This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Exceptional service in the national interest

USNCCM July 25-29, 2021

Exploiting Tensor-Product Structure in High-Order Finite Elements on Next-Generation Architectures

Nathan V. Roberts*,[1] and Mauro Perego[1]
nvrober@sandia.gov
[1] Sandia National Laboratories
SAND 2021-***** C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE

Outline

Sandia National Laboratories

1 Sum Factorization

2 Basis-Value Intermediates Algorithm

3 Point-Value Intermediates Algorithm

4 Implementation and Performance Results

5 Conclusion

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
  - standard assembly: $O(p^9)$ flops

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
  - standard assembly: $O(p^9)$ flops
  - sum factorization: $O(p^7)$ flops in general; $O(p^6)$ flops for constant-Jacobian case.

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
  - standard assembly: $O(p^9)$ flops
  - sum factorization: $O(p^7)$ flops in general; $O(p^6)$ flops for constant-Jacobian case.
- Most sum factorization implementations are specialized for particular problems; often also “bake in” assumptions about the execution environment (serial, OpenMP, CUDA).

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
    - standard assembly: $O(p^9)$ flops
    - sum factorization: $O(p^7)$ flops in general; $O(p^6)$ flops for constant-Jacobian case.
- Most sum factorization implementations are specialized for particular problems; often also “bake in” assumptions about the execution environment (serial, OpenMP, CUDA).
- Our goal: performance-portable, generic implementation.

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
  - standard assembly: $\mathrm{O}(\mathrm{p}^9)$ flops
  - sum factorization: $\mathrm{O}(\mathrm{p}^7)$ flops in general; $\mathrm{O}(\mathrm{p}^6)$ flops for constant-Jacobian case.
- Most sum factorization implementations are specialized for particular problems; often also “bake in” assumptions about the execution environment (serial, OpenMP, CUDA).
- Our goal: performance-portable, generic implementation.
- We also make a small adjustment to the usual sum factorization algorithm; this makes a big difference for CUDA performance.

Motivation

Sandia National Laboratories

- Assembly costs can dominate the runtime in high-order FEM.
- Hexahedra:
  - standard assembly: $O(p^9)$ flops
  - sum factorization: $O(p^7)$ flops in general; $O(p^6)$ flops for constant-Jacobian case.
- Most sum factorization implementations are specialized for particular problems; often also “bake in” assumptions about the execution environment (serial, OpenMP, CUDA).
- Our goal: performance-portable, generic implementation.
- We also make a small adjustment to the usual sum factorization algorithm; this makes a big difference for CUDA performance.
- So far, we have performance results for Poisson assembly in 3D. (More soon.)

Sum Factorization: The Core Idea

- Very simple idea: $(a_1 + a_2) * (b_1 + b_2)$ is cheaper to compute than $a_1 * a_1 + a_1 * b_2 + a_2 * b_1 + a_2 * b_2$.

Computation	Adds	Multiplies	Total Ops
$(a_1 + a_2) * (b_1 + b_2)$	2	1	3
$a_1 * a_1 + a_1 * b_2 + a_2 * b_1 + a_2 * b_2$	3	4	7

Sum Factorization: The Core Idea

Sandia National Laboratories

- Very simple idea: $(a_1 + a_2) * (b_1 + b_2)$ is cheaper to compute than $a_1 * a_1 + a_1 * b_2 + a_2 * b_1 + a_2 * b_2$.

Computation	Adds	Multiplies	Total Ops
$(a_1 + a_2) * (b_1 + b_2)$	2	1	3
$a_1 * a_1 + a_1 * b_2 + a_2 * b_1 + a_2 * b_2$	3	4	7

	Adds	Multiplies	Total Ops
$\sum_{i=1}^{N} \sum_{j=1}^{N} a_i b_j$	$N^2 - 1$	N^2	$2N^2 - 1$
$\sum_{i=1}^{N} a_i \sum_{j=1}^{N} b_j$	$2N - 2$	N	$3N - 2$

Standard Assembly

Sandia National Laboratories

Suppose we are assembling a matrix

$$G_{ij} = \int_K \phi_i \cdot \psi_j$$

where ϕ_i, ψ_j are vector-valued functions defined in physical space in terms of reference-space bases and some transformation M to physical space:

$$\phi_i \cdot \psi_j = \sum_{a=1}^{d_{\hat{\phi}}} \sum_{b=1}^{d_{\hat{\psi}}} \hat{\phi}_i^a \, M_{ab} \, \hat{\psi}_j^b.$$

Here, $\hat{\phi}_i^a$, $\hat{\psi}_j^b$ are scalar-valued components of the basis functions.

Standard Assembly, cont'd

Sandia National Laboratories

Now, suppose that $\hat{\phi}_i^a$, $\hat{\psi}_j^b$ have tensor components, such that

$$\hat{\phi}_i^a(\hat{\mathbf{x}}) = \prod_{r=1}^{R} \hat{\phi}_{i_r}^a(\hat{x}_r), \quad \hat{\psi}_j^b(\hat{\mathbf{x}}) = \prod_{r=1}^{R} \hat{\psi}_{j_r}^b(\hat{x}_r)$$

at each point $\hat{\mathbf{x}}$ in reference space. Let $w(\hat{\mathbf{x}}) = w(\hat{\mathbf{x}}_\mathbf{1}, \ldots, \hat{\mathbf{x}}_\mathbf{R})$ be the pointwise *cell measure*, so that

$$G_{ij} = \int_K \phi_i \cdot \psi_j = \sum_l \phi(\mathbf{x}_l) \cdot \psi(\mathbf{x}_l) w(\hat{\mathbf{x}}_l).$$

We may then expand the integral as:

$$G_{ij} = \sum_{a=1}^{d_{\hat{\phi}}} \sum_{b=1}^{d_{\hat{\psi}}} \sum_{l_1=1}^{L_1} \cdots \sum_{l_R=1}^{L_R} \left(\prod_{r=1}^{R} \hat{\phi}_{i_r}^a(\hat{x}_{l_r}) \hat{\psi}_{j_r}^b(\hat{x}_{l_r}) \right) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}).$$

Observations for Factorization

Sandia National Laboratories

$$G_{ij} = \sum_{a=1}^{d_{\hat{\phi}}} \sum_{b=1}^{d_{\hat{\psi}}} \sum_{l_1=1}^{L_1} \cdots \sum_{l_R=1}^{L_R} \left(\prod_{r=1}^{R} \hat{\phi}_{i_r}^{a}(\hat{x}_{l_r}) \hat{\psi}_{j_r}^{b}(\hat{x}_{l_r}) \right) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}).$$

Some points to observe:

- All terms are in reference space, except M_{ab} and w.
- The number of summands is $d_{\hat{\phi}} d_{\hat{\psi}} P$, where $P = L_1 \ldots L_R = O(p^N)$ is the number of quadrature points.

We may rewrite as:

$$G_{ij} = \sum_{a=1}^{d_{\hat{\phi}}} \sum_{b=1}^{d_{\hat{\psi}}} \sum_{l_1=1}^{L_1} \hat{\phi}_{i_1}^{a}(\hat{x}_{l_1}) \hat{\psi}_{j_1}^{b}(\hat{x}_{l_1}) \sum_{l_2=1}^{L_2} \cdots$$

$$\sum_{l_R=1}^{L_R} \hat{\phi}_{i_R}^{a}(\hat{x}_{l_R}) \hat{\psi}_{j_R}^{b}(\hat{x}_{l_R}) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}).$$

Sum Factorization

Sandia National Laboratories

$$G_{ij} = \sum_{a=1}^{d_{\hat{\phi}}} \sum_{b=1}^{d_{\hat{\psi}}} \sum_{l_1=1}^{L_1} \hat{\phi}_{i_1}^{a}(\hat{x}_{l_1}) \hat{\psi}_{j_1}^{b}(\hat{x}_{l_1}) \sum_{l_2=1}^{L_2} \cdots$$

$$\sum_{l_R=1}^{L_R} \hat{\phi}_{i_R}^{a}(\hat{x}_{l_R}) \hat{\psi}_{j_R}^{b}(\hat{x}_{l_R}) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}).$$

Note that the partial sums

$$\sum_{l_r=1}^{L_r} \cdots \sum_{l_R=1}^{L_R} \hat{\phi}_{i_R}^{a}(\hat{x}_{l_R}) \hat{\psi}_{j_R}^{b}(\hat{x}_{l_R}) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}).$$

can be reused for any G_{ij} whose multi-indices end in $i_r \cdots i_R$, $j_r \cdots j_R$.

Sum Factorization, cont'd

Sandia National Laboratories

We introduce partial sums $G^1, \cdots, G^R$:

$$G^R = \sum_{l_R=1}^{L_R} \hat{\phi}^a_{i_R}(\hat{x}_{l_R}) \hat{\psi}^b_{j_R}(\hat{x}_{l_R}) M_{ab}\, w(\hat{x}_{l_1}, \cdots, \hat{x}_{l_R}),$$

$$G^r(\alpha_r) = \sum_{l_r=1}^{L_r} \hat{\phi}^a_{i_r}(\hat{x}_{l_r}) \hat{\psi}^b_{i_r}(\hat{x}_{l_r}) G_{r+1}(\alpha_{r+1}).$$

Here, $\alpha_r = i_r j_r \cdots i_R j_R$; $\alpha_{r+1} = i_{r+1} j_{r+1} \cdots i_R j_R$.

Note that we require storage for the intermediates $G^r(\alpha_r)$; to save storage, we don't explicitly store G^1; we instead accumulate into the final matrix. Similarly, we fix i_R, j_R in an outer loop, so that G^R only requires a single storage location.

3D Sum Factorization (Basis-Value Intermediates)

Sandia National Laboratories

Algorithm 2 3D Sum Factorization (Basis-Value Intermediates)

```
1: Clear G                                   ▷ Integration matrix, indexed by (i, j) = (i_x i_y i_z, j_x j_y j_z)
2: for a = 1, ··· , d_φ̂, b = 1, ··· , d_ψ̂ do
3:     for i_z = 1, ··· , N_φ̂z, j_z = 1, ··· , N_ψ̂z do
4:         for l_x = 1, ··· , L_x do
5:             Clear G_y                     ▷ (p+1)^2 array, indexed by i_y, j_y
6:             for l_y = 1, ··· , L_y do
7:                 G_z ← 0                   ▷ One number
8:                 for l_z = 1, ··· , L_z do ▷ Integrate in z
9:                     G_z ← G_z + φ̂^{i_z}_{a,z}(ẑ_{l_z}) ψ̂^{j_z}_{b,z}(ẑ_{l_z}) M_ab(x) w(x)   ▷ O((p+1)^5) flops
10:                end for
11:                for i_y = 1, ··· , N_φ̂y, j_y = 1, ··· , N_ψ̂y do
12:                    G_y(i_y, j_y) ← G_y(i_y, j_y) + φ̂^{i_y}_{a,y}(ŷ_{l_y}) ψ̂^{j_y}_{b,y}(ŷ_{l_y}) G_z   ▷ O((p+1)^6) flops
13:                end for
14:            end for
15:            for i_x = 1, ··· , N_φ̂x, j_x = 1, ··· , N_ψ̂x do
16:                for i_y = 1, ··· , N_φ̂y, j_y = 1, ··· , N_ψ̂y do
17:                    i ← i_x i_y i_z, j ← j_x j_y j_z
18:                    G(i, j) ← G(i, j) + φ̂^{i_x}_{a,x}(x̂_{l_x}) ψ̂^{j_x}_{b,x}(x̂_{l_x}) G_y(i_y, j_y)   ▷ O((p+1)^7) flops
19:                end for
20:            end for
21:        end for
22:    end for
23: end for
```

3D Sum Factorization (Basis-Value Intermediates)

Sandia National Laboratories

Algorithm 2 3D Sum Factorization (Basis-Value Intermediates)

```
1: Clear G                                        ▷ Integration matrix, indexed by (i, j) = (i_x i_y i_z, j_x j_y j_z)
2: for a = 1,··· ,d_φ̂, b = 1,··· ,d_ψ̂ do
3:     for i_z = 1,··· ,N_φ̂z, j_z = 1,··· ,N_ψ̂z do
4:         for l_x = 1,··· ,L_x do
5:             Clear G_y                          ▷ (p+1)^2 array, indexed by i_y, j_y
6:             for l_y = 1,··· ,L_y do
7:                 G_z ← 0                        ▷ One number
8:                 for l_z = 1,··· ,L_z do        ▷ Integrate in z
9:                     G_z ← G_z + φ̂^{i_z}_{a,z}(ẑ_{l_z}) ψ̂^{j_z}_{b,z}(ẑ_{l_z}) M_ab(x) w(x)   ▷ O((p+1)^5) flops
10:                end for
11:                for i_y = 1,··· ,N_φ̂y, j_y = 1,··· ,N_ψ̂y do
12:                    G_y(i_y, j_y) ← G_y(i_y, j_y) + φ̂^{i_y}_{a,y}(ŷ_{l_y}) ψ̂^{j_y}_{b,y}(ŷ_{l_y}) G_z   ▷ O((p+1)^6) flops
13:                end for
14:            end for
15:            for i_x = 1,··· ,N_φ̂x, j_x = 1,··· ,N_ψ̂x do
16:                for i_y = 1,··· ,N_φ̂y, j_y = 1,··· ,N_ψ̂y do
17:                    i ← i_x i_y i_z, j ← j_x j_y j_z
18:                    G(i, j) ← G(i, j) + φ̂^{i_x}_{a,x}(x̂_{l_x}) ψ̂^{j_x}_{b,x}(x̂_{l_x}) G_y(i_y, j_y)   ▷ O((p+1)^7) flops
19:                end for
20:            end for
21:        end for
22:    end for
23: end for
```

accesses to basis values for adjacent points are generally separated in time — if adjacent points contiguous, this may be suboptimal…

3D Sum Factorization (Point-Value Intermediates)

Sandia National Laboratories

Algorithm 3 3D Sum Factorization (Point-Value Intermediates)

1: Clear G ▷ Integration matrix, indexed by $(i,j) = (i_x i_y i_z, j_x j_y j_z)$
2: **for** $a = 1,\cdots,d_{\hat\phi}$, $b = 1,\cdots,d_{\hat\psi}$ **do**
3: **for** $i_x = 1,\cdots,N_{\hat\phi_x}$, $j_x = 1,\cdots,N_{\hat\psi_x}$ **do**
4: Clear G_x ▷ $(p+1)^2$ array indexed by (l_y, l_z)
5: **for** $l_y = 1,\cdots,L_y$ **do**
6: **for** $l_z = 1,\cdots,L_z$ **do**
7: **for** $l_x = 1,\cdots,L_x$ **do**
8: $G_x(l_y,l_z) \leftarrow G_x(l_y,l_z) + \hat\phi^{i_x}_{a,x}(\hat x_{l_x})\hat\psi^{j_x}_{b,x}(\hat x_{l_x})M_{ab}(\mathbf{x})\,w(\mathbf{x})$ ▷ $O((p+1)^5)$ flops
9: **end for**
10: **end for**
11: **end for**
12: **for** $i_y = 1,\cdots,N_{\hat\phi_y}$, $j_y = 1,\cdots,N_{\hat\psi_y}$ **do**
13: **for** $l_z = 1,\cdots,L_z$ **do**
14: $G_y(l_z) \leftarrow 0$ ▷ $(p+1)$-length array, indexed by l_z
15: **for** $l_y = 1,\cdots,L_y$ **do**
16: $G_y(l_z) \leftarrow G_y(l_z) + \hat\phi^{i_y}_{a,y}(\hat y_{l_y})\hat\psi^{j_y}_{b,y}(\hat y_{l_y})G_x(l_y,l_z)$ ▷ $O((p+1)^6)$ flops
17: **end for**
18: **end for**
19: **for** $i_z = 1,\cdots,N_{\hat\phi_z}$, $j_z = 1,\cdots,N_{\hat\psi_z}$ **do**
20: $i \leftarrow i_x i_y i_z$, $j \leftarrow j_x j_y j_z$
21: **for** $l_z = 1,\cdots,L_z$ **do**
22: $G(i,j) \leftarrow G(i,j) + \hat\phi^{i_z}_{a,z}(\hat z_{l_z})\hat\psi^{j_z}_{b,z}(\hat z_{l_z})G_y(l_z)$ ▷ $O((p+1)^7)$ flops
23: **end for**
24: **end for**
25: **end for**
26: **end for**
27: **end for**

3D Sum Factorization (Point-Value Intermediates)

Sandia National Laboratories

Algorithm 3 3D Sum Factorization (Point-Value Intermediates)

1: Clear G ▷ Integration matrix, indexed by $(i,j) = (i_x i_y i_z, j_x j_y j_z)$
2: **for** $a = 1, \cdots, d_{\hat{\phi}}$, $b = 1, \cdots, d_{\hat{\psi}}$ **do**
3:     **for** $i_x = 1, \cdots, N_{\hat{\phi}_x}$, $j_x = 1, \cdots, N_{\hat{\psi}_x}$ **do**
4:         Clear G_x ▷ $(p+1)^2$ array indexed by (l_y, l_z)
5:         **for** $l_y = 1, \cdots, L_y$ **do**
6:             **for** $l_z = 1, \cdots, L_z$ **do**
7:                 **for** $l_x = 1, \cdots, L_x$ **do**
8:                     $G_x(l_y, l_z) \leftarrow G_x(l_y, l_z) + \hat{\phi}_{a,x}^{i_x}(\hat{x}_{l_x})\hat{\psi}_{b,x}^{j_x}(\hat{x}_{l_x})M_{ab}(\mathbf{x})\,w(\mathbf{x})$ ▷ $O((p+1)^5)$ flops
9:                 **end for**
10:             **end for**
11:         **end for**
12:         **for** $i_y = 1, \cdots, N_{\hat{\phi}_y}$, $j_y = 1, \cdots, N_{\hat{\psi}_y}$ **do**
13:             **for** $l_z = 1, \cdots, L_z$ **do**
14:                 $G_y(l_z) \leftarrow 0$ ▷ $(p+1)$-length array, indexed by l_z
15:                 **for** $l_y = 1, \cdots, L_y$ **do**
16:                     $G_y(l_z) \leftarrow G_y(l_z) + \hat{\phi}_{a,y}^{i_y}(\hat{y}_{l_y})\hat{\psi}_{b,y}^{j_y}(\hat{y}_{l_y})G_x(l_y, l_z)$ ▷ $O((p+1)^6)$ flops
17:                 **end for**
18:             **end for**
19:             **for** $i_z = 1, \cdots, N_{\hat{\phi}_z}$, $j_z = 1, \cdots, N_{\hat{\psi}_z}$ **do**
20:                 $i \leftarrow i_x i_y i_z, j \leftarrow j_x j_y j_z$
21:                 **for** $l_z = 1, \cdots, L_z$ **do**
22:                     $G(i,j) \leftarrow G(i,j) + \hat{\phi}_{a,z}^{i_z}(\hat{z}_{l_z})\hat{\psi}_{b,z}^{j_z}(\hat{z}_{l_z})G_y(l_z)$ ▷ $O((p+1)^7)$ flops
23:                 **end for**
24:             **end for**
25:         **end for**
26:     **end for**
27: **end for**

Here, basis values for adjacent points are accessed in a tight loop, possibly improving data locality

Some Implementation Features

Sandia National Laboratories

- Intrepid2 designed from the start for performance portability (Kokkos)
- Everything is templated on both a `DeviceType` (Serial, OpenMP, Cuda) and a `Scalar` (double, here)
- Have added data structures that "know" about tensor-product points and values, etc.
- Our goal: a single interface that intelligently selects the best algorithm for assembly, depending on the (tensor) structure of the data.
- Worth noting: we don't (yet) take advantage of potential symmetries.

Estimated Flops for Each Algorithm

Sandia National Laboratories

We use Poisson assembly on a 16^3 grid, with elementwise integrals of the form

$$K_{ij} = \int_K \nabla\phi_i \cdot \nabla\phi_j \, \partial K,$$

as our test problem. We implement a flop estimator (counting each add or multiply as one flop), with results:

p	Standard	Basis-Indexed	Speedup	Point-Indexed	Speedup
1	1.6e+07	2.7e+07	0.60x	2.9e+07	0.55x
2	5.3e+08	3.6e+08	1.5x	3.8e+08	1.4x
3	6.7e+09	2.4e+09	2.8x	2.5e+09	2.7x
4	4.9e+10	1.1e+10	4.5x	1.1e+10	4.5x
5	2.5e+11	3.7e+10	6.8x	3.9e+10	6.4x
6	1.0e+12	1.1e+11	9.1x	1.1e+11	9.1x
7	3.3e+12	2.7e+11	12x	2.7e+11	12x
8	9.6e+12	6.0e+11	16x	6.1e+11	16x

(Speedup values here are theoretical, based only on flop counts.)

Poisson Results: Serial

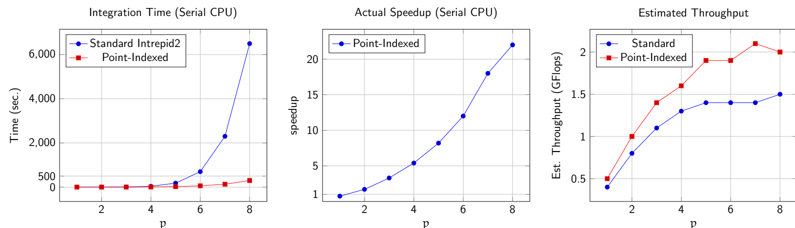

Figure: Serial (Intel Xeon W, 2.3 GHz) timing comparison for 3D Poisson integration, 4096 elements. (Optimal workset sizes for each case determined experimentally.)

Poisson Results: OpenMP

Sandia National Laboratories

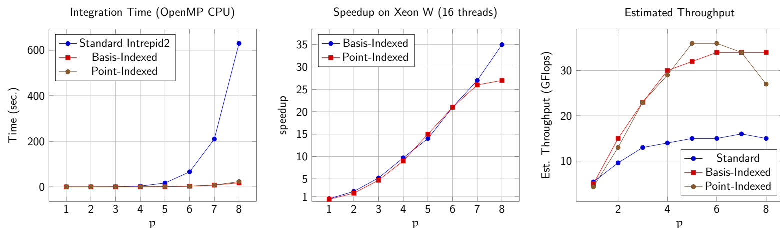

Figure: OpenMP (Intel Xeon W, 2.3 GHz, 16 threads) timing comparison for 3D Poisson integration, 4096 elements. (Optimal workset sizes for each case determined experimentally.)

Poisson Results: CUDA P100

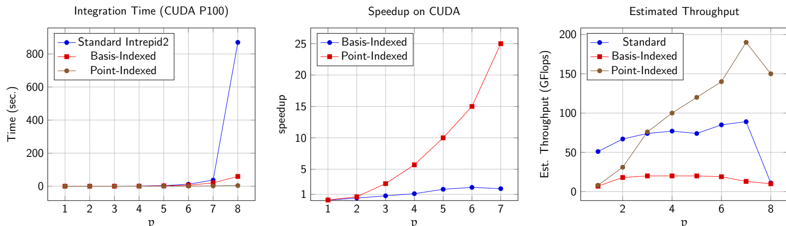

Figure: CUDA (P100) timing comparison for 3D Poisson integration, 4096 elements. (Optimal workset sizes for each case determined experimentally.)

Note: The $p = 8$ case has a dramatic slowdown for standard (for this case, the only workset size that ran to completion was 1); we exclude it from the speedup plot so as to not to throw off the scaling.

Conclusions

Sandia National Laboratories

- We have a general framework for sum factorization that performs efficiently under Serial, CUDA, and OpenMP implementations.
- We have tested this for Poisson, but expect similar results (modulo minor performance tweaks that may be required) for other problems.
- The point-indexed sum factorization algorithm makes a dramatic difference for CUDA, and is close in performance to the basis-indexed algorithm under OpenMP and Serial.
- Next step: performance tests for H^1, $\mathrm{H}(\mathrm{div})$, $\mathrm{H}(\mathrm{curl})$ norms.