples to generate distance function on objects with irregular shapes. (Credits: Batman logo is a trademark of Warner Bros. Tsubame emblem is a trademark of Tokyo Institute of Technology)

In general, by setting the initial values as the grid resolution $\Delta\lambda$ it takes two iterations to obtain the correct distance value next to the zero level subsequent iterations correct the second point from the zero level and thus on. In this work an Euler's integration is used for the time derivative and WENO scheme [82] is applied to solve the spatial derivative.

Initial tests performed on surfaces with irregular shapes are shown in *Fig. 4.5*, the resulting distances are shown below each surface. As it can be seen from this, this method proves to track different shapes and generate distances for every point in the domain.

With these promising tests as a guide the same method was applied to the Indian Ocean domain. Here the distance function definition came rather natural: positive values for points in land and negative values for points in the ocean (bathymetry). The *zero* level of the function is represented by the points along the shoreline ($z=0$). A depiction of the result is shown in *Fig. 4.6*, distances from the coastline are obtained for the domain. In this case, positive distances represent cells on land while negative distances represent cells on the ocean.

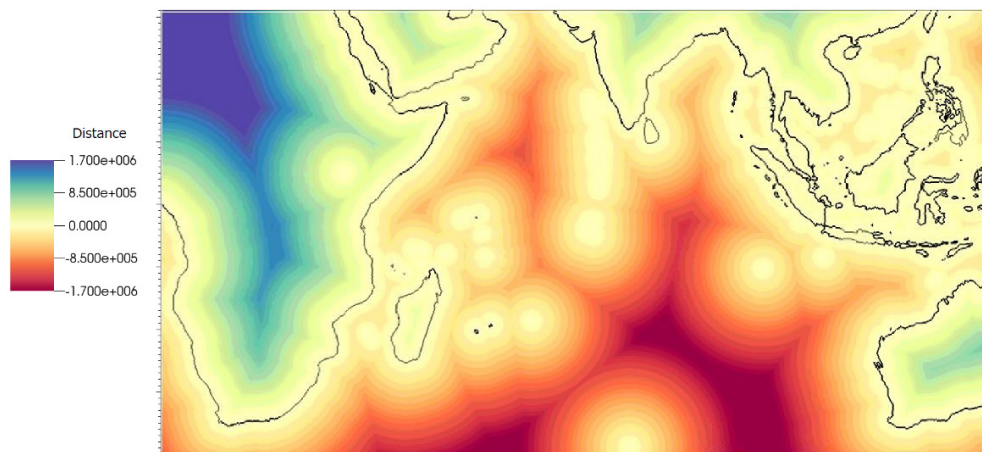


Fig. 4.6 Signed Distance Function. Positive values represent distances on land, negative values distance on water

Once the distance of all cells in a block is determined, each block is tested for refinement. Blocks with one cell or more within a certain distance from the coast (*refinement stripe*) are flagged for refinement until they reach the fine-target resolution. The width of the refinement stripe is problem dependent and is input by the user based on their needs. A different refinement stripe value can be chosen for every level, an illustration of this concept is shown in Fig. 4.7 for four levels.



Fig. 4.7 Refinement stripe representation for L2, L3 and L4.

The result of applying this refinement procedure to the Indian Domain is shown in Fig. 4.8. The refinement stripe distance is 100 km and halves on every subsequent level. The initial resolution at ground Level 1 is 2 arc-min; the fine-target resolution is 0.03125 arc-min (~56 m) for a total of 7 levels created.

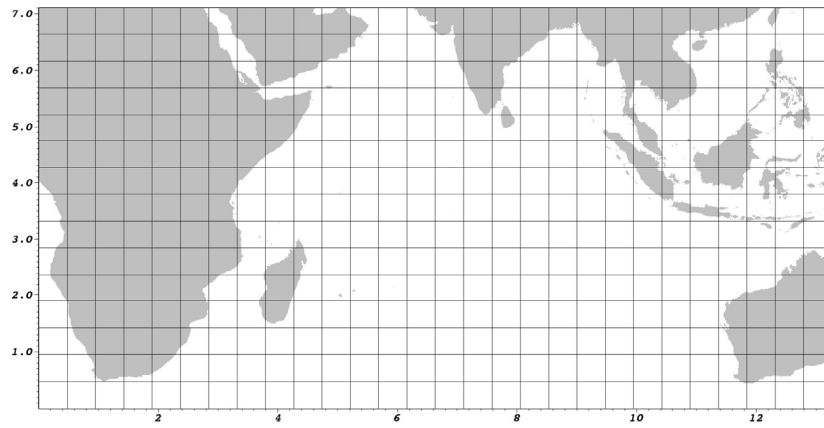


Fig. 4.8 (a) Level 1

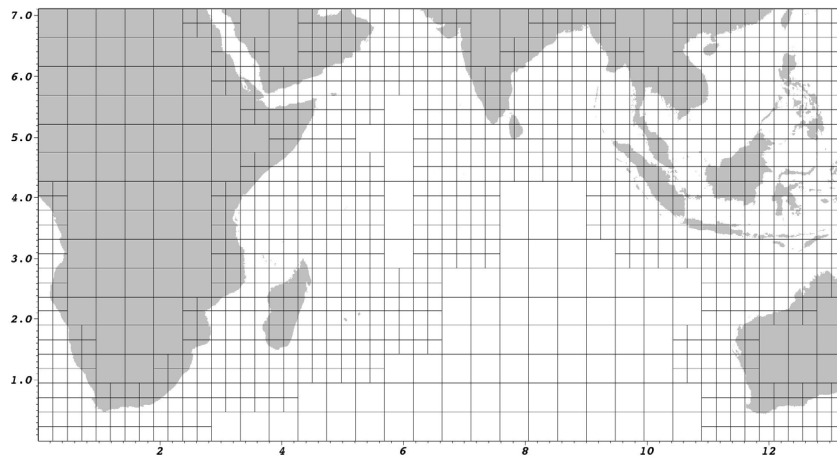


Fig. 4.8 (b) Level 2

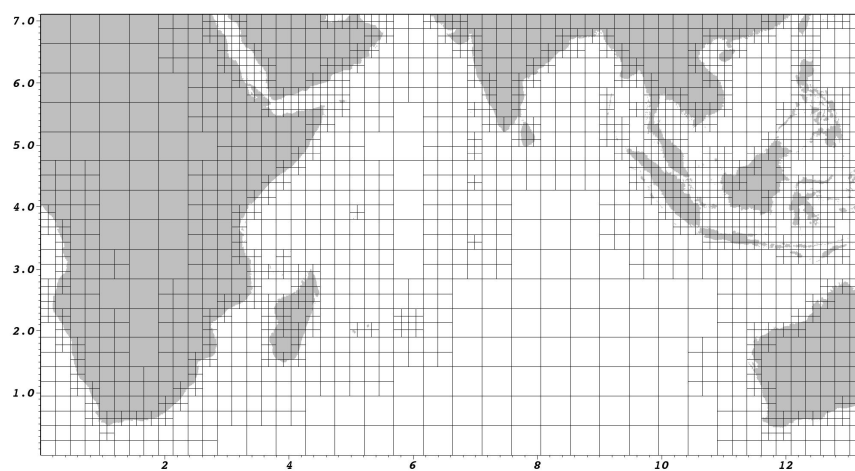


Fig. 4.8 (c) Level 3

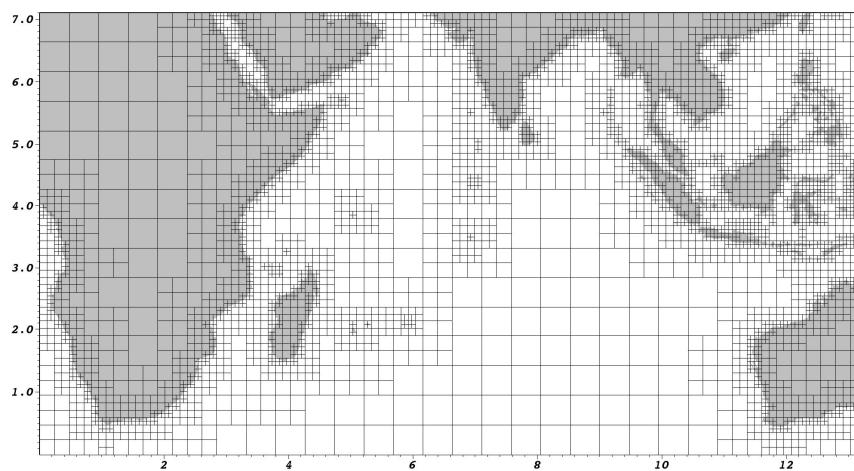


Fig. 4.8 (d) Level 4

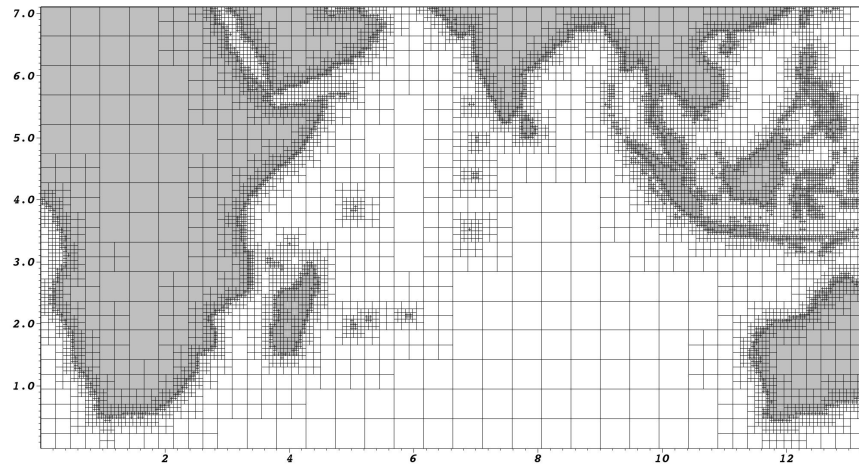


Fig. 4.8 (e) Level 5

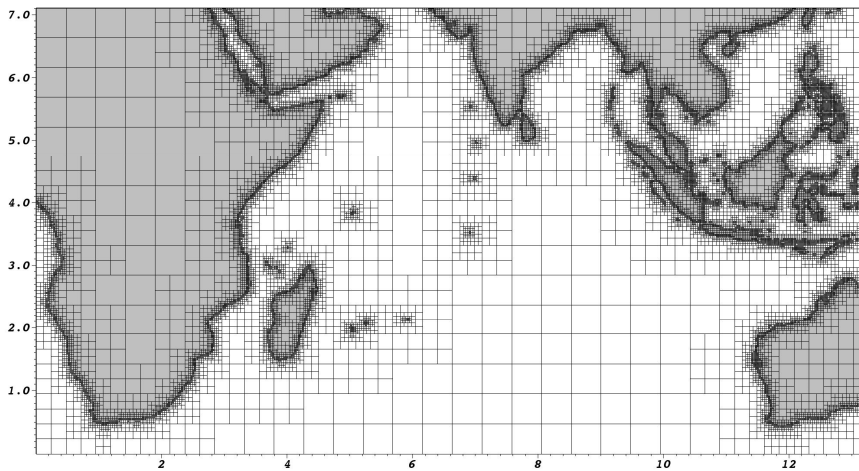


Fig. 4.8 (f) Level 6

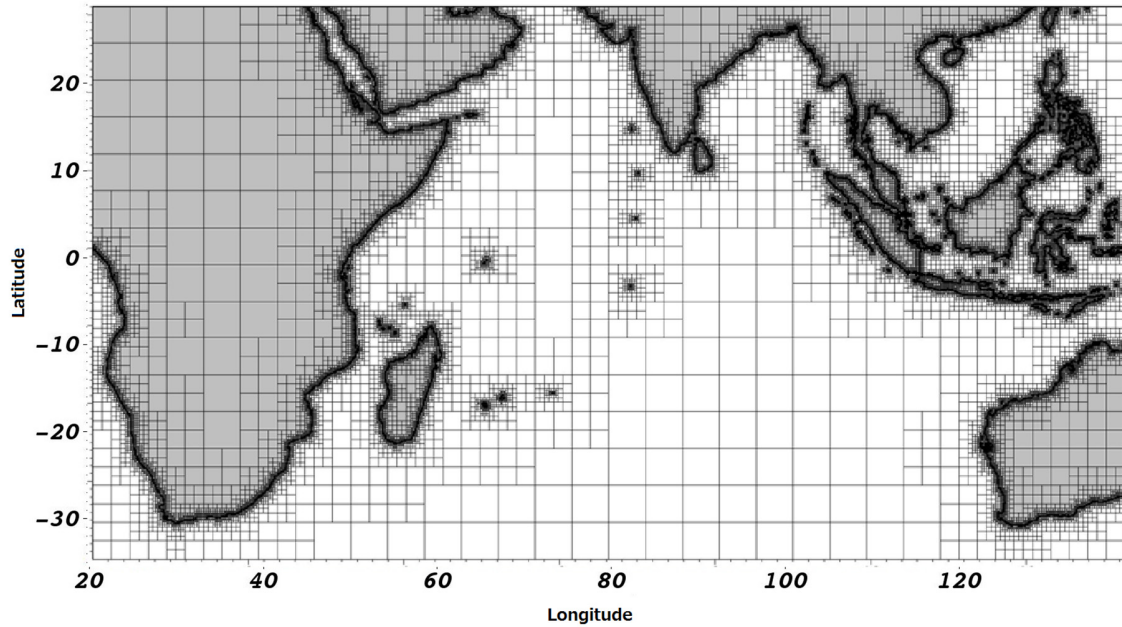


Fig. 4.8 Distance Refining Process Indian Ocean Domain using refinement stripe. Total 7 levels, highest resolution 50 m.

The mesh produced following can properly refine the complex coast shapes and cover the desired refinement stripe. An important observation is that independently of the distance, blocks that lie on the tsunami source fault also get flagged for refinement in order to obtain a more accurate initial condition and preserve the wave front better.

One downside of this refinement process is that the number of blocks generated can be considerably large, more than 230,000 in this case. Also the memory needed to store this mesh is above 120GB. Motivated by the objective of using resources efficiently and the need of high resolution just in certain specific areas a second refinement factor was introduced.

4.1.4 Refinement by Focal Area

The second factor of refinement is a constraint added to the first. This constraint consists in locating on the domain a convex polygonal area which serves as a refinement delimiter. This area is referred to as *focal area* (FA) and is possible to locate more than one. An example of a focal area is shown in *Fig. 4.10* (b) as a circle covering the domain.

Since this is an additional constraint to the first refinement step, only blocks flagged for refinement at the first step need to be tested again. On this second test, the block's four vertex coordinates are compared against the focal area vertices to determine if it is inside or outside the area. If a block is completely outside the focal area, then it is un-flagged for refinement. Only blocks partially or totally inside the focal area remained flagged for refinement.

The process of determining if a block lies inside or outside a focal area is based on collision detection theory. There are several methods available for this like the Grid-based approach, Bounding Box method, Discrete Collision Detection or the Separating Axis Theorem (SAT) ([83], [84]). We chose the latter because it is a well-known theorem applied to physical simulations [85] and consists of a relatively *light* algorithm for 2D, which allows to test large number of blocks rapidly.

The SAT states that: if two convex objects are not penetrating, there exists an axis for which the projection of the objects will not overlap (*Fig. 4.9*).

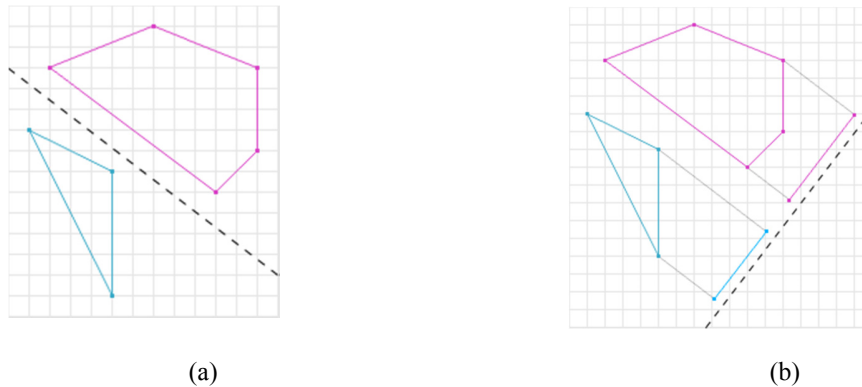


Fig. 4.9 SAT. (A): Two non intersecting convex polygons; (b) Projection of the non intersecting polygons [86].

Hence by using this theorem and the basic concepts of normal and projections, the algorithm to determine the intersection can be summarized as follows: the axis used can be the normal of edges, then first compute the normal of each edge of the polygon, second each normal has to be tested to the projection of the other polygon by using the dot product to determine if they lie or intersect. If, for all axes, the shape's projection overlap, then we can conclude that the shapes are intersecting.

A proof of concept of refinement by focal area using the SAT to determine if blocks are inside or out is shown in Fig. 4.10. Here the focal area is represented by a circle shaded in orange. As comparison, Fig. 4.10 (a) shows the distance refinement as explained in the previous; Fig. 4.10 (b) shows the result of including the FA, it is clear that only blocks inside the region continued refinement by the rest outside were left untouched.

It is important to note that since the focal area is an additional constraint, it can be toggled active after any chosen level. Hence, a specific number of levels can be refined without this constraint while the following are affected and delimited by the focal area. In the previous proof of concept blocks outside the FA were limited to level 3 while blocks inside had a target of 5 levels.

Furthermore, a focal area also indicates that blocks within, with the highest resolution, are to be treated as inundation areas if they include dry points.

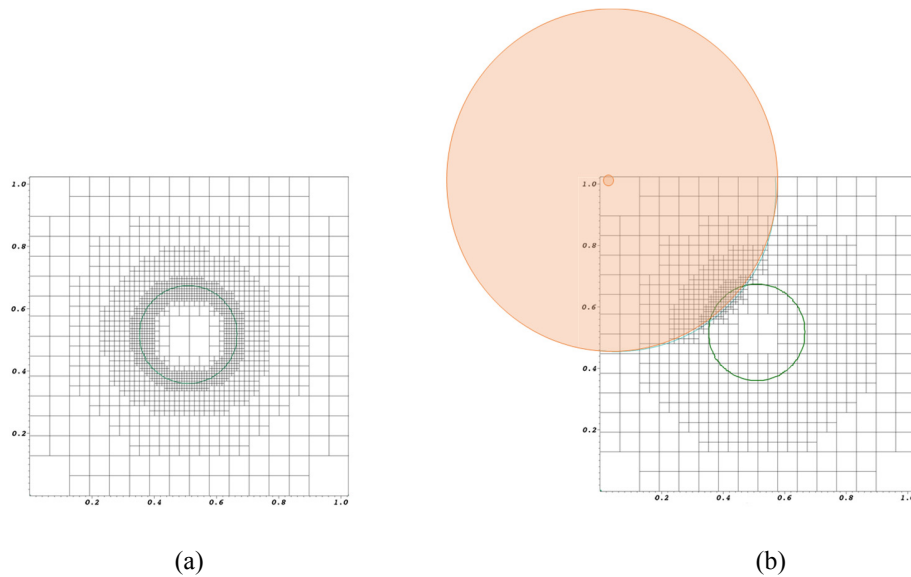


Fig. 4.10 Focal refinement proof of concept. (a): 5 level refinement with no FA; (b): 5 level refinement with FA represented by a circular shape.

As mentioned above, the focal areas are defined by the interest of the user and more than one can be included at the same time. For this work four FA are submitted by RIMES since they represent their research interest, named: Mozambique, Comoros, Seychelles and Sri Lanka. The extension and shape of each FA is different and was chosen to coincide with higher resolution bathymetry databases owned by RIMES and also because they represent areas of inundation that they need to be monitoring. These FA and their specific location and shape are shown in *Fig. 4.11* shaded in green.

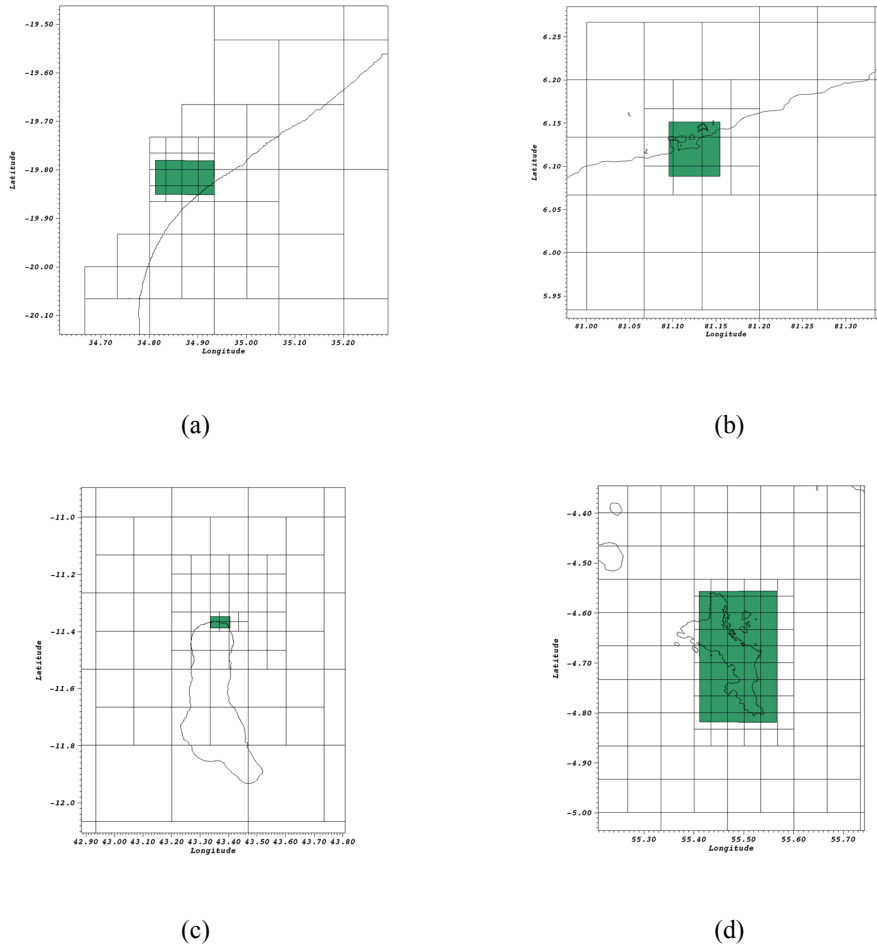


Fig. 4.11 Focal areas used in this work. a) Mozambique; b) Sri Lanka; c) Comoros; d) Seychelles

4.1.5 Dry area removal

The final step in the mesh generation process consist in the removal of *dry blocks*. Considering that tsunami inundations, with few exceptions, generally extend tens to hundreds of meters inland, then it becomes clear that blocks located deep inland are an unnecessary load in the computation. By using this insight all blocks whose cells' distances are larger than a *land-distance* threshold are considered *dry blocks*, and thus flagged for removal and deleted from the domain.

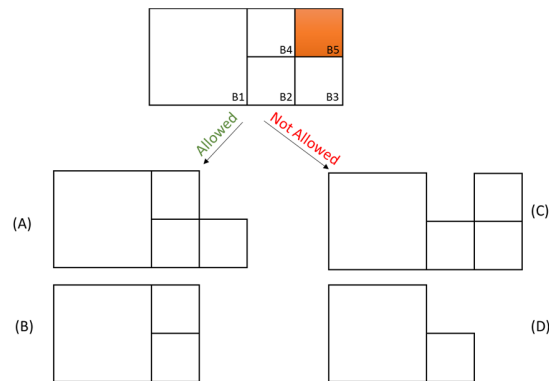


Fig. 4.12 Inland dry block removal cases allowed and not allowed

Fig. 4.12 show a graphical explanation of removing a block and the cares that must be taken. Cases (A) and (B) represent situations where a block can be removed, this happens when the complete edge of a block has a neighbor. On the other hand cases (C) and (D) show examples where a removal is not allowed even if the block was originally flagged for it. The reason this operation is not allow is that edges of a block are not completed covered by a neighbor block.

Finally, the complete result of implementing all these procedures, distance refinement, focal area refinement and dry-block removal, to generate the mesh in the Indian Ocean domain is shown in Fig. 4.13. The four focal areas used are: Mozambique, Comoros, Seychelles and Sri Lanka. The focal area constraint start after Level 3. The reason to choose this particular level has to do with its grid resolution and the bathymetry database. The initial uniform block at level 1 has a resolution of 2 arc-min, hence level 3 has a 30 arc-sec resolution which is also the highest resolution available for GEBCO bathymetry database (described in section 4.3). Hence by refining all coastlines up to level 3 it can be guarantee that the most realistic available values are used directly in the coastlines without any need of interpolation.

Comparing with Fig. 4.8 it can be seen that this time the refinement at higher levels is limited to within the focal areas (pointed by the arrows). Also, all dry blocks exceeding the land-distance threshold of 10km were removed from the mesh.

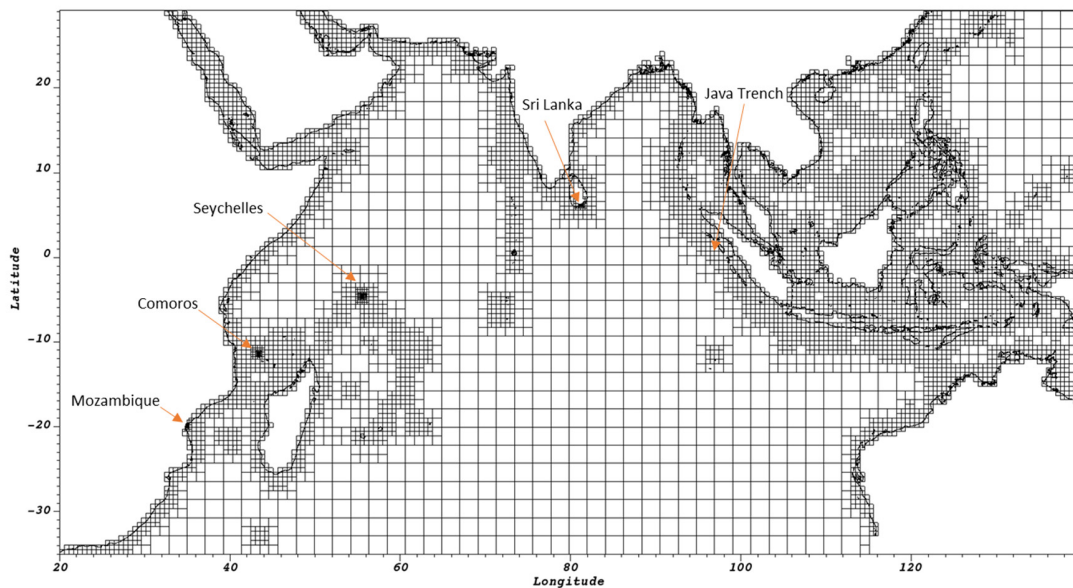


Fig. 4.13 Mesh Refinement for Indian Ocean Domain with 4 Focal Areas: Mozambique, Comoros, Seychelles and Sri Lanka.

In conclusion, the block-based refinement produced a mesh that can trace the complex coast shape in the domain, the effect of using focal areas served to focus high resolution just in areas of interest and by doing this also the number of blocks generated drastically decreased to 7847 while the memory needed to store them became less than 20GB. In total there are over 30,000,000 points in the domain. The resulting mesh represents the best balance between using resources efficiently and producing accurate results where needed. It is also important to notice that TRITON-G stores this mesh as a database thus no needing to generate it every time a simulation on this exact same domain is required.

4.2 Block Halo Update

Blocks have neighbors on their four edges, in order to compute on all the domain correctly, each block must communicate its results with its surrounding neighbors. After each time step, blocks must exchange results with their neighbors before the next iteration. For this purpose they share a boundary layer in their adjoining sides. This layer or halo extends over the neighbor's grid as depicted in *Fig. 4.15* and can represent one of three kinds of swapping: copying, coarsening or interpolating.

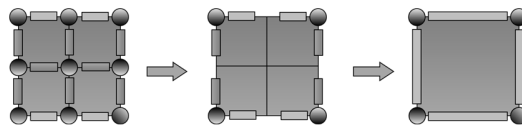


Fig. 4.14 Cell coarsening, averaging down

If two neighbor blocks are at the same level then the halo is readily updated by exchanging values directly without any further computation necessary, this represents a copying swap.

In the case of two neighbors at different levels (l and $l+1$) then additional computation is required before the halo swap.

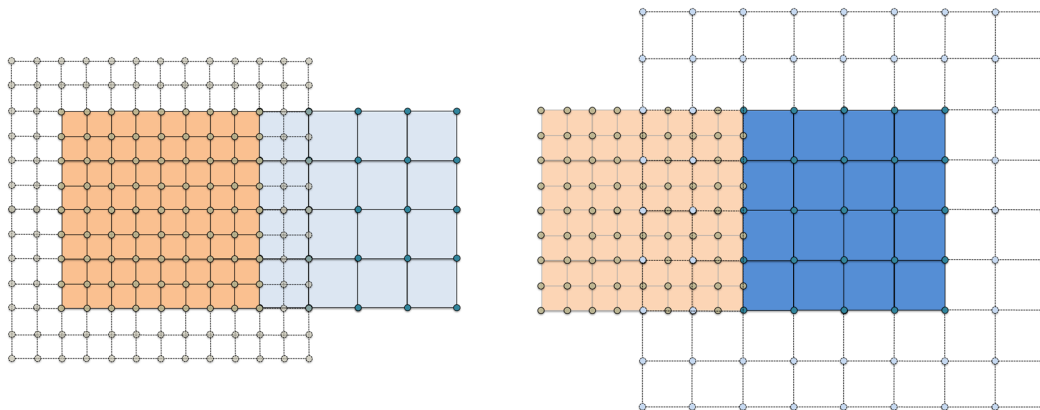


Fig. 4.15 Halo update for neighboring blocks; blue block at level l , orange block at level $l+1$.

Left: Interpolation case; Right: Coarsening case

If the block's neighbor is one level up (see right diagram on Fig. 4.15) then values for the halo are averaged down from the block with higher accuracy before swapping. Moreover, this coarsening has the effect of passing down better accuracy to blocks with lower resolution like in a cascade effect.

The last case, interpolating, occurs when the block's neighbor is one level down (see left diagram on Fig. 4.15). The values for the halo are interpolated from the neighbor block. In order to keep high accuracy and a smooth transit of the wave through these different levels a third-order polynomial interpolation is used, similarly as in equation (3.21). As an example, Fig 10 depicts the stencil used to interpolate in one dimension, only the portion where the blocks overlay is showed.

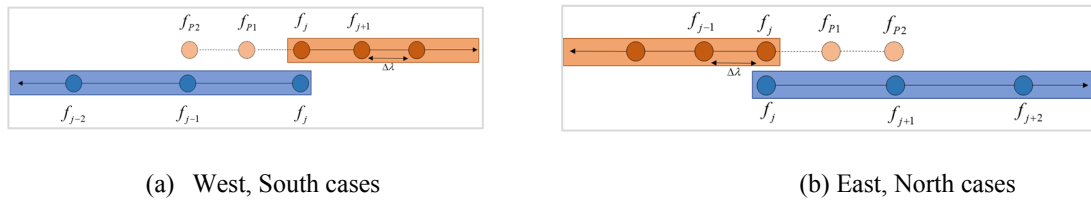


Fig. 4.16 Halo interpolation stencil for the four edges: north, east (a) and west, south (b)

Hence, the interpolated values for the halo can be found from:

$$\begin{aligned} f^{N,E}_{P1} &= \frac{1}{4}(f_j + 4f_{j+1} - f_{j+2}) \\ f^{N,E}_{P2} &= \frac{1}{4}(-f_j + 6f_{j+1} - f_{j+2}) \end{aligned} \quad (4.3)$$

for the north (N) and east (E) edges, which are analogous. And:

$$\begin{aligned} f^{S,W}_{P1} &= \frac{1}{4}(-f_j + 4f_{j+1/2} + f_{j+1}) \\ f^{S,W}_{P2} &= \frac{1}{4}(-f_j + 6f_{j+1} - f_{j+2}) \end{aligned} \quad (4.4)$$

for the south (S) and west (W) edges. In order to avoid spurious waves generated from interpolating the water height h , constant water level H is used instead. The original variable is recovered by using the relation $h=H-z$.

4.3 Topography and Bathymetry

The data used for bathymetry and topography comes from different sources. Initially, The General Bathymetric Chart of the Oceans (GEBCO) [87] database is used for the domain. GEBCO is available on 30 arc-seconds spatial resolution. When coarser resolution is needed, the values are averaged from this database. On the contrary, if finer resolution is needed, a third order interpolation (Eq. (3.21)) is implemented to generate the new values.

Where available, databases with more precise measurements are used to replace the original GEBCO values. In particular the focal areas should include better resolution than that provided by GEBCO. For this purpose in-situ measurements with higher precision are desirable. As mentioned earlier in the chapter four focus areas are used for this work, they represent the interest of study for RIMES: Mozambique, Comoros, Seychelles and Sri Lanka.

Databases generated by RIMES were provide to estimate the inundation more accurately in this regions. Additionally a fifth set of databases was provided to test on Phuket region.

The bathymetry resulting is shown in *Fig. 4.17*. Also a representation of the finer databases for the focal areas is shown in *Fig. 4.18*; light shaded areas represent a 150m resolution and dark areas 50m resolution.

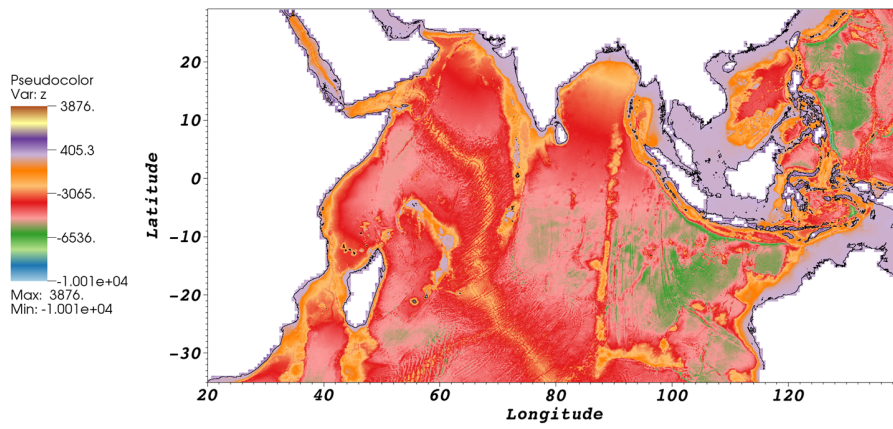
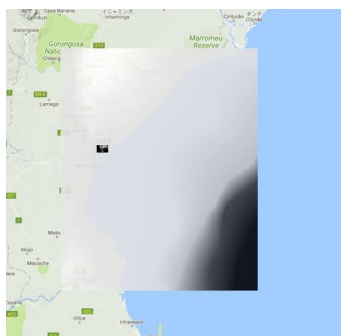
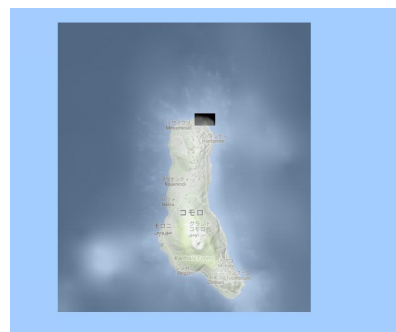


Fig. 4.17 GEBCO bathymetry and topography for the Indian Ocean domain



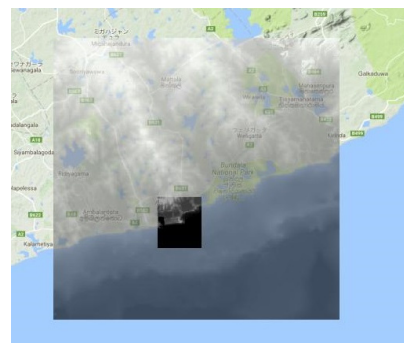
(a)



(b)



(c)



(d)

Fig. 4.18 Additional bathymetry databases for replacement with higher accuracy by RIMES. Light-gray shaded areas represent 150m resolution, dark shaded areas represent 50m resolution. (a) Mozambique, (b) Comoros, (c) Seychelles, (d) Sri Lanka

After the mesh generation all points in the blocks are updated with their respective bathymetry, the database chosen to supply the value is always the one with the highest accuracy available at that specific location.

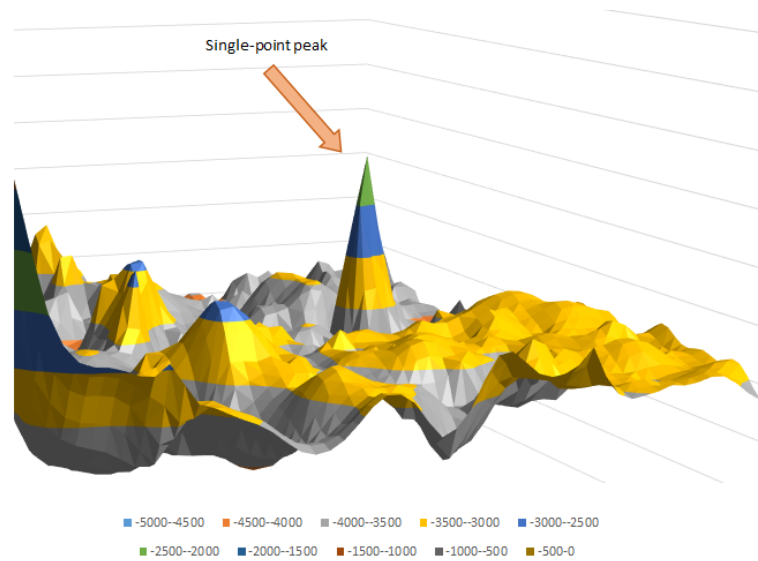


Fig. 4.19 Single-point peak bathymetry example

The bathymetry used in this work represent the real values as the best available databases provide. In order to get the most accurate and realistic result possible in the simulation great care is taken to guarantee that the original values remain untouched. There is a specific case however that needs special attention: the existence of solitary single-point coast values that create peaks with an unreasonable large gradient. *Fig. 4.19* shows an example of this kind of points in the right side of the domain. When these points are identified a cell of points around this value is updated with land values to avoid potential divergence when simulating. Since this cases are rare this simple technique proved to be non-invasive to the total model.

Chapter 5. GPU Computing

5.1 Introduction

The introduction of C-language extension CUDA [88] by nVIDIA® was a disruption in the traditional way simulations were performed. By providing a way to program their graphic cards for general purpose (known as GPGPU), researchers no longer had to rely solely on CPU processors to perform calculations. Due to the intrinsic parallelism of graphics, GPUs naturally evolved to deliver in a card hundreds, and later, thousands of processors more than CPUs. Ivy Bridge CPU-architecture chips are designed for up to 15 cores and a peak performance of 300 GFLOPS for double precision. In comparison, a GPU Tesla K40 card contains 2880 cores and a peak performance of 1.43 Tera FLOPS. The main reason behind the exceptional performance of GPUs lies in the specialized design for compute-intensive, highly parallel computation, with transistors dedicated exclusively to processing as opposed to flow control and data caching.

On GPU, cores are clustered to form an array of Streaming Multiprocessors (SMs). The programming model is based on a multithreaded problem partitioned in blocks of threads, each executed independently on the SMs.

5.2 CUDA

In this new architecture design to provide a GPU as a General Purpose Device we find CUDA. The Compute Unified Device Architecture was developed by nVIDIA, one of the

leading GPU manufactures in the world. This new language extension and hardware technology was introduced firstly in the GeForce 8 Series, Tesla and Quadro, being now a standard in almost all of the high performance nVIDIA cards. Tesla K40C cards are used in this research, three of them installed in a machine working in parallel are chosen as the configuration for operation.

CUDA provides a set of APIs designed to program the card in a readily easy way compared with before its introduction. This come to solve the problem of the steep learning curve since CUDA also is presented as a C language extension, a very common and known programming language. Therefore any C language programmer has the opportunity to easily understand the syntax and use of CUDA. The software part of CUDA provides libraries, runtime and drivers including its own compiler named *nvcc* which behaves generally speaking, like the also known GNU Compiler Collection *gcc*. CUDA provides address to the DRAM memory in a general way allowing now a flexible programming tool in scatter and gather memory operations. Now the memory can be written or read in and from any location just like in a CPU.



Fig. 5.1 nVIDIA Tesla K40C GPU used in this research, 2880 CUDA cores

5.2.1 Memory Model and Hierarchy

CUDA provides multiple memory access for every thread during execution. Each of these is designed for a particular purpose and with certain hierarchy. The programmer has the freedom to use these memories in the best suitable form for his particular problem. Each

memory has its own specific characteristics such as size and access speed that influence the application running performance.

The architecture is composed of a *Constant*, *Global* and *Texture* Memory at its first hierarchy and a *Shared* Memory, *Local* Memory and *Registers* in the next hierarchy.

Due to the present architecture of motherboards and GPU, when CUDA is used the data in the CPU, also referred as *Host*, cannot be directly read by the GPU, referred in CUDA as *device*, and a transfer must be performed before using it in any calculation inside the GPU. The hierarchy to access the memories is different between types of memory, as illustrated in *Fig. 5.2* the *Constant*, *Global* and *Texture* memories have direct communication to the CPU to send back and forth data, while the *Shared* and *Local* as well as the register are only to be accessed by a thread. Moreover a thread can also read from the *Global/Constant/Texture* memories however it cannot read directly to and from the CPU. To the scope of this research mainly the *Global* memory is used. Therefore these will be explained next in more detail.

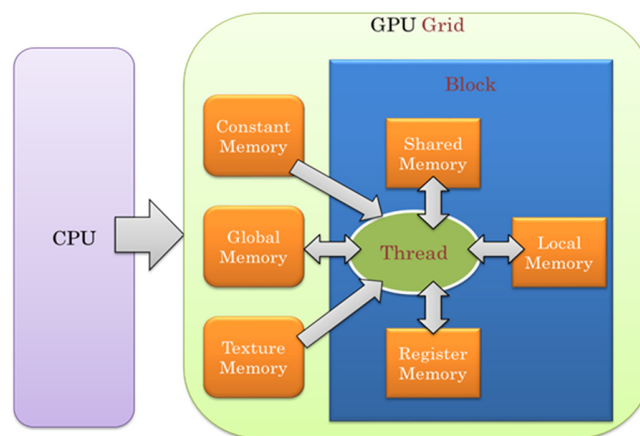


Fig. 5.2 CUDA Memory Model

The *Global* memory is the memory shared by the streaming processors and as mentioned it communicates directly with the CPU; when a program is run, the memory for

the variables must be initialized and the data uploaded from the CPU to the GPU. This memory delivers the highest memory bandwidth only when the global memory accesses can be coalesced within a half-warp so the hardware can then fetch the data in the fewest number of transactions. If the memory transaction cannot be coalesced, then a separate memory transaction will be issued for each thread in the half-warp, which is undesirable. Because of coalesced memory the threads should be arranged in a way to avoid bank conflicts when accessing the memory address, a non-coalesced global memory access pattern will reduce the performance by reducing the bandwidth and thus the speed of the calculation.

5.2.2 Programming Model

CUDA is an extension language that provides its own specific APIs. These API are readily usable with a familiar syntax and appearance. However the structure inside the GPU for the memory requires some attention since it is not usual to work in parallel schemes even using a single device. The nVIDIA's GPU provides a certain number of threads to be used in the calculation depending on its particular characteristics. These threads are organized in blocks and blocks are sorted in grids as shown in *Fig. 5.3*.

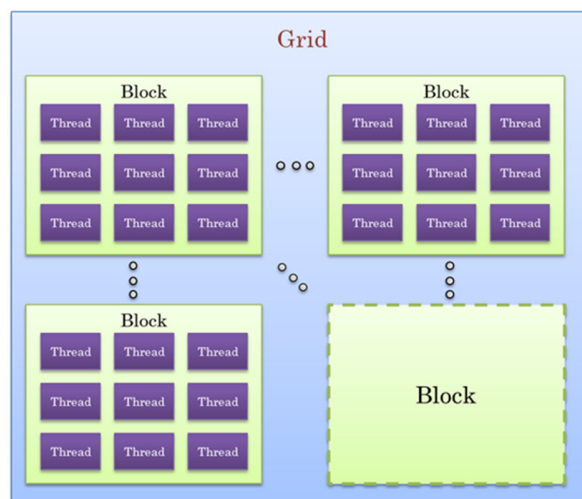


Fig. 5.3 Programming Model. Grid, Blocks, Threads hierarchy representation

Each thread is dedicated to compute the program submitted to the GPU and as stated in the previous section it has certain restrictions to access the memories. The part of the program that is designed to run in the GPU is called a *kernel*, and it has some specific characteristics that are discussed next. The qualifier `__global__` is used to declare a kernel and it tells the compiler that this part is to be run in the GPU, also a kernel with this qualifier can be called from the host only and executed in the device exclusively. For instance a pseudo code in C language for a kernel would look like:

```
__global__ void my_kernel(void)
{
    //Procedures
}
```

Additionally in order to launch a kernel in the GPU it should include sizes for the Grid, Blocks and Threads as well as in which stream (if using the asynchronous model) to be launched with the following syntax:

```
my_kernel <<< Grid, Blocks, Threads, Streams >>> ( A,B,C,D,... )
```

where *A,B,C,D* are any sample parameters to pass to *my_kernel*.

The dimensions that the grids, blocks and thread can have depends on the particular card however they can be arranged in one, two or three dimensional arrays. For this end CUDA provides the *gridDim* built in variable to define the dimensions.

5.2.3 Data Handling

Since the GPU and CPU cannot read directly from their memory, data used in the calculation must be moved to and from the CPU to the GPU during the calculation. CUDA

includes in its APIs particular functions to perform this task in the most efficient way possible. Thanks to the continuous development in hardware the data transfer bandwidth between CPU and GPU has been widened up. The function to handle these transfers is called *cudaMemcpy* and its syntax is:

```
cudaMemcpy(void* dst, const void* src, size_t count, enum cudaMemcpyKind kind);
```

where *dst* is the pointer where the data will be copied, *src* is the pointer where the data is located, *count* is the size in bytes of the memory and *kind* is one of the kinds of transfers available: Host to Device, Device to Device or Device to Host.

One useful element to improve the performance in a program is the asynchronous copy model. The previous copy function blocks the system until the transfer has been completed. However the asynchronous analogue returns the control immediately and allows to keep running the program.

5.2.4 Compilation for GPU Computing

CUDA provides its own compilation tool called NVCC. This compiler basically separates the host from the device code, generates a C object output to be compiled with another tool. Nvcc supports C language syntax and can be easily used with the gcc compiler to produce an executable file. Nvcc also provides its own specific set of flags for optimization during compilation, in some circumstances they can prove a useful way to increase performance. We utilize version 7.5 of nvcc in this work.

5.3 SSWE GPU Kernels

As mentioned before a kernel is the style CUDA provides to define C functions that get executed on GPU in parallel. The number of copies a kernel is executed depends on the block and thread partition defined in the call parameters. Furthermore, the clear analogy between CUDA blocks and mesh blocks provided a guide when deciding on how to organize the grid program for GPU execution in this study.

The SSWE are computed exclusively on GPU by processing the mesh blocks created during the domain refinement process. The two-dimensional mesh blocks have a size of $(length, height) = (65+4, 65+4)$, where the 4 corresponds to the total size of the halo. On the other hand, CUDA threads can be organized in any three-dimensional block configuration as needed by the problem. Since the GPU process in warps of 32 threads, using multiples of this number is desirable to avoid performance penalties. When possible, kernel configurations follow this guideline and any overflowed areas are treated as separate special cases.

Thusly, in order to process a single mesh block, first (Fig. 5.4 Top) CUDA threads are organized in two dimensional blocks of size: $(64, 4)$. Since the 64 threads in the x dimension cover the length of a mesh block, only one CUDA block is needed. For the y dimension, 16 blocks are requested for a total of $16 \times 4 = 64$ threads, thus covering the height of the mesh block (Fig. 5.4 Bottom left). Finally, in order to process all the mesh blocks, this two-dimensional CUDA block configuration is extended along the z -direction as many times as mesh blocks exist. The bottom right diagram in Fig. 5.4 shows this setting by simplifying the CUDA blocks as a single structure.

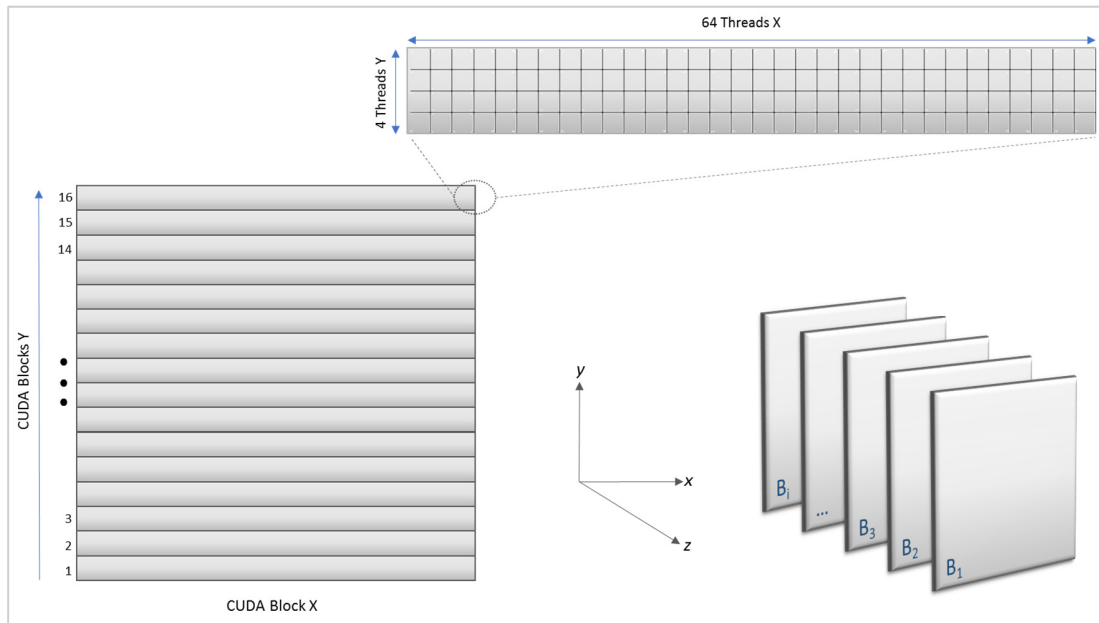


Fig. 5.4 CUDA blocks and threads diagram for the SSWE Kernel. Top: threads configuration per block; Bottom left: X and Y block configuration; Bottom right: Z block configuration

As mentioned above, any overflow cells are computed as a special case. Mesh blocks are squares of 65 cell-rows and 65 cell-columns while the CUDA block configuration covers a total area of 64×64 , hence a line is missed in each dimension during computation. In order to process these two lines a second kernel is launched with two blocks of 65 threads each. By fixing the indexes appropriately inside the kernel, one block completes the 65th row while the other completes the 65th column simultaneously. Although not ideal, this operation does not represent any noticeable performance penalty due to the small amount of data processing.

In the case of SWE kernel the implementation is based on the work by [72]. The grid chosen for this kernel is different than that of the kernel for SSWE; a grid is chosen with blocks of 16×16 threads covering the domain, the excess of threads at the edges is skipped by introducing a conditional *if* and computing threads with indices less than the grid number of the block.

5.4 GPU Halo swap

After each time-step the halo region of each mesh block is updated with the latest values from neighbor blocks. As explained in section 4.2 this represents three different kinds of swap: copying, coarsening or interpolating. A mesh block has four edges and each one can have a different type of halo. For this reason, initially three different lists are created which group by halo type the edges of all blocks. With this information three different kernels are configured to perform the updates:

- A) Copying: the thread configuration is straightforward since the only operation involved is copying data between two blocks with the same resolution. Hence CUDA blocks contain two lines of 64 threads each. The total number of CUDA blocks is equal to the number of edges in the copying list. Similarly as explained in the previous section, one cell is missed during the computation. However in this case, as opposed to launching a second kernel, the last thread of each line computes its own value and the missed cell. A second kernel would result in an unnecessary overhead since the copying kernel is considerably much lighter than the SSWE one.

- B) Coarsening: in this case again the number of CUDA blocks is equal to the number of coarsening edges in its list. As opposed to copying, this halo swap includes a series of operations to obtain the average on each cell before updating. However giving different instructions to a group of threads create divergent executions and may cause performance degradation. A conditional *if* inside a kernel creates a divergence in execution since threads cannot be processed in a single warp. In order to avoid this, instead of mapping one thread to one mesh cell and including several *if* conditionals, each CUDA block consist of a single 32-thread line, half the size of a mesh block.

With this setting each thread computes four different operations, reducing the conditional to a minimum while computing the average and update efficiently.

- C) Interpolating: as with the other two cases, the number of blocks is the same as the number of edges stored in its list. The process to interpolate requires several computations that differ on cells thus creating divergent paths. Similarly as with coarsening, the thread number in a CUDA block is half of that of a mesh block, 32, lied in a single line. Using the same reasoning than with coarsening, each thread carries more operations instead of declaring many threads and divert them with *if* conditionals.

5.5 Kernel Types

By analyzing the domain's bathymetry it is easy to notice that some mesh blocks contain only wet points while others are a combination of dry and wet points. This idea is used to create two kernels for solving the SSWE. One kernel, named *Wet*, is used to compute the wave free propagation on wet-only blocks. The other one, named *Dry*, is used to compute the wave propagation and coastline reflections in wet-dry mixed blocks. Hence the main difference between them is the additional treatment of wall boundaries at coastlines in the Dry kernel. Also, as previously mentioned, blocks with dry points inside focal areas represent a special case and are processed as inundation by using the run-up computation described in section 3.5.

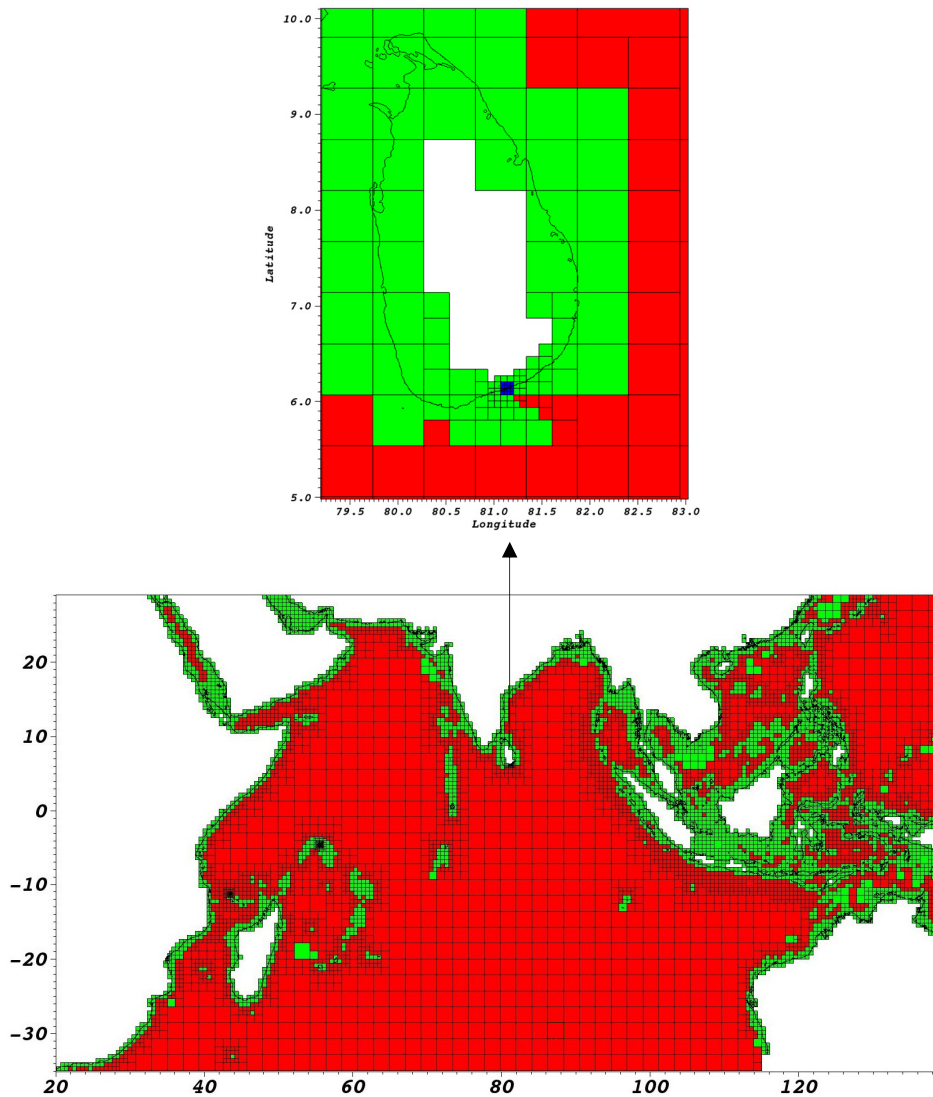


Fig. 5.5 Mesh blocks colored by kernel type. Red: Wet; Green: Wall; Blue: Inundation. Top: zoom over Sri Lanka FA

Therefore, TRITON-G is composed of three main kernels: *Wet*, *Wall* and *Inundation*. At mesh generation each block gets assigned a kernel type based on its bathymetry. This is illustrated in Fig. 5.5 where blocks flagged as *Wet* are shaded in red, *Dry* blocks in green and *Inundation* blocks in blue. As expected *Dry* kernel blocks tend to extend over coastlines while

Wet kernel blocks are spread out in the open ocean. When inside a focal area dry-type blocks at level 7 are re-flagged as Inundation type. This can be seen in the right image in Fig. 5.5 for the Sri Lanka FA with *inundation* blocks in blue. The total number of block per type is shown in Fig. 5.6.

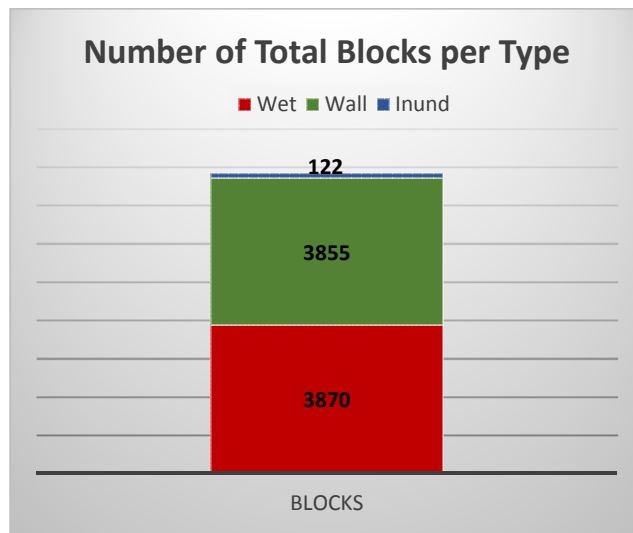


Fig. 5.6 Number of blocks per type

Whereas a single kernel would be too complicated and inefficient to compute the whole domain, splitting down the computation in three main kernels allows for better block handling and performance. Using specialized kernels for each case not only provide a simpler way to process the blocks through lists but also the ability to fine tune independently for higher performance.

5.6 Multi-GPU

With the GPU kernels at hand the next step was to implement the computation for Multi-GPU. As explained at the beginning of the chapter GPU memory cannot be directly accessed between different cards, hence a bridge to communicate is created through the CPU. Thus for this purpose CUDA and the message passing library Open MPI, are used to parallelize the code into multi-GPU. Each card processes a portion of the domain and in turn pass down to the CPU host any necessary halo information to exchange with other cards. A challenge that arises when using parallel computation is to guarantee that all processes are performing the same amount of work. This load balance concern is also true for multi-GPU computing and the way to determine that all GPUs receive the same amount of work is by applying a balance domain partition. This partition is rather straightforward for uniform mesh however for refined mesh a special technique has to be used. This technique is the use of a space filling curve (SFC). This method is explained in the following section and how it was implemented to our study case.

5.6.1 Domain Partition: Space Filling Curve

A correct domain partition is vital to obtain a balanced work load. In order to find this procedure a space filling curve (SFC) is used. SFC provides a continuous mapping from one-dimensional to two dimensions. Thus it is possible to linearize this space and use this information as a guide to partition the domain. SFC are constructed in a way that the curve visits each point exactly once. When refinement happens the SFC is modified to visit each of the points of the children instead of the parent. Thus locality is preserved; points closer in the SFC are typically close on the domain too. This characteristic of the SFC is of high interest for parallel computing since the locality means no need for extensive communications between processes.

There are several space filling curves available, for this work the Hilbert SFC is used [89]. It uses the rotations and inversions to keep points closer to their neighbors. Although block connectivity is kept using a quadtree structure, this does not provide information for domain partition. In order to track the order of the blocks the SFC is used.

i	Ordering				Orientation			
0	0	1	3	2	1	0	0	2
1	0	2	3	1	0	1	1	3
2	3	1	0	2	3	2	2	0
3	3	2	0	1	2	3	3	1




Table 5.1 Ordering and orientation tables for the Hilbert SFC in two dimensions

The template for the Hilbert ordering is shown on the left of *Fig. 5.7*, green values represent the orientation and numbers in black the ordering of the block. Each block has an assigned initial rotation, for instance block 0 has an orientation 1; by using Table 5.1 the orientation of children can be determined like this: using the value of the orientation as index i to look in the table, the resulting children will have orientations: $\{0,1,1,3\}$.

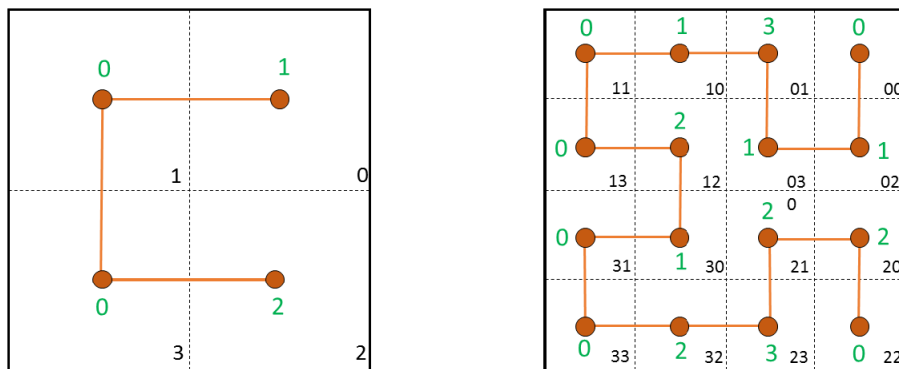


Fig. 5.7 Application of the Hilbert orientation tables to obtain the SFC after refinement

Similarly for block 1 with orientation 0 it is possible to determine the child orientation following the same method: using the orientation value as the index i to find the resulting orientation. Hence for orientation 0 the resulting children ordering is: $\{1,0,0,2\}$. As it can be seen, the original four blocks were refined and the Hilbert SFC covers all the children in a continuous line that preserves locality.

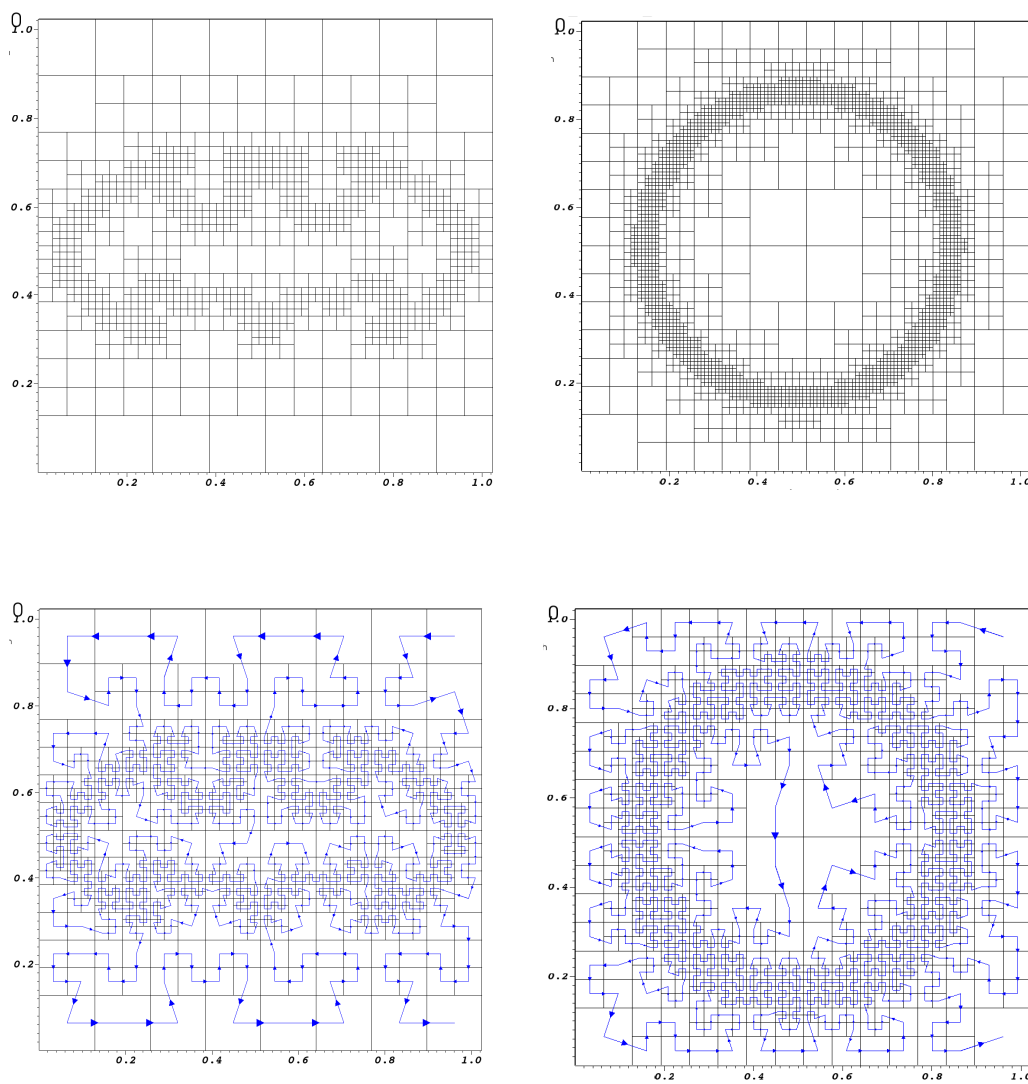


Fig. 5.8 Hilbert Space Filling Curve tests on domains with different geometry, refined levels and large number of blocks. It can be seen that the SFC (line in blue) traces all blocks exactly once.

By using this information it is possible to create a list that stores the order of the blocks in the domain. This one dimensional list has the advantage that show neighboring blocks close, hence there are no abrupt cuts when a partition is introduced.

Additionally the process of creating this curve is not computationally demanding and requires minimum memory resources. Thus, SFC proves to be the perfect solution to find the domain partition by simple splitting the one-dimensional list in equal parts.

Several tests were performed at the beginning of this research to test the best implementation of this method and to confirm that blocks are stored with locality. *Fig. 5.8* shows two of these tests with different geometries and several refinement levels.

Once the method of the SFC was correctly implement and showed excellent results in tests, it was applied to the Indian Ocean domain (*Fig. 5.9*).

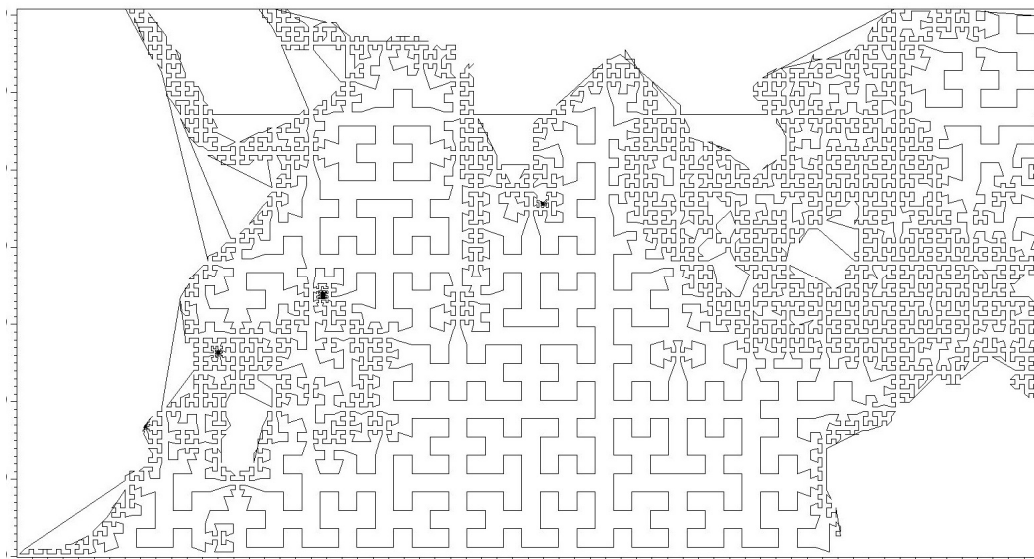


Fig. 5.9 Hilbert Space Filling Curve for Indian Ocean Domain

With the SFC as a guide, the domain is partitioned in portions that represent similar load for each GPU. *Fig. 5.10* shows the result for three GPUs, each portion is shaded in a different color.

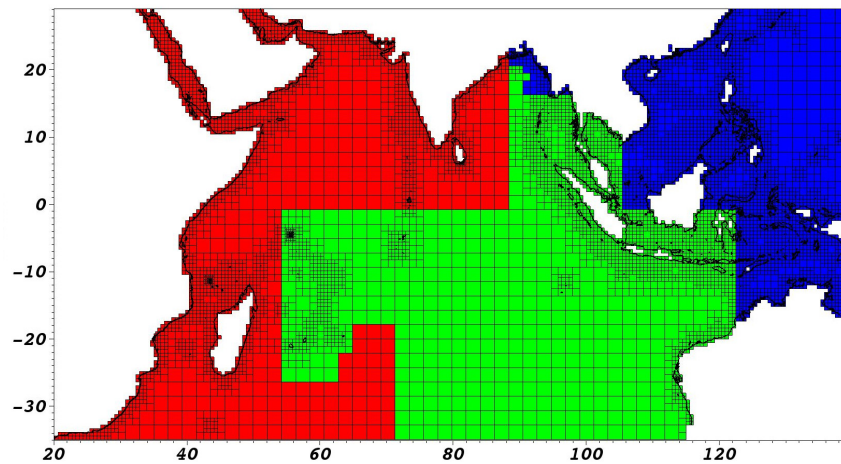


Fig. 5.10 Indian Ocean Domain Load Balance on 3 GPUs, each GPU represented by a different color

In general the total number of blocks get divided the number of GPU available and the resulting partition is never more than one block different.

5.6.2 Communication and Buffers

As mentioned above, a GPU cannot access a different card's memory directly. Hence in order to exchange data to update the halo a buffer is used. This process consists of the following steps:

- a) Prepare the buffer on the source GPU
- b) Transfer it from device (GPU) to host (CPU)
- c) Exchange them using MPI
- d) Upload to destiny GPU
- e) Read it to update the halo.

The preparation of the buffer represents a considerable challenge since the domain is not a uniform mesh. Not only the domain is composed of a large number of block, each with its own halo region but also the possibility that each edge transfers data to different GPUs make it a difficult task. The two main goals are:

- 1) Create a way to handle the data communication structure
- 2) Produce buffers that do not represent a large communication overhead

The traditional way to handle the first goal in tree-based refinement is with a look-up table that contains the list of adjacent edges, the type of halo and information needed to update it. Also it includes information about the source of the data and destiny. It is easy to imagine that maintaining this tables require additional memory and a careful track of the block connectivity. On top of this, there is the challenge of exchange all the required edges to the appropriate process. Sending multiple small messages down the network is known to be an inefficient way to communicate. Considering that large numbers of blocks might require to exchange data, the communication could become a large overhead.

In order to handle these two issues a different approach was taken. We inspired the solution based on cellular data communication. By implementing a similar design as the *user datagram protocol* (UDP) it is possible to eliminate the need for the existence of any look-up tables while at the same time making the buffer exchange smooth and simplified. UDP ([90], [91]) is a current standard way for query-and-response transaction in cellular networks; it has the ability to exchange different kinds of messages in a single and simple buffer. As depicted in *Fig. 5.11* a message or in our case a buffer can be composed of a series of messages separated by an identifying header in front of each. With a small 3-byte header the halo data can be transferred correctly to its destiny. The header includes values for: destiny, which represent the block it should go to; the edge, which represent which of the four edges in that block is the information for; size, which represent the total data size sent for that halo edge. By embedding this simple information in a header the necessity of a look-up table is removed completely. Also it gives an additional freedom of writing the data in any order as

long as the correct header proceeds it. Moreover in order to make it ever more compact, all the information needed to update the three variables h , hu and hv is stored in a single buffer continuously as opposed to having three different buffers.

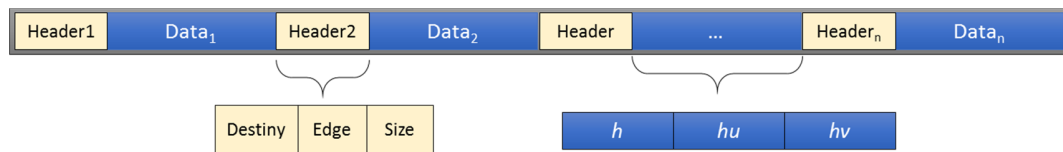


Fig. 5.11 Buffer packaging based on UDP structure

The second goal is partially covered by the inclusion of all the header and data in a single buffer. An improvement to this consist of collecting and packing all the edges that are required to be transferred in a single buffer. This concept is illustrated in Fig. 5.12 , the different edges (represented in different shades of blue) get all collected and stored in one single buffer following the UDP format.

Halo Type	Destiny	
	Length	Number Lines
Refine	1/2	3
Copy	1	2
Coarse	1	5

Source	Destiny		
	R0	R1	R2
R0	-	146	5
R1	105	-	102
R2	5	110	-

Table 5.2 Left: Number of lines and length sent by boundary type. Right: Amount of data exchanged between processors in the Indian Ocean case (in kB).

The total number of lines sent by each edge depends on the type of boundary it is, Table 5.2 show the values for each type; length represent the one dimensional grid size of a block. Table 5.2 also show the total amount of data exchanged between processors (noted as R for

rank), as it can be seen the total amount is considerable small. This is part possible due to the efficient domain partition that allowed to preserved locality as much as possible.

In this way we manage to drastically reduce the communications by firstly removing any need to create and store look-up tables and more importantly by creating single buffers to exchange between processors smooth and efficiently. Instead of requiring hundreds of small communications exchanging each block edge, only a single transfer between processors is needed without any noticeable additional memory used.

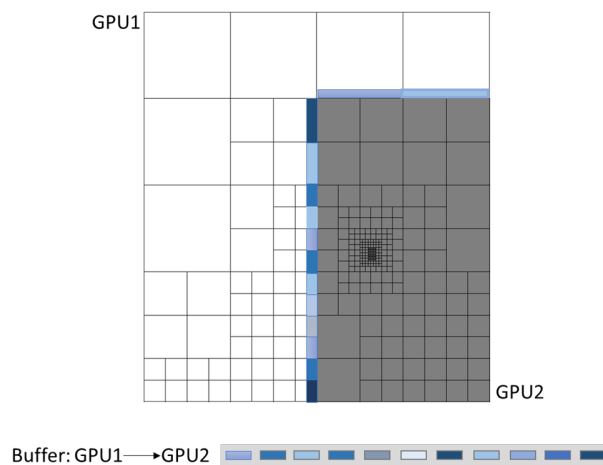


Fig. 5.12 GPU buffer. Data collected and packed for a single communication

The step of downloading and uploading the buffer from GPU to CPU and vice-versa is done through the CUDA API for memory copy. Also, this process can be implemented as an asynchronous operation. Hence while the computation continues, the transfer occurs simultaneously providing a smaller communication overhead.

The buffers created on GPU are transferred to host memory and exchanged with the appropriate processor using MPI in a traditional way. Since a single server is used in this study the communication speed does not depend on an external network but only on the machine PCI performance.

Once a buffer is uploaded into the destiny GPU memory it is read to compute the halo on the blocks that require it. The unpacking of the buffer becomes also a simple and smooth task by using UDP. The kernel simply read the header and readily process the information to update the halo just as it would with local memory. This gives the advantage of not having an additional overhead unpacking the buffer for each block but instead use it directly to update.

5.7 TRITON-G Output

TRITON-G handles three different kind of output: variables, gauges and images. In the case of variables these are stored just for FA inundation blocks. For images, two sets are generated, one for the whole domain and another for individual images of each FA; the frequency of this output is a parameter set by the user but as default the whole domain image is generated every 4 minutes and FA images every 5 seconds. Clearly generating output with such frequency is a large overhead; in order to overcome this issue special optimizations were introduced that will be explained in the following section.

The third type represent virtual gauges that the user can set in any location in the point and its purpose is to store the wave height in that point during the simulation; the frequency to do this is set by the user.

5.7.1 Type of Output

The variables are stored on GPU memory and represent values of interest for the user. Specifically the following values are store on each time step:

- i. Maximum inundation
- ii. Maximum wave height
- iii. Maximum wave velocity
- iv. Maximum Flux
- v. Arrival time

Again, this storage happens on GPU memory and the function to check the maximum value is computed on GPU. Since these values are computed just in inundation blocks, the computational time on GPU is almost neglectable, after a breakdown of the computation it was found that it represents less than 1% of the total running time.

The output variables are flushed down from GPU to CPU when needed and written as a simple ASCII files that user can process easily in any way they see fit. *Fig. 5.13* show the flow of the computation and the part that output takes during simulation.

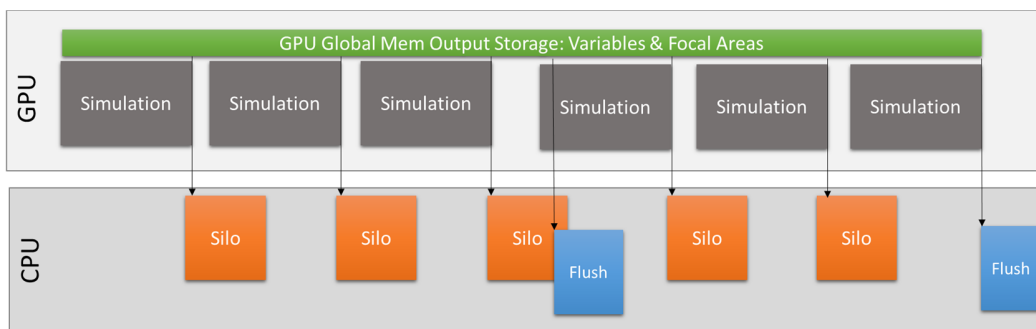


Fig. 5.13 TRITON-G computational flow

In the case of images, TRITON-G uses vISIT to generate the rendering. This is possible by writing SILO format [92] files with the mesh information. SILO is an open source utility that sits on the HDF5 format; this provides high efficiency to store large problems and particularly multi-level blocks. It naturally handles the connectivity between blocks to

generate a full composed image. In this way the user can visually follow the results produced by the simulation.

5.7.2 Optimizations

In order to obtain a smooth animation of the wave propagation in the focal areas output frequency must be considerable high, around every 5 seconds. Clearly, even for small data this frequent output represents an unnecessary overhead. The optimization to this issue came from postponing the image generation for the focal areas. Since users can track the wave propagation in the main domain it was slightly redundant to generate at the same time images for the FA. Hence to remove the overhead of the FA image output all values are stored on GPU memory and collected on the available free memory. Tesla K40C cards have 12GB of memory. TRITON-G requires around 2-3GB per GPU to keep the block information and maintain the code; thus there is considerable space available for storage. In case that the memory got full, the complete values are flushed down to CPU where are kept on RAM memory until the simulation finishes while GPU is reused. By using this idea, the total overhead of the FA image output was removed.

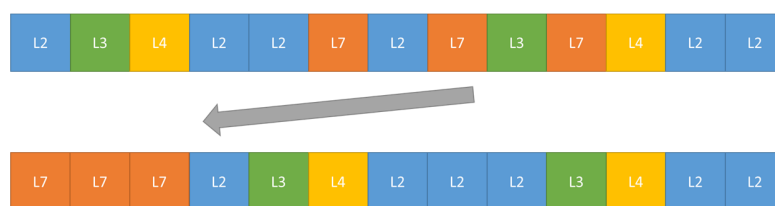


Fig. 5.14 Optimization by grouping the output blocks (L7) together

An additional idea used to speed up the transfers was to allocate all the blocks that require output together in memory, by reallocating them at the front of the memory array;

this is shown in *Fig. 5.14*. By doing this, it is possible to perform one single copy instead of doing several small ones which introduce a latency overhead.

The last optimization was a great improvement for the overall runtime. Even though the image generation for the full domain is not very frequent, the process of generating a SILO file for such a big mesh represented a considerable overhead. This process represented around 15-20% of the total runtime. In order to improve this, an ingenious solution was implemented. In general, the usual approach to remove output overhead consist of using asynchronous computation. *Fig. 5.15* shows in this idea in the diagram in the middle. This approach generate good results if the computing time is larger than the output time. However if the opposite happens we go back to the original issue of output overhead. In the case of TRITON-G the image output and generation took much longer time than the simulating kernels hence this approach would have proven not very efficient.

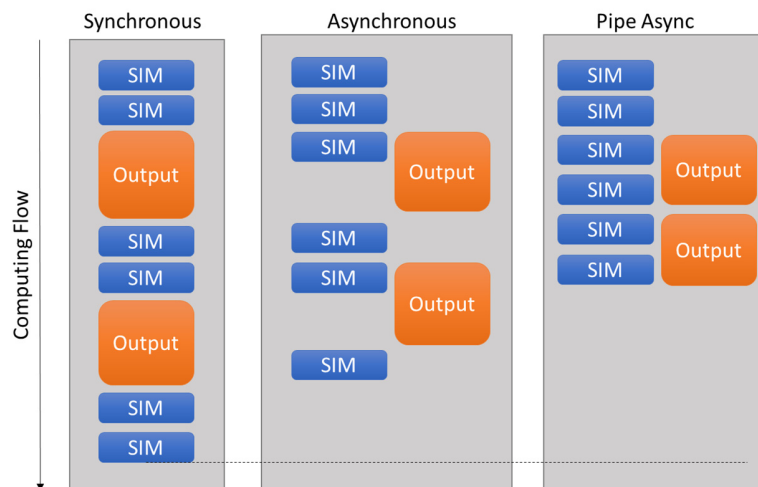


Fig. 5.15 Output overlap and optimization using Pipes

In order to minimize the effect of this image output we took advantage of Pipes. Pipe is a system call that creates a communication between two processes that run independently.

Thus, a parent program can call a child program and both perform completely different tasks. By using this concept, an utility to process to create SILO files for the full domain was created a stand-alone application. Hence now TRITON-G can call a subprogram that independently computes the output. However, this method has a limit, a memory array cannot be pass between parent-child processes. If the child cannot read the values of the parent it would be impossible to generate the output, nonetheless a clever solution was implemented to solve this issue as well: CPU shared memory. By using the available shared memory to store the arrays needed to generate the output it was possible for the child process to read the information and process the output. This process is illustrated in *Fig. 5.16*.

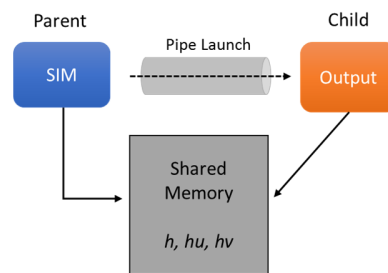


Fig. 5.16 Concept of the Pipe Asynchronous processing by using shared memory

FIG shows the advantage of implementing Pipe asynchronous output, unlike traditional asynchronous output that relays on a large computational time, the Pipe method provides the ability to hide the output processing behind several computing time-steps. The result of this is an almost total elimination of the overhead. Measurements before and after this optimization showed that the output process represented just 1-2% of the total time, practically removing totally this overhead.

5.7.3 Post-processing

During development TRITON-G developed not just into a single tsunami modeling program but in a full operational framework. *Fig. 5.17* show all the utilities that are part of TRITON-G, specific utilizes for rendering processing were developed along with the simulation model; also a tailored excel utility programmed in VBA was designed to process the gauges values and create automatic charts.

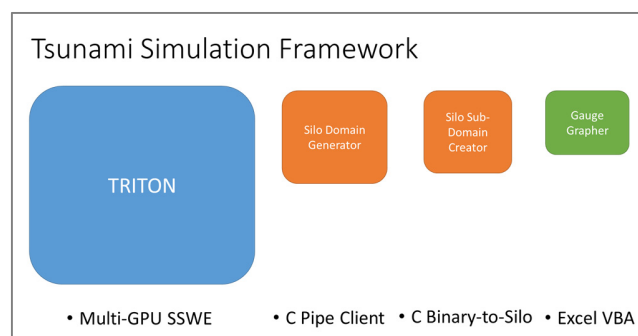


Fig. 5.17 TRITON-G Framework

As explained in the previous section, data to make the images for the FA is stored on GPU until the simulation completes. Once it has finished this data is copied to CPU and written as binary files. In a post-processing step, the Silo Sub-Domain utility converts these binary files into SILO files than in turn as used to generate the images using vISIT (*Fig. 5.18*).

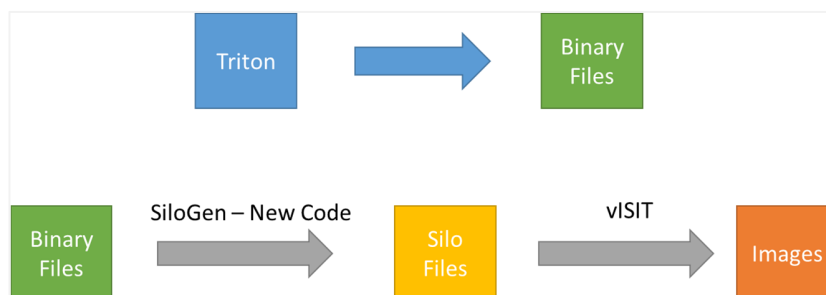


Fig. 5.18 FA Images generation process

5.8 TRITON-G Performance and Optimizations

For this study and under the collaboration project with RIMES a specific machine was assembled to use as the server to implement TRITON-G as their operational tsunami forecasting tool. Initially a machine with a single GPU Titan Blank was used for development to test the feasibility of the operational concept. Once promising results were obtained, a second machine intended for permanent installation was designed taken into account several considerations such as: budget, size, GPU performance, storage and power back up. The specification for this machine are detailed in *Table 5.3*.

RIMES Machine	Specifications
CPU	Intel Xeon E5-2620V3 2.4GHz
RAM Memory	DDR4-2133, 64 GB
GPU	nVIDIA TESLA K40C x 3 cards
HD	Seagate 3.5" 2TB SATA 6Gb/s 7.2KRPM
	SSD Intel DC S3500 480Gb SATA 6Gb/s
UPS	UPS APC 220VA
OS	CentOS 6.6
CUDA Version	7.5
Database	PostgreSQL 9.1
Web Server	Tomcat 6.1
MPI Version	Open MPI 1.8.6

Table 5.3 RIMES machine for TRITON-G

The best balance between all those requirements was met with those specifications. Nonetheless the aim to deliver a powerful system within a reasonable budget was never oversighted and this server proved it.

RIMES server is designed to be used as a web interface by the end user hence the mention to the webserver utilized. Through this interface users can adjust TRITON-G to their

specific needs. TRITON-G being an operational tool is far from a static single program but instead provide the options to be modified as requested. Such options include define focal areas, submit new bathymetry or manning databases, introduce the latest tsunami fault sources; as well as basic options like output frequency, total runtime and gauges location.

In the specifications apart from the three Tesla K40C GPU cards another characteristic stands out, the existence of two hard disks. The reasons behind installing two hard disks in the machine can be understood by looking at the type of disks they are. One disk is a traditional SATA HDD, this kind of disks have proven to be reliable, affordable and have grown in capacity, however their top writing speed is between 60-160MB/s. On the other hand Solid State disks (SSD) can deliver up to 600MB/s writing speed. The downside of SSD being the price and smaller capacity (240GB). In order to take advantage of these two devices they are configured to be used for different purposes. The SSD is assigned as the primary disk for output during computation. This allows for a fast I/O during simulation. The HDD is kept mainly for storage; once a simulation is finished, in a post-processing stage the results are transferred from SSD to HDD. In this way the SSD is kept mostly free and ready for real-time computation while the HDD with its largest capacity serves as a case storage device.

Hence our implementation and objectives are focus on the available three Tesla K40 cards [93].

TESLA K40C	
CUDA Capability Major/Minor version number	3.5
Total amount of global memory	11471 MBytes (12028608512 bytes)
(15) Multiprocessors, (192) CUDA Cores/MP	2880 CUDA Cores
GPU Max Clock rate	745 MHz (0.75 GHz)
Memory Clock rate	3004 Mhz
Memory Bus Width	384-bit
L2 Cache Size	1572864 bytes
Total amount of constant memory	65536 bytes
Total amount of shared memory per block	49152 bytes
Total number of registers available per block	65536
Warp size	32
Maximum number of threads per multiprocessor	2048
Maximum number of threads per block	1024
Max dimension size of a thread block (x,y,z)	(1024, 1024, 64)
Max dimension size of a grid size (x,y,z)	(2147483647, 65535, 65535)
Concurrent copy and kernel execution	Yes with 2 copy engine(s)
Run time limit on kernels	No
Support host page-locked memory mapping	Yes
Device has ECC support	Enabled

Table 5.4 Tesla K40C Main Specifications

5.8.1 Kernel Optimizations

Once TRITON-G reached a point where it showed stability and good agreement with benchmarks, several optimization were implemented in the SSWE kernels to obtain the best performance possible out of the available machine. These results are shown in *Fig. 5.19*.

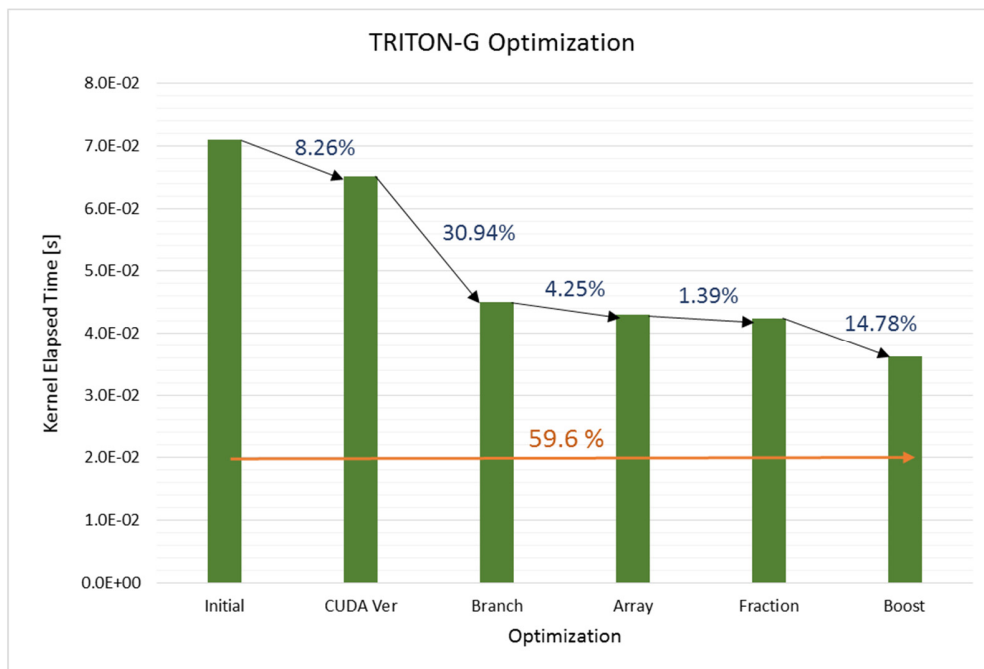


Fig. 5.19 TRITON-G Optimization, a total of 59.6% speed up was achieved

The first optimization utilized was the update of the CUDA toolkit and compiler version. TRITON-G was originally developed using version 6.5 of CUDA and to avoid changes on the machine OS configuration it did not get updated even though newer versions were release. Then, when the project reached the optimization part CUDA was updated to the, then latest version 7.5. By doing this an 8.26% speed up was achieved; this is due mainly to internal improvements to the compiler.

The second optimization is branch divergence. For GPU this event can represent a major performance penalty due to the way threads are computed. When a kernel includes a conditional that creates divergence the warp process all threads twice, deactivating the unused threads and putting the result back together at the end. Clearly the amount of work doubles which introduces a considerable overhead. Hence it is highly desirable to avoid any condition that creates threads to take different paths. Sometimes to achieve this the algorithms have to be rewritten, an example of this is the set of equations (3.27); re-written

the equations in this way creates a single path on the GPU which allows for higher efficiency. Another way used to decrease the divergence was to replace *if* conditionals for binary expressions that cancels the unnecessary part, e.g. the condition:

```
if a=1 then e=C
else e=D
```

Can be replaced by the expression:

$$e = \alpha C + \beta D$$

where α and β are expression that take the values 0 and 1 mutually exclusive and cancels out half of the values thus making the conditional unnecessary.

By introducing this optimization a 30.94% improvement was achieved, this was a big improvement and showed the damaging effect of branch divergence.

The next optimization was the elimination of unnecessary computing by storing results in a memory array. For example the trigonometrical expressions that appear in the model depend on the value of θ on each point, however this values does not change over time making it redundant to be computing the same values. Instead we compute the trigonometrical expressional once at the beginning, store them on GPU memory and simple reuse them each time step. Doing this reported a 4.25% improvement; the reason that performance increased in this case is due to the register count. As explained earlier the fastest memory available on the GPU are the registers, however they are scarce. Computing the trigonometrical expressions required loading values on registers but removing this computation registers were freed up and used for other tasks thus improving the performance.

The next optimization was the replacement of complicated exponential with built-in functions provided by CUDA. Especially the terms on the friction force (3.37) include exponentials of $7/3$, by using the built-in functions an improvement of 1.39% was found.

Lastly, another optimization that provided good results was the use of the Boost option (Appendix B). The Tesla K40C runs normally at a frequency of 745Hz, however this

clock can be changed to higher values to produce a boost on the speed. By setting the card to the highest available frequency of 845Hz we obtained a 14.78% improvement.

Overall we obtained a nearly 60% improvement by using all these optimization, effective cutting down the total running by around half of the time.

5.8.2 Sub-cycling

Encouraged by optimizing the code as much as possible, a sub-cycling technique was introduced in the simulation. Sub-cycling consists of using a large dt and making blocks cycle in smaller steps to reach the same dt . The number of cycles taken is called n . The advantage of this technique is that the global dt can be enlarged thus producing faster simulation, the potential disadvantage is that if the blocks sub-cycling are too many the whole computation is slow down. An illustration of how the sub-cycling works is shown in *Fig. 5.20*. As it can be seen there, blocks with the same number of sub-cycles can be group in a single list. In theory after a large dt is taken, appropriate boundary conditions should be computed by interpolation in time. This process usually introduce an overhead; since the objective of our simulation is to provide the fastest modeling tool for evacuation warning, it was decided to skip the boundary generation and use the values at time n . As it will be shown later, this decision proved to be correct judging by the excellent agreement of the simulation result with benchmark and real tsunami data comparison.

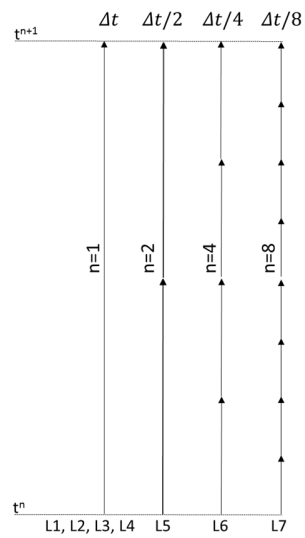


Fig. 5.20 Illustration of the sub-cycling process

The maximum dt values per level are shown in

Table 5.5, using CFL=0.8; the selection of 1.6 as global dt came from a balance between a larger dt and a small number of sub-cycles.

	h Max	dt Max	dt SC	n
L1	7816	1.07E+01	1.60E+00	1
L2	8524.75	5.13E+00	1.60E+00	1
L3	10006.75	2.37E+00	1.60E+00	1
L4	5143	1.65E+00	1.60E+00	1
L5	3902	9.47E-01	8.00E-01	2
L6	2944.51	5.45E-01	4.00E-01	4
L7	1297.51	2.57E-01	2.00E-01	8

Table 5.5 Maximum dt per level and the resulting sub-cycling number

Also as *Fig. 5.22* shows the largest number of block are located in level 3, this is not surprising considering that all coasts get refined up to level 3. It was our intention to avoid sub-cycling level 3 since it would produce a large overhead.

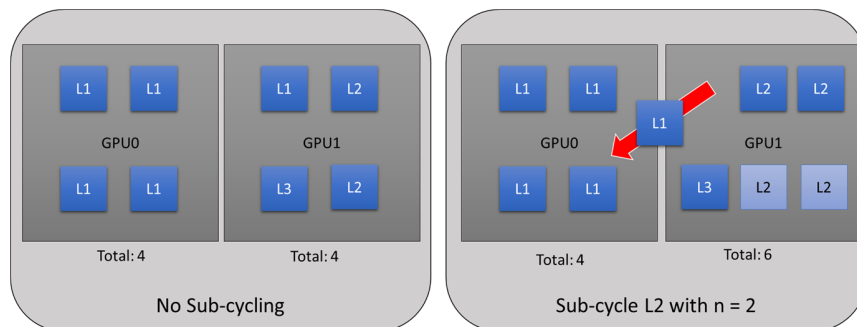


Fig. 5.21 Load Balance example due to the effect of sub-cycling

Finally, the introduction of sub-cycling produced an issue in the load balanced. As it can be seen in *Fig. 5.21* if a block needs sub-cycling that means that is equivalent to having several more blocks than those allocated in memory. In order to guarantee that the balance was still preserved a variation was introduced to the SFC. Instead of simply using the list originally created, a second weighted list is also created. This list gives more weight to blocks that need refinement hence when the partition is done the work load remains well balanced.

This effect on the increase of the number of blocks per level can be seen in *Fig. 5.22*, for example for level 7 the 100 original blocked represent 800 due to the $n=8$.

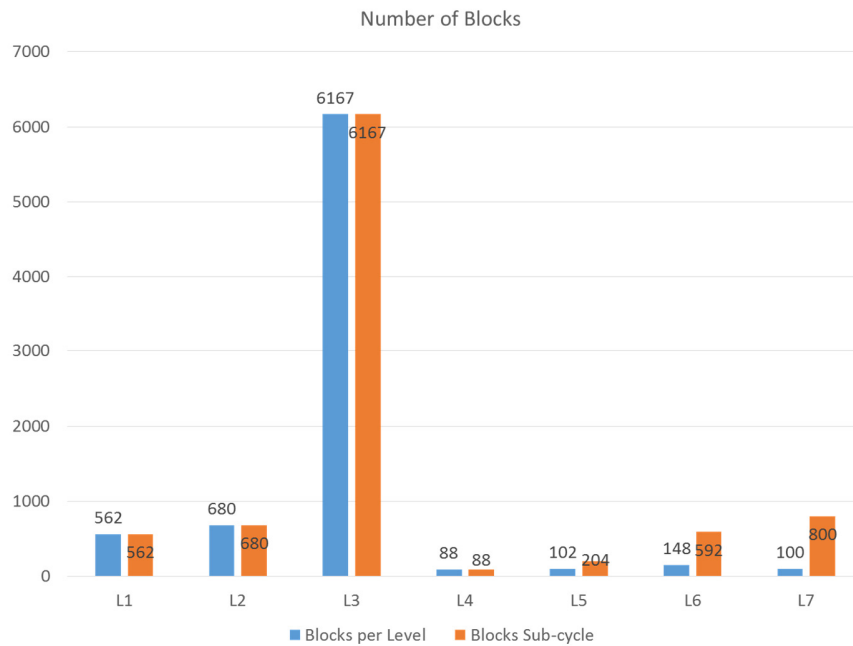


Fig. 5.22 Number of blocks per level before with (orange) an without (blue) sub-cycling

5.8.3 Performance Measurements

With a fully optimized code, output, kernel and load balance, measurements were made to estimate the performance of TRITON-G. Fig. 5.23 shows the breakdown of the main part of the computing. Wall stands for the wall kernel, Wet for the Wet kernel and X and Y for the dimension of the computation. The process of updating the halo, presented in the graph as *Bnd* for Interpolation, Coarsening and Copy represent only 9% of the total running time. It is also interesting to note that the Wet and Wall kernel have similar performance despite the fact that the wall include additional treatment for the coast boundaries. Since this treatment consists of many conditional and they were replaced on during optimization it is understandable that the performance is similar.

It is also interesting to notice that the inundation kernel represents around 21% of the total runtime even though it process only around 2% of the total number of blocks. Inside others

several values are included, most importantly the communications, it represents around 1-2% of the total running time. This confirms that the care taken to create efficient communications with packing buffers like in UDP format was the right decision and provide high performance.

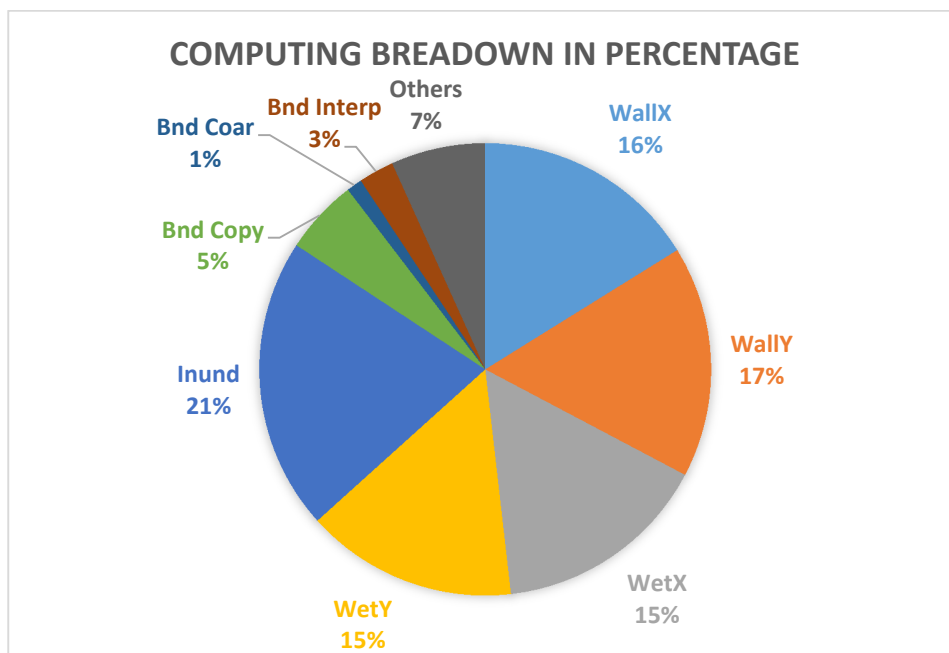


Fig. 5.23 Computing breakdown shown in percentage

Finally, results for the total runtime are presented. The simulation computes 10 hours on the Indian Ocean domain. Fig. 5.23 shows the total machine time consumed by TRITON-G to obtain this simulation. The first column represent the time before optimization were done, it was well above an hour. The next two columns show the results with the optimization and the boost option on and off, it can be seen that the total runtime with all optimization active is 40.40 minutes. This time includes all the image generation, gauges, variables' storage as well. It is an excellent result to produce such complicated scenario of 10 hours in just around 40 minutes machine time.

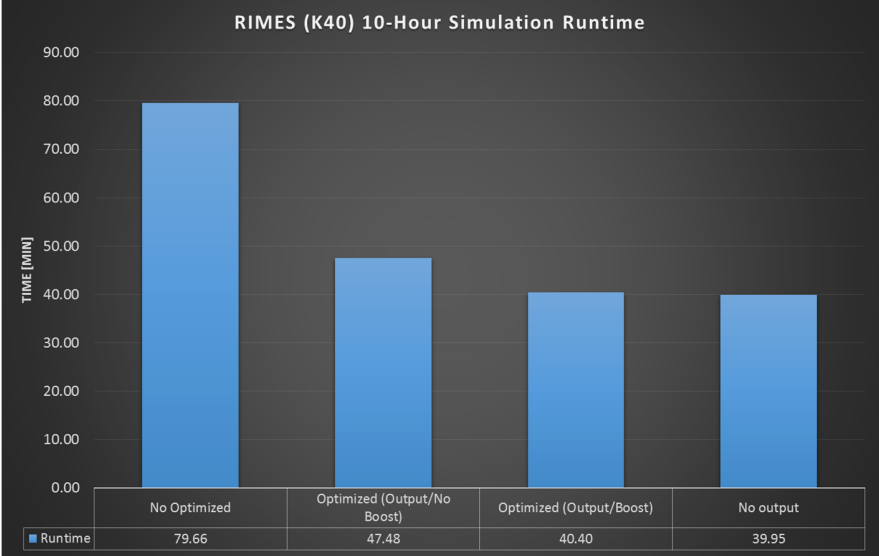


Fig. 5.24 A 10-hour Simulation Runtimes

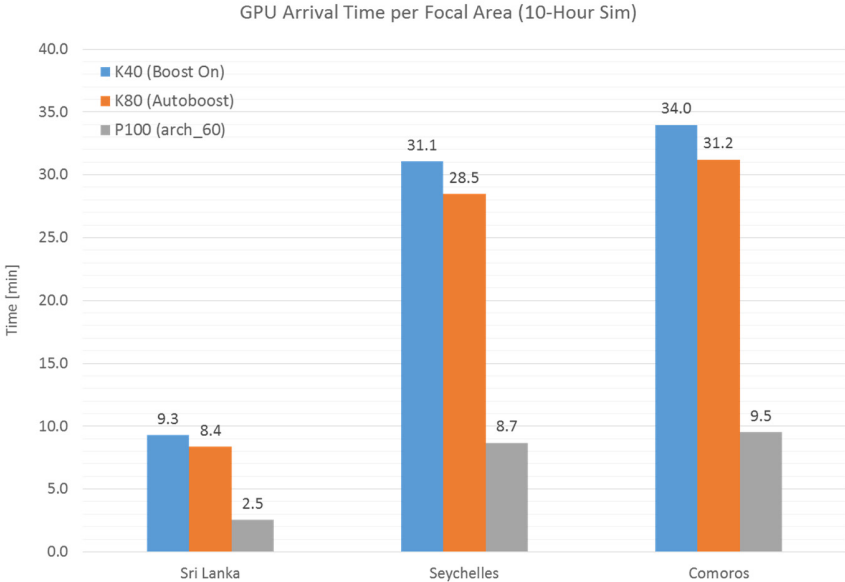


Fig. 5.25 Computing time required to obtain the first results during simulation

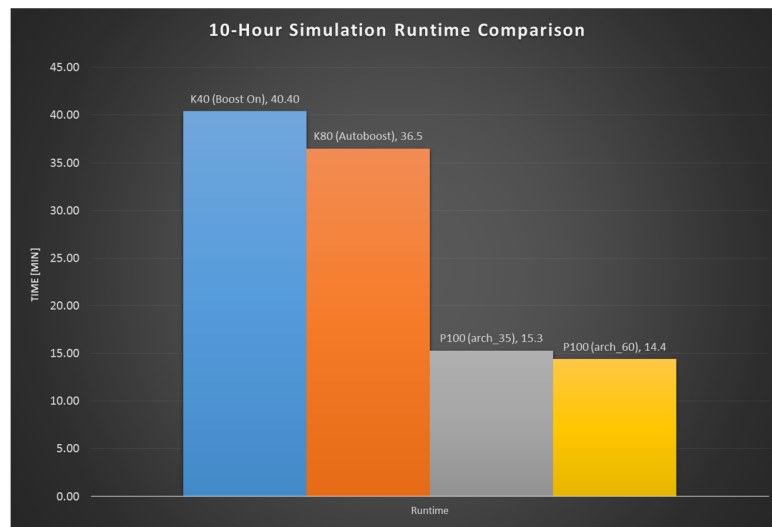


Fig. 5.26 A 10 hour simulation runtime comparison with 3 different GPUs

Furthermore, *Fig. 5.25* shows the machine time needed to compute the results of the first arrival wave in each focal area. For instance for Sri Lanka it takes just 9 minutes machine time to generate the results of the inundation in that area. Considering that the wave took around 2 hours from the source to Sri Lanka, obtaining results in 9 minute would give plenty of time to authorities to make a decision regarding evacuation.

A final result is presented to show how the program behaves under different GPU cards, *Fig. 5.26* show the results for Tesla cards model K40, K80 and P100. It is interesting to notice that despite the K80 card having double the number of register than K40 and a better automatic boost system we were able to achieve a very similar performance. In the case of P100 which contains more than 3000 cores and the latest technology results were obtained in just under 15 minutes for the same 10-hour simulation.

Chapter 6. Numerical Simulations: Application

Previous experience with GPU computing had inspired us to continuously exploring new and better ways to perform simulation, results for our previous work can be consulted in Appendix A. By taking GPGPU into more complex and large problems like the research that we present here TRITON-G was produced. It is the product of many years of constant development, looking always for better and more efficient ways to exploit GPU technology. Results of hindcasting the Indonesia 2004 tsunami are presented to show the comparison of propagation and inundation between TRITON-G and existing actual records of the event. The chapter finishes by presenting two inundation cases, one in Sri Lanka and the other in Phuket.

6.1 Application: Hindcast Indonesia Tsunami December 2004

Validation must be performed on different ways, not only on idealized situations but also on real cases with measured data. After obtaining promising results with the Gaussian idealized benchmark in the previous section, here a hindcast of the Indonesian 2004 Tsunami is presented. The ability of TRITON-G to model this event is demonstrated in this section by comparing its results with recorded tide gauges as well as with inundation maps.

There exist considerable information available about the Indonesian December 26th, 2004 massive earthquake and destructive tsunami [94]. Gauges recorded at the moment of this event are obtained from NOAA's tsunami events database. This event occurred at 7:58 am with a magnitude of 9.0 Mw generated by the subduction of the Indian Plate by the Burma plate; nearly 1600 km of fault was affected around 160km off the coast of Sumatra. This

massive earthquake generated a large tsunami that spread over the Indian Ocean in the following hours.

Firstly, the tsunami wave propagation computed by TRITON-G is depicted in *Fig. 6.1*; each snapshot represent an hour after the earthquake's main event. The parameters for the source were provided by RIMES and the result is shown in snapshot (a) of *Fig. 6.1*. The initial displacement represent height of 10 m and troughs of -5 m. In order to show the wave covering the totality of the Indian Ocean a slightly longer simulation was produced as opposed to the 10-hour test reported in the performance section. The results shown in the images represent 12 hours on real time and it took TRITON-G 51 minutes to complete the full simulation.

The images provide a visual record of the evolution of the tsunami wave train. Due to the initial location, several countries were rapidly affected by the arriving tsunami; Indonesia, Thailand and Malaysia received the first train of waves within minutes of the earthquake's main event. Also the positioning of the initial source made possible a direct hit towards Sri Lanka, India and later on the Maldives Islands. At around 5 hours, waves has spread reaching Australia and at around 6 hours after the earthquake the waves start to arrive in the north-east of Africa and then continue to spread covering all the domain. At 12 hours, it is still possible to see small oscillations all over the Indian Ocean still remaining from the original tsunami.

These synoptic maps also serve to show the magnitude of this event and the destructive characteristic of tsunamis waves with their ability to travel extremely large distances with enough energy to case damage.

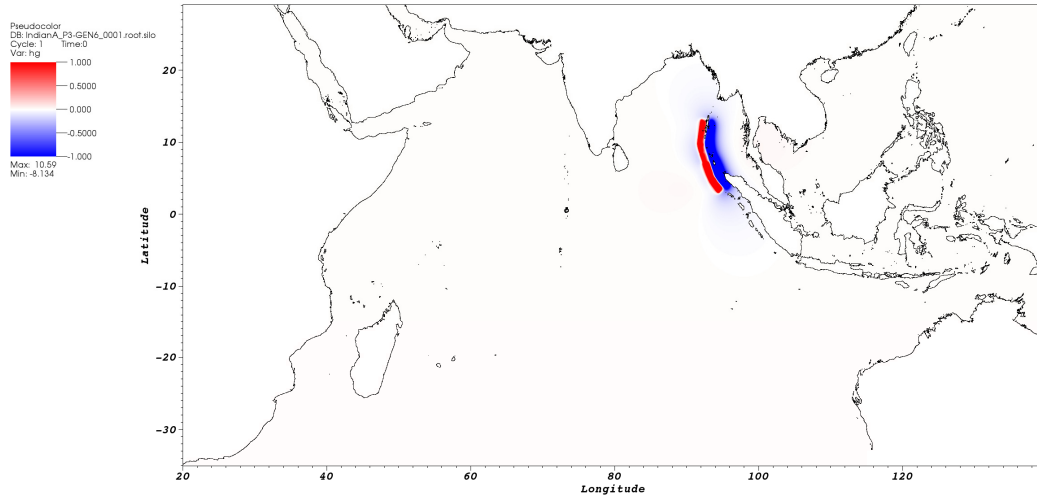


Fig. 6.1 (a) Time = 0 hours. Initial Source in Sumatra.

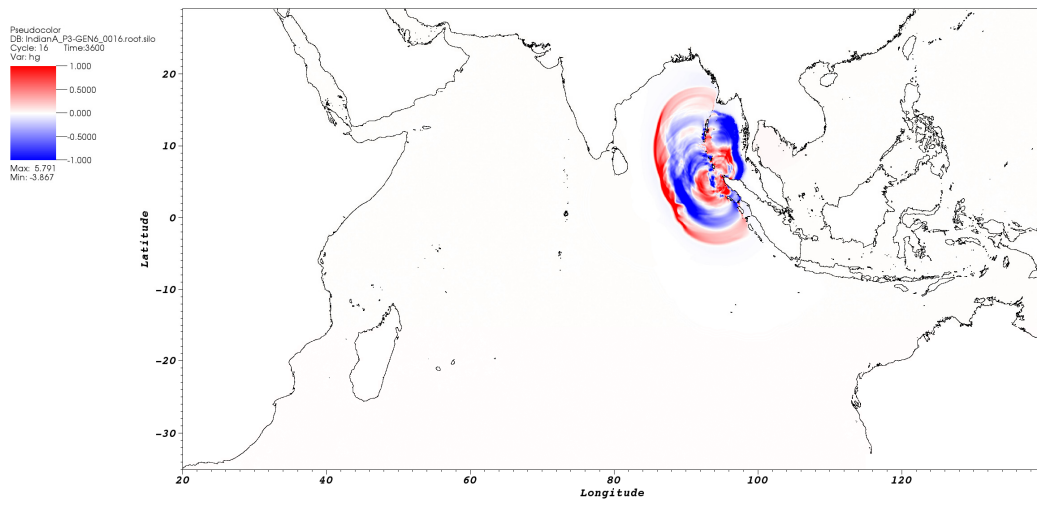


Fig. 6.1 (b) Time = 1 Hour

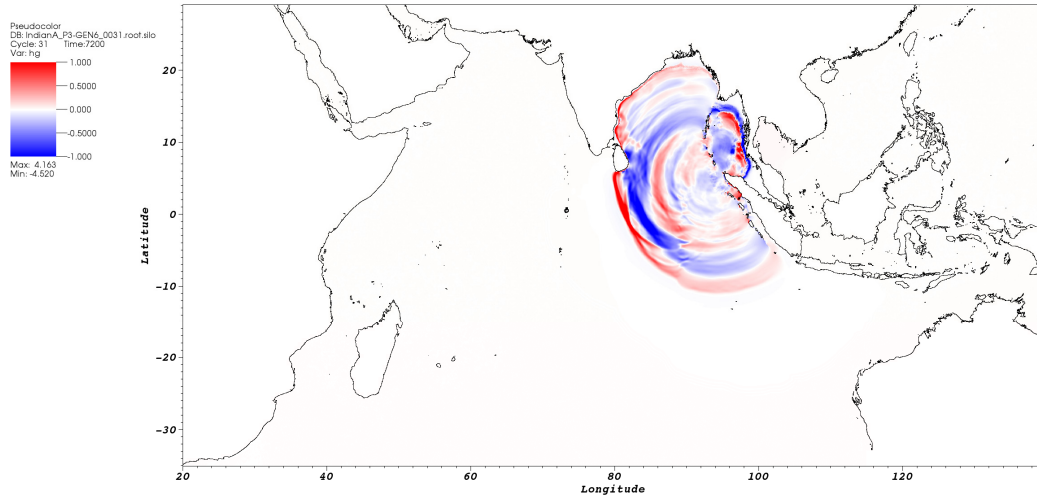


Fig. 6.1 (c) Time = 2 Hours

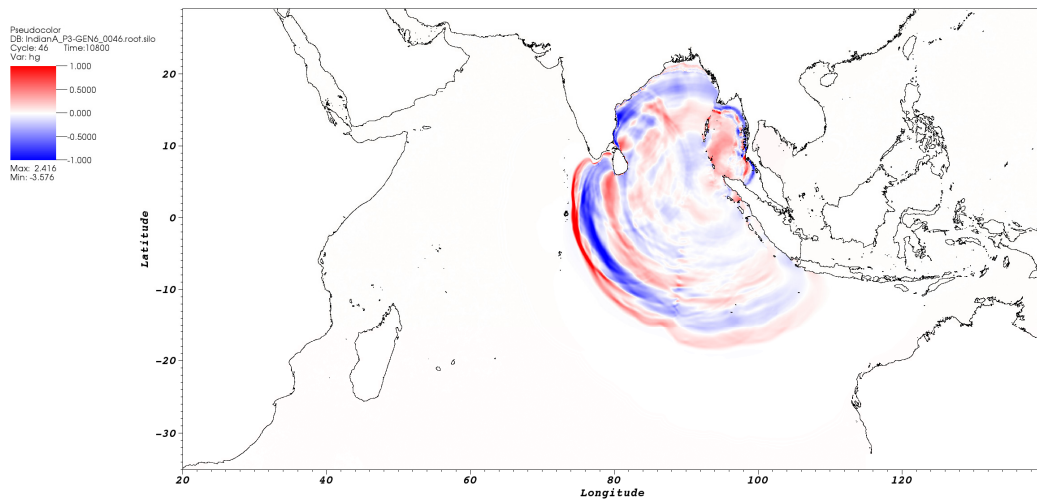


Fig. 6.1 (d) Time = 3 Hours

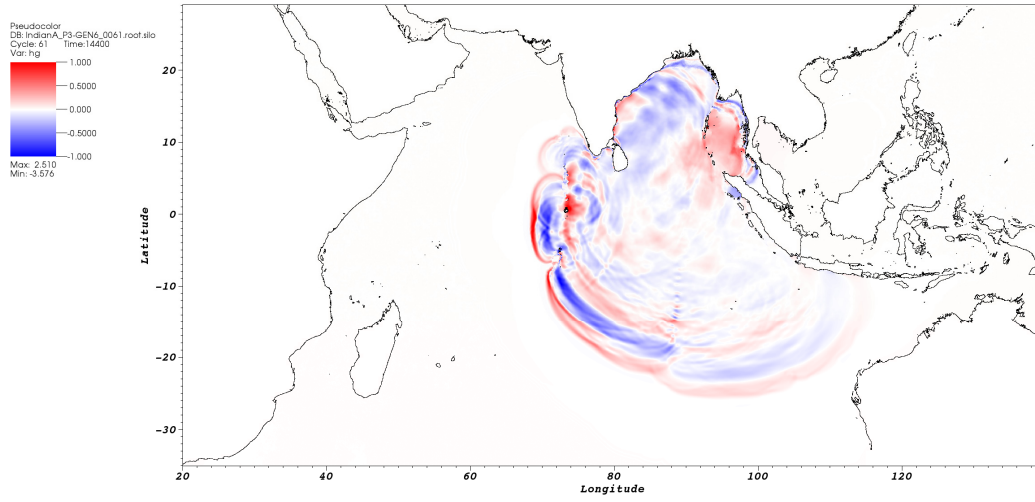


Fig. 6.1 (e) Time = 4 Hours

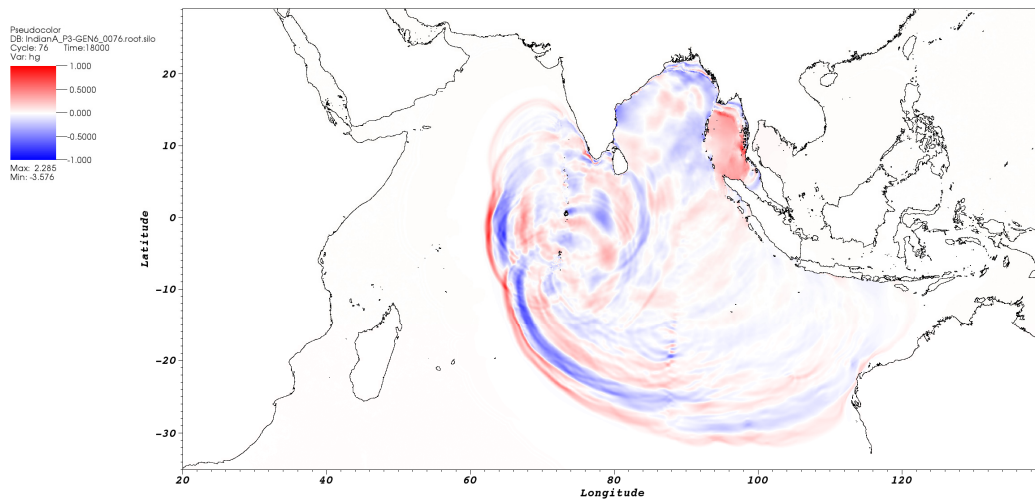


Fig. 6.1 (f) Time = 5 Hours

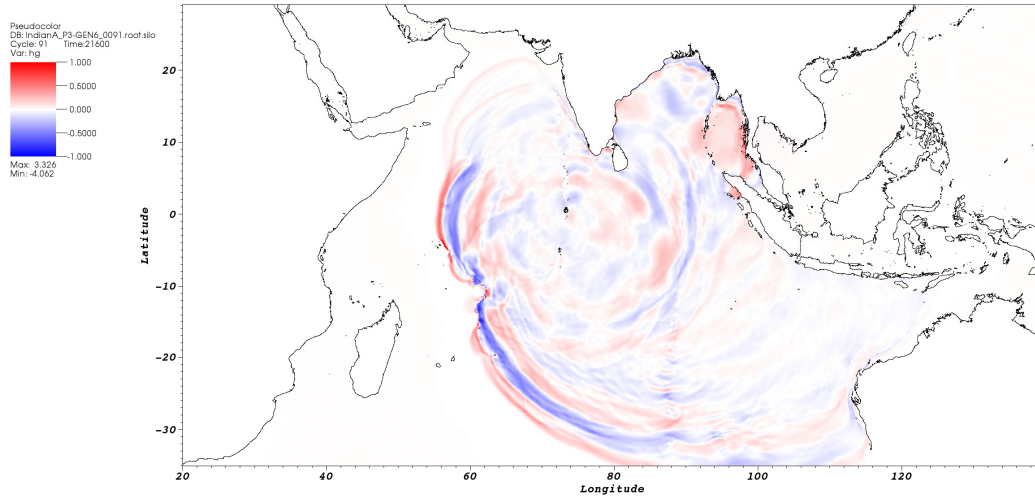


Fig. 6.1 (g) Time = 6 Hours

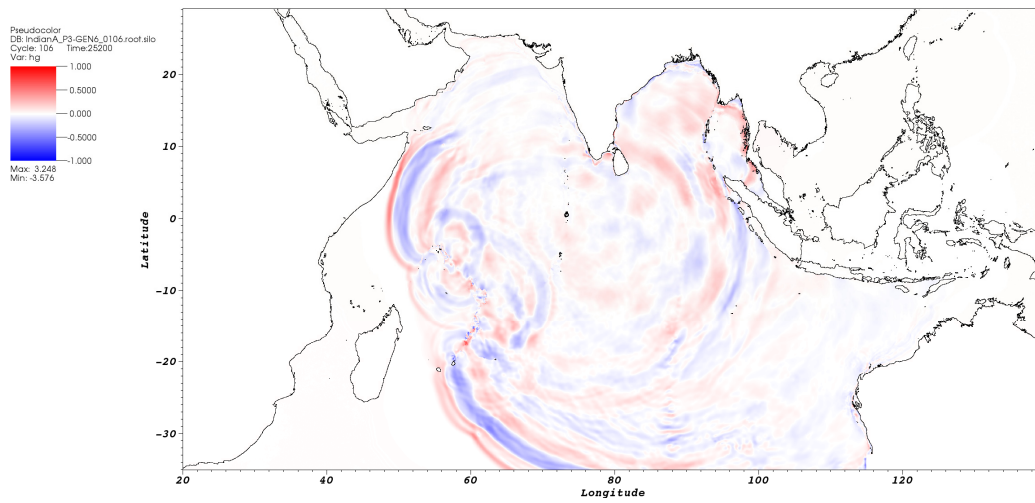


Fig. 6.1 (h) Time = 7 Hours

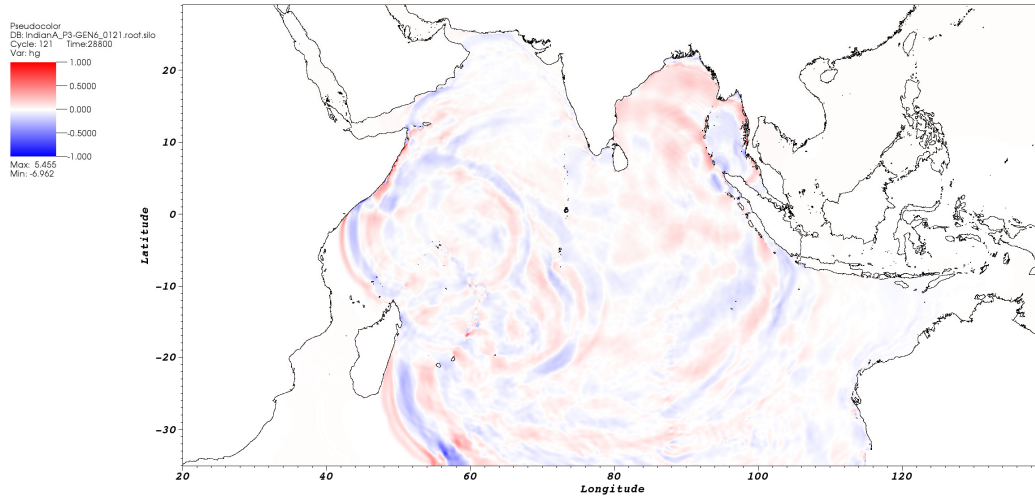


Fig. 6.1 (i) Time = 8 Hours

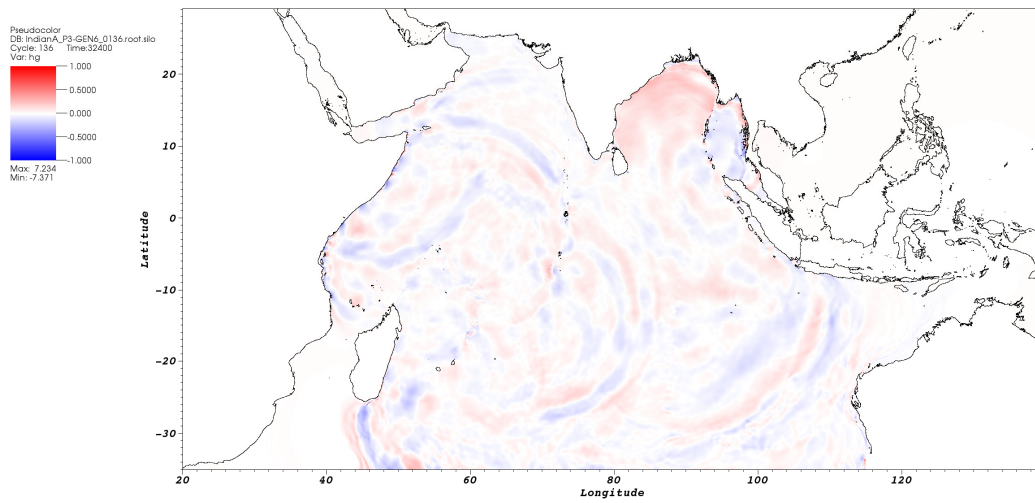


Fig. 6.1 (j) Time = 9 Hours

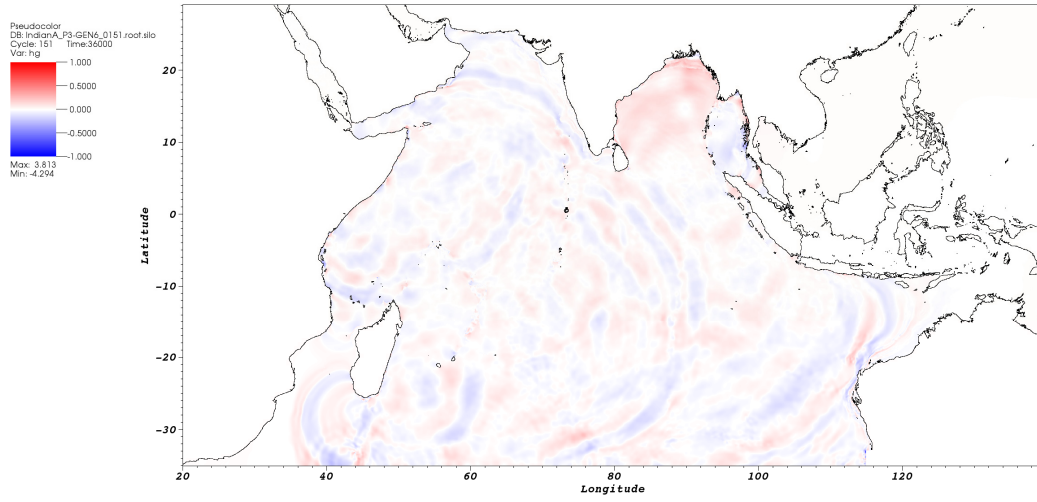


Fig. 6.1 (k) Time = 10 Hours

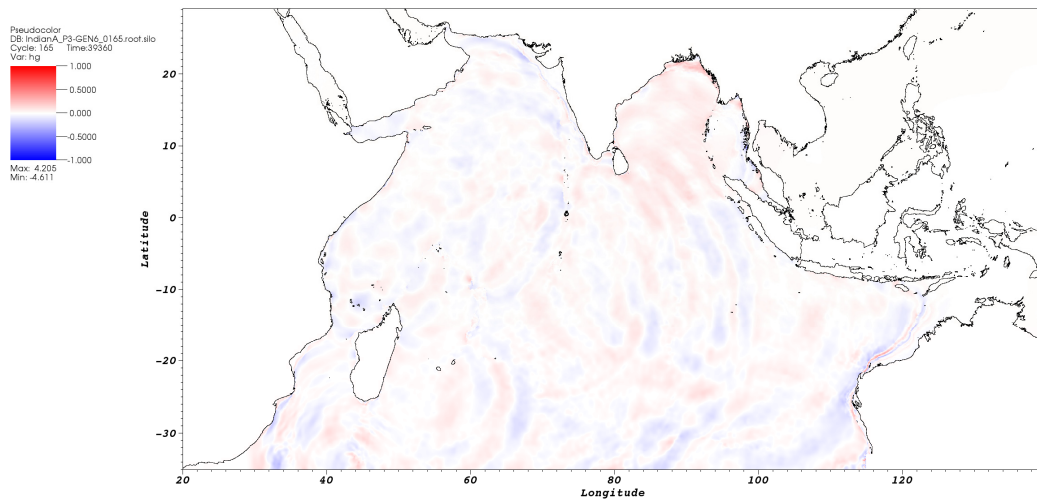
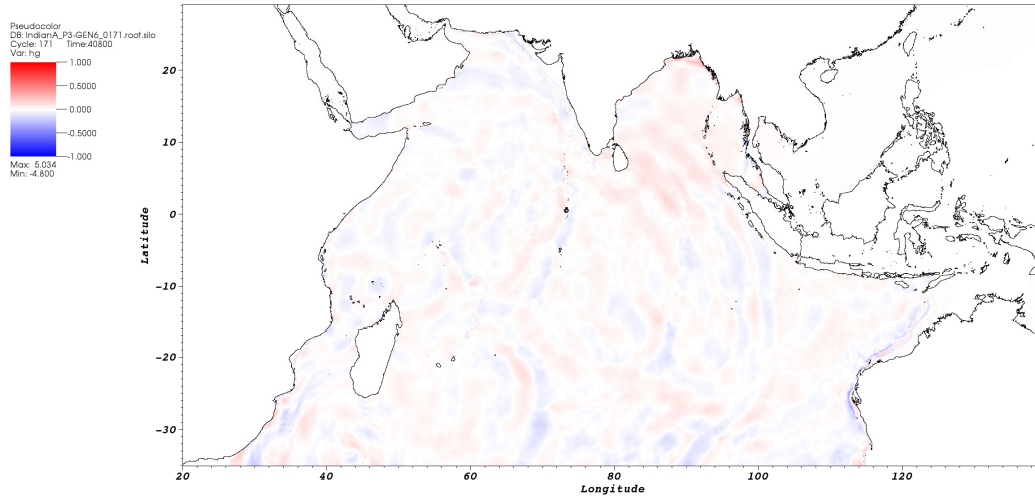


Fig. 6.1 (l) Time = 11 Hours



(m) Time = 12 Hours

Fig. 6.1 Hourly snapshots of the Indonesian 2004 tsunami propagation after the earthquake simulated by TRITON-G.

In conclusion, the animation generated by TRITON-G and shown in snapshots here above, was compared with existing simulations from different researches available at the tsunami events database [94]. From this qualitative comparison it was observed a correct wave propagation along the Indian Ocean; this served as a first confirmation that the results were accurate.

6.1.1 Tide gauge comparison

In order to properly estimate if the propagation was correct, several gauges were located in the domain during the simulation and then compared to the real values recorded by buoys. These buoys measure the ocean sea level at regular intervals and serve as a critical factor to determine tsunamis arrival times and heights. The recording frequency varies from system to system but in general is of 5 minutes. Although several sources are available, DART® (Deep-ocean Assessment and Reporting of Tsunamis) [95] buoy system is one of

the most well-known and accessible. In some charts presented in the results the initial ocean sea level is not zero even when the tsunami waves have not arrived yet, this is easily explained by the fact that the buoy stations report tide values which naturally rise and low ocean sea level. The location of the gauges used in this work is shown in *Fig. 6.2*, all these stations due to their position received tsunami waves directly from the source.

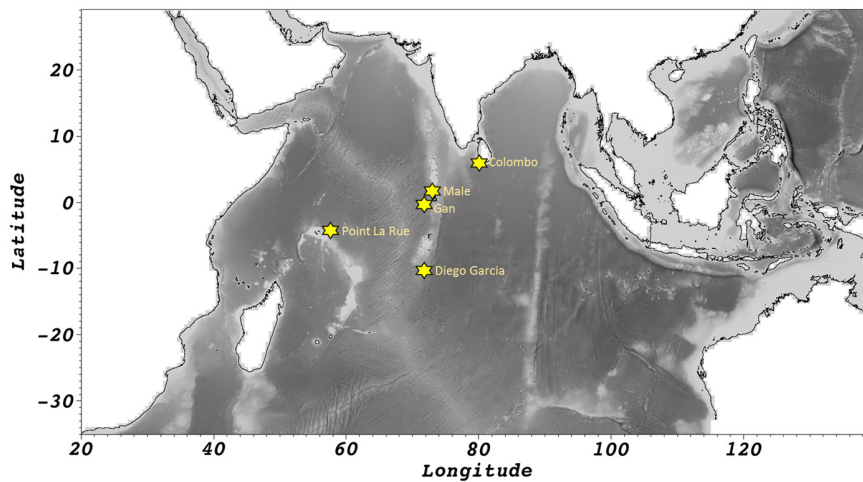


Fig. 6.2 Gauge locations in the Indian Ocean: Male, Gale, Diego Garcia, Colombo and Point Le Rue.

The first gauge compared is *Diego Garcia*, an atoll in the Chagos Archipelago, located at $7^{\circ}30'N$; $72^{\circ}38'E$ and approximately 2700 km from the west side of the fault. This station records every 6 minutes. The first wave arrived at 3 hours 46 minutes after the earthquake, *Fig. 6.3* shows the result of TRITON-G for this gauge. It is clear the arrival time agreement, TRITON-G predicted time is just a few minutes ahead from the actual event. Also the arrival height is in great accordance with the recorded gauge. One of the objective of this work is to produce a model that its results can be used for evacuation warnings, hence slightly overshooting do not affect this purpose.

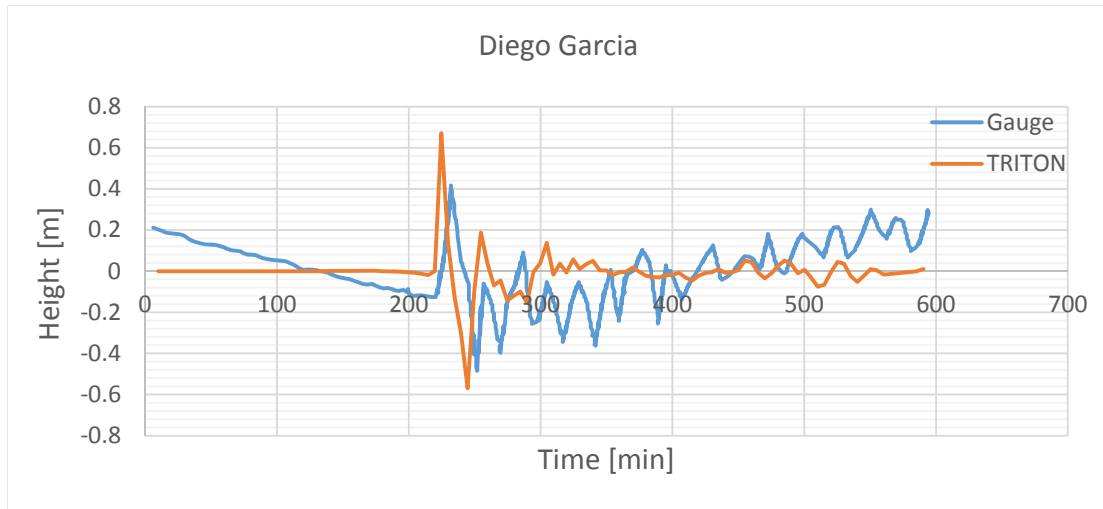


Fig. 6.3 Comparison of arrival times Diego Garcia tide gauge vs TRITON-G

This station served a second goal due to its location. The natural topography in the area between the source and this station show no obstacles hence allowing for an obstacle-free propagation of the wave. Thus, the results of this gauge serve also as an indirect benchmark to far-field propagation with real bathymetry. Just as shown in the first section, the implementation of the numerical model of TRITON-G appears to be correctly estimating the wave propagation in spherical coordinates and with complex bathymetry.

The second gauge compared is *Male*, near the Maldives Islands, located at $4^{\circ}18'N$; $73^{\circ}52' E$ and approximately 2000 km from the west side of the fault. This station is sampled every 4 minutes. Results for this gauge comparison are shown in Fig. 6.4, the arrival wave occurs 3 hours 17 min after the earthquake; again TRITON-G shows good agreement with this arrival time. Also, the main tsunami event peak is reproduced by our model, moreover it can be seen the excellent agreement as the tsunami continues to arrive; the two main receding wave are correctly estimated as well as the second and third tsunami wave train, with heights predicted within just a small difference of those measured by the gauge. All this despite the

considerable distance already traveled by the tsunami which confirm a good modeling of the far-field simulation.

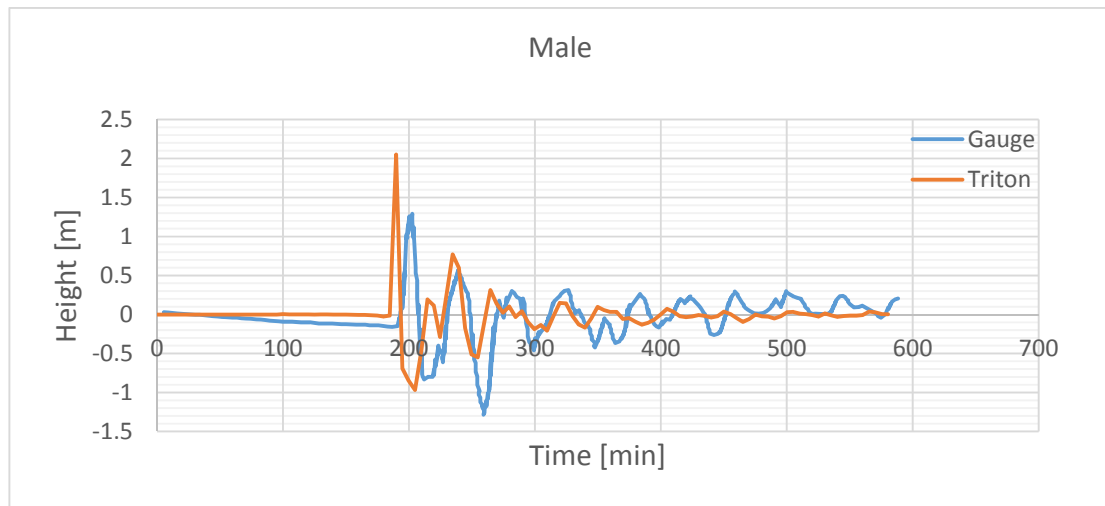


Fig. 6.4 Comparison of arrival times Male tide gauge vs TRITON-G

The third gauge presented is *Gan*, near the Maldives Islands, located at $0^{\circ}68'N$; $73^{\circ}17' E$ and approximately 2300 km from the west side of the fault. This station is sampled every 4 minutes. The comparison with our model is presented in Fig. 6.5. The arrival wave occurs at 3 hours 17 minutes after the earthquake; the model show good agreement with this arrival time within an acceptable small difference; also although there is overshooting present the peak of the first arrival wave and main event at this station is also reproduced consistent with the observed that the gauge location. The complexity of the bathymetry around this locate might affect the proper estimation of the arrival height while the shallow regions might be amplifying these values.

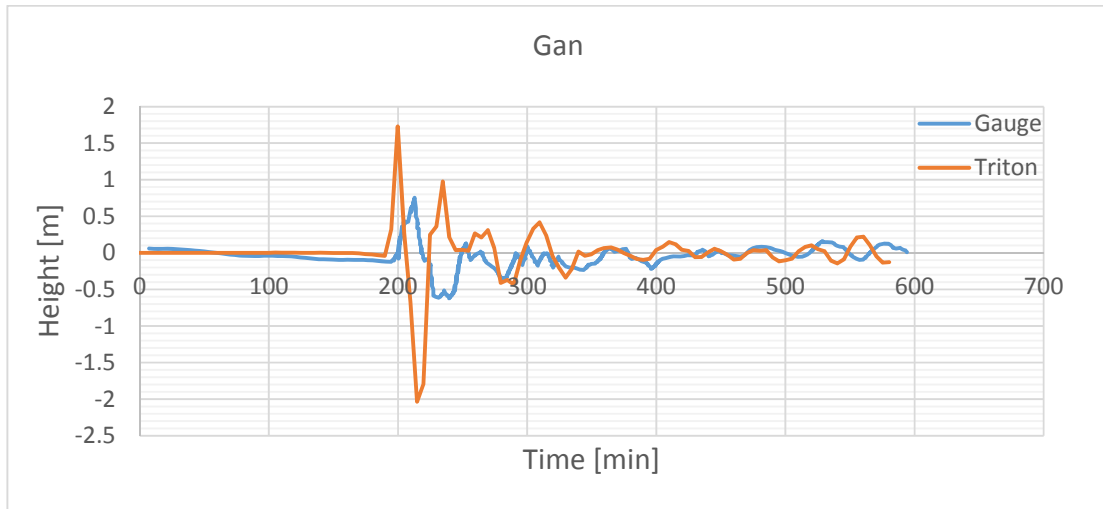


Fig. 6.5 Comparison of arrival times Gan tide gauge vs TRITON-G

The fourth gauge is *Colombo*, in Sri Lanka, located at $64^{\circ}93'N$; $79^{\circ}83' E$ and approximately 1400 km from the west side of the fault. This station is sampled every 2 minutes however due to the intensity of the arrival tsunami the gauge was damaged after the first wave, yet the recorded values until stop functioning prove valuable for comparison.

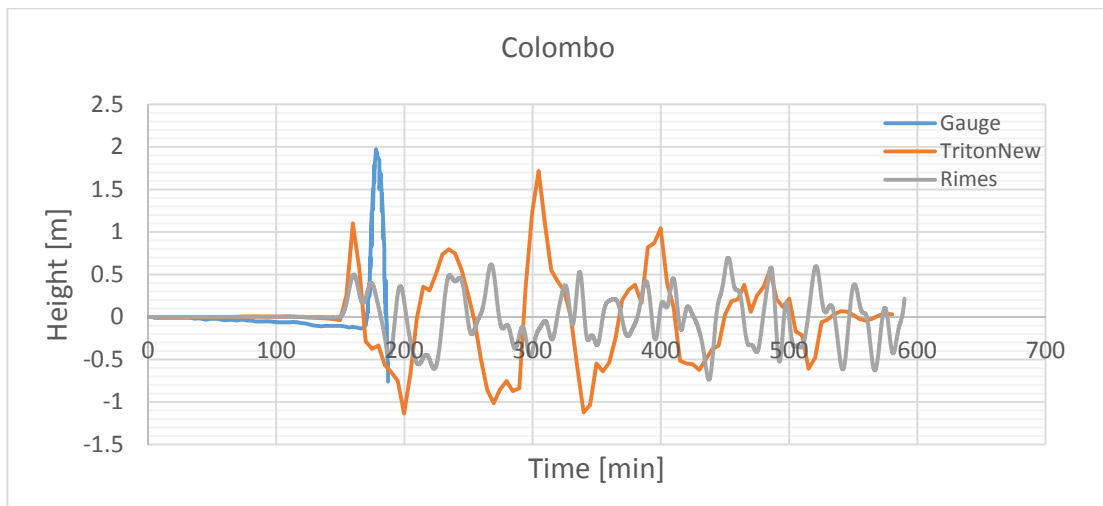


Fig. 6.6 Comparison of arrival times Colombo for tide gauge, TRITON-G, RIMES

In this station the arrival wave happens 2 hours 50 minutes after the earthquake; due to lack of data after the gauge’s damage, we rely on eyewitnesses accounts who reported that the first wave was not the biggest but the following ones.

The comparison of Colombo’s tide gauge with TRITON-G is shown in *Fig. 6.6*; the arrival time of the modeled wave is similar to that of the recorded gauge, the ahead time difference of the model is partly explained by the gauge location in the simulation; as explained in chapter 3, coastlines not marked for inundation have a wall boundary condition, hence gauges too close to the shoreline might not represent accurate results, in order to avoid this effect the gauge location in the simulation is located slight west of the actual buoy, this accounts for an earlier predicted arrival time. Besides the tide gauge and TRITON-G results *Fig. 6.6* also show the results from RIMES original model for comparison. This serve to illustrate how TRITON-G improved the height estimation compared to that of the previous model. Also looking at the wave train predicted by our model it can be seen that the largest peak does not occur with the first wave but instead at a later time a couple of hours later than the first event; this coincides with eyewitnesses accounts who reported larger waves later than the arrival one.

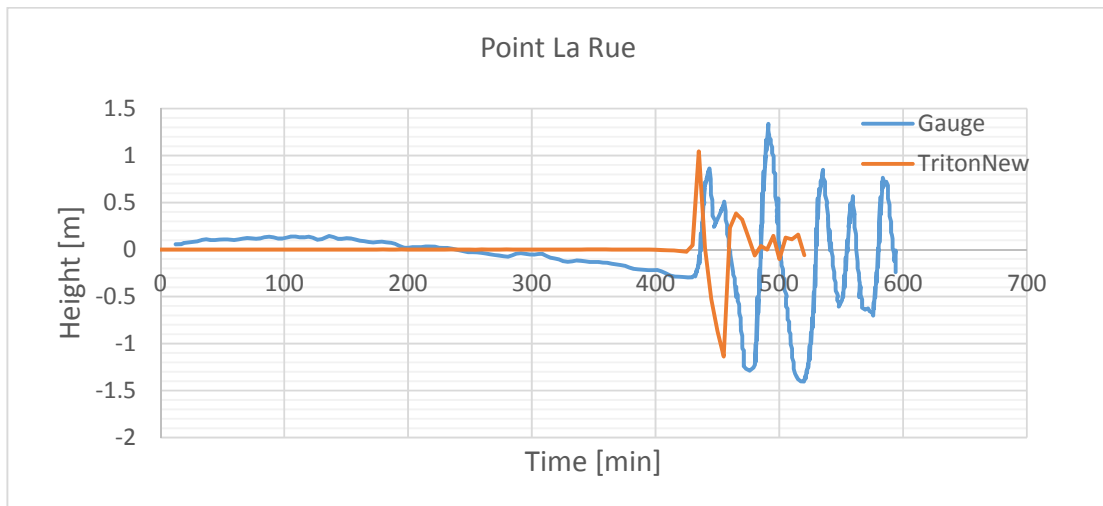


Fig. 6.7 Comparison of arrival times Point La Rue tide gauge vs TRITON-G

The last station is *Point La Rue*, near Seychelles, located at $4^{\circ}68'S$; $55^{\circ}53'E$ and approximately 4300 km from the south-west side of the fault. This station is sampled every 4 minutes. The arrival wave is reported at 7 hours and 4 minutes after the earthquake. The comparison of this gauge with our model is shown in *Fig. 6.7*. As it can be seen, the arrival time coincides with good agreement with that of the recorded values; similarly with the previous case the slight time difference is partly explained by the same reason, the location of the simulation gauge slight east of the actual buoy since this is very near the shoreline. This comparison represent very valuable information for the model validation; first, the location of this gauge accounts for a considerable far-field simulation, more than 4000 km are covered by the tsunami wave and yet the arrival time and wave height is within good agreement of the recorded values. Secondly, due to the location of the gauge, the traveling reaching it had gone through the Maldives islands, where complex bathymetry generate dispersion on the waves and coasts reflection affects as well; yet the model preserves the traveling wave well, there is no noticeable diffusion of dissipation of the wave that might reduce the arrival heights.

In conclusion, with the comparison of our model's results with real gauges it is possible to determine the good accuracy presented by TRITON-G. Arrival times for the traveling wave were all within good agreement with the recorded values, small differences can be partly explained by the simulation gauge location. Also it was shown that the main events could be reproduced; although a tendency to overshoot was noticed, this did not affect the ability of the model to transport the wave along far distances and in no case a wave sign was reported incorreccted, all crests coincides with observed and measured values. There are three possible reasons to the different wave oscillation after the main event as well as the peak height discrepancies:

- Bathymetry and Topography
- Initial Condition
- Dispersion

Although databases for bathymetry and topography with good accuracy are available, these are still far from representing in details the real shape of the Ocean's bottom and orography. This difference makes it challenging to reproduce the wave reflections on coasts and effects of traveling through the Ocean bed completely realistic. Hence it is expectable that some difference is found in the wave oscillations.

Also, every tsunami model is dependent on a good and accurate initial condition to obtain good simulations, source theory however is still a developing and challenging field and obtaining a realistic fault source is not always possible. The use of an inaccurate initial fault source can produce differences in the arrival wave heights, also in the direction and speed of the traveling wave.

Waves traveling through the Ocean bed experience physical dispersion due to the effect of the bathymetry. In general this dispersion is compensated by numerical dispersion introduced by the truncation error; however as it was explained in the methods section, for TRITON-G using the cubic interpolation upwind scheme has the advantage of minimizing dispersion and diffusion. The result is a homogeneous traveling wave with minimum dispersion effect thus reducing the possibility of seeing the high oscillatory behavior of the arrival tsunami wave in the gauges.

Finally, this kind of discrepancies between recorded gauges and modeling programs is not exclusive to us but instead rather common; the same challenges experienced in this work are reported in other operational models as well however our discrepancies tend to be less than those in other researches ([96], [97]).

6.1.2 Inundation Run-up: Focal Areas, Phuket

In order to continue the validation of our tsunami model this section present the results obtained for the inundation calculation of the original 4 focal areas: Mozambique, Comoros, Seychelles and Sri Lanka, plus an additional focal area in Phuket.

Although exact measured inundation maps of these areas does not exist, the results are based on RIMES existing simulated repository and reports [98], and on post-tsunami damage surveys; with collecting information concerning the damage produced by the inundation, it is possible to estimate the maximum heights and run-up on several locations.

The first case presented is the result for Sri Lanka focal area. Specifically this area covers the coastal region known as Hambantota. Eye witness accounts report the arrival time of the first tsunami wave around 9 am the morning of the 26th, some two hours after the initial earthquake in Sumatra. This arrival time coincides with results obtained with our model. It is important to mention that the result for this focal area were computed by TRITON-G in just 9 minutes, this show the speed advantage provided by our model; obtaining fast predictions allow authorities to make an evacuation decision quickly and save lives.

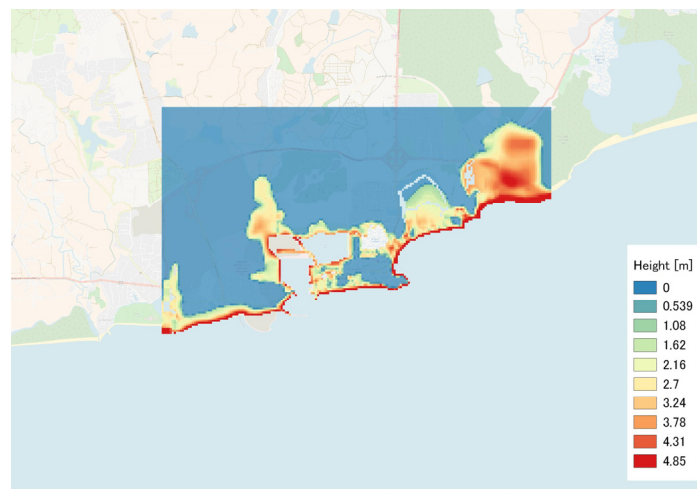


Fig. 6.8 Hambantota Inundation Map, Sri Lanka FA

The inundation map for Hambantota generated by TRITON-G is shown in *Fig. 6.8*. According to measurements done post-tsunami, it was determine that the arrival waves had heights of over 8 meters and produced run-ups inland in certain areas of up to 2 km. As it can be seen in *Fig. 6.8*, TRITON-G results reproduce this account, the inundation map show areas

up the coastal bay where inundation produced hundreds of meters deep run-ups in land. Also, the image on the left of *Fig. 6.10* show the maximum wave heights computed, the results show values of over 8 meters just as reported by witnesses, in fact wave heights of almost 10m were obtained during simulation.

These qualitative comparisons show good agreement with the observations made on field. In order to compare more accurately the result of our maps, a comparison between RIMES Hambantota inundation report and TRITON-G's simulation is presented in *Fig. 6.9*; the image on the right represents TRITON-G's result with the area for Hambantota trimmed to fit that of the report.

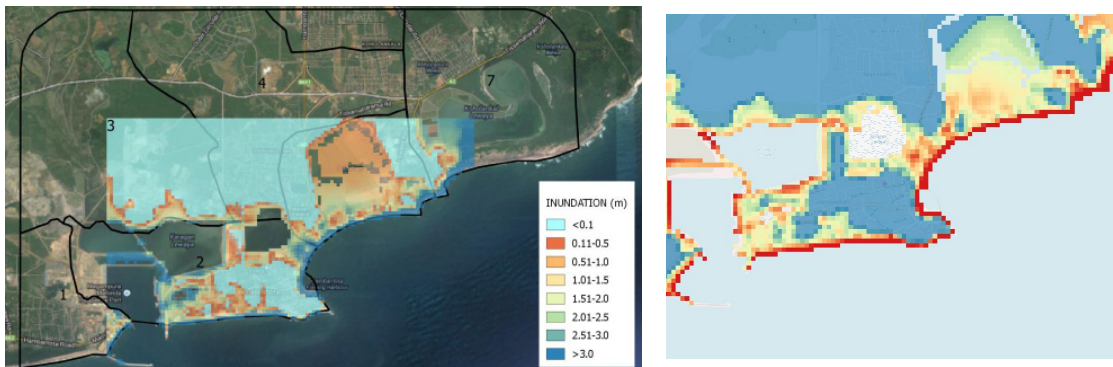


Fig. 6.9 Inundation comparison Hambantota, Sri Lanka; RIMES (left) vs TRITON-G (right)

Although the presence of topography, irregular bottom friction and complex interaction with the land makes difficult to compare directly between results, it is clear that in general both simulations have excellent agreement. Both simulations show agreement on the areas that were inundation and the areas that experienced no run-up. The decisive factor that made some areas more propense to inundation than others was the topography. The arrival tsunami wave hit the coast with heights of around 8-10 meters, coastal areas that faced the ocean with higher topographic heights were spared from being inundated. On the

contrary, coast shores that were practically flat were overtaken by the incoming wave as shown in the results.

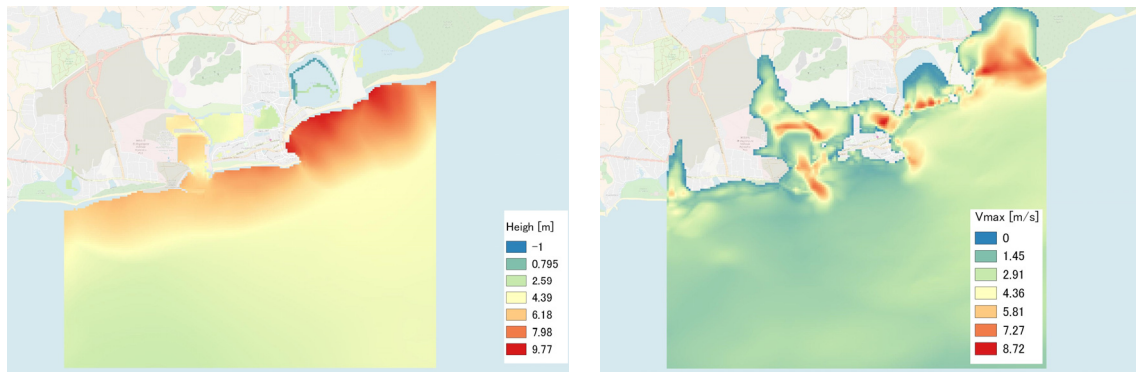


Fig. 6.10 Hambantota, Sri Lanka; left: maximum wave height; right: maximum wave velocity

Next, the results obtained for the rest of the focal areas are presented. It is important to mention the case of Mozambique for who maps were not generated. The reason of this is not oversight but a programming condition. As explained in the GPU chapters, output data for the focal areas start to get stored if the wave entering the focal area exceeds a certain target value, in this case 0.1 meters. The results for the Indonesian tsunami showed that the arriving waves at Mozambique were less than 0.1 m hence no output was recorded. Nonetheless, to guarantee that the four focal areas were able to compute inundation and estimate possible damage, a separated analysis was performed for the African region around Mozambique, based on the work of Aderito et.al. [99]. Results for this study case can be consulted in Appendix B (Source Fault Simulations section) of this work. Scenario 10, which correspond to a fault near Mozambique was chosen as the case to compare our result. TRITON-G proved to be stable and generate a propagation also in this region of the domain; since this is a theoretical case no actual gauges measurement exists, however from a qualitative comparison with the results in [99] we could determine that the heights and arrival times were within good agreement.

The inundation map for Seychelles is shown on the right of *Fig. 6.11*. Although no detailed simulation is available to compare the inundation, again we use the eye witness accounts and post-tsunami surveys. The report [100] found run-up in the east coast of the island, a region called Anse Royale, where infrastructure damage on this beach was found. The inundation map coincides with this studies as shown in this figure. Inundation with heights of 2-3 meters were obtained by TRITON-G and as shown in the map, the run-up went in land in the beach.

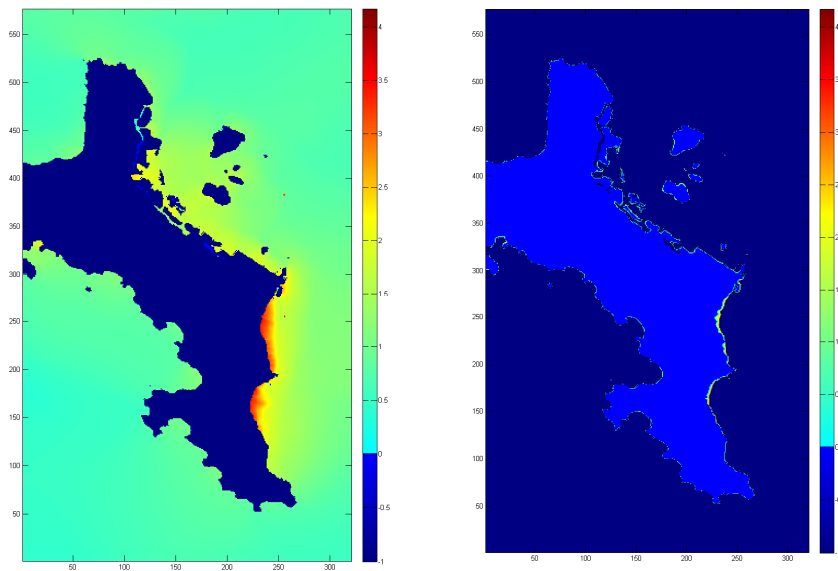


Fig. 6.11 Seychelles FA, left: maximum arrival wave; right: inundation map

Also according to [100] arrival wave heights of around 1.9 and 2.8m in Anse Royale were observed; this is in good agreement with the results obtained with TRITON-G, the image on the left of *Fig. 6.11* shows the maximum wave heights with values up to 3.5m.

Finally the results for Comoros is shown in *Fig. 6.12*, the arrival times estimated by our model of around 6 hours coincides with the accounts recorded. The map on the left show

the results for the maximum wave height and the image on the right show inundation. Due to the small arrival wave, the inundation did not produce a noticeable run-up for this case.

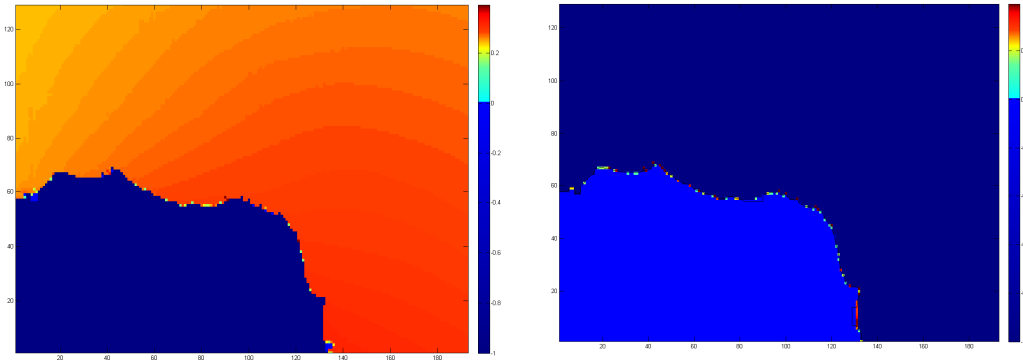


Fig. 6.12 Comoros FA, left: maximum arrivale wave; right: inundation map

An additional test was tested during TRITON-G development: Phuket. This case served the purpose not only to re-validate even more the good accuracy of TRITON-G but to test the flexibility of including a new focal area. Successfully, the new databases for this region were submitted to TRITON-G and a new mesh was generated around this focal area. With this new mesh the Indonesian tsunami was computed again to obtain results for the inundation.

The bathymetry databases kindly submitted by RIMES are shown in *Fig. 6.13*, just like the other FA two sets were used, one with 150m resolution and the other with 50m resolution, matching level 7. The Phuket focal area is highlighted in red and show an interesting case that did not appear before: the mix of bathymetry source inside on single focal area. In the cases of the other four focal areas, the highest resolution was always covered by a single dataset but in this case part of the focal area bathymetry, Kamala, is replaced directly from a fine 50m dataset while the other part, Patong, is interpolated from a coarser resolution. The effect of doing this will show the importance of accurate databases.

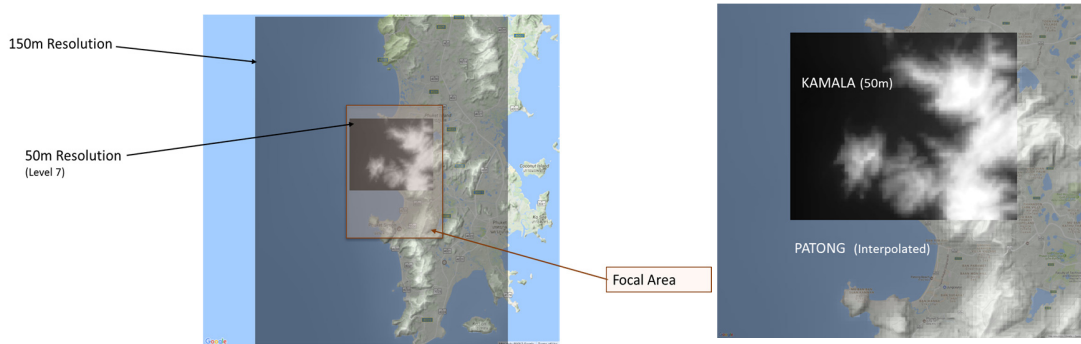


Fig. 6.13 Phuket Inundation: left bathymetry databases used; right: zoom on Kamala and Patong

The inundation map for the Phuket focal area, which includes the region of Kamala and Patong is shown in Fig. 6.14. The work by Suppasri [101] is to compare these results.

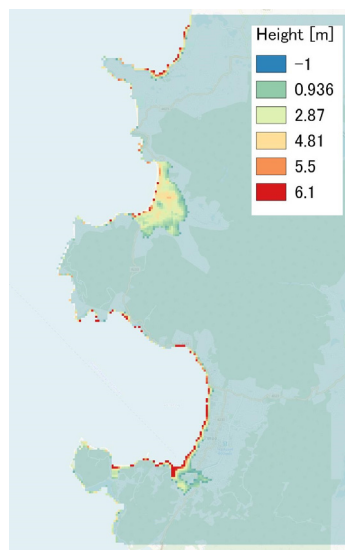


Fig. 6.14 Kamala and Patong maximum inundation map TRITON-G

The wave arrival time for this region is of around 181 minutes, which agrees with the values obtained by our model. The results to do this comparison are shown in Fig. 6.15, where

the focal area Phuket was trimmed to match the area reported in [101]. The image on the left present the inundation simulation obtained in the report while the image on the right depicts the results of our model. It can be readily see that in general, the results around the Kamala region coincide well between models. Both systems report maximum inundation heights of around 5-6 meters and the run-up distances follow the same pattern. This is another good validation for TRITON-G as a propagation and inundation modeling tool.

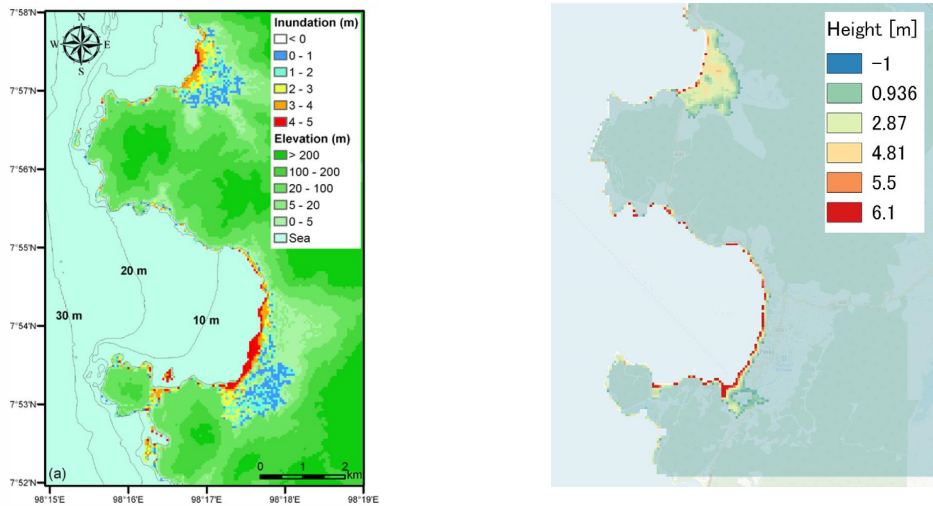


Fig. 6.15 Kamala inundation map comparison. Left: Suppasri et.al. [101]; right: TRITON-G

There is however certain discrepancy in the inundation run-up in the Patong region. As mentioned before the bathymetric dataset obtained for this part of focal area was originally of 150m, hence to obtain a 50m resolution interpolation was needed. Therefore while uniform and un-interpolated bathymetry and topography was used in [101], our model used a mix of interpolated and high accurate values. This effect explains this discrepancy and serves as a warning on the importance of having realistic and high accurate bathymetry and topography in order to obtain high resolution.

In conclusion, except for Mozambique, the focal areas reproduced correctly the inundation. Arrival times and maximum wave heights and inundation were correctly predicted by our model. More precise comparison on the Hambantota region and Kamala served as proof to support the effectiveness of our model. Moreover, the test on Phuket showed the importance of using real and accurate topography and bathymetry and the effect that using smoothen values can produce.

6.2 Concluding remarks

As seen in earlier chapters, TRITON-G delivers fast computation for complex and large simulations using GPU acceleration. And now by using benchmarks and real data comparisons it is shown that the model used by TRITON-G also provides accurate results. Hindcasting the Indonesian 2004 tsunami serves as a test to determine the behavior of the model under real circumstances. Comparison with several tide gauges across the Indian Ocean showed very good agreement with the wave arrival times and confirmed correct reproduction of the arrival wave peak event. Discrepancies in the arrival height can be explained by the effect of the initial conditional and bathymetric effects, while the discrepancies on the oscillatory wave behavior after the main event can be understood as a dispersion effect. The gauges also served to test the model stability and ability to preserve the traveling wave across long distances. Moreover by comparing results with existing post-tsunami damage surveys and other simulations it was possible to determine the correct prediction of inundation, in particular for Hambantota and Phuket regions. This last test also demonstrated the flexibility of TRITON-G to generate new FA and also was an excellent example of the influence in the results of using high accurate bathymetry as opposed to interpolated values.

Chapter 7. Conclusions and Future Work

7.1 Conclusions

In this work a novel, fast and accurate tsunami modeling was introduced. The combination of highly accurate numerical methods provided an excellent solution to the governing equations. Moreover the development of a customized refinement served to create a mesh optimized for the resources available and the interests of RIMES. A full-GPU implementation was proven successful, several optimizations introduced in the kernels and load balance as well as using multi-GPU allowed to obtain high performance from the Tesla K40 cards used. A large simulation of 10 hours in real time can be computed in 40 minutes machine time, including considerable-sized output. Hindcast of the Indonesian 2004 tsunami helped to compare results and find good agreement with propagation and inundation. Also the good agreements with benchmarks serves as a back up to the validity of this work.

The main conclusions are:

- A tsunami model was successfully developed that proved to be accurate, robust and outstandingly fast.
- A full-GPU operational model was properly implemented; multi-GPU delivered high speed simulation and performance.
- By using the proposed numerical methods, the governing equations correctly modeled the tsunami propagation and inundation accurately with smooth integration between different coordinate systems.
- A customized refinement based on tree-based refinement was developed to produce a mesh that provide accuracy, and efficient use of resources.
- By exhaustive testing, benchmark comparing, and hindcasting, the program developed proved to be reliable under a wide range of circumstances.

- The use of an easy interface to interact with the program through simple parameters files proved to give the degree of flexibility desired to the end user.

The work done in this research produced TRITON-G, a tsunami operation real-time model that has proved robust and accurate. It was tailored-made to fulfill the requirement of RIMES and to replace their previous program. At this time TRITON-G has been already deployed and has been under operation at RIMES headquarters with success. The continuous testing in the developing stage and now at operation stage has confirmed the robustness of the program. Moreover TRITON-G ended up being a full framework, providing additional utilities for data and parameters pre-processing, like database distance generation, and output post-processing, like inundation images, variables output, and gauges.

The innovative approach to implement the model as a full-GPU application proved to be an excellent decision. The large domain computing is accelerated by the used of multi-GPU. Even though only three Tesla K40 cards were available, it was possible to obtain great acceleration for simulation with great complexity. Being able to obtain results within minutes on areas close to the tsunami source is essential to warning systems. TRITON-G can fulfil this requirement single-handedly. The optimization implemented in the kernel provided the necessary boost to obtain the best performance out of the cards. Also working with the Pipe asynchronous work flow for output provide the advantage of delivering results as frequent as requested. In this way, results can be checked progressively as the simulation advances.

The two refinement factors custom developed for TRITON-G proved to be accurate and efficient. By handling different block levels, high resolutions of 50m in coastlines can be obtained while coarser resolutions of 2 arc-min remain in the open ocean. Moreover, despite the fractal structure of coastlines, the refinement was also able to track the complex coast shapes accurately. The introduction of the concept of focal areas worked perfectly to diminish the memory usage while not losing accuracy in the coastlines of interest.

When compared TRITON-G with existing data from the actual tsunami of Indonesia 2004, it proved yet again to be accurate; it reproduced arrival times in high accordance with the gauges. Also the arriving wave heights reproduced the peaks of the main events, always matching or improving simulations from the previous RIMES program. Although a small tendency to overshoot the heights was noticed, this can be manageable for warning systems since their aim is to look for worst-case scenarios to make evacuation decisions.

Finally, the idea behind collaborating in this project with RIMES not only aimed at producing a model that was accurate but also flexible. It is not so hard to create a specialized simulation for a single case with small variations however TRITON-G has the ability to be a flexible operational tool. The input information is easily handled by parameter files that make a smooth interface for the user. The block mesh can be custom made, any focal areas of interest can be set, the distance to refine can be changed or the number of levels, all these options available to the user. Furthermore the output variables and images can be easily controlled by the user with a simple parameter file.

It is our deepest hope that with this humble contribution to forecasting we help to move forward tsunami models into the next step, and by doing so, the lives of people at risk of being affected by a tsunami are better protected.

7.2 Future Work

A fresh and novel tsunami operational model has been presented in this work, TRITON-G. It has proven to be a fast program and generate accurate results. Moreover, it successfully fulfilled the requirements for the collaborating with RIMES. Since TRITON-G was developed under specific requirements it is understandable that certain limitations had to be taken. Nonetheless, outside of the collaborating project, it would be possible to take

TRITON-G to a new level of performance. Budget constraints made available only 3 GPU cards for this work, however TRITON-G is already developed with a multi-node, multi-GPU framework in mind. Hence with a few adjustments it could be deployed in larger servers, with more GPUs available or even in a Super Computer, like Tsubame 3.0 at Tokyo Institute of Technology. This new super computer is composed of a large cluster of GPUs, nVIDIA Tesla P100 cards. As it was shown in the GPU chapter, TRITON-G has been already tested on this P100 cards, achieving high performance. Also additional kernel tuning can be developed for newer cards. Another improvement would be the available memory, with more memory it would be possible to process larger domains i.e. more blocks, for example computing the whole Earth instead of a portion in the Indian Ocean. Furthermore, it would be also possible to implement TRITON-G as a tool for structural design, following the new NRA regulations this work can be modified to assess tsunami risks on areas where NPP buildings exist or are planned to be constructed. The flexibility and reliability of TRITON-G permits to test many inundation scenarios for different tsunami heights as well as do it swiftly by its GPU acceleration.

Secondly, a trend that is growing in the simulation community is the use of mixed precision. GPU cards start to include variables to handle this case. Hence, it is possible to apply single precision to areas where water depth is not relatively shallow (around -1000 m) and use double precision for deeper regions. This would be an improvement in speed because of the introduction of single precision computing but it also represent a challenge to connect areas with different precision smoothly as well as keeping a correct load balance on the cards.

Also, additional complexity could be included in the model to include effects that were neglected in this work for performance tradeoff. For instance wave dispersion or sea tides could be incorporated if more resources were available.

TRITON-G proved to be a robust model, however certain care had to be taken due to the presence of very large bathymetry gradients, peaks and plateaus. The appropriate election of a Δt together with an added small artificial viscosity served to overcome those problematic areas. However, where those approaches proved too general, a more specific and

dedicated approach could be taken by the use of morphometrics [102]. By determining the specific shapes that cause numerical instabilities they could be automatically identified using morphometrics thus making it easier to treat locally those points and drastically reducing the need of general solutions.

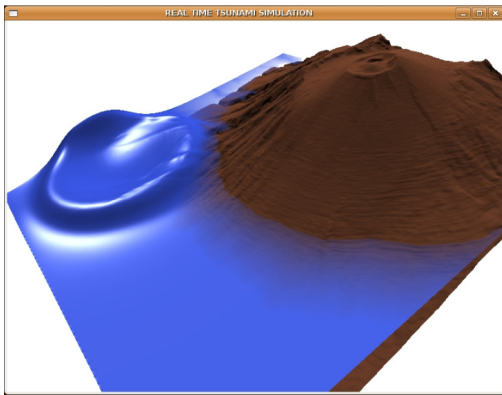
Finally, in recent years the influence of deep learning has grown exponentially due to the computing power of GPUs. Training a model on CPU could be a task of weeks and months, however with GPUs neural networks can be fed and trained in days or hours. A long term goal would be to implement deep learning for code auto-tuning and forecasting.

“The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man”

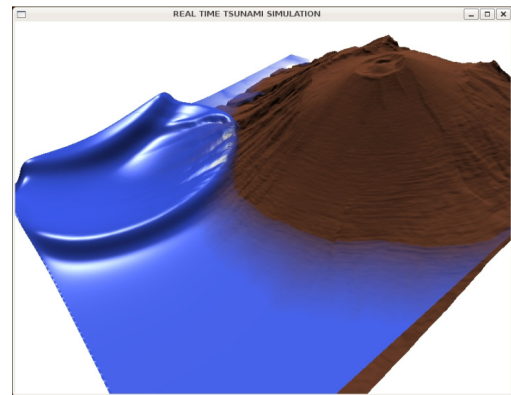
George Bernard Shaw, 1856-1950

Appendix A: Previous Work

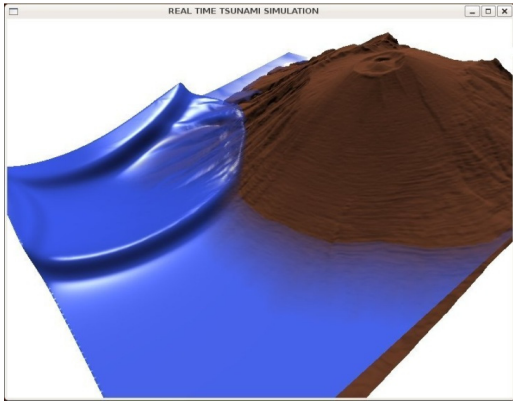
Our interest in GPGPU computing as a tool for accelerating simulations is not recent, starting in 2007 great success was achieved accelerating real-time tsunami simulations [103]. Even though the technology was still in the first stages of development, we could produce a simulation that outperformed CPU computing by up to 62 times on single GPU. This allowed us to create an interactive real-time visualization as well which was very innovative by then. Moreover multi-GPU was also explored and again great success was obtained in the simulation acceleration. GPU cards in that time had small memory which made no possible large simulation however even in those conditions our tests for big grids showed excellent scalability. Supercomputer Tsubame 1.2 was composed of a cluster of GPU, state-of-the-art in that moment, and this allowed us to test multi-node scalability as well.



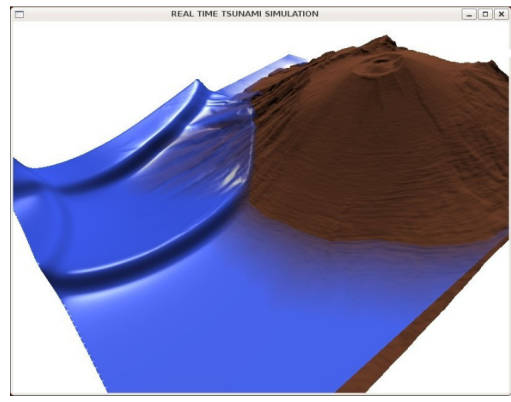
(a)



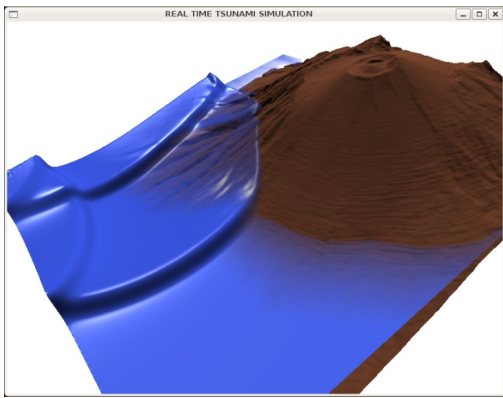
(b)



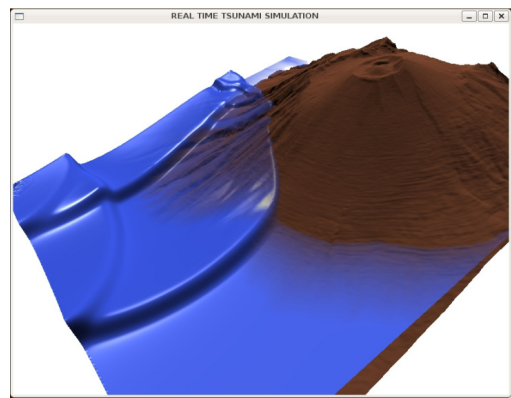
(c)



(d)



(e)



(f)

Fig. A.1 Tsunami Simulation running on Terrain II. Increments of aprox. 1min shown

The most outstanding result from that study was to achieve the same performance of 1000 CPUs (AMD Dual Opteron) with just 8 GPUs (Tesla S1070).

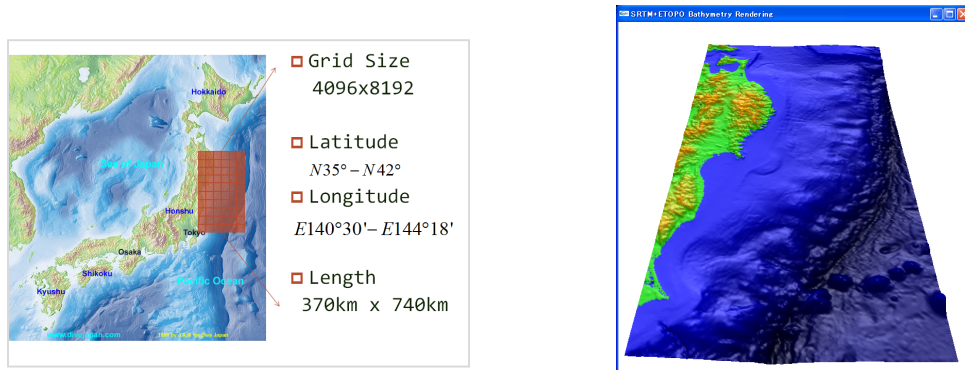


Fig. A.2 Japan Tohoku Region study case, SRTM and ETOPO merged Bathymetry [103]

Fig. A.1 shows the first successful real-time simulation obtained. Although the domain did not represent any specific area and had a relatively small size of 512x512, the promising results guided us to try real bathymetry as shown in Fig. A.2; the Tohoku region of Japan was used as the study case for the GPU acceleration tests. Results of the excellent scalability and outstanding speed up are shown in Table A.1.

GPU	Grid Size	Time	SpeedUp	Efficiency(%)
2GPU				
	512	9.349	1.330	66
	1024	24.86	1.931	97
	2048	96.34	1.941	97
	4096	381.3	1.934	97
4GPU				
	512	17.459	0.712	18
	1024	18.898	2.540	63
	2048	49.276	3.795	95
	4096	194.057	3.801	95
8GPU				
	512	139.14	0.089	1
	1024	147.341	0.326	4
	2048	141.815	1.319	16
	4096	112.68	6.546	82

Table A.1 Asynchronous Scalability for 2, 4 and 8 GPUs; Tsubame 1.5 Tesla S1070

Appendix B. GPU Boost

As explained in the Tesla K40 Application Note [104]:

In the Tesla K40 there is something called the “Base Clock” and “Boost Clock(s)”:

Base Clock: Selected based on worst-case reference workload. All Tesla K40 boards ship at the graphics core clock set at “base clock.” By default all Tesla K40 boards will run at this clock setting.

Boost Clock(s): These clocks are selected based on less power aggressive workloads. There may be more than one boost clock to provide deterministic performance for workloads that consume less than 235 W. In the case of the Tesla K40 there are two boost clocks. An end user can select one of the boost clocks using NVML or nvidia-smi. As long as the board power remains within 235 W the board will maintain the selected boost clock for the entire execution period.

In order to boost K40, the following command is used to change the card’s running clock:

```
sudo nvidia-smi -ac 3004,875 -i {CardID}
```

It is interesting to note that the K80 card also possess the Boost option, however in this case the option and configuration is automatically handled internally by the card. This self-adjusting behavior can be seen in the following measurements done during our work; the oscillation points on power and temperature for K80 charts show how the card changed clock configuration during TRITON-G execution while the K40 remained stable.

Boost Behavior Comparison K80 vs K40

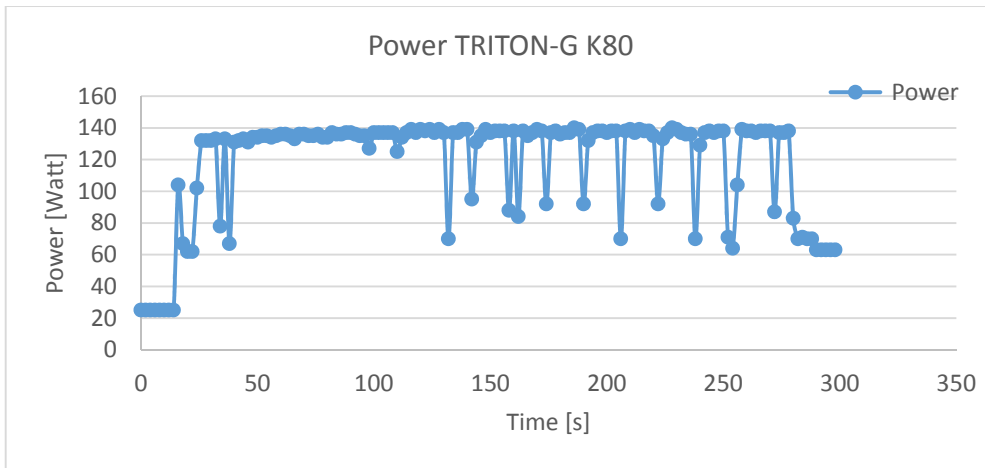


Fig. B.1 Power measurement for K80 Autoboot

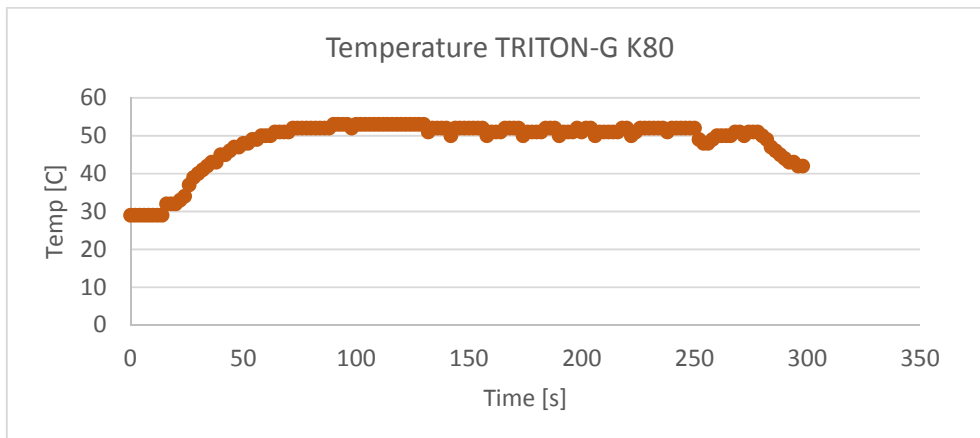


Fig. B.2 Temperature measurement for K80 Autoboot

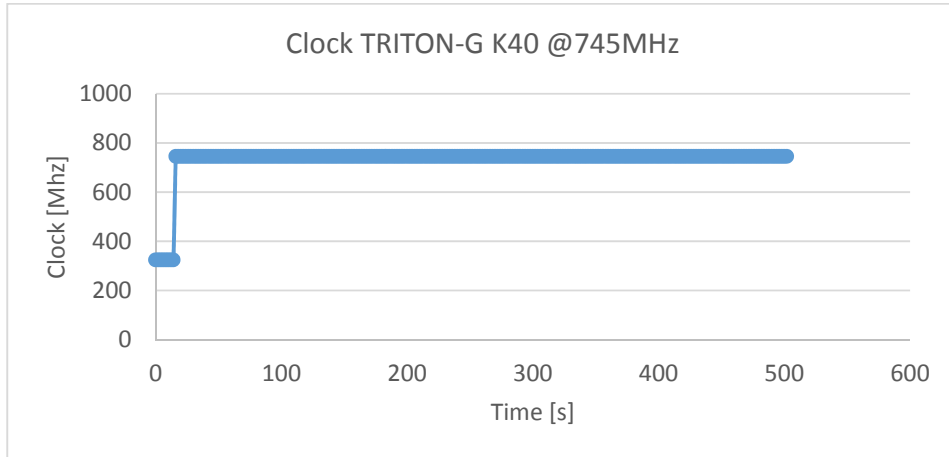


Fig. B.3 Clock measurement for K40 at 745Mhz

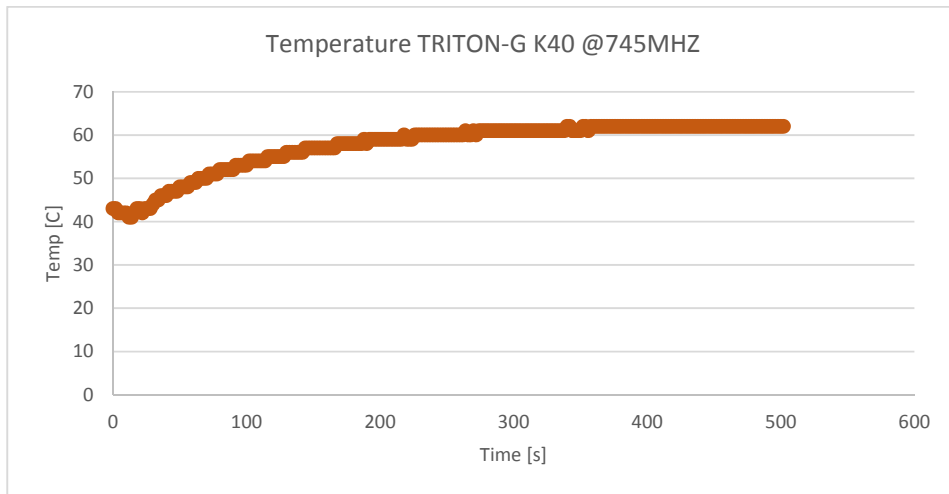


Fig. B.4 Temperature measurement for K40 at 745Mhz

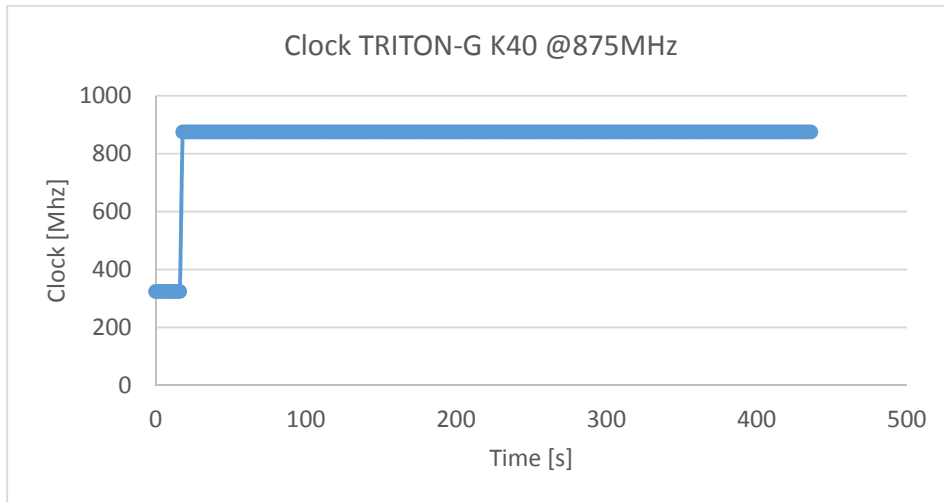


Fig B.5 Clock measurement for K40 at 875Mhz

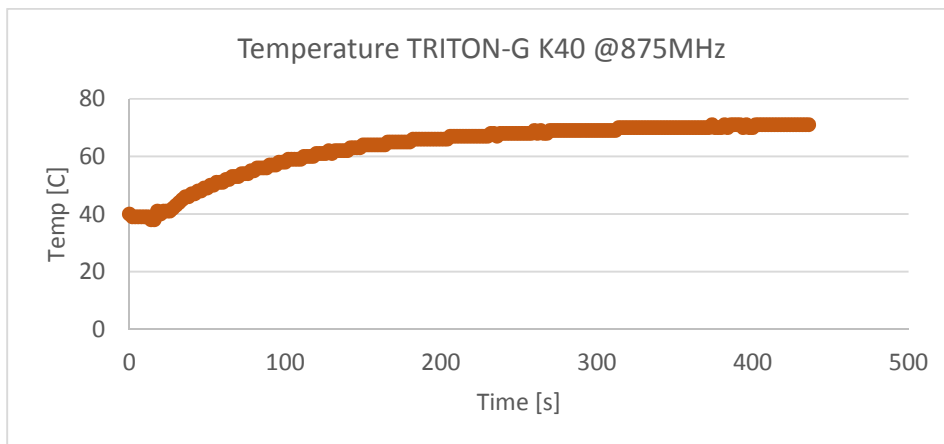


Fig B.4 Power measurement for K40 at 875Mhz

Appendix C Additional TRITON-G Tests

To further test TRITON-G stability under different circumstances and wavelengths, wave heights and initial locations, exhaustive testing was done, some of tests results are presented here divided into two groups: Gaussian Initial Conditions and Fault Sources.

1) Gaussian Waves Initial Condition

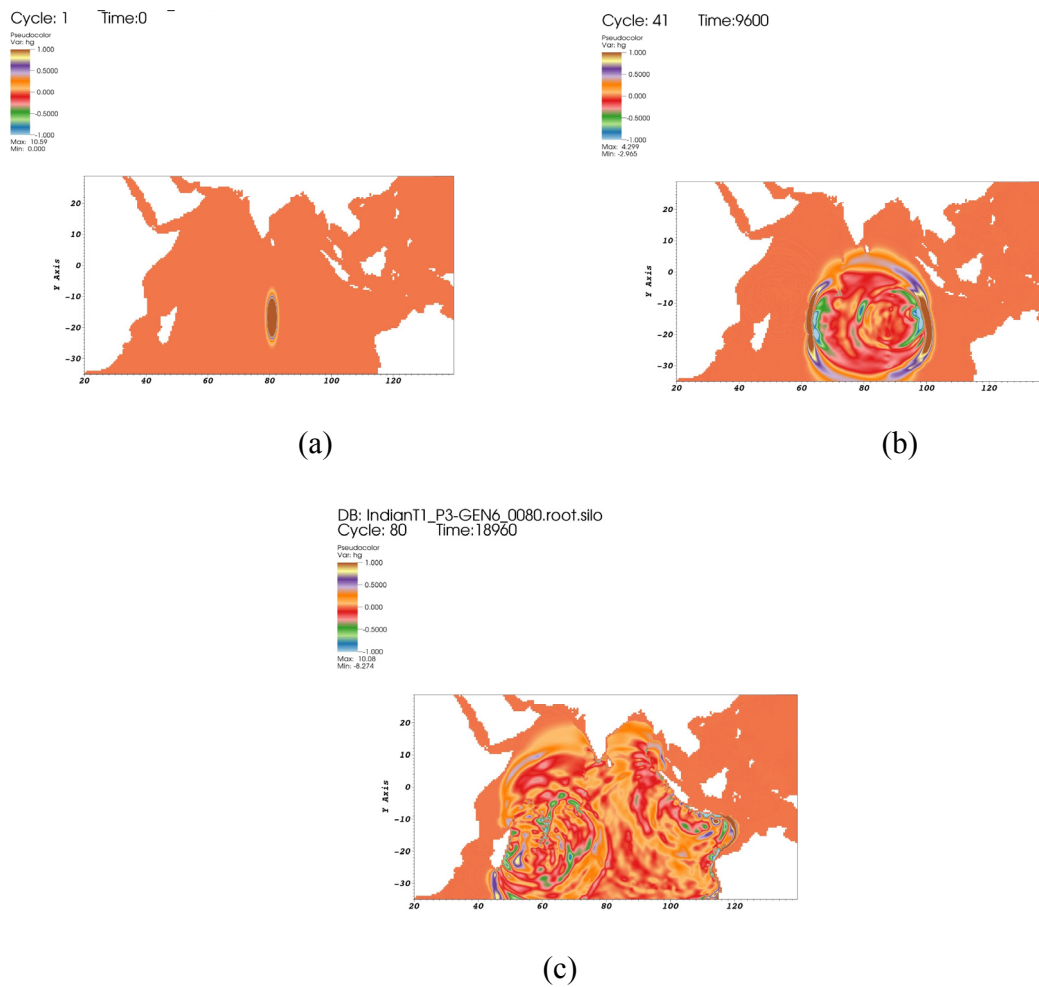
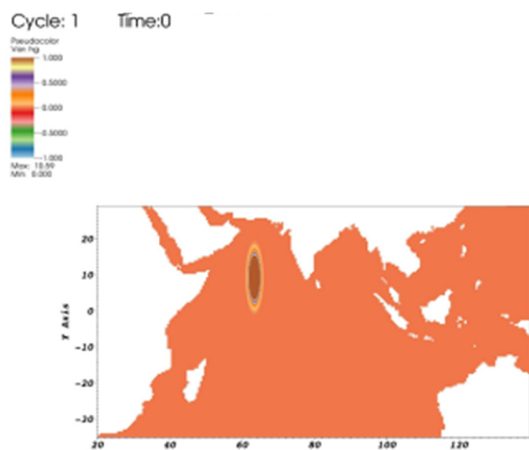
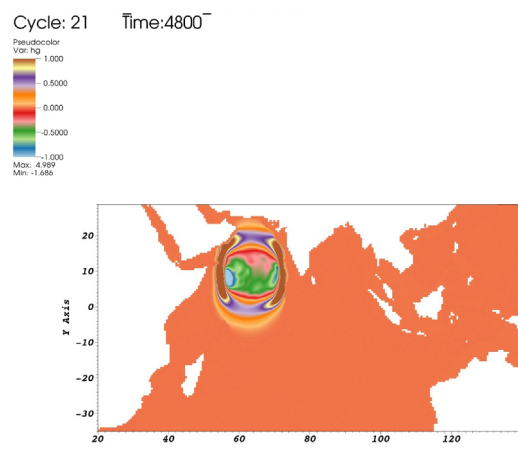


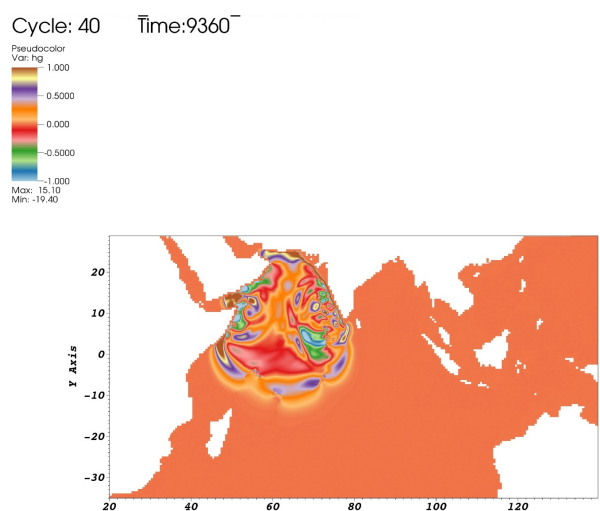
Fig. C.1 Gaussian 1



(a)

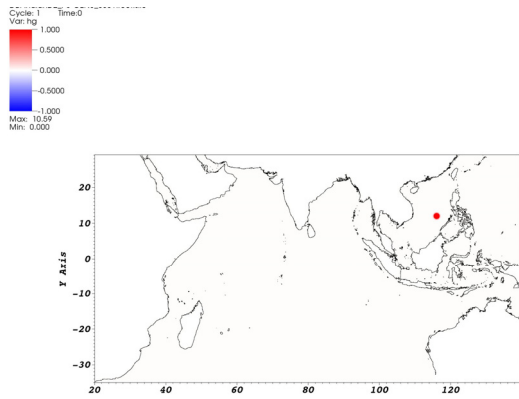


(b)

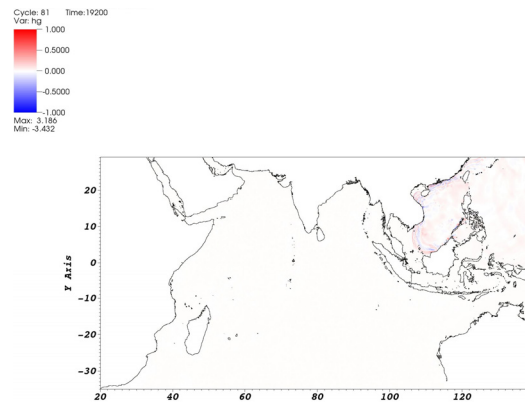


(c)

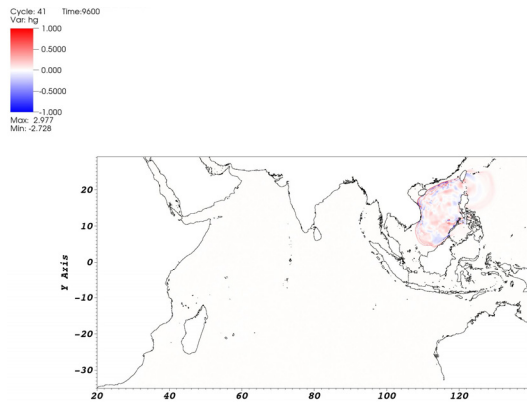
Fig. C.2 Gaussian 2



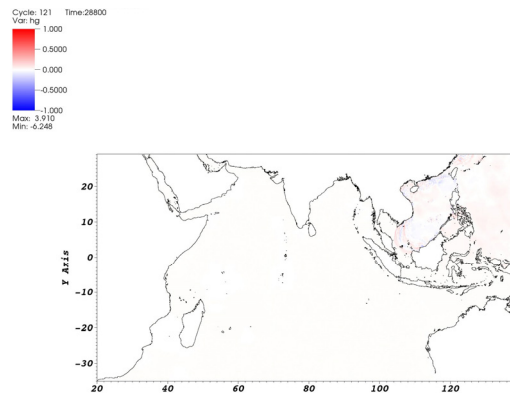
(a)



(c)

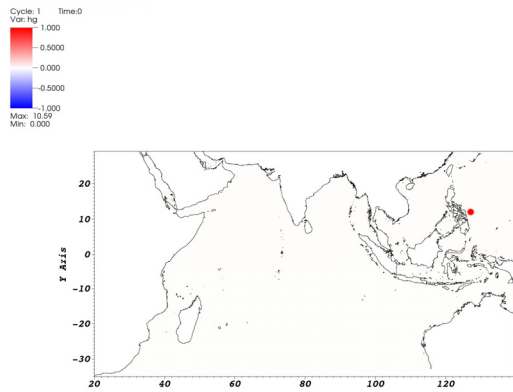


(b)

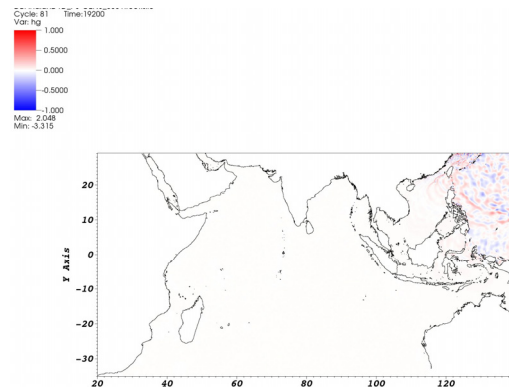


(d)

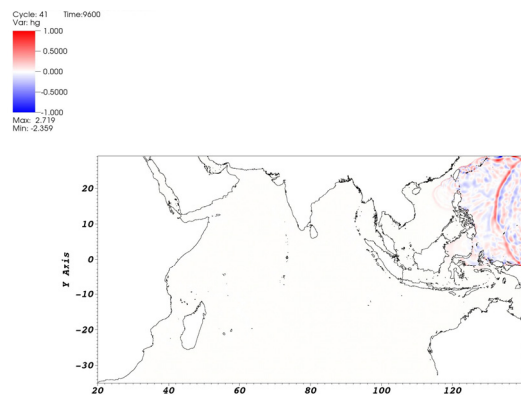
Fig. C.3 Gaussian 3



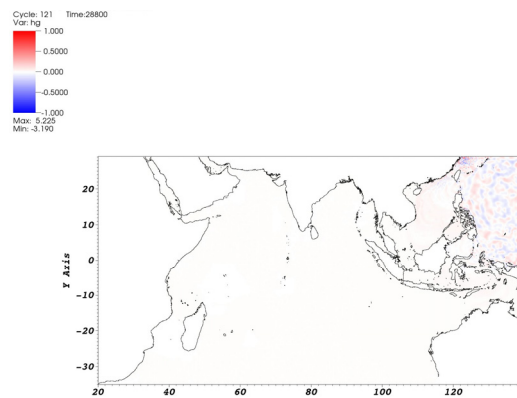
(a)



(c)



(b)



(d)

Fig. C.4 Gaussian 4

2) Fault sources

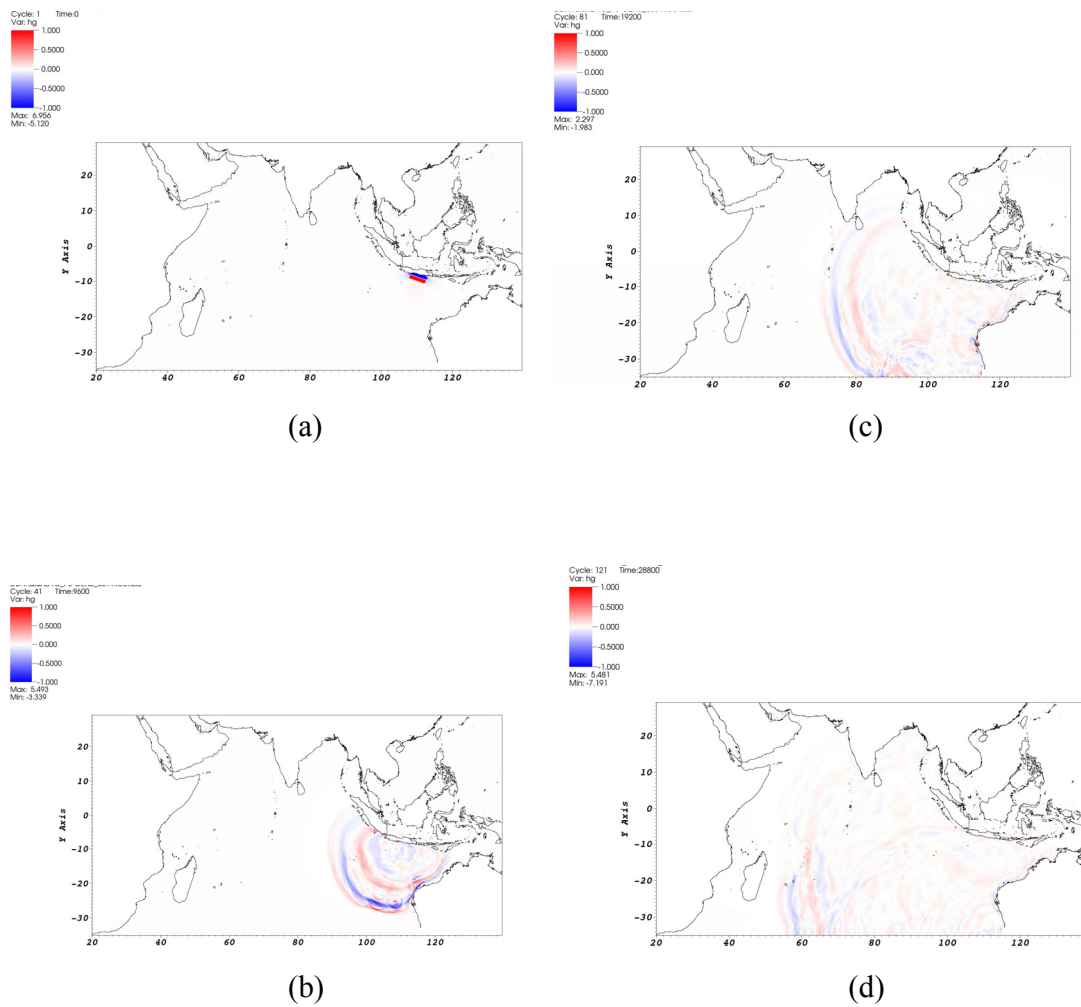
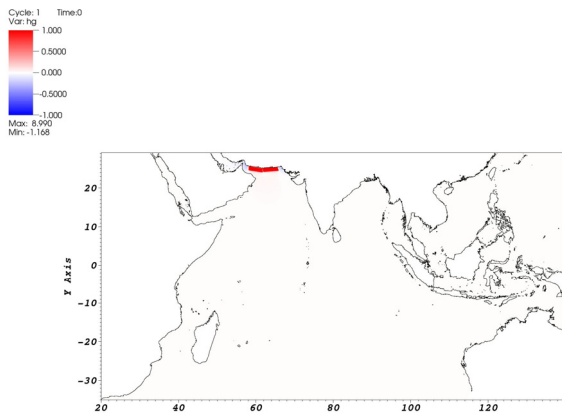
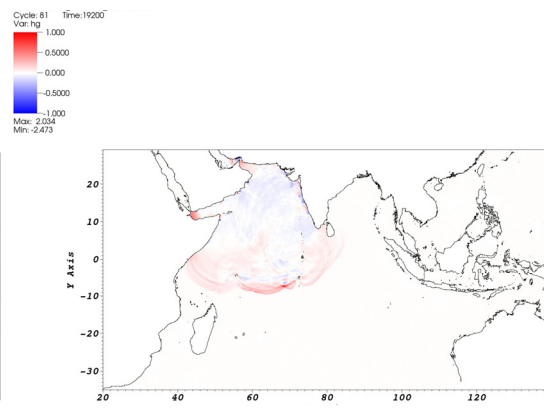


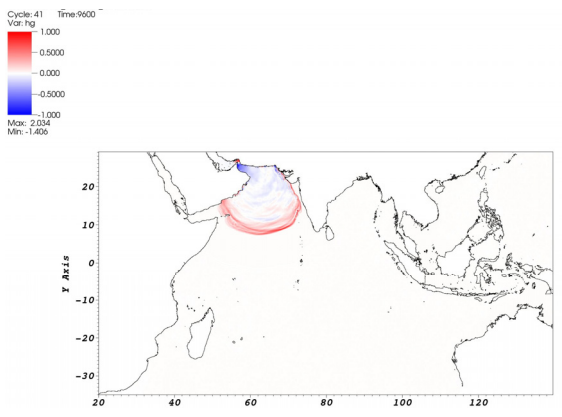
Fig. C.5 JAVA Full Rupture Mw9.0



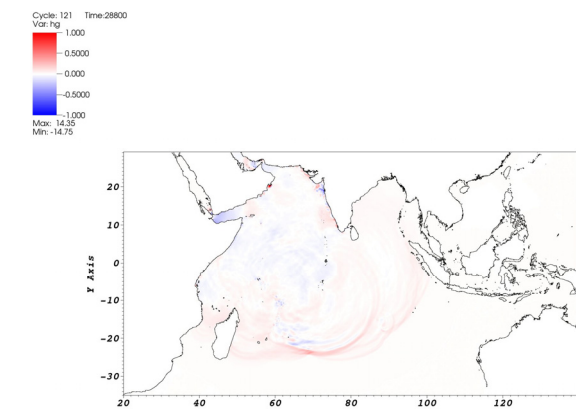
(a)



(c)

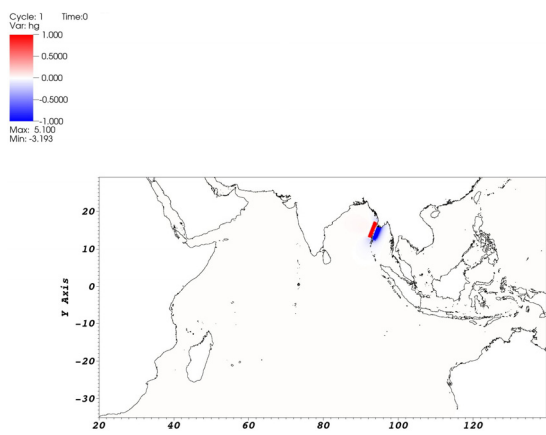


(b)

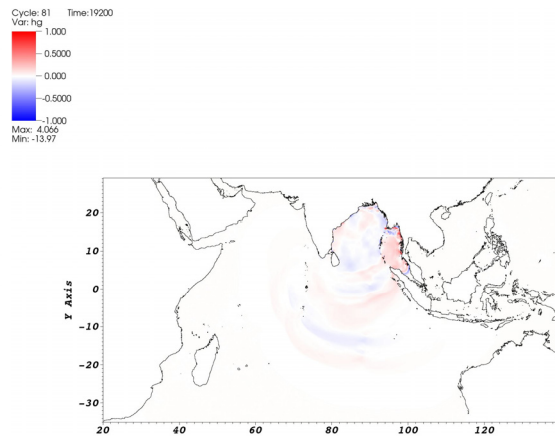


(d)

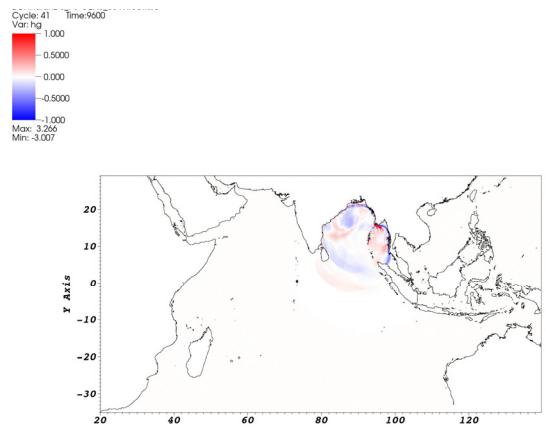
Fig. C.6 MAKRAN Full Rupture Mw 9.2



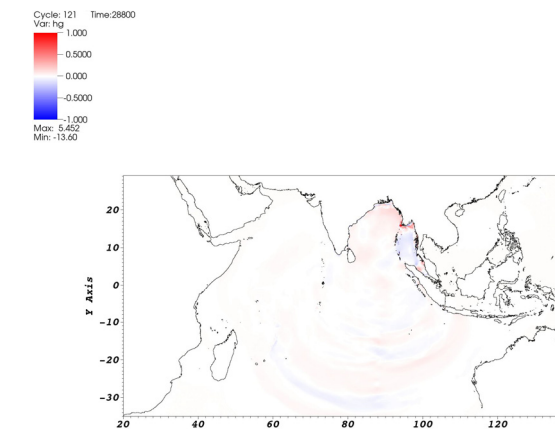
(a)



(c)

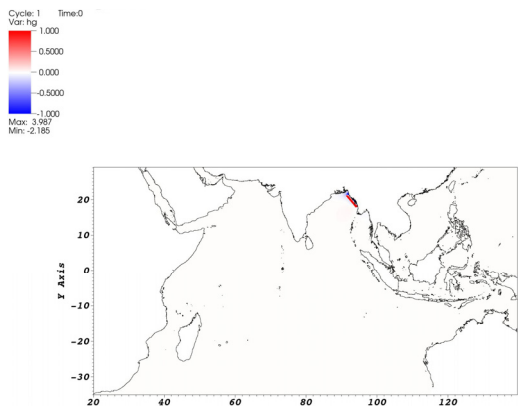


(b)

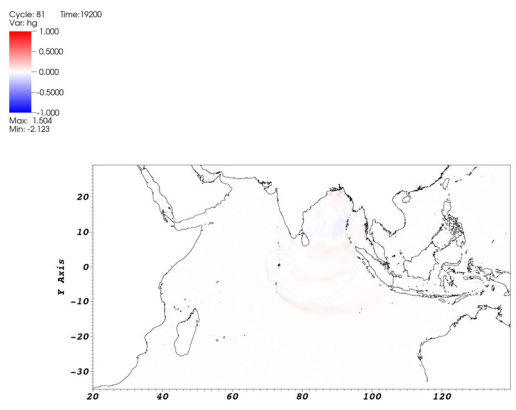


(d)

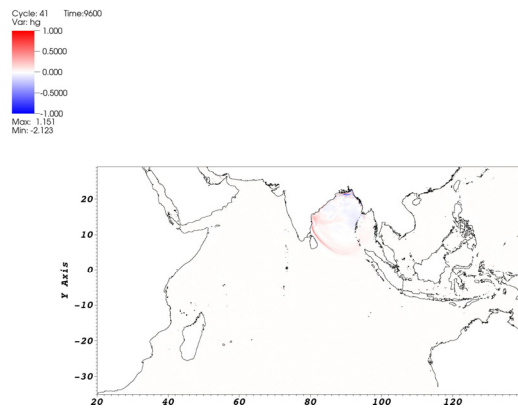
Fig. C.7 NORTH ANDAMAN Leftover 2004 Mw8.9



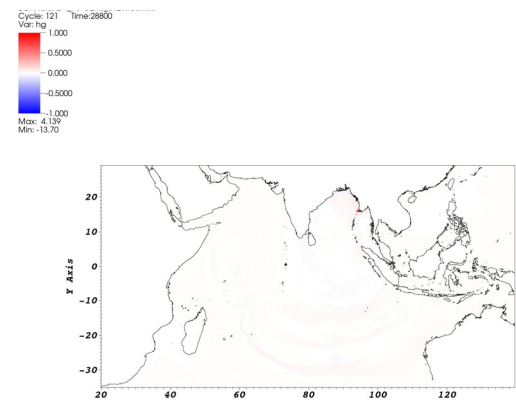
(a)



(c)

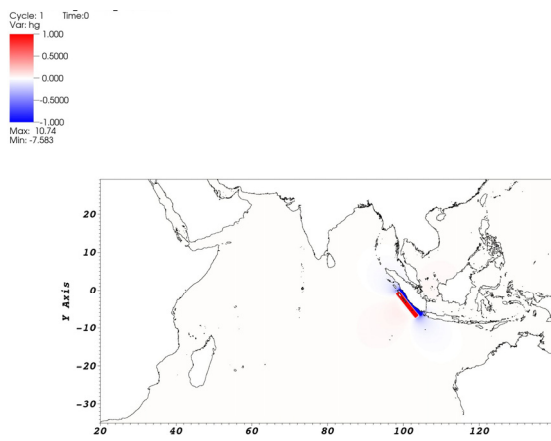


(b)

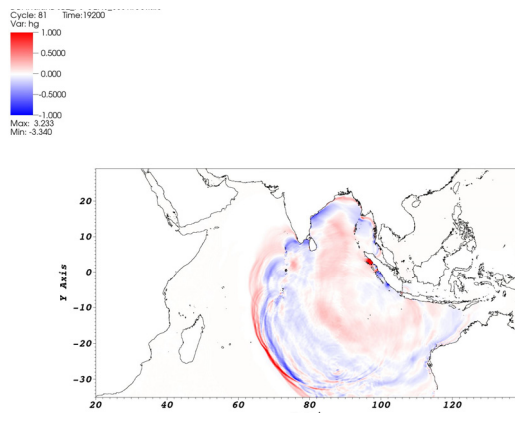


(d)

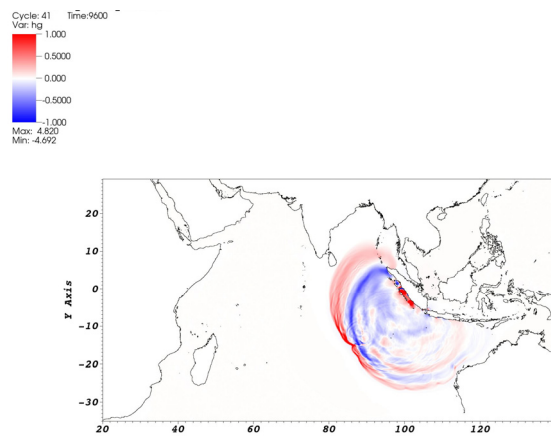
Fig. C.8 NORTH ANDAMAN Mw8.7



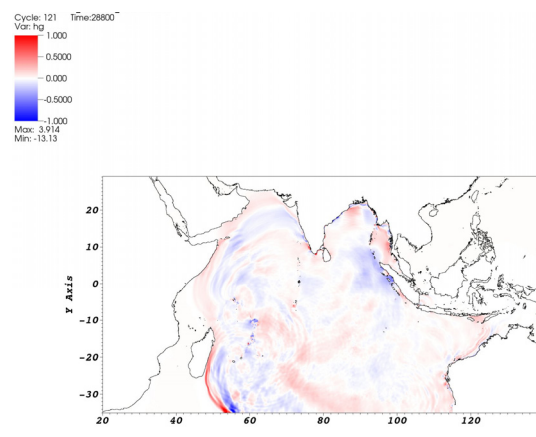
(a)



(c)

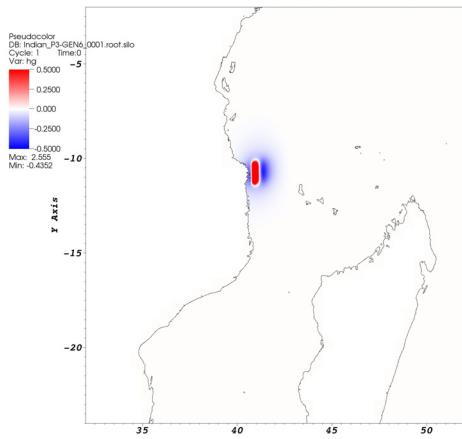


(b)

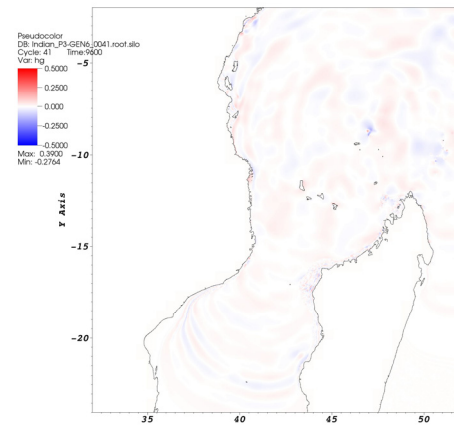


(d)

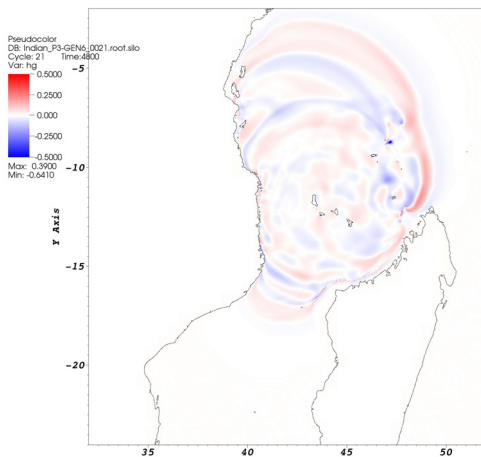
Fig. C.9 SOUTH SUMATRA Mw9.3



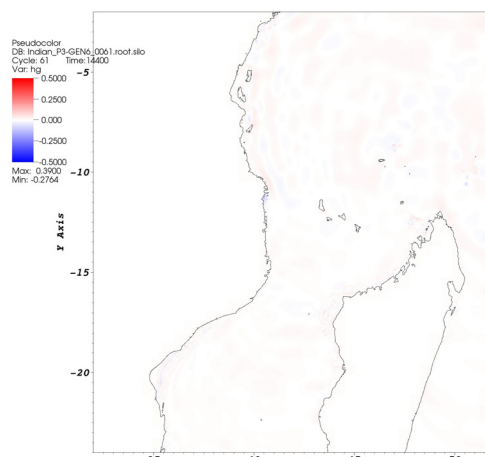
(a)



(c)



(b)



(d)

Fig. C.10 Mozambique Scenario 10 (As explained in [99])

Bibliography

- [1] Nuclear Regulatory Agency Japan, "Enforcement of the New Regulatory Requirements," Tokyo, 2013.
- [2] Tokyo Electric Power Company, "Building diagrams for 1F ground level and cross section," Official Communication, 2014.
- [3] nVIDIA, "CUDA Programming Guide," 2017. [Online]. Available: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz4rP4v1E69>.
- [4] D. Smith, S. Shi and R. Cullinford, "The Holocene Storegga slide tsunami in the United Kingdom," *Quaternary Science Review*, vol. 23, pp. 2291-2321, 2004.
- [5] S. Bondevik, L. F and M. J, "The Storegga Slide tsunami- comparing field observations with numerical simulations," *Marine and Petroleum Geology*, vol. 22, pp. 195-208, 2005.
- [6] T. Bugget and B. R. K. N, "The Storegga slide," *Philosophical Transactions of the Royal Society of London*, vol. Series A 325, pp. 357-388.
- [7] F. Nanayama, S. K and F. R, "Unusually large earthquakes inferred from tsunami deposits along the Kuril trench," *Nature*, vol. 424, pp. 660-663, 2003.
- [8] M. Atwarer, T. Satake, K. Ueda and D. Yamaguchi, "The Orpmane Tsunami of 1700," *USGS Professional Paper*, vol. 1707, p. 133.
- [9] K. Satake, K. Shimazaki, Y. Tsuj and K. Ueda, "Time and site of a giant earthquake in Cascadian inferred from Japanese tsunami records of January 1970," *Nature*, vol. 379, pp. 246-249, 1996.
- [10] K. Satake and Y. Tanioka, "New Guinea earthquake: Mechanism and quantification of unusual tsunami generation," *Pure and Applied Geophysics*, vol. 160, pp. 2087-2118, 2003.
- [11] S. T and R. Fiske, "Krakatau, 1883- the volcanic eruption and its effects," Smithsonian Institution Press, 1983.

- [12] N. Nomanhoy and K. Satake, "Generation mechanism of tsunamis from the 1883 Krakatau euption," *Geophysical Reseach Letters*, vol. 22, pp. 509-512, 1995.
- [13] F. Press and D. Harkrider, "Air-sea waves from the explosion of Krakatoa," *Science*, vol. 154, pp. 1325-1327, 1966.
- [14] E. Pelinsky, C. B, A. Stromkov and H. Kim, "Analysis of tide-gauge records of the 1883 Krakatau tsunami," Springer, 2005.
- [15] H. Kanamori, "Mechanism of tsunami earthquakes," *Physics of the Earth and Planetary Interiors*, vol. 6, pp. 246-259, 1972.
- [16] T. Y and S. K, "Fault parameters for the 1896 Sanriku tsunami earthquake estimated from tsunami numerical modeling," *Geophysical Research Letters*, vol. 23, pp. 1549-1552, 1996.
- [17] J. Lander and P. Lockridge, "United States Tsunamis," National Geophysical Data Center, Boulder, CO, 1989.
- [18] D. Miller, "giant waves in Lituya Bay, Alaska," *USGS Professional paper*, Vols. 354-C, pp. 51-86, 1960.
- [19] N. Shuto, "Numerical simulation of tsunamis- Its present and near future," *Natural Hazards*, vol. 4, pp. 171-191, 1991.
- [20] K. Abe, "Predominance of long periods in large Pacific tsunamis," *Science of Tsunami Hazards*, vol. 18, pp. 15-34, 2000.
- [21] N. Shuto and H. Matsutomi, "Field survey of the 1993 Hokkarido-Nansei-Oki earthquake tsunami," *Pire and Applied Geophysics*, vol. 144, pp. 649-663, 1993.
- [22] T. Takahashi, N. Shuto and F. Imamura, "Source models for the 1993 Hokkaido-Nansei-Oka earthquake tsunami," *Pure and Applied Geophysics*, vol. 144, pp. 747-767, 1995.
- [23] Y. Kawata, B. benson and J. Borrero, "Tsunami in Papua New Guinea was as intense as first thought," *Eos transactions American Geeophysical union*, vol. 80, pp. 101-105, 1999.
- [24] M. Matsuyama, W. J and H. Yeh, "The effect of bathymetry on tsunami characteristics at Sisano Lagoo, Papua New Guinea," *Geophysical Research letters*, vol. 26, pp. 3513-3516, 1999.

- [25] E. Geist, "Origin of the 17 July 1998 Papua New Guinea tsunami; earthquake or landslide?," *Seismological Research*, vol. 71, pp. 344-351, 2000.
- [26] Y. Lay, H. Kanamori and C. Ammon, "The great Sumatra-Andaman earthquake of 26 December 2004," *Science*, vol. 308, pp. 1127-1133, 2005.
- [27] "WHO: World Health Organization," 2013. [Online]. Available: <http://www.who.int/hac/crises/idn/sitreps/en/>.
- [28] N. Mori, Takahashi and T. Y. ., Yasuda, "Survey of 2011 Tohoku earthquake tsunami inundation and run-up," *Geophys. Res. Lett.*, vol. 38, 2011.
- [29] K. Motoki and N. Toshihiro, "Damage statistics (Summary of the 2011 off the Pacific Coast of Tohoku Earthquake damage)," *J Soils and Foundations*, vol. 52, no. 5, pp. 780-792, 2012.
- [30] International Atomic Energy Agency, "The Fukushima Daiichi Accident," IAEA, Austria, 2015.
- [31] W. Hansen, "Theorie zur errechnung des wasserstands und derstromungen in randemeeren," *J Pured and Applied Geophysics*, vol. 8, pp. 287-300, 1956.
- [32] G. Fischer, "Ein numerisches verfahren zur errechnung von windstau und gezeiten in randmeeren," *J of Geophysics*, vol. 11, pp. 60-76, 1959.
- [33] Z. Kowalik and T. S. Murty, Numerical Modeling of Ocean Dynamics. World Scientific, 1993, p. 481.
- [34] F. Imamura, Review of tsunami simulation, Word Scientific Publishing Co, pp. 25-42.
- [35] F. Imamura, C. Goto, Y. Ogawa and N. Shuto, "Numerical Method of Tsunami Simulation with the Leap-Frog Scheme," IUGG/IOC Time Project Manuals, 1995.
- [36] D. Nicolsky, E. Sileimani and R. Hansen, "Validation and verification of a numerical model for tsunami propagation and runup," *J Pure and Applied Geophysics*, vol. 168, no. 6, pp. 1199-1222, 2011.
- [37] V. Titov and C. Synolakis, "Evolution and runup of the breaking and nonbreaking waves using VTSC2," vol. 126, no. 6, pp. 308-316, 1995.
- [38] D. Burwell, E. Tolkova and A. Chawla, "Diffusion and dispersion characterization of a numerical tsunami model," *Ocean Modelling*, vol. 19, no. 1-2, pp. 10-30, 2007.

- [39] M. Berger and R. LeVeque, "Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems," *SIAM J. Numer. Anal.*, vol. 35, pp. 2298-2316, 1998.
- [40] D. Wang, N. C. Becker, D. Walsh, G. J. Fryer, S. A. Weinstein, C. S. McCreery, V. Sardina, V. Hsu, B. F. Hirshorn, G. P. Hayes, Z. Duputel, L. Rivera, H. Kanamori, K. Koyangai and B. Shiro, "Real-time forecasting of the April 11, 2012, Sumatra Tsunami," *Geophys. Res. Lett.*, vol. 39, no. 19, p. L19601, 2012.
- [41] A. Babeyko, "Fast Tsunami Simulation Tool for Early Warning," [Online]. Available: <https://docs.gempa.de/toast/current/apps/easywave.html>. [Accessed 2017].
- [42] D. Peregrine, "Long waves on a beach," *Journal of Fluid Mechanics*, vol. 27, no. 4, 1967.
- [43] O. Nwogu, "An alternative form of the Boussinesq equations for nearshore wave propagation," *Coastal, and Ocean Engineering*, vol. 119, pp. 618-638, 1993.
- [44] P. Lynett, T. Wu and P. Lui, "Modeling wave runup with depth-integrated equations," *Coastal Engineering*, vol. 46, no. 2, pp. 89-107, 2002.
- [45] F. Shi, J. T. Kirby, J. D. Geiman and S. Grilli, "A high-order adaptive time-stepping TVD solver for," *Ocean Modeling*, vol. 43, pp. 36-51, 2012.
- [46] G. Wei, J. Kirby, S. T. Grilli and R. Subramanya, "Fully nonlinear Boussinesq model for free surface waves. Part 1: Highly nonlinear unsteady waves," *J Fluid Mech*, vol. 294, pp. 71-92, 1995.
- [47] V. Roeber and K. F. Cheung, "Boussinesq-type model for energetic breaking waves in fringing reef environments," *Coastal Engineering*, pp. 1-20, 2012.
- [48] Y. Zhang and A. Baptista, "Aun efficient and robust tsunami model on unstructured grids," *Pure and Applied Geophysics*, vol. 165, pp. 2229-2248, 2008.
- [49] S. D. Abadie, S. Morichon, S. Grilli and S. Glockner, "Numerical simulation of waves generated by landslides using a multiple-fluid Navier–Stokes model," *Coastal Engineering*, vol. 57, no. 9, pp. 779-794.
- [50] J. Horrillo, G. Wood, B. Kim and A. Parambath, "A simplified 3-D Navier–Stokes numerical model for landslide tsunamis: Application to the Gulf of Mexico," *J Geophysics Res Oceans*, vol. 118, pp. 6934-6950.

- [51] S. D. Abadie, J. C. Harris, S. Grilli and R. Fabre, "Numerical modeling of tsunami waves generated by the flank collapse of the Cumbre Vieja Volcano (La Palma, Canary Islands) : Tsunami source and near field effects," *J Geophys. Res.*, vol. 117, p. C05030, 2012.
- [52] "RIMES: Regional Integrated Multi-hazard Early Warning System," [Online]. Available: <http://www.rimes.int/>. [Accessed August 2017].
- [53] D. S. Drumheller, "Introduction to Wave Propagation in Nonlinear Fluids and Solids," *J. Acoust. Soc. Am.*, vol. 111, no. 3, pp. 1142-1154, 2002.
- [54] F. Toro, Shock-capturing methods for free-surface shallow flows, London: John Wiley&Sons Ltd, 2010.
- [55] A. Bermúdez and M. Vázquez, "Upwind methods for hyperbolic conservation laws," *Comput Fluids* , vol. 8, pp. 1049-1071, 1994.
- [56] R. LeVeque, "Balancing source terms and flux gradients in high-resolution Godunov methods: the quasi-steady wave-propagation algorithm," *J. Comput. Phys.*, pp. 346-365, 1998.
- [57] D. Williamson, J. Drake, J. Hack, R. Jakob and P. Swarztraube, "A standard test set for numerical approximations to the," *J. Comput. Phys*, vol. 102, pp. 211-224, 1992.
- [58] P. Swarztrauber, D. Williamson and J. Drake, "The cartesian method for solving partial differential equations in spherical," *Dyn. Atmos. Oceans*, no. 27, pp. 679-706, 1997.
- [59] A. Staniforth and J. Cote, "Semi-Lagrangian integration schemes for atmospheric models – a review," *Mon. Weather Rev.* , vol. 119, pp. 2206-2223, 1991.
- [60] A. Zhukov, "Application of the method of characteristics to the numerical solution of one-dimensional problems of gas dynamics," *Trudy Mat. Inst. Steklov*, vol. 58, 1960.
- [61] V. Babu, Fundamentals of Gas Dynamics, Wiley, 2014.
- [62] V. Rusanov, "Characteristics of the general equations of gas dynamics," *Zhurnal Vychislistelnoi Matematiki Matematicheskoi Fiziki*, vol. 3, pp. 508-527, 1963.
- [63] T. Nakamura, R. Tanaka, T. Yabe and K. Takizawa, "Exactly conservative semi-Lagrangian scheme for multi-dimensional hyperbolic equations with directional splitting technique," *J. Comput. Phys*, vol. 174, pp. 171-207, 2001.

- [64] Y. Ogata and T. Yabe, "Multi-Dimensional Semi-Lagrangian Characteristic Approach to the Shallow Water Equations by the CIP Method," *International Journal of Computational Engineering Science*, vol. 05, no. 03, 2004.
- [65] J. J. Stoker, *Water Waves: The Mathematical Theory with Applications*, Wiley-Interscience, 1992.
- [66] T. Yabe and T. Aoki, "A universal solver for hyperbolic equations by Cubic-Polynomial Interpolation I. One-dimensional solver," *Comp. Physic Comm*, vol. 66, pp. 219-232, 1991.
- [67] T. Yabe, R. Tanaka, T. Nakamura and F. Xiao, "An Exactly Conservative Semi-Lagrangian Scheme (CIP-CSL) in One Dimension," *Monthly Weather Rev.*, vol. 129, pp. 332-344, 2001.
- [68] T. Utsumi, T. Kunugi and T. Aoki, "Stability and accuracy of the cubic interpolated propagation scheme," *J Comp. Phys*, vol. 101, no. 9, 1997.
- [69] J. T. Kirby, S. Fengyan, T. Babak, J. C. Harrisb and T. G. Stephan, "Dispersive tsunami waves in the ocean: Model equations and sensitivity to dispersion and Coriolis effects," *Ocean Modelling*, vol. 62, pp. 39-55, 2013.
- [70] Z. Kowalik, W. Knight, T. Logan and P. Whitmore, "Numerical Modeling of the global tsunami: Indonesian tsunami of 2004," *Science of Tsunami Hazards*, vol. 23, no. 1, pp. 40-45, 2005.
- [71] F. Lovholt, G. Pedersen and G. Gisler, "Oceanic propagation of a potential tsunami from the La Palma Island," *J. Geophysics Res*, vol. 114, 2008.
- [72] A. Sugiyama, T. Aoki and K. Honda, "A stable and higher-order computation for tsunami inundation using shallow water model," in *第20回計算工学講演会*, Tsukuba, 2015.
- [73] A. Sugiyama, T. Aoki and K. Honda, "インド洋に面した地域の津波ハザード・シミュレーションII -高精度スキームによる遡上シミュレーション-", in *第28回数値流体力学シンポジウム*, 東京, 2014.
- [74] J. G. Zhou, D. M. Causon, C. G. Mingham and I. D. M., "The surface gradient method for the treatment of source terms in the shallow-water equations," *J Comp. Physics*, vol. 168, pp. 1-52, 2001.

- [75] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31, Cambridge University Press, 2002.
- [76] W. C. Thacker, "Some exact solutions to the nonlinear shallow-water wave equations," *J. Fluid Mechanics*, vol. 107, pp. 499-508, 1981.
- [77] J. Johnston Stoker, *Water waves: The mathematical theory with applications*, John Wiley & Sons, Inc, 2011.
- [78] L. Smylie and D. Mansinha, "The Displacement Fields of Inclined Faults," *B. Seismological Soc. Ame.*, vol. 61, no. 5, pp. 1433-1440, 1971.
- [79] M. Berger and J. Olinger, "Adaptive mesh refinement for hyperbolic partial differential equations," *J Comp. Physics*, vol. 53, pp. 484-512, 1984.
- [80] M. Colella and P. Berger, "Local Adaptive Mesh Refinement for Shock Hydrodynamics," *J. Comp. Physics*, vol. 82, pp. 64-84.
- [81] S. Fedkiw and R. Osher, *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag, 2003.
- [82] L. Doron, G. Puppo and G. Russo, "A third order central WENO scheme for 2D conservation laws," *Applied Numerical Mathematics*, vol. 33, no. 1-4415-421, 2000.
- [83] T. Moller, N. Hoffman and E. Haine, *Real-Time Rendering*, AK Peters Ltd, 1999.
- [84] S. Gottschalk, L. Ming and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection.," 1996.
- [85] G. Szauer, *Game Physics Cookbook*, Amazon Digital Services , 2017.
- [86] dyn4j, "SAT (Separating Axis Theorem)," [Online]. Available: <http://www.dyn4j.org/2010/01/sat/>.
- [87] The General Bathymetric Chart of the Oceans (GEBCO) , "GEBCO," 2017. [Online]. Available: <http://www.gebco.net/>.
- [88] "CUDA Zone," [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed 2017].
- [89] H. Sagan, *Space-Filling Curves*, Universitext, 1994.
- [90] D. Reed, "User Datagram Protocol INTERNET STANDARD," RFC 768 , 1980.

- [91] D. Clark, "The design philosophy of the DARPA internet protocols," in *SIGCOMM '88 Symposium proceedings on Communications architectures and protocols*, 1988.
- [92] "Silo User's Guide," LLNL, 2017. [Online]. Available: <https://wci.llnl.gov/codes/silo/media/pdf/LLNL-SM-453191.pdf>.
- [93] "nVIDIA Tesla K40 Manual," [Online]. Available: http://www.nvidia.co.jp/content/PDF/kepler/Tesla-K40-Active-Board-Spec-BD-06949-001_v03.pdf. [Accessed 2017].
- [94] NOAA, "Tsunami Event - The Indian Ocean Tsunami, December 26, 2004," 2017. [Online]. Available: http://nctr.pmel.noaa.gov/indo_1204.html.
- [95] NOAA Center for Tsunami Research, "Deep-ocean Assessment and Reporting of Tsunamis," 2017. [Online]. Available: <http://nctr.pmel.noaa.gov/Dart/>.
- [96] M. Dao and P. Tkalich, "Tsunami Propagation modelling - a sensitivity study," *Natural Hazards and Earth System Sciences*, vol. 7, pp. 741-754, 2007.
- [97] S. Grilli, M. Ioualalen, J. Asavanant, J. Shi, T. Kirby and P. Watts, "Source Constraints and Model Simulation of the December 26, 2004 Indian Ocean Tsunami," *Port, Ocean and Coastal Engineering*, vol. 133, no. 6, pp. 414-428, 2007.
- [98] RIMES, "Tsunami Hazard and Risk Assessment and Evacuation Planning - Hambantota, Sri Lanka," August 2014.
- [99] A. Aramuge and Y. Fujii, "Tsunami hazard assessment in Mozambique coast," *Bulletin of the International Institute of Seismology and Earthquake Engineering*, vol. 44, pp. 19-24, 2010.
- [100] Geological Survey of Canada, "Effects of the 26 december 2004 Indian Ocean Tsunami in the Republic of Seychelles," UNESCO, 2005.
- [101] A. Supparsi, S. Koshimura and F. Imamura, "Developing tsunami fragility curves based on the satellite remote sensing and the numerical modeling of the 2004 Indian Ocean tsunami in Thailand," *J Natural Hazards and Earth Sc.*, vol. 11, pp. 173-189, 2011.
- [102] C. H. G, "Morphometric analysis in geographic information systems: applications of free software GRASS and R," *Computers & Geosciences*, vol. 30, no. 9, pp. 1055-1067, 2004.

- [103] M. Arce Acuna and T. Aoki, "Real-Time Tsunami Simulation Accelerated by GPU; Master Thesis," Tokyo, 2009.
- [104] nVIDIA, "Tesla K40 GPU Boost Application Note," 2017. [Online]. Available: <https://www.nvidia.com/content/PDF/kepler/nvidia-gpu-boost-tesla-k40-06767-001-v02.pdf>.

List of Publications

Refereed Journals

- [1] Marlon Arce Acuña, Takayuki Aoki, "Multi-GPU Computing and Scalability for Real-Time Tsunami Simulation", 3 reviewers. HPCS 2010, IPSJ Symposium Series Vol 2010. No 1. ISSN 1344-6040
- [2] Marlon Arce Acuña, Takayuki Aoki, "Tree-based Mesh-Refined GPU Accelerated Tsunami Simulator for Real Time Operation", Computers & Fluids, 2017 (Pending to submission)

International Conferences

- [1] Marlon Arce Acuña, Takayuki Aoki, "AMR Multi-GPU Accelerated Tsunami Simulation", The 32th Annual Conference. International Conference on Simulation Technology, JSST 2013
- [2] Marlon Arce Acuña, Takayuki Aoki, "Multi-GPU Tsunami Simulation on TSUBAME GPU Supercomputer". SIAM Conference on Mathematical & Computational Issues in the Geosciences, March 21-24, 2011. Long Beach CA, USA
- [3] Marlon Arce Acuña, Takayuki Aoki, "Real-time Tsunami Simulation accelerated by GPU" JOINT CONFERENCE 7th International Conference on Urban Earthquake Engineering (7CUEE) & 5th International Conference on Earthquake Engineering (5ICEE), March 3-5, 2010, Tokyo Institute of Technology, Tokyo, Japan
- [4] Marlon Arce Acuña, Takayuki Aoki, "Real-Time Tsunami Simulation Solving the Shallow Water Equations on Multi-Node GPU Cluster", ETHZ - Tokyo Tech

Workshop : Computing with GPUs, Cells, and Multicores, May 10-11, 2009. Zurich, Switzerland

- [5] Marlon Arce Acuña, Takayuki Aoki, "Real-Time Tsunami Solving the Shallow Water Equations on Multi-Node GPU Cluster", WCCM/APCOM, July 2010, Sydney Australia
- [6] Marlon Arce Acuña, Takayuki Aoki, "Multi-GPU Computing and Scalability for Real-Time Tsunami Simulation", GSIC International Workshop on GPGPU Applications, January 26, 2010. Tokyo
- [7] Marlon Arce Acuña, Takayuki Aoki, "Real-Time Tsunami Simulation on a Multi-Node GPU Cluster", Poster Presentation SC09, November 14-20, 2009 Portland Oregon, USA

Domestic Conferences

- [1] Marlon Arce Acuña, Takayuki Aoki, "Large-scale Mesh-Refined Multi-GPU Accelerated Tsunami Simulation on a Real Indian Ocean Scenario"; Student Presentation Award; 22nd Computational Engineering Conference, Omiya, May 31-June 2 JSCES 2017
- [2] Marlon Arce Acuña, Takayuki Aoki. "Large Scale AMR Multi-GPU Tsunami Simulation" 13-16 April, Compsafe 2014, Sendai
- [3] Marlon Arce Acuña, Takayuki Aoki, Kiyoshi Honda, "Tsunami Hazard Simulation in Indian Ocean Coasts I - Wide-area Simulation with Fine Mesh Adaptation" The 28th Computational Fluid Dynamics Symposium, 2014, Tokyo
- [4] Marlon Arce Acuña, Takayuki Aoki, "Large-Scale GPU Tsunami Simulation on a Multi-Level Mesh" IAM Symposium: Analysis on marine renewable energy dynamics and marine environment dynamics Dec. 16 -17 (Saturday), 2011, Fukuoka
- [5] Marlon Arce Acuña, Takayuki Aoki, "Mesh Refinement for Real-Time Tsunami Simulation", 第 16 回計算工学講演会, May 25-27, 2011, Chiba
- [6] Marlon Arce Acuña, Takayuki Aoki, "Multi-node GPU Real-Time Tsunami Simulation for Large Scale and Actual Study Case", 第 24 回数値流体力学シンポジウム, December 20-22, 2010. Tokyo

- [7] Marlon Arce Acuna, Takayuki Aoki, "Parallel GPU Computing for Real-Time Tsunami Simulation on an Actual Study Case"; 第 15 回 計算工学講演会, May 25-28, 2010. Fukuoka
- [8] Marlon Arce Acuna, Takayuki Aoki, "Multi-GPU Computing and Scalability for Real-Time Tsunami Simulation", HPCS 2010, IPSJ, January 14-15, 2010. Tokyo
- [9] Marlon Arce Acuna, Takayuki Aoki, "Real-time Tsunami Simulation Accelerated by Parallel GPUs"; JSME Fellow Award for Outstanding Young Engineers; 日本機械学会 第 22 回計算力学講演会 CMD2009, Kanazawa, Japan.
- [10] Marlon Arce Acuna, Takayuki Aoki, "Large-scale Real-Time Tsunami Simulation on Multi-node GPU Cluster", CFD2009. 第 23 回 数値流体力学シンポジウム, December 16-18, 2009 Sendai, Japan
- [11] Marlon Arce Acuna, Takayuki Aoki, "Real-Time Tsunami Simulation on Multi-node GPU Cluster"; Grand Prize Best Poster Award; Next-Generation Supercomputing Symposium 2009, October 7-8, 2009, Tokyo

Honors and Awards

- [1] Student Presentation Award. The 32th Annual Conference. International Conference on Simulation Technology, JSST 2013
- [2] JSME Fellow Award for Outstanding Young Engineers, Japanese Society of Mechanical Engineering, 2010
- [3] Grand Prize Best Poster Award, RIKEN Next-Generation Supercomputing Symposium 2009. Tokyo, Japan
- [4] Best Paper Presentation Award, Master Degree Presentation, 2009, Department of Nuclear Engineering, Tokyo Institute of Technology