Automated finite difference modelling on structured grids, and a variety of compute architectures

Satya P Jammy, Christian T Jacobs and Neil D Sandham

s.p.jammy@soton.ac.uk

UK Fluids Conference, September, 2016

UNIVERSITY OF Southampton

Motivation

- SBLI is a legacy CFD code developed at the University of Southampton
- Written in Fortran90, for solving the compressible Navier-Stokes equations on block structured grids
- Fourth order central differencing for spatial discretisation
- Currently capable of running on CPU clusters
- However, the architectures are evolving rapidly.

Motivation

UNIVERSITY OF Southampton

- Current architectures include, Multi-core CPU's, GPU's, XeonPhi cards
- Future architectures: Energy efficient systems from ARM and others
- Porting existing SBLI code on these architectures requires a complete rewrite and challenging
- Newer architecture might arrive while we are still porting to the current architectures

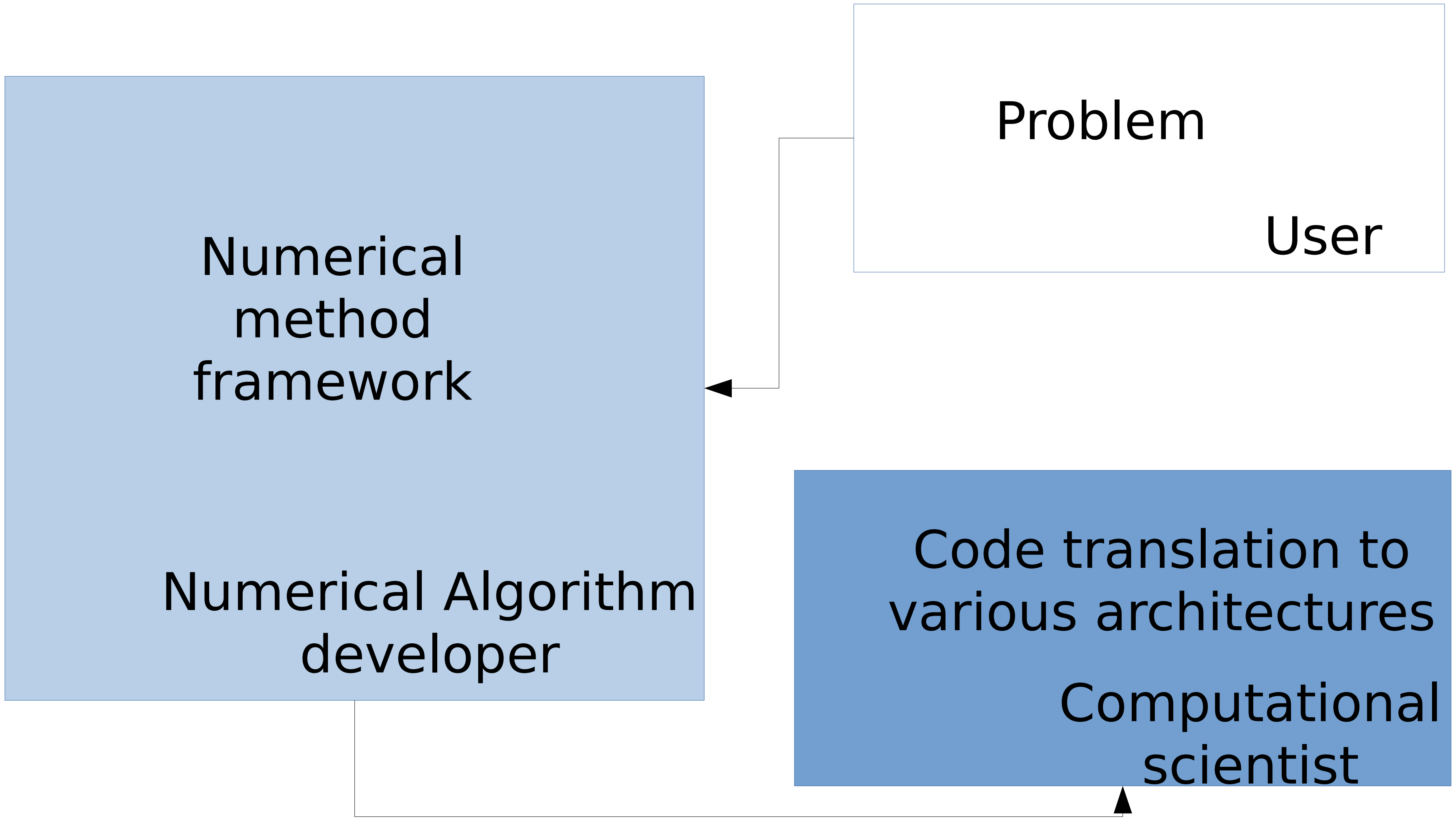
Ways to future proof codes?

Approach

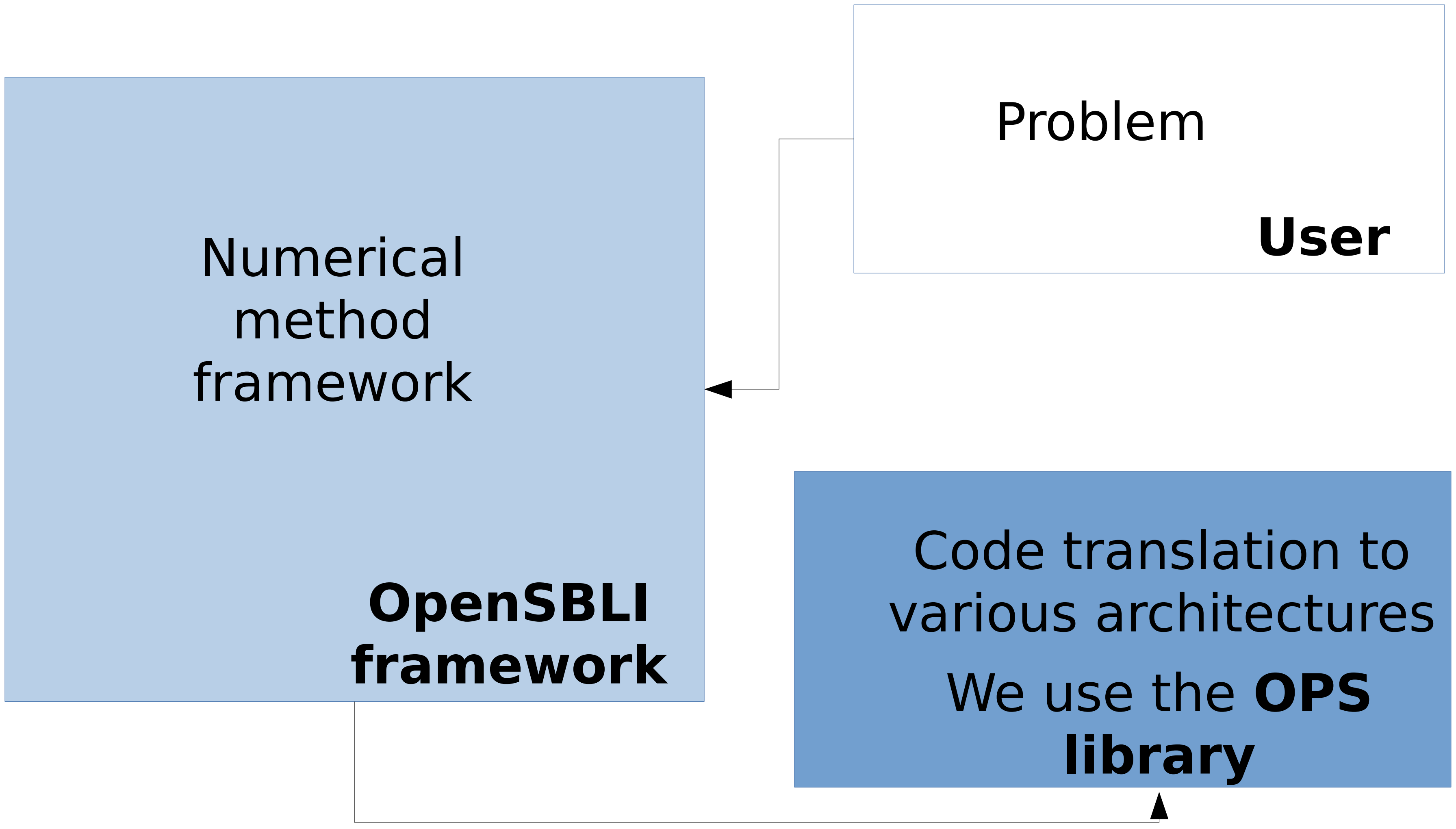
- Key components of numerical solution are
    - Problem description
    - Numerical method
    - Writing code
- One way to address future proofing is through separation of concerns using high level abstractions

Approach

UNIVERSITY OF Southampton


Problem
User
Numerical method framework
Numerical Algorithm developer
Code translation to various architectures
Computational scientist

Approach


Problem
User
Numerical method framework
OpenSBLI framework
Code translation to various architectures
We use the **OPS library**

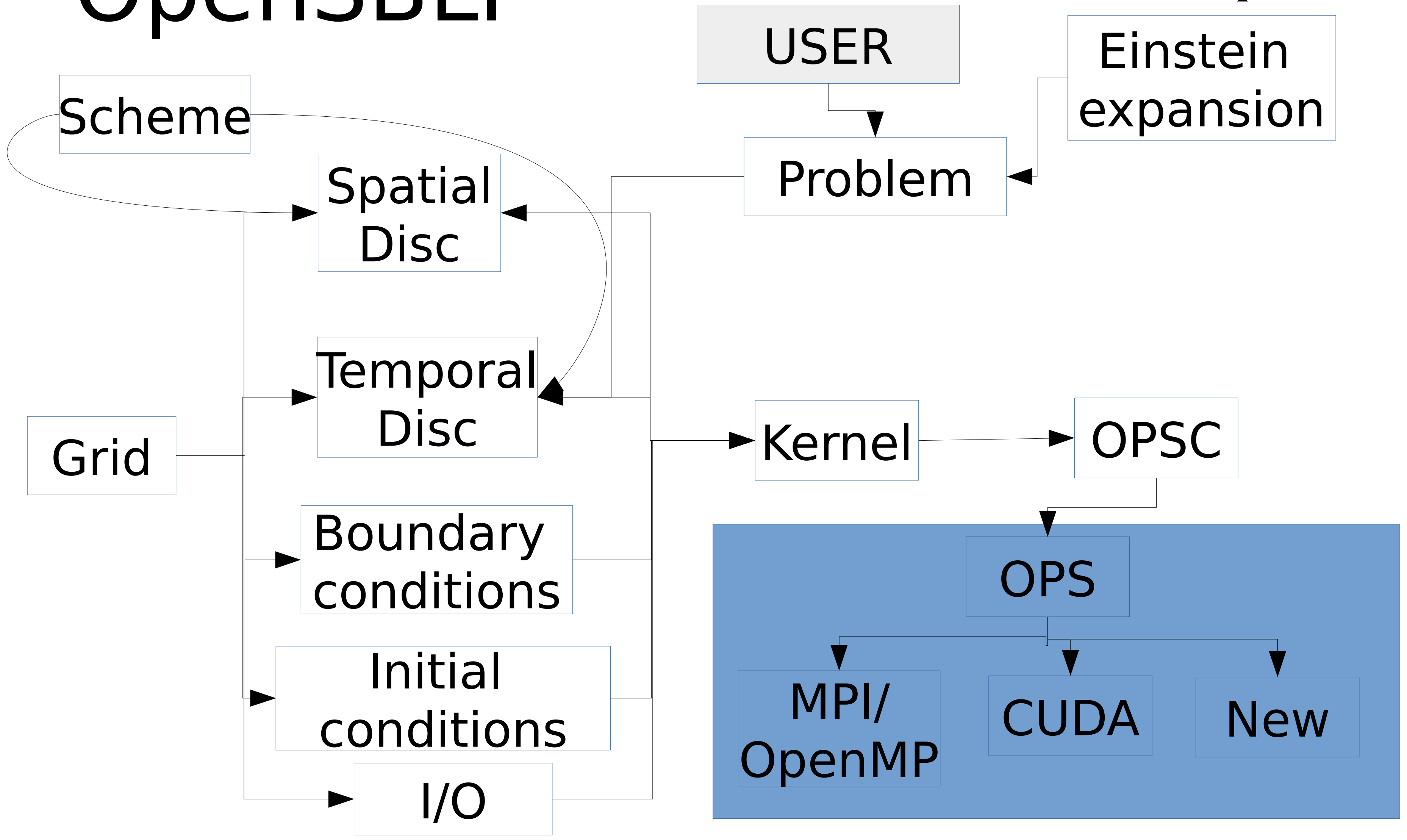
UNIVERSITY OF Southampton

OpenSBLI

- Written in Python and uses SymPy for the numerical method
- User specifies
    - Equations in Einstein notation
    - Order of accuracy of the spatial scheme (central finite differences)
    - Time stepping scheme
    - Boundary conditions
    - Initial conditions
- Output is C code that can be compiled on various architectures

OpenSBLI

UNIVERSITY OF Southampton



OpenSBLI

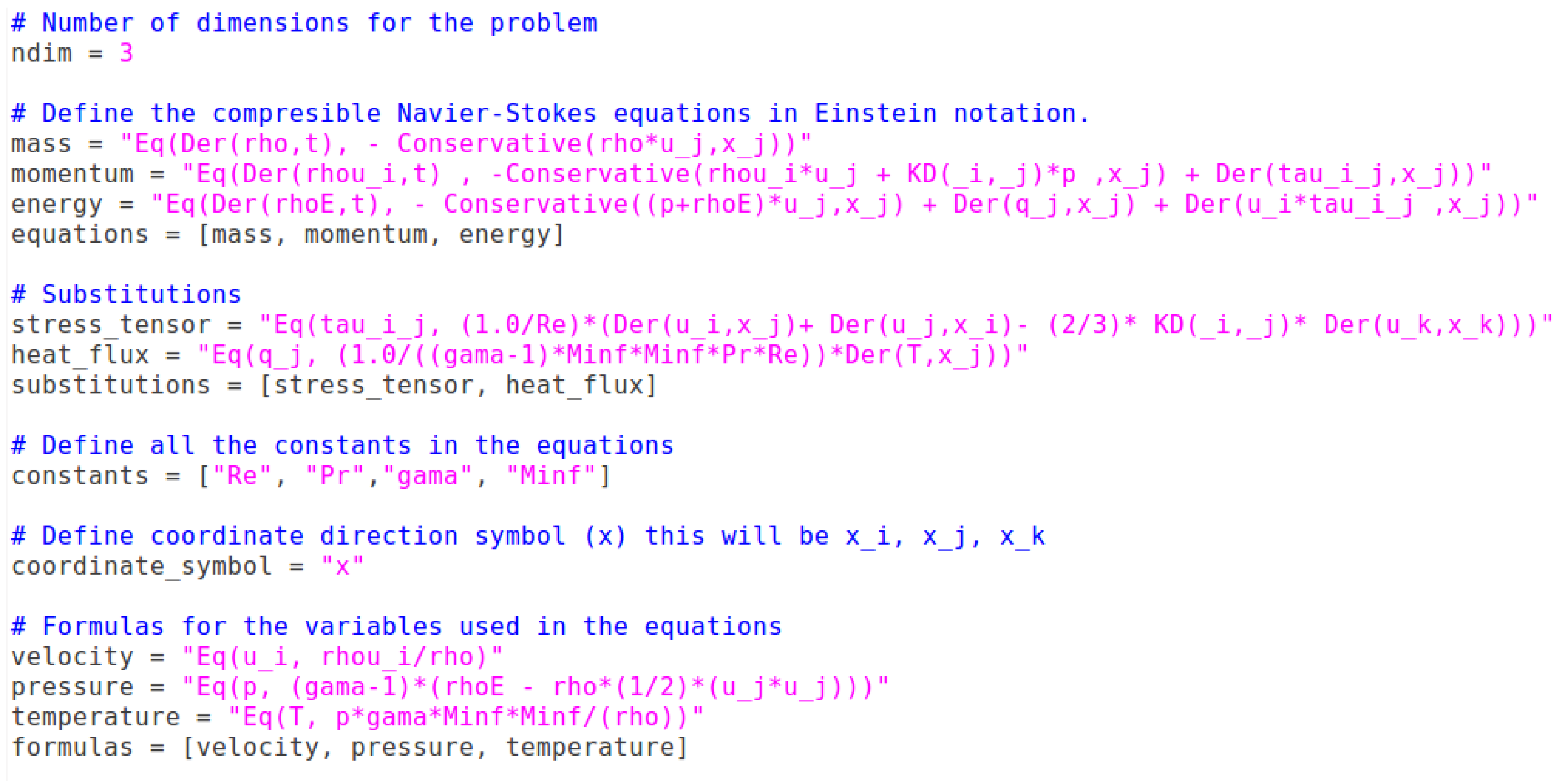
UNIVERSITY OF Southampton

- The OpenSBLI framework has the following additional advantages
  - Flexible choice of equations
  - Spatial order can be easily varied
  - Models need not require a rewrite for new architectures
- For eg, a 50 line high-level problem definition, for the 3D compressible N-S equations, results in 20K lines of generated code for MPI and CUDA
- Details of OpenSBLI framework can be found in http://arxiv.org/abs/1609.01277

OpenSBLI

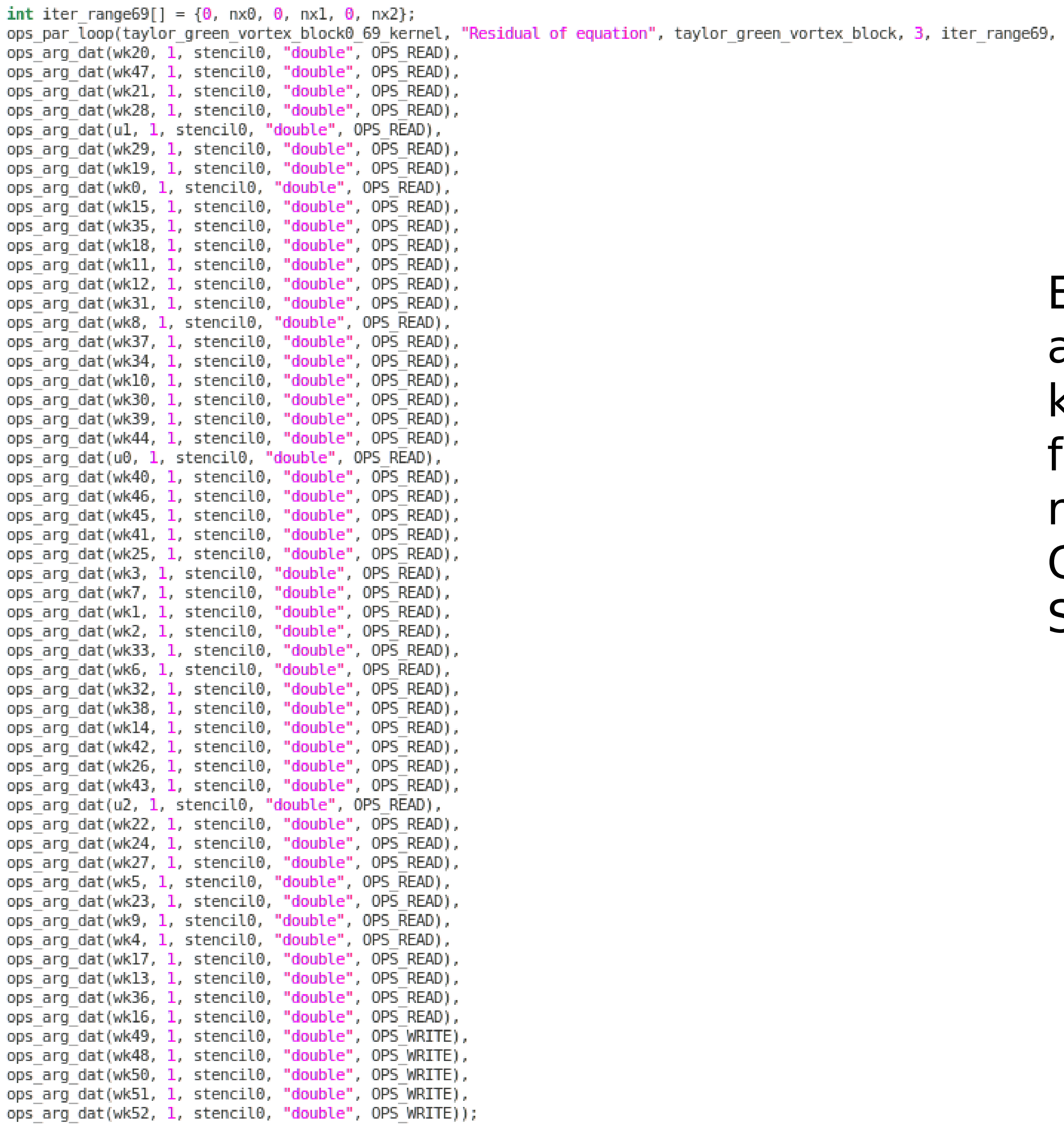
UNIVERSITY OF Southampton

Example



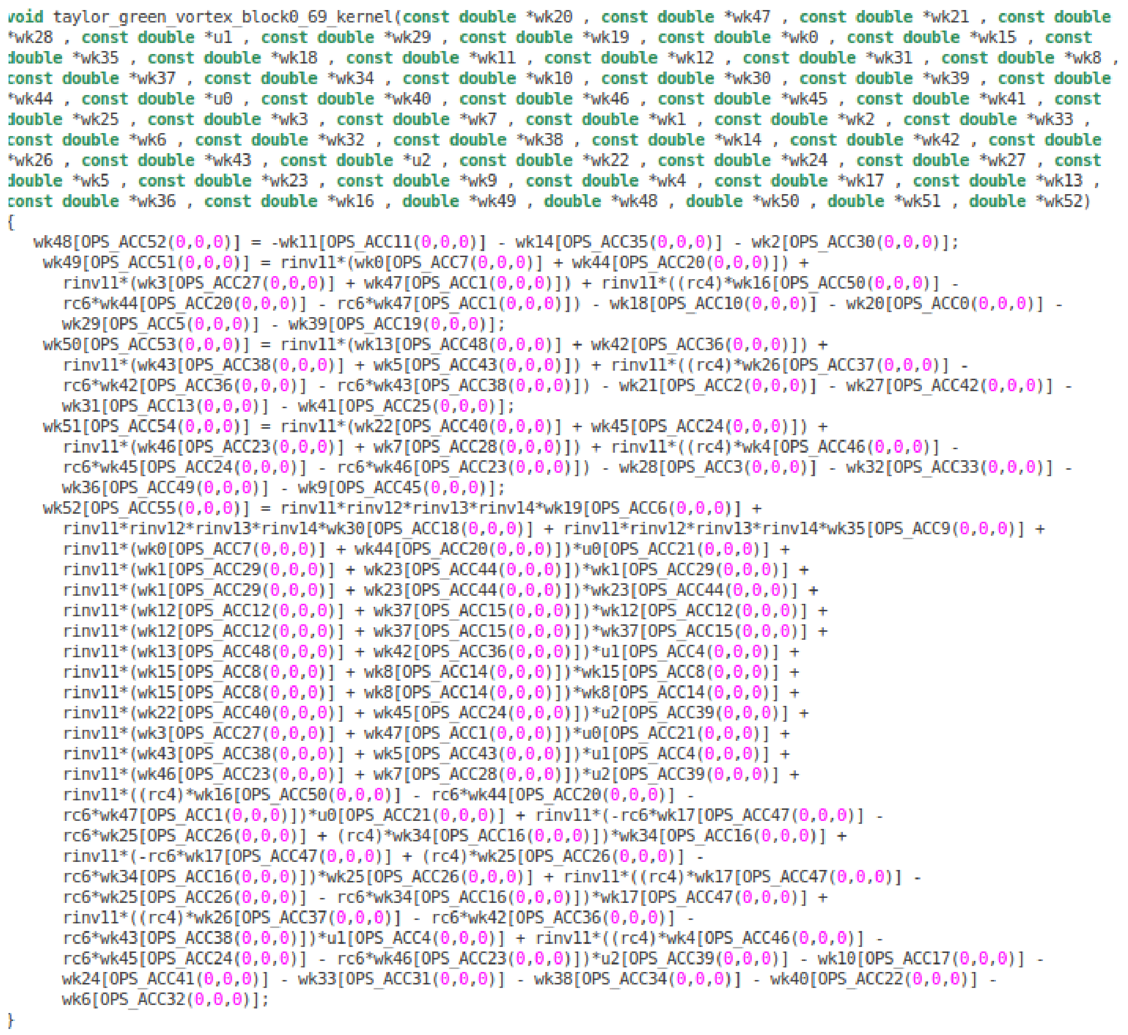
```
# Number of dimensions for the problem
ndim = 3

# Define the compresible Navier-Stokes equations in Einstein notation.
mass = "Eq(Der(rho,t), - Conservative(rho*u_j,x_j))"
momentum = "Eq(Der(rhou_i,t) , -Conservative(rhou_i*u_j + KD(_i,_j)*p ,x_j) + Der(tau_i_j,x_j))"
energy = "Eq(Der(rhoE,t), - Conservative((p+rhoE)*u_j,x_j) + Der(q_j,x_j) + Der(u_i*tau_i_j ,x_j))"
equations = [mass, momentum, energy]

# Substitutions
stress_tensor = "Eq(tau_i_j, (1.0/Re)*(Der(u_i,x_j)+ Der(u_j,x_i)- (2/3)* KD(_i,_j)* Der(u_k,x_k)))"
heat_flux = "Eq(q_j, (1.0/((gama-1)*Minf*Minf*Pr*Re))*Der(T,x_j))"
substitutions = [stress_tensor, heat_flux]

# Define all the constants in the equations
constants = ["Re", "Pr","gama", "Minf"]

# Define coordinate direction symbol (x) this will be x_i, x_j, x_k
coordinate_symbol = "x"

# Formulas for the variables used in the equations
velocity = "Eq(u_i, rhou_i/rho)"
pressure = "Eq(p, (gama-1)*(rhoE - rho*(1/2)*(u_j*u_j)))"
temperature = "Eq(T, p*gama*Minf*Minf/(rho))"
formulas = [velocity, pressure, temperature]
```

UNIVERSITY OF Southampton



```
int iter_range69[] = {0, nx0, 0, nx1, 0, nx2};
ops_par_loop(taylor_green_vortex_block0_69_kernel, "Residual of equation", taylor_green_vortex_block, 3, iter_range69,
ops_arg_dat(wk20, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk47, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk21, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk28, 1, stencil0, "double", OPS_READ),
ops_arg_dat(u1, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk29, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk19, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk0, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk15, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk35, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk18, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk11, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk12, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk31, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk8, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk37, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk34, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk10, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk30, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk39, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk44, 1, stencil0, "double", OPS_READ),
ops_arg_dat(u0, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk40, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk46, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk45, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk41, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk25, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk3, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk7, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk1, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk2, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk33, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk6, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk32, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk38, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk14, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk42, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk26, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk43, 1, stencil0, "double", OPS_READ),
ops_arg_dat(u2, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk22, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk24, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk27, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk5, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk23, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk9, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk4, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk17, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk13, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk36, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk16, 1, stencil0, "double", OPS_READ),
ops_arg_dat(wk49, 1, stencil0, "double", OPS_WRITE),
ops_arg_dat(wk48, 1, stencil0, "double", OPS_WRITE),
ops_arg_dat(wk50, 1, stencil0, "double", OPS_WRITE),
ops_arg_dat(wk51, 1, stencil0, "double", OPS_WRITE),
ops_arg_dat(wk52, 1, stencil0, "double", OPS_WRITE));
```

Example of auto-generated kernel call for computing right hand side of Compressible Navier-Stokes solution



```
void taylor_green_vortex_block0_69_kernel(const double *wk20 , const double *wk47 , const double *wk21 , const double
*wk28 , const double *u1 , const double *wk29 , const double *wk19 , const double *wk0 , const double *wk15 , const
double *wk35 , const double *wk18 , const double *wk11 , const double *wk12 , const double *wk31 , const double *wk8 ,
const double *wk37 , const double *wk34 , const double *wk10 , const double *wk30 , const double *wk39 , const double
*wk44 , const double *u0 , const double *wk40 , const double *wk46 , const double *wk45 , const double *wk41 , const
double *wk25 , const double *wk3 , const double *wk7 , const double *wk1 , const double *wk2 , const double *wk33 ,
const double *wk6 , const double *wk32 , const double *wk38 , const double *wk14 , const double *wk42 , const double
*wk26 , const double *wk43 , const double *u2 , const double *wk22 , const double *wk24 , const double *wk27 , const
double *wk5 , const double *wk23 , const double *wk9 , const double *wk4 , const double *wk17 , const double *wk13 ,
const double *wk36 , const double *wk16 , double *wk49 , double *wk48 , double *wk50 , double *wk51 , double *wk52)
{
   wk48[OPS_ACC52(0,0,0)] = -wk11[OPS_ACC11(0,0,0)] - wk14[OPS_ACC35(0,0,0)] - wk2[OPS_ACC30(0,0,0)];
    wk49[OPS_ACC51(0,0,0)] = rinv11*(wk0[OPS_ACC7(0,0,0)] + wk44[OPS_ACC20(0,0,0)]) +
      rinv11*(wk3[OPS_ACC27(0,0,0)] + wk47[OPS_ACC1(0,0,0)]) + rinv11*((rc4)*wk16[OPS_ACC50(0,0,0)] -
      rc6*wk44[OPS_ACC20(0,0,0)] - rc6*wk47[OPS_ACC1(0,0,0)]) - wk18[OPS_ACC10(0,0,0)] - wk20[OPS_ACC0(0,0,0)] -
      wk29[OPS_ACC5(0,0,0)] - wk39[OPS_ACC19(0,0,0)];
    wk50[OPS_ACC53(0,0,0)] = rinv11*(wk13[OPS_ACC48(0,0,0)] + wk42[OPS_ACC36(0,0,0)]) +
      rinv11*(wk43[OPS_ACC38(0,0,0)] + wk5[OPS_ACC43(0,0,0)]) + rinv11*((rc4)*wk26[OPS_ACC37(0,0,0)] -
      rc6*wk42[OPS_ACC36(0,0,0)] - rc6*wk43[OPS_ACC38(0,0,0)]) - wk21[OPS_ACC2(0,0,0)] - wk27[OPS_ACC42(0,0,0)] -
      wk31[OPS_ACC13(0,0,0)] - wk41[OPS_ACC25(0,0,0)];
    wk51[OPS_ACC54(0,0,0)] = rinv11*(wk22[OPS_ACC40(0,0,0)] + wk45[OPS_ACC24(0,0,0)]) +
      rinv11*(wk46[OPS_ACC23(0,0,0)] + wk7[OPS_ACC28(0,0,0)]) + rinv11*((rc4)*wk4[OPS_ACC46(0,0,0)] -
      rc6*wk45[OPS_ACC24(0,0,0)] - rc6*wk46[OPS_ACC23(0,0,0)]) - wk28[OPS_ACC3(0,0,0)] - wk32[OPS_ACC33(0,0,0)] -
      wk36[OPS_ACC49(0,0,0)] - wk9[OPS_ACC45(0,0,0)];
    wk52[OPS_ACC55(0,0,0)] = rinv11*rinv12*rinv13*rinv14*wk19[OPS_ACC6(0,0,0)] +
      rinv11*rinv12*rinv13*rinv14*wk30[OPS_ACC18(0,0,0)] + rinv11*rinv12*rinv13*rinv14*wk35[OPS_ACC9(0,0,0)] +
      rinv11*(wk0[OPS_ACC7(0,0,0)] + wk44[OPS_ACC20(0,0,0)])*u0[OPS_ACC21(0,0,0)] +
      rinv11*(wk1[OPS_ACC29(0,0,0)] + wk23[OPS_ACC44(0,0,0)])*wk1[OPS_ACC29(0,0,0)] +
      rinv11*(wk1[OPS_ACC29(0,0,0)] + wk23[OPS_ACC44(0,0,0)])*wk23[OPS_ACC44(0,0,0)] +
      rinv11*(wk12[OPS_ACC12(0,0,0)] + wk37[OPS_ACC15(0,0,0)])*wk12[OPS_ACC12(0,0,0)] +
      rinv11*(wk12[OPS_ACC12(0,0,0)] + wk37[OPS_ACC15(0,0,0)])*wk37[OPS_ACC15(0,0,0)] +
      rinv11*(wk13[OPS_ACC48(0,0,0)] + wk42[OPS_ACC36(0,0,0)])*u1[OPS_ACC4(0,0,0)] +
      rinv11*(wk15[OPS_ACC8(0,0,0)] + wk8[OPS_ACC14(0,0,0)])*wk15[OPS_ACC8(0,0,0)] +
      rinv11*(wk15[OPS_ACC8(0,0,0)] + wk8[OPS_ACC14(0,0,0)])*wk8[OPS_ACC14(0,0,0)] +
      rinv11*(wk22[OPS_ACC40(0,0,0)] + wk45[OPS_ACC24(0,0,0)])*u2[OPS_ACC39(0,0,0)] +
      rinv11*(wk3[OPS_ACC27(0,0,0)] + wk47[OPS_ACC1(0,0,0)])*u0[OPS_ACC21(0,0,0)] +
      rinv11*(wk43[OPS_ACC38(0,0,0)] + wk5[OPS_ACC43(0,0,0)])*u1[OPS_ACC4(0,0,0)] +
      rinv11*(wk46[OPS_ACC23(0,0,0)] + wk7[OPS_ACC28(0,0,0)])*u2[OPS_ACC39(0,0,0)] +
      rinv11*((rc4)*wk16[OPS_ACC50(0,0,0)] - rc6*wk44[OPS_ACC20(0,0,0)] -
      rc6*wk47[OPS_ACC1(0,0,0)])*u0[OPS_ACC21(0,0,0)] + rinv11*(-rc6*wk17[OPS_ACC47(0,0,0)] -
      rc6*wk25[OPS_ACC26(0,0,0)] + (rc4)*wk34[OPS_ACC16(0,0,0)])*wk34[OPS_ACC16(0,0,0)] +
      rinv11*(-rc6*wk17[OPS_ACC47(0,0,0)] + (rc4)*wk25[OPS_ACC26(0,0,0)] -
      rc6*wk34[OPS_ACC16(0,0,0)])*wk25[OPS_ACC26(0,0,0)] + rinv11*((rc4)*wk17[OPS_ACC47(0,0,0)] -
      rc6*wk25[OPS_ACC26(0,0,0)] - rc6*wk34[OPS_ACC16(0,0,0)])*wk17[OPS_ACC47(0,0,0)] +
      rinv11*((rc4)*wk26[OPS_ACC37(0,0,0)] - rc6*wk42[OPS_ACC36(0,0,0)] -
      rc6*wk43[OPS_ACC38(0,0,0)])*u1[OPS_ACC4(0,0,0)] + rinv11*((rc4)*wk4[OPS_ACC46(0,0,0)] -
      rc6*wk45[OPS_ACC24(0,0,0)] - rc6*wk46[OPS_ACC23(0,0,0)])*u2[OPS_ACC39(0,0,0)] - wk10[OPS_ACC17(0,0,0)] -
      wk24[OPS_ACC41(0,0,0)] - wk33[OPS_ACC31(0,0,0)] - wk38[OPS_ACC34(0,0,0)] - wk40[OPS_ACC22(0,0,0)] -
      wk6[OPS_ACC32(0,0,0)];
}
```

Example of auto-generated kernel for computing residual of Compressible Navier-Stokes solution

Results

- MMS tests for the 2D advection-diffusion equation

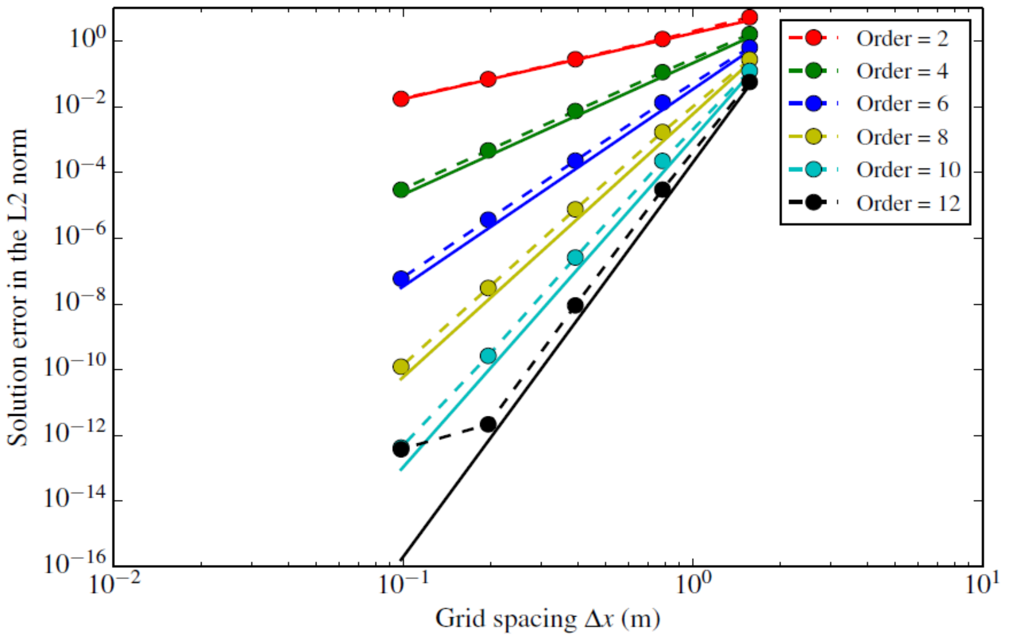


Figure 7: The absolute error (in the L2 norm) between the numerical solution ϕ and the exact/manufactured solution ϕ_m, from the suite of MMS simulations. The solid lines represent the expected convergence rate for each order.

Demonstration of solution convergence. Considering up to order 12. Image by Jacobs et al. (Submitted): http://arxiv.org/abs/1609.01277

Results

UNIVERSITY OF Southampton

- Compressible Navier-Stokes equations
- 3D Taylor-Green vortex problem
- Up to 1 billion grid points
- Re = 1600
- Architectures
  - CPU (ARCHER)
  - GPU (K40)

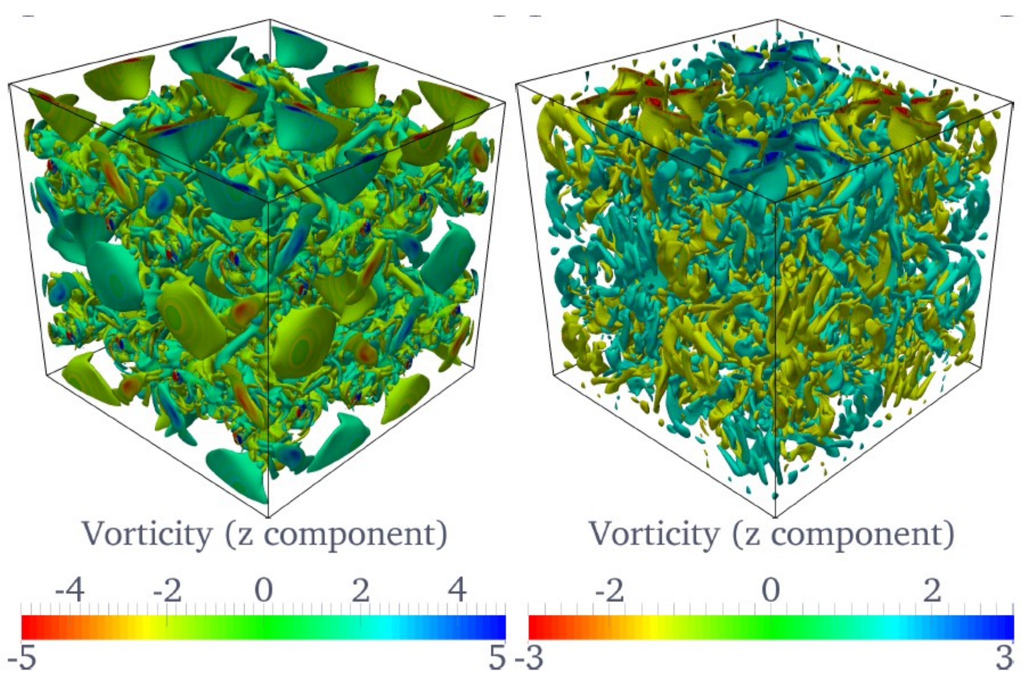


Figure 8: Visualisations of the non-dimensional vorticity (z-component) iso-contours, from the Taylor-Green vortex test case with a 256^3 grid, at various non-dimensional times. Top left to bottom right: non-dimensional time $t = 0, 2.5, 10, 20$.

Results from a Taylor-Green vortex test case. Image by Jacobs et al. (Submitted): http://arxiv.org/abs/1609.01277

Conclusions

- A new framework for the automated solution of finite difference methods on various architectures is developed and validated
- For easy debugging the framework writes the computations in Latex
- A 50 line high-level problem definition, for the compressible N-S equations, results in 20K lines of generated code for MPI and CUDA
- Separation of concerns enables better model maintainability, and future proofs the code as newer architectures arrive
- New algorithms and numerical methods can be readily implemented using the framework