

Firedrake

Automated simulation
right from the equations

$$\phi^{n+1/2} = \phi^n - \frac{\Delta t}{2} p^n$$
$$p^{n+1} = p^n + \frac{\int_{\Omega} \nabla \phi^{n+1/2} \cdot \nabla v \, dx}{\int_{\Omega} v \, dx} \quad \forall v \in V$$
$$\phi^{n+1} = \phi^{n+1/2} - \frac{\Delta t}{2} p^{n+1}$$

where

$$\nabla \phi \cdot n = 0 \text{ on } \Gamma_N$$
$$p = \sin(10\pi t) \text{ on } \Gamma_D$$

Firedrake allows mathematicians, scientists and engineers to work at the level of the equations.

```
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
```

Finite element simulations are specified in Python using high-level mathematical syntax.

```
LI_K_2_8[k] = (K[k] * LI_IPK_1_6[k]);
LI_K_2_9[k] = (K[k] * 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) {
        _mm256d ymm8 = _mm256_load_pd(&LI_K_2_4[k]);
        _mm256d ymm9 = _mm256_load_pd(&LI_J_2_7[j]);
        _mm256d ymm6 = _mm256_load_pd(&LI_K_2_2[k]);
        _mm256d ymm7 = _mm256_load_pd(&LI_J_2_5[j]);
        _mm256d ymm4 = _mm256_load_pd(&LI_K_3_0[k]);
        _mm256d 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(ymm8, ymm5),
            _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
            _mm256_mul_pd(ymm9, ymm4))));
        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][k]),
            _mm256_add_pd(_mm256_mul_pd(ymm8, ymm5),
            _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
            _mm256_mul_pd(ymm9, ymm4))));
        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+2][k]),
            _mm256_add_pd(_mm256_mul_pd(ymm8, ymm5),
            _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
            _mm256_mul_pd(ymm9, ymm4))));
        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+3][k]),
            _mm256_add_pd(_mm256_mul_pd(ymm8, ymm5),
            _mm256_add_pd(_mm256_mul_pd(ymm6, ymm7),
            _mm256_mul_pd(ymm9, ymm4))));
    }
}
```

High performance vectorised parallel code is generated, compiled and executed automatically on the fly as the simulation runs.

- ✓Limitless flexibility in specifying equations.
- ✓A huge range of sophisticated finite element spaces.
- ✓World-class linear and nonlinear solvers built in.
- ✓Seamless scaling from laptop to supercomputer.
- ✓Specialist support for high aspect ratio domains.

