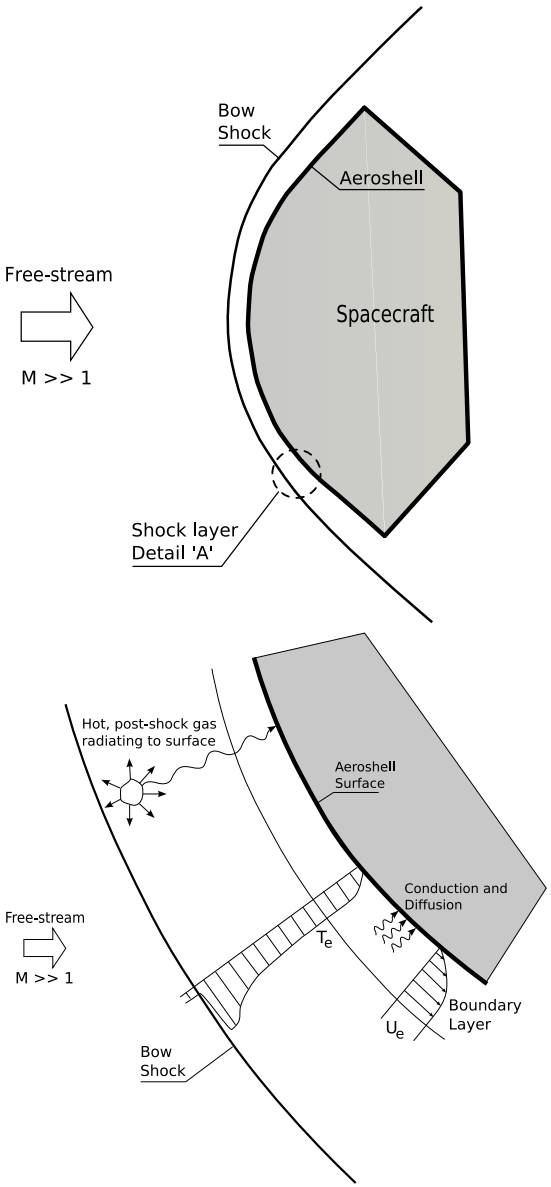
Eilmer 4.0
A toolkit for doing gas-dynamic calculations

Peter Jacobs

The University of Queensland

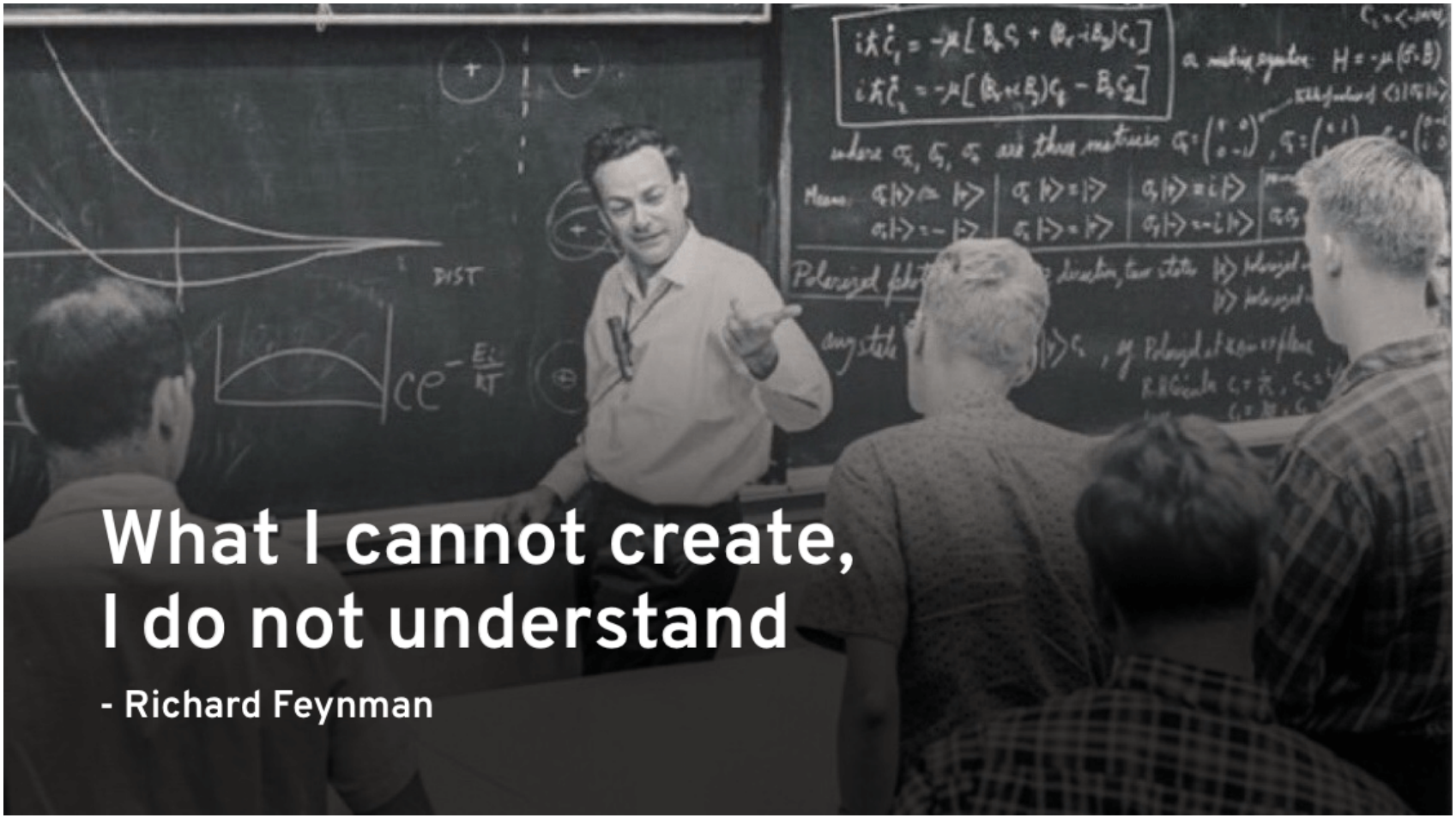
05 July 2018

When should we compute hypersonic flows?



- The flow physics are modelled well, but interactions are complex.
- Physical experimentation provides insights but has limitations, such as: scaling (time and length), boundary conditions, quantification of uncertainties, expense.
- Computer simulation complements physical experiments, and vice versa.
- Analysis via computer simulation (might) substitute when we don't have suitable experience.
- Computer analysis is good for 'what-if' studies, and design.

What I cannot simulate, I do not understand.



Compressible flow simulation via finite volumes

- What we compute: solution to the conservation equations for a viscous compressible flow
- How we compute: discretise in space and time
- Start with a known state and boundary conditions, then use an update algorithm
  - Convective fluxes: reconstruction-evolution approach, interpolation order, flux calculators, limiters
  - Viscous-transport fluxes: gradient estimation
  - Time integration

Form of the equations to solve

Integral form of a conservation law

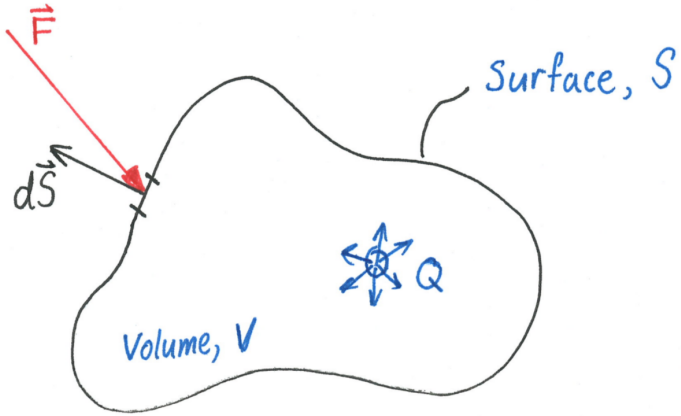
A general conservation law for quantity U is written in integral form as

$$\frac{\partial}{\partial t}\int_V U dV = -\oint_S \left(\vec{F}_c - \vec{F}_d\right)\cdot \hat{n}\, dA + \int_V Q dV\ , \qquad (1)$$

where S is the bounding surface and $\hat{n}$ is the outward-facing unit normal of the control surface.

What are the quantities U?

The conserved quantities in a compressible flow are **mass**, **momentum** and **energy**. In two dimensions, we can group these conservation equations with vector notation.



Integral form of conservation laws in vector form

For an ideal gas in two dimensions, the vector of conserved quantities is:

$$U = \begin{bmatrix} \rho \\ \rho u_x \\ \rho u_y \\ \rho E \end{bmatrix} , \tag{2}$$

with convective flux vector

$$\vec{F}_c = \begin{bmatrix} \rho u_x \\ \rho u_x^2 + p \\ \rho u_y u_x \\ \rho E u_x + p u_x \end{bmatrix} \hat{i} + \begin{bmatrix} \rho u_y \\ \rho u_x u_y \\ \rho u_y^2 + p \\ \rho E u_y + p u_y \end{bmatrix} \hat{j} , \tag{3}$$

and diffusive flux vector

$$\overline{F}_d = \begin{bmatrix} 0 \\ \tau_{xx} \\ \tau_{yx} \\ \tau_{xx} u_x + \tau_{yx} u_y + q_x \end{bmatrix} \hat{i} + \begin{bmatrix} 0 \\ \tau_{xy} \\ \tau_{yy} \\ \tau_{xy} u_x + \tau_{yy} u_y + q_y \end{bmatrix} \hat{j} . \tag{4}$$

The vector of sources Q is typically zero.

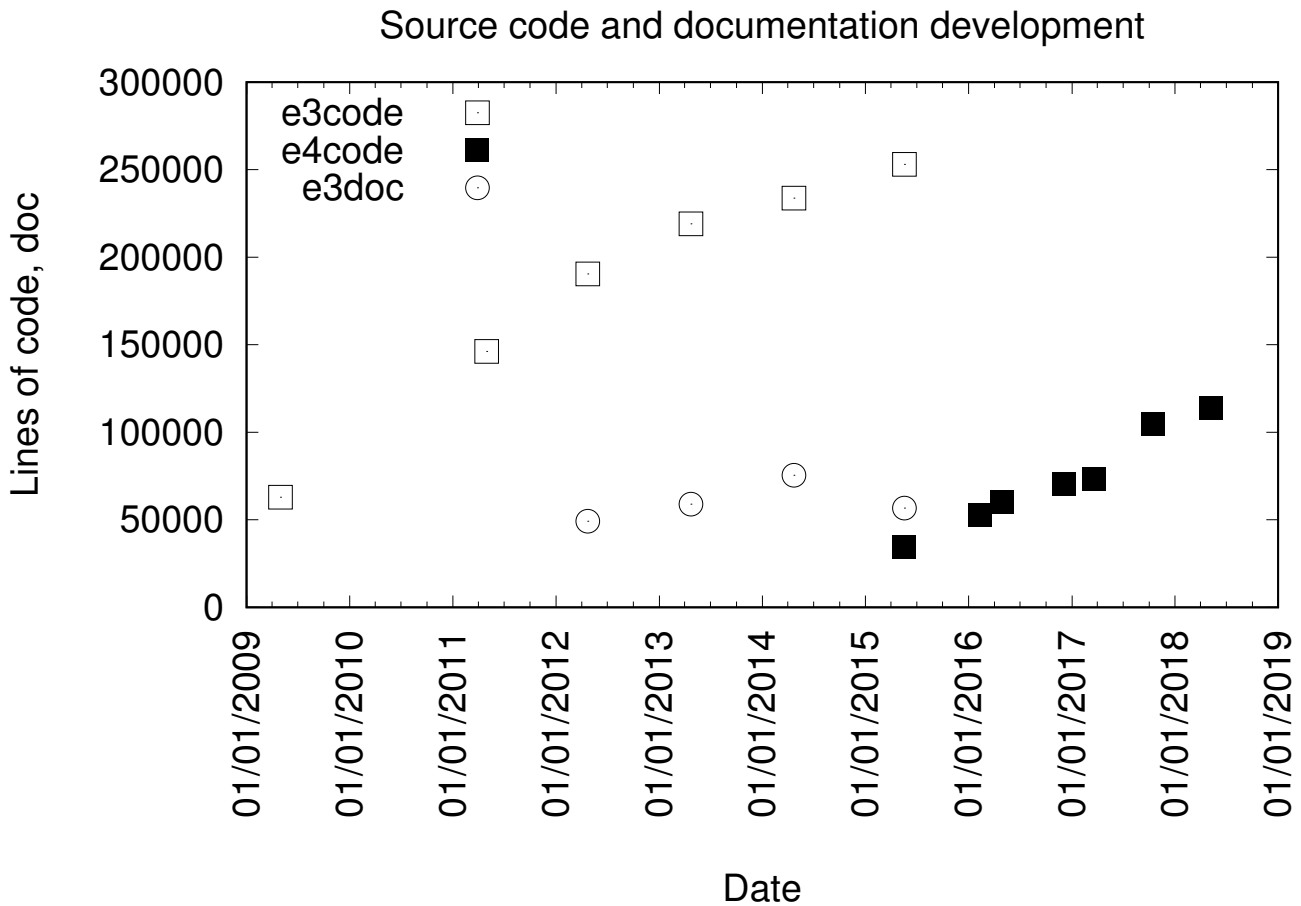
Other necessary (and optional) pieces...

Thermodynamic model of the gas
Finite-rate chemical kinetics
Radiation energy exchange (yet to port)
Boundary conditions
Features:

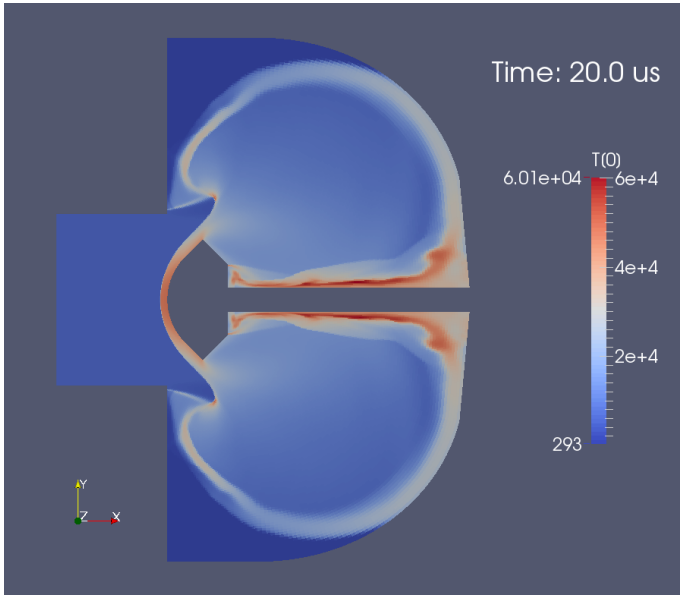
- 3D from the beginning, 2D as a special case
- structured- and unstructured-meshes for complex geometries
- moving meshes
- coupled heat transfer
- shared-memory parallelism for multicore workstation use
- block-marching for speed (nenzfr and nozzle design)
- GPGPU processing for thermochemistry
- MPI distributed-memory parallelism

Development progress, in lines of source code.

At 60 lines per page,
the Eilmer4 code is equivalent to a 1200 page document.



Development progress, features – 1/2



- 2D/3D compressible flow simulation.
- Gas models include ideal, thermally perfect, equilibrium.
- Finite-rate chemistry.
- Multi-temperature and state-specific thermochemistry.
- Inviscid, laminar, turbulent (k-ω) flow.
- Solid domains with conjugate heat transfer in 2D.
- User-controlled moving grid capability, with shock-fitting method for 2D geometries.
- Dense-gas thermodynamic models and rotating frames of reference for turbomachine modelling.

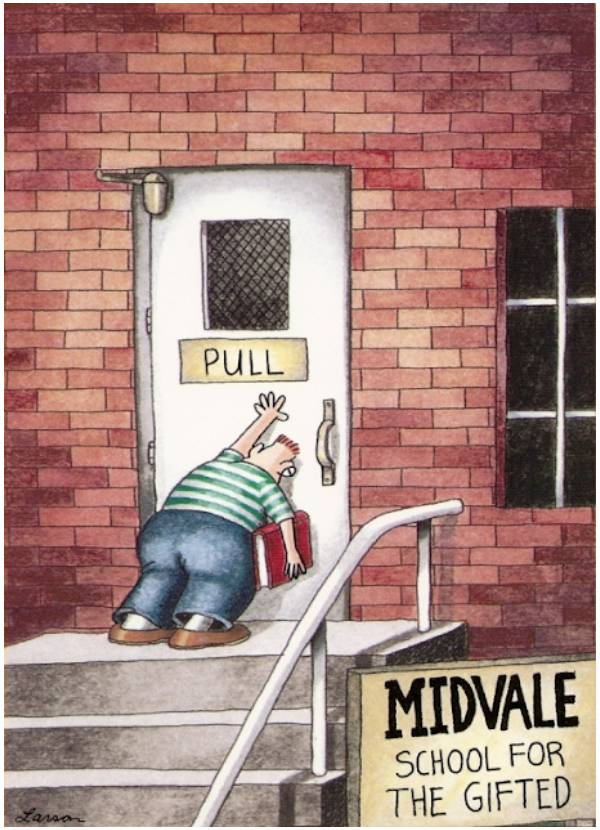
Development progress, features – 2/2

- Transient, time-accurate, using explicit Euler, PC, RK updates.
- Alternate steady-state solver with implicit updates using Newton-Krylov method.
- Parallel computation via shared-memory on workstations, and using MPI on a cluster computer.
- Multiple block, structured and unstructured grids.
- Native grid generation and import capability.
- Unstructured-mesh partitioning via Metis.

- `en.wikipedia.org/wiki/Eilmer_of_Malmesbury`
- Gas model calculator and compressible flow relations.

Development progress, documentation

- Web site: `cfcfd.mechmining.uq.edu.au/eilmer`
- Source code: `bitbucket.org/cfcfd/dgd`



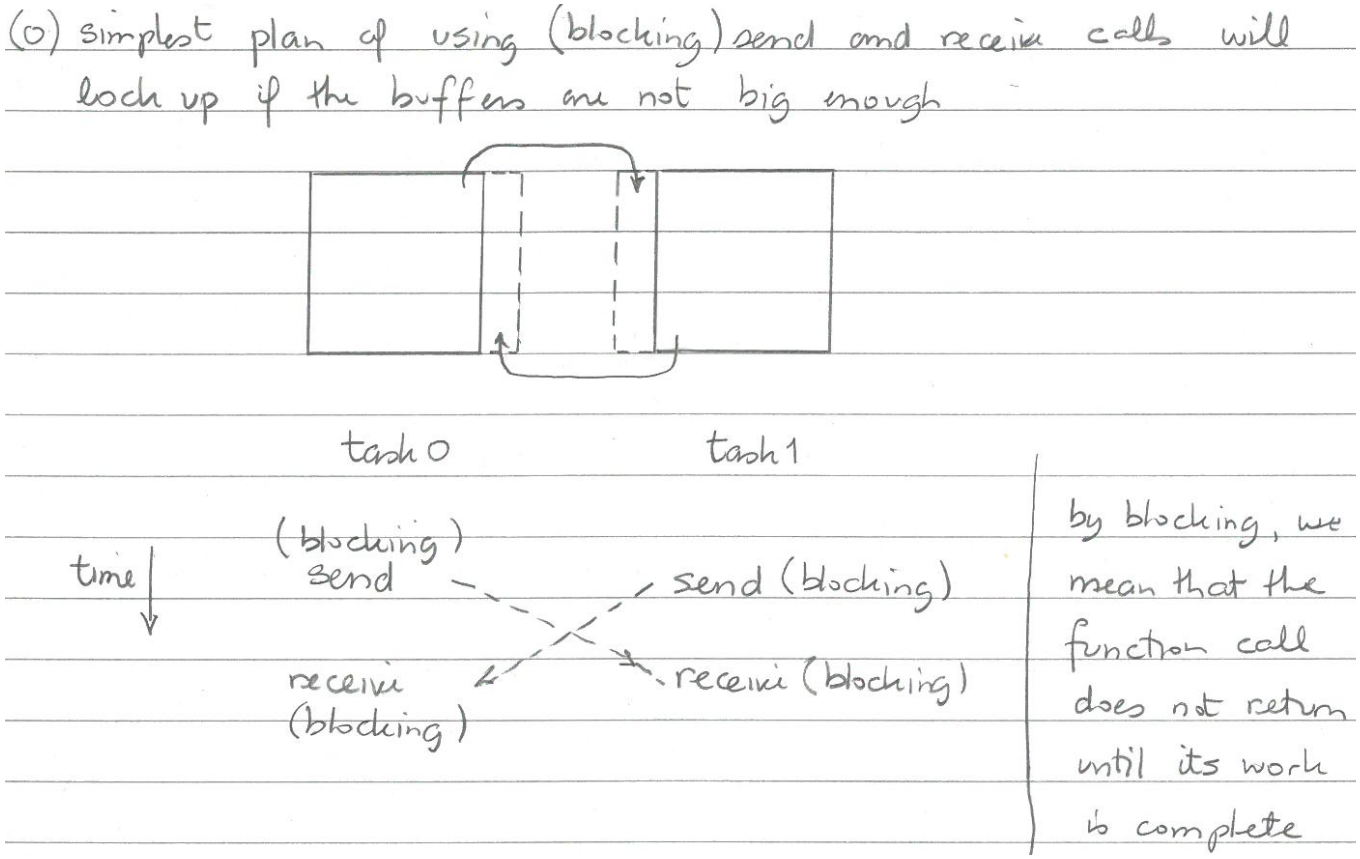
Documentation in the Eilmer 4.0 guides:

- Guide to the transient flow solver
- Guide to the basic gas models package
- Guide to the geometry package
- Formulation of the transient flow solver
- Reacting gas thermochemistry

Gary Larson, *The Far Side*

Development progress, parallel calculation with MPI

- Main development in the transient flow solver for 2018.
- Blocks partition the flow domain into connected pieces and compute the flow solution in the blocks in parallel.
- We distribute the blocks to the MPI tasks and regularly communicate the flow conditions at the edges of blocks.



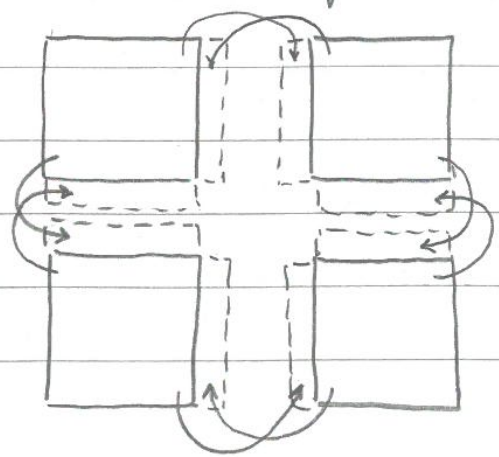
Message passing with blocking calls

Message passing with non-blocking calls

(1) better plan

task 0	task 1
post nonblocking receive	post nonblocking receive
send	send
wait until receive finished	wait until receive finished

- be careful of circular dependencies (Rowan caught this problem.)

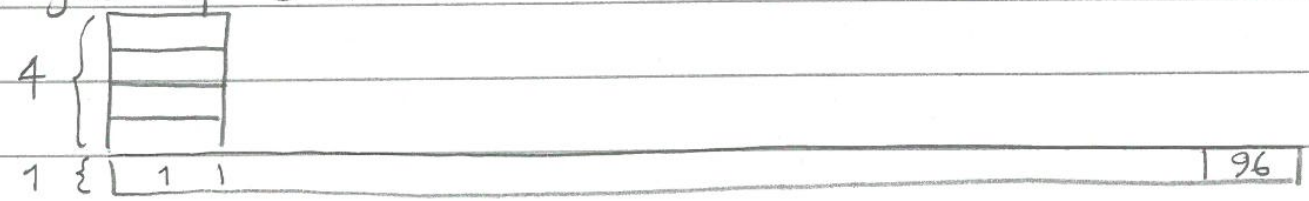


- we must post nonblocking receives for all boundaries on all blocks
- then all sends on all blocks
- then go around and check all receives are complete

Parallel computing for the impatient - 1

- let's say we have 8 jobs to do with 100 hours of work each
- each job is 96% parallel and 4% serial

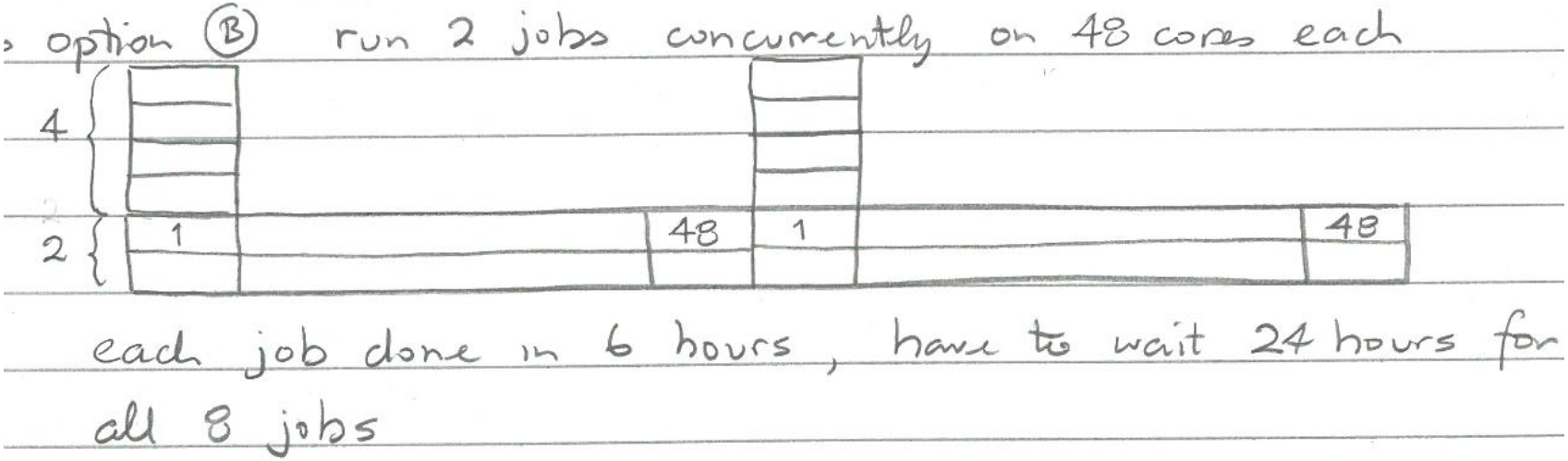
- option (A) apply 96 cores to each job.
  this will give the fastest completion time for each individual job of 5 hours



require 40 hours overall for all 8 jobs

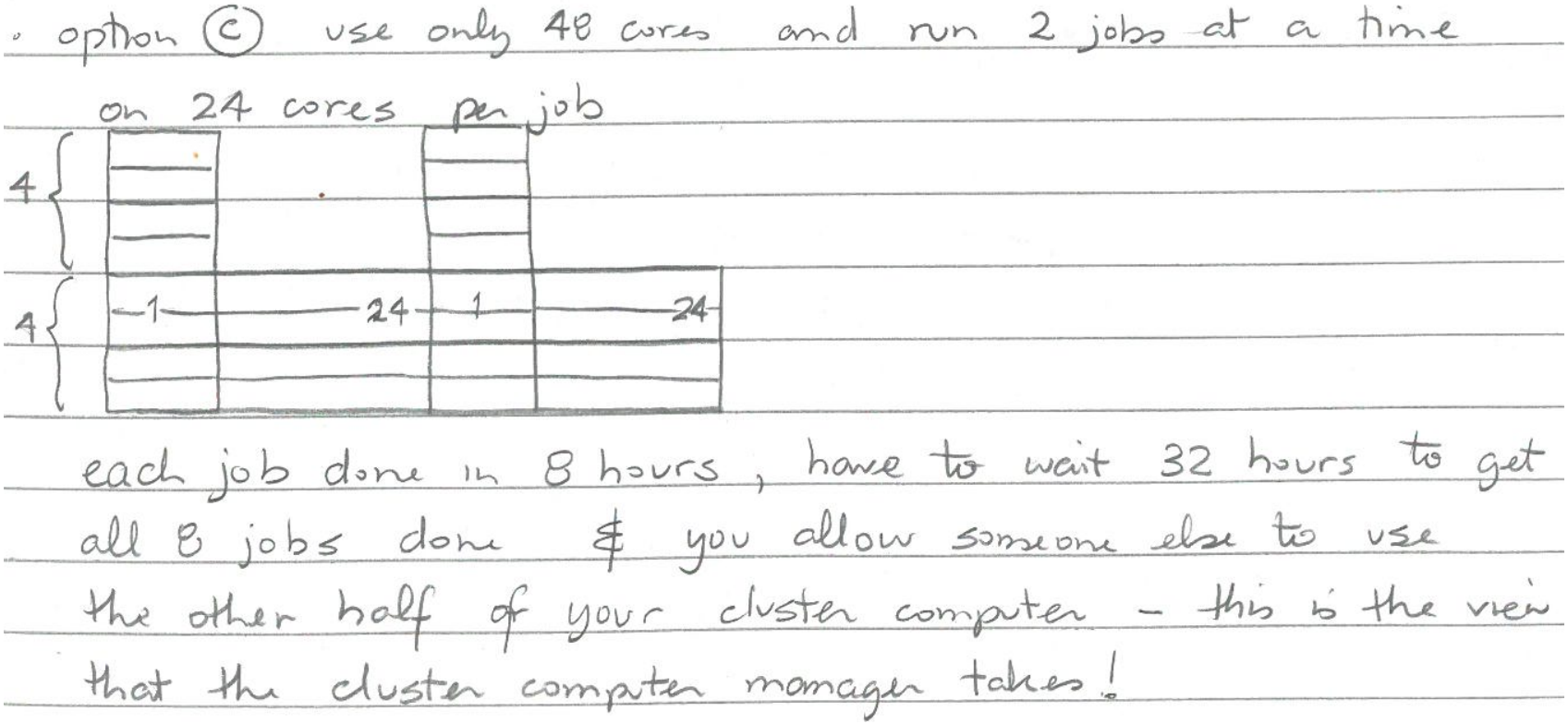
- This is good because we get nearly 5 weeks of computation done in less than 2 days.

Parallel computing for the impatient - 2



- Now we get our work done in one day on the same cluster computer.
- This is usually better...

Parallel computing for the impatient - 3

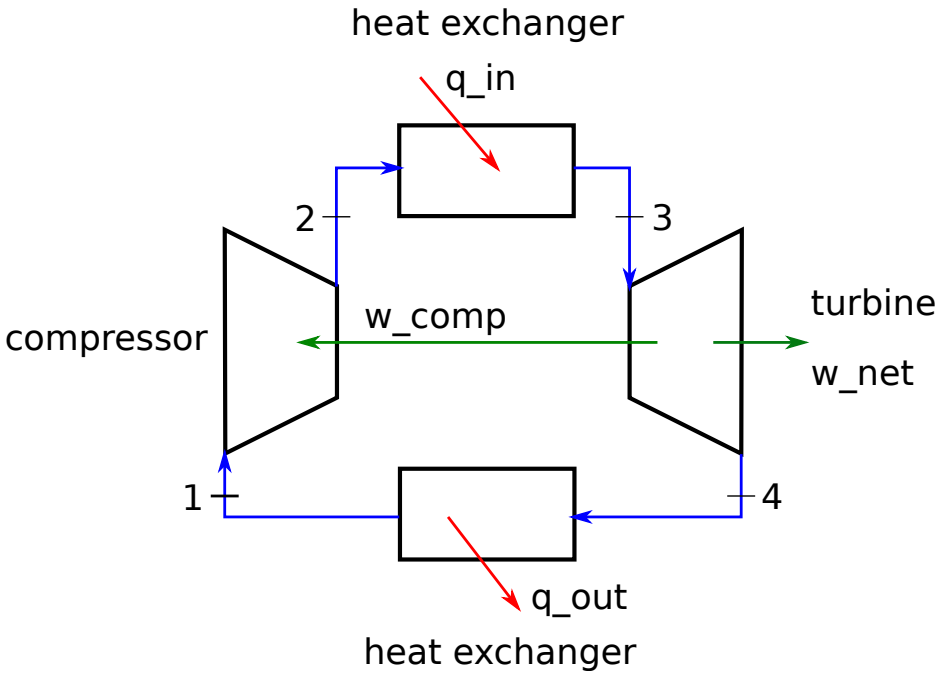


- ...but your colleagues might prefer this approach.

Eilmer as a programmable program

- Some of the internal functions are made accessible to the user to call via the Lua programming language.
- The Lua API allows Eilmer to be programmed and extended beyond our intital design.
- Examples of use:
  1. Gas state calculations for an ideal Brayton cycle
  2. Post-shock relaxation with finite-rate gas chemistry
  3. Making the set-up of a simulation easier – X3R shock tube

Using the gas functions to analyse a thermodynamic cycle



Çengel and Boles' thermodynamics text, example 9-5

"A gas-turbine power plant operating on an ideal Brayton cycle has a pressure ratio of 8. The gas temperature is 300 K at the compressor inlet and 1300 K at the turbine inlet. Utilizing the air-standard assumptions, determine (a) the gas temperature at the exits of the compressor and turbine, (b) the back work ratio, and (c) the thermal efficiency."

Brayton cycle – set up gas model

```
13  gasModelFile = "thermal-air-gas-model.lua"
14  -- gasModelFile = "ideal-air-gas-model.lua" -- Alternative
15  gmodel = GasModel:new{gasModelFile}
16  if gmodel:nSpecies() == 1 then
17     print("Ideal air gas model.")
18     air_massf = {air=1.0}
19  else
20     print("Thermally-perfect air model")
21     air_massf = {N2=0.78, O2=0.22}
22  end
23
24  print("Compute cycle states:")
25  Q = {} -- We will build up a table of gas states
26  h = {} -- and enthalpies.
27  for i=1,4 do
28     Q[i] = GasState:new{gmodel}
29     Q[i].massf = air_massf
30     h[i] = 0.0
31  end
```

Brayton cycle – compute gas states

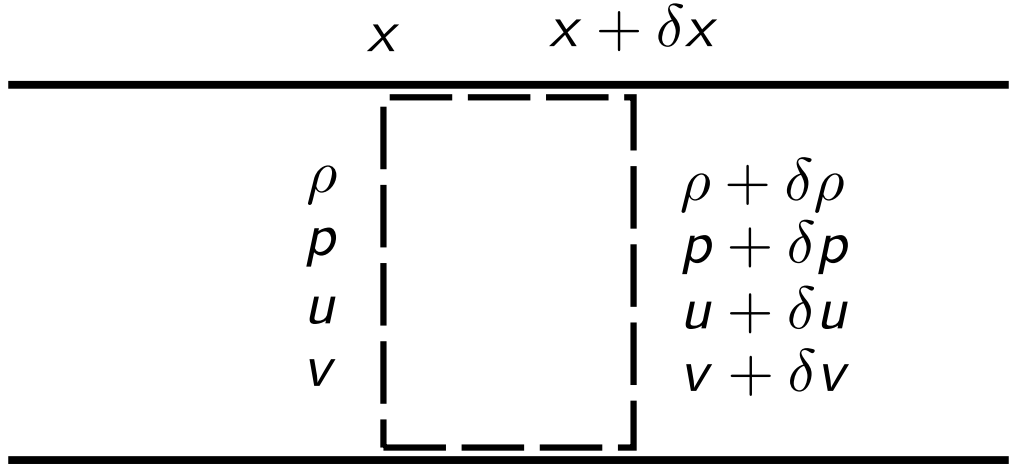
```
33  print("   Start with ambient air")
34  Q[1].p = 100.0e3; Q[1].T = 300.0
35  gmodel:updateThermoFromPT(Q[1])
36  s12 = gmodel:entropy(Q[1])
37  h[1] = gmodel:enthalpy(Q[1])
38
39  print("   Isentropic compression with a pressure ratio of 8")
40  Q[2].p = 8 * Q[1].p
41  gmodel:updateThermoFromPS(Q[2], s12)
42  h[2] = gmodel:enthalpy(Q[2])
43
44  print("   Constant pressure heat addition to T=1300K")
45  Q[3].p = Q[2].p; Q[3].T = 1300.0
46  gmodel:updateThermoFromPT(Q[3])
47  h[3] = gmodel:enthalpy(Q[3])
48  s34 = gmodel:entropy(Q[3])
49
50  print("   Isentropic expansion to ambient pressure")
51  Q[4].p = Q[1].p
52  gmodel:updateThermoFromPS(Q[4], s34)
53  h[4] = gmodel:enthalpy(Q[4])
```

Brayton cycle – report results

```
56  print("State   Pressure Temperature   Enthalpy")
57  print("             kPa           K      kJ/kg")
58  print("-----------------------------------------")
59  for i=1,4 do
60     print(string.format(" %4d %10.2f  %10.2f %10.2f",
61                  i, Q[i].p/1000, Q[i].T, h[i]/1000))
62  end
63  print("-----------------------------------------")
64  print("")
65  print("Cycle performance:")
66  work_comp_in = h[2] - h[1]
67  work_turb_out = h[3] - h[4]
68  heat_in = h[3] - h[2]
69  rbw = work_comp_in / work_turb_out
70  eff = (work_turb_out-work_comp_in) / heat_in
71  print(string.format("   turbine work out = %.2f kJ/kg", work_turb_out/1000))
72  print(string.format("   back work ratio = %.3f", rbw))
73  print(string.format("   thermal efficiency = %.3f", eff))
```

```
1   Thermally-perfect air model
2   Compute cycle states:
3      Start with ambient air
4      Isentropic compression with a pressure ratio of 8
5      Constant pressure heat addition to T=1300K
6      Isentropic expansion to ambient pressure
7
8   State   Pressure Temperature   Enthalpy
9                kPa           K      kJ/kg
10  ----------------------------------------
11      1     100.00      300.00       1.87
12      2     800.00      539.08     247.11
13      3     800.00     1300.00    1106.49
14      4     100.00      771.40     496.25
15  ----------------------------------------
16
17  Cycle performance:
18     turbine work out = 610.23 kJ/kg
19     back work ratio = 0.402
20     thermal_efficiency = 0.425
```

Steady, reacting, compressible flow in one dimension



For the control volume, shown as the dashed box, the conservation of mass, momentum and energy and the equation of state $p = f(\rho, u)$ can be linearized and be written as:

$$\begin{bmatrix} v & \rho & 0 & 0 \\ 0 & \rho v & 1 & 0 \\ vE & \rho E + p & 0 & \rho v \\ \frac{\partial f}{\partial \rho} & 0 & -1 & \frac{\partial f}{\partial u} \end{bmatrix} \cdot \begin{bmatrix} \delta\rho \\ \delta v \\ \delta p_{gda} \\ \delta u_{gda} \end{bmatrix} = \begin{bmatrix} 0 \\ -\delta p_{chem} \\ -\rho\, v\, \delta u_{chem} \\ 0 \end{bmatrix}$$

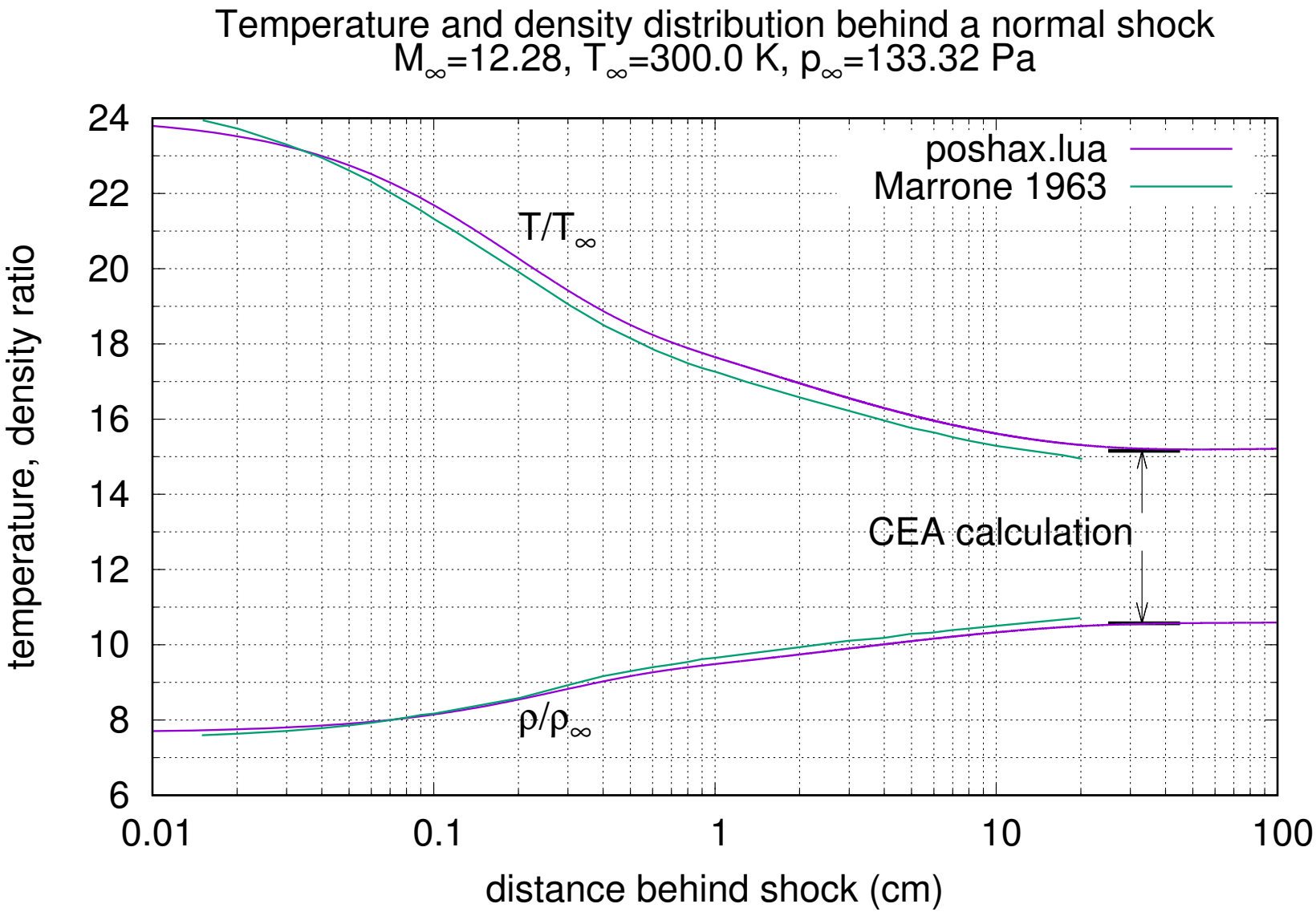
Poshax – setup and initial shock calculation

```
11  -- Typical use:
12  -- $ e4shared --custom-post --script-file=poshax.lua
13  --
14  ----------------------------------------------------------------------
15  print("Initialise a gas model.")
16  print("air 7 species Gupta-et-al reactions")
17  gmodel = GasModel:new{"air-7sp-gas-model.lua"}
18  chemUpdate = ChemistryUpdate:new{filename="air-7sp-chemistry.lua",
19                                   gasmodel=gmodel}
20  -- The example here matches the case discussed on page 63 of the thesis.
21  state1 = GasState:new{gmodel}
22  state1.p = 133.3 -- Pa
23  state1.T = 300.0 -- degree K
24  state1.massf = {N2=0.78, O2=0.22}
25  print("Free stream conditions, before the shock.")
26  gmodel:updateThermoFromPT(state1)
27  gmodel:updateSoundSpeed(state1)
28  print("state1:"); printValues(state1)
29  mach1 = 12.28
30  V1 = mach1 * state1.a
31  print("mach1:", mach1, "V1:", V1)
32
33  print("Stationary normal shock with chemically-frozen gas.")
34  state2, V2, Vg = gasflow.normal_shock(state1, V1)
35  print("    V2=", V2, "Vg=", Vg)
36  print("    state2:"); printValues(state2)
```

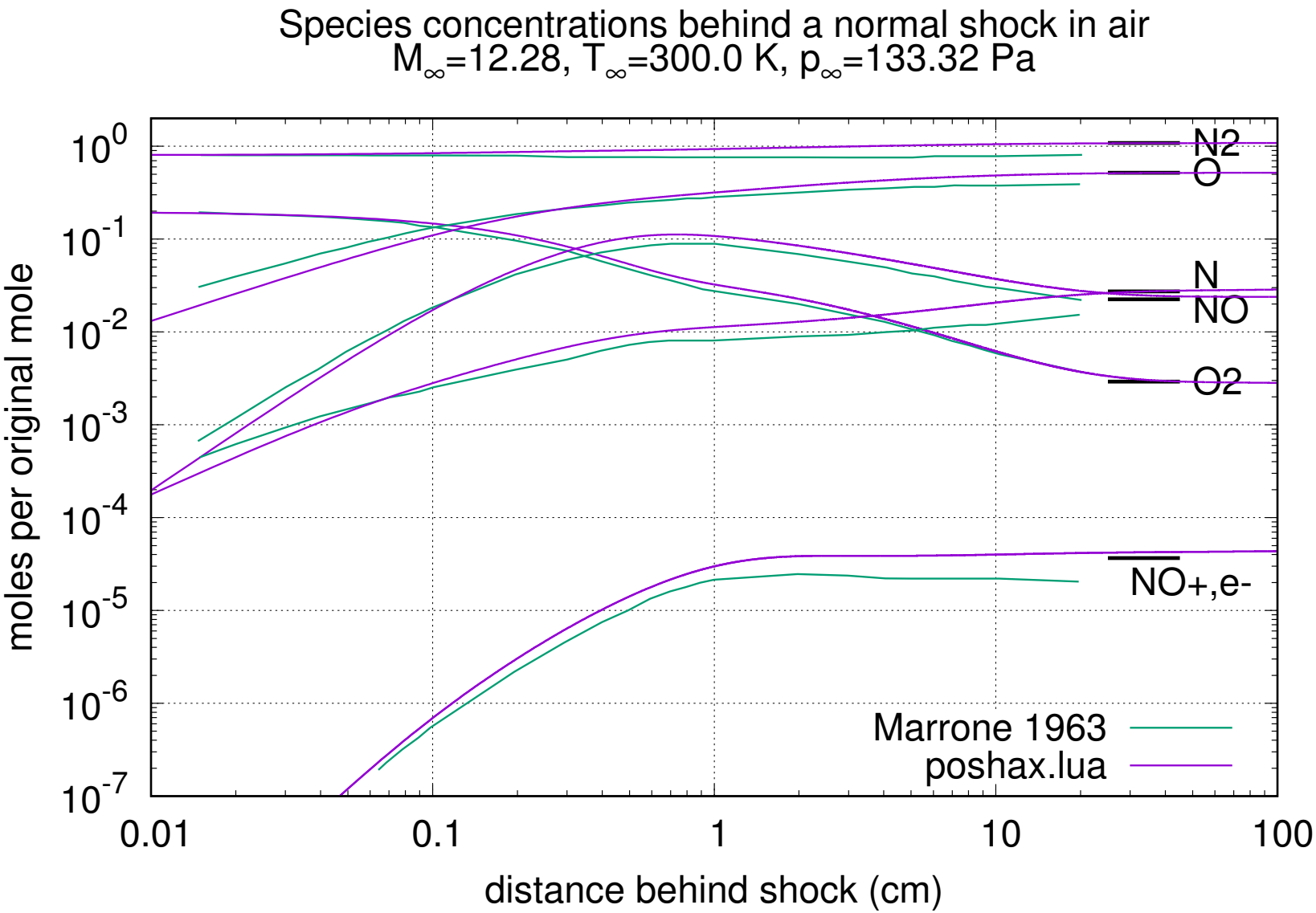
Poshax – loop with update step for relaxing gas

```
for j=1, nsteps do
   -- At the start of the step, we have GasState 0.
   -- Make some shorter names.
   local rho, T, p, u = gas0.rho, gas0.T, gas0.p, gas0.u
   -- Do the chemical increment.
   -- Make the new GasState as a clone and then update it.
   local gas1 = GasState:new{gmodel}
   copyValues(gas0, gas1)
   dt suggest = chemUpdate:updateState(gas1, t inc, dt suggest, gmodel)
   gmodel:updateThermoFromRHOU(gas1)
   local du chem = gas1.u - u
   local dp chem = gas1.p - p
   -- Do the gas-dynamic accommodation after the chemical change.
   local Etot = u + 0.5*v*v
   local dfdr, dfdu = eos derivatives(gas1, gmodel)
   -- Compute the accommodation increments using expressions from Maxima.
   local denom = rho*rho*v*v - dfdr*rho*rho - dfdu*p
   local drho = (dp chem - du chem*dfdu)*rho*rho / denom
   local dv = -(dp chem - du chem*dfdu)*rho*v / denom
   local dp gda = -(du chem*dfdu*rho*rho*v*v - dfdr*dp chem*rho*rho
                        - dfdu*dp chem*p) / denom
   local du gda = -(du chem*rho*rho*v*v - du chem*dfdr*rho*rho - dp chem*p) / denom
   -- Add the accommodation increments.
   gas1.rho = gas0.rho + drho
   v1 = v + dv
   p1 check = gas1.p + dp gda
   gas1.u = gas1.u + du gda
   gmodel:updateThermoFromRHOU(gas1)
   -- Have now finished the chemical and gas-dynamic update.
   t = t + t inc
   x = x + 0.5*(v + v1) * t inc
   f:write(string data(x, v1, gas1, dt suggest))
   if x > x final then break end
   -- House-keeping for the next step.
   v = v1
   copyValues(gas1, gas0) -- gas0 will be used in the next iteration
end
```

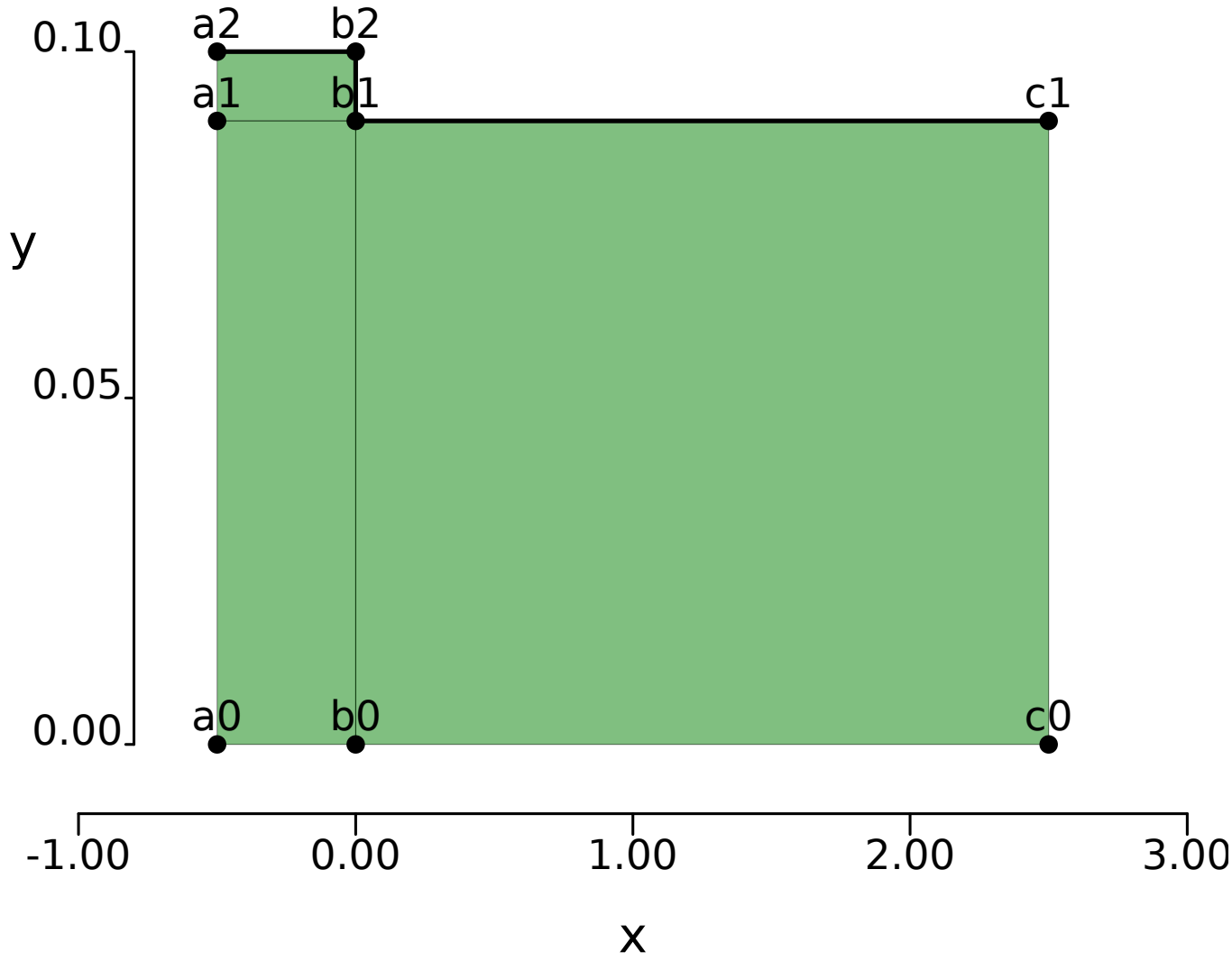
Poshax – results for Air test case



Poshax – results for Air test case



The X3R shock tube – sudden contraction half-way along



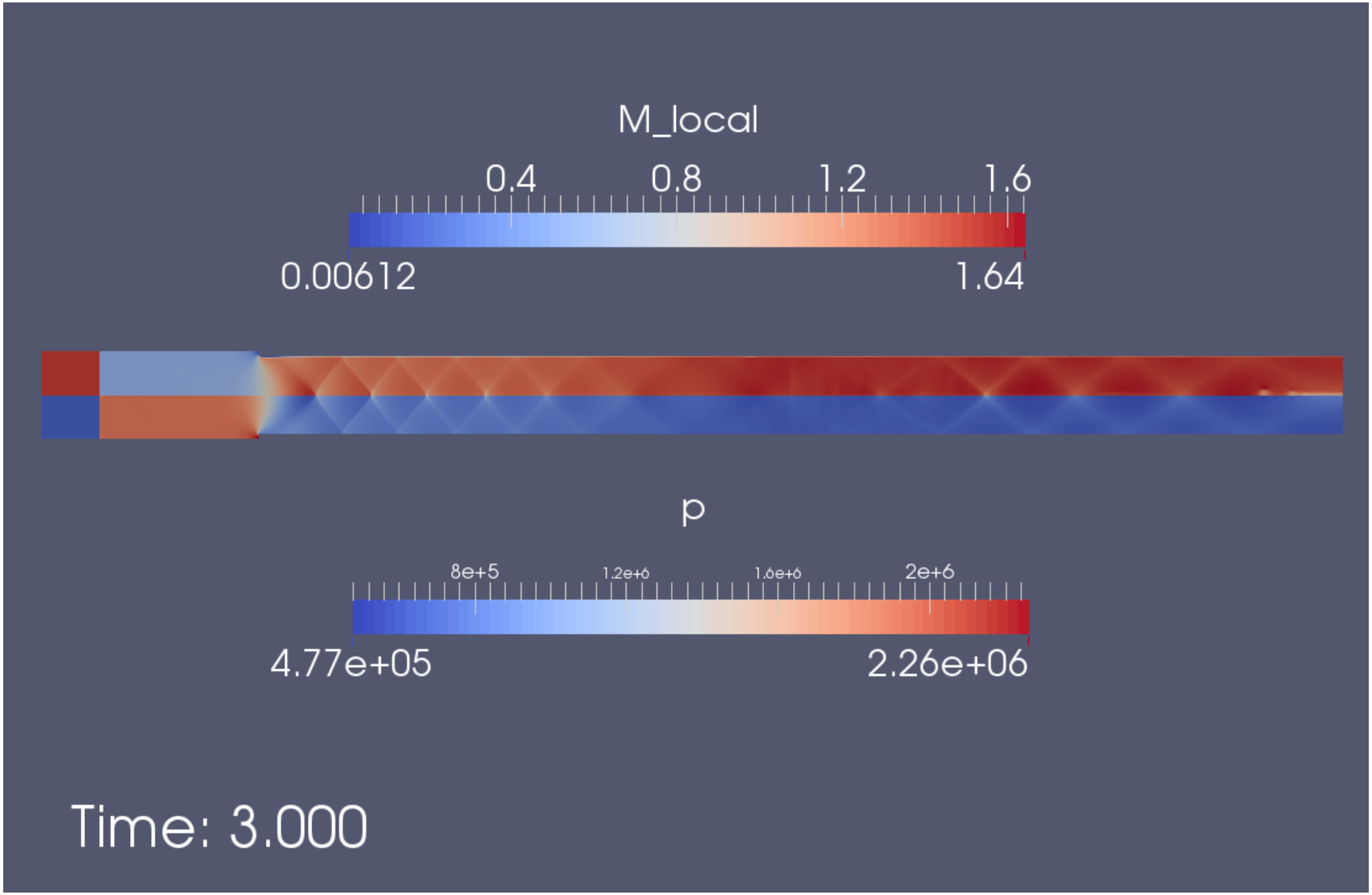
The X3R shock tube – shock-processed air flow

```
-- x3rs.lua
-- Peter J. 2018-06-14

config.title = "X3R shock tube with that unfortunate step."
print(config.title)
config.dimensions = 2
config.axisymmetric = true

-- Gas model and flow conditions.
nsp, nmodes, gm = setGasModel('ideal-air-gas-model.lua')
print("GasModel set to ideal air. nsp= ", nsp, " nmodes= ", nmodes)

-- Fill condition and shock speed from Sam Stennett.
state1 = GasState:new{gm}
state1.p=33.1e3 -- Pa
state1.T=298.1 -- degrees K
gm:updateThermoFromPT(state1); gm:updateSoundSpeed(state1)
Vs1 = 1353.34 -- m/s
Ms1 = Vs1/state1.a
print("sound speed in state 1=", state1.a, "shock Mach number=", Ms1)
state2, V2, Vg = gasflow.normal_shock(state1, Vs1)
print("state2: p=", state2.p, "T=", state2.T)
print("V2=", V2, "Vg=", Vg, "Mg=", Vg/state2.a)

initial = FlowState:new{p=state1.p, T=state1.T}
inflow = FlowState:new{p=state2.p, T=state2.T, velx=Vg}
```

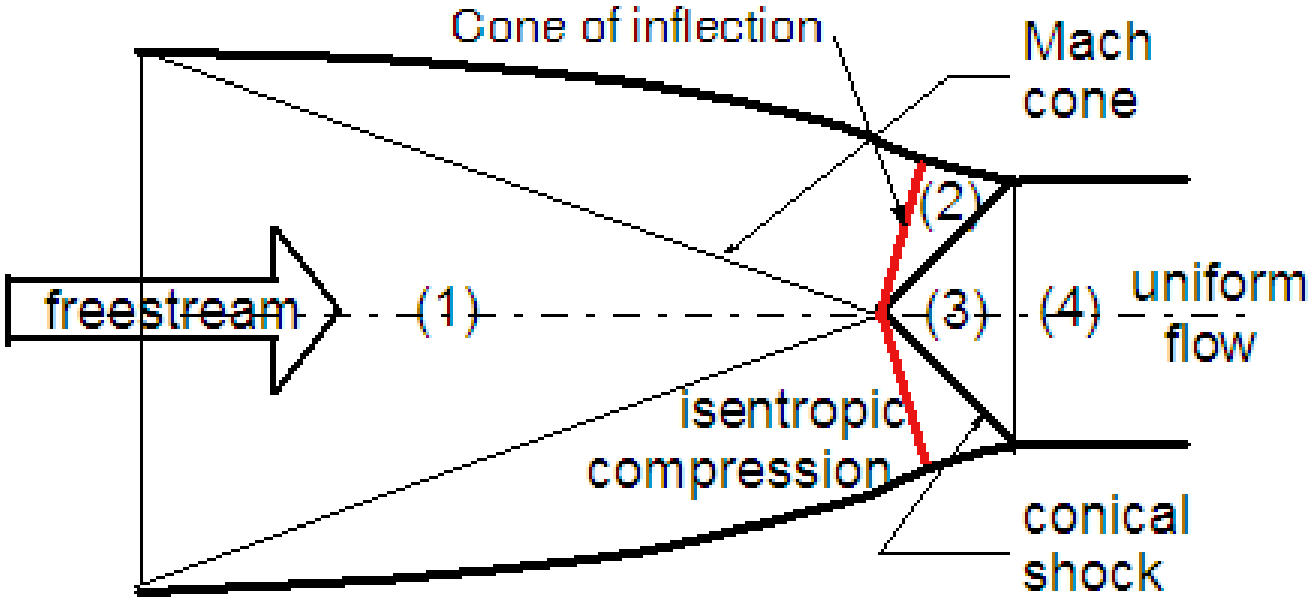
The X3R shock tube – defining the flow domain

```
28  -- Geometry of the flow domain.
29  L1 = 0.5 -- metres, upstream of step
30  L2 = 2.5 -- metres, downstream of step
31  R1 = 0.100 -- starting radius
32  R2 = 0.090 -- downstream radius
33
34  -- We put the points into a table because it will
35  -- then be easy to plot them all in a loop in the sketch.
36  pnts = {
37     a0 = Vector3:new{x=-L1, y=0},
38     a1 = Vector3:new{x=-L1, y=R2},
39     a2 = Vector3:new{x=-L1, y=R1},
40     b0 = Vector3:new{x=0, y=0.0},
41     b1 = Vector3:new{x=0, y=R2},
42     b2 = Vector3:new{x=0, y=R1},
43     c0 = Vector3:new{x=L2, y=0},
44     c1 = Vector3:new{x=L2, y=R2}
45  }
46
47  patches = {
48     surf0 = CoonsPatch:new{p00=pnts.a0, p10=pnts.b0, p11=pnts.b1, p01=pnts.a1},
49     surf1 = CoonsPatch:new{p00=pnts.a1, p10=pnts.b1, p11=pnts.b2, p01=pnts.a2},
50     surf2 = CoonsPatch:new{p00=pnts.b0, p10=pnts.c0, p11=pnts.c1, p01=pnts.b1},
51  }
```

The X3R shock tube – let flow develop for 3ms

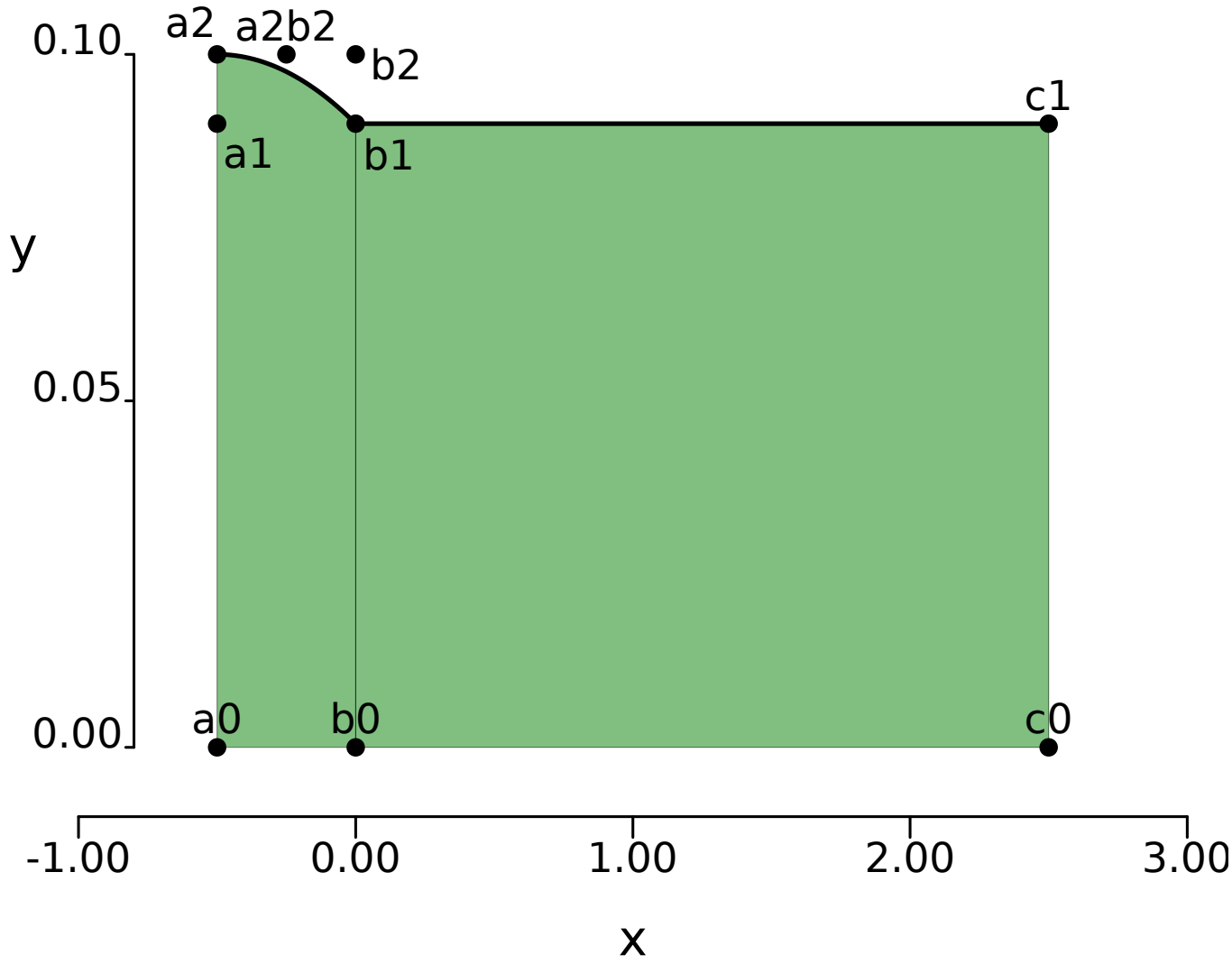


The X3R shock tube – use a Busemann inlet



- Sannu Mölder and Evgeny Timofeev "Hypersonic air-intake design for high performance and starting" RTO-EN-ATV-195

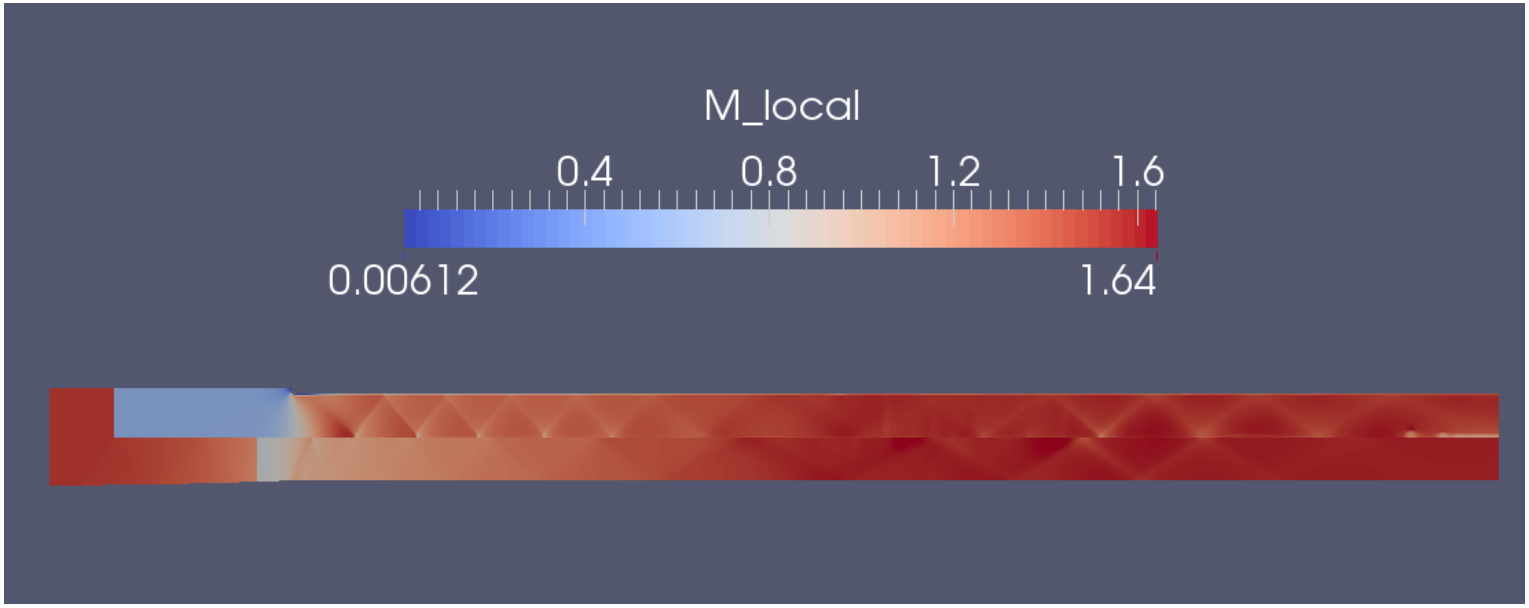
The X3R shock tube – contraction as a Bezier curve

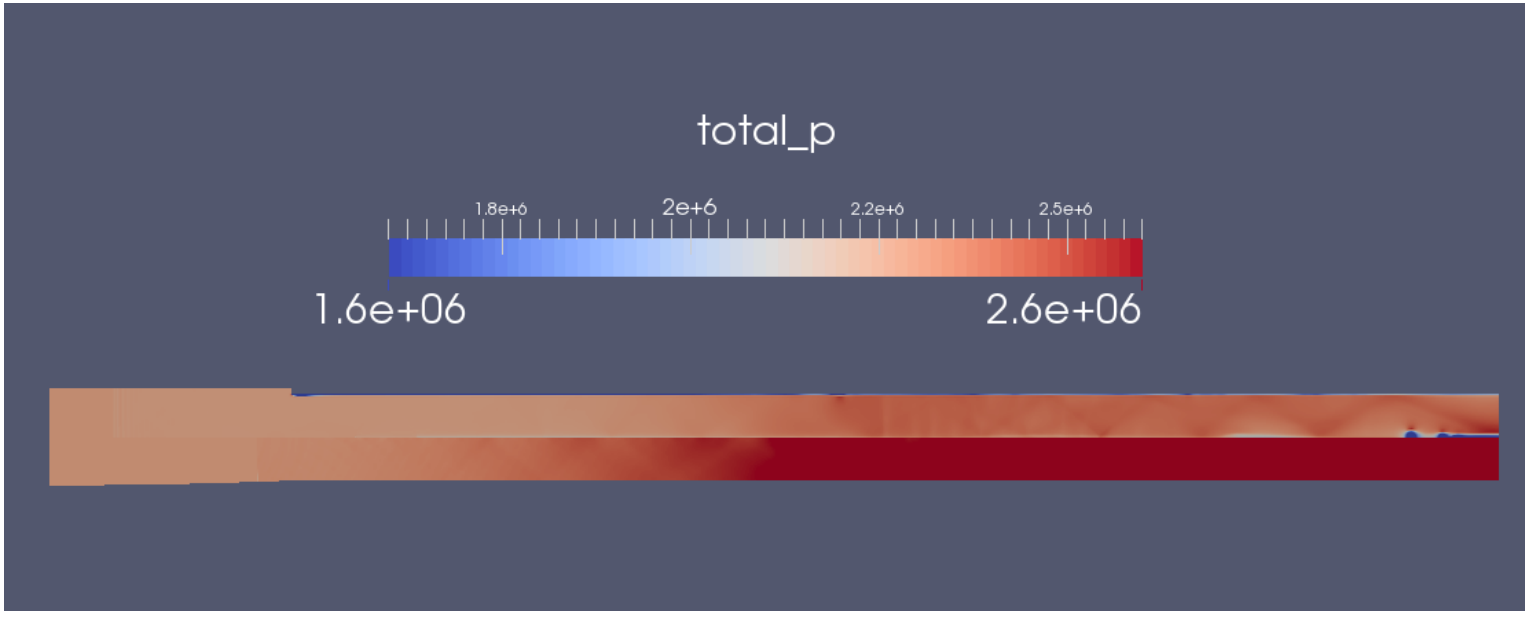


The X3R shock tube – defining the better flow domain

```
36  pnts = {
37     a0 = Vector3:new{x=-L1, y=0},
38     a1 = Vector3:new{x=-L1, y=R2},
39     a2 = Vector3:new{x=-L1, y=R1},
40     b0 = Vector3:new{x=0, y=0.0},
41     b1 = Vector3:new{x=0, y=R2},
42     b2 = Vector3:new{x=0, y=R1},
43     a2b2 = Vector3:new{x=-0.5*L1, y=R1},
44     c0 = Vector3:new{x=L2, y=0},
45     c1 = Vector3:new{x=L2, y=R2}
46  }
47
48  lines = {
49     nth = Bezier:new{points={pnts.a2, pnts.a2b2, pnts.b1}},
50     sth = Line:new{p0=pnts.a0, p1=pnts.b0},
51     wst = Line:new{p0=pnts.a0, p1=pnts.a2},
52     est = Line:new{p0=pnts.b0, p1=pnts.b1}
53  }
54
55  patches = {
56     surf0 = CoonsPatch:new{north=lines.nth, south=lines.sth,
57                            west=lines.wst, east=lines.est},
58     surf2 = CoonsPatch:new{p00=pnts.b0, p10=pnts.c0, p11=pnts.c1, p01=pnts.b1},
59  }
```

The X3R shock tube – let flow develop for 3ms





Concluding remark from Han Wei, NUS.