Mathematics & Supercomputing & Multidisciplinarity

Olivier Pironneau[1]

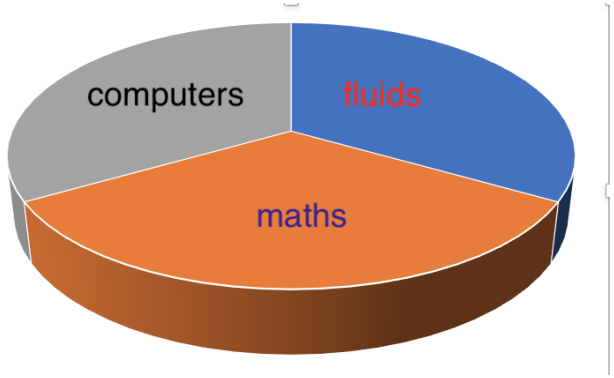
[1]Sorbonne University - LJLL & Académie des Sciences
http://ann.jussieu.fr/pironneau

Lecture dedicated to Dimitri Komatitsch (1970-2019)

Roscoff March 2019

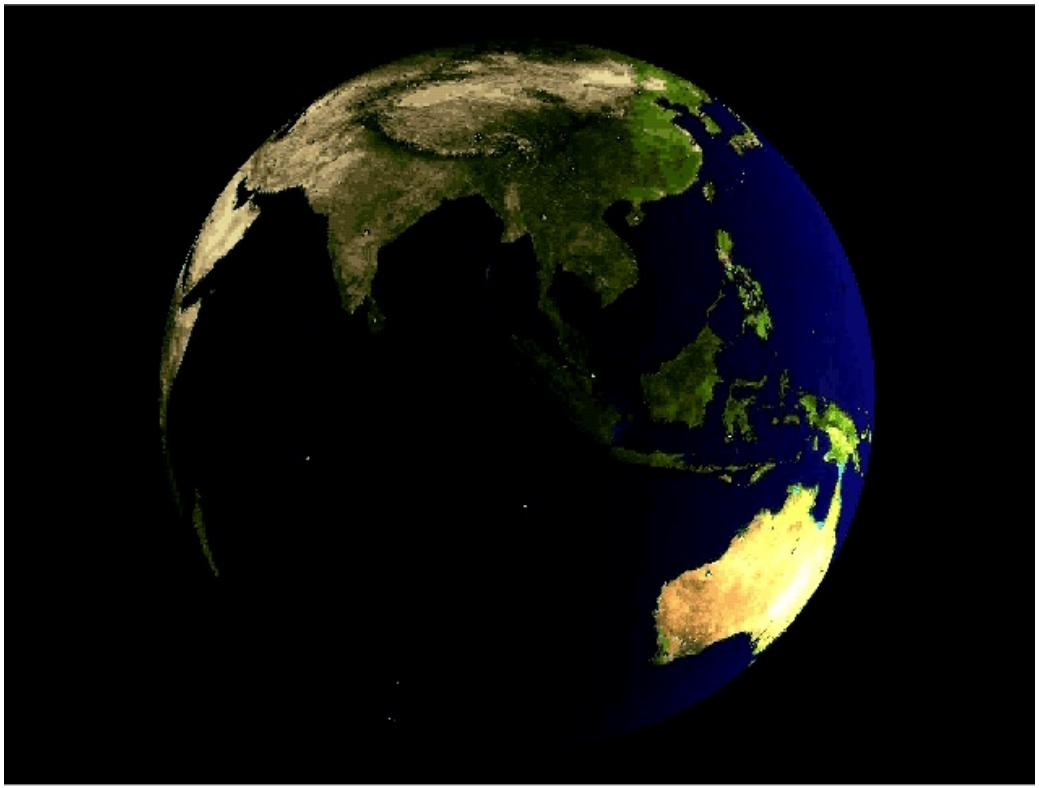
Multidisciplinarity and Science Policy

- Create a new lab: expensive and political
- Create a new activity within a lab: mostly at an interface of science
- JL Lions steering Laboria (INRIA applied math lab)
- The benefits of the triple point



Jack of all trades master of none

The 2008 Tsunami Simulated with SPECFEM3D



Wave propagation in the planet earth after the Sumatra earthquake of 2008 (D. Komatitsch et al)

The Simulation Code SPECFEM3D

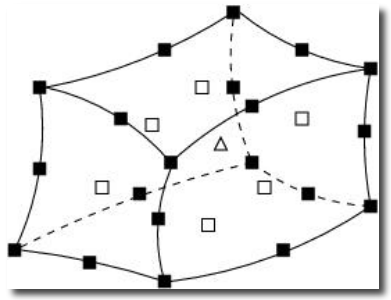
Dimitri Komatitsch and team designed, for earthquake, a Fluid-solid software that runs on a cluster of CPU/GPU .

- Linear Elasticity in S (wave eq.)+ incompressible flows ($p = \partial_t \phi$) in $F = \Omega \backslash S$:

$$\rho \partial_t v = -\nabla p,\ \partial_t p = -\kappa \nabla \cdot v \Leftrightarrow \int_\Omega \frac{1}{\kappa} \partial_{tt} \phi w + \int_\Omega \frac{1}{\rho} \nabla \phi \nabla w - \int_{F \backslash S} w n \cdot v = 0$$

$$\int_\Omega \rho\, w \cdot \partial_{tt} u + \int_\Omega \nabla w : C : \nabla u = \int_\Omega w \cdot f + \int_\Gamma \sigma \cdot n \cdot w$$

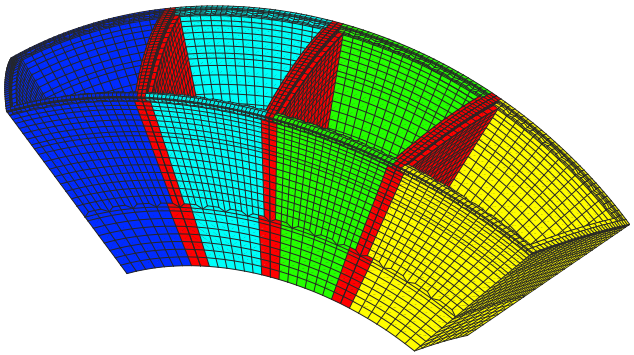
$$\sigma = C : \varepsilon, \qquad \varepsilon = \frac{1}{2}[\nabla u + \nabla u^T]$$



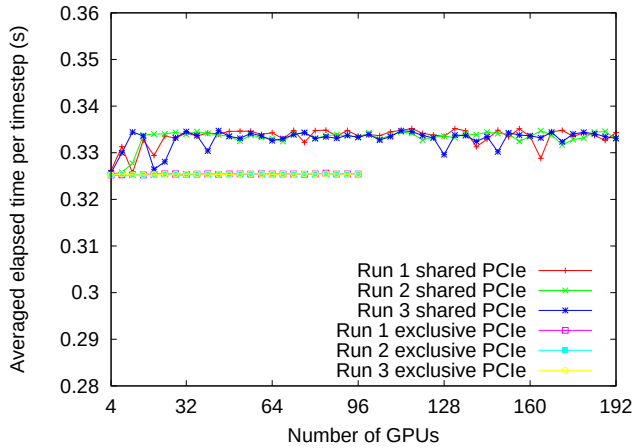
- It is in C++ and uses OpenMP, MPI and CUDA.
- It has a block allocation strategy for load balancing
- Each block is a spectral element fluid-Structure solver for the wave equation with an explicit in time discretization.
- When blocks are non fitting they are glued by the Mortar method
- Effort is made to compute locally in GPUs *during* the data transfers between blocks using non blocking MPI comm.

Performance using GPUs (2008)

The key is to allocate for each macro-element the inner nodes calculations to the GPU and the boundary nodes calculation and communications to CPUs.
Each spectral element on the GPU has 128 threads, one thread per Gauss-Lebato point of degree 4 (125 DOF)



Subdomain allocation

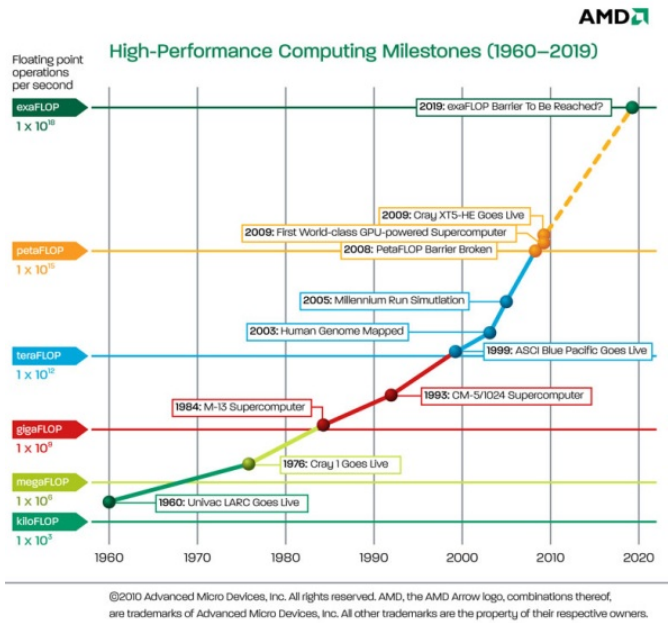


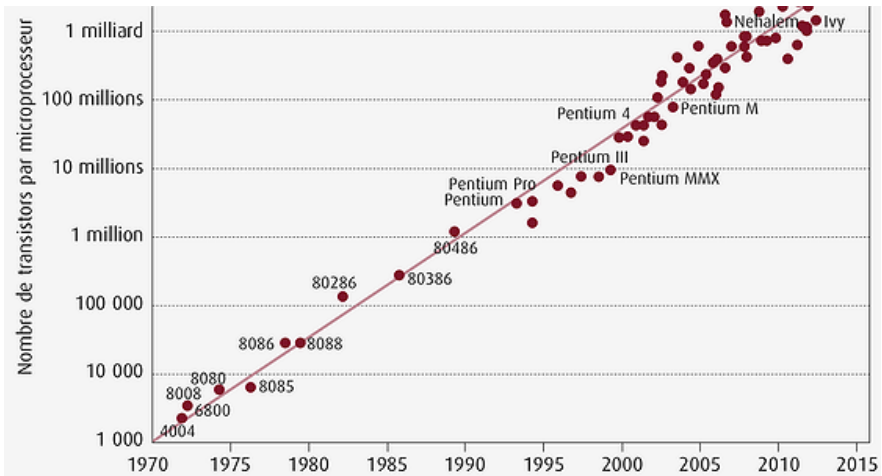
Performance

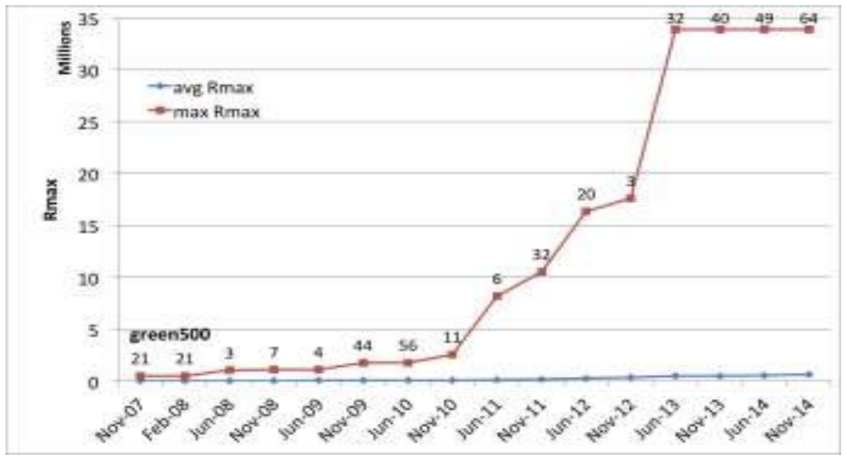
Curie Machine (7 Pflops)

High Performance Computing & Science

1. Nuclear engineering 1943 Los Alamos
2. Aerospace and Automobile : 1970
3. Geology for oil 1980
4. Ab-initio chemistry 1990 $\Leftarrow$ // Computing
5. Climate and meteorology 2000
6. Astrophysics 2010
7. Deep Learning and AI 2020
8. QCD: quantum chromodynamics 2030







Moore's

ref: AMD, Intel, DOI: 10.2298/CSIS150228063F,

Algorithmic Advances

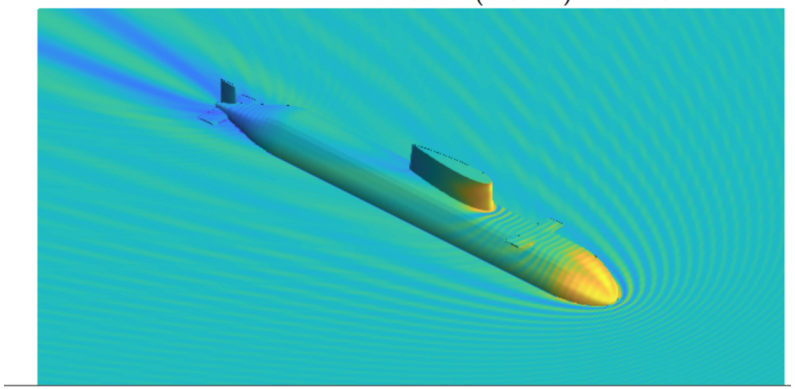
Solution of sparse linear systems

1. Gauss-Seidel / Jacobi iterative method
2. Conjugate gradient method
3. Preconditioned Conjugate gradients
4. Preconditioned GMRES/Quasi Newton
5. Algebraic Multigrids
6. Fast multipole methods & H-Matrix
7. Domain Decomposition

Neumann scattering :
$[H]\mu = -\partial_n P_{inc}$

100.000 ddl at 200 Hz

H-Matrix (8 cores) : 1000 s, 5.70 Go
Regularization : 101 s
LU factorisation : 604 s
Resolution : 2.22 s
Radiation (8 cores) : 1500 s

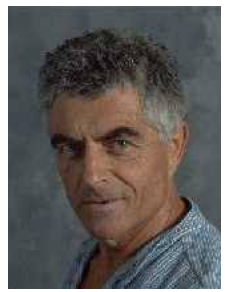


Courtesy of Mathieu Aussal's gypsilab

Sound wave from a submarine computed by BEM with 10^5 *nodes and* $\mathcal{H}^1$*-method on a PC in 1000sec. Traditional methods would require solving a spase linear system of size* 10^7 *at least.*







C. Gauss M. Hestenes K.vanDerVorst Y. Saad A. Brandt W.Hackbusch P.Schwarz

The Fast Mutlipole Method

- Vladimir Rokhlin imported the idea from astrophysics, Leslie Greengard did the proofs, Weng-Cho Chew, Eric Darve, Guillaume Sylvan, etc perfected the method.
- FMM is an integral equation solver $O(n \log n)$.
- For a given large integer J and two given sets of vectors $\{u_j\}_1^J$, $\{x_j\}_1^J$, let us compute $v_i = \sum_{j=1}^{J} \frac{u_j}{x_i - x_j} \quad i = 1..J$.
- If no trick is applied it takes J^2 operations. However one may do the following:

$$\frac{1}{x-y} = \frac{1}{x-z+z-y} = \frac{1}{(x-z)(1+\frac{z-y}{x-z})} = \sum_{m=0}^{M} \frac{(y-z)^m}{(x-z)^{m+1}} + o((\frac{z-y}{x-z})^M)$$

Hence $$\sum_{j=1}^{J} \frac{u_j}{x_i - x_j} = \sum_{m=0}^{M} [\sum_{j=1}^{J} (x_j - z)^m u_j] \frac{1}{(x_i - z)^{m+1}}$$

Now the operation count is $2M \times J + M \times M$.

Top 10 for Algorithms of the 20^{th} Century

1946 Metropolis Algorithm for Monte Carlo.

1947 Simplex Method for Linear Programming.

1950 Krylov Subspace Iteration Methods.

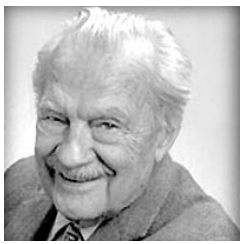
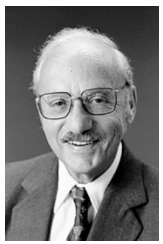
1951 The Decompositional Approach to Matrix Computations.

1957 The Fortran Optimizing Compiler.

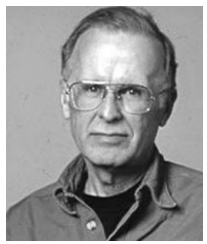
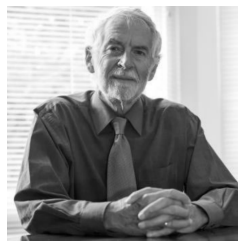
1959 QR Algorithm for Computing Eigenvalues.

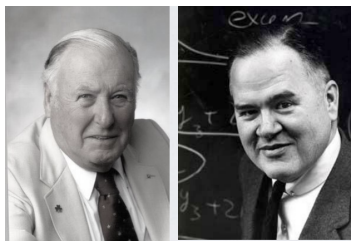
1962 Quicksort Algorithm for Sorting.

1965 Fast Fourier Transform.


Metropolis

Dantzig

Krylov

Householder

Backus

Francis

Hoare

Cooley-Tukey

Who Devised the Algorithms

Who was a professor?

Adams	**Argyris**	Backus	**Bashforth**
Bezier	**Brandt**	**Broyden**	**Clough**
Cooley	**Courant**	**Curtiss**	**Dahlquist**
Dantzig	**Daubechies**	**Davidon**	**de Boor**
de Casteljau	**Euler**	**Fedorenko**	**Fletcher**
Francis	**Friedrichs**	Garwin	**Gauss**
Gear	**Givens**	**Golub**	**Gottlieb**
Greengard	**Griewank**	**Hackbusch**	**Hestenes**
Heun	**Hirschfelder**	**Householder**	Liu
Jacobi	**Kahan**	**Kantorovich**	Karmarkar
Kublanovskaya	**Kutta**	**Lagrange**	**Lanczos**
Lax	**Legendre**	**Lewy**	**Metropolis**
Moler	Morlet	**von Neumann**	**Newton**
Orszag	**Powell**	**Raphson**	**Richardson**
Rokhlin	**Runge**	**Rutishauser**	**Saad**
Schoenberg	**Sorensen**	**Southwell**	**Stiefel**
Tukey	**Ulam**	**van der Vorst**	Vinsome
Wilkinson	Zuse		

Who was a mathematician?

Adams	Argyris	Backus	**Bashforth**
Bezier	**Brandt**	**Broyden**	Clough
Cooley	**Courant**	Curtiss	**Dahlquist**
Dantzig	**Daubechies (½)**	**Davidon**	**de Boor**
de Casteljau	**Euler**	**Fedorenko**	**Fletcher**
Francis	**Friedrichs**	Garwin	**Gauss (½)**
Gear	**Givens**	**Golub**	**Gottlieb**
Greengard	**Griewank**	**Hackbusch**	**Hestenes**
Heun	Hirschfelder	**Householder**	**Liu**
Jacobi	**Kahan**	**Kantorovich**	**Karmarkar**
Kublanovskaya	**Kutta**	**Lagrange**	**Lanczos**
Lax	**Legendre**	**Lewy**	Metropolis
Moler	Morlet	**von Neumann**	**Newton (½)**
Orszag	**Powell**	**Raphson**	**Richardson**
Rokhlin	**Runge**	**Rutishauser**	**Saad**
Schoenberg	**Sorensen**	Southwell	**Stiefel**
Tukey	**Ulam**	**van der Vorst**	Vinsome
Wilkinson	Zuse		

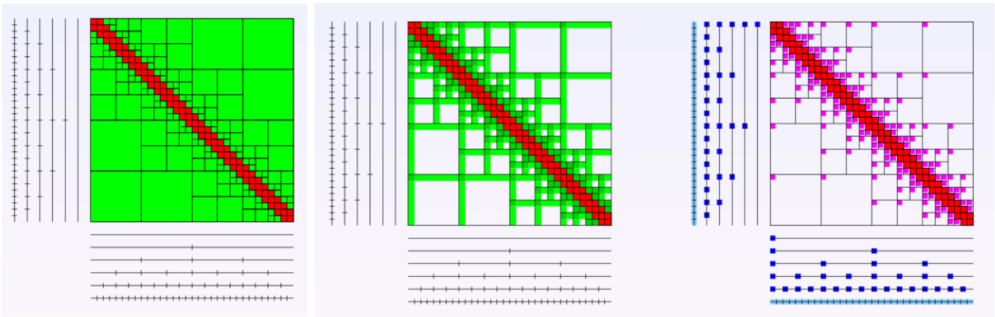
From Nick Trefethen's talk "Who invented the great numerical algorithms?"

Best Algorithms of late 20^{th} - early 21^{st} Century?

1. Page Rank algorithms
2. Stochastic/genetic/evolutionary algorithms for optimisation
3. Deep Neural Network
4. Quantum Computing
5. Fast Multipole Method
6. $\mathcal{H}$ - methods

$\mathcal{H}$ stands for Hierarchic. It uses the same idea as FMM: if the kernel f the differential operator can be approximated (e.g; a polynomial expansion) $K(x,y) \approx \sum_k \tilde{K}(x,\xi_k) b_k(y)$, then

$$\int_{S\times S} K(x,y)u_i(y)u_j(x)dxdy \approx \sum_k a_{ik}b_{jk};\ a_{ik} = \int_S \tilde{K}(x,\xi_k)u_i(x)dx,\ \ b_{jk} = \int_S b_k(y)u_j(y)dy$$



(from Mathieu Aussal) Full Matrix $\mathcal{H}$ - matrix $\mathcal{H}^2$ -matrix

Remarks on the Future of HPC

- Page ranking is perhaps the most use algorithm: use racks of PCs, cloud... Memories will have computing capabilities
- Banks are among the biggest users of supercomputing: Risk assessment. They don't use supercomputers
- Mining bitcoins is incredibly expensive, so far without HPC
- Deep learning uses GPUs. Supercomputing or Cloud GPU-computing?
- Exascale computers will have millions of cores. Requires new algorithms
- Green computing (10^4 ARM processors, slow but cool)? or Wind-mills attached to supercomputers?

Neural Network: existence of Solution

A Neural Network (Bruno Després) Let $f \in C^1(R) \cap W_1^\infty(R)$

$$
\begin{aligned}
f(x) &= \int_{-\infty}^{x} f'(y)dy = \int_R H(x-y)f'(y)dy \\
&\approx \sum_{j=-J}^{J} \phi(\frac{x}{\epsilon} - \frac{j\delta x}{\epsilon}) f'(j\delta x)\delta x = \sum_{-J}^{J} \omega_j \phi(a_j x + b_j)
\end{aligned}
$$

where $H(x)$ is Heaviside and ϕ a sigmoid to approximate H.

Theorem 3 (Hornik et al., Cybenko, 1989) *Feedforward network with a linear output layer and at least one hidden layer with any "squashing" activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.*

Mathematical Results Pertinent to Deep Learning

Deep learning requires computing resources; Cloud is prefered to HPC but it may change. Ten years of algorithmic improvements by computer scientists. But no convergence proof.

Consider a system with stochastic input α and output $\alpha \mapsto u(\alpha)$. Let us focus on

$$u_\rho(\alpha) := \mathbf{E}(u|\alpha), \quad \sigma_\rho^2 := \mathbf{E}\big[\|u(\alpha) - u_\rho(\alpha)\|^2 \big| \alpha\big]$$

The Deep Neural Network defines an approximation of the output $\alpha \mapsto u_n(\alpha)$.

If m the number of samples to train the network, and $\{\alpha_i\}_1^m$ are randomly chosen according to a probalility law on the set of all sample A, the best we can do is to

$$\text{choose } u_n \text{ the minimizer of } \mathcal{E}_m(u_n) := \frac{1}{m}\sum_{i=1}^m \|u_n(\alpha_i) - u(\alpha_i)\|^2$$

Theorem 1 (see Goodfellow)

$$\mathcal{E}(u_n) := \mathbf{E}_A[\|u_n(\alpha) - u(\alpha)\|^2] = \mathbf{E}_A[\|u_n(\alpha) - u_\rho(\alpha)\|^2] + \sigma_\rho^2$$

This means that the best one can expect to achieve is a neural network which generates $\alpha \mapsto u_\rho(\alpha)$.

Mathematical Results Pertinent to Deep Learning (II)

Precision improves with the number of samples according to
Theorem 2 (Cucker-Smale)

$$\mathcal{P}\left[|\mathcal{E}(u_n) - \mathcal{E}_m(u_n)| < \epsilon\right] \geq 1 - 2e^{-\frac{m\epsilon^2}{2\sigma^2 + \frac{1}{3}M^2\epsilon}}$$

where M is such that $|u_n(\alpha) - u(\alpha)| < M$ a.e. and σ is the variance of u_n. There remains the problem of generating u_ρ with a neural network.

Convergence of the stochastic gradient algorithm (like Adam's) can be a problem. There are versions of the Adaptive Adam Stochastic gradient which achieve precision $O(\frac{1}{\sqrt{N}})$ after N iterations and $O(\frac{1}{N})$ after N iterations in the batch setting (Rachel Ward, Xiaoxia Wu , and Léon Bottou, arxiv 1806.01811) .

No theorem to support that Deep NN are better!

More Topics

- Convergence of genetic algorithms
- Smooth Particle Hydrodynamics (SPH)