A high-performance profile of hybridized PDEs

Joseph McLaughlin

November 28, 2023

Motivation

PDEs often model physical systems are deployed wherever physical systems studied.

Application in earth sciences, physics, biology, finance, among many others.

Motivation

PDEs often model physical systems are deployed wherever physical systems studied.

Application in earth sciences, physics, biology, finance, among many others.

Large-scale systems of PDEs are computationally expensive and memory intensive.

Methods such as hybridization reduce the memory of the system allowing us to solve larger problems.

Constructing a hybrid problem

Continuous 2-D Poisson $$(u_{xx} + u_{yy}) \cdot u = f$$

$$\downarrow$$

Discrete linear system $$\mathbf{Au} = \mathbf{b}$$

$$\downarrow$$

Hybridized linear system $$\begin{bmatrix} \mathbf{M} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{D} \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{g}} \\ \bar{\mathbf{g}}_\delta \end{pmatrix}$$

Linear systems in elliptic PDES

Approximate solution

Exact solution

$$u \approx \mathbf{u} = \mathbf{A}^{-1}\mathbf{b}$$

Linear systems in elliptic PDES

Input vector derived from source

Sparse matrix operator

$$u \approx \mathbf{u} = \mathbf{A}^{-1}\mathbf{b}$$

Linear systems in elliptic PDES

Input vector derived from source

Sparse matrix operator

$$u \approx \mathbf{u} = \underbrace{\mathbf{A}^{-1}\mathbf{b}}$$

Linear system

Solving linear systems

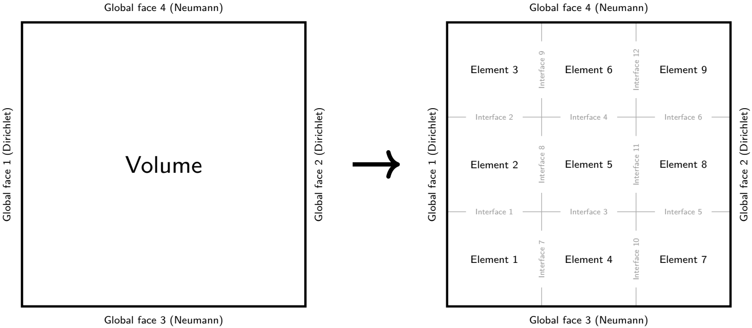
Direct methods

- Initial factorization can be expensive
- Fast, results are exact
- Requires more memory
- Less useful for big problems

Iterative methods

- Sought by refining an approximate solution
- Can be slower
- Requires less memory
- More useful for big problems

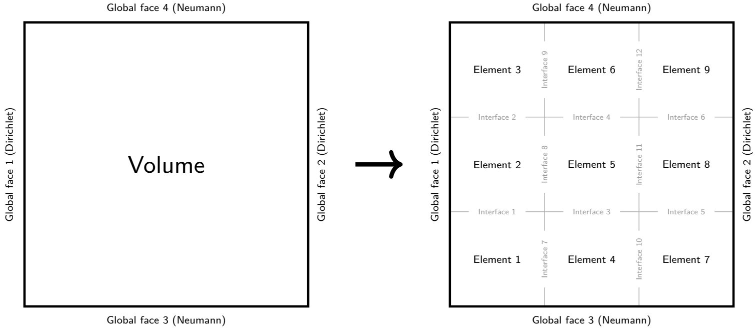
Hybridized systems



$$\mathbf{u} = \mathbf{A}^{-1}\mathbf{b}$$

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{M}_1^{-1}\bar{\mathbf{b}}_1 \\
\mathbf{u}_2 &= \mathbf{M}_2^{-1}\bar{\mathbf{b}}_2 \\
&\vdots \\
\mathbf{u}_9 &= \mathbf{M}_9^{-1}\bar{\mathbf{b}}_9
\end{aligned}$$

Hybridized systems



$$\mathbf{u} = \mathbf{A}^{-1}\mathbf{b}$$

Single linear system

$$\begin{cases} \mathbf{u}_1 = \mathbf{M}_1^{-1}\bar{\mathbf{b}}_1 \\ \mathbf{u}_2 = \mathbf{M}_2^{-1}\bar{\mathbf{b}}_2 \\ \vdots \\ \mathbf{u}_9 = \mathbf{M}_9^{-1}\bar{\mathbf{b}}_9 \end{cases}$$

Several independent systems

Hybridized systems

Boundary coefficient matrix

Block diagonal sparse matrix

$$\begin{bmatrix} \mathbf{M} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{D} \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{g}} \\ \bar{\mathbf{g}}_\delta \end{pmatrix}$$

Diagonal matrix

Hybridized systems

Source and boundary data

Volume solution

$$\begin{bmatrix} \mathbf{M} & \mathbf{F} \\ \mathbf{F}^{\top} & \mathbf{D} \end{bmatrix} \begin{pmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{g}} \\ \bar{\mathbf{g}}_{\delta} \end{pmatrix}$$

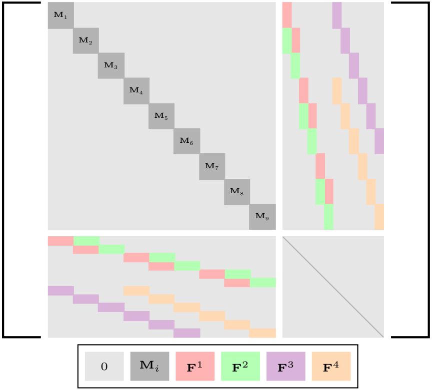
Internal interface solution

Hybridized systems

Boundary coefficient matrix

Block diagonal sparse matrix

$$\begin{bmatrix} \mathbf{M} & \mathbf{F} \\ \mathbf{F}^\top & \mathbf{D} \end{bmatrix} =$$



Hybridized systems

Global system

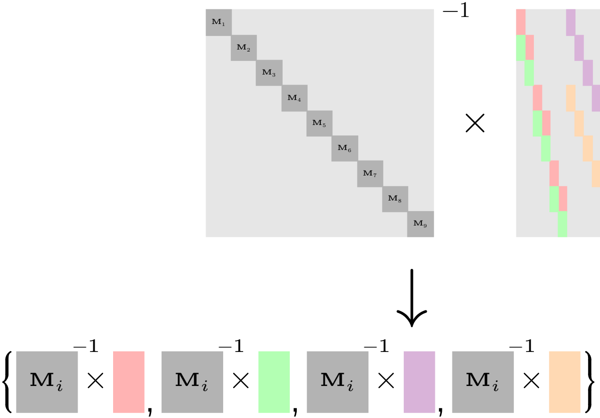
$$\boldsymbol{\lambda}_{\mathbf{A}} = \mathbf{D} - \mathbf{F}^{\top}\mathbf{M}^{-1}\mathbf{F}$$
$$\boldsymbol{\lambda}_{\mathbf{b}} = \bar{\mathbf{g}}_{\boldsymbol{\delta}} - \mathbf{F}^{\top}\mathbf{M}^{-1}\bar{\mathbf{g}}$$
$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_{\mathbf{A}}^{-1}\boldsymbol{\lambda}_{\mathbf{b}}$$

Local system

$$\mathbf{b} = \bar{\mathbf{g}} - \mathbf{F}\boldsymbol{\lambda}$$
$$\mathbf{u} = \mathbf{M}^{-1}\mathbf{b}$$

Hybridized systems

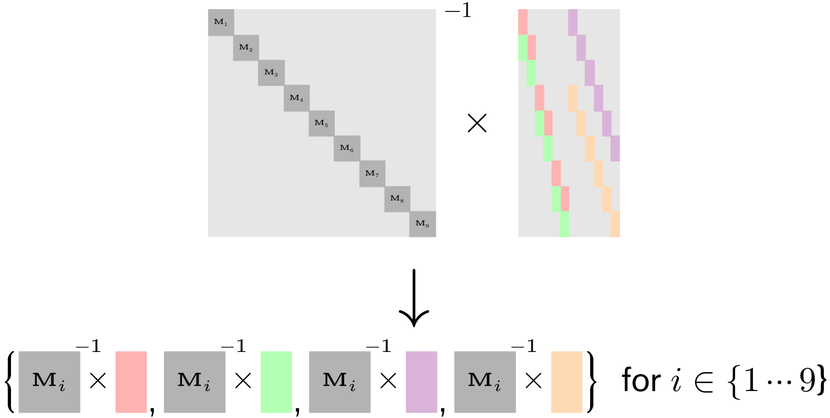
Example: $\mathrm{M}^{-1}\mathbf{F}$



for $i \in \{1 \cdots 9\}$

Hybridized systems

Example: $\mathrm{M}^{-1}\mathbf{F}$



Takeaway: hybridization reduces memory and allows us to compute several small operations.

Implementation details

Hybridized 2-D Poisson equation implemented in C++

Utilizing PETSc (blas for linalg operations)

OpenMP for thread parallelism

PAPI used for performance profiling

Tested on 14 cores (Xeon E5-2683 v3 on Talapas)

Memory balance at 14.6 Flops/byte

Scaling experiment

Parameters

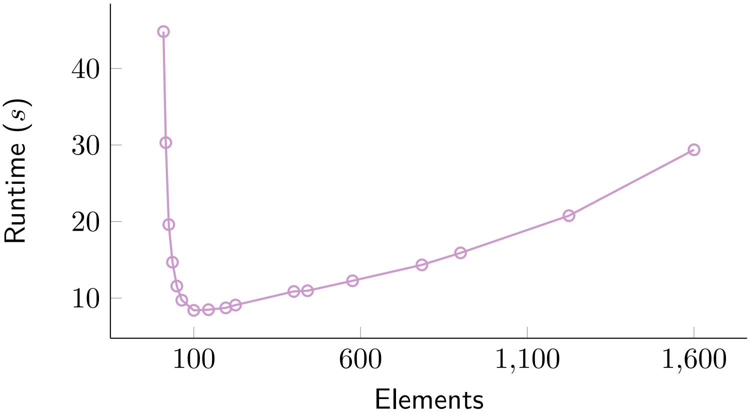
$$\text{Problem size (grid points)} = \bar{n}^2$$
$$\text{Number of elements} = \ell^2$$
$$\text{Local problem size} = n^2 = \bar{n}^2/\ell^2$$

Constraint

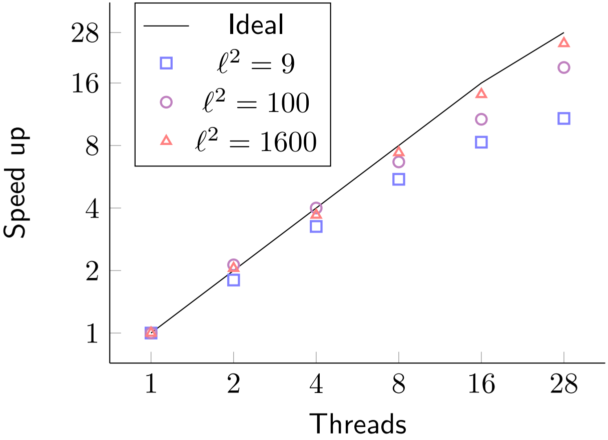
Fix $\bar{n}^2 = 705,600$ and vary ℓ^2 to evaluate strong scaling in terms of grid points.

Runtime with 28 threads

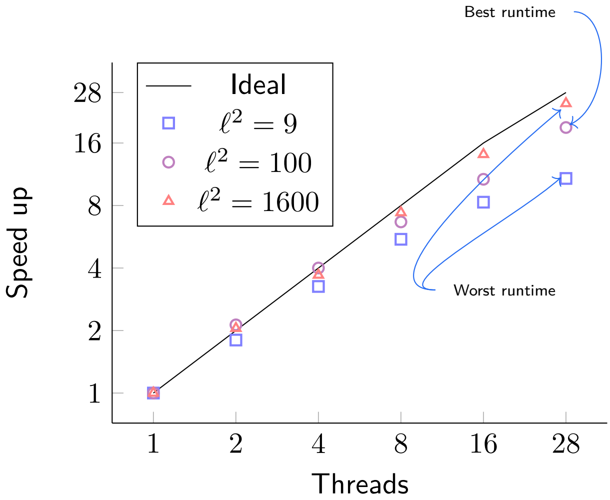
Performance is convex.
Best performance is at $\ell^2 = 100$ for $\bar{n}^2 = 705,600$.



Strong scaling does not correlate with runtime

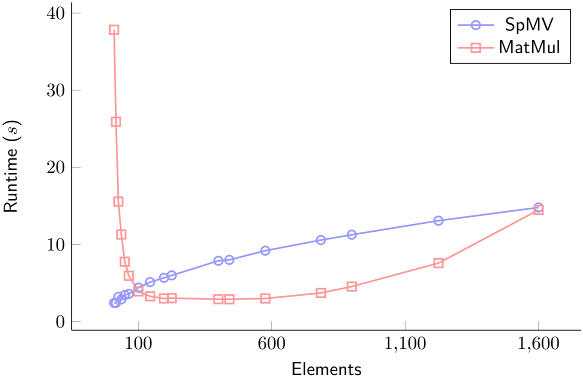


Strong scaling does not correlate with runtime



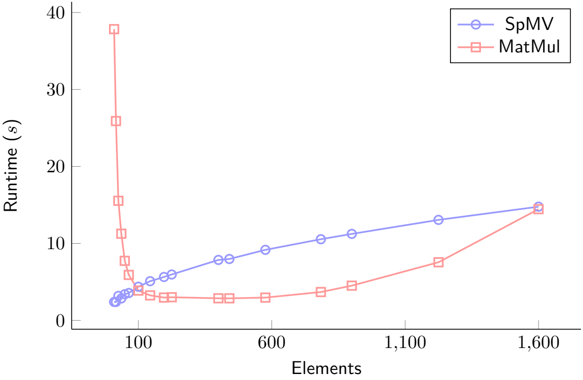
Two operations comprise majority of the runtime

- SpMV i.e., $\mathbf{F}^\top(\mathbf{M}^{-1}\bar{\mathbf{g}})$
- MatMul i.e., $\mathbf{F}^\top(\mathbf{M}^{-1}\mathbf{F})$
- The remaining operations comprise $< 9\%$ of runtime



Utilizing each operation

- SpMV cannot be reused as it uses source data.
- Results of MatMul can be reused if the geometry of the problem does not change.
- For our purposes we assume MatMul cannot be reused.



Analyzing SpMV

SpMV is *generally* memory bound.

$$\mathbf{F}^{\top} \times (\mathbf{M}^{-1}\bar{\mathbf{g}})$$

4 unqiue sub-matrices $\in \mathbb{R}^{n \times n^2}$

1 $\bar{n}^2$ vector partitioned into ℓ^2 length-n^2 vectors

Analyzing SpMV

SpMV is *generally* memory bound.

$$\mathbf{F}^\top \times (\mathbf{M}^{-1}\bar{\mathbf{g}})$$

Still the numbers of bytes in each unique $\mathbf{F}$ component is less than $\mathbf{F}$ outright.

Analyzing SpMV

SpMV is *generally* memory bound.

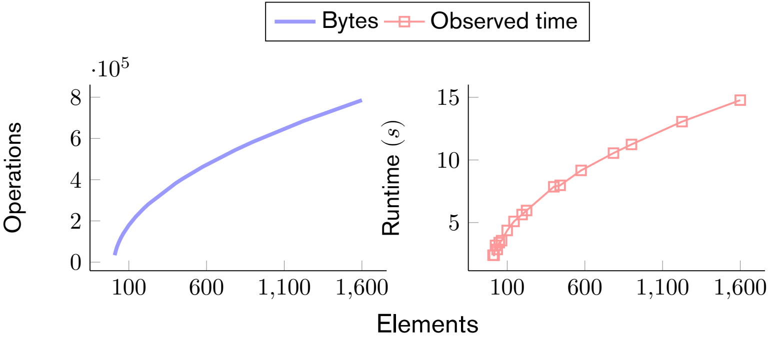
$$\mathbf{F}^\top \times (\mathbf{M}^{-1}\bar{\mathbf{g}})$$

Still the numbers of bytes in each unique $\mathbf{F}$ component is less than $\mathbf{F}$ outright.

More elements $\Rightarrow$ smaller $\mathbf{F}$ components $\Rightarrow$ greater data reuse.

Analyzing SpMV

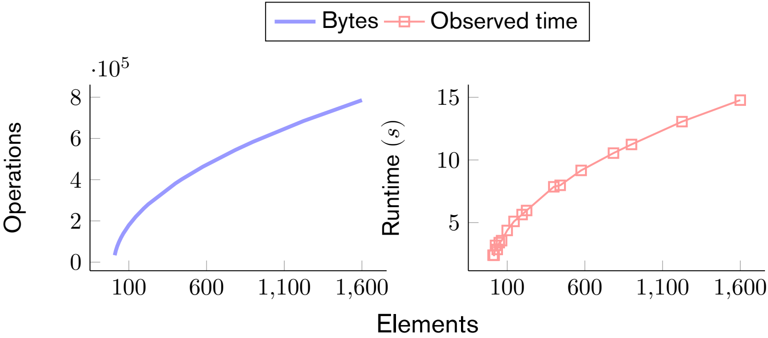
Runtime scales proportionally the number of bytes.



Analyzing SpMV

Runtime scales proportionally the number of bytes.

Takeaway: prefer fewer elements to minimize SpMV.



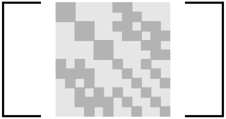
Analyzing MatMul

MatMul is implemented as a *batch* operation.

One thread is responsible for one MatMul.

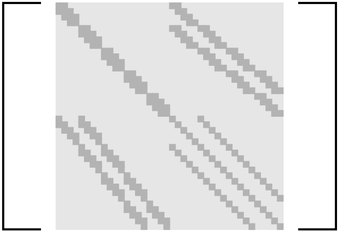
The number of MatMuls is a large sum related to the number of interfaces.

Additional elements increase the number of MatMuls, but make each MatMul smaller.



$\text{nnz} = 5.3\text{e}6$

$\ell^2 = 9$

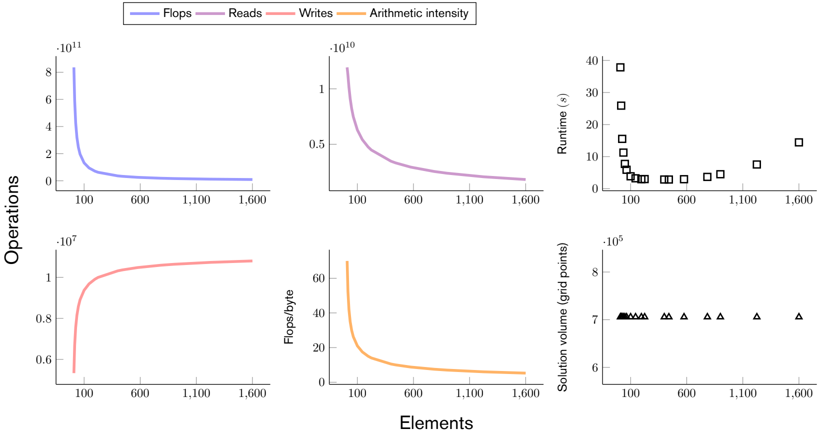


$\text{nnz} = 7.5\text{e}6$

$\ell^2 = 25$

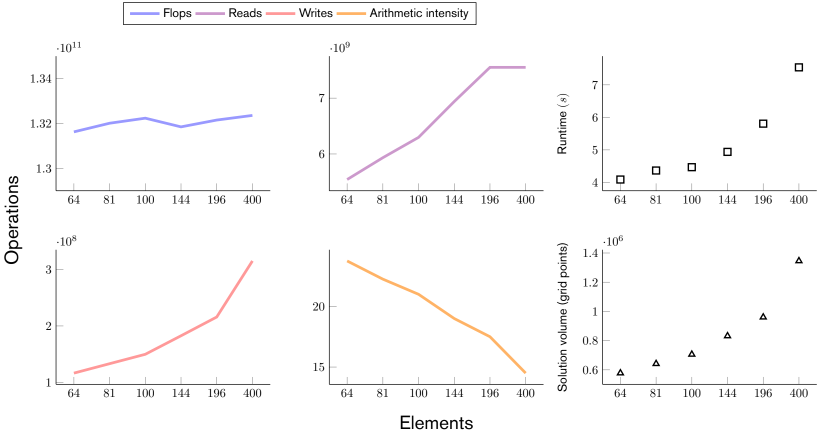
Analyzing MatMul

Case 1. Global volume ($\bar{n}$) is constant. Additional elements decrease the size of each local volume (n).



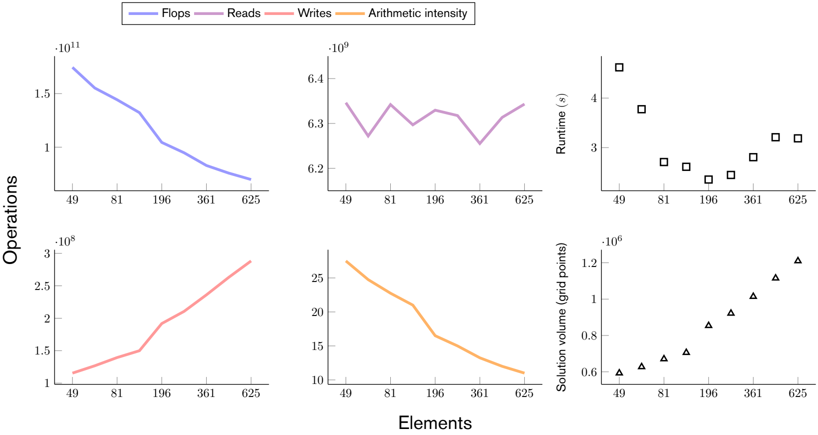
Analyzing MatMul

Case 2. Several problems chosen with a similar total MatMul work (flops).



Analyzing MatMul

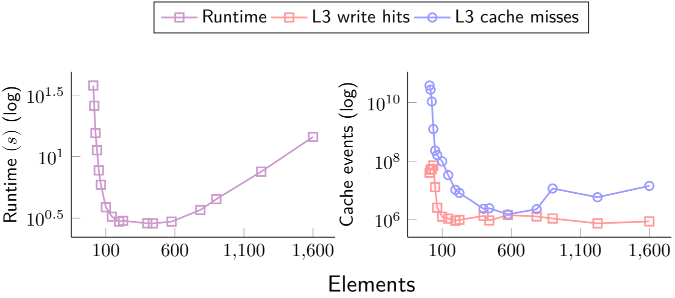
Case 3. Several problems chosen with a similar total MatMul problem size (bytes).



Profiling the cache

Case 1 reveals that cache misses increase while the total bytes read decrease and the total writes increase.

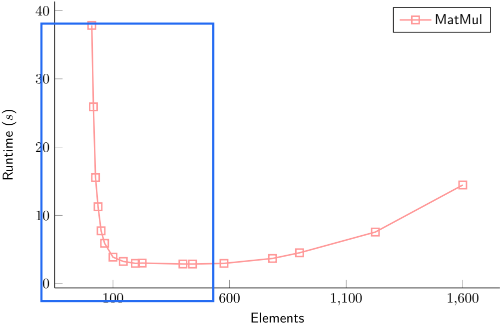
This suggests that this problem is write-miss bound.



MatMul performance

MatMul is compute bound above the system's memory balance (14 Flops/Byte).

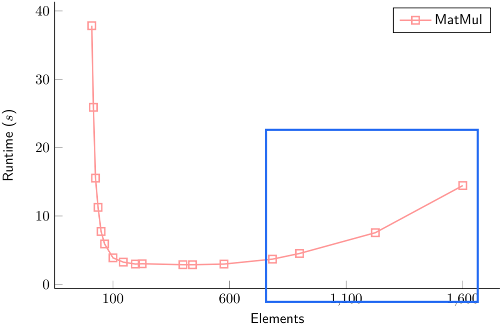
Below the system's memory balance the operation becomes memory bound and in particular, write bound.



MatMul performance

Reads outnumber writes by 10:1 but

- inputs to MatMul become increasingly small and often fits into the cache
- writes are only written to once, and much more likely to cache miss



Remarks

Building hybrid systems

Choosing the fewest elements is beneficial if you don't intend to remesh often.

If you need to frequently remesh, we found ideal performance near the system's memory balance.

Adding additional elements generally makes the global system expensive.

Future work

A specialized block sparse format may improve performance by eliminating writes in MatMul.

These results may help us develop a predictive model of performance for larger systems.