



Firedrake

Imperial College
London

Cross-Element Vectorization in Firedrake

TJ Sun (ts2914@ic.ac.uk), Lawrence Mitchell, David A Ham, Paul H J Kelly
June 2018

loo.py as our new backend

Introducing loo.py

- Andreas Klöckner et al. (UIUC)
- Based on polyhedral model of loops
- \approx model of loops + transformations
- “Do what I tell you”
- Support multiple backends
 - CPU
 - ISPC
 - OpenCL, PyOpenCL
 - Cuda

Code generation for assembly

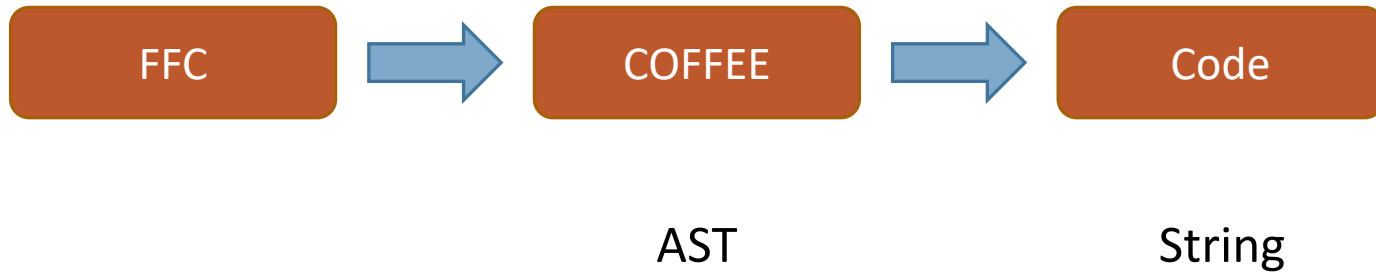
FFC



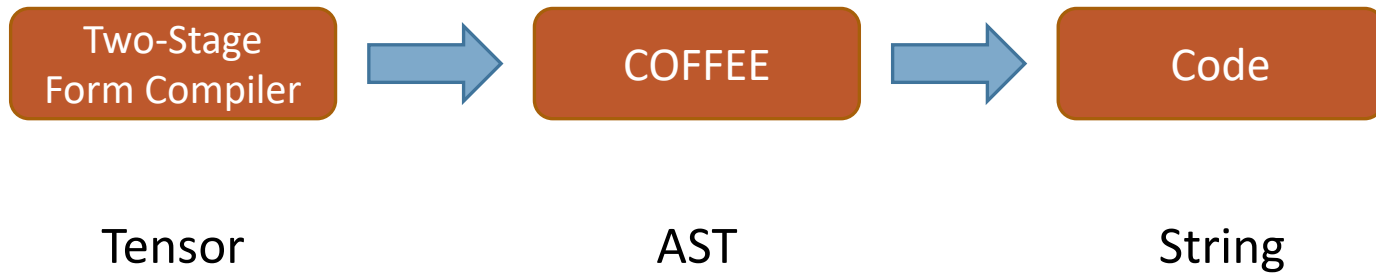
Code

string

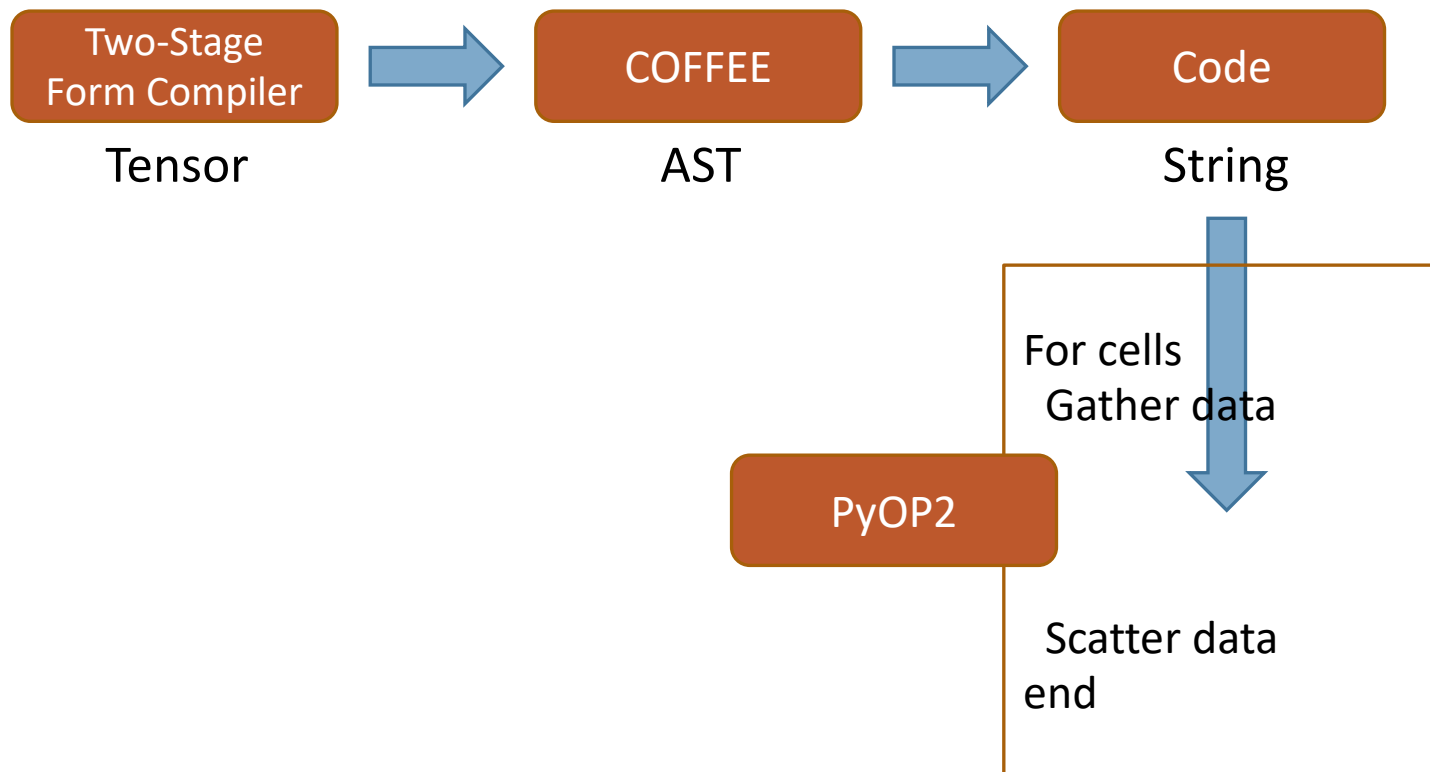
Code generation for assembly



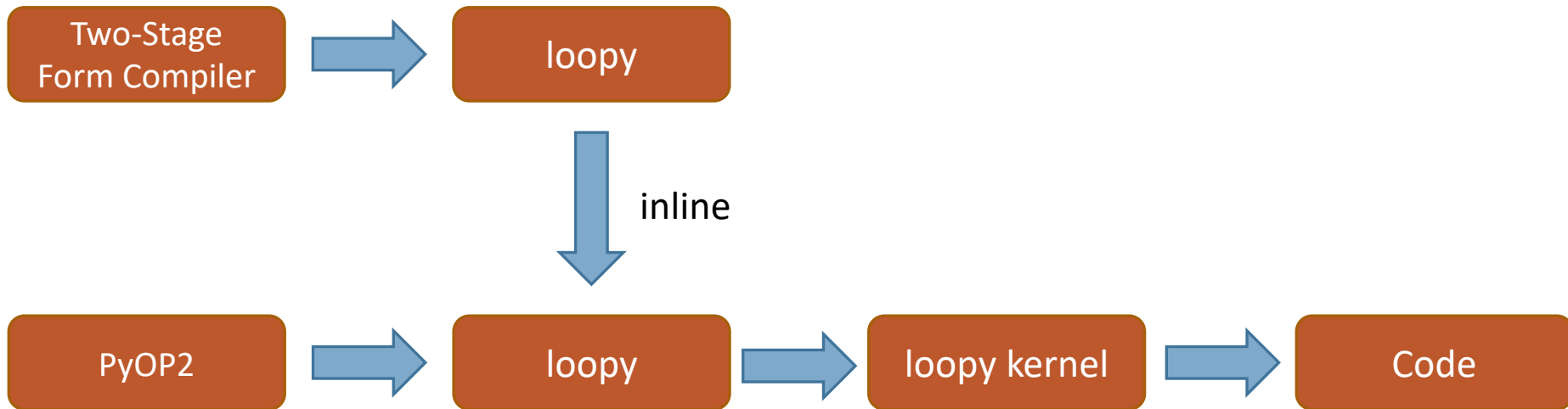
Code generation for assembly



Code generation for assembly

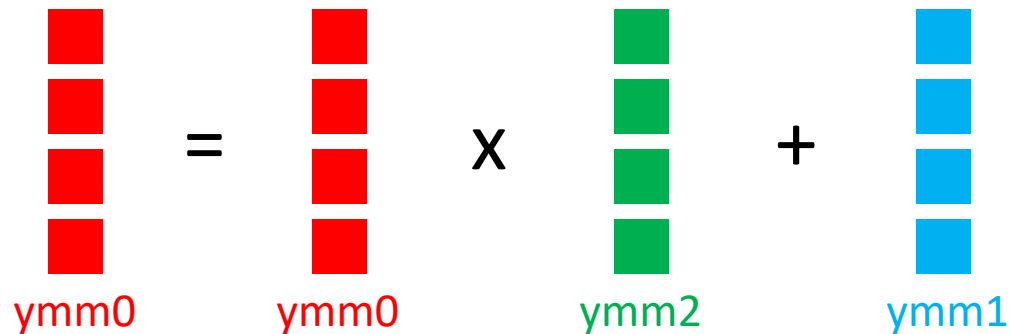


Code generation for assembly



What is vectorization

- SIMD (single instruction multiple data) programming model
- e.g. VFMADD213PD `yymm0`, `yymm1`, `yymm2` (in AVX2 instruction set)



- 8 double precision operations with 1 instruction
- Need to issue 2 FMA instructions in 1 cycle to get advertised performance
- SIMD width doubles every 4 years
 - AVX512 instructions can do 8 doubles
- Naïve code usually achieves <10% peak performance


```

3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6     /* ... */
7
8     for (int ip = 0; ip <= 5; ++ip)
9     {
10
11         /* ... */
12
13         for (int j0 = 0; j0 <= 9; ++j0)
14         {
15             A[2 * j0] = A[2 * j0] + t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
16             A[1 + 2 * j0] = A[1 + 2 * j0] + t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
17         }
18     }
19 }
20
21 void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
22 {
23     /* ... */
24     for (int n = start; n <= -1 + end; ++n)
25     {
26         for (int i7 = 0; i7 <= 2; ++i7)
27             for (int i8 = 0; i8 <= 1; ++i8)
28                 t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
29         for (int i13 = 0; i13 <= 9; ++i13)
30             for (int i14 = 0; i14 <= 1; ++i14)
31                 t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
32         for (int i1 = 0; i1 <= 9; ++i1)
33             for (int i2 = 0; i2 <= 1; ++i2)
34                 t2[2 * i1 + i2] = 0.0;
35
36         form0_cell_integral_otherwise(t2, t3, t4);
37
38         for (int i20 = 0; i20 <= 1; ++i20)
39             for (int i19 = 0; i19 <= 9; ++i19)
40                 dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41     }
42 }
43

```

Action of linear elasticity operator on triangle mesh, CG3 space

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6     /* ... */
7
8     for (int ip = 0; ip <= 5; ++ip)
9     {
10
11         /* ... */
12
13         for (int j0 = 0; j0 <= 9; ++j0)
14         {
15             A[2 * j0] = A[2 * j0] + t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
16             A[1 + 2 * j0] = A[1 + 2 * j0] + t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
17         }
18     }
19 }
20
21 void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
22 {
23     /* ... */
24     for (int n = start; n <= -1 + end; ++n)
25     {
26         for (int i7 = 0; i7 <= 2; ++i7)
27             for (int i8 = 0; i8 <= 1; ++i8)
28                 t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
29         for (int i13 = 0; i13 <= 9; ++i13)
30             for (int i14 = 0; i14 <= 1; ++i14)
31                 t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
32         for (int i1 = 0; i1 <= 9; ++i1)
33             for (int i2 = 0; i2 <= 1; ++i2)
34                 t2[2 * i1 + i2] = 0.0;
35
36         form0_cell_integral_otherwise(t2, t3, t4);
37
38         for (int i20 = 0; i20 <= 1; ++i20)
39             for (int i19 = 0; i19 <= 9; ++i19)
40                 dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41     }
42 }
43
```

Kernel

Wrapper

Action of linear elasticity operator on triangle mesh, CG3 space

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10
11 /* ... */
12
13 for (int j0 = 0; j0 <= 9; ++j0)
14 {
15     A[2 * j0] =
16     A[1 + 2 * j0] * t31;
17 }
18 }
19 }
20
21 void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
22 {
23 /* ... */
24 for (int n = start; n <= -1 + end; ++n)
25 {
26     for (int i7 = 0; i7 <= 2; ++i7)
27         for (int i8 = 0; i8 <= 1; ++i8)
28             t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
29     for (int i13 = 0; i13 <= 9; ++i13)
30         for (int i14 = 0; i14 <= 1; ++i14)
31             t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
32     for (int i1 = 0; i1 <= 9; ++i1)
33         for (int i2 = 0; i2 <= 1; ++i2)
34             t2[2 * i1 + i2] = 0.0;
35
36     form0_cell_integral_otherwise(t2, t3, t4);
37
38     for (int i20 = 0; i20 <= 1; ++i20)
39         for (int i19 = 0; i19 <= 9; ++i19)
40             dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41 }
42 }
43
```

Outer loop over all elements in the mesh

for (int n = start; n <= -1 + end; ++n)

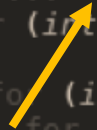
Wrapper

Action of linear elasticity operator on triangle mesh, CG3 space

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6 /* ... */
7
8 for (int ip = 0
9 {
10
11 /*
12     for (int i7 = 0; i7 <= 2; ++i7)
13         for (int i8 = 0; i8 <= 1; ++i8)
14             t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
15     for (int i13 = 0; i13 <= 9; ++i13)
16         for (int i14 = 0; i14 <= 1; ++i14)
17             t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
18     for (int i1 = 0; i1 <= 9; ++i1)
19         for (int i2 = 0; i2 <= 1; ++i2)
20             t2[2 * i1 + i2] = 0.0;
21 void wr
22 {
23 /* ...
24 for (int n = start; n <= -1 + end; ++n)
25 f
26     for (int i7 = 0; i7 <= 2; ++i7)
27         for (int i8 = 0; i8 <= 1; ++i8)
28             t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
29     for (int i13 = 0; i13 <= 9; ++i13)
30         for (int i14 = 0; i14 <= 1; ++i14)
31             t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
32     for (int i1 = 0; i1 <= 9; ++i1)
33         for (int i2 = 0; i2 <= 1; ++i2)
34             t2[2 * i1 + i2] = 0.0;
35
36     form0_cell_integral_otherwise(t2, t3, t4);
37
38     for (int i20 = 0; i20 <= 1; ++i20)
39         for (int i19 = 0; i19 <= 9; ++i19)
40             dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41 }
42 }
43
```

Indirect gathering of input data for kernel

```
for (int i7 = 0; i7 <= 2; ++i7)
    for (int i8 = 0; i8 <= 1; ++i8)
        t3[2 * i7 + i8] = dat1[2 * map1[3 * n + i7] + i8];
for (int i13 = 0; i13 <= 9; ++i13)
    for (int i14 = 0; i14 <= 1; ++i14)
        t4[2 * i13 + i14] = dat2[2 * map0[10 * n + i13] + i14];
for (int i1 = 0; i1 <= 9; ++i1)
    for (int i2 = 0; i2 <= 1; ++i2)
        t2[2 * i1 + i2] = 0.0;
```



Action of linear elasticity operator on triangle mesh, CG3 space

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10
11 /* ... */
12
13 for (int j0 = 0; j0 <= 9; ++j0)
14 {
15     A[2 * j0] = A[2 * j0] + t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
16     A[1 + 2 * j0] = A[1 + 2 * j0] + t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
17 }
18 }
19 }
20
21 void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
22 {
23 /* ... */
24 for (int n = start; n <= -1 + end; ++n)
25 {
26     for (int i7 = 0; i7 <= 2; ++i7)
27         for (int i8 = 0; i8 <= 1; ++i8)
28             t3[2 * i7 + i8] = dat1[2 * i7 + i8];
29     for (int i13 = 0; i13 <= 9; ++i13)
30         for (int i14 = 0; i14 <= 1; ++i14)
31             t4[2 * i13 + i14] = dat2[2 * i13 + i14];
32     for (int i1 = 0; i1 <= 9; ++i1)
33         for (int i2 = 0; i2 <= 1; ++i2)
34             t2[2 * i1 + i2] = 0.0;
35
36     form0_cell_integral_otherwise(t2, t3, t4);
37
38     for (int i20 = 0; i20 <= 1; ++i20)
39         for (int i19 = 0; i19 <= 9; ++i19)
40             dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41 }
42 }
43
```

Kernel "call", actually it is inlined

kernel(t2, t3, t4);

Action of linear elasticity operator on triangle mesh, CG3 space

```

3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6     /* ... */
7
8     for (int ip = 0; ip <= 5; ++ip)
9     {
10
11         /* ... */
12
13         for (int j0 = 0; j0 <= 9; ++j0)
14         {
15             A[2 * j0] = A[2 * j0] + t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
16             A[1 + 2 * j0] = A[1 + 2 * j0] + t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
17         }
18     }
19 }
20
21 void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
22 {
23     /* ... */
24     for (int n = start; n <= -1 + end; ++n)
25     {
26         for (int i
27             for (int
28                 t3[2 *
29
30         for (int i20 = 0; i20 <= 1; ++i20)
31             for (int i19 = 0; i19 <= 9; ++i19)
32                 dat0[2 * map0[10 * n + i19] + i20] += t2[2 * i19 + i20];
33
34         t2[2 * i1 + i2] = 0.0;
35
36         form_cell_integral_otherwise(t2, t3, t4);
37
38         for (int i20 = 0; i20 <= 1; ++i20)
39             for (int i19 = 0; i19 <= 9; ++i19)
40                 dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20];
41     }
42 }
43

```

Indirect scattering of local tensor to global tensor

Action of linear elasticity operator on triangle mesh, CG3 space

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10
11 /* ... */
12
13 for (int j0 = 0; j0 <= 9; ++j0)
14 {
15     A[2 * j0] = A[2 * j0] + t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
16     A[1 + 2 * j0] = A[1 + 2 * j0] + t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
17 }
18 }
19
20 void wrapper
21 * __restrict__
```

Kernel

Outer loop: contraction over quadrature points

```
22 for (int ip = 0; ip <= 5; ++ip)
23 {
24
25
26 /* ... inner loop over degrees of freedom
27
28 for (int j0 = 0; j0 <= 9; ++j0)
29 {
30
31     A[2 * j0] += t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
32     A[1 + 2 * j0] += t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
33 }
34 }
35
36 }
```

Action of linear elasticity operator on triangle mesh, CG3 space

Vectorization strategy

- “Intra-kernel” can be tricky
 - Trip count can be small and/or not multiple of SIMD width
 - Alignment to cache boundary
 - Stride 1 access
 - Operations outside of innermost loop not vectorized
 - Loop structure varies with PDE, discretization, mesh
 - And we have done many of these in firedrake [1]
- “Inter-kernel” provides a generic solution
 - Vector-expand the kernel to act on N elements together, $N = \text{SIMD width}$
 - All operations can be vectorized
 - Can always do this systematically
 - Downside: increasing working size

[1] F. Luporini, et al. ACM TACO 2015


```

3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict__
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10 /* ... */
11
12 for (int j0 = 0; j0 <= 9; ++j0)
13   for (int elem = 0; elem <= 3; ++elem)
14     {
15       A[elem + 4 * 2 * j0] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[elem]
16       A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip + j0] * t35[elem]
17     }
18 }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double const
22 {
23
24 /* ... */
25
26 for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
27 {
28   for (int i7 = 0; i7 <= 2; ++i7)
29     for (int i8 = 0; i8 <= 1; ++i8)
30       for (int n_inner = 0; n_inner <= 3; ++n_inner)
31         t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
32   for (int i13 = 0; i13 <= 9; ++i13)
33     for (int i14 = 0; i14 <= 1; ++i14)
34       for (int n_inner = 0; n_inner <= 3; ++n_inner)
35         t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
36   for (int i1 = 0; i1 <= 9; ++i1)
37     for (int i2 = 0; i2 <= 1; ++i2)
38       for (int n_inner = 0; n_inner <= 3; ++n_inner)
39         t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
40
41   kernel(t2, t3, t4);
42
43   for (int i20 = 0; i20 <= 1; ++i20)
44     for (int i19 = 0; i19 <= 9; ++i19)
45       for (int n_inner = 0; n_inner <= 3; ++n_inner)
46         dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] + t20[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20];
47 }
48 }

```

Action of linear elasticity operator on triangle mesh, batched by 4

Gathering input data for 4 elements

Arrays are vector-expanded

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict__
4 {
5
6 /* ... */
7 for (int ip = 0; ip
8 {
9 /* ... */
10
11 for (int i7 = 0; i7 <= 2; ++i7)
12   for (int i8 = 0; i8 <= 1; ++i8)
13     for (int n_inner = 0; n_inner <= 3; ++n_inner)
14       t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
15
16 for (int i13 = 0; i13 <= 9; ++i13)
17   for (int i14 = 0; i14 <= 1; ++i14)
18     for (int n_inner = 0; n_inner <= 3; ++n_inner)
19       t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
20
21 for (int i1 = 0; i1 <= 9; ++i1)
22   for (int i2 = 0; i2 <= 1; ++i2)
23     for (int n_inner = 0; n_inner <= 3; ++n_inner)
24       t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
```

Data for different elements packed to inner most dimension

```
26 for (int n_outer = (start /
27 {
28   for (int i7 = 0; i7 <= 2;
29     for (int i8 = 0; i8 <=
30       for (int n_inner = 0;
31         t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
32     for (int i13 = 0; i13 <= 9; ++i13)
33       for (int i14 = 0; i14 <= 1; ++i14)
34         for (int n_inner = 0; n_inner <= 3; ++n_inner)
35           t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
36     for (int i1 = 0; i1 <= 9; ++i1)
37       for (int i2 = 0; i2 <= 1; ++i2)
38         for (int n_inner = 0; n_inner <= 3; ++n_inner)
39           t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
```

```
40
41 kernel(t2, t3, t4);
```

```
42
43 for (int i20 = 0; i20 <= 1; ++i20)
44   for (int i19 = 0; i19 <= 9; ++i19)
45     for (int n_inner = 0; n_inner <= 3; ++n_inner)
46       dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20];
47 }
48 }
```

Action of linear elasticity operator on triangle mesh, batched by 4

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict__
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10 /* ... */
11
12 for (int j0 = 0; j0 <= 9; ++j0)
13 for (int elem = 0; elem <= 3; ++elem)
14 {
15     A[elem + 4 * 2 * j0] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[elem]
16     A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip + j0] * t33[elem]
17 }
18 }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double const
22 {
23
24 /* ... */
25
26 for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
27 {
28     for (int i7 = 0; i7 <= 2; ++i7)
29         for (int i8 = 0; i8 <= 1; ++i8)
30             for (int n_inner = 0; n_inner <= 3; ++n_inner)
31                 t3[8 * i7 + 4 * i8 + n_inner] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i7] + i8];
32     for (int i13 = 0; i13 <= 9; ++i13)
33         for (int i14 = 0; i14 <= 1; ++i14)
34             for (int n_inner = 0; n_inner <= 3; ++n_inner)
35                 t4[8 * i13 + 4 * i14 + n_inner] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
36     for (int i1 = 0; i1 <= 9; ++i1)
37         for (int i2 = 0; i2 <= 1; ++i2)
38             for (int n_inner = 0; n_inner <= 3; ++n_inner)
39                 t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
40
41     kernel(t2, t3, t4);
42
43     for (int i20 = 0; i20 <= 1; ++i20)
44         for (int i19 = 0; i19 <= 9; ++i19)
45             for (int n_inner = 0; n_inner <= 3; ++n_inner)
46                 dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20];
47 }
48 }
```

Kernel call

kernel(t2, t3, t4);

Action of linear elasticity operator on triangle mesh, batched by 4

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict__
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10 /* ... */
11
12 for (int j0 = 0; j0 <= 9; ++j0)
13 for (int elem = 0; elem <= 3; ++elem)
14 {
15     A[elem + 4 * 2 * j0] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[elem]
16     A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip + j0] * t35[elem]
17 }
18 }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double const
22 {
23
24 /* ... */
25
26 for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
27 {
28     for (int i7 = 0; i7 <= 2; ++i7)
29         for (int i8 = 0; i8 <= 1; ++i8)
30             for (int n_inner = 0; n_inner <= 3; ++n_inner)
31                 dat0[2 * map0[10 * (4 * n_outer + n_inner) + i7 + i8] + n_inner] =
32                     t3[8 * i7 + 4 * i8 + n_inner] + t4[8 * i7 + 4 * i8 + n_inner] + t5[8 * i7 + 4 * i8 + n_inner] + t6[8 * i7 + 4 * i8 + n_inner] + t7[8 * i7 + 4 * i8 + n_inner] + t8[8 * i7 + 4 * i8 + n_inner] + t9[8 * i7 + 4 * i8 + n_inner] + t10[8 * i7 + 4 * i8 + n_inner] + t11[8 * i7 + 4 * i8 + n_inner] + t12[8 * i7 + 4 * i8 + n_inner] + t13[8 * i7 + 4 * i8 + n_inner] + t14[8 * i7 + 4 * i8 + n_inner] + t15[8 * i7 + 4 * i8 + n_inner] + t16[8 * i7 + 4 * i8 + n_inner] + t17[8 * i7 + 4 * i8 + n_inner] + t18[8 * i7 + 4 * i8 + n_inner] + t19[8 * i7 + 4 * i8 + n_inner] + t20[8 * i7 + 4 * i8 + n_inner] + t21[8 * i7 + 4 * i8 + n_inner] + t22[8 * i7 + 4 * i8 + n_inner] + t23[8 * i7 + 4 * i8 + n_inner] + t24[8 * i7 + 4 * i8 + n_inner] + t25[8 * i7 + 4 * i8 + n_inner] + t26[8 * i7 + 4 * i8 + n_inner] + t27[8 * i7 + 4 * i8 + n_inner] + t28[8 * i7 + 4 * i8 + n_inner] + t29[8 * i7 + 4 * i8 + n_inner] + t30[8 * i7 + 4 * i8 + n_inner] + t31[8 * i7 + 4 * i8 + n_inner] + t32[8 * i7 + 4 * i8 + n_inner] + t33[8 * i7 + 4 * i8 + n_inner] + t34[8 * i7 + 4 * i8 + n_inner] + t35[8 * i7 + 4 * i8 + n_inner] + t36[8 * i7 + 4 * i8 + n_inner] + t37[8 * i7 + 4 * i8 + n_inner] + t38[8 * i7 + 4 * i8 + n_inner] + t39[8 * i7 + 4 * i8 + n_inner] + t40[8 * i7 + 4 * i8 + n_inner] + t41[8 * i7 + 4 * i8 + n_inner] + t42[8 * i7 + 4 * i8 + n_inner] + t43[8 * i7 + 4 * i8 + n_inner] + t44[8 * i7 + 4 * i8 + n_inner] + t45[8 * i7 + 4 * i8 + n_inner] + t46[8 * i7 + 4 * i8 + n_inner] + t47[8 * i7 + 4 * i8 + n_inner] + t48[8 * i7 + 4 * i8 + n_inner] + t49[8 * i7 + 4 * i8 + n_inner] + t50[8 * i7 + 4 * i8 + n_inner] + t51[8 * i7 + 4 * i8 + n_inner] + t52[8 * i7 + 4 * i8 + n_inner] + t53[8 * i7 + 4 * i8 + n_inner] + t54[8 * i7 + 4 * i8 + n_inner] + t55[8 * i7 + 4 * i8 + n_inner] + t56[8 * i7 + 4 * i8 + n_inner] + t57[8 * i7 + 4 * i8 + n_inner] + t58[8 * i7 + 4 * i8 + n_inner] + t59[8 * i7 + 4 * i8 + n_inner] + t60[8 * i7 + 4 * i8 + n_inner] + t61[8 * i7 + 4 * i8 + n_inner] + t62[8 * i7 + 4 * i8 + n_inner] + t63[8 * i7 + 4 * i8 + n_inner] + t64[8 * i7 + 4 * i8 + n_inner] + t65[8 * i7 + 4 * i8 + n_inner] + t66[8 * i7 + 4 * i8 + n_inner] + t67[8 * i7 + 4 * i8 + n_inner] + t68[8 * i7 + 4 * i8 + n_inner] + t69[8 * i7 + 4 * i8 + n_inner] + t70[8 * i7 + 4 * i8 + n_inner] + t71[8 * i7 + 4 * i8 + n_inner] + t72[8 * i7 + 4 * i8 + n_inner] + t73[8 * i7 + 4 * i8 + n_inner] + t74[8 * i7 + 4 * i8 + n_inner] + t75[8 * i7 + 4 * i8 + n_inner] + t76[8 * i7 + 4 * i8 + n_inner] + t77[8 * i7 + 4 * i8 + n_inner] + t78[8 * i7 + 4 * i8 + n_inner] + t79[8 * i7 + 4 * i8 + n_inner] + t80[8 * i7 + 4 * i8 + n_inner] + t81[8 * i7 + 4 * i8 + n_inner] + t82[8 * i7 + 4 * i8 + n_inner] + t83[8 * i7 + 4 * i8 + n_inner] + t84[8 * i7 + 4 * i8 + n_inner] + t85[8 * i7 + 4 * i8 + n_inner] + t86[8 * i7 + 4 * i8 + n_inner] + t87[8 * i7 + 4 * i8 + n_inner] + t88[8 * i7 + 4 * i8 + n_inner] + t89[8 * i7 + 4 * i8 + n_inner] + t90[8 * i7 + 4 * i8 + n_inner] + t91[8 * i7 + 4 * i8 + n_inner] + t92[8 * i7 + 4 * i8 + n_inner] + t93[8 * i7 + 4 * i8 + n_inner] + t94[8 * i7 + 4 * i8 + n_inner] + t95[8 * i7 + 4 * i8 + n_inner] + t96[8 * i7 + 4 * i8 + n_inner] + t97[8 * i7 + 4 * i8 + n_inner] + t98[8 * i7 + 4 * i8 + n_inner] + t99[8 * i7 + 4 * i8 + n_inner] + t100[8 * i7 + 4 * i8 + n_inner];
37
38     for (int i2 = 0; i2 <= 1; ++i2)
39         for (int n_inner = 0; n_inner <= 3; ++n_inner)
40             t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
41
42     kernel(t2, t3, t4);
43
44     for (int i20 = 0; i20 <= 1; ++i20)
45         for (int i19 = 0; i19 <= 9; ++i19)
46             for (int n_inner = 0; n_inner <= 3; ++n_inner)
47                 dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] + t1[8 * i19 + 4 * i20 + n_inner] + t2[8 * i19 + 4 * i20 + n_inner] + t3[8 * i19 + 4 * i20 + n_inner] + t4[8 * i19 + 4 * i20 + n_inner] + t5[8 * i19 + 4 * i20 + n_inner] + t6[8 * i19 + 4 * i20 + n_inner] + t7[8 * i19 + 4 * i20 + n_inner] + t8[8 * i19 + 4 * i20 + n_inner] + t9[8 * i19 + 4 * i20 + n_inner] + t10[8 * i19 + 4 * i20 + n_inner] + t11[8 * i19 + 4 * i20 + n_inner] + t12[8 * i19 + 4 * i20 + n_inner] + t13[8 * i19 + 4 * i20 + n_inner] + t14[8 * i19 + 4 * i20 + n_inner] + t15[8 * i19 + 4 * i20 + n_inner] + t16[8 * i19 + 4 * i20 + n_inner] + t17[8 * i19 + 4 * i20 + n_inner] + t18[8 * i19 + 4 * i20 + n_inner] + t19[8 * i19 + 4 * i20 + n_inner] + t20[8 * i19 + 4 * i20 + n_inner] + t21[8 * i19 + 4 * i20 + n_inner] + t22[8 * i19 + 4 * i20 + n_inner] + t23[8 * i19 + 4 * i20 + n_inner] + t24[8 * i19 + 4 * i20 + n_inner] + t25[8 * i19 + 4 * i20 + n_inner] + t26[8 * i19 + 4 * i20 + n_inner] + t27[8 * i19 + 4 * i20 + n_inner] + t28[8 * i19 + 4 * i20 + n_inner] + t29[8 * i19 + 4 * i20 + n_inner] + t30[8 * i19 + 4 * i20 + n_inner] + t31[8 * i19 + 4 * i20 + n_inner] + t32[8 * i19 + 4 * i20 + n_inner] + t33[8 * i19 + 4 * i20 + n_inner] + t34[8 * i19 + 4 * i20 + n_inner] + t35[8 * i19 + 4 * i20 + n_inner] + t36[8 * i19 + 4 * i20 + n_inner] + t37[8 * i19 + 4 * i20 + n_inner] + t38[8 * i19 + 4 * i20 + n_inner] + t39[8 * i19 + 4 * i20 + n_inner] + t40[8 * i19 + 4 * i20 + n_inner] + t41[8 * i19 + 4 * i20 + n_inner] + t42[8 * i19 + 4 * i20 + n_inner] + t43[8 * i19 + 4 * i20 + n_inner] + t44[8 * i19 + 4 * i20 + n_inner] + t45[8 * i19 + 4 * i20 + n_inner] + t46[8 * i19 + 4 * i20 + n_inner] + t47[8 * i19 + 4 * i20 + n_inner] + t48[8 * i19 + 4 * i20 + n_inner] + t49[8 * i19 + 4 * i20 + n_inner] + t50[8 * i19 + 4 * i20 + n_inner] + t51[8 * i19 + 4 * i20 + n_inner] + t52[8 * i19 + 4 * i20 + n_inner] + t53[8 * i19 + 4 * i20 + n_inner] + t54[8 * i19 + 4 * i20 + n_inner] + t55[8 * i19 + 4 * i20 + n_inner] + t56[8 * i19 + 4 * i20 + n_inner] + t57[8 * i19 + 4 * i20 + n_inner] + t58[8 * i19 + 4 * i20 + n_inner] + t59[8 * i19 + 4 * i20 + n_inner] + t60[8 * i19 + 4 * i20 + n_inner] + t61[8 * i19 + 4 * i20 + n_inner] + t62[8 * i19 + 4 * i20 + n_inner] + t63[8 * i19 + 4 * i20 + n_inner] + t64[8 * i19 + 4 * i20 + n_inner] + t65[8 * i19 + 4 * i20 + n_inner] + t66[8 * i19 + 4 * i20 + n_inner] + t67[8 * i19 + 4 * i20 + n_inner] + t68[8 * i19 + 4 * i20 + n_inner] + t69[8 * i19 + 4 * i20 + n_inner] + t70[8 * i19 + 4 * i20 + n_inner] + t71[8 * i19 + 4 * i20 + n_inner] + t72[8 * i19 + 4 * i20 + n_inner] + t73[8 * i19 + 4 * i20 + n_inner] + t74[8 * i19 + 4 * i20 + n_inner] + t75[8 * i19 + 4 * i20 + n_inner] + t76[8 * i19 + 4 * i20 + n_inner] + t77[8 * i19 + 4 * i20 + n_inner] + t78[8 * i19 + 4 * i20 + n_inner] + t79[8 * i19 + 4 * i20 + n_inner] + t80[8 * i19 + 4 * i20 + n_inner] + t81[8 * i19 + 4 * i20 + n_inner] + t82[8 * i19 + 4 * i20 + n_inner] + t83[8 * i19 + 4 * i20 + n_inner] + t84[8 * i19 + 4 * i20 + n_inner] + t85[8 * i19 + 4 * i20 + n_inner] + t86[8 * i19 + 4 * i20 + n_inner] + t87[8 * i19 + 4 * i20 + n_inner] + t88[8 * i19 + 4 * i20 + n_inner] + t89[8 * i19 + 4 * i20 + n_inner] + t90[8 * i19 + 4 * i20 + n_inner] + t91[8 * i19 + 4 * i20 + n_inner] + t92[8 * i19 + 4 * i20 + n_inner] + t93[8 * i19 + 4 * i20 + n_inner] + t94[8 * i19 + 4 * i20 + n_inner] + t95[8 * i19 + 4 * i20 + n_inner] + t96[8 * i19 + 4 * i20 + n_inner] + t97[8 * i19 + 4 * i20 + n_inner] + t98[8 * i19 + 4 * i20 + n_inner] + t99[8 * i19 + 4 * i20 + n_inner] + t100[8 * i19 + 4 * i20 + n_inner];
48 }
```

Scattering might have race condition

```
for (int i20 = 0; i20 <= 1; ++i20)
    for (int i19 = 0; i19 <= 9; ++i19)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
            dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] += t2[8 * i19 + 4 * i20 + n_inner];
```



Action of linear elasticity operator on triangle mesh, batched by 4

```
3 static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict__
4 {
5
6 /* ... */
7
8 for (int ip = 0; ip <= 5; ++ip)
9 {
10 /* ... */
11
12 for (int j0 = 0; j0 <= 9; ++j0)
13 for (int elem = 0; elem <= 3; ++elem)
14 {
15     A[elem + 4 * 2 * j0] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[elem];
16     A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip + j0] * t31[elem];
17 }
18 }
19 }
20 }
```

Kernel

```
for (int ip = 0; ip <= 5; ++ip)
{
/* ... */

for (int j0 = 0; j0 <= 9; ++j0)
for (int elem = 0; elem <= 3; ++elem)
{
A[elem + 4 * 2 * j0] += t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[elem];
A[elem + 4 * (1 + 2 * j0)] += t18[10 * ip + j0] * t32[elem] + t19[10 * ip + j0] * t31[elem];
}
}
```

“element” loop pushed to innermost

Trip count 4, stride 1, aligned, independent

```
34 for (int n_inner = 0; n_inner <= 3; ++n_inner)
35     t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
36 for (int i1 = 0; i1 <= 9; ++i1)
37 for (int i2 = 0; i2 <= 1; ++i2)
38 for (int n_inner = 0; n_inner <= 3; ++n_inner)
39     t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
40
41 kernel(t2, t3, t4);
42
43 for (int i20 = 0; i20 <= 1; ++i20)
44 for (int i19 = 0; i19 <= 9; ++i19)
45 for (int n_inner = 0; n_inner <= 3; ++n_inner)
46     dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20] = dat0[2 * map0[10 * (4 * n_outer + n_inner) + i19] + i20];
47 }
48 }
```

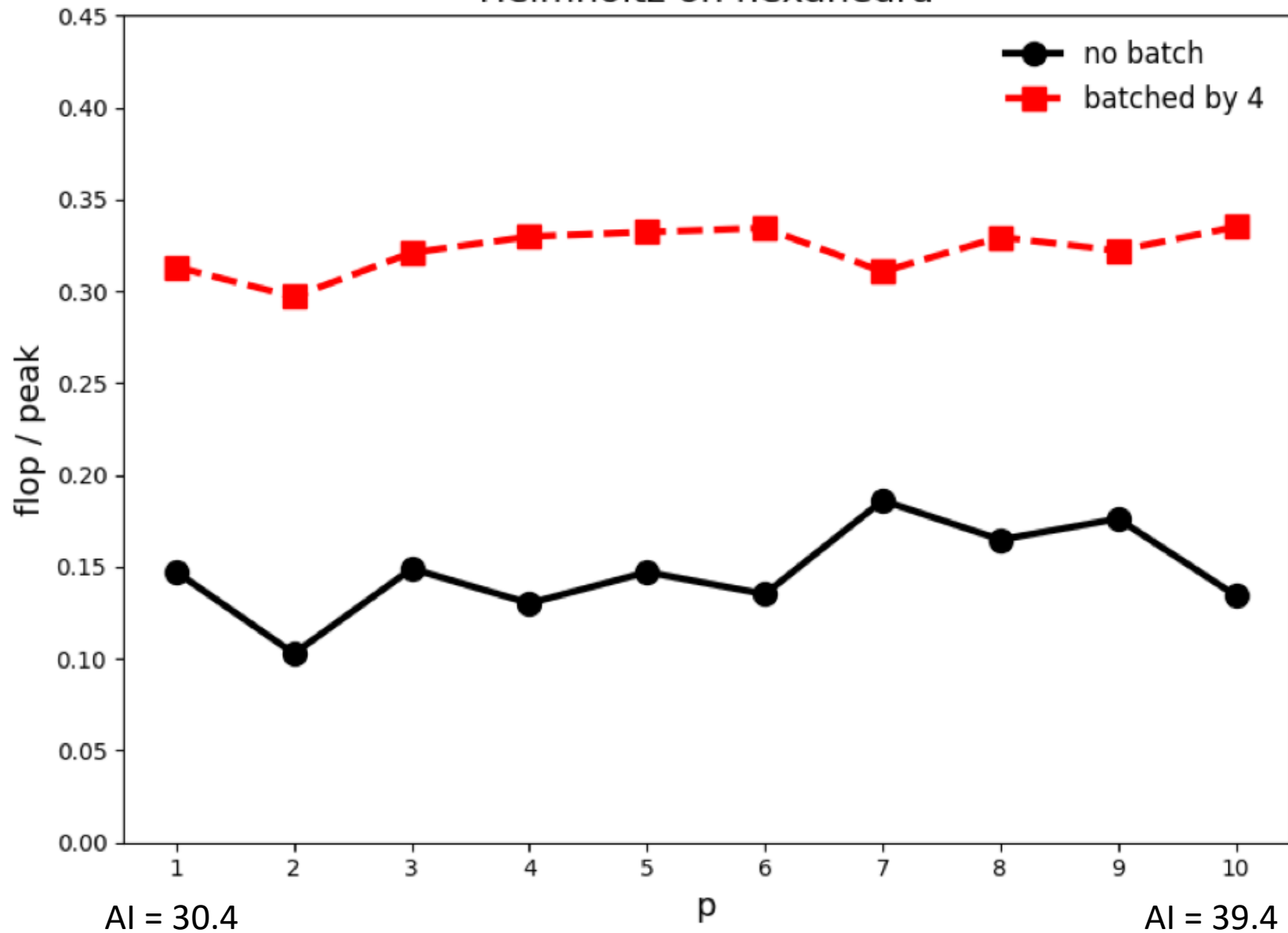
Action of linear elasticity operator on triangle mesh, batched by 4

Experimental setup

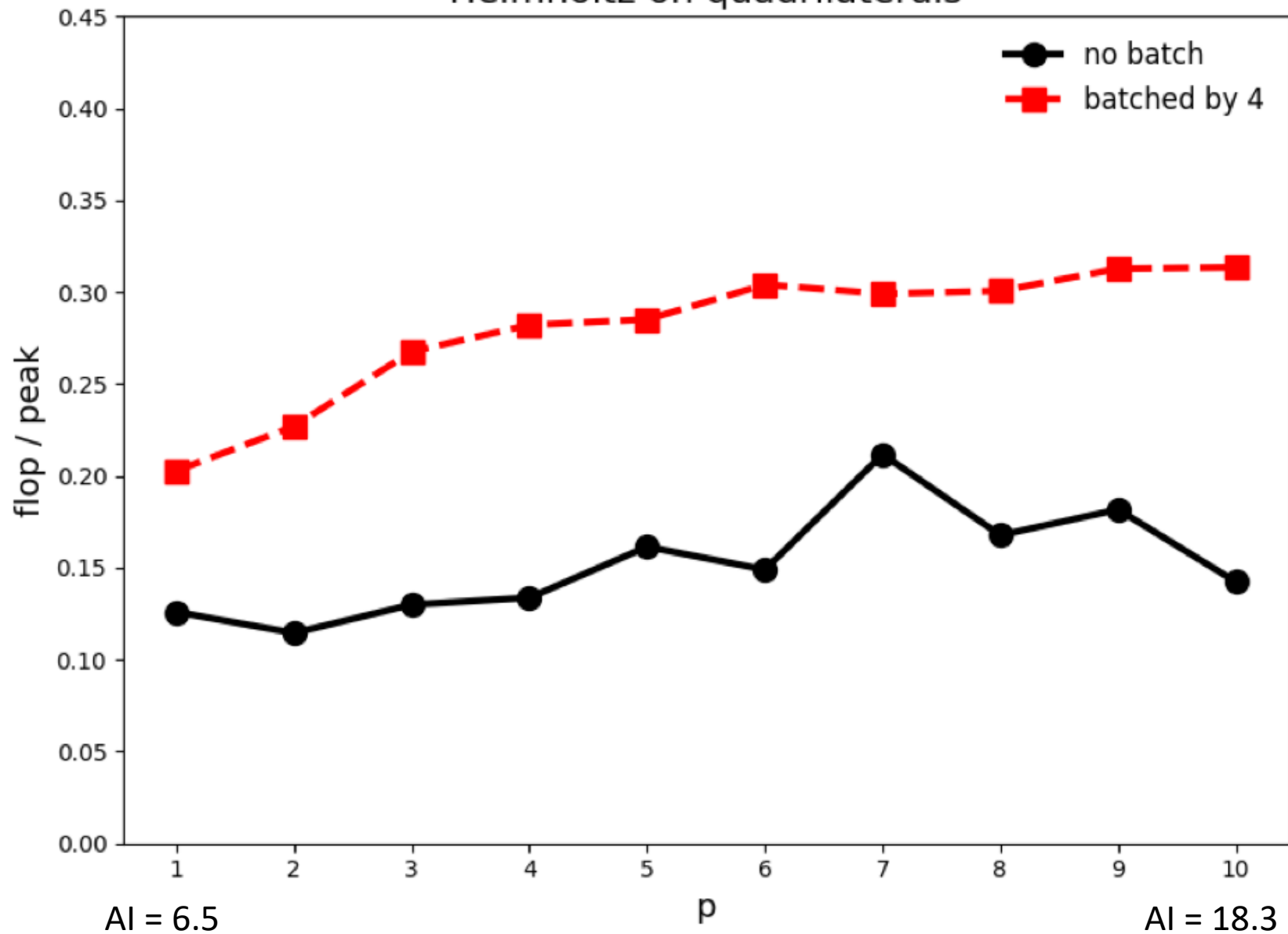
- Hardware: Haswell i7-4790 (single core measurement)
 - SIMD width = 4 (avx2)
 - Peak flop = 3.6 GHz x 4 (avx2) x 2 (fma) x 2 (issue) = 57.6 Gflops
 - Running Intel LINPACK binary: 51.0 Gflops
 - STREAM triad bandwidth: 10.4 GB / s
 - Roofline AI “regime switching point” = 5.54 flops / byte
- Mesh: hexahedra (3D tensor product element)
 - TSFC automates sum factorisation [2]
 - > Innermost loop trip count = polynomial degree + 1
- Action of Helmholtz operator
 - Arithmetic Intensity (perfect cache) 30.4 to 33.6 → compute bound
- We present achieved flops / 57.6 Gflops

[2] M. Homolya, et al. arXiv:1711.02473 (2017).

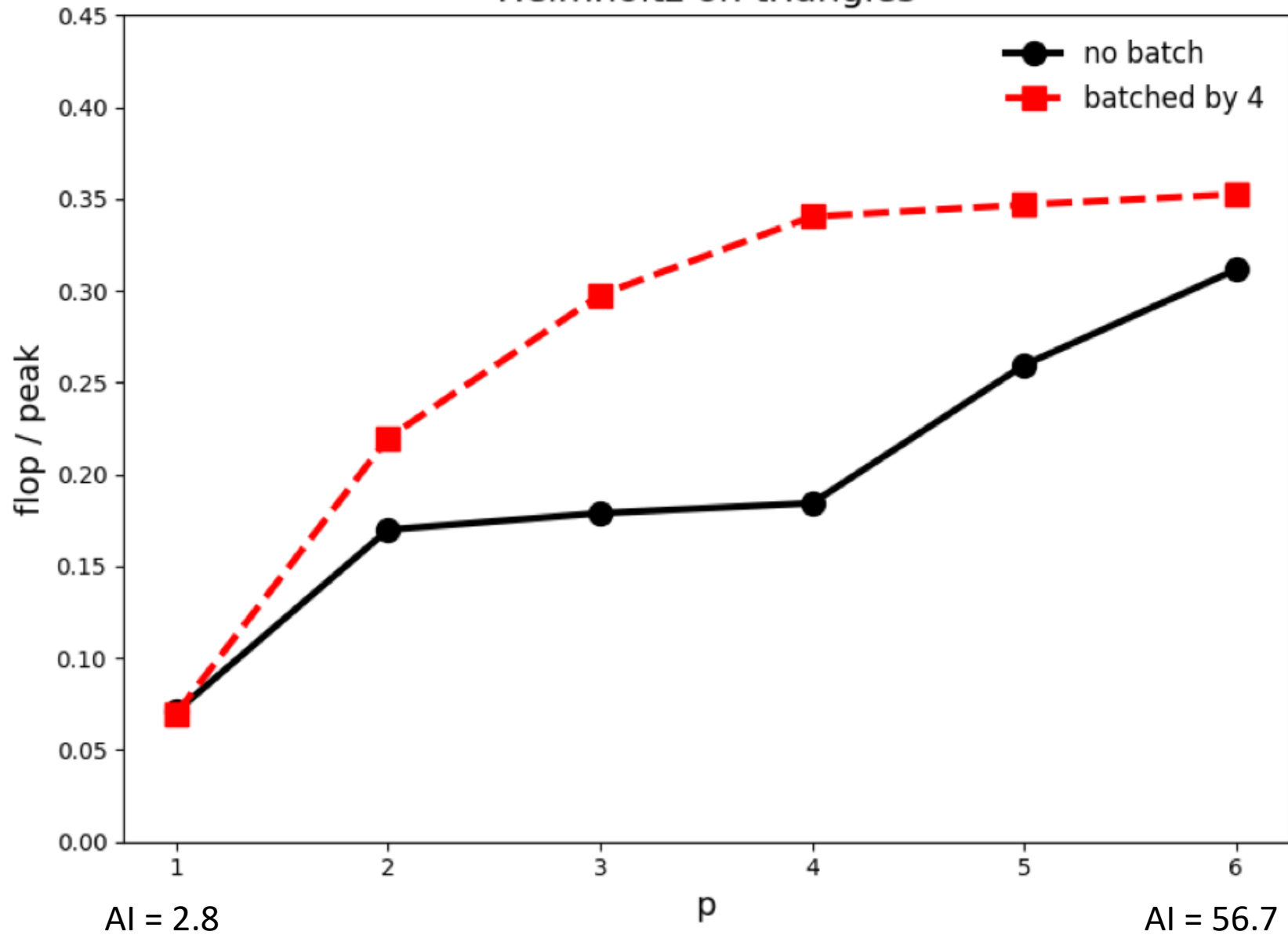
Helmholtz on hexahedra



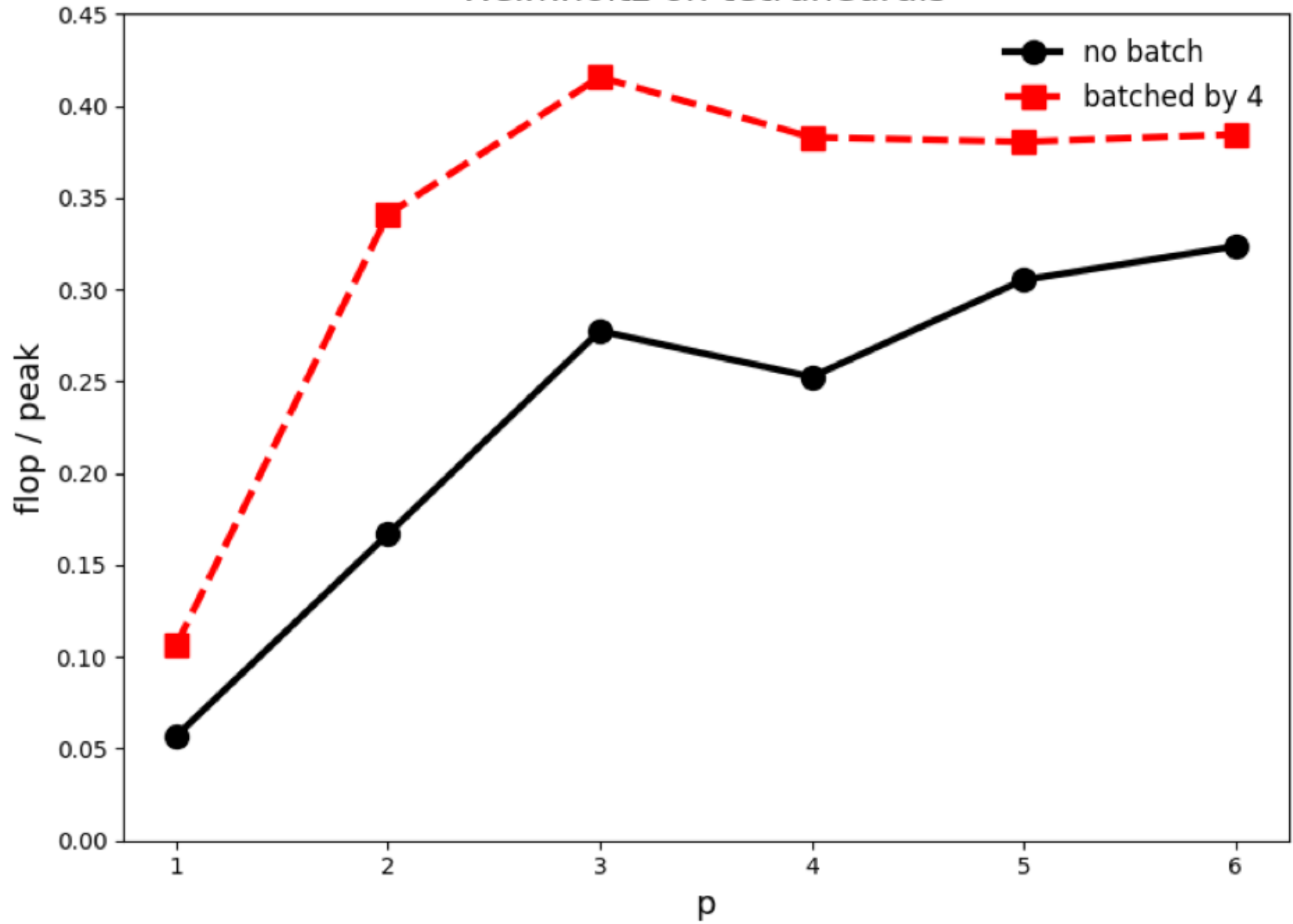
Helmholtz on quadrilaterals



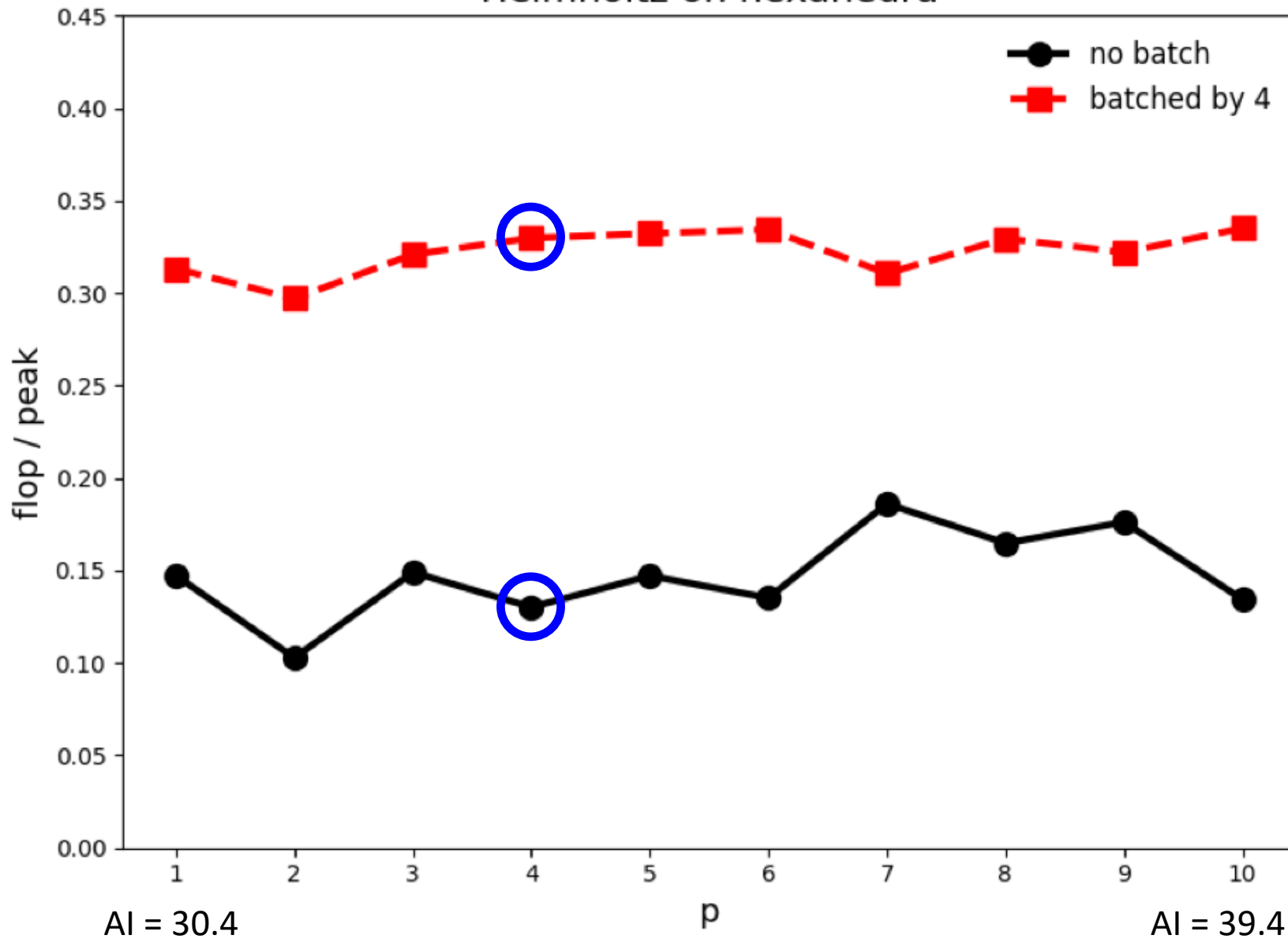
Helmholtz on triangles



Helmholtz on tetrahedrals



Helmholtz on hexahedra



Flop contributions by instruction types

■ AVX2 (4 doubles) ■ SSE (2 doubles) ■ scalar (1 double)

Only 0.4% of flops not vectorized

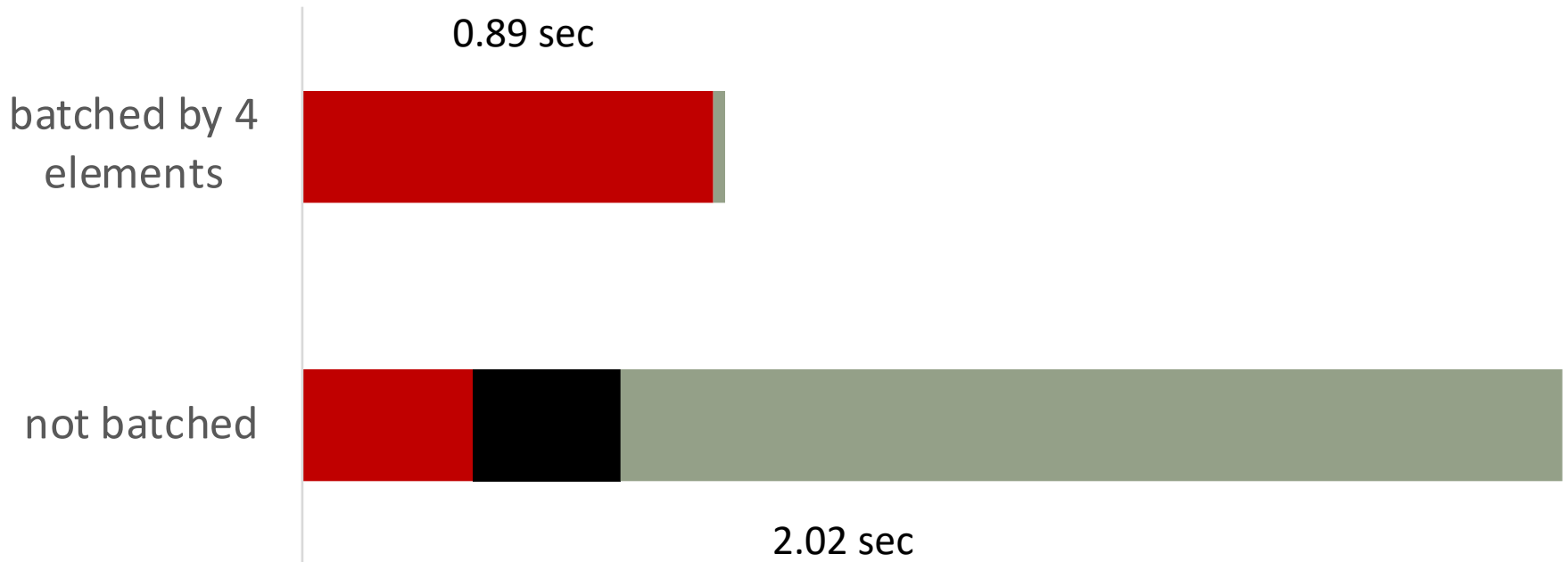
batched by 4 elements

not batched

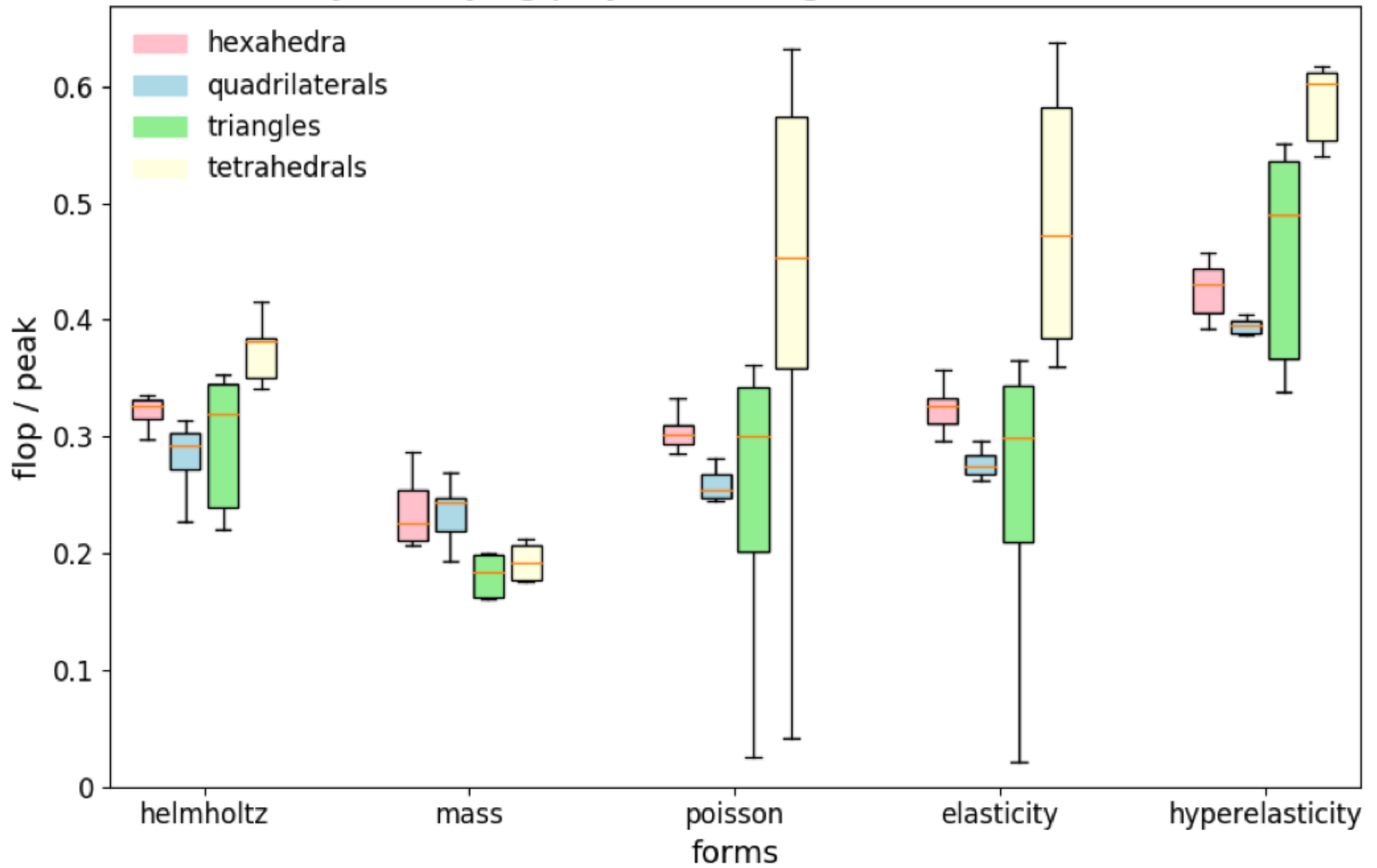


Instruction counts by instruction types

■ AVX2 (4 doubles) ■ SSE (2 doubles) ■ scalar (1 double)



batched by 4, varying polynomial degree for each form and mesh



To be continued...

- We are building an abstraction layer of loops (via loo.py)
- Pathway to GPUs
 - ...which requires a better performance model
- Try it out:
 - firedrake branch tsfc2loopy
 - tsfc branch tsfc2loopy
 - PyOP2 branch tsfc_loopying
 - loopy branch opaque-types



Firedrake


```

1  static double const form_t19[3] = ...
2  ...
3
4  void wrap_form00_cell_integral_otherwise(int const start, int const end, Mat const mat0,
5     double const *__restrict__ dat0, int const *__restrict__ map0)
6  {
7     double t2[3 * 3] __attribute__((aligned (64)));
8     ...
9
10    for (int n = start; n <= -1 + end; ++n)
11    {
12        for (int i13 = 0; i13 <= 2; ++i13)
13            for (int i14 = 0; i14 <= 1; ++i14)
14                t3[2 * i13 + i14] = dat0[2 * map0[3 * n + i13] + i14];
15        for (int i1 = 0; i1 <= 2; ++i1)
16            for (int i4 = 0; i4 <= 2; ++i4)
17                t2[3 * i1 + i4] = 0.0;
18        ...
19        for (int form_j_0 = 0; form_j_0 <= 2; ++form_j_0)
20            for (int form_ip_0 = 0; form_ip_0 <= 2; ++form_ip_0)
21            {
22                form_t23 = form_t22[3 * form_ip_0 + form_j_0] * form_t9[form_ip_0];
23                for (int form_k_0 = 0; form_k_0 <= 2; ++form_k_0)
24                    form_t24[3 * form_j_0 + form_k_0] += form_t22[3 * form_ip_0 + form_k_0] * form_t23;
25            }
26        ...
27        for (int form_k_2 = 0; form_k_2 <= 2; ++form_k_2)
28            for (int form_j_1 = 0; form_j_1 <= 2; ++form_j_1)
29                t2[3 * form_j_1 + form_k_2] += t2[3 * form_j_1 + form_k_2] + ...;
30
31        MatSetValuesBlockedLocal(mat0, 3, &(map0[3 * n]), 3, &(map0[3 * n]), &(t2[0]), ADD_VALUES);
32    }
33 }

```