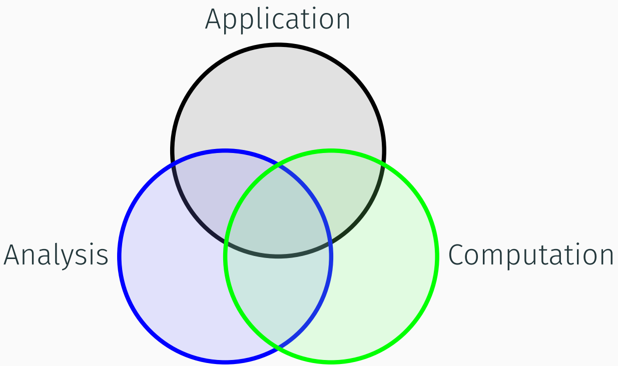
Toward Optimal Optimization in the Cloud

David Bindel

27 September 2019

Department of Computer Science
Cornell University

My Work: Computational Science & Engineering Picture



- MEMS
- Fusion
- Networks
- **Systems**

- Linear algebra
- Approximation theory
- Symmetry + structure
- **Optimization**

- **HPC / cloud**
- Simulators
- Solvers
- **Frameworks**

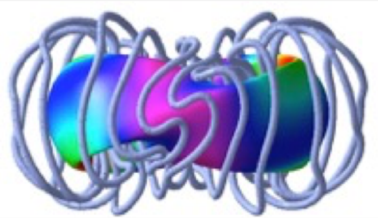
Optimizing with Expensive Physics (Stellarators)

Optimizer

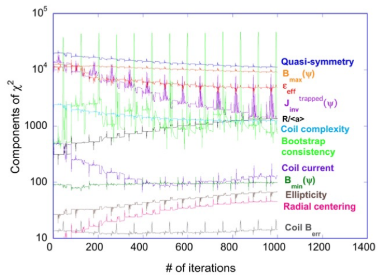
Calculate χ^2 (physics + engineering targets)

Adjust plasma boundary (or coil shape)

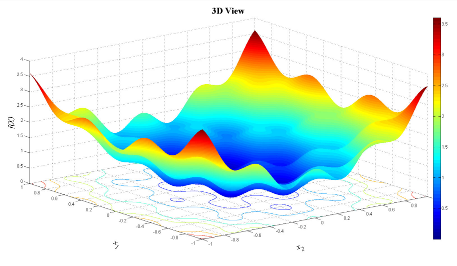
Solve 3D equilibrium



$$r(\phi, \theta) + iz(\phi, \theta) = \sum \alpha_{m,n} e^{i(m\phi - n\theta)}$$



Optimal Optimization



Two meanings for "optimization:"

- Mathematical programming / operations research:

$$\min_{x \in \Omega} c(x) \text{ s.t. constraints}$$

- High performance computing / code tuning:

  Minimize run time (usually) subject to resources

Too Slow, Now What?

What if I decide my optimization code is too slow?

- Simplify the problem?
- Use better algorithms (discretizations or optimizers)?
- Improve the initial guess or add information?
- *Parallelize*?

In the Cloud

There is no cloud.
It's just someone else's computer.

- Renting physical or virtual machines + net (IaaS)
- Not *that* different from cluster!
- Key distinctions: elasticity and shared tenancy

Some Key Characteristics

Properties of STELLOPT relevant to parallel code performance:

- Non-convex global optimization problem
- Gray box optimization
- Computationally intensive rather than data intensive
- May not need massively parallel *simulations*

Why might this be a good fit for a cloud?

Parallelizing Optimization

Different approaches for different problems:

- Big data: Parallelize across data / parameters
  - Partition data or model params across processors
  - Maybe sloppy about synchronization across processors
  - Often slow rates of convergence, but cheap steps
- Big compute (local opt): Parallelize evaluation (PDEs)
  - Partition problem domain across processors
  - May need synchronization at every time step / solver step
- Big compute (global opt): Parallelize across evaluations
  - Run concurrent evaluations while exploring design space
  - Can combine with parallelism within evaluations

How does the decomposition affect platform choices?

Performance Facts

Communication is key (across memories or nodes)

- Flops are cheap, messages are expensive
- Coordination requires communication
- Hard to scale algorithms with tight coordination

Commodity vs supercomputer: it's mostly in the network!

Performance Facts: Cloud Edition

> *There is no cloud.*
> *It's just someone else's computer.*

Overheads beyond a local compute cluster or supercomputer:

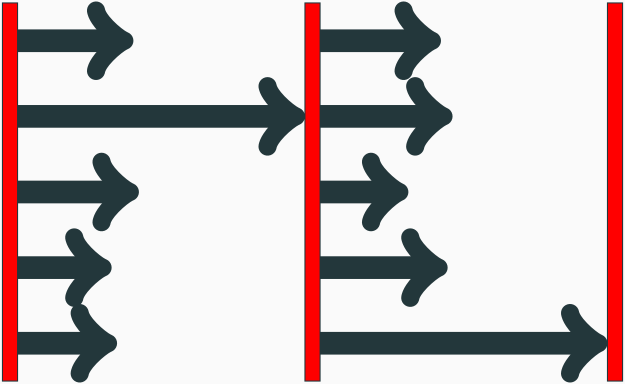
- Maybe virtualization (esp. NIC)
- Shared tenancy on nodes

Key woe: when these effects cause *high variance* in times

… at least, if tightly coupled

NB: Not unique to clouds, and not the only source of variance!

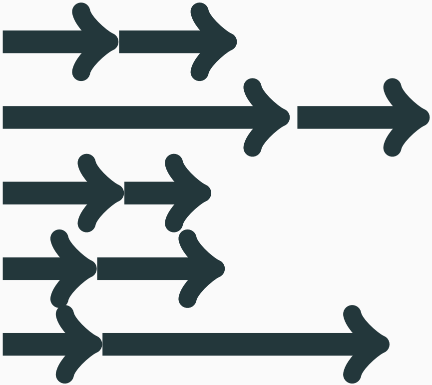
Dangerous Delays



Bad news for:

- Tightly coupled PDE solvers
- Standard step-by-step optimizers (Newton, BFGS, ...)

Breaking Barriers: Asynchronous Edition



Parallel Optimization

Exploration/exploitation tradeoff:

- Exploration: Enhance model where it is uncertain
- Exploitation: Use learned model to suggest good design

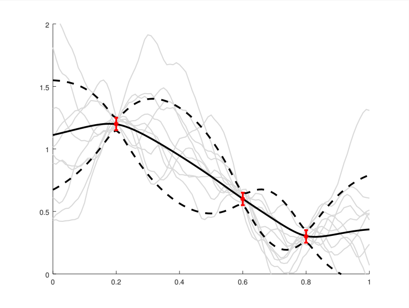
Global optimization has to balance theese.

From a parallelism perspective:

- Exploration: Independently evaluate unknown regions
- Exploitation: Finish current point to determine next

Exploration tends to be “pleasingly parallel,” so cloud-friendly.
But don’t want methods that *just* explore.

Surrogate-Based Methods



Idea: Build a *surrogate* to approximate function

- Start with initial experimental design sample
- *Exploit* surrogate to find predicted good points
- *Explore* via model of prediction uncertainty

Many such methods in PySOT: Python Surrogate Opt Toolbox

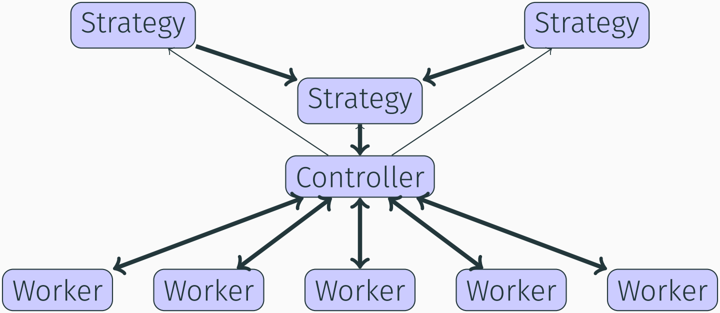
PySOT

Standard PySOT strategy:

- Fit surrogate based on experimental design
- After initial design done, assign new workers
    - Done adaptively (based on current surrogate)
    - *And* asynchronously – don't wait to dispatch
- Wrap on convergence or out-of-time

Separate optimization logic from asynchronous logistics?

POAP



- Plumbing for Optimization with Asynchronous Parallelism
- Provides an event-driven programming abstraction
- User writes *strategies* that get updates, request actions
- *Controller* handles action requests, manages workers

POAP and LibEnsemble

Fall 2018: Sabbatical semester at Argonne on LibEnsemble

- Middleware layer for ensemble calculations (opt, UQ)
- Primary target: big DOE HPC machines
- MPI only (when I started – not now)
- Shares some resemblance to POAP

Current project: Merge capabilities!

Optimal Optimization

What do we want? Good points!
When do we want them? Now!

Different notions of efficient optimization

- Sample efficiency
  - Cost measured in number of evaluations
  - Want best possible design within eval budget
  - Usual measure for $p = 1$ workers
- Time efficiency
  - Cost measured in time
  - Want best possible design within time budget
  - Usual measure for $p = P > 1$ workers
  - Design Q: Parallelize within evals, or across?

In the Cloud

Cloud elasticity $\implies$ time and samples not proportional!

- Two budgets: money (or processor-hours) and time
- Money can pay for
  - Faster individual evaluations (up to a point)
  - More concurrent evaluations
- Latter is easier in cloud, but helps mostly for exploration
- Goal: Best expected design within both budgets

Seems like a hard problem — in the early stages of this work!

Key Takeaways

- Clouds (IaaS) are good for *some* optimization problems
    - Not quite “rent-a-supercomputer”
    - Network overheads can be a stumbling block
    - We like: loosely-coupled, exploratory, asynchronous
- Need good abstractions for coding in this environment
- Opportunities/challenges in fully exploiting elasticity