# Scalable Simulation of 3D Wave Propagation in Semi-Infinite Domains Using the Finite Difference Method on a GPU Based Cluster

**Thales Luis Rodrigues Sabino**[1], **Marcelo Zamith**[1], **Diego Brandão**[1]
**Anselmo Montenegro**[1], **Esteban Clua**[1], **Maurício Kischinhevksy**[1]
**Regina C.P. Leal-Toledo**[1], **Otton T. Silveira Filho**[1], **André Bulcão**[2]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
Rua Passo da Pátria 156 - Niterói - Rio de Janeiro - Brazil

[2]Cenpes, Petrobrás
Ilha do Fundão, Rio de Janeiro - Brazil

{tsabino, mzamith, dbrandao, anselmo}@ic.uff.br

{esteban, kisch, leal, otton}@ic.uff.br, bulcao@petrobras.br

***Abstract.*** *The scattering of acoustic waves has been of practical interest for the petroleum industry, mainly in the determination of new oil deposits. A family of computational models that represent this phenomenon is based on finite difference methods. The simulation of these phenomena demands a high computational cost and large amounts of available memory. In this work we employ GPU based cluster environment for the development of scalable solvers for a 3D wave propagation problem with finite difference methods.*

## 1. Introduction

The scattering of acoustic waves has been of practical interest for oil and gas industries, mainly the determination of new oil deposits. The scattering process is commonly described with Hyperbolic Partial Differential Equations (PDEs). This kind of equation describes a large variety of physical phenomena governed by wave behavior.

In order to solve such PDEs, the use of numerical methods is employed. Try to solve the scattering problem with a minimum level of precision requires a huge computational effort. Solutions based on multi core CPU clusters have been widely used with successful results [Balevic et al. 2008]. With the advent of GPU computing technology, the numerical solution of PDEs can be done from 20 to 200 times faster than a traditional CPU implementation depending on the problem ([Zamith et al. 2010] and [Brandão et al. 2010]).

GPU computing has become an important choice for many parallel computational problems, since GPUs are potentially more powerful for massively parallel computations than CPUs. Cluster based environments usually needs to work with some kind of communication system. Message interfaces such MPI standard have been extensively used with numerical methods to simulate large complex scattering problems. The use of GPUs in MPI based clusters is almost automatic, since MPI deals with communication steps when GPUs deals with intense computational steps.

One needs to be aware that GPUs are made for graphical purposes and have some limitations. GPU works faster with single precision floating point numbers. Double precision support only became available with recent releases. In order to achieve maximum performance, GPUs dedicates more transistors to arithmetic units than control units so it is important to coalesce memory reads and writes. GPUs have a limited memory (6 GB for NVIDIA Tesla C2070).

Industry scale problems may require use of hundreds of GB of memory and a single GPU may not be suffice to solve such problems timely, not for computational power, but for memory limitations. Taking, for instance, a 3D scattering of acoustic wave problem with a domain discretized in a cube of side 700 requires, at least, 4GB of storage capacity and this requirement grows by a cubic factor.

Many different non-graphical computation, simulation and numerical problems, including Protein Structure Prediction [Langdon and W.Banzhaf 2008], Solution of Linear Equation Systems [Bolz et al. 2003], Options Pricing [Abbas-Turki and Lapeyre 2009], Flow Simulation [Rozen et al. 2008], Wave Propagation [Balevic et al. 2008], [Michea and D.Komatitsch 2010], have been solved in GPUs.

This paper presents a parallel implementation for the scattering of 3D acoustic waves in semi-infinite non-homogeneous domains in a heterogeneous GPUs based cluster capable of divide the problem domain in a way that each domain's partition is handled by one GPU making industry scale wave propagation problems solvable by up two orders of magnitude faster.

## 2. Acoustic Wave Equation

The wave equation is a second order linear differential equation that describes the behavior of sound waves over time. The acoustic wave field is described by $P(x, y, z, t)$ and $u(x, y, z, t)$, where $P$ is the pressure field the $u$ is the particle's displacement. The relation between $P$ and $u$ is given by $P(x, y, z, t) = -k\nabla^2 u(x, y, z, t)$. Thus, the 3D wave equation is given by:

$$\frac{\partial^2 P(x, y, z, t)}{\partial t^2} = c^2(x, y, z)\nabla P(x, y, z, t) + f(x, y, z, t) \tag{1}$$

where $x$, $y$ and $z$ are cartesian coordinates, $t$ is time, $c$ is the velocity acoustic wave and $f(x, y, z, t)$ is the source term.

To numerically solve the partial differential equation (Eq. 1), we first discretize it into a set of finite-difference (FD) equations by replacing partial derivatives with central differences. A central-difference approximation can be derived from the Taylor series. Thus, using a second order approximation for space and time, assuming $h = \Delta x = \Delta y = \Delta z$ and $t = n\Delta t$, Eq. 1 is rewritten as:

$$\begin{aligned} P^{n+1}_{(i,j,k)} = 2P^{n}_{(i,j,k)} - P^{n-1}_{(i,j,k)} + A\left[P^{n}_{(i-1,j,k)} + P^{n}_{(i+1,j,k)}\right] - \\ - A\left[6P^{n}_{(i,j,k)} + P^{n}_{(i,j-1,k)} + P^{n}_{(i,j+1,k)} + P^{n}_{(i,j,k-1)} + P^{n}_{(i,j,k+1)}\right] \end{aligned} \tag{2}$$

where $A = \left(\frac{c(x,y,z)\Delta t}{h}\right)^2$ and $n = 1, 2, \ldots$ represents the time slice. Since the velocity field does not vary with time, $c$ is not a function of time.

## 3. CUDA and GPU Computing

Since 2006 with NVIDIA's CUDA release, the world of high-performance computing became more accessible. A GPU with hundreds of cores is now available by a tenth of the price one need to build a CPU cluster with the same number of processor cores and this cost tends to fall further.

Few years ago, scientific computing based on GPU architecture was developed using graphical shader languages together with some graphics API that allow the execution of such shaders. The programmer needed to map the problem as a set of vertices and fragments to generate a texture representing the final solution. One of the advantages of programming using CUDA API instead of conventional Shader Language is that it allows one to work with familiar concepts while developing kernels that run on GPUs.

CUDA extends existing programming languages with a set of instructions that allows code execution on NVIDIA GPUs. Such extensions expose the GPU hardware memory hierarchy. The GPU memory structure is divided as global, texture and shared. Shared memory is a small but extremely fast memory. This speed comes from its physical proximity to the processing core. Texture memory is a read only memory slower than the shared one, but is almost twice as fast as global memory.

## 4. Finite Difference Method on GPU

As described in [Brandão et al. 2010], finite difference methods implemented on GPUs are more efficient when it take advantage of shared memory. Using a single GPU one need to allocate memory that is suffice to store tree times the domain size. To compute the next step, one needs the actual and the past steps stored. Figure 1 shows a 3D finite difference explicit scheme with a second order spatial FD operator.

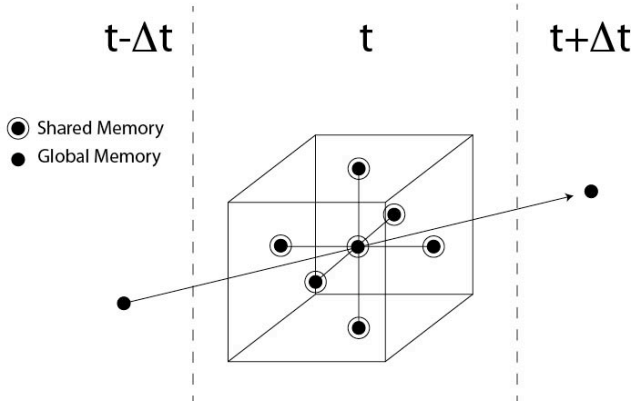


**Figure 1. Second order 3D finite difference operator. To compute the next step ($t + \Delta t$) it is necessary to fetch the actual ($t$) and the past one ($t - \Delta t$).**

A typical CUDA application workflow consists in 4 basic steps: (1) Initialize the necessary data on the host, (2) copy the data from host to device, (3) invoke the kernel that

will process the data in device and (4), read the data back from device to the host. Initially, the velocity field is allocated and copied to the device memory. Memory is allocated to contain the values for the past, present and future instants wave amplitude values. The amplitude values for $P^0_{(x,y,z)}$ and $P^1_{(x,y,z)}$ are set with a forward explicit approximation scheme. After initialization, Eq. 2 is then used for subsequent time steps.

A GPU can run millions of lightweight threads. To help the management of these threads, CUDA works with the concept of grid of thread blocks. Basically the threads are organized into thread blocks, which in turn are organized into a grid of blocks. Our approach considered that each new value $P^{n+1}_{(x,y,z)}$ is computed by only one thread. Hence, the 3D domain is divided in a set of 3D blocks, observing the constrains described in [NVIDIA 2010]. The maximum number of blocks that can be allocated is $65,535$ with $512$ threads each, giving a total of $33,553,920$ points that can be processed by a single GPU.

Shared memory is used to hold the values of some $P^n_{(x,y,z)}$ step. This avoids unnecessary reads from the global memory. Figure 1 shows that a thread must access seven amplitude values of the same instant $P^n_{(x,y,z)}$. Bringing this values to shared memory makes this accesses much faster. The values of instants $P^{n-1}_{(x,y,z)}$ are fetched direct from global memory, since all the threads of the same block will access only one position, we guarantee a coalesced memory read taking maximum performance of GPU architecture.

### 4.1. Work Division Across Multiple GPUs

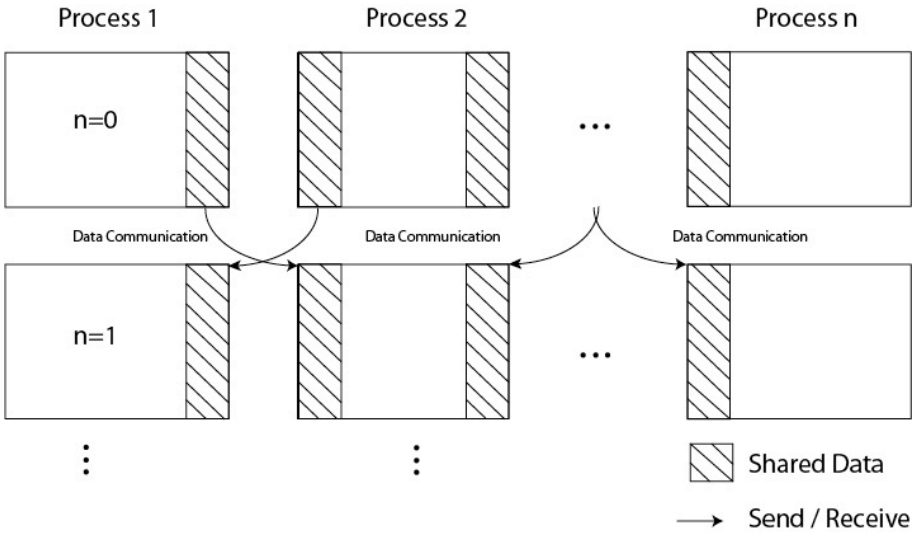


**Figure 2. Problem division across multiple devices. Each device is assumed to be handled by one CPU. Each CPU is running its own process and communication of necessary data is done using MPI when all devices finish computation of associated data.**

As we discussed before, a considered small problem for industry standards might not be handled entire by a single GPU. Our work implements a solution that consists in split problem domain in a way that each partition fits in a single GPU. This allows the use of a GPU based cluster to solve industry scale problems with the power of GPU.

Figure 2 shows how we organized the devices in an array with the data that need

to be communicated between time steps. For 3D problems, each device handles a parallelepiped whose borders need to be sent to other processes in order to have the entire array of devices solving one single problem. The division planes are chosen in way which communication data are reduced at minimum. This division ensures a maximum of two communication interfaces. Taking for instance a division scheme where a internal subportion of parallelepiped is chosen for a device, there is a need to communicate data up to six interfaces, which can become impractical for both communication costs and by memory consumption since data near communication interfaces are redundant.

The "data communication" stage is done at the of each simulation step. Processes need to communicate the edge values of each $P^{n+1}_{(x,y,z)}$ state, like the diagram shown in Figure 2. The amount of data traffic relies on the order the FD spatial operator chosen. Practical simulations uses $6^{th}$ up to $10^{th}$ order spatial operators, which can turn the communication time larger than computation time. In such situations, the final time of simulation may not be smaller as one that use a pure CPU implementation. Network switched technologies such Infiniband turn communication time almost negligible, which allows our scheme for use in practical situations.

The user might want to store the state of simulation at any instant. Take any instant $n$, the state of the entire simulation is scattered across the devices on the cluster. Knowing that a state can be composed of hundreds of GB of memory, such state may need to be written directly to a secondary memory device.

We present a communication scheme for 3D simulation of scattering of acoustic wave equation using a finite differences method that is capable of deal with industry scale problems that may not fit entire in a single GPU.

## 5. Results and Analysis

As discussed before, a considered small problem for industry standards may easily bypass a single GPU memory capacity. In order to run large finite difference simulations, we implement a communication scheme like the one shown in Figure 2. The amount of data transferred from one process to another is given mainly by the spatial order of the FD method being used. The amount of data that need to be transferred / received from one process to another is given by $x \times y \times d$, where $d$ is the method discretization order. We choose a domain division in a way that a GPU is fulfilled on its maximum capacity. One should note that doing a more refined division not necessarily generates more traffic in the network, since one process do not need to broadcast its data, only send / receive directly to / from another process.

Our test environment consists in a cluster composed by 64 quad-core CPUs and 128 NVIDIA Tesla C1060 GPUs. Each process uses a single GPU. We run simulations with different number of processes in order to evaluate communication bottlenecks. We run simulations with different domain sizes and kernel configurations. We measure the performance in giga samples per second (GSamples), meaning the number of calculations that can be performed per second, that is, $10^{-9}(nx \times ny \times nz)/(computing\_time)$, where $nx$, $ny$ and $nz$ are the number of discretized points in $x$, $y$ and $z$ axis, respectively. Our application was implemented using NVIDIA CUDA API and the communication stages are done with MPI.

The simulation configurations used are listed in Table 1. We choose these con-

figurations in a way that each GPU is used on its full capacity. For each configuration, the sizes of $xy$ planes are kept varying only the $z$-axis value. The size of $z$-axis chosen determine the number of GPUs necessary to solve the problem. This values are listed in Table 1 too.

| Simulation Configurations | | | | | |
|---|---|---|---|---|---|
| **z** | **Hosts** | **Processes** | **z** | **Hosts** | **Processes** |
| 64 | 1 | 1 | 1280 | 10 | 20 |
| 128 | 1 | 2 | 1408 | 11 | 22 |
| 256 | 2 | 4 | 1536 | 12 | 24 |
| 384 | 3 | 6 | 1664 | 13 | 26 |
| 512 | 4 | 8 | 1792 | 14 | 28 |
| 640 | 5 | 10 | 1920 | 15 | 30 |
| 768 | 6 | 12 | 2048 | 16 | 32 |
| 896 | 7 | 14 | 4096 | 32 | 64 |
| 1024 | 8 | 16 | 8192 | 64 | 128 |
| 1152 | 9 | 18 | | | |

**Table 1. Simulation configurations**

Figure 3 shows a theoretical GSample values. For each configuration, the theoretical GSample values are obtained taking the time spent for running the simulation in a single GPU times the number of GPUs that is going to be used. Figure 4 shows the measured GSamples values for the same simulations. The discrepancy between the theoretical and measured GSample values occurs because theoretical model does not counts the communication time. But, as one can see in Figure 4 the simulations scales linearly with the number of processes being used.

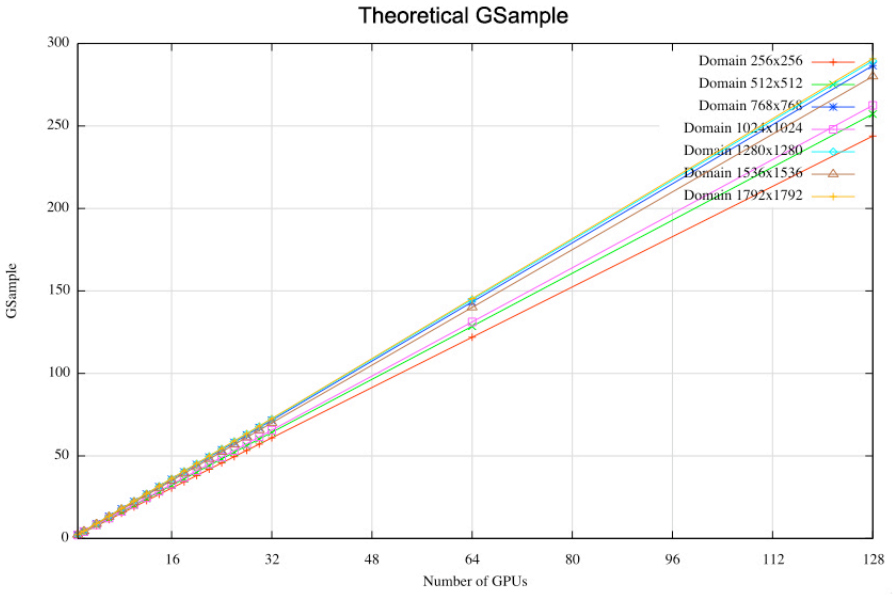


**Figure 3. Theoretical GSample result for various domain sizes.**

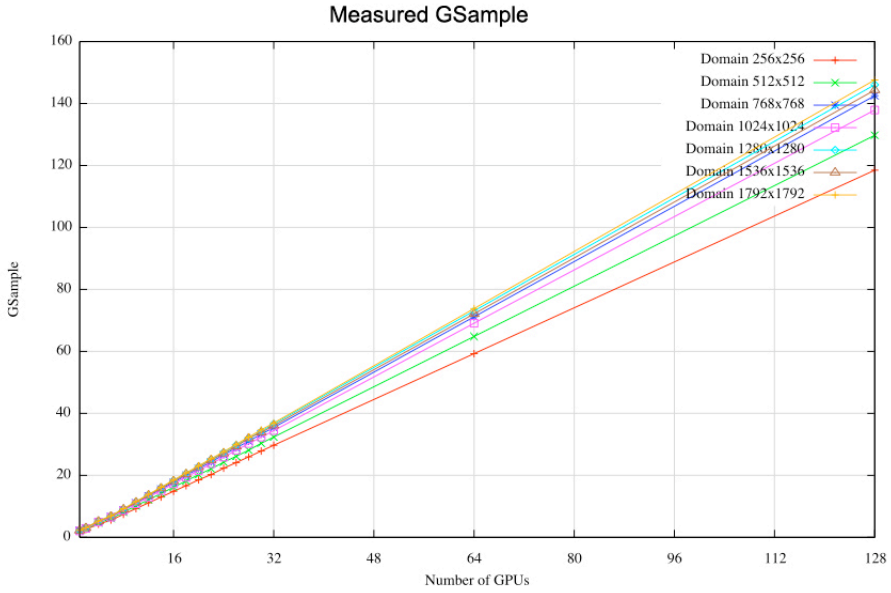


**Figure 4. Measured GSample result for various domain sizes.**

## 6. Conclusion and Future Works

Even GPUs being powerful arithmetic high-parallel multithreaded processors, its hardware memory and programming languages are limited. Due to such limitations, strategies must be designed to solve large-scale problems that may require massive storage capacity.

We present a communication scheme for 3D simulation of scattering of acoustic wave equation using a finite differences method that is capable of deal with industry scale problems that may not fit in a single GPU. We show that our implementation scales linearly with the number of GPUs being employed minus the communication time generated by MPI.

As a forthcoming work we propose the use of OpenCL with MPI in order to scale the application in a heterogeneous environment and a study on the impact of the discretization order in the final distributed execution time.

## 7. Acknowledments

The authors gratefully acknowledge Petrobás, CNPq, CAPES and FAPERJ for the financial support of this work.

## References

Abbas-Turki, L. A. and Lapeyre, B. (2009). American options pricing on multi-core graphic cards. *International Conference on Business Intelligence and Financial Engineering*, 0:307–311.

Balevic, A., Rockstroh, L., Tausendfreund, A., Patzelt, S., Goch, G., and Simon, S. (2008). Accelerating simulations of light scattering based on finite-difference time-domain method with general purpose gpus. *Computational Science and Engineering, IEEE International Conference on*, 0:327–334.

Bolz, J., Farmer, I., Grinspun, E., and Schroder, P. (2003). Sparse matrix solvers on the gpu: conjugate gradients and multigrid. In *ACM Transactions on Graphics: Proceedings of ACM SIGGRAPH*, pages 917–924.

Brandão, D., Zamith, M., Clua, E., Montenegro, A., Bulcão, A., Madeira, D., Kischinhevsky, M., and Leal-Toledo, R. (2010). Performance evaluation of optimized implementations of finite difference method for wave propagation problems on gpu architecture. In *Computer Architecture and High Performance Computing Workshops (SBAC-PADW), 2010 22nd International Symposium on*, pages 7 –12.

Langdon, W. B. and W.Banzhaf (2008). A simd interpreter for genetic programming on gpu graphics cards. In *Lecture Notes in Computer Science: Genetic Programming*, pages 73–85. Springer Berlin-Heidelberg.

Michea, D. and D.Komatitsch (2010). Accelerating a three-dimensional finite-difference wave propagation code using gpu graphics cards. *Geophysical Journal International*, 182:389–402.

NVIDIA (2010). *NVDIA - CUDA Programming Guide*. NVIDIA.

Rozen, T., Boryczko, K., and Alda, W. (2008). A gpu-based method for approximate real-time fluid flow simulation. *Machine Graphics and Vision International Journal*, 17(3):267–278.

Zamith, M. P. M., Brandão, D. N., Kischinhevsky, M., Leal-Toledo, R. C. P., Filho, O. T. S., Montenegro, A. A., and Bulcão, A. (2010). Simulation of wave propagation in semi-infinite domains using the finite difference method on a gpu based on cluster. pages 7147–7157.