Performance and Energy Efficiency Analysis of a Reverse Time Migration Design on FPGA

João Carlos Bittencourt[*], Joaquim Oliveira[†], Anderson Nascimento[†], Rodrigo Tutu[†], Lauê Jesus[†], Georgina Rojas[†], Deusdete Matos[†], Leonardo Fialho[†], André Lima[†], Erick Nascimento[†], João Marcelo Souza[†], Adhvan Furtado[†], and Wagner Oliveira[‡]
[*]Center of Technology and Exact Sciences, Federal University of Recôncavo da Bahia, Cruz das Almas, Brazil
[‡]Politechnical Institute, Federal University of Bahia, Salvador, Brazil
[†]Super Computing Center SENAI/CIMATEC, Salvador, Brazil
Email: adhvan@fieb.org.br

***Abstract*—Reverse time migration (RTM) modeling is a computationally and data intensive component in the seismic processing workflow of oil and gas exploration.Therefore, the computational kernels of the RTM algorithms need to access a large range of memory locations. However, most of these accesses result in cache misses, degrading the overall system performance. GPU and FPGA are the two endpoints in the spectrum of acceleration platforms, since both can achieve better performance in comparison to CPU on several high-performance applications. Recent literature highlights FPGA better energy efficiency when compared to GPU. The present work proposes a FPGA accelerated platform prototype targeting the computation of the RTM algorithm on an HPC environment. Experimental results highlight that speedups of 112x can be achieved, when compared to a sequential execution on CPU. When compared to a GPU, the energy consumption has been reduced up to 55%.**

***Index Terms*—RTM, Oil and Gas Exploration, FPGA, Performance Analysis, Heterogeneous Systems**

I. Introduction

Reconfigurable systems are becoming a key component in heterogeneous high-performance computing systems. Reconfiguration increases the computing flexibility for a given algorithm, allowing for different solutions to be deployed and tested on-the-fly. This ability is specially useful for systems that require a combination of high processing power with remote reconfiguration at circuit level.

Reconfigurable systems normally use highly flexible computing fabric, usually Field-Programmable Gate Arrays (FPGA). Nowadays, FPGA devices are adopted in a wide spectrum of applications, ranging from consumer electronics to critical systems such as vehicle automation or mobile network base station transceivers [1], [2]. More recently, FPGAs are being embedded into HPC cluster nodes starting a new revolution in high performance and cloud computing [3].

Seismic imaging algorithms for oil exploration, such as Reverse Time Migration (RTM), are a time consuming and memory bound class of application that usually demand large amounts of computational power. As the easy-to-exploit reserves are depleted, more accurate imaging methods are increasingly relevant in order to achieve better picture of the seabed complex, allowing for proper planning and efficient exploration of new reserves on complex geological surfaces.

Although current computing capabilities meet the requirements for RTM and Full Wave Inversion (FWI) algorithms on production environments, the cost of processing large amounts of data is high and must be taken into account. Efforts targeting the reduction of these costs by using different acceleration strategies are needed. Since the RTM algorithm uses Finite Differences (FD) method, any advancement in accelerating FD kernels may reduce design time of third-party solutions.

In the last decade, GPU devices have been widely used as the main alternative for acceleration in supercomputer environments [4], [5]. They can be categorized as loosely coupled accelerators as they are commonly attached to the host system through PCIe interfaces. System designs targeting GPUs are extremely flexible, when compared to HDL-based ones used for FPGAs. Modern frameworks, such as CUDA and OpenCL, provide high-level hardware abstraction layers that allow developers to implement solutions using C-like languages, while also hiding low-level implementation details.

Recent studies highlight that, in order to develop faster and more energy-efficient architectures, algorithmic evaluation should focus on optimized memory organization, set of operators, and interconnection topology [6]. Modern FPGA devices provide higher performance for dedicated application-specific circuits as well as hardware reconfiguration, providing design adaptability towards new features when needed. The use of FPGA-based hardware accelerators in critical HPC applications has shown promising results [7], [8]. Recent works highlight not only how fast they are, but also its reduced cost and better energy efficiency when compared to CPU/GPU counterparts [9]. Despite of this, FPGAs are still not the main hardware acceleration method used in these applications. The FPGA design flow requires modeling, simulating, debugging and synthesizing a processing architecture. Unlike other general processing units design flow, FPGA designs require specialized skills to deal with synchronization problems and bit-wise operations, among others hardware issues.

The present work describes the development and validation of a HPC FPGA acceleration platform for RTM. The system is a hardware/software co-design that provides an scalable model for seismic imaging on HPC environments. The main goal of this paper is to provide information about the accelerator ar-

978-1-7281-5999-7/19/$31.00 ©2019 IEEE
DOI 10.1109/H2RC49586.2019.00012

chitecture considering the computational challenges in seismic imaging for oil exploration. We use a custom structure for 2-D RTM computation, which optimizes memory bandwidth and explores the FPGA through a pipelined architecture. Our design provides a high performance and energy efficient alternative for accelerating seismic imaging processes through a combination of state of art RTM techniques [10], [11] and hardware-oriented optimizations [12], [13], [14]. The hardware prototype was developed for evaluation using an Intel Arria 10 GX FPGA Development Kit attached to an HPC node equipped with two Intel Xeon Gold 6148.

II. Reverse Time Migration

Reverse Time Migration is a computationally intensive component in the seismic processing workflow of oil and gas exploration, posing several computational challenges such as high bandwidth and large storage requirements. Traditional RTM algorithms demand the manipulation of a large amount of data in both memory and disks. Therefore, the computational kernels of an RTM algorithm need to access a large range of memory locations for each point being calculated. Most memory access attempts results in cache misses which degrades the overall system performance. To overcome such high demand for memory access in FPGA acceleration platforms, both architectural and algorithmic optimizations are necessary for better performance results.

The RTM algorithms was initially introduced as a migration method that uses the full wave equation. In the specific case of reverse time extrapolation, the seismogram is used as contour condition in the geologic model surface. The conventional RTM algorithm has three steps for each shot: (1) forward time extrapolation of the source wavefield $P_s(z,x,t)$; (2) backward time extrapolation of the receiver wavefield $P_r(z,x,t)$, where the shot seismogram is injected in the receiver positions; and (3) assessment of the image condition that shows where the reflections occurred.

The forward time extrapolation wavefield is defined by the finite-difference method presented in Eq. 1.

$$\begin{gathered}\frac{P^{n+1}_{i,j} - 2P^{n}_{i,j} + P^{n-1}_{i,j}}{\Delta t^2} = c^2 D_k(P^n_{i,j}) + f^n(z_s, x_s)\\ P^{n+1}_{i,j} = 2P^n_{i,j} - P^{n-1}_{i,j} + c^2\Delta t^2 D_k(P^n_{i,j}) + f^n(z_s,x_s)\end{gathered} \tag{1}$$

The receiver wavefield $P_r(z,x,t)$ is extrapolated by applying the Eq. 1, but backward in time, as depicted in Eq. 2.

$$P^{n-1}_{i,j} = 2P^n_{i,j} - P^{n+1}_{i,j} + c^2\Delta t^2 D_k(P^n_{i,j}) + f^n(z_r, \mathbf{x_r}) \tag{2}$$

where $f^n(z_r, x_r) = S_r d^k_{obs}$, and with d^k_{obs} the k-th shot and S_r is an operator that inserts the seismogram line at the receiver positions. The seismogram is inserted, slice by slice, at the surface, from time equal to record length to time zero.

For every shot, the resulting image is usually built by cross-correlation of the imaging condition, as depicted in Eq. 3

$$I_s(z,x) = \sum_{t=0}^{T} P_s(z,x,t)P_r(z,x,t) \tag{3}$$

The final image is the sum of the images of all shots (Eq. 4).

$$I_s(z,x) = \sum_{s=1}^{n_s} I_s(z,x) \tag{4}$$

Fields P_s and P_r, ranging from $t = T$ to $t = 0$, have maximum cross-correlation in the reflectors positions. Therefore, Eq. 3 is solved during the P_r extrapolation. Different strategies may be used in order to obtain P_s wavefield, some of them are discussed in [15]. Thus, one can store all time levels of the source wavefield P_s, and access them as needed during the backwards loop. This is perhaps the most intuitive solution – it requires only one forward modeling, but for large problems the required storage is such that it does not fit into the memory of typical modern computers (even with domain decomposition) and disk I/O operations become necessary. It is also possible to store every *n-th* step of the time history (thereby taking advantage of the time oversampling in the simulation) and estimate intermediate time levels by interpolation as needed during the backwards recursion. The associated computational cost is slightly higher than that of the previous case due to the interpolation, but the scheme results in a *n*-fold reduction in memory occupation. Further reduction in memory requirements can be obtained by compressing the records. Such a methodology is presented by the checkpoint technique proposed in [16].

The saving boundary technique stores the source wavefield P_s only in the model boundary for all time levels and then computes the wavefield at the points inside the mesh using the values saved from the edges as boundary conditions [17]. This technique requires to solve three modeling problems (two for source and one for receiver wavefield) to migrate each shot. Therefore, [10] presents an alternative based on non-coherent correlation of random events. In such an approach, the source propagation is done with wavefield that encounters the model boundary region pseudo-randomized. The forward propagation is done up to the maximum time by saving only the last two snapshots, and then the source wavefield is reverse-time extrapolated simultaneously with the wavefield of the receivers with Eq. 2. This method greatly reduces the memory requirements. Both in the forward and in the reverse propagation, the boundary region of the model are generated at each moment of time for the source wavefield. These boundaries conditions are used to propagate the source in RTM both forward and backwards. The distorted wavefield correlates poorly with the receiver wavefield, minimizing boundary artifacts [10].

A method based in two different boundary conditions is proposed in [14], which use an absorbing boundary condition on the upper boundary, to reconstruct the shot wavefield. Such new schemes could thus solve the free surface boundary problem and would not demand much memory. Both techniques presented by [10] and [14] present promising solution regarding reducing the memory demand of RTM algorithms. The hardware aspects of such implementations are discussed in the following section.

A. *Hardware Implementation Aspects*

The most significant effort toward hardware acceleration of RTM algorithm optimization is reducing the storage requirements. Towards this, the RTM models proposed in [10], [14] indicate the realization of the both reverse propagation using the final snapshot and random boundary as the most suited methodologies in order to reduce such storage requirements. The random boundary fundamentals are based on one copy of the source and receiver wavefields. Since only one copy of the source and receiver wavefields need to be stored into a memory system, such a technique becomes suitable for encapsulating the entire RTM computation into a single FPGA accelerator instance. This is particularly desirable to avoid external communication when processing a single shot and increase accelerator performance by scaling its computation.

The main disadvantage of random boundary lies in the need for the Free Surface Boundary Condition (FSBC), since the Random Boundary Condition (RBC) induces severe noise along the imaging profile at the surface boundary. FSBC is also harmful since the reflections from the surface will generate imaging illusions.

The method presented in [14] proposes a novel boundary condition to reconstruct the shot wavefield called Hybrid Boundary Condition (HBC), which adds an absorbing boundary condition to the upper boundary. The proposed scheme aims to solve the free surface boundary problem without increasing memory storage requirements. This hybrid methodology requires saving the upper boundary of the source wavefield such as the work presented in [15] which proposed saving all boundaries. The saving boundary technique stores only $d/2$ layers of data at every surrounding boundary, where d is the order of the finite-difference scheme. Additionally, by saving layers of the upper side, it is possible to decrease the computational cost, and this can reduce the overall complexity of its hardware implementation. In order to provide a comparison between RBC and HBC, Table I shows the storage requirements for three synthetic models for both methods against the conventional method, which saves all fields.

Table I
MEMORY REQUIREMENTS FOR RTM STRATEGIES.

Model	Conventional	Random Boundary	Hybrid Boundary
Marmousi	64.4 GB	0.05 GB	0.40 GB
Sigsbee	143.2 GB	0.12 GB	0.47 GB
Pluto	311.4 GB	0.25 GB	1.04 GB

Experimental results also show the effectiveness of the HBC to obtain good images of structures at subsurface, as well as the decrease of random noise present in the final results [14] when compared against the RBC. Therefore, the HBC with random and saving boundary was defined as our system methodology implementation model.

A multiple stencil operator and multiple time step hardware implementation strategy is proposed in [18]. The authors employ a deep pipelined approach over successive iterations to achieve linear scalability for multiple devices with constant memory bandwidth. Therefore, the hardware structure presented in [18] is used to specify the proposed micro-architecture design. Such a scalable structure is presented in Figure 1. The PSM (*Pipelined Stage Module*) is responsible for processing one time step. Each PSM receives a previous time step data from the previous PSM as well as forwards the processed data to the next PSM. This structure allows for less external memory access since the pressure wavefields P_0 and P_1 needed to calculate P_2 are produced inside the PSM.

The structure presented in Figure 1 is used for the stencil computation. However, the RTM computation requires extra steps which have to access the external memory in order to load data which cannot be generated internally as well as store intermediary results. Thus, a custom hardware structure is herein proposed taking into account the memory bandwidth limitations. Such a structure is organized in two steps: (i) store the data from the latest PSM into the external memory; and (ii) read and pass through the stored data to the first PSM. This operation is performed until the last time step computation.

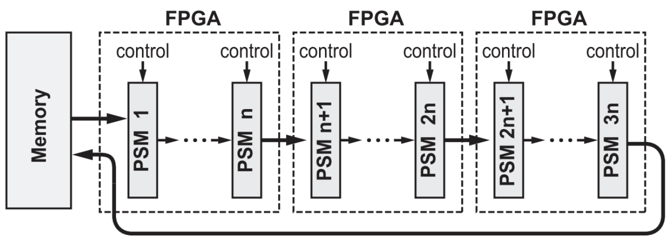


Figure 1. Hardware structure for stencil computation.

Each PSM module is capable of handling more than one type of RTM model. Hence, the proposed architecture considers programmable micro-operations through domain-specific programmability by using hardware and software layers. The PSM structure is composed of several Processing Elements (PEs) responsible for the actual RTM computation. The instructions stored in a programmable memory are responsible for conduct the operations performed inside the PEs. Furthermore, each PE is allocated to a configurable set of grid points.

The structure depicted in Figure 1 is composed of separated instruction and data memories, and a control module responsible for managing the communication between the PSMs and internal memories. The instruction memory keeps track of the instructions which will execute through the PSMs. On the other hand, the data memory is used as a cache from the external memory point of view, being a shared data memory between the PSMs.

Several implementations present in the literature proposes the use of custom numeric representations in order to achieve a better performance when computing the RTM on FPGA devices [19], [20], [21], [22]. Reducing the precision (bit width) has shown to greatly decrease the area costs and optimize the I/O bandwidth. However, it should be emphasized that a change in the numerical representation pattern requires a detailed study of the target algorithm as well as the data characterization [23]. The proposed design implementation uses a 24-bit fixed-point representation in order to manage the internal processing operations. Herein, the fixed-point representation is used in order to optimize the memory bandwidth on FPGA device.

III. Design Architecture

The reference application for the RTM method was designed using C language for both migration and modeling steps. The algorithm was designed from the Hybrid Boundary Condition [14] methodology since this strategy better fits the hardware needs without losing quality in final results. The workload and data structures management were built to be easily adapted to other processing units, such as FPGA and GPU, and to use different levels and methodologies to explore parallelism. It has been used the open source Seismic Unix utility, supported by the Center for Wave Phenomena (CWP) which provides a standard environment for seismic applications.

The proposed acceleration solution has two components: an FPGA Accelerated Functional Unit kernel (RTM FPGA IP-core) and a host user application (RTM Host Application). The following sections present the base model intended to running in software/IP-core co-design, including mechanisms to provide an effective FPGA resource/functionality exploitation in order to achieve better performance and mitigate computational challenges of the RTM algorithm.

A. RTM FPGA IP-core

The system prototype is composed of a computing node with two multi-core Intel Gold 6148 processors and an Intel Arria 10 GX FPGA Development Kit board. The system component set, presented in Figure 2, is composed by the RTM Core, an Avalon-MM Interconnection Fabric embedded with its standard components, an External DDR4 Memory Controller, and a PCIe controller. The Avalon-MM Interconnection Fabric has been used to provide a unified interface for the different IP-cores presented in Figure 2, besides their standard control and interface cores. The PCIe controller and PCIe interface components are responsible for providing the PCIe functionality, allowing the communication between the FPGA and the host computer. The DDR4 controller and DDR4 interface provide the necessary resources for the communication between the Intel Arria 10 GX FPGA Development Kit and the DDR4 external memory. The interface IP-cores depicted in Figure 2 were obtained through Intel IP database. The RTM core component represents the main result of this project by the hardware implementation of the hybrid boundary RTM functionality. Finally, the interface between the IP-cores has been implemented by the Intel FPGA Platform Designer.

The Figure 3 presents an overview of the RTM IP-core components designed in the present HDL code. It highlights the main components of the accelerated platform. The RTM core corresponds to the implementation of the RTM processing element on the target FPGA device. This core is composed

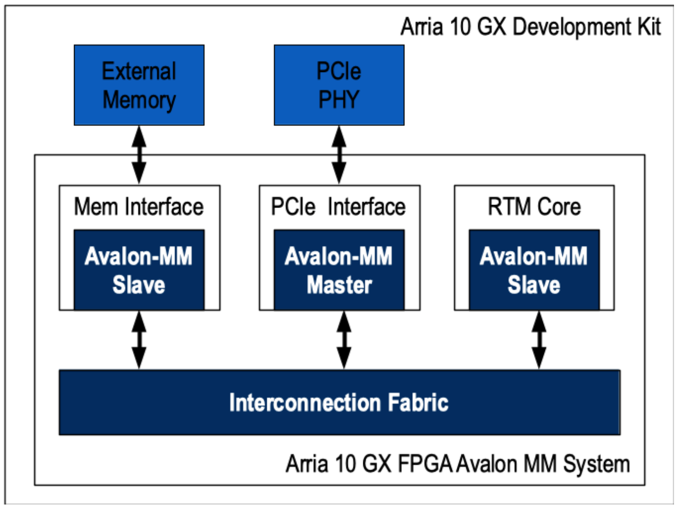


Figure 2. Arria 10 GX FPGA Development Kit conceptual design architecture and interfaces.

by three functional components: (1) a Main Control System; (2) the Scalable Streaming Array (SSA) of Pipelined Stage Modules (PSM); and (3) a Cross-correlation mechanism.

The SSA is responsible for processing the time steps of the stencil computation. Each PSM subcomponent receives a previous time step data from the previous PSM, and forwards the processed data to the next PSM. The cross-correlation mechanism performs the computation needed to assess the resulting image quality. Such computation is based in a cross-correlation of the source and receiver wavefields based in reflectors located in the correlated subsurface.

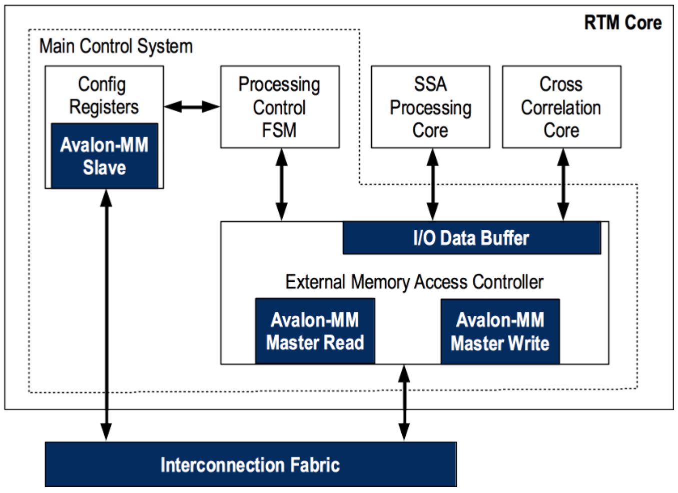


Figure 3. RTM Core main components and interface logical structure.

The Main Control System controls the internal data flow and external memory access through an Avalon control interface through the configuration registers and the External Memory Access Interface (EMIF). The processing control FSM controls the internal data flow and the external memory access through the Avalon-MM EMIF. The operation begins when the host sends the initialization signal, causing specific sub-modules of the main control system to start their operation by reading the memory data. This data is later stored into several internal I/O data buffers and sent to the PSM units inside the SSA and the cross-correlation components. At the same time, when a PSM or cross-correlation module need to write their data into the device memory, the I/O data buffers provide a mechanism for data to be written and to be sent to memory after that.

The SSA processing core is the main processing unit of the RTM Core, responsible for performing the time steps of the stencil computation. Such a processing is accomplished through several consecutive PSM, which compute the forward and backward propagation as well as the image reconstruction of the RTM algorithm. When operating in forward propagation mode, the SSA module can process up to N time steps, where N is the number of PSM units, starting with the time step $T = t+2$. By processing the time steps from $t+2$ to $t+2+N-1$ in a pipeline fashion, the PSM module calculates $t+2+N$. After the computation of the source wavefield time steps, the data values for $T = t+2+N$ and $T = t+2+N+1$ are stored into the platform memory, in order to calculate the following time steps starting from $T = t+2+N+2$ and calculating the following time steps up to $t+2+N+2+N-1$, until the SSA module finishes the calculation of all time steps. In the reverse propagation, the PSM units are divided into two groups of $N/2$. The PSM units ranging from 0 to $(\frac{N}{2}-1)$ are responsible for calculating the reverse propagation and the PSM units ranging from $\frac{N}{2} toN-1$ are responsible for the reconstruction of the propagated signal. The results of such calculations are forwarded to the correlation module for the image condition evaluation process.

The cross-correlation core performs the computation needed to asset the resulting image quality. Such a computation is carried out by the cross-correlation of the source and receiver wavefields based in reflectors located on the subsurface. The RTM is based on the computation of a full wave equation and uses the condition of cross-correlation image. Such an image condition establishes that reflectors are located at points in the subsurface where the wavefield of the source coincides in time and space with the receiver wavefield. Thus, the source wavefield is propagated, the receiver field is reverse-time extrapolated, and the cross-correlation between these fields is calculated. At the points where the fields coincide, the image condition will be a non-zero indicating the presence of a reflector in that position. The correlation module is responsible for the cross-correlation of the image condition. The correlation process is defined by the Equation 5 in each shot.

$$I_s(z,x) = \sum_{t=0}^{T} P_s(z,x,t)P_r(z,x,t) \tag{5}$$

where P_s is the source wavefield in forward propagation, and P_r the receiver wavefield in backward propagation. The final image is given by the sum of the images of all single shots.

B. RTM Host Application

The host system can be seen as a regular general-purpose computer system embedded with its own memory hierarchy and a multi-core CPU. An RTM Host Application (RTM-HA) runs on top of a GNU/Linux operating system, which provides built-in API for internal memory management and peripheral device access. The FPGA platform is seen as a hardware co-processor that receives input data and runtime parameters from the host system, processes seismic migration, and then sends back the resulting output data. There is no automatic memory coherence scheme between host and the FPGA platform. Therefore, every data transfer is explicitly handled by the user space application using the PCIe Communication Framework. The Figure 4 summarizes the general architecture emphasizing the conceptual division between the host and FPGA domains.

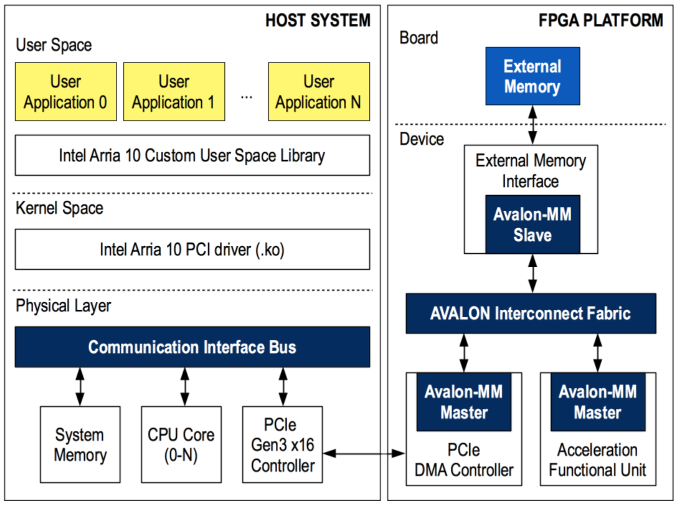


Figure 4. System integration architecture.

The PCIe communication framework includes a set of kernel modules and user space libraries on the host side, along with an embedded DMA controller on the FPGA side. It allows efficient data transfers between host and FPGA through the standard PCIe interface. The hardware implementation details are abstracted from host by using a configuration memory mapping, which points to the internal configuration registers and memory banks into a 64-bit PCIe address space, visible to user space. All data transfers are performed by the FPGA embedded DMA Controller, whose configuration is handled by an FPGA device driver on host machine (so called arriapci.ko).

The software integrates three distinct modules, each one responsible for performing an application domain: (i) data processing (pre- and post-migration); (ii) communication with the RTM FPGA IP-core; and (iii) image stacking. The data processing module includes functions for converting data from/to the target fixed-point representation and organizing data vectors into the required memory pattern of the RTM core. Internally, the RTM FPGA IP-core uses a 24-bit fixed-point numerical representation in order to achieve higher memory throughput. The RTM-HA thus converts all input data to such representation before sending them to the device. Data vectors are also organized into the memory package model required by the core. Additionally, the output data is converted back to floating-point for a proper image display.

The communication module contains structures and functions to represent and interact with RTM FPGA IP-core. The module maps specific internal registers and memory positions into appropriate C structures that can be easily used by the main user application. It extends the Arria 10 API library including specific details of the RTM FPGA IP-core operation, such as control register addresses, status, and debug signals. Finally, the image stacking module concatenates outputs for all executed seismic shots into a single migrated image of the subsurface. Since the RTM FPGA IP-core is designed to process one shot at a time – such feature allows shot parallelization when multiple FPGA boards are available.

IV. Results

In this section, experimental results for the conceived RTM algorithm hardware acceleration platform are presented. Such results were obtained using the Intel FPGA Quartus Prime Development Suite Pro Edition (v18.0), while the design itself was described using SystemVerilog-HDL language. The hardware platform is an Intel Arria 10 GX FPGA Development Kit, with a single 4 GB DDR4 external memory. The communication is performed through a standard PCIe Generation 3 connector. The system component set is composed by the RTM IP-core itself, an Avalon-MM interconnection fabric embedded with its standard components, an External DDR4 memory controller, and a PCIe controller.

In order to standardize the following results, it was necessary to define a set of design constraints regarding both the surface model and the hardware design itself. The Pluto modelis a synthetic 2-D acoustic model of size 369×375 ($NX \times NZ$). The Pluto model was chosen due to its larger size against other synthetic models in the literature. It is justified by the larger time and complexity of performing a functional verification over such models. By using the Pluto model for validation/debugging of the implemented design, it becomes possible to attest to the reliability of the reconfigured design for larger or even real models.

In order to provide better performance estimates and to guide the hardware synthesis flow, a set of design parameters has been defined. The main parameter is the target delay, which reflects the design maximum operation frequency. Delay/frequency are related and directly affect the performance and synthesis results (mainly used resources). In FPGA designs the maximum delay is derived from the number of Look-Up Tables (LUT) used for a given function, as well as (in some cases) the wire delay associated to the corresponding path. The target delay was set by successively reducing the clock delay from 10 ns until to obtain the lowest possible delay value in a well succeeded implementation. The timing requirements relies on the design implementation itself, as well as the FPGA device limitations in terms of device resource utilization and routability.

For an optimized memory bandwidth, the RTM IP-core uses in fixed-point numerical representation. An optimal word size for the hardware was determined empirically by comparing the outputs of a fixed-point RTM software with its 32-bit floating-point equivalent, used as reference. The goal was to determine the smallest word length with a satisfactory ”migration quality” defined by a group of geophysics on our laboratory. Each comparison generated two numerical results, Signal to Noise Ratio (SNR) and Universal Image Quality Index (IQI) Satisfactory migration images were obtained when SNR values ranged between -10 and -20 dB and with IQI above 70%. Following this method, 24-bit was the smallest length that met the word size requirements.

The platform memory interface uses a 512-bit standard data bus. The Scalable Streaming Array (SSA) is the main processing core and therefore is the main resource consuming component. The SSA is therefore composed of several Pipeline Stage Modules (PSM), responsible for processing the time steps of the stencil computation. Each PSM component receives a previous time step data from the previous PSM, process it through a Processing Element (PE) component, and forwards the processed data to the next PSM. Each PSM receives a 512-bit data set from input data management unit, thus the number of internal PEs is directed related to both the memory data bus size and the data word length. Therefore, the number of PEs was specified as depicted in Equation 6.

$$NPE = \frac{512}{24} = 21,33 \tag{6}$$

From Equation 6, one can conclude that the PSM uses 504 bits of the data memory bus. The number of PSMs within the SSA is a highly dependent parameter of both the FPGA device resources and the related timing. The project was tested on a series of synthesis runs, in order to estimate the amount of logical resources to implement the RTM Core as well as the device routing capability, by exploiting the maximum potential of the FPGA platform. Thus, it was defined that the amount of 16 or 24 PSMs corresponds to an adequate trade-off.

The platform design itself is based on the Arria 10 PCIe Gen3 x8 DMA reference design. The project includes a DMA with an Avalon-MM interface that connects to the PCIe hard IP-core. The reference design also includes Linux software drivers that set up the DMA transfer.

The Table II presents the design performance and energy efficiency results for both FPGA and GPU. The analysis also presents the results for the RTM computation on a Muti-thread environment with 16 nodes and 40 threads per node. The values presented in Table II for the Arria 10 were obtained place & route processing and static timing analysis with software parameters for timing optimization, namely Performance (High effort), after running at the hardware platform. The runtime values also include the time to transfer the data to/from the GPU/FPGA. The present analysis used the Pluto seismic model. Speedup and power measurements considered migrations of four seismic shots evenly spaced on the model surface.

Table II
RTM ACCELERATED PLATFORM PERFORMANCE ANALYSIS AND COMPARISON.

Device	#PSM	FMax (MHz)	Runtime (s)	Speedup	Energy (Wh)
CPU	N/A	N/A	87,495	1	N/A
Multi-thread	N/A	N/A	145.82	**150**	N/A
Titan XP	N/A	1,417	706	**123.9**	36
Arria 10	16	100	1,990	44	44
	16	180	1,125	77.8	26
	24	100	1,363	64.2	31
	24	180	781	**112**	**20**

The main processing core SSA can be configured to operate in three different modes: (i) direct propagation; (ii) reverse propagation; and (iii) image reconstruction. The execution time of all processing steps is determined by the size of the input matrix (NX and NZ values) and the number of iterations (NT). The results presented in Table II reflect the data obtained by means of system runtime, considering the previously highlighted configurations. The results highlight the speedup comparison of the proposed platform design against a fully serial CPU implementation and a GPU optimized model, developed in CUDA, running on a Titan XP. Such performance comparisons were made in relation to the serial model, which presented a runtime of 21,877 seconds.

The resulting operation of each PSM into the SSA processing core is equivalent to one time iteration, whether in direct or reverse propagation or in image reconstruction.

When compared to the serial implementation, speed-ups of about 112x can be achieved by using the proposed FPGA accelerated platform. Regarding the other implementation strategies the GPU is only 9% faster, while the runtime of Multi-thread MPI/OMP is about 26% lower. Although FPGA presents such performance decrease, it runs on around eight times slower frequency.

The FPGA design was synthesised for several configurations, by varying the number of PSMs – meaning the number of simultaneous interactions – and the frequency of operation. The experimental results demonstrate that, by increasing the number of PSMs, a relative speedup of about 30% can be achieved. A similar behavior is highlighted in lower frequenciesSuch results also indicates that higher speedups can be achieved if the design is synthesized for more recent devices such as Intel Spartan 10 FPGA, which has twice the number of logic elements, when compared with Arria 10 devices.

Regarding two implementations with close speedup values (see Table II), namely Titan XP GPU and Arria 10 with 24 PSMs at 180 MHz, the total power measure was generated using the Watts Up Pro Portable Power Meterduring the runtime of both accelerators. The energy consumption were obtained as a function of the migration time and the arithmetic average of all instantaneous samples collected on this period, as described in Equation 7, where T is migration time, $P(i)$ is the *n*th power sample and N is the number of collected samples.. Such analysis considered the entire server where the GPU/FPGA are hosted, using the same host for the GPU and the FPGA cards.

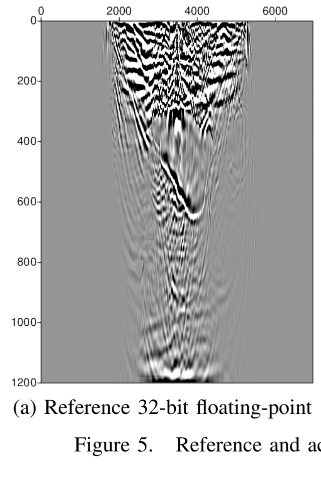
$$P = T/N \sum_{i=1}^{N} (P(i)) \tag{7}$$

The Titan XP GPU had a total power consumption of 36 Wh, while the Arria 10 with 24 PSMs at 180 MHz obtained only 20 Wh. So, the FPGA version is by far more energy-efficient, by consuming about 45% less power when compared to the Titan XP. Considering the speed-up per Watt as an efficiency metric, an efficiency of 5.6 Speedup/Wh can be achieved by using our accelerator solution, against 3.44 Speedup/Wh obtained by Titan XP GPU.

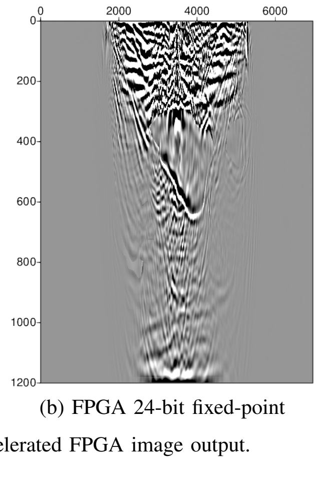
Design verification has been one of the most fundamental challenges of hardware design, and functional verification is the most common method of verifying FPGA-based design functionalities. The RTM-core verification environment is based in a set of Universal Verification Components (UVCs). Each design component has one UVC which emulates its functional behavior. The verification components were described using SystemVerilog-HDL and C languages, following the Universal Verification Methodology (UVM). The compilation/simulation were performed through the Mentor Questa Advanced Simulator tool (v10.6).

In order to verify the platform software interface according to the specifications, components of the FPGA accelerated platform software have been tested and validated using three distinct and complementary strategies, namely: simulation, bench tests and signal probing. In the simulation, the software components have been carried out based on hypothetical data from input files, generating output files, which were compared with expected values from a reference design.

Bench tests (involving the Arria 10 GX FPGA Development Kit), have aimed at validating functions of the communication and image stacking. Initially, memory transfers were made between the host computer and the FPGA in order to guarantee the correctness of the device driver. The bench tests resulted in the images presented in Figure 5.



(a) Reference 32-bit floating-point



(b) FPGA 24-bit fixed-point

Figure 5. Reference and accelerated FPGA image output.

Figure 5a represents the final image obtained by the software reference model using a 32-bit floating-point representation. Figure 5b is the resulting image obtained after processing the Pluto model on the FPGA accelerated platform. Finally, the signal probing tests consisted of sampling and analysis of FPGA internal test points. Such tests were conducted by using the SignalTap II tool of Intel FPGA Quartus Prime Development Suite and consisted in verify both control signals and data buses.

V. Conclusion

This paper described a heterogeneous FPGA-based accelerated platform prototype targeting the computation of the RTM algorithm on an HPC environment. Such a prototype has two separate modules: a host user application and a FPGA Accelerated Functional Unit kernel.

Hardware implementation aspects were correlated with RTM computation optimizations present in the literature, resulting in the proposed hardware/software co-design implementation. The main techniques used were the multiple stencil operator and multiple time step hardware implementation strategy as well as the Hybrid Boundary Condition methodology.

Since the proposed solution is parameterizable for different seismic synthetic models, the paper presented the main features related to configuration parameters and constraints used in this project. The paper also highlighted strategies for static temporal analysis and functional verification of hardware, as well as validation of software components. The exploration of fixed-point computation in geophysical applications on FPGA is novel. Thus, it remains possible to explore such method in other applications, such as in 3-D stencil operations.

The analysis of performance and power consumption, compared against GPU and CPU, highlights that speedups of 112x can be achieved, when compared to a Sequential CPU implementation. Although the design present lower speedup compared to GPU and CPU multi-threaded, our FPGA accelerator achieved better energy efficiency. The power consumption when compared to a GPU has been reduced up to 55% with an efficiency 60% greater.

Since the RTM FPGA IP-core is designed to process one shot at a time in one compute node, the scalability of the solution lies in the parallelization of shots when multiple FPGA boards are available in one or more compute nodes. Therefore higher scalability can be achieved by exploring temporal parallelism, by increasing the number of PSMs and consequently the number of iterations computed in parallel. Additionally spatial parallelism can also be explored by increasing the number os processing elements into each PSM, whith would result in more points being calculated at the same time. Future works aims to explore the above alternatives

References

[1] R. B. Atitallah and K. M. A. Ali, "FPGA-Centric High Performance Embedded Computing: Challenges and Trends," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, aug 2017, pp. 390–395.

[2] C. Sexton, N. J. Kaminski, J. M. Marquez-Barja, N. Marchetti, and L. A. DaSilva, "5G: Adaptable Networks Enabled by Versatile Radio Access Technologies," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 688–720, 2017.

[3] C. Kachris and D. Soudris, "A survey on reconfigurable accelerators for cloud computing," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*. Lausanne, Switzerland: IEEE, aug 2016, pp. 1–10.

[4] B. Zhao, J. Zhong, B. He, Q. Luo, W. Fang, and N. K. Govindaraju, "Gpu-accelerated cloud computing for data-intensive applications," pp. 105–129, 2014.

[5] P. V. Sukharev, N. P. Vasilyev, M. M. Rovnyagin, and M. A. Durnov, "Benchmarking of high performance computing clusters with heterogeneous cpu/gpu architecture," in *Young Researchers in Electrical and Electronic Engineering (EIConRus), 2017 IEEE Conference of Russian*. St. Petersburg, Russia: IEEE, 2017, pp. 574–577.

[6] R. Bittner, E. Ruf, and A. Forin, "Direct GPU/FPGA communication Via PCI express," *Cluster Computing*, vol. 17, no. 2, pp. 339–348, jun 2014.

[7] F. A. Escobar, X. Chang, and C. Valderrama, "Suitability Analysis of FPGAs for Heterogeneous Platforms in HPC," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 600–612, feb 2016.

[8] M. Yoshimi, R. Kudo, Y. Oge, Y. Terada, H. Irie, and T. Yoshinaga, "An FPGA-Based Tightly Coupled Accelerator for Data-Intensive Applications," in *2014 IEEE 8th International Symposium on Embedded Multicore/Manycore SoCs*. Aizu-Wakamatsu, Japan: IEEE, sep 2014, pp. 289–296.

[9] R. Kobayashi, S. Takamaeda-Yamazaki, and K. Kise, "Towards a low-power accelerator of many fpgas for stencil computations," in *2012 Third International Conference on Networking and Computing*. Okinawa, Japan: IEEE, Dec 2012, pp. 343–349.

[10] R. G. Clapp, *Reverse time migration with random boundaries*. Huston, USA: Society of Exploration Geophysicists, 2009, pp. 2809–2813.

[11] B. D. Nguyen and G. A. McMechan, "Five ways to avoid storing source wavefield snapshots in 2d elastic prestack reverse time migration," *GEOPHYSICS*, vol. 80, no. 1, pp. S1–S18, 2015.

[12] R. Cattaneo, G. Natale, C. Sicignano, D. Sciuto, and M. D. Santambrogio, "On how to accelerate iterative stencil loops: A scalable streaming-based approach," *ACM Trans. Archit. Code Optim.*, vol. 12, no. 4, pp. 53:1–53:26, Dec. 2015.

[13] P. Yang, J. Gao, and B. Wang, "Rtm using effective boundary saving: A staggered grid gpu implementation," *Computers & Geosciences*, vol. 68, no. Supplement C, pp. 64 – 72, 2014.

[14] H. Liu, R. Ding, L. Liu, and H. Liu, "Wavefield reconstruction methods for reverse time migration," *Journal of Geophysics and Engineering*, vol. 10, no. 1, p. 015004, 2013.

[15] E. Dussaud, W. W. Symes, P. Williamson, L. Lemaistre, P. Singer, B. Denel, and A. Cherrett, *Computational strategies for reverse-time migration*. Las Vegas, CA, USA: Society of Exploration Geophysicists, 2008, pp. 2267–2271.

[16] W. W. Symes, "Reverse time migration with optimal checkpointing," *GEOPHYSICS*, vol. 72, no. 5, pp. SM213–SM221, 2007.

[17] R. Clapp, "Reverse time migration: Saving the boundaries," Stanford Exploration Project, Technical Report SEP-136, Tech. Rep., 2008.

[18] K. Sano, Y. Hatsuda, and S. Yamamoto, "Multi-fpga accelerator for scalable stencil computation with constant memory bandwidth," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 695–705, March 2014.

[19] R. de Bragança, R. Silva, and M. E. Lima et al., *Seismic Modeling and RTM Migration on unconventional Hardware*. Rio de Janeiro, Brazil: Society of Exploration Geophysicists, 2014, pp. 1357–1361.

[20] M. Araya-Polo, J. Cabezas, M. Hanzich, M. Pericas, F. Rubio, I. Gelado, M. Shafiq, E. Morancho, N. Navarro, E. Ayguade, J. M. Cela, and M. Valero, "Assessing accelerator-based hpc reverse time migration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 1, pp. 147–162, Jan 2011.

[21] R. G. Clapp, H. Fu, and O. Lindtjorn, "Selecting the right hardware for reverse time migration," *The Leading Edge*, vol. 29, no. 1, pp. 48–58, 2010.

[22] H. Fu, W. Osborne, R. Clapp, O. Mencer, and W. Luk, "Accelerating seismic computations using customized number representations on fpgas," *EURASIP Journal on Embedded Systems*, vol. 2009, no. 1, p. 382983, Dec 2008.

[23] H. Fu, “Application-specific number representation,” Ph.D. dissertation, Imperial College London, 2 2009.