Cross-Element Vectorization in Firedrake

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

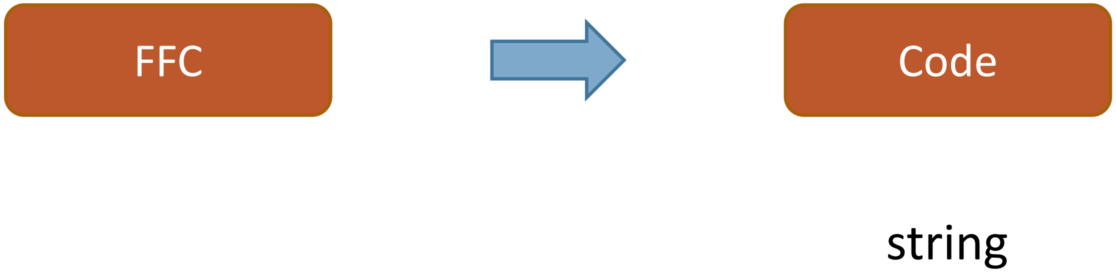
June 2018

loo.py as our new backend

Introducing loo.py

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

Code generation for assembly

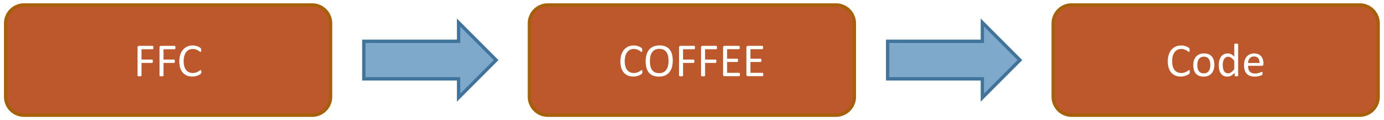


FFC

Code

string

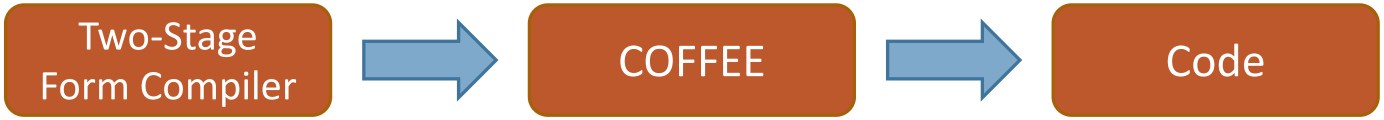
Code generation for assembly



AST

String

Code generation for assembly

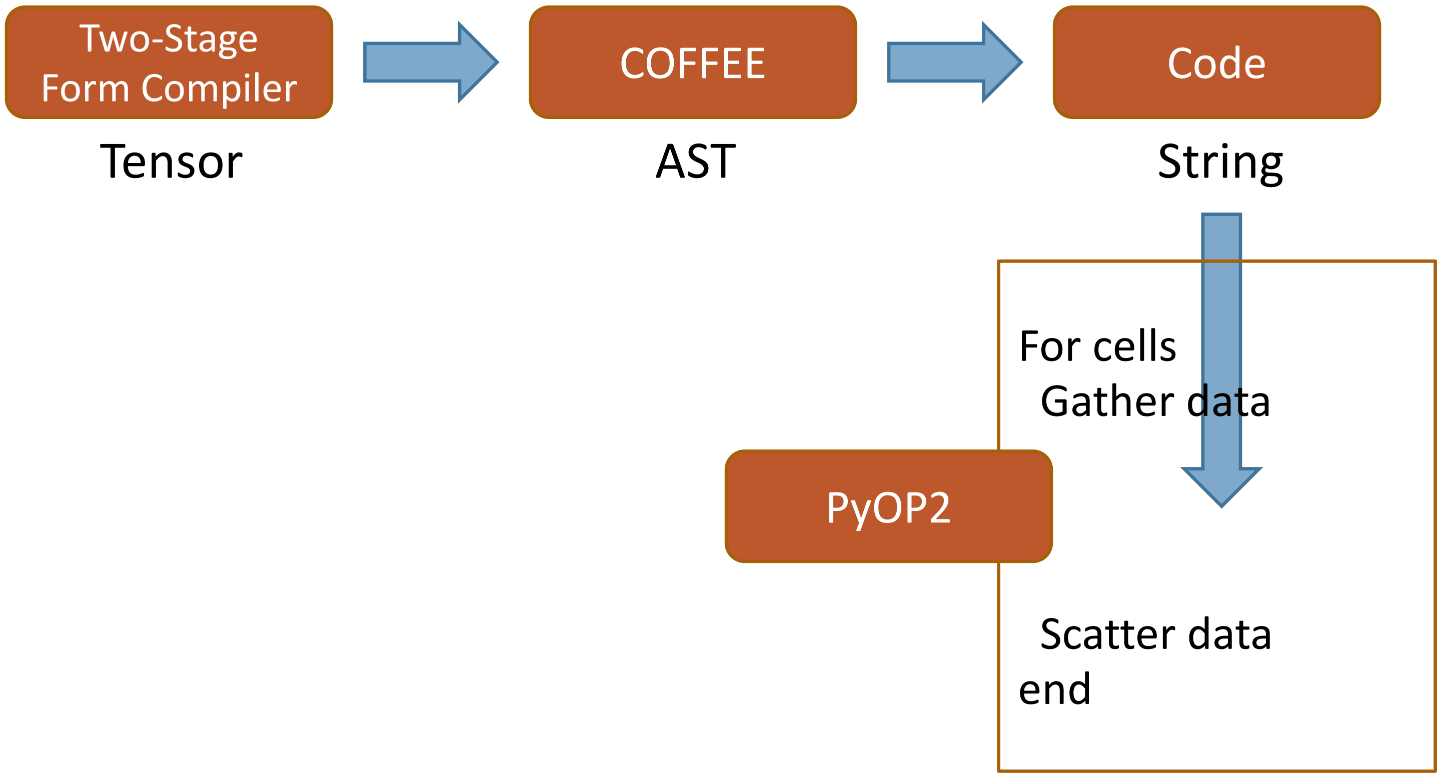


Tensor

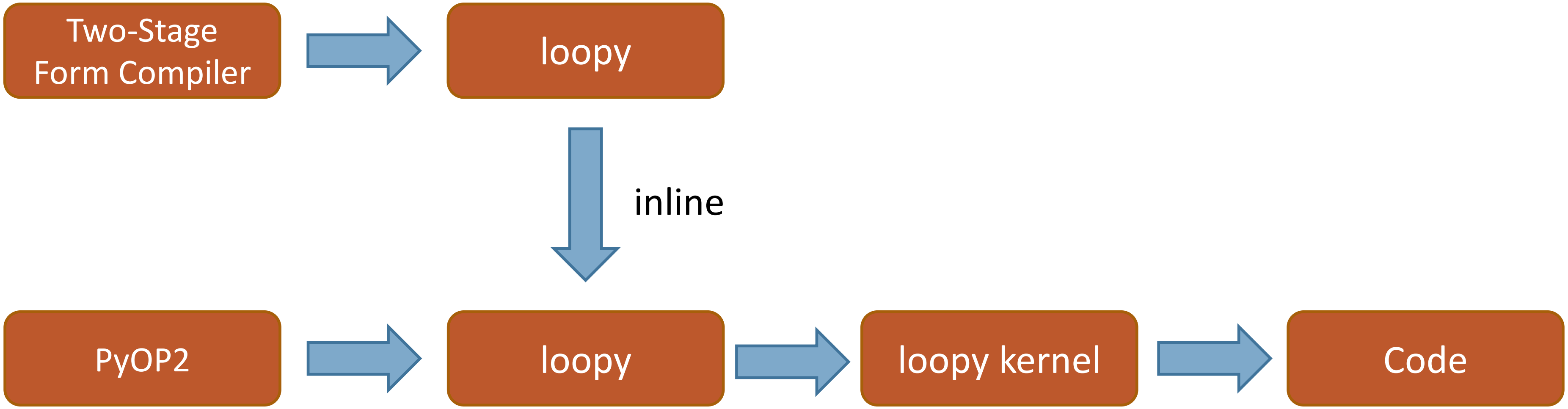
AST

String

Code generation for assembly



Code generation for assembly



Two-Stage Form Compiler

loopy

inline

PyOP2

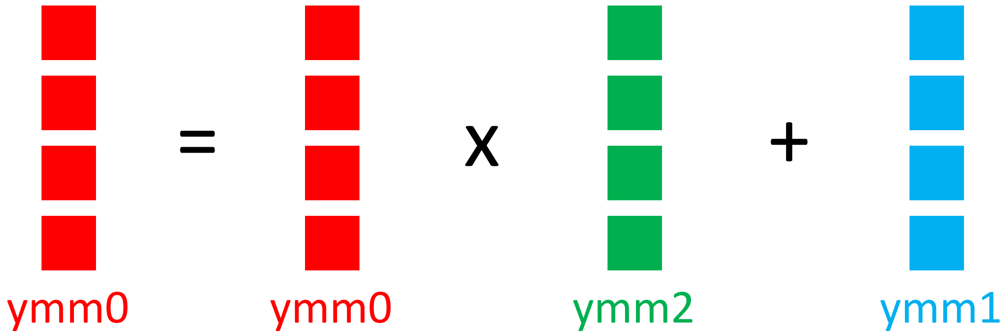
loopy

loopy kernel

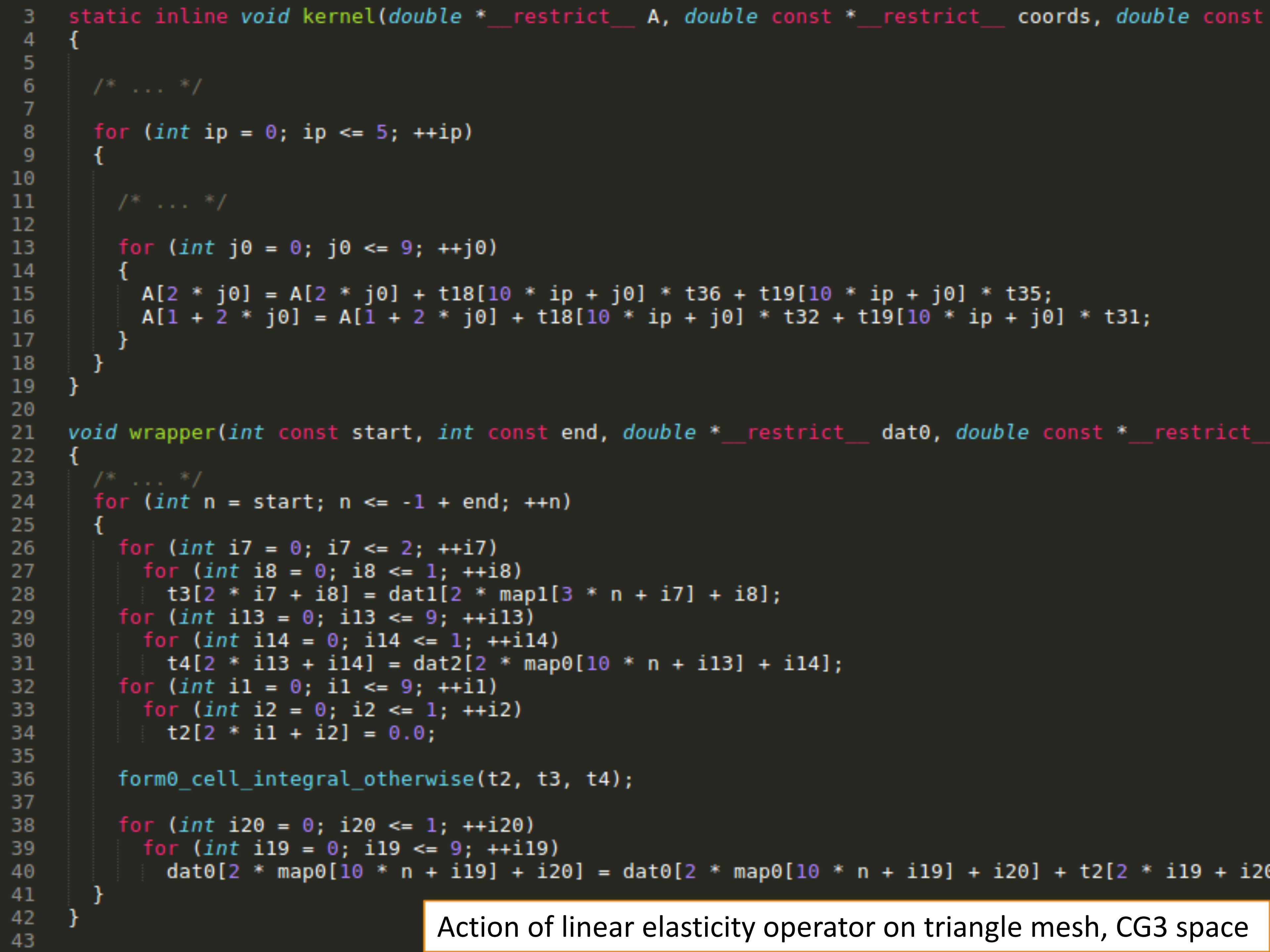
Code

What is vectorization

- SIMD (single instruction multiple data) programming model
- e.g. VFMADD213PD ymm0, ymm1, ymm2 (in AVX2 instruction set)

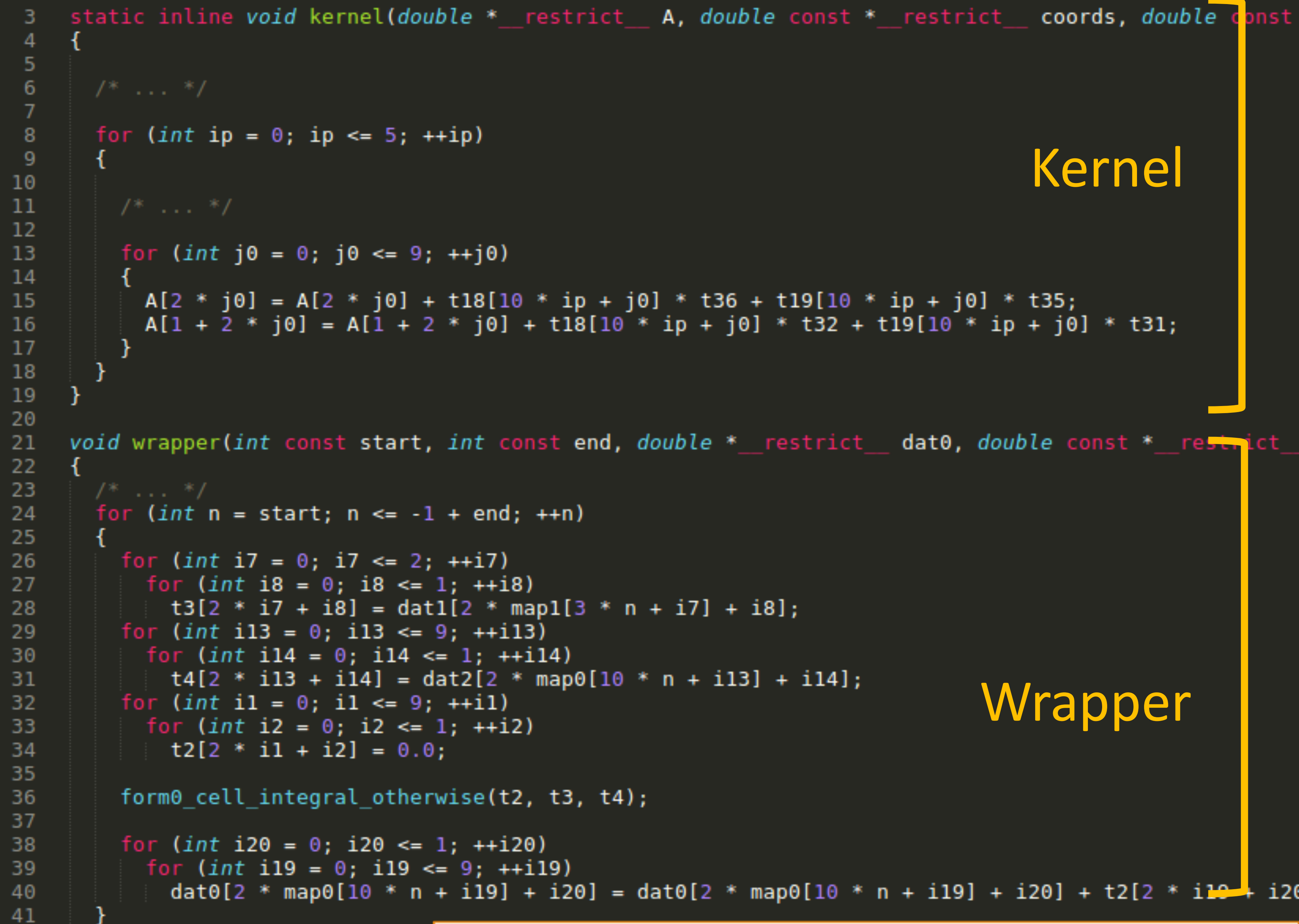


- 8 double precision operations with 1 instruction
- Need to issue 2 FMA instructions in 1 cycle to get advertised performance
- SIMD width doubles every 4 years
  - AVX512 instructions can do 8 doubles
- Naïve code usually achieves <10% peak performance



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

  /* ... */

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

    /* ... */

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

void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict_
{
  /* ... */
  for (int n = start; n <= -1 + end; ++n)
  {
    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 + i2
  }
}
```

Action of linear elasticity operator on triangle mesh, CG3 space

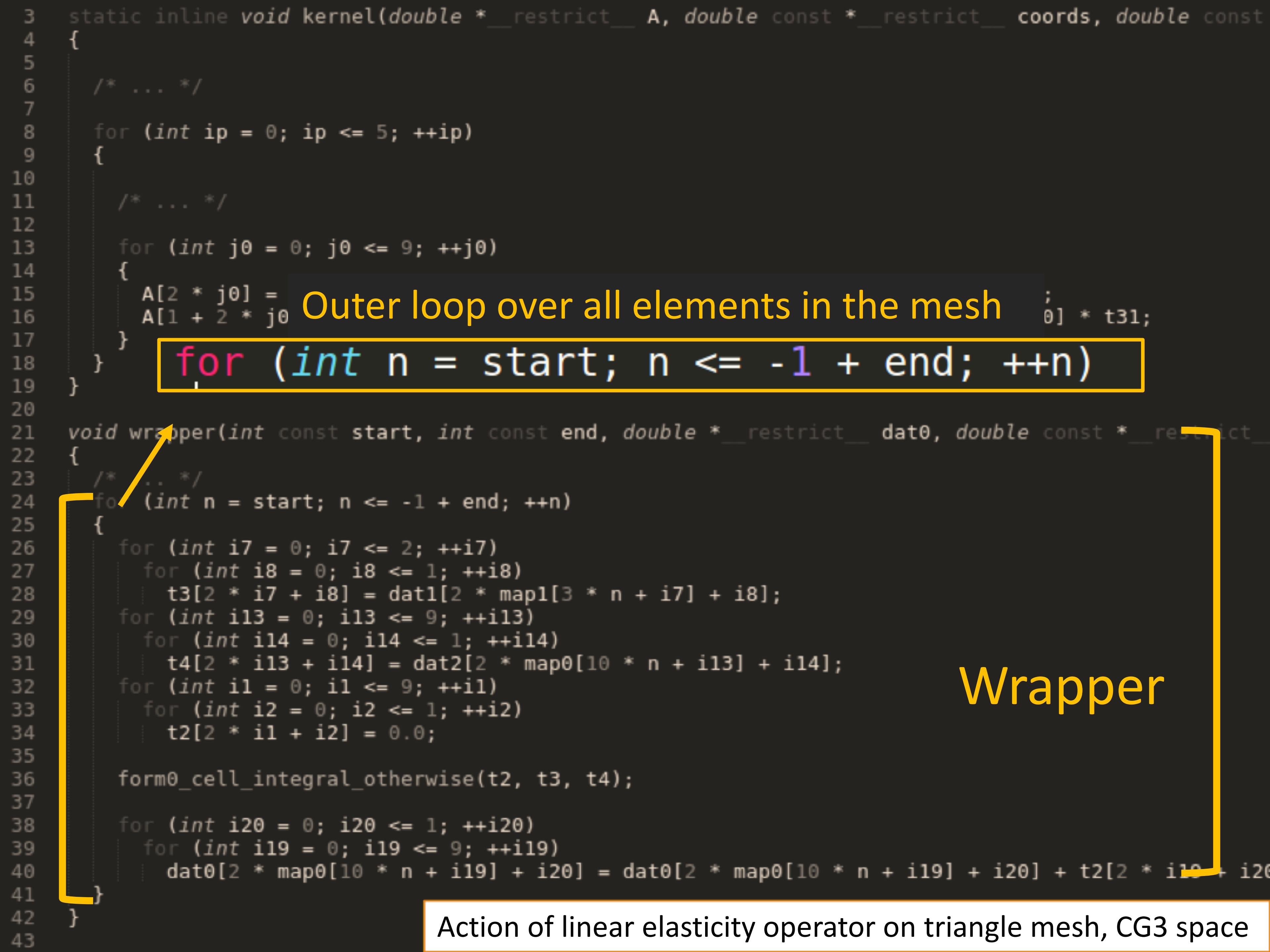


```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

  /* ... */

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

    /* ... */

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

void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
{
  /* ... */
  for (int n = start; n <= -1 + end; ++n)
  {
    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
  }
}
```

Kernel

Wrapper

Action of linear elasticity operator on triangle mesh, CG3 space



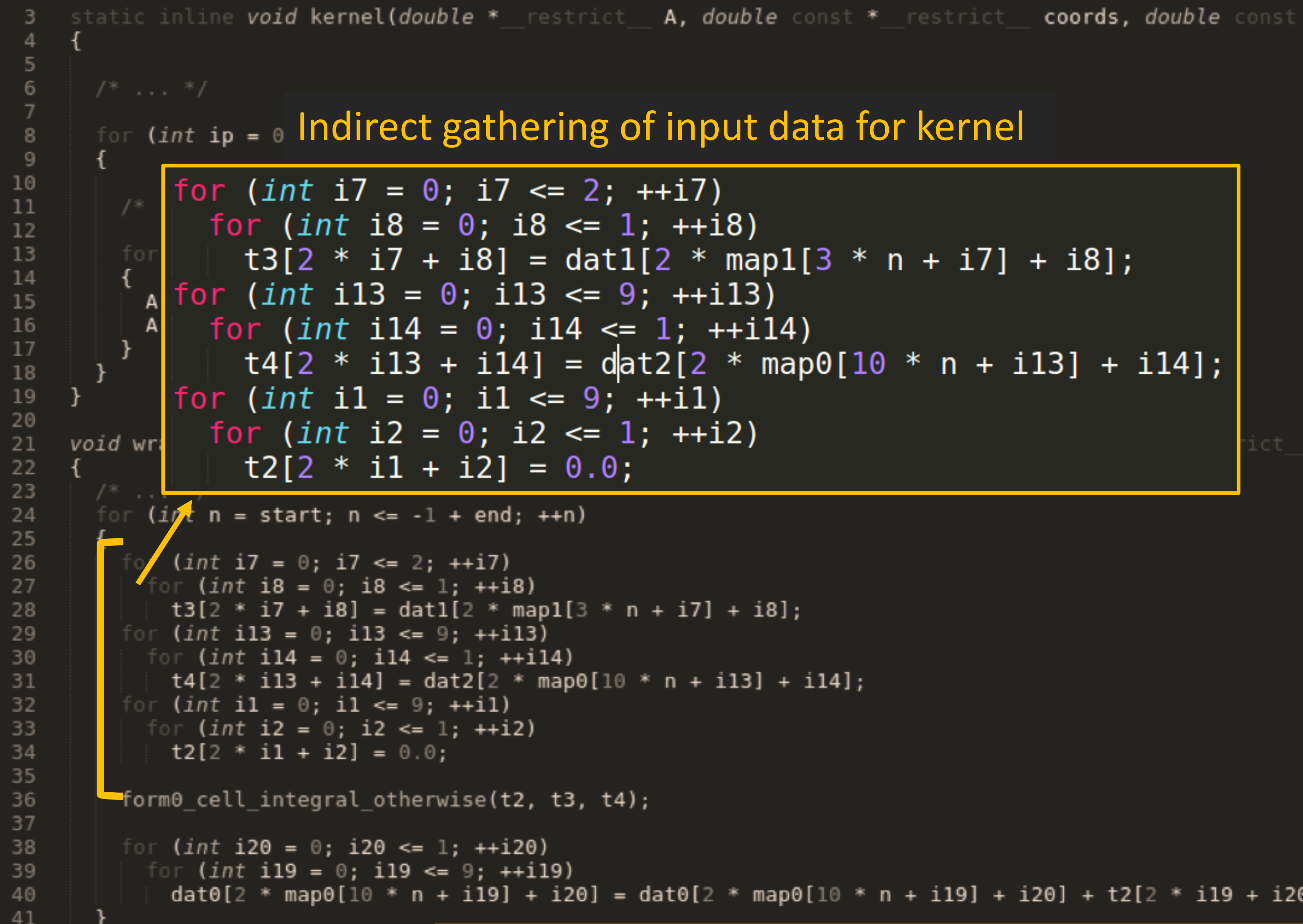
```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

  /* ... */

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

    /* ... */

    for (int j0 = 0; j0 <= 9; ++j0)
    {
      A[2 * j0] =                                                          ;
      A[1 + 2 * j0                                                         0] * t31;
    }
  }
}

void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
{
  /* ... */
  for (int n = start; n <= -1 + end; ++n)
  {
    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 + i2
  }
}
```

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

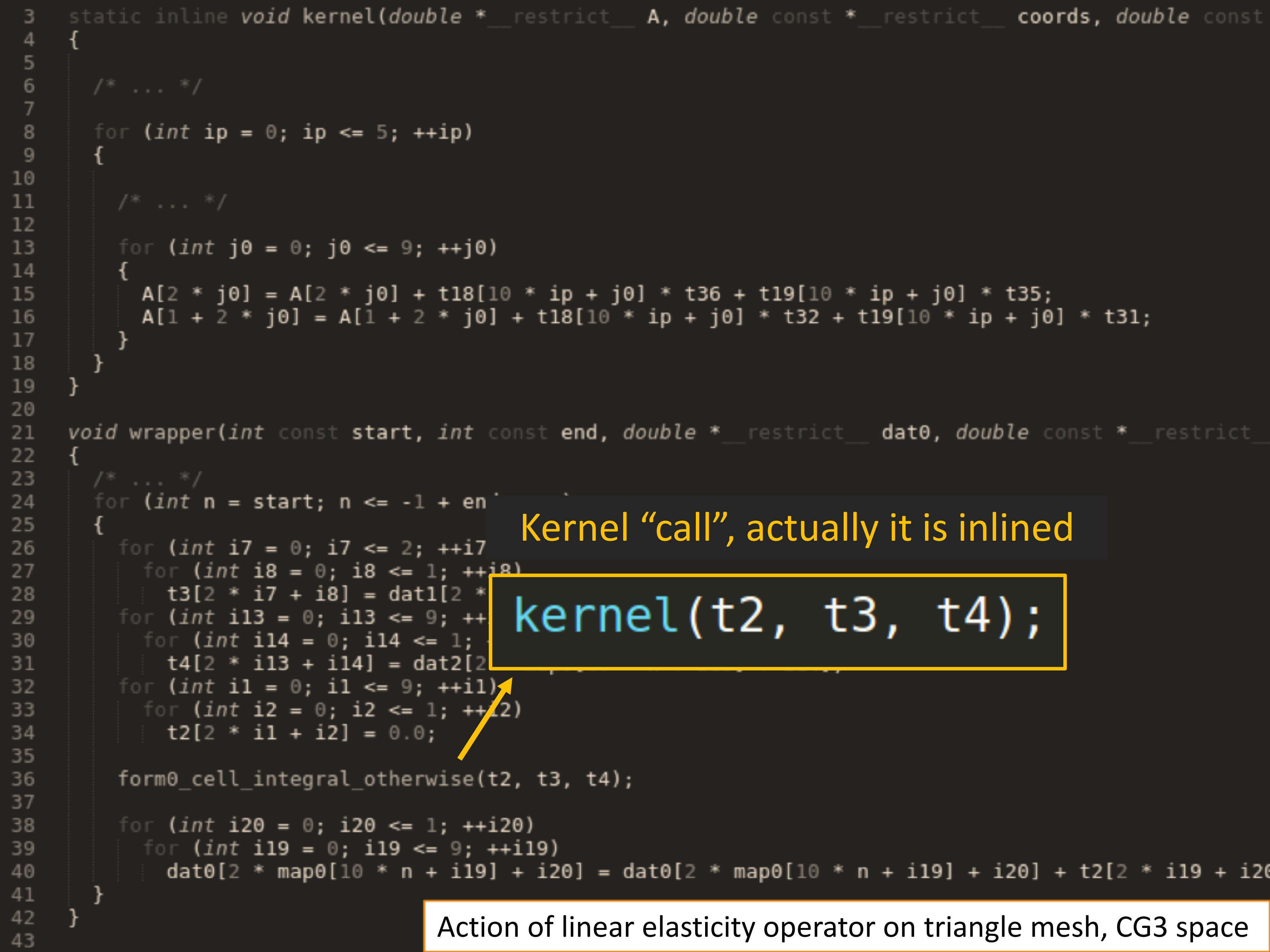


Indirect gathering of input data for kernel

```
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                                                              rict_
{
  /* ..
  for (int n = start; n <= -1 + end; ++n)
  {
    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 + i2
  }
}
```

Action of linear elasticity operator on triangle mesh, CG3 space

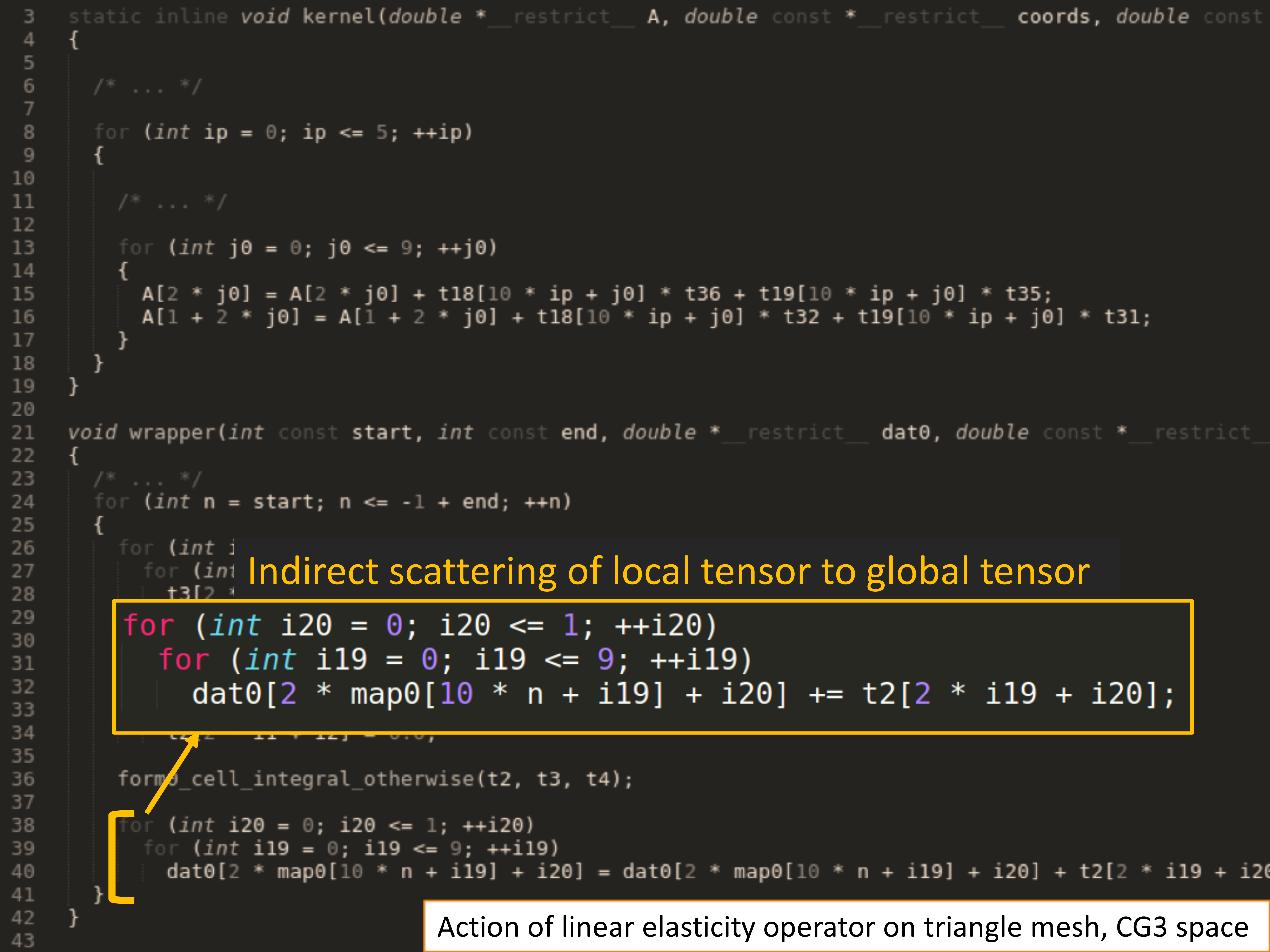


```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

 /* ... */

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

  /* ... */

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

void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict__
{
 /* ... */
 for (int n = start; n <= -1 + en
 {
  for (int i7 = 0; i7 <= 2; ++i7
   for (int i8 = 0; i8 <= 1; ++i8)
    t3[2 * i7 + i8] = dat1[2 *
  for (int i13 = 0; i13 <= 9; ++
   for (int i14 = 0; i14 <= 1;
    t4[2 * i13 + i14] = dat2[2
  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 + i2
 }
}
```

Kernel “call”, actually it is inlined

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

Action of linear elasticity operator on triangle mesh, CG3 space

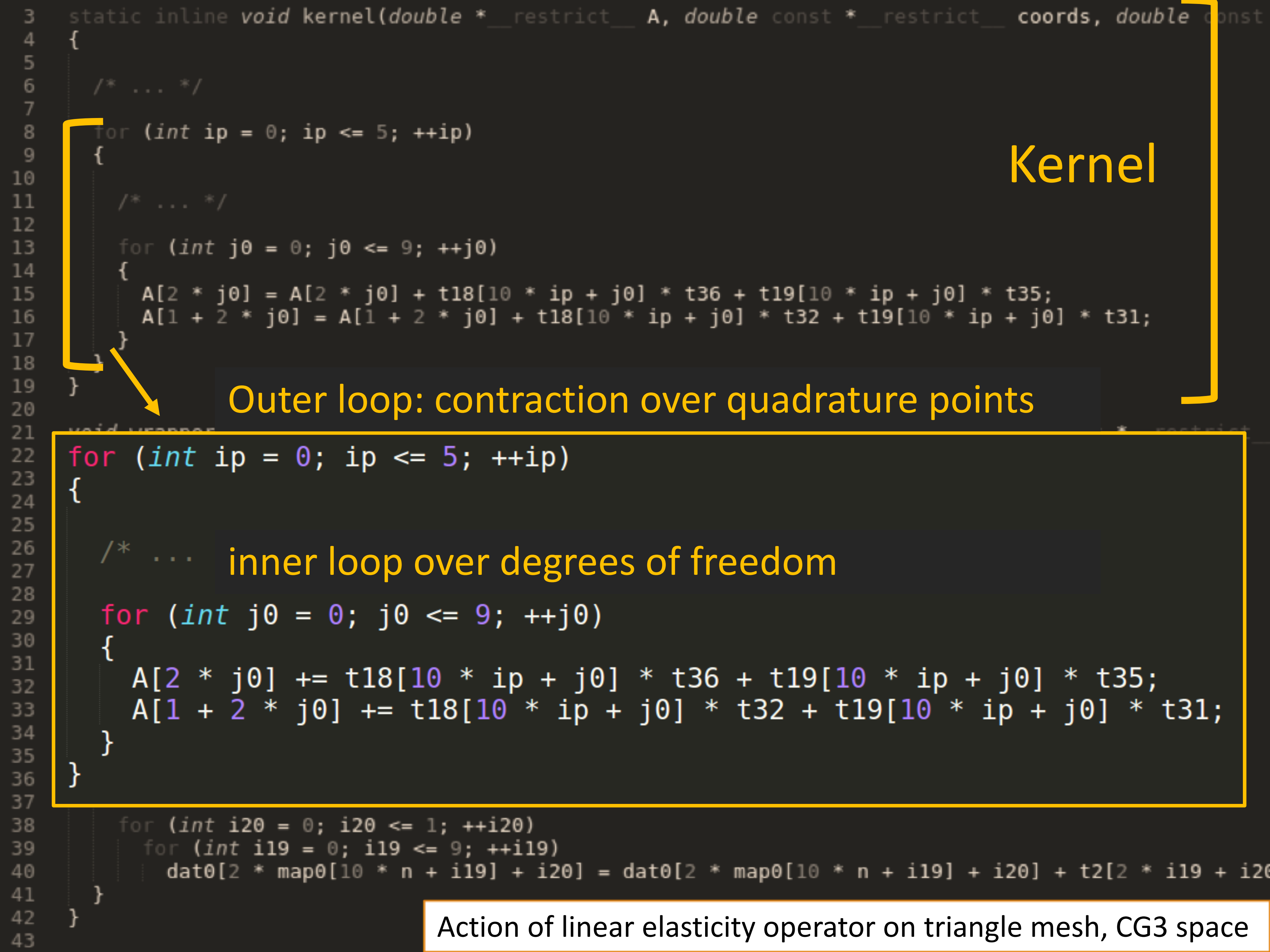


```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

  /* ... */

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

    /* ... */

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

void wrapper(int const start, int const end, double *__restrict__ dat0, double const *__restrict_
{
  /* ... */
  for (int n = start; n <= -1 + end; ++n)
  {
    for (int i
      for (int
        t3[2 *





        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 + i2
  }
}
```

Indirect scattering of local tensor to global tensor

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

Action of linear elasticity operator on triangle mesh, CG3 space



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const
{

  /* ... */

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

    /* ... */

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

void wrapper                                                              * restrict
```

Kernel

Outer loop: contraction over quadrature points

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

  /* ...

  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;
  }
}
```

inner loop over degrees of freedom

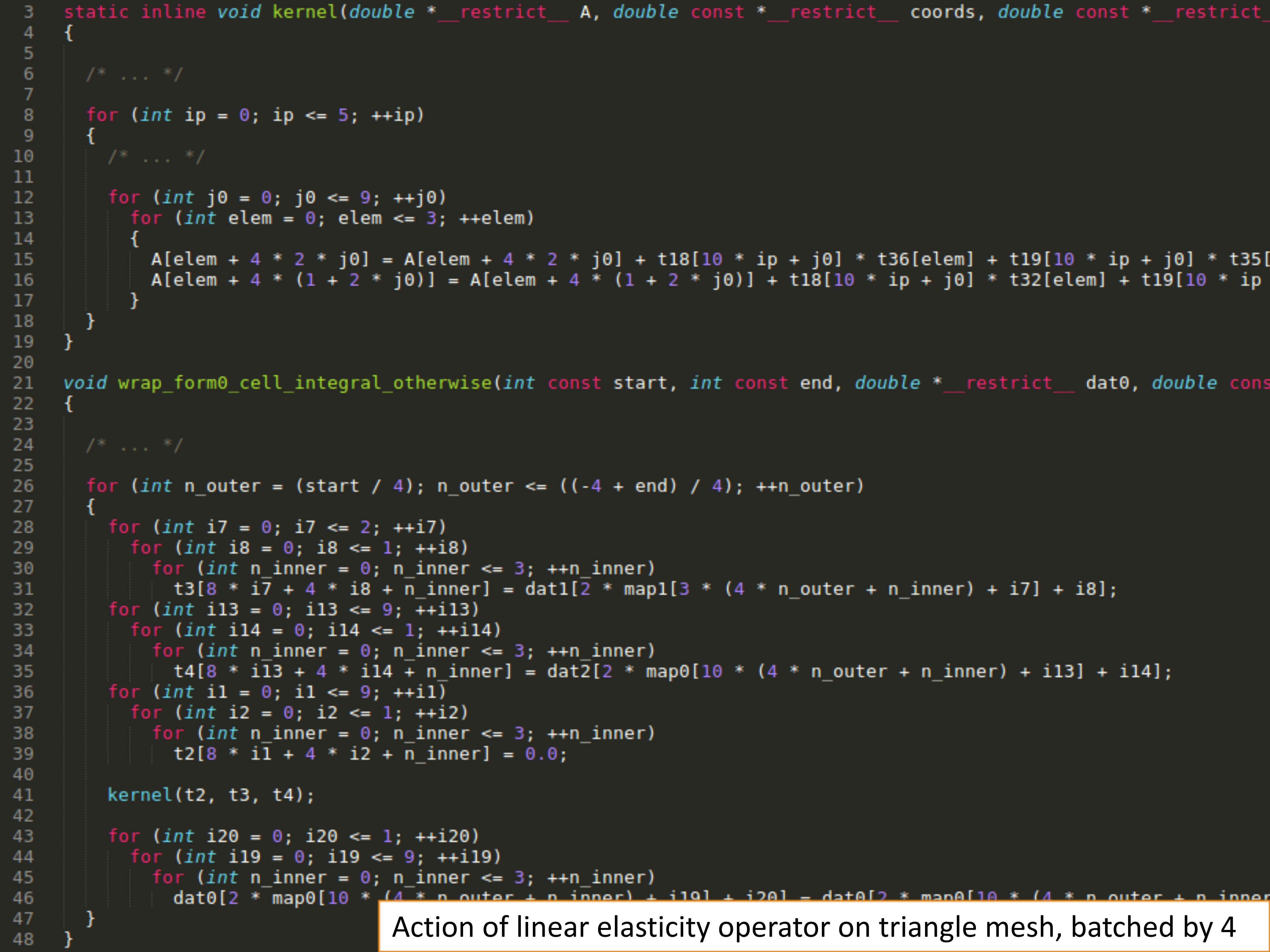
```
    for (int i20 = 0; i20 <= 1; ++i20)
      for (int i19 = 0; i19 <= 9; ++i19)
        dat0[2 * map0[10 * n + i19] + i20] = dat0[2 * map0[10 * n + i19] + i20] + t2[2 * i19 + i2
  }
}
```

Action of linear elasticity operator on triangle mesh, CG3 space

Vectorization strategy

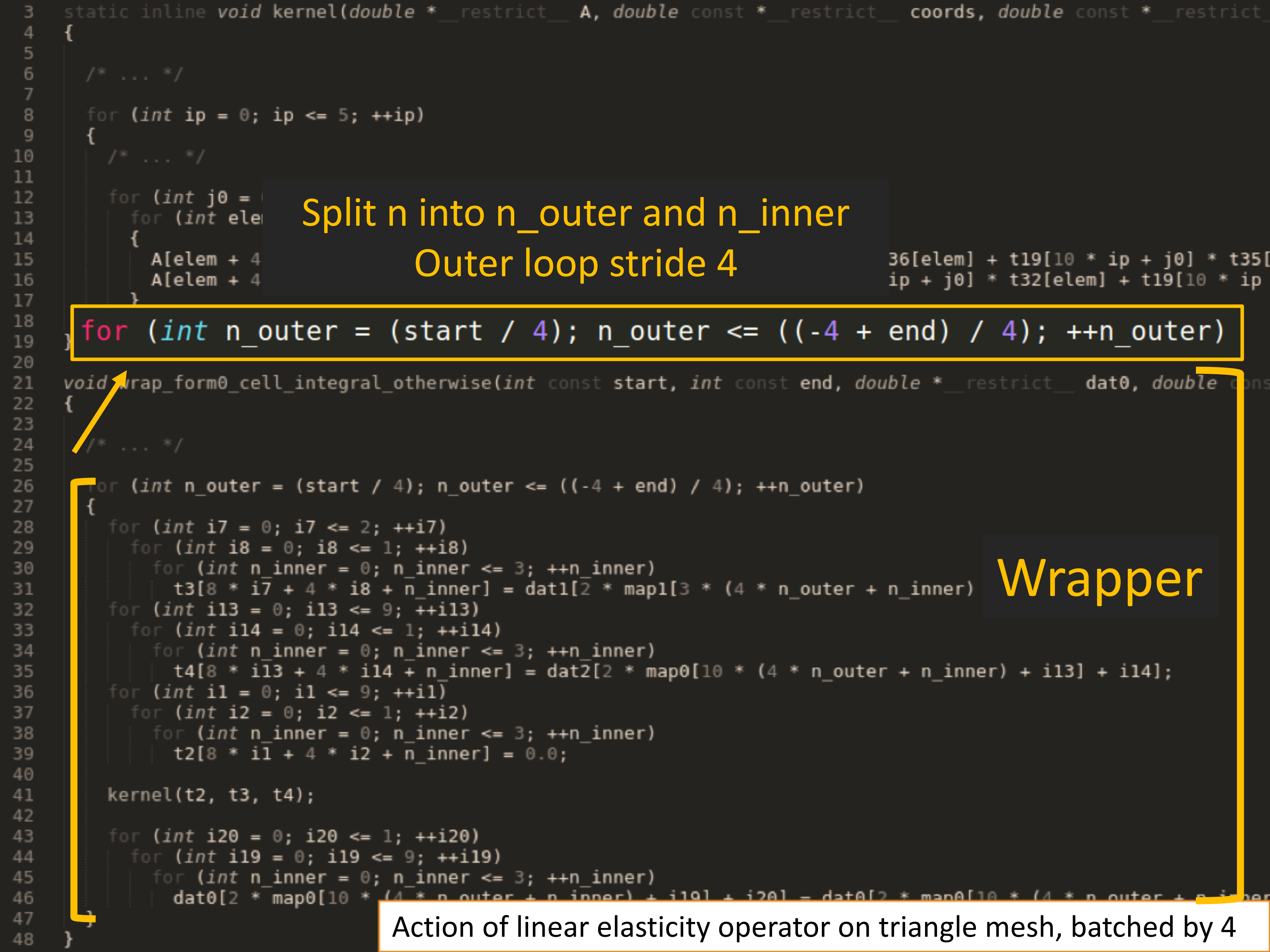
- “Intra-kernel” can be tricky
  - Trip count can be small and/or not multiple of SIMD width
  - Alignment to cache boundary
  - Stride 1 access
  - Operations outside of innermost loop not vectorized
  - Loop structure varies with PDE, discretization, mesh
  - And we have done many of these in firedrake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

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



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

  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] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[
        A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
      }
  }
}

void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
{

  /* ... */

  for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
  {
    for (int i7 = 0; i7 <= 2; ++i7)
      for (int i8 = 0; i8 <= 1; ++i8)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
    for (int i13 = 0; i13 <= 9; ++i13)
      for (int i14 = 0; i14 <= 1; ++i14)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
    for (int i1 = 0; i1 <= 9; ++i1)
      for (int i2 = 0; i2 <= 1; ++i2)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

    kernel(t2, t3, t4);

    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] = dat0[2 * map0[10 * (4 * n_outer + n_inner
  }
}
```

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



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

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

    for (int j0 =
      for (int ele
      {
        A[elem + 4                                            36[elem] + t19[10 * ip + j0] * t35[
        A[elem + 4                                            ip + j0] * t32[elem] + t19[10 * ip
      }

}

void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
{

  /* ... */

  for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
  {
    for (int i7 = 0; i7 <= 2; ++i7)
      for (int i8 = 0; i8 <= 1; ++i8)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner)
    for (int i13 = 0; i13 <= 9; ++i13)
      for (int i14 = 0; i14 <= 1; ++i14)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
    for (int i1 = 0; i1 <= 9; ++i1)
      for (int i2 = 0; i2 <= 1; ++i2)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

    kernel(t2, t3, t4);

    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] = dat0[2 * map0[10 * (4 * n_outer + n_inner
  }
}
```

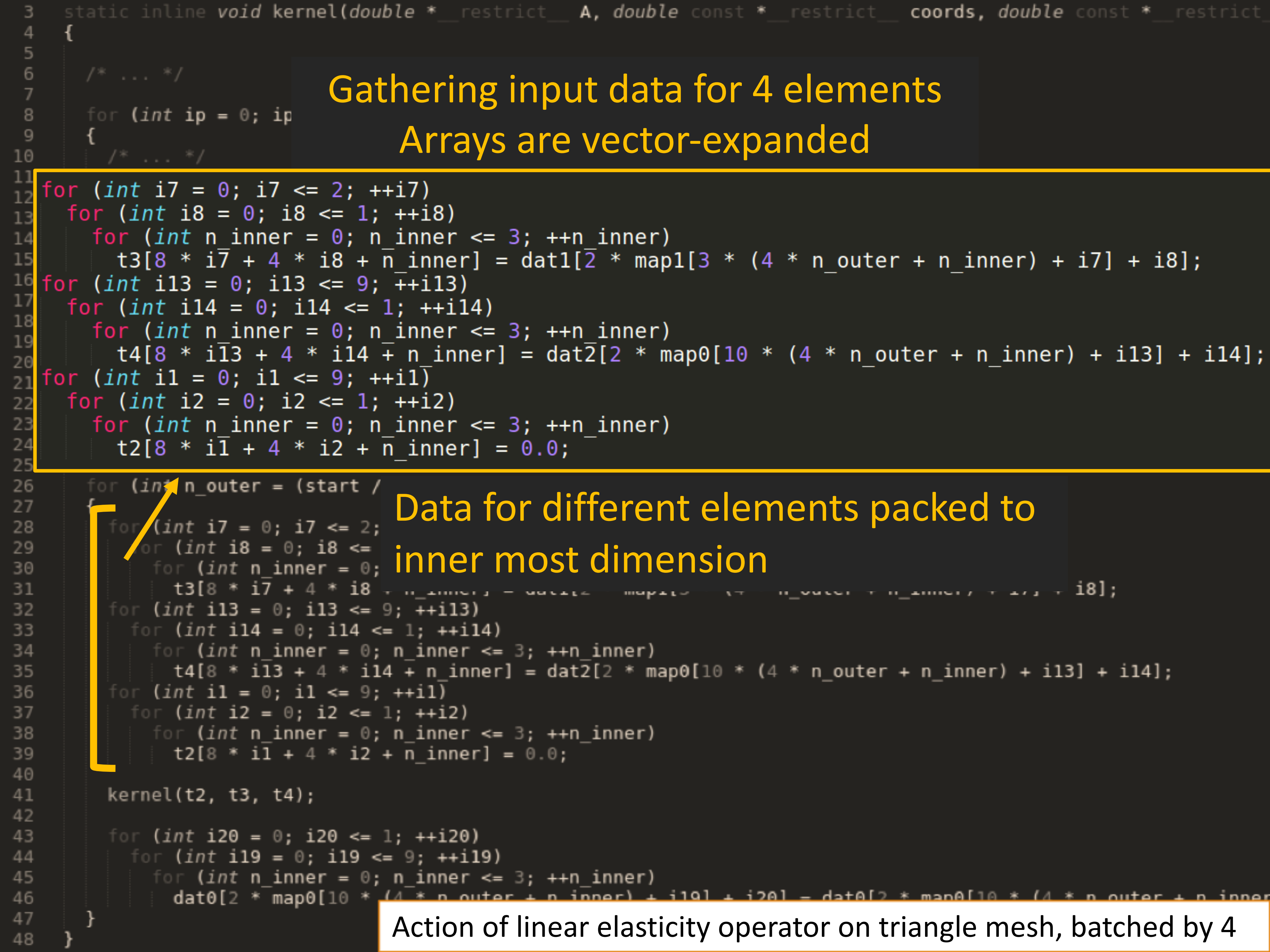
Split n into n_outer and n_inner

Outer loop stride 4

```
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



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

  for (int ip = 0; ip
  {
    /* ... */
```

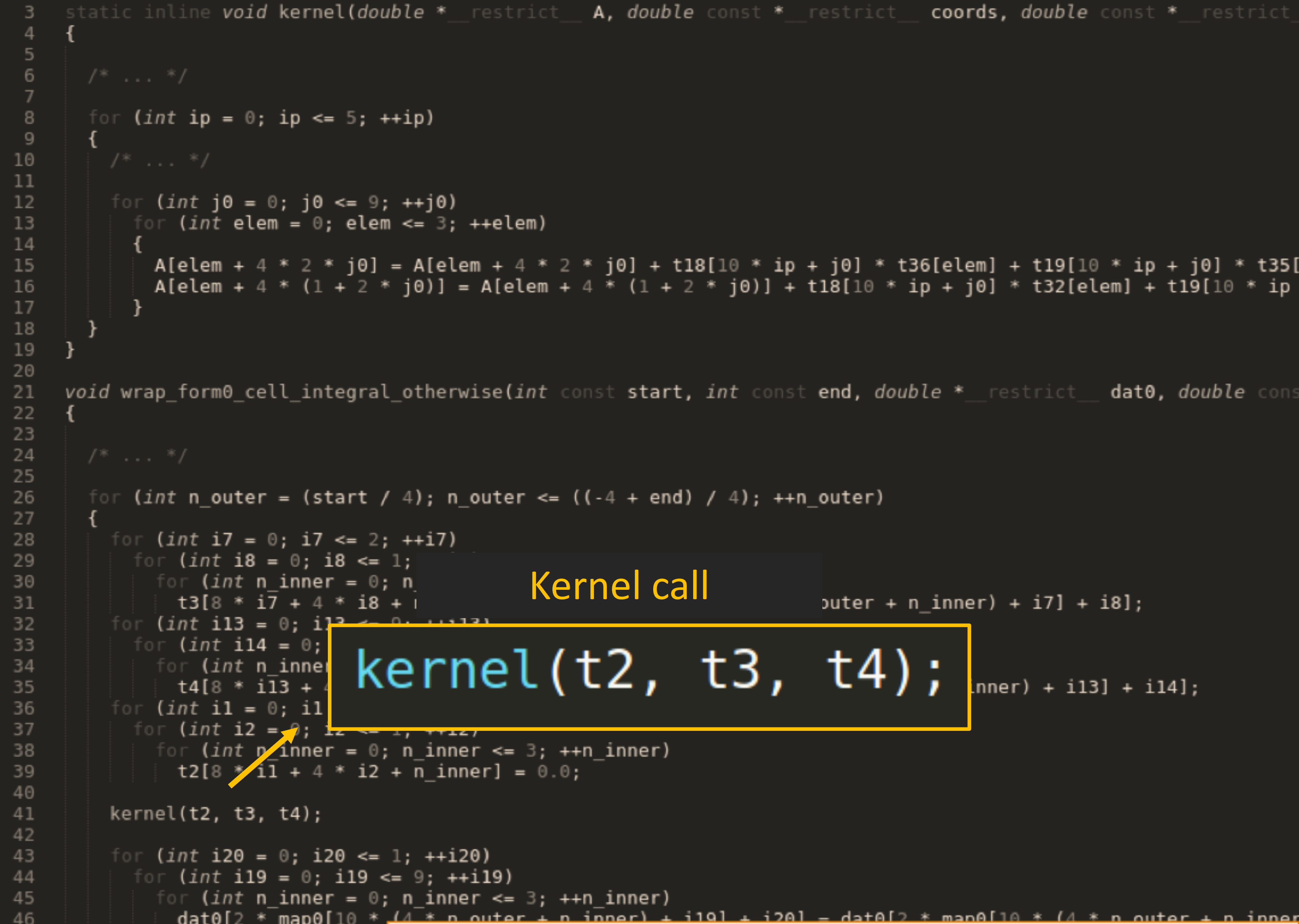
Gathering input data for 4 elements
Arrays are vector-expanded

```
for (int i7 = 0; i7 <= 2; ++i7)
  for (int i8 = 0; i8 <= 1; ++i8)
    for (int n_inner = 0; n_inner <= 3; ++n_inner)
      t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
for (int i13 = 0; i13 <= 9; ++i13)
  for (int i14 = 0; i14 <= 1; ++i14)
    for (int n_inner = 0; n_inner <= 3; ++n_inner)
      t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
for (int i1 = 0; i1 <= 9; ++i1)
  for (int i2 = 0; i2 <= 1; ++i2)
    for (int n_inner = 0; n_inner <= 3; ++n_inner)
      t2[8 * i1 + 4 * i2 + n_inner] = 0.0;
```

Data for different elements packed to inner most dimension

```
  for (int n_outer = (start /
  {
    for (int i7 = 0; i7 <= 2;
      for (int i8 = 0; i8 <=
        for (int n_inner = 0;
          t3[8 * i7 + 4 * i8 + n_inner] = dat1[2 * map1[3 * (4 * n_outer + n_inner) + i7] + i8];
    for (int i13 = 0; i13 <= 9; ++i13)
      for (int i14 = 0; i14 <= 1; ++i14)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
    for (int i1 = 0; i1 <= 9; ++i1)
      for (int i2 = 0; i2 <= 1; ++i2)
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

    kernel(t2, t3, t4);

    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] = dat0[2 * map0[10 * (4 * n_outer + n_inner
  }
}
```

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

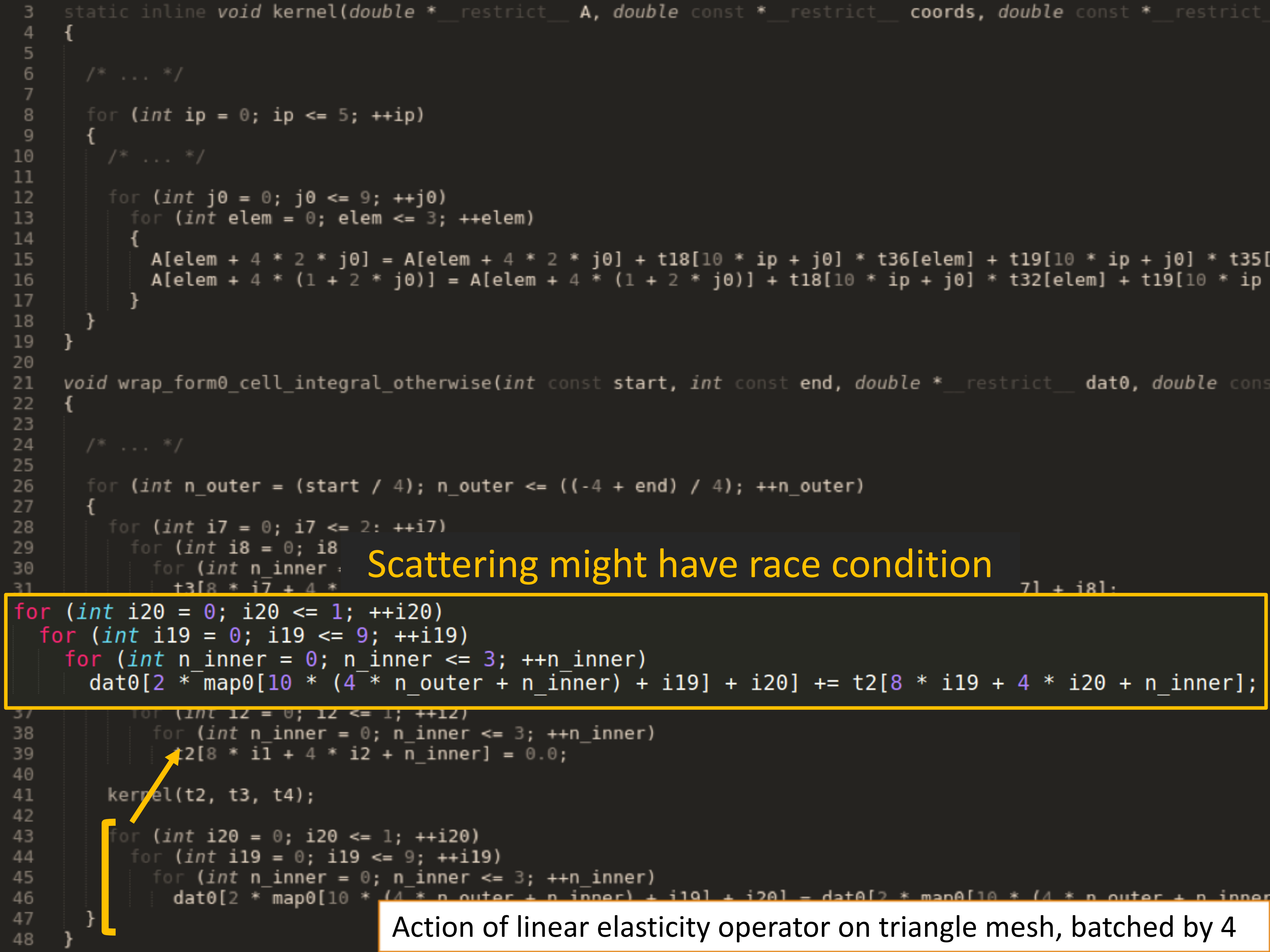


```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

  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] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[
        A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
      }
  }
}

void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
{

  /* ... */

  for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
  {
    for (int i7 = 0; i7 <= 2; ++i7)
      for (int i8 = 0; i8 <= 1;
        for (int n_inner = 0; n
          t3[8 * i7 + 4 * i8 +                              outer + n_inner) + i7] + i8];
    for (int i13 = 0; i1
      for (int i14 = 0;
        for (int n_inne
          t4[8 * i13 +                                                inner) + i13] + i14];
    for (int i1 = 0; i1
      for (int i2 = 0;
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

    kernel(t2, t3, t4);

    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] = dat0[2 * map0[10 * (4 * n_outer + n_inner
  }
}
```

Kernel call

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

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



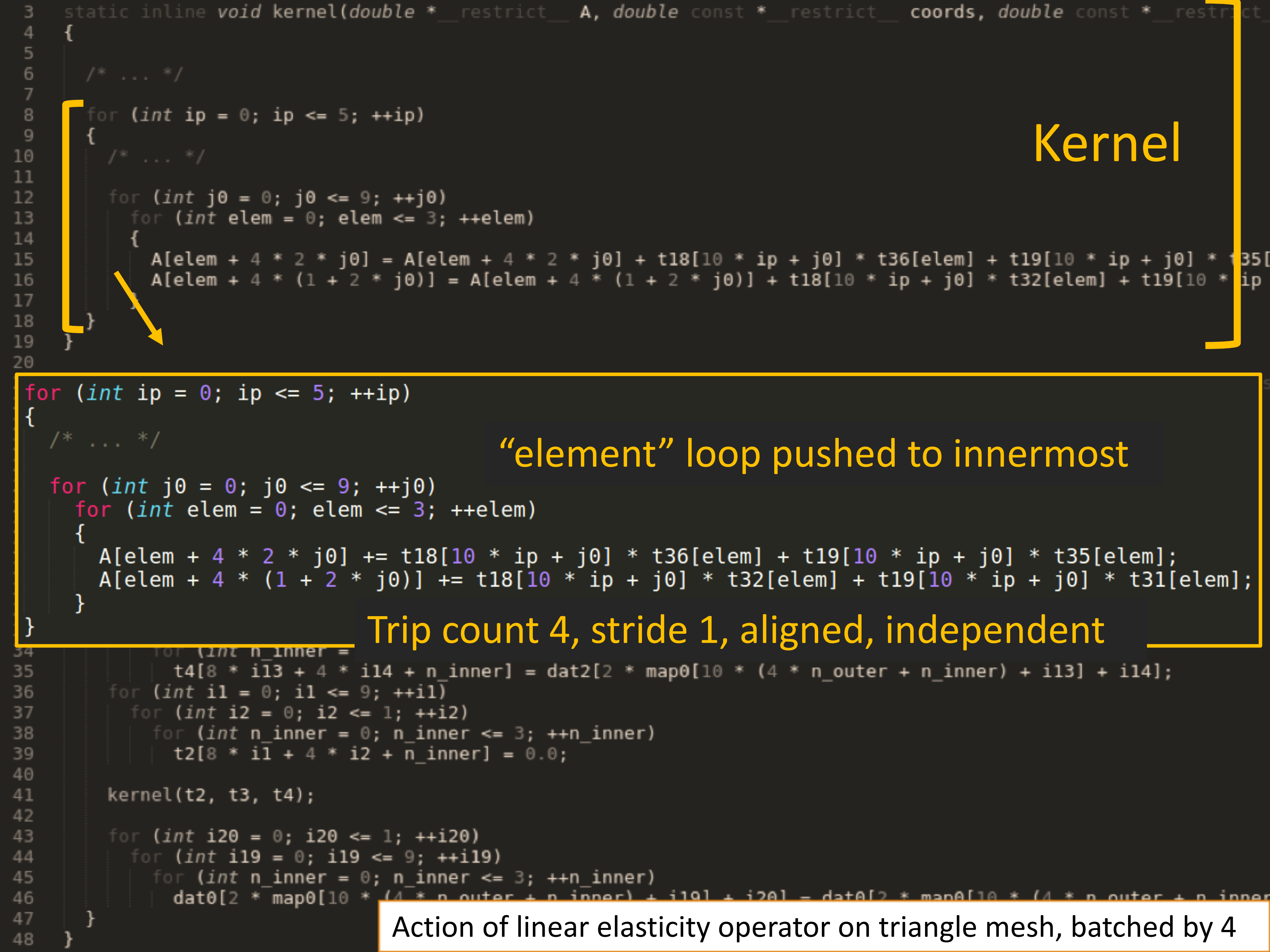
```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

  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] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[
        A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
      }
  }
}

void wrap_form0_cell_integral_otherwise(int const start, int const end, double *__restrict__ dat0, double cons
{

  /* ... */

  for (int n_outer = (start / 4); n_outer <= ((-4 + end) / 4); ++n_outer)
  {
    for (int i7 = 0; i7 <= 2; ++i7)
      for (int i8 = 0; i8
        for (int n_inner
          t3[8 * i7 + 4 *
```

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];
```

```
        for (int n_inner = 0; n_inner <= 3; ++n_inner)
          t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

    kernel(t2, t3, t4);

    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 *
  }
}
```

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



```
static inline void kernel(double *__restrict__ A, double const *__restrict__ coords, double const *__restrict
{

  /* ... */

  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] = A[elem + 4 * 2 * j0] + t18[10 * ip + j0] * t36[elem] + t19[10 * ip + j0] * t35[
        A[elem + 4 * (1 + 2 * j0)] = A[elem + 4 * (1 + 2 * j0)] + t18[10 * ip + j0] * t32[elem] + t19[10 * ip
      }
  }
}
```

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

```
          for (int n_inner =
            t4[8 * i13 + 4 * i14 + n_inner] = dat2[2 * map0[10 * (4 * n_outer + n_inner) + i13] + i14];
      for (int i1 = 0; i1 <= 9; ++i1)
        for (int i2 = 0; i2 <= 1; ++i2)
          for (int n_inner = 0; n_inner <= 3; ++n_inner)
            t2[8 * i1 + 4 * i2 + n_inner] = 0.0;

      kernel(t2, t3, t4);

      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] = dat0[2 * map0[10 * (4 * n_outer + n_inner
    }
  }
```

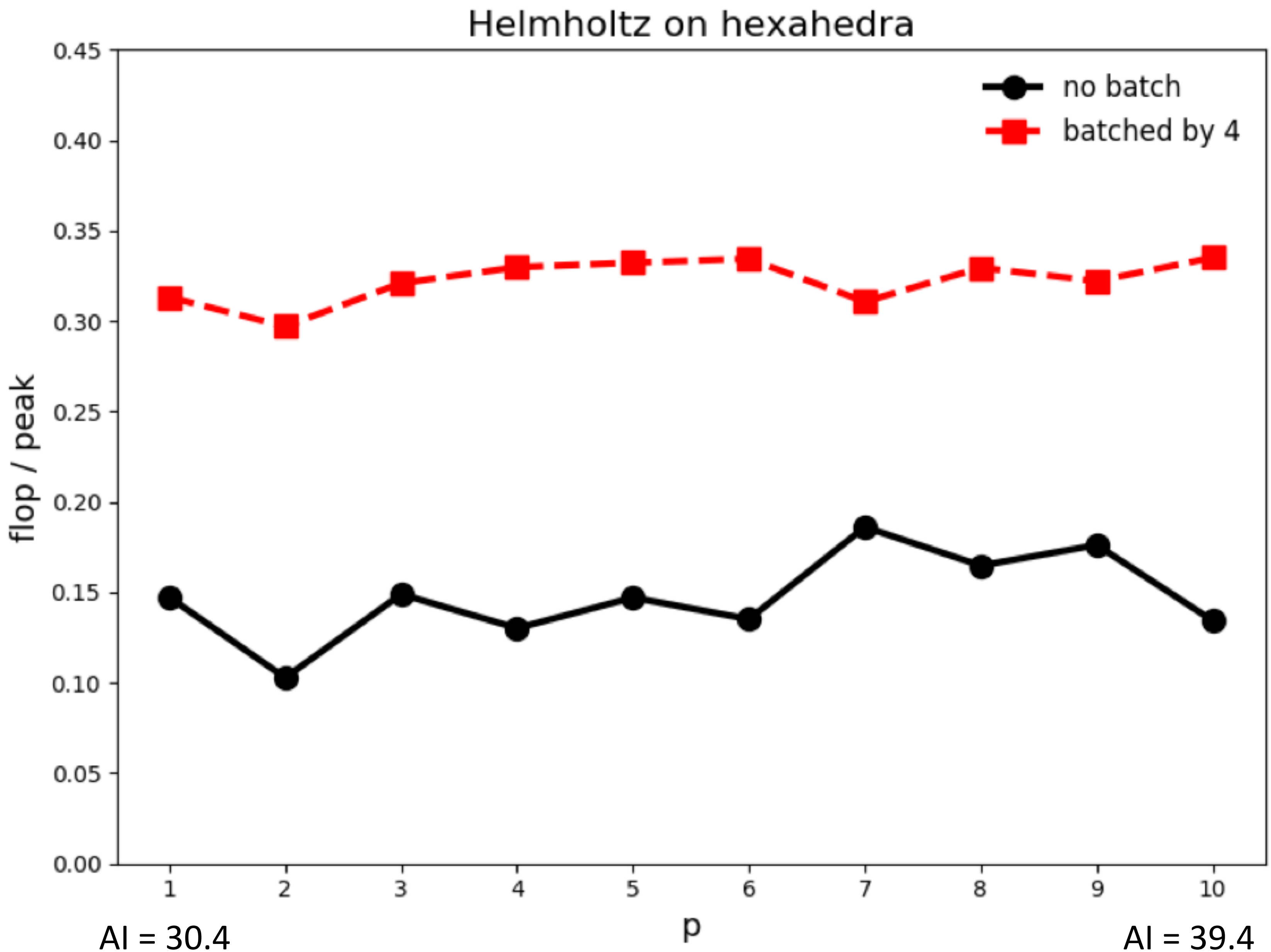
Action of linear elasticity operator on triangle mesh, batched by 4

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]

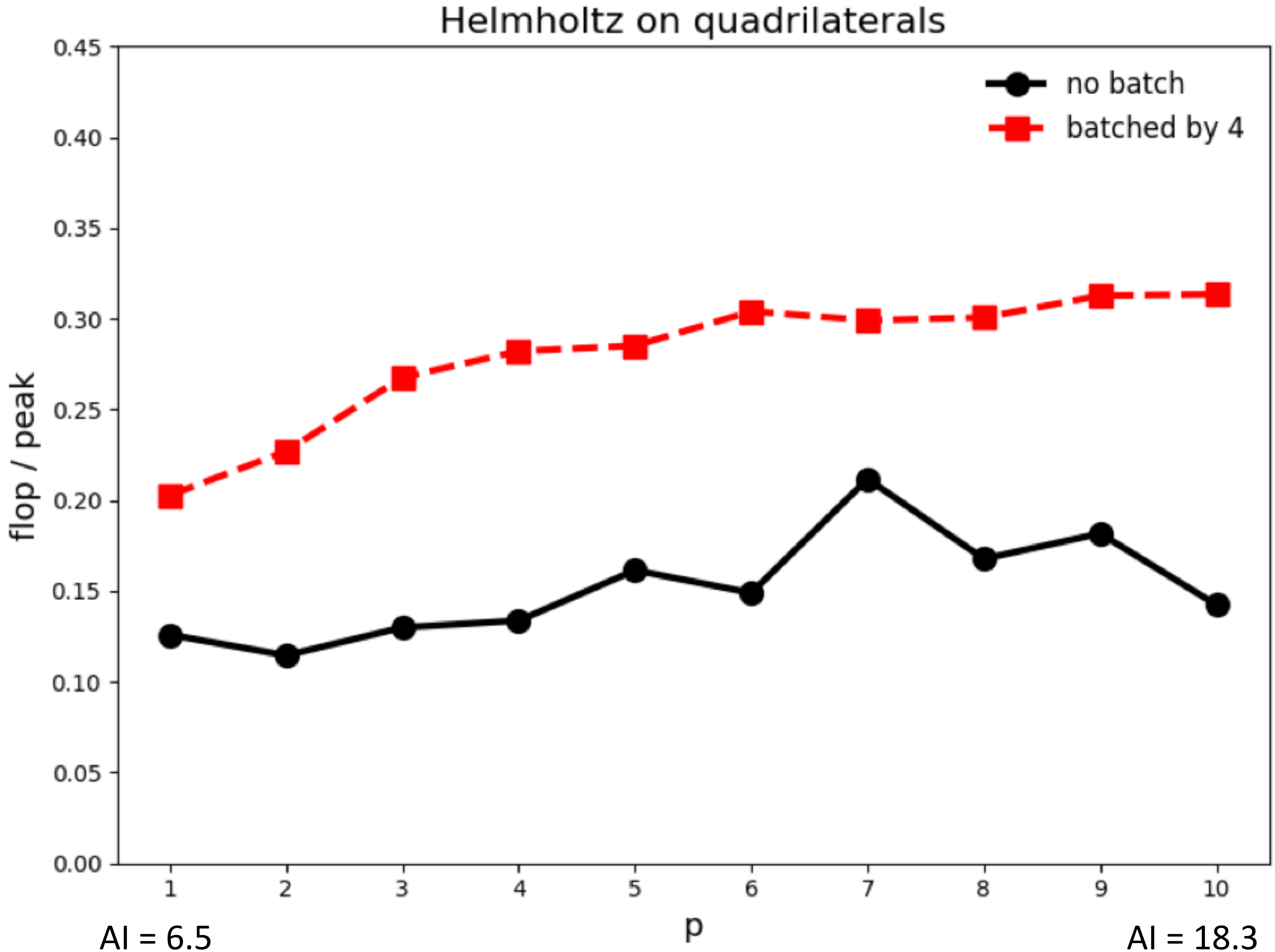
    -> Innermost loop trip count = polynomial degree + 1
- Action of Helmholtz operator
  - Arithmetic Intensity (perfect cache) 30.4 to 33.6 → compute bound
- We present achieved flops / 57.6 Gflops

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


Helmholtz on hexahedra
no batch
batched by 4
flop / peak
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
8
9
10
p

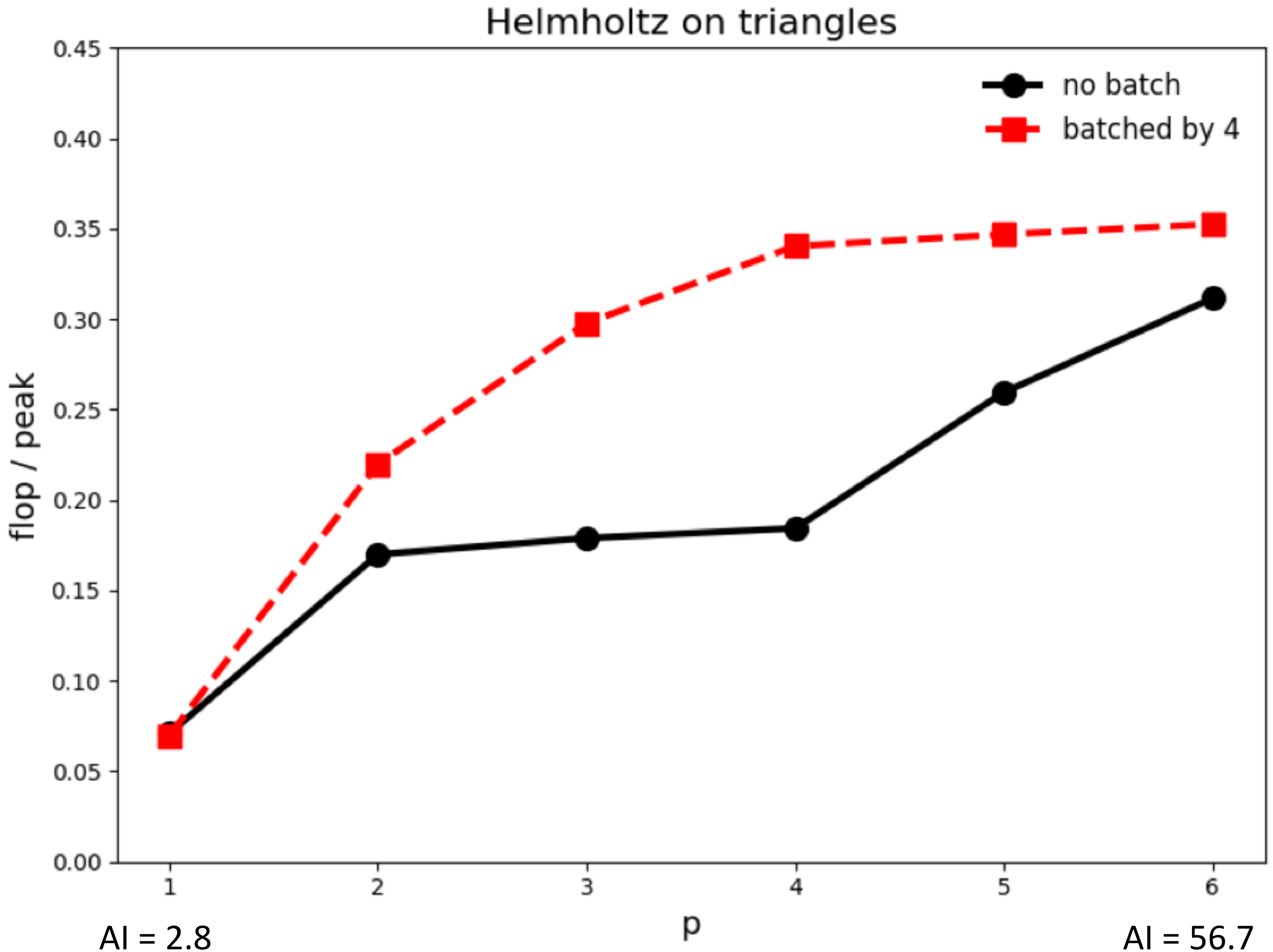
AI = 30.4

AI = 39.4


Helmholtz on quadrilaterals
no batch
batched by 4
flop / peak
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
8
9
10
p

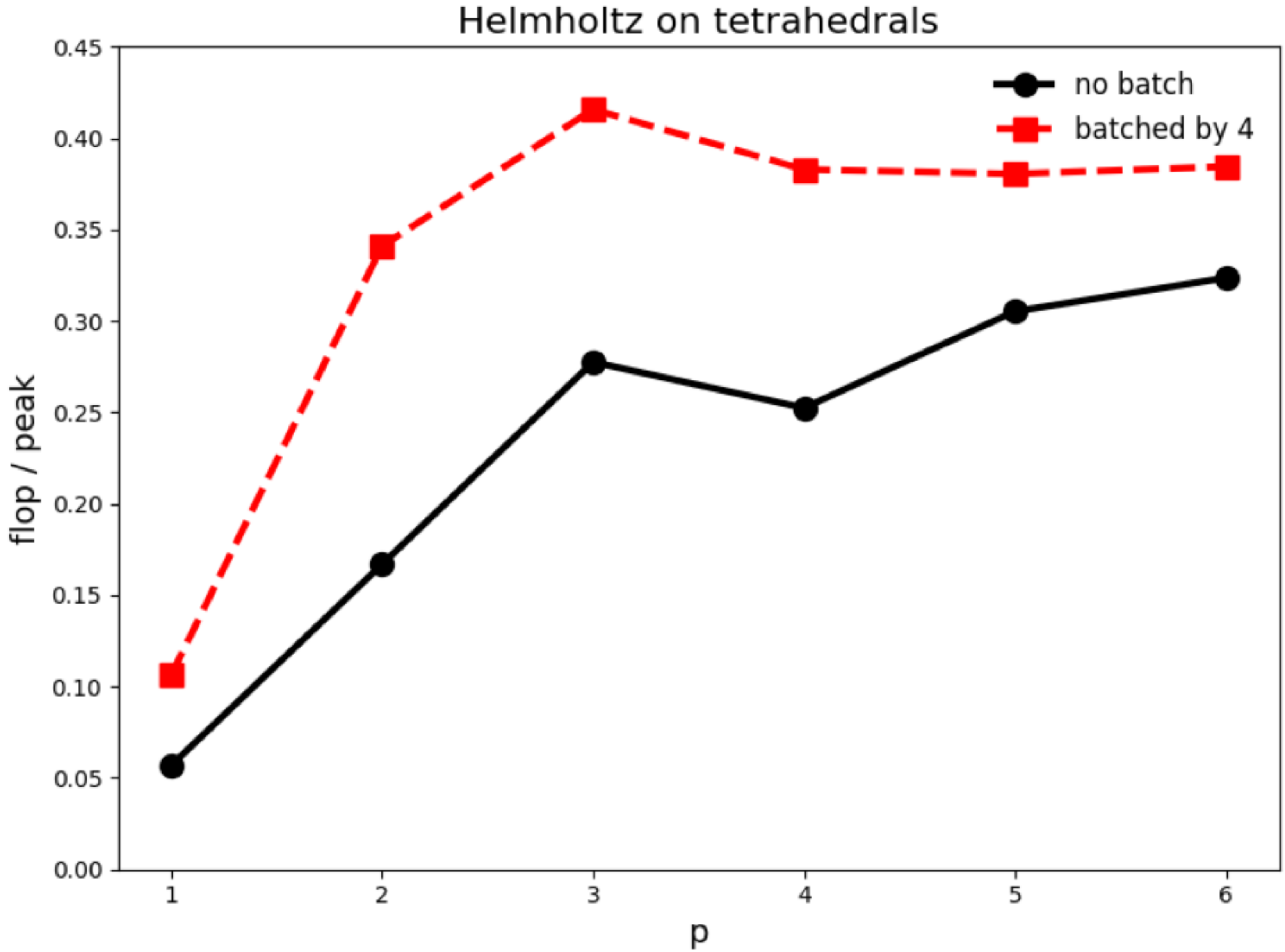
AI = 6.5

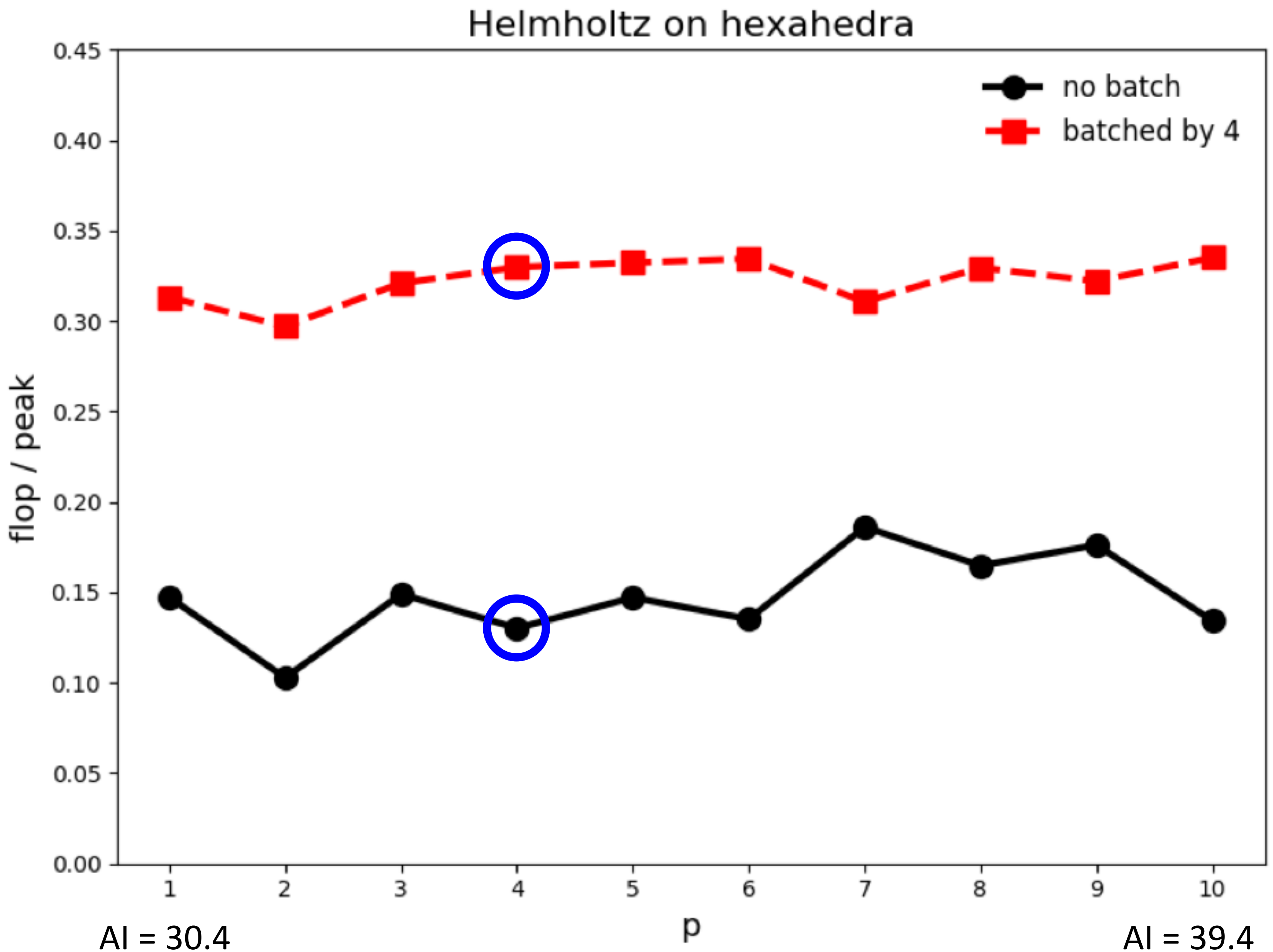
AI = 18.3


Helmholtz on triangles
no batch
batched by 4
flop / peak
p
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
1
2
3
4
5
6

AI = 2.8

AI = 56.7

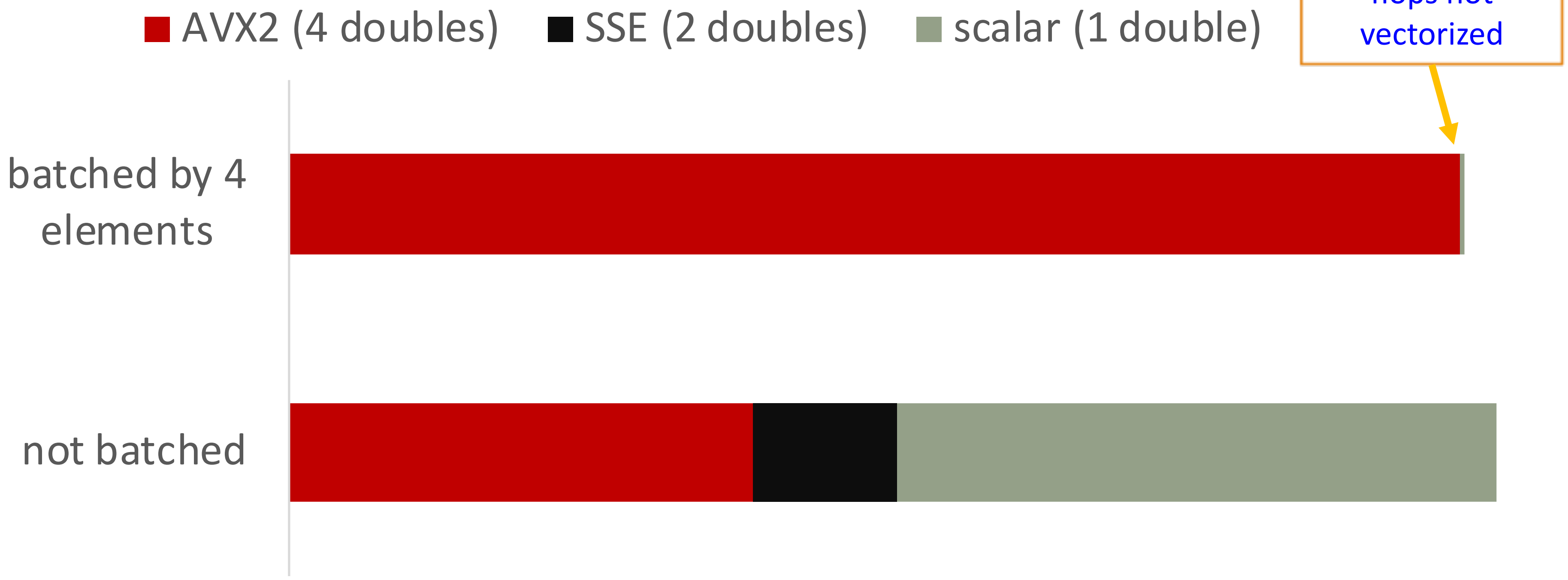

Helmholtz on tetrahedrals
no batch
batched by 4
flop / peak
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
p


Helmholtz on hexahedra
no batch
batched by 4
flop / peak
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
1
2
3
4
5
6
7
8
9
10
p

AI = 30.4

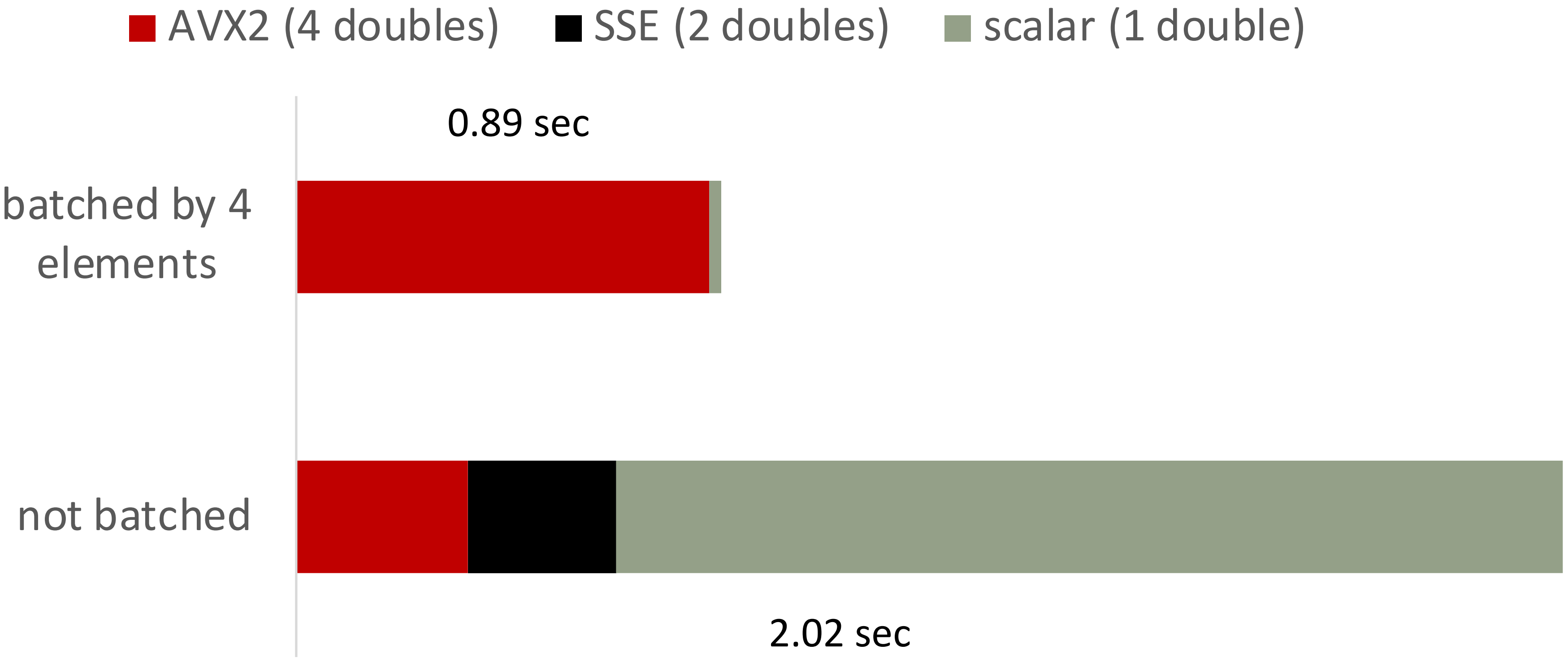
AI = 39.4

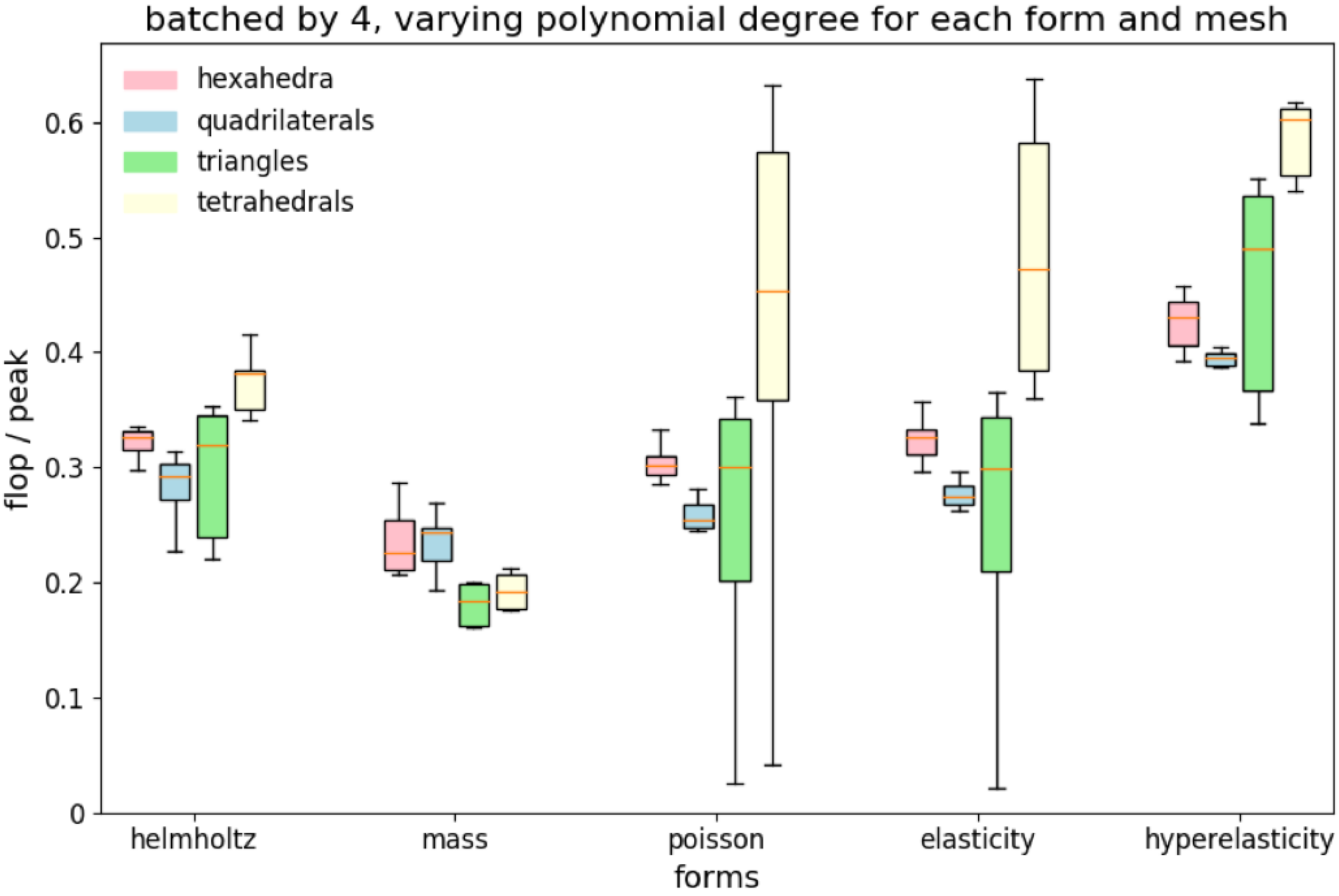
Flop contributions by instruction types



Only 0.4% of flops not vectorized

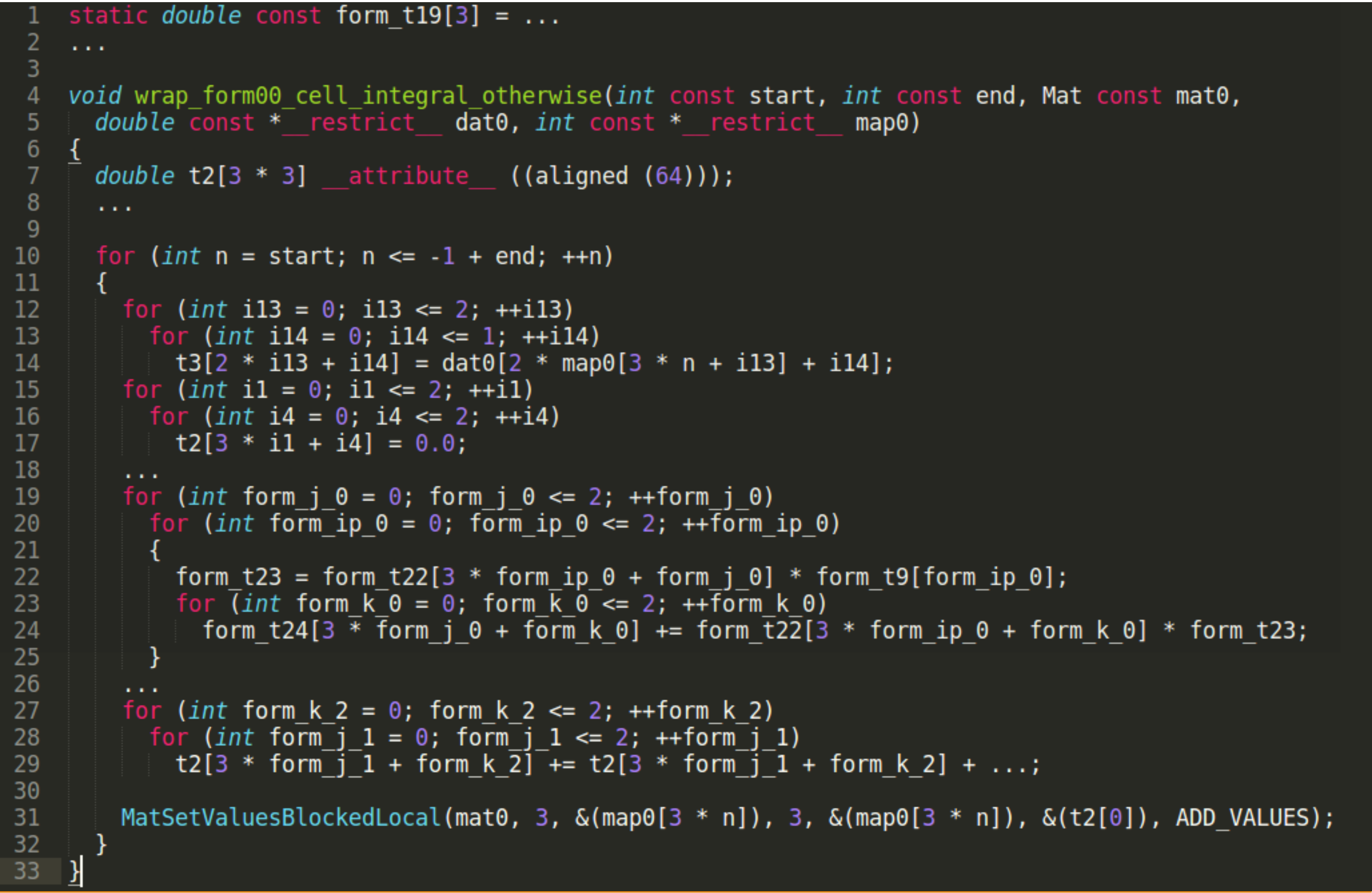
Instruction counts by instruction types




batched by 4, varying polynomial degree for each form and mesh
hexahedra
quadrilaterals
triangles
tetrahedrals
flop / peak
0
0.1
0.2
0.3
0.4
0.5
0.6
helmholtz
mass
poisson
elasticity
hyperelasticity
forms

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`



```
static double const form_t19[3] = ...
...

void wrap_form00_cell_integral_otherwise(int const start, int const end, Mat const mat0,
  double const *__restrict__ dat0, int const *__restrict__ map0)
{
  double t2[3 * 3] __attribute__ ((aligned (64)));
  ...

  for (int n = start; n <= -1 + end; ++n)
  {
    for (int i13 = 0; i13 <= 2; ++i13)
      for (int i14 = 0; i14 <= 1; ++i14)
        t3[2 * i13 + i14] = dat0[2 * map0[3 * n + i13] + i14];
    for (int i1 = 0; i1 <= 2; ++i1)
      for (int i4 = 0; i4 <= 2; ++i4)
        t2[3 * i1 + i4] = 0.0;
    ...
    for (int form_j_0 = 0; form_j_0 <= 2; ++form_j_0)
      for (int form_ip_0 = 0; form_ip_0 <= 2; ++form_ip_0)
      {
        form_t23 = form_t22[3 * form_ip_0 + form_j_0] * form_t9[form_ip_0];
        for (int form_k_0 = 0; form_k_0 <= 2; ++form_k_0)
          form_t24[3 * form_j_0 + form_k_0] += form_t22[3 * form_ip_0 + form_k_0] * form_t23;
      }
    ...
    for (int form_k_2 = 0; form_k_2 <= 2; ++form_k_2)
      for (int form_j_1 = 0; form_j_1 <= 2; ++form_j_1)
        t2[3 * form_j_1 + form_k_2] += t2[3 * form_j_1 + form_k_2] + ...;

    MatSetValuesBlockedLocal(mat0, 3, &(map0[3 * n]), 3, &(map0[3 * n]), &(t2[0]), ADD_VALUES);
  }
}
```

CG1 Poisson bilinear form