FPGA-based Hardware Software Co-design to Accelerate Brain Tumour Segmentation

Vinay Rayapati, Ravi Kiran Reddy Gogireddy, Ajay Kumar Gandi, Saketh Gajawada,
Gopala Krishna Reddy Sanampudi and Nanditha Rao
International Institute of Information Technology Bangalore, India
Email: {vinay.r, ravikiran.reddy515, ajaykumar.gandi, saketh.gajawada, gopalakirshna.reddy, nanditha.rao}@iiitb.ac.in

Abstract—**Brain tumors are a major concern, being the leading cause of cancer-related deaths. Computer-aided diagnosis significantly reduces the workload on physicians and improves cancer diagnosis and treatment. Brain tumor segmentation is a computationally intensive image-processing task. In this paper, we propose an FPGA-based Hardware-Software Co-design to accelerate this task using Watershed and Otsu thresholding algorithms. The FPGA handles parallel components, while the CPU manages sequential tasks in the same System-on-Chip (SoC). Using PolarFire Icicle FPGA platform, we process 20 MRI brain scan images (128x128) from the Kaggle dataset. Implementing both algorithms in parallel on the FPGA results in a 1.97× acceleration compared to a CPU-only implementation, mainly achieved by a 1973× reduction in latency when moving the Otsu algorithm from the CPU to the FPGA. This optimization employs DSP/MATH blocks, loop unrolling, and pipelining techniques.**

Index Terms—**Brain tumour segmentation, FPGA, Hardware-software co-design, Accelerator**

I. Introduction

Brain tumor segmentation is a critical task in medical image analysis that involves identifying tumor regions within MRI scans [1], [6]. This allows physicians to determine the exact shape and location of tumors. Computer Aided Detection (CAD) systems previously relied on processors to perform these compute-intensive tasks. However, processors have limited data processing speeds, making segmentation a computationally intensive task. To address this issue, researchers have explored hardware-based acceleration methods [4] - [6]. FPGAs offer parallel processing capabilities through reconfigurable hardware resources. This enables computationally demanding pre-processing and post-processing steps of segmentation, such as image thresholding, morphology operations, and intensity normalization to be accelerated.

Authors in [5] employed an iterative thresholding algorithm for segmentation. Most standard filters like Sobel can also be used for detecting edges or boundaries of tumors, but they lack the expected accuracy [5]. Otsu's global automatic image thresholding technique is implemented in [7], while [8] presents the Watershed algorithm on Xilinx Virtex-5 FPGA. The Watershed algorithm has also been implemented using different approaches such as an improvised version of Watershed [9], a parallel and pipelined implementation [10] and an FPGA-based hardware acceleration [10] [13] [16].

Hardware Software Co-design (HSC) is a popular design methodology that offers more flexibility and better performance compared to a pure hardware or software implementation. This methodology is a combination of software running on a processing system (PS) and hardware running on a programmable logic (PL). The PS and PL are typically part of the same SoC on the FPGA platform. The PL is an FPGA fabric-based accelerator that can implement parallel parts of the algorithm, which are usually compute-intensive. The PS is used to implement parts of the algorithm that are sequential. This approach of dividing tasks between the hardware and software based on their inherent algorithmic structures and compute-intensive nature forms the central idea of Hardware Software Co-design. In [11] [12] [13], an HSC methodology is used to implement a histogram of oriented gradients and image processing tasks. However, this method has not been explored much for tumor segmentation and identification purposes.

In this paper, we propose a hardware-software co-design architecture for accelerating brain tumor segmentation on an FPGA. The two algorithms chosen for acceleration are Otsu's Binarization for thresholding segmentation and the Watershed technique for region-based segmentation. Otsu's binarization algorithm has a stable and repetitive structure, making it well-suited for implementation on the programmable logic (PL). The Watershed method requires more complex processing and is run on the processor system (PS) or CPU. This allows leveraging both the hardware and software capabilities of the FPGA for higher performance. The threshold value needed by Otsu's algorithm is provided by the CPU-based Watershed method. Additionally, we use the faster DSP/MATH blocks along with loop optimization techniques such as pipelining and unrolling to improve the performance. We test our setup on twenty 128x128 MRI brain scan images from the Kaggle brain tumor dataset. The proposed architecture outperforms a CPU-only implementation by at least 2×. Processing many large-sized MRI images is typically time-consuming. Therefore, a 2× speed-up can also have major impacts on medical image applications. This improvement is attributed solely to the 1973× speedup obtained by implementing the Otsu algorithm on the FPGA.

II. Background

Otsu's binarization performs well for thresholding medical images to separate foreground objects (tumors) from the background. It provides a globally optimal threshold that maximizes the separability of the foreground and background pixel classes. This makes it suitable for the initial segmentation step. The Watershed technique is a region-based approach well-suited for refining the segmentation and separating touching or overlapping tumors. It identifies tumor boundaries more precisely than simple thresholding alone. Together, Otsu's binarization followed by Watershed segmentation provides a multi-step segmentation pipeline that leverages both intensity thresholding and region-based techniques. Therefore, we propose this hybrid approach to achieve high accuracy in brain tumor segmentation tasks.

The Watershed algorithm consists of key steps such as thresholding, morphology, dilation, and distance transform which are presented in Fig.ure 1. These steps need to be carried out sequentially; that is, the output of the previous step is required as an input to the next step. In the thresholding stage, the foreground and the background separation is done by making the pixels 0 or 255 compared with a manual threshold. Alternately, an optimum threshold can be taken as input from the Otsu thresholding. This is followed by morphology which removes noisy pixels in the image and dilation which expands the contour boundaries; this step is essential in detecting the edge of the contour. After dilation, we employ distance transform for better contour detection. In this stage, the nearest distance to the low pixel from the high pixel is calculated, and then the thresholding is performed to get a better contour.

The Otsu thresholding algorithm is an image thresholding technique that provides a single threshold value that separates the given image into two components: foreground and background. This threshold value is calculated using the Otsu thresholding algorithm by maximizing the inter-class variance or minimizing the intra-class (within-class) variance for the pixel intensity. The algorithm iterates through different threshold values and exhaustively searches for the single threshold value (T) that provides the minimum within-class variance.

$$\sigma_W^2(T) = W_b(T)\sigma_b^2(T) + W_f(T)\sigma_f^2(T) \tag{1}$$

where,

$$\begin{aligned} W_b(T) &= \frac{1}{N}\sum_{i=0}^{T-1} w(i) \\ W_f(T) &= \frac{1}{N}\sum_{i=T}^{N-1} w(i) \end{aligned} \tag{2}$$

In Equation (1), the within-class variance ($\sigma_W^2(T)$) is defined as the weighted sum of variances of two classes (background and foreground) where T is the threshold value. $W_b(T)$ and $W_f(T)$ defined in Equation (2) are the background and foreground class probabilities respectively. In Otsu thresholding, a histogram for the given input image is calculated first. The threshold value (T) is then iterated from 0 to N, where N=255 is the maximum value of a pixel for an 8-bit binary representation of the given image. Finally, the class probability (W), mean (μ), and variance (σ^2) are calculated to compute the within-class variance for each iteration of the threshold value. The threshold value that provides the minimum within-class variance is the output from the Otsu thresholding algorithm which is used to separate the foreground and background of an image.

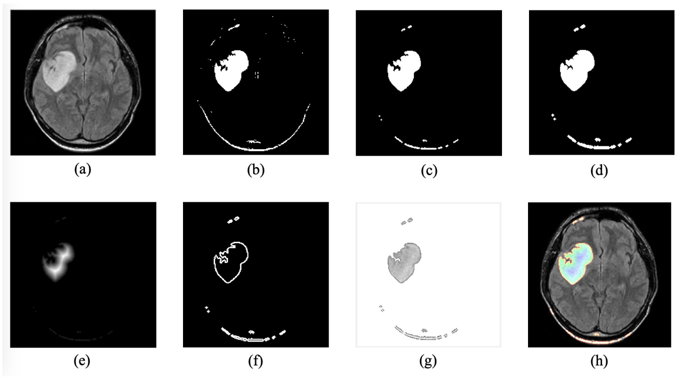

Fig. 1: Different stages of the Watershed Algorithm. a) Input Image b) Thresholding c) Morphology d) Dilation e) Distance Transform f) Detecting Contour g) Affected Region h) Output Image

III. Proposed Hardware Software Co-design approach and Experimental Setup

We adopt the Hardware-Software Co-design methodology to obtain design flexibility and better performance compared to a purely software or hardware implementation. In this methodology, (a) parts of the algorithm are implemented on a CPU or processing system (PS) as a software application, and (b) parts of the algorithm are implemented on the FPGA programmable fabric (PL) as a hardware implementation. The PS and PL are typically embedded in the same SoC on an FPGA platform. In the context of tumor segmentation, the Watershed algorithm has a non-pipelined structure and is, therefore, implemented on the software part (PS) of the FPGA. On the other hand, the Otsu thresholding algorithm has more parallel or structured operations and is compute-intensive. Therefore, we implement it on the PL (FPGA fabric).

Further, the Watershed algorithm requires an optimum threshold value as input. This is obtained from the output of the Otsu thresholding algorithm, thus connecting both the PS and PL components of the FPGA. The threshold value can also be set manually, but we designed it to obtain the threshold individually for each input image enabling runtime processing. This methodology of the division of algorithms based on their inherent structure provides more design flexibility and speeds up the given computation task.

We implement the HSC approach mainly on the Polar-Fire Soc Icicle FPGA kit and Xilinx Zedboard. We tested this approach on two different FPGAs to ensure our setup is easily portable. The real hardware setup with the segmented image output on the monitor is shown in Fig.ure 2. We tested the setup with nearly twenty 128x128 MRI grayscale images from the Kaggle brain tumor dataset and on images that did not have tumors. We use Microchip's Softconsole tool to implement the Watershed algorithm, while we use the High-Level Synthesis

(HLS) tool SmartHLS to implement the Ostu thresholding algorithm. The Otsu thresholding algorithm is exported as an IP from HLS into the Microchip Libero framework and programmed on the PL. The MRI images are transferred from the CPU and stored in the on-chip memory of the FPGA. We implement the Watershed algorithm using one of the four RISC-V cores in the FPGA and a 128kB scratch-pad L2 cache memory.

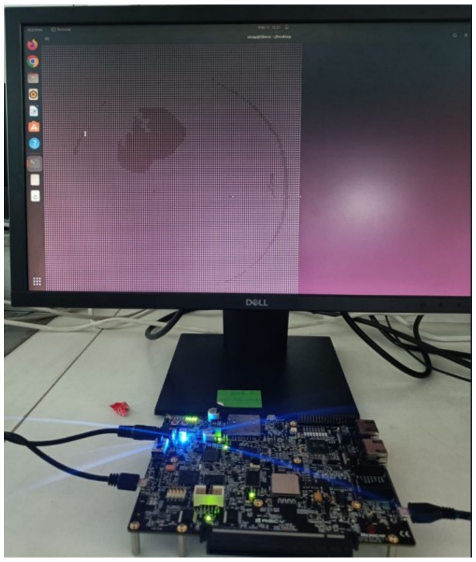

Fig. 2: Experimental setup showing the PolarFire FPGA kit. The segmented image is visible on the display.

We implement the following combinations of designs to enable a fair comparison:

1) Design 1: Otsu and Watershed implementations on PS at 100MHz
2) Design 2: Otsu and Watershed implementations on PS at 667MHz (This is the highest frequency of operation for the processor on the SoC)
3) Design 3: Otsu implementation on PL and Watershed implementation on PS at 100MHz (Hardware-Software co-design methodology)
4) Design 4: Otsu implementation on PL at 100MHz and Watershed implementation on PS at 667MHz (Hardware-Software co-design methodology)

IV. Results

In this section, we present the accuracy, latency, and resource utilization of the Otsu and Watershed algorithms (a) implemented entirely on the PS, without the FPGA intervention and (b) implemented with the co-design approach.

A. Accuracy

The input image is a 128×128 size MRI scan image comprising of 16384 pixels. Since the hardware implementation uses 8-bit registers, each pixel value ranges from 0 to 255. This setup has been tested for different MRI brain scan images.

TABLE I. Latency Comparison of various design combinations for WaterShed and Otsu algorithms on the FPGA (PS and PL)

Design	Otsu	Watershed	Freq (MHz)	Latency (us) PolarFire	Latency (ms) Zedboard
PS-only implementation	–	PS	667	2.06	-
PS-only implementation	–	PS	100	13.76	1.165
PS-only implementation	PS	–	667	2.01	-
PS-only implementation	PS	–	100	13.42	10.779
PL-only implementation	PL	–	100	0.0067	0.072
Design 1	PS	PS	100	27.18	11.863
Design 2	PS	PS	667	4.07	-
Design 3 (HSC)	PL	PS	100/100 (PS/PL)	13.766	1.456
Design 4 (HSC)	PL	PS	100/667 (PS/PL)	2.066	-

Six pairs of input images with the corresponding output image are shown in Fig.ure 3 . We observe that the FPGA-generated output image is just 24 pixels (out of 16384 pixels) different from the result obtained from the MATLAB-generated output image, indicating that the two outputs are nearly the same. The cause of the error could be the low bit-width implementation used in our design. This results in an accuracy of 99.85% for our implementation.

B. Latency comparison

In Table I, we present the runtimes or latency for the two algorithms as implemented on CPU and FPGA independently, in the first five rows. We also present the optimized design combinations in the rest of the table. The overall latency of Design 1 (both algorithms run on CPU) is 27.18 μs while the latency for Design 3 (Otsu on FPGA + Watershed on CPU) is 13.766 μs, resulting in an improvement of 1.97× with the co-design approach. We compare Designs 1 and 3 as they run at the same frequency (100MHz), while we compare Designs 2 and 4 which run at 667 MHz on the CPU. This improvement of 1.97× can be attributed solely to the speedup obtained by implementing the Otsu algorithm on the FPGA. The Otsu thresholding algorithm, when run on the FPGA, ran in 6.787 ns while the same algorithm ran on the CPU in 13.42μs. This results in a drastic 1973× improvement by moving the Otsu algorithm from CPU to an FPGA-based implementation. In spite of this major reduction in runtime for the Otsu algorithm, the overall processing time is still limited by the Watershed algorithm which runs on the CPU.

Further, we analyzed the cause of the latency reduction. We observe that parallelizing the Otsu algorithm in the FPGA reduces the latency for within-class variance computation. This can be explained as follows. The within-class variance is calculated for all the 256 threshold values to calculate the optimum threshold. We need to calculate the background and foreground class variance, mean, and class probabilities for each threshold value. This computation takes place sequentially in a processor. It would take 256× the actual time taken to calculate within-class variance for a single threshold value. However, in the case of an FPGA, these 256 computations can be done in parallel using multiple computing elements on the

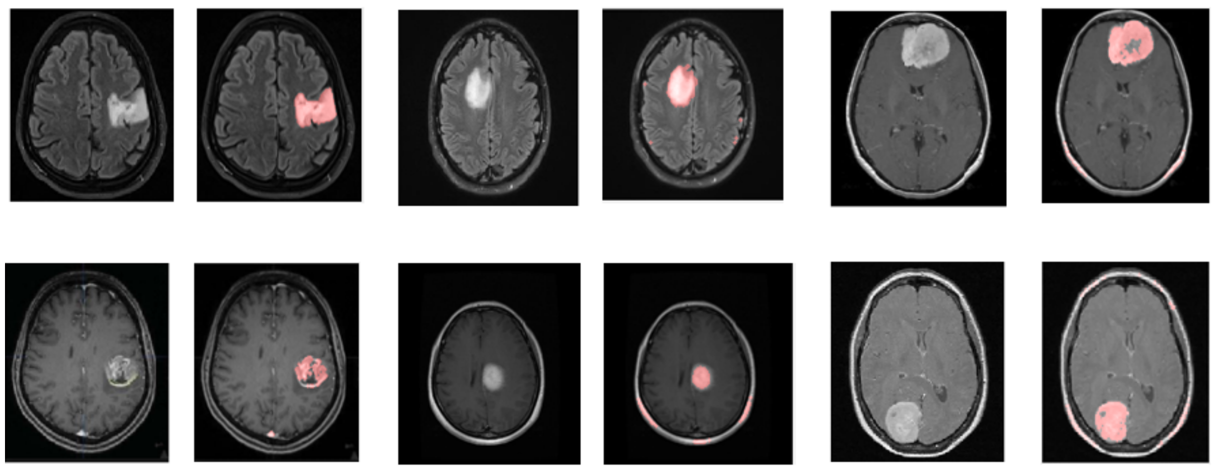
Fig. 3: The setup was tested on various MRI images: a few sample input-output pairs are shown

TABLE II. Latency, architecture and resource utilization: Comparison with related work. There has not been much work on hardware-software co-design and therefore we choose the most relevant work for comparison

Conference/ Year	**Proposed Work**		**Bioinform 2021 [14]**	**DSD 2022 [17]**	**ICM 2021 [15]**	**VLSID 2014 [16]**	**Elsevier Biosys 2020 [3]**
FPGA	PolarFire SoC	Zedboard	Alveo U280	ZCU104	Virtex 7	Virtex 5	Virtex-5
Execution Time / Freq	2.066 us	1.456 ms	0.15 s	1.9ms at 40MHz for 320x480	25.175 MHz	-	Path Delay 773 ns
Architectural novelty	Hardware Software Co-design	Hardware Software Co-design	Neural network model	SLIC superpixel	Classification using CNN	Otsu	Canny Edge and Watershed
LUTs	20451 (8.05%)	7386 (13.88%)	819956 (62.9%)	25,115 (8.66%)	21697 (7%)	-	123 (1%)
DSPs	21 (2.68%)	13 (5.90%)	5760 (63.83%)	-	69 (2%)	5 (3.9%)	-
BRAMs	-	17 (12.14%)	1504 (74.60%)	18 (13.46%)	19 (0.922%)	8 (2.7%)	-

TABLE III. Resource Utilization on the FPGA

Resource Type	**Used**	**Total Available**	**Percentage Used**
Fabric + Interface 4LUT	20451	254196	8.05
Fabric + Interface DFF	22290	254196	8.77
LSRAM/ on-chip memory	6	812	0.74
Math	21	784	2.68

PL. Thus migrating the design to an FPGA directly results in a 256× improvement for the within-class variance. This was improved further by using the following:

1) The DSP/MATH blocks, which are highly compute-efficient (speed up of nearly 10×).
2) Loop unrolling techniques to increase the parallelism of the computations.
3) Pipelining the different stages.

C. Resource utilisation and comparison with related work

In Table III, we present the hardware utilization for the implemented design on the PolarFire platform. The complex multiplication operations for within-class variance in the Otsu thresholding algorithm are mapped to the Math blocks in order to speed up the computations. Comparisons with threshold, histogram calculation, and separation of the background and foreground are mapped to the LUTs and DFFs. Existing works do not use the HSC approach, and therefore end up using more resources on the FPGA in addition to high latency. Our proposed design leverages efficient HSC techniques to achieve an execution time of just 2.06us. From Table II, we note that our design utilizes only 8.05% LUTs and 2.68% DSPs. Thus, the proposed design demonstrates significant improvements over prior work in terms of both speed and resource usage; and can be applied to large medical imaging datasets.

V. Conclusion

We presented a novel hardware-software co-design methodology to accelerate brain tumor segmentation on an FPGA. The proposed architecture is implemented on a PolarFireSoc Icicle FPGA platform. We tested the setup on nearly 20 MRI images of the brain size 128x128 from the Kaggle brain tumor dataset and observed an acceleration of 1.97×. This reduction is mainly attributed to a drastic reduction in time when the Otsu algorithm was moved from the CPU to an FPGA implementation resulting in a 1973× reduction in latency. This was achieved using DSP/MATH blocks, loop unrolling, and pipelining techniques on the FPGA.

References

[1] Gordillo N, Montseny E, Sobrevilla P. State of the art survey on MRI brain tumor segmentation. Magn Reson Imaging. 2013;31:1426–38.

[2] S. Sreekrishna Yadav and V. Mittal, ”FPGA Implementation of segmented feature fusion in MRI images using wavelet,” 2019 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2019

[3] V. Sivakumar, N. Janakiraman, ”A novel method for segmenting brain tumor using modified watershed algorithm in MRI image with FPGA,” 2019 International Conference on Communication and Electronics Elsevier Biosystems Volume 198, 2020

[4] Awan MG, Deslippe J, Buluc A, Selvitopi O, Hofmeyr S, Oliker L, Yelick K. ADEPT: a domain independent sequence alignment strategy for GPU architectures. BMC Bioinform. 2020;21:1–29.

[5] Remya Ajai A S, Sundararaman Gopalan, Comparative Analysis of Eight Direction Sobel Edge Detection Algorithm for Brain Tumor MRI Images, Procedia Computer Science, Volume 201, 2022, Pages 487-494, ISSN 1877-0509

[6] Angulakshmi, M., Lakshmi Priya, G.G., Automated brain tumour segmentation techniques A review, 2017, International Journal of Imaging Systems and Technology, 66-77-27-1,0899-9457

[7] J. G. Pandey, A. Karmakar, C. Shekhar and S. Gurunarayanan, ”A Novel Architecture for FPGA Implementation of Otsu’s Global Automatic Image Thresholding Algorithm,” 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems, Mumbai, India, 2014, pp. 300-305, doi: 10.1109/VLSID.2014.58.

[8] V. Sivakumar, N. Janakiraman, A novel method for segmenting brain tumor using modified watershed algorithm in MRI image with FPGA, Biosystems, Volume 198, 2020, 104226, ISSN 0303-2647

[9] C. J. Kuo, S. F. Odeh and M. C. Huang, ”Image segmentation with improved watershed algorithm and its FPGA implementation,” ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196), Sydney, NSW, Australia, 2001, pp. 753-756 vol. 2, doi: 10.1109/ISCAS.2001.921180.

[10] D. B. Khac Trieu and T. Maruyama, ”Implementation of a Parallel and Pipelined Watershed Algorithm on FPGA,” 2006 International Conference on Field Programmable Logic and Applications, Madrid, Spain, 2006, pp. 1-6, doi: 10.1109/FPL.2006.311267.

[11] Ghaffari S, Soleimani P, Li KF, Capson DW. A Novel Hardware–Software Co-Design and Implementation of the HOG Algorithm. Sensors. 2020; 20(19):5655.

[12] Chen AT-Y, Gupta R, Borzenko A, Wang KI-K, Biglari-Abhari M. Accelerating SuperBE with Hardware/Software Co-Design. Journal of Imaging. 2018; 4(10):122.

[13] A. Ahmad, M. A. Pasha and G. J. Raza, ”Accelerating Tiny YOLOv3 using FPGA-Based Hardware/Software Co-Design,” 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, Spain, 2020, pp. 1-5, doi: 10.1109/ISCAS45731.2020.9180843.

[14] Xiong, Siyu, et al. ”MRI-based brain tumor segmentation using FPGA-accelerated neural network.” BMC bioinformatics 22 (2021): 1-15.

[15] O. M. Elsayed, S. M. Ismail and M. A. Abd El Ghany, ”Register Transfer Level Model For CNN Tumor Detection on FPGA,” 2021 International Conference on Microelectronics (ICM), New Cairo City, Egypt, 2021, pp. 74-77

[16] J. G. Pandey, A. Karmakar, C. Shekhar and S. Gurunarayanan, ”A Novel Architecture for FPGA Implementation of Otsu’s Global Automatic Image Thresholding Algorithm,” 2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems

[17] A. Ghaderi, C. Ahlberg, M. Östgren, F. Ekstrand and M. Ekström, ”FP-SLIC: A Fully-Pipelined FPGA Implementation of Superpixel Image Segmentation,” 2022 25th Euromicro Conference on Digital System Design (DSD), Maspalomas, Spain, 2022, pp. 109-117