# Building a Community Model for Robustness and Extensibility

This talk:
https://jedbrown.org/files/20160303-MIMCommunity.pdf

**Jed Brown** jed@jedbrown.org (CU Boulder)
Collaborators: Matt Knepley (Rice), Dave May (ETH)

Melt in the Mantle, Newton Institute, Cambridge, 2016-03-03

# Requirements for community magma software

- Usability
- Extensibility
  - Materials
  - Boundary conditions
  - Discretization
  - Packaging and distribution
- Community
- Performance
  - Solvers
  - End-to-end workflow
- Verification and Validation
- Data Assimilation

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran!

- Renders HTML 10% faster than Firefox or Chromium.
- but only if there is no JavaScript
  - recompile to use JavaScript
- Character encoding compiled in
- Mutually incompatible forks
- No confusing run-time proxy dialogs, edit file and recompile
- Proxy configuration compiled in
- For security, HTTP and HTTPS mutually incompatible
- Address in configuration file, run executable to render page
- Tcl script manages configuration file
- Plan to extend script to recompile Firetran with optimal features for each page.

# Firetran struggles with market share

- Status quo in many scientific software packages
- Why do we tolerate it?
- Is scientific software somehow different?

# Usability: Packaging and distribution

- Code must be portable – any compiler, any platform
  - Need automatic tests to confirm
- Developers underestimate challenge of installing software
- User experience damaged even when user's fault (broken environment)
- Package managers (Debian APT, RedHat RPM, MacPorts, Homebrew, etc.)
- Binary interface stability critical to packagers

# Stokes modeling

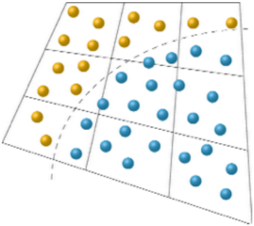
- Something we trust: **Conservation**

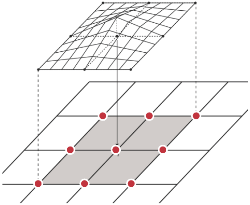
$$
\begin{aligned}
-\nabla \cdot \left[\eta Du - pI\right] &= \rho g && \text{momentum} \\
\nabla \cdot u &= 0 && \text{mass} \\
\frac{DT}{Dt} - \nabla \cdot (\kappa \nabla T) &= Q && \text{energy} \\
\frac{D\Phi_i}{Dt} &= 0 && \text{composition}
\end{aligned}
$$

  - $Du = \frac{1}{2}\left[\nabla u + (\nabla u)^T\right]$
  - Non-Newtonian Stokes, high-contrast coefficients $10^{10}$
  - Boussinesq approximation, high Rayleigh number, zero Reynolds
  - Free surface, near hydrostatic balance
- Something we don't: **Constitutive models**
  - $\eta(Du, p, T, \Phi_i)$ shear viscosity—viscoplastic, non-smooth
    - von Mises, Drucker-Prager, ...
  - $\rho(p, T, \Phi_i)$ density

# Spatial discretization

- Requirements
  - Stable for resolved and under-resolved features
  - Accurate for resolved scales
- $Q_2 - P_1^{\text{disc}}$ Finite Element
  - \+ Local mass conservation, hydrostatic mode built-
  - \+ Stable (not “stabilized”) velocity space
  - \+ ALE for moving free surface
  - \- not uniformly stable wrt. aspect ratio
- Staggered Finite Difference ($C$-grid)
  - \+ Fewer dofs for minimum resolution, full-space mu
  - \- no ALE, lower order of accuracy, stencil growth fo
- Material Point Method
  - Lagrangian marker particles
  - Nonlinearities evaluated at markers





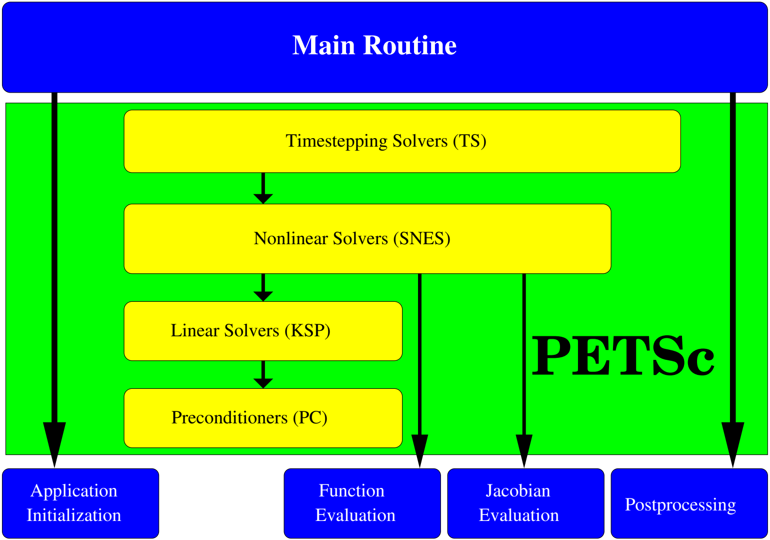
# What does magma add?

- Transport equation
- Nonlinear feedback
- Dispute about equation structure
  - 2-equation: simple to add to Stokes solver; ill-conditioned
  - 3-equation: better conditioning; local conservation issue
  - 4-equation: space compatibility, extra saddle point
- Zero-porosity limit
  - Dave says different equations in different domains
  - Arbitrary cutoff, dynamic switching
  - Perhaps it should be a variational inequality
- High porosity also matters

# Extensibility

- Easy to implement materials/rheology and boundary conditions
- Should not depend on discretization
- Packaging and distribution
  - Must be possible to package independently
  - Some authors wait for a paper to be published
  - Some authors want personal control of branding
  - Alternative is inline source modification/forking
- Must be a library – enables coupling and flexible UQ

# Flow Control for a PETSc Application



# User modifications versus plugins

- Fragmentation is expensive and should be avoided
- Maintaining local modifications causes divergence
- Better to contain changes to a plugin
- `dlopen()` and register implementations in the shared library
- Invert dependencies and avoid loops
  - `libB` depends on `libA`
  - want optional implementation of `libA` that uses `libB`
  - `libA-plugin` depends on both `libA` and `libB`
- Static libraries are anti-productive (tell your computing center)
  - Can sort-of do plugins with link line shenanigans
  - Still no reliable and ubiquitous way to handle transitive dependencies

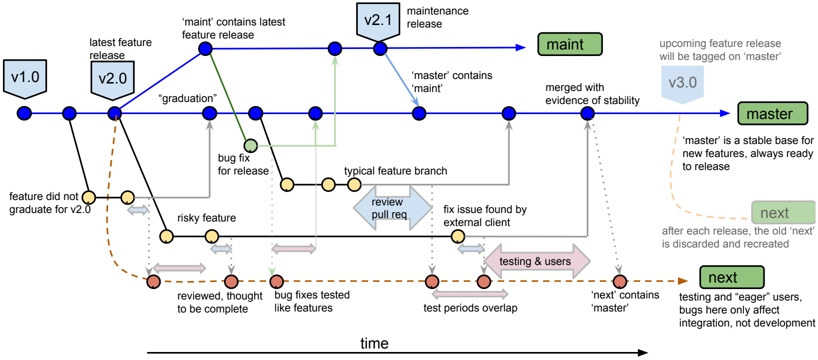
# Adjoints or not?

- Linear, backward in time model
- Efficiently evaluates
$$\frac{\partial(\text{small output})}{\partial(\text{large input})}$$
- Necessary for efficient evaluation of high-dimensional space
- Mathematical challenges
  - non-smooth models – subdifferentials
  - chaotic dynamics – sufficient averaging, non-differentiable forward map
- Technical challenges
  - Algorithmic differentiation
  - Hand differentiation
  - Needing more derivatives
  - Unsupported components inevitable

# Upstreaming and community building

- Maintainers should provide good alternatives to forking
- Welcoming environment for contributions
- Empower users – all major design decisions discussed in public
  - cf. Harvey Birdman Rule of copyleft-next
- Privacy, “scooping”, openness
  - My opinion: social problem, deal with using social means
- Major tech companies have grossly underestimated cost of forking
- In science, we cannot pay off technical debt incurred by forking
- Provide extension points to reduce cost of new development

# Simplified gitworkflows(7)




first-parent history of branch

merge history (not first-parent)

merges to be discarded when 'next' is rewound at next release

merge in first-parent history of 'master' or 'maint' (approximate "changelog")

merge to branch 'next' (discarded after next major release)

commit in feature branch (feature branches usually start from 'master')

commit in bug-fix branch (bug-fix branches usually start from 'maint' or earlier)

# Review of library best practices

- Namespace everything
  - headers, libraries, symbols (all of them)
  - use `static` and visibiliy to limit exports
- Avoid global variables
- Avoid environment assumptions; don't claim shared resources
  - `stdout`, `MPI_COMM_WORLD`
- Document interface stability guarantees, upgrade path
- Binary interface stability
- User debuggability
- Documentation and examples
- Portable, automated test suite
- Flexible error handling
- Support

# Compile-time configuration

- configuration in build system
- over-emphasis on "efficiency"
- templates are compile-time
    - combinatorial number of variants
- compromises on-line analysis capability
- create artificial IO bottlenecks
- offloads complexity to scripts and "workflow" tools
- limits automation and testing of calibration
- maintaining consistency complicates provenance
- PETSc Fail: mixing real/complex, 32/64-bit int

# Choose dependencies wisely, but practically

- Licenses
  - PETSc has a permissive license (BSD-2); anything more restrictive must be optional
  - ParMETIS license prohibits modification and redistribution
  - But bugs don't get fixed, even with patches and reproducible tests
  - Result: several packages now carry patched versions of ParMETIS – license violation and namespace collision
- Parallel ILU from Hypre
  - Users Manual says PILUT is deprecated – use EUCLID
  - EUCLID has memory errors, evidently not supported
  - Repository is closed; PETSc doesn't have resources to maintain
  - Tough luck for users
- Encapsulation is important to control complexity
- Reconfiguring indirect dependencies breaks encapsulation
- Single library may be used by multiple components in executable
  - diamond dependency graph
  - conflict unless same version/configuration can be used for both

# Verification and Validation

*Verification without validation is sport; validation without verification is magic. — Anthony Scopatz*

- Verification: solving the equations right
  - Manufactured solutions
  - Mesh refinement studies
  - Benchmarks for non-smooth/emergent behavior
- Validation: solving the right equations
  - Comparison with observations
  - Do we have good initial/boundary conditions?
  - Data assimilation

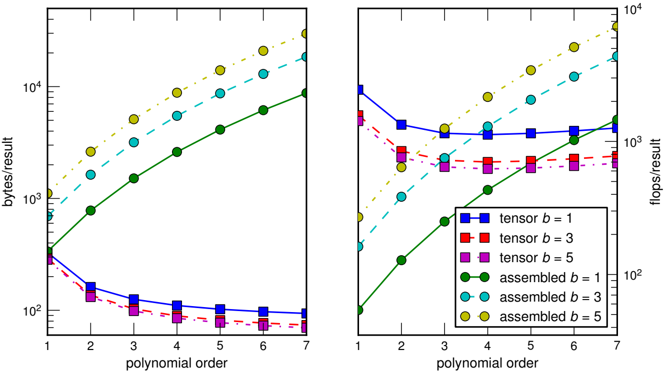
# Outline

Performance

# Solver Performance

- Bottleneck for most workflows; solver convergence plagues practitioners
- Direct – not viable in 3D
- Non-scalable iterative – not viable at high resolution
- Krylov is not magic – need quality preconditioner
- Linear Multigrid
  - Assembled vs unassembled
  - Algorithmic fundamentals depend on discretization
- Domain decomposition
  - Reliant on assembled matrices
  - New results with convergence guarantees; expensive setup
- Nonlinear
  - Newton-type methods – rely on global linearization
  - Nonlinear Multigrid or DD
  - Exciting adaptive methods, but need robustness first
- Reexamine implementation after working out convergence properties

# Performance of assembled versus unassembled




- High order Jacobian stored unassembled using coefficients at quadrature points, can use local AD
- Choose approximation order at run-time, independent for each field
- Precondition high order using assembled lowest order method
- Implementation $> 70\%$ of FPU peak, SpMV bandwidth wall $< 4\%$

# Hardware Arithmetic Intensity

| Operation | Arithmetic Intensity (flops/B) |
|---|---|
| Sparse matrix-vector product | 1/6 |
| Dense matrix-vector product | 1/4 |
| Unassembled matrix-vector product, residual | $\gtrsim 8$ |

| Processor | STREAM Triad (GB/s) | Peak (GF/s) | Balance (F/B) |
|---|---|---|---|
| E5-2680 8-core | 38 | 173 | 4.5 |
| E5-2695v2 12-core | 45 | 230 | 5.2 |
| E5-2699v3 18-core | 60 | 660 | 11 |
| Blue Gene/Q node | 29.3 | 205 | 7 |
| Kepler K20Xm | 160 | 1310 | 8.2 |
| Xeon Phi SE10P | 161 | 1060 | 6.6 |
| KNL (DRAM) | 100 | 3000 | 30 |
| KNL (MCDRAM) | 500 | 3000 | 6 |

# $Q_2$ tensor product optimization

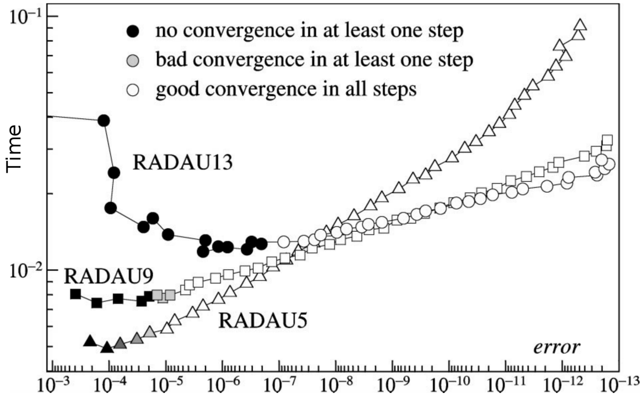
- Reference gradient $\mathscr{D}_\xi = [\hat{D} \otimes \hat{B} \otimes \hat{B}, \hat{B} \otimes \hat{D} \otimes \hat{B}, \hat{B} \otimes \hat{B} \otimes \hat{D}]$
- $\nabla_\xi \mathbf{x} = (\mathscr{D}_\xi \otimes I_3)(\mathscr{E}_e \otimes I_3)\mathbf{x}$ (29%)
- Invert $3 \times 3$ at quad. points: $\nabla_{\mathbf{x}} \xi$ (7%)

$$A\mathbf{u} = \sum_{e \in N_{\text{el}}} \underbrace{\mathscr{E}_e^T}_{\text{scatter accum}} \overbrace{\mathscr{D}_\xi^T}^{\text{tensor } 29\%} \underbrace{\Lambda\Big((\nabla_{\mathbf{x}}\xi)^T(\omega\eta)(\nabla_{\mathbf{x}}\xi)\Big)}_{\text{independent at quadrature points } 6\%} \overbrace{\mathscr{D}_\xi}^{\text{tensor } 29\%} \underbrace{\mathscr{E}_e}_{\text{gather}} \mathbf{u}$$

- Pack 4 elements at a time in vector-friendly ordering
- Intrinsics, 30% of peak AVX (SNB) and FMA (Haswell)
- Similar structure in HPGMG-FE

| Operator | Flops | Pessimal Cache Bytes | Pessimal Cache F/B | Perfect Cache Bytes | Perfect Cache F/B | Time (ms) | GF/s |
|---|---|---|---|---|---|---|---|
| Assembled | 9216 | — | — | 37248 | 0.247 | 42 | 113 |
| Matrix-free | 53622 | 2376 | 22.5 | 1008 | 53 | 22 | 651 |
| Tensor | 15228 | 2376 | 6.4 | 1008 | 15 | **4.2** | 1072 |
| Tensor C | 14214 | 5832 | 2.4 | 4920 | 2.9 | — | — |

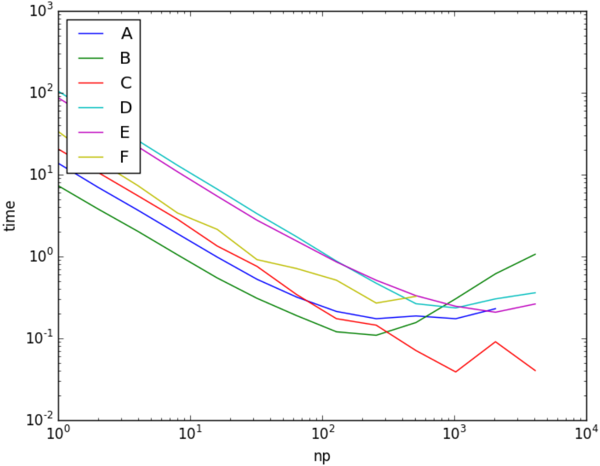
# Work-precision diagram: *de rigueur* in ODE community




[Hairer and Wanner (1999)]

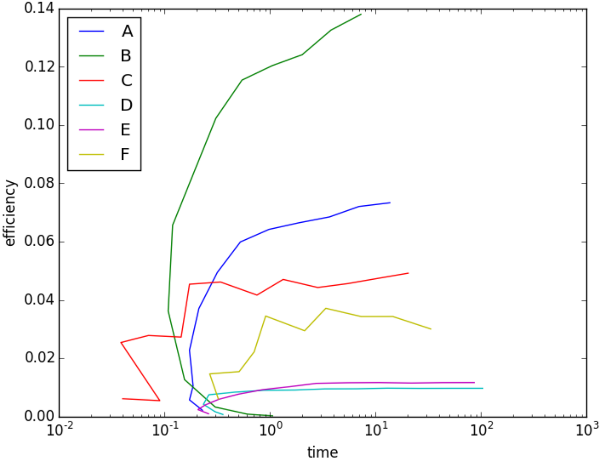
- Tests discretization, adaptivity, algebraic solvers, implementation
- No reference to number of time steps, flop/s, etc.
- Useful performance results inform *decisions* about *tradeoffs*.

# Strong Scaling: efficiency-time tradeoff



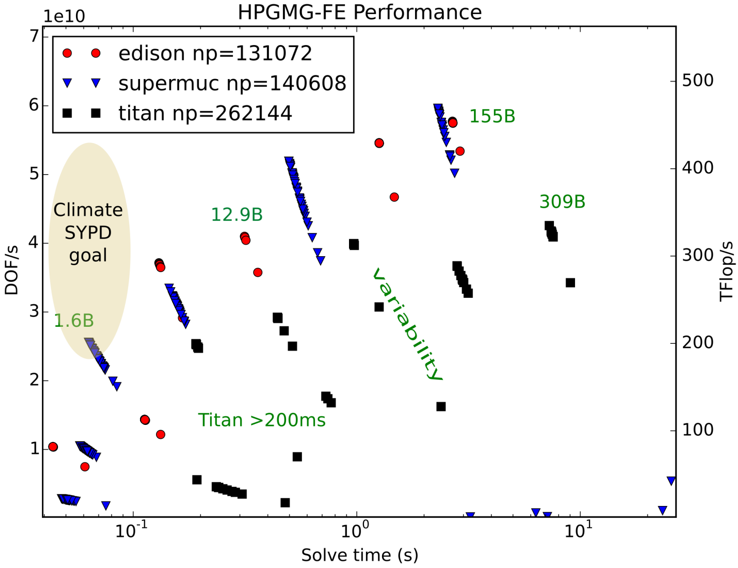

- Good: shows absolute time
- Bad: log-log plot makes it difficult to discern efficiency
  - Stunt 3: `http://blogs.fau.de/hager/archives/5835`
- Bad: plot depends on problem size

# Strong Scaling: efficiency-time tradeoff




- Good: absolute time, absolute efficiency (like DOF/s/cost)
- Good: independent of problem size for perfect weak scaling
- Bad: hard to see machine size (but less important)

# HPGMG-FE on Edison, SuperMUC, Titan



# End-to-end performance

- Education
- Preprocessing/custom implementation
- HPC Queue
- Execution time
  - Solvers
- I/O
- Postprocessing/visualization

# Exascale Science & Engineering Demands

- Model fidelity: resolution, multi-scale, coupling
  - Transient simulation is not weak scaling: $\Delta t \sim \Delta x$
- Analysis using a sequence of forward simulations
  - Inversion, data assimilation, optimization
  - Quantify uncertainty, risk-aware decisions
- Increasing relevance $\implies$ external requirements on time
  - Policy: 5 SYPD to inform IPCC
  - Weather, manufacturing, field studies, disaster response
- "weak scaling" [...] will increasingly give way to "strong scaling"

  [The International Exascale Software Project Roadmap, 2011]
- ACME @ 25 km scaling saturates at $< 10\%$ of Titan (CPU) or Mira
  - Cannot decrease $\Delta x$: SYPD would be too slow to calibrate
  - "results" would be meaningless for 50-100y predictions, a "stunt run"
- **ACME v1 goal of 5 SYPD is pure strong scaling.**
  - Likely faster on Edison (2013) than any DOE machine –2020
  - Many non-climate applications in same position.

# Tim Palmer's call for 1km (Nature, 2014)

Running a climate simulator with 1-kilometre cells over a timescale of a century will require 'exascale' computers capable of handling more than $10^{18}$ calculations per second. Such computers should become available within the present decade, but may not become affordable for individual institutes for another decade or more.

- Would require $10^4$ more total work than ACME target resolution
- 5 SYPD at 1km is like 75 SYPD at 15km, assuming infinite resource and perfect weak scaling
- ACME currently at 3 SYPD with lots of work
- Two choices:
    1. compromise simulation speed—this would come at a high price, impacting calibration, data assimilation, and analysis; or
    2. ground-up redesign of algorithms and hardware to cut latency by a factor of 20 from that of present hardware
- DE Shaw's Anton is an example of Option 2
- Models need to be constantly developed and calibrated
    - custom hardware stifles algorithm/model innovation
- Exascale roadmaps don't make a dent in 20x latency problem

# Outlook

- Scientific software shouldn't be "special"
- Usability is essential
- Defer all decisions to run time
- Plugins are wonderful for users and contributors
- Reviewing patches/educating contributors is a thankless task, but crucial
- Application scaling mode must be scientifically relevant
- Versatility is needed for model coupling and advanced analysis
- Abstractions must be durable to changing scientific needs
- Plan for the known unknowns and the unknown unknowns
- The real world is messy!