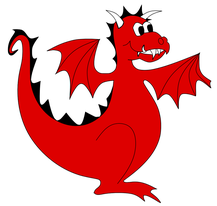
Imperial College London



Firedrake

$$\phi^{n+1/2} = \phi^n - \frac{\Delta t}{2} p^n$$

$$p^{n+1} = p^n + \frac{\int_\Omega \nabla \phi^{n+1/2} \cdot \nabla v \, \mathrm{d}x}{\int_\Omega v \, \mathrm{d}x} \qquad \forall v \in V$$

$$\phi^{n+1} = \phi^{n+1/2} - \frac{\Delta t}{2} p^{n+1}$$

where

$$\nabla \phi \cdot n = 0 \text{ on } \Gamma_N$$

$$p = \sin(10\pi t) \text{ on } \Gamma_D$$

```python
from firedrake import *
mesh = Mesh("wave_tank.msh")
V = FunctionSpace(mesh, 'Lagrange', 1)
p = Function(V, name="p")
phi = Function(V, name="phi")
u = TrialFunction(V)
v = TestFunction(V)
p_in = Constant(0.0)
bc = DirichletBC(V, p_in, 1)
T = 10.
dt = 0.001
t = 0
while t <= T:
    p_in.assign(sin(2*pi*5*t))
    phi -= dt / 2 * p
    p += assemble(dt * inner(grad(v), grad(phi))*dx) \
         / assemble(v*dx)
    bc.apply(p)
    phi -= dt / 2 * p
    t += dt
```

```
        LI_K_2_8[k] = (K[1] * LI_IPK_1_6[k]);
        LI_K_2_9[k] = (K[0] * LI_IPK_1_5[k]);
        LI_K_3_0[k] = ((((LI_C_1_0 * LI_K_2_0[k]) + (LI_C_1_0 * LI_K_2_8[k])) + (LI_C_1_0 * LI_K_2_3[k])) +
                       (LI_C_1_0 * LI_K_2_5[k])) + (LI_C_1_0 * LI_K_2_7[k])) + (LI_C_1_0 * LI_K_2_1[k])) +
                      ((LI_C_1_0 * LI_K_2_9[k]) + (LI_C_1_0 * LI_K_2_6[k])) * const0);


    }
    for (int j  = 0; j < 3; j += 4) {
#pragma vector aligned
        for (int k  = 0; k < 4; k += 4) {
            __m256d ymm8  = _mm256_load_pd(&LI_K_2_4[k]);
            __m256d ymm9  = _mm256_load_pd(&LI_J_2_7[j]);
            __m256d ymm6  = _mm256_load_pd(&LI_K_2_2[k]);
            __m256d ymm7  = _mm256_load_pd(&LI_J_2_5[j]);
            __m256d ymm4  = _mm256_load_pd(&LI_K_3_0[k]);
            __m256d ymm5  = _mm256_load_pd(&LI_J_3_0[j]);
            _mm256_store_pd(&A[j][k], _mm256_add_pd(_mm256_load_pd(&A[j][k]),
                                _mm256_add_pd(_mm256_mul_pd(ymm4, ymm5),
                                              _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
                                                            _mm256_mul_pd(ymm8, ymm9)))));
            ymm8 = _mm256_permute_pd(ymm8, 5);
            ymm6 = _mm256_permute_pd(ymm6, 5);
            ymm4 = _mm256_permute_pd(ymm4, 5);
            _mm256_store_pd(&A[j+1][k], _mm256_add_pd(_mm256_load_pd(&A[j*1+1][k]),
                                _mm256_add_pd(_mm256_mul_pd(ymm4, ymm5),
                                              _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
                                                            _mm256_mul_pd(ymm8, ymm9)))));
            ymm8 = _mm256_permute2f128_pd(ymm8, ymm8, 1);
            ymm6 = _mm256_permute2f128_pd(ymm6, ymm6, 1);
            ymm4 = _mm256_permute2f128_pd(ymm4, ymm4, 1);
            _mm256_store_pd(&A[j+2][k], _mm256_add_pd(_mm256_load_pd(&A[j*1+2][k]),
                                _mm256_add_pd(_mm256_mul_pd(ymm4, ymm5),
                                              _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
                                                            _mm256_mul_pd(ymm8, ymm9)))));
            ymm8 = _mm256_permute_pd(ymm8, 5);
            ymm6 = _mm256_permute_pd(ymm6, 5);
            ymm4 = _mm256_permute_pd(ymm4, 5);
            _mm256_store_pd(&A[j+3][k], _mm256_add_pd(_mm256_load_pd(&A[j*1+3][k]),
                                _mm256_add_pd(_mm256_mul_pd(ymm4, ymm5),
                                              _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
                                                            _mm256_mul_pd(ymm8, ymm9)))));
        }
    }
}
```

Automating the finite element method by composing abstractions

David A. Ham, Lawrence Mitchell, Florian Rathgeber, Andrew T.T. McRae, Gheorghe-Teodor Bercea, Fabio Luporini, Michael Lange, Miklós Homolya, Christian Jacobs, Paul H.J. Kelly

Department of Mathematics Department of Computing Department of Earth Science and Engineering Grantham Institute