IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 23, 2019, accepted August 29, 2019, date of publication September 2, 2019, date of current version September 20, 2019.

*Digital Object Identifier 10.1109/ACCESS.2019.2939100*

# Distributed-Memory Load Balancing With Cyclic Token-Based Work-Stealing Applied to Reverse Time Migration

**ÍTALO A. S. ASSIS[1], ANTÔNIO D. S. OLIVEIRA[1], TIAGO BARROS[1], IDALMIS M. SARDINA[2], CALEBE P. BIANCHINI[3], AND SAMUEL XAVIER-DE-SOUZA[1], (Senior Member, IEEE)**

[1]Departamento de Engenharia de Computação e Automação, Universidade Federal do Rio Grande do Norte, Natal RN 59078-970, Brazil
[2]Escola de Ciências e Tecnologia, Universidade Federal do Rio Grande do Norte, Natal RN 59078-970, Brazil
[3]Faculdade de Computação e Informática, Universidade Presbiteriana Mackenzie, São Paulo SP 01302-000, Brazil

Corresponding author: Ítalo A. S. Assis (italoaug@ufrn.edu.br)

This work was supported in part by the Shell Brazil through the project Novos Métodos de Exploração Sísmica por Inversão Completa das Formas de Onda at the Universidade Federal do Rio Grande do Norte (UFRN), in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), and in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

**ABSTRACT** Reverse time migration (RTM) is a prominent technique in seismic imaging. Its resulting subsurface images are used in the industry to investigate with higher confidence the existence and the conditions of oil and gas reservoirs. Because of its high computational cost, RTM must make use of parallel computers. Balancing the workload distribution of an RTM is a growing challenge in distributed computing systems. The competition for shared resources and the differently-sized tasks of the RTM are some of the possible sources of load imbalance. Although many load balancing techniques exist, scaling up for large problems and large systems remains a challenge because synchronization overhead also scales. This paper proposes a cyclic token-based work-stealing (CTWS) algorithm for distributed memory systems applied to RTM. The novel cyclic token approach reduces the number of failed steals, avoids communication overhead, and simplifies the victim selection and the termination strategy. The proposed method is implemented as a C library using the one-sided communication feature of the message passing interface (MPI) standard. Results obtained by applying the proposed technique to balance the workload of a 3D RTM system present a factor of 14.1 % speedup and reductions of the load imbalance of 78.4 % when compared to the conventional static distribution.

**INDEX TERMS** Load balancing, reverse time migration, work-stealing, one-sided communication, distributed memory.

## I. INTRODUCTION

The migration of seismic data is the process that attempts to build an image of the Earth's interior from recorded field data. Migration places these data into their actual geological position in the subsurface using numerical approximations of either wave-theoretical or ray-theoretical approaches to simulate the propagation of seismic waves [1].

The wave-theoretical approach to the propagation of seismic waves employs the finite difference method (FDM) [2], [3] to numerically solve the equation describing the movement of the waves [1], [4]. This approach is prevalent among the geophysical community, due to its capacity of dealing with substantial velocity variations in complex geology (e.g., pre-salt).

Reverse time migration (RTM) [5]–[9] implements this approach. It is one of the most known FDM-based migration methods. RTM is computationally intensive in terms of data storage and handling, and its use of high-complexity algorithms. Therefore, exploiting parallelism is mandatory for RTM implementations in 3D Earth models (3D RTM) [10].

Parallel architectures can be classified as shared memory, when there is a single memory address space available to all processing units (e.g., nodes or cores), or distributed memory otherwise [11]. Many scientific and industrial computational resources are distributed memory systems composed of multi-processor nodes with shared memory systems. A hybrid parallel application works at these two levels

The associate editor coordinating the review of this manuscript and approving it for publication was Yilun Shang.

  

of parallelism. It can distribute the total workload among the nodes of a distributed memory system. Each node, then, distributes its subset of the workload among the processing units of its shared memory system. Parallel machines can also be described as heterogeneous when they have processing units built from different types of hardware, or homogeneous otherwise [12].

One of the main concerns in parallel computing is the efficient use of the available computational resources. Some applications such as RTM may suffer from load imbalance. A way of dealing with this issue is to employ load balancing techniques, which usually refer to the distribution of the workload among the available computational resources (e.g., nodes, processors, cores). The objective is to minimize the idling of the computational resources while there are still tasks remaining to be processed.

Ensuring the load balancing for an RTM is especially challenging in distributed memory systems. Distributing the workload in equal amounts of tasks for each computational node may not be optimal. Even for homogeneous computational systems with an evenly distributed workload, several factors may be a source of load imbalance. It can be intrinsic to the application itself or caused by program-external factors such as runtime environment routines (e.g., system calls) and resource availability. The competition for shared resources, such as the parallel file system or the network, can cause idling due to resource contention as the availability of the resources may differ across the nodes and along time.

Work-stealing (WS) is one of the main load balancing strategies. The fundamental idea of WS methods is that idle processing units steal tasks from the others [13] in an attempt to avoid the performance overhead of a centralized entity being responsible for the task scheduling. Processes stealing tasks are the thief processes, whereas the processes with stolen tasks are the victim processes. Nevertheless, in the context of distributed systems, some WS implementations present problems with too many failed steal attempts, with communication overhead, with the victim selection, and with the termination strategy.

This paper proposes a cyclic token-based work-stealing (CTWS) algorithm for distributed memory systems applied to RTM. The novel cyclic token approach reduces the number of failed steals, avoids communication overhead, and simplifies the victim selection and the termination process. The proposed work-stealing method was implemented in C using the message passing interface (MPI) [14] standard. The communication was implemented by remote memory access (RMA) using MPI one-sided communication [15]. This communication model allows the thief processes to perform the work-stealing without directly involving (or interrupting) the victim processes, thus further reducing communication overhead. Our 3D RTM code was implemented in C using MPI for distributed-memory parallelism across nodes, and OpenMP [16] for thread-level parallelism within nodes.

The contribution of this paper to the fields of distributed load balancing and RTM are:

1) the proposition of a novel approach to implementing load balancing with WS in distributed systems based on a cyclic token;
2) the mitigation of important WS implementation problems in distributed systems by the proposed cyclic token approach as it avoids failed steals and simplifies the victim selection and the termination strategy;
3) the reduction of the communication overhead of the WS distributed implementation by the use of MPI one-sided communication to implement the cyclic token approach;
4) a detailed evaluation of the conventional load balancing technique for 3D RTM showing that load imbalance is significant due to resource contention;
5) improvements in the execution time of 3D RTM of about 14 % and reductions of the load imbalance of about 78 %.

The rest of this paper is organized as follows. Section II shows the basics of RTM and describes our RTM implementation. Section III introduces the work-stealing method proposed in this work. Section IV details the application of the proposed technique to the RTM. Section V discusses the performance of the RTM with and without the proposed approach. Section VI presents a literature review of related works contrasting them with the proposed approach. Finally, Section VII summarizes this work and proposes future research.

## II. RTM AND STATIC LOAD BALANCING

In a seismic reflection survey, an acoustic source at a given location (a ''seismic shot'') generates a wave that propagates into the subsurface. Each time the wave travels through an interface between two layers with different impedance, part of its energy is reflected and is eventually registered at a set of receivers. This procedure is repeated for different shot locations in order to cover the whole area of interest. The data recorded by a single receiver for a single seismic shot is called a seismic trace, and a set of traces is called a seismogram. The seismograms can pass through many processing steps to finally provide an image of the subsurface.

Migration is one of the most critical steps in processing seismic data. It aims to position the reflection interfaces properly in the subsurface. A migrated section is an image representing the geological structures in the region of interest. This section can be used for interpretation purposes, often to locate and characterize oil and gas reservoirs.

Reverse time migration (RTM) [6], [9] is one of the most known migration methods. The main steps of an RTM are presented in Algorithm 1. The first step, the forward propagation, simulates the incident wavefield by propagating a source wavelet through the region of interest. The backward propagation generates the reflected wavefield by propagating the seismogram comprised of the seismic traces from a shot, a common shot gather, in reverse time order.

Both forward and backward propagation can be performed by iteratively solving, over a discrete grid, the acoustic wave

**Algorithm 1** Main Steps of a Reverse Time Migration

1: **for all** (shots locations) **do**
2: forward propagation
3: backward propagation of the common shot gather
4: image condition
5: **end for**

equation, described as:

$$\frac{\partial^2 u(\mathbf{x})}{\partial x_1^2} + \frac{\partial^2 u(\mathbf{x})}{\partial x_2^2} + \frac{\partial^2 u(\mathbf{x})}{\partial x_3^2} = \frac{1}{c(\mathbf{x})^2}\frac{\partial^2 u(\mathbf{x})}{\partial t^2} + s(t). \quad (1)$$

In (1), $\mathbf{x} = (x_1, x_2, x_3)$ are the spatial dimensions, $u(\mathbf{x})$ is the pressure wavefield, $c(\mathbf{x})$ is a velocity model, $t$ is the time dimension and $s(t)$ is the source, i.e., a wavelet representing the seismic shot.

The finite difference method (FDM) is often used to numerically solve (1) by approximating its PDEs (partial differential equations). Approximations of higher orders provide more accurate results, with smaller numerical errors. Spatial and time restrictions should be observed when solving finite differences by a numerical approach [17].

The content of the velocity model, $c(\mathbf{x})$, plays an important role from the geophysical perspective. Its complexity is the reason why RTM is used. It also influences the computational cost of the RTM as it determines the spatial and time resolutions. In other words, the maximum and minimum values of the velocity model, $c_{\min}$ and $c_{\max}$, directly influence on the total number of operations performed by an RTM. However, for a fixed $f_{\max}$, two models having the same $c_{\min}$ and $c_{\max}$ demand the same time and spatial resolutions, and generally incurs in the same computational cost, no matter they have different geological structures.

The wave propagation via FDM is performed over a limited grid representing the region of interest. Nevertheless, the region where the seismic survey takes place is not restricted to that region of interest. For this reason, it is common practice to add extra points to the limits of the grid, in order to absorb the energy reaching the borders of the model [18].

RTM relies on the principle that the incident and the reflected wavefields, $u_{\mathrm{i}}(\mathbf{x}, t)$ and $u_{\mathrm{r}}(\mathbf{x}, t)$, correlate at the reflection interfaces. An image condition with the following mathematical description performs this correlation.

$$I(\mathbf{x}) = \int_{t=0}^{T} u_{\mathrm{i}}(\mathbf{x}, t) \cdot u_{\mathrm{r}}(\mathbf{x}, t)\mathrm{d}t, \quad (2)$$

where $T$ is the total time of the the simulation.

The three RTM steps (as Algorithm 1 shows) are repeated for each shot location generating one migrated section per shot. The final migrated section is achieved by summing up all shot migrations.

The RTM method used in the experiments described in this paper is an extension of the RTM introduced by Nunes-do-Rosario *et al.* [19]. It is implemented in C with a hybrid parallel approach. MPI is used to distribute the workload of different shots among computational nodes of a distributed system, and OpenMP is employed to parallelize internal loops of each shot processing. The implemented parallel RTM code is described in Algorithm 2.

**Algorithm 2** Reverse Time Migration With Work-Stealing Load Balancing. $ns$ Is the Number of Time Steps. $i_{\text{shot}}$ Is the Number of the Shot Being Processed

1: statically distribute shots among nodes using MPI
2: read RTM parameters
3: compute absorbing boundaries coefficients
4: #OpenMP parallel section begin
5: **for all** (shots locations of the process) **do**
6: read shot seismogram
7: **for** ($t_i = 0$ to $ns$) **do**
8: #OpenMP for
9: **for** (all grid points) **do**
10: compute the wavefield
11: **end for**
12: add the source wavelet
13: write wavefield to disk
14: **end for**
15: **for** ($t_i = ns - 1$ to 0) **do**
16: #OpenMP for
17: **for all** (grid points) **do**
18: compute the wavefield
19: **end for**
20: #OpenMP for
21: **for all** (receivers location) **do**
22: inject observed data samples at time $t_i$
23: **end for**
24: read forward wavefield at $t_i$ from disk
25: #OpenMP for
26: **for all** (main grid points) **do**
27: perform image condition
28: **end for**
29: **end for**
30: **end for**
31: #OpenMP parallel section end
32: reduce all nodes migrated sections

Absorbing boundaries are implemented in our RTM code as reduction coefficients (Line 3 of Algorithm 2) that taper the wavefield amplitudes in a layer of grid points surrounding the mesh as proposed in [18]. The acoustic wave equation (1) is solved for each propagation by the FDM with a second order approximation in time and eighth order approximation for each spatial dimension (Lines 10 and 18 of Algorithm 2).

The incident wavefield is stored on disk (Line 13 of Algorithm 2) at each forward wave propagation time step. At each backward wave propagation time step, the incident wavefield is read from disk (Line 24 of Algorithm 2) in order to perform the image condition (Line 27 of Algorithm 2).

The load balancing of the RTM described by Algorithm 2 is static. An equal amount of shots, or nearly equal, is allocated to each node at the beginning of the algorithm (Line 1). From this point on, no more load balancing decisions are taken. If a process finishes processing all its shots, i.e., leaves the shots loop (Lines from 5 to 30), it has to wait for the slowest process in order to collectively summing up all shot migrations, i.e., performing the reduction operation, through the command MPI_Reduce, in Line 32. Since MPI does not provide task schedulers, the static schedule is often used because of its ease of implementation.

## III. CYCLIC TOKEN-BASED WORK-STEALING

Cyclic token-based work-stealing (CTWS) is the load balancing method for distributed memory systems introduced in this paper. It is a library implemented in C using MPI. In order to reduce communication overhead, CTWS is implemented using MPI one-sided communication.

Since MPI-2 [15], MPI specification includes the concept of one-sided communication. This MPI feature implements RMA, which allows processes to make a portion of their local memory available for access by other processes. In one-sided communications, the process that accesses the memory is called the origin process while the process whose memory is accessed is called the target process [20]. All processes involved in one-sided communication must collectively create windows. A window is a structure with information on the memory regions which the processes make available for RMA.

Many operations are available on MPI one-sided communication. Our work mainly uses the operations MPI_Put and MPI_Get, which are used to write to and read from remote memory, respectively. These operations are passive, i.e., the target process is not involved in the operation. Therefore, the target process keeps computing its tasks while the RMA operation is performed.

A token and a list of remaining tasks per process are the two main elements of the proposed work-stealing technique. Both are implemented as MPI one-sided communication windows. The token was implemented as an integer number. It is initialized as 0, meaning that, according to the list of remaining tasks, there are tasks to be stolen. The first process to figure out that no more tasks can be stolen sets the token to 1, i.e., sets the token to finish.

The token gets passed around through an MPI_Put operation in a round-robin fashion. Only the process owning the token can update the list of tasks per process and steal tasks. This strategy avoids deadlocks that would be caused by two processes trying to steal from each other at the same time. In such a case, both of them would have to grant access to both of their lists of remaining tasks. If both of them granted access to one of these lists, a deadlock would occur.

In the initialization (Line 1 of Algorithm 3), the token must be allocated to a single process. The shots to be processed are equally distributed among the processes. Each process has its copy of this list of tasks per process. This list is implemented as an array of integers where the $i$-th element is the number of remaining tasks of the $i$-th process. The functions *getTask*() and *updateList*() are responsible for managing the token and the list of tasks.

**Algorithm 3** Cyclic Token-Based Work-Stealing. $t_{\text{id}}$ Is the Task Identification Number

```
1: Initialize CTWS variables
2: t_id = getTask()
3: while (t_id ≠ −1) do
4:    for all (iterations of task t_id) do
5:       updateList()
6:       Compute an iteration of t_id
7:    end for
8:    t_id = getTask()
9: end while
```

The proposed strategy is designed for applications with iterative tasks. At each task iteration, the function *updateList*() is called by each process. It first verify whether it possesses the token (Line 5 of Algorithm 3). Should it have the token and it is not set to finish, the process updates its number of remaining tasks in its list and copies its list to the next process through an MPI_Put operation in a ring fashion. When the process does not have the token, it simply continues working on its tasks. By doing so, any process has a close approximation of the current amount of remaining tasks of each process. This information is then used to lead the stealing stage.

The core of the proposed work-stealing strategy is the function *getTask*() (Lines 2 and 8 of Algorithm 3), which is detailed by the flow chart of Fig. 1. When a process has shots to be processed, *getTask*() returns the first of them. Otherwise, the process attempts to steal tasks from other processes.

Only the process possessing the token can try to steal tasks. For this reason, the first step of the proposed strategy is to ensure that the process has the token. If it does not, it will perform a busy-wait by repeatedly verifying whether it posses the token. Once the token arrives, the thief process checks whether the token is set to finish. If so, no work-stealing is needed, and the process continues to the reduction operation.

However, when the token is not set to finish, the thief process tries to steal from the process with more remaining tasks, according to the thief's list of tasks. Since the list of remaining tasks per process is an approximation, the real number of remaining tasks may have changed by the stealing time. For this reason, the thief process verifies the actual number of remaining tasks of the victim process through an MPI_Get operation. If the victim process does not have tasks to be stolen, the thief process updates its list of tasks and restarts the procedure by finding a new victim process in its updated list of tasks. Should there be no more tasks left to be stolen, then the thief process sets the token to finish,

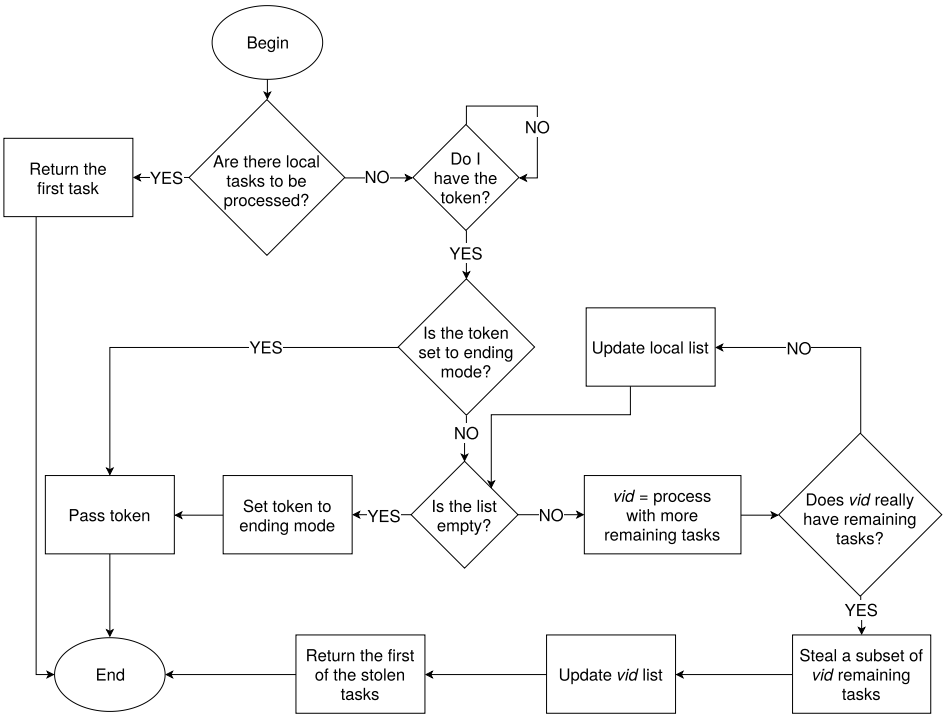


**FIGURE 1.** **Detailed flow chart of the function *getTask*() that is responsible for determining which is the next task to be processed by each process. In this work, the task unit to be processed is the RTM of each shot gather.**

forwards it to the next process and continues to the reduction operation.

When the thief process finds a victim process with tasks to be stolen, it uses MPI one-sided communication (MPI_Put and MPI_Get) to steal a subset of the remaining tasks of the victim process. For the tests in this work, half of the remaining tasks are stolen. As discussed by Dinan *et al*. [21], stealing half of the tasks of the victim increases the number of possible victims for the next steals. This strategy aims to improve scalability by reducing the time to locate and steal tasks. Finding an optimal number of tasks to be stolen is not of the scope of this paper.

This stealing procedure is seamless to the victim process, i.e., the victim process will not stop processing its current task to communicate with the thief process. At this point, the thief process starts to work on the first of its stolen tasks. Should the victim process have a single remaining task, then the thief process will try to steal it. In case the victim process also tries to start processing its only left task at the same time, the race condition is avoided by the one-sided communication operators *MPI_Win_lock* and *MPI_Win_unlock* set to the type *MPI_LOCK_EXCLUSIVE*. These commands ensure mutual exclusion allowing a single process to access the window at a time.

## IV. CTWS APPLIED TO RTM

In this work, the task unit to be processed is the RTM of each shot gather. In other words, the iterations of the shots loop (Lines from 5 to 30 of Algorithm 2) are distributed to the nodes of a distributed system using CTWS. For this reason, the commands controlling the shots loop of the RTM must be replaced by the commands controlling the tasks loop of CTWS. Line 5 of Algorithm 2 is replaced by Lines 2 and 3 of Algorithm 3, and Line 30 of Algorithm 2 is replaced by Lines 8 and 9 of Algorithm 3. In this context, the task identification number, $t_{\text{id}}$, represents the number of the shot gather. The function *updateList*() is called inside of both the forward propagation loop (Lines from 7 to 14 of Algorithm 2) and the backward propagation loop (Lines from 15 to 29 of Algorithm 2).

The larger the number of processes, the shorter the time that each process will have the token. Because of that, the overhead caused by *updateList*() in the RTM is proportionally smaller for larger numbers of processes and larger input sizes. On the other hand, by running *updateList*() and having the token in each process fewer times, the list of remaining tasks per process is more prone to be out of date. This way the number of unsuccessful steals attempts performed by *getTask*() may increase and so its overhead.

## V. RESULTS AND DISCUSSION

The experiments were performed on Yemoja, an 856 node supercomputer. Each computational node hosts two processors 10-core Intel Xeon E5-2690 Ivy Bridge v2 at 3.00 GHz. 200 nodes are equipped with 256 GB RAM and 656 nodes with 128 GB RAM. This supercomputer employs an 850 TB Lustre parallel distributed file system. Yemoja is located at the Manufacturing and Technology Integrated Campus of the

National Service of Industrial Training (SENAI-CIMATEC). Both the 128 GB RAM and the 256 GB RAM were used in the following experiments. Since the total amount of RAM required by our RTM implementation is significantly inferior to 128 GB, this fact does not influence the algorithm performance.

In order to validate the 3D wave propagator, which underlies the 3D RTM algorithm used in the experiments, we compared a seismic trace generated by our propagator with the analytical solution, computed according to [22], in a homogeneous velocity model. The source was a Ricker wavelet with a peak frequency of 20 Hz. The distance between source and receiver is 200 m. The medium has a constant velocity of 2000 m/s. In this experiment, our wave propagator provided a very accurate approximation to the 3D waveform analytical solution with a mean squared error of $6 \times 10^{-14}$.

For the following experiments, the size of the input grid is $401 \times 401 \times 401$, the peak frequency of the source wavelet is 20 Hz, the time sampling is 1 ms, the spatial sampling is 10 m, and the number of time steps is 3501. $c(\mathbf{x})$ is a two layers model with a horizontal interface positioned at the center of the vertical dimension. The velocity is 1400 m/s for the top layer and 2000 m/s for the bottom layer.

The programs were compiled with the *gcc* compiler using the optimization flag *-O3* and OpenMPI 3.1.2 for all experiments. A single MPI process was created at each computational node. We used HPCToolkit performance tools [23] to measure the execution times and the overhead of our strategy. For all the following experiments using CTWS, the load balancing overhead was inferior to 0.4%. A single experiment was performed at a time in order to avoid multiple tests competing for the shared resources of the cluster.

Firstly, we measured the load imbalance of the 3D RTM without applying a dynamic load balancing technique. For that we ran the RTM of 40, 80, 160, 320 and 640 shots with 4, 8, 16, 32 and 64 nodes, respectively. As shown in Fig. 2 and 3, for the experiment with 4 nodes, the average idle time per node is 2.7 % of the total time of 18.4 h. As the number of nodes increases up to 64, the average idle time per node increases to 23.4 % of the total time of 39.2 h. Although the number of shots per node is the same for each experiment, the competition for shared resources of the cluster (e.g., network and parallel file system) increases the runtime as the number of nodes increase.

Fig. 4 details the execution of the 3D RTM ran over 64 nodes without applying a dynamic load balancing technique. Although the workload is distributed evenly among the homogeneous nodes, the runtime of a single shot RTM ranges from 1.5 to 9.3 h. The fastest node stays idle for 17.6 h while the other nodes finish their tasks, i.e., 45% of the total runtime. Factors as a race condition for the network and the parallel storage system can cause such load imbalance.

Fig. 3 also shows results generated by the proposed work-stealing technique employed in the same set of experiments. The proposed technique presented a maximum average idle

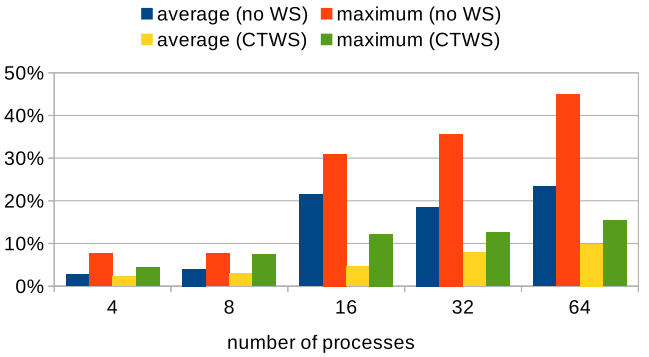


**FIGURE 2.** 3D RTM maximum process idle time and average process idle time with 4, 8, 16, 32 and 64 nodes. Both RTM implementations with and without the proposed work-stealing method (CTWS) process 10 shots per node.

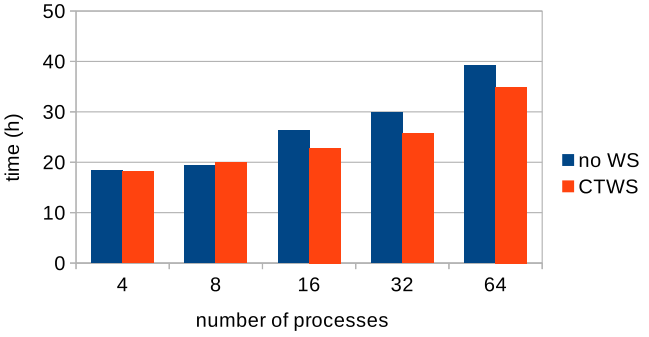


**FIGURE 3.** 3D RTM total runtime with 4, 8, 16, 32 and 64 nodes. Both RTM implementations with and without the proposed work-stealing method (CTWS) process 10 shots per node.

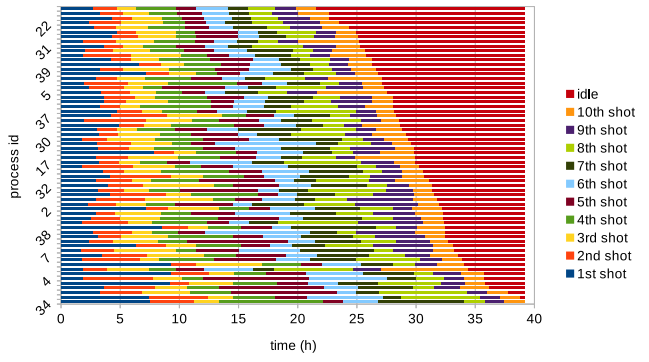


**FIGURE 4.** Example of 3D RTM runtime per process and shot ran over 64 nodes. The shots are numbered in the order they are processed in the node they were assigned to. The processes are sorted by their idle time.

time of 9.9%, showing its effectiveness in balancing the load. For the experiment with 4 nodes, the average idle time per node is 2.3 % of the total time of 18.2 h. As the number of nodes increases up to 64, the average idle time per node slightly increases to 9.9 % of the total time of 34.9 h.

The total execution times displayed in Fig. 3 show that the proposed technique outperforms the 3D RTM with the conventional static load balancing when using a more substantial number of nodes. The total runtime was reduced by 13.4%, 14.1% and 10.8% when ran over 16, 32 and

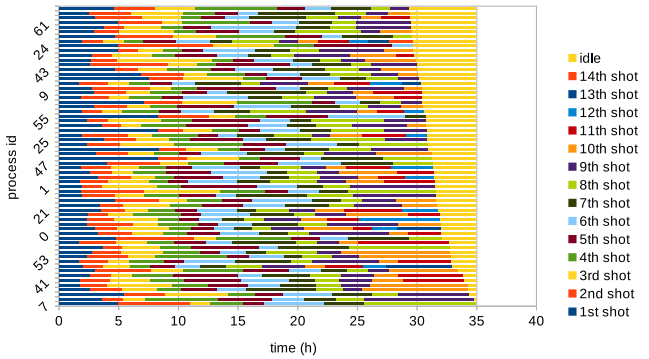


**FIGURE 5.** Example of 3D RTM runtime per process and shot ran over 64 nodes using the proposed work-stealing. The shots numbers refer to the order they were processed within its node. The processes are sorted by their idle time.

64 nodes, respectively. For the fewest number of nodes, however, the proposed technique performance was similar to the static load balancing. The total runtime increased by 3.2% when ran over 8 nodes and decreased by 0.9% when ran over 4 nodes. Industry-scale RTM, however, is usually performed over a large number of computational nodes. Regarding the load imbalance, as shown in Fig. 2, the proposed method was able to reduce the average idle time for all the performed tests. In the best scenario, when ran over 16 nodes, the idle time was reduced from 21.5 % to 4.5 %, representing a 78.4 % improvement in the effective use of the resources.

Fig. 5 details the 3D RTM execution over 64 nodes, using the proposed work-stealing technique. Although there are differences between the processing times of a single shot, nodes with better resource availability steal shots from others that are slower, thus improving load balancing. The least busy node processed only 6 shots while the busiest node processed 14 shots.

Table 1 presents all steal attempts of the example of Fig. 5. It shows that 37 out of 38 steal attempts were successful. In five cases, 13.2 % of the steals, a task was stolen for the second time:

1) in steal 18, process 20 stole the task 387 from process 21. Before that, in stealing 5, process 21 stole the same task from process 38;
2) in steal 23, process 28 stole the task 178 from process 29. Before that, in stealing 8, process 29 stole the same task from process 17;
3) in steal 24, process 34 stole the task 478 from process 9. Before that, in stealing 10, process 9 stole the same task from process 47;
4) in steal 28, process 21 stole the task 307 from process 16. Before that, in stealing 3, process 16 stole the same task from process 30;
5) in steal 29, process 41 stole the task 318 from process 18. Before that, in stealing 9, process 18 stole the same task from process 31;

In our test case, stealing the same task multiple times does not represent an additional overhead since there is no

**TABLE 1.** Steal attempts of the example of fig. 5.

| stealing attempt | thief process | victim process | stolen tasks |
|---|---|---|---|
| 1 | 63 | 20 | 205,206,207 |
| 2 | 19 | 24 | 245,246,247 |
| 3 | 16 | 30 | 305,306,307 |
| 4 | 49 | 34 | 346,347 |
| 5 | 21 | 38 | 386,387 |
| 6 | 10 | 55 | 556,557 |
| 7 | 22 | 7 | 77,78 |
| 8 | 29 | 17 | 177,178 |
| 9 | 18 | 31 | 317,318 |
| 10 | 9 | 47 | 477,478 |
| 11 | 15 | 8 | 88 |
| 12 | 50 | 51 | 518 |
| 13 | 30 | 24 | 248 |
| 14 | 62 | 25 | 258 |
| 15 | 32 | 55 | 558 |
| 16 | 63 | 57 | 578 |
| 17 | 17 | 59 | 598 |
| 18 | 20 | 21 | 387 |
| 19 | 2 | 1 | 19 |
| 20 | 23 | 4 | 49 |
| 21 | 22 | 5 | 59 |
| 22 | 42 | 8 | 89 |
| 23 | 28 | 29 | 178 |
| 24 | 34 | 9 | 478 |
| 25 | 35 | 12 | - |
| 26 | 35 | 13 | 139 |
| 27 | 30 | 14 | 149 |
| 28 | 21 | 16 | 307 |
| 29 | 41 | 18 | 318 |
| 30 | 49 | 37 | 379 |
| 31 | 13 | 38 | 389 |
| 32 | 15 | 39 | 399 |
| 33 | 50 | 54 | 549 |
| 34 | 18 | 57 | 579 |
| 35 | 5 | 58 | 589 |
| 36 | 0 | 59 | 599 |
| 37 | 3 | 60 | 609 |
| 38 | 38 | 61 | 619 |

**TABLE 2.** Steal attempts varying the number of processes.

| number of processes | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|
| total steal attempts | 0 | 2 | 3 | 12 | 38 |
| failed steal attempts | 0 | 0 | 0 | 0 | 1 |

significant extra cost to move a task between processes. This fact occurs because all the data is available to all processes through the Yemoja's parallel file system. A method that considers the cost of moving tasks is left to future work.

Table 2 shows the total number of steal attempts and failed steals varying the number of processes. In this test set, 1 out of 55 steal attempts was unsuccessful, i.e., 98.2% of the steal attempts were successful.

In terms of weak scalability, both the RTM with and without CTWS are not scalable as the total runtime increases when the number of shots and nodes are doubled. This means that RTM with CTWS may also be affected by the concurrency for shared resources of the distributed system as the number of nodes increases. However, by moving tasks to nodes with better resource availability, RTM with CTWS was able to deliver up to 14.1% speedup when compared to using a static load distribution.

## VI. RELATED WORKS

Several authors have proposed strategies to address the load imbalance for shared memory systems. Barros *et al.* [24] introduced a runtime method based on coupled simulated annealing (CSA) [25] to auto-tune the workload distribution of 3D acoustic wave propagation implemented with the FDM method. Andreolli *et al.* [26], [27] proposed a compilation-time auto-tuning based on genetic algorithms to find the best set of parameters (e.g., workload distribution, compilation flags) for seismic applications. Sena *et al.* [28] used cache blocking for the 3D RTM and proposed a procedure called Min-Worst-Min Block (MWMB) to find an efficient block size. Hofmeyr *et al.* [29] introduced a dynamic load balancing for multicore systems using runtime tools. Tchiboukdjian *et al.* [30] proposed a method that ensures all the data in the cache memory is used before being replaced. This method was designed for applications with linear access to memory. Imam and Sarkar [31] presented a work-stealing scheduler based on task priority queues. Balancing the computational load at the shared memory level can lead to a significant reduction in the execution time. Our work aims to achieve further improvement by balancing the workload at the distributed memory level.

Other authors provide methods to deal with the load imbalance of distributed memory systems. Khaitan *et al.* [32] proposed a master-slave based load balancing approach. Tesser *et al.* [33]–[35] proposed a simulation-based strategy to evaluate the performance and tune the dynamic load balancing of iterative MPI applications and applied it to a 3D wave propagation. Padoin *et al.* [36], [37] proposed combining a load balancing with techniques of processor frequency control in order to reduce energy consumption along with execution time. These approaches differ from this work for being centralized, i.e., a single or a few computational processes take the load balancing decisions. This behavior may lead to overload at the central element and significantly degrade performance [32].

To avoid losses of performance caused by a centralized load balancing element, some authors proposed decentralized load balancing strategies. Sharma and Kanungo [38] presented a technique to balance the computational load in heterogeneous multicore clusters, where no prior knowledge about the computational resources is required. Zheng *et al.* [39] introduced a periodic load balancing strategy, where the balancing decisions are taken hierarchically in a tree fashion. Different from this work, in these methods, the processes involved in the load balancing decisions have to synchronize to exchange information. This communication synchronization overhead may reduce parallel performance.

Work-stealing algorithms [13] have been used to provide decentralized load balancing methods for distributed systems. Martinez *et al.* [40] used StarPU, a task-based runtime system, to distribute the load balance of the 3D isotropic elastic wave propagation among processors and graphics processing units (GPUs) simultaneously. They compared centralized load-balancing and decentralized work-stealing algorithms from StarPU. Khaitan and Mccalley [41] applied dynamic load balancing with work-stealing to a contingency analysis application while Mor and Maillard [42] proposed an MPI library for load balancing branch and bound applications. These approaches use asynchronous communication to reduce the communication overhead as the processes which originate the communication may keep working while waiting for replies from their messages. Different from our work, these papers employ two-sided communication, i.e., both the origin and the destination processes are involved in the communication. This way, the destination processes have to interrupt their computation eventually to reply to the messages they have received. Also, this kind of non-blocking communication may imply in the origin process having to wait for its reply even when overlapping communication with computation.

One-sided communication is an alternative to reduce communication overhead. This model of communication allows a process to read and write data from a remote memory region without the target process being involved. Some authors have used it in the recent literature. Li *et al*. [43] used profiling information to estimate the task grain size and guide the asynchronous work-stealing. Kumar *et al.* [44] introduced a load-aware work-stealing based on a policy to choose a victim that completely avoids the failed steals. Dinan *et al.* [21] discussed the design and scalability aspects of work-stealing for distributed memory systems. They also proposed a runtime system for supporting work-stealing, which implements several techniques to achieve scalability in distributed memory systems. These methods employ one-sided communication through partitioned global address space (PGAS), a programming model that provides a globally shared address space for distributed memory. On the contrary, our work employs relies on MPI one-sided communication, which has no global address space. According to Fu *et al.* [45], employing PGAS often demands an important development effort to exploit these programming models thoroughly. Moreover, a vast majority of scientific codes use MPI either directly or via third-party software.

Other authors have recently used MPI implementations of one-sided communication in areas such as large-scale multimedia content analysis [46], graph processing [45] and matrix operations [47], [48]. According to Diaz *et al.* [11], MPI has been the *de facto* standard in HPC for the last decades. In the context of load balancing, Vishnu and Agarwal [49] introduced a work-stealing method using MPI one-sided communication for machine learning and data mining algorithms. Different from our proposal, their approach for victim selection is either random, which may increase the number of network requests, or prone to network contention because of having multiple thief processes trying to steal the same victim. Moreover, Vishnu and Agarwal employ a termination strategy that does not look at the entire victim set, potentially causing some processes to finish while there are remaining tasks to perform. Differently, we employ a termination strategy in which the processes

**TABLE 3. Literature review on load balancing methods. The proposed work-stealing method benefits from MPI one-sided communication to further reduce communication overhead and is applied to RTM in distributed memory systems.**

| | distributed memory | decentralized | work-stealing | asynchronous communication | one-sided communication | PGAS | MPI RMA | WS with global load information | RTM |
|---|---|---|---|---|---|---|---|---|---|
| Barros *et al.* [24] | | | | | | | | | |
| Andreolli *et al.* [26] | | | | | | | | | |
| Andreolli *et al.* [27] | | | | | | | | | |
| Sena *et al.* [28] | | | | | | | | | x |
| Hofmeyr *et al.* [29] | | x | | | | | | | |
| Tchiboukdjian *et al.* [30] | | | x | | | | | | |
| Imam and Sarkar [31] | | x | x | | | | | x | |
| Khaitan *et al.* [32] | x | | | | | | | | |
| Tesser *et al.* [33] | x | | | | | | | | |
| Tesser *et al.* [34] | x | | | | | | | | |
| Tesser *et al.* [35] | x | | | | | | | | |
| Padoin *et al.* [36] | x | | | | | | | | |
| Padoin *et al.* [37] | x | | | | | | | | |
| Sharma and Kanungo [38] | x | x | | | | | | x | |
| Zheng *et al.* [39] | x | x | | | | x | | | |
| Martinez *et al.* [40] | x | x | x | x | | | | | |
| Khaitan and Mccalley [41] | x | x | x | x | | | | | |
| Mor and Maillard [42] | x | x | x | x | | | | | |
| Li *et al.* [43] | x | x | x | x | x | x | | | |
| Kumar *et al.* [44] | x | x | x | x | x | x | | | |
| Dinan *et al.* [21] | x | x | x | x | x | x | | | |
| Vishnu and Agarwal [49] | x | x | x | x | x | | x | | |
| Our proposal | x | x | x | x | x | | x | x | x |

only finish when there are no more victims to be stolen, and the token is set to finish. This is only achieved because, in our proposed work-stealing algorithm, we share and update *global load information* without the need for synchronization. Sharing and updating the global load information has the main advantages of helping the thief processes to make better decisions and preventing failed stealing attempts.

Regarding load balancing strategies to RTM, little effort has been employed to schedule RTM tasks among nodes of distributed systems. Several authors implement parallel RTM for distributed systems using static scheduling [50]–[57]. As shown in Section V, the use of static distribution may lead to inefficient use of the distributed computational resources as faster nodes may wait idly for the slower ones to finish their tasks. On the other hand, our work-stealing strategy allows moving tasks among nodes in order to keep all resources busy as much as possible. In this work, we compare our proposed load balancing approach to static scheduling as it is arguably the conventional strategy to distribute RTM shots among the computing nodes in distributed systems.

In summary, this work distinguishes itself from the others as it proposes a decentralized work-stealing method to balance the load of the RTM in distributed systems. It employs MPI one-sided communication to reduce its overhead by communicating asynchronously and without the victim's involvement. By keeping global load information, our method lowers the cost of victim selection and process termination. Consequently, it reduces the number of failed stealing attempts.

Table 3 presents a summary of the works related to load balancing mentioned above, highlighting their main characteristics in comparison to the method proposed in this work.

## VII. CONCLUSION

We have presented a decentralized work-stealing strategy with asynchronous communication to balance the load of a 3D reverse time migration for distributed computing systems. Each process communicates in a round-robin fashion to maintain a close approximation of the remaining tasks list. This list is used to lead the stealing when processes are idle. This strategy decentralizes the dynamic load balancing and avoids the overhead of centralized decisions. A token avoids deadlocks by ensuring that two processes cannot steal each other at the same time. The MPI one-sided communication prevents race conditions by serializing access to a memory space by multiple processes. By using MPI one-sided communication, the stealing is seamless to the victim processes since they do not stop processing their tasks during the stealing, avoiding unnecessary communication.

In the presented experiments, the 3D RTM had up to 23.4 % of average idle time when ran over 64 nodes. This imbalance might be significantly reduced should the proposed work-stealing be applied. For the set of experiments performed in this paper, the proposed method has reduced the total execution time of the 3D RTM in up to 14.1 % and its load imbalance in the order of 78.4 % when compared to the conventional static distribution.

Further investigation is necessary to assess whether additional improvement can be achieved by adjusting the frequency of checking the token, the number of shots to be stolen, the method used to update the list of remaining tasks and the technique to avoid deadlocks. Also, future work should focus on different aspects of a distributed system that may influence the load imbalance such as the use of fault tolerance protocols (e.g., [58], [59]) and heterogeneous

computational systems. A comparison of our method against other load balancing methods is left to future work.

## ACKNOWLEDGMENT

The authors gratefully acknowledge the strategic importance of the support given by ANP through the R&D levy regulation. The authors are also thankful to the High-Performance Computing Center at UFRN (NPAD/UFRN) and the Manufacturing and Technology Integrated Campus of the National Service of Industrial Training (SENAI CIMATEC) for making computer resources available. Finally, the authors would like to thank Jorge Lopez from Shell and the anonymous reviewers for providing essential comments on this article.

## REFERENCES

[1] O. Yilmaz, *Seismic Data Analysis: Processing, Inversion, and Interpretation of Seismic Data* (Investigations in Geophysics), vol. 10. Tulsa, OK, USA: Society of Exploration Geophysicists, 2001.

[2] Z. Alterman and F. Karal, Jr., "Propagation of elastic waves in layered media by finite difference methods," *Bull. Seismol. Soc. Amer.*, vol. 58, no. 1, pp. 367–398, 1968. [Online]. Available: http://www.bssaonline.org/content/58/1/367.short

[3] K. R. Kelly, R. W. Ward, S. Treitel, and R. M. Alford, "Synthetic seismograms: A finite-difference approach," *Geophysics*, vol. 41, no. 1, pp. 2–27, 1976.

[4] J. F. Claerbout and S. M. Doherty, "Downward continuation of moveout-corrected seismograms," *Geophysics*, vol. 37, no. 5, pp. 741–768, 1972.

[5] C. H. Hemon, "Equations d'onde et modeles," *Geophys. Prospecting*, vol. 26, no. 4, pp. 790–821, 1978. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2478.1978.tb01634.x

[6] E. Baysal, D. Kosloff, and J. W. C. Sherwood, "Reverse time migration," *Geophysics*, vol. 48, pp. 1514–1524, Nov. 1983.

[7] G. McMechan, "Migration by extrapolation of time-dependent boundary values," *Geophys. Prospecting*, vol. 31, no. 3, pp. 413–420, 1983. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-2478.1983.tb01060.x

[8] N. D. Whitmore, "Iterative depth migration by backward time propagation," in *Proc. SEG Tech. Program Expanded Abstr.*, 1983, pp. 382–385.

[9] D. D. Kosloff and E. Baysal, "Migration with the full acoustic wave equation," *Geophysics*, vol. 48, no. 6, pp. 677–687, 1983.

[10] M. Araya-Polo, F. Rubio, R. De La Cruz, M. Hanzich, J. M. Cela, and D. P. Scarpazza, "3D seismic imaging through reverse-time migration on homogeneous and heterogeneous multi-core processors," *Sci. Program.*, vol. 17, nos. 1–2, pp. 185–198, 2009.

[11] J. Diaz, C. Muñoz-Caro, and A. Niño, "A survey of parallel programming models and tools in the multi and many-core era," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 8, pp. 1369–1386, Jan. 2012.

[12] P. Pacheco, *An Introduction to Parallel Programming*, T. Green and N. McFadden, Eds. Burlington, VT, USA: Morgan Kaufmann, 2011.

[13] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/324133.324234

[14] L. Clarke, I. Glendinning, and R. Hempel, "The MPI message passing interface standard," in *Programming Environments for Massively Parallel Distributed Systems*, K. M. Decker and R. M. Rehmann, Eds. Basel, Switzerland: Birkhäuser, 1994, pp. 213–218.

[15] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming With the Message Passing Interface*, 2nd ed. Cambridge, MA, USA: MIT Press, 1999.

[16] L. Dagum and R. Menon, "OpenMP: An industry standard API for shared-memory programming," *IEEE Comput. Sci. Eng.*, vol. 5, no. 1, pp. 46–55, Mar. 1998.

[17] J. M. Carcione, G. C. Herman, and A. P. E. ten Kroode, "Seismic modeling" *Geophysics*, vol. 67, no. 4, pp. 1304–1325, 2002.

[18] C. Cerjan, D. Kosloff, R. Kosloff, and M. Reshef, "A nonreflecting boundary condition for discrete acoustic and elastic wave equations," *Geophysics*, vol. 50, no. 4, pp. 705–708, 1985.

[19] D. A. Nunes-do-Rosário, S. Xavier-de-Souza, R. C. Maciel, and J. C. Costa, "Parallel scalability of a fine-grain prestack reverse time migration algorithm," *IEEE Geosci. Remote Sens. Lett.*, vol. 12, no. 12, pp. 2433–2437, Dec. 2015. [Online]. Available: http://ieeexplore.ieee.org/document/7307125/

[20] M. Y. Park and S. H. Chung, "Detecting race conditions in one-sided communication of MPI programs," in *Proc. 8th IEEE/ACIS Int. Conf. Comput. Inf. Sci. (ICIS)*, Jun. 2009, pp. 867–872.

[21] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proc. Conf. High Perform. Comput. Netw., Storage Anal. (SC)*, 2009, pp. 53:1–53:11. [Online]. Available: http://doi.acm.org/10.1145/1654059.1654113

[22] A. T. de Hoop, "A modification of Cagniard's method for solving seismic pulse problems," *Appl. Sci. Res. B*, vol. 8, no. 1, pp. 349–356, Dec. 1960.

[23] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCTOOLKIT: Tools for performance analysis of optimized parallel programs," *Concurrency Comput., Pract. Exper.*, vol. 22, no. 6, pp. 685–701, 2010.

[24] T. Barros, J. B. Fernandes, I. A. Souza-de Assis, and S. X. de Souza, "Auto-tuning of 3D acoustic wave propagation in shared memory environments," in *Proc. 1st EAGE Workshop High Perform. Comput. Upstream Latin Amer.*, 2018. [Online]. Available: http://www.earthdoc.org/publication/publicationdetails/?publication=94579. doi: 10.3997/2214-4609.201803072.

[25] S. Xavier-de Souza, J. A. K. Suykens, J. Vandewalle, and D. Bollé, "Coupled simulated annealing," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 40, no. 2, pp. 320–335, Jul. 2010.

[26] C. Andreolli, P. Thierry, L. Borges, C. Yount, and G. Skinner, "Genetic algorithm based auto-tuning of seismic applications on multi and manycore computers," in *Proc. EAGE Workshop High Perform. Comput. Upstream*, 2014. doi: 10.3997/2214-4609.20141920.

[27] C. Andreolli, P. Thierry, L. Borges, G. Skinner, and C. Yount, "Characterization and optimization methodology applied to stencil computations," in *High Performance Parallelism Pearls*, J. Jeffers and J. Reinders, Eds. Boston, MA, USA: Elsevier, 2015, ch. 23, pp. 377–396. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128021187000236

[28] A. C. Sena, A. P. Nascimento, C. Boeres, V. Rebello, and A. Bulcao, "An approach to optimise the execution of RTM algorithm in multicore machines," in *Proc. IEEE 7th Int. Conf. eScience*, Dec. 2011, pp. 403–410.

[29] S. Hofmeyr, J. A. Colmenares, C. Iancu, and J. Kubiatowicz, "Juggle: Proactive Load Balancing on Multicore Computers," in *Proc. 20th Int. Symp. High Perform. Distrib. Comput. (HPDC)*, 2011, pp. 3–14.

[30] M. Tchiboukdjian, V. Danjean, T. Gautier, F. Le Mentec, and B. Raffin, "A work stealing scheduler for parallel loops on shared cache multicores," in *Euro-Par 2010 Parallel Processing Workshops* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2011.

[31] S. Imam and V. Sarkar, "Load balancing prioritized tasks via work-stealing," in *Euro-Par 2015: Parallel Processing*, J. L. Träff, S. Hunold, and F. Versaci, Eds. Berlin, Germany: Springer, 2015, pp. 222–234.

[32] S. K. Khaitan, J. D. McCalley, and A. Somani, "Proactive task scheduling and stealing in master-slave based load balancing for parallel contingency analysis," *Electr. Power Syst. Res.*, vol. 103, pp. 9–15, Oct. 2013.

[33] R. K. Tesser, L. L. Pilla, F. Dupros, P. O. A. Navaux, J.-F. Méhaut, and C. Mendes, "Improving the performance of seismic wave simulations with dynamic load balancing," in *Proc. 22nd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process. (PDP)*, Feb. 2014, pp. 196–203. [Online]. Available: https://doi.org/10.1109/PDP.2014.37

[34] R. K. Tesser, L. M. Schnorr, A. Legrand, F. Dupros, and P. O. A. Navaux, "Using simulation to evaluate and tune the performance of dynamic load balancing of an over-decomposed geophysics application," in *Euro-Par 2017: Parallel Processing*, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, Eds. Cham, Switzerland: Springer, 2017, pp. 192–205.

[35] R. K. Tesser, L. M. Schnorr, A. Legrand, F. C. Heinrich, F. Dupros, and P. O. Navaux, "Performance modeling of a geophysics application to accelerate over-decomposition parameter tuning through simulation," *Concurrency Comput., Pract. Exper.*, vol. 31, no. 11, p. e5012, 2018.

[36] E. L. Padoin, M. Castro, L. L. Pilla, P. O. A. Navaux, and J. Méhaut, "Saving energy by exploiting residual imbalances on iterative applications," in *Proc. 21st Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2014, pp. 1–10.

[37] E. L. Padoin, L. L. Pilla, M. Castro, P. O. A. Navaux, and J.-F. Méhaut, "Exploration of load balancing thresholds to save energy on iterative applications," in *High Performance Computing*, C. J. Barrios Hernández, I. Gitler, and J. Klapp, Eds. Cham, Switzerland: Springer, 2017, pp. 76–88.
[38] R. Sharma and P. Kanungo, "Dynamic load balancing algorithm for heterogeneous multi-core processors cluster," in *Proc. 4th Int. Conf. Commun. Syst. Netw. Technol.*, Apr. 2014, pp. 288–292.
[39] G. Zheng, A. Bhatelé, E. Meneses, and L. V. Kalé, "Periodic hierarchical load balancing for large supercomputers," *Int. J. High Perform. Comput. Appl.*, vol. 25, no. 4, pp. 371–385, 2011.
[40] V. Martínez, D. Michéa, F. Dupros, O. Aumage, S. Thibault, H. Aochi, and P. O. Navaux, "Towards seismic wave modeling on heterogeneous many-core architectures using task-based runtime system," in *Proc. Int. Symp. Comput. Archit. High Perform. Comput.*, 2016, pp. 1–8.
[41] S. K. Khaitan and J. D. Mccalley, "SCALE: A hybrid MPI and multithreading based work stealing approach for massive contingency analysis in power systems," *Electr. Power Syst. Res.*, vol. 114, pp. 118–125, Sep. 2014.
[42] S. Mor and N. Maillard, "Dynamic workload balancing deques for branch and bound algorithms in the message passing interface," *Int. J. High Perform. Syst. Archit.*, vol. 3, pp. 77–86, May 2011.
[43] S. Li, J. Hu, X. Cheng, and C. Zhao, "Asynchronous work stealing on distributed memory systems," in *Proc. 21st Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process. (PDP)*, Feb. 2013, pp. 198–202.
[44] V. Kumar, K. Murthy, V. Sarkar, and Y. Zheng, "Optimized distributed work-stealing," in *Proc. 6th Workshop Irregular Appl., Archit. Algorithms (IA3)*, Nov. 2016, pp. 74–77.
[45] H. Fu, M. Gorentla Venkata, S. Salman, N. Imam, and W. Yu, "SHMEM-Graph: Efficient and balanced graph processing using one-sided communication," in *Proc. 18th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2018, pp. 513–522.
[46] H. Essafi and P. Hède, "FRAMSTIM: Framework for large scale multimedia content feature extraction based on MPI one-sided communication," in *Proc. 2nd Int. Conf. Internet Things, Data Cloud Comput. (ICC)*, 2017, pp. 41:1–41:6. [Online]. Available: http://doi.acm.org/10.1145/3018896.3018936
[47] S. Ghosh, J. R. Hammond, A. J. Peña, P. Balaji, A. H. Gebremedhin, and B. Chapman, "One-sided interface for matrix operations using MPI-3 RMA: A case study with elemental," in *Proc. 45th Int. Conf. Parallel Process. (ICPP)*, Aug. 2016, pp. 185–194.
[48] A. Lazzaro, J. VandeVondele, J. Hutter, and O. Schütt, "Increasing the efficiency of sparse matrix-matrix multiplication with a 2.5 D algorithm and one-sided MPI," in *Proc. Platform Adv. Sci. Comput. Conf. (PASC)*, 2017, pp. 3:1–3:9. [Online]. Available: http://doi.acm.org/10.1145/3093172.3093228
[49] A. Vishnu and K. Agarwal, "Large scale frequent pattern mining using MPI one-sided model," in *Proc. IEEE Int. Conf. Cluster Comput. (ICCC)*, Sep. 2015, pp. 138–147.
[50] R. Abdelkhalek, H. Calandra, O. Coulaud, G. Latu, and J. Roman, "Fast seismic modeling and reverse time migration on a graphics processing unit cluster," in *Concurrency Comput., Pract. Exper.*, vol. 24, no. 7, pp. 739–750, 2012.
[51] A. Qawasmeh, M. R. Hugues, H. Calandra, and B. M. Chapman, "Performance portability in reverse time migration and seismic modelling via OpenACC," *Int. J. High Perform. Comput. Appl.*, vol. 31, no. 5, pp. 422–440, 2017.
[52] S. K. Akanksha and G. N. Kumar, "Parallelization of reverse time migration using MPI+OpenMP," in *Proc. Int. Conf. Adv. Commun. Control Comput. Technol. (ICACCCT)*, May 2017, pp. 695–697.
[53] C. Chu and P. L. Stoffa, "A pseudospectral-finite difference hybrid approach for large–scale seismic modeling and RTM on parallel computers," in *Proc. SEG Tech. Program Expanded Abstr.*, 2008, pp. 2087–2091.
[54] M. Perrone, L. K. Liu, L. Lu, K. Magerlein, C. Kim, I. Fedulova, and A. Semenikhin, "Reducing data movement costs: Scalable seismic imaging on blue gene," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2012, pp. 320–329.
[55] C. Chu, P. L. Stoffa, and R. Seif, "3D seismic modeling and reverse-time migration with the parallel Fourier method using non-blocking collective communications," in *Proc. SEG Tech. Program Expanded Abstr.*, 2009, pp. 2677–2681.
[56] L. Lu and K. Magerlein, "Multi-level parallel computing of reverse time migration for seismic imaging on blue gene/Q," in *Proc. 18th ACM SIGPLAN Symp. Principles Pract. Parallel Program. (PPoPP)*, 2013, pp. 291–292. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442550
[57] S. R. Paul, M. Araya-Polo, J. Mellor-Crummey, and D. Hohl, "Performance analysis and optimization of a hybrid seismic imaging application," *Procedia Comput. Sci.*, vol. 80, pp. 8–18, Jan. 2016.
[58] Y. Shang, "Resilient multiscale coordination control against adversarial nodes," *Energies*, vol. 11, no. 7, p. 1844, 2018.
[59] Y. Shang, "Resilient consensus of switched multi-agent systems," *Syst. Control Lett.*, vol. 122, pp. 12–18, Dec. 2018.

**ÍTALO A. S. ASSIS** received the M.Sc. degree in computer engineering from the Universidade Federal do Rio Grande do Norte (UFRN), Brazil, in 2015,where he is currently pursuing the Ph.D. degree in computer engineering. His current research interests include high-performance computing and optimization.

**ANTÔNIO D. S. OLIVEIRA** received the M.Sc. degree in computer science from UERN/UFERSA, Brazil, in 2013. He is currently pursuing the Ph.D. degree in computer engineering with the Universidade Federal do Rio Grande do Norte. His current research interest includes high-performance computing and scheduling algorithms.

**TIAGO BARROS** received the B.S., M.Sc., and Ph.D. degrees from the Universidade Estadual de Campinas–Brazil, in 2008, 2012, and 2018, respectively, all in electrical engineering. He is currently a Postdoctoral Researcher in computer engineering with the Universidade Federal do Rio Grande do Norte, Brazil. His current research interests include digital signal processing applied to seismic data analysis, and computational geophysics with a focus on seismic processing and inversion, especially, in tomographic and waveform inversion methods.

**IDALMIS M. SARDINA** received the M.Sc. degree in computational modeling from the Universidade do Estado do Rio de Janeiro (UERJ), Brazil, in 2000, and the Ph.D. degree in computer science from the Universidade Federal Fluminense (UFF), Brazil, in 2010. She is currently a Professor with the School of Science and Technology and a Researcher with the Laboratory of Parallel Architectures for Signal Processing (LAPPS), Digital Metropolis Institute (IMD), Universidade Federal do Rio Grande do Norte (UFRN), Brazil. Her current research interest includes computer science, with emphasis on parallel and distributed processing.

**CALEBE P. BIANCHINI** received the M.Sc. degree in computer science from UFSCar, Brazil, in 2002, and the Ph.D. degree in computer engineering from USP, Brazil, in 2009. He has been involved in a variety of projects in the last 15 years. Most of them are research and development projects in the HPC field, such as Intel Parallel Computing Center (IPCC/UNESP). He is currently involved in MackCloud Project, which is a Scientific Computer Center at the Universidade Presbiteriana Mackenzie. He is also an Associate Professor with the Universidade Presbiteriana Mackenzie leading small in-house projects in HPC and software engineering. His current research interests include HPC and grid computing (former), and software engineering and networks (latter).

**SAMUEL XAVIER-DE-SOUZA** (SM'19) was born in Natal, Brazil. He received the degree in computer engineering from the Universidade Federal do Rio Grande do Norte (UFRN), Brazil, in 2000, and the Ph.D. degree in electrical engineering from the Katholieke Universiteit Leuven, Belgium, in 2007. He was a Software/Hardware Engineer with IMEC, Belgium, and a High-Performance Computing Consultant with the Flemish Supercomputing Center, Belgium. In 2009, he joined the Department of Computer Engineering and Automation, UFRN, where he is currently an Associate Professor. He is also a Founder and the Director of the High-Performance Computing Center-NPAD, UFRN. In 2016, he received the Royal Society-Newton Advanced Fellowship.