eCSE09-04: CVODE support for FEniCS: Technical Report

Chris Richardson
University of Cambridge
chris@bpi.cam.ac.uk

Christoforos Hadjigeorgiou
University of Cambridge
ch741@cam.ac.uk

June 29, 2018

Abstract

This project interfaced the SUNDIALS CVODE time integration package with FEniCS, a widely used open-source finite element framework. A target application exploiting the functionality developed in this project is micromagnetic simulations. The new developments allow users to leverage parallel computation for applicationsthat were previously limited to serial implementations. Development took place on ARCHER (EPCC) and CSD3 (Cambridge), and packages that include the new developments are available on both systems. *This work was funded under the embedded CSE (eCSE) programme of the ARCHER UK National Supercomputing Service (`http://www.archer.ac.uk`)*

1 Project objectives

The objective of this project was to interface the SUNDIALS CVODE [2] time stepping package to the open-source FEniCS Project libraries [3], supported by concrete application to the field of micromagnetics. SUNDIALS is written in C, whilst FEniCS has C++ and Python interfaces. The code developed in this project allows the FEniCS user to access SUNDIALS CVODE interface from C++ or Python.

In the field of micromagnetics, it is often necessary to integrate equations in time which have different characteristic timescales, and this requires sophisticated time-stepping tools, such as CVODE from SUNDIALS. In this report, we outline the demonstration implementations for two examples: a diffusion problem in 3D, and a micromagnetics application.

2 FEniCS interface to CVODE

FEniCS 2018.1.0 can now be built with SUNDIALS CVODE support by compiling with the `SUNDIALS_DIR` environment variable set to the directory containing an installation of SUNDIALS (version 3.0.0 or later). When built with SUNDIALS support, the `CVode` class becomes available to use in C++ or Python. The examples shown here will focus on Python, as this is the interface of choice for the magnetics community.

FEniCS with SUNDIALS/CVODE support is available on ARCHER (module temporarily at `/work/ecse0904/shared/FEniCS/fenics-module`), and CSD3 (use `module load fenics`).

We provide here a brief description of the Python user interface. The `CVode` class must be subclassed, to define the `derivs()` method, which provides the time derivatives which we wish to integrate. In the example below, `CVode` is subclassed as `MyCVode`, and is instantiated with a choice of Adams or Newton methods for time-stepping, and functional or BDF for iteration. See the CVODE documentation [1] for details. A vector (usually from a FEniCS `Function`) is attached to the `CVode` instance with the `init()` method, along with absolute and relative tolerances. The initial time is assumed to be zero, though this can also be set manually. Subsequently, it is just a matter of calling `step()` with the desired timestep, to move the equations forward. A simple example for the diffusion equation is shown below.

```
from dolfin import *

nx = 12
mesh = UnitCubeMesh(MPI.comm_world, nx, nx, nx)
Q = FunctionSpace(mesh, "CG", 1)
phi = Function(Q)

gaussian = Expression(
 "exp(-a*(pow(x[0]-0.5, 2) + pow(x[1]-0.5, 2) + pow(x[2]-0.5,2)))",
 a=20., degree=1)
phi.interpolate(gaussian)
v = TestFunction(Q)
u = TrialFunction(Q)
a = u*v*dx
A = assemble(a)
Adiag = PETScVector()
A.init_vector(Adiag, 0)
Adiag[:] = 1.0
Adiag = A*Adiag
ufn = Function(Q)

class MyCVode(CVode):
    def derivs(self, t, u, udot):
        uvec = ufn.vector()
        uvec[:] = u[:]
        w = assemble(-dot(grad(ufn), grad(v))*dx)
        udot[:] = w[:]/Adiag[:]

cv = MyCVode(CVode.LMM.CV_ADAMS, CVode.ITER.CV_FUNCTIONAL)
cv.init(phi.vector(), 1e-6, 1e-3)

vfile = XDMFFile(mesh.mpi_comm(), filename)

nstep = 50
dt = 0.001
vfile.write(phi, 0.0)
for i in range(nstep):
    t = cv.step(dt)
    vfile.write(phi, t)
```

The above diffusion demo is set up with an initial Gaussian peak in the centre of a cube, which spreads out over time according to the equation:

$$\frac{\partial u}{\partial t} = \nabla^2 u$$

The results are shown in Figure 1, on a mesh with 23 665 872 cells, running on 256 cores on CSD3. Timings are shown in Figure 2. Weak scaling for explicit time stepping is difficult to demonstrate as the number of time steps (and hence derivative evaluations) increases with mesh refinement in order to maintain stability, so the amount of work increases as the problem size increases, even with the same number of degrees of freedom per node. To show the reasonableness of the weak scaling, we also present the total time spent in the timestepping code divided by the number of evaluations, and it remains constant as the problem size is increased.

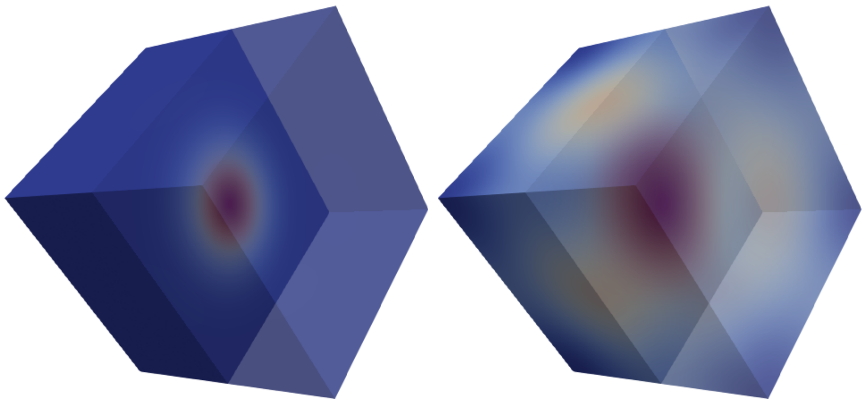

Figure 1: Diffusion demo, showing (a) initial state, (b) state after 50 steps with CVODE

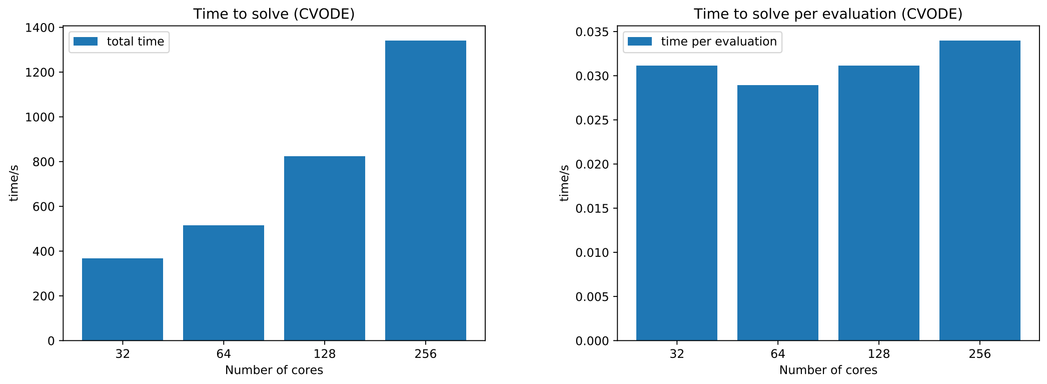

Figure 2: Run time for diffusion equation with CVODE on CSD3, weak scaling. The number of degrees-of-freedom is fixed at 16 000 per core, so that the largest problem (on 256 cores) is of size 4 096 000. Increasing the problem size increases the number of derivative evaluations. The right hand plot shows the time *per evaluation*.

Strong scaling can also be demonstrated by taking a large model ($100 \times 100 \times 100$ cube) on 32 cores, and solving the same problem on more processors. The results are shown in Figure 3. The ideal scaling is shown in orange, and the actual times in blue. Efficiency decreases as the core count increases, as the ratio of communication to computation increases. However, most of the time is spent in the user-defined `derivs()` callback, which calculates the derivatives. Future work could look at reducing the amount of communication in this step, and increasing the efficiency, although this is outside the scope of the current project.

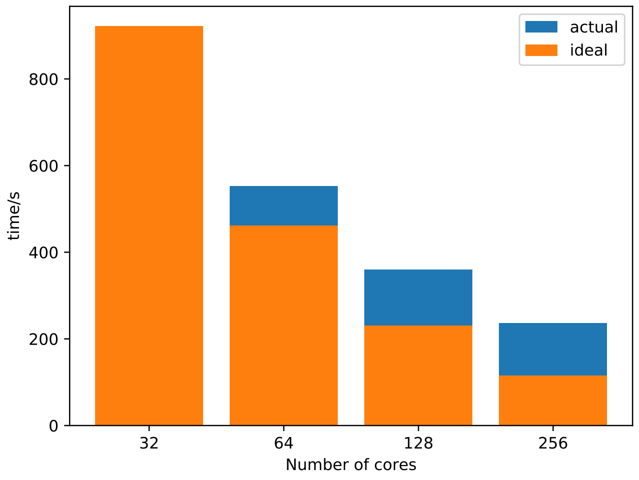

Figure 3: Strong scaling of the diffusion problem on CSD3: solve time in CVODE for a $100 \times 100 \times 100$ cube problem. Using the 32 core result as a baseline, the actual and ideal runtimes are shown for up to 256 cores

3 Micromagnetics

A typical magnetics problem involves alignment of the magnetisation vector of a sample to an applied field. In this process, the magnetisation processes around the direction of the applied field, ultimately aligning with it. The magnetisation $\mathbf{M}$ evolves according to the following equation:

$$\frac{d\mathbf{M}}{dt} = -\frac{\gamma}{1+\alpha^2}\mathbf{M} \times \mathbf{H}_{\text{eff}} - \frac{\alpha\gamma}{1+\alpha^2}\mathbf{M} \times (\mathbf{M} \times \mathbf{H}_{\text{eff}})$$

where the effective field $\mathbf{H}_{\text{eff}}$ is also a known function of the magnetisation. There is an analytical solution for the simple case of alignment to a constant field (Figure 4), and the numerical results are in good agreement with the analytical solution.

A larger model was used for testing on HPC systems. A unit cube mesh of size $64 \times 64 \times 64$ was used to calculate the alignment of the magnetisation to an external field. A typical output (of a smaller version of the same model) is shown in Figure 5. The initial magnetisation vectors were set to random values, and their behaviour over time under an applied field was calculated. As with the diffusion example, strong scaling was demonstrated from 32 to 256 cores on CSD3 and from 24 to 96 cores on ARCHER, with the results shown in Figures 6 and 7. At first glance, the efficiency seems to be poor at higher core count, but closer inspection (right hand plot, Figure 6) shows that the majority of the time is not spent in CVODE, but in the user supplied `derivs()` method. More work needs to be done to improve the efficiency of this user supplied part of the code, which encapsulates the physics of the problem being solved. It should be stressed to end users that the efficiency of the method is only as good as the efficiency of the user-supplied code for the time derivatives. A similar picture emerged in tests on the KNL processors on CSD3 (Figure—8). Although we demonstrated operation up to 1024 cores, the performance had saturated by this point.

Time-stepping with the Newton Method

If information about the derivatives of the `derivs()` function is available, this can be used in the Newton method in CVODE (See [1, section 2.1]). Because we are only interested in parallel implementations, we have only implemented an interface to `CVSPILS` in CVODE. When

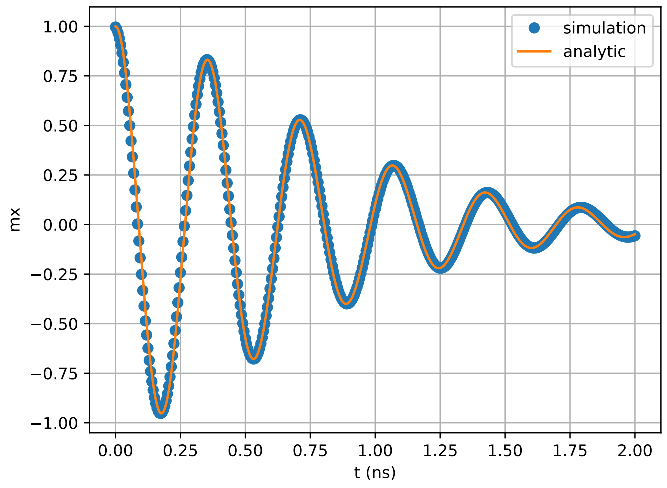

Figure 4: Analytic solution vs computed solution for field alignment using CVODE

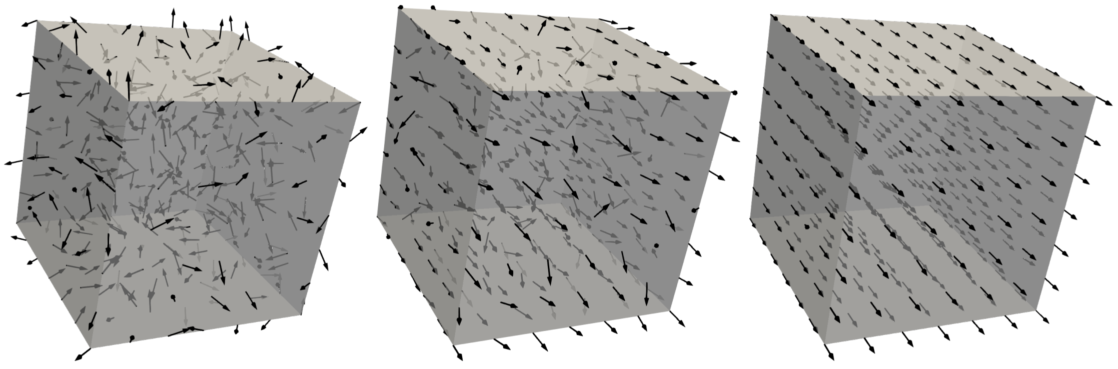

Figure 5: Magnetics demo, showing initial state, an intermediate step, and final state

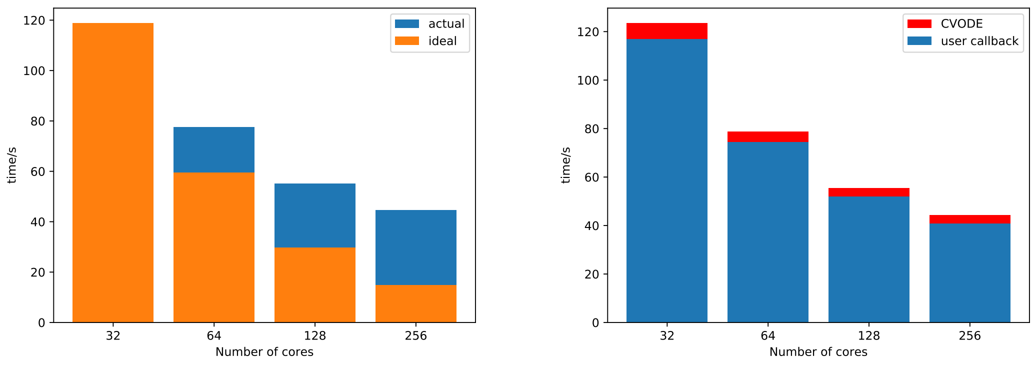

Figure 6: Strong scaling of magnetics demo on CSD3. On the left is shown the runtime of CVODE for four different core counts, and the ideal speedup (compared to the 32 core case). On the right is shown the breakdown between the time spent in CVODE, and the time spent in the user routine (`derivs()` above).

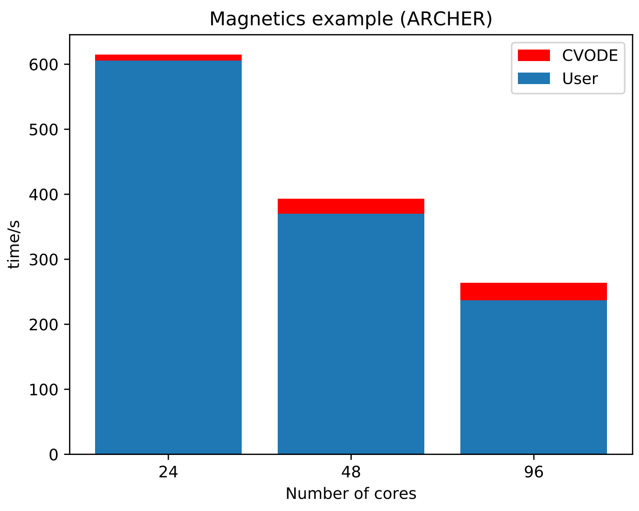

Figure 7: Strong scaling of magnetics demo on ARCHER. The breakdown is shown between the time spent in CVODE and the time spent in the user routine (`derivs()` above).

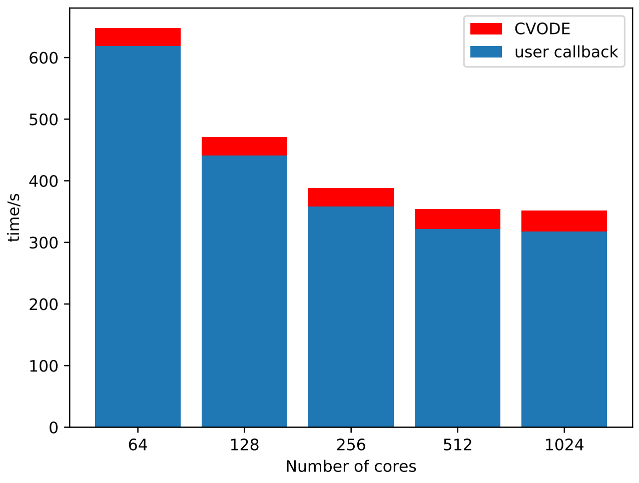

Figure 8: Strong scaling of magnetics demo on KNL processors (CSD3) showing the breakdown between user code and CVODE (as Figure 7).

using the Newton method, in the subclassed `CVode` class, one must also specify `jacobian()` and `psolve()` functions, providing the functions `CVSpilsJacTimesVecFn` and `CVSpilsPrecSolveFn` as described in the CVODE manual.

4 Conclusions

We have implemented an interface to CVODE for FEniCS, with user interfaces in C++ and Python. The new interface is fully integrated into the FEniCS release 2018.1.0, and will be available on platforms where FEniCS can be compiled with SUNDIALS 3.0.0 or later. Testing on CSD3 in Cambridge and ARCHER has shown good performance for the test problems – a diffusion problem, and magnetic alignment to an applied field.

References

[1] CVODE online documentation. `https://computation.llnl.gov/sites/default/files/public/cv_guide.pdf`.

[2] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.

[3] A. Logg, K.-A. Mardal, and G. N. Wells, editors. *Automated Solution of Differential Equations by the Finite Element Method*, volume 84 of *Lecture Notes in Computational Science and Engineering*. Springer, 2012.