The University Of Sheffield.

Taking up development of existing research codes

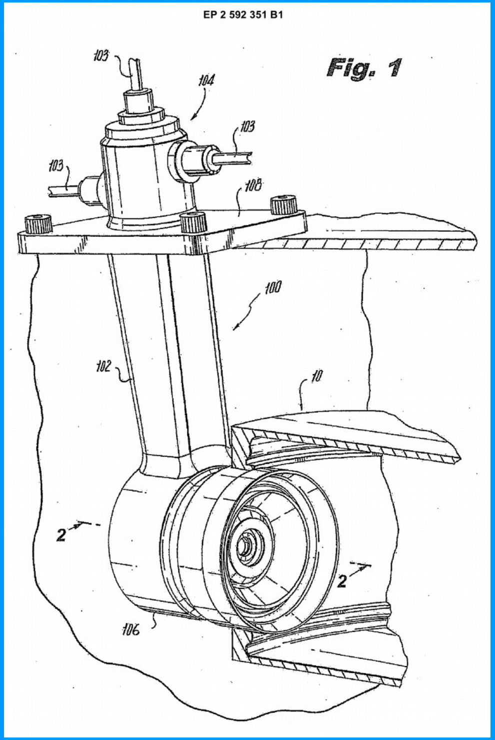
Dr. Spiridon Siouris
s.siouris@sheffield.ac.uk
Low Carbon Combustion Centre (LCCC), Mechanical Engineering

A WORLD TOP 100 UNIVERSITY

Contents

My background

- Research fellow in fluids (Mech.Eng.)
- Worked on several projects in Low Carbon Combustion Centre and Aerodynamics groups such as:
  - Modelling of fuel deposition in gas turbine injectors (FINCAP)
  - Modelling of fuel and lubricant thermo-chemical degradation in gas turbine systems
  - Aerodynamics, active flow control and plasma modelling



My background

- No formal CS education, but I like:
  - Computers/Programming/HPC
  - Tidy and efficient things
  - Adhering to standards
- Lots of experience in software development for modelling fluids and/or chemical reacting flows over many years

Software in academia

- Great opportunities for developing very interesting software, but…
- Projects often start late
- Results driven – the quicker the better and no questions asked!
- Often written by researchers where programming is not part of their formal education or primary focus
- PI’s usually not involved at programming level to provide input

Software in academia

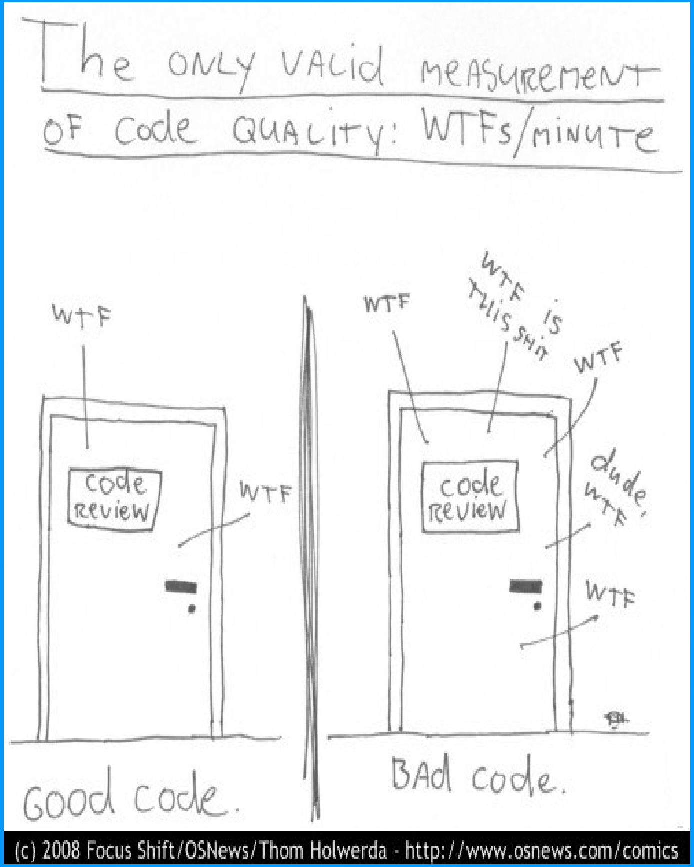
- Aim to generate results often leading to quick and dirty programming!
- Software generated without adhering to good programming practises
- Code is passed on from one researcher to another
- Elements are added on and on without any restructuring of code

Software in academia

- Lots of software generated with high impact, but:
  - Time is lost figuring out what various things do in the software
  - Can end up with several versions of the same software but very little information, if lucky!
  - Not written in the best way…
- Principle of least astonishment https://en.wikipedia.org/wiki/Principle_of_least_astonishment

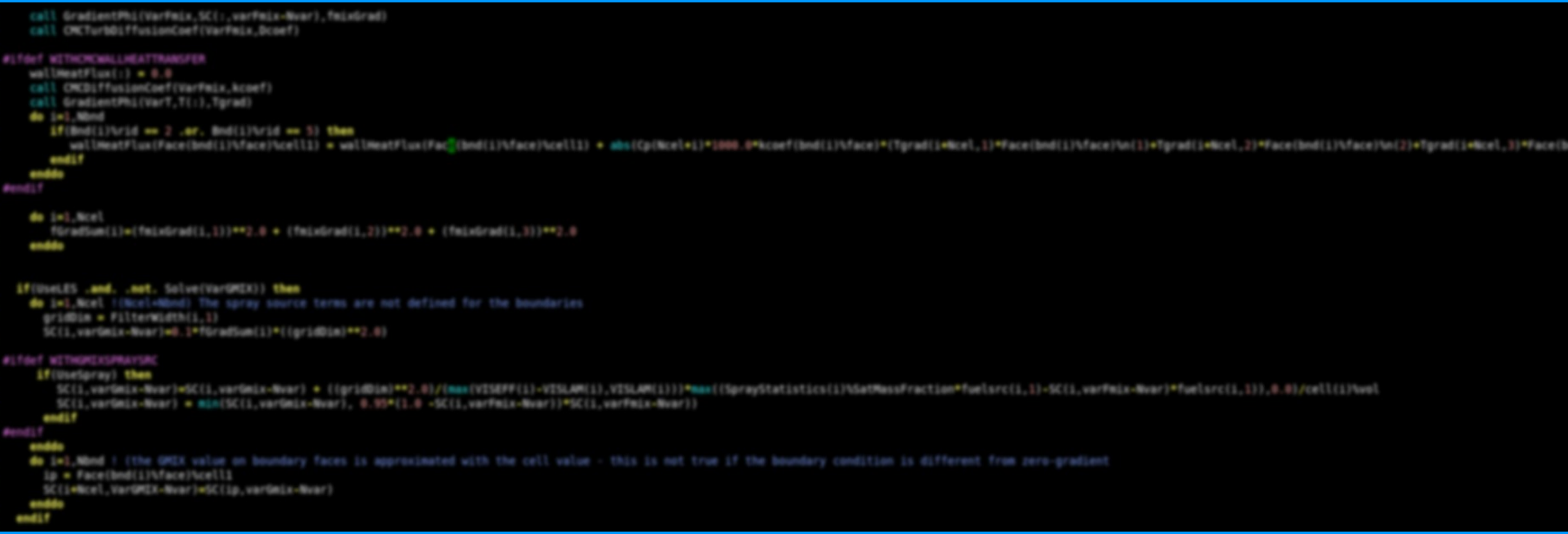
Software in academia

- Lots of software generated with high impact, but:
  - Time is lost figuring out what various things do in the software
  - Can end up with several versions of the same software but very little information, if lucky!
  - Not written in the best way…
- Principle of least astonishment https://en.wikipedia.org/wiki/Principle_of_least_astonishment


The only valid measurement of code quality: WTFs/minute
WTF
WTF
code review
Good code.
WTF
WTF is this shit
WTF
code review
dude WTF
WTF
BAd code.
(c) 2008 Focus Shift/OSNews/Thom Holwerda - http://www.osnews.com/comics

Getting to know a project

- Look at a random source file. How is the formatting/coding? Did people take care? Is it easy to understand?



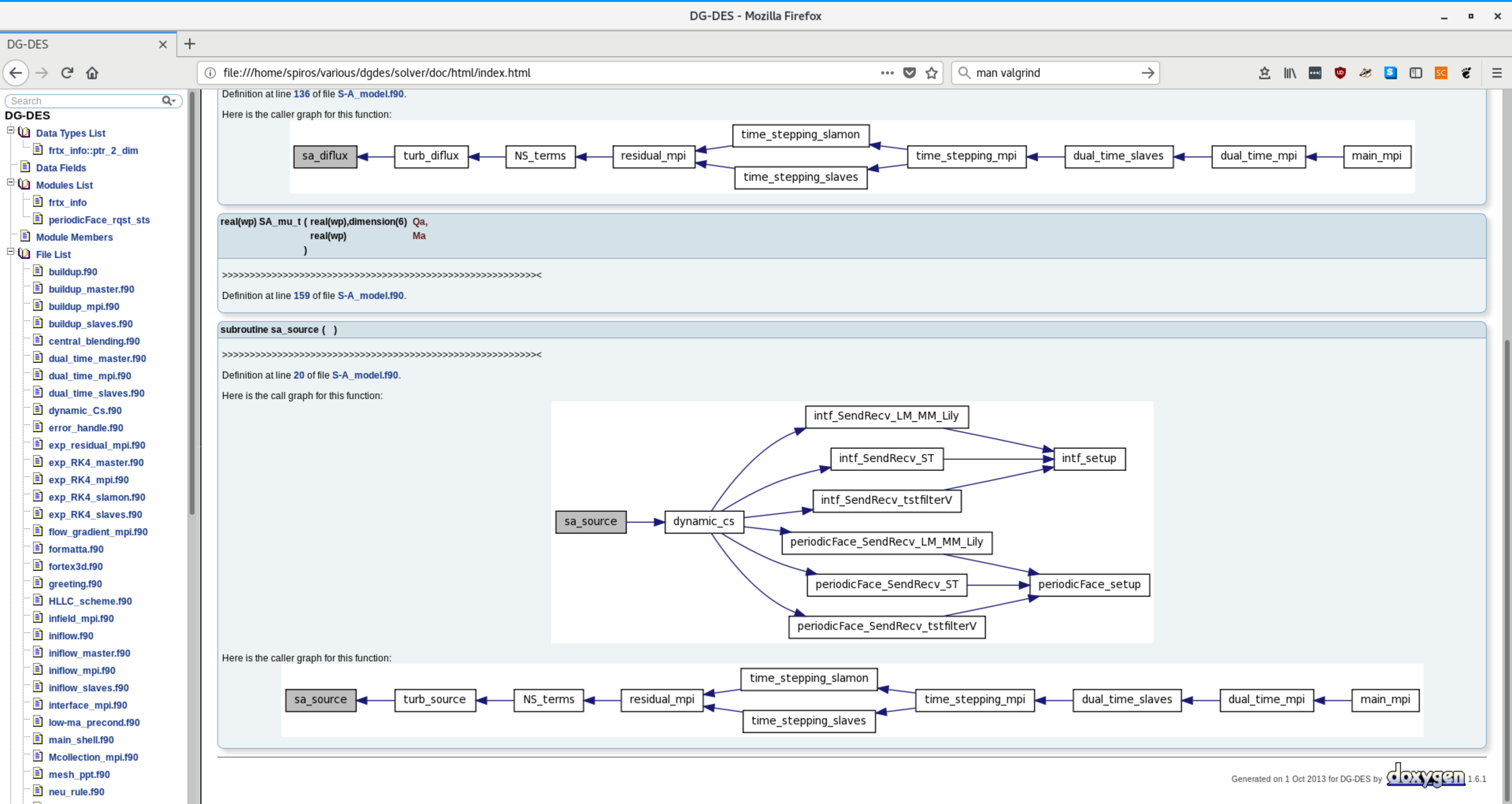
```
    call GradientPhi(VarFmix,SC(:,varFmix-Nvar),fmixGrad)
    call CMCTurbDiffusionCoef(VarFmix,Dcoef)

#ifdef WITHCMCWALLHEATTRANSFER
    wallHeatFlux(:) = 0.0
    call CMCDiffusionCoef(VarFmix,kcoef)
    call GradientPhi(VarT,T(:),Tgrad)
    do i=1,Nbnd
       if(Bnd(i)%rid == 2 .or. Bnd(i)%rid == 5) then
          wallHeatFlux(Face(bnd(i)%face)%cell1) = wallHeatFlux(Face(bnd(i)%face)%cell1) + abs(Cp(Ncel+i)*1000.0*kcoef(bnd(i)%face)*(Tgrad(i+Ncel,1)*Face(bnd(i)%face)%n(1)+Tgrad(i+Ncel,2)*Face(bnd(i)%face)%n(2)+Tgrad(i+Ncel,3)*Face(b
       endif
    enddo
#endif

    do i=1,Ncel
       fGradSum(i)=(fmixGrad(i,1))**2.0 + (fmixGrad(i,2))**2.0 + (fmixGrad(i,3))**2.0
    enddo


  if(UseLES .and. .not. Solve(VarGMIX)) then
    do i=1,Ncel !(Ncel+Nbnd) The spray source terms are not defined for the boundaries
      gridDim = FilterWidth(i,1)
      SC(i,varGmix-Nvar)=0.1*fGradSum(i)*((gridDim)**2.0)

#ifdef WITHGMIXSPRAYSRC
     if(UseSpray) then
        SC(i,varGmix-Nvar)=SC(i,varGmix-Nvar) + ((gridDim)**2.0)/(max(VISEFF(i)-VISLAM(i),VISLAM(i)))*max((SprayStatistics(i)%SatMassFraction*fuelsrc(i,1)-SC(i,varFmix-Nvar)*fuelsrc(i,1)),0.0)/cell(i)%vol
        SC(i,varGmix-Nvar) = min(SC(i,varGmix-Nvar), 0.95*(1.0 -SC(i,varFmix-Nvar))*SC(i,varFmix-Nvar))
      endif
#endif
    enddo
    do i=1,Nbnd ! (the GMIX value on boundary faces is approximated with the cell value - this is not true if the boundary condition is different from zero-gradient
      ip = Face(bnd(i)%face)%cell1
      SC(i+Ncel,VarGMIX-Nvar)=SC(ip,varGmix-Nvar)
    enddo
  endif
```

Getting to know a project

- Compile to check coding errors/warnings using:
  - `gcc -Wall -Wextra -Wpedantic`
- Run and check for memory leaks using:
  - `valgrind --leak-check=full`
- Generate documentation using:
  - `doxygen`
  - Suitable for C/C++, Fortran, bash, python, Java, TCL
  - Visualise relations between functions
  - Navigate around parts of code with hyperlinks
  - Can output HTML, pdf

The University Of Sheffield.

Getting to know a project



DG-DES - Mozilla Firefox

DG-DES

file:///home/spiros/various/dgdes/solver/doc/html/index.html

man valgrind

Search

DG-DES

- Data Types List
  - frtx_info::ptr_2_dim
- Data Fields
- Modules List
  - frtx_info
  - periodicFace_rqst_sts
- Module Members
- File List
  - buildup.f90
  - buildup_master.f90
  - buildup_mpi.f90
  - buildup_slaves.f90
  - central_blending.f90
  - dual_time_master.f90
  - dual_time_mpi.f90
  - dual_time_slaves.f90
  - dynamic_Cs.f90
  - error_handle.f90
  - exp_residual_mpi.f90
  - exp_RK4_master.f90
  - exp_RK4_mpi.f90
  - exp_RK4_slamon.f90
  - exp_RK4_slaves.f90
  - flow_gradient_mpi.f90
  - formatta.f90
  - fortex3d.f90
  - greeting.f90
  - HLLC_scheme.f90
  - infield_mpi.f90
  - iniflow.f90
  - iniflow_master.f90
  - iniflow_mpi.f90
  - iniflow_slaves.f90
  - interface_mpi.f90
  - low-ma_precond.f90
  - main_shell.f90
  - Mcollection_mpi.f90
  - mesh_ppt.f90
  - neu_rule.f90

Definition at line 136 of file S-A_model.f90.

Here is the caller graph for this function:

real(wp) SA_mu_t (real(wp),dimension(6) Qa,
real(wp) Ma
)

>>><

Definition at line 159 of file S-A_model.f90.

subroutine sa_source ( )

>>><

Definition at line 20 of file S-A_model.f90.

Here is the call graph for this function:

Here is the caller graph for this function:

Generated on 1 Oct 2013 for DG-DES by doxygen 1.6.1

Getting to know a project - summary

1. Have a look at a few source files
2. Compile with all warnings on
3. Run with memory checker
4. Generate documentation

Working on a project

1. Backup what you receive (ie projectname_date.tar.gz)
2. Set up source code version control (I like `git`)
3. Backup again
4. Ground rules. Discuss with PI or main developer the extent of your input (ie what is your freedom to change code beyond your scope?)
5. Be kind and avoid: ``Code is rubbish and needs major work''!

The University Of Sheffield.

Working on a project

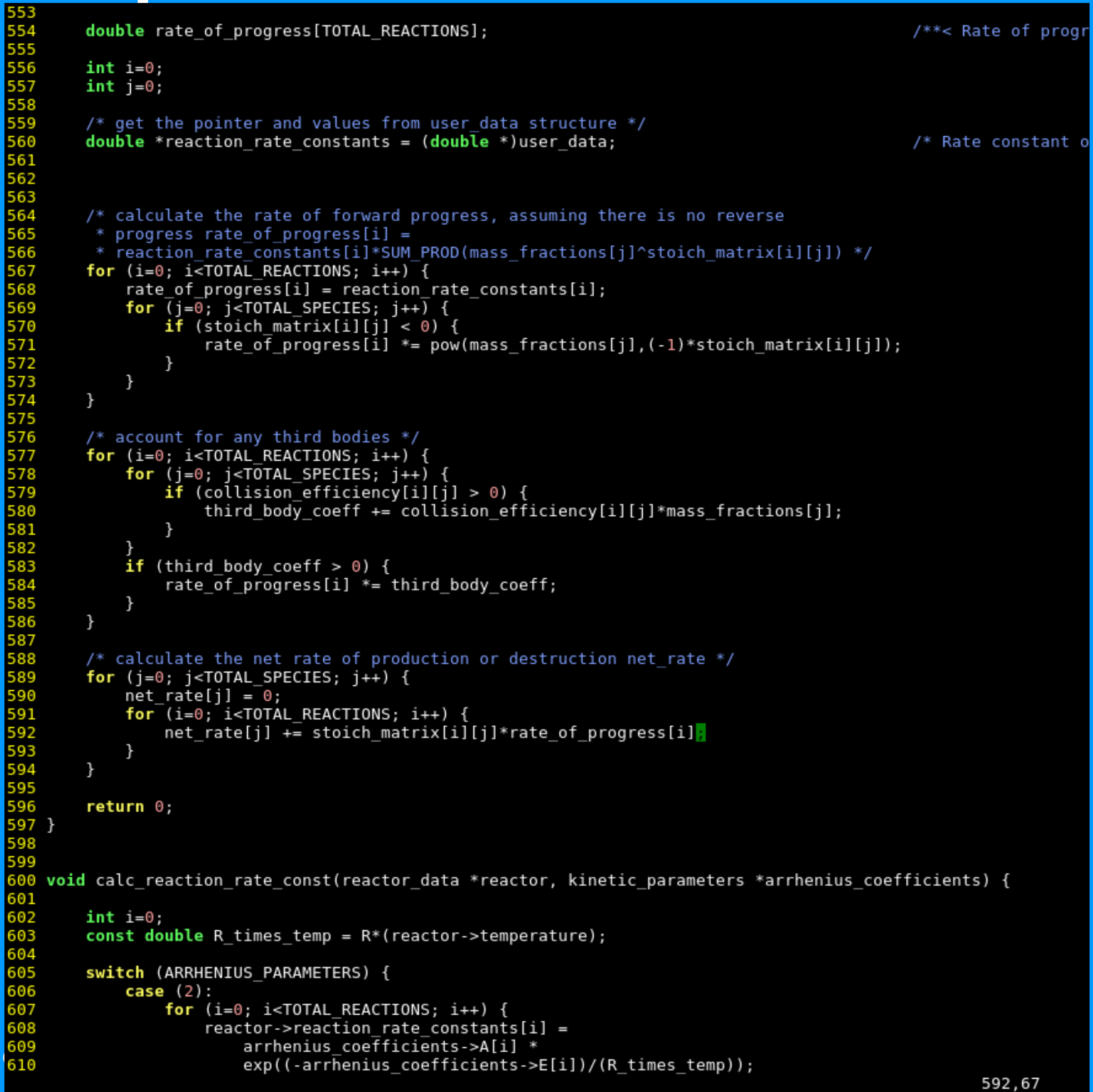
6. Carry on with your coding work
7. If you see anything that needs changing, then change it! (ie variables, functions, data structures)
8. Use GCC during development, and Intel for production
    - Intel is good for compilation and running speed, but supports non standard features – not portable!
    - GCC is good for errors and warnings, but slower than Intel (not by much though!) - portable!

A WORLD TOP 100 UNIVERSITY

Good programming practises

- Naming variables
  - Should be clear and meaningful without ambiguity (and don’t try to be funny)
  - No coded variables
  - Should not need any comments
  - Searchable, ie no single-letter variables
  - Length of name should roughly correspond to the size of its scope

Good programming practises



```
    double rate_of_progress[TOTAL_REACTIONS];                                          /**< Rate of progr

    int i=0;
    int j=0;

    /* get the pointer and values from user_data structure */
    double *reaction_rate_constants = (double *)user_data;                             /* Rate constant o



    /* calculate the rate of forward progress, assuming there is no reverse
     * progress rate_of_progress[i] =
     * reaction_rate_constants[i]*SUM_PROD(mass_fractions[j]^stoich_matrix[i][j]) */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        rate_of_progress[i] = reaction_rate_constants[i];
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (stoich_matrix[i][j] < 0) {
                rate_of_progress[i] *= pow(mass_fractions[j],(-1)*stoich_matrix[i][j]);
            }
        }
    }

    /* account for any third bodies */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (collision_efficiency[i][j] > 0) {
                third_body_coeff += collision_efficiency[i][j]*mass_fractions[j];
            }
        }
        if (third_body_coeff > 0) {
            rate_of_progress[i] *= third_body_coeff;
        }
    }

    /* calculate the net rate of production or destruction net_rate */
    for (j=0; j<TOTAL_SPECIES; j++) {
        net_rate[j] = 0;
        for (i=0; i<TOTAL_REACTIONS; i++) {
            net_rate[j] += stoich_matrix[i][j]*rate_of_progress[i];
        }
    }

    return 0;
}


void calc_reaction_rate_const(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients) {

    int i=0;
    const double R_times_temp = R*(reactor->temperature);

    switch (ARRHENIUS_PARAMETERS) {
        case (2):
            for (i=0; i<TOTAL_REACTIONS; i++) {
                reactor->reaction_rate_constants[i] =
                    arrhenius_coefficients->A[i] *
                    exp((-arrhenius_coefficients->E[i])/(R_times_temp));
                                                                                   592,67
```

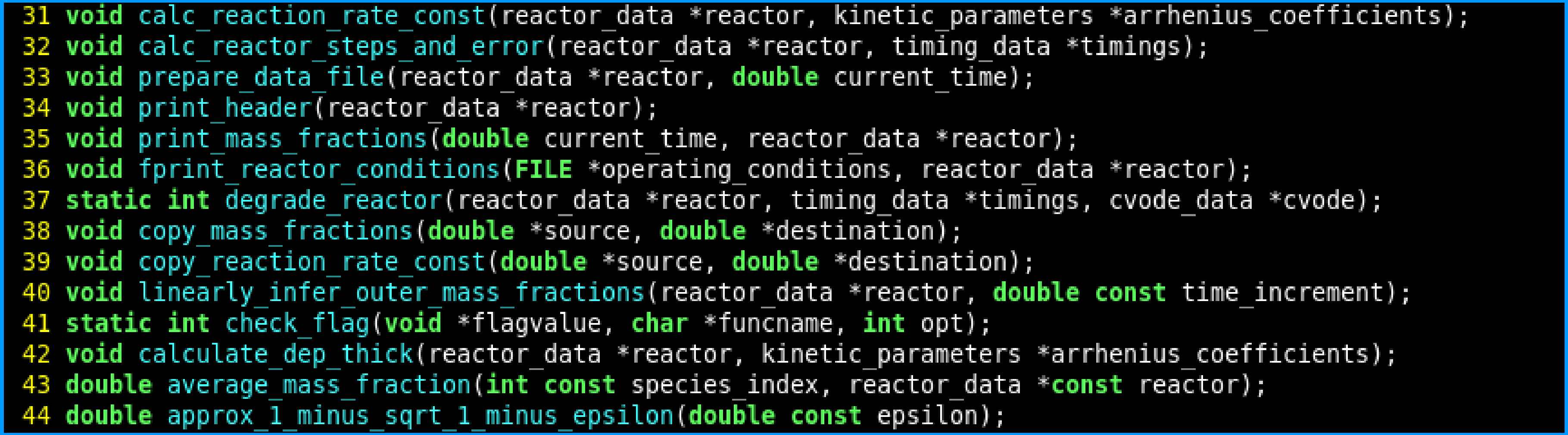
The University Of Sheffield.

Good programming practises

- Functions
  - The shorter the better
  - Two to three arguments but no more
  - Do one thing
  - No side effects
  - Error handling
  - Should always test in isolation

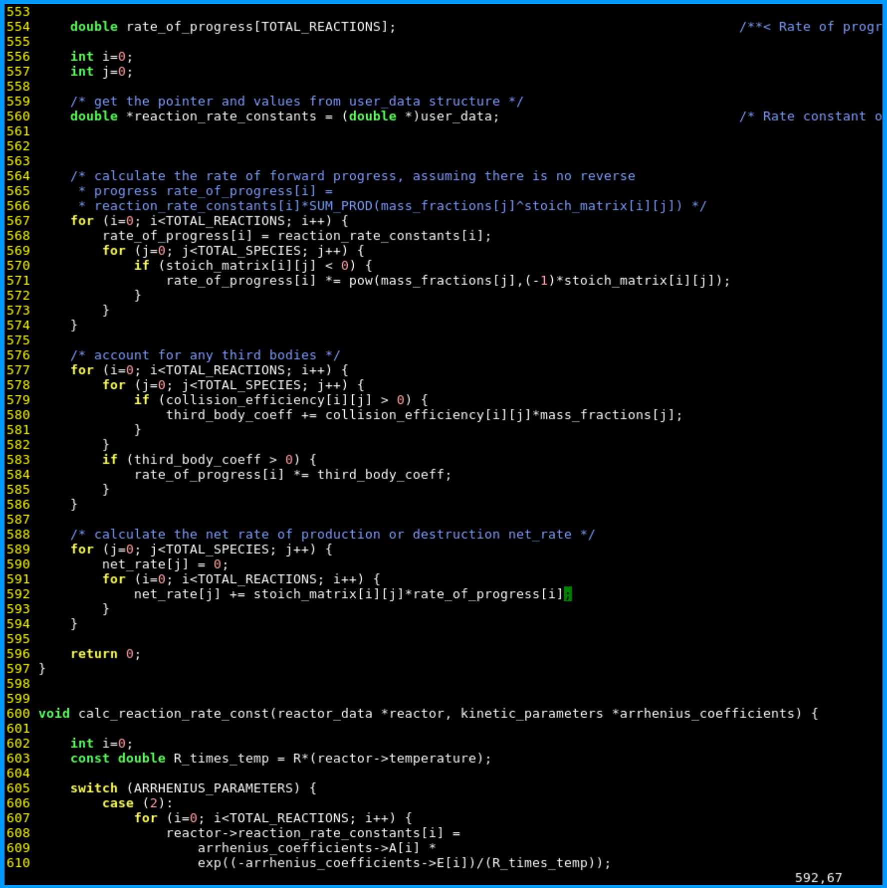
A WORLD TOP 100 UNIVERSITY

Good programming practises

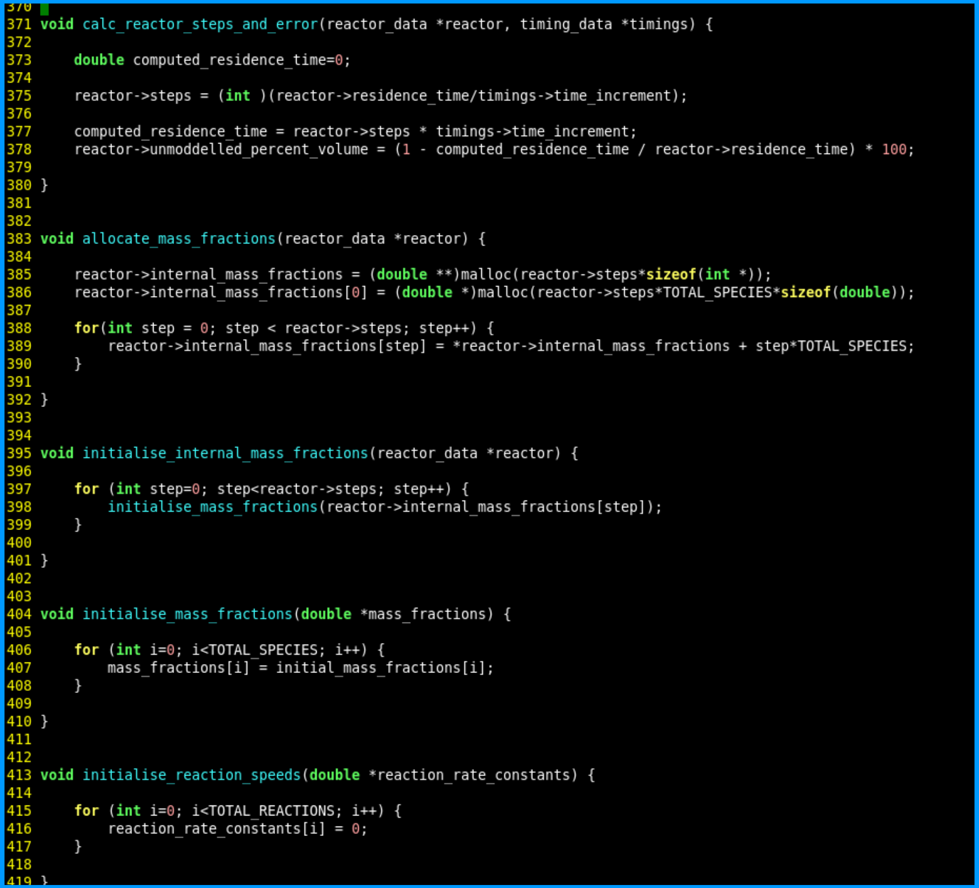


```c
void calc_reaction_rate_const(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients);
void calc_reactor_steps_and_error(reactor_data *reactor, timing_data *timings);
void prepare_data_file(reactor_data *reactor, double current_time);
void print_header(reactor_data *reactor);
void print_mass_fractions(double current_time, reactor_data *reactor);
void fprint_reactor_conditions(FILE *operating_conditions, reactor_data *reactor);
static int degrade_reactor(reactor_data *reactor, timing_data *timings, cvode_data *cvode);
void copy_mass_fractions(double *source, double *destination);
void copy_reaction_rate_const(double *source, double *destination);
void linearly_infer_outer_mass_fractions(reactor_data *reactor, double const time_increment);
static int check_flag(void *flagvalue, char *funcname, int opt);
void calculate_dep_thick(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients);
double average_mass_fraction(int const species_index, reactor_data *const reactor);
double approx_1_minus_sqrt_1_minus_epsilon(double const epsilon);
```

Good programming practises



```
    double rate_of_progress[TOTAL_REACTIONS];                                   /**< Rate of progr

    int i=0;
    int j=0;

    /* get the pointer and values from user_data structure */
    double *reaction_rate_constants = (double *)user_data;                      /* Rate constant o



    /* calculate the rate of forward progress, assuming there is no reverse
     * progress rate_of_progress[i] =
     * reaction_rate_constants[i]*SUM_PROD(mass_fractions[j]^stoich_matrix[i][j]) */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        rate_of_progress[i] = reaction_rate_constants[i];
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (stoich_matrix[i][j] < 0) {
                rate_of_progress[i] *= pow(mass_fractions[j],(-1)*stoich_matrix[i][j]);
            }
        }
    }

    /* account for any third bodies */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (collision_efficiency[i][j] > 0) {
                third_body_coeff += collision_efficiency[i][j]*mass_fractions[j];
            }
        }
        if (third_body_coeff > 0) {
            rate_of_progress[i] *= third_body_coeff;
        }
    }

    /* calculate the net rate of production or destruction net_rate */
    for (j=0; j<TOTAL_SPECIES; j++) {
        net_rate[j] = 0;
        for (i=0; i<TOTAL_REACTIONS; i++) {
            net_rate[j] += stoich_matrix[i][j]*rate_of_progress[i];
        }
    }

    return 0;
}


void calc_reaction_rate_const(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients) {

    int i=0;
    const double R_times_temp = R*(reactor->temperature);

    switch (ARRHENIUS_PARAMETERS) {
        case (2):
            for (i=0; i<TOTAL_REACTIONS; i++) {
                reactor->reaction_rate_constants[i] =
                    arrhenius_coefficients->A[i] *
                    exp((-arrhenius_coefficients->E[i])/(R_times_temp));
                                                                                  592,67
```

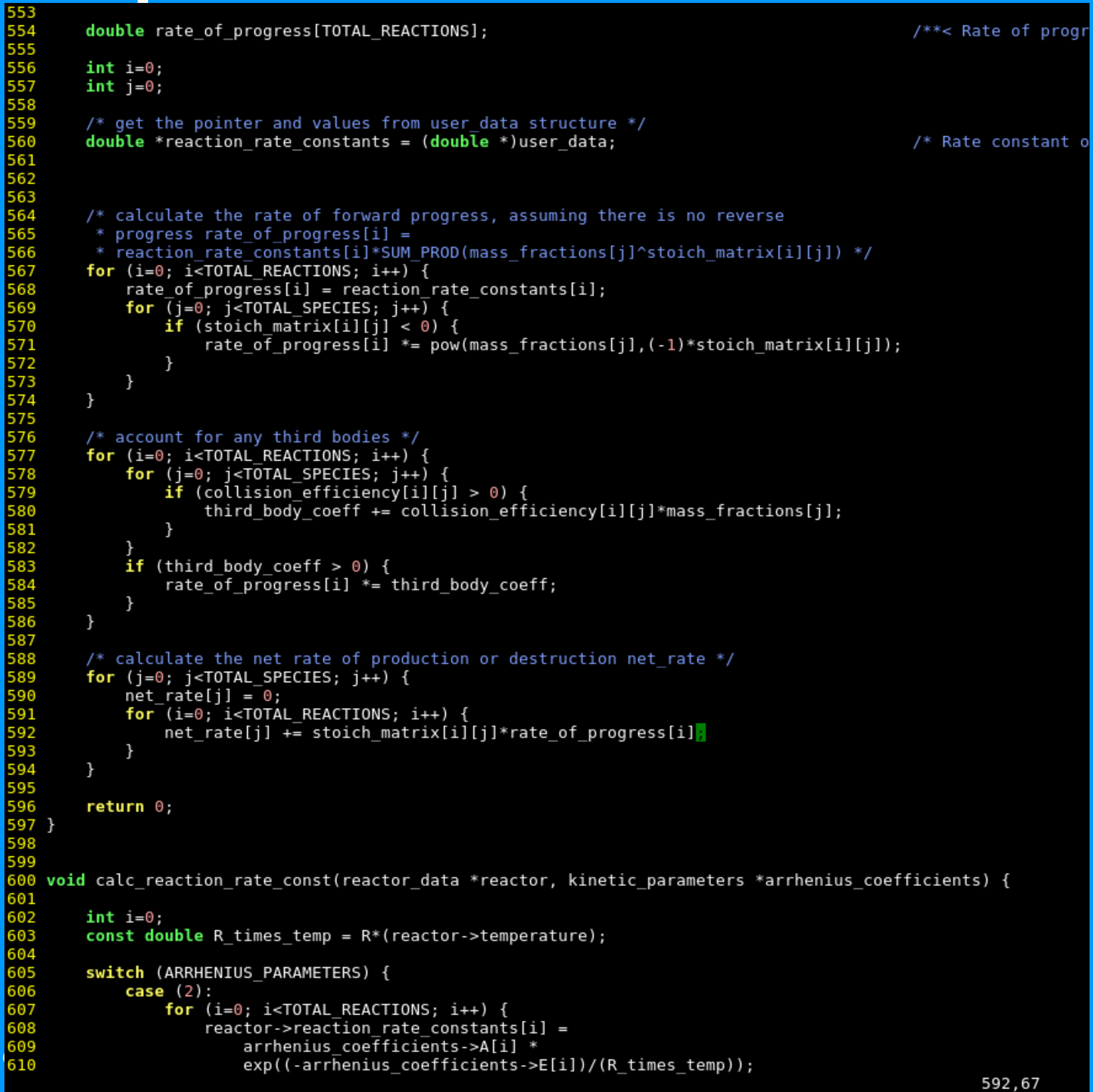


```

void calc_reactor_steps_and_error(reactor_data *reactor, timing_data *timings) {

    double computed_residence_time=0;

    reactor->steps = (int )(reactor->residence_time/timings->time_increment);

    computed_residence_time = reactor->steps * timings->time_increment;
    reactor->unmoddelled_percent_volume = (1 - computed_residence_time / reactor->residence_time) * 100;

}


void allocate_mass_fractions(reactor_data *reactor) {

    reactor->internal_mass_fractions = (double **)malloc(reactor->steps*sizeof(int *));
    reactor->internal_mass_fractions[0] = (double *)malloc(reactor->steps*TOTAL_SPECIES*sizeof(double));

    for(int step = 0; step < reactor->steps; step++) {
        reactor->internal_mass_fractions[step] = *reactor->internal_mass_fractions + step*TOTAL_SPECIES;
    }

}


void initialise_internal_mass_fractions(reactor_data *reactor) {

    for (int step=0; step<reactor->steps; step++) {
        initialise_mass_fractions(reactor->internal_mass_fractions[step]);
    }

}


void initialise_mass_fractions(double *mass_fractions) {

    for (int i=0; i<TOTAL_SPECIES; i++) {
        mass_fractions[i] = initial_mass_fractions[i];
    }

}


void initialise_reaction_speeds(double *reaction_rate_constants) {

    for (int i=0; i<TOTAL_REACTIONS; i++) {
        reaction_rate_constants[i] = 0;
    }

}
```

Good programming practises

- Commenting
  - Use with caution
  - Better to improve naming of functions, data variables than to rely on comments
  - Useful for citing articles, algorithm names, etc. where appropriate
  - Watch out for line length. Do not assume everyone has a large wide screen monitor

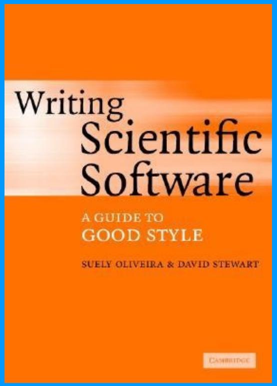
Good programming practises

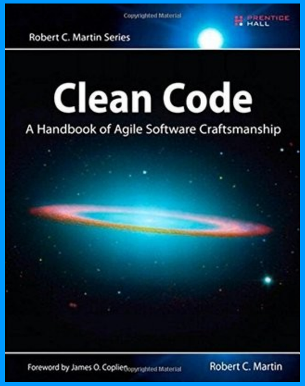


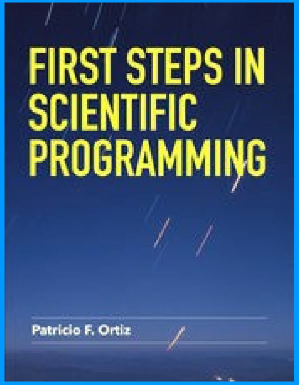
```
    double rate_of_progress[TOTAL_REACTIONS];                                           /**< Rate of progr

    int i=0;
    int j=0;

    /* get the pointer and values from user_data structure */
    double *reaction_rate_constants = (double *)user_data;                              /* Rate constant o



    /* calculate the rate of forward progress, assuming there is no reverse
     * progress rate_of_progress[i] =
     * reaction_rate_constants[i]*SUM_PROD(mass_fractions[j]^stoich_matrix[i][j]) */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        rate_of_progress[i] = reaction_rate_constants[i];
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (stoich_matrix[i][j] < 0) {
                rate_of_progress[i] *= pow(mass_fractions[j],(-1)*stoich_matrix[i][j]);
            }
        }
    }

    /* account for any third bodies */
    for (i=0; i<TOTAL_REACTIONS; i++) {
        for (j=0; j<TOTAL_SPECIES; j++) {
            if (collision_efficiency[i][j] > 0) {
                third_body_coeff += collision_efficiency[i][j]*mass_fractions[j];
            }
        }
        if (third_body_coeff > 0) {
            rate_of_progress[i] *= third_body_coeff;
        }
    }

    /* calculate the net rate of production or destruction net_rate */
    for (j=0; j<TOTAL_SPECIES; j++) {
        net_rate[j] = 0;
        for (i=0; i<TOTAL_REACTIONS; i++) {
            net_rate[j] += stoich_matrix[i][j]*rate_of_progress[i];
        }
    }

    return 0;
}


void calc_reaction_rate_const(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients) {

    int i=0;
    const double R_times_temp = R*(reactor->temperature);

    switch (ARRHENIUS_PARAMETERS) {
        case (2):
            for (i=0; i<TOTAL_REACTIONS; i++) {
                reactor->reaction_rate_constants[i] =
                    arrhenius_coefficients->A[i] *
                    exp((-arrhenius_coefficients->E[i])/(R_times_temp));
                                                                                                    592,67
```

Good programming practises

- Further reading:
  - Oliveira S., Stewart D., 2006, Writing scientific software. A guide to good style, Cambridge University press
  - Martin R. C., 2013, Clean code, A handbook of agile software craftmanship, Prentice Hall
  - Ortiz P. F., 2018, First steps in scientific programming, Amazon







Good programming practises

- Further reading
  - Ledgard H., Green R., Coding guidelines: Finding the art in the science, ACM queue, 2011
  - ASTG coding standard, Recommended coding styles for software development in Fortran, C++, Java, and Python, Ver. 1.7, 2015

Good programming practises

- Further reading for C/C++
  - NASA C style guide, August 1994, SEL-94-003
  - SEI CERT C coding standard, Rules for developing safe, reliable and secure systems in C++, 2016
  - SEI CERT C++ coding standard, Rules for developing safe, reliable and secure systems in C++, 2016
  - High integrity C++ Coding Rules, Programming Research Ltd., www.codingstandard.com
  - Joint strike fighter air vehicle C++ coding standards for the system development and demonstration program, December 2005

Good programming practises

- Further reading for Fortran
  - Fortran coding standards for new JULES code, Joint UK land environment simulator, June 2010
  - European standards for writing and documentign exchangeable Fortran 90 code, Ver. 1.1, Andrews P., 1995

Optimising code

- Very subjective and debatable topic
- Done to save time and computational resources
- Which one are you most interested in saving?
- Scrutinise every advice you receive, including mine
- Work out what is best for **you**
- Let’s start with a quote…

Optimising code

The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; premature optimization is the root of all evil (or at least most of it) in programming.

Knuth D., 1974, Computer Programming as an Art, Communications of the ACM, Vol. 17, No. 12

The University Of Sheffield.

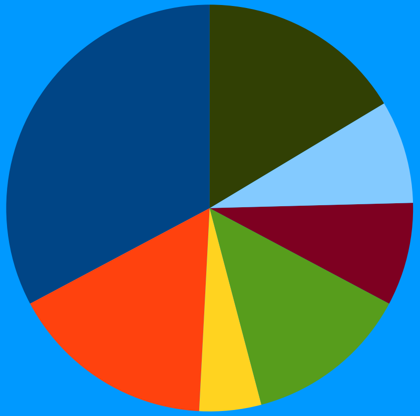
Optimising code

- Rule No. 1: Code optimisation should be the last thing you should do.
  - More important to adhere to good programming practises
  - Make sure code is as scalable as possible
  - Find something else to optimise in your workflow
- Rule No. 2: If you need to do it, use proper evidence for it
  - No data – no optimising
  - Use profilers for performance analysis, eg `gprof` for single-core, and/or `tau` for multi-core

A WORLD TOP 100 UNIVERSITY

Optimising code

- Example of a typical week of mine:



- What is taking up most of my time?
- Does it make sense for me to put effort to speed up HPC running?
- How about your workflow?

Optimising code

- Better for me to spend time on:
  - Making the code more understandable - Good programming practises!
  - Automating post processing
- Organising order of tasks is very important too as simulations can be running in the background
  - Is it possible to submit a job before a 2 hour meeting? If so, 2 hours saved!

Before optimising code

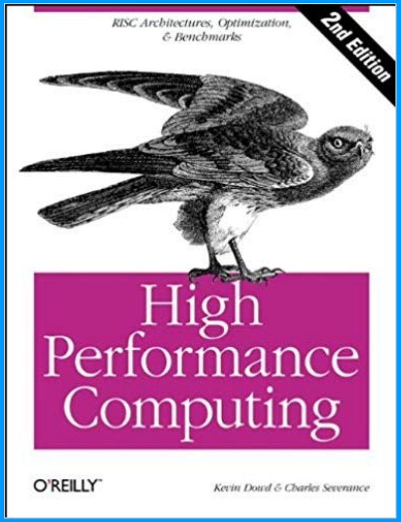
- Make sure the code is correct and bug free
- Make it portable
- Are the algorithms numerically stable/fast? What is the latest literature?
- Use libraries as much as possible as they are (most likely) already efficient – no point reinventing the wheel

Before optimising code

- Do you have a table with a design of experiments?
- Are the simulations run with the correct settings, boundary conditions, etc?
- Is the time/space discretisation suitable enough?, and do you really need a 1M cell mesh?
- After having checked all the above, then....

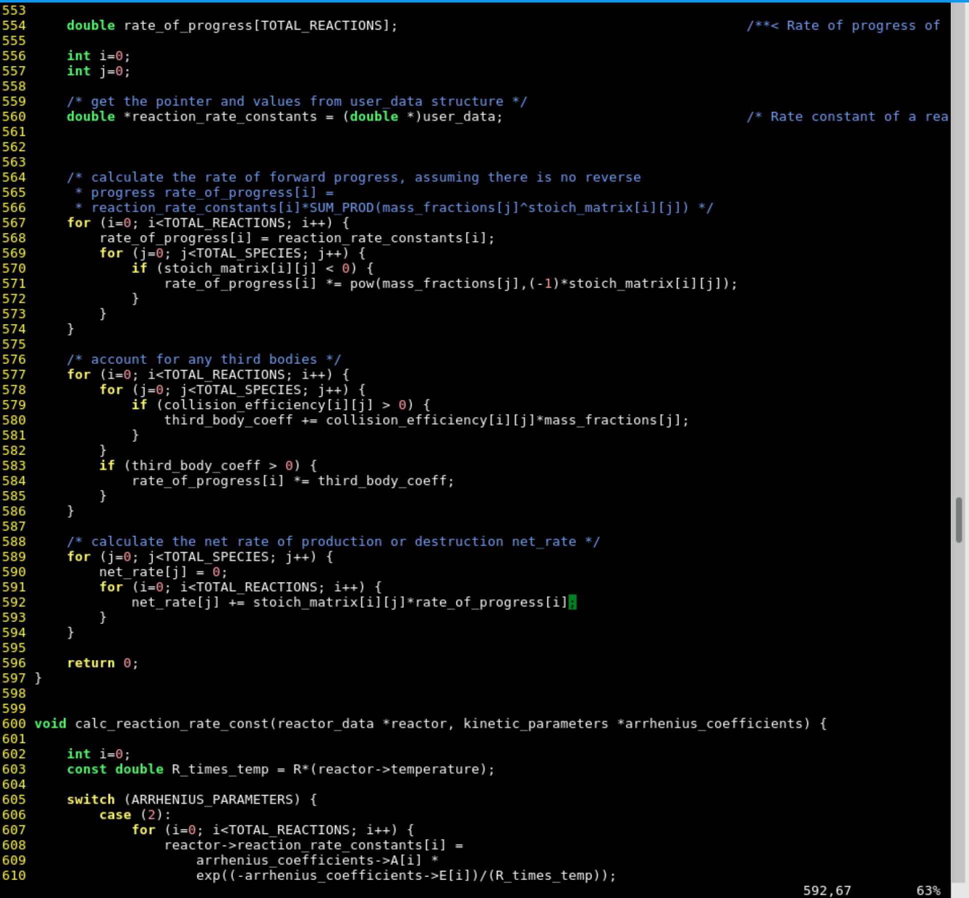
Optimising code

- Learn (briefly) how CPU’s work, eg registers, cache
- Too many techniques to list here
- Very good read (old, but still useful)
  - Severance C. and Dowd K., High performance computing (RISC architectures, optimization and benchmarks), 2nd ed., O’Reilly, 1998
  - Or even look at anything from the early days of game programming, LAPACK, BLAS, etc where every kB of memory and CPU cycles matters!
- Plenty other references exist that are more recent, but the core knowledge remains the same

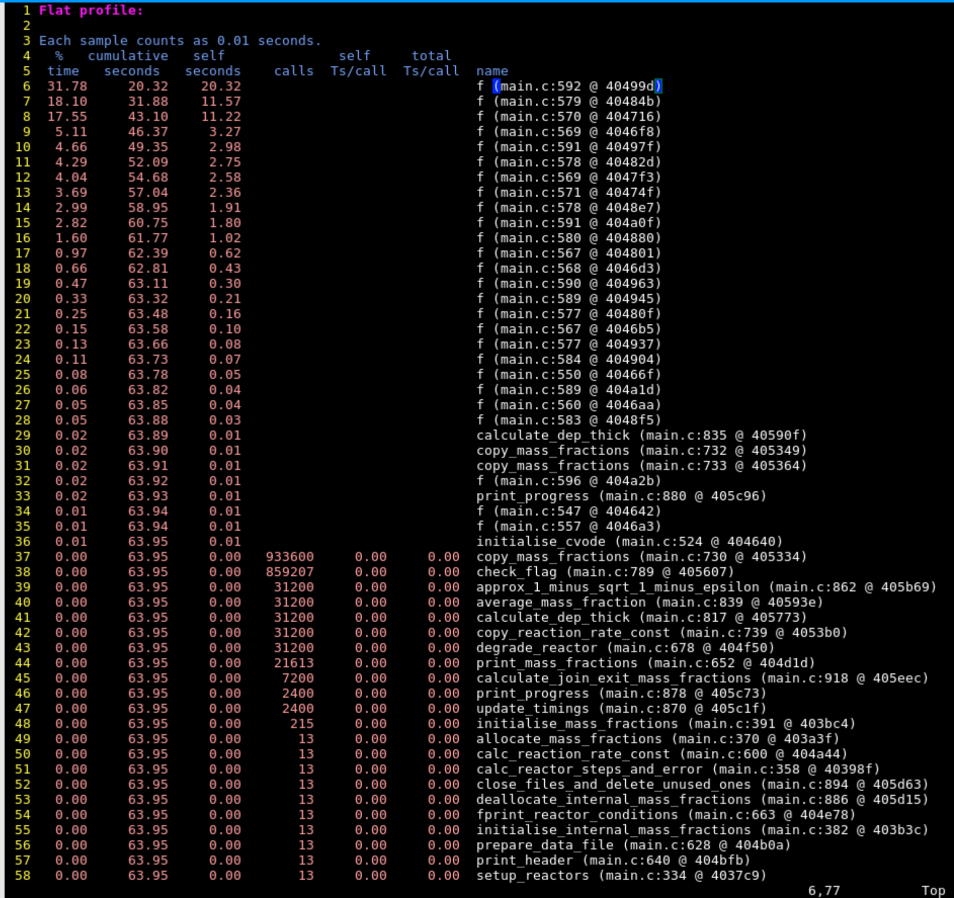

RISC Architectures, Optimization, & Benchmarks
2nd Edition
High Performance Computing
O'REILLY
Kevin Dowd & Charles Severance

Optimising code

- Profile first to generate data and then optimise the most time consuming parts of the code

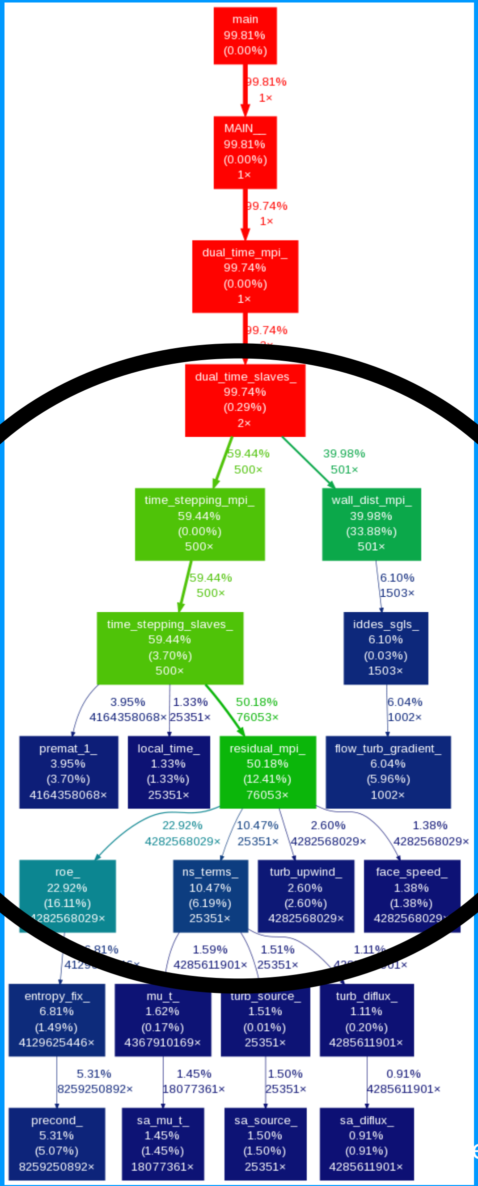


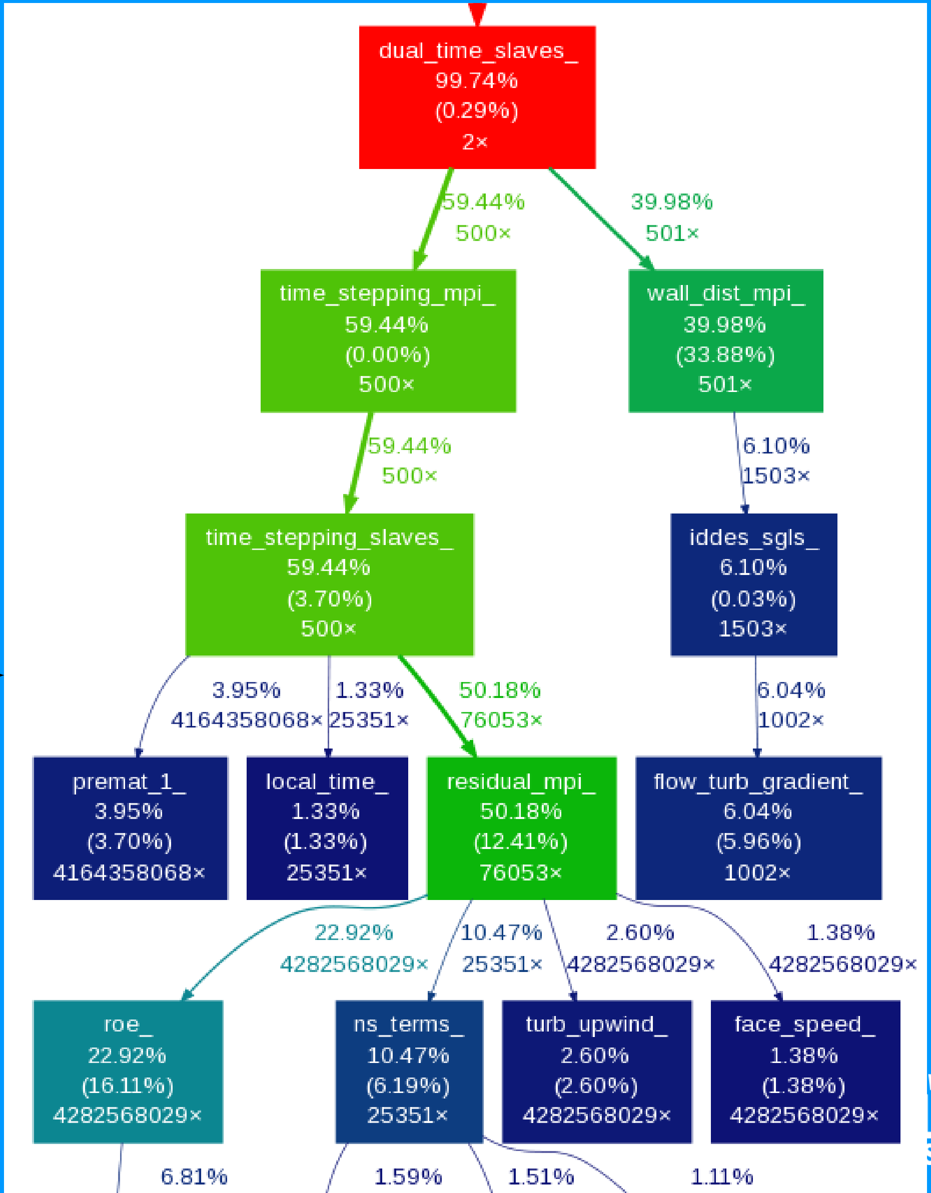
```
553
554     double rate_of_progress[TOTAL_REACTIONS];                                       /**< Rate of progress of
555
556     int i=0;
557     int j=0;
558
559     /* get the pointer and values from user_data structure */
560     double *reaction_rate_constants = (double *)user_data;                          /* Rate constant of a rea
561
562
563
564     /* calculate the rate of forward progress, assuming there is no reverse
565      * progress rate_of_progress[i] =
566      * reaction_rate_constants[i]*SUM_PROD(mass_fractions[j]^stoich_matrix[i][j]) */
567     for (i=0; i<TOTAL_REACTIONS; i++) {
568         rate_of_progress[i] = reaction_rate_constants[i];
569         for (j=0; j<TOTAL_SPECIES; j++) {
570             if (stoich_matrix[i][j] < 0) {
571                 rate_of_progress[i] *= pow(mass_fractions[j],(-1)*stoich_matrix[i][j]);
572             }
573         }
574     }
575
576     /* account for any third bodies */
577     for (i=0; i<TOTAL_REACTIONS; i++) {
578         for (j=0; j<TOTAL_SPECIES; j++) {
579             if (collision_efficiency[i][j] > 0) {
580                 third_body_coeff += collision_efficiency[i][j]*mass_fractions[j];
581             }
582         }
583         if (third_body_coeff > 0) {
584             rate_of_progress[i] *= third_body_coeff;
585         }
586     }
587
588     /* calculate the net rate of production or destruction net_rate */
589     for (j=0; j<TOTAL_SPECIES; j++) {
590         net_rate[j] = 0;
591         for (i=0; i<TOTAL_REACTIONS; i++) {
592             net_rate[j] += stoich_matrix[i][j]*rate_of_progress[i];
593         }
594     }
595
596     return 0;
597 }
598
599
600 void calc_reaction_rate_const(reactor_data *reactor, kinetic_parameters *arrhenius_coefficients) {
601
602     int i=0;
603     const double R_times_temp = R*(reactor->temperature);
604
605     switch (ARRHENIUS_PARAMETERS) {
606         case (2):
607             for (i=0; i<TOTAL_REACTIONS; i++) {
608                 reactor->reaction_rate_constants[i] =
609                     arrhenius_coefficients->A[i] *
610                     exp((-arrhenius_coefficients->E[i])/(R_times_temp));
                                                                                            592,67        63%
```



```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 31.78     20.32    20.32                             f (main.c:592 @ 40499d)
 18.10     31.88    11.57                             f (main.c:579 @ 40484b)
 17.55     43.10    11.22                             f (main.c:570 @ 404716)
  5.11     46.37     3.27                             f (main.c:569 @ 4046f8)
  4.66     49.35     2.98                             f (main.c:591 @ 40497f)
  4.29     52.09     2.75                             f (main.c:578 @ 40482d)
  4.04     54.68     2.58                             f (main.c:569 @ 4047f3)
  3.69     57.04     2.36                             f (main.c:571 @ 40474f)
  2.99     58.95     1.91                             f (main.c:578 @ 4048e7)
  2.82     60.75     1.80                             f (main.c:591 @ 404a0f)
  1.60     61.77     1.02                             f (main.c:580 @ 404880)
  0.97     62.39     0.62                             f (main.c:567 @ 404801)
  0.66     62.81     0.43                             f (main.c:568 @ 4046d3)
  0.47     63.11     0.30                             f (main.c:590 @ 404963)
  0.33     63.32     0.21                             f (main.c:589 @ 404945)
  0.25     63.48     0.16                             f (main.c:577 @ 40480f)
  0.15     63.58     0.10                             f (main.c:567 @ 4046b5)
  0.13     63.66     0.08                             f (main.c:577 @ 404937)
  0.11     63.73     0.07                             f (main.c:584 @ 404904)
  0.08     63.78     0.05                             f (main.c:550 @ 40466f)
  0.06     63.82     0.04                             f (main.c:589 @ 404a1d)
  0.05     63.85     0.04                             f (main.c:560 @ 4046aa)
  0.05     63.88     0.03                             f (main.c:583 @ 4048f5)
  0.02     63.89     0.01                             calculate_dep_thick (main.c:835 @ 40590f)
  0.02     63.90     0.01                             copy_mass_fractions (main.c:732 @ 405349)
  0.02     63.91     0.01                             copy_mass_fractions (main.c:733 @ 405364)
  0.02     63.92     0.01                             f (main.c:596 @ 404a2b)
  0.02     63.93     0.01                             print_progress (main.c:880 @ 405c96)
  0.01     63.94     0.01                             f (main.c:547 @ 404642)
  0.01     63.94     0.01                             f (main.c:557 @ 4046a3)
  0.01     63.95     0.01                             initialise_cvode (main.c:524 @ 404640)
  0.00     63.95     0.00   933600     0.00     0.00  copy_mass_fractions (main.c:730 @ 405334)
  0.00     63.95     0.00   859207     0.00     0.00  check_flag (main.c:789 @ 405607)
  0.00     63.95     0.00    31200     0.00     0.00  approx_1_minus_sqrt_1_minus_epsilon (main.c:862 @ 405b69)
  0.00     63.95     0.00    31200     0.00     0.00  average_mass_fraction (main.c:839 @ 40593e)
  0.00     63.95     0.00    31200     0.00     0.00  calculate_dep_thick (main.c:817 @ 405773)
  0.00     63.95     0.00    31200     0.00     0.00  copy_reaction_rate_const (main.c:739 @ 4053b0)
  0.00     63.95     0.00    31200     0.00     0.00  degrade_reactor (main.c:678 @ 404f50)
  0.00     63.95     0.00    21613     0.00     0.00  print_mass_fractions (main.c:652 @ 404d1d)
  0.00     63.95     0.00     7200     0.00     0.00  calculate_join_exit_mass_fractions (main.c:918 @ 405eec)
  0.00     63.95     0.00     2400     0.00     0.00  print_progress (main.c:878 @ 405c73)
  0.00     63.95     0.00     2400     0.00     0.00  update_timings (main.c:870 @ 405c1f)
  0.00     63.95     0.00      215     0.00     0.00  initialise_mass_fractions (main.c:391 @ 403bc4)
  0.00     63.95     0.00       13     0.00     0.00  allocate_mass_fractions (main.c:370 @ 403a3f)
  0.00     63.95     0.00       13     0.00     0.00  calc_reaction_rate_const (main.c:600 @ 404a44)
  0.00     63.95     0.00       13     0.00     0.00  calc_reactor_steps_and_error (main.c:358 @ 40398f)
  0.00     63.95     0.00       13     0.00     0.00  close_files_and_delete_unused_ones (main.c:894 @ 405d63)
  0.00     63.95     0.00       13     0.00     0.00  deallocate_internal_mass_fractions (main.c:886 @ 405d15)
  0.00     63.95     0.00       13     0.00     0.00  fprint_reactor_conditions (main.c:663 @ 404e78)
  0.00     63.95     0.00       13     0.00     0.00  initialise_internal_mass_fractions (main.c:382 @ 403b3c)
  0.00     63.95     0.00       13     0.00     0.00  prepare_data_file (main.c:628 @ 404b0a)
  0.00     63.95     0.00       13     0.00     0.00  print_header (main.c:640 @ 404bfb)
  0.00     63.95     0.00       13     0.00     0.00  setup_reactors (main.c:334 @ 4037c9)
                                                                                       6,77          Top
```

Optimising code





effield

Optimising code

- Optimise with focus on memory as well as CPU cycles and communication between cores
- Memory is not cheap and can get consumed very easily if not careful

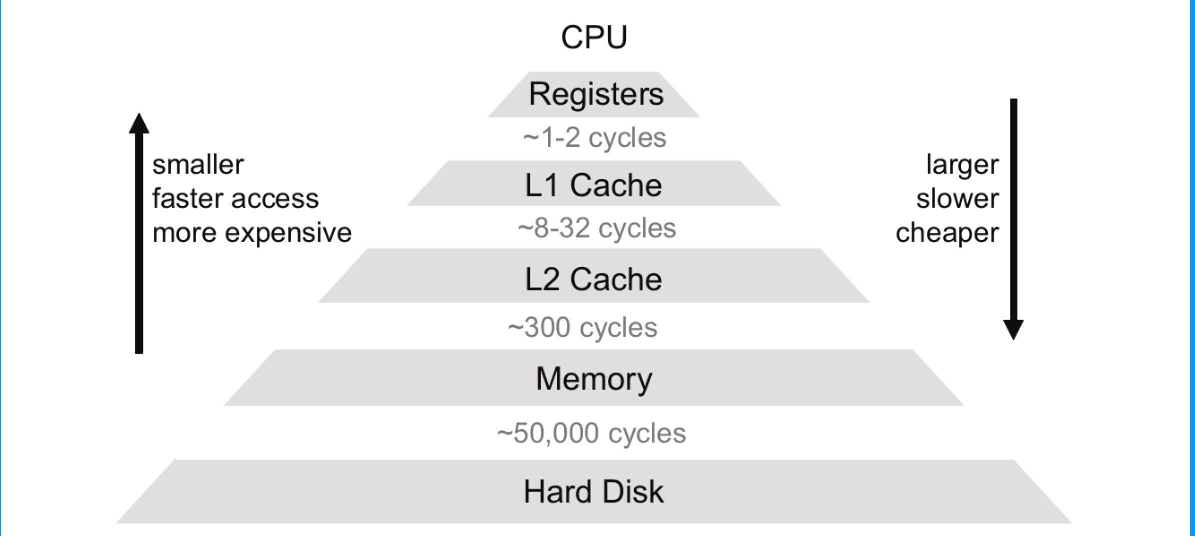


Fig. 5. Memory hierarchy. Typical latencies for data transfers from the CPU to each of the levels are shown. The numbers shown here are only an indication, and the actual numbers will depend on the exact architecture under consideration.

Chellappa S., Franchetti F., Puschel M., 2008, How to write fast numerical code: A small introduction, Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007.

Summary

- Try and write quality code with focus on being:
  - Readable, simple, and therefore maintainable
  - Correct, error/warning free
  - Portable
  - Stable
  - Can handle errors where most likely to occur
- Optimise your whole workflow, and lastly work on speeding up your code
- If you optimise be careful not to speed-up at the expense of code quality
- Good luck!