Imperial College London

Continuous Integration for Nektar++ with Buildbot

Chris Cantwell

CI for HPC Workshop, Imperial College
10th November 2017

Nektar++: Spectral/hp element framework

NEKTAR++

SPECTRAL/HP ELEMENT FRAMEWORK

Provide an efficient open environment which allows the seamless use of high- and low-order finite element methods.

Nektar++: Spectral/hp element framework

NEKTAR++

SPECTRAL/HP ELEMENT FRAMEWORK

Provide an efficient open environment which allows the seamless use of high- and low-order finite element methods.

- Collaboration between Imperial College London and University of Utah
- Contributions from collaborators in UK, Germany, Sweden, Denmark, US, South Korea
- Development driven by challenging science
- 11 years old

Nektar++: Spectral/hp element framework

NEKTAR++

SPECTRAL/HP ELEMENT FRAMEWORK

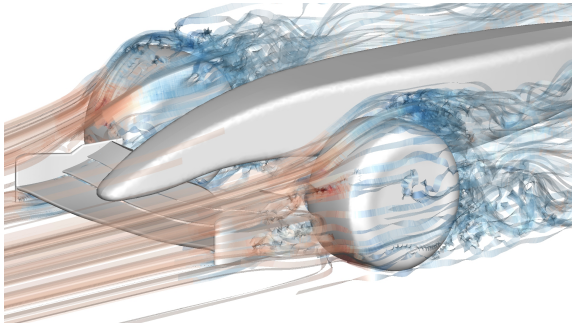
Provide an efficient open environment which allows the seamless use of high- and low-order finite element methods.

- Collaboration between Imperial College London and University of Utah
- Contributions from collaborators in UK, Germany, Sweden, Denmark, US, South Korea
- Development driven by challenging science
- 11 years old

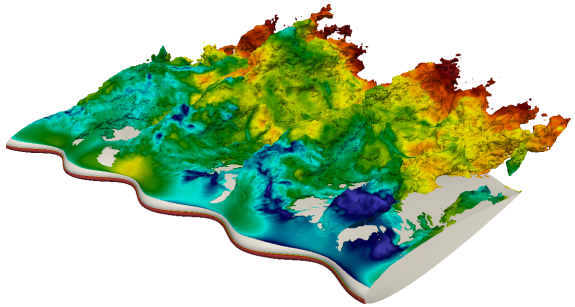



Application Area: Aerodynamics

Formula 1 Car (Lombard, Sherwin)

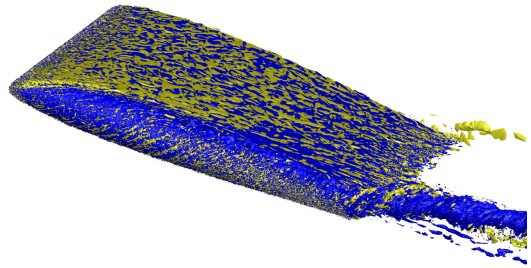


NACA 0012 with wavy leading-edge (Serson)

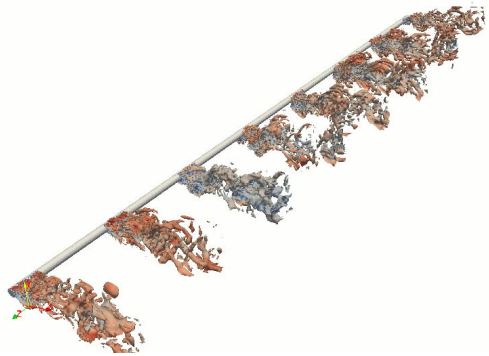


LES of wingtip vortex at $Re = 1.2$M

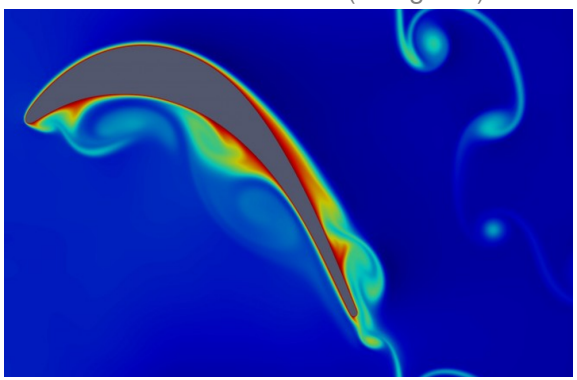
(Lombard, Moxey, Sherwin)



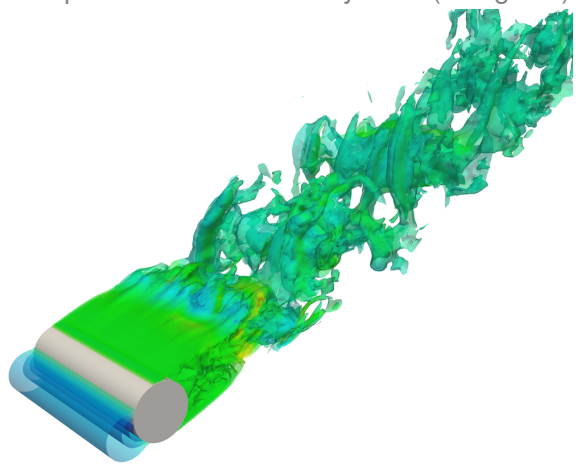
Vortex-induced vibration of cylinder (Bao)



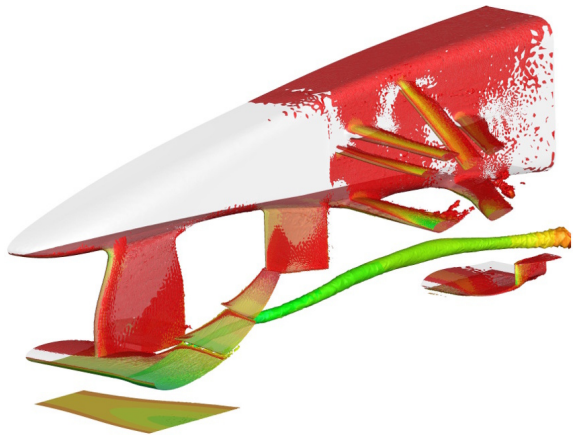
T106C turbine blade (Mengaldo)



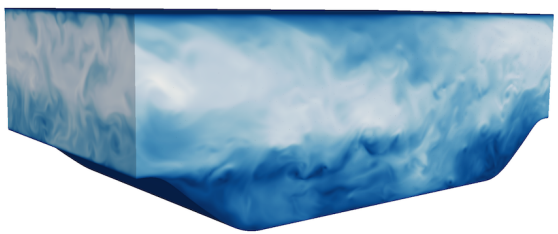
Compressible Flow over a cylinder (Mengaldo)



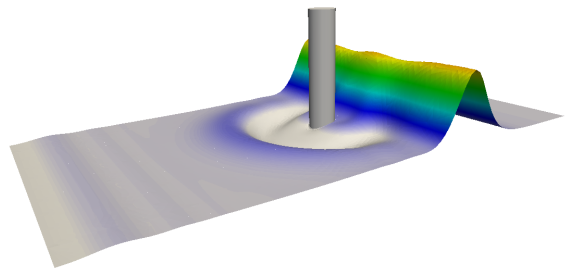
Y250 Wing on F1 car (Lombard, Sherwin)



Turbulent hill (Moxey)

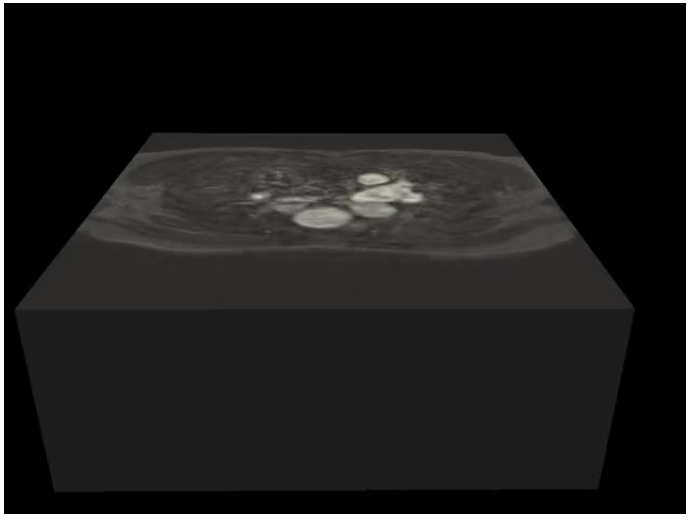


Shallow Water Equations (Eskilsson)

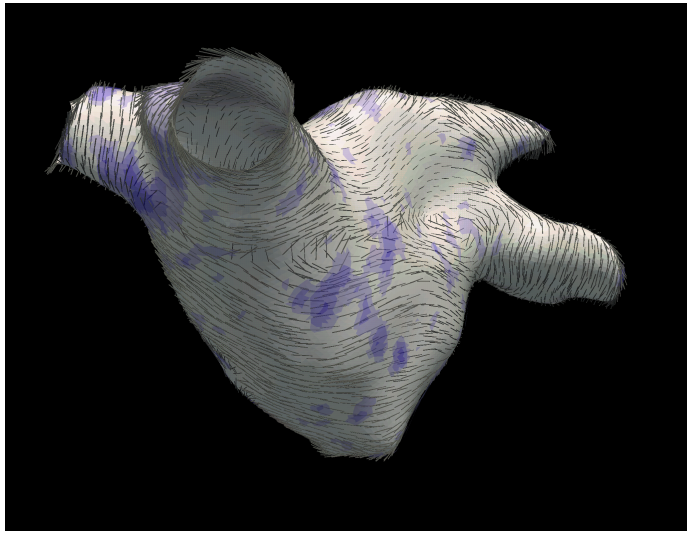


Application Areas: Biomedical

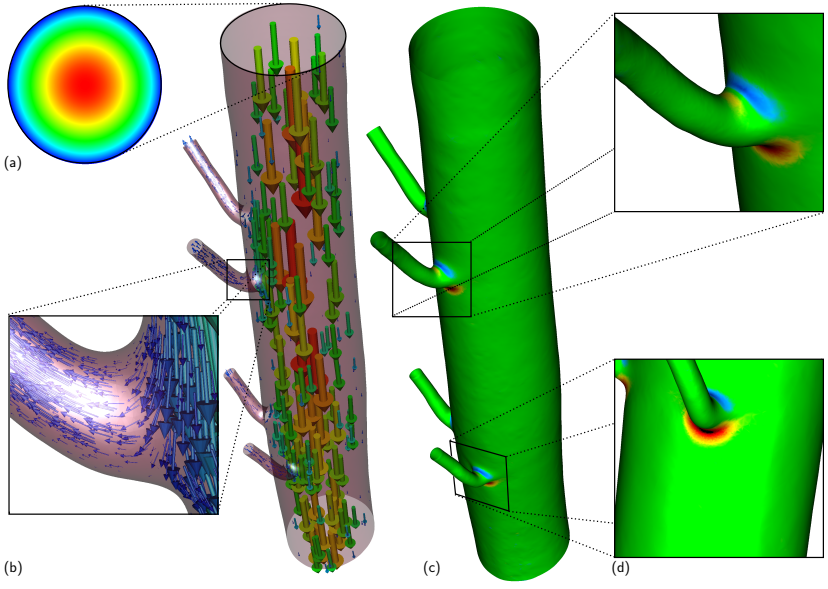
Cardiac arrhythmia (Cantwell)



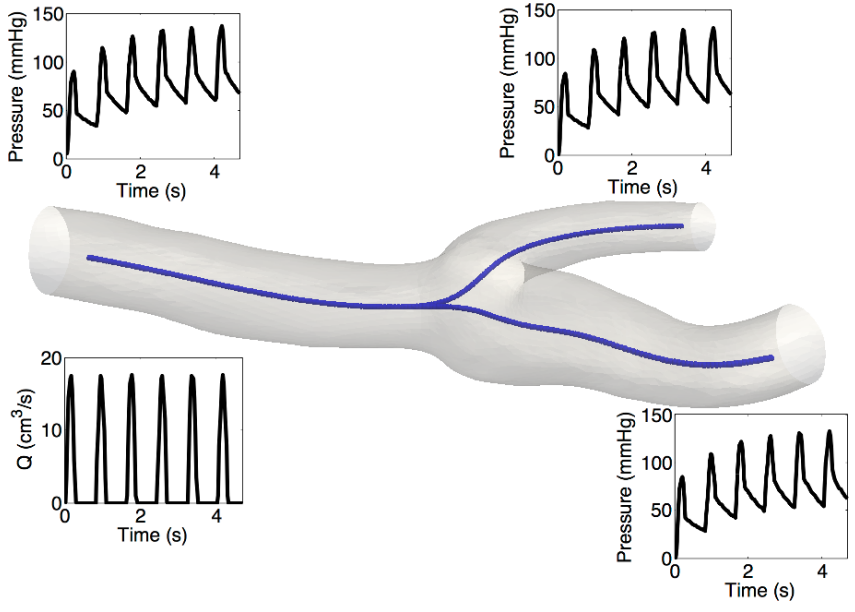
Personalised cardiac electrophysiology (Ali, Cantwell)



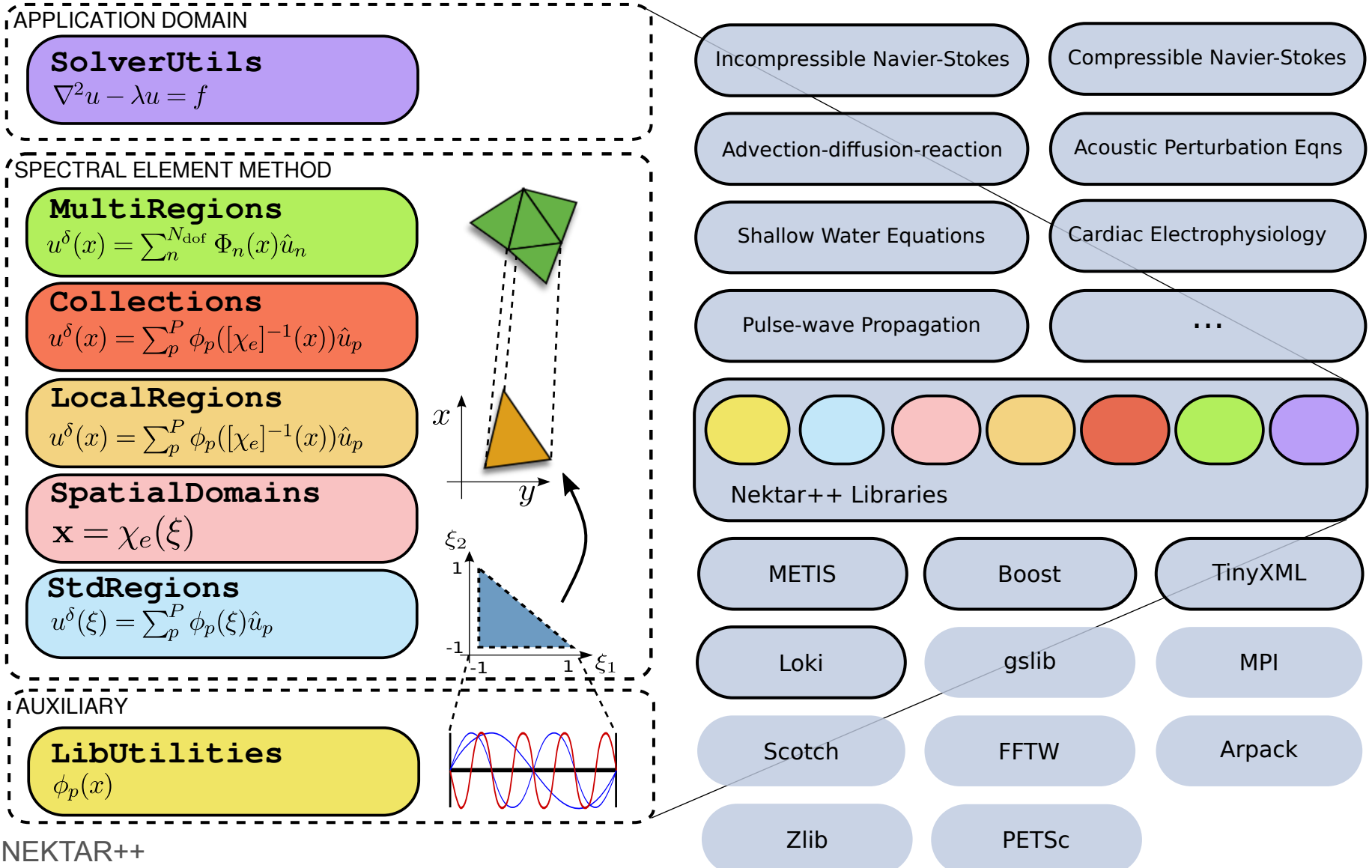
Formation of atherosclerosis (Mohamied)



Arterial 1D pulse wave propagation (Comerford)



Nektar++: Architecture



Nektar++: RSE Summary

- C++ object-oriented, templated code (Lines: 265k libraries, 50k solvers)
- Cross-platform (Linux, OS X, Windows)
- CMake (and CTest)
- Support for a range of optional features / 3rd-party libraries
- Documentation: **Tutorials** for new users, **User Guide** (LaTeX).
- Python bindings (in development)
- **Binary packages** built for a number of platforms, automated install scripts available for others.

Nektar++: RSE Summary

- C++ object-oriented, templated code (Lines: 265k libraries, 50k solvers)
- Cross-platform (Linux, OS X, Windows)
- CMake (and CTest)
- Support for a range of optional features / 3rd-party libraries
- Documentation: **Tutorials** for new users, **User Guide** (LaTeX).
- Python bindings (in development)
- **Binary packages** built for a number of platforms, automated install scripts available for others.
- **Tested** on a wide range of desktop operating systems

- . . . as well as on a range of **supercomputers**:
  - SGI ICE (CX2, Imperial College London)
  - Blue Gene/Q (Mira, Argonne National Laboratory)
  - Cray XC30 (ARCHER, UK National Supercomputer)
- Regular releases (current is 4.4.1)

Testing Nektar++

- Custom 'Tester' tool in Nektar++
- Around 500 unit and regression tests (serial / parallel)
- Test input file specifies job: `Helmholtz_P7.tst`

```
<test>
    <description>Helmholtz 2D CG with P=7</description>
    <executable>Helmholtz2D</executable>
    <parameters>-v Helmholtz2D_P7.xml</parameters>
    <files>
        <file description="Session File">Helmholtz2D_P7.xml</file>
    </files>
    <metrics>
        <metric type="L2" id="1">
            <value tolerance="1e-8">6.82372e-07</value>
        </metric>
        <metric type="Linf" id="2">
            <value tolerance="1e-8">9.43712e-07</value>
        </metric>
    </metrics>
</test>
```

Development Process

Key tools:

- Git (repository, branching, diffs)
- GitLab (web-interface to git, issues, merge requests)
- Buildbot (CI)

Branch merge process:

1. Merge in latest master
2. Create merge request (update CHANGELOG)
3. Run buildbot (address any issues)
4. Other developers review / check docs / check buildbot
5. Senior developer performs merge

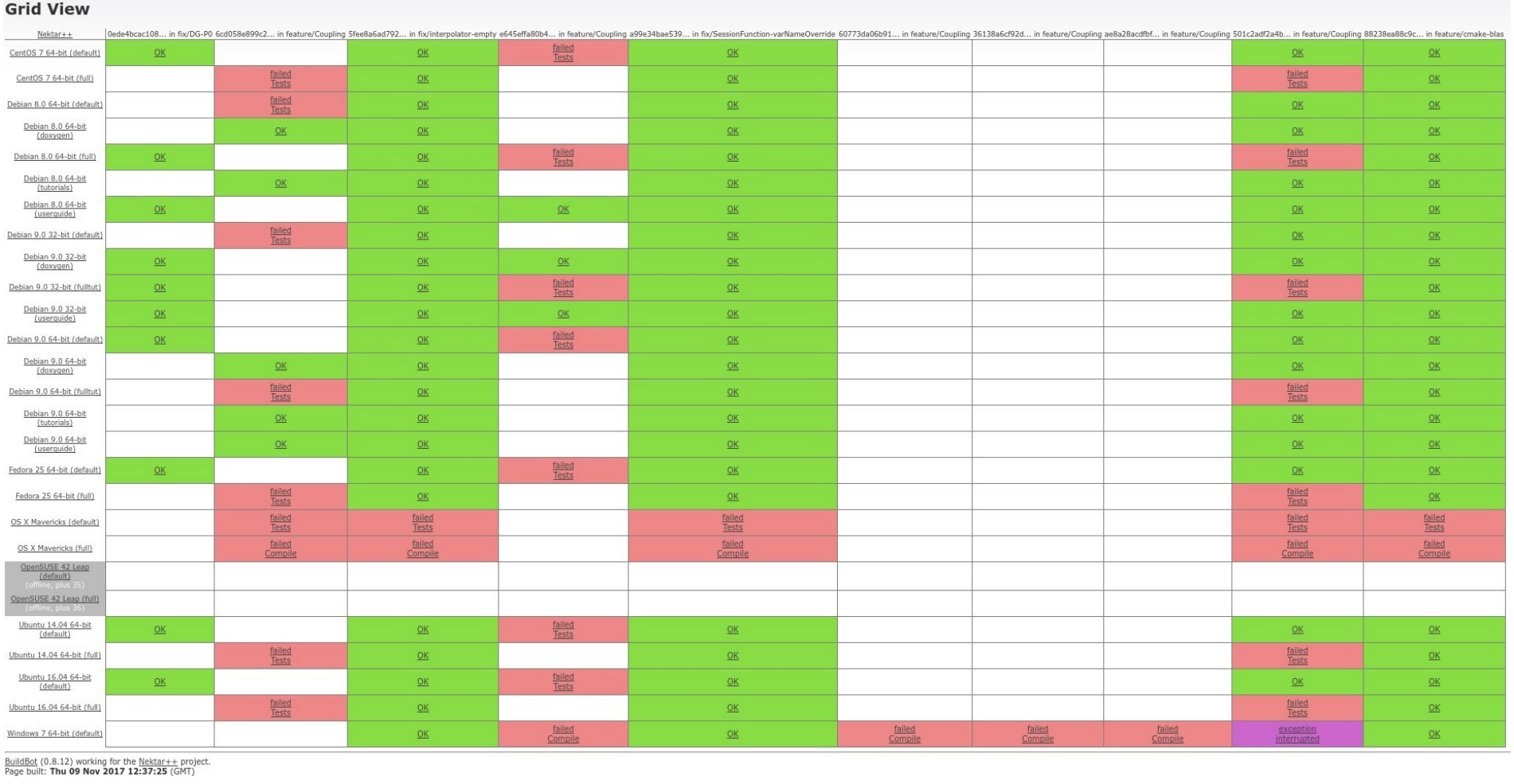
Nektar++ Buildbot motivation

Growing developer-base required automated tools to ensure retain code correctness.

- Flexibility compared to other available tools (at the time). We have:
  - Multiple languages
  - Complex builds (different optional packages)
  - Multiple types (main code, tutorial codes)
  - Package generation
- Free, open-source, big community
- Easy to set up (packages for most OS distributions)
- Fast interface
- Hooks into GitLab.

Nektar++ Buildbot webstatus

http://buildbot.nektar.info



Grid View

Nektar++	0ede4bcac108... in fix/DG-P0	6cd058e899c2... in feature/Coupling	5fee8a6ad792... in fix/interpolator-empty	e645effa80b4... in feature/Coupling	a99e34bae539... in fix/SessionFunction-varNameOverride	60773da06b91... in feature/Coupling	36138a6cf92d... in feature/Coupling	ae8a28acdfbf... in feature/Coupling	501c2adf2a4b... in feature/Coupling	88238ea88c9c... in feature/cmake-blas
CentOS 7 64-bit (default)	OK		OK	failed Tests	OK				OK	OK
CentOS 7 64-bit (full)		failed Tests	OK		OK				failed Tests	OK
Debian 8.0 64-bit (default)		failed Tests	OK		OK				OK	OK
Debian 8.0 64-bit (doxygen)		OK	OK		OK				OK	OK
Debian 8.0 64-bit (full)	OK		OK	failed Tests	OK				failed Tests	OK
Debian 8.0 64-bit (tutorials)		OK	OK		OK				OK	OK
Debian 8.0 64-bit (userguide)	OK		OK	OK	OK				OK	OK
Debian 9.0 32-bit (default)		failed Tests	OK		OK				OK	OK
Debian 9.0 32-bit (doxygen)	OK		OK	OK	OK				OK	OK
Debian 9.0 32-bit (fulltut)	OK		OK	failed Tests	OK				failed Tests	OK
Debian 9.0 32-bit (userguide)	OK		OK	OK	OK				OK	OK
Debian 9.0 64-bit (default)	OK		OK	failed Tests	OK				OK	OK
Debian 9.0 64-bit (doxygen)		OK	OK		OK				OK	OK
Debian 9.0 64-bit (fulltut)		failed Tests	OK		OK				failed Tests	OK
Debian 9.0 64-bit (tutorials)		OK	OK		OK				OK	OK
Debian 9.0 64-bit (userguide)		OK	OK		OK				OK	OK
Fedora 25 64-bit (default)	OK		OK	failed Tests	OK				OK	OK
Fedora 25 64-bit (full)		failed Tests	OK		OK				failed Tests	OK
OS X Mavericks (default)		failed Tests	failed Tests		failed Tests				failed Tests	failed Tests
OS X Mavericks (full)		failed Compile	failed Compile		failed Compile				failed Compile	failed Compile
OpenSUSE 42 Leap (default) (offline, plus 35)										
OpenSUSE 42 Leap (full) (offline, plus 36)										
Ubuntu 14.04 64-bit (default)	OK		OK	failed Tests	OK				OK	OK
Ubuntu 14.04 64-bit (full)		failed Tests	OK		OK				failed Tests	OK
Ubuntu 16.04 64-bit (default)	OK		OK	failed Tests	OK				OK	OK
Ubuntu 16.04 64-bit (full)		failed Tests	OK		OK				failed Tests	OK
Windows 7 64-bit (default)			OK	failed Compile	OK	failed Compile	failed Compile	failed Compile	exception interrupted	OK

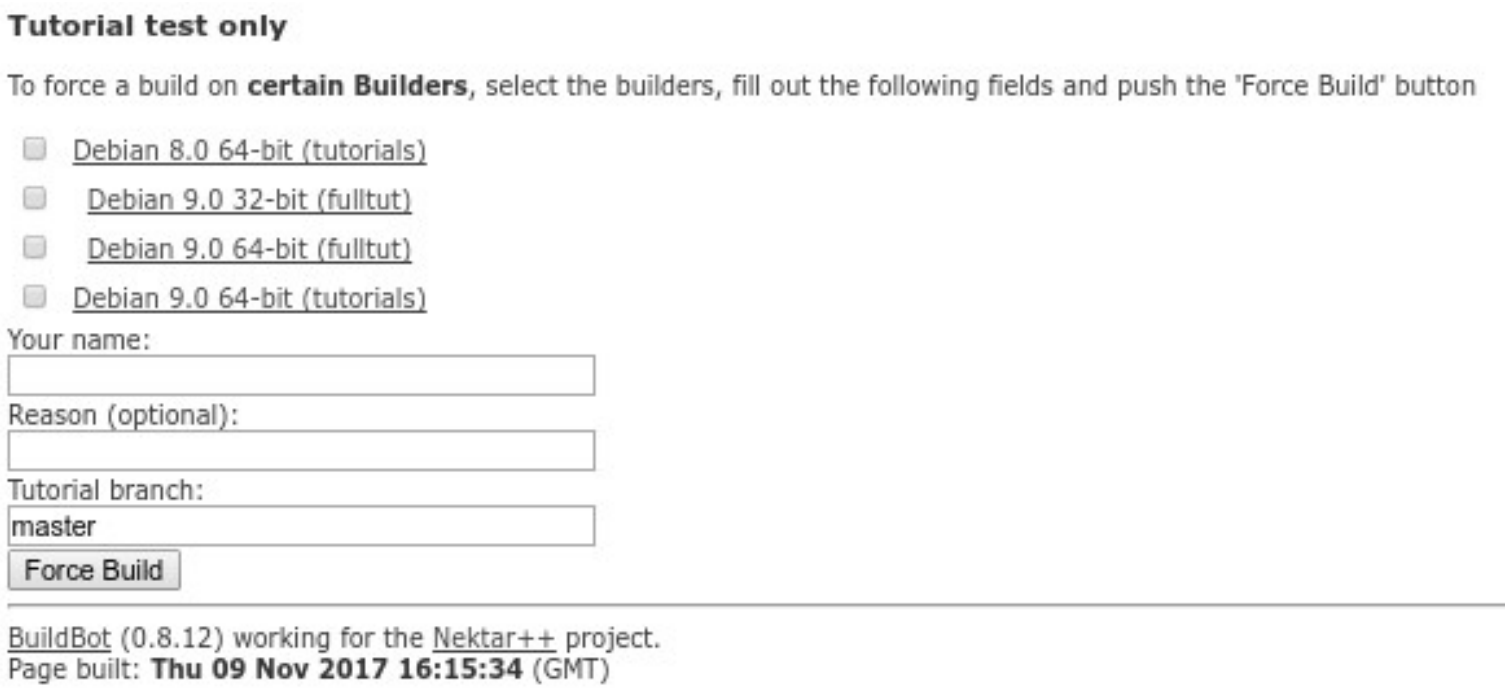
BuildBot (0.8.12) working for the Nektar++ project.
Page built: **Thu 09 Nov 2017 12:37:25** (GMT)

Multiple builders, configurations and build types; forks; submodules

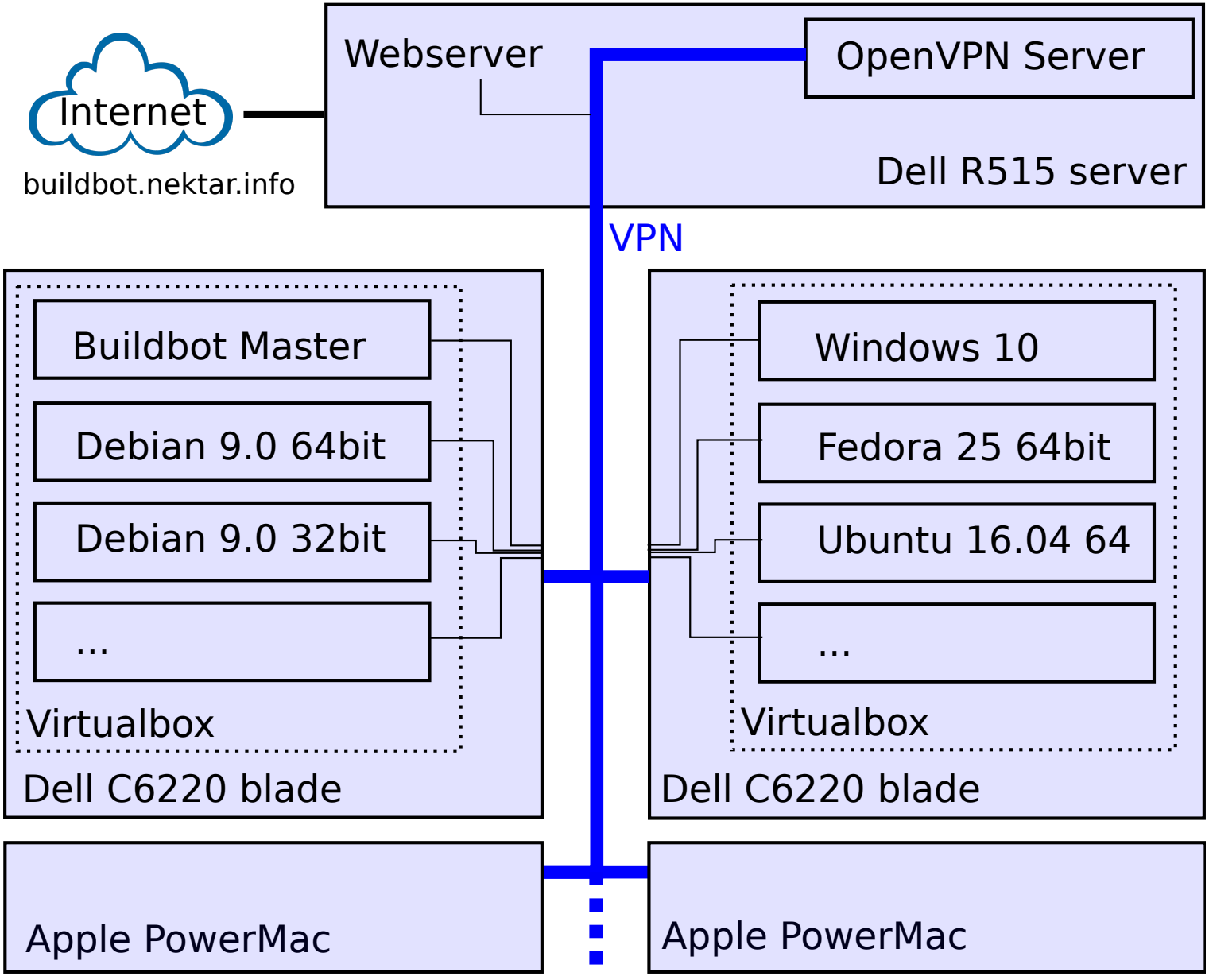
Nektar++ Buildbot forcing builds

Developers can force builds:

- Different build configurations (complete, tutorials only, quick test)
- Force on all builders
- Force on selected builders



Nektar++ Buildbot Infrastructure



Nektar++ Buildbot Configuration

Buildbot is written in Python, leverage that flexibility to configure:

```
slaves = [
  {'desc': "Debian 8.0 64-bit",     'name': "debian8",
   'pwd' : "#####",                 'type': "linux",
   'spec': "gcc-4.9.2,boost-1.55", 'ncpus' : 4,
   'cfgs': ["default", "full", "doxygen", "userguide", "tutorials"]},
  ...
```

Build steps are used to generate build factories:

```
linuxConfig = shell.ShellCommand(
    name="Configure",
    workdir="build/builds",
    command=['cmake',buildtype,alltests,timings,tploki,tptinyxml,'../
    env={'PATH': pathstr, 'LD_LIBRARY_PATH': ldpathstr},
    locks=[ waterloo_disk.access('counting')]
)
```

Locks avoid oversubscribing blades. Map configurations to build factories:

```
cfgs = {
    "default"    : { "linux": f1, "win32" : f1winxp, "win64": f1win7,
                     "osx" : f1 },
    "full"       : { "linux": f2, "osx"   : f2 },
    ...
}
```

Nektar++ Buildbot Configuration

Finally, we populate the list of slave / builder combinations:

```
for s in slaves:
    c['slaves'].append(BuildSlave(
                s['name'], s['pwd'], max_builds=1,
                notify_on_missing="c.cantwell@imperial.ac.uk",
                missing_timeout=3600,
                properties={'ncpus' : int(floor(s['ncpus']/2))}))

    for t in s['cfgs']:
        if s['type'] in cfgs[t]:
            bname = s['desc'] + " (" + t + ")";
            c['builders'].append({'name'       : bname,
                                  'slavenames' : s['name'],
                                  'builddir'   : s['name'] + "-" + t,
                                  'factory'    : cfgs[t][s['type']]})
            names.append(bname)
```

Add schedulers to trigger builds or allow forced builds on subsets of builders.

Summary

- Continuous Integration is critical in creating robust software.
- CI is an integral part of the Nektar++ development process.
- Nektar++ CI is used daily and on each merge to the codebase.
- Buildbot was chosen for the Nektar++ project due to flexibility.
- Resource costs for cross-platform coverage.
- Time costs associated with maintenance.

What next?:

- Continuous Performance Monitoring.
  - Test for performance degradation
  - Need dedicated hardware
  - How to measure / verify - hardware specific?

Thank you for listening!