Hardware Accelerated Rician Denoise Algorithm for High Performance Magnetic Resonance Imaging

Efstathios Sotiriou-Xanthopoulos, Sotirios Xydis, Kostas Siozios, George Economakos and Dimitrios Soudris
School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
Email: {stasot, sxydis, ksiop, geconom, dsoudris}@microlab.ntua.gr

***Abstract*—Rician denoising is a mandatory task of Magnetic Resonance Imaging (MRI), as it enables higher-quality image processing, which is crucial for correct diagnosis. However, denoising is a slow task, especially because of the increased image resolution and the need for high image clarity. A solution towards this need is the implementation of rician denoise algorithm onto hardware. In this paper, we propose a hardware implementation of rician denoise, which processes the MR image into segments in a pipelined manner, while avoiding further processing on already denoised pixels of the image. Using a synthetic MRI scan separated into 16 segments, the proposed implementation achieves a speedup of $6.8\times$ with comparable image quality, as compared to a software-only approach running on Intel Core2Duo.**

Keywords—Magnetic Resonance Imaging; Rician Denoise; Hardware Acceleration; High Performance

I. Introduction

Magnetic Resonance Imaging (MRI) is a domain of utmost importance in medical imaging, as it enables the accurate diagnosis of diseases in complex human organs, like the brain or the cardiovascular system, by depicting their inner status and possible structural abnormalities. A typical MRI pipeline comprises four basic tasks [1]: Reconstruction, Denoising, Registration and Segmentation. Reconstruction converts the signal produced by the MRI scanner to a 3D image of $X \times Y \times Z$ pixels, each of which depicts an inner point of the organ-under-examination. As long as the MRI scanning induces noise to the 3D image, a denoising algorithm restores the image content. For the scope of this paper, we focus on denoising of images with rice-distributed (or rician) noise. Afterwards, the denoised image is further processed by the registration and the segmentation phases, where image differences and distinctive parts of the organ are detected respectively.

However, the requirement for images of high clarity and the high resolution of typical MRI scans have increased the execution time of rician denoising: A typical software-only implementation [2] [3] may take tens of seconds [4] or even minutes, depending on the image resolution and the denoising effort. Such delays should be avoided especially when the physician has to examine multiple MRI scans and make a correct diagnosis in very strict deadlines. A commonly used solution to alleviate this problem is to implement rician denoising onto hardware. Apart from the execution time improvement, another advantage is that a hardware implementation can be used as a PC peripheral or as a mobile-scale embedded system, which enables the physician to remotely examine a MRI scan from a low-end laptop or tablet, e.g. in case of emergency.

In [1], a study on domain-specific architectures shows the benefits of using specialized processing elements (e.g. GPUs, FPGAs, etc) in such applications, while another survey for accelerating medical applications is presented in [5]. Also, heterogeneous many-accelerator systems-on-chip architectures, featuring multiple CPUs, memories, hardware cores and FPGAs, have been proposed for offering significant performance and/or energy improvements in MRI applications [6]. To enable rician denoising parallelization, a typical approach is to analyze the dataflow graph (DFG) of each algorithm, in order to examine which arithmetic operations (additions, multiplications, square roots, etc.) can be parallelized [7], and map each operation onto specific parallel-running hardware components. In such a selection, the performance improvements come from the parallelization opportunities of the algorithm DFG.

This paper extends the existing hardware mapping and parallelization solutions, by presenting an efficient pipelined hardware architecture for rician denoising of MRI 3D scans, which enables the image processing into parallel segments, while respecting the data dependencies among the image segments, in order to fully retain the denoising accuracy. The basic contribution of this work, as compared to the existing ones, is that the parallelization is achieved not only by parallelizing the arithmetic operations, but also by processing in parallel multiple image parts which do not have data dependencies among each other. In addition, it is noticed that some areas of the 3D image (e.g. those which are close to the edges) are denoised much earlier than others (e.g. at the center of the image). The proposed architecture avoids unnecessary processing of those areas for sake of performance, while achieving comparable denoising quality.

For the scope of this paper, the evaluation of the proposed architecture will be achieved by using a representation of the overall system, also known as Virtual Platform (VP) [8], where each component of the hardware architecture is virtually modeled in software. Afterwards, the VP components are simulated altogether, so that the designer can validate the system functionality (i.e. examine the correctness of data processing) and explore the best combinations of architectural parameters (e.g. clock frequency, memory size, memory type, parallelization degree, etc.) towards performance, area and power consumption. For VP modeling, we will use SystemC [9], which combines the software execution with hardware description language features, thus allowing the modeling of any hardware component in multiple levels of abstraction.

The rest of the paper is organized as follows: Section II presents a theoretical background of rician denoise. Section III describes the proposed pipelined hardware implementation. Section IV shows the effectiveness of the proposed architecture. Finally the conclusions are presented in Section V.

MOBIHEALTH 2014, November 03-05, Athens, Greece

DOI 10.4108/icst.mobihealth.2014.257361

II. Background on Rician Denoise

Rician denoise is a total variation algorithm which tries to remove noise n from an observed image f (obtained from reconstruction phase), by finding a denoised image u, such that

$$f = u + n \tag{1}$$

Every possible u which satisfies Equation 1 belongs to a domain Ω of images, which is the domain of function f. To achieve this goal, the first step is to model the rician noise distribution. A most common one is the Rudin-Osher-Fatemi (ROF) model, firstly presented in [10]. According to ROF model, the objective for MRI denoising is expressed in Equation 2:

$$\min_{u \in BV(\Omega)} \int_{\Omega} |\nabla u| \, dx + \lambda \int_{\Omega} \left[\frac{u^2 + f^2}{2\sigma^2} - \log I_0 \left(\frac{uf}{\sigma^2} \right) \right] dx \tag{2}$$

In Equation 2, $I_0()$ is the zero-order Bessel function, σ is the Rician noise constant parameter, while $\lambda > 0$ denotes the variation step of rician denoising; the higher the value of λ, the more intense the variation will be, albeit at the expense of denoising quality. This minimization objective is achieved by using a gradient descent expressed in Equation 3:

$$\frac{\partial u}{\partial t} = \nabla \frac{\nabla u}{|\nabla u|} - \frac{\lambda}{\sigma^2} u + \frac{\lambda}{\sigma^2} \frac{I_1 \left(\frac{uf}{\sigma^2} \right)}{I_0 \left(\frac{uf}{\sigma^2} \right)} f \tag{3}$$

The proposed hardware architecture is based on the gradient descent software implementation of [2]. According to the source code and the documentation [11] of the software implementation, the term $\nabla \frac{\nabla u}{|\nabla u|}$ of Equation 3 for each pixel $u_{x,y,z}$ can be approximated by the following equation:

$$\nabla \frac{\nabla u}{|\nabla u|} \approx \sum_{(x',y',z') \in N_{x,y,z}} (g_{x',y',z'}) \tag{4}$$

Where $N_{x,y,z}$ is the set of neighbor coordinates of (x, y, z) and $g_{x,y,z}$ is a gradient function approximating $\frac{1}{|\nabla u|}$:

$$\begin{aligned} g_{x,y,z} = [&(u_{x+1,y,z} - u_{x,y,z})^2 + (u_{x-1,y,z} - u_{x,y,z})^2 + \\ &+ (u_{x,y+1,z} - u_{x,y,z})^2 + (u_{x,y-1,z} - u_{x,y,z})^2 + \\ &+ (u_{x,y,z+1} - u_{x,y,z})^2 + (u_{x,y,z-1} - u_{x,y,z})^2 + \epsilon]^{-1/2} \end{aligned} \tag{5}$$

In addition, the gradient descent is implemented as an iterative minimization algorithm, where the output of each iteration k is used as input to iteration $k+1$. The algorithm stops when total convergence is achieved, i.e. the difference between the input and output becomes lower than a given threshold, for all the image pixels. To enable this iterative implementation, Equation 3 is re-written as follows:

$$u^{k+1}_{x,y,z} = \frac{u^k_{x,y,z} + D \left(G(g^k_{x,y,z} u^k_{x,y,z}) + \frac{\lambda}{\sigma^2} \frac{I_1 \left(\frac{u^k_{x,y,z} f_{x,y,z}}{\sigma^2} \right)}{I_0 \left(\frac{u^k_{x,y,z} f_{x,y,z}}{\sigma^2} \right)} f_{x,y,z} \right)}{1 + D \left(G(g^k_{x,y,z}) + \frac{\lambda}{\sigma^2} \right)}, \tag{6}$$

$$G(p_{x,y,z}) = \sum_{(x',y',z') \in N_{x,y,z}} (p^k_{x',y',z'}) \tag{7}$$

Equations 4, 5 and 6 also describe the data dependencies of the rician denoising algorithm. In particular, they imply that a $5 \times 5 \times 5$ cube of neighbor pixels is required, the center of which will be the pixel $u_{x,y,z}$ to be denoised. Finally, in order to achieve lower execution time, the zero-order and first-order Bessel functions I_0 and I_1 are polynominally approximated, according to [11].

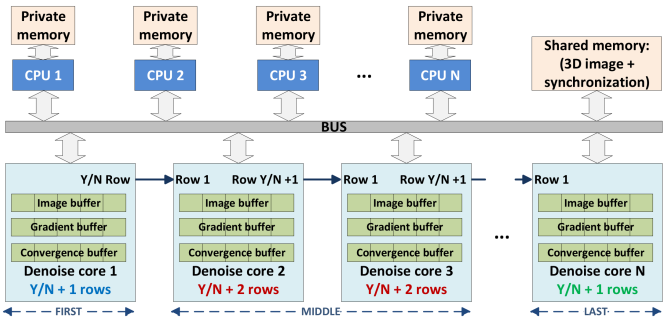


Fig. 1. The proposed hardware implementation for rician denoise

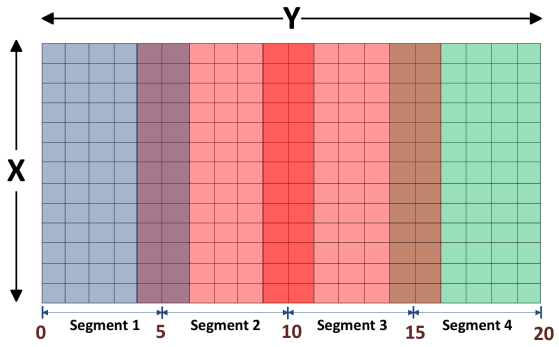


Fig. 2. Segmentation of each image slice

III. Efficient Hardware Implementation of Rician Denoise

Figure 1 depicts the pipelined architecture of the proposed implementation of rician denoise. The pipeline consists of N hardware cores, with private image, gradient and convergence buffers, each of which processes a $X \times (Y/N) \times Z$ segment of the 3D image, denoising one image pixel each time. When a core $i \in [1...N]$ processes a 2D slice in coordinate z_i of Z axis, the previous core $i-1$ can process in parallel a 2D slice in the next coordinate $z_{i-1} = z_i + 1$.

Management of the data dependencies among hardware cores: At the first and the last rows of each segment, some data dependencies for gradient computation may be violated. To avoid such a violation, each core in the middle (i.e. except first and last core) also takes as input two extra rows, one at the beginning and another at the end of its input, thus covering in total $Y_c = (Y/N) + 2$ rows. Also, the first and the last core include one extra row, at the end and at the beginning respectively, thus $Y_c = (Y/N) + 1$. Such a technique leads to overriden image rows, as in Figure 2, where a segmentation example for $N = 4$ demonstrates which rows of the 2D slice are processed by each core. However, in any case, the actual denoising output of every core is an image segment with (Y/N) rows, while the extra rows are utilized only for inner calculations.

In addition, the image segment denoising in core i requires the preceding neighbor pixels to be already denoised, which may also lead to dependency violations at the limits of two adjacent segments. To override this issue, each core $i = 0, ..., (N-1)$ sends the denoised row before last (i.e. (Y/N) and $(Y/N)+1$ respectively) to the next core $j = i+1$, $j = 1, ..., N$, which stores the row pixels into a buffer. Both techniques are of utmost importance, as they ensure the correctness and accuracy of the hardware implementation.

Buffer analysis: When applying the rician denoise algorithm on pixel $u_{x,y,z}$, a set of buffers is used in order to cover data dependencies, avoid unnecessary re-computations by storing previously calculated internal results, and trace the early-converged image areas.

In particular, a "gradient buffer" is utilized for storing previously-calculated gradients: It is noticed that the gradients $g_{x-1,y,z}$, $g_{x,y-1,z}$ and $g_{x,y,z-1}$, for $x > 0$, $y > 0$ and $z > 0$, have already been calculated, during denoising of the preceding pixels. Thus, unneccessary arithmetic operations can be avoided, e.g. denoising $u_{x,y,z}$ uses gradient $u_{x,y,z+1}$, which can be re-used for denoising $u_{x-1,y,z+1}$. Thus, when denoising a pixel $u_{x,y,z}$ for $y > 0$, only gradient $g_{x,y,z+1}$ can be calculated, while for the remaining gradients previously-calculated values can be re-used. The only exception is for $y = 0$, only preceding gradients can be re-used, while all succeeding gradients should be calculated as well. In total, the re-usable gradients range from $g_{x,y,z-1}$ to $g_{x,y,z+1}$. Thus, the length of the gradient buffer should be $[2 \times X \times Y_c + 1]$ elements.

An "image buffer" stores the neighbor pixels required for denoising, as well as for gradient computation. The pixels needed for denoising range from $u_{x,y,z-1}$ to $u_{x,y,z+1}$, however the calculation of gradient $g_{x,y,z+1}$ requires pixel $u_{x,y,z+2}$ too. To satisfy all the data dependencies, a pixel range from $u_{x,y,z-1}$ to $u_{x,y,z+2}$ has to be covered, thus the length of the image buffer should be $[3 \times X \times Y_c + 1]$ elements.

Finally, during denoising, some image areas (e.g. at the edges) may converge earlier than others (e.g. at the center). Formally, in iteration k, there are coordinates (x, y, z), where $\left|u^k_{x,y,z} - u^{k-1}_{x,y,z}\right| < T$. In such cases, those pixels will only be used for gradient calculation, while no further denoising may be applied. Towards this direction, a $[X \times Y_c \times Z]$ memory module called "convergence buffer" is used for marking the early-converged pixels of the 3D image.

Hardware core manipulation: Each hardware core is controlled by a dedicated CPU via a bus. The running software executes every iteration of the algorithm, by manipulating the input/output data of the hardware core. A CPU can obtain/store the initial image, the intermediate image segments and the final output by accessing a shared memory module. In addition, the shared memory stores the appropriate synchronization mechanisms (e.g. semaphores, etc.), in order to ensure that the rows to be sent from core i to core $i+1$ will not be corrupted.

The CPUs need $X \times Y_c \times (Z+2)$ accesses to their hardware cores in order to fully process an image segment:

1) The first $2 \times X \times Y_c$ accesses populate the image buffer, while no denoising is made.
2) In each of the next $X \times Y_c \times (Z-2)$ accesses, the hardware core stores the input pixel $u_{x,y,z}$ to the image buffer and denoises the pixel $u_{x,y,z-2}$ by using the preceding and succeeding pixels which are all stored in the image buffer. Afterwards, the denoised pixel replaces the initial one in the buffer. During denoising, the gradient and convergence buffers are also populated. If the pixel to be denoised has converged in an earlier iteration, according to convergence buffer, no further denoising is applied and the pixel is returned to the CPU unaffected.
3) The final $2 \times X \times Y_c$ accesses flush the image buffer, the remaining pixels of which are denoised.

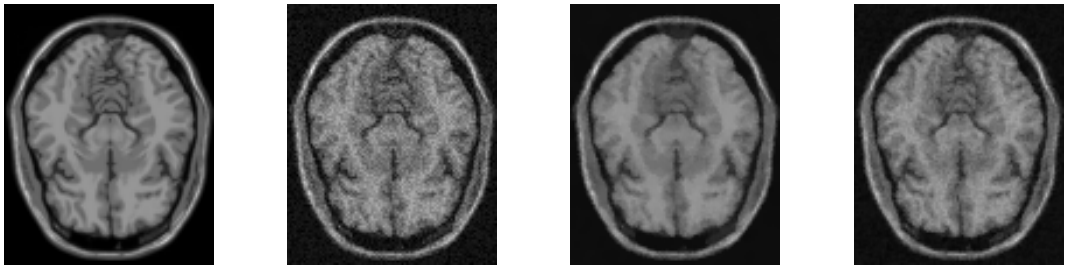
IV. Experimental Results

For evaluation purposes, a virtual platform of the proposed architecture with $N = 16$ denoising cores has been deployed. The hardware cores are controlled by ARM11 CPUs, each having 32-kilobyte instruction and data caches. The operation frequency of hardware cores is 333 MHz, while each ARM11 CPU operates at 500 MHz. The functionality and the delay of each hardware core are modeled in custom-made SystemC modules, while the CPU, the memories and the bus were obtained from SoCLib [12] component library. The system can be emulated by compiling the source code of the VP and running the produced simulator. For the interaction among the VP components, the simulator includes a scheduler and other event handling mechanisms which are provided by the SystemC kernel.

The 3D image to be used for simulation is a synthetic MRI scan of a healthy brain, with dimensions $90 \times 112 \times 90$, obtained from the BrainWeb [13] database. However, the proposed implementation can denoise MRI scans of any organ and in any resolution. As BrainWeb only supports Gaussian noise, we obtained a clear 3D image, a slice of which is shown in Figure 3a, and we added rice-distributed noise. The maximum noise value is equal to the 9% of the maximum luminosity of the 3D image. The respective slice of the resulted image is depicted in Figure 3b. To validate the denoising quality of the proposed architecture, a reference denoising has been made (Figure 3c) by using the software implementation available in [2]. The result of denoising by using the proposed hardware implementation is shown in Figure 3d.

In both software and hardware implementation, rician denoise requires 12 iterations in order to denoise the input 3D image. However, there is a result difference between the hardware and the software implementation, which relies on the fact that the hardware implementation does not apply further denoising in early-converged areas of the image segments. However, after a careful analysis, the difference ranges only from -0.4% to +0.4% for 98.9% of the 3D image, while no difference exists for the 77.6% of the image. In addition, it is noticed that some details depicted in Figure 3a can be better shown in Figure 3d rather than in Figure 3c, which means that the depiction of some structural parts of the brain can be better achieved when avoiding the early-converged image areas.

To investigate the time improvements of the proposed hardware implementation, a 16-segment implementation is firstly compared to a single-segment hardware implementation,



(a) (b) (c) (d)

Fig. 3. (a) A slice of the synthetic MRI scan, obtained from BrainWeb; (b) The MRI scan with rice-distributed noise; (c) Denoising with the reference software implementation [2]; (d) Denoising with the proposed 16-core hardware implementation

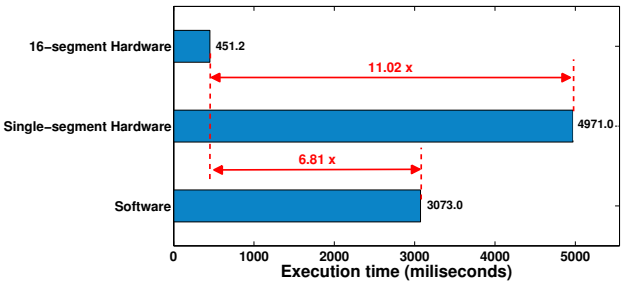


Fig. 4. Comparison of execution time

derived from the same VP by setting $N = 1$. According to Figure 4, the pipeline of the proposed architecture can result to $11\times$ lower execution time, thus covering both the synchronization overhead and the extra accesses needed for populating the buffers.

In addition, the 16-segment hardware architecure is compared to the reference software implementation, which has been executed on Intel Core2Duo[1], running at 2.4 GHz. As Figure 4 depicts, the hardware architecure results to time improvements reaching $6.81\times$, which are very significant if we take into consideration that the proposed architecture incorporates slower CPUs. These gains were resulted not only by pipelining, but also by not applying denoising on early-converged image parts. This is also noticed when comparing the single-segment hardware implementation with the reference software. In particular, the use of slower CPUs to the hardware implementation and the need for populating the buffers before actual denoising might result to a time overhead reaching 62%, as compared to the software implementation. However, as compared to the overall time gains, this overhead remains low, while a significant part of the time overhead is also covered by avoiding the early-converged image parts.

V. Conclusion

This paper presents a hardware implementation for rician denoising, which processes the input MRI scan in segments, in a pipelined manner, thus achieving significant performance improvements. In addition, the implementation is able to avoid further denoising on early-converged pixels of the 3D image, thus resulting to further performance enhancement. In total, when denoising a synthetic MRI scan divided in 16 segments, the proposed hardware implementation can achieve execution time improvements reaching $6.81\times$, as compared to the reference software implementation.

[1]Only one of the CPU cores is utilized, as the software implementation is single-threaded.

Acknowledgment

This work has been partially supported by the EC funded project Swan-iCare (FP7-ICT, Project No. 317894) and "TEAChER: TEach AdvanCEd Reconfigurable architectures and tools" project funded by DAAD (2014).

References

[1] A. Bui, K.-T. Cheng, J. Cong, L. Vese, Y.-C. Wang, B. Yuan, and Y. Zou, "Platform characterization for domain-specific computing," in *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, Jan 2012, pp. 94–99.

[2] (2014) Center of domain-specific computing, CDSC image processing pipeline. https://code.google.com/p/cdsc-image-processing-pipeline/.

[3] P. Coupé. (2014) MRI denoising software. https://sites.google.com/site/pierrickcoupe/softwares/denoising-for-medical-imaging/mri-denoising.

[4] M. Vatsa, R. Singh, and A. Noore, "Denoising and segmentation of 3d brain images," in *IPCV'09*, 2009, pp. 561–567.

[5] J. Cong, V. Sarkar, G. Reinman, and A. Bui, "Customizable domain-specific computing," *Design Test of Computers, IEEE*, vol. 28, no. 2, pp. 6–15, March 2011.

[6] J. Cong, M. Ghodrat, M. Gill, B. Grigorian, H. Huang, and G. Reinman, "Composable accelerator-rich microprocessor enhanced for adaptivity and longevity," in *IEEE International Symposium on Low Power Electronics and Design (ISLPED), 2013*, Sept 2013, pp. 305–310.

[7] J. Cong, M. A. Ghodrat, M. Gill, B. Grigorian, and G. Reinman, "Charm: A composable heterogeneous accelerator-rich microprocessor," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '12. New York, NY, USA: ACM, 2012, pp. 379–384. [Online]. Available: http://doi.acm.org/10.1145/2333660.2333747

[8] R. Leupers, F. Schirrmeister, G. Martin, T. Kogel, R. Plyaskin, A. Herkersdorf, and M. Vaupel, "Virtual platforms: Breaking new grounds," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, March 2012, pp. 685–690.

[9] (2014) SystemC official webpage. http://www.accellera.org/home.

[10] L. I. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Phys. D*, vol. 60, no. 1-4, pp. 259–268, Nov. 1992. [Online]. Available: http://dx.doi.org/10.1016/0167-2789(92)90242-F

[11] (2014) Center of domain-specific computing, riciandenoise: 2d and 3d total variation based rician denoising. http://cdsc-image-processing-pipeline.googlecode.com/files/riciandenoise.pdf.

[12] (2011) SoCLib tlm2.0 library website. http://www.soclib.fr.

[13] (2006) Brainweb: Simulated brain database. http://brainweb.bic.mni.mcgill.ca/brainweb/.