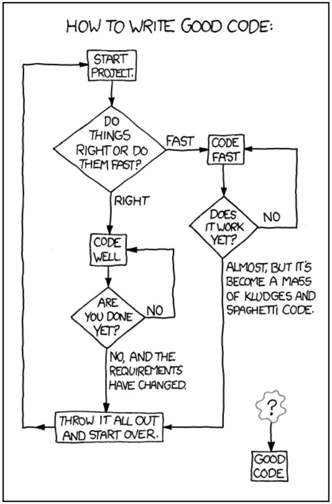
Eilmer4: What's happening?

Peter Jacobs
School of Mechanical and Mining Engineering
The University of Queensland

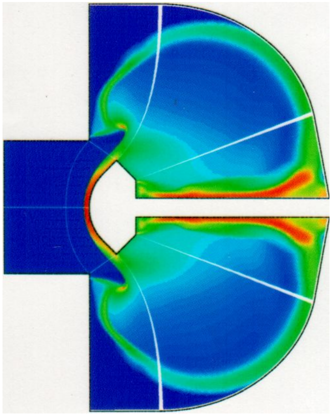
21 May 2015

Eilmer4: the quest for good code


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
RIGHT
DOES IT WORK YET?
NO
CODE WELL
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
?
THROW IT ALL OUT AND START OVER.
GOOD CODE

- That is, code that doesn't make your eyes bleed.
- Authors (so far): PJ and Rowan Gollan
- Where to get it: The Compressible Flow CFD Project http://cfcfd.mechmining.uq.edu.au/
- On the left: http://xkcd.com/844/

Eilmer3 in a nutshell



- Eulerian description of the gas (finite-volume, axisymmetric or 3D)
- Transient, time-accurate
- Shock capturing
- Multiple-block, structured grids
- Parallel computation on a cluster computer, using MPI
- *Really good* thermochemistry module by Rowan Gollan and Dan Potter

Mathematical gas dynamics (in differential form, by RJG)

Conservation of mass:

$$\frac{\partial}{\partial t}\rho + \nabla \cdot \rho \mathbf{u} = 0 \tag{1}$$

Conservation of species mass:

$$\frac{\partial}{\partial t}\rho_i + \nabla \cdot \rho_i \mathbf{u} = -\left(\nabla \cdot \mathbf{J}_i\right) + \dot{\omega}_i \tag{2}$$

Conservation of momentum:

$$\frac{\partial}{\partial t}\rho \mathbf{u} + \nabla \cdot \rho \mathbf{u}\mathbf{u} = -\nabla p - \nabla \cdot \left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\} \tag{3}$$

Conservation of total energy:

$$\begin{aligned}\frac{\partial}{\partial t}\rho E + \nabla \cdot (e + \frac{p}{\rho})\mathbf{u} &= \nabla \cdot [k\nabla T + \sum_{s=1}^{N_v} k_{v,s}\nabla T_{v,s}] + \nabla \cdot \left[\sum_{i=1}^{N_s} h_i \mathbf{J}_i\right] \\ &\quad - \left(\nabla \cdot \left[\left\{-\mu(\nabla \mathbf{u} + (\nabla \mathbf{u})^{\dagger}) + \tfrac{2}{3}\mu(\nabla \cdot \mathbf{u})\boldsymbol{\delta}\right\} \cdot \mathbf{u}\right]\right) - Q_{\mathrm{rad}}\end{aligned} \tag{4}$$

Conservation of vibrational energy:

$$\frac{\partial}{\partial t}\rho_i e_{v,i} + \nabla \cdot \rho_i e_{v,i}\mathbf{u} = \nabla \cdot [k_{v,i}\nabla T_{v,i}] - \nabla \cdot e_{v,i}\mathbf{J}_i + Q_{T-V_i} + Q_{V-V_i} + Q_{\mathrm{Chem}-V_i} - Q_{\mathrm{rad}_i} \tag{5}$$

More maths...

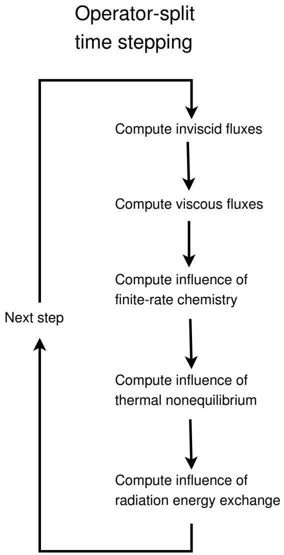
Thermodynamic model of the gas...
Finite-rate chemical kinetics...
Radiation energy exchange...
Boundary conditions...

Numerical Methods

- nonlinear function solvers – secant, Newton, fixed-point methods
- linear equation solvers – full, direct methods, iterative Jacobi, Gauss-Seidel
- single- and multidimensional-interpolation and reconstruction of data – polynomials, Bezier curves, splines, NURBS
- finite-differences (to turn our PDEs into algebraic equations or ODEs)
- quadrature / integration – Newton-Cotes, Gaussian
- ordinary-differential-equation integrators – Euler, predictor-corrector, Runge-Kutta schemes

Software implementation

- C++ for the core solver and update computations for the physical processes
- Parallel computation on a cluster computer, using MPI
- Python scripting for pre- and post-processing
- Lua for user-controlled run-time configuration in boundary conditions and source terms
- Lua for thermochemical configuration.


Operator-split time stepping
Compute inviscid fluxes
Compute viscous fluxes
Compute influence of finite-rate chemistry
Compute influence of thermal nonequilibrium
Compute influence of radiation energy exchange
Next step

Prehistory (of Eilmer, at least)

- in the late 1980s, the state of the art for scramjet simulations involving reactive flow was JP Drummond *SPARK* code
- Flow solver component based on Bob McCormack's (1969) finite-difference shock-capturing technique.
- All configuration hard-coded into the Fortran source code and compiled to run on a Cray supercomputer.
- To capture shocks in the T4 scramjet experiments, needed excessively large artificial-pressure coefficients to suppress oscillations.
- In the 1980s, a new CFD technology (upwind flux) was being developed by the applied mathematics people and parallel computing environments were being developed by the computer science people (cluster computers).

The early years (of Eilmer) – finite-volume formulations

- Dec 1990: following a CFD lesson on the chalk-board from Bob Walters and Bernard Grossman, *cns4u* was started with the intention to be like SPARK but with new technology
- Dec 1992: back in Brisbane started *L1d* (at WBM-Stalker) to reverse-engineer other groups shock tunnels
- 1993 built *sm3d*, a space-marching code for 3D scramjet flows
- 1995 through 1999: the postgrad years expanded scope of experimentation and application
  - *sm3d+* Chris Craddock, chemistry, optimization, scramjets
  - *pamela* Andrew McGhee, MPL parallelization
  - *u2de* Paul Petrie, unstructured, adaptive grid, shock tubes
  - *sf2d, sf3d* Ian Johnston, structured moving grid with chemistry and shock capturing, aerothermodynamics of entry vehicles
- 1996: code reformulation around fluxes (frequent discussions with Mike Macrossan); all code still in C with a preprocessor having a little command interpreter built in.

The middle years – parallel calculation and scripting

- 1997: discovered scripting languages Tcl and Python
- 1999-2000: Vince Wheatley, rarefied flows, L1d, another go at finite-rate chemistry
- 1999-2000: James Faddy, experiment with solution-adaptive 2D solver
- 1999-2002: Richard Goozee, SPH experiment, get to grips with MPI
- 2001-2002: Michael Elford: validation exercise with Fastran (commercial code)
- May 2003: *scriptit.tcl* provided fully programmable environment for simulation-preparation.
- Aug 2004: *Elmer* began as a hybrid code using Python and C.

Growing up – C++ and Python

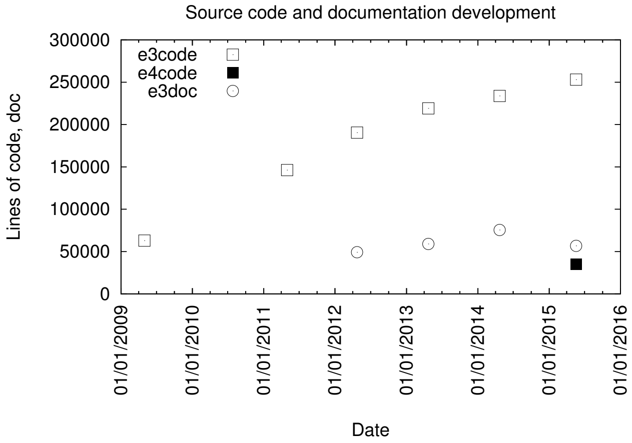
- Feb 2005: Realized that Python was much nicer for occasional users. (*scriptit.py*)
- Jun 2005: rewrite of *Elmer(2)* in C alone so that Andrew Denman could get on with his thesis
- 2005: more experimentation in CFD methods; Joseph Tang wrote a hierarchical-grid solver based on virtual-cell embedding.
- Mar 2006: *mbcns2* was a rewrite of *mbcns* but with C++ to manage code complexity.
- Jul 2006: rewrite *Elmer2* in C++
- Sep 2007: this is crazy; we should merge the 2D and 3D codes
- Jan 2008: make *mbcns2* look more like *Elmer2*
- Nov 2008: and call it *Eilmer3*.

Letting other people get some work done.

- User Guide (458pp) and Theory Book (plus guides for turbulent flows, block-marching, etc).
- class-based implementation of boundary conditions; easier to extend and maintain.
- generalization of the solution files; expandable content
- generalization of the postprocessing code as a library; specialized postprocessors are easy and can be mixed with preparation of new simulations.
- rework of house-keeping so that we can scale.
- generalization of thermochemistry, easily extendable multiple temperature model (Rowan Gollan and Dan Potter)
- coupled radiation (Dan Potter)
- turbomachinery (Carlos Ventura, Jason Czapla, Jason Qin)
- meso-scale combustion and heat transfer (Anand, Xin, Rowan)

Source-code line count

At 60 lines per page, it's equivalent to a 5000 page document.



Eilmer4 – Let's do it right, again.

Fred Brooks, in the "Mythical Man-Month: Essays on software engineering"

> *Sooner or later the first system is finished, and the architect, with firm confidence and a demonstrated mastery of the class of systems, is ready to build a second system. ...*
> *This second is the most dangerous system a man ever designs. ...*
> *The general tendency is to over-design the second system, using all the ideas and frills that were cautiously sidetracked on the first one.*

We're OK, this is not our second system.
cns4u, mbcns, mbcns2, Elmer, Elmer2, Eilmer3 ... Eilmer4.

Eilmer – patron saint of Computational Fluid Dynamics.

Stained-glass window in Malmesbury Abbey.
https://en.wikipedia.org/wiki/Eilmer_of_Malmesbury

Eilmer4 – features.

- 3D from the beginning, 2D as a special case
- structured- and unstructured-meshes for complex geometries (presently vapourware, but the new design is accommodating)
- refined thermochemistry (Rowan)
- moving meshes (Jason Qin)
- simplified and generalized boundary conditions (Daryl Bond's suggestion)
- coupled heat transfer
- shared-memory parallelism for multicore workstation use
- block-marching for speed (nenzfr and nozzle design)
- programmed in D with Lua scripting
- small kitchen sink

More Software Engineering

- Languages
  - Fortran is OK, but why bother.
  - C/C++ is for experts, and all roads lead to C.
  - D is the way to write for statically-typed, compiled code and retain some sanity.
  - Python convenient for the top-level code that the user sees but
  - Lua is easily embeddable (core C/C++/D code calls functions written in Lua)
- Other tools
  - Editors (emacs) with syntax-highlighting.
  - Source code revision control – mercurial.
  - OS environment – Linux on cluster computers.
  - Compilers (to machine code)
  - Code checkers – static checks by compiler, memory access by Valgrind, gdb to show where the program goes off into the weeds

Collecting the low-hanging fruit of parallelism

```d
// Determine the allowable time step -- serial version.
double dt_allow = 1.0e9;
foreach (myblk; gasBlocks) {
    if (!myblk.active) continue;
    double local_dt_allow = myblk.determine_time_step_size(dt_global);
    dt_allow = min(dt_allow, local_dt_allow);
}



// Determine the allowable time step -- parallel version.
shared double dt_allow = 1.0e9;
foreach (myblk; parallel(gasBlocks,1)) {
    if (!myblk.active) continue;
    double local_dt_allow = myblk.determine_time_step_size(dt_global);
    dt_allow = min(dt_allow, local_dt_allow);
}
```

Notes:

- Need to keep most data thread local.
- D Compiler expands "parallel" into code that hands out tasks to the default ThreadPool.

Verification and Validation Examples

Verification:

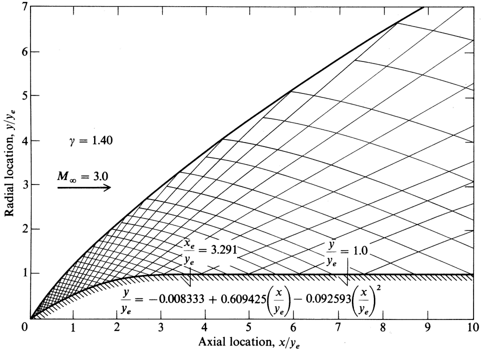
- Are we solving the equations correctly?
- Compare with numerical solutions from other codes.
- Exact solution: oblique detonation wave (idealized chemistry).
- Manufactured solution that we must match (using special source terms and BCs).

Validation:

- Are we solving the correct gas-dynamic equations?

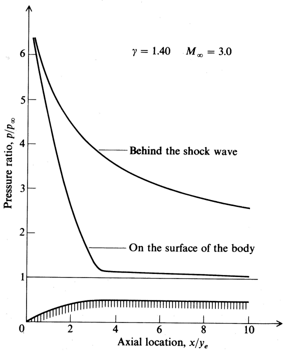
Example 1: sharp-nosed projectile

- Original Zucrow & Hoffman; also Anderson's Hypersonics text
- Shape of surface defined by polynomial equation
- Can compare numerical solutions



Radial location, y/y_e
Axial location, x/y_e
$\gamma = 1.40$
$M_\infty = 3.0$
$\frac{x_e}{y_e} = 3.291$
$\frac{y}{y_e} = 1.0$
$\frac{y}{y_e} = -0.008333 + 0.609425\left(\frac{x}{y_e}\right) - 0.092593\left(\frac{x}{y_e}\right)^2$

FIGURE 5.5
A typical characteristics mesh. (*From Zucrow and Hoffman, Ref. 53.*)



Input script – gas model and flow

```lua
-- sharp.lua
config.title = "Mach 3 flow over a sharp 2D body"
print(config.title)

nsp, nmodes = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)
initial = FlowState:new{p=5955.0, T=304.0, velx=0.0, vely=0.0}
inflow = FlowState:new{p=95.84e3, T=1103.0, velx=2000.0, vely=0.0}

```

Notes:

- user's input script is Lua source code
- arguments to function calls delimited by ()
- tables delimited by {}
- object model by convention as described in Ierusalimschy's book "Programming in Lua"

Input script – user-defined functions

```
-- Geometry of flow domain.
function y(x)
   -- (x,y)-space path for x>=0
   if x <= 3.291 then
      return -0.008333 + 0.609425*x - 0.092593*x*x
   else
      return 1.0
   end
end

function xypath(t)
   -- Parametric path with 0<=t<=1.
   local x = 10.0 * t
   local yval = y(x)
   if yval < 0.0 then
      yval = 0.0
   end
   return {x, yval, 0.0}
end
```

Notes:

- global variables unless stated otherwise
- can return tables

Input script – geometry definition

```
a = Vector3:new{-1.0, 0.0}
b = Vector3:new{ 0.0, 0.0}
c = Vector3:new{10.0, 1.0}
d = Vector3:new{10.0, 7.0}
e = Vector3:new{ 0.0, 7.0}
f = Vector3:new{-1.0, 7.0}
ab = Line:new{a, b}; bc = LuaFnPath:new{"xypath"} -- lower boundary in
surface
fe = Line:new{f, e}; ed = Line:new{e, d} -- upper boundary
af = Line:new{a, f}; be = Line:new{b, e}; cd = Line:new{c, d} -- verti
-- Mesh the patches, with particular discretisation.
ny = 60
clustery = RobertsFunction:new{end0=true, end1=false, beta=1.3}
clusterx = RobertsFunction:new{end0=true, end1=false, beta=1.2}
grid0 = StructuredGrid:new{psurface=makePatch{fe, be, ab, af},
                           cfList={nil,clustery,nil,clustery},
                           niv=17, njv=ny+1}
grid1 = StructuredGrid:new{psurface=makePatch{ed, cd, bc, be},
                           cfList={clusterx,nil,clusterx,clustery},
                           niv=81, njv=ny+1}
```

Note:

- alternatively, could import grids

Input script – flow domain with boundary conditions

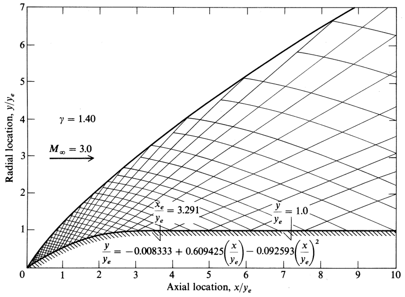
```
-- Define the flow-solution blocks.
blk0 = SBlock:new{grid=grid0, fillCondition=inflow}
blk1 = SBlock:new{grid=grid1, fillCondition=initial}
-- Set boundary conditions.
identifyBlockConnections()
blk0.bcList[west] = SupInBC:new{flowCondition=inflow}
blk1.bcList[east] = ExtrapolateOutBC:new{}

config.max_time = 15.0e-3  -- seconds
config.max_step = 2500
config.dt_init = 1.0e-6
```

Notes:

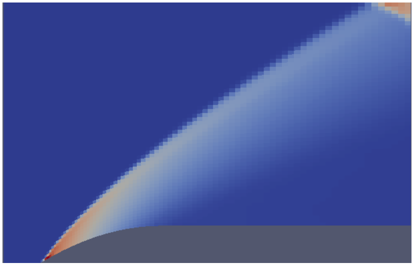
- We have separated grid generation from block definition.
- fillCondition could be given as a (user-defined) function of position (x,y,z).
- Also, could provide lists of boundary conditions to the block constructors.

Result – pressure field

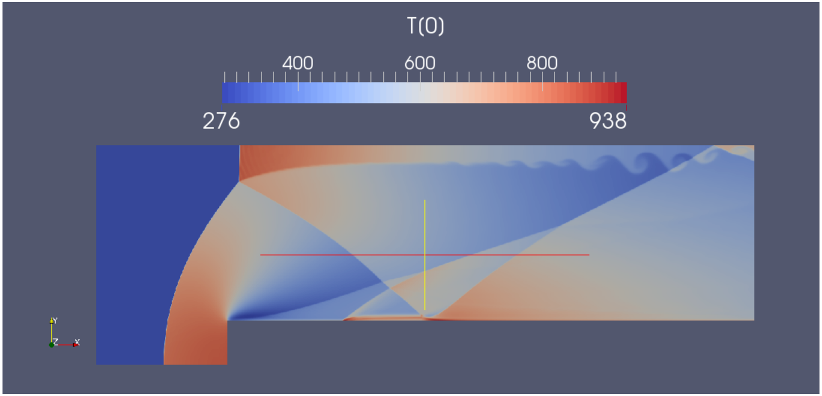


$\gamma = 1.40$
$M_\infty = 3.0$
$\frac{x_e}{y_e} = 3.291$
$\frac{y}{y_e} = 1.0$
$\frac{y}{y_e} = -0.008333 + 0.609425\left(\frac{x}{y_e}\right) - 0.092593\left(\frac{x}{y_e}\right)^2$
Radial location, y/y_e
Axial location, x/y_e

FIGURE 5.5
A typical characteristics mesh. (*From Zucrow and Hoffman, Ref. 53.*)



Example 2: supersonic flow over a forward-facing step

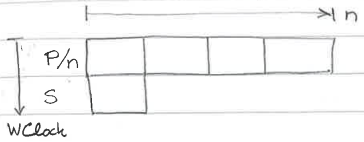


Note:

- Hurry up and show the animation.

Parallel performance – Amdahl's law

Model of a calculation done in parallel, n CPUs



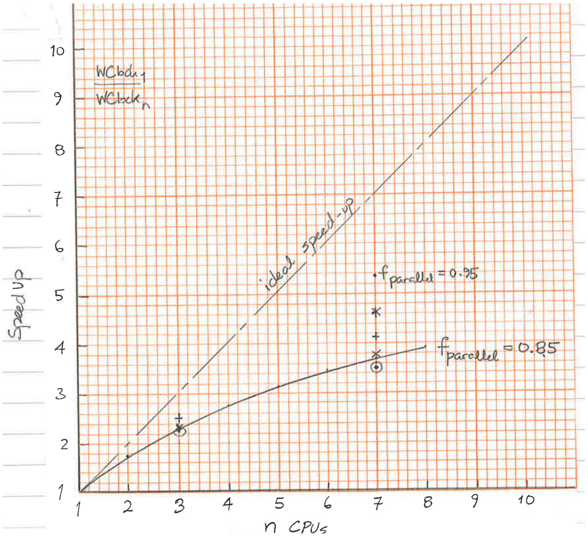
$$WC_n = \frac{P}{n} + S, \quad WC_1 = P + S \;\Big\}\; WC_n = \frac{P}{n} + WC_1 - P$$

$$\text{Speedup} = \frac{WC_1}{WC_n} \; ; \; f_{\text{parallel}} = \frac{P}{WC_1}$$

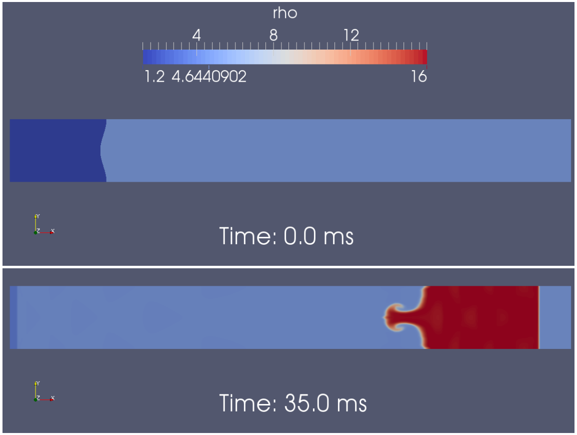
$$\frac{1}{\text{SpeedUp}} = 1 - f_{\text{parallel}}\left(\frac{n-1}{n}\right) \; ; \; f_{\text{parallel}} = \left(\frac{\text{SpeedUp} - 1}{\text{Speedup}}\right)\left(\frac{n}{n-1}\right)$$

- How quickly can we do a fixed-size computation if we employ n processors?

Parallel performance – forward-facing-step, 21 blocks



Example 3: Richtmyer-Meshkov Instability



Animations:

- density, pressure, temperature, velocity gradient