Cross-Element Vectorization in Firedrake

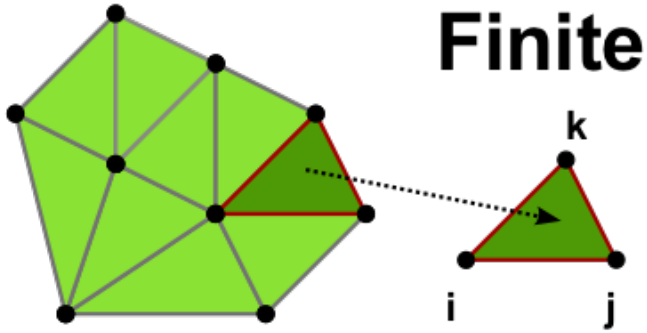
TJ SUN, LAWRENCE MITCHELL, DAVID HAM, PAUL H J KELLY

MARCH 2018

What is Firedrake

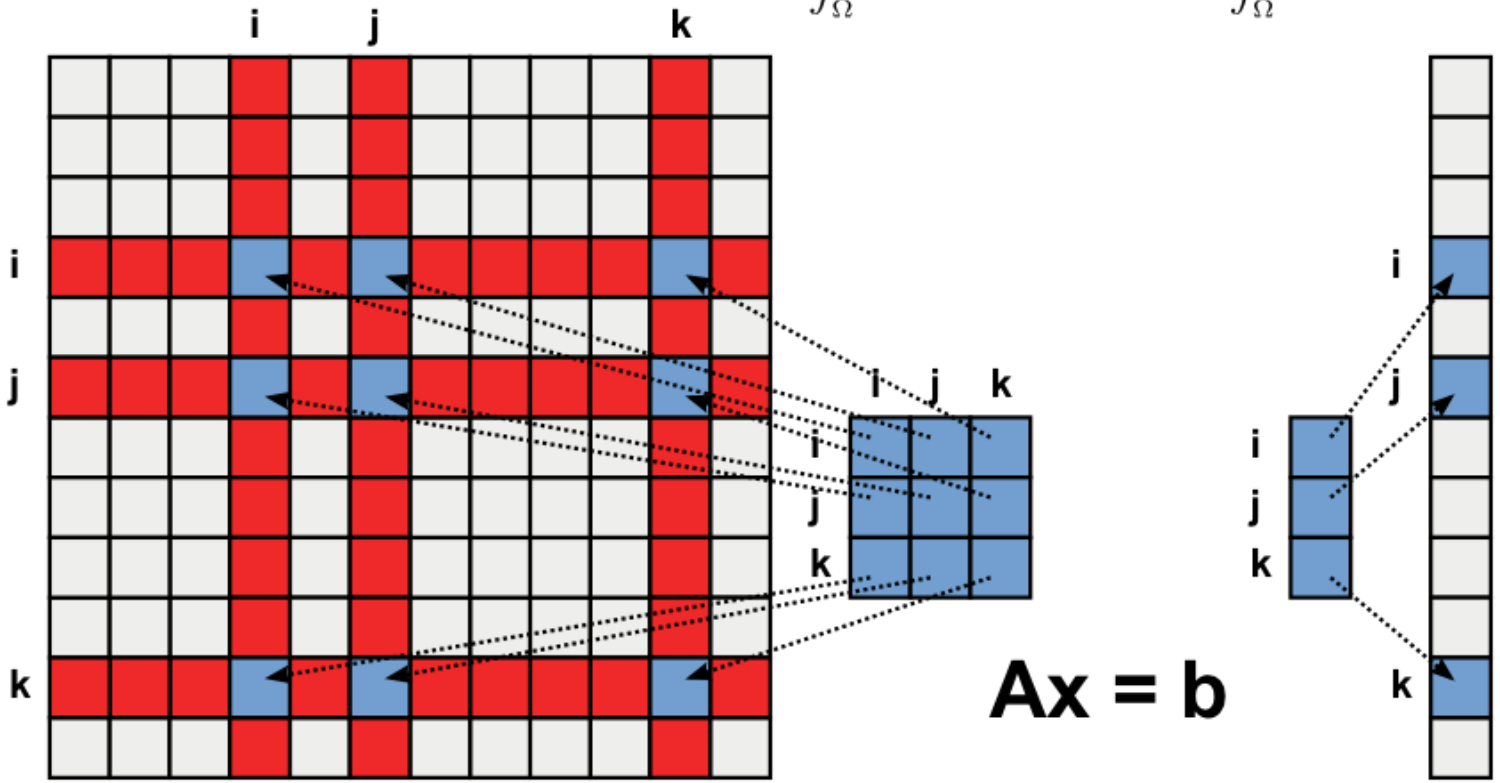
- We care about features and performance
  - Achieve this through abstraction layers and code generation
- Unstructured mesh
- Finite element ≈ integration ≈ assembly

Finite-element assembly



The weak form of the Helmholtz equation:

$$\int_\Omega \nabla v \cdot \nabla u - \lambda v u \, \mathrm{d}V = \int_\Omega v f \, \mathrm{d}V$$



Ax = b

Source: Rathgeber

```
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, Lagrange element of degree 3

```
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, Lagrange element of degree 3

```
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] =                                          t35;
16       A[1 + 2 * j(                                  ) + 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
```

Loop through all elements in the mesh

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

Wrapper

Action of linear elasticity operator on triangle mesh, Lagrange element of degree 3

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;
```

```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{
  /* ... */
  for (int ip = 0
  {
    /*
    for
    {
      A
      A
    }
  }
}
void wr
{
  /* ..
  for (in
  {
    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;

    form0_cell_integral_otherwise(t2, t3, t4);

    for (int i20 = 0; i20 <= 1; ++i20)
      for (int i19 = 0; i19 <= 9; ++i19)
        dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i20
  }
}
```

Action of linear elasticity operator on triangle mesh, Lagrange element of degree 3

```
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 + en
25   {
26     for (int i7 = 0; i7 <= 2; ++i7
27       for (int i8 = 0; i8 <= 1; ++i8)
28         t3[2 * i7 + i8] = dat1[2 *
29     for (int i13 = 0; i13 <= 9; ++
30       for (int i14 = 0; i14 <= 1;
31         t4[2 * i13 + i14] = dat2[2
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, Lagrange element of degree 3

```
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     for (int i20 = 0; i20 <= 1; ++i20)
30       for (int i19 = 0; i19 <= 9; ++i19)
31         dat0[2 * map0[10 * n + i19] + i20] += t2[2 * i19 + i20];
32
33
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 scattering of local matrix to global matrix

Action of linear elasticity operator on triangle mesh, Lagrange element of degree 3

```
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                                                                    * restrict
```

Kernel

Outer loop: contraction over quadrature points

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

  /* ...
```

inner loop over degrees of freedoms

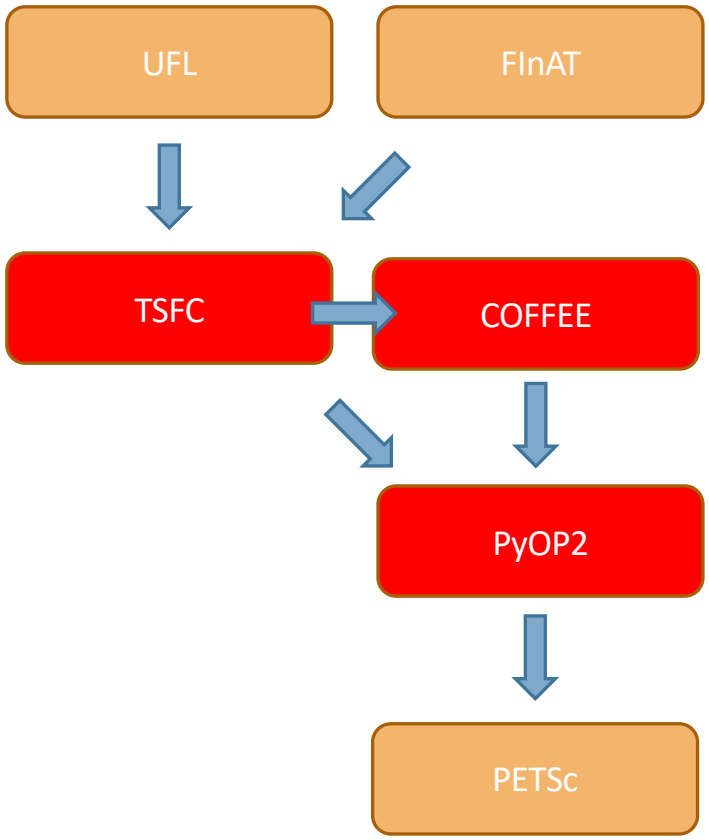
```
  for (int j0 = 0; j0 <= 9; ++j0)
  {
    A[2 * j0] += t18[10 * ip + j0] * t36 + t19[10 * ip + j0] * t35;
    A[1 + 2 * j0] += t18[10 * ip + j0] * t32 + t19[10 * ip + j0] * t31;
  }
}
```

```
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, Lagrange element of degree 3

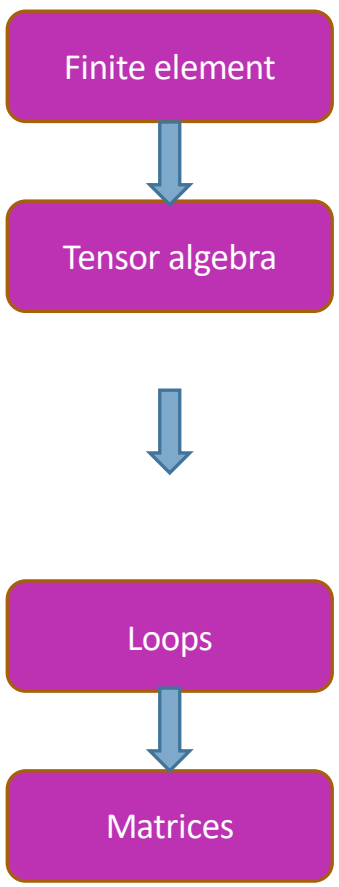
What's the problem with vectorization?

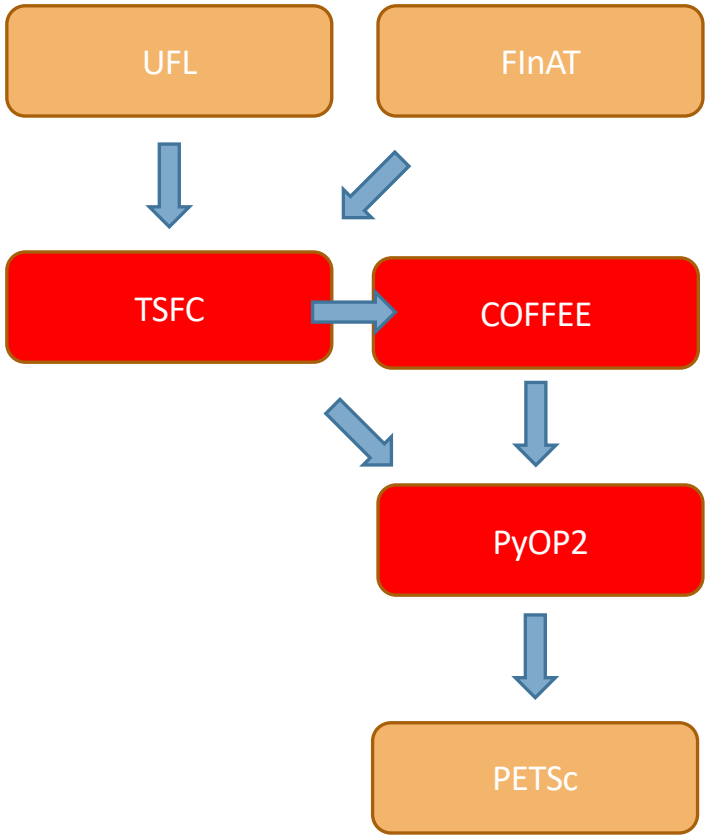
- SIMD width doubles every 4 years
  - AVX512 (2017) can do 8 doubles
- Vectorizing loops in one kernel (intra-kernel) is not easy
  - Trip count can be small and/or not multiple of SIMD width
  - Possible dependencies
  - Alignment to cache boundary
  - Stride 1 access
  - Varies with PDE, discretization, mesh
- Inter-kernel vectorization provides a generic solution
  - Vector-expand the kernel to act on N elements together, N=SIMD width
  - Can always do this systematically
  - Downside: increasing working size



Firedrake

Abstraction layers



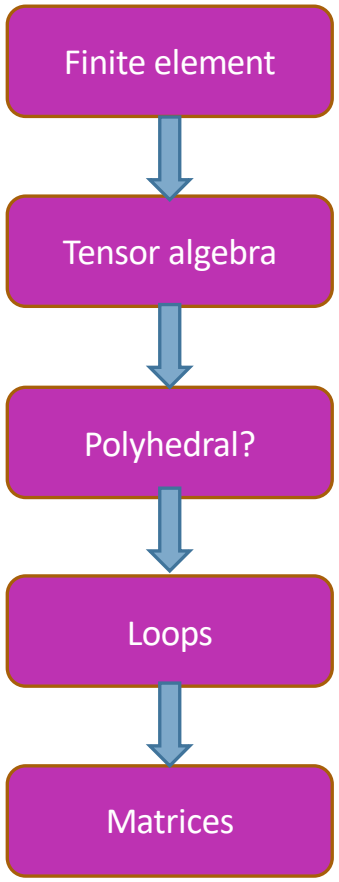


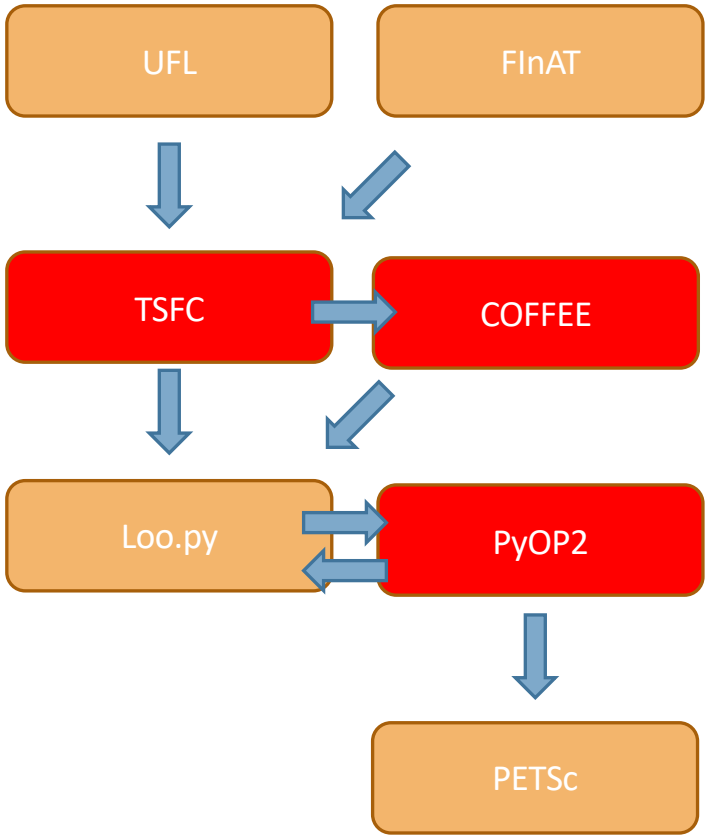
Firedrake

Introducing loo.py

- Andreas Klöckner et. al (UIUC)
- ≈ isl model of loops + transformations
- Not a blackbox
  - But handy if you tell it exactly what to do
- Support multiple backends
  - CPU
  - ISPC
  - OpenCL, PyOpenCL
  - Cuda

Abstraction layers





Firedrake

```
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[
16         A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
17       }
18   }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
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
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 =
13       for (int ele
14       {
15         A[elem + 4
16         A[elem + 4
17       }
18
19   }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
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)
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 *
47   }
48 }
```

Split n into n_outer and n_inner
Outer loop stride 4

36[elem] + t19[10 * ip + j0] * t35[
ip + j0] * t32[elem] + t19[10 * ip

```
for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
```

Wrapper

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
9    {
10     /* ... */
11
12 for (int i7 = 0; i7 <= 2; ++i7)
13   for (int i8 = 0; i8 <= 1; ++i8)
14     for (int n_inner = 0; n_inner <= 3; ++n_inner)
15       t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
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 for (int i1 = 0; i1 <= 9; ++i1)
21   for (int i2 = 0; i2 <= 1; ++i2)
22     for (int n_inner = 0; n_inner <= 3; ++n_inner)
23       t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
24
25
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
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 *
47   }
48 }
```

Gathering input data for 4 elements
Arrays are vector-expanded

Data for different elements packed to inner most dimension

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[
16         A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
17       }
18   }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
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;
30         for (int n_inner = 0; n
31           t3[8 * i7 + 4 * i8 +                                  outer + n_inner) + i7] + i8];
32     for (int i13 = 0; i13 <= 9; ++i13)
33       for (int i14 = 0;
34         for (int n_inner
35           t4[8 * i13 +                                          nner) + i13] + i14];
36     for (int i1 = 0; 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
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[
16         A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
17       }
18   }
19 }
20
21 void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
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
30         for (int n_inner
31           t3[8 * i7 + 4 *                                                    7] + i8];
```

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];
```

```
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 *
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[
16         A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
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

```
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 *
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[
16          A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
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

```
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 *
47      }
48    }
```

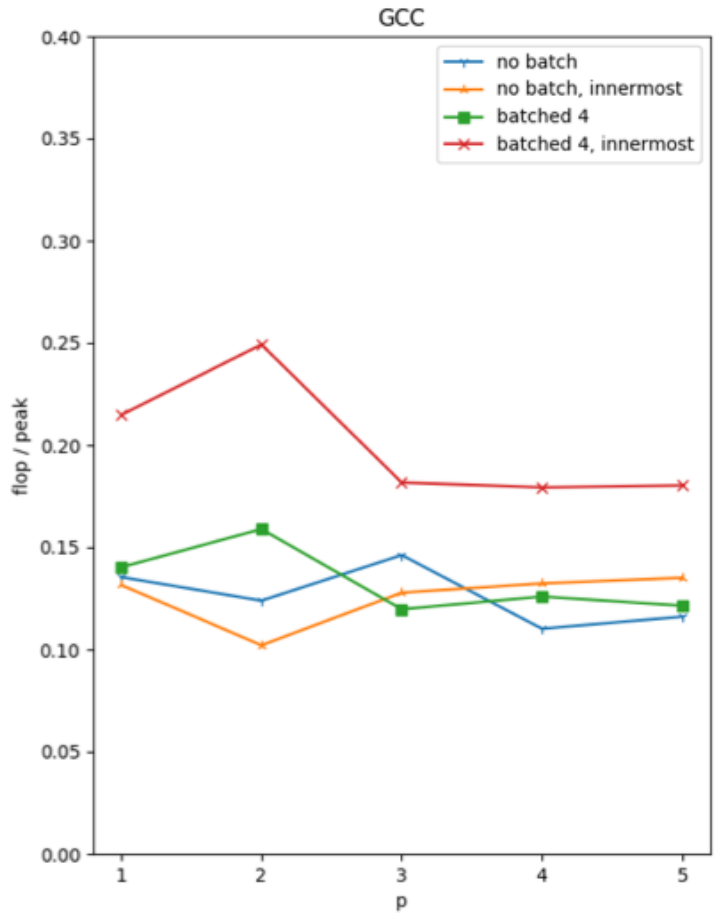
“easily vectorizable, right?”

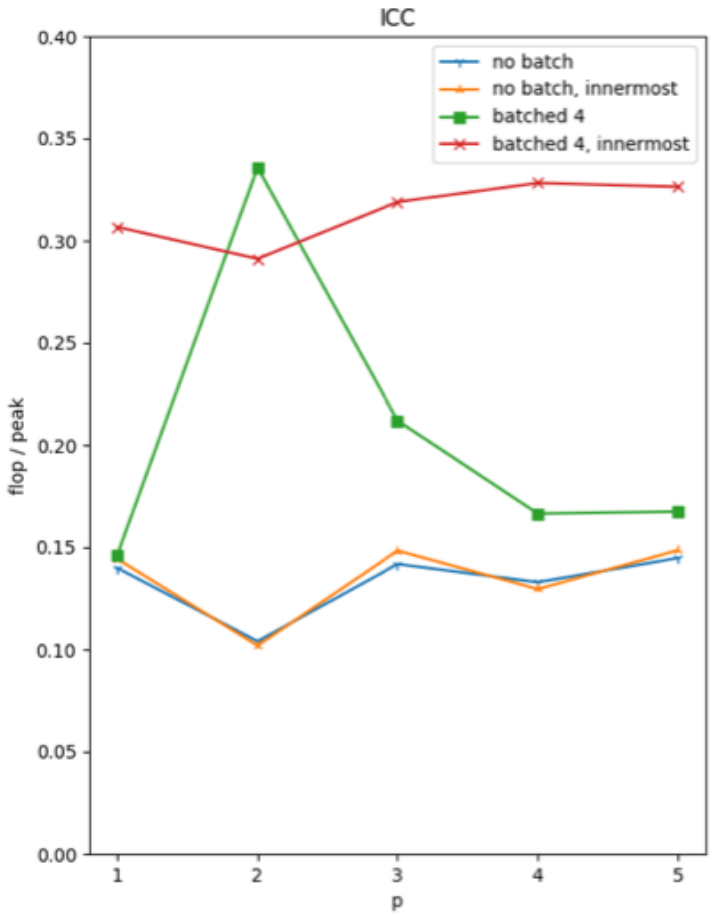
Action of linear elasticity operator on triangle mesh, batched by 4

Experimental setup

- Hardware: Haswell i7-4790 (single core measurement)
  - 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
- In reality we don’t have enough registers, so expect a bottleneck in L1 cache access to spilled values
- Mesh: hexagon (3D)
- Action of Helmholtz operator
  - AI (perfect cache) 6.5 to 33.6
- We present achieved flops / 57.6 Gflops

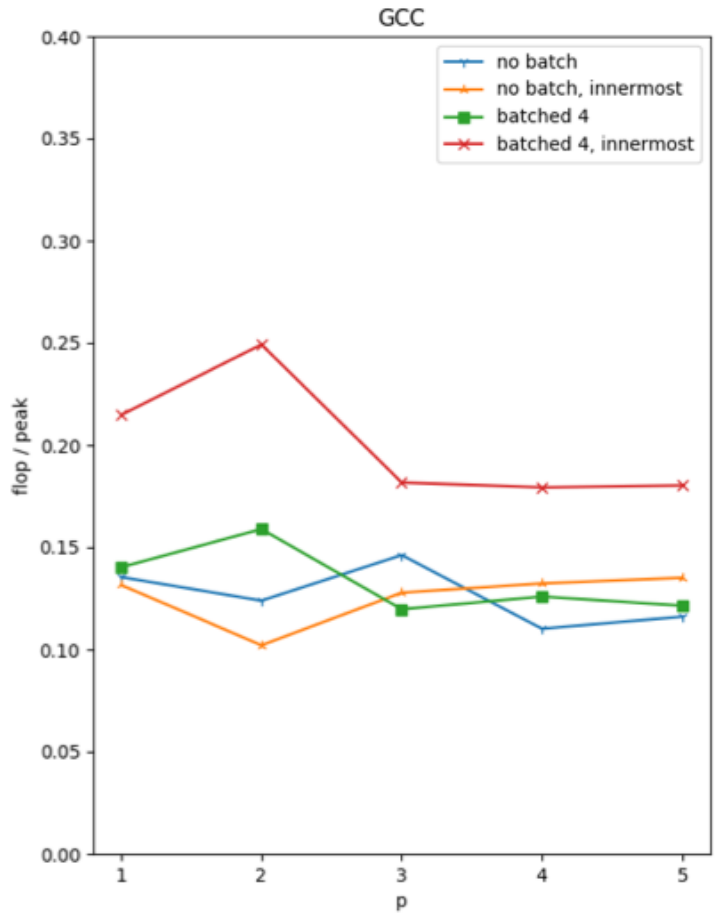
…need to tell compiler to vectorise the innermost loop

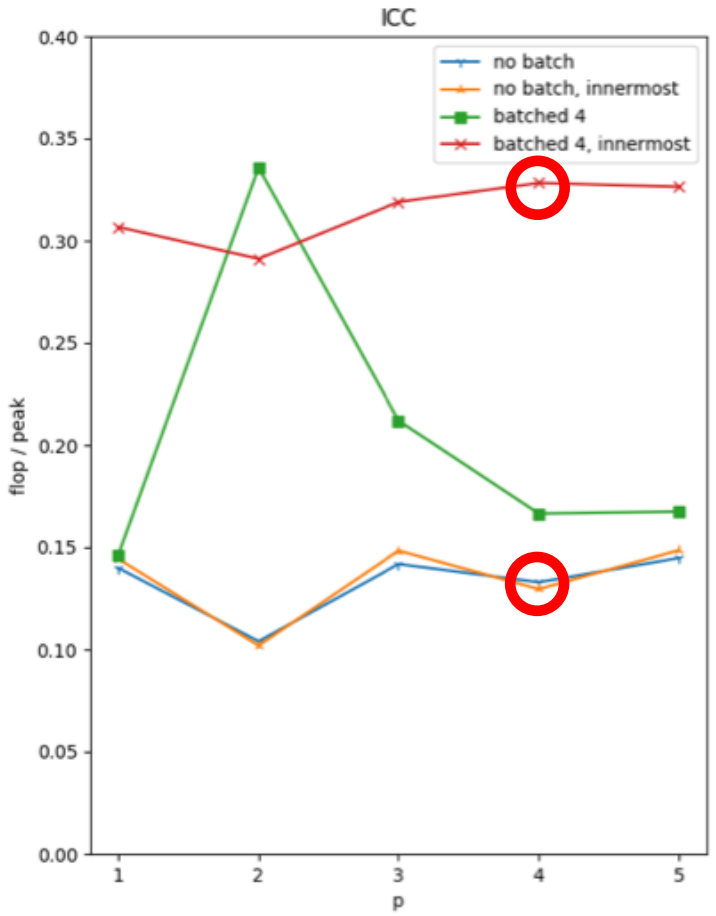




Higher polynomial degree (more degrees of freedoms)

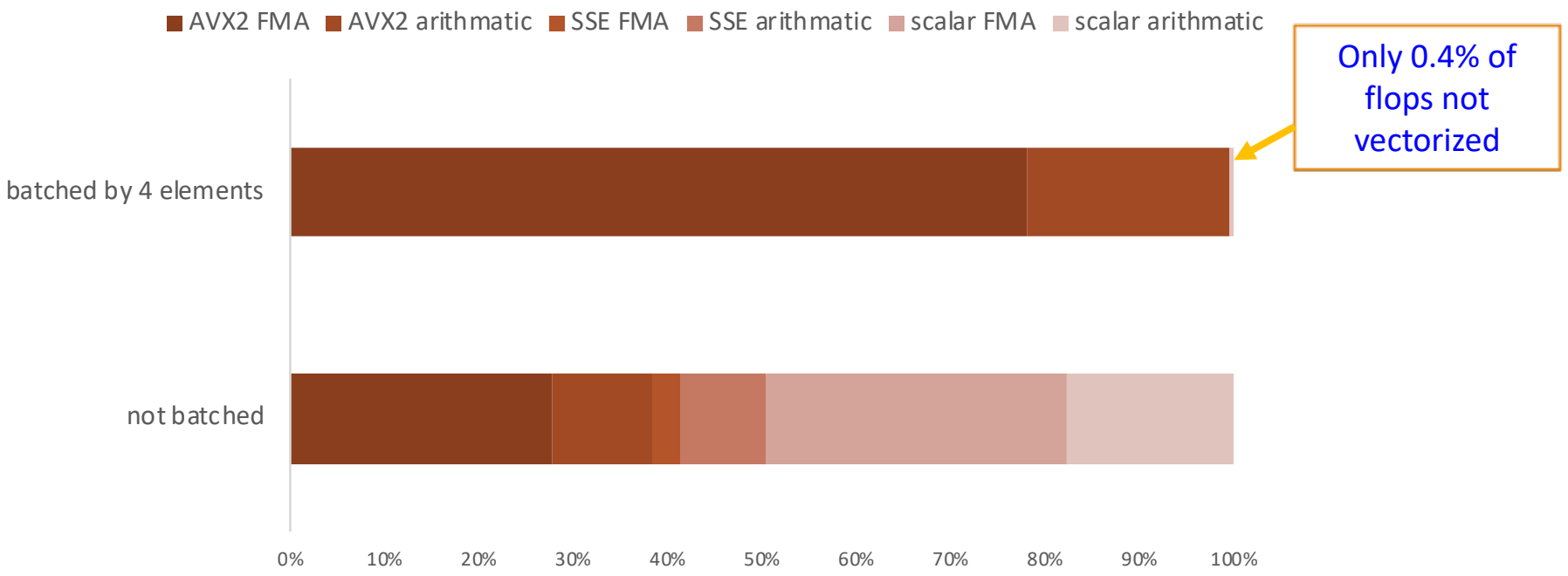
…need to tell compiler to vectorise the innermost loop





Higher polynomial degree (more degrees of freedoms)

Flop contribution by instruction types



Helmholtz on hexahedron, degree 4

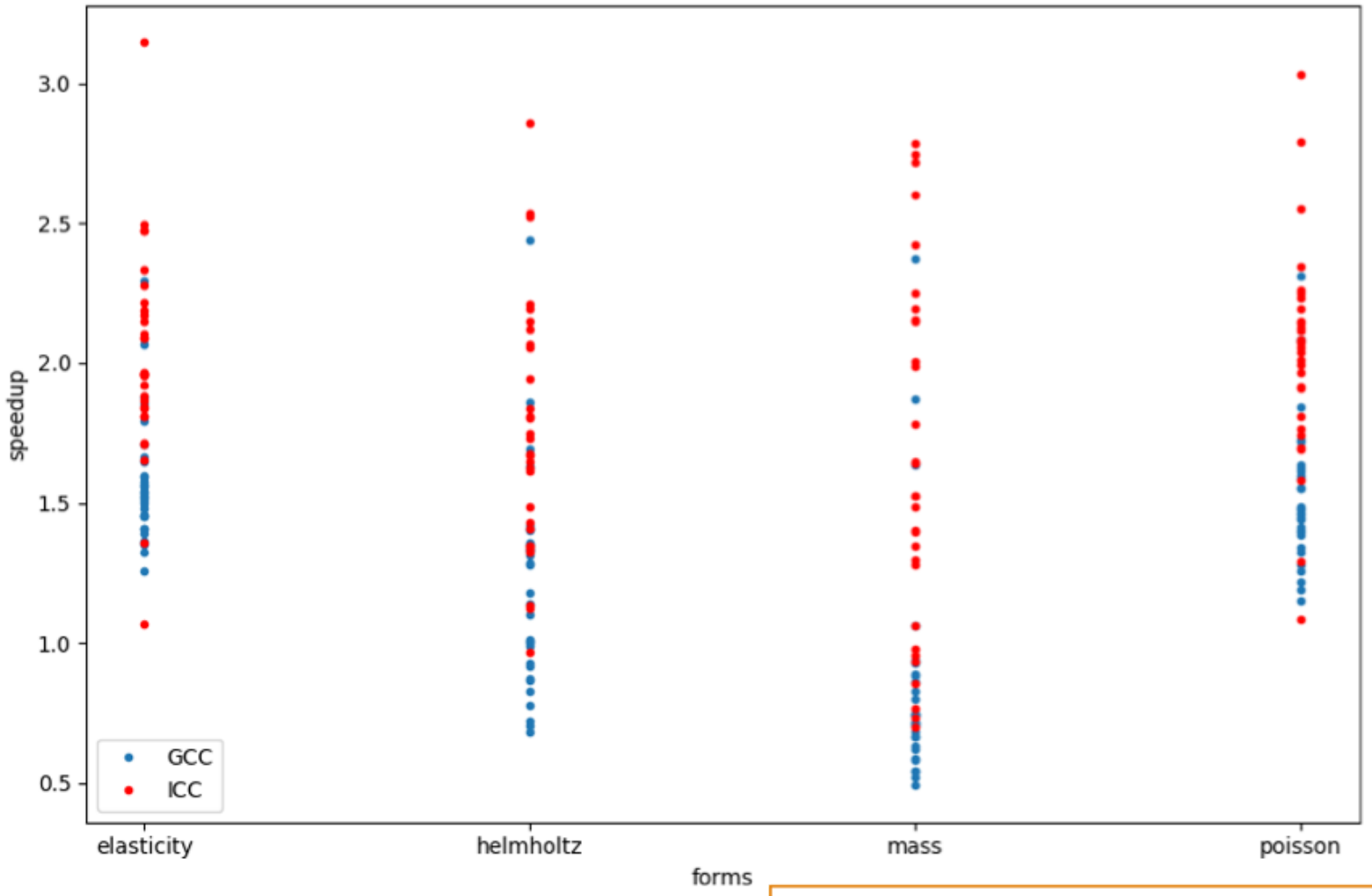
What we think so far

- Batch is effective way to utilize SIMD resources
  - If not outright better, certainly smoother
- Register pressure likely to be the problem now
  - But there might be low-hanging fruits
- Tell compilers to vectorize the inner most loop
- Haswell is quite old, oddities might disappear on more recent architecture

To be continued…

- Help the compiler with register allocation?
- Trade-off between compute and storage
  - Currently we minimize flop
- Bigger picture: what we really aiming at is the abstraction of loops
  - Cross-element vectorization is almost a by-product
  - Pathway to GPUs (and beyond)…
  - … which requires a performance model and/or autotuning

Speedup vs not batched



Various forms, polynomial degrees, meshes