THE UNIVERSITY of York

High Performance Computing - Tools

Prof Matt Probert

http://www-users.york.ac.uk/~mijp1

Overview

- Compilers
- Libraries
- Static Analysis
- Debuggers
- Memory Access/Leak Profiling
- Coverage Analysis
- Version Management

Compilers

- The compiler is of primary importance in coding
  - A good optimising compiler will know about the underlying hardware and how best to re-write your code for optimal efficiency
- Know what your compiler can do
  - Typically can have very large and complex set of flags/switches to fine-tune behaviour
  - READ THE MAN PAGE! On my old Alpha ‘man f90’ gave ~4000 lines of terse description of the command, including 93 principal switches and 493 sub-options!
  - gfortran is sneaky – only 979 lines and 291 options, but there are many more lurking in the gcc & ld manpages!
  - Check carefully what is the default behaviour – varies between compilers!

Compiler Flags

- Of course, not all compilers or languages support all flags on all platforms, but some that are very common and useful [syntax may vary] include:

  f90 –g –C –std08 –O0 myprog.f90

  ⇒(-g) include extra debugging info in binary

  ⇒(-C) turn on run-time bounds checking (array limits, etc) [NOT C/C++]

  ⇒(-std08) turn on strict standard compliance checking

  ⇒(-O0) disable all optimisation

  f90 –c myprog.f90

  ⇒compile and do not link – useful if want to split source across different source files and then link them at end – good with "make" tool

  f90 –o myprog –O myprof.f90

  ⇒(-o) name output file "myprog" and not default (e.g. a.out)

  ⇒(-O) use default optimisation (typically safe code transformations that should not change any numerical results – see later lectures)

More Compiler Flags

f90 –pg myprog.f90

⇒(-pg) include extra code in binary for profiling with “gprof”

f90 –check … –show … -warn … myprog.f90

⇒(-check) enable additional runtime checking, e.g. trapping over/underflow, bounds checking, etc.

⇒(-show) enable additional compiler output to separate listing file

⇒(-warn) enable additional compiler checks, e.g. warn about argument mismatch in procedure calls, about using uninitialised variables, etc.

f90 –arch … myprog.f90

⇒(-arch) specify which version of CPU to generate code for – generic, host, or a particular generation. Use additional instructions found only in later CPUs and hence not run (without emulation) on older ones.

Pre-processor

Conditional compilation can be useful.

```
program testpp
  implicit none
#ifdef debug
  write(*,*)'Code was compiled
with -Ddebug'
#endif
end program testpp
```

N.B. to invoke pre-processor Fortran extension must be .F90 not .f90

Compile with e.g.

```
f90 -Ddebug mycode.F90

cc -Ddebug mycode.c
```

```
#include <stdio.h>
void main() {
#ifdef debug
  printf("This code was compiled with -debug\n");
#endif
}
```

Example of Pre-Processor Usage

```
#ifdef BLAS

  ! C = A X B using BLAS where A=MxK & B=KxN
  call
dgemm('N','N',m,n,k,1.0_dp,A,m,B,k,0.0_dp,C,m)

#endif

#ifndef BLAS

  ! C = A X B using F90 intrinsic MATMUL
  C = MATMUL(A,B)

#endif
```

Choosing a Compiler

- What if there is more than one compiler available? How do you choose which is best?
  - Benchmark standard codes
  - Library support
  - Adherence to language definition standards
  - Bug detection proficiency?
  - Cost?
- See e.g. http://www.polyhedron.com for some useful Fortran compiler comparison tables
- Intel compiler generates fast code on Intel hardware but crippled (deliberately) on AMD. AOCC?
- GNU compilers produce good code everywhere but not so hardware specific.
- Portland Group good for GPU programming
- NAG best for standards compliance and testing

Automating Compilation

- Use `make` (or `gmake`)
  - Very common and useful utility for automating a sequence of tasks with varying dependencies
- Very useful with multi-file projects
  - Good idea to split a large, complex source file into smaller, simpler ones, e.g. one module/file
  - A `Makefile` is a list of *targets* and *dependencies*
    - To build executable need to compile all source files in some appropriate order
    - But if edit one source file, should only need to recompile that one and any others that depend upon it
    - Hence big time saver for large projects
    - Can use for other tasks too, e.g. creating documentation from LaTeX source files – `make thesis` – etc.

```
[mijp1@ludwig tmp]$ make
g95 -O2 -c -o global.o global.f90
g95 -O2 -c -o input.o input.f90
g95 -O2 -c -o set.o set.f90
g95 -O2 -c -o initial.o initial.f90
g95 -O2 -c -o timer.o timer.f90
g95 -O2 -c -o tridia.o tridia.f90
g95 -O2 -c -o gamma.o gamma.f90
g95 -O2 -c -o output.o output.f90
ar r libdiffuse.a global.o input.o set.o initial.o timer.o tridia.o gamma.o output.o
rm global.o input.o set.o initial.o timer.o tridia.o gamma.o output.o
g95 -O2 -c -o diffuse.o diffuse.f90
g95 -o /home/mijp1/bin/diffuse.exe diffuse.o  -L/home/mijp1/diffuse -ldiffuse /
-L/usr/local/lib -lhrgf77
```

First full compilation

```
[mijp1@ludwig tmp]$ touch output.f90
[mijp1@ludwig tmp]$ make
g95 -O2 -c -o output.o output.f90
ar r libdiffuse.a global.o input.o set.o initial.o timer.o tridia.o gamma.o output.o
g95 -o /home/mijp1/bin/diffuse.exe diffuse.o  -L/home/mijp1/tmp -ldiffuse /
-L/usr/local/lib -lhrgf77
```

‘top up’ compilation

Libraries

- One way of improving your code in terms of efficiency and functionality – in a (hopefully) error-free way – is by using libraries.
  - Simple to create your own libraries using `ar` command
  - Can link libraries to your code using compiler flags (-L and -l)
  - Can download ready-written libraries from NetLIB, etc.
  - Your O/S and/or compiler may well come with its own version of some common libraries, e.g. BLAS and LAPACK are included within the Intel MKL (Math Kernel Library) and within ACML (AMD Core Math Library), and should be vendor-optimised to the underlying hardware for maximum efficiency …

Common Free Libraries (I)

- BLAS (Basic Linear Algebra Subprograms)
  - A portable interface to various "building block" routines for performing basic vector and matrix operations. Level 1 does vector-vector operations, Level 2 does matrix-vector operations, and Level 3 does matrix-matrix operations. Often vendor optimised and used as basis for LAPACK, etc. L3 best for arithmetic intensity ...
- LAPACK (Linear Algebra Package)
  - Routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems, etc. in real and/or complex form.
  - Special versions exist for dense and banded matrices but not sparse matrices – see ScaLAPACK instead.
  - Designed for efficiency on serial, shared-memory vector and parallel processors by using block-matrix approach based upon Level 3 BLAS wherever possible.

Common Free Libraries (II)

- ATLAS (Automatically Tuned Linear Algebra Software)
  - A version of BLAS which attempts to self-tune itself for maximum performance on the given hardware upon installation (fine-tuning block-sizes for cache size, etc).
- OpenBLAS
  - Good all-round performance on most platforms
- FFTW (Fastest Fourier Transform in the West)
  - A portable FFT library that is comparable in performance to (non-portable) vendor-tuned libraries – http://www.fftw.org
- CDF (Common Data Format)
  - A library and toolkit for storing, manipulating, and accessing multi-dimensional data sets in a device independent manner – see http://cdf.gsfc.nasa.gov/ . See also NetCDF and HDF
- FoX-XML (Fortran library for XML)
  - A library to read/write/modify XML and derivatives, e.g. CML and KML – see http://www1.gly.bris.ac.uk/~walker/FoX

Commercial Libraries

- NAG (Numerical Algorithms Group)
  - A commercial library that is commonly available at many HPC centres. Not cheap. Also compilers.
  - "Spin-out" from academia – http://www.nag.co.uk
  - Contains many high-level routines and has been regularly updated since 1971 for latest algorithmic developments.
  - Includes roots of polynomials, sum of series, numerical integration and differentiation, ODE and PDE solvers, curve fitting, optimisation, linear algebra (includes LAPACK), statistics, sorting, special functions, random number generators, etc.

Static Analysis

- A way of detecting coding errors/standard extensions/anomalies by detailed examination of code without execution
  - “lint” is a standard C-code static analyser that helps remove all the fluff in your code (hence name), e.g. assignment of long variables to short, unreachable areas of code, uninitialised variables & functions, etc.
  - FortranLint is commercial equivalent
  - Linter-gfortran has just been released under MIT licence
  - “Forcheck” and “FPT” are commercial packages that does same (and more) for FORTRAN – very useful! Can also be useful when inheriting existing/legacy code – generates cross-reference tables of each program unit, the complete program and produces a call-tree.

Forcheck Output …

```
 -- file: geometry.F90

       - program unit: GEOM_FUNCS

IERR
 (file: geometry.F90, line:      97)
 **[681 I] not used
ITER
 (file: geometry.F90, line:     425)
 **[323 I] variable unreferenced
RANDOM_UNIFORM_RANDOM
 (file: geometry.F90, line:     420)
 **[674 I] procedure, program unit, or entry not referenced
PRESSURE_TOL
 (file: geometry.F90, line:     668)
 **[313 I] possibly no value assigned to this variable
 if (gg == 0.0) return
 (file: geometry.F90, line:     683)
 **[342 I] eq.or ineq. comparison of floating point data with
   zero constant
```

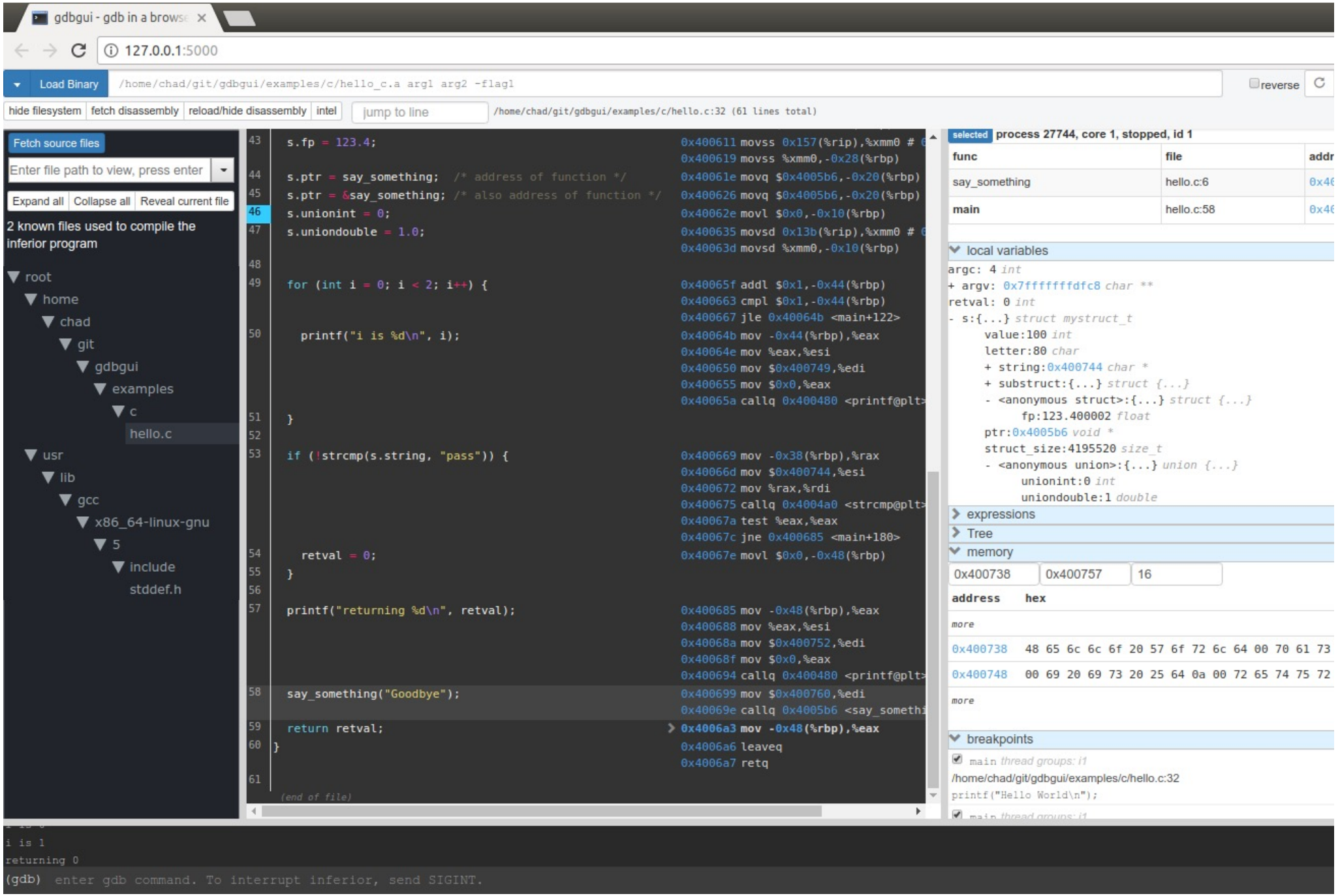
Debuggers (I)

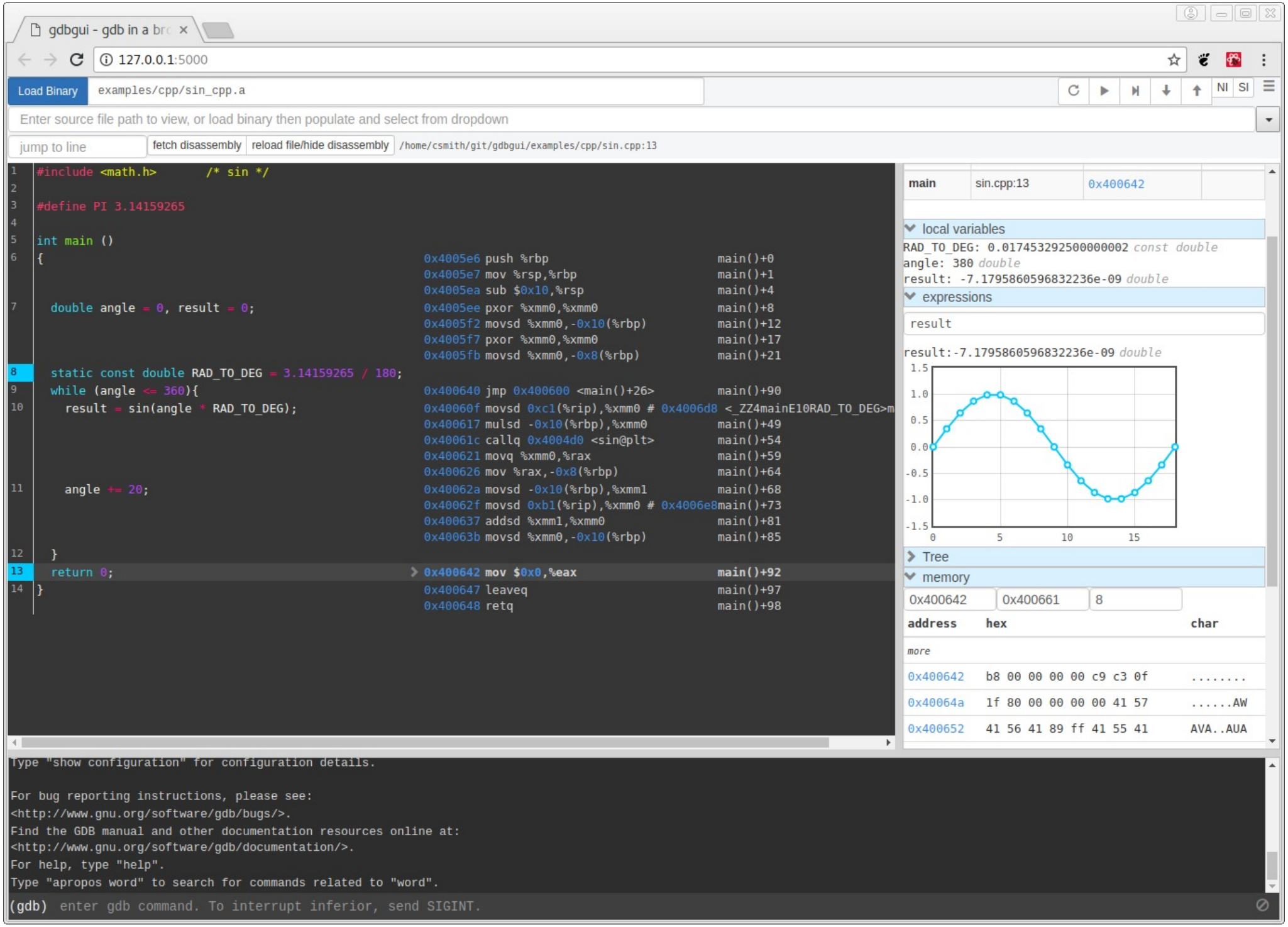
- What is a debugger?
  - A special tool used to help find bugs in your code!
- Typically allows you to:
  - Interact with code whilst executing and see source at the same time, so can
    - start program and stop it at any point in its execution by setting *breakpoints*, and then
    - examine/modify values of variables/memory/structures, and/or
    - examine state of call stack, and/or
    - set *watches* on variables/conditions, and then
    - continue execution until next interrupt
- Can also use to extract useful information from a "core dump" if program has crashed
  - e.g. find line number of crash

Debuggers (II)

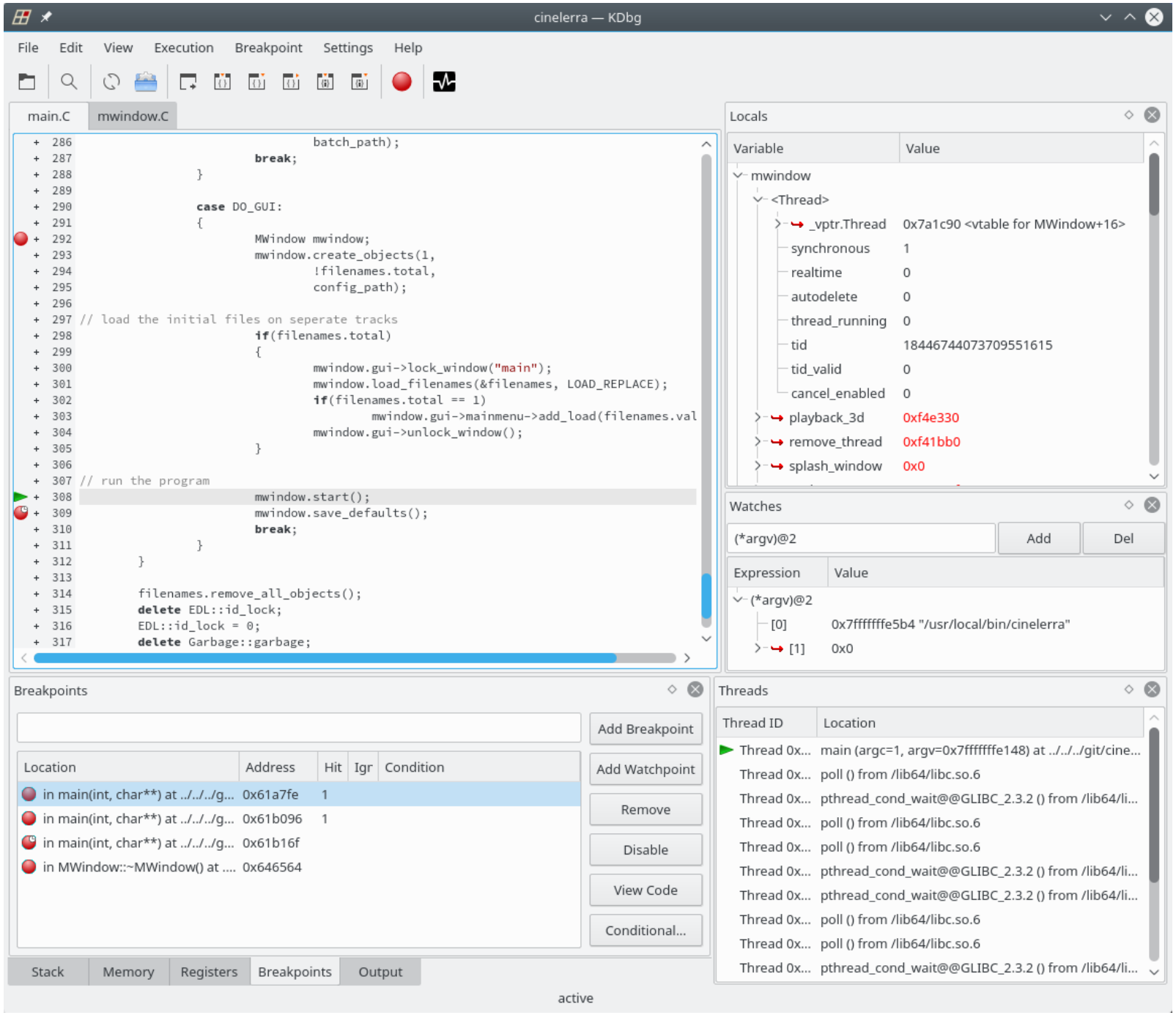
- gdb/dbx/xdb are all very similar debuggers
  - All require compiling code with ‘-g’ option
  - gdb is from GNU and will understand any supported language
    - e.g. C, C++, F77, Pascal, F90 etc.
    - Based on C so F90 arrays and derived types can seem a bit odd …
  - Graphical Front-ends are also available (e.g. kdbg)
  - gdbgui has some very nice features for spotting troublesome data – will display current value of scalar if “hover” cursor over text in source, or plot a 1D arrays
  - Also built into some editors e.g. `emacs M-x gdb`
- Some compilers come with own debugger e.g. Intel compilers come with idb debugger
- Parallel debugger (e.g. TotalView or DDT) harder!

gdbgui in action




gdbgui - gdb in a br
127.0.0.1:5000
Load Binary
examples/cpp/sin_cpp.a
NI
SI
Enter source file path to view, or load binary then populate and select from dropdown
jump to line
fetch disassembly
reload file/hide disassembly
/home/csmith/git/gdbgui/examples/cpp/sin.cpp:13
1 #include <math.h> /* sin */
2
3 #define PI 3.14159265
4
5 int main ()
6 {
7 double angle = 0, result = 0;
8 static const double RAD_TO_DEG = 3.14159265 / 180;
9 while (angle <= 360){
10 result = sin(angle * RAD_TO_DEG);
11 angle += 20;
12 }
13 return 0;
14 }
0x4005e6 push %rbp main()+0
0x4005e7 mov %rsp,%rbp main()+1
0x4005ea sub $0x10,%rsp main()+4
0x4005ee pxor %xmm0,%xmm0 main()+8
0x4005f2 movsd %xmm0,-0x10(%rbp) main()+12
0x4005f7 pxor %xmm0,%xmm0 main()+17
0x4005fb movsd %xmm0,-0x8(%rbp) main()+21
0x400640 jmp 0x400600 <main()+26> main()+90
0x40060f movsd 0xc1(%rip),%xmm0 # 0x4006d8 <_ZZ4mainE10RAD_TO_DEG>m
0x400617 mulsd -0x10(%rbp),%xmm0 main()+49
0x40061c callq 0x4004d0 <sin@plt> main()+54
0x400621 movq %xmm0,%rax main()+59
0x400626 mov %rax,-0x8(%rbp) main()+64
0x40062a movsd -0x10(%rbp),%xmm1 main()+68
0x40062f movsd 0xb1(%rip),%xmm0 # 0x4006e8main()+73
0x400637 addsd %xmm1,%xmm0 main()+81
0x40063b movsd %xmm0,-0x10(%rbp) main()+85
0x400642 mov $0x0,%eax main()+92
0x400647 leaveq main()+97
0x400648 retq main()+98
main sin.cpp:13 0x400642
local variables
RAD_TO_DEG: 0.017453292500000002 const double
angle: 380 double
result: -7.1795860596832236e-09 double
expressions
result
result:-7.1795860596832236e-09 double
1.5
1.0
0.5
0.0
-0.5
-1.0
-1.5
0
5
10
15
Tree
memory
0x400642
0x400661
8
address hex char
more
0x400642 b8 00 00 00 00 c9 c3 0f ....
0x40064a 1f 80 00 00 00 00 41 57 ......AW
0x400652 41 56 41 89 ff 41 55 41 AVA..AUA
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) enter gdb command. To interrupt inferior, send SIGINT.

kdbg in action



Backwards Debuggers

- If your code runs and then crashes it may produce a core dump
  - If it came from a ‘-g’ compiled binary then can load the core file in a debugger to find out what line/function triggered the crash
- ‘backwards debugger’ does more
  - lets you run your code backwards in time from crash, inspecting variables, etc.
  - Slow but can be invaluable!

Memory Access/Leak Profiling

- What if your code is syntactically correct but produces random crashes?
  - One possibility is improper use of memory
    - either trying to access an area of memory that is not under your control (e.g. array bounds error)
    - or which has not been properly initialised (e.g. reading an element of an array that has never been written)
- Or runs for a while and then crashes?
  - One possibility is running out of memory due to a memory leak
    - E.g. a local routine requires some temporary storage, so allocates some memory but fails to deallocate it afterwards, so second call creates another copy, etc.
- Easy to diagnose and fix using a memory access/leak profiling tool, such as “valgrind” or “google-perftools”.

Valgrind Usage

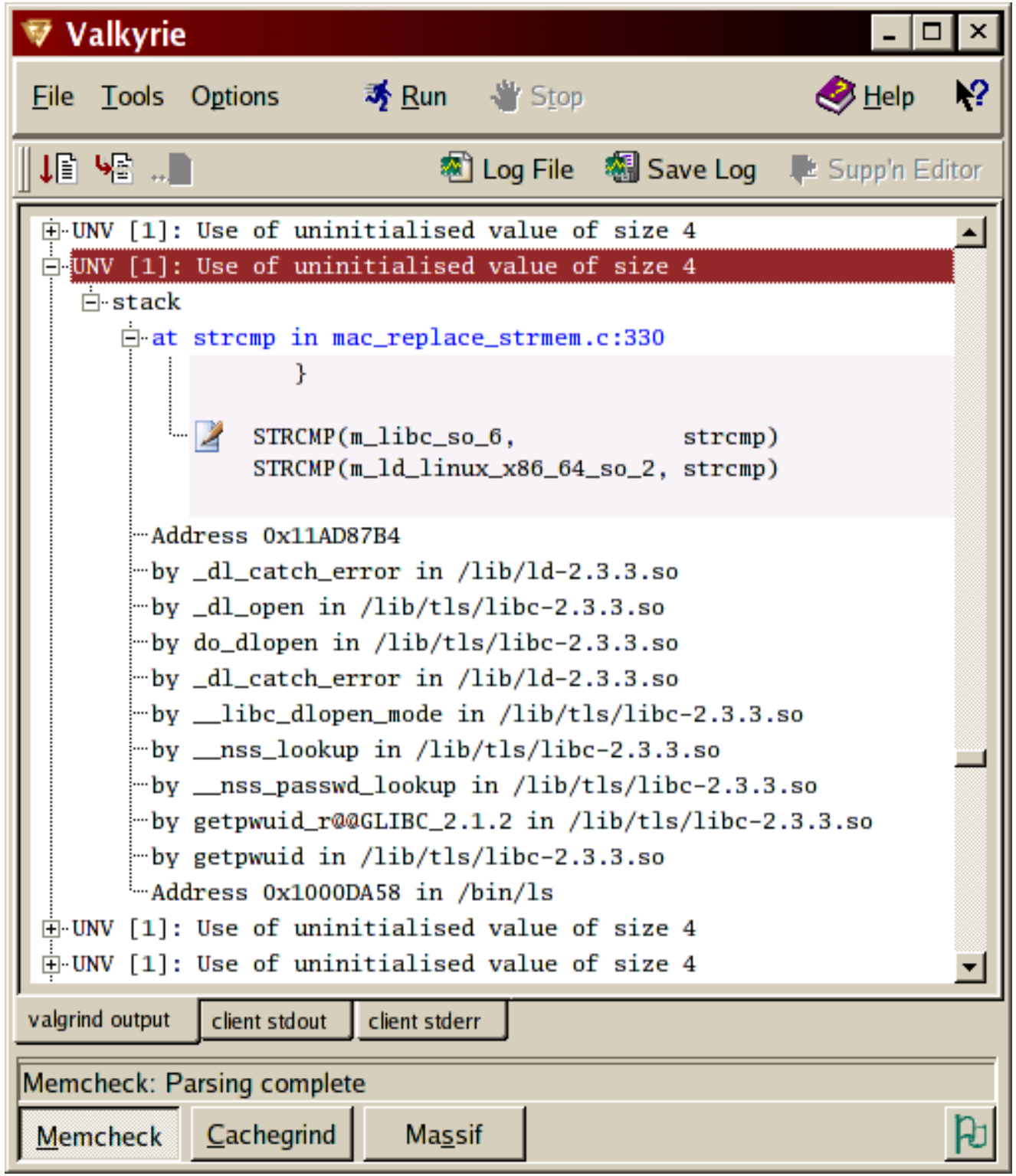
- Works with programs written in any language
  - Uses dynamic binary translation so no need to modify, recompile or relink code – even works with binaries for which you do not have the source!
  - Targeted at programs written in C and C++, because programs written in these languages tend to have the most memory bugs! But can also be used with Java, Perl, Python, Fortran, etc – alone or in any combination.
- Can combine valgrind with gdb
  - attach it to program at the point(s) where errors are detected, so that you can examine code & variables.
- Slows down program – valgrind uses a "synthetic software CPU" so slowdown might be 5 – 100 times.

Valgrind Features

- **Memcheck** detects memory-management problems by monitoring all reads/writes of memory & uses of malloc
  - Catches use of uninitialised memory, using inappropriate areas of memory, memory leaks, etc. giving code line number at which error occurred. Requires a lot of memory and slows down code by 10-30x.
- **Cachegrind** profiles cache usage to pinpoint source of cache misses etc giving line numbers, etc.
  - Slows down code by 20-100x.
- **Massif** is a heap profiler
  - Takes regular snapshots of the heap to produce graph of heap usage over time, and identifies which parts of the code are responsible for the most memory allocations. Slowdown ~ 20x
- **Used by coders working on developing new versions of Firefox, OpenOffice, KDE, GNOME …**

Valgrind GUI = Valkyrie

- Valgrind is a command line application. Various graphical frontends exist e.g. valkyrie or KCachegrind



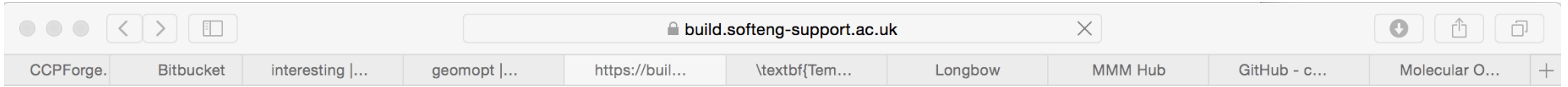
OProfile and Statistical Profiling

- gprof can slow down code by 20% by constantly interrupting flow of execution
  - Valgrind is x15 slower
- OProfile uses the hardware counters in modern CPUs to do *statistical profiling*
  - No recompilation required
  - Needs Linux kernel support
  - No runtime overhead
  - Can combine with KCachegrind for GUI
  - See https://www.mjr19.org.uk/sw/op2kcg/ for more

Coverage Analysis

- So, your code appears to be free of bugs. Is it then safe to use?
  - Depends on how well tested it is!
  - Do you know how much of the functionality of the code was exercised in each test? All of it?
  - In a complex project, it is very easy for certain branches of the code to escape testing!
- Need a "coverage analysis" tool
- To identify any "cold-spots" – lines of code that were not executed. Might indicate a bug in conditional logic or the need for tests with different inputs, etc.
  - Might also identify "hot-spots" – lines of code that have been executed many times – hence useful for profiling – see "profiling" lecture for more details.
  - Can use `gcov` with GNU or `icov` with Intel
  - Also GUI such as `lcov` or `icov` with HTML o/p

lcov



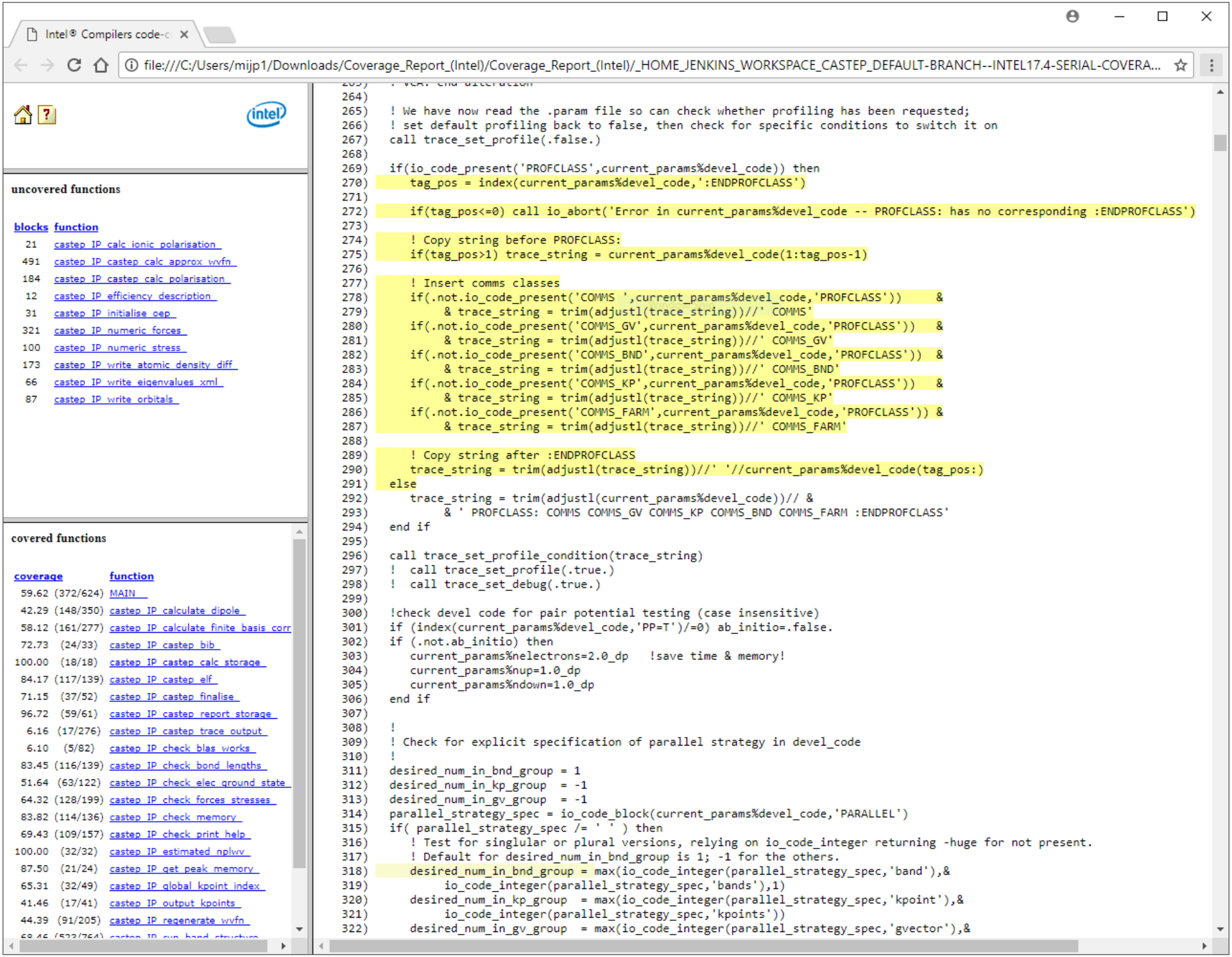
Back to default-branch--GCC-5.4-MPI-coverage | index | **Zip**

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level				
Test:	castep.info	Lines:	94624	164231	57.6 %
Date:	2017-11-01	Functions:	2008	3168	63.4 %

Directory	Line Coverage		Functions	
Source	55.1 %	1322 / 2401	74.3 %	26 / 35
Source/Functional	57.3 %	48140 / 84049	69.8 %	1024 / 1466
Source/Fundamental	62.8 %	40840 / 64992	70.3 %	748 / 1064
Source/Utility	33.9 %	4322 / 12759	34.8 %	210 / 603
obj/linux_x86_64_gfortran5.0--coverage-mpi	0.0 %	0 / 30	-	0 / 0

Generated by: LCOV version 1.10




```
264)
265)  ! We have now read the .param file so can check whether profiling has been requested;
266)  ! set default profiling back to false, then check for specific conditions to switch it on
267)  call trace_set_profile(.false.)
268)
269)  if(io_code_present('PROFCLASS',current_params%devel_code)) then
270)     tag_pos = index(current_params%devel_code,':ENDPROFCLASS')
271)
272)     if(tag_pos<=0) call io_abort('Error in current_params%devel_code -- PROFCLASS: has no corresponding :ENDPROFCLASS')
273)
274)     ! Copy string before PROFCLASS:
275)     if(tag_pos>1) trace_string = current_params%devel_code(1:tag_pos-1)
276)
277)     ! Insert comms classes
278)     if(.not.io_code_present('COMMS ',current_params%devel_code,'PROFCLASS'))     &
279)          & trace_string = trim(adjustl(trace_string))//' COMMS'
280)     if(.not.io_code_present('COMMS_GV',current_params%devel_code,'PROFCLASS'))   &
281)          & trace_string = trim(adjustl(trace_string))//' COMMS_GV'
282)     if(.not.io_code_present('COMMS_BND',current_params%devel_code,'PROFCLASS'))  &
283)          & trace_string = trim(adjustl(trace_string))//' COMMS_BND'
284)     if(.not.io_code_present('COMMS_KP',current_params%devel_code,'PROFCLASS'))   &
285)          & trace_string = trim(adjustl(trace_string))//' COMMS_KP'
286)     if(.not.io_code_present('COMMS_FARM',current_params%devel_code,'PROFCLASS')) &
287)          & trace_string = trim(adjustl(trace_string))//' COMMS_FARM'
288)
289)     ! Copy string after :ENDPROFCLASS
290)     trace_string = trim(adjustl(trace_string))//' '//current_params%devel_code(tag_pos:)
291)  else
292)     trace_string = trim(adjustl(current_params%devel_code))// &
293)          & ' PROFCLASS: COMMS COMMS_GV COMMS_KP COMMS_BND COMMS_FARM :ENDPROFCLASS'
294)  end if
295)
296)  call trace_set_profile_condition(trace_string)
297)  !  call trace_set_profile(.true.)
298)  !  call trace_set_debug(.true.)
299)
300)  !check devel code for pair potential testing (case insensitive)
301)  if (index(current_params%devel_code,'PP=T')/=0) ab_initio=.false.
302)  if (.not.ab_initio) then
303)     current_params%nelectrons=2.0_dp   !save time & memory!
304)     current_params%nup=1.0_dp
305)     current_params%ndown=1.0_dp
306)  end if
307)
308)  !
309)  ! Check for explicit specification of parallel strategy in devel_code
310)  !
311)  desired_num_in_bnd_group = 1
312)  desired_num_in_kp_group  = -1
313)  desired_num_in_gv_group  = -1
314)  parallel_strategy_spec = io_code_block(current_params%devel_code,'PARALLEL')
315)  if( parallel_strategy_spec /= ' ' ) then
316)     ! Test for singlular or plural versions, relying on io_code_integer returning -huge for not present.
317)     ! Default for desired_num_in_bnd_group is 1; -1 for the others.
318)     desired_num_in_bnd_group = max(io_code_integer(parallel_strategy_spec,'band'),&
319)          io_code_integer(parallel_strategy_spec,'bands'),1)
320)     desired_num_in_kp_group  = max(io_code_integer(parallel_strategy_spec,'kpoint'),&
321)          io_code_integer(parallel_strategy_spec,'kpoints'))
322)     desired_num_in_gv_group  = max(io_code_integer(parallel_strategy_spec,'gvector'),&
```

Version Management

- So, code appears to be free from bugs and has been thoroughly tested. What then?
  - Archive it to CD? Put it on the Web?
  - What if you want to add more features at a later date? What if you want other people to help with the coding?
- Need a version management tool!
  - Git is a powerful (unfriendly) command-line tool – developed for managing the Linux kernel
  - Github is an easier (!) way to use git
  - Other popular VCS include mercurial (used by mozilla) and subversion (used by apache)
  - Also BitBucket as web client for mercurial, etc.

Version Management Features

- Record the history of your source files.
  - The first version of each file is stored completely and then for subsequent revisions only the differences are stored. Saves a lot of disk space!
  - Can easily retrieve any stored version of each file at any later time, e.g. if a modification introduces a bug which is not spotted immediately, can backtrack through different versions, repeating the tests to see when the bug appeared, and hence find the offending code & fix the bug.
- Eases group working – multiple people on one project
  - What if you find and fix a bug, but someone else also finds and fixes the same bug but in a different way? Which version of the code should be used? What if two people want to modify the same file at the same time? Need to have access control and/or locks – say who is allowed to make changes – and a mechanism for merging differences.

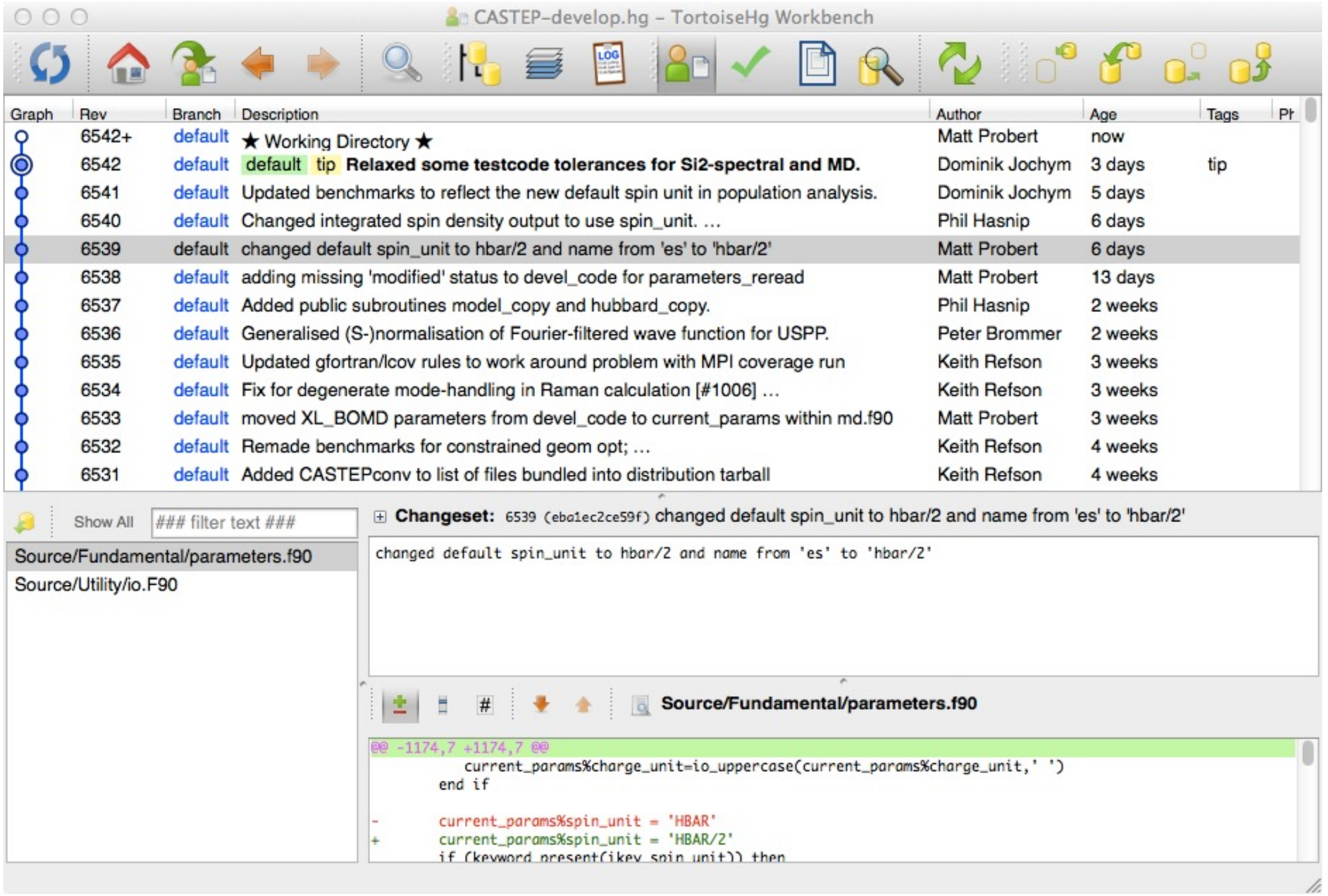
Mercurial Features

- **`hg`** does all of the above and more
  - Can use as a simple way of archiving project as it develops: create a repository, add files, make changes, commit changes (or revert), browse old versions, etc.
  - Allows multiple users to work with a central repository and clone a local copy to work from, and decide when to pull/push changes to main repository
  - Built in facilities for merging changes and managing conflicting modifications
  - Works with change sets not individual files so better at keeping things in sync
  - Can *tag* any version to make it easy to get a release
  - Can have multiple *branches* with different developments
  - Command-line tool with many GUIs e.g. tortoiseHg

Example Mercurial

- Create a repository (hidden directory called `.hg`) in current dir: **`hg init`**
- Add all files in current directory to repository: **`hg add`**
- Save current state of all files to repository: **`hg commit`**
- Show current state of all files in repository: **`hg log`**
- Revert changes (previous state of all files) to repository: **`hg revert`**
- Check out any particular state of repository: **`hg update`**

- This is for a local repository. Can also use a remote repository.
- See '`man hg`' for more details and other options


CASTEP-develop.hg – TortoiseHg Workbench
Graph
Rev
Branch
Description
Author
Age
Tags
Ph
6542+ default ★ Working Directory ★ Matt Probert now
6542 default default tip Relaxed some testcode tolerances for Si2-spectral and MD. Dominik Jochym 3 days tip
6541 default Updated benchmarks to reflect the new default spin unit in population analysis. Dominik Jochym 5 days
6540 default Changed integrated spin density output to use spin_unit. … Phil Hasnip 6 days
6539 default changed default spin_unit to hbar/2 and name from 'es' to 'hbar/2' Matt Probert 6 days
6538 default adding missing 'modified' status to devel_code for parameters_reread Matt Probert 13 days
6537 default Added public subroutines model_copy and hubbard_copy. Phil Hasnip 2 weeks
6536 default Generalised (S-)normalisation of Fourier-filtered wave function for USPP. Peter Brommer 2 weeks
6535 default Updated gfortran/lcov rules to work around problem with MPI coverage run Keith Refson 3 weeks
6534 default Fix for degenerate mode-handling in Raman calculation [#1006] … Keith Refson 3 weeks
6533 default moved XL_BOMD parameters from devel_code to current_params within md.f90 Matt Probert 3 weeks
6532 default Remade benchmarks for constrained geom opt; … Keith Refson 4 weeks
6531 default Added CASTEPconv to list of files bundled into distribution tarball Keith Refson 4 weeks
Show All
filter text
Source/Fundamental/parameters.f90
Source/Utility/io.F90
Changeset: 6539 (eba1ec2ce59f) changed default spin_unit to hbar/2 and name from 'es' to 'hbar/2'
changed default spin_unit to hbar/2 and name from 'es' to 'hbar/2'
Source/Fundamental/parameters.f90
@@ -1174,7 +1174,7 @@
current_params%charge_unit=io_uppercase(current_params%charge_unit,' ')
end if
- current_params%spin_unit = 'HBAR'
+ current_params%spin_unit = 'HBAR/2'
if (keyword_present(ikey_spin_unit)) then

Generic Resources

- GNU for gcc/gfortran/gmake/emacs/gdb etc at http://www.gnu.org
- NETLIB for many excellent libraries (e.g. BLAS, LAPACK, ATLAS, etc) at http://www.netlib.org
- Valgrind – a free debugger/profiler/leak detector for x86 Linux at http://valgrind.org
- Intel free software (Fortran, C++, MKL, Vtune, etc) at https://software.intel.com/en-us/qualify-for-free-software
- AMD compiler suite (including Clang & Flang) at https://developer.amd.com/amd-aocc/
- Mercurial homepage is at https://mercurial-scm.org
- EPSRC-supported initiatives include

  https://www.software.ac.uk

  http://society-rse.org

FORTRAN Specific Resources

- See PGI for CUDA and OpenACC support at https://developer.nvidia.com/openacc-toolkit
- Linux Fortran Information page at http://www.nikhef.nl/~templon/fortran.html
- FPT is at http://www.simconglobal.com
- FORCHECK is at https://codework.com/solutions/developer-tools/forcheck-fortran-analysis/
- Commercial Fortran products and support at http://www.polyhedron.com
- Fortran compiler list – lot more than just gfortran – at https://fortran-lang.org/en/compilers/