Towards a Multi-GPU Implementation of a Seismic Application

Pedro H. C. Rigon[1,2](✉), Brenda S. Schussler[1,2], Edson L. Padoin[1,2], Arthur F. Lorenzon[1,2], Alexandre Carissimi[1,2], and Philippe O. A. Navaux[1,2]

[1] Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
{phcrigon,bsschussler,aflorenzon,asc,navaux}@inf.ufrgs.br, padoin@unijui.edu.br

[2] Regional University of Northwestern Rio Grande do Sul, Porto Alegre, Brazil

Abstract. This study explores the implementation and analysis of a Multi-GPU system for the application of the Fletcher Method in geophysical exploration, essential in the discovery and extraction of energy sources such as oil and gas. The scalability of the software for the use of multiple GPUs (Graphics Processing Units) allows for improved performance of these applications due to their parallel processing capacity. The proposed strategy emphasizes a judicious approach to workload division, considering the data location and the GPU's processing capacity. This implementation stands out as the first in the seismic application field to utilize multiple V100 GPUs and assess the impact on performance. The experiments results demonstrated that the proposed Multi-GPU implementation provides significant performance improvements over the Single-GPU version (e.g., 2.77 times using 4 GPUs). Furthermore, the Multi-GPU implementation exhibits linear growth in performance and efficiency as the input grid size increases.

Keywords: Multi-GPU · Fletcher · Performance · GPU · CUDA

1 Introduction

Geophysical exploration methods play a vital role in our society as they enable the discovery of fundamental resources (e.g., oil and gas) that drive the economic development of nations. However, pursuing new oil reservoirs usually involves destructive practices like drilling in environmentally sensitive areas and improper waste disposal. Hence, researchers have developed applications that simulate seismic imaging for oil detection to mitigate these adverse effects and enhance drilling precision. On top of that, given that these applications involve a huge amount of data and naturally lend themselves to parallel processing, graphics processing units (GPUs) have become extensively employed to accelerate such tasks [Lukawski et al., 2014].

GPUs are architectures designed with a single instruction, multiple data (SIMD) approach and incorporate thousands of processing cores. This design makes them well-suited as accelerator devices for executing applications that efficiently handle array and matrix data structures. However, despite their impressive computing capabilities, GPUs demand significant power during their operation. Consequently, optimizing the utilization of the available hardware resources on GPUs, such as cores and memory, becomes imperative when executing parallel applications [Lorenzon and Beck Filho, 2019]. By doing this, we can effectively reduce energy consumption while mitigating the associated environmental and economic impacts [Navaux et al., 2023].

With the increasing availability of multiple GPUs in high-performance servers, one can further explore the processing potential through Multi-GPU systems [Papadrakakis et al., 2011]. In this scenario, by distributing the workload among all the GPUs available in the system, one can take advantage of the parallel processing power of each one, achieving significant performance improvements [Liu et al., 2019]. Furthermore, the improvement in scalability provided by multiple GPUs allows the handling of increasingly larger datasets, improving the analysis capacity and quality of the generated seismic images through the greater density of data incorporated in the final result.

However, effectively implementing seismic applications that can fully leverage the processing power of Multi-GPU systems poses a significant challenge. A key obstacle lies in achieving efficient utilization of GPUs. Therefore, striking a balance in workload distribution across the GPUs becomes crucial to optimize the available processing power. Additionally, employing efficient strategies for workload partitioning and thread coordination is vital to prevent resource underutilization or overload on the GPUs. By addressing these challenges, one can maximize the efficiency and performance of parallel applications on Multi-GPU systems.

Considering the aforementioned scenario, we propose a Multi-GPU implementation for the Fletcher Method. Our main objective is to provide a workload division strategy that considers fundamental aspects such as data locality and maximizes GPU processing capacity. To validate the proposed implementation, we performed extensive experiments using twenty-nine different grid input sets on a system with eight GPUs. With that, we can verify the performance gains and energy consumption reductions a Multi-GPU implementation provides as the grid input set changes.

Through the experiments, we demonstrate that the proposed Multi-GPU implementation can provide significant performance improvements over the Single-GPU version (e.g., 2.77 times using 4 GPUs). Moreover, the results indicate that more GPUs are associated with greater throughput, highlighting scalability as a critical aspect of optimizing performance in this application. We also show that the performance and efficiency of multi-GPU implementations are directly proportional to the size of the input grid. However, it is essential to highlight that for smaller input grids, the multi-GPU performance is degraded due to the cost of inter-GPU synchronization and data communication, characteristics inherent to these implementations.

The remainder of this paper is organized as follows. In Sect. 2 we describe the Fletcher model and list the Related Work. In Sect. 3, the proposed Multi-GPU implementation is discussed. The methodology followed during the experiments is described in Sect. 4. Performance and power demand results are discussed in Sect. 5 while the final considerations are drawn in Sect. 6.

2 Background and Related Work

2.1 Fletcher Modeling

Fletcher modeling works as a technique for simulating wave propagation over time. This propagation is expressed through the acoustic Eq. (1), where the velocity varies according to the specific geological layers (Eq. 2). Referring to the equations, $p(x, y, z, t)$ indicates the pressure at each location in the domain with respect to time, $V(x, y, z)$ is a representation of the propagation velocity, and $\rho(x, y, z)$ reflects the density [Fletcher et al., 2009].

$$\frac{1}{V^2}\frac{\partial^2 p}{\partial t^2} = \nabla^2 p \tag{1}$$

$$\frac{1}{V^2}\frac{\partial^2 p}{\partial t^2} = \nabla^2 p - \frac{\nabla \rho}{\rho} \cdot \nabla p \tag{2}$$

Seismic modeling initializes by collecting data in a seismic survey, as illustrated in (Fig. 1). The procedure begins with equipment attached to a ship, which at regular intervals emits seismic waves that reflect and refract in interactions with different environmental undergrounds, working as a sonar to map geological structures. When these waves return to the ocean's surface, specific sensors installed on cables towed by the ship capture and record seismic variations. These variations, a.k.a. seismic traces, correspond to the set of signals obtained by each sensor during the wave emission. Therefore, with each emission of waves, the seismic traces of all the microphones on the cable are recorded, providing an understandable overview of the subsoil. During this operation, the ship continues to move and emit signals periodically, thus producing a detailed image of the seabed and underground [Chu et al., 2011].

The Fletcher method models the acoustic wave propagation in a Tilted Transversely Isotropic (TTI) environment through a three-dimensional grid, in which the size of each dimension (x,y, and z) is defined by the variables sx, sy, and sz, respectively. Each point on this grid represents a point in the physical environment being modeled, and this point is associated with physical characteristics such as pressure, density, and wave velocity. In the case of TTI media, the wave velocity varies depending on the direction. That is, each point has an associated slope direction.

We illustrate the single-GPU implementation of the Fletcher method in the Algorithm 1. It requires as input the following parameters: the number of iterations the wave will propagate ($endTime$), a value that defines the period in which the state of the wave will be stored in the disk ($threshToWriteWave$), and the

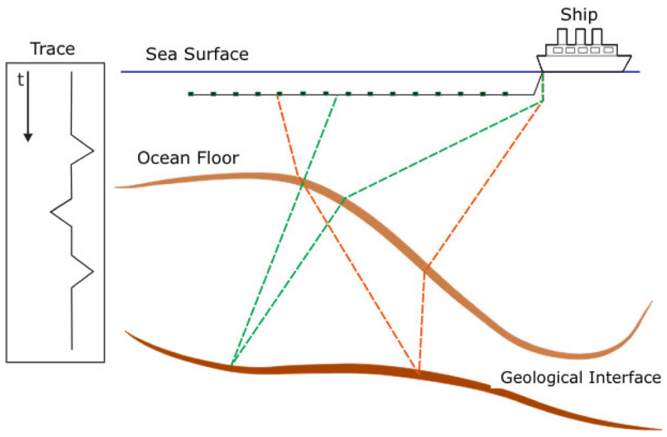

Fig. 1. Data collection in a marine seismic survey

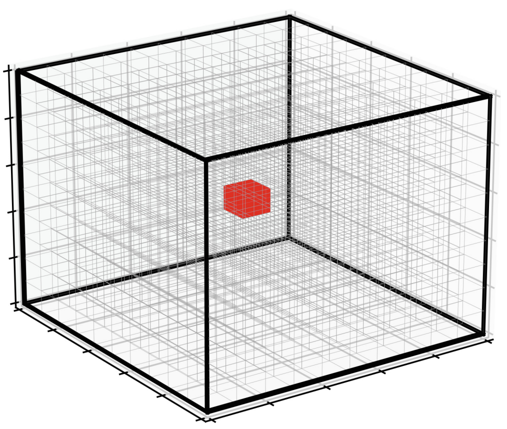

Fig. 2. The pressure point inserted (in red) at the center of the three-dimensional pressure vector. (Color figure online)

dimensions of the grid (*sx, sy,* and *sz*). The procedure starts by initializing the grid with the physical characteristics of the environment through the *initialize-Grid()* function. Initially, a pressure point that represents the amplitude of the seismic wave for a given instant is inserted at the central position of the three-dimensional pressure vector (Fig. 2). Then, before the kernel starts the execution on the GPU, this three-dimensional array is mapped to a one-dimensional array, following the traditional (x, y *and* z) order. This means that the points x of the same line are mapped contiguously in the one-dimensional vector resulting from the mapping.

Algorithm 1. Fletcher: Single-GPU Implementation

```
Input: $endTime$: number of iterations the wave will propagate.
    $threshToWriteWave$: number of iterations where the wave will be stored in disk.
    $sx$: size of dimension x.
    $sy$: size of dimension y.
    $sz$: size of dimension z.
1: $initializeGrid(grid, sx, sy, sz)$
2: $initPropagatePointers(grid, initPoint)$
3: $allocateDataDevice()$
4: $copyDataToDevice()$
5: $calculateExecutionConfiguration(blocks, threadsPerBlock, sx, sy, sz)$.
6: for each $dt$ in $endTime$ do
7:     $insertSourcePointToDevice()$
8:     $kernelPropagate <<< blocks, threadsPerBlock >>> (...)$
9:     $updatePointers()$
10:    if $dt == threshToWriteWave$ then
11:        $writeWave()$
12:    end if
13: end for
```

The loop from line *6* to *13* is responsible for iterating until the simulation is performed. Then, for each iteration, a modulated Gaussian pulse representing the amplitude of the seismic wave at a given time instant is inserted in the center of the three-dimensional grid (*insertSourcePointToDevice()*). Then, the CUDA Kernel is launched for execution, which will propagate this pressure point in time. The propagation of the seismic wave is based on the computation of a 5-point stencil during the Kernel execution. Stencil computation is a technique that involves computing a center point based on reading neighboring points that are the results of previous kernel computation. This approach is widely used in parallel processing algorithms and [Pearson et al., 2020] image processing. During computation using the Stencil (Fig. 3) technique, there is no dependency between the calculations of individual points, which means that they can be computed independently and in parallel. This property makes processing highly parallelizable, allowing multiple points to be calculated simultaneously, speeding up execution [Pavan et al., 2019].

Once the point associated with the acoustic wave is computed, the wave state is propagated to the previous state to proceed with the next iteration. In this scenario, two buffers are used: *pp* (previous state) and *pc* (current state). The current state of the wave is moved to the *pp* buffer, which now becomes the previous state. At the same time, the newly calculated next state is stored in the *pc* buffer, which now becomes the current state. Furthermore, when the number of iterations reaches a defined threshold, the wave is written to the disk (*writeWave()*).

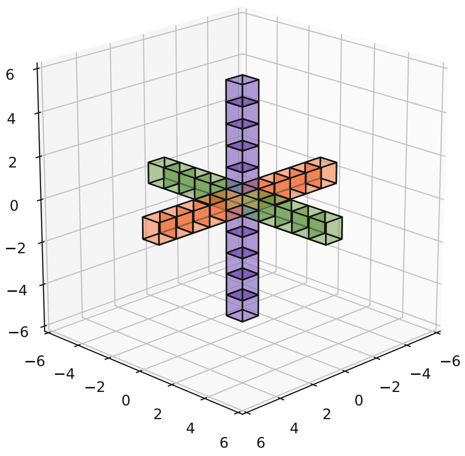

Fig. 3. 5-points 3D Stencil representation

In summary, for each time step, the pressure values at each grid point are updated based on the wave equation and the previous pressure, density, and wave velocity values of the point and its neighbors. To avoid artificial reflections from the border of the grid, which can interfere with the wave propagation characteristic, an absorption zone of 16 points is applied. The seismic waves are artificially damped in this region according to the distance from the inner grid. In this way, the closer to the border of the grid, within the absorption region, the greater the smoothing velocity at these points, so the velocity set at the border is zero.

2.2 Related Work

In this section, we list the works that exploit the parallelism of seismic applications. They are organized in chronological order.

[Liu et al.,2019] explore using GPUs to accelerate the Reverse Time Migration (RTM) algorithm. The parallelization scheme focuses on using two GPUs by employing a workload division strategy. The authors also consider a version that relies on the unified memory scheme available in CUDA. The results suggest that the workload division strategy presents better results than unified memory in a multi-GPU environment. Furthermore, the authors argue that computational efficiency grows linearly with the increase in GPUs. In contrast, our paper extends the scope to use Multi-GPU systems with up to eight GPUs, with a balanced workload distribution approach across all these components.

[Serpa and Mishra, 2022] address optimizing the Fletcher method on multicore and single-GPU architectures focusing on portability. The paper analyzes the performance, energy consumption, and energy efficiency of two versions of the code, an original version and an optimized version for OpenMP, OpenACC, and CUDA. The results indicate that the CUDA version has the best performance and energy efficiency among all evaluated versions. While it focuses on multicore architectures and single-GPU only, our work focuses on Multi-GPU architectures and addresses related topics such as border exchange between algorithm iterations and the performance and energy improvements as the grid size increases.

[Liu et al., 2012] discuss the implementation of the GPU-accelerated RTM algorithm. It also addresses specific issues such as uneven topography and anisotropic environment, i.e., it focuses on various technical aspects of implementing GPU-accelerated RTM. Different from it, our work focuses on the implementation of a Multi-GPU system and on the performance and energy efficiency results.

[Pearson et al., 2020] explore techniques to improve 3D stencil communication on heterogeneous supercomputers using strategies such as hierarchical partitioning and optimization of data exchange between GPUs. Through tests on up to 256 nodes, the authors demonstrate the efficiency of these techniques in improving communication and reducing data exchange time. On the other hand, our work uses concepts of stencil communication to implement Fletcher's method of propagating seismic waves in a Multi-GPU environment in CUDA, exploring aspects such as performance and energy efficiency.

[Okamoto et al., 2010] describe how the use of multiple GPUs can significantly speed up simulations of seismic wave propagation. A particular challenge faced when using multiple GPUs is non-contiguous memory alignment in the overlapping regions between subdomains processed by different GPUs. This can lead to delays in data transfer between the device and the host node. Differently, in our work, we address this scenario and show that with the increase in the grid input size, there is a proportional increase in the synchronization time between the subdomains processed by different GPUs.

3 Multi-GPU Implementation of Fletcher

Given the Single-GPU implementation of the Fletcher method described in Sect. 2, we discuss next the modifications we have done for the Multi-GPU version. Because proper workload distribution and memory management across many GPUs are key performance aspects in Multi-GPU environments, we consider a workload division that can take advantage of (*i*) the intrinsic parallelism available in the calculation of each grid point; (*ii*) the memory management between the GPUs; and (*iii*) the data locality aspect [Padoin et al., 2013].

For the workload distribution, we consider the division of the *z-axis* among all GPUs, as illustrated in Fig. 4 for the distribution across 4 GPUs. This partitioning divides the three-dimensional grid into subdomains in the *z-axis* direction, and each of these subdomains is assigned to a specific GPU for processing. The outcome of this strategy is an even distribution of the workload across the available GPUs. This strategy aims to optimize the data locality since the execution of the CUDA kernel is based on the computation of a 5-point Stencil (Fig. 3). Hence, when mapping the three-dimensional grid to a one-dimensional grid in

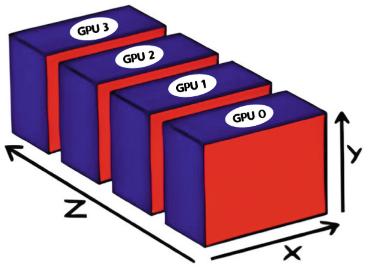

Fig. 4. Workload distribution strategy along z-axis across 4 GPUs

the traditional way (x, y, z), it is essential to note that the neighboring addresses along the x and y axes will be closer in memory than the neighboring points along the z-axis. In this scenario, assigning a continuous block in the z-axis direction to each GPU can increase memory locality on L1 and L2 GPU caches, reducing the need for slower global memory accesses.

Moreover, we explore memory coalescence because GPUs are designed to be more efficient when threads of the same warp access data stored in contiguous memory addresses. Hence, decreasing the distance between the points ensures that memory data access will happen more cohesively. This allows GPU memory reads and writes to be combined into fewer memory transactions, resulting in a more efficient use of memory bandwidth. However, although there is immediate parallelism during kernel execution, it is worth mentioning that each call to the CUDA kernel advances the solution by a single *dt* time step. Hence, to propagate the wave by several time steps, the kernel needs to be iterated, and between these iterations, we must ensure the exchange of information between the different GPUs. This communication occurs through synchronizing and updating variables in the border regions of the computing domain.

For the 5-point Stencil computation, the edges represent intersection zones of the three-dimensional grid data mapped to the GPUs used for processing. These borders are needed to synchronize data across all GPUs, ensuring the correct reading of data throughout the execution of the CUDA kernel. Hence, computing the Stencil requires that each subdomain, present on each GPU, have a 5-point border in the z dimension for each intersection point with the subdomain of other GPUs. Therefore, after the kernel execution, it is necessary to exchange the upper and lower borders across the neighbors' GPUs to propagate the updated data to the next iteration of wave propagation. During this exchange operation, *cudaMemCpyDeviceToDevice* was used to assess the impact of the border exchange considering an indirect communication. When using this function, the communication always passes through the Host.

Moreover, the workload distribution along the *z-axis* also provides benefits in reducing the number of operations performed only to exchange borders through GPUs. Although it is not possible to eliminate all communications, this strategy minimizes the need for inter-GPU communication because threads on each GPU

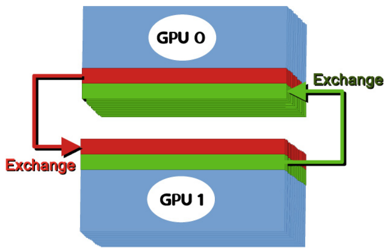

Fig. 5. Border exchange across GPUs along the z-axis.

can process their grid points independently without synchronizing and updating data from other GPUs. In addition, the cost associated with border synchronization reduces since this overhead increases with the growth of the grid size (Fig. 5).

4 Methodology

The experiments were performed on a *p3.16xlarge* AWS instance (Table 1), which is equipped with 64 Intel Xeon E5-2686 v4 (Broadwell) VCPUs, each supporting two threads per core, resulting in a total of 128 available execution threads. In addition, the instance has 8 NVIDIA Tesla V100-SXM2 16Gb GPUs and offers 488 GiB of RAM. Also, the following versions were used: CUDA v.12.0, NVIDIA driver 525.85.12, and gcc 9.4.0 with the −O3 optimization flag.

Table 1. Specifications of the Architecture

Processor Specification	
Processor	Intel Xeon E5-2686 v4
Architecture	Broadwell
Processor/GPU	Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30 GHz, 64 VCPUs
Memory	1 MiB L1d, 1 MiB L1i, 8 MiB L2, 90 MiB L3
GPU Specifications	
GPU	NVIDIA Tesla V100-SXM2
Architecture	Volta
Processor/GPU	GV100
Registers	256 KB/SM, 20480 KB/GPU
Memory	4096-bit HBM2, 16 GB, 6144 KB L2 Cache

We have considered twenty-nine different input grid sizes for the Fletcher method: ranging from 88 to 984 (the maximum size we could allocate in the architecture), in intervals of 32. We have chosen to use 3D input vectors with a dimension multiple of 32 to match the size of the CUDA warps. In this scenario, the following versions were implemented and tested: Single-GPU, 2-GPUs, 4-GPUs, and 8-GPUs, indicating the number of used GPUs.

We compare the Single and Multi-GPU versions regarding performance and energy consumption. The performance is represented by the number of samples computed per second (MSamples/s). We collected the energy consumption via the NVIDIA-SMI command line tool provided by NVIDIA. The results presented in the next section are the average of 10 runs with a 95% confidence interval based on the *Student's t* distribution. In addition, each graph identifies the confidence intervals of the results for each problem size and GPU version.

5 Results

In this section, we present, analyze, and discuss the results obtained from the experiments. First, we discuss the performance of the Multi-GPU implementation of the Fletcher method for 1, 2, 4, and 8 GPUs. The energy efficiency is analyzed, comparing the average power demand (in Watts) of the GPU during the iteration of the CUDA kernel with the performance (in MSamples/s) to assess the energy efficiency of the application.

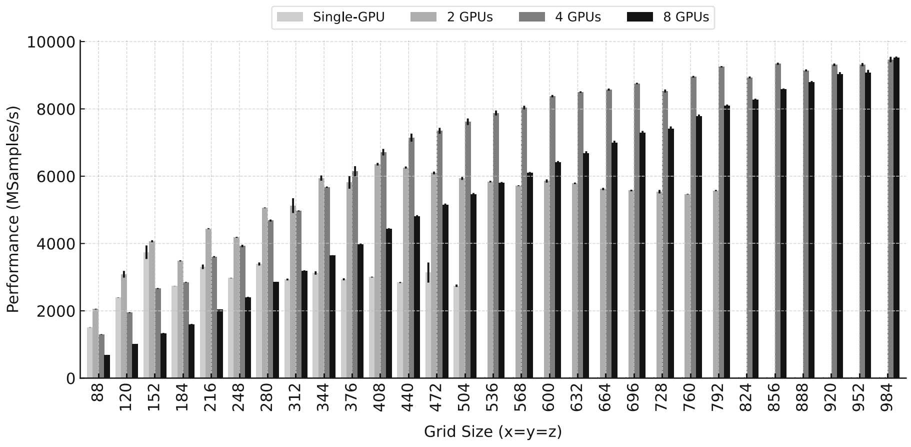

Fig. 6. Performance results for each grid size and implementation. The higher the bar, the better the performance.

Figure 6 shows the performance results for the entire experiment set. It is worth mentioning that the grid set computed on the Single-GPU and 2-GPUs versions is limited by the GPU VRAM, which in this case is 16Gb per Device (NVIDIA Tesla V100-SXM2). Therefore, the first observation is that the Multi-GPU implementation of Fletcher allows the execution of larger grid sizes, improving the capability and quality of seismic images generated through higher data density incorporated into the final result.

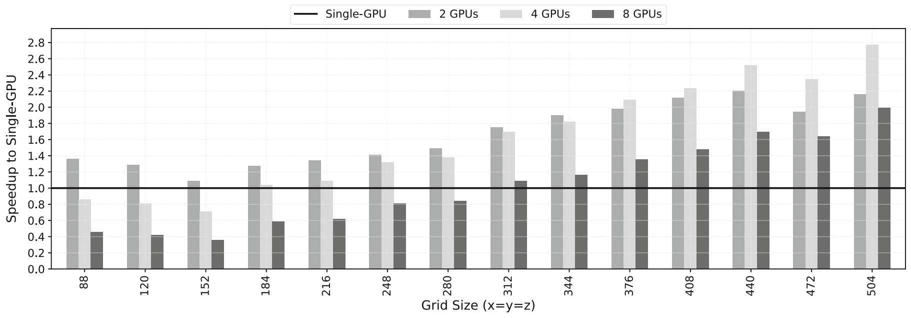

Fig. 7. Performance Speedup over Single-GPU implementation.

By analyzing the behavior of Fig. 7, one can highlight that the performance grows along with the increase in the grid size, allowing the user to increase application throughput. As an example, for a grid size equal to 504, the maximum speedup of 2.77 is achieved over the Single-GPU with 4 GPUs. Moreover, by using 2 GPUs, we observed a speedup over the Single-GPU greater than 2 for grid sizes 376 and larger, indicating efficient scalability for this configuration. However, when increasing the number of GPUs to eight, we could not obtain proportional gains to this increase in computational capacity because of the cost of inter-GPU communication. That is, with a small grid size, parallelism in Kernel computation is not exploited to the maximum, and the cost of border synchronization between the GPU is more significant in relation to the execution time of CUDA kernel computing. Therefore, we argue that the effectiveness of an 8-GPU strategy would require a larger problem, where the cost of communication would have a smaller impact in relation to the throughput gain.

This statement is corroborated by the analysis of Fig. 6, which demonstrates that the performance of multi-GPU implementations exhibits a positive linear trend with increasing input grid size. This implies that the effectiveness of Multi-GPU computing amplifies proportionally to the scale of the problem. Thus, its performance becomes especially notable for large-scale computational tasks, where the ratio between the cost of inter-GPU communication and Kernel CUDA computation is optimized. Additionally, the use of 2 GPUs reaches its Performance peak for input grid sizes close to 408. Afterward, a slight decrease in performance is observed as the grid size increases, until GPU memory capacity (VRAM) limits computation, which it does for input sizes greater than 792.

Figure 8 illustrates the maximum power achieved while running the CUDA kernel for different versions and grid input sizes. The maximum power increases as the number of GPUs also increase. The Single-GPU approach consistently

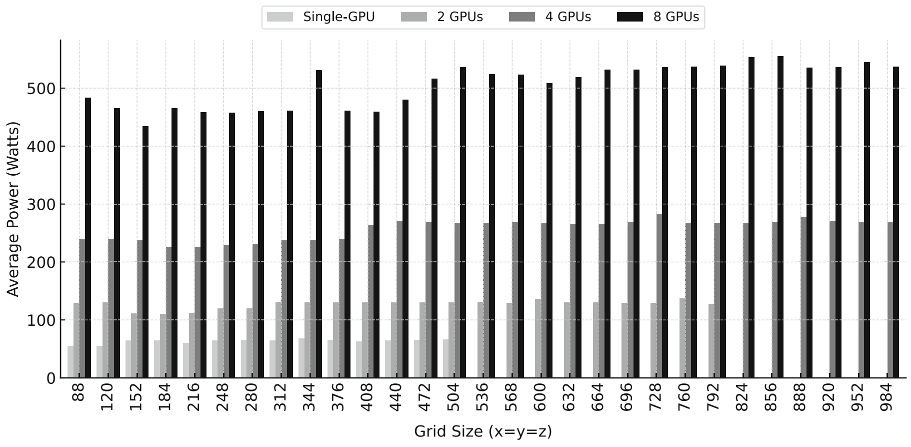

Fig. 8. Maximum Power dissipation for each version and grid input set.

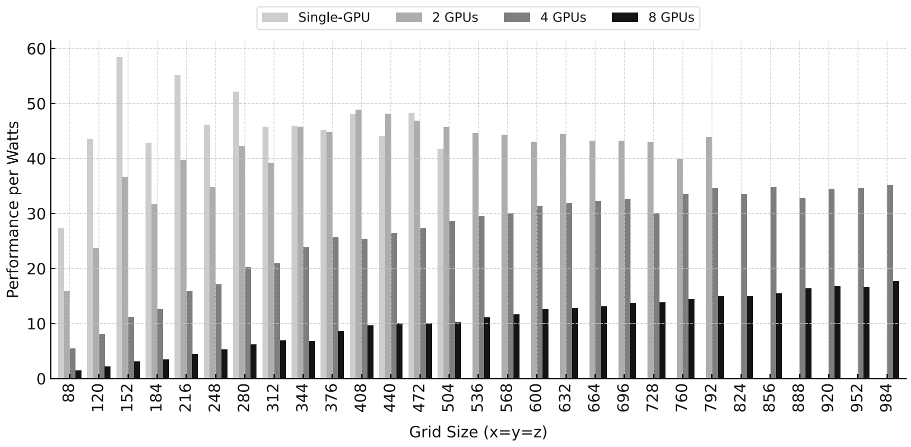

Fig. 9. Performance per Watts comparison

presents the lowest power across all problem sizes. This increase in maximum power with the use of more GPUs is expected, as more processing units mean more power dissipated. However, it is important to highlight that an increase in power does not always translate into a proportional increase in performance, highlighting the importance of considering energy efficiency when analyzing and optimizing applications for multi-GPU systems, aiming to achieve the best balance between performance and power consumption.

Extending the analysis through the data of the average power consumed during the execution of the CUDA kernel. Figure 9 shows that the single-GPU implementation has high efficiency for small input grid sizes. This is due to the fact that there is no cost of inter-GPU edge synchronization and also the fact that computing power does not become a performance limiting factor for these input sizes, due to the reduced scale of the grid Furthermore, we confirmed the low power efficiency of multi-GPU implementations for lower problem sizes. This is because the cost of inter-GPU synchronization and data communication inherent in such implementations results in unnecessarily high power consumption for issues that could be efficiently managed by a single GPU.

Furthermore, the energy efficiency of multi-GPU grows linearly with the size of the input grid. This indicates that to achieve high efficiency with multi-GPU, a large input set is needed, in order to ensure that throughput inherent to the multi-GPU implementation significantly outweighs the cost associated with data synchronization. This reinforces the idea that the best performance between single-GPU and multi-GPU implementations depends on factors such as the input grid and the complexity of modeling wave and medium characteristics.

6 Conclusions and Future Work

In this work, we have explored the implementation of Fletcher's multi-GPU method and compared it with the single-GPU approach. This implementation provided an innovative technical analysis for seismic applications using Multi-GPU systems with NVIDIA Tesla V100. Therefore, we studied the variations in performance and energy efficiency according to the variation in the size of the input grid. The presented results reinforce the importance of choosing the appropriate implementation method, given the size of the grid and the complexity of the modeled problem to be treated.

The results indicate that the multi-GPU implementation offers greater scalability, allowing the handling of larger input sets. However, it should be noted that for smaller input grids, multi-GPU performance is degraded due to the cost associated with inter-GPU edge synchronization. We have found that the performance of multi-GPU implementations is directly proportional to the input grid size, reaching peak efficiency and performance, with a Speedup of 2.77 when employing 2 GPUs for grid sizes around 408. However, within the dataset analyzed, the configuration with 8 GPUs did not generate gains proportional to the increase in the available computational load, being more suitable for problems with a larger grid dimension, where the cost of inter-GPU communication is small concerning the throughput provided. In future work, we intend to reduce the cost of inter-GPU communication by implementing a direct approach, which incorporates peer-to-peer (P2P) communication and the use of NVIDIA's NVLINK technology.

Acknowledgment. This work has been partially supported by Petrobras under number 2020/00182-5, by the call CNPq/MCTI/FNDCT - Universal 18/2021 under grants 406182/2021-3, and by the Coordenação de Aperfeioamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001.

References

Chu, C., Macy, B.K., Anno, P.D.: Approximation of pure acoustic seismic wave propagation in TTI media. Geophysics **76**(5), WB97–WB107 (2011)

Fletcher, R.P., Du, X., Fowler, P.J.: Reverse time migration in tilted transversely isotropic (TTI) media. Geophysics **74**(6), WCA179–WCA187 (2009)

Liu, G.-F., Meng, X.-H., Yu, Z.-J., Liu, D.-J.: An efficient scheme for multi-GPU TTI reverse time migration. Appl. Geophys. **16**(1), 56–63 (2019)

Liu, H., Li, B., Liu, H., Tong, X., Liu, Q., Wang, X., Liu, W.: The issues of prestack reverse time migration and solutions with graphic processing unit implementation. Geophys. Prospect. **60**(5), 906–918 (2012)

Lorenzon, A.F., Beck Filho, A.C.S.: Parallel computing hits the power wall: principles, challenges, and a survey of solutions. Springer Nature (2019)

Lukawski, M.Z., et al.: Cost analysis of oil, gas, and geothermal well drilling. J. Petrol. Sci. Eng. **118**, 1–14 (2014)

Navaux, P.O.A., Lorenzon, A.F., da Silva Serpa, M.: Challenges in high-performance computing. J. Braz. Comput. Soc. **29**(1), 51–62 (2023)

Okamoto, T., Takenaka, H., Nakamura, T., Aoki, T.: Accelerating large-scale simulation of seismic wave propagation by multi-GPUS and three-dimensional domain decomposition. Earth Planets Space **62**(12), 939–942 (2010)

Padoin, E.L., Pilla, L.L., Boito, F.Z., Kassick, R.V., Velho, P., Navaux, P.O.: Evaluating application performance and energy consumption on hybrid CPU+ GPU architecture. Clust. Comput. **16**, 511–525 (2013)

Papadrakakis, M., Stavroulakis, G., Karatarakis, A.: A new era in scientific computing: Domain decomposition methods in hybrid cpu-gpu architectures. Comput. Methods Appl. Mech. Eng. **200**(13), 1490–1508 (2011)

Pavan, Pablo J.., Serpa, Matheus S.., Carreño, Emmanuell Diaz, Martínez, Víctor., Padoin, Edson Luiz, Navaux, Philippe O. A.., Panetta, Jairo, Mehaut, Jean-François.: Improving Performance and Energy Efficiency of Geophysics Applications on GPU Architectures. In: Meneses, Esteban, Castro, Harold, Barrios Hernández, Carlos Jaime, Ramos-Pollan, Raul (eds.) High Performance Computing: 5th Latin American Conference, CARLA 2018, Bucaramanga, Colombia, September 26–28, 2018, Revised Selected Papers, pp. 112–122. Springer International Publishing, Cham (2019). https://doi.org/10.1007/978-3-030-16205-4_9

Pearson, C., Hidayetoğlu, M., Almasri, M., Anjum, O., Chung, I.-H., Xiong, J., Hwu, W.-M.W.: Node-aware stencil communication for heterogeneous supercomputers. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 796–805. IEEE (2020)

Serpa, M., Mishra, P.: Performance evaluation and enhancement of the fletcher method on multicore architectures (2022)