AN ALGORITHM FOR THE OPTIMISATION OF FINITE ELEMENT INTEGRATION LOOPS

Fabio Luporini, David Ham, Paul Kelly

Imperial College London

18/04/2016

PRISM Workshop on Embracing Accelerators

MOTIVATION: AUTOMATED CODE GENERATION

Firedrake

UFL

TSFC

COFFEE

...

REDUCE FLOPS

~~LOW LEVEL OPT~~
~~e.g., VECTORISATION~~

SIMPLE OPERATOR (1): MASS MATRIX

Math (UFL)

dot(v, u)*dx

Loop nest

```
for (int ip  = 0; ip < m; ++ip) {
  for (int j  = 0; j < n; ++j) {
    for (int k  = 0; k < o; ++k) {
      A[j][k] += (det * W[ip] * B[ip][k] * B[ip][j]);
    }
  }
}
```

SIMPLE OPERATOR (2): HELMHOLTZ LHS

Math (UFL)

(v*u + dot(grad(v), grad(u)))*dx

Loop nest

```
for (int ip  = 0; ip < m; ++ip) {
  for (int j  = 0; j < n; ++j) {
    for (int k  = 0; k < o; ++k) {
      A[j][k] += (((B[ip][k] * B[ip][j]) + (((((K[2] * B0[ip][k]) + (K[5] * B1[ip][k]) + (K[8] * B2[ip][k])) * ((K[2] * B0[ip][j]) + (K[5] * B1[ip][j]) + (K[8] * B2[ip][j]))) + (((K[1] * B0[ip][k]) + (K[4] * B1[ip][k]) + (K[7] * B2[ip][k])) * ((K[1] * B0[ip][j]) + (K[4] * B1[ip][j]) + (K[7] * B2[ip][j]))) + (((K[0] * B0[ip][k]) + (K[3] * B1[ip][k]) + (K[6] * B2[ip][k])) * ((K[0] * B0[ip][j]) + (K[3] * B1[ip][j]) + (K[6] * B2[ip][j])))) * F1 * F0)) * det * W[ip]);
    }
  }
}
```

MORE COMPLEX OPERATOR: HYPERELASTICITY LHS

Math (UFL)

derivative((inner(F*diff(lmbda/2*(tr(((I + grad(u)).T*(I + grad(u)) - I)/2)**2) + mu*tr(((I + grad(u)).T*(I + grad(u)) - I)/2*((I + grad(u)).T*(I + grad(u)) - I)/2), ((I + grad(u)).T*(I + grad(u)) - I)/2), grad(v)) - inner(B, v))*dx, u, du)

Loop nest

```
for (int ip  = 0; ip < m; ++ip) {
  for (int j  = 0; j < n; ++j) {
    for (int k  = 0; k < o; ++k) {
         A[j][k] += ((((K[2] * BC10[0][j]) + (K[5] * BC11[0][j]) + (K[8] * BC12[0][j])) * (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((((((K[8] * F2) + (K[5] *
F1) + (K[2] * F0)) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] *
F3)) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0)) + (((((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] *
F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0))) * F9) + (((K[6] * F5) + (K[3] * F4) +
(K[0] * F3)) * ((((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k]))
* ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] *
BC01[0][k]) + (K[6] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))
+ (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0)) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0]
[k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] *
BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] * BC01[0][k]) + (K[6] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) +
(((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] *
F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0))) * F9) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((((((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] *
F4) + (K[0] * F3))) + (((K[6] * F8) + (K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] *
F0) + 1.0))) / 2.0)) + (((((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[6] * F8) + (K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] *
F6) + 1.0)) + (((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0))) * F9) + (((((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) *
((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0]
[k]) + (K[7] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] *
BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[7] * F5) + (K[4]
* F4) + (K[1] * F3) + 1.0))) / 2.0)) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k])
+ (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] *
..................[5] * F4) + (K[2] * F3))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) +
(K[8] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] *
BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))) / 2.0) + (((((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[7] *
F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) +
(K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * BC10[0]
[k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0)) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] * F5) + (K[4] * F4)
+ (K[1] * F3) + 1.0))) / 2.0) + ((((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6]
* BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC10[0][k]) +
(K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) ….
    }
  }
}
…
```

MORE COMPLEX OPERATOR: HYPERELASTICITY LHS

```
((((K[2] * BC10[0][j]) + (K[5] * BC11[0][j]) + (K[8] * BC12[0][j])) * (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((((((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[7] *
F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[7] * F5) + (K[4] *
F4) + (K[1] * F3) + 1.0))) / 2.0)) + (((((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] * F6)) * ((K[8] * F8) + (K[5] *
F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0))) * F9) + (((K[6] * F5) + (K[3] * F4) + (K[0] * F3)) * ((((((K[2] *
BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4)
+ (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] * BC01[0][k]) + (K[6] * BC02[0][k]))
* ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC00[0][k]) + (K[5] *
BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) +
(K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) *
((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] * BC01[0][k]) + (K[6] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0]
[k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0)))
/ 2.0))) * F9) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((((((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[6] * F8) +
(K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0) + (((((K[8] *
F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[6] * F8) + (K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F2) + (K[5] *
F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0))) * F9) + (((((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] *
F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[8] *
F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) +
(K[7] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) /
2.0)) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) *
((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * ..................[5] * F4) + (K[2] *
F3))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[8] *
F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC20[0][k]) + (K[5] * BC21[0][k])
+ (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))) / 2.0) + ((((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) +
(((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[7] * F2) +
(K[4] * F1) + (K[1] * F0))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] *
BC12[0][k])) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0)) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0) +
((((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) +
(K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] *
BC12[0][k])) * ((K[6] * F5) + ((((K[2] * BC10[0][j]) + (K[5] * BC11[0][j]) + (K[8] * BC12[0][j])) * (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((((((K[8] * F2) + (K[5] *
F1) + (K[2] * F0)) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] *
F3)) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0)) + (((((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[7] * F8) + (K[4] * F7) + (K[1] *
F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0))) * F9) + (((K[6] * F5) + (K[3] * F4) +
(K[0] * F3)) * ((((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k]))
* ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] *
BC01[0][k]) + (K[6] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))
+ (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0]
[k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[2] * BC10[0][k]) + (K[5] *
BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[0] * BC00[0][k]) + (K[3] * BC01[0][k]) + (K[6] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) +
(((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[6] *
F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0))) * F9) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((((((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] *
F4) + (K[0] * F3))) + (((K[6] * F8) + (K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] *
F0) + 1.0))) / 2.0) + (((((K[8] * F5) + (K[5] * F4) + (K[2] * F3)) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) + (((K[6] * F8) + (K[3] * F7) + (K[0] * F6)) * ((K[8] * F8) + (K[5] * F7) + (K[2] *
F6) + 1.0)) + (((K[8] * F2) + (K[5] * F1) + (K[2] * F0)) * ((K[6] * F2) + (K[3] * F1) + (K[0] * F0) + 1.0))) / 2.0))) * F9) + (((((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) *
((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0]
[k]) + (K[7] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] *
BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0)) + (((K[2] * BC10[0][k]) + (K[5] * BC11[0][k]) + (K[8] * BC12[0][k])) * ((K[7] * F5) + (K[4]
* F4) + (K[1] * F3) + 1.0))) / 2.0) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k])
+ (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] *
..................[5] * F4) + (K[2] * F3))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[8] * F2) [illegible] * BC00[0][k]) + (K[5] * BC01[0][k]) +
(K[8] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8 [illegible] F7) + (K[2] * F6) + 1.0)) + (((K[2] *
BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))) / 2.0) + [illegible] ][k]) + (K[7] * BC22[0][k])) * ((K[7] *
F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[7] * F8) [illegible] * BC00[0][k]) + (K[4] * BC01[0][k]) +
(K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * BC00[0][k]) + (K[4] * BC01[0][k]) + (K[7] [illegible] F1) + (K[1] * F0))) + (((K[1] * BC10[0]
[k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0)) + (((K[1] * BC10[ [illegible] 12[0][k])) * ((K[7] * F5) + (K[4] * F4)
+ (K[1] * F3) + 1.0))) / 2.0) + ((((K[0] * BC20[0][k]) + (K[3] * BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) + ( [illegible] 20[0][k]) + (K[3] * BC21[0][k]) + (K[6]
* BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC1 [illegible] (K[0] * F3))) + (((K[0] * BC10[0][k]) +
(K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K [illegible] * ((K[7] * F2) + (K[4] * F1) + (K[1] *
F0))) + (((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) [illegible] * BC11[0][k]) + (K[8] * BC12[0][k])) *
((K[7] * F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0) + (((((K[2] * BC20[0][k]) + (K[5] * BC21[0][k] [illegible] + (K[1] * F6))) + (((K[1] * BC10[0][k])
+ (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[8] * F5) + (K[5] * F4) + (K[2] * F3))) + (((K[1] * BC [illegible] * ((K[8] * F2) + (K[5] * F1) + (K[2] *
F0))) + (((K[2] * ..................[5] * F4) + (K[2] * F3))) + (((K[2] * BC00[0][k]) + (K[5] * BC01[0][k]) + (K[8] [illegible] ) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC00[0][k]) + (K[5] *
BC01[0][k]) + (K[8] * BC02[0][k])) * ((K[8] * F2) + (K[5] * F1) + (K[2] * F0))) + (((K[2] * BC20[0][k]) + [illegible] [0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * F7) + (K[2] * F6) + 1.0))
+ (((K[2] * BC20[0][k]) + (K[5] * BC21[0][k]) + (K[8] * BC22[0][k])) * ((K[8] * F8) + (K[5] * [illegible] ))) / 2.0) + ((((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) + (K[7] * BC22[0][k]))
* ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC20[0][k]) + (K[4] * BC21[0][k]) [illegible] * ((K[7] * F8) + (K[4] * F7) + (K[1] * F6))) + (((K[1] * BC00[0][k]) + (K[4] *
BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) + (((K[1] * [illegible] ) + (K[4] * BC01[0][k]) + (K[7] * BC02[0][k])) * ((K[7] * F2) + (K[4] * F1) + (K[1] * F0))) +
(((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] * F5) + (K[4 [illegible] * F3) + 1.0)) + (((K[1] * BC10[0][k]) + (K[4] * BC11[0][k]) + (K[7] * BC12[0][k])) * ((K[7] *
F5) + (K[4] * F4) + (K[1] * F3) + 1.0))) / 2.0) + (((((K[0] * BC20[0][k]) + (K[3] * BC2 [illegible] ][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[0] * BC20[0][k]) + (K[3] *
BC21[0][k]) + (K[6] * BC22[0][k])) * ((K[6] * F8) + (K[3] * F7) + (K[0] * F6))) + (((K[ [illegible] BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) + (K[3] * F4) + (K[0] * F3))) +
(((K[0] * BC10[0][k]) + (K[3] * BC11[0][k]) + (K[6] * BC12[0][k])) * ((K[6] * F5) +

...
```

THREE SIMPLE EXAMPLES

I) FACTORISATION ENABLES CODE MOTION

```
for i in #integration points
 for j in #test functions
  for k in #trial functions

   A[j][k] += …

     1) B[i][j]*C[i][k] + B[i][j]*D[i][k]*f

     2) B[i][j]*C[i][k] + B[i][j]*D[i][k]*f

     3) B[i][j]*(C[i][k] + D[i][k]*f)

     4) B[i][j]*TMP[i][k]
```

2) EXPANSION ENABLES FACTORISATION

```
for i in #integration points
 for j in #test functions
  for k in #trial functions

   A[j][k] += …

    1) (B[i][j]*C[i][k] + … + …)*f +
       (B[i][j]*D[i][k] + … + …)*g

    2) (B[i][j]*C[i][k] + … + …)*f +
       (B[i][j]*D[i][k] + … + …)*g

    3) B[i][j]*(C[i][k]*f + D[i][k]*g)
       + …*f + …*f + …*g + …*g ……
```

3) COMMON SUB-EXPRESSIONS ELIMINATION

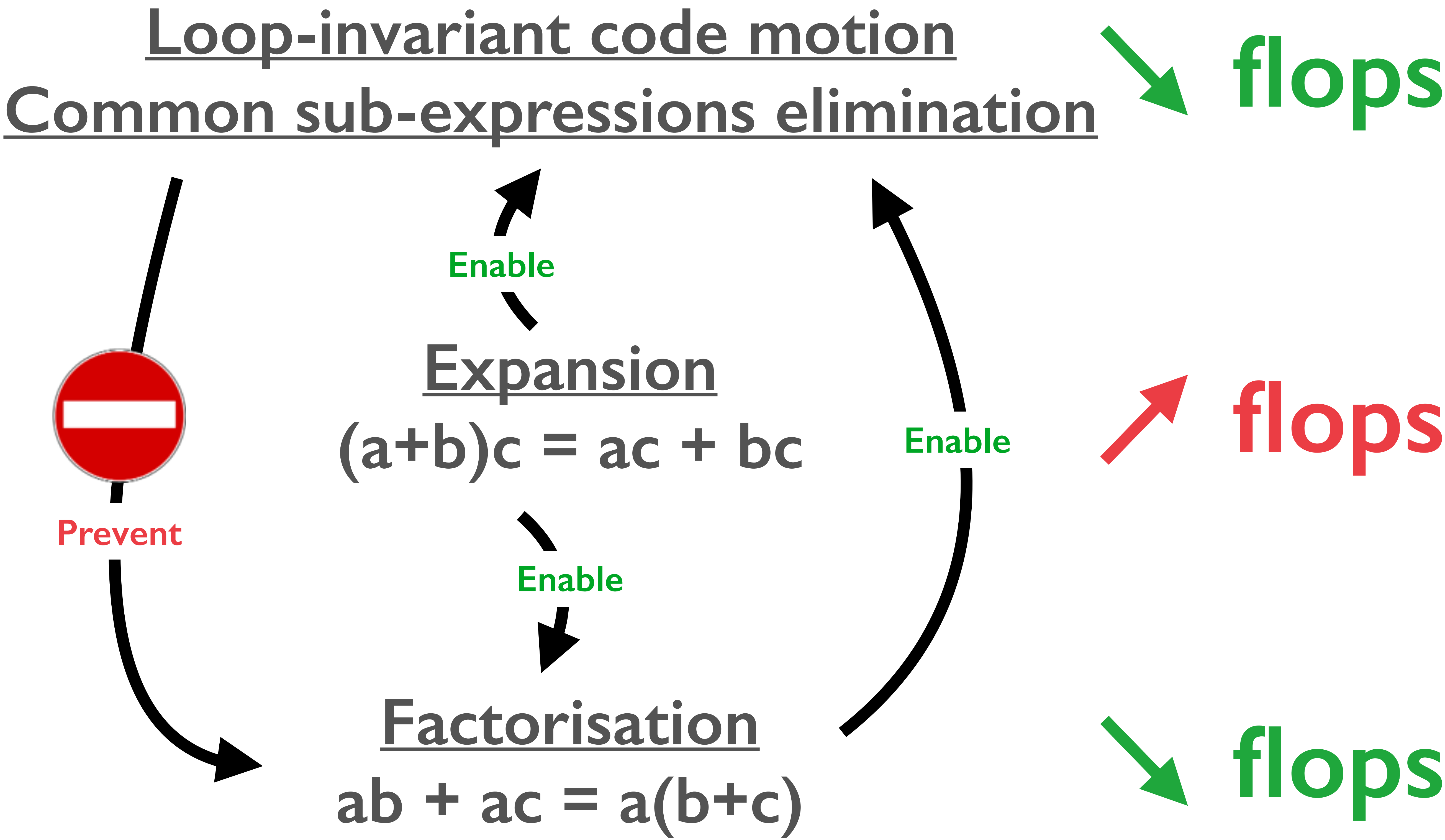
```
for i in #integration points
 for j in #test functions
  for k in #trial functions

   A[j][k] += …

     1) (B[i][j]*C[i][k] + … + …)*f +
        (B[i][j]*D[i][k] + … + …)*g + …
        …
        (B[i][j]*C[i][k] + … + …)*h + …

     2) (B[i][j]*C[i][k] + … + …)*f +
        (B[i][j]*D[i][k] + … + …)*g + …
        …
        (B[i][j]*C[i][k] + … + …)*h + …
```

ARSENAL FOR REDUCING FLOPS



QUESTION

How do we orchestrate the application of these rewrite operators ?

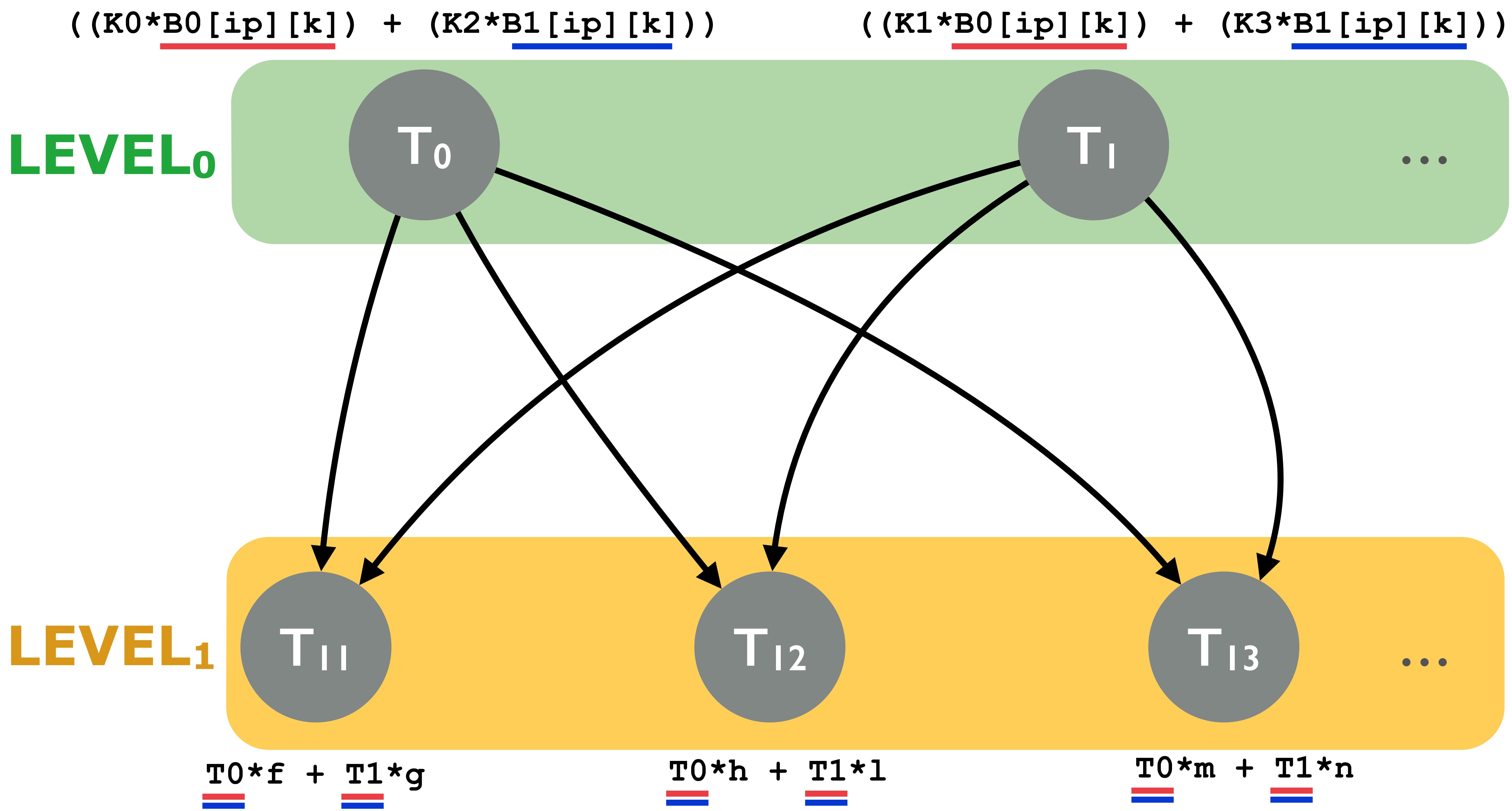
ANSWER

EXPLOIT LINEARITY

+

TEMPORARIES GRAPH

Node t_i = temporary (for a sub-expression occurring > 1)
Edge (t_i, t_j) = temporary t_j reads temporary t_i



Node t_i = temporary (for a sub-expression occurring > 1)

Edge (t_i, t_j) = temporary t_j reads temporary t_i


```
T0*f + T1*g
```

```
((K0*B0[ip][k]) + (K2*B1[ip][k]))*f + ((K1*B0[ip][k]) + (K3*B1[ip][k]))*g
K0*B0[ip][k]*f + K2*B1[ip][k]*f + K1*B0[ip][k]*g + K3*B1[ip][k]*g
B0[ip][k]*(K0*f + K1*g) + B1[ip][k]*(K2*f + K3*g)
```

By "injecting" temporaries, I can trade

common sub-expressions elimination

for

expansion + factorisation + code motion

The cost model iteratively (across multiple levels) compares these two alternatives

AN ILP FORMULATION FOR A LOCAL OPTIMUM

- **The temporaries graph analysis is applied twice, to test functions and (bilinear forms) trial functions**

- **The resulting expression will have temporaries depending on either test or trial functions**

```
A[j][k] += (T1[i][j]*T2[i][j] + … + B[i][j]*D[i][k] + … + …)*g +
           (T7[i][k]*T1[i][j] + … + …)*h + …
```

- **A simple ILP formulation finds the optimal factorisation**

Expression → **ILP** min … two sets of constraints → **Best factorisation strategy**

RESULTS ACHIEVED

Operation count

- The algorithm finds a local optimum by minimising the operation count within the inner loops AND through smart expression scheduling

- Sometimes finds a global optimum (i.e., best possible operation count)

Execution time

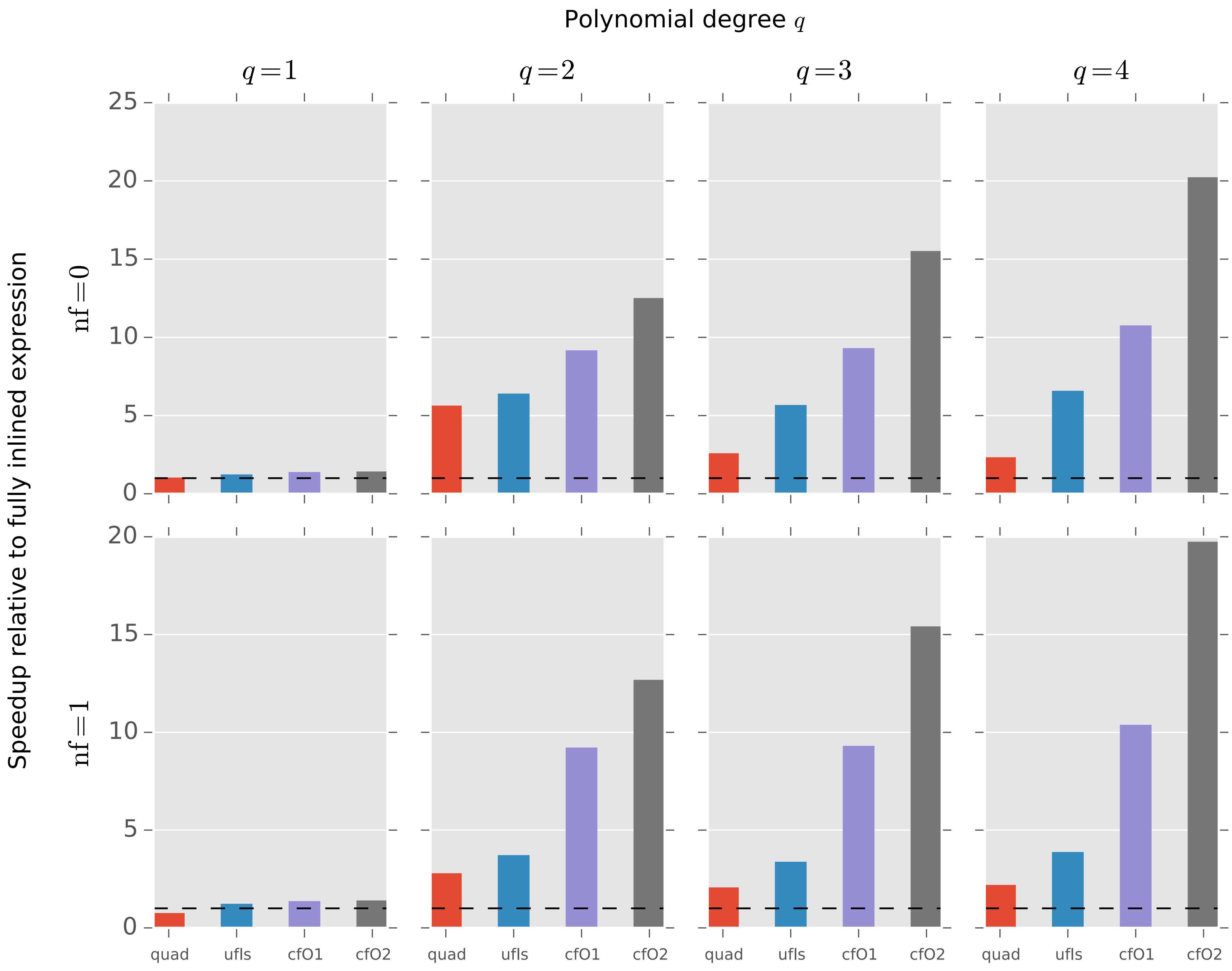
- In-depth experimentation with operators of increasing complexity
  Mass matrix => Helmholtz => Elasticity => Hyperelasticity

- Many parameters varied: polynomial order, coefficient functions, domain dimension

- Many compilers tried: GCC, Intel, Cray, LLVM — and many compilation options!

- Hundreds of test cases, winning (run-rime) in > 95% of them (over state-of-the-art code generation systems)

FOCUS ON HYPERELASTICITY



- Hyperelasticity
- Sandy Bridge (icc)
- Small 3D mesh (fit L3)

FFC-opt

UFLACS

COFFEE-v1

This talk

OPEN QUESTION — ACCELERATORS

- Extensive performance evaluation on CPUs
- Observation: code motion + common sub-expressions elimination require storage (increase in working set size!)
- Low order + L2/L3 cache on CPU: not a big issue.
  But what happens on GPUs?
  ===> Trade-off computation vs temporaries ?

CONCLUSIONS

- Shown: an algorithm for reducing the operation count of finite element integration loops. More info in the paper (arxiv):

  "An algorithm for the optimization of finite element integration loops"

- Exploit (simple) math behind variational formulations and uses a simple model to orchestrate rewrite operators

- Implementation in COFFEE: a tiny computer algebra system that "sees" loop nests and provides rewrite operators

- Extensive evaluation (more in the paper)