ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2012

Sources of complexity

- problem size
  - cpu time, amount of memory, cpu count
- project size
  - asset complexity: lines of code, entry points, files
  - dependencies: number of modules, third party libraries
  - run-time complexity: number of object types and instances
- project longevity
  - duty cycle, life cycle
  - cost/benefit of reuse
  - managing change: people, software, hardware, technology
- locality of needed resources
  - computation and persistence: where, how, when, who
- usability
  - learning curve, ease of access, interfaces, security, etc.

Risk mitigation

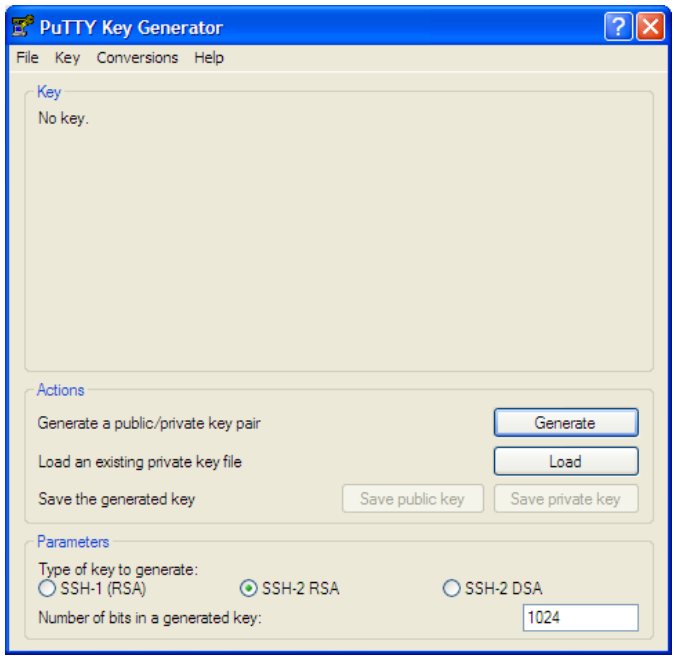
- verification and validation
  - invariants, consistency checks, code introspection
  - benchmark problems
  - reality checks
- key software practices
  - source control: e.g. cvs; svn; git, hg, bzr
  - uniform build procedure: platforms, compilers, dependencies, but also build types
  - testing: unit, component, application; mutation, regression; *automate*
  - tracking of bugs and feature requests: gnats, bugzilla, trac
  - documentation: doxygen, epydoc; wiki
  - code reviews
- project coherence
  - design methodology
  - flexible software architecture
  - coding and documentation standards
  - regular meetings

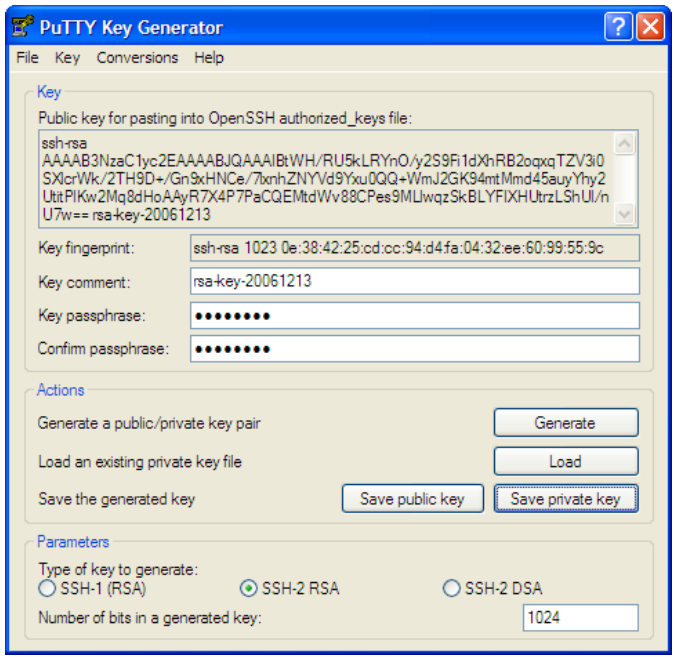
Remote access to computational resources

- `ssh`, the *secure shell*:
  - currently the most flexible and secure method
  - authentication, strong data encryption
  - interactive sessions, remote shell services, data transfer, command execution
- authentication requires a *private* and a *public* key
  - the private key is protected by a *passphrase*; must be long enough to be difficult to guess, but memorable because it is not recoverable
  - you only need one private key, and it only needs to be installed on machines whose keyboard you actually touch
  - you install your public key on every machine to which you require access
    - on unix machines: in `~/.ssh/authorized_keys`
  - keys are created using `ssh-keygen`; on windows there is `PuTTY`
- most installations support `ssh-agent`
  - a wrapper around your interactive session that asks you for your passphrase, loads your private key in memory, and negotiates authentication with `ssh` servers automatically

Key generation with PuTTY

- the PuTTY screen before and after a key has been generated





Bazaar – setup

- we are going to use `bzr`, a source control system, for both homework and final project submissions
- installation instructions, documentation and tutorials available from
  `http://bazaar.canonical.com`
- bazaar keeps track of authors in its revision history, so it's a good idea to give it your name and an email address

```
#> bzr whoami "Michael Aivazis <aivazis@caltech.edu>"
```

- you should verify that it worked by asking

```
#> bzr whoami
Michael Aivazis <aivazis@caltech.edu>
```

- the discussion here is specialized to the class repository, but the ideas generalize easily

Bazaar – initialization

- first, decide where you would like to place the folder with the homework repository, open a terminal window, and change your working directory to
- create a new repository

```
#> bzr init acm114
Created a standalone tree (format: 2a)
```

- `bzr` will create the new folder for you and initialize it; change your working directory to the new folder

```
#> cd acm114
```

- you can ask `bzr` for details about the current repository

```
#> bzr info
Standalone tree (format: 2a)
Location:
  branch root: .
```

- `bzr` commands are *repository global*: they apply to the entire repository, unless explicitly restricted to a particular file or folder

Getting information from the class server

- `bzr` is a *distributed* source control system: each repository has a copy of the entire change history, and can evolve independently
- you are responsible for sharing information among repositories by invoking the repository synchronization commands *explicitly* and providing the location of the remote repository
- populate your new `acm114` repository by *pulling* from the class server

```
#> bzr pull --remember bzr+ssh://acm114@acm114.caltech.edu
+N assigment-1/
+N assigment-1/20120120.pdf
+N assigment-1/20120120.tex
+N assigment-1/setup.tex
All changes applied successfully.
Now on revision 1.
```

- `--remember` asks bzr to use the class repository as the default: afterwards, you can just `bzr pull` without mentioning the source
- `bzr+ssh` is the connection scheme that employs `ssh` for authentication
- `acm114@acm114.caltech.edu` is the address of your homework repository on the class server

Working in the repository

- one of the problems in the first assignment asks you to write a routine that computes the first few terms of some series in a language of your choice; let's assume that you picked `python`, and you created a file `dilog.py`

```
#> cd assignment-1
#> vi dilog.py
```

- you can inform `bzr` that you would like to place `dilog.py` under source control

```
#> bzr add dilog.py
adding assigment-1/dilog.py
```

- let's check what `bzr` knows about the repository

```
#> bzr status
added:
  assigment-1/dilog.py
```

Committing and submitting

- after *exhaustive* editing and testing, you are exhausted; time to take a break:

```
#> bzr commit -m "assignment-1: first draft of the dilog routine"
#>   dilog.py
Committing to: /home/mga/courses/acm114
added assigment-1/dilog.py
Committed revision 2.
```

- rinse, repeat...
- time to *push* your work to the course server

```
#> bzr push --remember bzr+ssh://acm114@acm114.caltech.edu
...
Pushed up to revision 7
```

- again, the `--remember` stores the location of the remote repository so that subsequent push operations do not need to specify it