Accelerating a Medical 3D Brain MRI Analysis Algorithm using a High-Performance Reconfigurable Computer

by

Jahyun J. Koo

Department of Electrical and Computer Engineering

McGill University, Montreal

A thesis submitted to McGill University in partial fulfillment

of the requirements of the degree of Master of Engineering.

September 30, 2007

Acknowledgments

First of all, I would like to thank my thesis supervisors, Dr. Warren J. Gross and Dr. Alan C. Evans. Your thoughtful concerns and enthusiastic support made this thesis possible. Once again, thank you for giving me this amazing opportunity!!

I would also like to thank to Simon Hong, Kelvin Mok, Jaejin Ryu, Eric Lee, and Phil Chopp who proof-read my thesis. Especially to Phil and Simon, thank you for your great revisions and insightful comments. I am further very grateful to my fellow researchers who I met during my studies at McGill University. I will never forget those joyful memories I shared with Dani Tannir, Tarek Alhajj, Baker Haddadin, Kelvin Lee, Sadok Aouini, Ahmed Abdel-Aty, Mohammad Taherzadsh, Terrance Kao, and Yousef El-Kurdi at the MACS Lab. Although the MACS lab soccer team never won the championship, I am extremely proud of every single player who participated on AC MACS and MACS UNITED. Thanks to David Fernández and Ashraf Haddad for working extremely hard to publish our first paper. Without your help, the ASAP paper was not possible. I would also like to thank Dr. Uicheul Yoon, Dr. Junki Lee, Hosung Kim and Claude Lepage for their tremendous help on image processing algorithms and statistical analysis.

For last, I dedicate this thesis to my lovely parents who relinquished their life in Korea and moved to Canada for me and my brother. This thesis would not exist without your insightful advice and encouragement. Thank you for your patience and understanding!!!

Contributions of Authors

This thesis is based on two manuscripts([1] and [2]) which will appear in the proceedings of International Conference on Application-specific System, Architectures and Processors (ASAP'07) and International Conference on Filed Programmable Logic and Applications (FPL'07). I am the first author of both manuscripts and have involved implementing and analyzing the proposed study in this thesis. In [1], I was the primary project manager and was in charge of optimizing the primary implemented algorithms and developing test plans for verification. In [2], as a primary research of the project, I implemented and evaluated the proposed algorithm using many different design and test strategies.

The following list summarizes the contributions of co-authors:

1. **Dr. Warren J. Gross** : Supervisor of the project.
2. **Dr. Alan C. Evans** : Co-Supervisor of the project.
3. **David Fernández** : developed the initial implementation of the Dense Matrix Vector Multiplication (DMVM) algorithm in [1].
4. **Ashraf Haddad** : developed the initial implementation of the Spherical Boundary Conditions in Molecular Dynamics (SB) algorithm in [1].

Abstract

Many automatic algorithms have been proposed for analyzing Magnetic Resonance Imaging (MRI) data sets. These algorithms allow clinical researchers to analyze their quantitative data with consistently accurate results. With the increasingly large data sets being used in brain mapping, there has been a significant rise in the need for methods to accelerate these algorithms, as their computation can consume many hours. This thesis presents the results from a study on implementing such quantitative analysis algorithms on High-Performance Reconfigurable Computers (HPRCs). The Partial Volume Estimation (PVE), a brain tissue classification algorithm for MRI, was implemented on two SGI RASC RC100 systems using the Mitrion-C High-Level Language (HLL). The CPU-based PVE algorithm was profiled to identify the computationally intensive functions and two floating-point functions, estimating the probability densities (PDs) of tissues and the prior information, were implemented on FPGA-accelerators. Several simulated and real human brain MR images were used to evaluate the accuracy and performance improvement of the FPGA-based PVE algorithm. The *Sensitivity* and *Kappa* coefficients were calculated to verify the accuracy of the images resulting from the FPGA-based implementation. The FPGA-based PDs estimation and prior information estimation function achieved an average speedup of 2.5× and 9.4×, respectively. The overall performance improvement of the FPGA-accelerated PVE algorithm over the conventional CPU-based algorithm was 5.1× with four FPGAs.

Résumé

Plusieurs algorithmes ont été proposés pour l'analyse des données d'imagerie par résonance magnétique (IRM). Ceux-ci ont permis aux chercheurs cliniques d'analyser leurs données avec précision jusqu'à tout récemment. Mais avec l'augmentation des données quantitatives à analyser dans le domaine de l'imagerie du cerveau, il y a un besoin maintenant pour des méthodes pour accélérer ces algorithmes surtout que leur temps de calcul peut prendre plusieurs heures. Cette thèse présente le résultat d'une étude sur l'implémentation de ces algorithmes sur des ordinateurs reconfigurables à haute performance. L'estimation de volume partiel (PVE), un algorithme de classification du tissu de cerveau a été implémenté sur deux systémes SGI RASC RC100 qui utilisent le langage de haut niveau Mitrion-C. L'algorithme PVE sur processeur a été profilé pour identifier les fonctions intensives en temps de calcul et deux fonctions à virgule flottante estimant les densités de probabilité de tissus et l'information antérieure ont été implémenté sur des accélérateurs FPGA. Plusieurs images de cerveau humain simulées et réelles ont été utilisées pour vérifier la précision et l'amélioration en performance de l'algorithme PVE sur FPGA. Les coefficients de sensitivité et kappa ont été mesurés dans le but de vérifier la précision des images de l'implémentation sur FPGA. L'estimation des densités de probabilité sur FPGA et la fonction d'estimation d'information antérieure ont eu pour résultat des gains de performance de 2.5 et 9.4, respectivement. La performance globale de l'amélioration de l'algorithme PVE sur FPGA comparativement à l'algorithme établie sur processeur a été de 5.1 sur quatre processeurs.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Magnetic Resonance Imaging (MRI) has become a paramount medical imaging technology in brain anatomical studies due to its spatial resolution and superior *in-vivo* tissue contrast. As a result, there has been a significant increase in the need for quantitative analysis of 3D MRI data [3]. Such analysis provides an enormous amount of information about the anatomy of the human brain to researchers. In most quantitative MRI studies, several image processing algorithms are usually performed on the 3D MRI data sets to identify and segment particular brain structures. Traditionally, MRI data sets for the human brain were manually processed by researchers who are very familiar with anatomical structures.

Human brain anatomical studies usually acquire a large number of brain images to conduct comparisons and determine the differences among the brains. For example, the study in [4] compared 176 healthy normal brains, aged 7 to 87 years, while [5] collected approximately 500 children brains, aged 7 days to 18 years, to study human brain development. As the number of subjects acquired in these typical studies has increased for more assurable and accurate analysis, some limitations of the manual algorithms, such as long computation time and issues regarding reproducibility, have surfaced. Over the past decade, automatic analysis methods, which reduce computation time and improve data consistency have been proposed and developed [6] [7] [8] [9] [10] [11]. Even though these algorithms are fully automatic, some can take from a few minute to several hours to process a single subject due to their complexities of the algorithms. As a result, acceleration of these algorithms has become a very important task, as the researchers require faster and more accurate processing of a very large number of 3D MRI data sets.

MRI data analysis algorithms can be accelerated using an emerging architecture, High-Performance Reconfigurable Computers (HPRCs) which combine high performance CPUs with reprogrammable accelerators such as Field Programmable Gate-Arrays (FPGAs). The recent studies show that HPRCs are capable of accelerating floating-point scientific applications up to 100 times over conventional CPUs [1] [12] [13] [14]. HPRCs reduce the computation time of floating-point scientific applications using their capability to take advantage of both the coarse-grained parallelism offered by traditional CPU-based multiprocessors and fine-grained parallelism offered by reprogrammable devices. Implementing the MRI data analysis algorithm on HPRC would be a challenging task for researchers with only a software background, as basic hardware knowledge and low-level hardware description languages (HDLs) such as VHDL or Verilog are required to program FPGAs. Several commercial high-level languages (HLLs) for programming FPGAs, such as Handel-C [15], Impulse C [16] and Mitrion-C [17], have been proposed to overcome some of these hardware-dependent limitations associated with FPGA-based implementations .

In this study, the 3D MRI tissue classification algorithm, Partial Volume Estimation (PVE) is implemented on an SGI Altix 350 with two RASC RC100 systems using the Mitrion-C HLL. Chapter 2 provides an overview of 3D brain MRI analysis algorithms as well as the HPRC system and HLL used in this study. Moreover, a detailed overview of the PVE algorithm is also described in Chapter 2. The design strategies used to implement the PVE algorithm on the FPGA-accelerators are proposed in Chapter 3. The CPU and FPGA-based PVE algorithms are performed on both simulated and real images and the results are compared each other. Chapter 4 provides the performance and accuracy comparisons between the two implementations. Chapter 5 discusses the differences between the images generated from two implementations and future works. Conclusions are also provided in Chapter 5.

Chapter 2

Background

2.1 Magnetic Resonance Imaging

As the interests in brain mapping have increased, many medical imaging methods have been developed to produce images of the human brain. These images commonly capture the fine details of the anatomical brain structures, such as three major brain tissue types: Gray Matter, White Matter and cerebrospinal fluid as shown in Figure 2.1. Grey matter (GM) is one of the major components of the Central Nervous System (CNS) and is the region where the functional stimuli are processed. White matter (WM) is other major component of the CNS and is connective fibers responsible for passing messages between functional areas, while cerebrospinal fluid (CSF), surrounding the surface of GM, protects the brain from mechanical pressure.

The common modern imaging technologies used for clinical and research purposes are computed tomography (CT), positron emission tomography (PET) and magnetic resonance imaging (MRI). Among these technologies, MRI has grown rapidly since the first scanning of a human brain in the late 1970's. MRI scanner generates multiple 2D cross-sectional images with a greater spatial resolution and immensely superior soft tissue contrast compare to other medical imaging modalities [18]. Each volume element (voxel) of the generated MR image represents the signal intensity of brain tissue and any image artifact introduced during the image

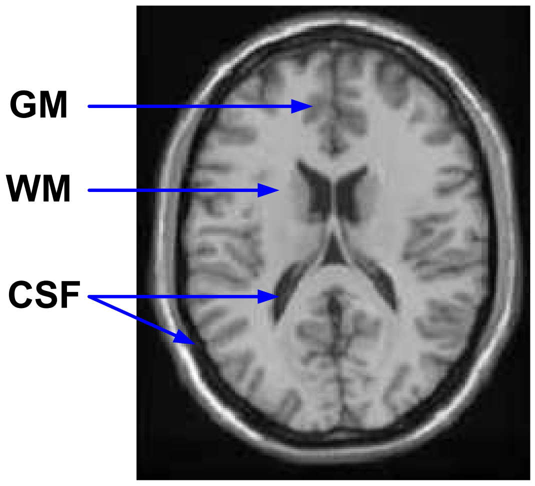


Figure 2.1: Three major tissue classes of the brain.

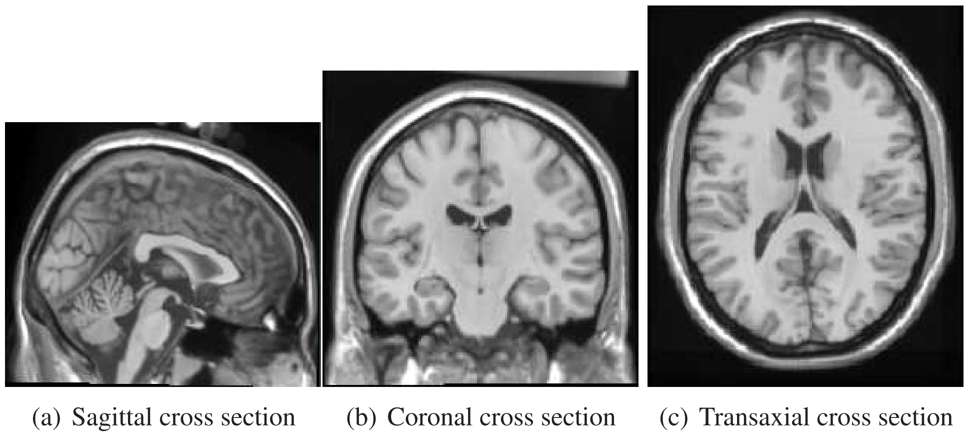

(a) Sagittal cross section
(b) Coronal cross section
(c) Transaxial cross section

Figure 2.2: MRI cross-sectional brain images.

acquisition. Figure 2.2 illustrates the cross-sectional images generated by an MRI scanner. The slice orientation of each image refers to one of orthogonal directions.

Due to the superior tissue contrast among *in vivo* soft brain tissues, MR imaging has become a powerful noninvasive technology in many medical and physiological studies. The tissue contract in MRI originates from the hydrogen proton density in each tissue and the tissue-specific transverse and longitudinal relaxation rates. The relaxation of the magnetization of proton can be described phenomenologically by the Bloch equation:

$$\frac{d\mathbf{M}}{dt} = \mathbf{M} \times \gamma\mathbf{B} - \frac{M_x\mathbf{i} + M_y\mathbf{j}}{T_2} - \frac{(M_z - M_o)\mathbf{k}}{T_1}, \tag{2.1}$$

where **i**, **j** and **k** are unit vectors in the x,y, and z directions respectively. In Equation 2.1, **M** and γ represent the magnetization vector of proton and the gyromagnetic ratio respectively, while **B** includes the various magnetic fields applied. M_o is the magnitude of the equilibrium magnetization when only B_o is applied and M_x, M_y and M_z represent the three orthogonal components of the **M**. T_1 and T_2 are longitudinal and transverse relaxation time constants.

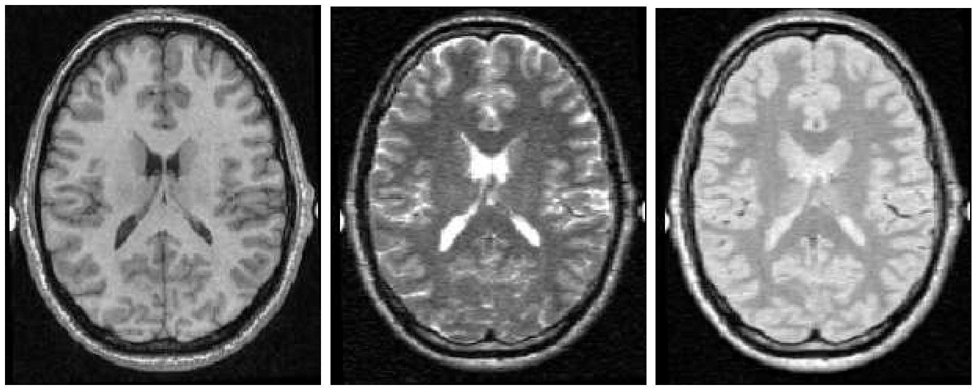
A magnetization moment **M** is produced when B_o is presented and will be "excited" when the radio frequency (RF) pulse is applied in the transverse direction. Following an excitation, the transverse component of the magnetization starts decaying away as:

$$M_{xy} = M_0 e^{\frac{-t}{T_2}}. \tag{2.2}$$

On the other hand, the longitudinal component starts its recovery to the equilibrium state as:

$$M_z = M_o(1 - e^{\frac{-t}{T_1}}). \tag{2.3}$$

Since different tissues have different T1 and T2 time constants, the transverse and longitudinal magnitudes of the magnetized tissue vary in different rate depending on the T1 and T2 values as a function of time. As a result, the soft tissue contrast is maximized if an image is acquired when the maximum magnitude difference is presented between the magnetized tissues. If the tissue contrast in an image is mainly due to the different T1 or T2 values of the tissues, the image generated from an MRI scanner is called T1-weighted or T2-weighted image respectively. Another image often acquired using an MRI scanner is PD-weighted image. The contrast presented in this image is mainly due to the different proton density between tissues. As shown in Figure 2.3, the T1-weighted image has very clear contrasts among the all tissues (typically, CSF, GM and WM), however, the T2-weighted and PD-weighted image show less clear contrast between the tissues.



(a) T1-weighted image (b) T2-weighted image (c) PD-weighted image

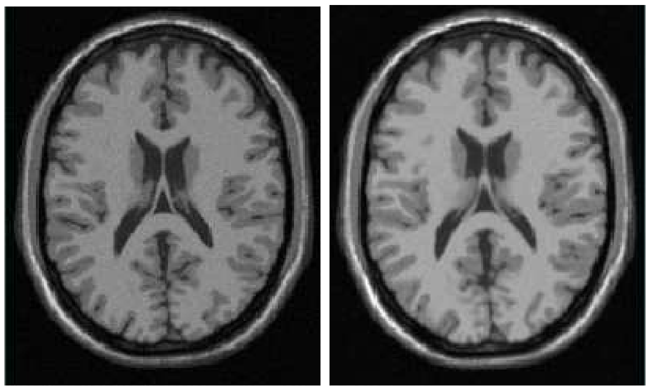
Figure 2.3: T1, T2 and PD contrast.

2.2 Automatic MR Imaging Analysis Algorithm

Due to the high resolution and the explicit tissue contrast, many clinical studies have used MR images for quantitative analysis of human brain structure in many clinical studies. Traditionally, these studies manually processed the MRI data sets to identify different tissue types and to segment neuroanatomical structures in human brain. However, these manual procedures have become too time-consuming as the sizes of data sets in many studies have tremendously increased. Moreover, the manual tissue classification and segmentation are expensive and labor intensive since they require well-trained individuals who are very familiar with neuroanatomical structures. Furthermore, the consistency of results are limited in manual techniques. Therefore, fully automatic MRI analysis algorithms have been developed to overcome these limitations. The automatic algorithms provide more consistent results and allow researchers to process larger set of 3D MRI in a shorter time period. The following subsections present brief descriptions of the major algorithms used for quantitative analysis of 3D brain MRI.

2.2.1 Non-uniformity Correction

One of artifacts often seen across an MR image is the intensity non-uniformity (INU). This artifact is usually caused by poor radio frequency (RF) field uniformity



(a) T1-weighted image with INU (b) INU corrected image

Figure 2.4: INU correction algorithm.

and eddy currents driven by the switching of field gradients [6]. The artifacts may not be visually apparent on the image, however, they cause tissue within homogeneous areas to have different intensities. As a result, an algorithm to correct the INU, such as [6], is usually performed in the early stage of automated image processing so that the artifacts presented on MRI do not degrade the performance of any other algorithms. Figure 2.4 illustrates a non-uniformity corrected image.

2.2.2 Registration

In human brain mapping, all image volume within as well as between populations are compared to each other for a reliable quantitative analysis. As a result, an algorithm has been developed to map all image volumes in a study into the same spatial coordinate system so that data from a similar location in different brains can be compared. This algorithm is distinguished into liner or non-linear registration depending on how the algorithm transforms the image volumes to the stereotaxic space. The linear registration algorithm uses 9-parameters (translation, rotation and scale) to globally transfer the native MRI data set into a pre-defined atlas. The commonly used atlas for linear registration is produced from an averaged MRI brain aligned with the Talairach sterotaxic coordinate system [7]. On the other hand, the non-linear registration generates a high dimensional deformable field and locally transfers the image volumes into the stereotaxic space using the field [19].

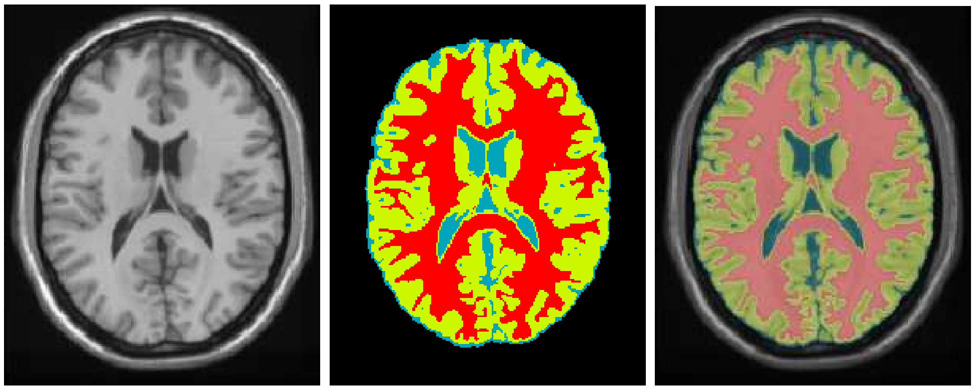
2.2.3 Tissue Classification

After the non-uniformity of MR image is corrected and the image is transferred into the stereotaxic space, researchers use tissue classification algorithm to classify each voxel of the image into one of four tissue classes (CSF, GM, WM and background) as shown in Figure 2.5. The tissue classification algorithm usually exploits the tissue contrast presented in MR images to identify brain tissues. The algorithm first generates a histogram of each tissue type and determines the intensity distribution of each tissue type. The classifier then evaluates the intensities of voxels and labels the voxels into appropriate tissue classes.

The commonly used tissue classification algorithms are usually distinguished into *supervised* or *unsupervised* methods depending on whether a *priori* information is used to classify the voxels or not [20]. In the supervised classification algorithm, a 3D "training" volume, which contains tissue probability information is first generated from several hundred MR images that have been classified by a semi-automatic or automatic method. The MR image is then classified using a classifier and the generated "training" volume. The commonly used classifiers in the algorithms are maximum likelihood [11], k-nearest neighbors (KNN) [9] or artificial neural network (ANN) [21]. On the other hand, the unsupervised classification algorithm clusters voxels in an image only based on statistics such as *K-means* clustering algorithm without any pre-defined training data set [20]. The algorithm classifies voxels into K different tissue classes by minimizing the sum of squares of distance between data and the centroid of clustered data.

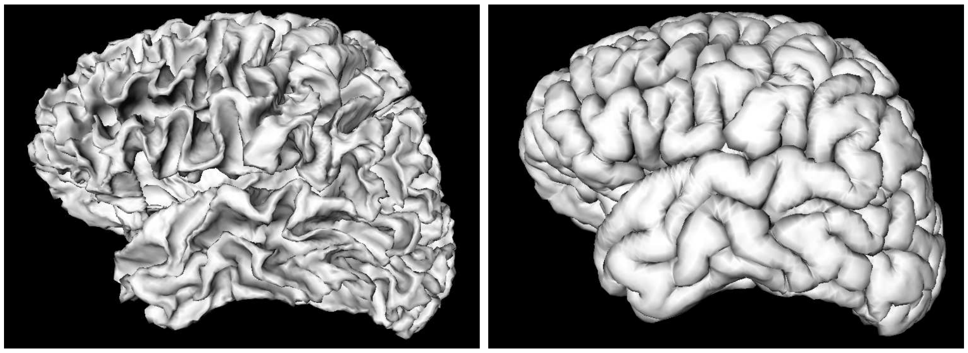
2.2.4 Surface Extraction

One of critical analysis stages for human brain mapping is the extraction of inner and outer cortical surface of the human cerebrum from a 3D MR image. A surface extraction algorithm such as Constrained Laplacian Anatomic Segmentation using Proximity (CLASP) uses a 3D deformable polygonal mesh model to construct the 3D cortical surfaces [10]. The algorithm first deforms an ellipsoid of the model



(a) T1-weighted image (b) Classified image (c) Overlaid

Figure 2.5: Tissue classification algorithm. Yellow, red and teal represent GM, WM and CSF respectively.



(a) WM surface (b) GM surface

Figure 2.6: Surfrace extraction algorithm.

inward in an iterative fashion until the classified WM surface is successfully constructed. The GM surface is then constructed by deforming the model outward from the constructed WM surface. The process of expending the deforming model to the classified GM surface has been very challenging due to the folded nature of the cortex. As a result, a laplacian map, which is generated by subtracting the skeletonized CSF from the classified cortical GM, is used to maintain topological correctness during extracting the GM surface. Researchers then measure cortical thickness of brain by calculating the distance between the two extracted surfaces. Figure 2.6 illustrates the WM and GM surfaces constructed from the CLASP algorithm.

2.3 Partial Volume Estimation (PVE)

The process of classifying three tissue types (WM, GM and CSF) in 3D MR images of the human brain is made difficult due to the *partial volume effect*. This effect blurs the intensity distinction of tissues at the boundary areas as shown in Figure 2.7. An algorithm, known as Partial Volume Estimation (PVE), has been developed to overcome the difficulties occurred during tissue classification due to the partial volume effect and to estimate the proportion of each tissue type present in a voxel more accurately.

Several different PVE algorithms have been proposed and distinguished based on how they model the partial volume effect. The PVE algorithm developed in [11] uses a statistical model, based on Markov Random field (MRF) theory, to address the partial volume effect. This model, first proposed in [22], assumes that the image intensity value of each *mixel* can be described as a sum of weighted random variables (RVs).

Statistical model for Mixel

Let the acquired 3D MR image be denoted by $X = \{x_i : i = 1, \ldots , N\}$ where *N* is the total number of voxels. In the image X, a voxel x_i composed of a mixture of several different tissue types is known as a *mixel* [22]. Let $j = \{1, \ldots , M\}$ represent the set of possible pure tissues present in the image and l_j represent the RV describing the tissue type *j*. The content of a mixel, x_i can there be expressed in term of the RVs as:

$$x_i = \sum_{j=1}^{M} w_{i_j} l_j, \tag{2.4}$$

where the weighting terms w_{i_j}, known as partial volume coefficients (PVCs), represent the fractional amount of tissue type *j* present in the voxel x_i ($w_{i_j} \in [0, 1]$ and $\sum_{j=1}^{M} w_{i_j} = 1$). The PVE algorithm is therefore divided into two main stages. The first stage classifies voxels in the volume into appropriate tissue class, while

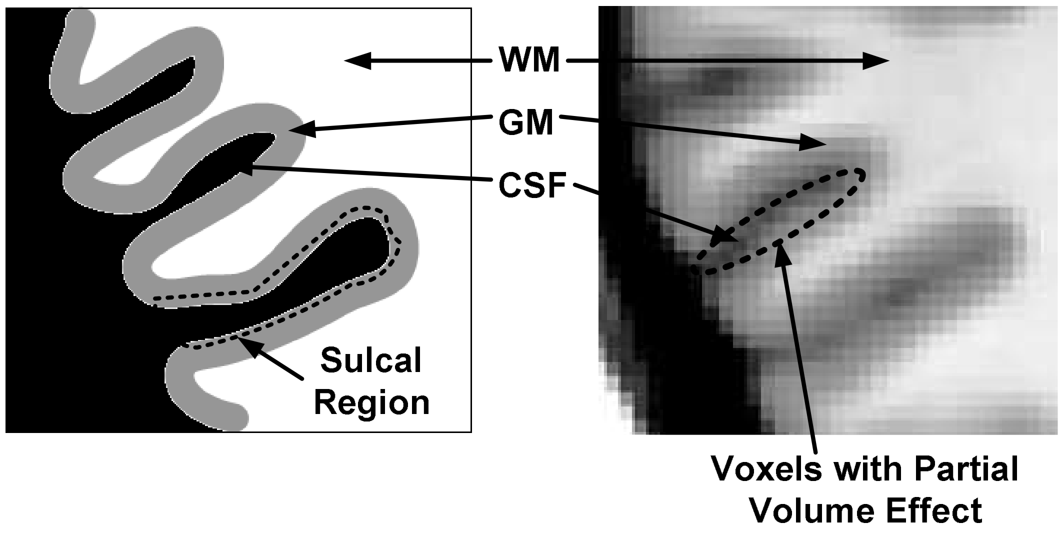


Figure 2.7: Partial volume effect on Sulcal region.

the second stage estimates the fractional amount (PVCs) of each pure tissue type present in each voxel.

2.3.1 Partial Volume (PV) classification

In the Partial Volume (PV) classification stage, each voxel is classified and labeled as one of three pure tissue types (CSF, WM and GM) or two mixed tissue types. The two possible mixed tissue types are mixed white matter/grey matter (WMGM) or mixed gray matter/CSF (GMCSF).

Let ***C*** represent a partial volume context image and voxels in ***C*** are labeled to one of five possible tissue classes. The estimation of the context image, ***C****, then can be calculated for the given MR image *X* using the maximum a posteriori (MAP) criterion as:

$$\mathbf{C}^* = \arg \underset{C}{max}\, P(\mathbf{C}|X) = \arg \underset{C}{max}\, P(\mathbf{C}) \prod_{i=1}^{N} P(x_i|c_i), \tag{2.5}$$

where $\prod_{i=1}^{N} p(x_i|c_i)$ and *P(C)* are the probability densities and the prior information for all possible tissue classes, respectively [11].

Estimation of the probability densities

The probability densities for all five possible tissue classes are first required to be calculated for the PVE classification. The mean and covariance of the three pure tissue types (CSF, GM, WM) are calculated based on the intensity distribution of the pre-classified voxels for each pure tissue. Since the probability densities of the pure tissues types which are equal to the multivariate Gaussian pdf [11], the probability density for pure tissue voxel x_i can be determined as:

$$p(x_i|c_i) = g\left[x_i; \mu_j, \sum_j\right] = \frac{1}{\sqrt{2\pi\Sigma_j}} e^{\frac{(x-\mu_j)^2}{2\Sigma_j}}, \tag{2.6}$$

where μ_j and $\sum_j$ represent the mean and covariance of the pure tissue type j. If voxel x_i is a *mixel* that is composed by tissue type *j* and *k*, the probability densities for this voxel can be also obtained from the integration of the multivariate Gaussian pdf [11] as :

$$p(x_i|c_i = \{j, k\}) = \int_0^1 g\left[x_i; \mu(w), \sum(w)\right] dw \tag{2.7}$$

where

$$\mu(w) = w\mu_j + (1 - w)\mu_k\,, \tag{2.8}$$

$$\sum(w) = w^2 \sum_j + (1 - w)^2 \sum_k . \tag{2.9}$$

The $\mu(w)$ and $\sum(w)$ represent the mean and covariance of the mixed tissue class, respectively. In Equation 2.7, {*j*, *k*} indicates voxel x_i is a *mixel* that is composed by tissue type *j* and *k*.

Estimation of the prior information

Since adjacent voxels can be assumed to likely have similar tissue types in data sets generated by the 1 mm resolution MRI scanner, a MRF model with 26-neighbour voxels is used to represent the prior information for all possible tissue classes due to its flexibility in defining the neighborhood system [11] [22] [23]. The prior information can be modeled using a MRF as:

$$P(C) \propto \exp\left(\beta \sum_{i=1}^{N} \sum_{k \in N_i} \frac{a_{ik}}{d(i,k)}\right), \tag{2.10}$$

where β is the MRF weighting parameter, *i* is the index of a voxel in the image *X*, *N* is the total number of voxels in image *X*, N_i is the 26-neighbour voxels around voxel *i*, *k* is the index of a voxel from N_i and *d(i,k)* is the distance between the voxel *i* and its neighbour voxel *k*. The value of a_{ik} is determined based on the tissue relationship between voxel *i* and *k* [11], that is,

$$a_{ik} = \begin{cases} 2 & : c_i = c_k \\ 1 & : c_i \text{ and } c_k \text{ share a component} \\ -1 & : \text{otherwise.} \end{cases} \tag{2.11}$$

During the tissue classification, the MRF model usually oversmoothes the boundaries between GM and CSF tissues in the deep sulcal regions of the brain shown as in Figure 2.7. As a result, [23] proposed a solution to overcome this artifact on sulcal regions using modified a_{ik} term and a curvature image, which pre-defined the CSF tissues in sulcal regions. The author modified the weighting term a_{ik} to ensure that the voxels in deep sulcal regions to be labeled as CSF or GMCSF tissue. The modified weighting term a_{ik} provided by [23] is

$$a_{ik} = \begin{cases} -2 & : c_i = c_k \\ -1 & : c_i \text{ and } c_k \text{ share a component} \\ & : c_i \notin \{GMCSF, CSF\} \text{ or } C(i) \geq 0 \\ f(C(i)) & : c_i \in \{GMCSF, CSF\} \text{ and } C(i) < 0 \\ 1 & : \text{otherwise,} \end{cases} \tag{2.12}$$

where *C(i)* is the value of the curvature at voxel *i* and *f(x)* is a “transform” function,

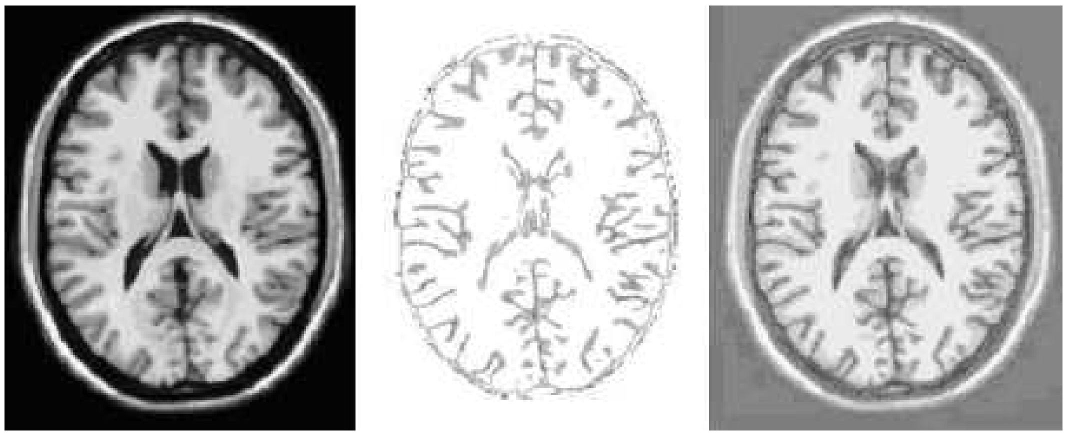


Figure 2.8: T1-weighted image and curvature image. (a) T1-weight MR image (b) Curvature image (c) Curvature image is overlaid on top of T1-weight

such as

$$f(x) = \frac{A}{1 + e^{-B(x+C)}} - 1, \tag{2.13}$$

which is used to calculate an appropriate MRF weight term for the voxel in the sulcal region. Note that overlapped image in Figure 2.8 illustrates that how well the predefined sulcal regions in a curvature image are correspond to the real regions in a T1 weighted image.

After the probability densities and prior information for every possible tissues class are estimated, a voxel i is classified as the tissue type which generated the maximum $P(c_i|x_i)$ term. The Iterated Conditional Modes (ICM) algorithm is used to determine the newly estimated context image C* in Equation 2.5. This algorithm iteratively re-estimates the context image using the tissue labels calculated in the previous iteration until convergence is reached.

2.3.2 PVC Estimation

After every voxel is labeled by one of the possible tissue types in the PV classification stage, the fraction of each pure tissue present in a mixel is calculated in the PVC estimation stage. If voxel *i* is a mixel containing tissue *k* and *j*, the fractional amount of tissue k and j within the voxel *i* can be calculated by employing the maximum-likelihood principle as:,

$$w_{i_k}* = 1 - w_{i_j}* \tag{2.14a}$$

$$\begin{aligned} w_{i_j}* &= \arg\max_{w\in[0,1]} g(x_i|\mu(w), \sum(w)) \\ &= \arg\max_{w\in[0,1]} \ln g(x_i|\mu(w), \sum(w)) \\ &= \arg\min_{w\in[0,1]} [(x_i - \mu(w))^T \sum(w)^{-1}(x_i - \mu(w)) \\ &+ \ln\det(\sum(w))], \end{aligned} \tag{2.14b}$$

respectively [11]. In Equations 2.14a and 2.14b, $\mu(w)$ and $\sum(w)$ are mean and covariance of the mixed tissue type, respectively and PVCs for all other tissues types are zero. A grid search algorithm is used to solve the maximum-likelihood PVC estimation [11].

2.3.3 Overview of the PVE Algorithm

Figure 2.9 shows four input images required to perform the PVE algorithm on a single 3D MR brain image. Figure 2.9(a) is a normal T1-weighted MR image from an MR scanner. Pre-generated curvature image is given in Figure 2.9(b) and this image contains the information of CSF in the sulcal region. The classified image is used to calculate the mean and covariance of the pure tissue classes. In Figure 2.9(c), white, grey, dark grey and black regions represent WM, GM, CSF and BG, respectively. The Skull mask image in Figure 2.9(d) is used to reduce the computation complexity by eliminating the voxels in the background.

An overview of the described PVE algorithm is provided in Figure 2.10. The algorithm first calculates the mean and covariance of pure tissue type from the pre-classified image before performing the PV classification stage. The algorithm then evaluates whether the voxel is in background or not. If the voxel is not located in background, the PVE algorithm performs the PV classification stage and labels the

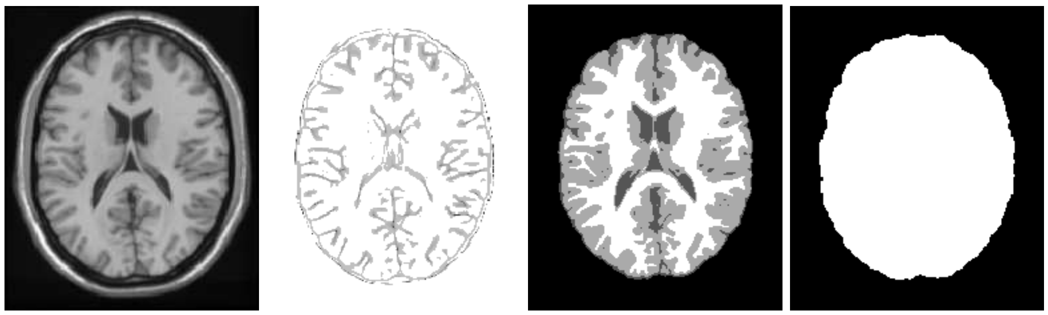

(a) T1-weighted image (b) Curvature image (c) Classified image (d) Skull mask image

Figure 2.9: Partial Volume Estimation input images.

voxel with an appropriate tissue type from the calculated probability densities and prior information of all possible tissues as described in Section 2.3.1. Otherwise, the algorithm classifies the voxel as background. The PV classification stage is iteratively executed and terminates when the total number of changed voxels during the previous iteration is lower than the total number of changed voxels during the current iteration or the maximum number of iterations (20) is reached. After the classification is done, the fractions of each pure tissue class within the voxels are estimated as described in Section 2.3.2.

Figure 2.11 illustrates four output images generated from the PVE algorithm. Each voxel in the PVE classified image is labeled to the dominant tissue type among all possible tissue types as shown in Figure 2.11(d). Each color represents different tissue types. Purple, blue, teal, green and yellow regions represent CSF, GM, WM, GNCSF, WMGM and background respectively. The PVE CSF, GM and WM images provide the proportion of CSF, GM and WM present in each voxel as shown in Figures 2.11(a), 2.11(b), 2.11(c), respectively.

2.4 HPRC System and High-Level Language

High-Performance Reconfigurable Computers (HPRCs) combine high performance CPUs with reprogrammable accelerators such as Field Programmable Gate-Arrays (FPGAs). The computationally intensive portions of the floating-point scientific applications can be implemented on FPGAs to achieve full fine-grained parallelism,

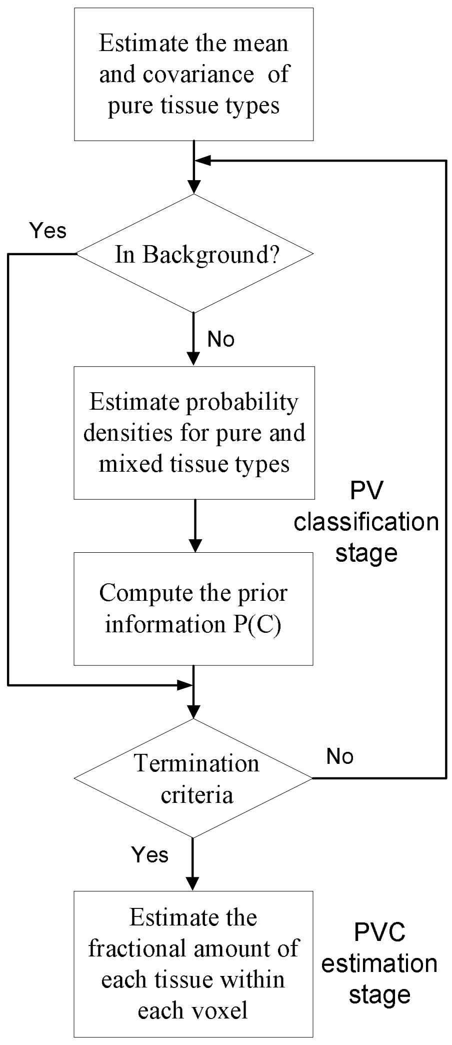

Estimate the mean and covariance of pure tissue types
Yes
In Background?
No
Estimate probability densities for pure and mixed tissue types
PV classification stage
Compute the prior information P(C)
Termination criteria
No
Yes
Estimate the fractional amount of each tissue within each voxel
PVC estimation stage

Figure 2.10: Partial Volume Estimation algorithm.

while other portions that exploit the coarse-grained parallelism can be implemented on the conventional CPU. However, the complex nature of hardware design for FPGAs requires specialist engineering knowledge and presents a significant barrier to scientific users with only a limited amount of programming background. Recently, a number of High-Level Languages (HLL) for programming FPGAs have emerged that aim to lower this barrier and abstract away hardware-dependent details.

Several commercial HPRCs and HLLs have been developed and became available for the researchers to accelerate different scientific applications. The following subsections present SGI RASC RC100 HPRC system and Mitrion-C HLL and their

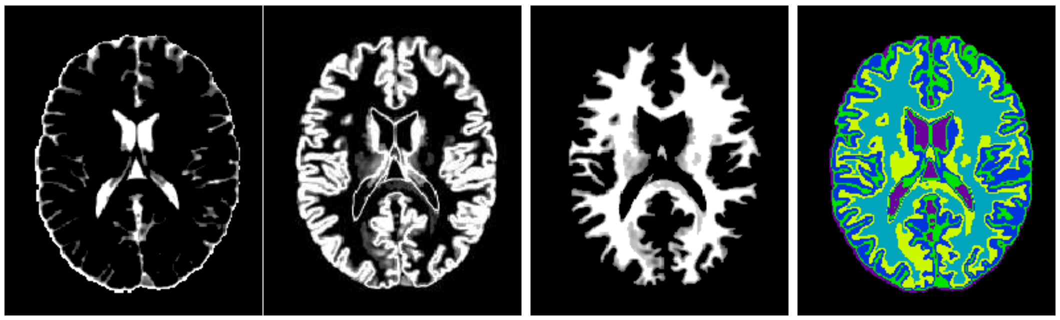

(a) PVE CSF (b) PVE GM (c) PVE WM (d) PVE classified

Figure 2.11: Partial Volume Estimation output images.

special features.

2.4.1 SGI RASC RC100 System

Silicon Graphic Inc. (SGI) Reconfigurable Application Specific Computing (RASC) technology is one of the commercial HPRC systems that allow researchers to dramatically reduce the algorithm computation time by exploiting the fine-grain parallelism provided by FPGA-accelerators. The RC100 system is the third generation of RASC technology hardware module and is connected to the Altix 350 multiprocessor system via a NUMAlink interconnect network as shown in Figure 2.12. The Altix 350 has eight 1.5GHz Intel Itanium 2 CPUs with 16GB of shared physical memory.

As illustrated in Figure 2.13, a RASC RC100 system is composed of two Xilinx Virtex-4 LX200 FPGAs, two TIO interface ASICs and a bitstream loader FPGA [24]. Table 2.1 provides a summary of available hardware resource of a Virtex-4 LX200 FPGA. Each computational FPGA has five 8MB QDR SRAM DIMM memory banks, although the core services currently provide access to 32MB per FPGA. The TIO ASIC is a peer-to-peer I/O brick which handles the interface between NUMAlink interconnect and IO buses from the algorithm FPGAs. A loader FPGA enables fast bitstream loading into the computational FPGAs.

The bitstream loaded into algorithm FPGAs consists of two different blocks: *algorithm block* and *core service block* as shown in Figure 2.14. The components

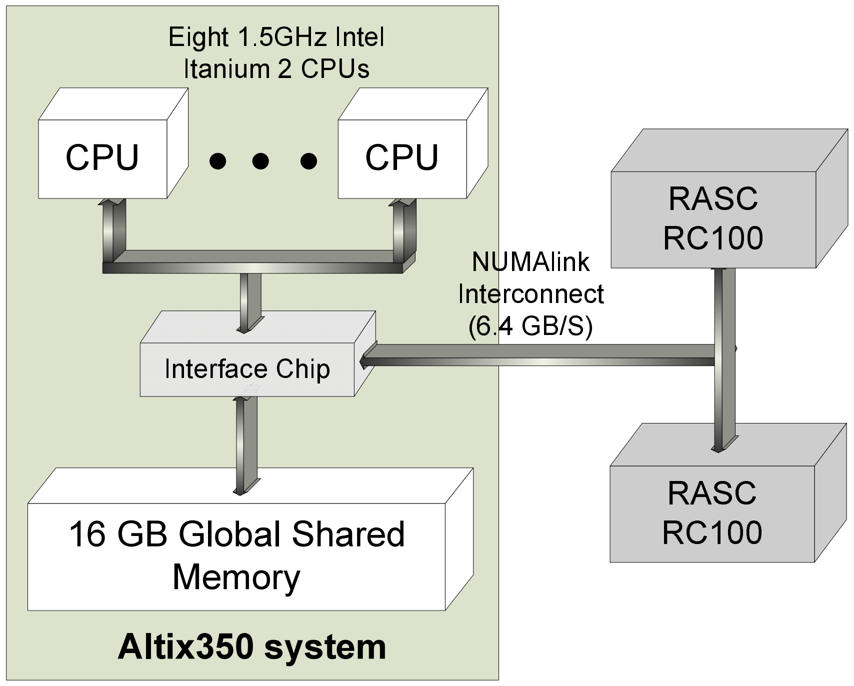

Eight 1.5GHz Intel Itanium 2 CPUs
CPU
CPU
RASC RC100
NUMAlink Interconnect (6.4 GB/S)
Interface Chip
RASC RC100
16 GB Global Shared Memory
Altix350 system

Figure 2.12: SGI Altix350 and RASC RC100.

Table 2.1: A summary of Virtex-4 hardware resource.

Resource	Availability
Logic Cells	200,448
Slices	89,088
18 X 18 multiplier	96
18kb Block RAM	336

inside of the pre-synthesized core service block help execution of the user implemented algorithm in the algorithm block. The read/write Direct Memory Access (DMA) and SSP interface on the core service block control the data transfer between the external SRAMs of FPGA and the main memory on the Altix system. The algorithm block has an access to the debug port and algorithm defined registers via Peer Input/ouput (PIO) block. Memory Mapped Registers (MMR) handle the registers used to control the FPGAs. Algorithm control block includes logics used to control the implementation on the algorithm block.

The RASC abstract layer provides Application Programmable Interface (API) functions through which the host program on the Altix system can communicate

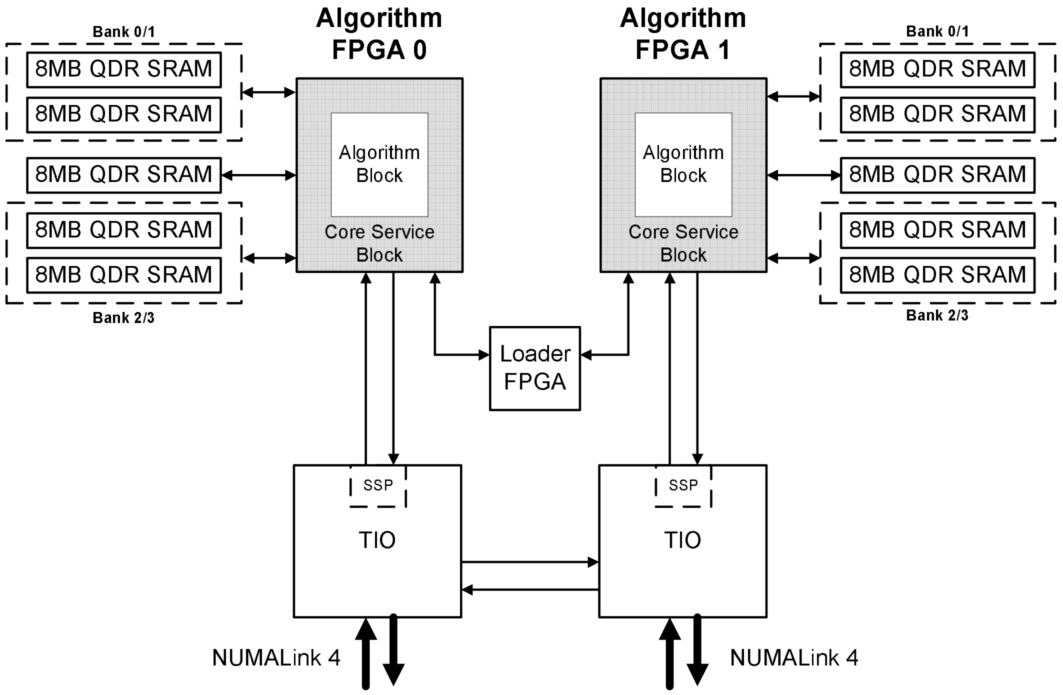


Figure 2.13: RASC RC100 system.

with the RASC RC100 modules. The developers can implement the abstract layer as an *Algorithm* or a *Co-processor (COP)* layer. The Algorithm layer is built on top of the *COP* layer and treats multiple devices as a single large logical device, whereas the *COP* layer treats each device individually. Due to the difference between the two layers, the developer cannot use both layers simultaneously and should decide which layer to use based on the algorithm beforehand. The API functions provide developers some standard and advanced interfaces required to run the implemented algorithm. Table 2.2 provides a summary of the abstract layer functions. When a host program calls the *rasclib* functions, they are not executed immediately. Instead, the host program first queues the functions within the RASC abstract layer. When the $rasclib\_algorithm\_commit$ function is executed, the host program then sends the queued commands to the driver. When all of the committed commands are finished, the $rasclib\_algorithm\_wait$ function returns a validated value to the host program [24].

Two special features, *streaming* and *wide-scaling*, help efficiently execute algorithms on large data sets in the SGI RASC RC100 system. Streaming reduces

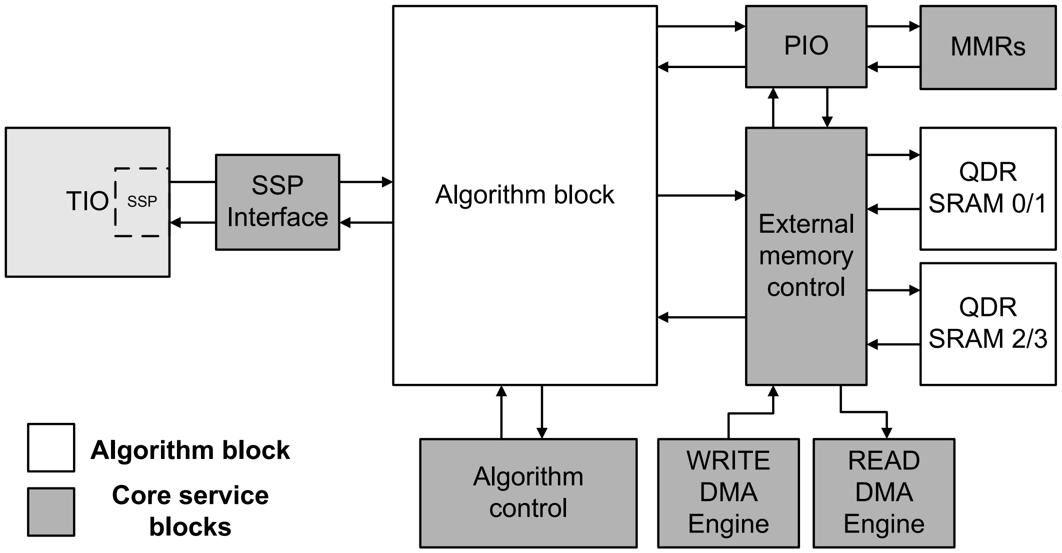


Figure 2.14: The block diagram of Algorithm FPGA.

Table 2.2: A summary of RASC Abstract Layer Functions.

Functions	Description
$rasclib\_resource\_alloc$	allocate the devices
$rasclib\_resource\_free$	free the allocated devices
$rasclib\_algorithm\_open$	open bitstream
$rasclib\_algorithm\_send$	send input buffer to external memory
$rasclib\_algorithm\_go$	start computing algorithm
$rasclib\_algorithm\_receive$	receive output buffer from external memory
$rasclib\_algorithm\_commit$	commit queued commands
$rasclib\_algorithm\_wait$	wait till all commands are executed
$rasclib\_algorithm\_close$	close the allocated bitstream

the overhead of data transfer by overlapping the data loading and unloading with algorithm execution. As shown in Figure 2.15(a), the streaming feature requires the designer to split the 32MB SRAM into two memory blocks; one for input memory (Bank A) and the other for output memory (Bank B). These memory banks are further segmented into two sub-blocks so that the implemented algorithm can process data in Bank A1 and write the results in Bank B0, while the algorithm loads the next input data into Bank A0 and unloads the final computed results from Bank B1 to the host buffer. When the algorithm processes all data in Bank A1, the FPGA starts executing on the loaded segment (Bank A0), while the freed segment (Bank

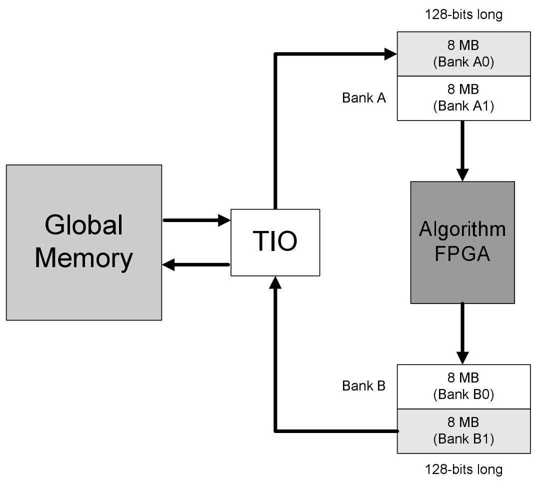
A1) begins loading the next input data set.

The wide-scaling feature allows the algorithm to be automatically scaled over multiple FPGAs, by equally distributing large input data sets to the available FPGAs to process them simultaneously. This feature uploads the generated bitstream into all available FPGAs and re-assembles the computed results from each FPGA into a single results data set automatically. As shown in Figure 2.15(b), the wide-scaling, therefore provides an implementation to easily exploit coarse-grained parallelism. In the cases where the algorithm requires different bitstreams for each FPGA or a non-uniform data partitioning, the programmer needs to manually scale the application.

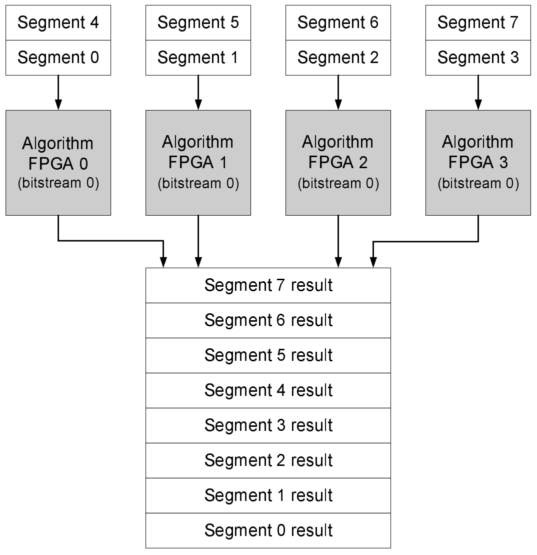
2.4.2 Mitrion-C HLL

Dataflow-oriented Mitrion-C is a fully parallel programming HLL. Since its syntax is very similar to ANSI-C, it is readily available to programmers with only software background. As shown in Figures 2.16, 2.17 and 2.18, the Mitrion-C HLL is a single-assignment language which emphasizes defining data dependencies instead of order of the execution [25]. Therefore, the programmer only needs to be concerned about expressing the dataflow of the algorithm, and not any hardware-based concerns such as timing. The programmer can express parallelism using explicit language constructs and exploit it in the form of *vectorization* and/or *pipelining*, based on the selection of data types and the parallel constructs provided by Mitrion-C [17].

There are two available array data types in Mitrion-C HLL: *lists* of elements accessible sequentially, and *vectors* in which all elements can be addressed at once in any order. The resulting architecture is determined by the selection of data type (list or vector) and the parallel operator applied to it. A parallel *foreach* construct indicates a data-parallel loop. While combining the vector data type with the foreach loop creates explicit parallel structures with multiple processing elements (PEs), this design style rapidly consumes FPGA resources as shown in Figure 2.16. In contrast, applying a foreach loop to a list implements a pipelined architecture, where



(a) Streaming [1]



(b) Wide-scaling

Figure 2.15: Special features provide by SGI RASC technology.

a single PE is generated to process the input data sequentially as shown in Figure 2.17.

The programmers can also use a combination of both lists and vectors as illustrated in Figure 2.18. While this design approach requires more resources than the list-foreach approach, it requires less than the vector-foreach approach. By contrast, this design style exploits more parallelism than the list-foreach design and less than the vector-foreach design. As a result, the programmer needs to select an appropriate data type for the application to fully utilize available hardware resources

```
Mitrion-C 1.2;
// options: -cpp

main ()
{
    int:8[6] data = [0 .. 5];  // Vector data type
    res = foreach(i in data){
        temp = i * i);
    }temp;
}(res);
```

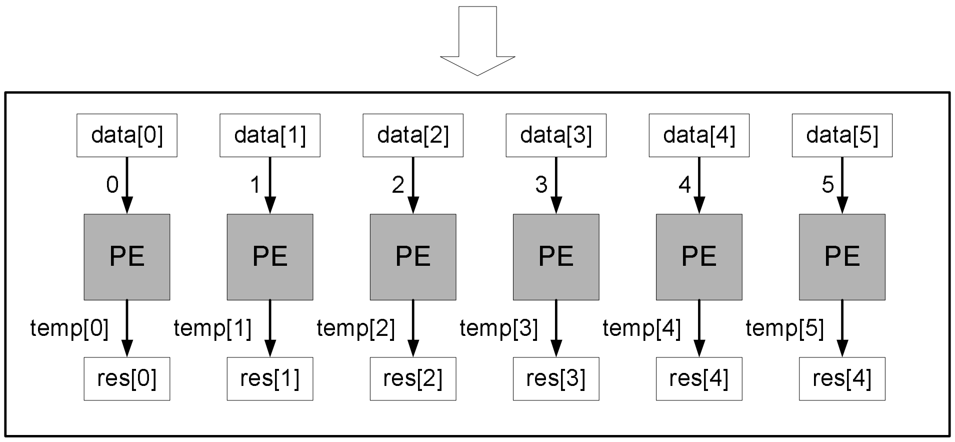


Figure 2.16: Mitrion-C vector data type with foreach loop.

and parallelism, since there clearly is a tradeoff between the circuit area and the execution time.

Mitiron Software Development Kit (SDK) helps the programmers to develop FPGA applications that are time consuming. The simulator/debugger in SDK generates a graphical representation of the hardware data flow graph and is the key in identifying data-dependency and bottlenecks in the design. In the Mitrion simulator/debugger, developers can also use breakpoint and watch functions similar to the ones used in software debuggers to determine program errors. Figure 2.19 is an example of the graphical simulator/debugger generated from the code example in Figure 2.18. As shown in Figure 2.19, the graphical simulator/debugger shows the data-dependency and variables to help programmers optimize and debug the design. Another tool in Mitrion SDK, the Mitrion Compiler, generates the Mitrion machine code from Mitrion-C source code. The machine code is used to generate a

```
Mitrion-C 1.2;
// options: -cpp

main ()
{
    int:8<6> data = <0 .. 5>;  // List data type
    res = foreach(i in data){
        temp = i * i);
    }temp;
}(res);
```

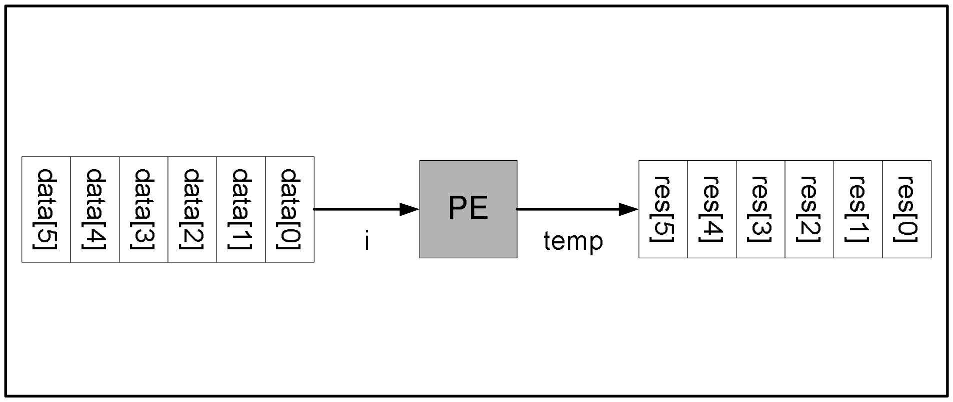
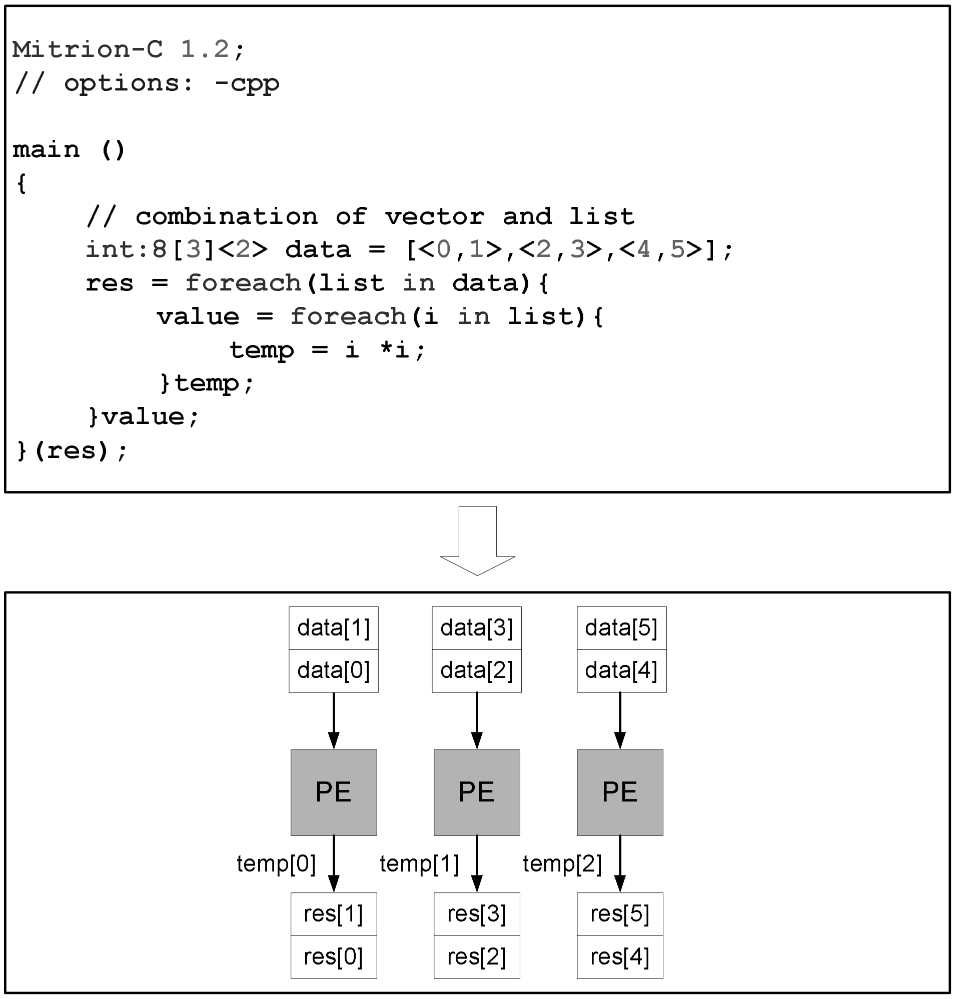


Figure 2.17: Mitrion-C list data type with foreach loop.

fine-grained massively parallel Mitrion Virtual Processor (MVP) correspondent to the application developed in the Mitrion-C HLL [17].

Figure 2.20 shows the overview of the Mitrion design flow. The Mitrion compiler generates Mitrion machine code which can be used either for simulation in the debugger/simulator or for Mitrion Processor Configurator depending on the stage of the development cycle. The Mitrion Processor Configurator generates VHDL which corresponds to the MVP constructed from the Mitrion machine code and the pre-defined target FPGA architecture. The generated MVP is designed to run at a pre-fixed frequency and is wrapped in the core services which provide memory and communications interfaces between the HPRC system and CPUs in the host system. Mitrion platform currently supports several commercial HPRC systems such as SGI



```
Mitrion-C 1.2;
// options: -cpp

main ()
{
     // combination of vector and list
     int:8[3]<2> data = [<0,1>,<2,3>,<4,5>];
     res = foreach(list in data){
          value = foreach(i in list){
               temp = i *i;
          }temp;
     }value;
}(res);
```

Figure 2.18: Combination of vector and list data with foreach loop.

RASC RC100, Cray XD1, Nallatech BenDATA-DD, and DRC Reconfigurable Processor Unit (RPU) [26]. Third party Computer-Aided Design (CAD) tools, such as Synplify Pro and Xilinx ISE, are used to synthesize and place-and-route the design to generate bitstream which will be uploaded into the FPGA-accelerator.

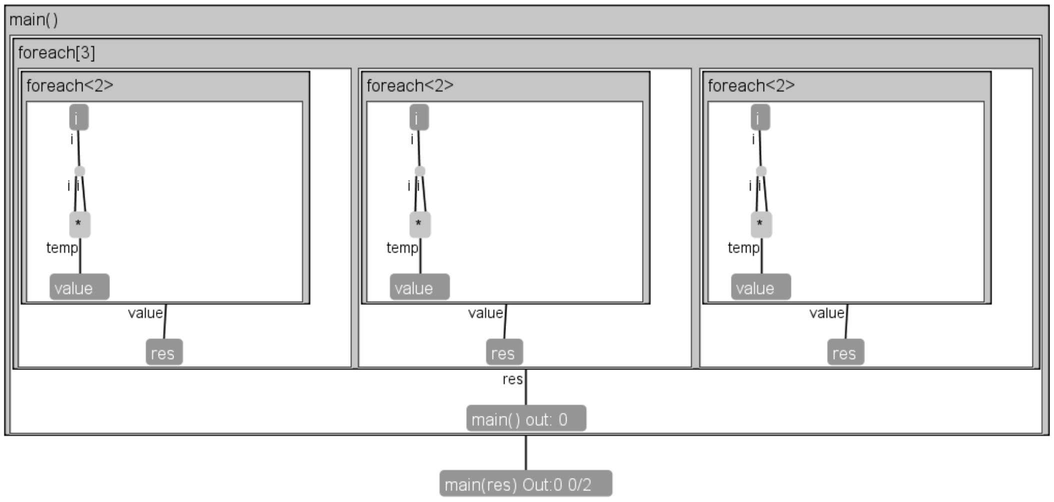


Figure 2.19: Mitrion graphical simulator/debugger.

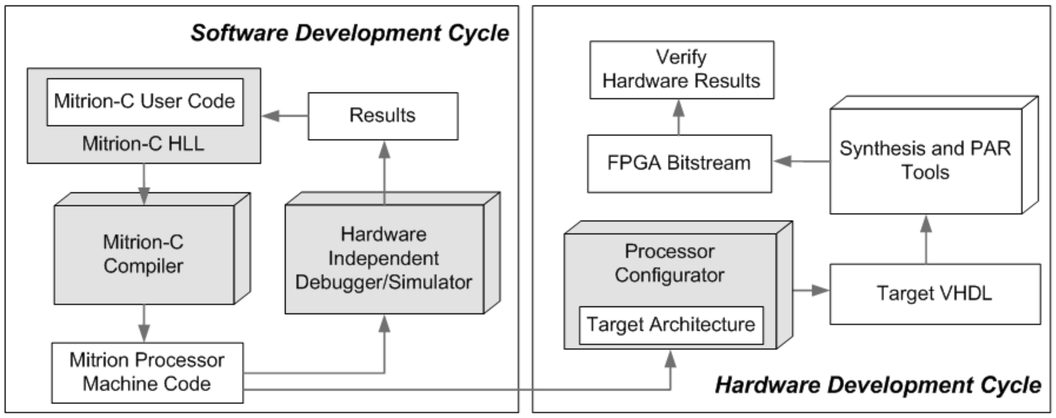


Figure 2.20: Mitrion design flow.

Chapter 3

Hardware Implementation of PVE

3.1 Introduction

The PVE algorithm described in Section 2.3 is very attractive for hardware acceleration due to its ability to independently process voxels. We implemented the PVE algorithm on RASC RC100 FPGA-accelerators using the Mitrion-C HLL to reduce its computation time. The FPGA-implemented PVE algorithm was optimized to fully utilize available parallelism and hardware resources on the RASC RC100.

3.2 Overview of Design Flow

In developing the hardware implementation, we first profiled the CPU-based PVE algorithm in order to identify the computationally intensive functions. The identified functions were then re-developed using the Mitrion-C HLL for hardware implementation. We verified their functionality using Mitrion simulator/debugger before the time-consuming synthesis and place-and-route operations. After the verification, we generated MVP from the source code and performed synthesis and place-and-route operations to generate its bitstreams. After the bitstreams were successfully generated, we uploaded them to the RASC bitstream registry using the device manager (devmgr). We also developed a host program which maintains communication between the Altix 350 system and the algorithm FPGAs on the RC100 modules using C programming language. Finally, we verified the results from the FPGA-based algorithm against the ones from the CPU-based algorithm. The following sections present the detailed design concepts that we used to implement the PVE algorithm on the RASC RC100 using the Mitrion-C HLL.

3.3 Profiling CPU-based PVE Algorithm

Unfortunately, not every function in the PVE algorithm can be implemented on the RASC RC100 FPGA-accelerators, due to the limited availability of hardware resources and memory. As a result, it is very important to identify the primary performance bottleneck of the PVE algorithm and to implement it on FPGA-accelerators for hardware acceleration. This process can be completed by profiling the CPU-based PVE algorithm, as a report generated by such a profiling tool will indicate which parts of a program are consuming most of the execution time and which functions are called while it was executing.

We profiled the CPU-based PVE algorithm using GNU *gprof* [27] to identify the computationally intensive portions and places where data-parallelism could be exploited on the FPGA. The resulting profiling report is shown in Table 3.1. This report indicates that the PV classification stage, described in Section 2.3.1, is the primary performance bottleneck, as the stage consumes approximately 94% of the total computation time. Especially, in the PV classification stage, the estimation of the probability densities function consumes approximately 13% of the total PVE algorithm computation time due to the expensive computation of the stage. Moreover, approximately 81% of the total PVE algorithm computation time is consumed by the estimation of the prior information for all tissue types in the PV classification stage. This stage becomes one of the main computation bottleneck of the CPU-based PVE algorithm because it requires to re-estimate the prior information iteratively. As a result, these two functions are the most ideal candidate for implementation on the RASC RC100 FPGA-accelerators. Figure 3.1 shows the distribution of the PVE algorithm functions on the Altix 350 system and RASC RC100.

Table 3.1: GNU gprof profiling report of the CPU-based PVE algorithm.

Functions	CPU-based
†The estimation of the prior information	301s (81%)
†The estimation of the probability densities	49.8s (13%)
PVC Estimation	3.8s (1%)
IO functions	18.2s (5%)
Total	372.8s

† Two subsections within the PV classification stage

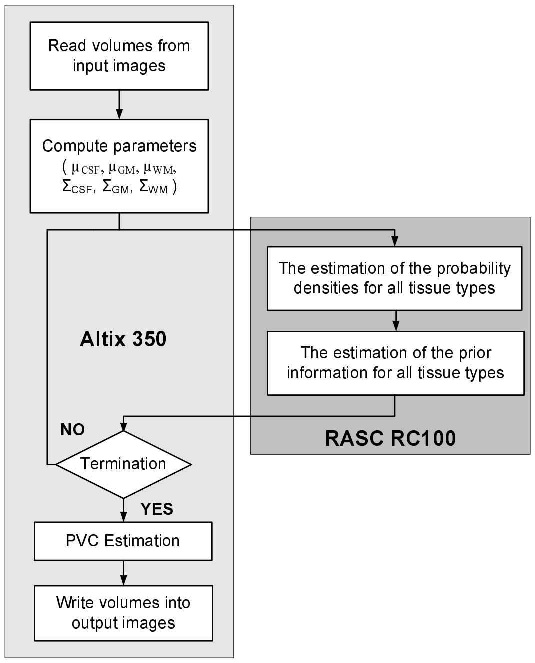

Read volumes from input images
Compute parameters (μ_{CSF}, μ_{GM}, μ_{WM}, Σ_{CSF}, Σ_{GM}, Σ_{WM})
The estimation of the probability densities for all tissue types
The estimation of the prior information for all tissue types
Altix 350
RASC RC100
NO
Termination
YES
PVC Estimation
Write volumes into output images

Figure 3.1: PVE algorithm on RASC RC100.

After we identified the computationally intensive functions from profiling, we estimated the total possible performance improvement that can be achieved by the FPGA-based PVE algorithm using Amdahl's law,

$$SU = \frac{T_{SW}}{T_{HW}} = \frac{1}{(1 - F_{HW}) + \frac{F_{HW}}{Speedup_{HW}}}, \tag{3.1}$$

where T_{SW} and T_{HW} represent the total computation time of the CPU-based and FPGA-based implementations, respectively, while the SU represents the overall speedup. In Equation 3.1, F_{HW} is the fraction of the computation time of the function which is implemented on the FPGA and $Speedup_{HW}$ is the performance gain from the hardware-implemented function. Amdahl's law states that the overall performance improvement (SU) is limited by the fraction of the function (F_{HW}) that can be accelerated [28]. The FPGA-based PVE algorithm can achieve an overall speedup of 5× when the performance of the PV classification implemented on FPGA-accelerator (F_{HW}=0.94) is enhanced by approximately a factor of 6.7.

3.4 FPGA-implemented PD Estimation

The PD estimation function estimates the probability densities (PDs) of every possible tissue class for voxels in the volume. The PDs are only estimated once at the first iteration of the PV classification stage. Algorithm 1 provides an overview of the CPU-based PD estimation function. The CPU-based implementation, which is developed in C, first reads the mean and covariance of pure tissue types (CSF, GM and WM) which are pre-calculated from the previous step. The algorithm reads the intensity of the voxel in the skull mask volume and verifies whether the voxel belongs to background or not. If the voxel is part of the background, the algorithm sets the PDs of all tissue classes to zero and labels the voxel as BG. Otherwise, the PDs of pure and mixed tissue for the voxel are calculated from Equations 2.6 and 2.7, respectively. The five generated PDs are normalized to ensure that the summation of all PDs is equal to one. After normalization, the algorithm classifies the tissue type of the voxel as the tissue class which has the highest PD among the five. The PD estimation terminates when the described processes are performed on every voxel in the T1-weight volume.

We first identified the functions that can benefit from hardware parallelism to implement the PD estimation function on the RASC RC100 FPGA-accelerators. Since the tasks of computing PDs of pure and mixed tissue are independent, we designed and implemented them to perform simultaneously. Moreover, since the tissue PDs of voxels are independent, we designed the hardware implementation to

Algorithm 1 Estimation of probability density for all possible tissue classes

```
Require: The first iteration of the PV classification
  INPUT : Skull mask image, T1-weighted image
  $\mu_{CSF} \Leftarrow$ mean of CSF
  $\mu_{GM} \Leftarrow$ mean of GM
  $\mu_{WM} \Leftarrow$ mean of WM
  $\Sigma_{CSf} \Leftarrow$ covariance of CSF
  $\Sigma_{GM} \Leftarrow$ covariance of GM
  $\Sigma_{WM} \Leftarrow$ covariance of WM
  for $i = 0$ to $last\_voxel - 1$ do
    if voxel $\in$ background then
      $pdf_i[CSF] \Leftarrow 0$
      $pdf_i[GM] \Leftarrow 0$
      $pdf_i[WM] \Leftarrow 0$
      $pdf_i[WMGM] \Leftarrow 0$
      $pdf_i[GMCSF] \Leftarrow 0$
      $class_i \Leftarrow$ background
    else
      x $\Leftarrow$ intensity value of voxel from T1-weighted image
      for all possible tissue classes do
        if tissue = CSF, GM or WM then
          $pdf_i[tissue] \Leftarrow$ gaussian(x, $\mu_{tissue}$, $\Sigma_{tissue}$)
        else {tissue = WMGM or GMCSF}
          $pdf_i[tissue] \Leftarrow 0$
          for w = 0 to 49 do
            compute $\mu_{tissue}(w)$
            compute $\Sigma_{tissue}(w)$
            $pdf_i[tissue]$ = $pdf_i[tissue]$ + gaussian(x, $\mu_{tissue}(w)$, $\Sigma_{tissue}(w)$)
          end for
        end if
      end for
      $norm\_pdf_i \Leftarrow$ nomalize $pdf_i$
      $class_i \Leftarrow \underset{tissue}{\text{argmin}}\{norm\_pdf_i[tissue]\}$, tissue = CSF, . . . , GMCSF
    end if
    CSF pdf volume[i] $\Leftarrow norm\_pdf_i[CSF]$
    GM pdf volume[i] $\Leftarrow norm\_pdf_i[GM]$
    WM pdf volume[i] $\Leftarrow norm\_pdf_i[WM]$
    WMGM pdf volume[i] $\Leftarrow norm\_pdf_i[WMGM]$
    GMCSF pdf volume[i] $\Leftarrow norm\_pdf_i[GMCSF]$
    classified volume[i] $\Leftarrow class_i$
  end for
  OUTPUT : CSF pdf volume, GM pdf volume, WM pdf volume, WMGM pdf
  volume, GMCSF pdf volume, classified pdf volume,
```

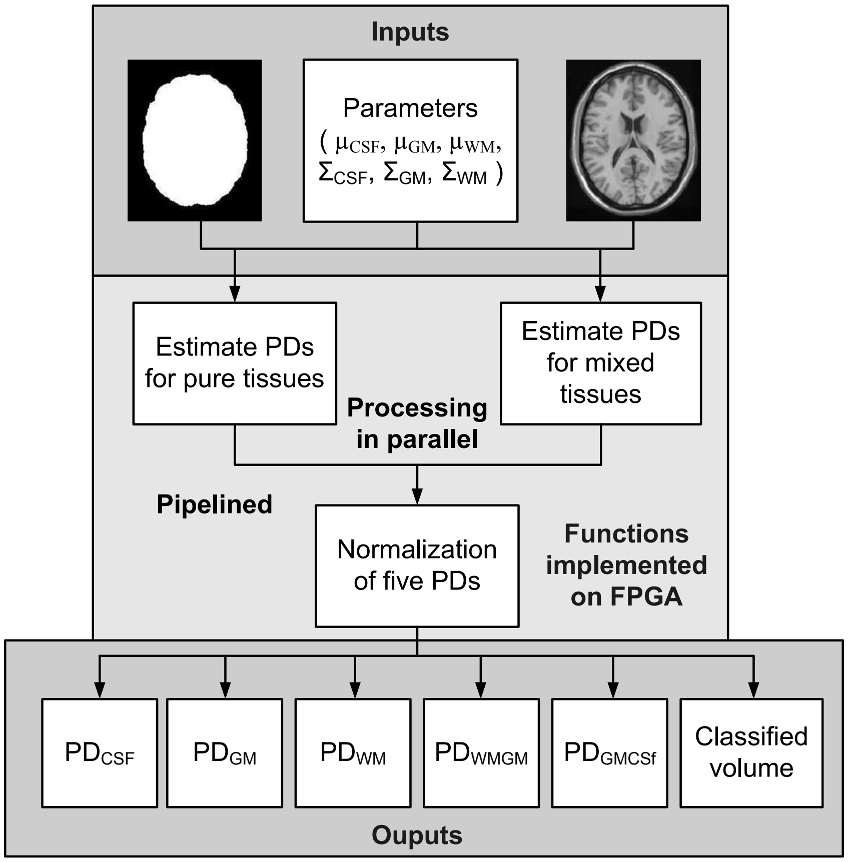

Inputs
Parameters
(μCSF, μGM, μWM, ΣCSF, ΣGM, ΣWM)
Estimate PDs for pure tissues
Estimate PDs for mixed tissues
Processing in parallel
Pipelined
Normalization of five PDs
Functions implemented on FPGA
PDCSF
PDGM
PDWM
PDWMGM
PDGMCSf
Classified volume
Ouputs

Figure 3.2: Overview of the FPGA-implemented PDs estimation function.

process the voxels in a pipelined fashion. Figure 3.2 illustrates the overview of the FPGA-implemented PDs estimation function. The following sections presents the detailed design principles and techniques used to implement PD estimation functions on FPGAs.

3.4.1 Input and Output Buffers

The PD estimation function requires two input images: the T1-weight and the skull mask image. Each image is composed of 181 slices and each slice contains 39277 voxels (217 $\times$ 181). The intensity values of voxels in the T1-weight image are stored in unsigned short data type (2 Bytes). The values in the skull mask image are stored in unsigned char data type (1 Byte) to identify the location of voxel (0 $\Rightarrow$ if it is in the background, 1 $\Rightarrow$ otherwise). The PD estimation function also generates six resulting volumes: PD$_{CSF}$, PD$_{GM}$, PD$_{WM}$, PD$_{WMGM}$, PD$_{CSFWM}$

Table 3.2: Summary of input/output file formats for the PDs estimation function.

Volume	Data type	Total Size
T1-weighted image (Input)	Unsigned short (2B)	13.56 MB
Skull image (Input)	Unsigned char (1B)	6.78 MB
PDs (Output)	Unsigned short (2B)	13.56 MB
Classified image (Output)	Unsigned chart (1B)	6.78 MB

and a classified image. Each PD volume contains PDs of voxels for a specific tissue type. The calculated floating-point PDs are converted to unsigned short data type and stored in a file to reduce data transfer overhead. The classified image uses unsigned char data type to store the classified tissue type of each voxel. Table 3.2 shows the summary of the input and output files for the PDs estimation function.

Due to the limitation of the available external memories in RASC RC100, we first carefully designed the usage of the off-chip memory before we ported the PDs estimation function into the Mitrion-C HLL and implemented on FPGA-accelerators. Figure 3.2 indicates that the input buffer of normalization function includes five single precision floating-point PD volumes. Therefore, the total size of the input buffer is approximately 142.38MB (5 $\times$ 7109137 voxels $\times$ 4B), which is larger than the size of the available external memories on the RASC RC100 module (16MB). As a result, we applied the RASC streaming technique described in Section 2.4.1 to reduce data transfer overhead between the host system and the RASC RC100. The available input and output off-chip memories were set to 8MB instead of 16MB and then the total number of voxel can be stored in 8MB SRAM were manually calculated as:

$$\begin{aligned} \text{number of voxel per 8MB segment} &= \frac{\text{total number of voxel}}{\text{number of segment}} \\ &= \frac{181 \times 217 \times 181}{\text{142.38 MB / 8MB}} \\ &\simeq 399456 \text{ voxels.} \end{aligned} \tag{3.2}$$

Since each segment can transfer 399456 voxels, we needed to transfer eighteen

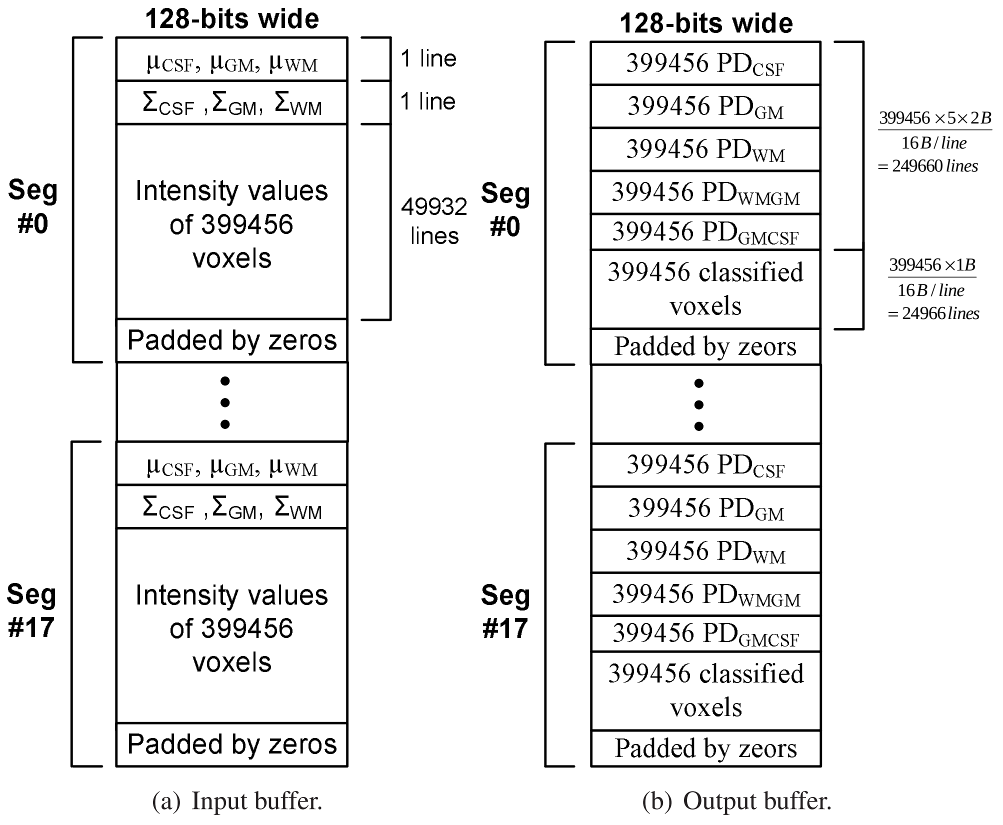


Figure 3.3: The structure of input and output buffer for FPGA-implemented PDs estimation.

segments (7109137 voxels / 399456 voxels) from the Altix 350 system to the off-chip memory of the algorithm FPGAs. Figures 3.3(a) and 3.3(b) show the structure of input and output buffer used for FPGA-implemented PDs estimation function, respectively.

3.4.2 FPGA-based PD Estimation for Pure Tissue Classes

We first implemented the PD estimation for pure tissue classes on the hardware accelerators. Figure 3.4 illustrates its functional building blocks. The implemented design first reads the mean and covariance of pure tissues from the external memory, and then stores them in the internal memory of FPGA. The implemented design calculates the single precision floating-point PDs of three pure tissue classes using the mean and covariance values and then writes them into the external SRAM in the order shown in Figure 3.4.

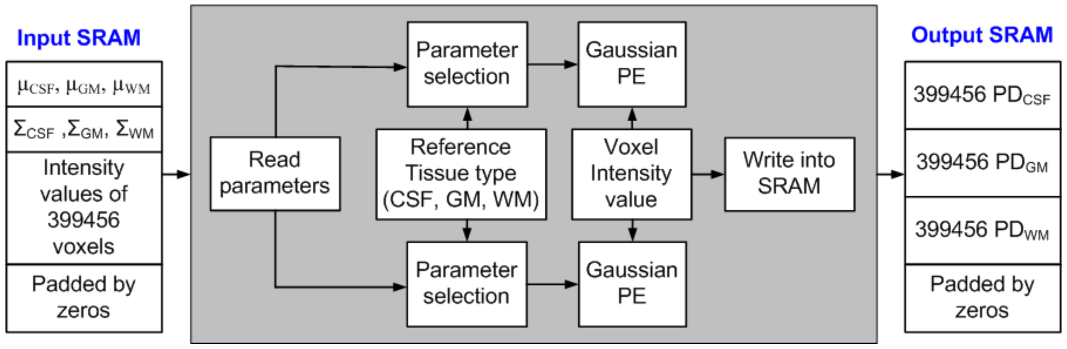


Figure 3.4: Functional blocks of FPGA-implemented PD estimation for pure tissue types.

We implemented the gaussian probability density function, provided in Equation 2.6, on FPGA-accelerators using the Mitrion-C HLL to compute PDs of three pure tissue classes. The gaussian PE was composed of six multipliers, one divider, one square root function and one exponent function, as shown in Figure 3.5. Mitrion-C constructed these single precision floating-point arithmetic functions using 18-bit $\times$ 18-bit multipliers and logic cells provided by the Xilinx Virtex-4. We were only able to implement two gaussian PEs to compute the PDs simultaneously due to the limited hardware resources available on the Xilinx Virtex-4. The Mitrion-C list data type was also used with *foreach* loop such that the PEs can generate outputs in a pipelined fashion.

3.4.3 FPGA-based PD Estimation for Mixed Tissue Classes

We also ported the PD estimation for mixed tissue classes into Mitrion-C HLL and implemented on FPGA-accelerators. The implementation was very similar to the one described in Section 3.4.2, however, the required integral calculation made it more complicated. Equation 2.7 estimates the PDs of the two mixed tissue classes by integrating the gaussian PDs from zero to one with an increment of w. Figure 3.6 shows the functional building blocks of the FPGA-implemented PD estimation for mixed tissues. The implemented design first reads the mean and covariance of pure tissues from the external memory, and then stores them in the internal memory. The mean and covariance of the mixed tissue class are calculated based on the reference

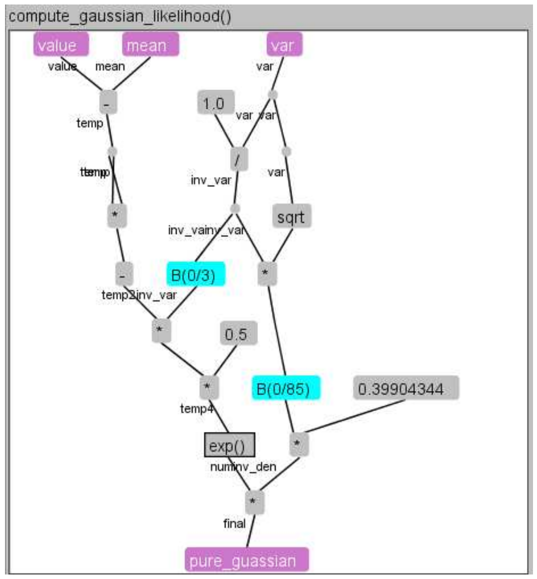


Figure 3.5: Gaussian PE from Mitrion simulator/debugger.

tissue type and the value of w, as shown in Equations 2.8 and 2.9, respectively. The integration step size, w, is set to 0.02 as in the CPU-based implementation. The implemented design generates the single precision floating-point PDs of two mixed tissue classes and writes them into the external SRAM in the order shown in Figure 3.6.

We implemented the integral operation over fifty different gaussian PD functions using a reduction tree design technique [29] [30] instead of a sequential loop. As we were only able to implement two gaussian PEs on one algorithm FPGA due to the limit number of available logic blocks, we designed and implemented one-level single precision floating-point reduction tree. A single precision floating-point accumulator was then used to accumulate the outputs generated by the reduction tree. Moreover, since the algorithm requires intermediate results $(1-w)$, w^2 and $(1-w)^2$ for fifty different w values between zero and one to calculate the mean and covariance for mixed tissues, we pre-calculated the values and stored in tables to reduce computational complexity. We also used a combination of the Mitrion-C list and vector data types with *foreach* loop to estimate the PDs of the two mixed tissue classes in a pipelined fashion.

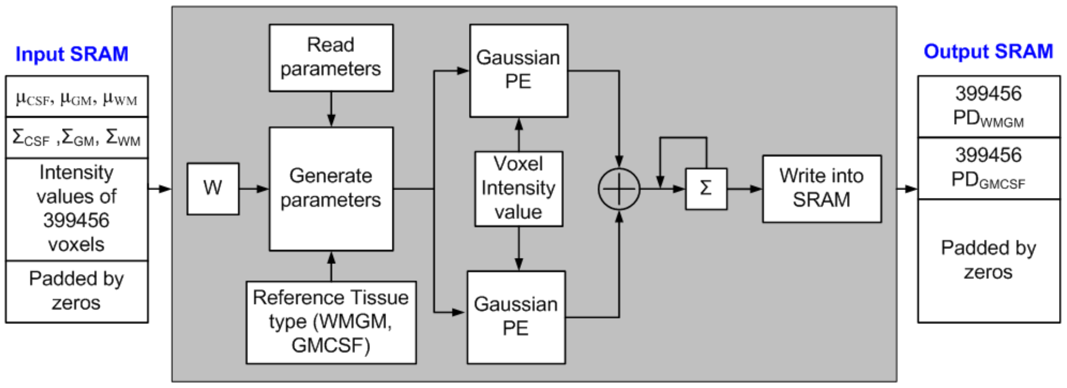


Figure 3.6: Functional blocks of FPGA-implemented PDs estimation for mixed tissue types.

3.4.4 FPGA-based PDs Normalization

We implemented a function on the FPGA-accelerators to normalize the five single precision floating-point PDs, generated as described in Sections 3.4.2 and 3.4.3. Moreover, we also designed the implementation to evaluate the voxel value of mask skull volume. The FPGA-implemented normalization function first reads the PDs of all tissue types and the voxel value of the mask skull from the external SRAM. If the voxel is in the background, the implemented design sets the normalized PDs of every tissue class to zero and labels the voxel as background. Otherwise, the five PDs of a voxel are normalized by a normalization PE. After the PDs are normalized, the implemented design determines the tissue type that generates the maximum PD among them and classifies the voxel according to the tissue type. The converted normalized PDs and classified tissue labels are then written into the external SRAM in the order as shown in Figure 3.7.

Figure 3.8 provides the detailed functional building blocks of the normalization PE are provided. The sum of the five PDs was computed from the three-level reduction tree. Each PD was multiplied by the inverse of the calculated sum rather than being divided by the sum, as single precision floating multiplication operation is computationally and logically less expensive than the division operation. The normalized single precision floating point PDs were multiplied by 65535 (2^{16}-1) and converted to unsigned short data type using the Mitrion-C type casting fea-

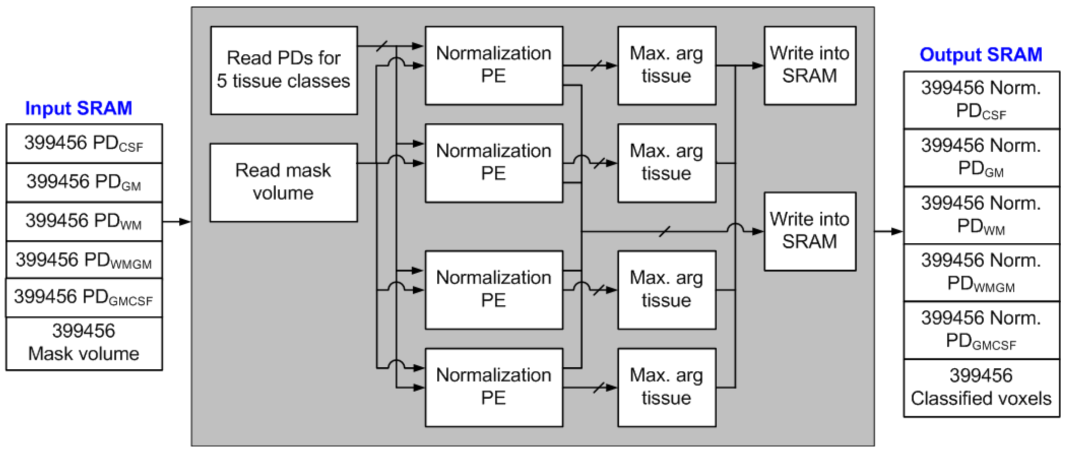


Figure 3.7: Functional blocks of FPGA-implemented normalization function for PDs.

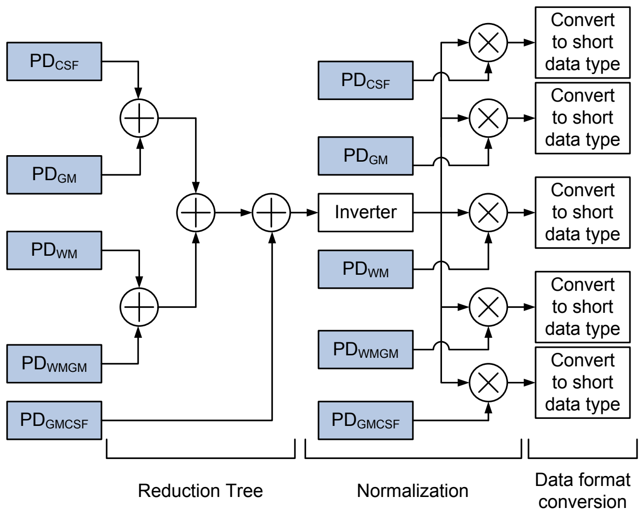


Figure 3.8: Normalization PE.

ture. Since normalization of the PDs is an independent process for every voxel, we employed four normalization PEs to normalize the tissue PDs for four voxels simultaneously as shown in Figure 3.7.

3.4.5 Host Program for the FPGA-based PD Estimation

After bitstreams were successfully generated for the implementations described in Sections 3.4.2, 3.4.3 and 3.4.4, we developed a host program to allocate memory for input/output buffers and to handle the data transfer between the Altix 350 system and the RASC RC100 algorithm FPGAs. This host program was also responsible for controlling the sequence of operations that are required to run the implemented algorithm on the FPGA-accelerators.

The host program uses a special RASC API function, $rasclib\_huge\_alloc$, to allocate the memory space that is required to store the input and output buffers. This function allows developers to allocate memory from the specially pre-reserved spaces by RASC library, such that a DMA can transfer the data faster between the global shared memory in the Altix 350 and the external memory of the FPGA. After all of the spaces have been allocated, the host program writes the data required to perform the algorithms into the allocated input buffer using the $memcpy$ function provided in C.

As shown in Figure 3.2, we designed PD estimation to perform simultaneously for pure and mixed tissues. As a result, the host program employs the POSIX threads (Pthreads) API [31] [32] to invoke the two FPGA-implemented algorithms at the same time. It first calls the $rasclib\_resource\_alloc$ function to allocate the available FPGAs for both algorithms. After the available devices are allocated, the host program calls the $pthread\_create()$ function to create two individual threads and to initialize the attributes used by the threads. Each thread sequentially calls the RASC API functions, described in Section 2.4.2, to transfer data to and from the device and to start the algorithm implemented on each FPGA. When a thread has completed its work, the host program calls the $pthread\_exit()$ function and receives the termination statuses of the threads. For last, the host program calls the $pthread\_join()$ function to ensure synchronization between the two threads. Figure 3.9 shows a summary of Pthreads operation.

The current Altix 350 system contains two RASC RC100 modules such that four FPGA-accelerators are available to execute multiple algorithms in parallel. Therefore, properly partitioning the implemented algorithms over four available FPGAs is essential to achieving optimal performance. Since every PD needs to be estimated before normalization, two PD estimation algorithms for pure and mixed tissue classes are partitioned over the four FPGAs using the RASC wide-scaling feature described in Section 2.4.1. Table 3.3 shows several different partitioning

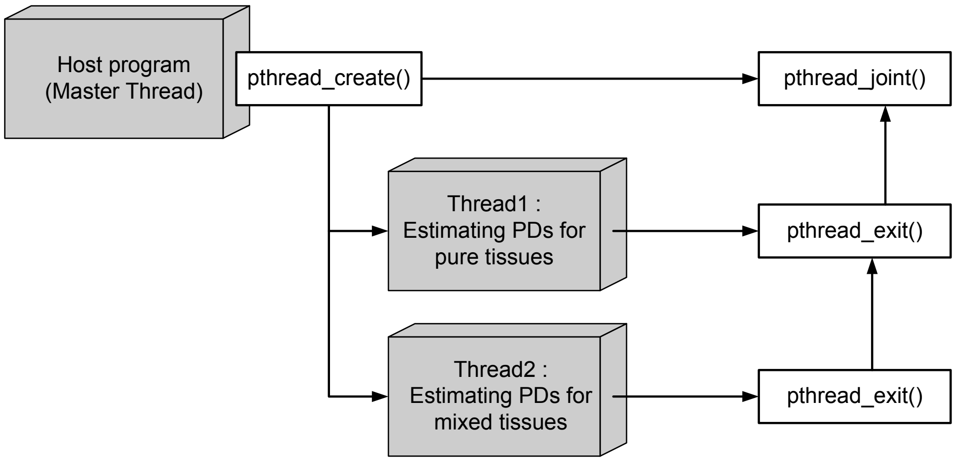


Figure 3.9: Summary of Pthreads operation.

Table 3.3: Possible scenarios to distribute over four FPGAs for PD estimation function for pure and mixed tissues classes

# of FPGAs used to estimate the PD for pure tissues	# of FPGAs used to estimate the PD for mixed tissues	Time$_{pure}$	Time$_{mixed}$	Total time
1	3	0.359s	13.54s	17.35s
2	2	0.188s	19.32s	23.12s
3	1	0.161s	38.62s	42.4s

scenarios over four FPGAs and indicates that estimating PDs of the mixed tissue classes dominates most of the computation time. As a result, the best performance improvement is achieved by using three FPGAs and one FPGA to estimate PDs of mixed tissue and pure classes respectively.

After the FPGA-implemented algorithm estimates PDs of all tissue classes for voxels, the host program reconstructs the input buffer, as shown in Figure 3.7. At the same time, the four FPGAs allocated for the PD estimation algorithms are unreserved and re-allocated for the normalization algorithm. The host program also partitions and invokes the normalization function over four FPGAs using wide scaling feature. Figure 3.10 illustrates the final design of the hardware-implemented PD estimation and normalization algorithm.

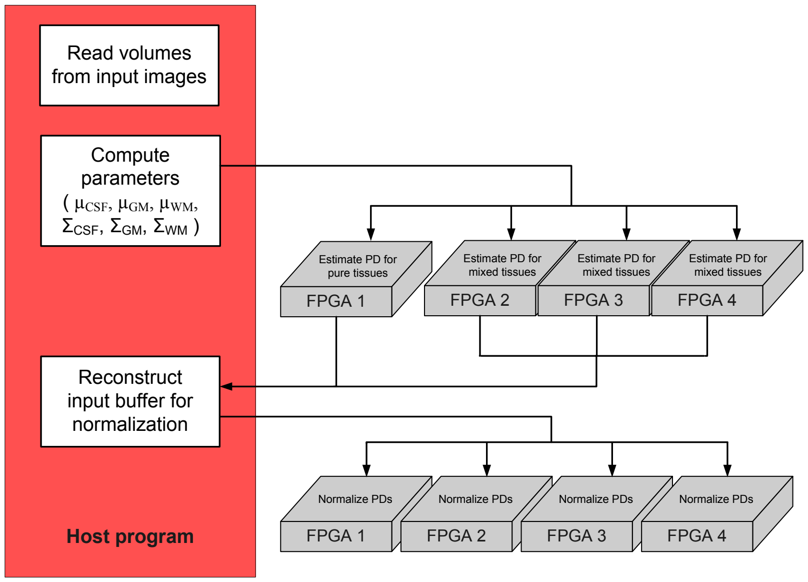

Read volumes from input images
Compute parameters (μ_{CSF}, μ_{GM}, μ_{WM}, Σ_{CSF}, Σ_{GM}, Σ_{WM})
Estimate PD for pure tissues
FPGA 1
Estimate PD for mixed tissues
FPGA 2
Estimate PD for mixed tissues
FPGA 3
Estimate PD for mixed tissues
FPGA 4
Reconstruct input buffer for normalization
Normalize PDs
FPGA 1
Normalize PDs
FPGA 2
Normalize PDs
FPGA 3
Normalize PDs
FPGA 4
Host program

Figure 3.10: Hardware-implemented PD estimation and normalization algorithm with four FPGAs.

3.5 FPGA-implemented Prior Information Estimation

The CPU-based prior information estimation function is designed to estimate the prior information, P(C), terms, for every possible tissue class using the MRF model provided in Equation 2.10. Although the PVE algorithm performs the PD estimation function only once during the PV classification stage, it invokes the prior information estimation function iteratively. The algorithm re-classifies the tissue classes of voxels using the new estimated P(C) terms in each iteration. The current iteration terminates when the total number of changed voxels during the previous iteration is lower than the total number of changed voxels during the current iteration or the maximum number of iterations (20) has been reached. Algorithm 2 provides an overview of the CPU-based prior information estimation for each tissue class of every voxel in the volume.

The CPU-based prior information estimation function first reads a tissue class of a $voxel_i$ from the classified volume, generated from the PD estimation stage (i $\in \{0$ . . . total number of voxel -1$\}$). If the voxel is labeled as background, the algorithm skips every step. Otherwise, the algorithm estimates the prior terms to determine the appropriate tissue class for the voxel. The algorithm reads the 26-neighborhood

Algorithm 2 Estimation of the prior information for all possible tissue classes

```
Require: Run until the criteria in Section 2.3.3 is satisfied.
  INPUT : Curvature image, classified image and five PDs volumes
  for $i = 0$ to $last\_voxel - 1$ do
    $current\_tissue \Leftarrow$ tissue type of $voxel_i$ from the classified image
    if $current\_tissue$ = background then
      $new\_tissue\_class_i \Leftarrow$ background
    else
      $curve\_var \Leftarrow$ value of $voxel_i$ in the curvature volume
      for $ref\_tissue$ = CSF ... GMCSF do
        sum $\Leftarrow 0$
        for $k = 0$ to 25 do
          if $voxel_i$ is located at the border of volume then
            $tissue_k \Leftarrow 0$
          else
            $tissue_k \Leftarrow$ tissue class of one of $voxel_i$'s 26 neighborhood voxels
          end if
          if $tissue_k$ and $ref\_tissue$ are same then
            $a_ik \Leftarrow$ -2
          else if $tissue_k$ and $ref\_tissue$ are similar then
            if $curve\_var < 0$ then
              $a_ik \Leftarrow \frac{-1}{1+e^{-25(curve\_var-1)}} - 1$
            else
              $a_ik \Leftarrow -1$
            end if
          else
            $a_ik \Leftarrow 1$
          end if
          compute $\frac{a_ik}{d(i,k)}$
          sum $\Leftarrow$ sum + $\frac{a_ik}{d(i,k)}$
        end for
        P(C) $\Leftarrow e^sum$
      end for
      final[CSF] $\Leftarrow pdf_i[CSF] \times$ CSF P(C)
      final[GM] $\Leftarrow pdf_i[GM] \times$ GM P(C)
      final[WM] $\Leftarrow pdf_i[WM] \times$ WM P(C)
      final[WMGM] $\Leftarrow pdf_i[WMGM] \times$ WMGM P(C)
      final[GMCSF] $\Leftarrow pdf_i[GMCSF] \times$ GMCSF P(C)
      $new\_tissue\_class_i \Leftarrow \underset{tissue}{\text{argmin}}\{final[tissue]\}$, tissue = CSF, ..., GMCSF
    end if
  end for
  OUTPUT : Classified image
```

voxels around $voxel_i$ from the classified volume to calculate P(C) for the reference tissue type using the MRF model as provided in Equation 2.10. If $voxel_i$ is located at the boundary of the volume, some of its 26-neighborhood voxels might be located outside the volume. Therefore, the algorithm manually sets the neighborhood voxels located outside the volume to the background instead of reading from the classified volume. The algorithm compares the 26-neighborhood voxels with the reference tissue classes (CSF, GM, WM, WMGM and GMCSF) to compute the similarity terms, a_{ik}, as shown in Equation 2.12. The 26 calculated a_{ik} terms are divided by the corresponding d(i,k), which represents the distances between $voxel_i$ and neighborhood $voxel_k$. A sequential loop then sums up the 26 calculated $\frac{a_{ik}}{d(i,k)}$ terms to compute the prior information. This process is repeated for every other reference tissue type. After the algorithm generates the prior information for every tissue class, it is multiplied by the corresponding PDs generated from the implementations describe in Section 3.4 and the final products are normalized. As a final step, the algorithm classifies the tissue class of the $voxel_i$ as the type that generates the maximum term among the final five products of the prior information and PDs. The algorithm increase a variable, $changed\_num$, if the tissue class of $voxel_i$ is changed. After the described steps are performed on every voxel in the volume, the algorithm compares the final value of $changed\_num$ to the value from the previous iteration to verify the termination criteria.

We first evaluated the functions in the CPU-based prior information estimation implementation to identify any data-parallelism that could be exploited on the FPGA-accelerators. Since the 26 a_{ik} terms required to compute a P(C) term are independent, we implemented these to perform simultaneously. Moreover, we designed the FPGA-implemented algorithm in a pipelined fashion because the prior information for different tissue types is independent. The following sections present the detailed design principles and techniques used in implementing prior information estimation on FPGA-accelerators.

Table 3.4: Summary of input/output file formats for the prior information estimation function.

Volume	Data type	Total Size
Classified image (Input)	Unsigned char (1B)	6.78 MB
Curvature image (Input)	floating point (4B)	27.12 MB
five PDs (Input)	Unsigned short (2B)	5 $\times$ 13.56 MB
classified image (Output)	Unsigned chart (1B)	6.78 MB

3.5.1 Input and Output Buffer

To run the FPGA-based prior information estimation algorithm, three images are required: the classified image, the curvature image and an image that contains the probability densities of voxels for different tissue classes. These images are composed of 181 slices, where each slice contains 39277 voxels (217 $\times$ 181). The classified image generated from the PD estimation function or the previous iteration of the prior information estimation function stores the tissue classes of voxels using unsigned char data type (1B). The value of the curvature image is stored using a single precision floating point data type (4B). An unsigned short data type (2B) is used to store each of the voxel values in the five PD volumes (PD$_{CSF}$, PD$_{GM}$, PD$_{WM}$, PD$_{WMGM}$ and PD$_{CSFWM}$). The prior information estimation algorithm also generates one classified image which uses unsigned char data type to store the re-classified tissue classes of voxels. Table 3.4 provides a summary of the input and output files for the prior estimation function.

Due to the limited size of the external SRAMs on the RASC RC100 module, we cannot sent all three input images to the FPGA at once. As a result, we utilized the RASC streaming feature to reduce the overhead of the data transfer between the memory on the Altix 350 and the off-chip memory. We set the available input and output off-chip memories to 8MB and manually calculated the total number of slice can be stored in 8MB SRAM as:

$$\begin{aligned}
8\text{MB} &= ((\text{x}+2) \times 399456 \text{ voxels} \times 1\text{B}) \\
&\quad + (\text{x} \times 399456 \text{ voxels} \times 4\text{B}) \\
&\quad + 5 \times (\text{x} \times 399456 \text{ voxels} \times 2\text{B}) \\
\therefore \text{x} &= 13 \text{ slices per 8MB}.
\end{aligned} \tag{3.3}$$

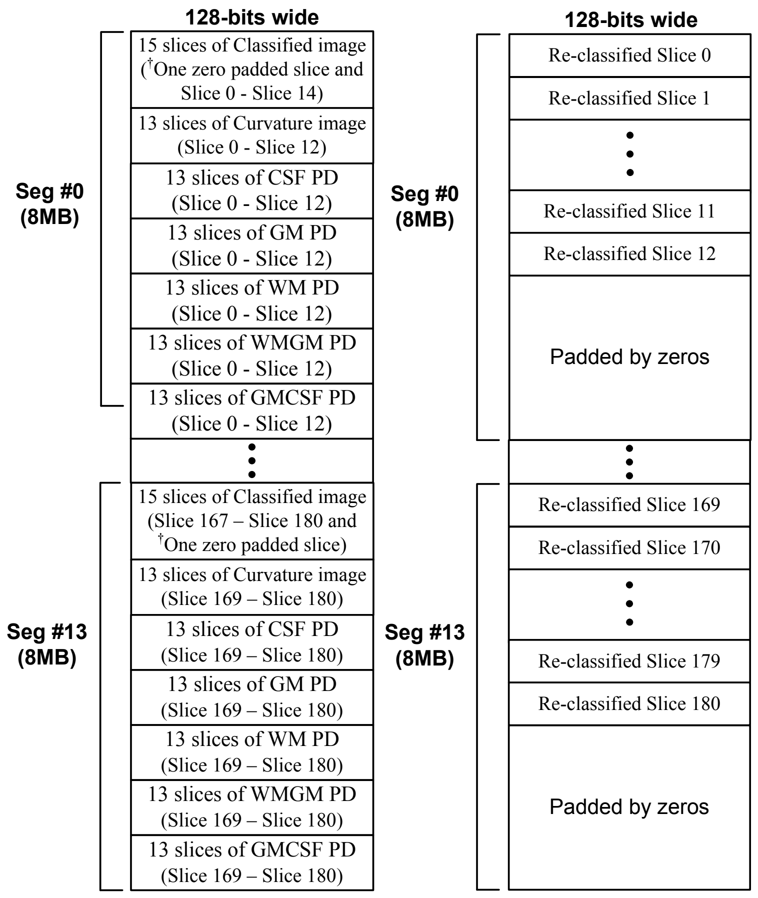
We transferred extra two slices of the classified image (front and back) in each segment in order to obtain 26-neighborhood voxels.

Moreover, we transferred fourteen 8MB segments from the Altix 350 system to the external memory of the RASC RC100 module using the streaming feature, as each volume contains 181 slices. Figures 3.11(a) and 3.11(b) show the structure of input and output buffer used for FPGA-implemented prior information estimation function, respectively.

3.5.2 FPGA-based MRF Function

We re-programmed the C-kernel of the CPU-based $P(C)$ estimation function using the Mitrion-C HLL to generate the bitstreams for the FPGA-accelerators. As shown in Figure 3.12, the hardware-implemented algorithm first reads three rows of each slice (slice α-1, α and α+1) of the classified image from the input SRAM and stores them in the internal Block RAMs on the FPGA. The CPU-based algorithm stores the voxel values of the classified image in an array of unsigned *char* (8-bits) data type. However, in order to reduce hardware resources and achieve better performance, we stored the rows read from the classified volume in 3-bit internal Block RAMs in the FPGA-based implementation, as all possible tissue classes can be represented as 3-bit data. The implemented design also reads a single row of the curvature image and the five probability densities volumes for all possible tissue types and stores in the internal Block RAMs.

The implementation then reads the 26-neighborhood voxels around the current $voxel_i$ from the internal Block RAMs, where three rows of each slice are stored as



128-bits wide
15 slices of Classified image (†One zero padded slice and Slice 0 - Slice 14)
13 slices of Curvature image (Slice 0 - Slice 12)
13 slices of CSF PD (Slice 0 - Slice 12)
13 slices of GM PD (Slice 0 - Slice 12)
13 slices of WM PD (Slice 0 - Slice 12)
13 slices of WMGM PD (Slice 0 - Slice 12)
13 slices of GMCSF PD (Slice 0 - Slice 12)
Seg #0 (8MB)
15 slices of Classified image (Slice 167 – Slice 180 and †One zero padded slice)
13 slices of Curvature image (Slice 169 – Slice 180)
13 slices of CSF PD (Slice 169 – Slice 180)
13 slices of GM PD (Slice 169 – Slice 180)
13 slices of WM PD (Slice 169 – Slice 180)
13 slices of WMGM PD (Slice 169 – Slice 180)
13 slices of GMCSF PD (Slice 169 – Slice 180)
Seg #13 (8MB)
128-bits wide
Re-classified Slice 0
Re-classified Slice 1
Re-classified Slice 11
Re-classified Slice 12
Padded by zeros
Seg #0 (8MB)
Re-classified Slice 169
Re-classified Slice 170
Re-classified Slice 179
Re-classified Slice 180
Padded by zeros
Seg #13 (8MB)

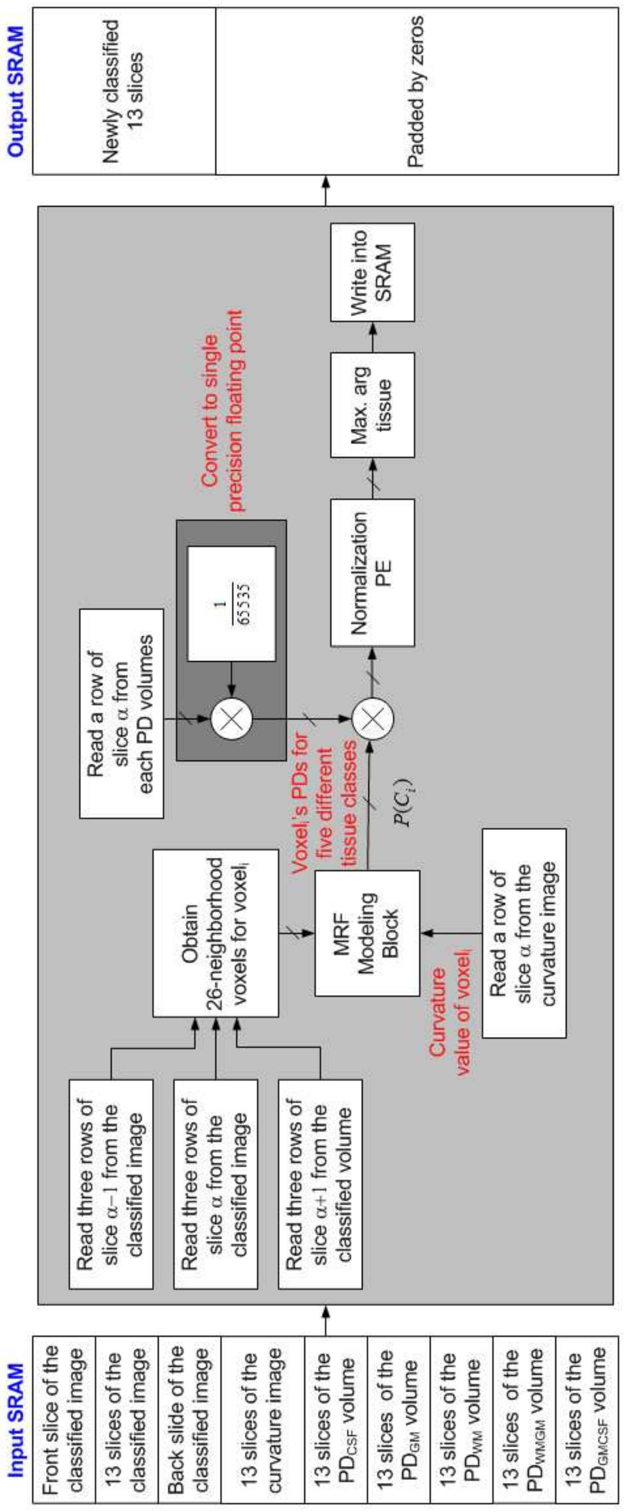
(a) Input buffer. (b) Output buffer.

† Zero padded slices are introduced in Section 3.5.3

Figure 3.11: The structure of input and output buffer for FPGA-implemented prior information estimation.

shown in Figure 3.13. For an example, if the current $voxel_i$ is stored at the address 183 of a internal Block RAM, its eight of 26-neighborhood voxels are stored at the addresses 2, 3, 4, 182, 184, 362, 363 and 364. The remaining neighborhood voxels can be simply obtained from the other internal Block RAMs, which store the three rows of slices α-1 and α+1.

After the implementation reads the 26-neighborhood voxels, it generates the prior information for all possible tissue classes using the MRF modeling block. Figure 3.14 provides an overview of the MRF modeling block. Since the algorithm requires 26 $\frac{a_{ik}}{d(i,k)}$ terms to compute P(C) as shown Equation 2.10, we designed and implemented a PE which generates a single $\frac{a_{ik}}{d(i,k)}$ term on the FPGA-accelerator using the Mitrion-C HLL as shown in Figure 3.15. Each a_{ik} PE first verifies the relationship between the reference tissue class and one of its 26-neighborhood voxels, and then determines the appropriate a_{ik} term. To reduce the complexity of the verification process, we implemented a truth table, shown in Table 3.5, to validate the similarity between two tissue classes. The weighting term is multiplied by the inverse of the distance between $voxel_i$ and its neighborhood voxel $voxel_k$, instead of dividing, to avoid costly floating-point division operations. Since the inverted distance between two voxels are fixed, we pre-calculated and stored these values in a table to reduce the computational complexity.



Input SRAM
Front slice of the classified image
13 slices of the classified image
Back slide of the classified image
13 slices of the curvature image
13 slices of the PD$_{CSF}$ volume
13 slices of the PD$_{GM}$ volume
13 slices of the PD$_{WM}$ volume
13 slices of the PD$_{WMGM}$ volume
13 slices of the PD$_{GMCSF}$ volume
Read three rows of slice $\alpha-1$ from the classified image
Read three rows of slice α from the classified image
Read three rows of slice $\alpha+1$ from the classified volume
Obtain 26-neighborhood voxels for voxel$_i$
MRF Modeling Block
Curvature value of voxel$_i$
Read a row of slice α from the curvature image
Voxel$_i$'s PDs for five different tissue classes
$P(C_i)$
Read a row of slice α from each PD volumes
$\frac{1}{65535}$
Convert to single precision floating point
Normalization PE
Max. arg tissue
Write into SRAM
Output SRAM
Newly classified 13 slices
Padded by zeros

Figure 3.12: Hardware-implemented prior information estimation function.

V_1	V_2	V_3	V_4	. . .	V_{177}	V_{178}	V_{179}	V_{180}
V_{181}	V_{182}	V_{183}	V_{184}	. . .	V_{356}	V_{357}	V_{359}	V_{360}
V_{361}	V_{362}	V_{363}	V_{364}	. . .	V_{537}	V_{538}	V_{539}	V_{540}

Figure 3.13: An internal Block RAM stores three rows of Slice α.

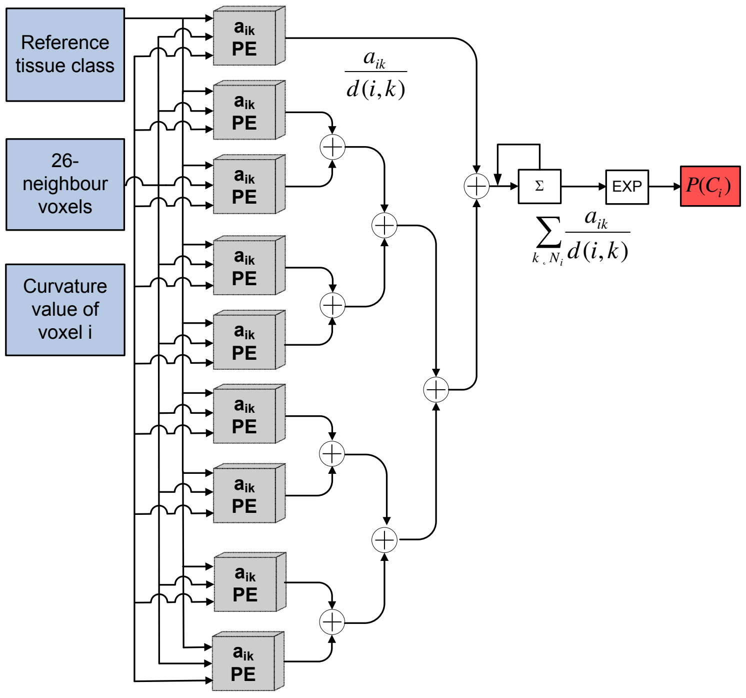


Figure 3.14: Functional blocks of MRF modeling block.

Since the 26 $\frac{a_{ik}}{d(i,k)}$ terms are completely independent, we implemented several a_{ik} PEs in parallel to generated these terms simultaneously. Due to the limited availability of hardware resources, we were only able to utilize nine a_{ik} PEs in the proposed implementation. We also exploited a four-level single precision reduction tree to accumulate the output of each PE instead of a data-dependent for-loop and implemented a sequential loop-dependent accumulator at the end of the reduction tree. The final $P(C)$ was then generated using a single precision floating-point exponential operation, as shown in Figure 3.14.

The implementation then reads the five PDs of $voxel_i$ from the internal Block RAMs and they are multiplied by $\frac{1}{65535}$ to convert them to single precision floating-

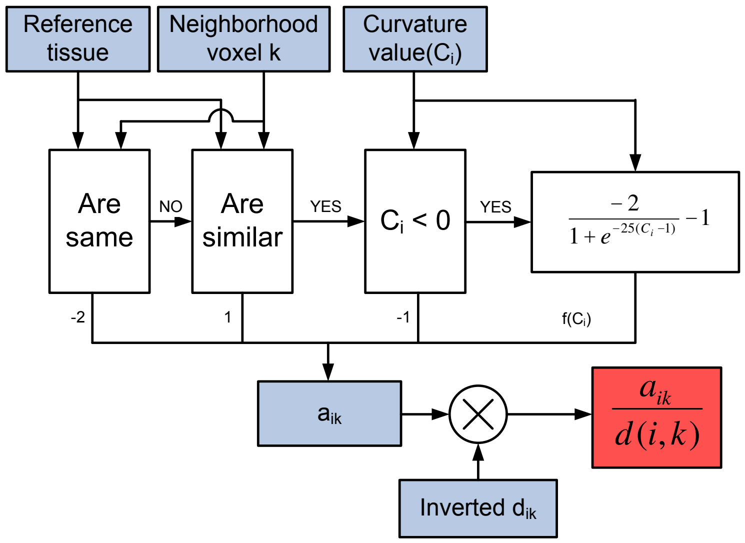


Figure 3.15: Generating a_{ik}.

Table 3.5: Truth table used to identify the similarity between the reference tissue and the tissue class of neighbor voxel

tissue	CSF	GM	WM	WMGM	GMCSF
CSF	0	0	0	0	1
GM	0	0	0	1	1
WM	0	0	0	1	0
WMGM	0	1	1	0	0
GMCSF	1	1	0	0	0

point values using the Mitrion-C type casting function. The generated $P(C)$s are multiplied by the corresponding single precision tissue probability densities in a pipelined fashion, and the final products are normalized using the normalization PE described in Section 3.4.4. As a final step, the implementation determines the tissue type which generates the maximum product of $P(C)$ and PD and classifies the tissue class of the $voxel_i$ as the determined tissue type. The newly classified tissue type is then written to the output RAM.

3.5.3 Host Program for FPGA-based Prior Information Estimation

We developed the host program for the FPGA-based prior information estimation function in C and its responsibilities were very similar to those described in Section 3.4.5. The host program first allocates the memory space for input and output buffers using the RASC $rasclib\_huge\_alloc$ function. The FPGA-based implementation requires a special scheme to handle the 26-neighborhood voxels of a voxel located on the border of the volume. To reduce computational complexity, the host program pads the volume with zeros such that the neighborhood voxels located outside of the volume are automatically set to zero, which represents the background. The $memcpy$ function provided in C writes the required input data into the allocated memory space as shown in Figure 3.11(a).

Since the prior information for each voxel can be computed independently, the algorithm execution is distributed among the four available FPGA-accelerators to exploit coarse-grained parallelism. We modified and partitioned the implementation described in Section 3.5.2 into four different implementations such that each partition can compute one quarter of the voxels in each row of the slice that is being processed as shown in Figure 3.16. Unfortunately, we could not use the RASC wide-scaling feature in this algorithm because the generated bitstreams from the four partitions are not identical. As a result, we generated four threads which manually uploaded each bitstream onto FPGAs and invoked the implemented algorithms simultaneously.

When the prior information estimation algorithm is performed successfully, the CPU-based PVE algorithm evaluates whether the iteration termination criteria is satisfied or not. The evaluation process requires to compare the classified image from the current and previous iteration and then it determines how many voxels have been classified to a new tissue class in the current iteration with respect to the previous iteration. As a result, this function is not a good candidate to be implemented on FPGA-accelerators because there is a data-dependency between voxels

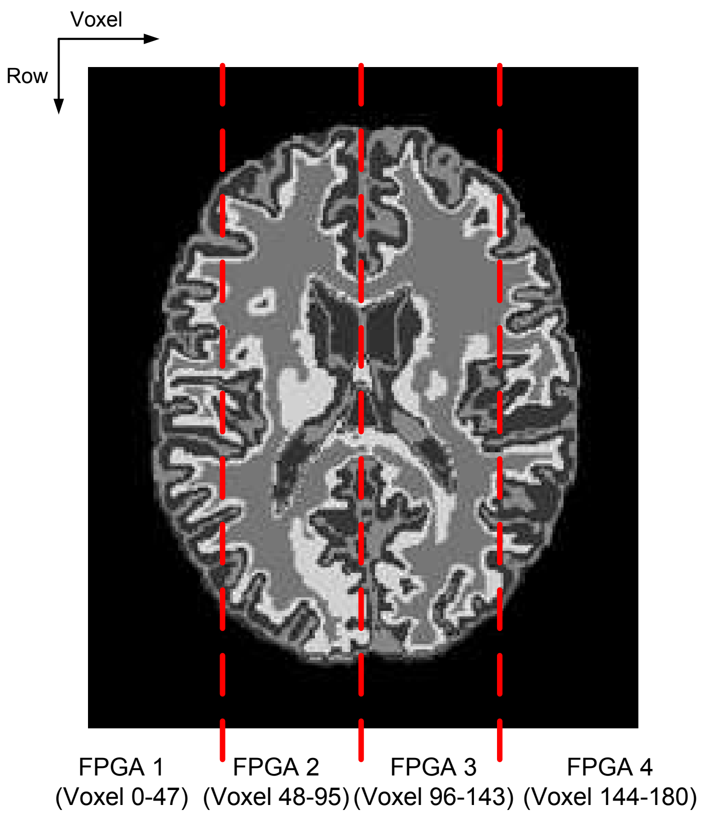


Figure 3.16: Partitioned FPGA-based prior information function.

from two classified images. Therefore, we implemented the evaluation function in the host program instead of on the FPGAs using a counter and a sequential for-loop provided in C. Since the evaluation function in the host program performs extremely fast, the computation time spent by the evaluation function is neglected.

3.6 Summary

We profiled the CPU-based PVE algorithm using GNU gprof to determine the most computationally intensive functions in the algorithm. The functions were then implemented on SGI RASC RC100 FPGA-accelerators using the Mitrion-C HLL to reduce the computation time. This chapter describes the design concepts used to implement the primary performance bottleneck of PVE algorithm, the PV classification stage, on the RASC RC100. Figure 3.17 illustrates the completed implementation. We developed the host program for the FPGA-based PVE algorithm using

SGI RASC library 2.1, while the hardware-implemented functions are programmed using Mitrion-C 1.2. We then used the third party tool, Xilinx ISE 8.2i, to generate bitstreams by performing synthesize and place-and-route operations on the implemented designs. Table 3.6 shows the hardware resources used and the time spent to perform the synthesis and place-and-route on each implemented function.

Table 3.6: Hardware resources summary of functions implemented on a Xilinx Virtex-4 LX200.

Resource (total)	Pure tissue PD estimation function	Mixed tissue PD estimation function	Normalization function	P(C) estimation function
Slices (89,088)	38,957 (43%)	79,232 (88%)	56,784 (63%)	77,954 (87%)
Flip Flops (178,176)	46,362 (26%)	86,739 (48%)	62,350 (34%)	82,159 (46%)
4-inputs LUTs (178,176)	42,886 (24%)	96,489 (54%)	65,579 (36%)	96,479 (54%)
Multipliers (96 18×18)	64 (66%)	64 (66%)	64 (66%)	65 (67%)
18kb Block RAMs (336)	39 (11%)	57 (16%)	25 (7%)	37 (11%)
†CAD Time	$\approx$ 6 hour	$\approx$ 6 hour	$\approx$ 6 hour	$\approx$ 9.5 hours

† Synthesis and place-and-route operations are done by a server with 2.0 GHz Intel Xeon processor and 4GB memory.

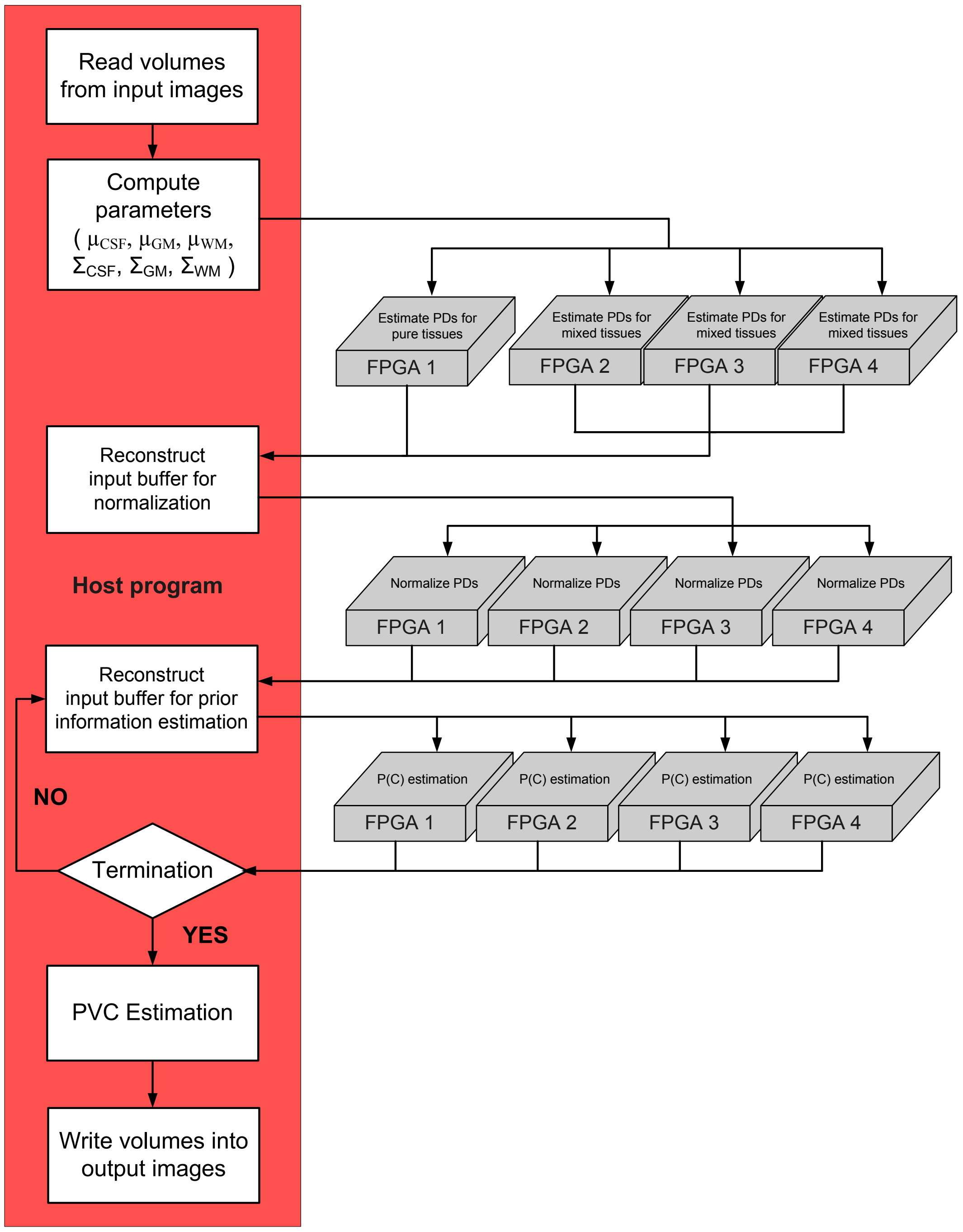

Read volumes from input images
Compute parameters (μ_{CSF}, μ_{GM}, μ_{WM}, Σ_{CSF}, Σ_{GM}, Σ_{WM})
Estimate PDs for pure tissues
FPGA 1
Estimate PDs for mixed tissues
FPGA 2
Estimate PDs for mixed tissues
FPGA 3
Estimate PDs for mixed tissues
FPGA 4
Reconstruct input buffer for normalization
Host program
Normalize PDs
FPGA 1
Normalize PDs
FPGA 2
Normalize PDs
FPGA 3
Normalize PDs
FPGA 4
Reconstruct input buffer for prior information estimation
P(C) estimation
FPGA 1
P(C) estimation
FPGA 2
P(C) estimation
FPGA 3
P(C) estimation
FPGA 4
NO
Termination
YES
PVC Estimation
Write volumes into output images

Figure 3.17: Final design of HPRC-implemented PVE algorithm

Chapter 4

Results

In order to evaluate the performance improvement and accuracy of the proposed FPGA-based PVE algorithm, we performed several tests with both simulated and real MR brain images. We first used the simulated images to evaluate the performance improvement and accuracy of the implementation, as they are fairly rigorous representation of a true MR brain image. Moreover, several real MR brain images were randomly selected from the International Consortium for Brain Mapping (ICBM) data set [33] and processed using the FPGA-based PVE algorithm to ensure the performance improvement and the robustness of the implementation. In order to evaluate the performance improvement, a timing function in the host program measured the computation times of both CPU-based and FPGA-based PVE algorithm and the results were compared. We also used several statistical similarity measurements to verify images that resulted from both the CPU-based and FPGA-based PVE algorithms. The following sections provide detailed descriptions of the test data sets and the measurements that we used to evaluate the performance improvement and accuracy of the FPGA-based PVE algorithm.

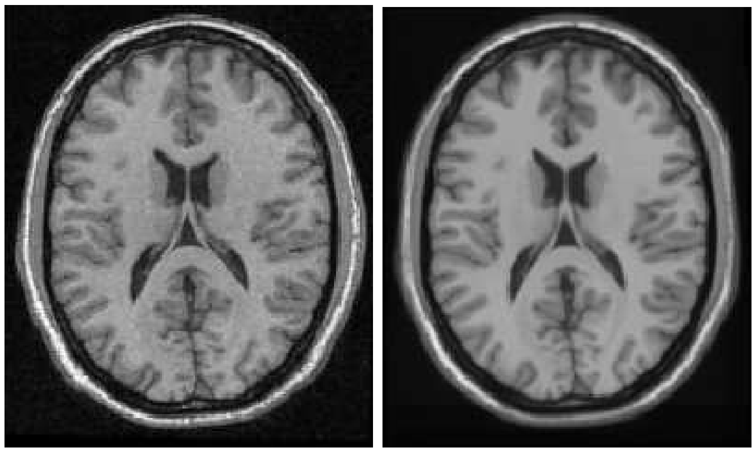

(a) Single T1 image. (b) Averaged T1 image (colin27).

Figure 4.1: Single and averaged T1-weighted colin image.

4.1 Test Data Set

4.1.1 The Digital Phantom

The digital phantom image, known as “colin27”, was generated by averaging the intensity of the 27 distinct T1-weighted MRI scans that are acquired from a single subject [34]. The signal-to-noise ratio of the intensity-averaged image was enhanced, and the contrast between tissues was also significantly improved, as shown in Figure 4.1. This image has been used during the development and performance evaluation of the FPGA-based PVE algorithm, as it minimizes classification errors due to the noise and poor tissue contrast found in typical MR brain images.

4.1.2 The Simulated Images

Noise and intensity non-uniformity (INU), two major acquisition artifacts found in MR images, are a significant challenge for automated brain MR image analysis algorithms because they can degrade the performance of these algorithms. In the past, it has been very difficult for developers to determine the behavior of the algorithm on images with artifacts, because no information is known about the artifacts before the image is processed. As a result, researchers have developed a sophisticated MRI simulator [35] to obtain realistic brain MR images with a user-specified quantity of artifacts. The simulator uses the Bloch equation to generate 3D simulated MR images from pre-defined tissue templates and a 3D brain phantom, by varying the specific imaging parameters and artifacts.

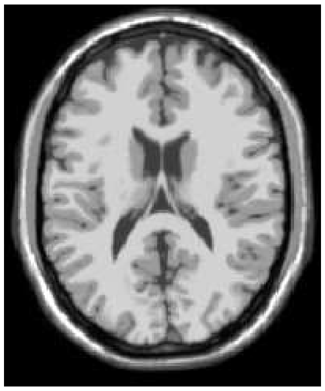
The simulated images have been very useful for many anatomical brain mapping communities because they accurately represent the major morphological features of the human brain. As a result, many 3D MR brain image analysis algorithm developers perform their algorithms on the simulated images and use them as the "gold standard" for quantitative analysis of their techniques [9] [23]. Moreover, as the MRI simulator can easily introduce different amounts of artifacts on the images, algorithm developers have used the generated images to validate the performance of their algorithms over different quantities of artifacts.

Images from the flexible and convenient MRI simulator are available to the public through an online interface. The McConnell Brain Image Center (BIC) has developed the *BrainWeb* [36] for the neuroimaging community so that researchers can have an access to images from the pre-computed simulated brain database (SBD) [1]. Figures 4.2 and 4.3 provide the simulated images downloaded from the BrainWeb with different noise and INU parameters respectively. The denotation $pn3rf20$ is used to represent an image with 3% of noise and 20% of INU.

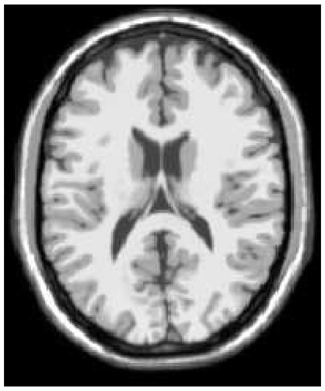
4.1.3 The Real Human Brain

For last, we used several real brain images from the ICBM data set, in order to validate the performance and the robustness of the FPGA-based PVE algorithm. The ICBM has been collecting human brain images since 1992, to develop a probability atlas and reference system for the human brain. The first ICBM data set contains 88 male subjects and 66 female subjects with healthy, normal brains. The age range of subjects is from 18 to 22 years. Figure 4.4 provides T1-weighted images selected from the ICBM data set.

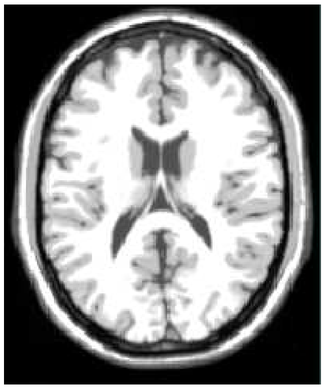
[1] http://www.bic.mni.mcgill.ca/brainweb/



(a) pn0rf0 : 0% noise and 0% RF noise.

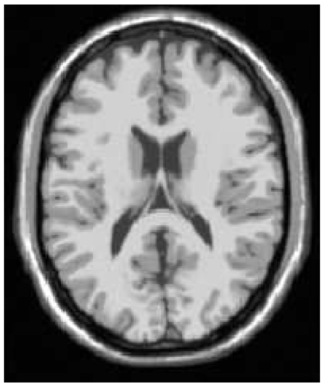


(b) pn0rf20 : 0% noise and 20% RF noise.

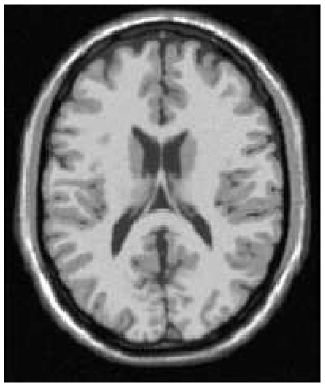


(c) pn0rf40 : 0% noise and 40% RF noise.

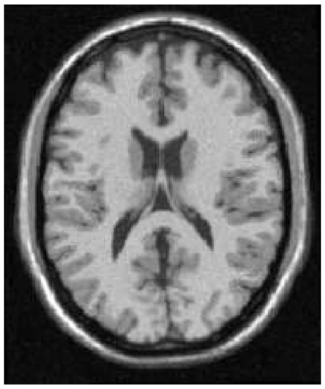
Figure 4.2: The simulated images with INU (typically 20% of INU is present on an image).



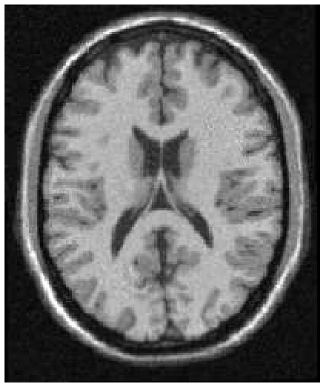
(a) pn1rf0 : 1% noise and 0% RF noise.



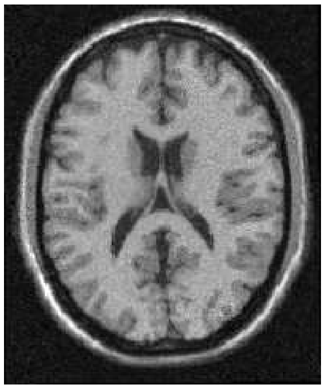
(b) pn1rf0 : 3% noise and 0% RF noise.



(c) pn1rf0 : 5% noise and 0% RF noise.

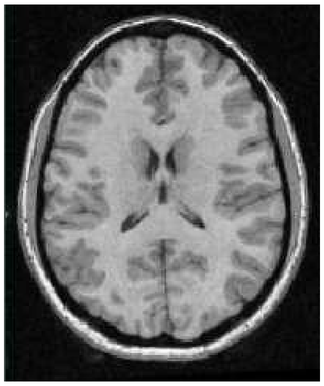


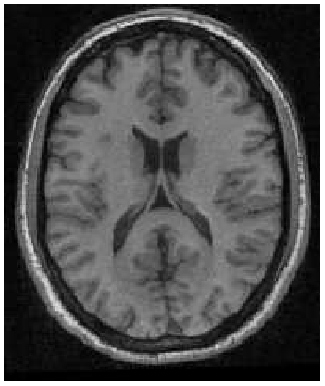
(d) pn1rf0 : 7% noise and 0% RF noise.

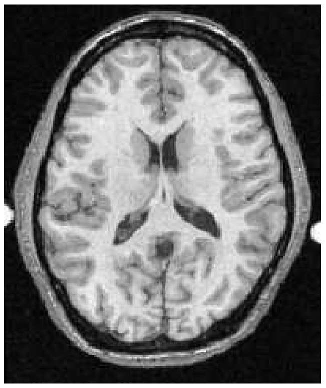


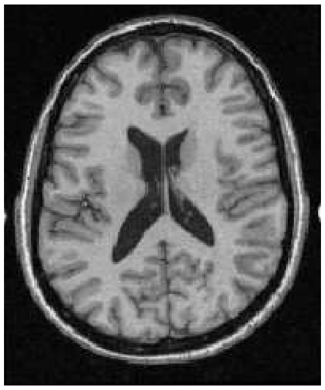
(e) pn1rf0 : 9% noise and 0% RF noise.

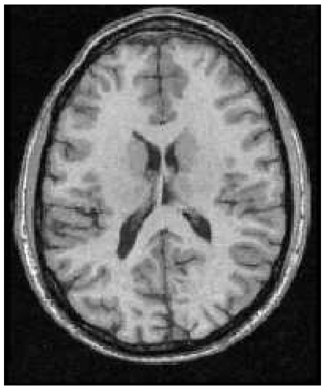
Figure 4.3: The simulated images with noise (typically 5% of noise is present on an image).

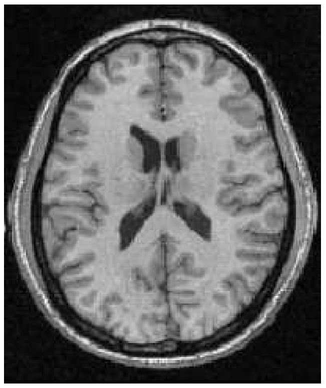

(a) ICBM101.

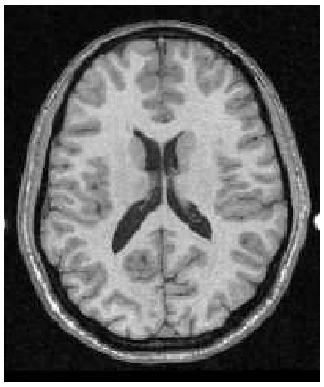

(b) ICBM102.

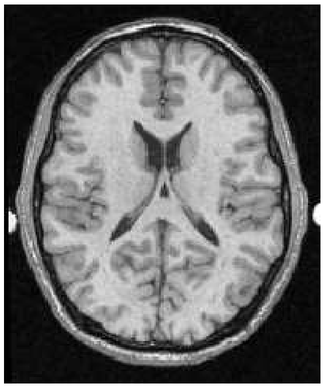

(c) ICBM103.


(d) ICBM104.


(e) ICBM105.


(f) ICBM106.


(g) ICBM107.


(h) ICBM108.

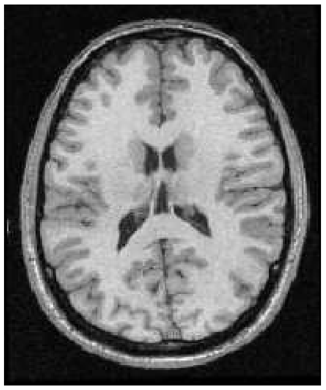

(i) ICBM109.

Figure 4.4: Real human brain images from the ICBM data set.

4.2 Performance Comparison

We compiled the original CPU-based PVE algorithm and the host program of the FPGA-based PVE algorithm useing GNU gcc 4.1.0 with level 2 optimization option to compile . They were executed using a single 1.5 GHz Itanium 2 processor on an Altix 350 system. The hardware-implemented portions of the FPGA-based PVE algorithm were distributed among four FPGA-accelerators. The computation time of the CPU-based and the FPGA-based algorithm were measured using $gettimeofday$ function, which reads the current CPU time from the host program. To allow for a fair comparison, the hardware execution time included the overhead associated with preparing the input buffer, loading bitstreams onto FPGAs, transmitting input buffers, running the bitstream and receiving resulting data [37].

Table 4.1 demonstrates the performance improvement achieved by the FPGA-based PD estimation function on the simulated and real images. The average computation time of the CPU-based and FPGA-based PD estimation function were 47.16 seconds and 18.70 seconds respectively, while the overall performance enhancement of the FPGA-based PD estimation implementation was an average 2.5× over the CPU-based PD estimation function.

The FPGA-based prior information estimation implementation also achieved a significant speedup over the CPU-based algorithm, as shown in Table 4.2. The average execution time per iteration of the CPU-based and the FPGA-based prior estimation function were 21.9 seconds and 1.9 seconds respectively, which represents a 11.5× performance enhancement. However, as the number of iterations required to satisfy the termination criteria differ between the two implementations, the speedup of the overall prior information estimation function was 9.4× instead of 11.5×. On average, the CPU-based prior information estimation function required 12 iterations for a total computation time of 263.40 seconds. Conversely, the FPGA-based algorithm required an average of 15 iterations and a total computation time of 28.52 seconds.

Table 4.3 provides the overall computation time spent by the CPU-based and the

Table 4.1: Performance enhancement achieved by FPGA-based PD estimation function.

Subject	CPU-based PD estimation	FPGA-based PD estimation	Speedup
colin27	47.97s	18.67s	2.6×
ICBM101	47.69s	18.69s	2.6×
ICBM102	46.7s	18.66s	2.5×
ICBM103	46.92s	18.77s	2.5×
ICBM104	46.94s	18.73s	2.5×
ICBM105	46.86s	18.76s	2.5×
ICBM106	46.74s	18.72s	2.5×
ICBM107	46.58s	18.66s	2.5×
ICBM108	46.73s	18.67s	2.5×
ICBM109	47.00s	18.7s	2.5×
ICBM110	47.28s	18.67s	2.5×
ICBM111	47.62s	18.72s	2.5×
ICBM112	46.82s	18.7s	2.5×
ICBM113	47.57s	18.72s	2.5×
ICBM114	47.11s	18.65s	2.5×
ICBM115	46.95s	18.64s	2.5×
ICBM116	47.09s	18.68s	2.5×
ICBM117	47.66s	18.73s	2.5×
ICBM118	47.89s	18.65s	2.6×
ICBM119	47.01s	18.65s	2.5×
ICBM120	47.30s	18.67s	2.5×
Average	47.16s	18.70s	2.5×

FPGA-based PVE algorithms, along with the overall speedup enhancement. The average computation time of the CPU-based and the FPGA-based PVE algorithms were 331.60 seconds and 65.09 seconds respectively. The overall computation time and speedup varied depending on the subject due to the number of the required iterations. The maximum speedup enhancement achieved by the FPGA-based PVE algorithm was 5.7× (ICBM102) when the same number of iterations were performed for both implementations. On the other hand, the minimum achieved speedup was 4.3× (ICBM114). The FPGA-based PVE algorithm achieved an average speedup of 5.1× over the CPU-based PVE algorithm.

Table 4.2: Performance enhancement achieved by FPGA-based P(C) estimation function. The number inside of bracket represents the number of iteration performed

Subject	CPU-based P(C) estimation[1]	FPGA-based P(C) estimation[2]	Speedup
colin27	246.93s (11)	27.59s (14)	9.0×
ICBM101	245.61s (11)	31.00s (16)	7.9×
ICBM102	305.60s (14)	27.74s (14)	11.0×
ICBM103	263.08s (12)	27.75s (14)	9.5×
ICBM104	284.96s (13)	34.64s (18)	8.2×
ICBM105	218.84s (10)	27.77s (14)	7.9×
ICBM106	262.31s (12)	27.32s (14)	9.6×
ICBM107	239.32s (11)	25.80s (13)	9.3×
ICBM108	279.37s (13)	25.72s (13)	10.9×
ICBM109	284.00s (13)	25.72s (13)	11.0×
ICBM110	243.12s (11)	27.33s (14)	8.9×
ICBM111	311.48s (14)	33.33s (17)	9.4×
ICBM112	240.18s (11)	25.62s (13)	9.4×
ICBM113	260.03s (12)	25.85s (13)	10.1×
ICBM114	219.59s (10)	29.28s (15)	7.5×
ICBM115	328.28s (15)	32.35s (17)	10.2×
ICBM116	285.51s (13)	32.34s (17)	8.8×
ICBM117	283.74s (13)	26.09s (13)	10.9×
ICBM118	284.71s (13)	29.15s (15)	9.8×
ICBM119	284.99s (13)	30.62s (16)	9.3×
ICBM120	264.7s (12)	26.01s (13)	10.2×
Average	263.40s (12)	28.52s (15)	9.4×

[1] Number of iteration × 21.9 seconds
[2] Number of iteration × 1.9 seconds

Table 4.3: Overall performance enhancement. The number inside of bracket represents the number of iteration performed

Subject	CPU-based PVE	FPGA-based PVE	Speedup
colin27	311.1s (11)	62.46s (14)	5.0×
ICBM101	309.3s (11)	67.56s (16)	4.6×
ICBM102	368.1s (14)	64.23s (14)	5.7×
ICBM103	326.1s (12)	64.64s (14)	5.0×
ICBM104	348.2s (13)	71.59s (18)	4.9×
ICBM105	281.8s (10)	64.50s (14)	4.4×
ICBM106	324.8s (12)	63.90s (14)	5.1×
ICBM107	302.1s (11)	62.68s (13)	4.8×
ICBM108	342.3s (13)	62.54s (13)	5.5×
ICBM109	347.4s (13)	62.57s (13)	5.6×
ICBM110	306.7s (11)	64.22s (14)	4.8×
ICBM111	375.20s (14)	70.08s (17)	5.4×
ICBM112	302.9s (11)	62.1s (13)	4.9×
ICBM113	323.5s (12)	62.4s (13)	5.2×
ICBM114	282.6s (10)	65.77s (15)	4.3×
ICBM115	391.2s (15)	68.9s (17)	5.7×
ICBM116	348.6s (13)	68.85s (17)	5.1×
ICBM117	347.2s (13)	62.63s (13)	5.5×
ICBM118	348.6s (13)	65.65s (15)	5.3×
ICBM119	347.8s (13)	66.98s (16)	5.2×
ICBM120	328s (12)	62.56s (13)	5.2×
Average	331.60s (12)	65.09s (15)	5.1×

4.3 Accuracy Comparison

We used the same digital phantom and real human brain 3D MR images to verify the accuracy of the images that were generated using the FPGA-based PVE algorithm. Furthermore, the simulated images from BrainWeb were used to evaluate the accuracy of implementation on the images with artifacts. We measured several statistical values to quantify the level of agreement between the images resulting from the two implementations.

The first statistic used to measure the accuracy of the resulting images was $sensitivity$. This value represents the ratio of the total number of voxels correctly classified as tissue i ($HW_i \cap SW_i$) to the total number of voxels classified as tissue i in the "golden standard" image (SW_i), that is,

$$Sensitivity = \frac{HW_i \cap SW_i}{SW_i}. \tag{4.1}$$

In this study, we employed the image classified using the CPU-based PVE algorithm as the "golden standard" image. The sensitivity values for all possible tissue classes were then calculated to evaluate the performance of the FPGA-based PVE algorithm, as these values gave a concrete idea of how well the FPGA-based PVE algorithm classified each tissue type with respect to the CPU-based PVE algorithm.

The second statistic used to measure the accuracy of the FPGA-based PVE algorithm was the $Kappa$ coefficient (K) [38]. Since the Kappa statistic is a relative value, which varies from 0 to 1 and represents the level of agreement, it is usually used to compare the accuracy of different algorithms. A Kappa coefficient of 1 indicates perfect agreement, whereas a value of 0 represents agreement equivalent to chance. Moreover, due to its simplicity and robustness, many fields of research, such as radiographic interpretation, diagnostics, linguistics and brain MRI classification have used the Kappa statistic [23], [9], [39], [40], [41], [42].

The Kappa coefficient provides a chance-correct measure of similarity between two classifications and is the ratio of the actual present agreement to the agreement expected by chance. It is defined as:

$$K = \frac{P_o - P_e}{1 - P_e}, \tag{4.2}$$

where P_o and P_e represent the observed and expected agreement respectively. For a classification of C different tissues, P_o is known as the "accuracy" and is defined as:

$$P_0 = \frac{1}{N}\sum_{i=1}^{C} a_i, \tag{4.3}$$

where the total number of samples is denoted as N and a_i represents the number of correctly classified tissue i. P_e is defined as:

$$P_e = \frac{1}{N^2}\sum_{i=1}^{C} c_i t_i, \tag{4.4}$$

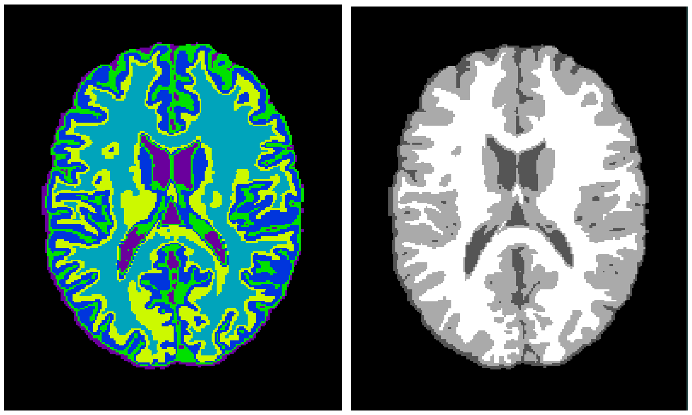
where c_i and t_i represent the the number of samples classified as tissue i and the number of the true number of samples in class i respectively.

Since anatomical studies and human brain mapping are usually only interested in the pure tissue classes (CSF, GM, and WM), in this study, we only calculated the Kappa coefficient over these three tissue types to evaluate the accuracy of the FPGA-based PVE algorithm. As a result, the images classified using each PVE implementations were re-classified to ensure that the mixed tissue classes, WMGM or CSFWM, were labeled as dominant pure tissue types and that biologically realistic contours were maintained. If a $voxel_i$ was classified as WMGM and 80% of the voxel was composed of GM, this voxel was re-classified as GM. Figure 4.5 provides the classified image from the PVE algorithm and the newly re-classified image.

Table 4.4 illustrates the calculated sensitivity and Kappa coefficients of the simulated and real 3D brain MR images. In order to validate the robustness of the FPGA-based PVE algorithm against different artifacts, we did not perform the non-uniformity correction algorithm on the simulated images downloaded from the BrainWeb. The average sensitivity of CSF, GM, WM, GMCSF and WMGM tissue

Table 4.4: Accuracy comparison.

Subject	CSF	GM	WM	GMCSF	WMGM	Kappa
colin27	0.994	0.998	0.999	0.996	0.998	1
pn0rf0	0.947	0.999	0.999	0.999	0.999	0.998
pn1rf0	0.983	0.997	0.999	0.996	0.997	0.999
pn3rf0	0.999	0.999	0.999	0.999	0.999	0.999
pn5rf0	0.999	0.999	0.999	0.999	0.998	1
pn7rf0	0.998	0.998	0.999	0.998	0.997	0.999
pn9rf0	0.997	0.997	0.998	0.998	0.995	0.999
pn0rf20	0.946	0.999	0.999	0.999	0.999	0.998
pn0rf40	0.961	0.999	0.999	0.999	0.999	0.998
ICBM101	0.996	0.997	0.999	0.994	0.995	0.999
ICBM102	0.996	0.996	0.998	0.995	0.995	0.999
ICBM103	0.996	0.998	0.998	0.995	0.996	0.999
ICBM104	0.996	0.997	0.999	0.996	0.997	0.999
ICBM105	0.997	0.998	0.998	0.995	0.996	1
ICBM106	0.996	0.997	0.998	0.996	0.995	0.999
ICBM107	0.991	0.998	0.998	0.997	0.996	0.999
ICBM108	0.995	0.997	0.998	0.997	0.996	0.999
ICBM109	0.994	0.998	0.998	0.998	0.997	0.999
ICBM110	0.998	0.997	0.998	0.997	0.997	0.999
ICBM111	0.993	0.997	0.998	0.996	0.996	0.999
ICBM112	0.995	0.997	0.999	0.998	0.996	0.999
ICBM113	0.994	0.998	0.999	0.997	0.996	0.999
ICBM114	0.995	0.998	0.999	0.998	0.995	0.999
ICBM115	0.995	0.997	0.998	0.998	0.996	0.999
ICBM116	0.992	0.997	0.999	0.998	0.995	0.999
ICBM117	0.990	0.997	0.999	0.997	0.996	0.999
ICBM118	0.993	0.997	0.999	0.997	0.996	0.999
ICBM119	0.997	0.998	0.999	0.994	0.996	0.999
ICBM120	0.992	0.998	0.999	0.997	0.996	0.999
Average	0.990	0.998	0.999	0.997	0.992	0.999

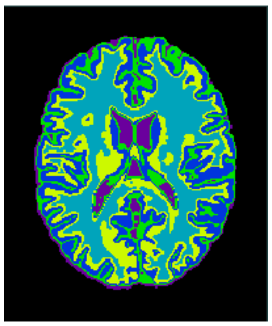


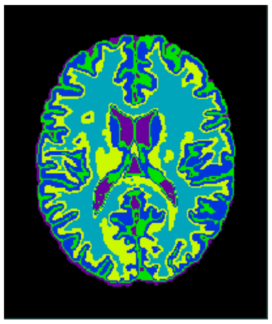
(a) 5 different tissues. (b) 3 different tissues.

Figure 4.5: The classified image from the FPGA-based PVE algorithm and the re-classified image (colin27).

over the whole image volume were 0.990, 0.998, 0.999, 0.997 and 0.992 respectively. The average Kappa coefficient calculated over the three pure tissues was 0.999.

Since the sensitivity and Kappa coefficients only indicate the level of agreement between two implementations, we performed a further test to evaluate the location of the differences. The voxels in the images classified using the CPU-based and FPGA-based PVE algorithms were compared and marked if the tissue was classified differently. This test was performed on both the simulated and real 3D brain images. Figure 4.6 shows the resulting digital phantom images. A single slice of the resulting classified image from each implementation is illustrated in Figures 4.6(a) and 4.6(b). The mismatched 1 mm^3 voxels were overlaid on the T1 images and represented by red, as shown in Figure 4.6(c). Figures 4.7 and 4.8 illustrates the resulting ICBM and simulated images respectively.


(a) CPU-based


(b) FPGA-based

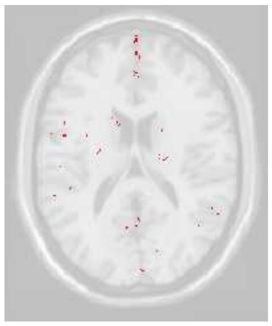
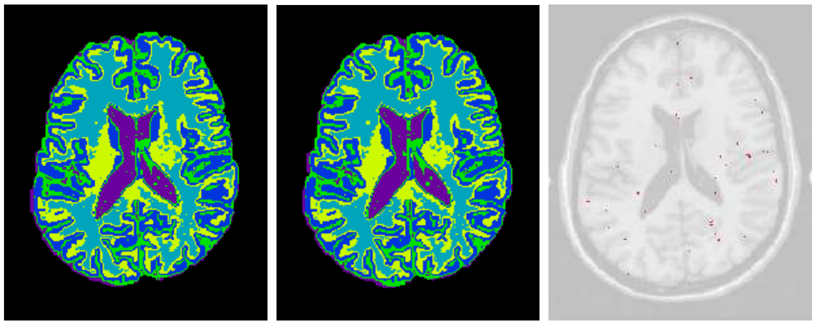

(c) Differences overlaid on the T1 image

Figure 4.6: Difference between two implementations (colin27).


(a) ICBM104

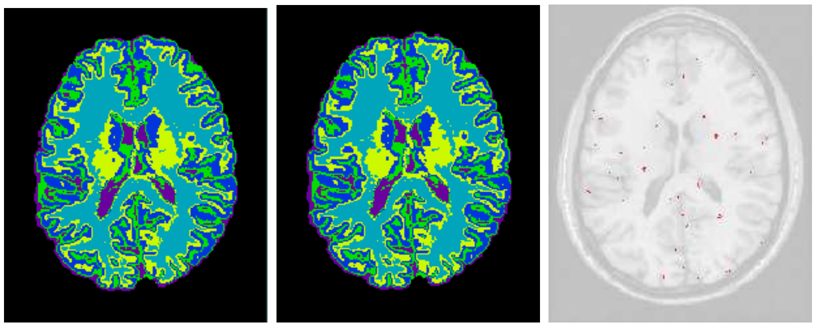
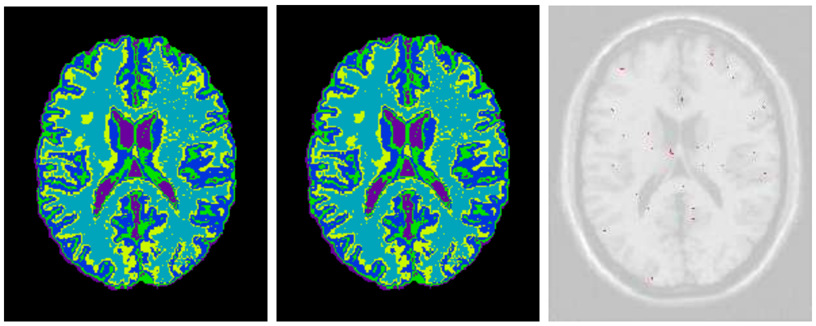
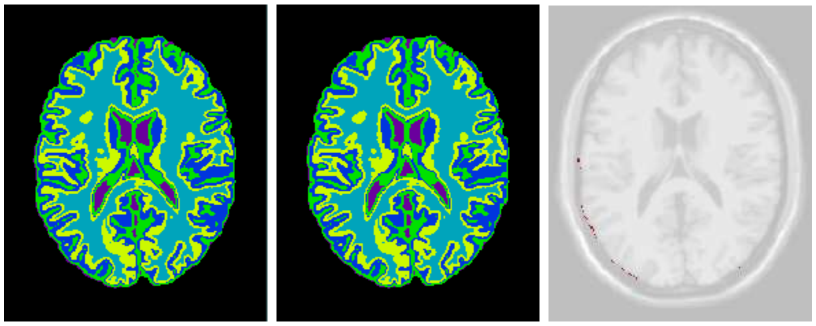

(b) ICBM105

Figure 4.7: ICBM images. CPU-based PVE(left), FPGA-based PVE(middle), Differences overlaid on the T1 image(right).



(a) PN9RF0



(b) PN0RF20

Figure 4.8: Simulated images with artifact. CPU-based PVE(left), FPGA-based PVE(middle), Differences overlaid on the T1 image(right).).

Chapter 5

Discussions and Conclusions

5.1 Discussions

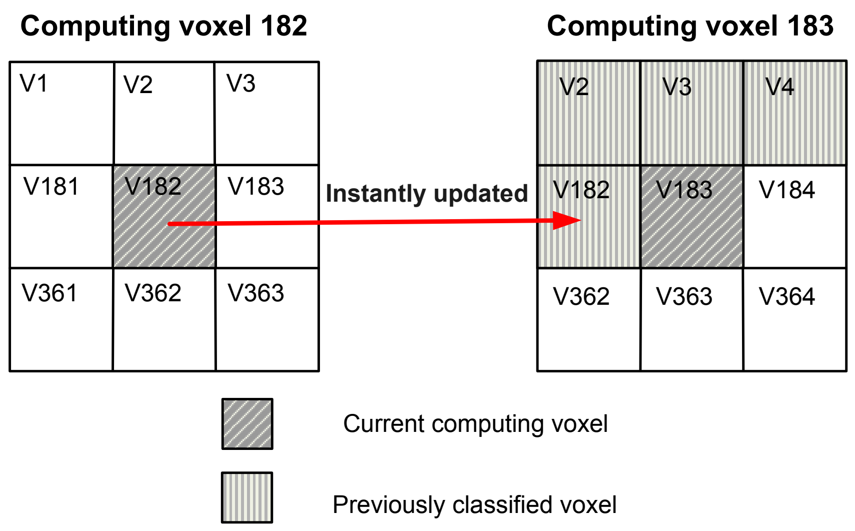
In this study, we implemented the 3D brain MRI tissue classification algorithm PVE on HPRC using the Mitrion-C HLL. Two computationally intensive functions, PD and P(C) estimation, were identified and accelerated using four FPGA-accelerators on two SGI RASC RC100s. The average performance enhancements achieved by the PD estimation and P(C) estimation function were 2.5× and 9.4× respectively. Moreover, the FPGA-based PVE algorithm achieved an average of 5.1× speedup over the conventional CPU-based PVE algorithm (varying from 4.3× to 5.7×). We also verified the accuracy of the images resulting from the FPGA-based PVE algorithm using several different statistics. The sensitivity and Kappa coefficients, described in Section 4.3, indicated that the results from the two implementation were slightly different. However, Figures 4.6, 4.7 and 4.8 illustrated that the observed difference would not cause significant errors in clinical studies, as the number of mismatched voxels was minimal and they were diversely distributed. Moreover, Table 4.4 demonstrated that the performance of the FPGA-based PVE algorithm is not degraded as the amount of artifacts increases.

The variable speedup and quantities of mismatched voxels observed for different subjects were due to differing strategies used for updating the estimated tissue type between two implementations. The CPU-based algorithm classified voxels sequentially and instantly stored the new classified tissue types into memory. As a result, the 26-neighbor voxels of the current computing voxel were instantly updated as they were changed. However, we could not implement a similar updating scheme for the FPGA-based PVE implementation due to data-dependencies between voxels. In order to fully exploit the parallelism offered by the FPGA, voxels needed to be classified independently. In the FPGA-based implementation, the independently classified voxels were first stored in the external SRAMs of the FPGA and then transferred to the memory on the Altix 350 system when the algorithm has processed all voxels. As a result, even the tissue class of one of 26-neighbor voxels was changed, it was not used to classify its adjacent voxel.

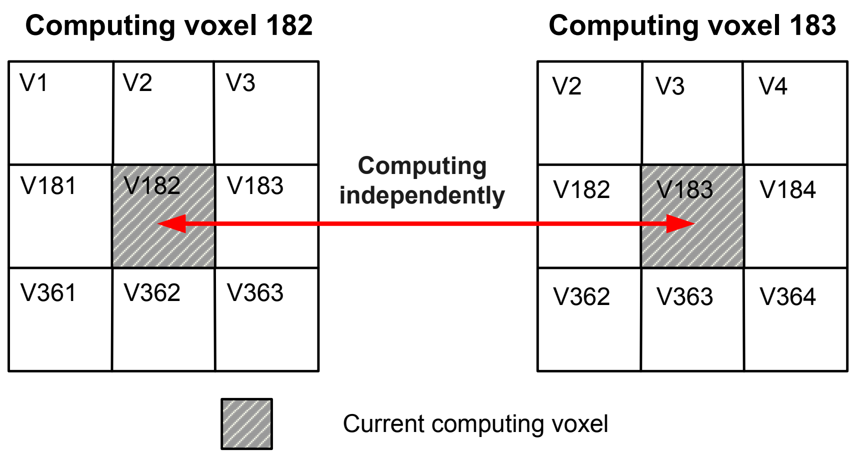
Figure 5.1 illustrates the two different tissue updating strategies described above. Note that in this figure, 8-neighbor voxels are used instead of 26-neighbor voxels to explain the difference between two methodologies. As shown in Figure 5.1(a), the CPU-based PVE algorithm previously classified certain neighbors (voxel 2, 3, 4 and 182) of voxel 183, before voxel 183 is classified. Therefore, when voxel 183 is classified, the new tissue types for the neighbor voxels are used instead of the tissue type from the previous iteration. On the other hand, the FPGA-based implementation classified every voxel independently, as shown in Figure 5.1(b). Therefore, the newly classified tissue types of voxel 2, 3, 4 and 182 are not used to classify voxel 183 because they are not yet updated.

The difference between two tissue updating schemes also degraded the performance enhancement of the FPGA-based implementation. As Tables 4.2 and 4.3 illustrated, the FPGA-based algorithm performed an average of 3 more iterations to converge than the CPU-based algorithm. As a result, if we can design a new tissue update methodology, which ensures the algorithm to be converged within the same number of iterations as the CPU-based implementation does, more performance enhancement can be achieved. The new scheme will reduce the total computation time of the proposed design by an average of 5.7 seconds (8.8%).

We profiled the FPGA-based PVE algorithm again and the new report is provided in Table 5.1. The report indicated that the computation time, spent by the



(a) CPU-based



(b) FPGA-based

Figure 5.1: Two different classified voxels updating scheme.

prior information estimation function in the FPGA-based PVE implementation, was significantly reduced from 79.4% to 44.2% of the total computation time. Moreover, the accelerated PVE algorithm spent 27% of the total computation time on the non hardware-accelerated functions, PVC estimation and I/O function. On the other hand, the computation time spent by the FPGA-based PD estimation function was actually increased from 15.4% to 29.0% of the total computation time. This indicated that the performance enhancement achieved by the FPGA-based PD estimation function was not significant compared to what was achieved by the FPGA-based prior information estimation function.

Table 5.1: GNU gprof profiling report of the FPGA-based PVE algorithm.

Functions	CPU-based	FPGA-based
†The estimation of the prior information	246.39s (79.4%)	27.59s (44.2%)
†The estimation of the probability densities	47.97s (15.4%)	18.67s (29.0%)
PVC Estimation	3.8s (1.2%)	3.8s (6.1%)
IO functions	12.4s (4%)	12.4s (19.9%)
Total	311.1s	62.7s

† Functions implemented on FPGA-accelerators

Of the execution time required for the FPGA-based PD estimation function, an average of 1.33 seconds were consumed normalizing the five computed PDs and determining the tissue type with the largest PD among the five. An average of 17.35 seconds were spent estimating the PD of five possible tissue classes for each voxel. Within the 17.35 seconds, approximately 78% was spent estimating PD of two mixed tissue classes due to the requirement for several floating point arithmetic operations and integration over fifty elements. These results indicated that estimation of the GMCSF and WMGM PDs was the bottleneck of the hardware implemented PD estimation function. This function could be improved using several alternative optimization techniques. One solution is to implement the fixed-point or custom-bitwidth arithmetic units for faster operation instead of floating point arithmetic units. This method also reduces the hardware resources, such that we can implement a higher level of the reduction tree to improve the performance of the integration operation. The other solution is to replace the primitive numerical integration algorithm used in the implementation with an advanced algorithm such as the Gauss-Legendre quadrature. This solution allows rapid and accurate integration with bigger integration step size.

While the CPU-based PVE algorithm used double-precision floating-point to estimate PD of tissues and P(C), the FPGA-based implementation used single-precision floating-point to reduce the required hardware resources and to achieve faster arithmetic operation. This difference did not cause any errors in the output images because the minimal precision error will not degrade determining the tissue that generates the maximum $P(C)P(x|c)$ term.

We used four CPUs to execute the bitstreams on four FPGAs by invoking Pthread functions which start four direct memory accesses (DMAs) simultaneously. Although three extra CPUs were utilized for the FPGA-based algorithm, the workload on the extra CPUs can be neglected since their execution times were minimal.

The performance of the FPGA-based PVE algorithm demonstrated that the RASC RC100 is a promising HPRC platform to accelerate the computationally intensive algorithms. Although the proposed implementation was running at 100MHz, it achieved 5.1$\times$ performance improvement over the CPU-based PVE algorithm, which was running at 1.5 GHz. The sophisticated RASC streaming feature was also very beneficial when processing the large input images utilized by the PVE algorithm as it increased efficiency by reducing the data overhead. Moreover, the wide-scaling was very simple to use and allowed the algorithm to fully exploit coarse-grained parallelism, which is provided by all available FPGA-accelerators.

When performing the FPGA-based PVE algorithm, we sequentially uploaded three different bitstreams (PD estimation, normalization and P(C) estimation) to the FPGA-accelerators using a loader FPGA, which was specially designed for fast bitstream uploading and downloading. However, the actual time required to upload a new bitstream into the FPGAs was approximately three seconds. In fact, approximately 14% (9 seconds) of the total computation time of the FPGA-based PVE algorithm was spent uploading the three bitstreams into the FPGA-accelerators. If the bitstream uploading and downloading time could be reduced to within one second, the average performance improvement achieve by the FPGA-based PVE implementation should become 5.6$\times$ instead of 5.1$\times$, as the total computation time is reduced.

The Mitrion-C HLL was an effective and useful tool for implementing the PVE algorithm on the RASC RC100 FPGA-accelerators. It abstracted most of the hardware details from the designer, meaning they can focus on the algorithm data flow. Only minor hardware consideration was required during the implementation phase to define the memory layout and load variables into the external SRAMs. Although the learning curve demanded by the language varied among different individuals, engineers with basic background in programming were able to develop simple algorithms and to achieve significant performance improvement in relatively short time [1].

Although it has been demonstrated that the Mitrion-C HLL can be used to overcome the hardware barrier imposed by tradition low-level methodologies, certain improvements were required. Sometimes the preliminary hardware usage report generated by the Mitrion compiler was not convincing. Several times the report indicated that a design consumed less than 50% of the FF and did not fully utilize the slices, however, it did not successfully complete place-and-route even after 9 hours. Generating a consistent hardware usage report is very critical for the developers so that they could reduce the design, before the time consuming place-and-route is performed.

The current version of the Mitrion-C HLL does not support eight available 64-bit wide registers, which can be very useful for storing parameters required by the algorithm. In the proposed implementation, we sent these parameters (mean and covariance of pure tissue classes) with input data in every segment. This approach was inefficient, as it increased the data overhead.

5.2 Conclusions

In the described study, we accelerated a quantitative analysis algorithm for 3D MR imaging using HPRC. The computationally intensive portions of the PVE algorithm were identified from the algorithm profiling report and were implemented on RASC RC100 FPGA-accelerators using Mitrion-C HLL. We used several simulated and real human brain MR images in order to evaluate accuracy and the performance enhancement of the FPGA-based PVE algorithm. The PD estimation function, implemented on four FPGAs, achieved an average speedup of 2.5$\times$ over the CPU-based implementation, while the performance improvement of the P(C) estimation function was approximately 9.4×. The overall performance improvement of the FPGA-based PVE algorithm was approximately 5.1× over the conventionally CPU-based algorithm. We also calculated the sensitivity and Kappa coefficients and used them as mean of verification. Furthermore, the images generated by both algorithms were then compared in order to verify the accuracy of the resulting images by the FPGA-based algorithm. These tests demonstrated that while there were very slight differences between the images generated by the two implementations, they were not critical in clinical studies.

5.3 Future Work

We have designed the proposed implementation to process 3D brain MR imaging with a resolution of 1 mm, which is the common resolution used in clinical studies. As the demand for higher resolution images increases, some subjects have been scanned with higher resolution. If the resolution of an MR image increases from 1 mm to 0.5 mm, the computation time of the conventional CPU-based PVE algorithm is expected to be increased by a factor of 8 since the number of voxels in an image also increases by a factor of 8. The FPGA-based PVE algorithm proposed in this study can also achieve a significant performance improvement when it is used to process a brain MRI data set with a resolution greater than 1 mm. These adjustments would require some minor memory structure modification and new partitioning strategies.

The CPU-based PVE algorithm is also developed to process multispectral MRI data sets. The multispectral PVE algorithm classifies brain tissues using the contrasts presented on the sets of T1, T2 and PD-weighted images, not only the contrast in T1-weighted image. As a result, the algorithm can classify the tissue classes more accurately, as more intensity information of tissues are available to estimate PDs of tissues. However, as the algorithm employs two more images, the computation time of PDs estimation function is increased because more voxels are needed to be processed. The proposed FPGA-based PVE algorithm can be used to process

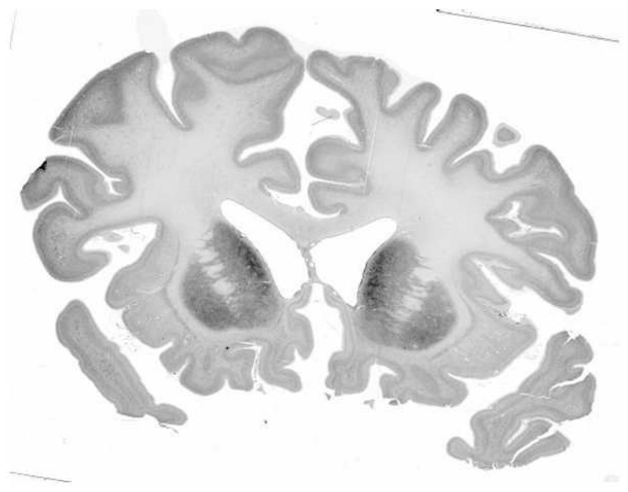


Figure 5.2: A single histological slice (Mohlberg, "Cytoarchitectonic Brain Mapping" presentation at MNI, May 18, 2007).

the multispectral data sets and to accelerate the computation. However, it also requires some memory structure and PDs estimation modifications such that the new algorithm can estimate PDs of tissues from the three images.

The histology on human brain focuses on the anatomical study of the microscopic structure of brain tissues. In the histological examination of a human brain, the postmortem brain is first frozen or embedded and then cut into very thin slices using a microtome. Each slice is then stained to contrast the different tissues and a high resolution microscope captures a picture of each slice for cytoarchitectonic mapping. One of the postmortem brain data sets available at MNI has a resolution of 20 μm. It is constructed of 7408 slices, where each slice contains 101200000 voxels (11500 $\times$ 8800 voxel), as shown in Figure 5.2. Due to the high resolution of the image, the histological data has been very attractive to many human brain mapping researchers. As a result, there has been a great interest in processing the histological images with the existing automated 3D brain MRI analysis algorithms, as they will generate more consistent and rapid results. However, the computation time of the algorithms is expected to be increased significantly, since the histological images consist of a tremendous number of voxels as compared to the conventional 3D brain MR images. Therefore, accelerating the algorithms to process the histological data using HPRC should be very beneficial to many brain clinical studies.

Bibliography

[1] J. J. Koo, D. Fernández, A. Haddad, and W. J. Gross, "Evaluation of High-Level-Language Methodology for High-Performance Reconfigurable Computer," *to appear in the Processings of the IEEE 18th International Conference on Application-specific System, Architectures and Processors (ASAP)*, 2007.

[2] J. J. Koo, A. C. Evans, and W. J. Gross, "Accelerating a Medical 3D Brain MRI Analysis Algorithm using a High-Performance Reconfigurable Computer," *to appear in the Proceedings of the IEEE 17th International Conference on Field Programmable Logic and Applications (FPL'07)*, 2007.

[3] A. P. Zijdenbos, R. Forghani, and A. C. Evans, "Automatic "pipeline" analysis of 3-D MRI data for clinical trials: application to multiple sclerosis," *IEEE Transactions on Medical Imaging*, vol. 21, no. 10, pp. 1280–1291, October 2002.

[4] E. R. Sowell, B. S. Peterson, P. M. Thompson, S. E. Welcome, A. L. Henkenius, and A. W. Toga1, "Mapping cortical change across the human life span," *Nature Neuroscience*, vol. 6, pp. 309–315, January 2003.

[5] A. C. Evans, "The NIH MRI study of normal brain development," *NeuroImage*, vol. 30, pp. 184–202, 2006.

[6] J. G. Sled, A. P. Zijdenbos, and A. C. Evans, "A nonparametric method for automatic correction of intensity nonuniformity in MRI data," *IEEE Transactions on Medical Imaging*, vol. 17, no. 1, pp. 87–97, February 1998.

[7] D. L. Collins, P. Neelin, T. M. Peters, and A. C. Evans, "Automatic 3D Inter-Subject Registration of MR Volumetric Data in Standardized Talairach Space," *Journal of Computer Assisted Tomography*, vol. 18(2), pp. 192–205, March/April 1994.

[8] D. L. Collins, C. J. Holmes, T. M. Peters, and A. C. Evans, "Automatic 3-D model-based neuroanatomical segmentation," *Human Brain Mapping*, vol. 3, no. 3, pp. 190–208, 1995.

[9] C. A. Cocosco, A. P. Zijdenbos, and A. C. Evans, "A fully automatic and robust brain MRI tissue classification method," *Medical Image Analysis*, vol. 7, no. 4, pp. 513–527, December 2003.

[10] J. S. Kim, V. Singh, J. K. Lee, J. Lerch, Y. Ad-Dab'bagh, D. MacDonald, J. M. Lee, S. I. Kim, and A. C. Evans, "Automated 3-D extraction and evaluation of the inner and outer cortical surfaces using a Laplacian map and partial volume effect classification," *Neuroimage*, vol. 27, no. 1, pp. 210–221, August 2005.

[11] J. Tohka, A. Zijdenbos, and A. Evans, "Fast and robust parameter estimation for statistical partial volume models in brain MRI," *NeuroImage*, vol. 23, no. 1, pp. 84–97, September 2004.

[12] Y. El-Kurdi, W. J. Gross, and D. Giannacopoulos, "Sparse Matrix-Vector Multiplication for Finite Element Method Matrices on FPGAs," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*, vol. 00. IEEE Computer Society, 2006, pp. 293–294.

[13] J. S. Beeckler and W. J. Gross, "FPGA Particle Graphics Hardware," in *Proceedings of the 13th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'05)*, vol. 00. IEEE Computer Society, 2005, pp. 85–94.

[14] T. Todman, G. Constantinides, S. Wilton, P. Cheung, W. Luk, and O. Mencer, "Reconfigurable Computing: Architectures and Design Methods," *IEE Pro-*

ceedings: Computer and Digital Techniques, vol. 152, no. 2, pp. 193–205, March 2005.

[15] Celoxica, *Handel-C Language Reference Manual for DK4 version*, Celoxica Limited., 2005.

[16] Impulse Accelerated Technologies, Inc., “C Programming Tools for FPGA Platforms,” 2007, http://www.impulsec.com/.

[17] S. Mohl, *The Mitrion-C Programming Language*, Mitrionics Inc., 2005.

[18] D. G. Nishimura, *Principles of Magnetic Resonace Imaging*. Stanford University, 1995.

[19] D. L. Collins, T. M. Peters, and A. C. Evans, “An automated 3D non-linear deformation procedure for determination of gross morphometric variability in human brain,” in *Proceedings of the International Conference on Visualization in Biomedical Computing*, vol. 2359. SPIE, 1994, pp. 180–190.

[20] R. O. Duda and P. E. Hart, *Patern Classification and Scene Analysis*. A Wiley-interscience publication, 1973.

[21] A. P. Zijdenbos, R. Forghani, and A. C. Evans, “Automatic Quantification of MS Lesions in 3D MRI Brain Data Sets: Validation of INSECT,” in *Proceedings of the First International Conference on Medical Image Computing and Computer-Assisted Intervention*, vol. 1496. Springer, 1998, pp. 439–448.

[22] H. S. Choi, D. R. Haynor, and Y. Kim, “Partial volume tissue classification of multichannel magnetic resonance images-A mixel model,” *IEEE Transactions on Medical Imaging*, vol. 10, no. 3, pp. 395–407, September 1991.

[23] V. Singh, “Use of a Non-Stationary Markov Random Field in Brain Tissue Partial Volume Segmentation,” Master’s thesis, Dept. of Electrical and Computer Eng, McGill University, February 2005.

[24] Silicon graphics Inc., *Reconfigurable Application-specific Computing User's Guide*, 007th ed., 2006.

[25] A. Dellson, G. Sandberg, and S. Mhl, "Turning FPGAs Into Supercomputers," in *Cray User Group (CUG) Conference*. Cray User Group, 2006.

[26] Mitrionics, "Mitrionics," March 2004, http://www.mitrion.com.

[27] Free Software Foundation, "GNU gprof - Table of Contents," November 1998, http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html.

[28] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. Morgan Kaufmann Publishers, 2003.

[29] L. Zhuo and V. K. Prasanna, "Sparse Matrix-Vector Multiplication on FPGAs," in *Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays(FPGA'05)*. ACM Press, 2005, pp. 63–74.

[30] G. R. Morris, V. K. Prasanna, and R. D. Anderson, "A Hybrid Approach for Mapping Conjugate Gradient onto an FPGA-Augmented Reconfigurable Supercomputer," in *Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06)*. IEEE Computer Society, 2006, pp. 3–12.

[31] S. Kleiman, D. Shah, and B. Smaalders, *Programming With Threads*, 1st ed. Prentice Hall, 1996.

[32] B. Nichols, D. Buttlar, and J. P. Farrell, *PThreads Programming*, 1st ed. O'Reilly Media, Inc., 1996.

[33] J. C. Mazziotta, A. W. Toga, A. Evans, P. Fox, and J. Lancaster, "A probabilistic atlas of the human brain: theory and rationale for its development. The International Consortium for Brain Mapping (ICBM)," *NeuroImage*, vol. 2, no. 2, pp. 89–101, June 1995.

[34] C. J. Holmes, R. Hoge, L. Collins, R. Woods, A. W. Toga, and A. C. Evans, "Enhancement of MR images using registration for signal averaging," *Journal of computer assisted tomography*, vol. 22, no. 2, pp. 324–333, March 1998.

[35] R. K.-S. Kwan, A. C. Evans, and G. B. Pike, "MRI simulation-based evaluation of image-processing and classification methods," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1085–1097, November 1999.

[36] C. A. Cocosco, V. Kollokian, R. K.-S. Kwan., G. B. Pike, and A. C. Evans, "BrainWeb: Online Interface to a 3D MRI Simulated Brain Database," in *Proceedings of 3rd International Conference on Functional Mapping of the Human Brain*. NeuroImage, May 1997, p. S425.

[37] V. Kindratenko, D. Pointer, D. Raila, and G. Steffen, "Comparing CPU and FPGA Application Performance," Natinal Center for Supercomputing Application (NCSA)," White Paper, February 2006.

[38] J. Cohen, "A Coefficient of agreement for nominal scales," *Education and Psychological Measurment*, vol. 60, pp. 37–60, 1960.

[39] S. Ruan, C. Jaggi, J. Xue, J. Fadili, and D. Bloyet, "Brain tissue classification of magnetic resonance images using partial volume modeling," *IEEE Transactions on Medical Imaging*, vol. 19, no. 12, pp. 1179–1187, December 2000.

[40] A. P. Zijdenbos, B. M. Dawant, R. A. Margolin, and A. C. Palmer, "Morphometric Analysis of White Matter Lesions in MR Images: Method and Validation," *IEEE Transactions on Medical Imaging*, vol. 13, no. 4, pp. 716–724, December 1994.

[41] A. J. Viera and J. M. Garrett, "Understanding Interobserver Agreement : The Kappa Statistic," *Family Madicine*, vol. 37, no. 5, pp. 360–363, May 2005.

[42] J. Carletta, "Assessing Agreement on Classification Tasks: The Kappa Statistic," *Computational Linguistics*, vol. 22, no. 2, pp. 249–254, 1996.