Implementation of a Tumor Growth Prediction System in Reconfigurable Logic

Konstantinos Malavazos

Department of Electrical and Computer Engineering
Technical University of Crete

This dissertation is submitted for the degree of
Master of Science

Committee Members
Professor Ioannis Papaefstathiou
Professor Apostolos Dollas
Professor Michail Zervakis

July 2018

Declaration

I hereby declare that this thesis presented for the degree of Master of Science, has been composed solely by myself and that it has not been submitted, in whole or in part for consideration for any other degree or qualification in this, or any other university. Except where states otherwise by reference or acknowledgment, the work presented is entirely my own.

Konstantinos Malavazos
July 2018

Acknowledgements

I would like to acknowledge the Thesis Supervisor Professor Ioannis Papaefstathiou for giving me the opportunity to expand my knowledge on hardware and FPGA designing. The door to Prof. Papaefstathiou office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this Thesis to be my own work, but steered me in the right direction whenever he thought I needed it. Also, I would like to thank Prof. Apostolos Dollas, Prof. Michalis Zervakis and Prof. Dionisios Pnevmatikatos for their constructive comments on my work in this Thesis and for their participation in the Committee. Furthermore, I would like to thank PhD Candidate Maria Papadogiorgaki for introducing me to the topic, providing me the Model application's codes, which are her intellectual property, as well for her support on the way. Moreover, I would like to thank all my labmates in Microprocessor and Hardware Lab (MHL) and especially PhD Candidates Pavlos Malakonakis, who was always there to push me with his great ideas and finally, Andreas Brokalakis, system administrator of MHL, who was always willing to help me extract important information about these machines.

Abstract

In the last few years, the biomedical community is increasingly taking advantage of the increasing computational power, both to manage and analyze data and to model biological processes. Biomedical software applications usually require significant computational power, especially when they include the processing and analysis of large amounts of data, such as medical image sequences. This Master Thesis targets the acceleration of three different mathematical models, which are developed in the Technical University of Crete to model and predict the spatio-temporal evolution of glioma, using Reconfigurable Logic. Glioma is a rapidly evolving type of brain cancer, well known for its aggressive and diffusive behavior. The modeling applications presented in this Thesis fall under the category of multi-compartmental continuum approaches and aim to simulate the spatio-temporal evolution of a glioma tumor in an isotropic and heterogeneous brain tissue, which consists of different anatomic structures. The first model is the Oxygen-Glucose Diffusion-Proliferation 2D Model, simulating the proliferative cells and the necrotic core as a result of hypoxic and hypoglycemic-cells death, in single MRI images. The second is the Simple Diffusion-Proliferation 3D Model, which simulates only the proliferative cells in a sequence of MRI slices. The last is the Oxygen-Glucose Diffusion-Proliferation 3D Model, which simulates the same behaviors of the glioma as the 2D Model, in a sequence of MRI slices. The hardware acceleration is achieved using the Trenz platform, model TE0808 UltraSOM, which consists of a Xilinx Zynq UltraScale+ FPGA and an ARM Cortex A-53. The FPGA implementations of these Models are compared with the corresponding OpenMP software implementations on two different high-end Server systems, with up to 40 threads (Hyper-Threading). The results showed that the FPGA accelerators achieved runtime speedup and are up to 14 times more power efficient.

Table of contents

List of figures

List of tables

List of Plots

Nomenclature

Acronyms / Abbreviations

AXI	Advanced eXtensible Interface
CPU	Central Processing Unit
DDR SDRAM	Double Data Rate Synchronous Dynamic Random-Access Memory
DM	DataMover
FPGA	Field-Programmable Gate Array
GPU	Graphical Processing Unit
HP	High Performance
JTAG	Joint Test Action Group
PL	Programmable Logic
PS	Processing System
SoC	System on Chip
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus

Chapter 1

Introduction

Biomedical software applications usually require a significant computational cost, especially when they include the processing and analysis of large amounts of data, such as medical image sequences. Thus, those applications may spend a lot of time in order to provide final results, in many cases even days. To this respect, the hardware implementation of particular biomedical applications would allow the reduction of their execution time aiming to accelerate the generation of the corresponding results. According to the World Health Organization, cancer is the second leading cause of death in developed countries and is among the three leading causes of death for adults in developing countries (WHO). A steadily increasing proportion of elderly people in the world will result in approximately 16 million new cases of cancer by the year 2020 (IARC). For that reason, a significant research is being conducted at the moment in order to investigate the mechanisms and invasion of tumor, with the aim of predicting its evolution.

1.1 Motivation

This Thesis was conceived in order to accelerate three mathematical models, in Reconfigurable Logic, which are developed by Post Master research fellow, in Technical University of Crete, Maria Papadogiorgaki. These models are targeting in modeling and predicting the spatio-temporal evolution of glioma. The identification of cancer characteristics and the prediction of tumor growth can lead to useful insight into the disease dynamics, which may improve clinical outcomes. In particular, glioma is a rapidly evolving type of brain cancer, well known for its aggressive and diffusive behavior [1] that has lead several research efforts to explore its progression. Along this direction, several mathematical models have been developed investigating the mechanisms that govern glioma tumors' evolution, with the aim of predicting their future spatial and temporal evolution [2]. Such models attempt to identify the interactions between cancer cells and tissue microenvironment, which play an important role in the mechanism of the tumor formation and progression. The mathematical models for tumor prediction, which are being developed, are very computational and resource costly. The FPGAs offer the computational power and the low power consumption that is needed for that kind of applications, making them one of the most efficient platforms.

1.2 Thesis Contributions

This thesis presents a novel work that achieved the acceleration of these very computationally demanding models. However, although several cancer-progression modeling algorithms are computationally efficient when they are implemented in software, they are still time consuming, requiring considerable execution intervals. Aiming to minimize the execution time of specific glioma-growth modeling approaches [3], [4], this thesis deals with the hardware implementation of the models applied on realistic medical images. These models have already been implemented in software and run on CPU. Due to the nature of the algorithms, the software applications run many hours in order to simulate the glioma progression for some days. The software applications were initially run on an Intel Core 2 Duo CPU @ 2.8 GHz. In this thesis, the models run on two Server Systems named Zeus and Kronos, respectively. Zeus has two Intel Xeon E5-2430 v2 @ 2.8 GHz, 15 MB SmartCache and 64 GB RAM. Each Xeon CPU has 6 cores and 12 physical threads (hyper-threading). The second server, Kronos, is a much more powerful system, having two Intel Xeon E5-2630 v4 @ 2.2 GHz, 25 MB Smartcache and 125 Gigabytes RAM. Each Xeon processor has 10 physical threads and 20 Hyperthreading. In total 40 Hyperthreading threads. The platform that is used for acceleration is Trenz TE0808 UltraSOM, having Xilinx Zynq UltraScale+, which is a SoC system with an ARM Cortex A53 and an UltraScale+ FPGA. The FPGA accelerator, for the most demanding Model, achieved the significant speed up of 1.85x over the high-end Server System (Kronos) with 40 parallel threads (Intel Hyper-Threading) and it is 14x more power-efficient. The achieved acceleration of these kind of algorithms helps the further research on tumor prediction, as it makes it computational and power feasible.

1.3 Thesis Outline

This thesis is separated in five different chapters. The second chapter describes the theoretical background of the mathematical models and the software implementation each and it also presents some previous related work. The third chapter describes extensively the implementation in hardware, the difficulties and the problems that were overcame, in order to achieve the final result. There are also detailed images, that describe and illustrate the main ideas, which were used to implement the hardware accelerators. In the fourth chapter, the experiments and the results are described and evaluated. The fifth and final chapter summarizes previous chapters and concludes in final results. In the same chapter, there is also the future work.

Chapter 2

Theoretical Background

2.1 Related Work

In the last years, the biomedical applications and their integration in hardware platforms for acceleration, are taking on an increasingly critical role in research as their algorithms are becoming more and more computational intensive. The medical imaging of the interior of a body for clinical analysis and medical intervention consists of processing images, a procedure very computationally intensive. Some of these applications even consist real-time analysis of image-data. Such work is that of Richard M. Jiang and Danny Crookes [5]. They implemented a novel area-efficient high-throughput 3D DWT architecture. Medical imaging systems or telemedicine applications require real-time speed in their image compression and multi-resolution analysis. The 3D discrete wavelet transform (DWT) is a widely applied method for these medical applications because of its perfect reconstruction property and lack of blocking artifacts. They used VHDL and mapped their design in a Xilinx Virtex-E FPGA. Their final implementation consists of a low area cost and can run up to 85 MHz, which can perform a five-level 3D wavelet analysis for seven 128×128×128 volume images per second.

The image-guided interventions (IGI) are medical procedures that use computer-based systems to provide virtual image overlays to help the physician precisely visualize and target the surgical site. This field has been greatly expanded by the advances in medical imaging and computing power over the past 20 years [6]. The low-dose computed tomography (CT) and 3D ultrasound to provide, real-time accurate anatomical information intra-operatively, is one emerging trend in IGI work-flow. The images, however, are characterized by quantum (in low-dose CT) or speckle (in ultrasound) noise. The work of Dandekar et al. [7], is a FPGA approach for real-time preprocessing of intra-operative 3D images. They implemented in hardware the anisotropic diffusion filtering and median filtering, that have been shown to be effective in enhancing and improving the visual quality of these images. They compared their hardware design, in an Altera Stratix-II, with a software implementation on an Intel Xeon @3.6 GHz workstation with 2 Gigabytes of DDR2. They achieved 208x Voxel processing rate (MHz) over the software implementation for the 3D anisotropic diffusion filtering, and 74x for the 3D median filtering.

A new non-invasive method, that promises early diagnosis breast cancer before metastases, has been developed. Three-dimensional ultrasound computer tomography (3D USCT) promises high-quality images of the breast, but is currently limited by a time-consuming synthetic aperture focusing technique based image reconstruction. The work of M. Birk et al. [8] targets in accelerating the image reconstruction by a GPU, and by the FPGAs embedded in a custom data acquisition system. They compared their results with a multi-core CPU and show that both platforms are able to accelerate processing. The basic idea is to accumulate many low-quality images, recorded by transducers at different geometric positions in order to create one high resolution image. This procedure is Synthetic Aperture Focusing Technique (SAFT) and targets to compute up to 1024^3 discrete grid of voxels (volume pixels). The CPU-based implementation used an Intel Core i7-920 @2.67 GHz and a DDR3 memory. The CUDA implementation was tested on a Nvidia Geforce GTX 580 and the hardware implementation on Custom Data Acquisition (DAQ) system with 81 Altera Cyclone II FPGAs. The final results showed that the multi-threaded software, on the Intel Core I7 machine, using 8 threads and Hyper-threading, achieved 1.02 GVoxel/s. The FPGA achieved 1.6 GVoxels/s and the GPU up to 8.2 GVoxels/s, making the GPU the better-performing platform, by far.

Another method for Breast Cancer Detection is the Microwave Imaging (MI). In the work of Pagliari et al. [9] two alternative methods for breast cancer MI, were examined. They used HLS tools to accelerate the critical sections of these corresponding imaging algorithms. The first method, *MIST Beamforming*, requires the execution of a set of simple operations on a massive amount of input data. The second, *MUSIC-Inspired (MUSIC-I)*, processes a relatively smaller amount of data but requires the execution of a set of operations that are significantly more complex. As they observed, the performance of the corresponding accelerator-based implementations are likely communication bound for MIST Beamforming and computation bound for MUSIC-I. They used SystemC for implementing a FPGA and an ASIC design. The target device for the FPGA is Xilinx Artix-7, and the ASIC is a 32-nm CMOS standard-cell. The HLS tool they used is the Cadence CtoSilicon. The implementation of the FPGA designs were achieved by Xilinx Vivado Design Suite. The FPGA implementations have been compared with a software running on the Zynq Cortex-A9 CPU at 667MHz and the ASIC results were evaluated against the execution time of an identical implementation of the Cortex-A9 in a 32-nm CMOS technology. The maximum achieved speedup is similar for the two FPGA accelerators, close to 25×. For the ASIC speed up to 160× for the MIST method and 2000x for MUSIC-I.

The usage of Magnetic Resonance Imaging (MRI) has intrigued the scientific community, as it is a non-invasive procedure. For that purpose, a lot of work has been done, in order to extract edge-defined and noise-free MRI scans. In the work of Sami Hasan, Said Boussakta and Alex Yakovlev [10] is implemented a generalized parallel 2-D MRI filtering algorithm with their FPGA-based implementation in a single unified architecture. They used Xilinx System Generator tool within the Xilinx ISE 12.3 development suite and two Xilinx Virtex-6 FPGA boards. Their work achieved the enhancement of the filtered MRI scans, in order to be edge-defined and noise-free grayscale imaging in a 64x64 MRI grayscale scan. Their FPGA implementation consumes 0.86 Watt and achieved maximum frequency of up to 230 MHz.

Another work targeting the MRI exported images is done by G. Poorna Sri Ramya and Subburayalu Bhuvaneshwari [11]. Their work is targeting in identifying sudden changes in an image and is optimized for feature detection, more specifically, the Canny edge detector and the Harris corner detector. Edge detection is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. However, Edge detectors perform poorly at corners. Corners provide repeatable points for matching, so are worth detecting. Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. In their work, they claim they achieved a new flexible architecture for Canny and Harris feature detectors. The algorithm simplifications, did not significantly change the algorithm's reliability, as they claim that the proposed method gives more accurate results.

2.2 State of the Art

Several mathematical and computational models have appeared in literature, which investigate the mechanisms that govern glioma's progression and invasion, with the aim of predicting its future spatial and temporal evolution [2]. Current modeling techniques follow a discrete cell-based description, or a continuum framework that deals with the evolution of local cellular densities.

Discrete models simulate individual cell behavior and are able to incorporate cellular parameters and biological rules, as well as address intracellular processes and intercellular communication issues [12], [13]. Cancer cells are usually represented by vectors of variables that determine their position and speed in space, their cell cycle phase, etc [14], [15]. However, they are computationally intensive and they fail to be efficiently initialized with tumors of clinically significant size that already consist of over 1 billion cells; thus, such models are limited to the exploration of small tumors, below the level of clinical detection.

On the other hand, continuum models that describe tumors at macroscopic level as tissues represent alternative approaches in exploring tumor progression dynamics [16], [17], [18]. They assume tumors as spatial distributions of cell densities, exploring volume expansion of tumors of clinically significant size, thus computational requirements are substantially reduced. Additionally, they are capable of accounting for the dynamic changes of chemical ingredients contained in the tumor microenvironment [19], [20] and they can take into account brain tissue heterogeneity and anisotropy [21]. However, they fail to effectively simulate specific cellular factors, (e.g. discontinuous changes) and they are less efficient than discrete models in describing cellular-scale flux factors such as chemotaxis, haptotaxis and cell to cell adhesion. Continuum approaches are most usually implemented by means of reaction-diffusion equations [22], where the cell proliferation factor is alternatively expressed by the Gompertz law, the second order polynomial equation, or the simple linear function [21], [23].

Recently, hybrid models have been developed to overcome particular limitations of continuum and discrete approaches. These models exploit the continuum method in order to simulate the tumor microenvironment, while they implement discrete approaches for cell to cell interactions [16], [24], [25], [26], [27]. Some of these models are multiscale, incorporating the molecular and cellular level [28]. However, hybrid models are still constrained by significant computational costs.

Alternatively, multi-compartmental continuum models are able to include multiphase heterogeneous populations and appear to be effective in describing how subpopulations of various cell types proliferate and diffuse, while they overcome the computational deficiencies of the discrete models [17], [18], [29], [30]. According to such approaches, cells are grouped based on phenotype depending on their access to the necessary nutrients, or the level of differentiation [31]. Invasion, proliferation, changes in phenotype and necrosis are readily expressed as additions or subtractions from the cell densities within each compartment [32]. The intratumoral availability of oxygen is explored most often, since its inadequacy leads to local hypoxia [31], [32], while some approaches apart from oxygen involve glucose as a second vital nutrient [3], [19], [29]. The influence of acidity on cancer cell survival, proliferation, and invasion is explored in the research efforts of [29], [33].

2.3 Theoretical System Description

The modeling applications presented in this Thesis fall under the category of multi-compartmental continuum approaches and focus on predicting the spatio-temporal evolution of glioma, which is a rapidly evolving type of brain cancer, well known for its aggressive and diffusive behavior. These models aim to simulate the spatio-temporal evolution of a glioma tumor in an isotropic and heterogeneous brain tissue, which consists of different anatomic structures. The brain tissue heterogeneity is incorporated through the variable diffusion coefficient according to the different expansion rate depending on the invaded area. More specifically, since glioma diffuses faster in white than in grey matter, the local diffusion coefficient is higher (usually a fivefold difference is considered), while cerebrospinal fluid (csf) is rarely invaded by glioma cells.

In the primary stage of the tumor growth its size and density are small, thus all cells are sufficiently supplied with nutrients, namely oxygen and glucose through the intratumoral and surrounding ECM [20], [27]. At this time they are normoxic/normoglycemic and proliferative, yet they compete for space with neighboring cells [3], [21]. As time proceeds, the tumor radius increases, the local ECM is destroyed by MDEs produced by the tumor and the concentrations of nutrients decrease in its central part. At a certain point, the intratumoral ECM is completely degraded, and nutrients concentrations fall below critical levels that are not sufficient to meet the needs of all cells [29]. Then, the local tumor cells become hypoxic and/or hypoglycemic and the corresponding regions appear in the core of the tumor. The hypoxic and hypoglycemic cells do not proliferate, while their metabolism changes, consuming different amounts of nutrients in order to cope with the new conditions [3]. Finally, when hypoxic/hypoglycemic cells migrate to an area with adequate oxygen/glucose, they convert back to proliferative. As the tumor evolves, hypoxic and/or hypoglycemic populations grow in the central part of the tumor to finally form a hypoxic and/or a hypoglycemic core. Furthermore, the metabolic pathway of cancer cells results to the production of cellular wastes, such as lactate and hydrogen ions, the excess of which leads to the decrease of the local pH value [29]. As a result, an acidic region appears close to the tumor core, where cells stop proliferating and turn to quiescent, while trying to escape this toxic microenvironment.

At a certain time, nutrients are depleted and hypoxic/hypoglycemic cells start dying and turning to necrotic, forming a necrotic region at the tumor's core, where nutrients can no longer diffuse due to the long distance from the tumor surroundings. Additionally, the acidic cells join the necrotic population when they die due to the local waste excess that leads to further pH reduction [29]. Eventually tumor consists of different regions, namely the necrotic core, the hypoxic, hypoglycemic and/or acidic zones around it and the outer proliferating area.

The different cell and chemical compartments, which constitute the building blocks of each model, employ continuum variables (cell densities and concentrations of chemical ingredients), express spatial concentration changes during time and they are governed by diffusion principles. These concentration changes are subjected to diffusion-reaction equations, i.e. second order partial differential equations that can be solved using numerical methods. The mathematical formulas that express the change of the tumor extent in respect to time, are based on the following reaction-diffusion equation [3]:

$$\frac{\partial C}{\partial t} = \nabla \cdot J(C,t) + S(C,t) - T(C,t) \tag{2.1}$$

where J expresses the invasion of tumor cells C, S stands for their increase by proliferation and T represents their decrease because of their conversion to another cell category, or death due to absence of vital nutrients, or effect of the applied pharmaceutical treatment.

The simplest three-dimensional (3D) model involves only one category of cancer cells, i.e. the proliferative cells, since the tumor growth is based only on diffusion and proliferation:

$$\frac{\partial C(l,t)}{\partial t} = D(l,t)\nabla^2 C(l,t) + f(C(l,t)) \tag{2.2}$$

$C(l,t)$ expresses the concentration of proliferative cancer cells at time t at tumor location l, which stands for (x,y) for 2D and (x,y,z) for 3D models, $D(l,t)$ is the diffusion coefficient and $f(C(l,t))$ is the proliferation rate that is expressed according to the second order polynomial equation

$$f(C(l,t)) = p \cdot C(l,t) \cdot (1 - \frac{C(l,t)}{Cm})) \tag{2.3}$$

without the influence of nutrients [4].

When the oxygen is incorporated in the model it involves three different categories of cancer cells, i.e. the proliferative, the hypoxic and the necrotic cells [34]. The tumor growth is based on diffusion and proliferation, which is directly related to the oxygen availability. More specifically, oxygen concentration affects on cellular phenotype, invasion, proliferation and survival, since its change leads to the formation of heterogeneous tumor cell populations, consisting of proliferative, hypoxic and necrotic cell types expressed into distinct cellular compartments.

When oxygen and glucose are incorporated, the model also follows the continuum approach and involves four different cancer cells phenotypes, i.e. the proliferative, the hypoxic, the hypoglycemic and the necrotic [3]. The tumor growth is based on diffusion and proliferation, which depends on the nutrients availability, i.e. oxygen along with the glucose. The model takes into consideration the combined and the separate effect of each nutrient in tumor evolution and composition, leading to the formation of proliferative, hypoxic and hypoglycemic, with different metabolic profiles, as well as proliferation and invasion properties. Additionally, oxygen and glucose concentration changes affects on the survival of tumor cells, since the nutrients inadequacy forces cells to necrosis.

The invasion of proliferative cancer cells is based on the following equation [3]:

$$\begin{aligned}\tfrac{\partial C(l,t)}{\partial t} &= \nabla\cdot(D_c(l,t)\cdot(1-T)\nabla C(l,t)) + f(C(l,t)) + H(l,t)\cdot g_h\cdot(1-n_h)-\\ &\quad -C(l,t)\cdot b_h\cdot n_h - C(l,t)\cdot N(l,t)\cdot a_h\end{aligned} \tag{2.4}$$

where, $f(C(l,t))$ is the cell net proliferation rate and $D_c(l,t)$ represents the variable coefficient of cell diffusion depending on the brain tissue where the tumor spreads. The diffusion and proliferation factors are directly associated to the histological grade of the glioma tumor. Moreover, $H(l,t)$, denotes the concentrations of quiescent cancer cells which can be hypoxic or hypoglycemic, while $N(l,t)$ are the necrotic-cells density. The factor T stands for the fraction of the local cell density to the tissue maximum carrying capacity C_m, such as:

$$T = \frac{C(l,t)+H(l,t)+N(l,t)}{C_m} \tag{2.5}$$

Furthermore, b_h, is the conversion rate of proliferative cells to hypoxic, or hypoglycemic, an the conversion rate to necrotic due to the contact with the necrotic region, g_h, the conversion rates of hypoxic or hypoglycemic cells back to proliferative (under oxygen or glucose excess respectively) and n_h, represents the factor related to the critical oxygen or glucose concentration for the proliferative cells to turn to hypoxic or hypoglycemic respectively.

The local proliferation rate f(C(l,t)) is expressed by the second order polynomial equation, since it takes in account the ratio of the local cell density and the maximum carrying capacity of the tissue, in order to limit the cell proliferation [16], [28]:

$$f(C(l,t)) = n_{con}\cdot p\cdot C(l,t)\cdot(1-T) \tag{2.6}$$

where p corresponds to the proliferation rate constant and n_{con} is the coefficient that reduces the proliferation rate in relation to the local chemical ingredients concentration, which stand for the nutrients, i.e. oxygen and glucose.

As mentioned before, the proliferative cells are converted to hypoxic and/or hypoglycemic under oxygen and/or glucose inadequacy respectively, which turn to necrotic due to the nutrients' depletion. The diffusion of hypoxic/hypoglycemic cells follows the equation below

[3]:

$$\frac{\partial H(l,t)}{\partial t} = \nabla \cdot (D_h(l,t) \cdot (1-T)\nabla H(l,t)) + C(l,t) \cdot b_h \cdot n_h - H(l,t) \cdot g_h \cdot (1-n_h) - \\ - H(l,t) \cdot a_{hn_1} \cdot n_{nh_1} - H(l,t) \cdot a_{hn_2} \cdot n_{nh_2} - H(l,t) \cdot N(l,t) \cdot a_n \quad (2.7)$$

where $D_h(l,t)$ is the local diffusion coefficient, a_{hn_1} is the conversion rate of hypoxic/hypoglycemic cells to necrotic due to the absence of oxygen/glucose, a_{hn_2} is the conversion rate of hypoxic/hypoglycemic cells to necrotic due to the simultaneous inadequacy of glucose and oxygen and n_{nh_1}, n_{nh_2} stand for the respective thresholds to turn to necrotic.

Finally, the change in necrotic cells-density is expressed by the following equation [3]:

$$\frac{\partial N(l,t)}{\partial t} = C(l,t) \cdot N(l,t) \cdot a_n + H(l,t) \cdot N(l,t) \cdot a_n + H(l,t) \cdot a_{hn_1} \cdot n_{nh_1} + H(l,t) \cdot a_{hn_2} \cdot n_{nh_2} \quad (2.8)$$

The concentration-changes of nutrients (oxygen and glucose) are similarly governed by diffusion equations, including their production through the ECM and their consumption by the tumor cells [30], as presented in the following formula:

$$\frac{\partial n(l,t)}{\partial t} = D_n \nabla^2 n(l,t) + \beta_n \cdot e(l,t) - \varepsilon_n \cdot n(l,t) - \gamma_c \cdot C(l,t) - \gamma_h \cdot H(l,t) \quad (2.9)$$

where D_n is the diffusion coefficient the nutrient, β_n is its production rate, ε_n stands for its natural decay rate and γ_c, γ_h are its consumption rates by the proliferative and the hypoxic/hypoglycemic cells respectively.

The formed linear system is expanded in two or three spatial dimensions and the differential equations are numerically approximated by means of the mathematical framework of Finite Differences. More specifically, the implicit scheme of Forward Euler is used in order to derive the direct solutions, which are estimated by iteratively calculating the next-time approximation of spatial concentrations (at each tumor point and time instance) as a linear combination of the previous-time values [4]. The parameter initialization on the above-described equations is performed using either patient-specific, or literature-based reference values that cover a wide range, allowing models to investigate different malignant cases. These multi-compartmental modeling approaches can be validated through experimental results of glioma models available in the literature and evaluated by clinical experts, as they derive reliable results in accordance to clinical assumptions and considerations. Moreover, they can be extended to incorporate additional factors, such as cellular metabolic wastes (lactate and H^+ ions), molecular pathways, biomechanical interactions and targeted cancer therapy, providing patient-specific simulation of different tumor evolution scenarios.

These models can be initialized by virtual spherical tumors that simulate glioma spheroids, or by real tumors derived from imaging modalities (such as MRI sequences). The oxygen-glucose model consists of heterogeneous tumor cell populations incorporating the interactions between different glioma cell phenotypes into distinct cellular compartments, namely proliferative, hypoxic, hypoglycemic and necrotic, as well as their tissue microenvironment. Moreover, it includes the effect of the host tissue, i.e. the extracellular matrix (ECM), and the matrix-degradative enzymes (MDEs) concentrations on tumor cell metabolic profile, survival, proliferation and invasion.

Chapter 3

Implementation

In this chapter, there will be an extensive description of the design methodology that have been used, the designs that were implemented and the difficulties that have been encountered throughout the implementation, for each tumor prediction Model.

Three different growth models, namely the simple 3D, as well as the 2D and the 3D version of the oxygen-glucose (nutrients) model, were applied on a real glioma tumor in order to evaluate their ability to simulate on a realistic brain structure. The tumor was segmented from an MRI sequence taken from the RIDER-NEURO database [35] and embedded in the brain anatomic structure that was derived from the SRI24 atlas [36], an MRI-based atlas of normal human brain anatomy [29]. The final MRI sequence consists of 129 consecutive slices, each having size of 240 x 240 pixels and resolution of 1mm/pixel depicting a heterogeneous brain tissue of different anatomic structures. A medium-diffusion/medium-proliferation combination was simulated on the initial-tumor size, which is approximately 15x30mm. More specifically the diffusion coefficient in the white matter, the gray matter and in Cerebrospinal Fluid (CSF) was assumed as $Dw = 0.03mm^2/day$, $Dg = 0.2 * Dw$ and $Dcsf = 0.01 * Dw$ respectively and the medium proliferation rate was defined as $p = 0.01/day$. All the other initial model-parameter values for the models were taken from the software implementation in [3]. Each model simulation was performed for spatial grid dimensions equal to $dx = dy = dz = 1mm$, a specific evolution time, as well as the determined time step, which corresponds to $dt = 0.001$ days for the three models. Since the tumor is limited to evolve inside the skull boundary, the model application is carried out within the slices (33, 97) and the pixels (50, 190) and (40, 210) on x, y dimension respectively. In this Thesis the results only for the above dataset are shown. In order to use another dataset with different dimension in x and y axis, the designs have to be synthesized again and new bitstream files to be exported. This procedure will be described later in more detail. The prediction results concern the tumor expansion after 1, 3, 6 and 12 months.

The work-flow of the designs, that is followed for every design in every Model, is shown in the next figure.

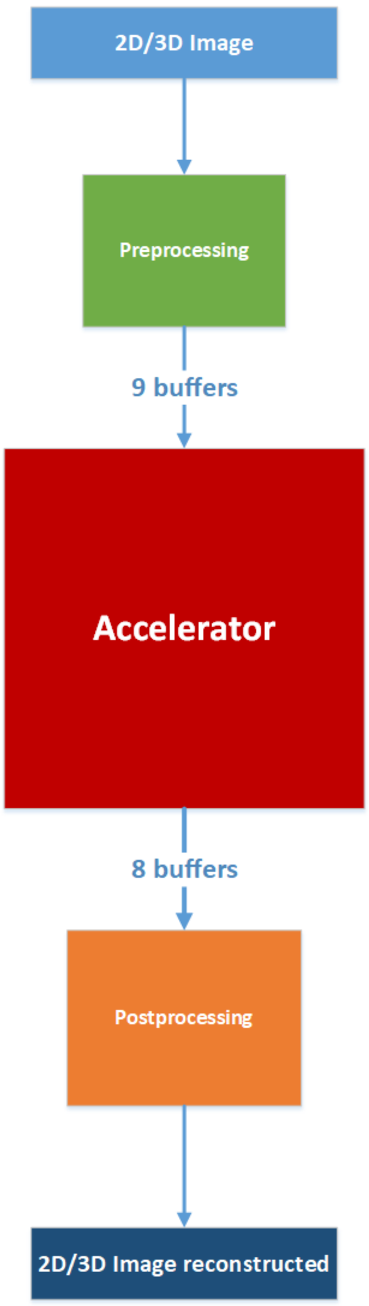


Fig. 3.1 Workflow of the designs

The figure 3.1 shows that the sequence of the MRI are read and then a preprocessing occurs. In the preprocessing the image/images are translated into nine input buffers. The size of each buffer is equal to the size of the MRI images, 240x240 pixels. Each buffer corresponds to a different attribute of the tumor and the brain structure (e.g. concentration nutrients, concentration proliferative cells, etc). After that, these buffers are fed to the hardware accelerators, in which the main simulation processing occurs. Finally, the output buffers are read again and the final image/images are reconstructed.

3.1 Model 2D Oxygen-Glucose Diffusion-Proliferation

The first model analyzed, is the Oxygen-Glucose Diffusion-Proliferation 2D tumor prediction model, that follows the continuum approach and involves four different cancer cells phenotypes, i.e. the proliferative, the hypoxic, the hypoglycemic and the necrotic. There are nine different buffers, in which the brain structure is mapped for different characteristics, such as the proliferative cells concentration, the hypoxic cells concentration etc. The code consists of three parts. The first part is the preprocessing, in which the main nine arrays are being initialized, after the image initialization. The next part is the main part, in which there is the model algorithm. In the final part, the output arrays are being analyzed and the new image is being created (image reconstruction). The main part consists of three loops. The first loop is the time-step loop and the other two are used for traversing through the two-dimensional arrays (X and Y dimensions). The X axis is the line dimension and the Y axis is the column dimension. Every array has a certain value for each pixel, so that the initial image of the brain, could be described in nine different arrays. The process taking place in the algorithm, is using five pixels (neighbor-cells) from each array, at most, in order to calculate any given pixel. Specifically, to calculate the (i,j) pixel the "cross" pixels are needed. The cross pixels are the (i+1,j), the (i-1,j), the (i,j-1), the (i,j+1) and of course the (i,j). As understood, all the algorithm process is nested inside the innermost loop. There are also output arrays with the same size, as the input ones. There are eight output buffers that will be used as input in the following time-step, as will be described later. So, for every pixel (i,j) there are eight outputs, one for every output array, that is written in the (i,j) position of these output arrays. The algorithm uses the double-buffering technique, in which the input and the output buffers are being swapped in every time-step. Thus, in the first time-step the initial input arrays are being read and the result is written into the output arrays, but in the second time-step the output buffers of the first time-step are being used as input and the input buffers as output. This procedure is being used in every time-step. The names of the initial input buffers are D, m, n, f, gl, C, H, Q, N and the outputs mtmp, ntmp, ftmp, gltmp, Ctmp, Htmp, Qtmp, Ntmp. The D array behaves only as input (read-only). The original software uses the double-buffering technique, which is used in to implement the hardware accelerator as well. The basic idea to implement the accelerator for this model, using the double buffering technique, is shown in the following figure.

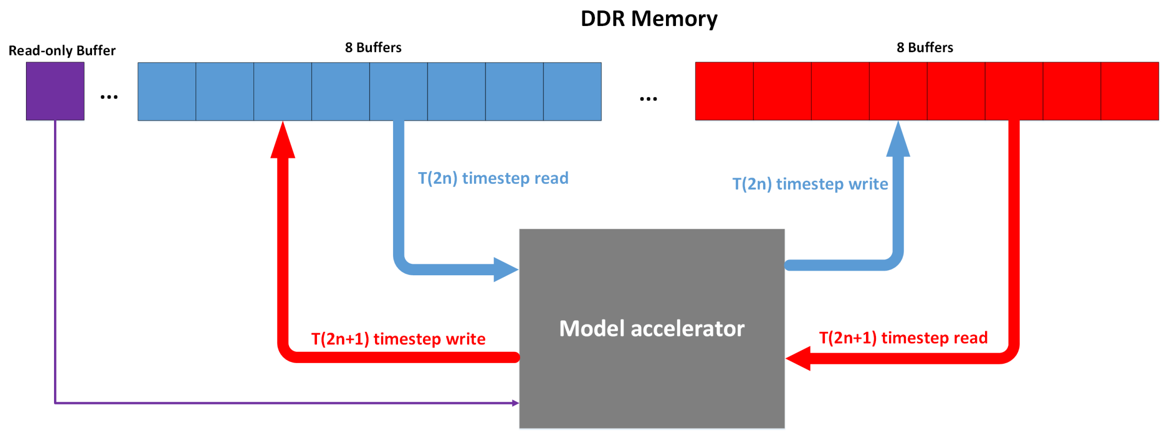


Fig. 3.2 Double buffering in hardware.

3.1.1 Xilinx Vivado HLS Implementation

The kernel that had to be implemented into hardware was the kernel of main part. The original strategy was to use a memory mapped design. To reach this level, firstly the code had to be implemented in Vivado HLS 2016.2 tool. This was one of the first times using an HLS tool to implement a hardware component and for that reason the code just synthesized as it was originally written. The target clock frequency is 200 MHz. At first glance, the result was disappointing, as the design needed 105% of the available LUTs and the target for the clock was not met. But the Vivado HLS does not make code optimizations automatically. There are plenty optimizations that can be made by the user and each one can be used for different situations.

In this situation, three different manual optimizations are used. The first one, is used to meet the timing constraints. To achieve maximum clock frequency, the mathematical or logical functions have to be as simple as possible. For example, the operation *int result = a + b + c;* has to be transformed into simpler operations that contain only two operands e.g. *int temp_a_b = a + b; int result = temp_a_b + c;* This procedure simplifies the code to the HLS compiler. The HLS tries to transform every single line of C code to HDL language blindly, resulting in synthesizing complex operations into a single clock cycle, thus increasing the clock's period, if the target clock period is very small. This optimization could be used in other situations, even in the conditional statements. The HLS synthesizes complex statements into enormous trees of multiplexers, increasing the clock's period to drive their inputs to their outputs.

The next optimization targets to reduce the resources. The HLS does not reuse results that have been already calculated and stored in registers, whereas it uses new LUTs to create new hardware, e.g. *int k1 = a + b; int k2 = a + b + c;*. In this case, HLS will use LUTs to create the *k1*, but for *k2* will not use the *k1*. A sophisticated compiler would predict that *a + b* is *k1*, thus it could be reused and the final code would be *int k1 = a + b; int k2 = k1 + c;* As is already described this optimization could also help HLS meet the target clock frequency.

One more optimization that has been used extensively is the avoidance of division operations. Division is the costliest mathematical operation, for that reason, every single division was replaced by a multiplication with the inverted divisor. This optimization could be used only when the divisor is a constant variable e.g. *double k = a/b;* is transformed to *double b_mult = 1.0/b; double k = a*b_mult;*. The *b*, in this example, is a constant variable. The optimizations described before made the code more complicated for a user but simpler for the compiler and reduced dramatically the resources. Specifically:

Table 3.1 Resource utilization of Original 2D Model (Oxygen-Glucose Diffusion-Proliferation) C Code by Xilinx Vivado HLS

	BRAM_18K	DSP48E	FF	LUT
Total	68	384	82675	69829
Available	1824	2520	548160	274080
Utilization (%)	3	15	15	25

As is already described the code became much more complicated, making it extremely difficult to develop and debug. That is why the implementation and the further developing of the code took a significant amount of time.

3.1.1.1 Streaming Procedure

As the target of this Master Thesis is High Performance Computing, a memory mapped approach could never succeed satisfying speedup, instead it would speed down the algorithm. That is why a different approach is used. The data should not be accessed randomly in DDR, as the design would become I/O bound, thus streaming I/O is used. In this case the data had to be rearranged in Vivado SDK, so that they could be streamed into the hardware core without stalls. For a streaming procedure, a memory controller needs two very fundamental information. The first is the address in DDR, in which the streaming of the data will start from. The second is the amount of data (payload) that it has to transfer. All the models, that have been implemented in hardware in this Master Thesis, have the same structure. The main part has the loop that simulates the time interval. In every time-step the hardware core has to send this fundamental information to the DDR's memory controller in order to start the streaming of data. For the highest possible performance these two information have to be sent by the accelerator at the start of every time-step iteration. This loop, as it is already described, simulates the time interval of the tumor progression and the double buffering occurs. For these reasons, in every time-step the main kernel has to be fed with new data by sending the address and the payload of each buffer, that has to be streamed.

For streaming transfers, Xilinx Vivado Design Suite provides some Integrated Peripherals (IPs) that can be used for data movement. The address and payload information can be fed to these IPs, in order to stream data to/from DDR memory. The first one is the AXI Direct Memory Access (AXI DMA) IP core, that provides high-bandwidth direct memory access between the AXI4 memory mapped and AXI4-Stream IP interfaces. However, the AXI DMA is not used in the implementation of the models, because the two information (address, payload), could only be sent by the CPU (ARM Cortex A53) and not the hardware core IP (Programmable Logic side). This could slow down the I/O, as the CPU has to poll the Hardware Core in order to find out whether the time-step has ended. When a time-step ends the CPU must send a special command to AXI DMA, with the information needed to start the new streaming transfer. This procedure could take up to 50 to 70 cycles, for every time-step. The second one, that is used extensively in this Master Thesis, is the AXI DataMover, which is a key interconnect infrastructure IP that enables high throughput transfer of data between the AXI4 memory-mapped and AXI4-Stream domains. The AXI DataMover provides the Memory Map to Stream (MM2S) and Stream to Memory Map (S2MM) AXI4-Stream channels that operate independently in a full-duplex like method. The AXI DataMover IP core provides burst mode and the ability to queue multiple transfer requests using nearly the full bandwidth capabilities of the AXI4-Stream protocol for the UltraScale+ FPGA. Furthermore, the AXI DataMover provides memory reads and writes to any byte offset location. For that reason, the second IP (AXI DataMover), that is described above, was used extensively. The AXI DataMover is shown in the following figures:

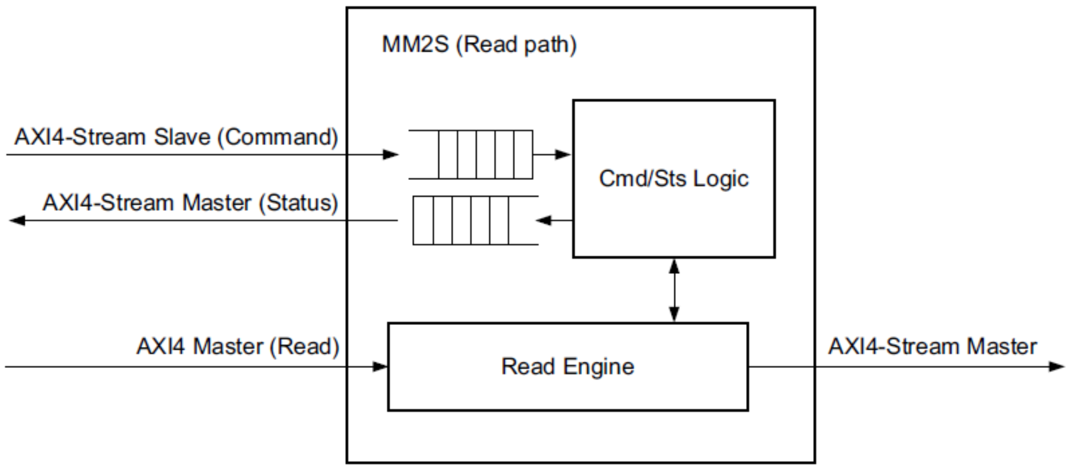


Fig. 3.3 AXI DataMover read datapath [37]

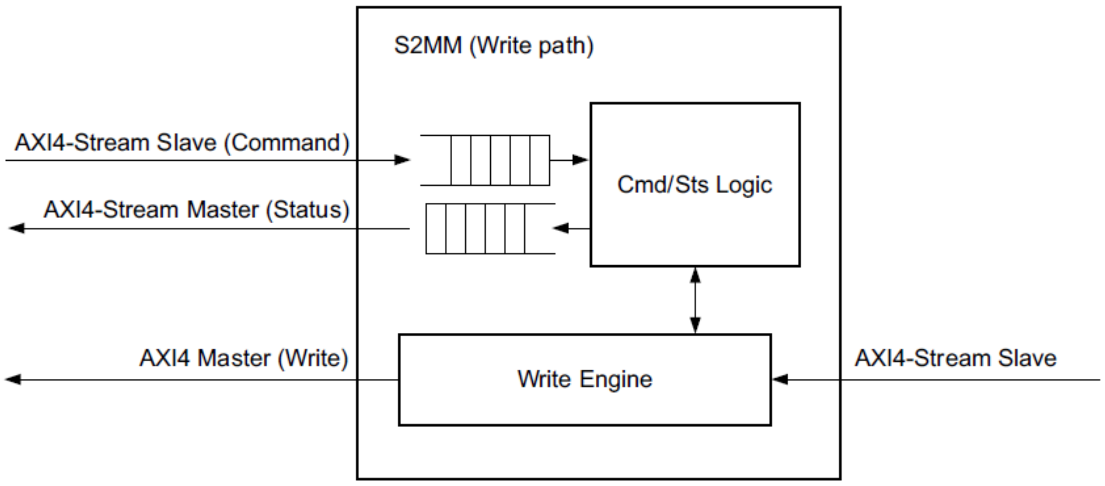


Fig. 3.4 AXI DataMover write datapath [37]

As it can been seen, the AXI DataMover operations are controlled by an AXI4 slave stream interface that receives transfer commands from the user logic. The MM2S and the S2MM each have a dedicated command interface. A command is loaded with a single data beat on the input command stream interface. The width of the command word is normally 72 bits if 32-bit AXI addressing is being used in the system. However, the command word width is extended (by parameterization) if the system address space grows beyond 32 bits. For example, a 64-bit address system requires the command word to be 104 bits wide to accommodate the wider starting address field. The command interface is an AXI4-Stream interface so the total number of bits should be an integer multiple of 8. For example, if the address space is configured as 33 bits, the address portion in the command should be padded to make it 40 bits. This is to ensure compliance with the AXI4-Stream protocol. AXI Datamover will internally ignore the padding bits. The command format allows the specification of a single data transfer from 1-byte to 8,388,607 bytes (7FFFFF hex bytes). A command loaded into the command interface is often referred to as the parent command of a transfer. The DataMover automatically breaks up large transfers into intermediate bursts (child transfers) that comply with the AXI4 protocol requirements. The structure of the command is shown in the next figure.

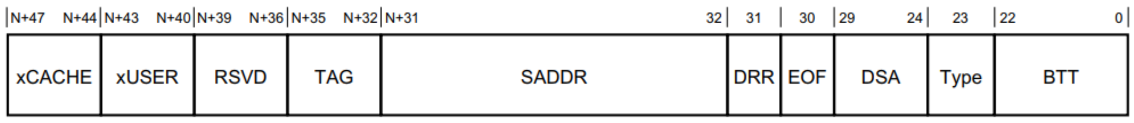

N+47 N+44 N+43 N+40 N+39 N+36 N+35 N+32 N+31 32 31 30 29 24 23 22 0
xCACHE
xUSER
RSVD
TAG
SADDR
DRR
EOF
DSA
Type
BTT

Fig. 3.5 AXI DataMover Command Word Layout [37]
N = Address width for memory map and should be a multiple of 8

In order to practice using this IP, several simpler designs were implemented, using Zedboard. The Zedboard uses 32-bit AXI addressing and for that reason the command word is 72 bits. The command that has been used is the following:

- 0-22: payload size in bytes.
- 23-31: a constant number 0x81, which is 10000001 in binary. These '1's, are setting the first and the last bit from that particular range. The first '1' is 23rd bit of the command word, which determines the type of AXI4 access. Setting this to 1 enables INCR. This bit enables the DataMover to increase the address after every transaction and not reading/writing into the same position (address). The second '1' bit is only used if the optional DRE bit is included by parameterization. The bit indicates that the DRE alignment needs to be re-established prior to the execution of the associated command.
- 32-63: This field indicates the starting address to use for the memory-mapped side of the transfer requested by the command.
- 64-71: These bits were set to '0', as they are not needed.

In the design, in order to use the AXI4 Stream protocol, some C++ templates were set. Firstly, a C struct was implemented that creates the basic AXI4 stream data structure. The struct contains the following:

```
struct AXI_VALUE{
 ap_uint<64> data;
 ap_uint<1> last;
 ap_uint<8> keep;
};
```

In the above structure there are 64 bits of DATA (8 bytes) that a double precision floating point number needs, the LAST bit is used to indicate if streaming has ended and the KEEP bits indicate if every byte of DATA (1 bit for every byte) received and/or transmitted is valid. In this case, the KEEP bits will always be 255 or 11111111, which means that all bytes are valid. Although the designs implemented on Zedboard were very simple (9 reads, some additions and 1 write), it was the first step in order to understand the AXI4 stream protocol and the AXI DataMover IP module. These designs provided the necessary experience in the Xilinx Vivado Design Suite toolchain (Xilinx Vivado HLS – Xilinx Vivado – Xilinx SDK).

The design needed a sophisticated approach in using AXI4 Streaming Interface. In order to achieve the maximum performance a streaming data movement requires to transfer data continuously without sending new requests to DDR's memory controller for data in different addresses. For that reason, the approach that has been followed is the following. As it is described above, for every pixel to be calculated, 5 pixels are required. At the start of each time-step, inside the loop, 2 lines of each input buffer are streamed and stored in local buffers (Block RAM) inside the FPGA. These two lines of each buffer are stored in two different buffers, whose size is the size of the line Y. Having those 2 buffers, the calculation can start for the 1st pixel of the 2nd line. The first line is initialized to zero ('0'). In that case, the data for the pixel is: (i,j) from the second buffer, (i-1,j) from the first buffer, (i,j-1) assumed as '0', (i,j+1) from the second buffer and finally for the (i+1,j) a new 64-bit data is received from the corresponding stream. After the calculation is done, the first buffer in j position copies the value from the second buffer in j position. The j-1 position in second buffer copies the value from j position in the same buffer. After that, the j position from the second buffer copies the new value, that has been streamed. That way the first and the second buffer are "moving" throughout the entire corresponding input buffer. To sum up, 2 buffers for each stream are initialized the first time inside the time-step loop and after that, these 2 buffers have the necessary data (cross pixels), plus the streaming data for (i+1,j) to calculate every pixel's value. The following images show the procedure step by step. The gray colored boxes are the data that are not stored in the Block RAM. The red boxes are the data of the first buffer and the purple of the second one. The yellow box is the data that are being streamed in every iteration.

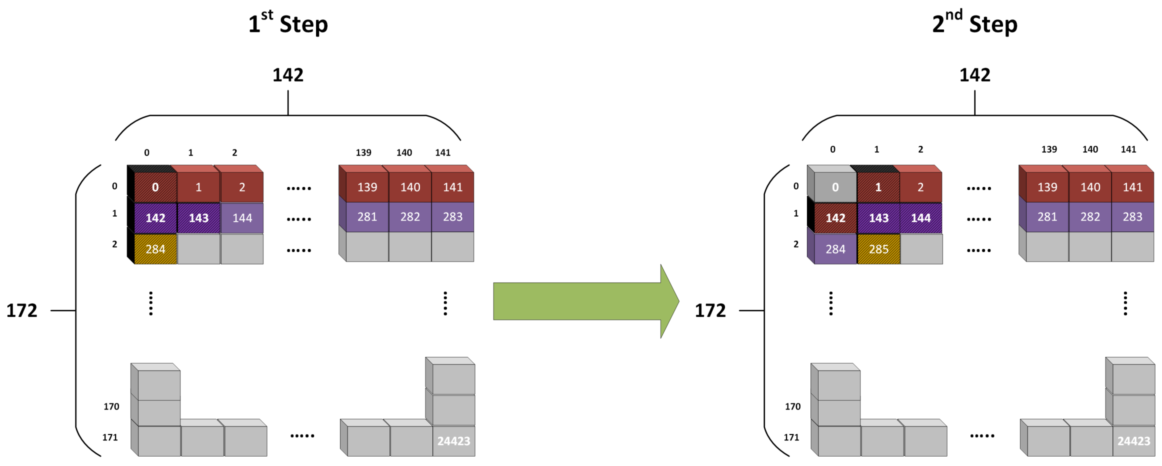


Fig. 3.6 The data inside the 1st and 2nd buffer for the first two iterations.

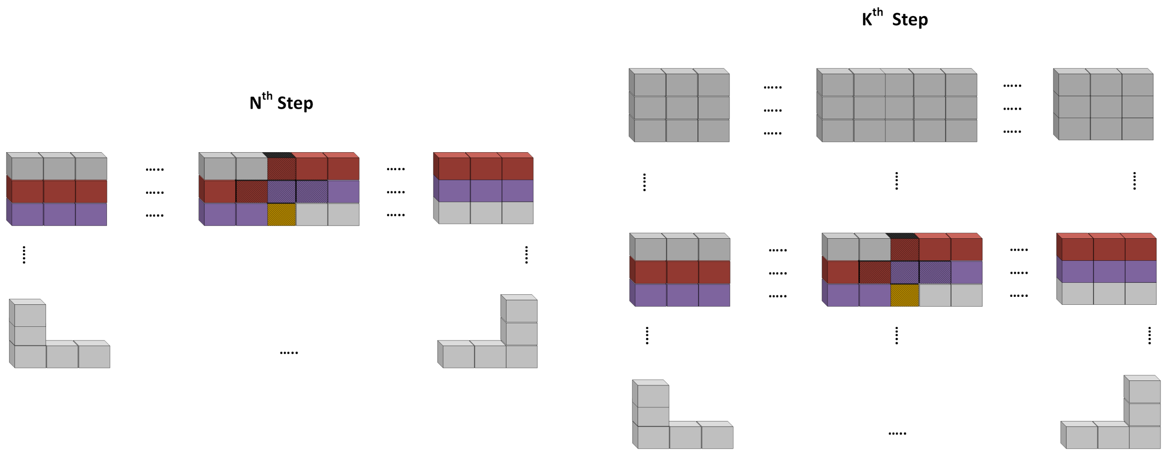


Fig. 3.7 The data inside the 1st and 2nd buffer for two iterations.

To achieve the best performance, the HLS PIPELINE directive has been used. The PIPELINE directive, is a HLS directive, that allows the concurrent execution of operations and reduces the initiation interval for a function or a loop. In this case, it has been used to speed up the innermost loop, because if it was used in one of the other loops it would create a costly design, as it fully unrolls the inner loops. The PIPELINE directive can process new data every 1 clock cycle (Initiation Interval=1). This was achieved by using temporary variables that could propagate the data to the next iteration of the loop, without dependencies and without data hazards. One more technique that was used, in order to achieve II=1, was the duplicated memory. The Block RAM (BRAM) is dual port, that means it allows two operations in every cycle (2 reads, 1 read – 1 write, 2 writes). When PIPELINE directive is being used, in order to not create data hazards, the HLS compiler creates a design with internal stalls when it sees data dependencies. In this case, for every temporary buffer from the 2 buffers that had been used, more than 2 ports were required. For that reason, more temporary buffers were used that were copies of the originals (duplicated buffers).

3.1.2 Xilinx Vivado Project

The second step of the implementation is the IP integration in a Vivado project. In Vivado a new block design was created. Inside the block design the IP of the core was added, that had been created in HLS and had been exported. The core IP has 9 input-AXI4-Streams (Din, C, H, Q, Ncapital, m, n, f, gl) and one slave s_axilite interface, in which there are bundled the remaining inputs (constant variables) and the main axilite control signals (ap_start, ap_idle, ap_ready, ap_done) and 2 more inputs, clock and reset. The outputs of the IP are 8 AXI4 Stream (Ctmp, Htmp, Qtmp, Ncapitaltmp, mtmp, ntmp, ftmp, gltmp). (At this point, it is worth mentioning that the Vivado HLS 2016.2 tool shows a significant bug. As it is known, unlike the C programming language is case-sensitive, the HDL languages are not. The *Ncapital* buffer's original name was *N* but the synthesis compiler of Vivado HLS could not distinguish the *n* buffer and the *N*. As the result of that, the HLS synthesis compiler created wrong architecture and the hardware's behavior was abnormal. For that reason, the *N* buffer was named *Ncapital*.) There are also the AXI4 Streams that are used as commands for the AXI DataMovers. There are 9 commands for the input streams (read) and 8 commands for the output streams (write). For that reason, 9 DataMovers are used. The 8 DataMovers (DMs) are used as reading and writing to DDR and 1 only for reading. This particular DM is used to the Din stream, as it is the only input buffer that does not need to write back in DDR. For every DM specific settings are used. Firstly, all DMs are set to FULL Channel Type, Memory Map Data width and Stream Data width to 64 bits, width of BTT field to 23 bits, address width 64 bits (UltraScale+) and Maximum Burst size to 256 "beats". The "beat" refers to a single data word, in this case 64 bits. The last one is one of the most important settings. The DM streams data in every clock cycle, but there is a limit. The DM after finishing sending "Maximum Burst Size" data, it has to send again a new request. When the Maximum Burst size is at the maximum value, which is 256, the DM sends fewer requests and the streaming procedure is much more efficient. The trade-off is the resource utilization that increases as the Maximum Burst Size increases. This happens as Vivado creates big FIFOs inside the DMs in order to store and propagate the data. The Platform also includes an APU and specifically a 4-core ARM Cortex A53 (PS APU). After that, some settings were set in the needs of the 2D model application. First of all, in order to achieve maximum bandwidth, 6

High Performance ports were used. Two of these ports are cache coherent, but they were used as simple High-Performance Ports. In every port 128 bit of data width is set . The reason is that Trenz's Zynq PS shows a bug when 64 bits are used and lose data during transmission. This behavior was shown in every implementation, in every model in this particular Master Thesis. The clock speed of Programmable Logic (PL) of the Zynq System was set to 150 MHz, as it is nearly impossible to achieve better frequency. The reason is the complexity of the design. Vivado, in order to achieve high clock frequency, tries to reduce area utilization inside the FPGA, but this creates congestion because of the limited routing points. As a result, for Vivado to synthesize and implement the design, the clock speed was set to 150 MHz. The synthesis and implementation, in order the bitstream file to be created, takes too long, nearly 4 hours. The final block design is shown in the following figures. The first figure shows the block design as it is shown in Xilinx Vivado.

Table 3.2 Xilinx Vivado resource utilization report for 2D Model Oxygen-Glucose Diffusion-Proliferation

	Block RAM Tile	DSP48E	FF	LUT	CLB Logic Distribution
Total	155.5	1421	169994	120651	26972
Available	912	2520	548160	274080	34260
Utilization (%)	17.05	56.39	31.01	44.02	78.73

The table 3.2 shows the final recourse utilization, as extracted by the Vivado. The report shows that a 2-core design can be implemented. However, in Chapter 4 in table 4.5 is shown that a 2-Core design could not achieve performance, as the bandwidth is already saturated by the Single-Core.

The next figure shows the same block design in a more abstract way, in order to be more understandable.

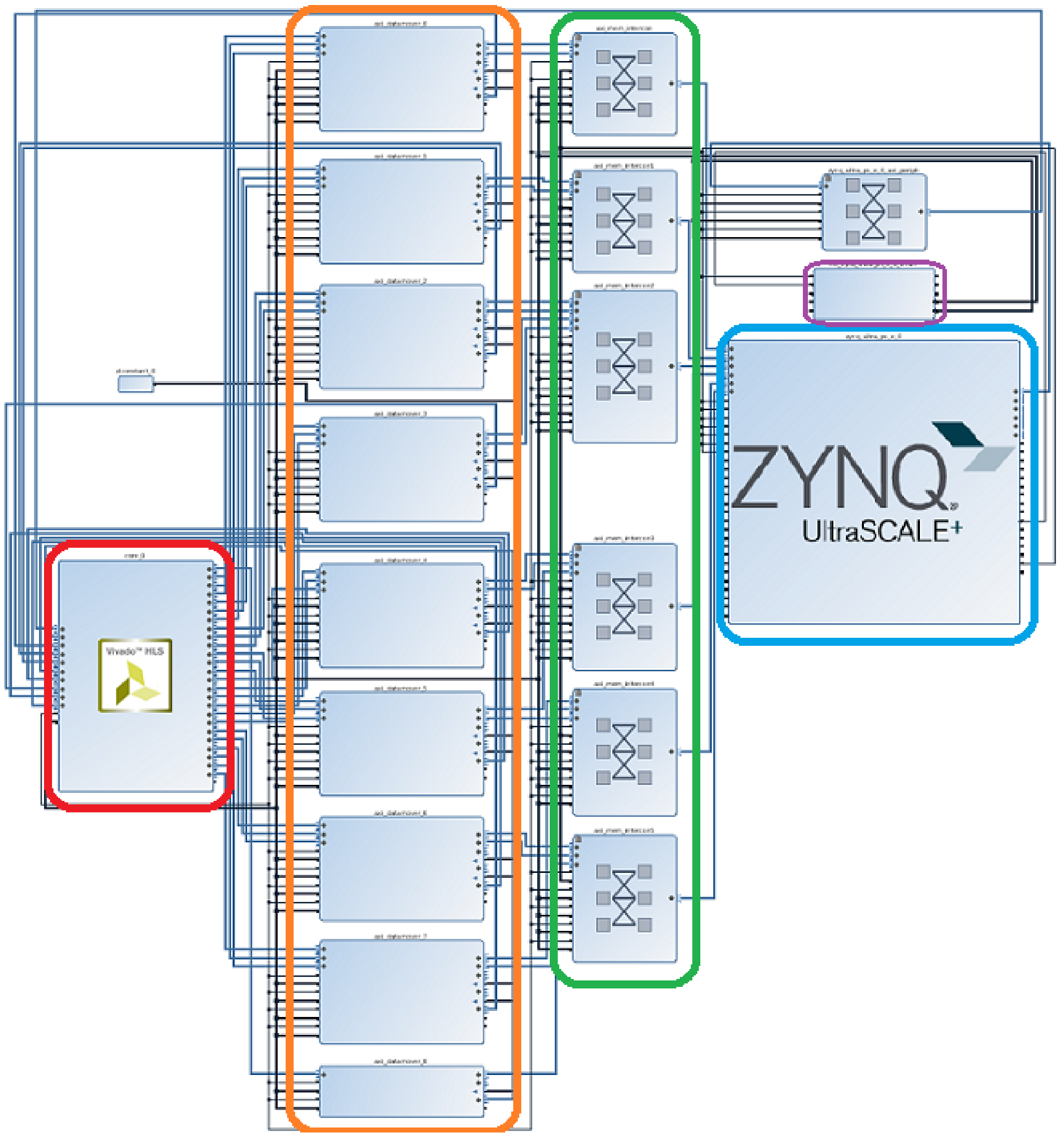


Fig. 3.8 Model 2D Block Design from Xilinx Vivado.

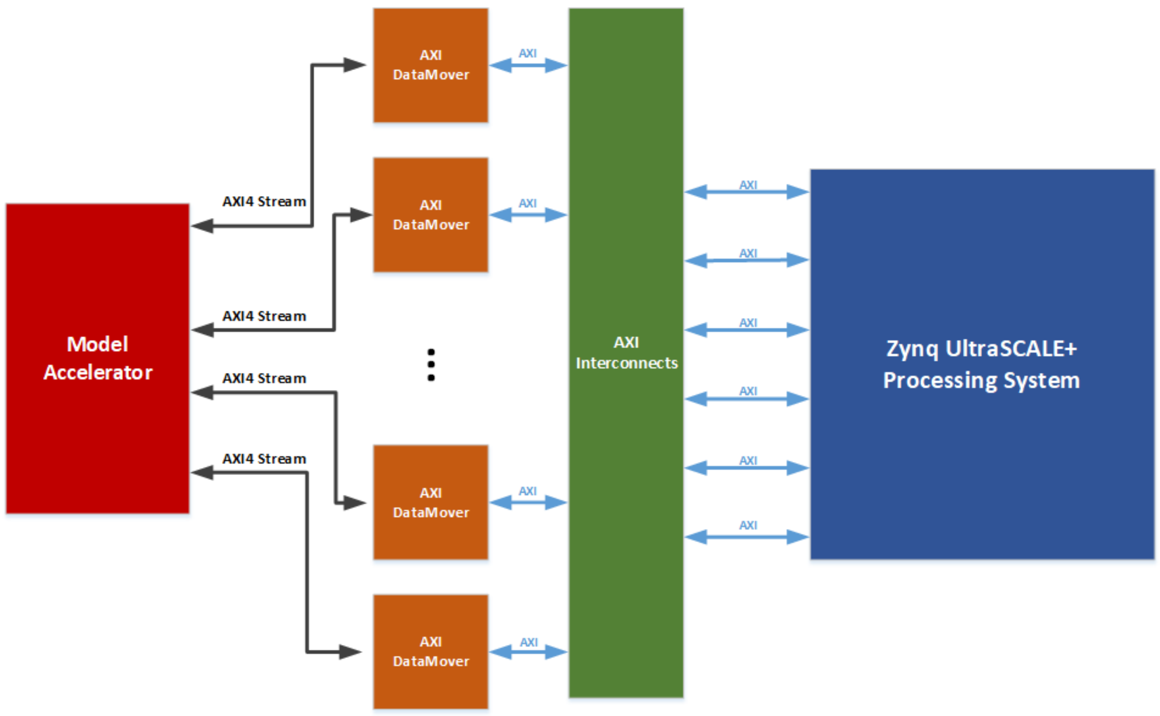


Fig. 3.9 Model 2D Block Design.

In the first figure, the red circled module is the Hardware HLS Accelerator, the orange modules are the AXI DataMovers, the green are the AXI Interconnects, the blue module is the Zynq PS and the purple is the PS system reset module. In this design there is only one clock domain for the PL, thus there is only one PS system reset. The colors for the second figure are similar to the first one.

3.1.3 Xilinx SDK application

After the implementation was done and the bitstream file was ready, a new project was created in Xilinx SDK. This tool is a SDK tool that allows the programming of the corresponding PS of the platform, in this case, inside this platform the ARM Cortex A53. When a new IP is exported from Vivado HLS, some drivers are exported too. These drivers are used as an API, that can be used later in Xilinx SDK. In this case, the IP core of the 2D model had exported drivers only for the s_axilite interface, as this is the only interface that is “visible” from PS. The AXI Streams are managed by the DMs, in fact the IP core is the Master and the DMs are the Slaves, unlike the PS, which is the Master of the IP core. The drivers that had been extracted for the core IP have the basic control functions (isReady, isIdle, Start, isDone) and the functions that set the constant variables for the algorithm. These functions are writing to specific registers in the FPGA the corresponding values that the user sets. The 2D model can perform the calculation in the entire image of MRI, but it is not necessary, as a large percentage of the image is black. That is why the 2D model starts the calculation in a specific window, where the brain is located. This particular feature has been utilized in every tumor prediction model algorithm in this Master Thesis in order to achieve the maximum streaming interface utilization. The data of the image were rearranged and only this particular window is stored to DDR. That way the streaming could be continuous, without stalling.

The Xilinx SDK application starts with the initialization of the UART peripheral. The SDK tool provides numerous drivers for every peripheral that can be connected with the PS. These drivers allow the the communication of the CPU of the platform with a serial terminal. A host PC is used with an installed serial terminal program e.g. GTKTerm, Tera Term, which sends the this image. That way the initial MRI image is sent to the UART peripheral. The UART is set to baudrate 115200 (bits/second) and the image's size is 97696 bytes ($x*y*sizeof(integer) = 142*172*4$). The transfer of the image from a host PC to the platform takes 6.7 seconds approximately. Some of these programs installed are GTK Term and Tera Term. After the transfer of the image 17 arrays are initialized (9 inputs, 8 outputs). These arrays' sizes are the window of the image that is useful (brain). The initialization follows the exact same procedure as the original first part of the code. The addresses (Read/Write) of each buffer are extremely important to be set to a specific Base Address and a specific region, as this address would be the Base Address that DM would start the streaming procedure. As it can be seen, this Base address is a variable that is sent to the core IP. After this initialization, the IP core is initialized, as well, via the drivers that are provided. Then the APU calls the core IP inside the FPGA that starts by the Start function. The APU polls the FPGA, via the isDone function, to determine the end of the execution. After the execution is complete the C, H, Q, N arrays that are stored in DDR are used to determine the grayscale colors of the final buffer. This buffer, whose size is the same as the initial image (97696 bytes), is sent to the host PC, where a C program reconstructs it to the final image.

The Trenz platform is an evaluation model for academic use that has many problems. One of the most important problems was the malfunction of JTAG. This malfunction did not allow to program the FPGA with a simple USB and debug the application via an ILA core. For every change in the SDK code the programming of the FPGA, was achieved by storing a BOOT file in a SD card. This problem made extremely difficult the debugging process, that took around 2 months.

3.2 Model 3D Simplified

The second model analyzed, is the Simple Diffusion-Proliferation 3D tumor prediction model. This model simulates the three-dimensional evolution of glioma. As the previous model, Oxygen-Glucose Diffusion-Proliferation 2D model, this model consists three parts: initialization, evolution processing and image reconstruction.

As it is understood, the 3D model uses three dimensions X, Y and the Slice, as an MRI scan consists of many images. To read the value of every pixel the three dimensions that are needed are line (i), column (j) and the third dimension is the slice (sliceNo), which is the index of the corresponding image. To sum up, the index for every pixel is (sliceNo, i, j). In the first part, the program starts by reading 129 text files. Every file corresponds to an image (sliceNo). Each file consists values for every pixel in the image. The values correspond to values of grayscale images but normalized between 0 and 4. In this model there are only 2 input buffers and 1 output, as it only simulates the concentration of the proliferative cells. The input buffers are C and Din and the output is Ctmp. For that reason, the C and Ctmp buffers are taking part in the double buffering procedure in each time-step. Although the image is three dimensional, the buffers are mapped in one dimension. That way the transformation of the C code to HLS code is made simpler. During the text file reading, these buffers are initialized to some other specific values according the initial values of the image files (0-4).

After the first part of the code, the second part is the main part, in which all the calculations of the model are taking place. In this part there are four nested loops. The first loop is the time-step loop. The time-step loop simulates the time-step of the calculations. At the start of each time-step the double buffering procedure occurs. The next nested loop is the loop that defines the slice (image) index and the other two nested loops define the line and the column. As it was described earlier, the buffers are mapped to one dimension and thus the index of each pixel must be mapped respectively. The index of each pixel is a calculation of the three indexes (sliceNo,i,j), which is:

$$index = sliceNo * XMAX * YMAX + i * YMAX + j \tag{3.1}$$

The XMAX and YMAX are the number of lines and columns respectively. In this model the calculation for every pixel needs seven “neighbor 3D-cross” pixels. The positions of these pixels are: the position of the current pixel, the position of the right pixel (next column), the position of the left pixel (previous column), the position of the above pixel (previous line), the position of the below pixel (next line), the same position of the pixel in the previous image and the same position of the pixel in the next image (z axis). The index for each of these pixels, given the previous formula 3.1 is:

Current pixel: index
Right pixel: index + 1
Left pixel: index - 1
Above pixel: index - YMAX
Below pixel: index + YMAX
Previous image pixel: index - XMAX*YMAX
Next image pixel: index + XMAX*YMAX

This way every pixel needed for any calculation can be mapped into one-dimensional buffer, as the original software uses triple pointer arrays. The model prediction process for every pixel is not so complex as in the 2D model.

The third part of the code is the reconstruction of the final image, with the extra information after the model prediction simulation. The final C buffer, after a specific number of time-steps, is analyzed and the final values are normalized to numbers 0-4. These numbers are mapped to 129 grayscale images.

3.2.1 Single Core Implementation

3.2.1.1 Xilinx Vivado HLS Implementation

The code that had to be implemented into hardware was the whole main part. The strategy was to be implemented to a streaming design directly, without trying to be implemented firstly into a memory mapped one. Having much more experience in Vivado HLS by the previous model design (2D Oxygen-Glucose), the transformation of the original code to an optimized code, was much simpler. The techniques, that are used in 2D model, are used again. The first technique is the simplification of complex calculations in order to achieve maximum clock frequency. An example of this technique, in this particular model, is:

Original C code:

```
Cx=((C[address+xmax]-2*C[address]+C[address-xmax])/(dx*dx));
```

Transformed HLS Code:

```
double Cz_zero_center_2 = 2*Cz_zero_center_reg;
double Cz_down_reg_Cz_zero_center_2 = Cz_down_reg-Cz_zero_center_2;
double Cx=((Cz_down_reg_Cz_zero_center_2+Cz_up_reg));
```

The second technique, that is used extensively, is the reduction of resources. This is achieved by creating temporary variables. After some code analysis, the repetitive calculations and variables were detected. These variables and calculations are saved to temporary registers and the access is very fast.

Original C code:

```
Cp1=(C[address]*proliferationRate*dt);
Cp2=(pow(C[address],2)*proliferationRate*dt)/Cm;
```

The "C[address]*proliferation*dt" is the same for these two lines of codes and the transformed code is:

Transformed HLS Code:

```
double Cz_zero_proliferationRate_dt =
                Cz_zero_center_reg*proliferationRate*dt;
double Cp1=(Cz_zero_proliferationRate_dt);
double Cp2_temp = Cz_zero_center_reg*Cz_zero_proliferationRate_dt;
double Cp2=(Cp2_temp)*Cm_mult;
```

The final technique is the usage of multiplications instead of division operations on the constant divisor variables, to reduce resources.

Streaming Procedure

After the first step of transformations of the original C code, the next transformation of the code into a streaming I/O design took place. The main interface used is AXI4 Stream, in order to use the AXI DataMovers as the key interconnect infrastructure IP to stream data in DDR. The 2D model uses two buffers, for every streaming array, whose size is YMAX (number of the columns). In this model a new technique similar to 2D is applied. The main difference is the 3rd dimension that is introduced in the 3D model. As it is described above, every output pixel needs seven other pixels. In order to achieve streaming, only one pixel is read in every iteration. The most optimized way is by storing two whole images to buffers in Block RAM and a an extra buffer, whose sizes are MaxSlices*XMAX*YMAX and YMAX respectively, for every input stream. Having these two images in BRAM, only the third image is needed to be streamed pixel by pixel. The small buffer is used for storing the *Above pixel (index – YMAX)* in every iteration. The following figure shows the describes this procedure.

The two temporary buffers are "moving" throughout the images, showing the data that are stored in them in every iteration. The small buffer is not included in the figures, as it used for optimization mainly. The blue buffer-image stores the N-1 image, the green stores the N image and are located in Block RAM. The gray boxes are the data that are not stored in Block RAM. The orange is the data that is streamed in every iteration.

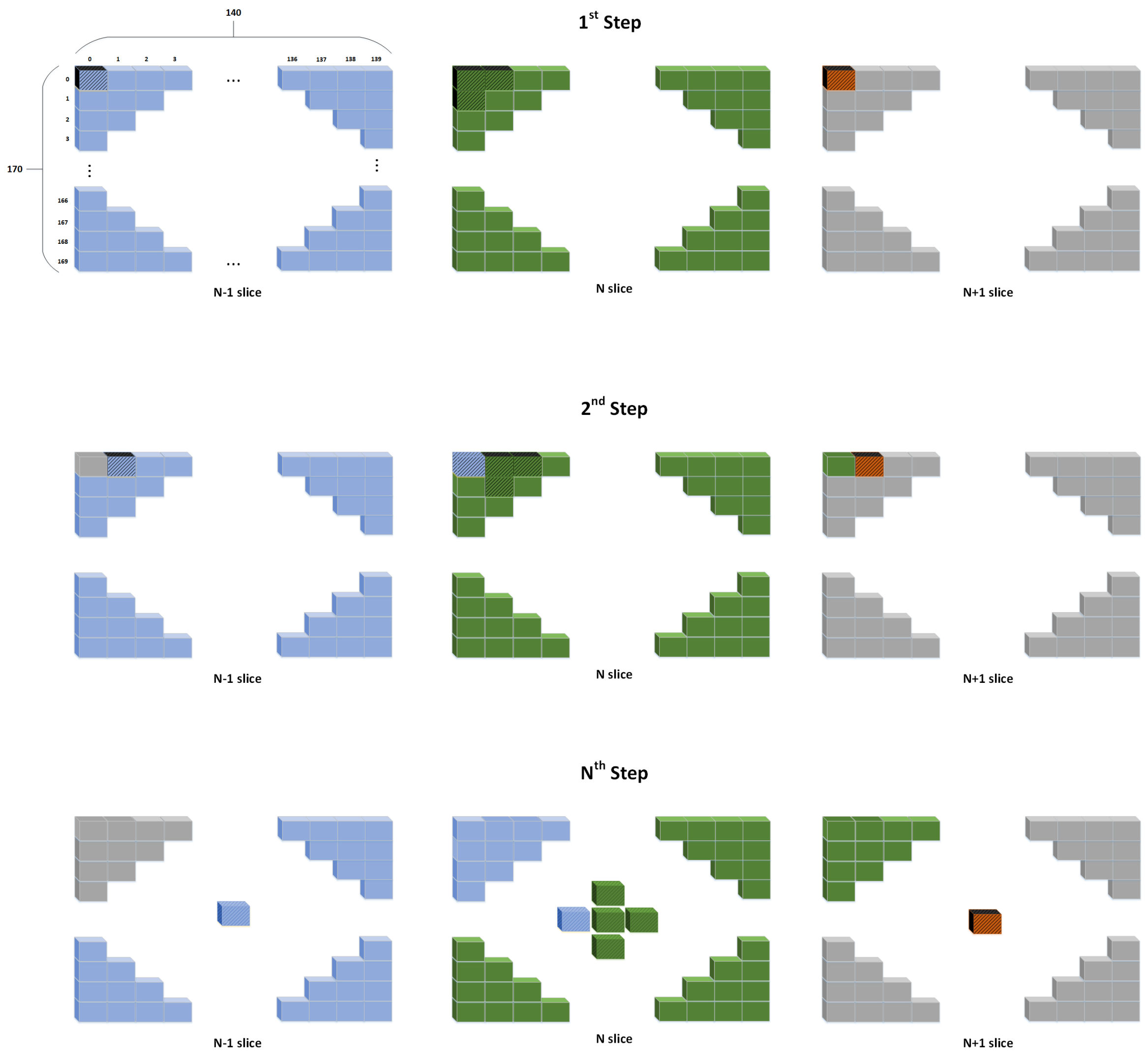


Fig. 3.10 The data inside the 1st and 2nd buffer for some iterations.

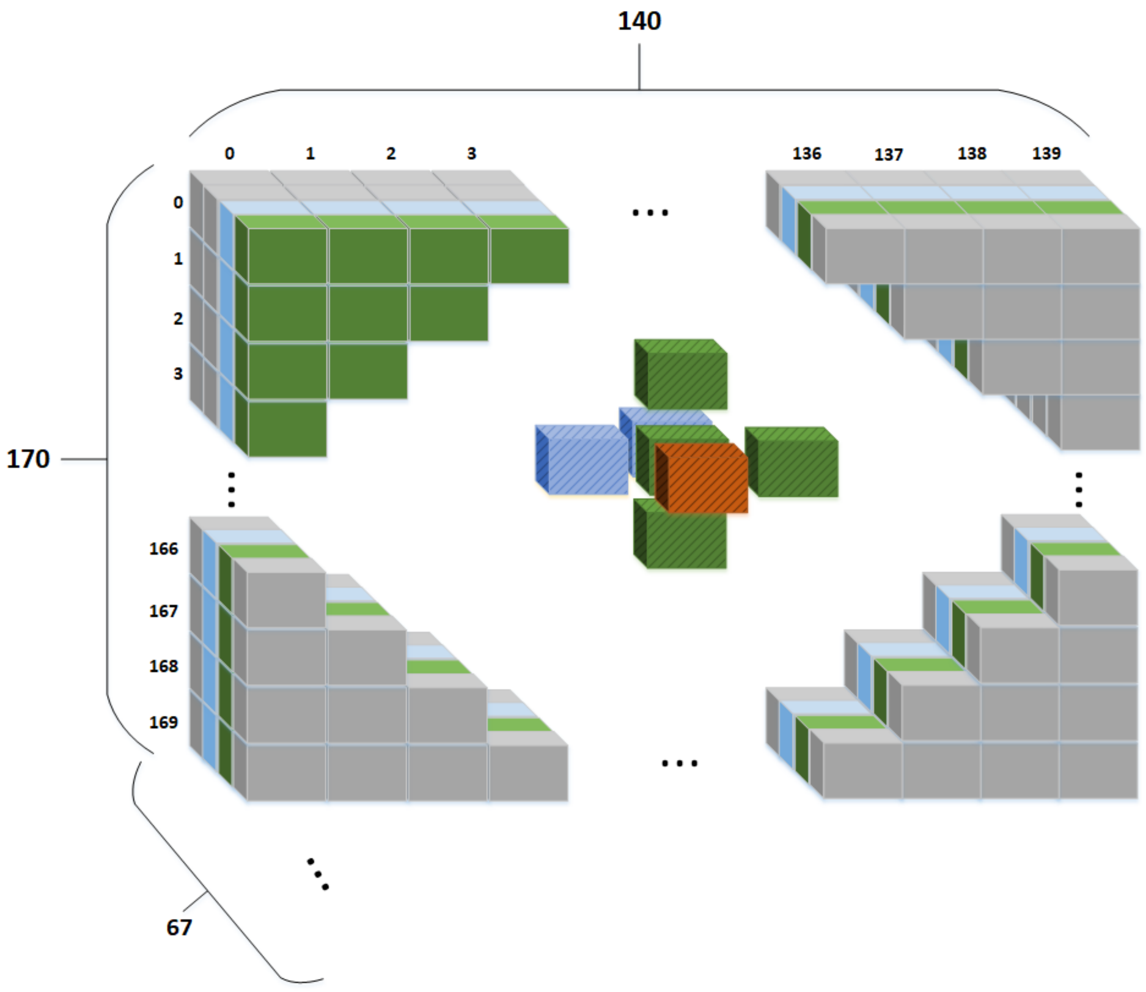


Fig. 3.11 The data inside the 1st and 2nd buffer for the Nth iteration in 3 dimensions.

The pixels: Current pixel: index, Right pixel: index + 1, Left pixel: index - 1, Below pixel: index + YMAX, Previous image pixel: index - XMAX*YMAX are all stored in the first two temporary buffers (images). The Above pixel: index - YMAX is stored in the temporary buffer, whose size is YMAX. The Next image pixel: index + XMAX*YMAX is being streamed in every iteration in the innermost loop. The names of these buffers are Cz_up[YMAX], which is the small buffer, Cz_minus[XMAX*YMAX], which is the buffer of the N-1 image, Cz_zero[XMAX*YMAX], which is the buffer for the N image. These first three local buffers are initialized at the start of every time-step and thus some clock cycles are spent. After the calculation of the output pixel, the local buffers and some registers are updated with the next pixels, in order to be prepared for the next iteration. The extra registers help the propagation of the of the next pixels to the next iteration. The indexes of the loops, that traverse through the different images, are sliceNo (image index), i (line index) and j (column index). So, Cz_minus buffer in (i,j) position and the Cz_up buffer in j position copy the value from the Cz_zero buffer in (i,j) position. Finally, the Cz_zero buffer in (i,j) position copies the new value, that has been streamed (orange pixel in 3.11), in that particular iteration. That way the first and the second buffer, which are the two buffers that two whole images are stored in them, are "moving" throughout the entire input 3D image stream, which consists many images, by been overwritten with new pixels in every iteration. The small buffer is "moving", as well, from up to down in every image.

The images, where the brain and the tumor are mapped, are 33 to 97 (65 images) from the initial 129 images. Each image has size of 240x240 pixels, but the useful pixels are lines 40 to 210 (170 lines) and columns 50 to 190 (140 columns). The algorithm starts the calculations in 33 image and pixel (40,50). However, this pixel needs information from the image 32 (N-1 image) and the final image 97 also needs information from the next image 98 (N+1). However, the useful data is written to the images 33 – 97 and in pixels (40-210, 50-190). This characteristic is very important during the design of the hardware. As it will be described later, In Xilinx SDK the data is rearranged accordingly for the application, in order only the useful to be streamed in the accelerator. For the C buffer 67 images are streamed (32-98), whose pixels are (40-210, 50-190), for the D buffer, which is read-only, 65 images (33-97) are streamed, whose pixels are (40-210, 50-190). During the writing, 65 images are written (33-97). The pixels that are located at the limits of the useful window (40-210, 50-190), need data that are not streamed in the design e.g. the pixel (40, 157) needs the pixel (39,157), in order to complete its calculations. The values of all these data are considered as zeros ('0'). As it has already been described, for different dimensions in input images the design needs to be exported again by the Vivado HLS and the Vivado, because the sizes of the buffers in Block RAM must be static.

The iterations for the loops in this particular design are the following. First of all, the time-step loop, whose number of iterations depends on the input variable of time (tmax, dt). The user can affect the duration and accuracy of the simulation by setting different values for tmax and dt. Inside the time-step loop there are two blocks of three nested loops each. In the first block of the three nested loops are used for the initialization of the local buffers for every input stream. The outmost loop iterates 2 times, the next nested loop iterates XMAX times and the innermost loop YMAX times. This way the two local buffers for every input are initialized with two whole images. In the innermost loop the stream sets a value to the corresponding pixel of each of the two buffers. At the very start of the tumor simulation, the first buffer (Cz_minus) is initialized with the image 32 and the next buffer (Cz_zero) is initialized with the image 33. After these loops the second block of loops starts, where the main loops of the algorithm are taking place. There are three nested loops again. For this dataset the first loop iterates 65 times, the next 170 times and the innermost 140 times. The outermost loop iterates 65 times because 2 iterations have already taken place in the previous initialization loops. This also helps the output streaming process to DDR (writing), as the writing is taking place too, inside the very same loops of the tumor simulation processing. Every output pixel is written to the (i,j) position of the in the corresponding slice. Every stream inside the innermost loop transfers and receives data according the addresses that are assigned to DataMovers.

In this design the DataMovers are used again, with a command same as in the 2D model, in a 64-bit APU architecture:

- 0-22: payload size in bytes.
- 23-31: a constant number 0x81, which is 10000001 in binary. These '1's, are setting the first and the last bit from that particular range. The first '1' is 23rd bit of the command word, which determines the type of AXI4 access. Setting this to 1 enables INCR. This bit enables the DataMover to increase the address after every transaction and not reading/writing into the same position (address). The second '1' bit is only used if the optional DRE bit is included by parameterization. The bit indicates that the DRE alignment needs to be re-established prior to the execution of the associated command.
- 32-95: This field indicates the starting address to use for the memory-mapped side of the transfer requested by the command.
- 96-103: These bits were set to '0', as they are not needed.

It is very important that the payload size has the exact value as the number of data that will be streamed. When the payload size is a wrong number of bytes, or if there are more/less reads or writes, the DataMovers will stop working. Every stream (read or write), inside this particular design, has its own unique command.

Inside the innermost loop the HLS PIPELINE directive is applied. The design was not ready to achieve the maximum speed for this particular application. The pipeline directive could achieve Initiation Interval (II) 3 and the reason is that all these local buffers, stored in BRAM, have two ports for read/write, as it is described in 2D model section. To achieve II=1 some buffers had to be duplicated. The buffer that is more stressed is the Cz_zero buffer, which is the current image that is processed in any given moment. Inside the algorithm, the Cz_zero is accessed 3 times for reading and 1 for writing. For that reason, 2 extra duplicated buffers were created, that are copies of the Cz_zero. The duplicated buffers were named

Cz_zero_copy1 and Cz_zero_copy2. The use of these duplicated buffers achieved the target II=1.

The resource utilization with target clock 5 ns is the following:

Table 3.3 Resource utilization of optimized 3D Model (Simplified) by Xilinx Vivado HLS

	BRAM_18K	DSP48E	FF	LUT
Total	348	116	14652	13603
Available	1824	2520	548160	274080
Utilization (%)	19	4	2	4

3.2.1.2 Xilinx Vivado Project

First of all, in Vivado HLS the IP core was exported. Afterwards, this IP was imported into a new project, that was created in Vivado. In this design the IP core has 2 input streams and 1 output, 1 slave axilite, clock and reset. Therefore, there are 2 commands for the input streams (read) and 1 command for the output stream (write). For that reason, 2 AXI DataMovers are used. The first one is used for streaming from the DDR and for the Din input stream. This DM has the "write" channel disabled. The second one uses dual channel (read and write) and it is utilized by the C input stream and Ctmp output stream. These two streams utilize the double buffering procedure. The settings of the AXI DMs are the same as the DMs that are used in the 2D application. These settings utilize the maximum potential of the DMs. The PS is used as Master of the core IP, as it is in model 2D application. The Slave Axilite interface is connected to an AXI interconnect as Slave and the Master General Purpose Port from Precessing System (PS) as Master. The settings of the PS are 2 High Performance Ports, with 128-bit data width and Clock Frequency of the Programmable Logic (PL) to 200 MHz. Two figures are shown below. The first shows the real block design as it is exported from Xilinx Vivado, while the second shows an abstract block design for the accelerator and the Zynq PS system.

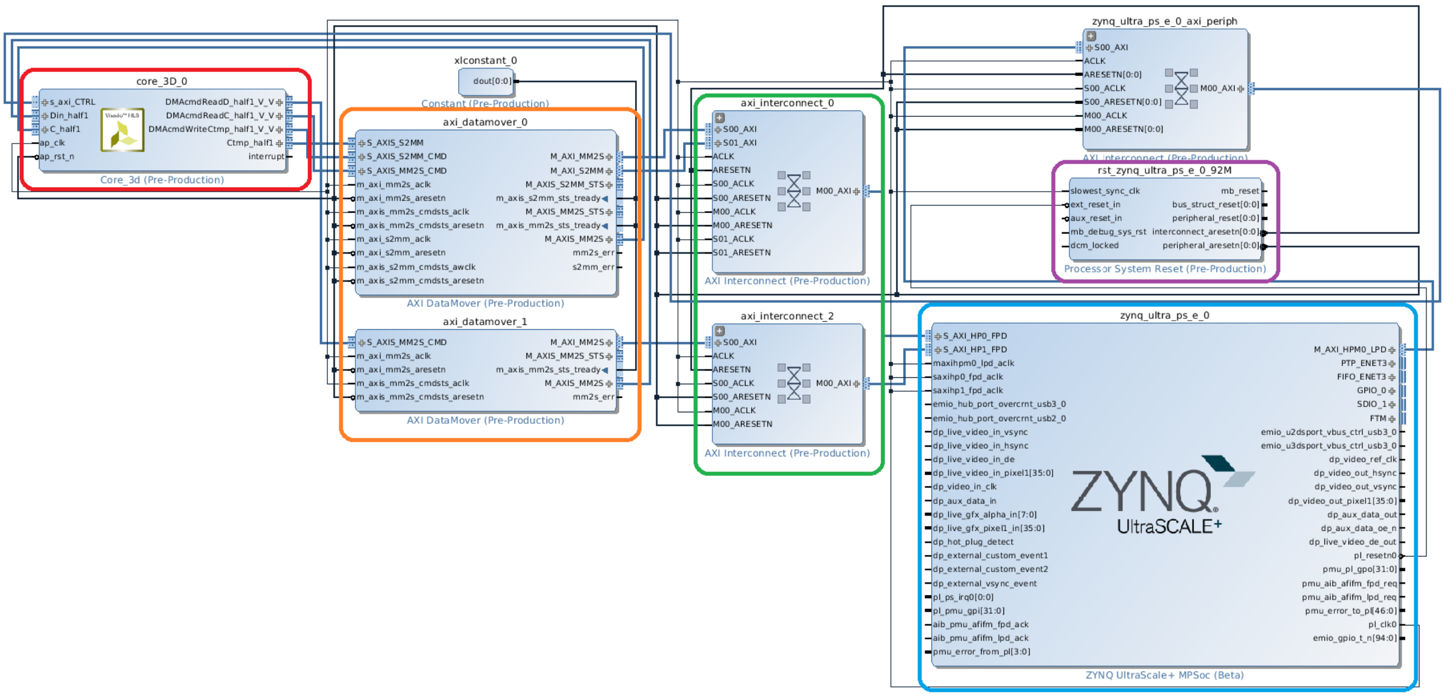

core_3D_0
s_axi_CTRL
Din_half1
C_half1
ap_clk
ap_rst_n
DMAcmdReadD_half1_V_V
DMAcmdReadC_half1_V_V
DMAcmdWriteCtmp_half1_V_V
Ctmp_half1
interrupt
Core_3d (Pre-Production)
xlconstant_0
dout[0:0]
Constant (Pre-Production)
axi_datamover_0
AXI DataMover (Pre-Production)
axi_datamover_1
AXI DataMover (Pre-Production)
axi_interconnect_0
AXI Interconnect (Pre-Production)
axi_interconnect_2
AXI Interconnect (Pre-Production)
zynq_ultra_ps_e_0_axi_periph
AXI Interconnect (Pre-Production)
rst_zynq_ultra_ps_e_0_92M
Processor System Reset (Pre-Production)
zynq_ultra_ps_e_0
ZYNQ UltraSCALE+
ZYNQ UltraScale+ MPSoc (Beta)

Fig. 3.12 Model 3D Simplified Block Design for single-core design from Xilinx Vivado.

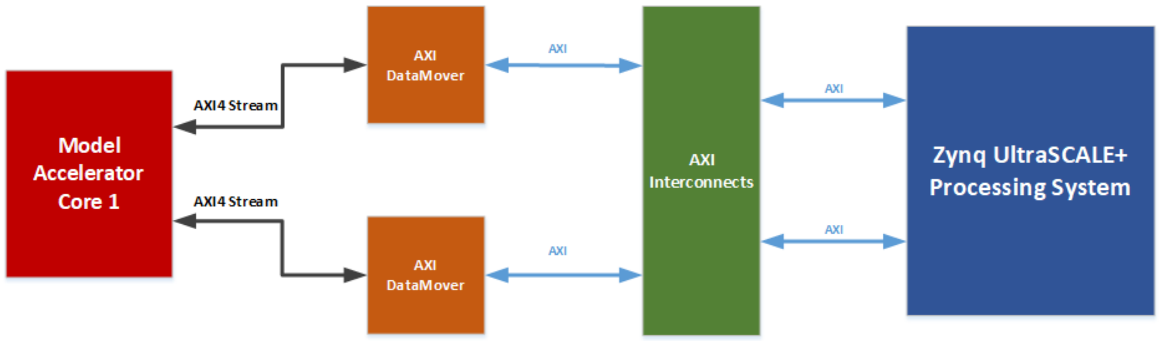


Fig. 3.13 Model 3D Simplified Single-Core Block Design.

In the first figure, the red circled module is the Hardware HLS Accelerator, the orange modules are the AXI DataMovers, the green are the AXI Interconnects, the blue module is the Zynq PS and the purple is the PS system reset module. In this design there is only one clock domain for the PL, thus there is only one PS system reset. The colors for the second figure are similar to the first one.

3.2.1.3 Xilinx SDK application

The produced bitstream file was used to create a new project in Xilinx SDK. In this new project, the final application was developed. The application starts by initializing the UART driver and reading via the USB the input file. This file has been created in a C program and contains all the 129 text files into a single one, which size is 7.4 MB. The time to send this buffer takes around 10 minutes and 30 seconds. Then the program follows the similar pattern as the original one. The initialization part starts by creating the 3 buffers (2 input and 1 output). The buffer's addresses and the payloads, for every DM, are predefined and are sent as inputs to the corresponding arguments of the hardware accelerator IP. The size of these buffers is much smaller than the original. The reason is that the hardware core calculates only the data that are located in the useful window (images: 32-98, X: 40-210, Y: 50-190), as it was described above. That means the payloads, that are sent to DMs, are $67 * 170 * 140 * 8 = 12756800$ bytes for C, Ctmp streaming buffers and $65 * 140 * 170 * 8 = 12376000$ bytes for Din buffer. After the initializations of the input/output buffers the initialization of the hardware core takes place, by using the corresponding drivers, that have been generated by Vivado HLS. Finally, the hardware core starts and the results are written back to C buffer after a number of time-steps.

At the first try the Hardware did not work and the function isDone never returned 1. It was observed that if the hardware core runs for fewer images (e.g. 10 images and the same pixels in each image), the isDone function returned 1 and the application was able to exit successfully. The payloads when 10 images are used are 10*170*140*8 = 1904000 bytes for C, Ctmp and $8 * 170 * 140 * 8 = 1523200$ bytes for Din. These payloads are much less than 8388607 bytes, that is the limit of the AXI DataMovers and for that reason the hardware core could run properly. In order to overcome this problem, the streaming process of each input stream had to be cut in half and two different DM commands to be sent with different payloads. The AXI DataMover can store multiple commands in FIFOs in each channel and execute them sequentially.

For the above reason, the commands and the HLS core had to be changed. The first change is that commands had to be sent two times instead of one. The payload had to be cut in half, thus it was decided that the first command would send 34 images and the second 33, for the C and Ctmp buffers when they are read and for the Din buffer 33 and 32 respectively. During the writing the commands' payloads are 33 for the first and 32 for the second for each of the two buffers C and Ctmp. The interface of the HLS core remained the same, as only the internal code of the HLS design has been changed to send two commands through the same interfaces sequentially. At the start of the outer loop, which simulates the time-step, the two commands for each stream are set and sent sequentially for each stream. The payloads of each stream are:

Read payloads

- C, Ctmp 1^{st} payload: $34 * 170 * 140 * 8 = 6473600$ bytes
- C, Ctmp 2^{nd} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- Din 1^{st} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- Din 2^{nd} payload: $32 * 170 * 140 * 8 = 6092800$ bytes

Write payloads

- C, Ctmp 1^{st} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- C, Ctmp 2^{nd} payload: $32 * 170 * 140 * 8 = 6092800$ bytes

The second change that has been made was in the axilite interface, as there were created some more inputs for the hardware core. These inputs were the base addresses and their corresponding payloads for every command. These arguments were bundled in the global bundle of axilite interface in the hardware accelerator. For these new arguments the HLS generated the corresponding drivers, that could be used later in Xiinx SDK in order to set them with the proper values. The final core has almost the same resource utilization. In Vivado the same project is used and there are no changes at all, except the import of the updated IP core. In Xilinx SDK some changes were made regarding the addresses and the payloads. In the first part of the code, where the initializations are taking place, two addresses and two payloads were generated for every buffer. As an example, the first address (Base address) for Din is 0x0000000010000000, while the second is generated by adding $33 \times 140 \times 170 \times 8$ bytes. By using this procedure, all addresses were generated. The updated payloads and addresses were sent by the corresponding generated drivers of the hardware accelerator IP. After that the application functioned as expected.

3.2.2 Implementation with multiple parallel cores

3.2.2.1 Xilinx Vivado HLS Implementation

The previous design (single-core) inspired the idea of a new design. The new design is more optimized than the first one, as it utilizes the parallel processing between the images in a single time-step. In every time-step every image can be processed independently from the others. However, in every time-step the process has to be serialized because of the double buffering procedure, that occurs. The time-step loop can not be parallelized, as it would occur data hazard.

The parallelization, as it is described, occurs in the slice loop and is achieved by making minor changes in the single-core HLS code. First of all, the target of this design was the full parameterization and re-usability of the hardware core, in order to create multiple parallel cores. The first approach targeted in creating a 2-core design. The first core would process the first 34 images and the second one the rest 33. The changes that had to be made were few. All base addresses and the payloads of each core, in which the accelerator would start the process, became arguments of the function. The max iterations for the slice loop became an argument of the function too. The core could be called for different base addresses, payloads and could be called for any number of images. For that reason, this core can be instantiated many times and a multi-core design can be created. The final change was two single bit signals, that were added in the design, to avoid deadlocks and data hazards. These signals are core_this_finish (output) and core_others_finish (input). Data hazard that would occur if a core simulated the tumor evolution in the Nth time-step the other starts the simulation for the $N^{th}+1$ time-step. All cores must start at the same time inside every time-step, otherwise the double buffering procedure will occur data hazards. The signal core_this_finish, which is an output 1-bit signal, is set to '1' for each core when every time-step starts, in a multi-core design. After that, each core stalls until the core_others_finish, which is an input 1-bit, becomes 1. This stalling is achieved by a while loop that waits the input signal turns to 1 (while(!core_others_finish);). When the core_others_finish signal becomes 1 the core turns its core_this_finish bit to 0, which indicates that the process for this particular time-step is taking place. The two cores are communicating using these two signals by driving the core_this_finish of the first core to the core_others_finish signal of the second core and vice versa the core_this_finish of the second core with the core_others_finish of the first core. This way, every core starts its process at the same time as the other, for each time-step.

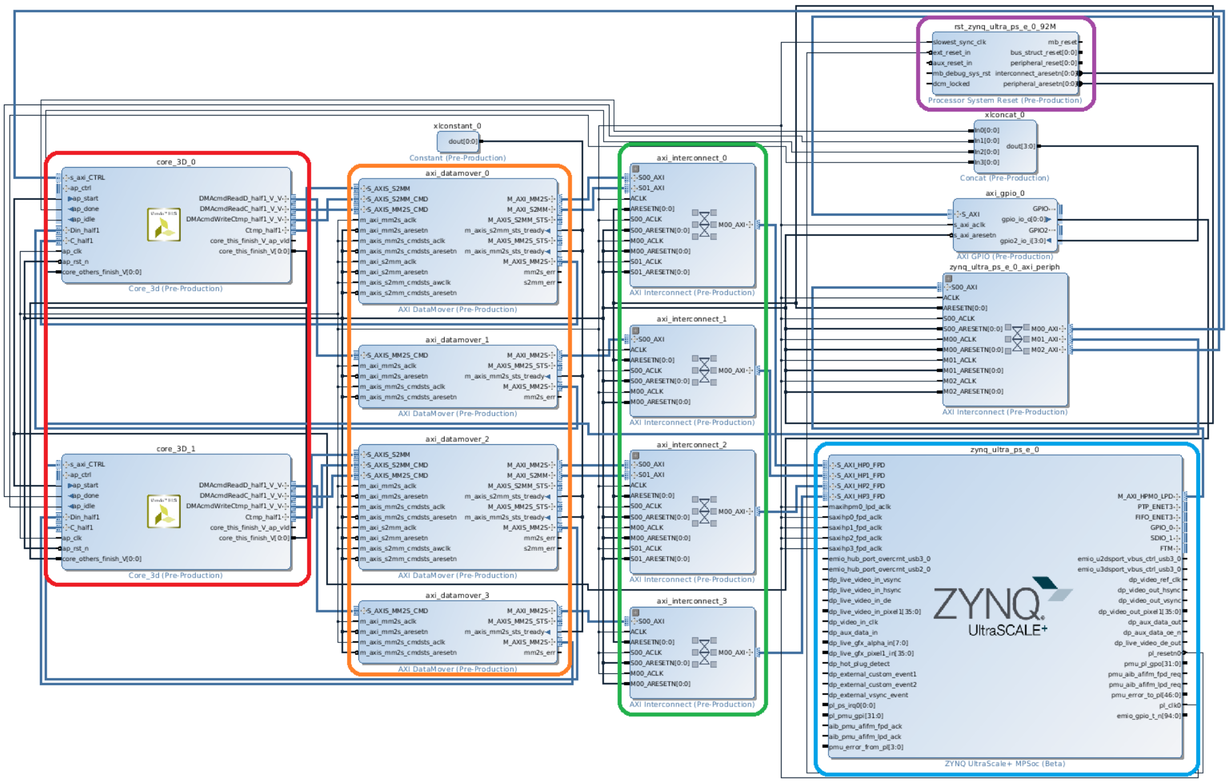
3.2.2.2 Xilinx Vivado Project

For the first multi-core implementation 2 cores are used. A new project was created in Vivado, in which the hardware accelerator is instantiated two times. For each of these 2 cores, in order to achieve maximum parallelization, 2 DataMovers are used for each stream. The Din, C and Ctmp streams are streamed parallel to these 2 cores. Every DM's settings are set as the previous design's settings. For this design the Processing System (PS) is set to have 4 slave High Performance Ports, that is 2 HP ports for each core. Each DM was connected to each HP port by an AXI Inteconnect. The Clock frequency is also set to 171.6 MHz. The single bit signals of each core are connected directly, as it is described above. The Utilization Report showed that the resource utilization is almost doubled as it is expected.

Table 3.4 Xilinx Vivado resource utilization report for simplified 3D Model using 2 cores

	Block RAM Tile	DSP48E	FF	LUT
Total	374	232	45055	34152
Available	912	2520	548160	274080
Utilization (%)	41	9.21	8.22	12.46

The two figures show the block design from Xilinx Vivado and an abstract block design for the two accelerators and the Zynq PS System.



rst_zynq_ultra_ps_e_0_92M
Processor System Reset (Pre-Production)
xlconcat_0
Concat (Pre-Production)
axi_gpio_0
AXI GPIO (Pre-Production)
zynq_ultra_ps_e_0_axi_periph
AXI Interconnect (Pre-Production)
xlconstant_0
Constant (Pre-Production)
core_3D_0
Core_3d (Pre-Production)
core_3D_1
axi_datamover_0
axi_datamover_1
axi_datamover_2
axi_datamover_3
AXI DataMover (Pre-Production)
axi_interconnect_0
axi_interconnect_1
axi_interconnect_2
axi_interconnect_3
AXI Interconnect (Pre-Production)
zynq_ultra_ps_e_0
ZYNQ UltraSCALE+
ZYNQ UltraScale+ MPSoc (Beta)

Fig. 3.14 Model 3D Simplified Block Design for 2-core design from Xilinx Vivado.

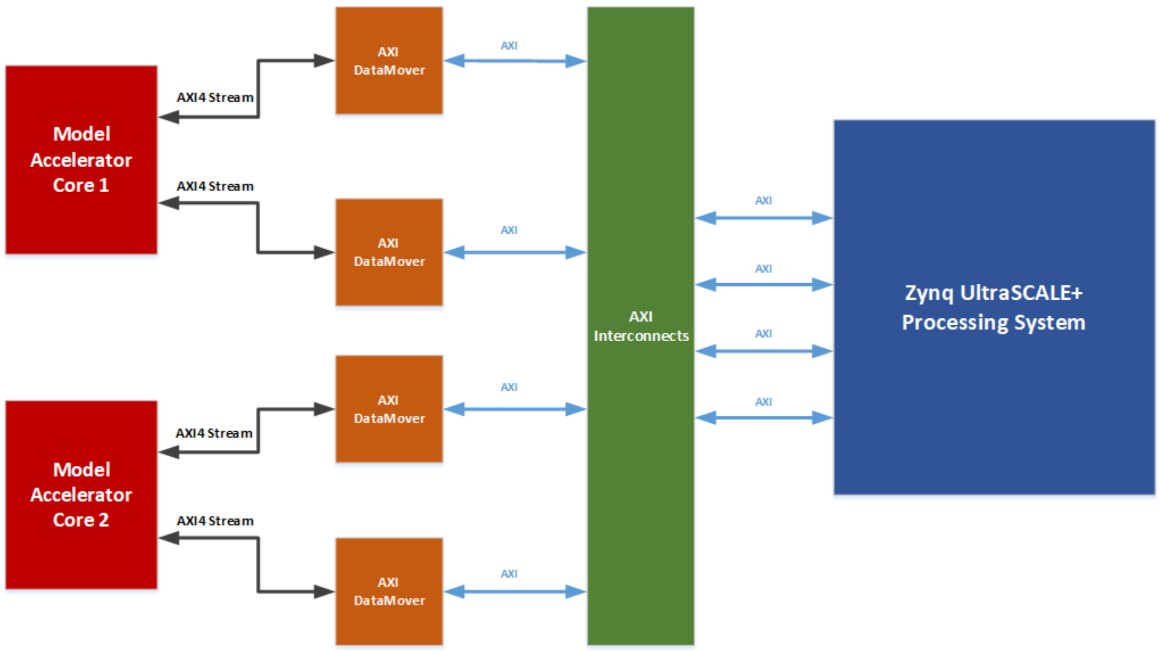


Fig. 3.15 Model 3D Simplified 2-Core Block Design.

In the first figure, the red circled module are the Hardware HLS Accelerators, the orange modules are the AXI DataMovers, the green are the AXI Interconnects, the blue module is the Zynq PS and the purple is the PS system reset module. In this design there is only one clock domain for the PL, thus there is only one PS system reset. The colors for the second figure are the same to the first one.

3.2.2.3 Xilinx SDK application

By using the new exported bitstream file, a new project – application was created. The new application is very similar with the single core one. The first part, which is the initialization of the buffers, is almost the same. The most important difference is the change of the base addresses and payloads of every stream. The changes targeted the avoidance of data hazard. The first core starts reading from the base addresses that are assigned by the single core application, but the end of the stream has to be the 35th image and not the 34th. This way the core can process the first 34 images. The second core's base address starts at the 33rd image and not in 34th. The reason is that the second core needs 34 images, in order to process 33 images. The write addresses for the first and the second core are the same as the single-core one. The payloads of each core are assigned respectively according the needs of every core. So, the first and the second core's payloads are:

Read payloads - 1st Core

- C, Ctmp payload: $35 * 170 * 140 * 8 = 6664000$ bytes
- Din payload: $33 * 170 * 140 * 8 = 6283200$ bytes

Read payloads - 2nd Core

- C, Ctmp payload: $34 * 170 * 140 * 8 = 6473600$ bytes
- Din payload: $32 * 170 * 140 * 8 = 6092800$ bytes

Write payload - 1st Core

- C, Ctmp payload: $33 * 170 * 140 * 8 = 6283200$ bytes

Write payload - 2nd Core

- C, Ctmp payload: $32 * 170 * 140 * 8 = 6092800$ bytes

After that the initialization of the buffers takes place. In the main part each core is initialized by using the drivers that have been exported by the Vivado HLS. The main difference here is the new argument that initializes the value of maximum iterations of each core. The first core's iterations are set to 35 images and the second to 34 images, as it was described above.

3.2.3 Implementation with 4 parallel cores

The final design of 3D model includes four parallel cores in order to achieve better parallelization. This new design targets in the full utilization of all HP ports and the FPGA's computational-power potential. For the final design a new Vivado project was created. In this design the hardware core is instantiated four times and four identical cores are created. The four single bit signals core_this_finish, of each core, are driven as inputs to a 4x1 AND gate and the output of the AND gate is driven to each core_others_finish as input of the cores. The Processing System (PS) settings are 6 HP ports and 150 MHz clock frequency. Each core uses two AXI DataMovers, thus there are eight DMs. They are connected to 6 HP ports, thus some DMs shared the same HP port. The utilization report shows that the resources are doubled comparing to the previous design (2-core):

Table 3.5 Xilinx Vivado resource utilization report for simplified 3D Model using 4 cores

	Block RAM Tile	DSP48E	FF	LUT
Total	748	464	89778	68056
Available	912	2520	548160	274080
Utilization (%)	82	18.41	16.38	24.83

The two figures below show the block design from Xilinx Vivado and an abstract block design for the four accelerators and the Zynq PS System.

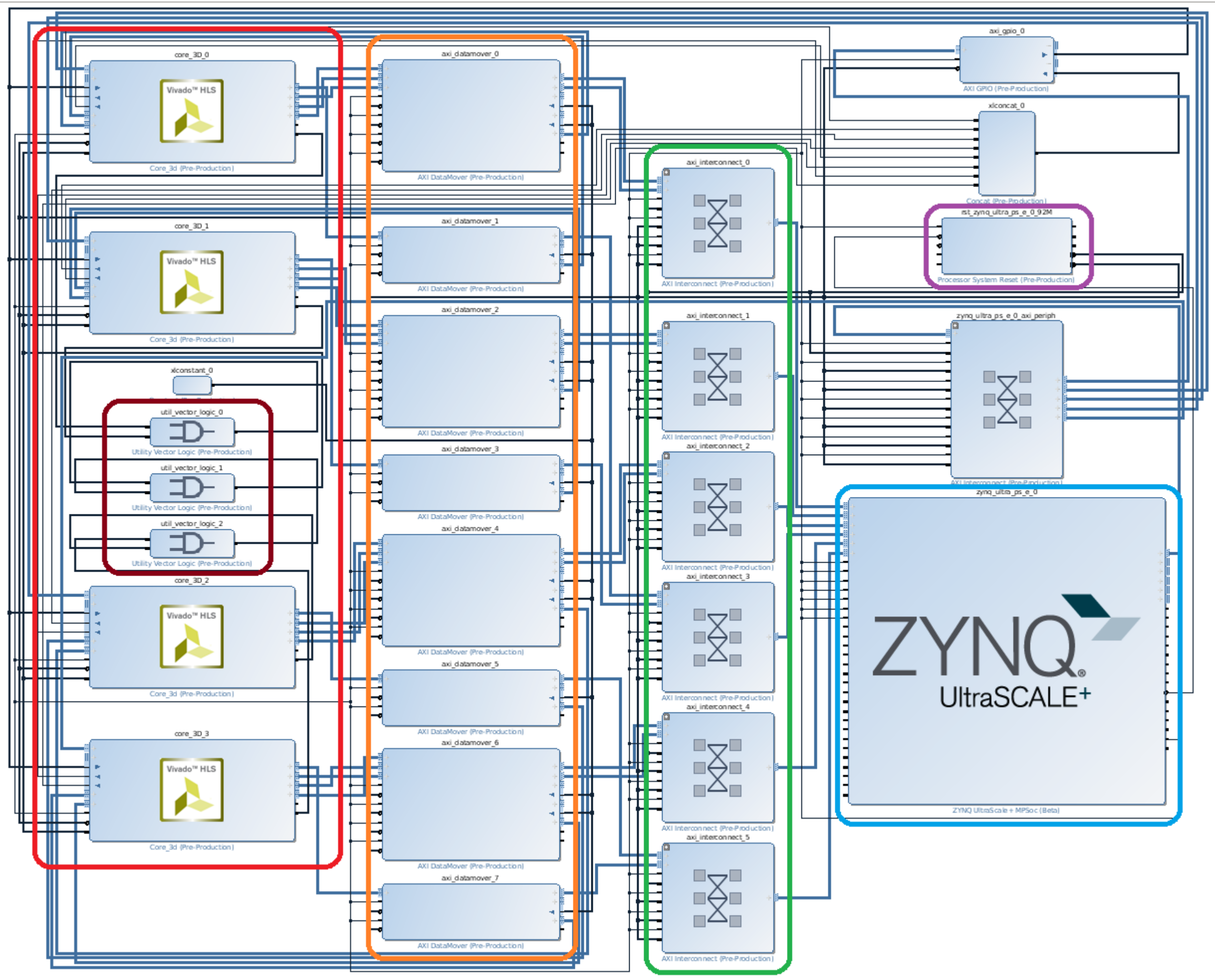


Fig. 3.16 Model 3D Simplified Block Design for 4-core design from Xilinx Vivado.

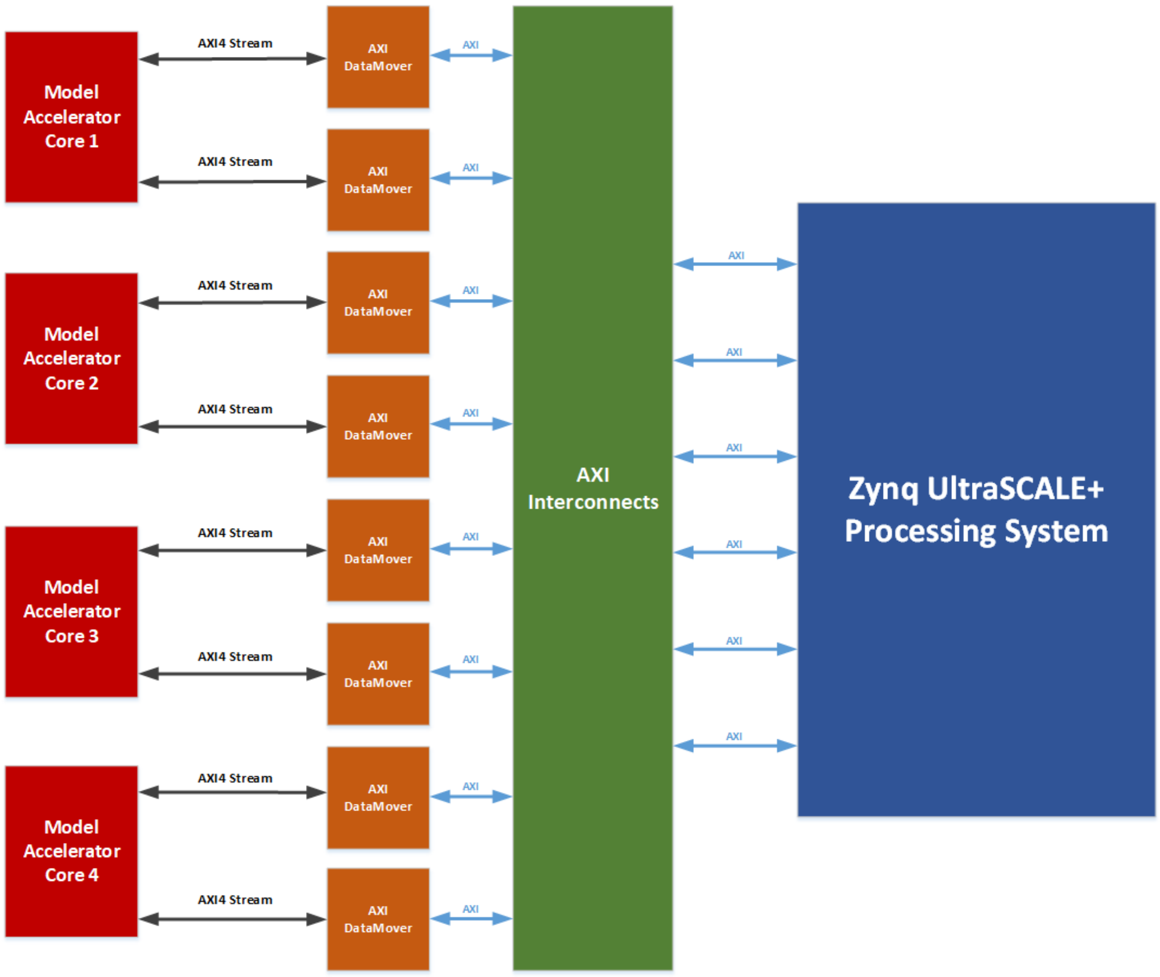


Fig. 3.17 Model 3D Simplified 4-Core Block Design.

In the first figure, the red circled modules are the Hardware HLS Accelerators, the orange modules are the AXI DataMovers, the green ones are the AXI Interconnects, the blue module the Zynq PS and the purple is the PS system reset module. In this design there is only one clock domain for the PL, thus there is only one PS system reset. The colors for the second figure are similar to the first one. Also, the brown colored modules are the AND gates that drive the signals *core_this_finish* and *core_others_finish* (3.2.2.1).

In Xilinx SDK a new application was created. The base addresses and the payloads are changed as they are changed in the previous design. For the C and Ctmp buffer the first core reads the images 32-49 and processes 16 images, the second reads 48-67 and processes 17 images, the third reads 66-85 and processes 17 images and finally the fourth core reads 84-98 and processes 15 images.
The corresponding payloads are:

Read payloads - 1st Core

- C, Ctmp payload: $18 * 170 * 140 * 8 = 3427200$ bytes
- Din payload: $16 * 170 * 140 * 8 = 3046400$ bytes

Read payloads - 2nd Core

- C, Ctmp payload: $19 * 170 * 140 * 8 = 3617600$ bytes
- Din payload: $17 * 170 * 140 * 8 = 3236800$ bytes

Read payloads - 3rd Core

- C, Ctmp payload: $19 * 170 * 140 * 8 = 3617600$ bytes
- Din payload: $17 * 170 * 140 * 8 = 3236800$ bytes

Read payloads - 4th Core

- C, Ctmp payload: $17 * 170 * 140 * 8 = 3236800$ bytes
- Din payload: $15 * 170 * 140 * 8 = 2856000$ bytes

Write payload - 1st Core

- C, Ctmp payload: $16 * 170 * 140 * 8 = 3046400$ bytes

Write payload - 2nd Core

- C, Ctmp payload: $17 * 170 * 140 * 8 = 3236800$ bytes

Write payload - 3rd Core

- C, Ctmp payload: $17 * 170 * 140 * 8 = 3236800$ bytes

Write payload - 4th Core

- C, Ctmp payload: $15 * 170 * 140 * 8 = 2856000$ bytes

After the buffer initialization the initialization of each core takes place, by using the corresponding drivers. The maximum iterations for each core is set to 18, 19, 19, 17 respectively. In the final part of code, the initial tumor image is set again with normalized values, by using the C buffer. This final buffer is sent via USB to a host PC, where all images are reconstructed again with the new values. The procedure to send back the final image takes several minutes, the same as the transfer of the initial images. The final images were verified with images that the original software extracted. As it will be described later, this design did not achieve better speed up than the 2-core design, as it would be expected, and the reason is the DDR's bandwidth limit.

3.3 Model 3D Oxygen-Glucose Diffusion-Proliferation

The third and final model analyzed, is the Model 3D Oxygen-Glucose Diffusion-Proliferation that follows the continuum approach and involves three different categories of cancer cells, i.e. the proliferative, the hypoxic and the necrotic cells, as the 2D model but in 3 dimensions. The code follows the same pattern as the previous two models. The following hardware designs of its particular model involve all the knowledge of the previous and combine all the techniques that have been used in the previous architectures. The first part initializes the model, in the second part which is the main part, the process of the model takes place and in the third and final part, the final reconstruction of the images occurs. In the first one there are 9 buffers to be initialized, as in model 2D hardware design, and each buffer is three dimensional. For the purpose of HLS transformation a preprocessing mapping to one-dimensional buffers took place, as in 3D model. The index-address of each pixel can be found by the given formula 3.1. The initialization is succeeded by reading the same 129 sample text files that were used in the simplified 3D model. In the second-main part of the algorithm the model processing is much more complicated than in the Simplified 3D Model, as this process is similar to 2D model processing, but it extends in three-dimensions. Each output pixel, in order to be calculated, needs 7 pixels from each input streaming buffer except the f and D buffer. The pixels needed are the same as in Simplified 3D Model. The output of the model are 8 buffers named mtmp, ntmp, ftmp, gltmp, Ctmp, Htmp, Qtmp, Ntmp. The D buffer is read-only again. The final part reconstructs the 129 images again to grayscale Portable Network Graphics (PNG) files, by reading the C, H, Q and N buffers.

3.3.1 Memory Mapped Design

3.3.1.1 Xilinx Vivado HLS

The transformation of the original software code to HLS format follows the same optimization techniques again but even more strictly as the current model is much more demanding than the previous models. The first design is a memory mapped implementation that helped a lot in verification. Also, by reviewing this design's resource utilization, the next step of optimizations and a new design, with different approach, is decided, as it will be described in 3.3.2.

The interfaces, that are used, are m_axi and s_axilite. The arguments of the core function, that is synthesized, are the buffers and some constant variables, that are needed in calculations. The code uses each buffer as a pointer in DDR memory. These buffers are mapped to AXI Master interface (m_axi). The core is the master to these interfaces in order to exchange data (read/write) when needed. The HLS provides the feature to bundle same interfaces to a single interface. However, this particular feature is not used in this design, thus every master interface has its own bundle, with the corresponding names of each buffer. The rest variables are mapped as slaves via the s_axilite interface and are set from the Processing System (PS) at the start of the application. These interfaces with the main IP control interface use the bundle feature and bundles them to a single s_axilite interface that is named CTRL. In this design every array behaves as a pointer to DDR. Thus, the arrays in the list of the core function are pointers of double data type. The model in the process part needs 7 pixels to read and 1 to write in every iteration. That way the accelerator hits the DDR memory for every transaction (read or write), even when some data are used more than once inside the process and for that reason a new optimization is introduced in this design. The 7 pixels needed for the process of each pixel are stored in local registers located in the FPGA. That way when the data of a pixel are reused in the model, the core does not communicate again with the DDR's memory controller in order to read them, instead it reads the corresponding local register and the data are ready in 1 or 2 clock cycles maximum. This technique is very useful for memory mapped designs to decrease the overall latency.

In this design the only directive, that is used except the interfaces, is the HLS PIPELINE directive. The target II is 1, but achieved 7. The reason, as it was described, is that there is only one port for read/write for all interfaces. For each buffer the core needs to read 7 pixels and write 1 in every clock cycle and the PIPELINE must stall for 7 cycles.

After synthesis the resource utilization report with target clock 5 ns (200 MHz clock frequency) is the following:

Table 3.6 Resource utilization report of Memory Mapped design on optimized 3D Model Oxygen-Glucose Diffusion-Proliferation by Xilinx Vivado HLS

	BRAM_18K	DSP48E	FF	LUT
Total	68	291	82881	72075
Available	1824	2520	548160	274080
Utilization (%)	3	11	15	26

3.3.1.2 Xilinx Vivado Project

After the extraction of the IP core by Vivado HLS, a new project in Xilinx Vivado was created and has the components: imported model-Accelerator IP, Processing System (PS), some interconnects and the Processor System Reset IP. The imported IP core is connected directly to PS via the AXI Interconnects. The Processing System (PS) is set to have all 6 slave High Performance ports active and clock domain at 150 MHz. Each HP port was set to transfer 128 data width because of the problem that was described in the subsection 3.1.2. The AXI Master interfaces are connected to the Slave ports of the AXI Interconnects, thus the AXI Interconnects are the Masters of the HP ports. The Master Interfaces of IP core are 19 in total (9 inputs and 8 outputs) and as it is understood, more than one was connected to each Slave High Performance port. This feature is provided by the AXI Interconnects. The clock frequency for the core IP is the 150 MHz pl_clock that is provided by the PS and the reset is the global reset signal that is provided by the Processor System Reset IP. The target clock frequency is slower than the target frequency in Vivado HLS and the reason is that the 200 MHz cannot be achieved. However, the target clock is decided to be faster in Vivado HLS in order the HLS compiler to create a more demanding hardware and drive the clock signal more optimized. The strategies of synthesis and implementation in Vivado were also changed to achieve the targeted clock. The Synthesis strategy is the "Flow Area Optimized High" [38] that tries to reuse modules and optimize the area utilization of the FPGA by performing general area optimizations including forcing ternary adder implementation, applying new thresholds for use of carry chain in comparators, and implementing area-optimized multiplexers. The implementation strategy, that is used, is "Performance Extra Timing Opt" [39] that tries to achieve better clock frequencies by using an alternate set of algorithms for timing-driven placement during the later stages of implementation (routing stage). The two figures below show the block design from Xilinx Vivado and an abstract block design for the accelerator and the Zynq PS System.

Table 3.7 Xilinx Vivado resource utilization report for 3D Model Oxygen-Glucose Diffusion-Proliferation Memory Mapped

	Block RAM Tile	DSP48E	FF	LUT	CLB Logic Distribution
Total	101	207	142345	94593	24182
Available	912	2520	548160	274080	34260
Utilization (%)	11.07	8.21	25.97	34.51	70.58

The above table 3.7 shows that resources are low but the distribution is high enough and the next streaming design should achieve about the same clock frequency. Moreover the duplicate buffer technique can be utilized as well to achieve better Initiation Interval (II) in Pipeline.

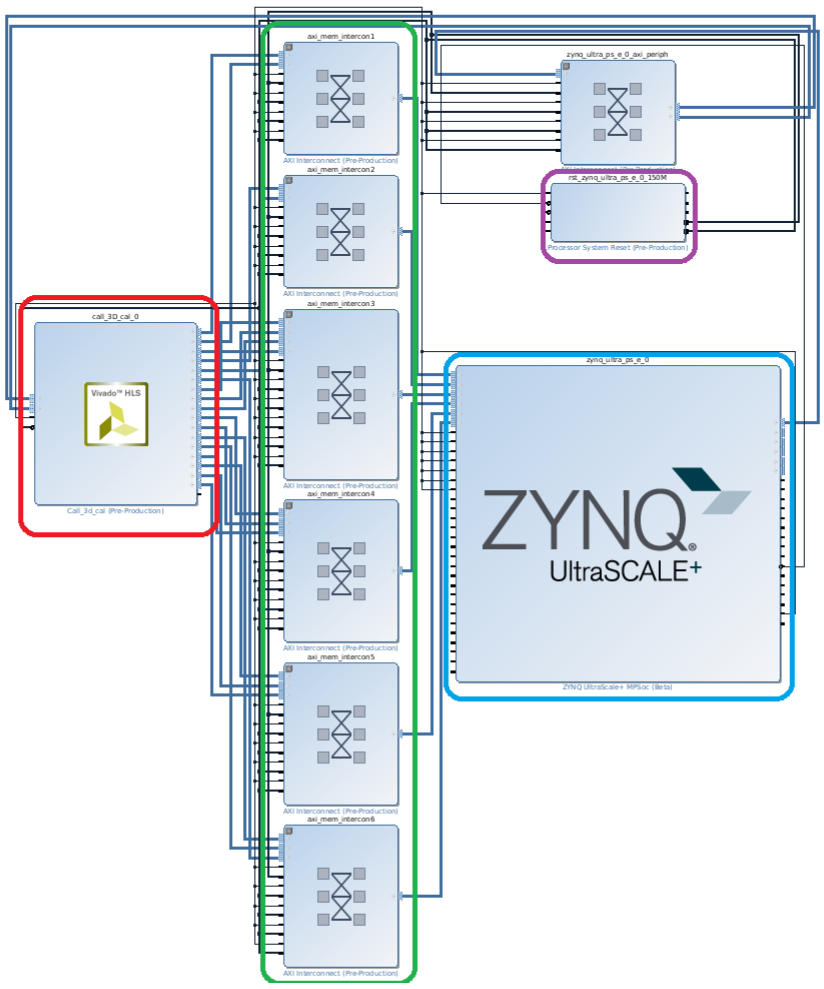

axi_mem_intercon1
AXI Interconnect (Pre-Production)
axi_mem_intercon2
axi_mem_intercon3
axi_mem_intercon4
axi_mem_intercon5
axi_mem_intercon6
zynq_ultra_ps_e_0_axi_periph
rst_zynq_ultra_ps_e_0_150M
Processor System Reset (Pre-Production)
call_3D_cal_0
Vivado™ HLS
Call_3d_cal (Pre-Production)
zynq_ultra_ps_e_0
ZYNQ UltraSCALE+
ZYNQ UltraScale+ MPSoc (Beta)

Fig. 3.18 Model 3D Oxygen Glucose Block Design Memory Mapped design from Xilinx Vivado.

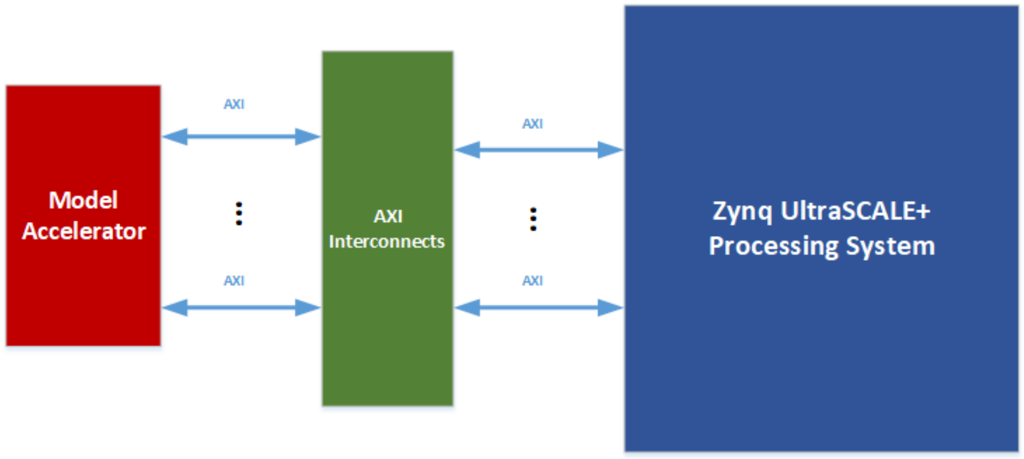


Fig. 3.19 Abstract Model 3D Oxygen Glucose Block Design Memory Mapped Design.

In the first figure, the red circled module is the Hardware HLS Accelerator, the green the AXI Interconnects, the blue module is the Zynq PS and the purple is the PS system reset module. In this design there is only one clock domain for the PL, thus there is only one PS system reset. The colors for the second figure are the same to the first one.

3.3.1.3 Xilinx SDK application

The exported bitstream file by Xilinx Vivado was used to create a new application in Xilinx SDK. In Xilinx SDK the same structure as the original C code of the 3D Model Oxygen-Glucose is followed to implement the application. In the first part of the code, 129 text files are read to initialize the 9 buffers. The difference is that only one file is read that contains the data of the 129 files. The reading is achieved via the UART module that is set by the corresponding drivers. In this application the original C software is also included, in order to run on the ARM Cortex A53 CPU for debugging and verification purposes. The results of software and hardware implementation are compared at the end of the application. In this particular application the total memory for hardware and software implementation that is used is about 2.2 Gigabytes. The Trenz platform provides only 2 Gigabytes and for that purpose a workaround came up to bypass this problem. The solution is simple enough and is based on the re-usability of memory. Another temporary buffer is used in order to store the output data of the buffer, after the ARM's software execution, that is targeted to be compared with the corresponding output buffer, that it is extracted by the hardware. After that, all buffers are initialized again and used by the hardware IP core. In the final part the temporary buffer and the corresponding hardware buffer are compared and the results are printed to a Serial Terminal. This procedure and this design helped a lot in the overall debugging and verification procedure.

3.3.2 Streaming Design

3.3.2.1 Xilinx Vivado HLS

The previous implementation could not achieve speedup, as it will be described in the next chapter. The new design, introduced in this section, followed and based on the approach of the previous models hardware implementations, as it used the streaming interface. As in hardware implementation of 2D Model, there are 9 input arrays and 8 output arrays that are streamed to or from DDR memory, thus 9 AXI DataMovers are needed. The difference is that much more data are streamed, as the 3D Model Oxygen-Glucose extends to three dimensions. The AXI DataMovers' commands follow the same pattern as the previous designs. For this design the same problem, as in simplified 3D Model, occurred. The payload of each stream is much larger than the maximum payload a DataMover can handle. The same workaround is used again to overcome it. Each stream is divided to two streams, which are streaming less data. For the m, n, gl, C, H, Q, N read buffers and the corresponding mtmp, ntmp, gltmp, Ctmp, Htmp, Qtmp, Ntmp write buffers the corresponding payloads are:

Read payloads

- 1^{st} payload: $34 * 170 * 140 * 8 = 6473600$ bytes
- 2^{nd} payload: $33 * 170 * 140 * 8 = 6283200$ bytes

Write payloads

- 1^{st} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- 2^{nd} payload: $32 * 170 * 140 * 8 = 6092800$ bytes

For the rest buffers D, f read buffers and the ftmp write buffer the corresponding payloads are:

Read payloads

- 1^{st} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- 2^{nd} payload: $32 * 170 * 140 * 8 = 6092800$ bytes

Write payloads

- 1^{st} payload: $33 * 170 * 140 * 8 = 6283200$ bytes
- 2^{nd} payload: $32 * 170 * 140 * 8 = 6092800$ bytes

This accelerator's design follows the same structure as the previous simplified 3D model. Firstly the time-step loop comes up. Inside this loop each command is initialized and prepared in order to be sent in every time-step. Then there are three nested loops, which initialize the first two temporary buffers with the values of the first two images of each stream. The purpose of these buffers is to achieve streaming data in or out of the FPGA in every clock cycle (*see 3.10, 3.11*). After that there are three nested loops again, in which the main process of the model takes place.

This design also utilizes the duplicated buffers technique. The purpose of this technique is the achievement of pipeline with II=1. However, The design did not meet the target II. The reason is the Block RAM utilization, as it is very costly to have three copies of each temporary buffer. As it was described in the previous section, some duplicated buffers must be used again in order to achieve the best possible II in the pipeline. Every pixel, as it was described before, needs 7 pixels in order to complete its calculations in every iteration. In every iteration the accelerator can read/write to the temporary buffer of each stream up to two pixels, as the Block RAM is dual port. For that reason the duplicated buffer technique comes in handy to overcome this problem. For each Xz_zero buffer (where X is one of the m, n, gl, C, H, Q, N), there is one more duplicated buffer, which is called Xz_zero_copy1 and has the same data as the original in any given time. The II of pipeline after this procedure achieved 2 clock cycles. In order to achieve the II=1 another duplicated buffer is needed, but it is impossible with the current FPGA and its resources.

The resource utilization report after Xilinx Vivado HLS synthesis is the following with target clock frequency 200 MHz:

Table 3.8 Resource utilization report of streaming 3D Model Oxygen-Glucose Diffusion-Proliferation by Xilinx Vivado HLS with target clock 5 ns

	BRAM_18K	DSP48E	FF	LUT
Total	1211	954	120612	120985
Available	1824	2520	548160	274080
Utilization (%)	66	37	22	44

As it can be seen by the above table, the Block RAM utilization is the barrier in order to achieve II=1. In order to achieve the best possible clock frequency, in the Xilinx Vivado project, the target clock changed to 2 ns and the resource utilization increased as it was expected.

Table 3.9 Resource utilization report of streaming 3D Model Oxygen-Glucose Diffusion-Proliferation by Xilinx Vivado HLS with target clock 2 ns

	BRAM_18K	DSP48E	FF	LUT
Total	1226	686	168897	133553
Available	1824	2520	548160	274080
Utilization (%)	67	27	30	48

3.3.2.2 Xilinx Vivado project

In Vivado a new project was created after the extraction of the core IP (accelerator) from Vivado HLS. The accelerator has 9 input AXI streams and one slave axilite interface, which initializes the basic constant variables, needed for the process of this particular model. The outputs of the accelerator are 8 streams which write to DDR memory and 2 stream commands (read command and write command) for each stream. In the block design the accelerator is connected again to 9 AXI DataMovers. Each channel (read or write) of AXI DataMover is set to Memory Map data width 64 bits, Stream Data width 64 bits, Maximum burst size 256 beats, Width of BTT field 23 bits and Address width 64 bits. The write channel of one DataMover is disabled as the D stream is connected to it and the D buffer is read-only. The PS is set with all 6 High Performance ports enabled and 128 bits data width. In this design there are used two different clock domains for Programmable Logic (PL). The clock domain's frequencies of the PL are set to 150 MHz for the accelerator and the DataMovers and 200 MHz for the I/O (HP ports). The reason two PL clocks are used is for helping the Vivado's implementation algorithms to route the design easier and achieve the target 150 MHz clock frequency for the accelerator. The 200 MHz clock domain, also, accelerates the I/O procedure, as it manages the transfer of the data in a higher rate. The same strategies for synthesis and implementation are used again, as in the 3D Model Oxygen-Glucose Diffusion-Proliferation memory mapped implementation. The bitstream file creation takes about 4 hours, as it is a very demanding design. Most of the time is consumed in the routing process.

The final resource utilization report after the Place Design step is showed in the following table:

Table 3.10 Xilinx Vivado resource utilization report for 3D Model Oxygen-Glucose Diffusion-Proliferation

	Block RAM Tile	DSP48E	FF	LUT	CLB Logic Distribution
Total	688.5	686	210405	124353	29983
Available	912	2520	548160	274080	34260
Utilization (%)	75.49	27.22	38.38	45.37	87.52

The two images below show the block design from Xilinx Vivado and an abstract block design for this architecture.

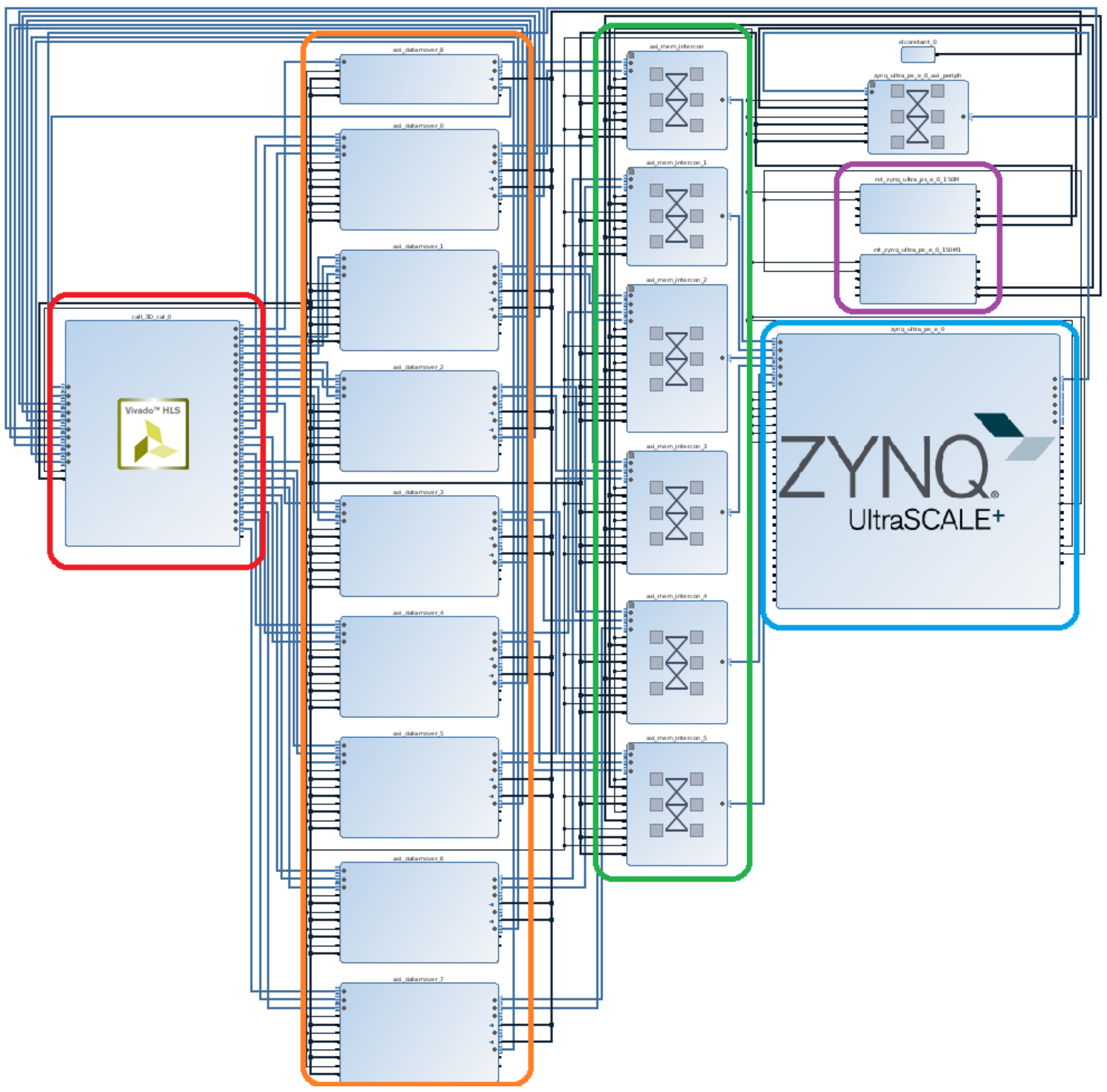


Fig. 3.20 Model 3D Oxygen-Glucose Block Design from Xilinx Vivado.

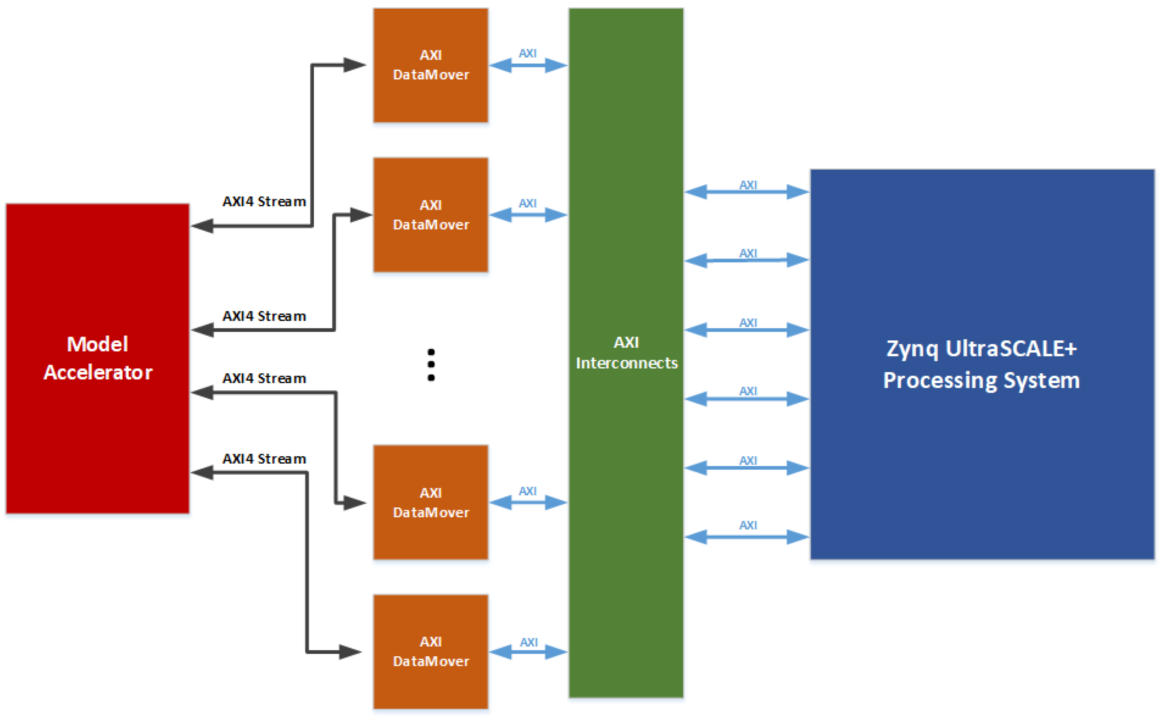


Fig. 3.21 Model 3D Oxygen-Glucose Diffusion-Proliferation Block Block Design.

In the first figure, the red circled module is the Hardware HLS Accelerator, the orange modules are the AXI DataMovers, the green are the AXI Interconnects, the blue module the Zynq PS and the purple are the PS system reset modules. In this design there are two clock domains for the PL, thus there are two PS system reset modules, as it can be seen in the first image. The colors for the second figure are the same to the first one.

3.3.2.3 Xilinx SDK application

In the Xilinx SDK a new application is created again, using the extracted bitstream file. The application follows the same pattern as the previous designs. At the start of the algorithm the read of the image file takes place. In the previous models, in order to transfer the file with the initial values extracted by the MRI images, the UART drivers and a serial terminal is used. This procedure takes about 11 minutes to transfer the file with baudrate 115200. In this design utilized the xilffs drivers which are setting the Generic Fat File System of the Processing System (PS). The xilffs drivers in the PS can handle any external file system. Thus the input file was stored in SD memory card and could be read immediately by the PS in runtime. As it was described, the FPGA could only be programmed by a SD memory card. This feature was exploited in order to read the input file in a very short amount of time. After the reading of the file the initialization of the buffers takes place. Every buffer is one dimensional and its size is the size of the useful data windows, as it was described in this section. The addresses of each buffer were determined with great accuracy in order to not create any problem to the runtime of the application. The last initializations are the payload ones for every stream, according to the previous restrictions. Then the accelerator in the FPGA is set by the corresponding drivers and the process in the FPGA starts. Finally, the output buffers, which are stored in DDR memory, are used to set the normalized values of the final buffer-image. This buffer holds the data of all 129 images and is transferred via UART to host PC, where a C program transform this buffer to 129 png image files. The model was initially verified by the software implementation, which was executed on the ARM CPU, by comparing the checksum of each buffer in the same simulation time, similar to the previous memory-mapped implementation verification method 3.3.1.3. The second verification was done by comparing the extracted images with the original software running on a high-end CPU for large simulations.

Chapter 4

Experiments and Results

In this chapter, there is an extensive description and analysis of each experiment that was conducted with their results. Firstly it will be introduced the servers used, in order to evaluate this Thesis' work and after that the Hardware Platform used for acceleration. After that, it will be shown every model's execution time, power and energy consumption in both platforms, in order, the work of this Thesis, to be evaluated.

4.1 Software Platforms

Two Server Systems are used for the evaluation of the hardware accelerators. These Servers were provided by the Microprocessor and Hardware Laboratory (MHL) of Technical University of Crete. The first Server, named Zeus, was used for running the original software of each model and debugging. Zeus is a Dell PowerEdge R520, with two Xeon E5-2430v2 CPUs, 64GB DDR3 ECC, and Centos 7 Operating System. Each Intel Xeon E5-2430 v2 is clocked at 2.8 GHz, and has 15 MB SmartCache. Each Xeon CPU, also, has 6 physical cores and 12 physical threads (Intel Hyper-threading). The second Server, which is named Kronos, was used to measure the power consumption of the models and is a more powerful machine than Zeus. This server is a Dell PowerEdge R530 (2x Intel Xeon E5-2630v4, 128GB DDR4 ECC, 2x SSD in RAID 1, Centos 7). The CPUs are clocked at 2.2GHz, have 25 MB Cache and Max Turbo Boost frequency at 3.1 GHz. Each CPU has 10 physical cores and 20 threads (Intel Hyper-threading). For the power consumption measurement, the iDRAC version 8 Enterprise Edition module was used. The Dell Remote Access Controller or DRAC is an out-of-band management platform on certain Dell servers. It uses mostly separate resources to the main server resources, and provides a browser-based or command-line interface (or both) for managing and monitoring the server hardware [40]. However, the iDRAC module was available only in the Kronos Server and for that reason only the power consumption in this Server is evaluated against the FPGA.

The experiments, that were conducted, were based on the runtime and power consumption in each platform. In order to evaluate the work, a multi-threaded software implementation for each model had to be created. This is achieved by using the API for OpenMP (Open Multi-Processing). OpenMP is an application programming interface (API) that supports a set of compiler directives, library routines, and environment variables that influence run-time behavior. OpenMP is an implementation of multi-threading, a method of parallelizing whereby a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. In general, OpenMP spawns threads and parallelizes the process of an application. The compiler directive that was used in OpenMP extensively, was the ***#pragma omp parallel for***, which is the combination of two directives the ***#pragma omp parallel*** and the ***#pragma omp for***. The first one spawns a group of threads, while the second one divides loop iterations between the spawned threads. For software model implementation a set of a different number of threads was tested in order to observe the scaling of execution time and power consumption.

In the same OpenMP directives, that were used, another option was enabled, named *schedule* [41]. Four different loop scheduling types (kinds) can be provided to OpenMP. The first one is *static*, which divides the loop into equal-sized chunks. In the case where the number of loop iterations is not evenly divisible by the number of threads multiplied by the chunk size, the OpenMP tries to divide the chunks to each thread as equal as possible. By default, chunk size is $loop\_count/number\_of\_threads$. The second is *dynamic*, which uses the internal work queue to give a chunk-sized block of loop iterations to each thread. When a thread is finished, it retrieves the next block of loop iterations from the top of the work queue. By default, the chunk size is 1. This kind of schedule also, adds some time overhead, in order to schedule every thread. The third is *guided*, which works similar to dynamic scheduling, but the chunk size starts off large and decreases to better handle load imbalance between iterations. The optional chunk parameter specifies them minimum size chunk to use. By default the chunk size is approximately $loop\_count/number\_of\_threads$. The other two are *auto* and *runtime* schedule options, which are not used in these three models. The final command of OpenmMP with the extra option looks like this: ***#pragma omp parallel for schedule(guided) private(param1, param2, ...)***. In each model different set of schedule directives are used.

In order to compare the power consumption, the energy consumption is evaluated, as well. The calculation of the energy is resulted by the following formula.

$$Energy(Joules) = Power(Watts) \times Time(seconds) \tag{4.1}$$

To calculate the energy efficiency, each iteration's power and runtime have to be calculated in each platform (Kronos and FPGA). It is observed, as it will be described later, the power consumption remains stable, for every iteration in each platform. The runtime of each iteration is:

$$Runtime_{(1\ \mathrm{Iteration})} = \frac{Runtime_{(\mathrm{N\ Iterations})}}{N\ Iterations} \tag{4.2}$$

After that the product of the runtime of 1 iteration times the power consumption for each platform is calculated. The efficiency is calculated by dividing these two products. The formula is the following:

$$Efficiency = \frac{(Runtime_{(1\ \mathrm{Iteration})} \times Power\ Consumption)_{(\mathrm{Kronos})}}{(Runtime_{(1\ \mathrm{Iteration})} \times Power\ Consumption)_{(\mathrm{FPGA})}} \tag{4.3}$$

The formula 4.3 calculates the final efficiency of the FPGA platform against the Kronos Server. The efficiency is compared for different number of threads running on Kronos, as well.

4.1.1 Hardware Platform

In hardware all experiments were conducted, for the same number of simulation times with the software implementations, on Trenz platform, model TE0808 UltraSOM (ZYNQ-UltraScale+), which consists the Zynq UltraScale+ *xczu9eg-ffvc900-1-i-es1* chip. In every experiment for each model the results that were extracted are the hardware accelerator execution time, the memory bandwidth with the use of a different number of HP ports and the power consumption. The power measurement is achieved by using a power meter which shows the power consumption of the entire platform, in real-time. The energy consumption for every model is calculated by using the formulas 4.1 and 4.2. The final efficiency is calculated by the formula 4.3. 178

In order to calculate the maximum Read/Write bandwidth using 6 HP ports, a new project was implemented using an HLS simple accelerator. This accelerator has 6 AXI Streams for Read and 6 for Write. This whole project simulates the reading and writing process of all Models' accelerators. In each HP and HPC port, a read and write channel of an AXI DataMover is connected. The PL side and I/O side (ports) are clocked to 200 MHz. The design showed the **maximum practical bandwidth** of the platform. This project was used to compare the Bandwidths of every hardware implementation of this Thesis by modifying it to the needs of every model's accelerator. The Memory Bandwidth *(GigaBytes/second)* for read and write for each experiment was calculated by the following formulas.

$$ReadBandwidth = (TotalReadData)/(ExecutionTime) \tag{4.4}$$

$$WriteBandwidth = (TotalWriteData)/(ExecutionTime) \tag{4.5}$$

The next table shows the maximum practical bandwidth for different number of streams and HP ports. The project modified having 6 AX4 Streams and 6 HP ports (4 HP and 2 HPC), 4 AXI4 Streams and 4 HP ports and 2 AXI4 Streams and 2 HP ports.

Table 4.1 Maximum Practical Bandwidth for different number AXI4 Streams and HP ports

Number of Streams and HP Ports	Read (GB/s)	Write (GB/s)	Total (GB/s)
2 Streams - 2 HP ports	3.19	3.19	6.38
4 Streams - 4 HP ports	5.23	5.23	10.46
6 Streams - 6 HP ports	5.33	5.33	10.66

The table 4.1 shows that the use of more ports can actually give better bandwidth, thus better performance to and accelerator using them. It is worth noting that the HPC ports do not give significant overall I/O throughput and this is shown by the results of the 4 Stream-project and the 6 Stream-project, in which their main difference is the use of 2 extra streams connected to the 2 available HPC ports. It is, also, worth noting that the full Bandwidth is distributed between the Read and Write channels. In the work of Paolo Gorlani, in his Master Thesis [42], the Trenz platform's bandwidth is measured with the same method and exported similar results.

4.2 Results

4.2.1 Model 2D Oxygen-Glucose Diffusion-Proliferation

4.2.1.1 Software Results

The experiments for the software version of the model, were conducted with different number of threads and model simulation times. The software model simulation was set with time-step on the real glioma 0.001 days and the spatial grid dimensions are dx=dy=1mm. The first experiments targeted the observation of the scaling of the runtime of the model in a particular simulation time on a CPU based system. For that reason, the simulation for 30 days (30000 time-step iterations) was used for benchmarking the software, as it takes about 6 minutes on Zeus Server and the scaling of the runtime could be observed easily. The number of threads that were used were 1, 2, 4, 8, 10, 12, 20, 24, 40 and 48. The OpenMP schedule directives used, are *guided* for main kernel code, where the execution takes place, and *static* for the write-back to the input arrays. The results of the execution time of this particular model with different number of threads are shown below:

Table 4.2 Software execution times for Model 2D Oxygen-Glucose Diffusion-Proliferation, for 30 days - 30000 iterations, with different number of threads on Zeus

30 Days - 30000 Iterations	
Threads	Execution Time (seconds)
1	296.14
2	164.67
4	89.22
8	47.14
10	38.07
12	33.05
20	28.18
24	26.11
40	27.42
48	28.61

Until the number of 8 threads, as it can be seen, the execution time scales down in a linear way with respect to the number of threads used. After the 8 threads the execution time does not scale the same way as the previous runs, as it is shown by the table 4.2. Although it was not used any profiling tool, it could be assumed that the memory bandwidth is the bottleneck for this application after the 8 threads. For the software implementations the best performance is achieved when 24 threads run on parallel. After that the runs with number of threads more than 24 increase the execution time. The reason is that the Server System has 24 physical and Hyper-Threading threads, thus the 40 and 48 threads increase the execution time by a very small factor, as it is more costly to spawn and schedule new threads.

In order to measure the power and energy consumption of the 2D Model Oxygen-Glucose software, the Kronos Server was used. All the previous runs were conducted again and the new execution times were extracted, with power consumption. The power consumption was measured by the iDRAC module. As it was observed the power consumption remained stationary for the total execution time of the main kernel code (tumor simulation). The next table shows the execution time, power and energy consumption for different number of threads.

Table 4.3 Model 2D Oxygen-Glucose Diffusion-Proliferation software power and energy consumption for different number of threads

Threads	Execution Time (seconds)	Power (Watts)	Energy (Kilojoules)
1	183.653	131	24.05
2	107.982	139	15.01
4	66.366	149	9.88
8	38.128	161	6.13
10	31.823	169	5.37
12	28.058	178	4.99
20	19.622	208	4.08
24	19.025	210	3.99
40	**16.254**	**215**	**3.49**
48	18.863	213	4.01

By observing the above table, the least power that could be consumed by the server, is by using only 1 thread. However, the total energy that is spent is much more than exploiting the computing power of more threads. Kronos in idle consumes about 120 Watts per hour and the usage of only 1 thread does not increase the overall power that much. The most efficient number of threads, that could be used in parallel, is 40 as it is very clear from the above table. Kronos uses all its physical and Hyper-threading threads and the overall energy is less than the other runs, because the execution time is the shortest, as it can be calculated by the formula 4.1. However, the total energy is not even close to the energy that is spent by the FPGA to complete the same number of iterations.

4.2.1.2 Hardware Accelerator Results

In this model, the execution time, for the hardware accelerator, was extracted for different model simulation times. The time step of the model simulations on the real glioma is $dt = 0.001$ days while the spatial grid dimensions are $dx = dy = 1$mm. All these settings are the same as the software implementation, in order to verify and evaluate the hardware design. The *tmax* was 30, 90, 180 and 365, which are 1 month, 3 months, 6 months and 1 year respectively. The total iterations for each *tmax* with the particular time-step is 30000, 90000, 180000 and 365000 respectively. The execution times, for each of these number of iterations, are shown below, in the next table.

Table 4.4 Hardware Accelerator execution times for Model 2D Oxygen-Glucose Diffusion-Proliferation, in different model simulation times

Number of Days	Iterations	Execution Time (seconds)
30 days	30000	9.9
90 days	90000	29.4
180 days	180000	59.5
365 days	365000	120.3

As it can be seen, the execution time of the accelerator scales up linearly with respect to the number of iterations and the model simulation time.

The total read data for this model are from the window of the image that the brain is located. This means there are 170 pixels in x axis and 140 pixels in y axis. So the total read data are: $TotalReadData = X * Y * 9\ streams * 8\ bytes * Iterations$. For the total write data are: $TotalWriteData = (X-2) * Y * 8\ streams * 8\ bytes * Iterations$. For this model there are 6 HP ports and the clock frequency for the entire PL is 150 MHz. The results are: $ReadBandwidth = 5.16\ Gigabytes/second$ and $WriteBandwidth = 4.54\ Gigabytes/second$. The project that was implemented, in order to measure the maximum bandwidth of the platform, is used to compare and evaluate the bandwidth of the 2D Model accelerator.

Table 4.5 Comparison of Bandwidth for Model 2D accelerator

	Model 2D accelerator (GB/s)	Maximum Bandwidth (GB/s)
Read Channel	5.16	5.33
Write Channel	4.54	5.33
Total Bandwidth	9.7	10.66

As it can be seen, the accelerator is very close to the practical maximum bandwidth. The accelerator cannot achieve the the maximum bandwidth, as there are 9 streams for input and 8 for output distributed in the 6 HP ports. Some of the ports, as it can be understood, manage more payload and more requests than the others, for Read or Write. It is, also, be concluded that a 2-Core design could not achieve better performance, as the bandwidth is already saturated. A 2-Core design would require the twice the number of AXI DataMovers, as well and that increases even more the workload of each HP port.

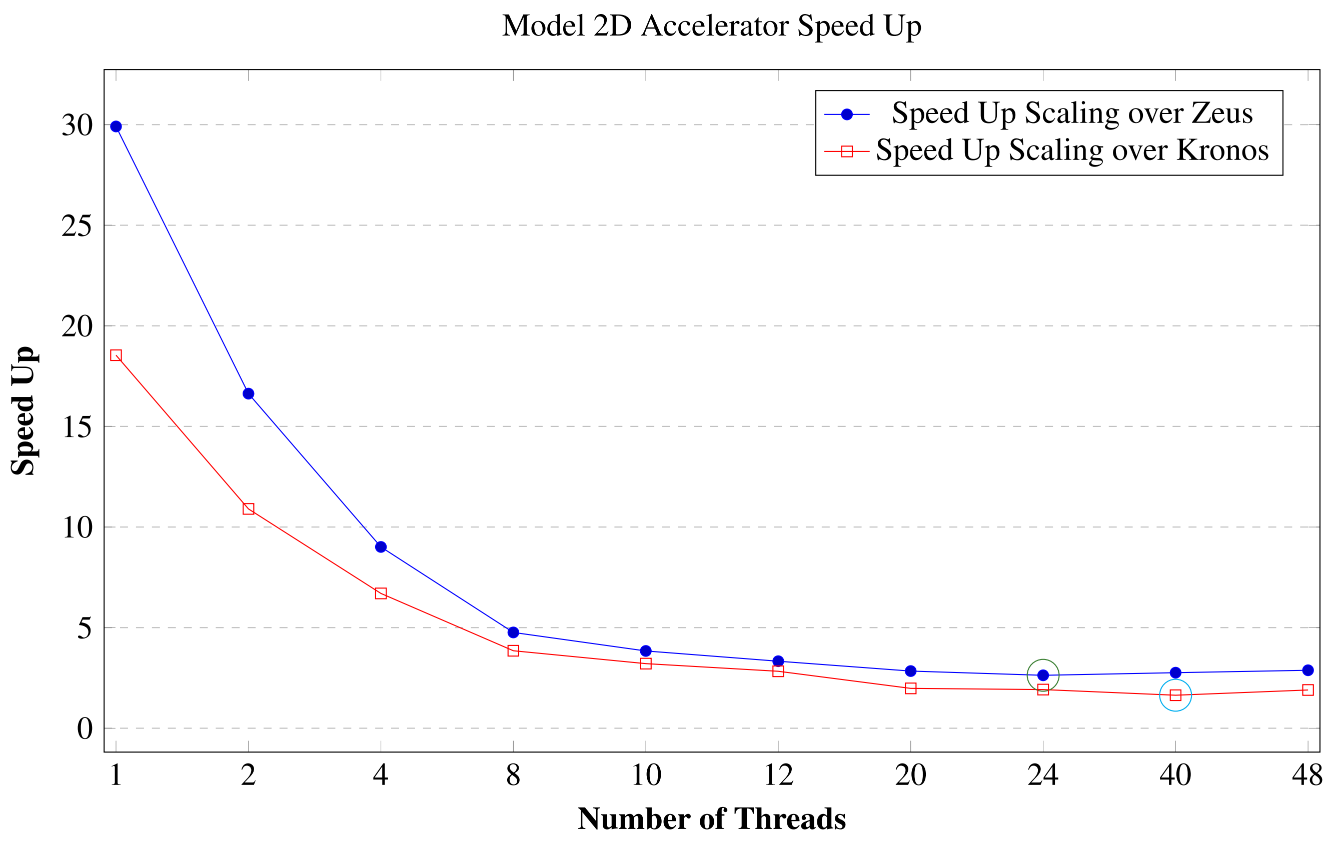
For the power consumption the power meter was used. The FPGA consumes about 25 Watts for 9.9 seconds. That means that the total energy spent is $25\ Watts \times 9.9\ seconds = 247.5\ Joules$.

4.2.1.3 Performance Comparison and Evaluation

The hardware implementation of the model showed very good results, as it achieved speed up over the two Intel Xeon CPUs of the Zeus and the Kronos Server. The speed up according to different numbers of threads in software is shown in the table and the graph below. The speedup is calculated for the tumor prediction simulation of 30 days (30000 iterations), in which the accelerator took **9.9 seconds** to complete.

Table 4.6 Model 2D Oxygen-Glucose Diffusion-Proliferation runtime Speed Up over the two servers

Threads	Runtime Zeus	Speed Up Zeus	Runtime Kronos	Speed Up Kronos
1	296.14	29.91	183.56	18.54
2	164.67	16.63	107.98	10.90
4	89.22	9.01	66.36	6.70
8	47.14	4.76	38.12	3.85
10	38.07	3.84	31.82	3.21
12	33.05	3.33	28.05	2.83
20	28.18	2.84	19.62	1.98
24	**26.11**	**2.63**	19.02	1.92
40	27.42	2.76	**16.25**	**1.64**
48	28.61	2.88	18.86	1.90



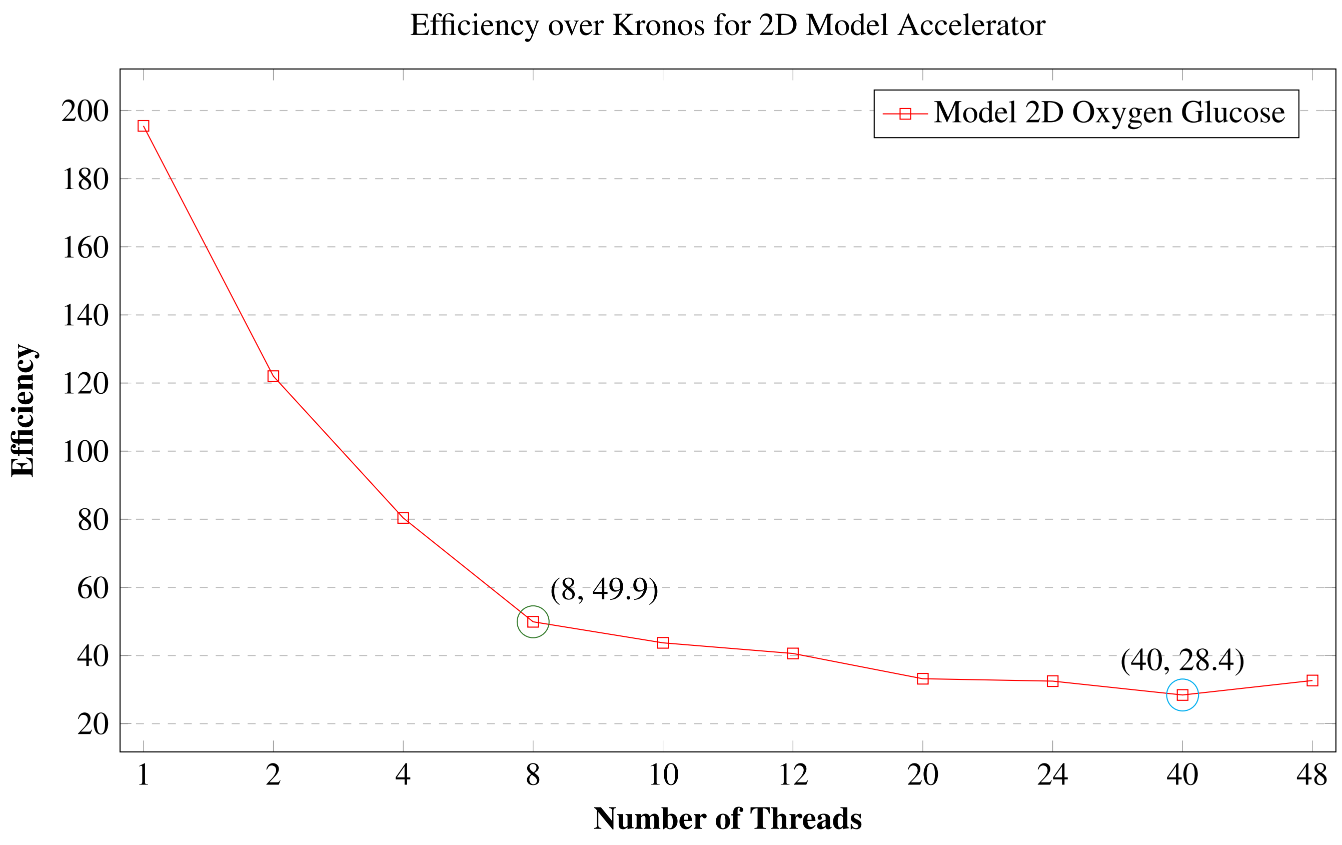
Plot 4.1 Speed Up of Model 2D Oxygen-Glucose Diffusion-Proliferation Hardware Accelerator in 30000 time-steps.

The least runtime speed up over Zeus achieved, is in 24 threads, which is **2.63x**, using 150 MHz clock frequency in the Hardware Accelerator. The accelerator also succeeded a **1.64x** runtime speed up over the most efficient usage of number of threads in Kronos Server. To calculate the efficiency, each iteration's power and runtime have to be calculated in each platform. The power consumption remains stable, for every iteration in each platform. The next table shows the calculation of the efficiency step by step using the formulas 4.2, 4.1 and 4.3.

Table 4.7 Model 2D Oxygen-Glucose Diffusion-Proliferation Hardware accelerators Energy per iteration.

	Hardware Accelerator	Software 40 threads
Runtime - 1 Iteration (Seconds)	0.000165	0.0005418
Power (Watts)	25	215
Energy (Joules) / Iteration	0.0041	0.1164
Efficiency	**28.4**	-

The final efficiency is 28.4x over the Kronos Server. However, this is the least efficiency that could be achieved. The FPGA is even more efficient than Kronos for different number of threads. The efficiency for different number of threads over the Kronos server is shown in the next graph.



Plot 4.2 Efficiency over Kronos. The circled values show efficiency against a high-end PC and a high-end server computer (Kronos).

It is very clear that a high-end server system, like Kronos, is not accessible by any user. A modern PC usually has up to 8 threads (Hyper-Threading). Moreover, such a CPU with 8 threads usually has higher clock frequencies, thus the runtime for 8 threads may be changed to such a machine compared to Kronos. However, the higher frequency means more power consumption. As it is shown above the efficiency over 8 threads is close to **50x**. For that reasons, the FPGA implementation is the most efficient solution for the Model 2D Oxygen-Glucose.

4.2.1.4 Simulation Results

As it is demonstrated by the images of 4.1 the initial tumor (first column) contains only proliferative cells (white), while after one month (second column), a hypoglycemic cell-population appears in the central part of the tumor (dark gray), which is surrounded by a hypoxic region (gray). Two months later (three months after the detection, (third column) a necrotic core (darker gray) has already been developed as a result of hypoxic and hypoglycemic-cells death. This core is surrounded by a hypoxic along with a hypoglycemic zone and the outward proliferative region. Six months after the assumed detection (fourth column) the tumor has significantly expanded, while the discrete zones are depicted. Finally, after one year (fifth column), the tumor reaches a steady state, where the different areas still remain distinct and the necrotic region occupies most of tumor body.

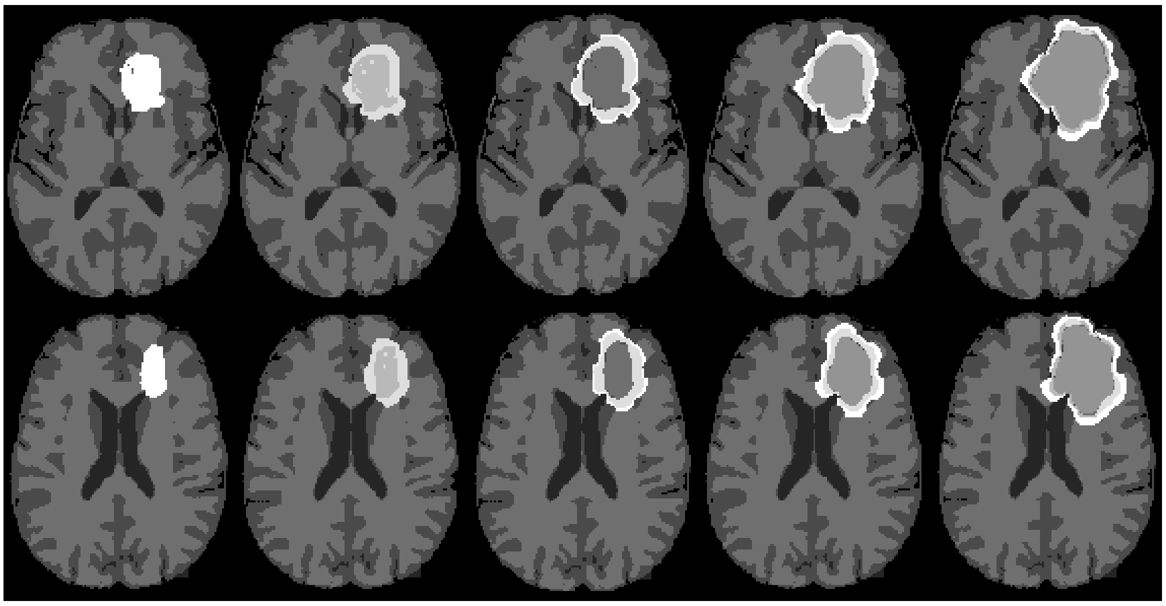


Fig. 4.1 Simulation results of the nutrients 2D Model on a medium-diffusion / medium proliferation tumor. Lines: two indicative frames. Columns: Initial tumor (1st), evolution after: 1 (2nd), 3 (3rd), 6 (4th) and 12 (5th) months.

4.2.2 Model 3D Simplified

4.2.2.1 Software Results

The software application was optimized again, for both servers, using OpenMP. The experiments were conducted for a different number of threads and the 30 days simulation was used for benchmarking the software. The spatial grid dimensions are $dx = dy = dz = 1$ and time-step $dt = 0.001$ (30000 iterations). The number of threads that were used, in both servers, were 1, 2, 4, 8, 10, 12, 20, 24, 40 and 48. The OpenMP schedule directives used, are *static* for main kernel code, where the execution takes place, and *static* for the write in the input arrays. The results for the Zeus server are shown below:

Table 4.8 Software execution times for Model 3D Simplified, for 30 days - 30000 iterations, with different number of threads on Zeus

30 Days - 30000 Iterations	
Threads	Execution Time (seconds)
1	671.00
2	359.13
4	216.73
8	209.64
10	183.53
12	177.99
20	109.95
24	129.78
40	138.85
48	147.53

Table 4.8 shows that this model's runtime does not scale down in a linear way with respect to the number of threads in Zeus server, like the 2D Model. This software is modeling the tumor expansion with a much simpler algorithm, thus less work per thread. So, every thread finishes its task very quickly. The scheduler of the OpenMP here can not decide the load for every thread in the most optimal way for a small number of threads and for that reason, the first 6 runs do not scale as expected (in a linear way). After the 20 thread-run, the runtime is increased. This behavior is expected, as one or two threads are being used by the Operating System and the maximum Hyper-threading is 24. That way, more threads are used than the whole system can provide. This procedure slows down the application, as it has to spawn and schedule new threads.

The same experiments were conducted again for the Kronos Server. These experiments are targeting to measure the power and the energy consumption of the software application. The power consumption remained stable for the total execution time, of the core kernel, of the application. The next table shows the execution time, power and energy consumption for a different number of threads on Kronos Server.

Table 4.9 Model 3D Simplified software execution time, power and energy consumption for a different number of threads in 30000 iterations.

Threads	Execution Time (seconds)	Power (Watts)	Energy (Kilojoules)
1	553.032	133	75.5
2	346.997	141	48.9
4	187.178	160	29.9
8	107.985	171	18.4
10	97.671	190	18.5
12	89.047	193	17.1
20	74.299	218	16.1
24	77.991	219	17.0
40	**70.472**	**222**	**15.6**
48	84.252	200	16.8

Table 4.9 shows that the model's execution time scales down in a nearly linearly way with respect to the number of threads used, until the 8 threads. The linearity stops after the 8 threads, where the application scales down in a much slower rate. This Model, as it is already described, is more simple than the 2D Model, as it simulates only the proliferative cells expansion. Kronos is a more powerful machine and so, the software's best performance is achieved faster. The best speed up is achieved by using 40 threads. Like the Zeus server, the use of more threads than the available do not give extra runtime speed up. For that reason, by calculating the energy by the formula 4.1, the energy consumption is the lowest in 40 threads, as Kronos works in the most efficient way.

4.2.2.2 Hardware Accelerators Results

For this model there are three different implementations, which are targeting further speed up of the original software. Each implementation used different numbers of accelerators, which vary from 1 to 4 cores. The first implementation uses 1 core, the second 2 and the third 4. The last two implementations (2 cores and 4 cores) target to divide the process to the corresponding cores. These two implementations utilize much more bandwidth of DDR memory than the single-core, thus they have larger I/O transactions than the first one (single-core). The time-step that is used for every implementation is $dt = 0.001$ days and the $tmax$ varies from 1 month to 1 year, as in the 2D model. The execution times (seconds) for the single-core, 2-core and 4-core implementations are shown below.

Table 4.10 Hardware Accelerators' execution times for different number of cores, for Model 3D Simplified, in different model simulation times

Number of Days	Iterations	Single-core	2-core	4-core
30 days	30000	239.1	132.2	140.5
90 days	90000	717.9	396.6	421.5
180 days	180000	1435.8	793.2	843.1
365 days	365000	2911.4	1608.4	1709.4

The execution time scales up linearly with respect to the number of iterations, for every design of the 3D Model Simplified. The max power consumption of the whole system (FPGA - ARM CPU - DDR memory - Cooling System) for each implementation, measured, is 20.5, 22.2 and 23 Watts, respectively.

The memory bandwidth was measured for every implementation, for this model, as well. For the single-core design, the Read/Write payload is calculated $X\ pixels \times Y\ pixels \times Number\ Of\ Slices \times Number\ Of\ Iterations \times 8\ Bytes\ for\ every\ pixel$. As it was described in a previous section, the useful data are located in the 32-98 slices and between 40-210 lines and 50-190 columns. Thus the Read/Write payload for Read/Write streams are $140 \times 170 \times 67 \times Number\ Of\ Iterations \times 8$. The read-only stream is $140 \times 170 \times 65 \times Number\ Of\ Iterations \times 8$. The Write stream is $140 \times 170 \times 67 \times Number\ Of\ Iterations \times 8$. For every iteration the total Read Payload is 37.9 MB and the total Write Payload is 12.4 MB. The Bandwidths for read and write using the formulas 4.4, 4.5 are shown in the next table:

Table 4.11 Comparison of Bandwidth for Model 3D accelerator using 1 core

	Model 3D 1 core (GB/s)	Maximum Bandwidth (GB/s)
Read Channel	3.15	3.19
Write Channel	1.51	3.19
Total Bandwidth	4.66	6.38

The previous project used to measure the Bandwidth of the 6 HP and HPC ports was transformed in order to use only 2 HP ports. The single core implementation uses only 2 HP ports as well. From the above table, it can be seen that the bandwidth of the hardware implementation of the model is very close to the maximum practical one.

The second design uses 2 cores, thus uses twice the number of AXI DataMovers and HP ports. Specifically it uses 4 DataMovers and 4 HP ports. The bandwidth for Read and the Write is the following. The payload for Read is about the same as the single-core design. The difference is that in this implementation in order to preserve the data the slice 34 and 33 are read twice from the two cores, thus the total payload is a bit more. The total Read payload is 38.6 MB. The total payload Write remains the same as the single core implementation, thus 12.4 MB. The Bandwidths for read and write using the formulas 4.4, 4.5 are shown in the next table:

Table 4.12 Comparison of Bandwidth for Model 3D accelerator using 2 cores

	Model 3D 2 cores (GB/s)	Maximum Bandwidth (GB/s)
Read Channel	5.78	5.23
Write Channel	2.73	5.23
Total Bandwidth	8.51	10.46

The table 4.12 shows that the design almost doubled the bandwidth, as it is expected. The runtime, also, affected as it is nearly 2x faster than the Single-Core (table 4.10). Although the Read Bandwidth of the 2-core implementation is more than the Maximum theoretical, the useful Bandwidth resides in the sum of Read and Write channel. Thus the maximum theoretical Bandwidth is 10.46 GB/s and the hardware implementation achieved 8.51 GB/s. The Bandwidth loss occurred because of the amount of data that the AXI DataMovers, the DDR and the AMBA have to transfer in every cycle. The AXI DataMovers have to be initialized and send a new request to DDR every 256 64-bit data (maximum burst size 256). Every request takes about 50-60 clock cycles of PL.

The 4-core targeted the further parallelization of the process of this Model and use the maximum bandwidth of the platform. The total Read Data for this design is larger than the previous' one, because of the extra images that have to be reread from the 4 cores. Specifically, the slices 16, 17, 33, 34, 50 and 51 need to be read twice from the accelerators in order to preserve the data. Thus the total Read payload is 40.2 MB. The total Write data remain the same as the previous designs, thus 12.4 MB. The Bandwidths for read and write using the formulas 4.4, 4.5 are shown in the next table:

Table 4.13 Comparison of Bandwidth for Model 3D accelerator using 4 cores

	Model 3D 4 cores (GB/s)	Maximum Bandwidth (GB/s)
Read Channel	5.6	5.33
Write Channel	2.6	5.33
Total Bandwidth	8.2	10.66

By observing the above table 4.13, it can be concluded that the use of 6 ports did not increase the performance, instead it reduced it. The reason is that there are more Read/Write

Channels for every port. The 4-core design has 8 Read and 4 Write channels. All these channels are distributed among the 6 HP ports. Each HP port and the memory controller have more requests and payload from DataMovers to manage, as 4 of these ports are connected with at least two DataMovers. This architecture reduces the maximum performance of the I/O and the result is the speed down of the application compared to the 2-core design. The next notable information is that the HPC ports offer a very small bandwidth. This can be seen by comparing the bandwidth of 2D Model implementation 4.5 with the 4-core bandwidth 4.13.

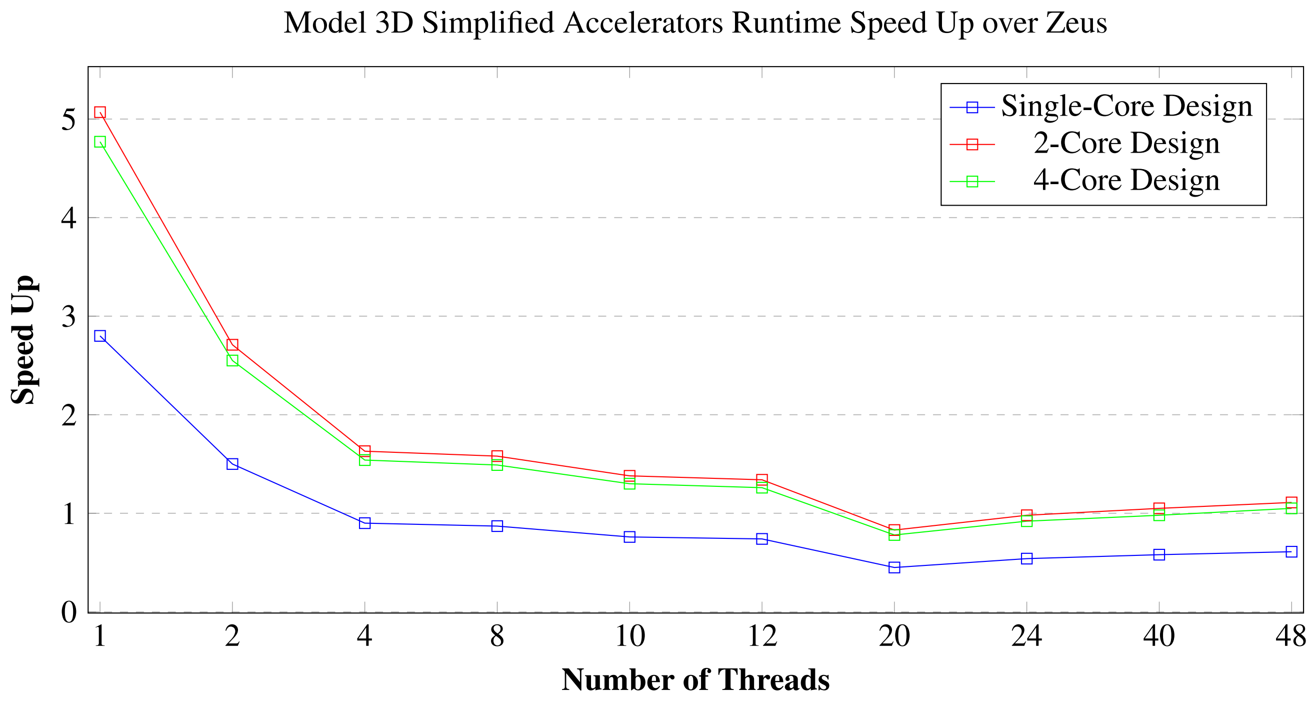
4.2.2.3 Performance Comparison and Evaluation

The hardware implementation did not achieve runtime speed up over Kronos. However, as it will be shown in this section, the FPGA remains the most efficient platform. The runtime speed up according to a different number of threads in software is shown in the table below.

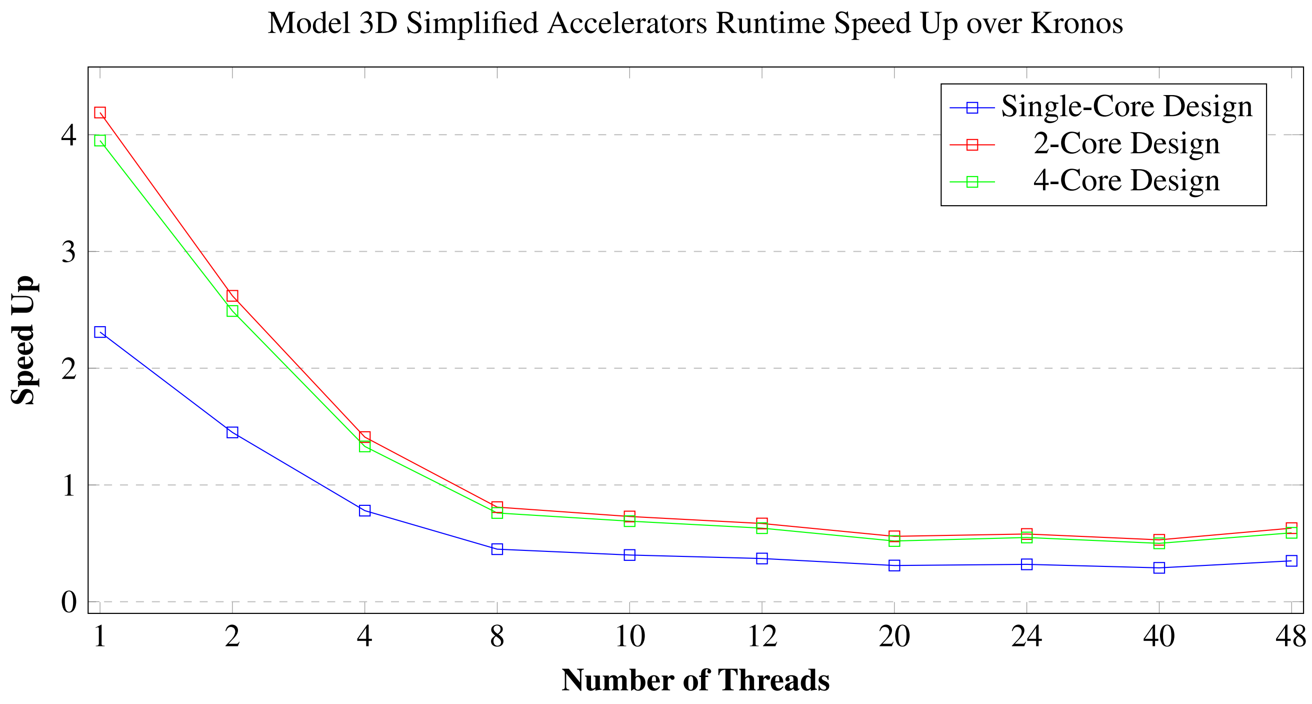
Table 4.14 Model 3D Simplified runtime Speed Up over the two servers

	Runtime Servers (seconds)		Single-Core		2-Core		4-Core	
Threads	Zeus	Kronos	Zeus	Kronos	Zeus	Kronos	Zeus	Kronos
1	671.00	553.03	2.80	2.31	5.07	4.19	4.77	3.95
2	359.13	346.99	1.50	1.45	2.71	2.62	2.55	2.49
4	216.73	187.17	0.90	0.78	1.63	1.41	1.54	1.33
8	209.64	107.98	0.87	0.45	1.58	0.81	1.49	0.76
10	183.53	97.67	0.76	0.40	1.38	0.73	1.30	0.69
12	177.99	89.04	0.74	0.37	1.34	0.67	1.26	0.63
20	109.95	74.29	0.45	0.31	0.83	0.56	0.78	0.52
24	129.78	77.99	0.54	0.32	0.98	0.58	0.92	0.55
40	138.85	70.47	0.58	0.29	1.05	0.53	0.98	0.50
48	147.53	84.25	0.61	0.35	1.11	0.63	1.05	0.59

The table 4.14 shows the software runtime in seconds for both servers and the corresponding speedup for each accelerator against different number of threads. It is clear that this Model, as it is simpler than the 2D model, the FPGA achieves speedup over the servers for fewer number of threads than the previous Model. The 2-core design is the hardware design that is the most efficient amongst the other two implementations (Single-Core and 4-core). This design can achieve speedup even against 12 cores of Zeus server and 4 cores of Kronos. Below there are graphs that illustrate the speed up for every accelerator against both servers.



Plot 4.3 Speed Up of Model 3D Simplified Single, 2 and 4 Core Hardware Accelerators over Zeus in 30000 time-steps.



Plot 4.4 Speed Up of Model 3D Simplified Single, 2 and 4 Core Hardware Accelerators over Kronos in 30000 time-steps.

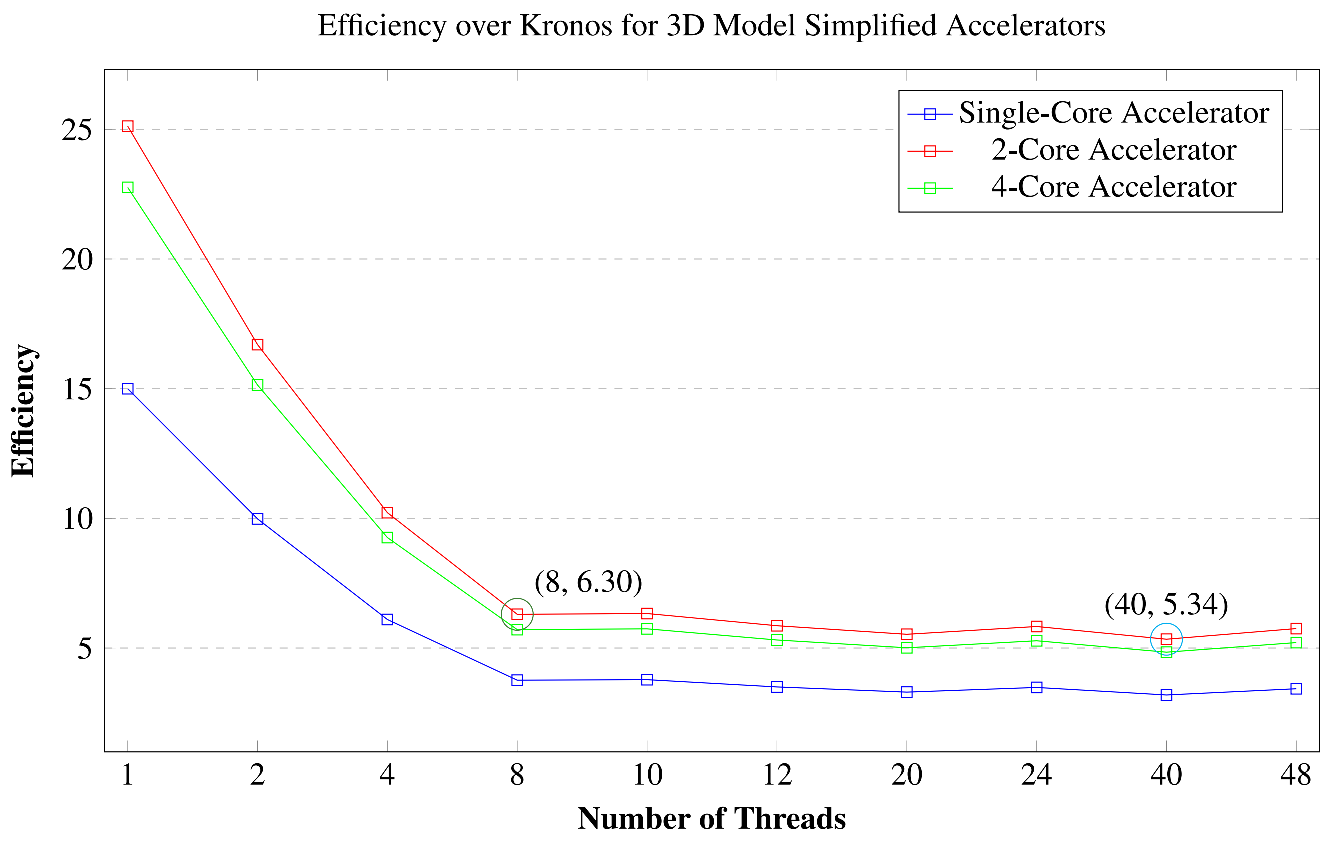
Firstly the plot 4.3 shows that the two accelerators 2-core and 4-core are very similar. However the 2-core design is better than the 4-core, as it uses the full possible bandwidth, for this Model only, thus the runtime is better. Then the plot 4.4 shows the same results as the previous plot but with the corresponding scale for the Kronos server.

The 2-core design is concluded to be the most optimal. Moreover it can achieve significant speedup against a thread in both servers and even more threads. Although the servers are faster for more than 4 or 8 threads, the 2-core hardware accelerator remains more efficient. The efficiency can be calculated by the formulas 4.2 and 4.3. The runtime, the energy and the efficiency for 1 iteration against the Kronos using 40 threads, is shown in the next table.

Table 4.15 Model 3D Simplified Hardware accelerators Energy per iteration.

	Single-Core	2-Core	4-Core	Software 40 threads
Runtime - 1 Iteration (Seconds)	0.0079	0.0044	0.0046	0.0023
Power (Watts)	20.5	22.2	23	222
Energy (Joules) / Iteration	0.1619	0.0976	0.1058	0.5106
Efficiency	3.1x	**5.3**	4.8x	-

As it is very clear, the most efficient hardware implementation is the 2-core design which achieves **5.3x efficiency over the Kronos Server**. It is worth noting that this is the least efficiency could be achieved. For fewer number of threads the efficiency of the hardware accelerator is even better and this is shown in the next graph.



Plot 4.5 Efficiency over Kronos. The circled values show efficiency against a high-end PC and a high-end server computer (Kronos).

The graph 4.5 shows that all accelerators, even the single-core, are more efficient than the Kronos. The 2-core design, which is the best accelerator for this application, can achieve 6.3x efficiency over an 8-thread machine. Such a machine is a modern high-end PC, which is more easily accessible.

From the above, it is concluded that the FPGA implementation is the most efficient solution for this Model, as well.

4.2.2.4 Simulation Results

In the next figure the simulation results for different number of days are illustrated. The MRI images are extracted by the 2-core Hardware application.

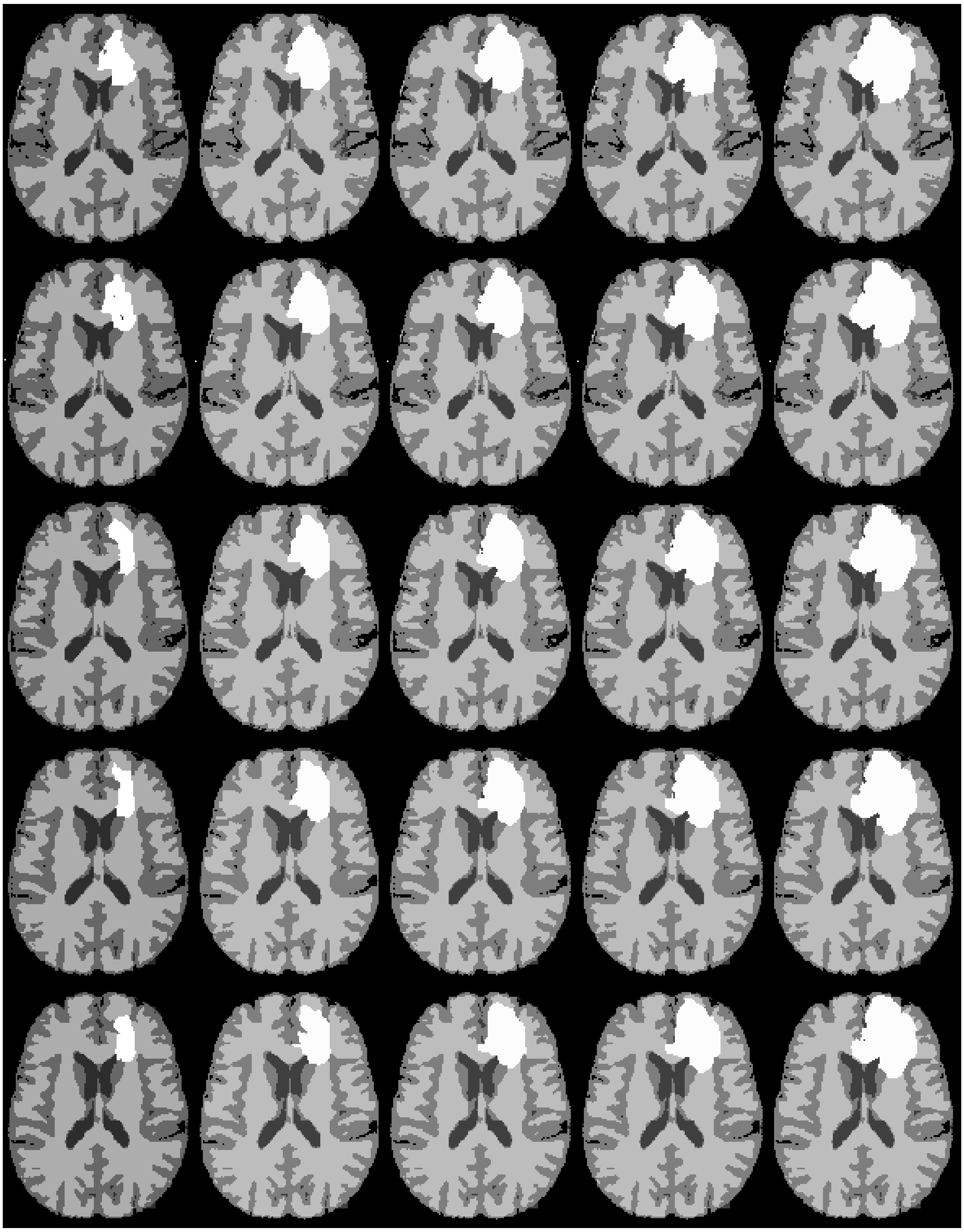


Fig. 4.2 Simulation results of the simple 3D model on a medium-diffusion / medium proliferation tumor. Lines: 5 consecutive frames. Columns: Initial tumor (1st), evolution after: 1 (2nd), 3 (3rd), 6 (4th) and 12 (5th) months.

Figure 4.2 illustrates the tumor expansion on five consecutive frames (lines) of the MRI sequence concerning four different simulation time intervals, namely 1, 3, 6 and 12 months (columns). It can be observed that the tumor tend expanding towards the white matter (lightest brain tissue), as opposed to the gray matter (darker tissue), where the diffusion is lower and the CSF (black regions), where the invasion is substantially reduced.

4.2.3 Model 3D Oxygen-Glucose Diffusion-Proliferation

4.2.3.1 Software Results

The original software was optimized using the OpenMP API, in order to create a multi-threaded application, and the experiments were conducted again to both Servers. The simulation time, that was used for benchmarking, is 30000 iterations (30 Days prediction). The time-step is $dt = 0.001$ days and the spatial grid dimensions are $dx = dy = dz = 1$. In both servers the number of threads, that were utilized, are 1, 2, 4, 8, 10, 12, 20, 24, 40 and 48. The OpenMP schedule directives used, are *dynamic* for main kernel code, where the execution takes place, and *dynamic* for the write in the input arrays. The first results were extracted by the Zeus Server and the execution time is shown in the next table, for different number of threads.

Table 4.16 Software execution times for Model 3D Oxygen-Glucose Diffusion-Proliferation, for 30 days - 30000 iterations, with a different number of threads on Zeus

30 Days - 30000 Iterations	
Threads	Execution Time (seconds)
1	15870.28
2	10386.54
4	5955.51
8	4071.90
10	3534.01
12	3400.93
20	2966.95
24	2995.73
40	3013.31
48	3074.93

In table 4.16 is shown that the execution time scales down in a linear way with respect to the number of threads used, until the 8 threads, like the 2D Model (4.2). After the 8 threads the execution time scales down much slower. I can be concluded that the application is processing bound until the 8 threads. After that the overall bandwidth needed by each thread make the application I/O bound, thus the application's runtime scales down with lower rate. Again, the best execution time is achieved by using 20 threads. The reason is, the same as in the 2D model (4.2.1.1). The Server System has 24 physical threads, thus the 24, 40 and 48 threads increase the execution time by a very small factor, as it is more costly, for the Operating System, to spawn and schedule new threads. As it is also known, some of the threads are used to maintain the Operating System's functionality.

In Kronos the same experiments, with same parameters, were conducted again, in order to evaluate not only the processing power of Kronos, but its power and energy consumption, as well. The similar behavior, as in Zeus, has been observed again. The difference is that Kronos achieves even better execution time by using even more threads, as it is a more powerful Server. The Power and Energy consumption, as well as the execution time, is shown in the following table.

Table 4.17 Model 3D Oxygen-Glucose Diffusion-Proliferation software power and energy consumption for a different number of threads

Threads	Execution Time (seconds)	Power (Watts)	Energy (Kilojoules)
1	9366.28	130	1217.6
2	6349.58	140	888.9
4	3517.02	177	622.5
8	2099.31	183	384.1
10	1871.96	194	363.1
12	1694.68	210	355.8
20	1329.01	219	291.1
24	1277.95	223	284.9
40	**1207.02**	**226**	**272.7**
48	1222.13	224	273.7

Firstly, the table 4.17 shows that the software scales down with a similar way, as the Zeus Server (4.16), with the difference in the Kronos the runtime is even better. The least runtime is achieved by using 40 threads, because there are 40 available threads in this system. After that, the use of more threads does not help the software to speed up any further. It is also clear, by viewing the same table, that the server consumes the least power when utilizing one thread. However, the total energy that is spent is much more than exploiting the computing power of more threads. The least overall energy is when 40 threads are used. The reason is that the energy is the product of Power and Time 4.1.

4.2.3.2 Hardware Accelerator Results

The 3D Model Oxygen-Glucose Diffusion-Proliferation was the most difficult model to be implmented to a hardware design, as it the most computationally intensive Model amongst the three. This model was based and built by the knowledge of the previous models, in algorithmic and High Performance Computing basis. The first design that was implemented, is the Memory-Mapped design and the second is the Streaming design. The first design's clock domain was clocked at 150 MHz, while the second design utilizes two clock domains. The first one, which is clocked at 150 MHz, is used by the accelerator and the second one at 200 MHz and is used by the I/O of the platform (AXI Interconnects and HP ports). The target of the memory mapped design was not to achieve acceleration, but instead to be used in order to create the next design (Streaming). Also, this design was used to verify and debug the main kernel of the second design's accelerator. The second design (streaming approach) exceeded the expectations, as it achieved runtime and efficiency speed up up to 1.85x and 14x respectively, although it has Pipeline Initiation Interval II=2.

More specifically, the Hardware accelerators' runtime simulations for a different number of iterations, using time-step interval $dt = 0.001$ days and spatial grid dimensions $dx = dy = dz = 1$ mm, are shown in the next table.

Table 4.18 Hardware Accelerators execution times (seconds) for Model 3D Oxygen-Glucose Diffusion-Proliferation, in different model simulation times

Number of Days	Iterations	Streaming Design	Memory-Mapped Design
30 days	30000	653	67050
90 days	90000	1959	201150
180 days	180000	3918	402300
365 days	365000	7944.8	815775

In table 4.18 is shown that the execution time scales up linearly with respect to the number of iterations. It is also very clear that the Memory-Mapped design could not achieve speedup. The memory-mapped design could be more optimized, but as it was already mentioned, this design is only for verification purpose. This design, of course, can never achieve the results of the streaming one. The streaming design's achieved speedup over the memory-mapped one is shown in the next table.

Table 4.19 Streaming Hardware Accelerator speedup over the Memory-Mapped Accelerator

Steaming Speedup		
Number of Days	Iterations	Speedup
30 days	30000	102.6x

The Memory Bandwidth was measured for the streaming model, as well. The total Read-/Write payload is $X \times Y \times Slices \times 8\ bytes \times Number\ Of\ Iterations \times Number\ Of\ Streams$. So, the Read payload is $140 \times 170 \times 67 \times 8 \times Number\ Of\ Iterations \times 9\ Streams$, and the Write payload is $140 \times 170 \times 65 \times 8 \times Number\ Of\ Iterations \times 8\ Streams$. For 30000 iterations the Read and Write payload is about 3.13 and 2.70 TeraBytes, respectively. The same project, which was used in 2D Model to compare the maximum I/O of the 6 HP ports and the achieved Bandwidth, was used again to measure and compare the 3D Model's Oxygen-Glucose Bandwidth. The achieved Bandwidth by this design is shown in the next table.

Table 4.20 Comparison of Bandwidth for Model 3D Oxygen-Glucose accelerator

	Model 3D accelerator (GB/s)	Maximum Bandwidth (GB/s)
Read Channel	5.21	5.33
Write Channel	4.43	5.33
Total Bandwidth	9.64	10.66

As it can be seen, the accelerator is very close to the maximum practical bandwidth. The design loses some of the available bandwidth, because there are 9 streams for Read and 8 streams for Write distributed in 6 HP ports. Some of these ports are connected to multiple different streams and have to manage more requests and data at the same time.

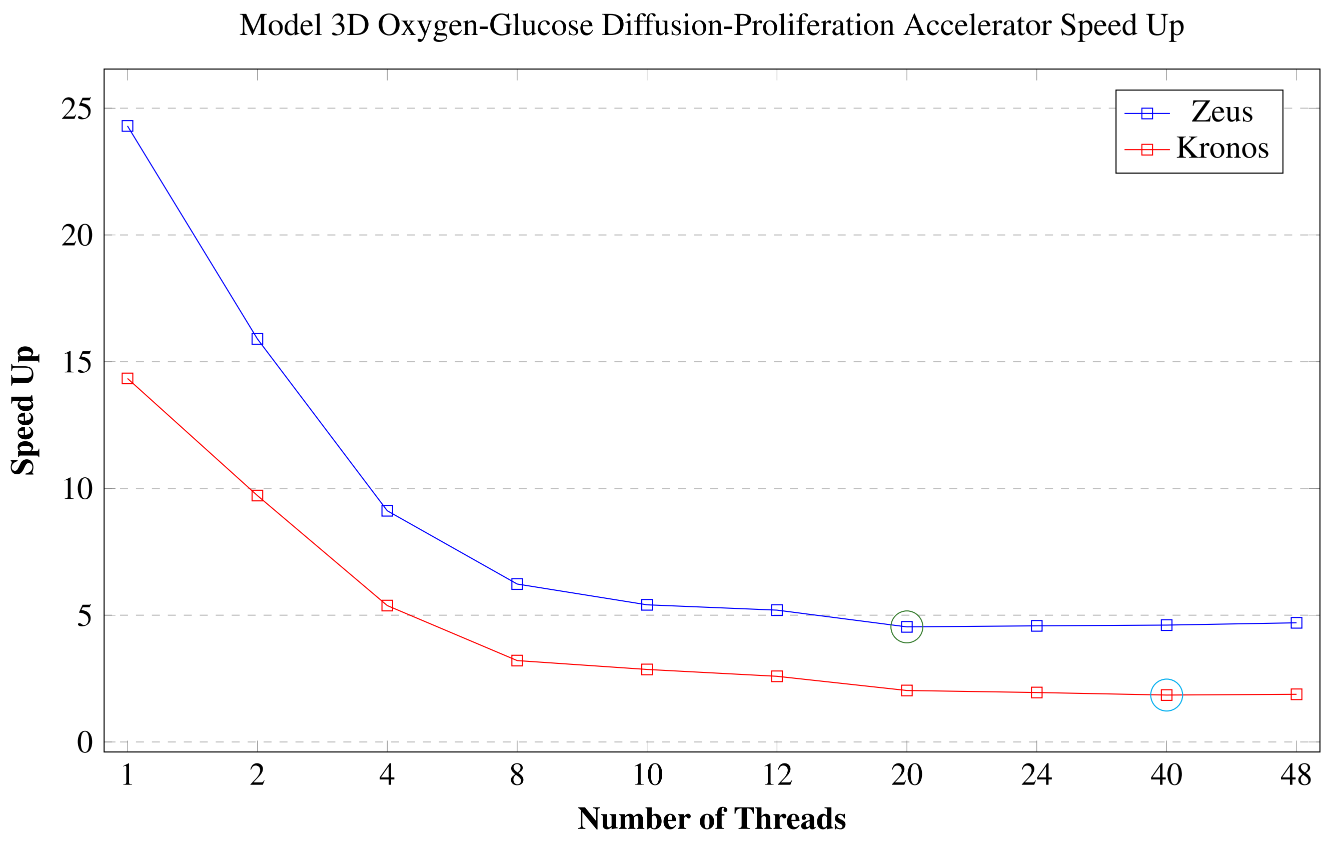
In the measurement of power consumption, the same power meter was used. The power remained steady throughout the whole execution of the Model by the FPGA accelerator. The FPGA system (CPU-FPGA-DDR memory-Cooling System) consumes 30 Watts per hour. The total energy spent, for the 30 days tumor evolution simulation, is $30\ Watts \times 653\ seconds = 19.5\ Kilojoules$.

4.2.3.3 Performance Comparison and Evaluation

The hardware implementation of the model showed very good results, as it achieved speed up over the two Intel Xeon CPUs of the Zeus and the Kronos Server. The speed up according to a different numbers of threads in software is shown in the table and the graph below:

Table 4.21 Model 3D Oxygen-Glucose Diffusion-Proliferation runtime Speed Up over the two servers

Threads	Zeus (seconds)	Speed Up Zeus	Kronos (seconds)	Speed Up Kronos
1	15870.28	24.30	9366.28	14.34
2	10386.54	15.90	6349.58	9.72
4	5955.51	9.12	3517.02	5.38
8	4071.90	6.23	2099.31	3.21
10	3534.01	5.41	1871.96	2.86
12	3400.93	5.20	1694.68	2.59
20	2966.95	**4.54**	1329.01	2.03
24	2995.73	4.58	1277.95	1.95
40	3013.31	4.61	1207.02	**1.85**
48	3074.93	4.70	1222.13	1.88



Plot 4.6 Speed Up of Model 3D Oxygen-Glucose Diffusion-Proliferation Hardware Accelerator in 30000 time-steps.

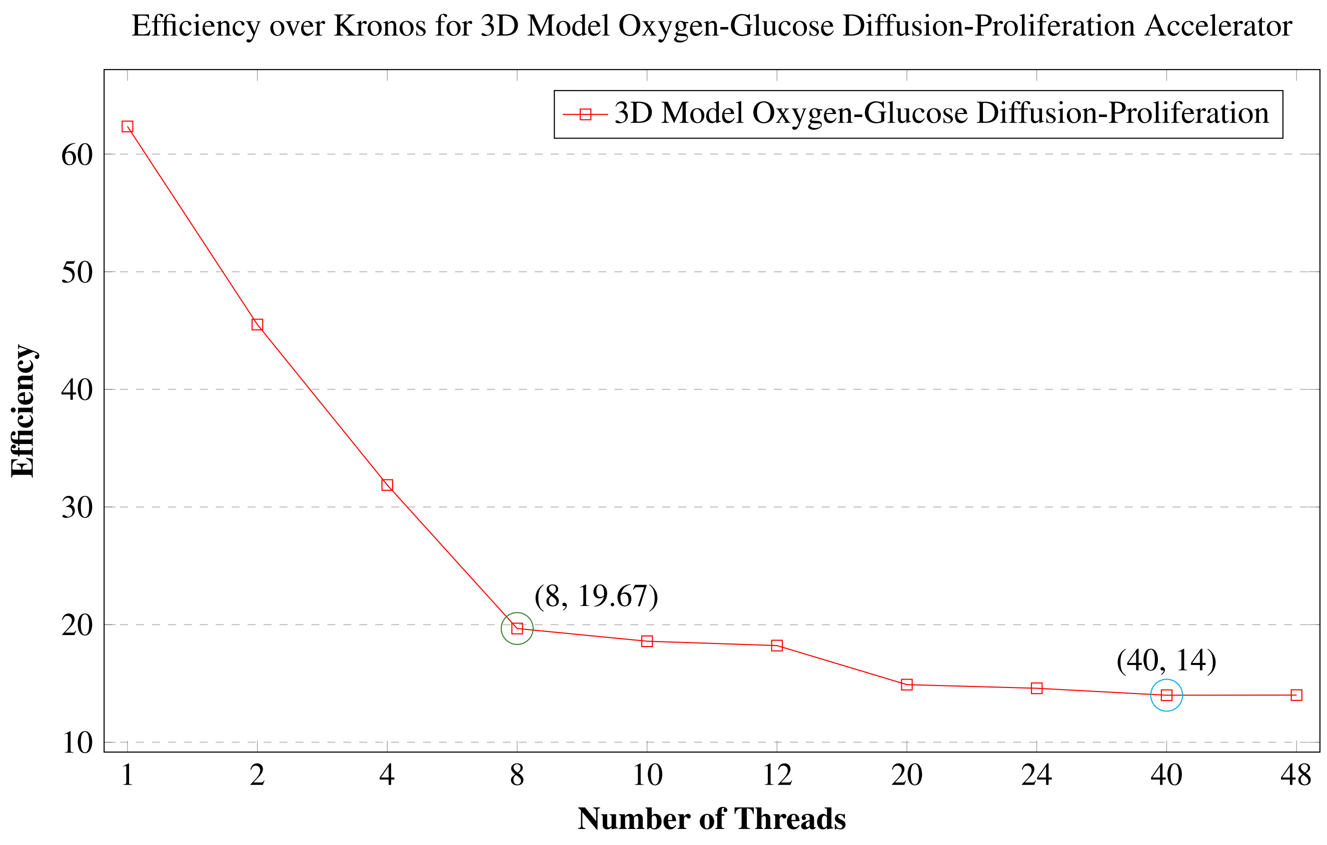
As it can be seen in the graph above, although the accelerator is clocked at 150 MHz, it can achieve the significant **4.54x** runtime speedup over 20 threads (green circle) of Zeus. The accelerator also achieved a **1.85x** runtime speed up over the most efficient usage of number of threads (cyan circle) in Kronos Server.

To calculate the efficiency, each iteration's power and runtime have to be calculated in each platform. The power consumption remains stable, for every iteration in each platform and it can be calculated by the formulas 4.2 and 4.3. The runtime, the energy for 1 iteration and the final efficiency are shown in the next table.

Table 4.22 Model 3D Oxygen-Glucose Diffusion-Proliferation Hardware accelerators Energy per iteration.

	Hardware Accelerator	Software 40 threads
Runtime - 1 Iteration (Seconds)	0.0217	0.0402
Power (Watts)	30	226
Energy (Joules) / Iteration	0.651	9.092
Efficiency	**14x**	-

The final efficiency is 14x over the Kronos Server. This is the least efficiency that can be achieved, of course. The efficiency, against each different number of threads, is calculated by using the formulas 4.2, 4.1 and 4.3 and are shown in the next graph.



Plot 4.7 Efficiency over Kronos. The circled values show efficiency against a high-end PC and a high-end server computer (Kronos).

In the graph 4.7 is shown that the accelerator can achieve even 62x efficiency against one thread and 19.6x against 8 threads of Kronos (green circle). The efficiency against 8 threads is noted again, because a modern high-end PC can provide up to 8 threads. However, the accelerator can even be more efficient a high-end Server system, like Kronos.

The same feature that extracted from 3D Model simplified accelerators (4.2.5), the efficiency of the hardware accelerator is better in another way, as well. If the Hardware Accelerator consumes as much energy as the server for the same amount of time of tumor simulation (30 Days), the iterations have to be increased thus the *dt* interval has to be decreased, as well. The new time interval *dt* is about 0.00007 days. This will help the model extract much more detailed images, as it samples the tumor growth many more times in a single day. Thus, the FPGA not only is much more energy-efficient, but can also give more quality results. The same images in order to be extracted by Kronos would require significantly more energy. The FPGA implementation is the most efficient solution for the Model 3D Oxygen-Glucose, as well.

4.2.3.4 Simulation Results

In Fig. 4.3 it can be noticed that one month after the tumor's assumed detection (second column) it already consists of distinct areas, namely the central hypoglycemic extent (dark gray), along with the hypoxic cell-population around it and the very thin external proliferative zone (white). By the end of the third month (third column), a necrotic core has appeared (darker gray), surrounded by the hypoxic along with the hypoglycemic region and the outward proliferative ring. The necrosis expands within the next three months (fourth column) and after one year (fifth column), the tumor has no significant further expansion while the necrotic region occupies almost the entire tumor body.

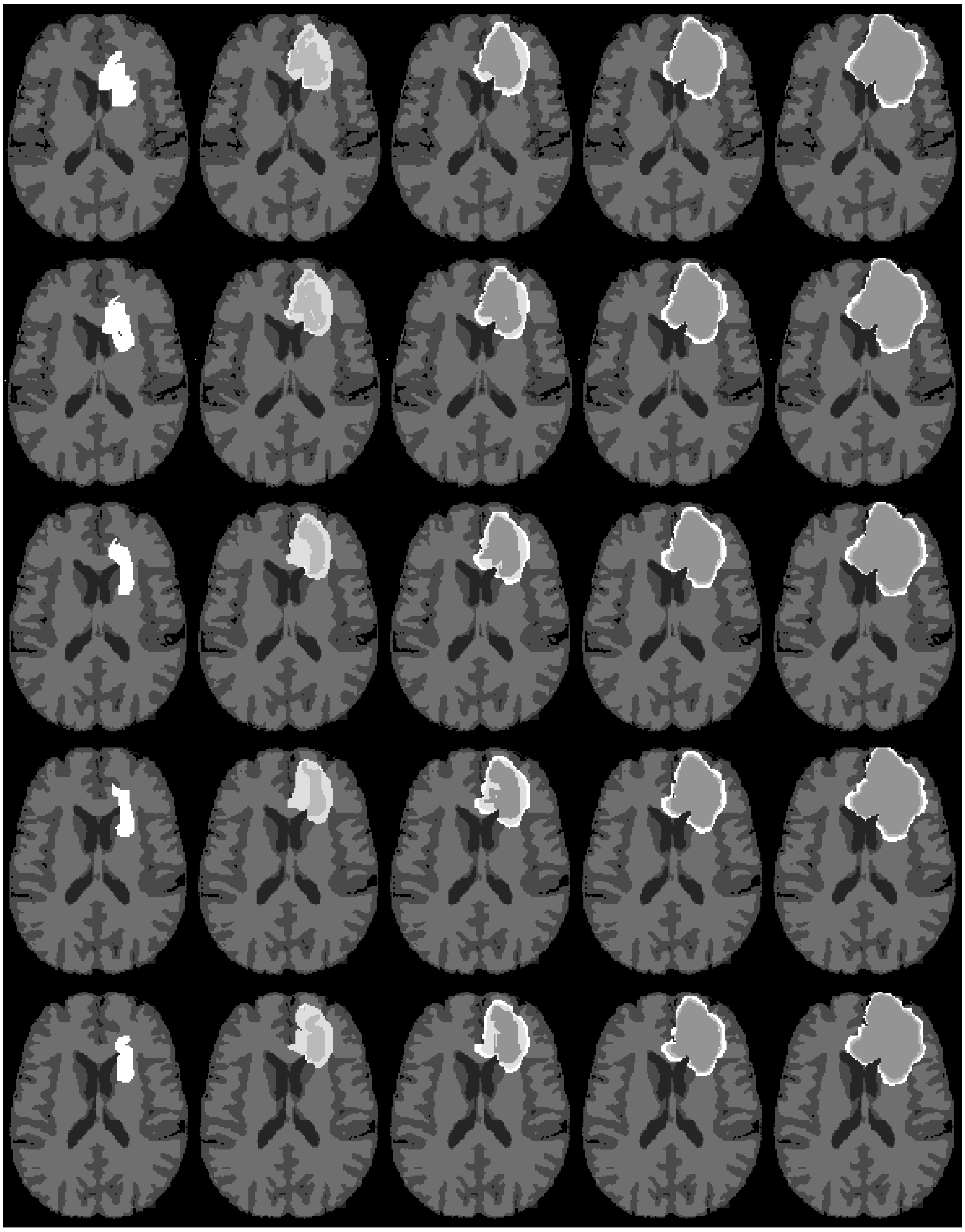


Fig. 4.3 Simulation results of the nutrients 3D model on a medium-diffusion / medium proliferation tumor. Lines: five consecutive frames. Columns: Initial tumor (1st), evolution after: 1 (2nd), 3 (3rd), 6 (4th) and 12 (5th) months.

4.2.4 Results Summary

In this chapter, the results, for each model in different platforms, were showed. Firstly, the software implementations, in two powerful server systems, showed that the use of OpenMP API reduces significantly the runtime. Specifically, in Zeus, the speedup when using the maximum threads available against one thread, in every model (2D Oxygen-Glucose, 3D Simplified, 3D Oxygen-Glucose) are 11.6, 6.1 and 5.3 respectively. In Kronos, the speedup results are 11.3, 7.8 and 7.7 respectively. However, it is also important to be noted by viewing the speedup results, that the increase in the number of threads does not speed up the application linearly. This is a normal behavior for applications that process big datasets, as these tumor prediction models. The reason is that the software becomes I/O-bound (memory throughput), as the number of threads increase.

Secondly, in this chapter is shown how the power and energy consumption scale with respect to the number of threads utilized. The only Server System, that was accessible to measure its power consumption, was Kronos. In the Kronos Server, the whole system consumes about 130 Watts in idle mode. When OpenMP utilizes only one thread, the power consumption is very low as well, about 131-133 Watts. However, the runtime for the whole application is very large. That way, the energy spent is equally high, as well, because the energy is proportional both to power and time (4.1). For that reason, it is proved, that the use of more threads not only speed up the application but also reduced the energy consumption. The above conclusions apply in every tumor progression Model application.

The main target of this Thesis was to accelerate the three Model's runtime, by using Reconfigurable Logic. In this chapter, it is shown that the hardware accelerators implemented, not only achieved runtime speed up but also they are more efficient than the software implementations in both servers. First of all, although the hardware designs are clocked at 150-200 MHz, they can achieve significant speedup against one thread in both the Zeus and the Kronos Server, in which their clock's frequencies are 15 times greater, at least. In fact, the hardware implementations succeeded in accelerating the models in such a degree, that they can achieve speedup over the corresponding multi-threaded software implementations. The Runtime Speed Up, for every model, is shown below. For the 3D Model Simplified, only the 2-core architecture's results are shown, as it is the most efficient accelerator amongst the others.

Table 4.23 Speedup over both Servers, for every Model and Accelerator, for a different number of threads.

Runtime Speedup						
	2D Model Oxygen-Glucose		3D Model Simplified		3D Model Oxygen-Glucose	
Threads	Zeus	Kronos	Zeus	Kronos	Zeus	Kronos
1	29.91	18.54	5.07	4.19	24.30	14.34
2	16.63	10.90	2.71	2.62	15.90	9.72
4	9.01	6.70	1.63	1.41	9.12	5.38
8	4.76	3.85	1.58	0.81	6.23	3.21
10	3.84	3.21	1.38	0.73	5.41	2.86
12	3.33	2.83	1.34	0.67	5.20	2.59
20	2.84	1.98	0.83	0.56	4.54	2.03
24	2.63	1.92	0.98	0.58	4.58	1.95
40	2.76	1.64	1.05	0.53	4.61	1.85
48	2.88	1.90	1.11	0.63	4.70	1.88

The table 4.23 shows that even the 3D Model's design succeeded in accelerating the application up to 1.58x over 8 threads of Zeus and 1.41x over 4 threads of Kronos. In the other two Models, the hardware accelerators are even faster than all the available threads in both Servers.

The FPGAs are platforms with very low power consumption. The Models implemented in the UltraScale+ FPGA are even more efficient than the servers. The efficiency resulting by the formulas 4.2, 4.1 and 4.3, proved that the FPGAs' power and energy consumption for computationally intensive applications, make them the most efficient platforms. The next table shows the efficiency, for a different number of threads, against the Kronos Server.

Table 4.24 Efficiency over Kronos Server, for every Model and Accelerator, for a different number of threads.

Efficiency			
Threads	2D Model Oxygen-Glucose	3D Model Simplified	3D Model Oxygen-Glucose
1	161.32	19.53	62.35
2	122.28	16.70	45.51
4	78.11	10.22	31.87
8	48.79	6.30	19.67
10	42.78	6.33	18.59
12	38.51	5.86	18.22
20	32.27	5.53	14.90
24	31.19	5.83	14.59
40	29.70	5.34	14.00
48	30.60	5.75	14.01

The table 4.24 shows that although the 3D Model Simplified accelerator did not achieve runtime speedup over 40 threads, succeeded in becoming 5.34 times more efficient than Kronos. This proves that an FPGA implementation, for a very computationally intensive algorithm with many memory accesses, is more efficient than any software implementation running on many threads. Moreover, the other two Models' accelerators, for 2D Model Oxygen-Glucose and 3D Model Oxygen-Glucose, achieved the significant 29.7x and 14x more efficiency against Kronos (40 threads), respectively. As it is already mentioned, these two Servers are high-end Servers, that are not accessible to any user. A modern high-end computer usually has up to 8 threads. The achieved efficiencies, over 8 threads for each Model, are even greater, up to 48.8x, 6.3 and 19.7 respectively.

Finally, the great efficiency speedup, achieved by the accelerators, allow extracting another significant feature. The hardware implementations can extract more detailed images by processing more time-steps for the same time of simulation. The accuracy of a model is determined by the time-step interval. By decreasing this interval for a specific simulation e.g. 30 days simulation, the accuracy increases, as the algorithm samples the tumor evolution more frequently.

4.2.5 3D Model Experimentation using different time-steps

Since the approximation error of these glioma growth models is decreased proportionally to the applied time step (dt), the following experiment has been conducted. The simple 3D model was applied twice on the initial tumor derived from same MRI sequence, for prediction time equal to 1 month and defining exactly the same parameter values, but with a different time-step between the two simulations, namely, $dt = 3$ days and $dt = 3 \cdot 10^{-7}$ days. The results, which are depicted in Figure 4.4, indicate that the boundaries of the estimated tumor are slightly different in these two cases and specifically the evolution is more “detailed” and less rough concerning the case of the lower time step ($dt = 3 \cdot 10^{-7}$ days). Thus, in order to reduce the approximation error, the time step should be decreased, but taking into account the total execution time that is increased for lower time steps, since they lead to higher numbers of iterations. However, such simulations are far more efficient when applied on hardware in comparison with the respective software, as the power consumption in FPGA is much lower than the Kronos. Hence, glioma-growth modeling applications on hardware can produce more detailed estimations of tumor evolution, minimizing the execution time step and consequently the corresponding approximation error, yet keeping the total execution time relatively low. In conclusion, the implementation of such biomedical applications on hardware not only reduces the total execution time, but also the total approximation error, leading to more realistic estimations.

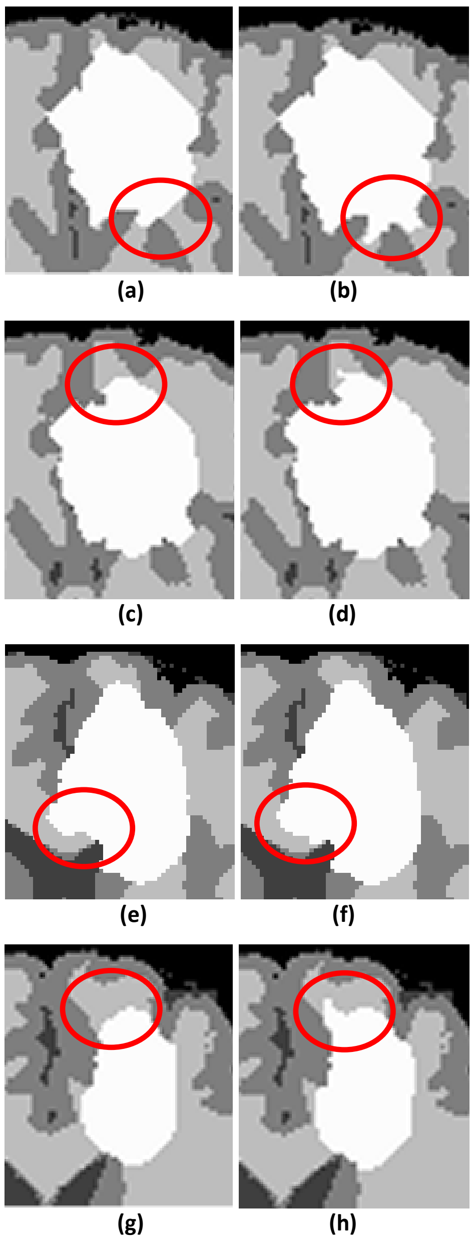


Fig. 4.4 Simple-model simulation of tumor growth for 1 month. Lines: four indicative slices, Subfigures: (a), (c), (e), (f): execution for time step = 3 days and (b), (d), (f), (h): execution for $time - step = 3 \cdot 10^{-7}$ days. The main differences are denoted inside the red circle.

The extracted images are out of the boundaries of this design's efficiency. However, it is more efficient to achieve this quality of images in hardware rather than a Server System.

Chapter 5

Conclusion and Future Work

This chapter will sum up and evaluate this Thesis' work. It will also present some future work and how this Thesis expands further the High-Performance Computing engineering for accelerating more biomedical applications. Finally, the difficulties, that have been overcome and the lessons that have been learned throughout this Thesis, will be presented at the end.

5.1 Conclusion

The purpose of this Thesis was to accelerate three models of a tumor growth prediction system, using Reconfigurable Logic. The target was achieved by creating hardware accelerators for UltraScale+ FPGA. The whole procedure required a deeper understanding of each model's tumor evolution kernel code, in order the most efficient and faster accelerator to be implemented. The main approach used, is the streaming of the data in and out of the accelerator. This procedure required a very sophisticated and complex approach, as it is very important the streaming procedure to be utilized to the fullest. The final accelerators implemented, utilized almost all the resources of the FPGA.

Another important work of this Thesis is the measurement of the bandwidth of Trenz's platform. Until now, the maximum bandwidth of this platform, using Streaming and all High-Performance ports enabled, was never been measured before. Firstly, it is shown (4.1), that the maximum bandwidth resides in both Read and Write channels. Secondly, the High-Performance Cache-Coherent ports do not give any significant extra bandwidth over the 4 HP ports. The results about the bandwidth of the platform, show that the accelerators have also a limit and can not speed up the Model's applications after that. If the FPGA could fit many hardware cores in order to parallelize a Model, this may not achieve significant speedup. The reason is that the hardware implementation becomes more and more I/O bound, as the application parallelized further.

The FPGA implementation succeeded in speeding up even the most computationally intensive Models. Each accelerator was compared to two different high-end Servers. The results showed that FPGA not only achieved impressive runtime speedup over the single-threaded software implementations but also achieved significant speedup against the multi-threaded implementations running on the two Servers. However, the FPGA platforms show another impressive feature, which is the low power consumption. The Trenz platform consumes at most 30 Watts per hour in full power. In this measurement is not only consisted the FPGA, but also the CPU (ARM Cortex A-53), the DDR4 memory, the power supply unit and all the fans for cooling the entire system.

On the other hand, the Servers that were used to compare the Hardware Accelerator's performance are consuming much more power. The Kronos Server, which was the only server that its power could be measured, consumes 130 Watts in idle mode, which means it consumes 3.6x more than the Trenz's maximum power consumption. Kronos' system reached the 226 Watts of power consumption, using 40 threads for the most computationally intensive tumor growth prediction Model. For this measurement, the cooling system of the entire system was not taken into account. The Server is located in a server room, alongside other server racks, in the Technical University of Crete. The measurement of the power of the whole cooling system of the entire Server's system was impossible, but it is known that it is very costly. Could it be measured, the efficiency of the SoC systems would be even clearer. The final results showed that the FPGA implementation can spend up to 29.7 less energy than 40 threads of Kronos and at the same time, run the simulation 1.64x faster. Also, if the FPGA consumed the same energy as the Kronos Server for the same simulation, the efficiency achieved by the accelerators, allow them to extract more detailed images, as it is already described in section 4.2.5. The same images, in order to be extracted by Kronos, would be very inefficient, due to its high power consumption.

Another important information extracted by this Thesis, is the power of HLS tools. A software so complicated as the three tumor growth prediction models, it would be nearly impossible to be coded and optimized in an HDL language in a reasonable amount of time. Therefore, the man-hours to code and debug the entire application would be many more. The modern tools are not yet optimized to achieve best results and sometimes they show significant bugs. However, the assistance of a complete toolchain, as Xilinx Vivado HLS, Xilinx Vivado, and Xilinx SDK made it possible to achieve this Thesis's target results.

5.2 Future Work

The High Performance Computing science is offering the computer power to simplify the most computationally intensive problems. Many of these problems lay in bioinformatics science, one of these is the tumor prediction models, which are the applications accelerated in Reconfigurable Logic in this Thesis. The acceleration of these models opens the opportunities to accelerate further tumor prediction models, that are under development, taking into consideration a few extra parameters, that affect the tumor evolution, appealing to the real world even more. The models that were used in this Thesis are predicting only the invasion and the progression of a tumor. Another set of models are being developed right now, as well, intend to predict the behavior of a tumor under specific medication and could offer to the doctors a new powerful tool for fighting tumors. To sum up, using a non-invasive procedure, such as a simple MRI scan, the doctor could predict the behavior of a tumor in its early stages and decide the proper medication for every patient, by using a low-cost method, an FPGA accelerator. Finally, an extra feature that could be used is the testing of new developing models, using this type of hardware accelerators and helping the biomedical science community to research even further, one of the most fatal diseases, tumors.

The modern FPGA platforms offer even more resources, that could accelerate the runtime of such computational intensive applications and offer much more efficiency. An FPGA with 30% extra Block RAM could achieve pipeline II=1. However, this is not enough, as it is very clear by the maximum bandwidth, which was measured and is presented in the previous chapter 4.5. A new high-end FPGA should offer even more bandwidth, in order to achieve even better performance.

The FPGA platforms are not the only option to accelerate such models. A GPU acceleration should be a very intriguing approach, as well. The new high-end GPUs offer enormous computational power. This approach should not only accelerate the models over a Server System, but it is expected to speed up even more than the FPGA's approach. However, the question is the efficiency of a high-end GPU. It is known, that such systems as a GPU, consume a lot of power. Thus, a GPU approach may be better for the acceleration, but still the FPGA may have better efficiency.

5.3 Lessons Learned

The work of this Thesis exceeded the expectations and achieved impressive results. However, many difficulties had to be overcome, in order to achieve these results. As it has already be described in previous chapters, the toolchain and the platform decided and used for completing this Master Thesis, have some significant bugs. The first one that has been encountered is the weakness of the Vivado HLS 2016.2 to distinguish whether the letters are capital or not, as it translates the case sensitive C programming language to an HDL non-case sensitive one. This bug delayed the extraction of the first model's functional architecture. Another bug overcame, is located in Vivado Design Suite 2016.2, in which the HP ports had to set in Data Width 128 bits in order to transfer 64 bits Data. This bug is solved in newer versions. Furthermore, the Trenz platform, as it is mentioned, is an evaluation model and one of the first platforms that carry an UltraScale+ FPGA. The platform has some serious hardware bugs and the most important is the malfunction of the JTAG connector which made the whole programming, evaluation and verification of its architecture quite difficult, as the Xilinx Software Command-Line Tool (XSCT) commands are not available. This malfunction, also, made the debugging even more difficult, as it was impossible to use the Integrated Logic Analyzer (ILA) cores and some workarounds had to be done. Every bug that has been encountered, in order to be solved, required a lot of complicated thinking to find the root of the problem. Therefore, a lot of time spent in debugging the designs' cores that were already very complex and sophisticated. However, this procedure helped me to gain great experience with the HLS tools' design logic and the Xilinx Vivado Toolchain. Finally, another minor problem was the Trenz platform's cooling system, which is not enough to maintain the temperature in normal functional condition and for that reason another cooling fan was used. In conclusion, the choice of this particular subject for Master Thesis not only gave me the opportunity to expand my knowledge on the High Performance Computer field, but also expanded the understanding of the FPGA's architecture in greater depth and the awareness on biomedical applications.

References

[1] Szeto MD, Chakraborty G, Hadley J, "Quantitative metrics of net proliferation and invasion link biological aggressiveness assessed by mri with hypoxia assessed by fmiso-pet in newly diagnosed glioblastomas.," *The Journal of Cancer Research(1916-1930) | The American Journal of Cancer(1931-1940)*, vol. 69, no. 10, pp. 4502–4509, 2009.

[2] Harpold H.L.P., Alvord E.C., Swanson K.R., "The evolution of mathematical modeling of glioma proliferation and invasion," *Journal of Neuropathology & Experimental Neurology*, vol. 66, no. 1, pp. 1–9, 2007.

[3] Papadogiorgaki Maria , Koliou Panagiotis, Kotsiakis Xenofon, Zervakis Michalis E, "Mathematical modeling of spatio-temporal glioma evolution," *Theoretical Biology and Medical Modeling*, vol. 10, no. 1, pp. 47–83, 2013.

[4] Doulamis N., Papadogiorgaki M., Zervakis M., "Texture driven mathematical modeling for brain tumor evolution. pervasive technologies related to assistive environments," in *PETRA 2013, 29-31 May, 2013, Rhodes Island, Greece*.

[5] R. M. Jiang and D. Crookes, "Fpga implementation of 3d discrete wavelet transform for real-time medical imaging," in *2007 18th European Conference on Circuit Theory and Design*, pp. 519–522, Aug 2007.

[6] K. Cleary and T. M. Peters, "Image-guided interventions: Technology review and clinical applications," *Annual Review of Biomedical Engineering*, vol. 12, no. 1, pp. 119–142, 2010. PMID: 20415592.

[7] O. Dandekar, C. Castro-Pareja, and R. Shekhar, "Fpga-based real-time 3d image preprocessing for image-guided medical interventions," *Journal of Real-Time Image Processing*, vol. 1, pp. 285–301, Jul 2007.

[8] M. Birk, A. Guth, M. Zapf, M. Balzer, N. Ruiter, M. Hübner, and J. Becker, "Acceleration of image reconstruction in 3d ultrasound computer tomography: An evaluation of cpu, gpu and fpga computing," in *Proceedings of the 2011 Conference on Design Architectures for Signal Image Processing (DASIP)*, pp. 1–8, Nov 2011.

[9] D. J. Pagliari, M. R. Casu, and L. P. Carloni, "Accelerators for breast cancer detection," *ACM Trans. Embed. Comput. Syst.*, vol. 16, pp. 80:1–80:25, Mar. 2017.

[10] S. Hasan, S. Boussakta, and A. Yakovlev, "Fpga-based architecture for a generalized parallel 2-d mri filtering algorithm," *American Journal of Engineering and Applied Sciences*, vol. 4, no. 4, pp. 566–575, 2011.

[11] G. P. S. Ramya and S. Bhuvaneshwari, "Fpga based implementation of edge and corner detection in mri brain tumor image," 2015.

[12] Drasdo D., "Coarse graining in simulated cell populations," *Advances in Complex Systems*, vol. 8, no. 2–3, pp. 319–363, 2005.

[13] Hatzikirou H., Deutsch A., "Cellular automata as microscopic models of cell migration in heterogeneous environments," *Current Topics in Developmental Biology*, vol. 81, pp. 401–434, 2008.

[14] Wolfram S, *Cellular Automata and Complexity: Collected Papers*. USA: Addison-Wesley, 1994.

[15] Kansal AR, Torquato S, Harsh GI, Chiocca EA, Deisboeck TS, "Simulated brain tumor growth dynamics using a three-dimensional cellular automaton," *Journal of Theoretical Biology*, vol. 203, no. 4, pp. 367–382, 2000.

[16] Anderson, A.R.A., Chaplain, M.A.J., Newman, E.L., Steele, R.J.C., Thompson, A.M., "Mathematical modeling of tumor invasion and metastasis," *Journal of Theoretical Medicine*, vol. 2, no. 2, pp. 129–154, 1999.

[17] Roose, T., Chapman, S.J., Maini, P.K., "Mathematical models of avascular tumor growth," *Society of Industrial and Applied Mathematics*, vol. 49, no. 2, pp. 179–208, 2007.

[18] Swanson, K.R., Bridgea, C., Murray, J.D., Alvord, E.C., "Virtual and real brain tumors: using mathematical modeling to quantify glioma growth and invasion," *Journal of the Neurological Sciences*, vol. 216, no. 1, pp. 1–10, 2003.

[19] Kiran, K.L., Jayachandran, D., Lakshminarayanan, S., "Mathematical modeling of avascular tumor growth based on diffusion of nutrients and its validation," *The Canadian Journal of Chemical Engineering*, vol. 87, pp. 732–740, 2009.

[20] Roniotis, A., Manikis, G., Sakkalis, V., Zervakis, M., Karatzanis, I., Marias, K., "High-grade glioma diffusive modeling using statistical tissue information and diffusion tensors extracted from atlases," *IEEE Transactions on Information Technology in Biomedicine*, vol. 16, no. 2, pp. 255–263, 2012.

[21] Roniotis, A., Marias, K., Sakkalis, V., Tsibidis, G.D., Zervakis M., "A complete mathematical study of a 3d model of heterogeneous and anisotropic glioma evolution," in *31st Annual International Conference of the IEEE EMBS, 2-6 September 2009, Minneapolis, Minnesota, USA.*, pp. 2807–2810.

[22] Greenspan HP, "Models for the growth of a solid tumor by diffusion," *Studies in Applied Mathematics*, vol. 51, no. 4, pp. 317–340, 1972.

[23] Clatz O, Sermesant M, Bondiau PY, Delingette H, Warfield SK, Malandain G, Ayache N, "Realistic simulation of the 3d growth of brain tumors in mr images coupling diffusion with biomechanical deformation," *IEEE Transactions on Medical Imaging*, vol. 24, no. 10, pp. 1344–1346, 2005.

[24] Anderson, A.R.A., “A hybrid mathematical model of solid tumour invasion: the importance of cell adhesion,” *Mathematical Medicine and Biology: A Journal of the IMA*, vol. 22, no. 2, pp. 163–186, 2005.

[25] Tanaka ML, Debinski W, Puri IK, “Hybrid mathematical model of glioma progression,” *Cell Proliferation in basic and clinical sciences*, vol. 42, no. 5, pp. 637–646, 2009.

[26] Jeon, J., Quaranta, V., Cummings, P.T., “An off-lattice hybrid discrete-continuum model of tumor growth and invasion,” *Biophysical Journals*, vol. 98, no. 1, pp. 37–47, 2010.

[27] Rejniak, K.A., Anderson, A.R., “Hybrid models of tumor growth,” *Wire’s Systems Biology and Medicine*, vol. 3, no. 1, pp. 115–125, 2010.

[28] Jiang Y., Pjesivac-Grbovic J., Cantrell C., Freyer J.P., “A multiscale model for avascular tumor growth,” *Biophysical Journals*, vol. 89, no. 6, pp. 3884–3894, 2005.

[29] Papadogiorgaki, M., Koliou, P., Zervakis, M.E., “Glioma growth modeling based on the effect of vital nutrients and metabolic products,” *Springer M.E. Medical & Biological Engineering & Computing*, pp. 1—-15, 2018.

[30] K. R. Swanson, R. C. Rockne, J. Claridge, M. A. Chaplain, E. C. Alvord, and A. R. Anderson, “Quantifying the role of angiogenesis in malignant progression of gliomas: In silico modeling integrates imaging and histology,” *Cancer Research*, vol. 71, no. 24, pp. 7366–7375, 2011.

[31] A. Martínez-González, G. F. Calvo, L. A. PérezRomasanta, and V. M. Pérez-García, “Hypoxic cell waves around necrotic cores in glioblastoma: A biomathematical model and its therapeutic implications,” *Bulletin of Mathematical Biology*, vol. 74, pp. 2875–2896, Dec 2012.

[32] E. Tzamali, R. Favicchio, A. Roniotis, G. Tzedakis, G. Grekas, J. Ripoll, K. Marias, G. Zacharakis, and V. Sakkalis, “Employing in-vivo molecular imaging in simulating and validating tumor growth,” in *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 5533–5536, July 2013.

[33] R. A. Gatenby, E. T. Gawlinski, A. F. Gmitro, B. Kaylor, and R. J. Gillies, “Acid-mediated tumor invasion: a multidisciplinary study,” *Cancer Research*, vol. 66, no. 10, pp. 5216–5223, 2006.

[34] M. Papadogiorgaki and M. Zervakis, “Visualization and modeling of cancer progression,” in *2013 IEEE International Conference on Imaging Systems and Techniques (IST)*, pp. 106–111, Oct 2013.

[35] “The cancer imaging archive - RIDER NEURO MRI.” https://wiki.cancerimagingarchive.net/display/Public/RIDER+NEURO+MRI.

[36] “SRI24 ATLAS: Normal adult brain anatomy.” https://www.nitrc.org/projects/sri24.

[37] Xilinx, Inc., *AXI DataMover v5.1 LogiCORE IP Product Guide (PG022).* Xilinx, Inc.

[38] Xilinx, Inc., *Vivado Design Suite User Guide: Synthesis (UG901).* Xilinx, Inc.

[39] Xilinx, Inc., *Vivado Design Suite User Guide: Implementation (UG904).* Xilinx, Inc.

[40] "Dell Remote Access Controller and iDRAC." https://en.wikipedia.org/wiki/Dell_DRAC.

[41] "OpenMP Loop Scheduling." https://software.intel.com/en-us/articles/openmp-loop-scheduling.

[42] Paolo Gorlani, "High Level Synthesys of OpenCL kernels for Molecular Dynamics," Master's thesis, SISSA, Master in High Performance Computing (MHPC), ICTP, 2017.