PETSc: Technical and social aspects of library development

This talk: `https://jedbrown.org/files/20160225-PETSc.pdf`

Jed Brown `jed@jedbrown.org` (CU Boulder)
Satish Balay, Matt Knepley, Lois Curfman McInnes, Karl Rupp, Barry Smith

Scientific Software Days, UT Austin, 2016-02-25

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.

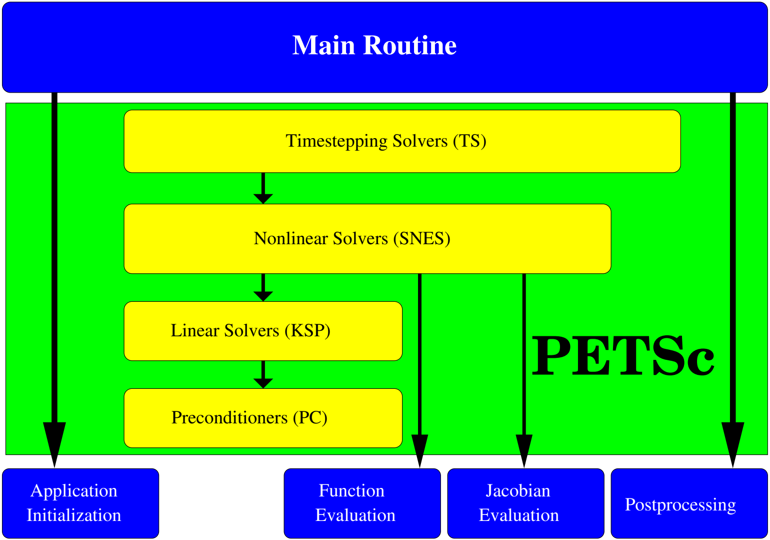
Firetran!

- 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?

Flow Control for a PETSc Application



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

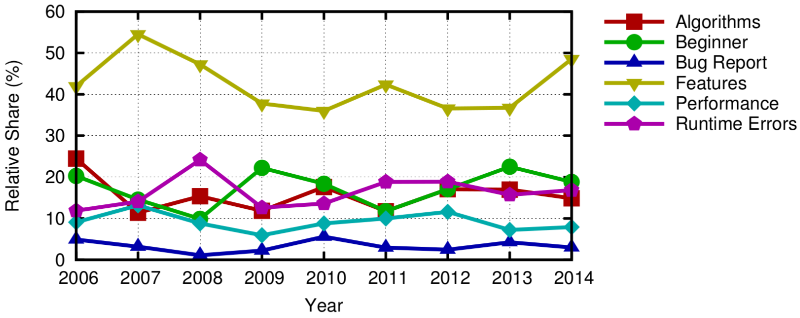
Choose dependencies wisely, but practically

- 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

Packaging and distribution

- 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
- PETSc has made changes to install schema to help packagers

Support: petsc-users mailing list



- 964 emails in 2006 $\to$ 3947 emails in 2014
- Also have petsc-dev and petsc-maint
- Hard to tell at first contact if user is worth helping
  - Lots of work
  - Success stories are very satisfying
- 12 contributors in 2006–2007, 46 contributors in 2015

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

Controlling the Binary interface

- Recompiling code is wasted productivity
- Implementation concerns (private variables, new virtual methods) should not require recompiling user code
- PETSc uses opaque pointers and the “delegator” (aka. “pointer to implementation”) pattern.
- Static function overhead insignificant, incremental cost less than 2 cycles
- Better for debugging
- Easier to expose libraries to dynamic programming languages

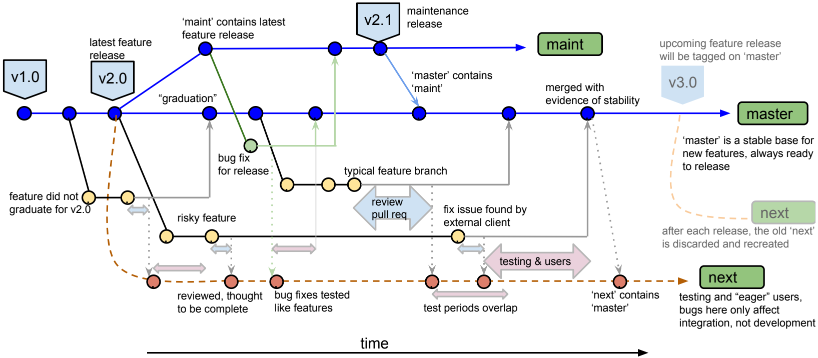
Upstreaming and community building

- 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

Workflow ideals

- 'master' is always stable and ready to release
- features are complete and tested before appearing in 'master'
- commits are minimal logically coherent, reviewable, and testable units
- related commits go together so as to be reviewable and debuggable by specialist
- new development is not disrupted by others' features and bugs
- rapid collaboration between developers possible
- `git log --first-parent maint..master` reads like a changelog
- bugs can be fixed once and anyone that needs the fix can obtain it without side-effects

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)

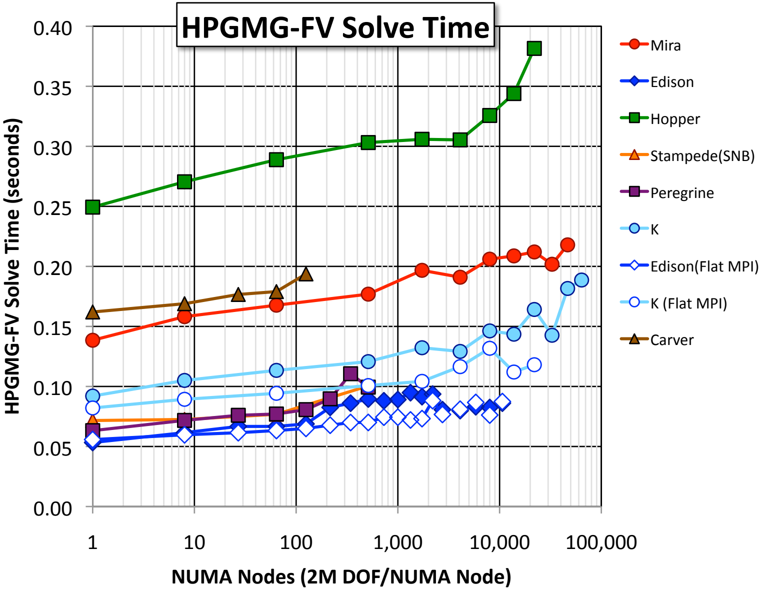
Best practices

- Every branch has a purpose
- Distinguish integration branches from topic branches
- Do all development in topic branches
  - `git checkout -b my/feature-branch master`
- Namespace your branches if working on a shared repository
- Merge integration branches “forward”
  - `maint-1` $\rightarrow$ `maint` $\rightarrow$ `master` $\rightarrow$ `next`
  - `git checkout -b my/bugfix-branch maint-1`
- Write clear commit messages for reviewers and people trying to debug your code
- Avoid excessive merging from upstream
  - Always write a clear commit message explaining what is being merged and why
- Always merge topic branches as non-fast-forward (`merge --no-ff`)
- Gracefully retry if you lose a race to shared integration branch
  - This maximizes utility of `--first-parent` history

Messaging from threaded code

- Off-node messages need to be packed and unpacked
- Many MPI+threads apps pack in serial – bottleneck
- Extra software synchronization required to pack in parallel
  - Formally $O(\log T)$ critical path, T threads/NIC context
  - Typical OpenMP uses barrier – oversynchronizes
- `MPI_THREAD_MULTIPLE` – atomics and $O(T)$ critical path
- Choose serial or parallel packing based on T and message sizes?
- Hardware NIC context/core now, maybe not in future
- What is lowest overhead approach to message coalescing?

HPGMG-FV: flat MPI vs MPI+OpenMP (Aug 2014)



- c/o Sam Williams

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 Δx: 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 important
- Support requires debugging via email
- 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!