Modern algorithms in mathematical research, parallelism and languages
The intersection of theoretical and practical issues

Fernando Pérez

Applied Mathematics
University of Colorado, Boulder
&
Helen Wills Neuroscience Institute
University of California, Berkeley

NSF - Washington
October 31, 2007
Outline

1 Motivation
Outline

1. Motivation
2. A case study
Realistic modeling of natural systems

- (Most) small problems are done.
  - For those, the algorithms don’t matter...
  - because they can be brute forced.
- Larger $N$ (whatever $N$ is). Do we have algorithms with the correct scaling?
- Multiscale & heterogeneous problems
  - Chemistry: QM of many e- atoms & complex molecules.
  - Materials: from Quantum Mechanics to macrostructures.
  - Biology: from the physical chemistry of ion channels to the whole heart.
  - Drug design from first principles.
  - Multifactor climate models.
Experimental datasets are getting very large

- fMRI (functional MRI, neuroimaging) → gigabytes per scan
- gene sequencing data
- astronomical multispectral data
- Large Hadron Collider (LHC at CERN, Geneva) → simply *monstrous*.
Common features of these problems

- Many degrees of freedom.
- Multiple relevant time/space scales, often each of them modeled in different manners.
- This combination will require a lot of mathematical, algorithmic and computational innovation.
Why do we need fast algorithms?

As computers get bigger/faster, algorithms matter more!
Why do we need fast algorithms?

As computers get bigger/faster, algorithms matter more!
Why do we need fast algorithms?

As computers get **bigger/faster**, algorithms matter **more**!
Why do we need fast algorithms?

As computers get **bigger/faster**, algorithms matter **more**!
Why do we need fast algorithms?

As computers get bigger/faster, algorithms matter more!

![Graph showing time vs. problem size for different algorithms and time constraints.](image-url)
Why do we need fast algorithms?

As computers get \textbf{bigger/faster}, algorithms matter \textbf{more}!
Motivation

A case study

Languages

Algorithms

Outlook & questions

Why do we need fast algorithms?

As computers get **bigger/faster**, algorithms matter **more**!

How do you get a fast algorithm?

**Compress**: factorize, find an alternate representation, approximate, *decouple*.
The realities of physics for computing
The realities of physics for computing

Not pretty:

Sources: Intel, Microsoft (Sutter), Stanford (Olukotun, Hammond) & Berkeley (Yelick)
We can’t escape thermodynamics

Moore’s Law Extrapolation:
Power Density for Leading Edge Microprocessors

Power Density Becomes Too High to Cool Chips Inexpensively

Sources: Shekhar Borkar, Intel Corp & Kathy Yelick, UC Berkeley
We can’t escape thermodynamics

Nor relativity! The speed of light: \( c = 3 \times 10^8 \text{m/s} \).

At 3 GHz, a signal can travel only \( 10 \text{ cm} \) between two clock ticks!
What we’re being sold

The landscape is becoming very heterogeneous, but the recurring theme is \textit{multi-something}, not faster anything!

- Multicore chips: even in your laptop.
- Graphics cards for general computing (GPGPU): $> 128$ ‘processors’ per $300$ card.
- High-density clusters: SiCortex ($> 5000$ processors in a cabinet).
- IBM/Toshiba/Sony Cell architecture (PlayStation)
- Hybrid CPU/GPU systems with multiple heterogeneous cores.
- Pervasive multithreading (Sun Niagara)
What we’re being sold

The landscape is becoming very heterogeneous, but the recurring theme is multi-something, not faster anything!

- Multicore chips: even in your laptop.
- Graphics cards for general computing (GPGPU): > 128 ‘processors’ per $300 card.
- High-density clusters: SiCortex (> 5000 processors in a cabinet).
- IBM/Toshiba/Sony Cell architecture (PlayStation)
- Hybrid CPU/GPU systems with multiple heterogeneous cores.
- Pervasive multithreading (Sun Niagara)

What is a petaflop computer?

\[(\text{gigahertz CPU}) \times (\text{few flops per cycle}) \times (\text{thousands of CPUs})\]

Not what (at least I) wanted to hear!
Outline

1. Motivation
2. A case study
3. Tools & Algorithms, part I
4. Tools & Algorithms, part II
5. Outlook & questions
Physical problems formulated as PDEs

- The Laplace/Poisson equations

\[-\Delta u = f\]

- The Schrödinger equation (for stationary states)

\[\left(-\frac{1}{2} \Delta + V\right) \psi = E \psi\]

- The modified Stokes equation (time-stepping schemes for Navier-Stokes):

\[\alpha v - \mu \Delta v + \nabla p = f\]
\[\nabla \cdot v = 0\]

A good fraction of the world’s (scientific) computing time is devoted to the solution of this type of problem.
Numerical approaches

Broadly and loosely speaking, there are two main options; each has its own set of difficulties.

**Discretize differential operator and invert**

- Sparse matrices (fast solvers)
- That represent unbounded operators...
Numerical approaches

Broadly and loosely speaking, there are two main options; each has its own set of difficulties.

- Discretize differential operator and invert
  - Sparse matrices (fast solvers)
  - That represent unbounded operators...
  - And hence are **ill-conditioned** and require pre-conditioning.
Numerical approaches

Broadly and loosely speaking, there are two main options; each has its own set of difficulties.

Discretize differential operator and invert

- Sparse matrices (fast solvers)
- That represent unbounded operators...
- And hence are **ill-conditioned** and require pre-conditioning.
Numerical approaches

Broadly and loosely speaking, there are two main options; each has its own set of difficulties.

Discretize differential operator and invert

- Sparse matrices (fast solvers)
- That represent unbounded operators...
- And hence are **ill-conditioned** and require pre-conditioning.

Write integral formulation and apply integral operator (Green’s functions)

- Well-conditioned objects...
- That lead to **dense matrices**
Numerical approaches

Broadly and loosely speaking, there are two main options; each has its own set of difficulties.

### Discretize differential operator and invert

- Sparse matrices (fast solvers)
- That represent unbounded operators...
- And hence are **ill-conditioned** and require pre-conditioning.

### Write integral formulation and apply integral operator (Green’s functions)

- Well-conditioned objects...
- That lead to **dense matrices**
- And don’t easily generalize to multiple dimensions.
**Multiresolution algorithms in multiple dimensions**

- **Project boundaries**
  1. Numerical algorithms with *finite but controlled precision*.
  2. **Multiscale**, fully *adaptive* algorithms.
  3. Approximations are a cousin of the Fast Multipole Method (*FMM*), but easier to generalize in dimension and kernel.
Multiresolution algorithms in multiple dimensions

**Project boundaries**

1. Numerical algorithms with *finite but controlled precision*.
2. **Multiscale**, fully *adaptive* algorithms.
3. Approximations are a cousin of the Fast Multipole Method (**FMM**), but easier to generalize in dimension and kernel.

**Key mathematical ideas**

1. Multiresolution analysis (**wavelets**): sparse matrix representations for a large class of kernels.
2. **Separated representations**: reduction of dimensionality cost.
Multiresolution algorithms in multiple dimensions

- **Project boundaries**
  1. Numerical algorithms with **finite but controlled precision**.
  2. **Multiscale**, fully **adaptive** algorithms.
  3. Approximations are a cousin of the Fast Multipole Method (**FMM**), but easier to generalize in dimension and kernel.

- **Key mathematical ideas**
  1. **Multiresolution analysis (wavelets)**: sparse matrix representations for a large class of kernels.
  2. **Separated representations**: reduction of dimensionality cost.

- **Group effort over many years (1988-today)**
  1. Gregory Beylkin, Lucas Monzón, Christopher Kurcz - **CU Boulder**
  2. Martin Mohlenkamp - **Ohio University**
  3. Robert Harrison, George Fann, Takeshi Yanai, Zhengting Gan - **ORNL**
  4. Vani Cheruvu - (now at **NCAR**)
  5. Robert Cramer - (now at **Raytheon**)

F. Pérez (CU.B/UC.B) Algorithms, parallelism & languages NSF-Washington 14 / 40
Represent functions adaptively

Standard adaptive $2^d$-trees in $\mathbb{R}^d$
Functions: adaptive, controlled accuracy decompositions

- $N_{\text{nod}} = 12, \epsilon = 1.0 \times 10^{-10}, N_{\text{blocks}} = 21$
- $N_{\text{nod}} = 10, \epsilon = 5.0 \times 10^{-11}, N_{\text{blocks}} = 634$
Functions: adaptive, controlled accuracy decompositions

$N_{\text{nod}} = 12, \epsilon = 1.0 \times 10^{-10}, N_{\text{blocks}} = 21$

$N_{\text{nod}} = 10, \epsilon = 5.0 \times 10^{-11}, N_{\text{blocks}} = 634$
Sparse operators

Instead of applying this:
Sparse operators

Instead of applying this:

We apply this (blue is zero, not done):
Adaptive natural-scale application
A graphical representation

Redundant tree of input (output skeleton)

Terminal
Non-terminal
Operators \((d > 1):\) Gaussians to the rescue

Use Gaussians for approximations

\[
\frac{1}{\| r - r' \|} \approx \sum_{m=1}^{M} w_m e^{-\tau_m \| r - r' \|^2},
\]

with controlled accuracy \(\varepsilon:\)

The problem factorizes partially: a practical solution for \(d > 1!\)

\[
\phi_{i j k} = \sum_{i' j' k' = 1}^{N} G_{i' j' k'} \rho_{i' j' k'} = \sum_{m=1}^{M} w_m \sum_{i'} \sum_{j'} \sum_{k'} F_{i j' k'}^m \rho_{i' j' k'}
\]
Quantum mechanics (H atom)

Schrödinger’s equation in an integral (Lippman-Schwinger) formulation:

\[-\frac{1}{2}\Delta \psi - \frac{1}{r}\psi = E\psi,\]

We write

\[\phi = -2G_\mu V\phi,\]

where \(G_\mu = (-\Delta + \mu^2 I)^{-1}\) is the Green’s function for some \(\mu\) and \(V = -1/r\) is the nuclear potential. This can be solved iteratively.
Quantum mechanics (H atom)

Schrödinger’s equation in an integral (Lippman-Schwinger) formulation:

\[
- \frac{1}{2} \Delta \psi - \frac{1}{r} \psi = E \psi,
\]

We write

\[
\phi = -2 G_\mu V \phi,
\]

where \( G_\mu = (-\Delta + \mu^2 I)^{-1} \) is the Green’s function for some \( \mu \) and \( V = -1/r \) is the nuclear potential. This can be solved iteratively.

1. Initialize with some value \( \mu_0 \) and function \( \phi \).
2. Apply the Green’s function \( G_\mu \) to the product \( V \phi \) to compute:

\[
\phi_{\text{new}} = -2 G_\mu V \phi.
\]

3. Compute the energy for \( \phi_{\text{new}} \),

\[
E_{\text{new}} = \frac{1}{2} \langle \nabla \phi_{\text{new}}, \nabla \phi_{\text{new}} \rangle + \langle V \phi_{\text{new}}, \phi_{\text{new}} \rangle \Big/ \langle \phi_{\text{new}}, \phi_{\text{new}} \rangle.
\]

4. Set \( \mu = \sqrt{-2E_{\text{new}}} \), \( \phi = \phi_{\text{new}} / \| \phi_{\text{new}} \| \) and return to Step 2.
Error for the Hydrogen atom ground state eigenvalue

![Graph showing the error in energy for iterations against atomic units, with a horizontal line at $\epsilon = 10^{-6}$]
Algorithmic and language considerations

- Algorithm development and implementation were done together.
  - *Dictionary:* builtin flexible, sparse, multidimensional data structure.
  - *Dynamic code generation:* string input → C source (for multiple dimensions) → run-time dynamically compiled and loaded library.
  - *Reuse* of open (LAPACK/SciPy) and in-house Fortran codes.
  - Easy to *hand-tune* (in C, C++ or Fortran) critical sections.
  - *Interactive* 2d/3d visualization, lots of object introspection and diagnostics.
  - Good prototype for modern, high-level parallelization for massive problems.
    - Robert J Harrison & George Fann (ORNL): MADNESS → Cilk-inspired parallelization of these ideas targeted at $O(10^5)$ CPUs, in C++.
Languages - what do we want?

Expressiveness
- Beyond arrays. We’re past LAPACK.
- Rich data types - hash tables, strings, sets.
- Seamless access to arbitrary/extended precision (Mathematica).

User productivity
- Interactivity.
- Visualization.
- Rich libraries (also for the non-mathematical core)

Performance
- Seamless access to C/C++/Fortran.
- Easy to optimize the high-level, flexible forms.

Parallelism
- High-level constructs to express it...
- Safely...
- Without sacrificing performance.
- HARD.
Treasure trove of info: K. Yelick, J. Demmel at UC Berkeley!

- MPI (‘the standard’ for much of today’s real parallel codes).
- OpenMP.
- Intel Threading Building Blocks (open sourced): library.
- Cilk (MIT) / Cilk++ (startup)
- Fortress (SUN - DARPA)
- X10 (IBM - DARPA)
- Titanium: Java (UC Berkeley)
- Unified Parallel C, UPC (UC Berkeley)
- Matlab/Mathematica’s Parallel Toolboxes
- InteractiveSuperComputing *p (MIT-private)
- IPython (B. Granger, B. Ragan-Kelley, FP)
Python in scientific computing

- Free, open source, interactive, highly portable language.
- Extremely readable syntax ("executable pseudo-code").
- Rich built-in types: lists, sets, dictionaries (hash tables), strings, ...
  - Very expressive for complex algorithm development.
- Comprehensive standard library (batteries included):
  - Text processing, networking protocols, threading, GUIs, ...
- Visualization: 2D and 3D
- Standard libraries for Matlab/IDL-like arrays.
- Easy to wrap existing C, C++ and FORTRAN codes.
  - Easy to couple with either hand-optimized C,
  - The C can also be auto-generated from a restricted subset of the language.
Who is doing science with Python?

- **NASA**: Hubble, JPL
- **CASA**: Common Astronomy Software Applications (National Radio Astronomy Observatory).
- **MADNESS** (DOE project from ORNL)
- **DANSE** (NSF/DOE at the Spallation Neutron Source-ORNL)
- **Sandia NL**: Trilinos solvers.
- **Geodynamics.org** (NSF funded): open tools for geophysics.
- **NiPy** (UC Berkeley, NIH funded): neuroimaging.
- **SAGE**: open source mathematics, very ambitious project (U. Washington).
- Many others...
Python: Matlab-like usage

```python
import math, numpy
from scipy.integrate import quad
from scipy.special import j0

def j0i(x):
    def integrand(phi):
        return math.cos(x*math.sin(phi))
    return (1.0/math.pi)*quad(integrand,0,math.pi)[0]

x = numpy.linspace(0,20,200)  # sample grid; 200 points between 0 and 20
y = j0(x)  # sample J0 at all values of x
x1 = x[::10]  # subsample the original grid every 10th point
y1 = map(j0i,x1)  # evaluate the integral form at all points in x1

plot(x,y,label=r'J_0(x)')
plot(x1,y1,'r',label=r'\texttt{integ}(x)')
axhline(0,color='green',label='\texttt{nolegend}');
title(r'\texttt{Verify } J_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \phi) d\phi')
xlabel('$x$');
legend();

matshow(numpy.random.random((32,32)))
```

Verify $J_0(x) = \frac{1}{\pi} \int_0^\pi \cos(x \sin \phi) d\phi$
Python: Correlation analysis of seizure data
J. D. Hunter, University of Chicago
Final location of epileptic foci for surgery
Python - SAGE: open source mathematics
William Stein, University of Washington (NSF funding)

```
show(graphs.CubeGraph(5).plot3d())
```

```
show(graphs.CubeGraph(6).plot3d())
```

```
show(plot(sin(x^2)+x, -pi, pi, hue=0.7, thickness=3))
```
IPython: interactive Distributed/Parallel computing

- Think of Python as ’the CPU’
- IPython exposes it over the network for interactive use.

Control, analysis, etc
What does IPython offer here?

- **Easy reuse** and distribution of existing serial (‘normal’) codes.
- High-level abstractions for ‘embarrassingly parallel’ problems.
  - Out-of-the-box, low latency **task farming** tools.
  - Can be integrated into more complex codes.
- Implement any approach to parallelism you want:
  - Task farming.
  - Nodes can use traditional Message Passing (MPI).
  - Manually control bits and pieces of distributed execution.
  - Control multiple parallel codes for multiscale modeling (A. Jaramillo-Botero, Caltech)
- **Actively developed** (Colorado, Berkeley).
- **Free and fully open source.**
The curse of dimensionality.

Effective theories (homogenization, renormalization group, multiresolution...)

Non-linear, inexact (but controllable) expansions for numerical optimality?

The role of random algorithms.

We can afford inexact answers, but we need to control our errors.
A few modern algorithms

- FMM (Greengard & Rohklin)
- Separated representations (Mohlenkamp & Beylkin)
- NS-Form integral operators (Beylkin, Cheruvu, FP)
- Diffusion geometry (Coifman et al)
- Randomized low-rank approximations (Martinsson & Rohklin)
- Non-linear/non-orthogonal dimensionality reduction - beyond ICA/PCA (F. Meyer, others).

... 

- Have they had truly significant impacts yet? If not, why not?
Outline

1. Motivation
2. A case study
3. Tools & Algorithms, part I
4. Tools & Algorithms, part II
5. Outlook & questions
Algorithms will become the critical issue for computational scientific progress on many problems in the next decade.

There are issues both very theoretical and very practical.

We need reusable tools out of our effort. Implementation complexity (and hence cost) is simply too high.

These tools must be openly available: LAPACK/Netlib for the next generation of problems.

The ideas of reproducible research must be part of the whole toolchain (from code development to publication)

We need tools (languages, libraries, etc) that scale from interactive desktop exploratory work to full-scale production.
THANK YOU
Practicality

- Productivity: ease of implementation, debugging, visualization...
  - Legacy codes: integration issues.
- Computers are programmed using languages (grad students and postdocs notwithstanding)
  - Models of parallelism
  - Rapidly changing hardware landscape