Architect of an Open World™

Portage de codes sismiques sur GPU

Gunter Roeth, Mathieu Dubois
gunter.roeth@bull.net
Equipe Applications & Performances

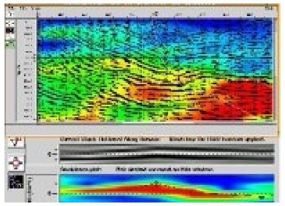
LIBERATE IT

“Upstream” workflow: Find new Oilfields ...

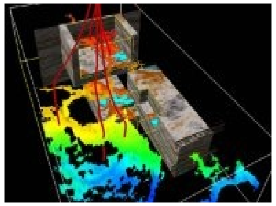
Data Acquisition

Data Management

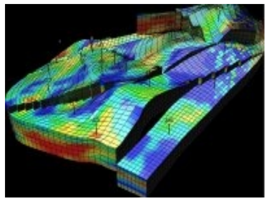
Seismic Processing



Visual Interpretation

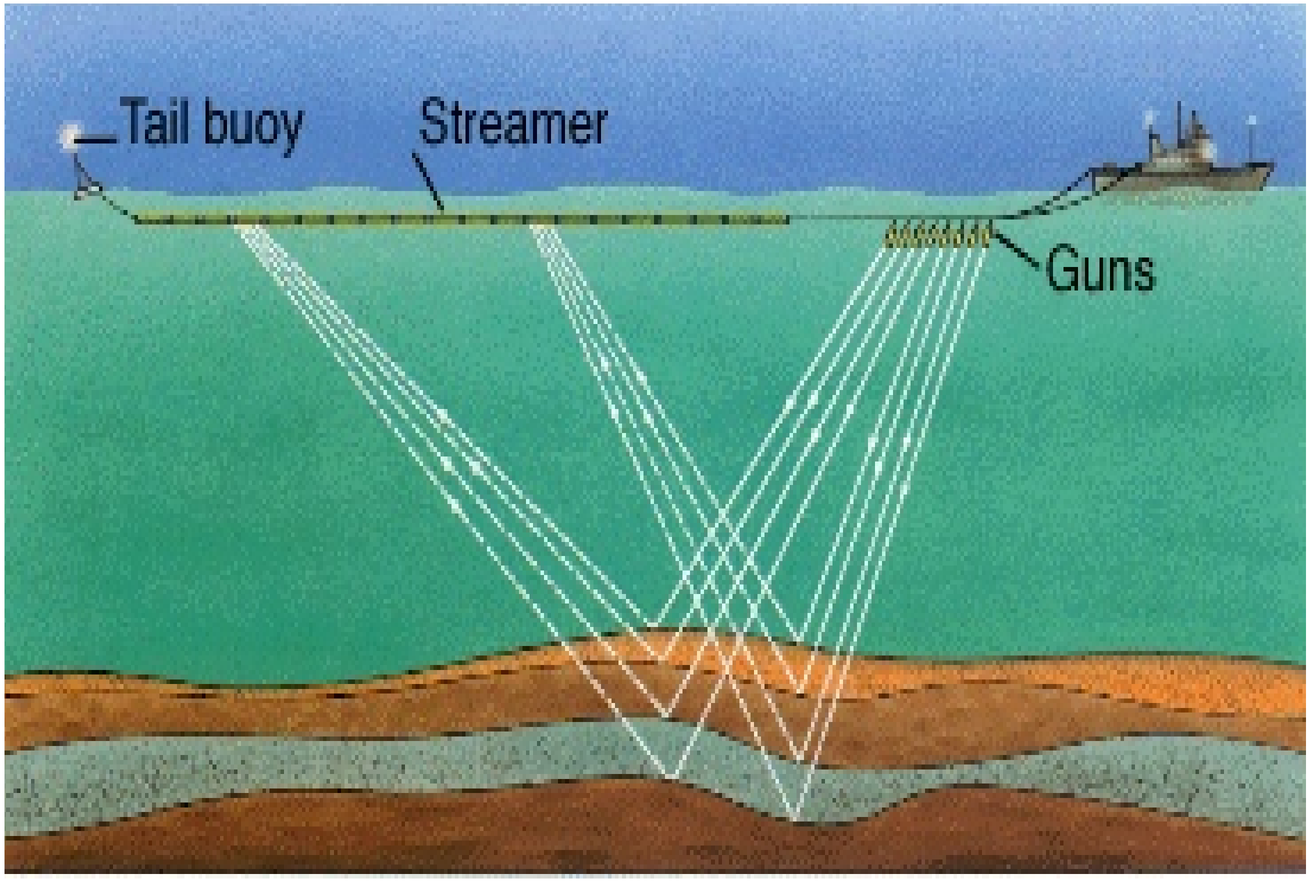


Reservoir Simulation



- Oil-companies empty their reserves and have to find new ones.
- Seismic methods are used in computer simulations to retrieve an Earth model (2000/5000m) from the data collected on the surface (1000km^2)
  - Geophysics is an indirect science : no verification can be done, other to drill a hole.
- Massive disk and storage arrays for data management
- Graphic WS and graphic server for visualisation
- HPC clusters with high speed interconnects for processing and simulation

Acquiring Marine Seismic Data



Seismic is the key source of information used to understand subsurface geology

Today : Reverse Time Migration

Migration is any data-processing program that converts a data plane to an subsurface image for a given model.

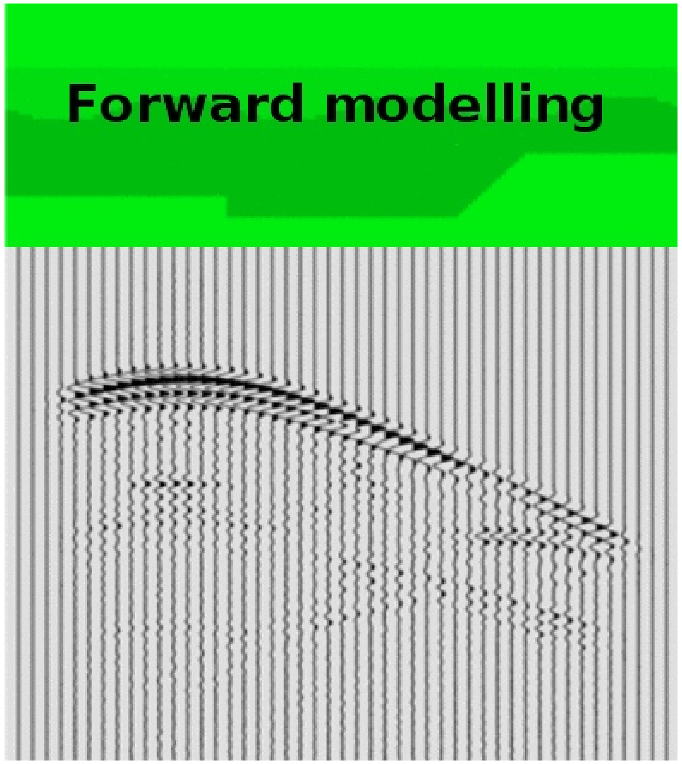
Typical Reverse Time Migration on a 3D domain solving the two way wave equation :

- Source wavefield is propagated forward in time
- Receiver wavefield is propagated backward in time

$$Image(\boldsymbol{x}) = \int F(\boldsymbol{x}, t) R(\boldsymbol{x}, t) dt$$

- Memory and/or I/O management
  - Storing all wavefields.
  - Checkpointing some wavefields and interpolate the missing from the existing when backpropagating (Symes)
  - Record at every time step wavefield at boundaries of model.
  - Use « random boundaries » techniques (Clapp)
- Little animation from Geophysic's Institut in Hamburg please go to *http://www.geophysics.zmaw.de/index.php?id=262*

Animation : Reverse Time Migration



Common propagators and functions per shot

- MPI implemented domain decomposition to fit local memory
- Finite Difference Modelisation
  - Stencils (2nd order in time, 8th order in space)
  - Increasing computational complexity with anistropy and TTI
  - Can deliver >50% of CPU peak (on Intel)
- Timestep loop with inner 3D loops
  - Possibility to parallelize with OpenMP
- 3D FFT : Often used from libraries : FFTW, MKL

- All pre-requisites for a succesful GPU port
  - NVIDIA (Micikevicius) provides an optimal stencil implementation
  - Timeloop is long enough to hide GPU/CPU transfers
  - OpenMP loops become GPU kernels
  - Cuda FFTs are available and perform well

Proofs of Concept : RTM migration

Show to customers that GPUs are a valid alternative.

- RTM migration already written and optimzed for GPUs with HMPP (2011) using the BullX 505 GPU blade server.

- Proofs Of Concept including porting of source code
  - Case 1 (2009) : RTM base on finite differences
  - Case 2 (2011) : RTM using a seismic spectral method

  - Several implementations tested :

NVIDIA GPU Computing CUDA

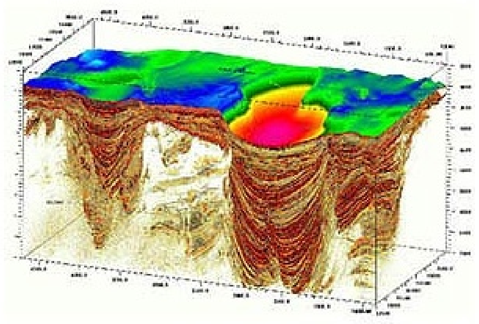
CUDA Fortran

hmpp hybrid multicore parallel programming

GPU Case Study – 2011

- 3D TTI Seismic Modeling and Reverse Time Migration
  - Efficient domain decomposition
  - Use of concurrency for maximum efficiency
    - Overlapping data transfers and CUDA kernel execution
    - Hand written HMPP kernels that efficiently exploit GPU memory

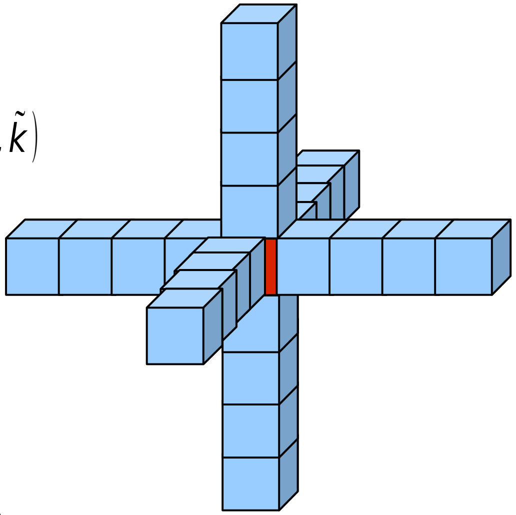
Seismic Modeling can be done very efficiently on GPUs



1st POC : RTM finite differences

- Original RTM implementation (Baysal,Kossloff,Sherwood) :

$$U_{i,j,k}^{t+1} = 2 \cdot U_{i,j,k}^{t} - U_{i,j,k}^{t-1} + v_{i,j,k} \cdot \sum_{(\tilde{i},\tilde{j},\tilde{k}) \in \varphi(i,j,k)} b_{\delta} . U_{(\tilde{i},\tilde{j},\tilde{k})}^{t}$$



- GPU parallelization of the spatial 3D loops (i,j,k)
- Wavepropagation in a grid of 500x600x600
- Absorbing boundaries (20 points)
- 8th order in space (4 neighbours in every direction)
- Forward wavefield is saved at every time step, but the disk I/O was taken out.

1st POC : Porting to C Cuda

- Fortran source code
- Porting needs include :
  - → 3 days program analysis and Fortran/C interfaces
  - → 1 day for Cuda calling sequences on the host
  - → 2 days for stencil Cuda kernels including the halo layers
  - → 1 day for Cuda kernels of the domain boundaries (PML)
  - → 1 day for seismic source injection
  - → 4 days for debugging and validation
- original source code : 732 lines – GPU code : 2188 lines

- Important Investment

1st POC Hardware and Software Configuration

- Server Novascale R422-E1 (twin) + NVIDIA Tesla S1070
  - → 2 Processors Intel Xeon (4 cores) @ 2.5 GHz (par CN)
  - → 16 GB memory
  - → 4 GPUs Tesla C1060

- Software
  - → OS : Bull Linux XBAS5 v1.2 (based on RHEL 5.1)
  - → NVIDIA kernel module 177.67, CUDA 2.3
  - → Intel Compiler (C et Fortran) version 10.1.017
  - → HMPP v 1.0

1st POC: Results C Cuda

- Execution times for 1000 time steps

Code Version	Time / time step (sec)	Speed-Up
Original	6,09	1,0
CUDA with copy of wavefield	0,72	8,5
CUDA without copy of wavefield	0,31	19,6
CUDA with asynchronous copy of wavefield	0,33	18,5

- Important speed-up for an important investment

1st POC : Conclusions

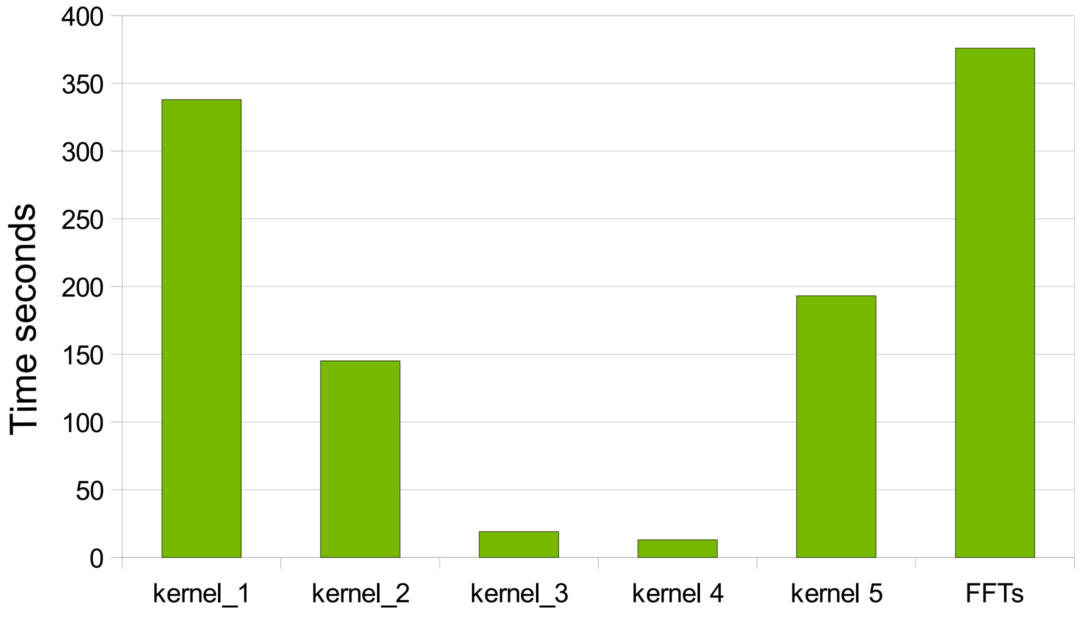
- Study showing that GPUs can be used in seismic RTM migration codes.
- C CUDA port shows :
  - → Important programming effort (C/Fortran)
  - → Code size is growing
  - → Limit port to hot spots
  - → Keep data on the GPU (perhaps recompute to avoid transfers)
  - → Overlap GPU/CPU communications with computation
  - → FD code memory bandwidth bound (try to get more compuation)
  - → Improvements possible with better data alignment and cache hierachies on the GPU.

2nd POC : spectral RTM

- Code based on a spectral method
- Original source code in Fortran 90 using OpenMP and MKL FFTs.
- One shot per node ..

2nd POC : Application profile

- 85 % of the execution time are spent in a single subroutine
- In this subroutine 7 kernels are identified and ported to GPU
  - → kernel_1
  - → kernel_2
  - → kernel_3
  - → kernel_4
  - → kernel_5
  - → FFTs



2nd POC goals

- Server bullx b505
  - → 2 sockets Intel Xeon (4 cores) @ 2.67 GHz
  - → 24 GB DDR3@1333MHz
  - → 2 GPUs Tesla M2050
- Software
  - → Bullx Supercomputer Suite (based on RHEL 6)
  - → NVIDIA kernel module 270.24, CUDA 4.0 (EA) for faster FFTs
  - → Intel Compiler (C et Fortran) version 12.0.2 and Intel MPI 4.0.1
  - → PGI Compilers 11.1 (CUDA 3.2)

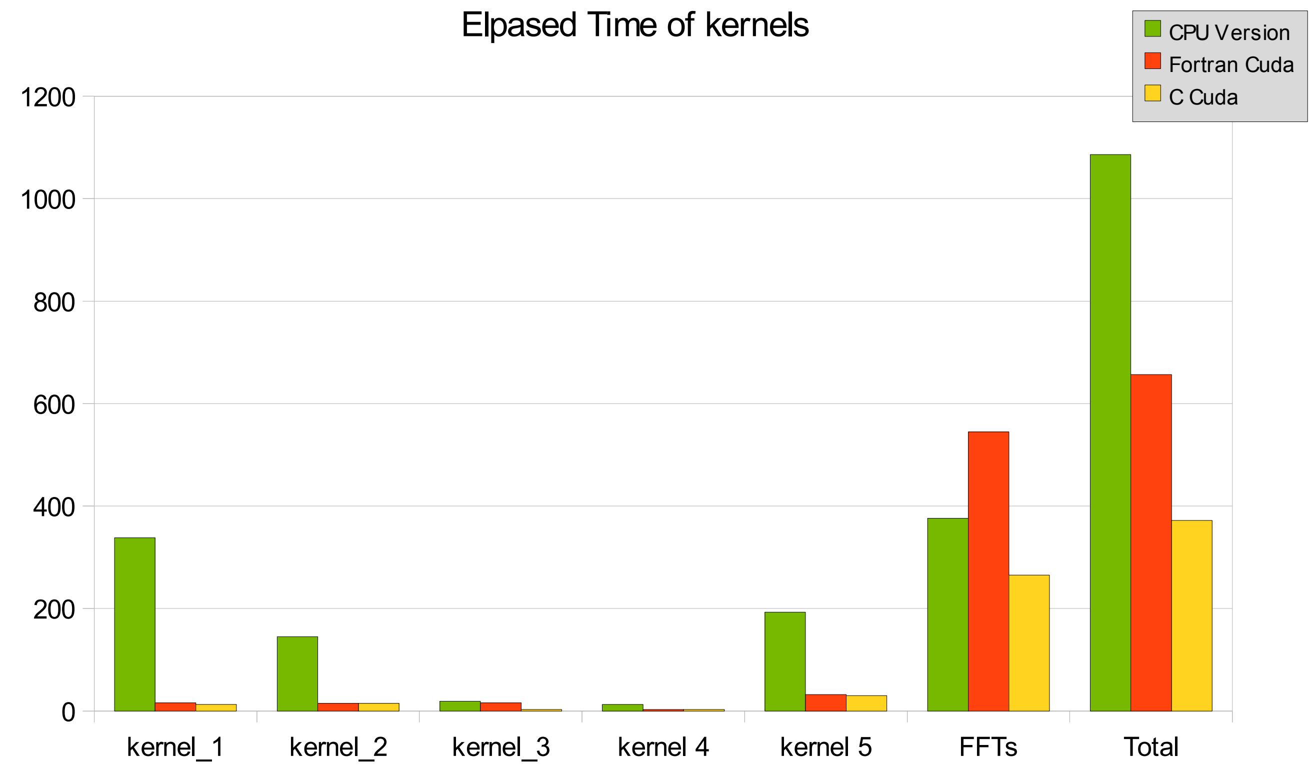
- Use Cuda FFTs and write call wrappers
- One Cuda kernel for each of the 7 subroutine kernels. Allowing to reuse the data on the GPU.
- Compare CUDA C (NVIDIA) , Fortran CUDA (PGI) and HMPP (CAPS)
  - → CUDA C – optimised CUFFT (pre-release CUDA 4.0)
  - → Fortran CUDA 11.1 based on CUDA 3.2

Using Fortran Cuda from PGI

- Simplified Fortran porting
- No difficult Fortran → C → CUDA interfaces
- No problem with conversion of unit memory stride acces in multidimensional arrays
- API simplified and identical with Fortran90

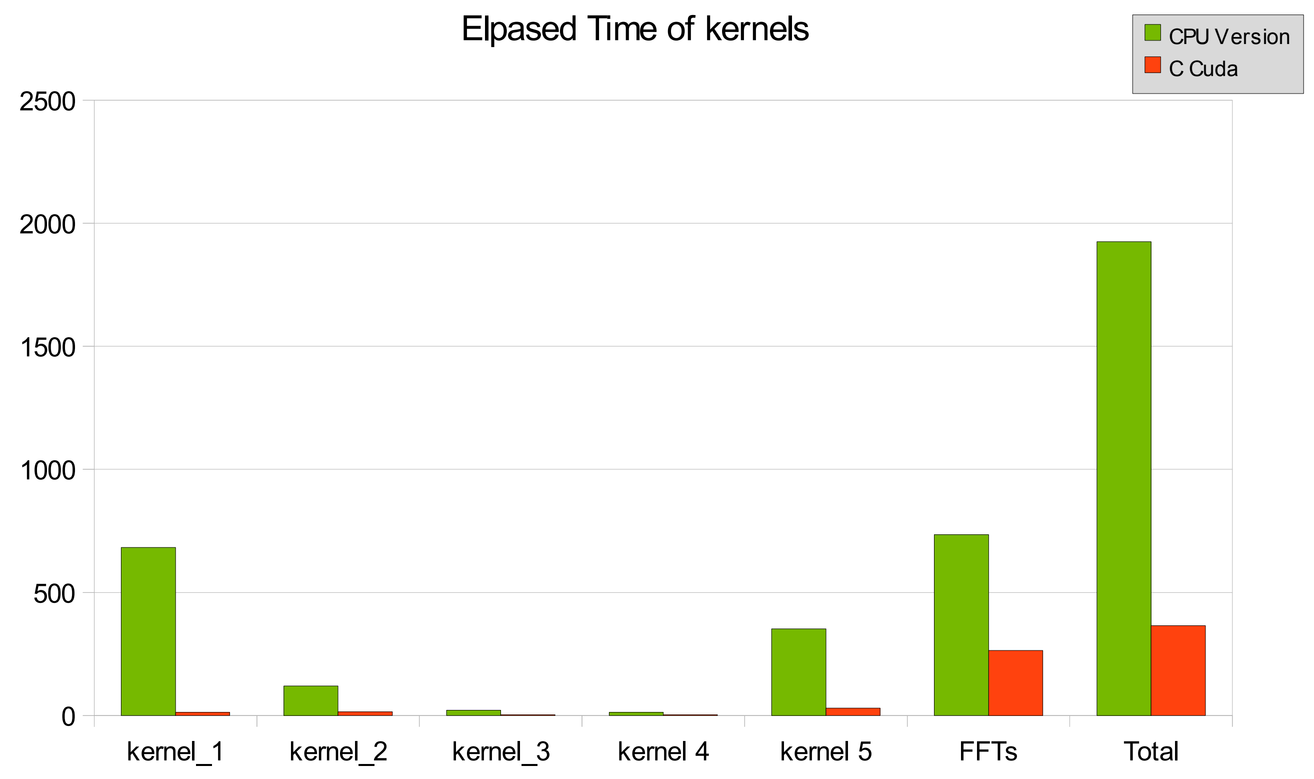
```
!Define variables on CPUs
real, pinned, allocatable, dimension(:,:,:) :: A_host
!Define variables on GPUs
real, device, allocatable, dimension(:,:,:) :: A_device

!allocate them in a single call
allocate( A_host(nx,ny,nz), A_device(nx,ny,nz) )

!transfer data between CPU/GPU
A_device = A_host
```

2nd POC : Timings one shot



- 1 shot per node
- CPU execution runs on 8 OpenMP threads, including FFTs
- CUDA C and Fortran are identical, except FFTs
- significant improvement for CUFFT V3.2 (used by Cuda Fortran) to V4.0 (used byCuda C)

2nd POC : Timings 2 shots



- 2 shots per node as there are 2 GPUs on the server
- CPU code runs with 4 OpenMP/MKL threads per shot instead of 8
- CPU code is roughly 2x slower
- GPU version runs perfectly well the 2 shots, each on one GPU

2nd POC : Conclusions

- Spectral RTM gives moderate speed-ups on GPUs (3x)
- Original code is highly optimised for OpenMP / MKL
- May run several shots per GPU
- GPU performance is limited by the FFTs
- Fortran CUDA facilitates porting with identical performance
- Further problems :
  - → Small data set for benchmarking.
  - → Production runs may use the whole CPU memory for a single shot
  - → Problem size may be too big to fit in GPU memory
- HMPP port is under way

Conclusions

- RTM codes have the structure to be ported to GPUs
  - → Work is concentrated in some kernels
  - → Kernels can be written for GPUs
  - → Transfers can be hidden during timeloop
  - → FFTs are working, and run faster
- CAPS HMPP and PGI Cuda Fortran facilitate the Fortran port of the kernels
- Shots may not fit in memory of a single GPU
- Implementation of the wavefield storage is important
  - → Random boundaries avoid I/O
  - → Checkpoint or boundaries, need an interpolation on the GPU
  - → Extreme case : I/O to disk
- Bull GPU blades (dual sockets, dual GPU) help to optimize performance, and delivering full bandwidth for CPU/GPU communication.

Architect of an Open World™

LIBERATE IT