Tuning a CPU-Based Stencil Computation in a DPC++ Multi-Device Environment

Tiago Conceição Oliveira[1*], Murilo Boratto[1], Antônio Horácio Rodrigues[1], Orlando Mota Pires[1], Leonardo Rodrigues Soares[1]

[1]SENAI CIMATEC, Supercomputing Center; Salvador, Bahia, Brazil

Reverse Time Migration (RTM) uses the finite-difference (FD) method to compute numerical approximations for the acoustic wave equation. It is a computational bottleneck for RTM applications and therefore needs to be optimized to guarantee timely results and efficiency when allocating resources for hydrocarbon exploration. This article describes our experience reengineering a migrated CUDA-based RTM code to SYCL into a multi-device RTM.

Keywords: Multi-Device RTM. OneAPI. SYCL. Heterogeneous Computing.

Introduction

RTM method, which stands for Reverse Time Migration, was first proposed by Baysal and colleagues [1] and McMechan [2] in 1983 but gained attention recently due to advances in computer capabilities [3]. It is a two-way wave equation that intends to build a high-quality and accurate image of the subsurface. Its bottleneck concerns to the need to compute two wave fields, one for the source (computed as a forward propagation) and other for the receiver (computed as a backward propagation) for each data point in the velocity model and for each shot (accurate data usually have thousands of shots) [4]. The Finite Difference Time Domain is a standard numerical solution used to model the wave propagation in RTM. The stencil data arrangement is used to compute this approximation at each grid point.

Despite the advantages intrinsic to the method, two significant computational difficulties characterize it: The high number of floating-point operations during the propagation step and the difficulty storing the wavefields in the main memory. Engineering seeks to explore both the intrinsic parallelism of tasks and the optimization of computational resources, designing solutions capable of running on different accelerated processing units, for example, to mitigate the effect of these problems. The optimization of this method represents an excellent economic advantage for exploration geophysics since it reduces the chances of errors in well-drilling.

Materials and Methods

Reengineering the DPC++ Based RTM Application for Multi-GPU Execution

In this paper, we consider a reference RTM algorithm written in C++. Algorithm 1 demonstrates the simplified execution of the RTM algorithm (Figure 1). The vector P store the state of pressure points in different time steps. The stencil computation needed for solving the finite-difference method dominates the total runtime; therefore, this is the main kernel to be accelerated on GPU devices. In this scenario, forward and backward propagation happens one after another in a serialized way. In these implementations, there is a dependence between the two propagation: during forward time steps, the source wavefields are stored and used in backward propagation to provide wavefields reconstruction.

We proposed to explore a multi-GPU environment. So, we worked on the dependencies between forward and backward propagation to execute them simultaneously in two GPU

Received on 15 December 2022; revised 10 January 2023.
Address for correspondence: Tiago Conceição Oliveira. Av Comendador Franco, No. 8115, Condomínio Terra, apt 22, bloco A4. Zipcode: 81560001. Curitiba PR. E-mail: tiagocompuesc@gmail.com.

J Bioeng. Tech. Health 2023;6(Suppl.1):24-31.

devices. This is only possible by building strategies to the previous model and storing the observed data to be used in the backward propagation without needing a reconstruction. Algorithm 2 presents a pseudocode describing the main steps for building RTM code without dependencies (Figure 2). The data used by RTM are generated previously and are not described in Algorithm 2. The method responsible to generates data is called modeling. Some steps were suppressed in Algorithm 2 and presented as functions, but those methods are described in Algorithm 1.

After building a code without dependencies, it is possible to propose an implementation of the RTM code using multiple devices. Figure 3 shows

Figure 1. Algorithm 1.

Algorithm 1 RTM Base CPU Version

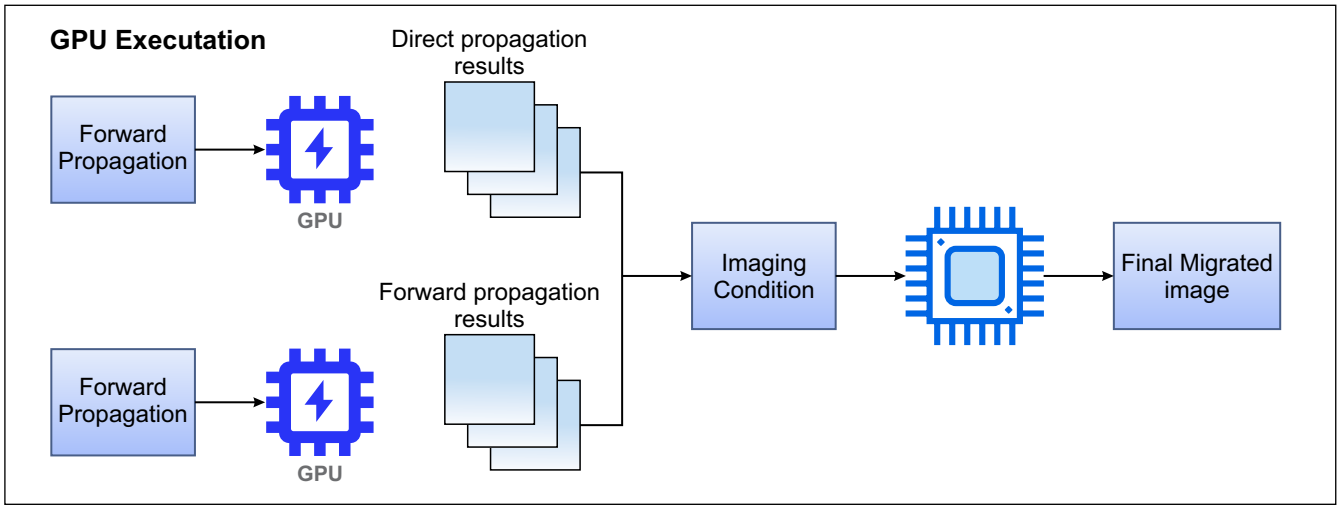
```
1:  fd_init();                                              ▷ Used to initialize all structures
2:  for it  0 to nt do                                      ▷ Loop to forward propagation
3:      stencil(p, pp, vel2, nx, nz)                        ▷ Calculate Finite Differences
4:      src(pp)                                             ▷ Add shot source
5:      for ix = 0 to nx do                                 ▷ Save data to use in wave reconstruction
6:          data[ix][it] = p[ix+nxb][gz]
7:      end for
8:      for ix = 0 to nx do                                 ▷ Save wavefield to use in image condition
9:          for iz = 0 to nz do
10:             swf[it][ix][iz] = p[ix+pad][iz+pad]
11:         end for
12:     end for
13: end for
14: for it = 0 to nt do                                     ▷ Loop to backward propagation
15:     stencil(p, pp, vel2, nx, nz)
16:     for ix = 0 to nx do                                 ▷ Reading wavefield
17:         p[ix+nxb][gz] +=  data[ix][it]
18:     end for
19:     for ix = 0 to nx do                                 ▷ Apply image condition
20:         for iz = 0 to nz do
21:             imloc[ix][iz] += swf[nt-it-1][ix][iz] * p[ix+pad][iz+pad]
22:         end for
23:     end for
24: end for
```

Figure 2. Algorithm 2.

Algorithm 2 RTM Base Without Dependencies Version

```
1:  fd_init();                                              ▷ Used to initialize all structures
2:  dobs = read_observed_data()                             ▷ Read precomputed observed data
3:  for it = 0 to nt do                                     ▷ Loop to forward propagation
4:      stencilIp, pp, vel2, nx, nz)                        ▷ Calculate Finite Differences
5:      src(pp)                                             ▷ Add shot source
6:      saving_wvf(sfw, p)                                  ▷ Saving forward wavefield
7:  end for
8:  for it = 0 to nt do                                     ▷ Loop to Backward propagation
9:      stencil(p, pp, vel2, nx, nz)
10:     for ix = 0 to nx do                                 ▷ Adding precomputed observed data to pressure field
11:         p[ix+nxb][gz] += dobs[ix][it]
12:     end for
13:     saving_wvf(rfw, p)                                  ▷ Saving backward wavefield
14: end for
15: imloc = apply_image_condition(swf, rwf)
```

Figure 3. Multi-GPU without dependence 2D-RTM flowchart.



The figure shows a structure of independent foward propagation and backward propagation computation in multiple devices. Synchronization is made only in the CPU.

a simplified version of the new code structure. The approach is that propagations are independently processed, and synchronization between devices is only necessary at the end of the operation. Finally, the image condition is applied. The result of both propagations, forward and backward, is the RTM migration image.

One way to control the flow between devices is to use threads. In this approach, we use threads to process propagations in parallel. A parallel zone starts with two threads for executing and synchronizing the RTM model. After the threading process on the GPU ends, the data return to the CPU. After parallel threads, the CPU computes the image condition, as described in Algorithm 3 (Figure 4).

The SYCL Thread Hierarchy

The thread hierarchy exploration aims to maximize the occupancy of the GPU resources. In an SYCL kernel, the programmer can affect the work distribution by structuring the kernel with proper workgroup size and sub-group size and organizing the work items for efficient vector execution (Figure 5). Writing efficient vector kernels is crucial for high performance on GPU.

Figure 4. Algorithm 3.

Algorithm 3 RTM Base CPU Version

```
1:  fd_init();                                  ▷ Used to initialize all structures
2:  dobs = read_observed_data()                 ▷ These data is the input for the backward propagation
3:  init_GPUs()                                 ▷ Init all structures to use Multi-GPU
4:  omp_set_num_threads(2)                      ▷ Each propagation runs in a different thread
5:  #pragma omp parallel{
6:  tid = get_thread_num();
7:  if tid == 0 then
8:      forward(P, PP, Vel, swf, tid)           ▷ Run in GPU 1
9:  else
10:     backward(PR,PPR, Vel, rwf, tid)         ▷ Run in GPU 2
11: end if
12: }
13: #pragma omp barrier
14: imloc = apply_image_condition(swf, rfw)
```

Laplacian Kernel

Laplacian is the core of the RTM algorithm, which is highly time-consuming in the application. Figure 6 shows how the prominent part of the Laplacian was implemented using the SYCL thread hierarchy. This implementation focuses on workgroup and sub-group size selection. SYCL does not provide a mechanism to set the number of threads directly in a workgroup. However, it can use workgroup size and SIMD sub-group size to set the number of threads. Thread contexts are easy to utilize, starting with selecting the number of threads in a workgroup (Figure 6, lines 4 to 9).

Results and Discussion

Computational Results

In experiments using base serial code, execution will use only one CPU core. We use two devices in experiments using the GPU version of the code. Table 1 describes a GPU device and a CPU device. An Intel® DevCloud node was chosen for the preliminary experiments. Table 1 shows the hardware description of the node.

Reference Input Data

Since the vector P represents the pressure field, its characteristics are directly related to the characteristics of the initial velocity model. Both are represented as 2-D matrices with the exact dimensions. We begin considering three seismic velocity models illustrated by Figure 7 with *nx* = 151 and *nz* = 151, Figure 8 with *nx* = 369 and *nz* = 375, Figure 9 with *nx* = 351 and *nz* = 367, as the base models for execution and migration evaluation.

The models presented here were our reference to build matrix P, which is the input parameter of the function that performs the stencil.

Experimental Results

The results of the experiments are available in Table 2. A comparison of the serial and parallel versions is also available. As we can see, the multi-device RTM is faster for all models tested.

Figure 10 is a serial reference for a parallel approach, while Figure 11 shows an image migration using the multi-device RTM code for a 3-layer model (Figure 7). Figure 12 is a subtraction for Figures 10 and 11 using the *farith package*. Figure 14 shows an image migration using the multi-device RTM code for the *SPluto model* (Figure 9).

Figure 13 is a serial reference for a parallel approach, and Figure 15 is a subtraction for Figures 13 and 14 using the *SPluto model*. Figures 12 and 15 show that the migrated multi-device image version does not differ significantly from the CPU-migrated image version.

Figure 5. The relationship diagram among ND-Range, work-group, sub-group, and work-item.

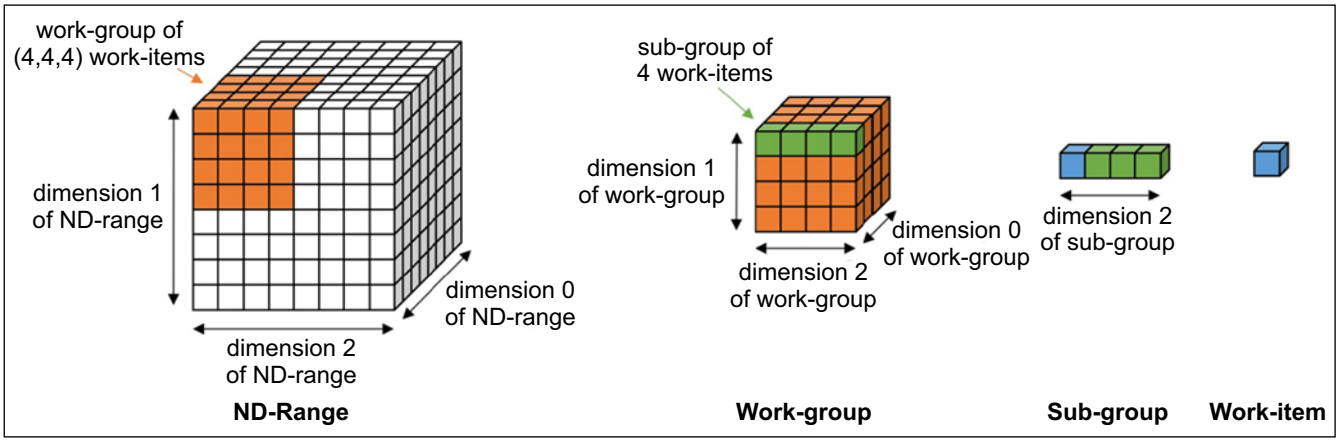


Figure 6. Kernel for laplacian on GPU using SYCL with multiple thread hierarchy.

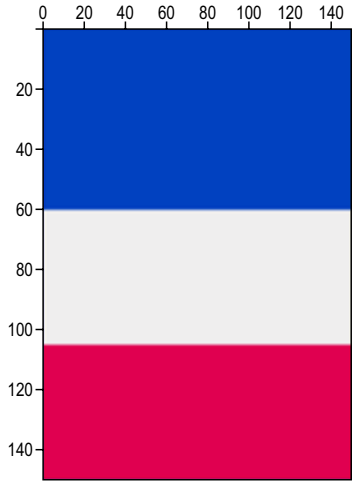
```
void kernel_lap(int order, int nx, int nz, float * __restrict__ p, float * __restrict__ lap,
    float * __restrict__ coefsx, float * __restrict__ coefsz, sycl::nd_item<2> item_ct1){
    int half_order=order/2;
    int i = half_order +
            item_ct1.get_group(0) * item_ct1.get_local_range().get(0) +
            item_ct1.get_local_id(0); // Global row index
    int j = half_order +
            item_ct1.get_group(1) * item_ct1.get_local_range().get(1) +
            item_ct1.get_local_id(1); // Global column index
    int mult = i*nz;
    int aux;
    float acmx = 0, acmz = 0;

    if(i<nx - half_order)
    {
        if(j<nz - half_order)
        {
            for(int io=0;io<=order;io++)
            {
                aux = io-half_order;
                acmz += p[mult + j+aux]*coefsz[io];
                acmx += p[(i+aux)*nz + j]*coefsx[io];
            }
            lap[mult +j] = acmz + acmx;
            acmx = 0.0;
            acmz = 0.0;
        }
    }

}
```

The figure shows a snippot for forward propagation and nackward propagation computation in multiple devices.

Table 1. DevCloud node description.

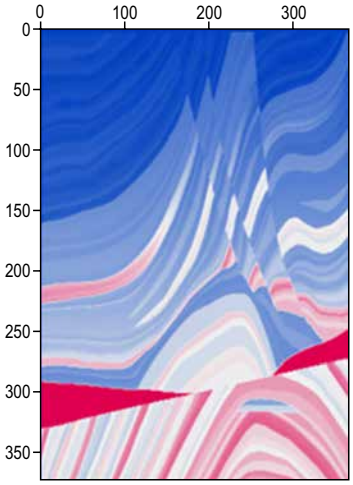
	Description
CPU	Intel(R) Xeon(R) Gold 6336Y CPU @ 2.40GHz
GPU	Intel® HD Graphics P630
Memory	128 GB
GCC	9.4.0-1ubuntu1~20.04.1
DPC++	2022.0.0 (2022.0.0.20211123)

Figure 7. seismic velocity 1.



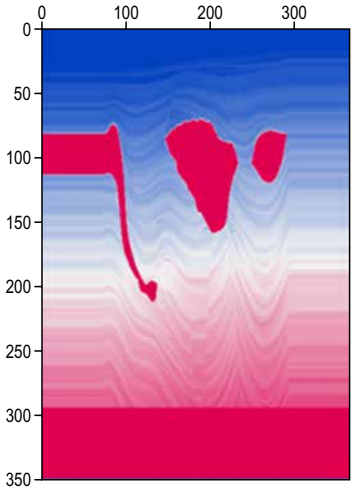
nx = 151 and *nz* = 151

Figure 8. seismic velocity 2.



nx = 369 and *nz* = 375

Figure 9. seismic velocity 3.



nx = 351 and *nz* = 367

Table 2. A CPU serial RTM and multi-device parallel RTM execution time comparison on DeCloud environment.

Velocity Model	Serial Time(s)	Parallel Time(s)
3 layer	4.76	1.08
Marmousi	42.93	3.6
SPluto	54.91	5.5

Figure 10. CPU-based RTM migration for 3 layer models.

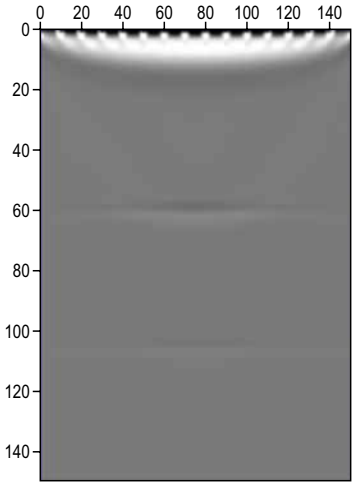


Figure 11. Muti-device-based RTM migration for 3 layer models.

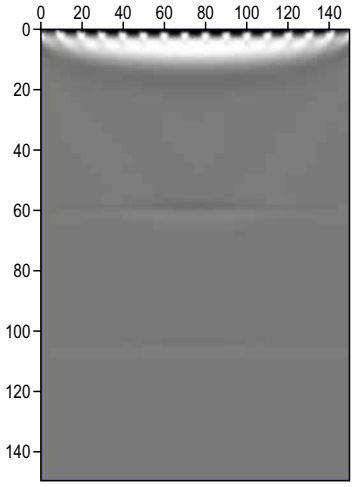


Figure 12. Difference between CPU and Multi-device RTM migration for 3 layer models.

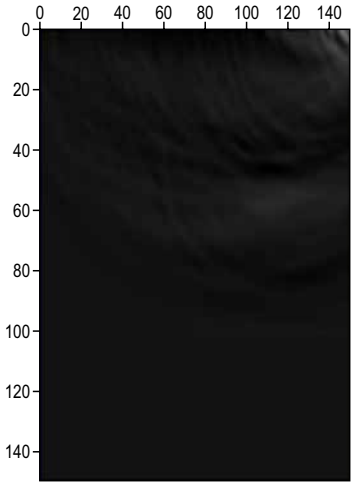


Figure 13. CPU-based RTM migration for the SPluto model.

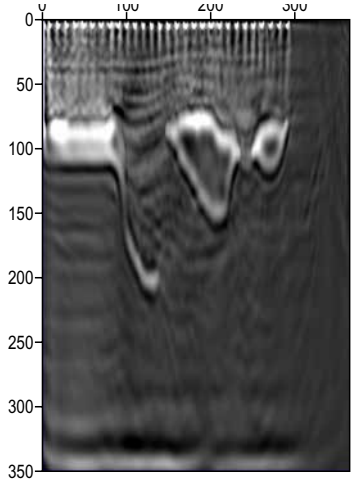


Figure 14. Multi-device RTM migration for the SPluto model.

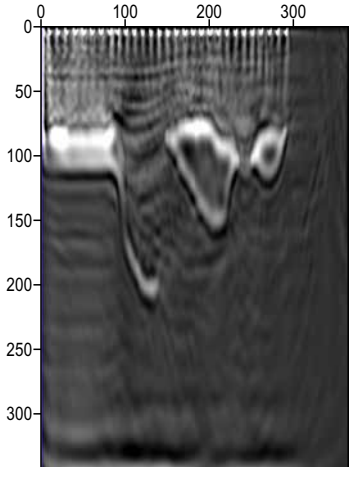
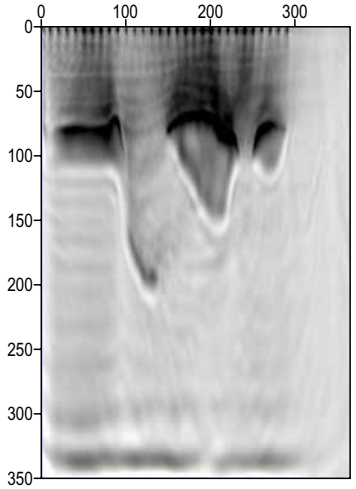


Figure 15. Difference between CPU and Multi-Device RTM migration for SPluto model.



Conclusion

The first step involves migrating a serial base RTM, already changed to provide simultaneous forward/backward propagation, to DPC++ and the improvements related to the DPC++ source code implementation. This step was achieved, as shown in the image migration in the results section. In the second step, researchers focused on possible optimizations to achieve those objectives: the application was rewritten, focusing on thread hierarchy, as shown in Figure 6.

Finally, the Multi-devices RTM application was successfully developed and tested. Intel® tools also helped to decide what resource to use and correctness check. The Multi-devices RTM code is a small workload but highly time-consuming. Further work could explore larger workloads, aiming to use the whole GPU memory and the main memory.

Acknowledgments

We thank the editors and reviewers for their constructive comments and suggestions. We would like to thank FINEP for the support of the Supercomputing Center of SENAI CIMATEC.

References

1. Baysal E, Kosloff DD, Sherwood JW. Reverse time migration. Geophysics 1983;48(11):1514–1524.
2. McMechan GA. Migration by extrapolation of time-dependent boundary values. Geophysical Prospecting 1983;31(3):413–420.
3. Yang P, GaoJ, WangB. RTM using effective boundary saving: A staggered grid GPU implementation. Computers & Geosciences 2014;68:64–72.
4. Zhang L, Slob E. Free-surface and internal multiple elimination in one step without adaptive subtraction. Geophysics 2019;84(1):A7–A11.
5. Zhang L, Slob E. Marchenko multiple elimination of a laboratory example. Geophysical Journal International 2020;221(2):1138–1144.