Dynamic Load Balancing of Massively Parallel Unstructured Meshes

Gerrett Diamond, Cameron W. Smith, Mark S. Shephard

Scientific Computation Research Center
Rensselaer Polytechnic Institute

November 13, 2017

Outline

Motivation

Many evolving distributed simulations have:

- Complex relational structures.
- Irregular forms of computational and communication costs.
- Evolving imbalance of work.
- Multiple criteria that need balancing simultaneously.

Common Methods for Partitioning

- Multilevel Graph Methods
  - ParMETIS
  - Zoltan
- Geometric Methods
  - Recursive Coordinate Bisection (RCB)
  - Recursive Inertial Bisection (RIB)
  - Multi-Jagged
- Diffusive Methods
  - Label Propagation

What is EnGPar?

- A partitioning tool to complement existing multi-level and geometric methods.
- Provides a diffusive load balancing algorithm for partition improvement and supports multi-criteria partitioning.
- Utilizes a specialized multigraph structure to represent relation based data.
- Implemented to support efficient data parallel operations on accelerators and vector units in many core processors.

Software

EnGPar's source can be found at `scorec.github.io/EnGPar/`.

- Written in C++ using MPI.
- Provides C/C++ and FORTRAN APIs.
- Uses PCU for sparse neighborhood exchange peer to peer communications.
  - Found at `github.com/SCOREC/core/tree/master/pcu`

N-graph

The N-graph is a multigraph with two modes of operation: traditional or hypergraph.

The N-graph is defined as the following:

- A set of vertices V representing the atomic units of work.
- If using the traditional graph mode:
  - N sets of edges $E_0, ..., E_{n-1}$ for each type of relation.
  - Each edge connects two vertices $u, v \in V$.
- If using the hypergraph mode:
  - N sets of hyperedges $H_0, ..., H_{n-1}$ for each type of relation.
  - N sets of pins $P_0, ..., P_{n-1}$ corresponding to each set of hyperedges.
  - Each pin in P_i connects a vertex, $v \in V$, to a hyperedge $h \in H_i$.

Mapping structures to the N-graph

To map to the N-graph simulations must:

- Define units of work as the vertices.
- Decide on the mode of edges to use.
- Create (hyper)edges between the vertices whose corresponding work relate to each other.

Mapping structures to the N-graph

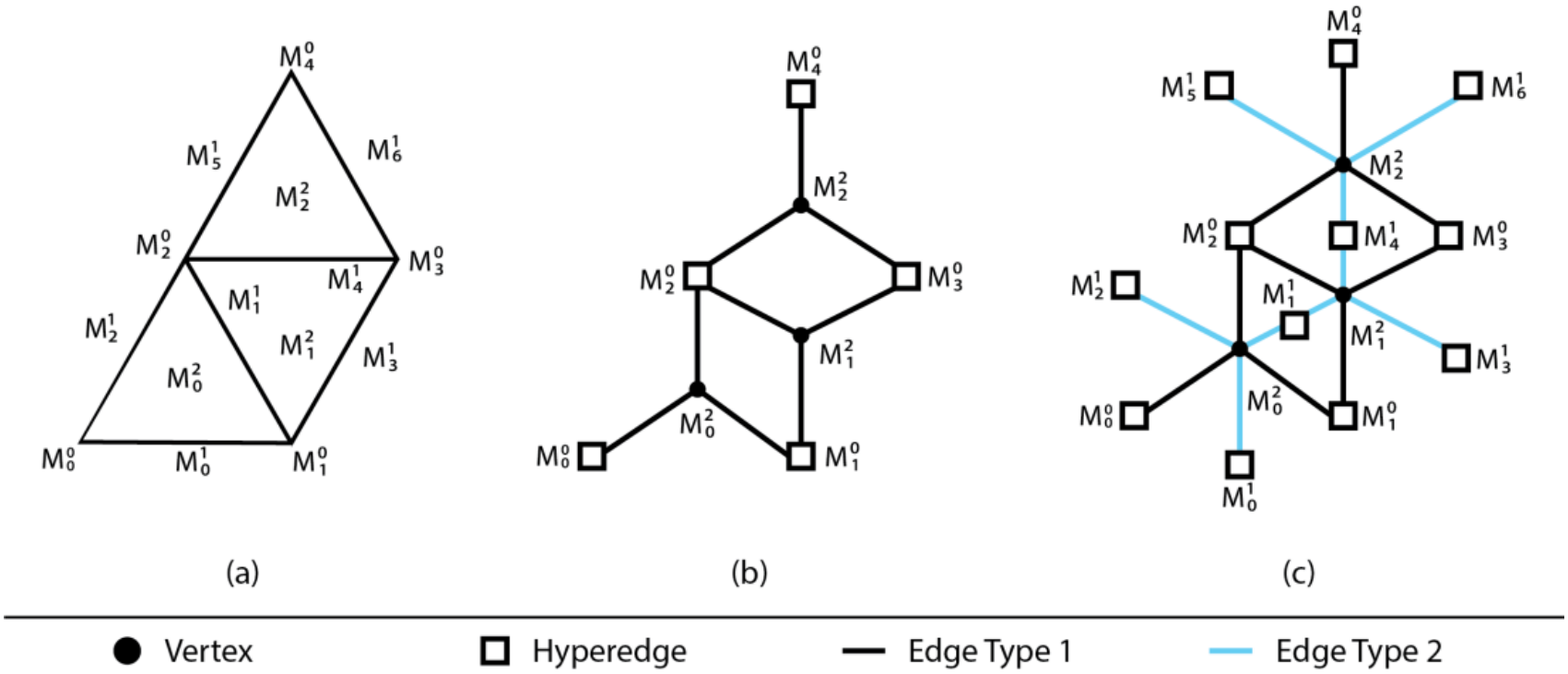


Figure: Converting a triangular mesh(a) to the N-graph with an edge type for mesh vertices (b) and an additional edge type for mesh edges (c).

Diffusive Terminology

- Sides
  - Each part determines which parts are its neighbors.
  - Determines a measurement of the area between each part.
- Cavity
  - Defined by (hyper)edges that cross a part boundary.
  - Includes all the vertices that bound the (hyper)edge.

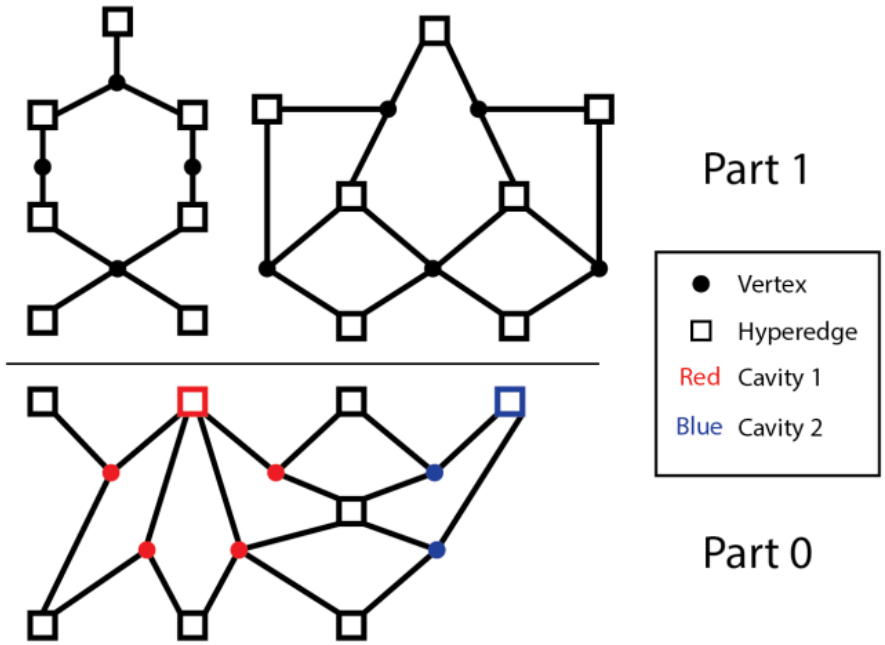


Figure: Two parts of an N-graph with a side of size 4. Two cavities are shown in red and blue for part 0.

Diffusive Terminology

- Weights
  - Each part computes its weight of the current target entity types, w_i.
  - This weight is shared with all of the part's neighbors(sides).
- Targets
  - The neighbors that the part will send weight to.
  - A part, i, will send weight to a neighbor, j, if:
    - $w_i > w_j$
    - the area between the parts (s_{ij}) is less than the average of all part boundaries.
  - Weight to send from part i to part j is $\alpha(w_i - w_j) * \frac{\text{size}(s_{ij})}{\text{size}(s)}$
    - α is an input parameter that limits how much weight is sent in each iteration.

Diffusive Partitioning

Algorithm 1 Diffusive Load Balancing Framework

```
1:  procedure BALANCE(ngraph,entity_types)
2:      for all t ∈ entity_types do
3:          while imbalance of t > tolerance do RUNSTEP(ngraph,t)
4:              if Balancing Stagnates then
5:                  break
6:  procedure RUNSTEP(ngraph,t)
7:      sides = makeSides(ngraph)
8:      weights = makeWeights(ngraph, sides, t)
9:      targets = makeTargets(ngraph, sides, weights)
10:     queue = makeQueue(ngraph)
11:     plan = select(ngraph, targets, queue)
12:     ngraph.migrate(plan)
```

Queue

The queue provides an ordering of the (hyper)edges on the part boundary for selection.

This is done in two steps:

- A breadth-first traversal starting at the part boundary to determine the furthest (hyper)edges as the center of the part.
- A breadth-first traversal starting at the center (hyper)edges to compute topological distance for each (hyper)edge on the part boundary.

When parts are not fully connected, this operation is performed on each component separately.

The queue is then ordered with shallowest components before deeper components.

Queue

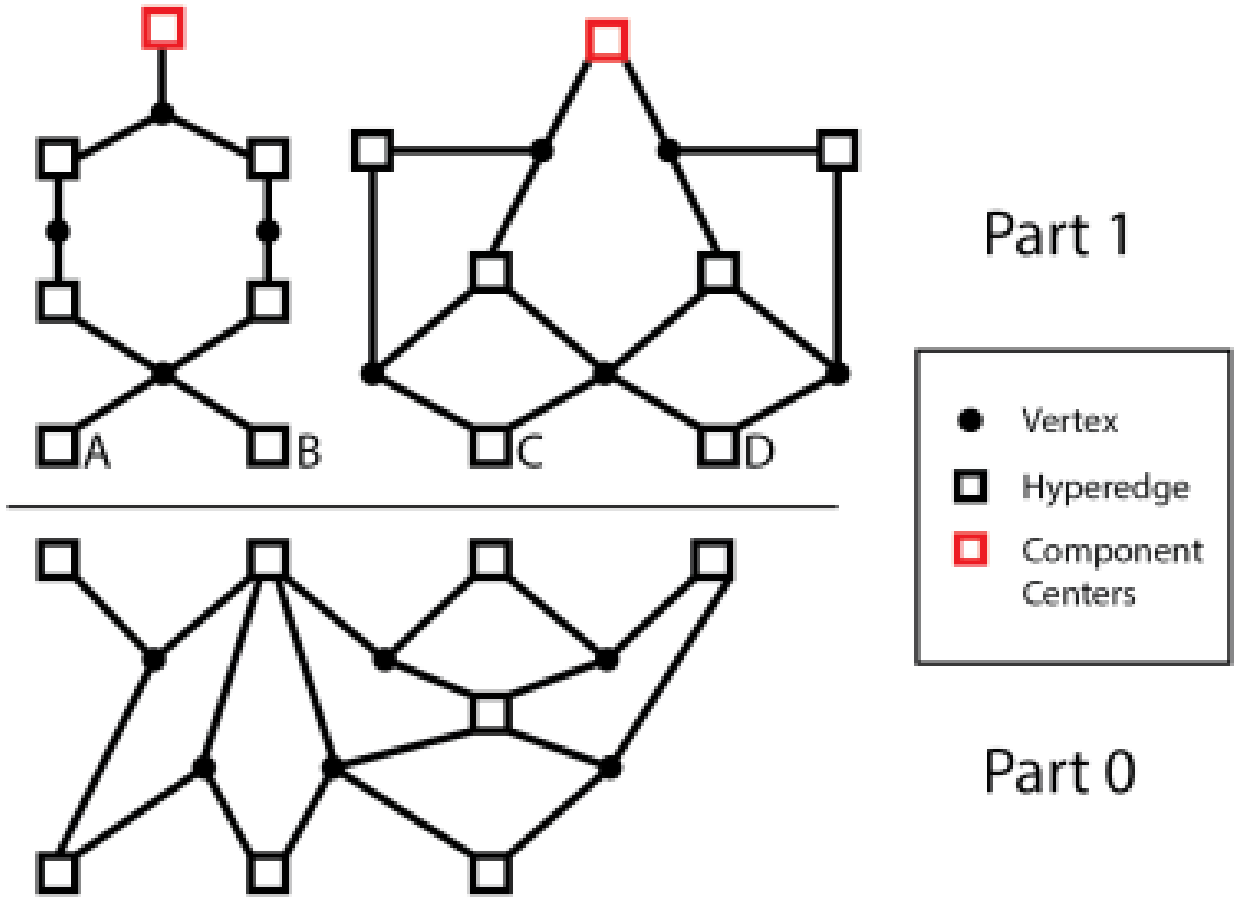
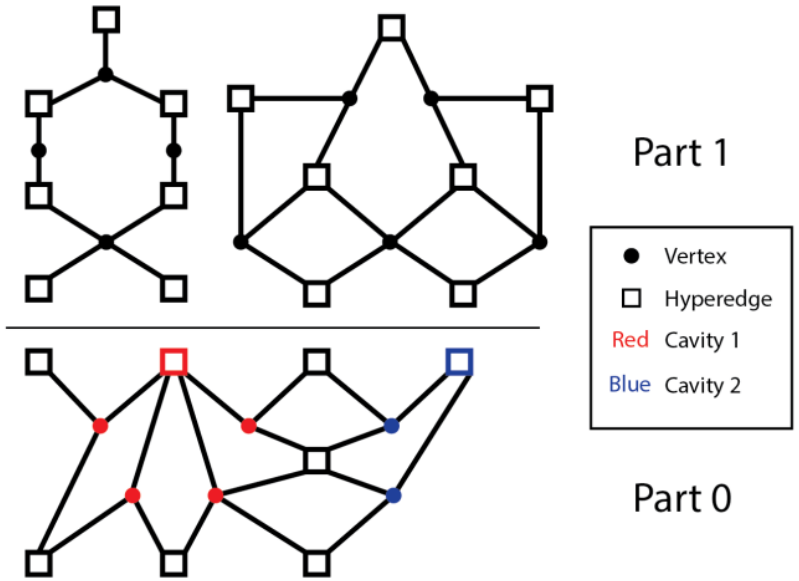


Figure: The queue computation for two disconnected components in Part 1. The centers of each component are marked in red. The queue for Part 1 is C,D,A,B

Selection

- Iterates over (hyper)edges that cross a part boundary.
- The cavity defined by the (hyper)edge is chosen for migration if:
  - The part that the (hyper)edge is shared with is a target part.
  - The target part has not been sent more weight than the limit.
  - The size of the cavity is small.



Problem Setup

We compare EnGPar's performance to its predecessor ParMA, which is built to operate directly on unstructured meshes.

ParMA and EnGPar are set to balance a mesh for a finite element analysis where:

- Scalability of matrix formation is sensitive to mesh element imbalance.
- Linear algebra routines are sensitive to the imbalance of degrees of freedom.
- For this we assume the degrees of freedom are associated with mesh vertices.

To partition for this, both balance mesh vertices followed by elements with a target imbalance of 1.05.

Problem Setup

Tests were run on a billion element mesh.

Initial partitions are built using:

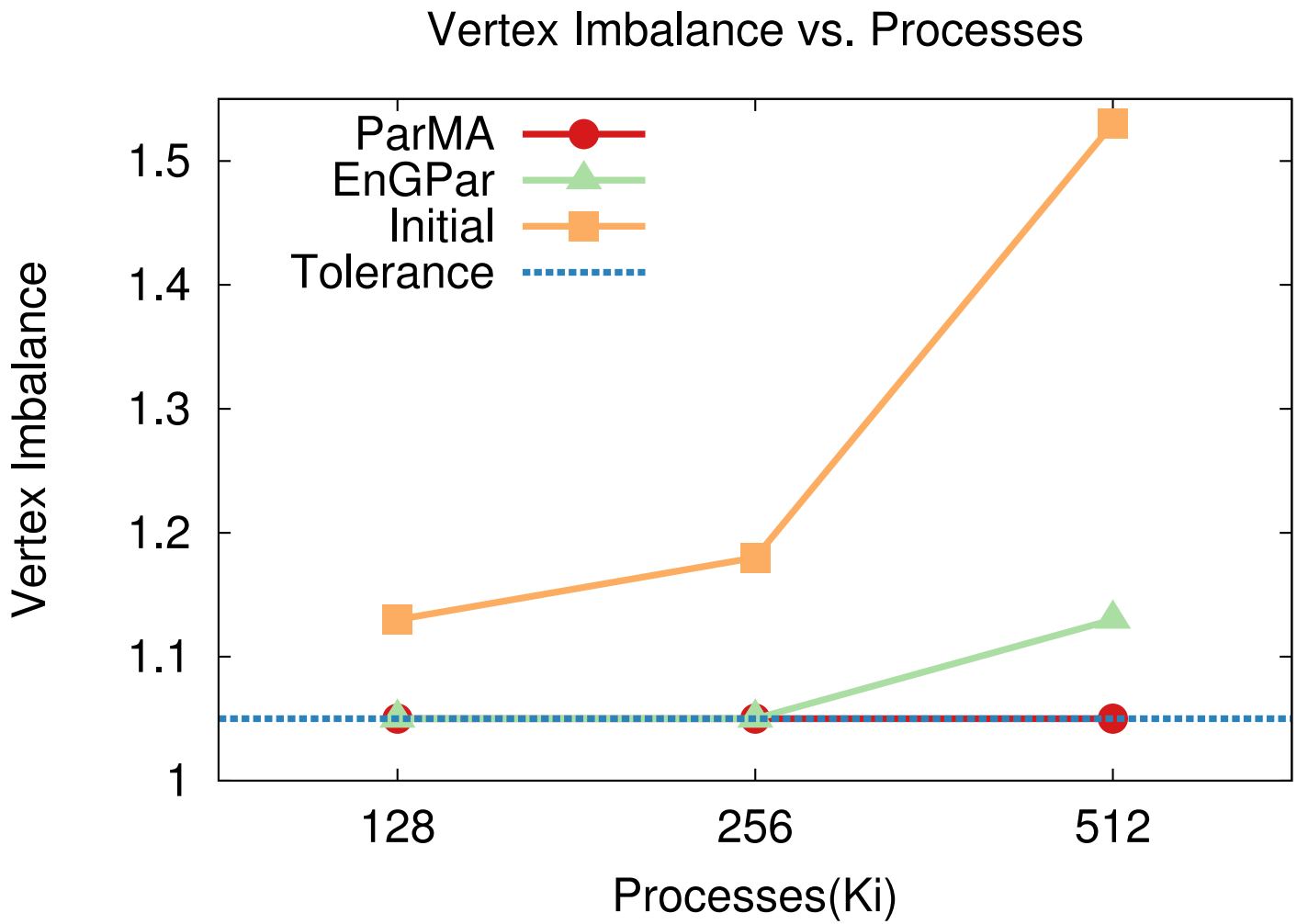
- Global ParMETIS part k-way to 8Ki($8 * 2^{10}$) parts.
- Local ParMETIS part k-way from 8Ki to 128Ki, 256Ki, and 512Ki parts.

The partitions before using EnGPar or ParMA are as such:

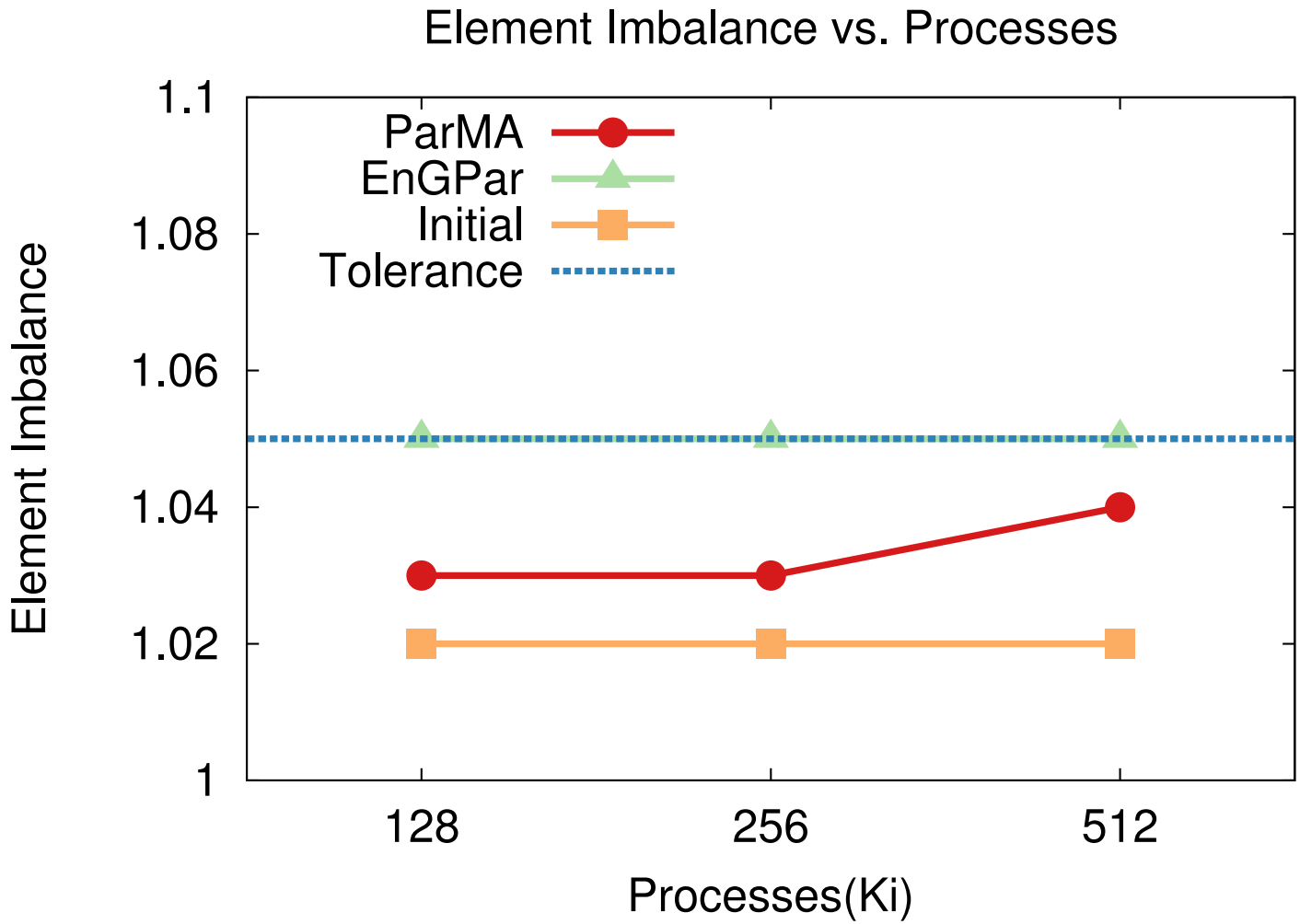
Number of Parts	128Ki	256Ki	512Ki
Elements per part	9,836	4,918	2,459
Vertex imbalance	1.13	1.18	1.53
Element imbalance	1.02	1.02	1.02

All experiments were run on the Mira BlueGene/Q system with one process per hardware thread.

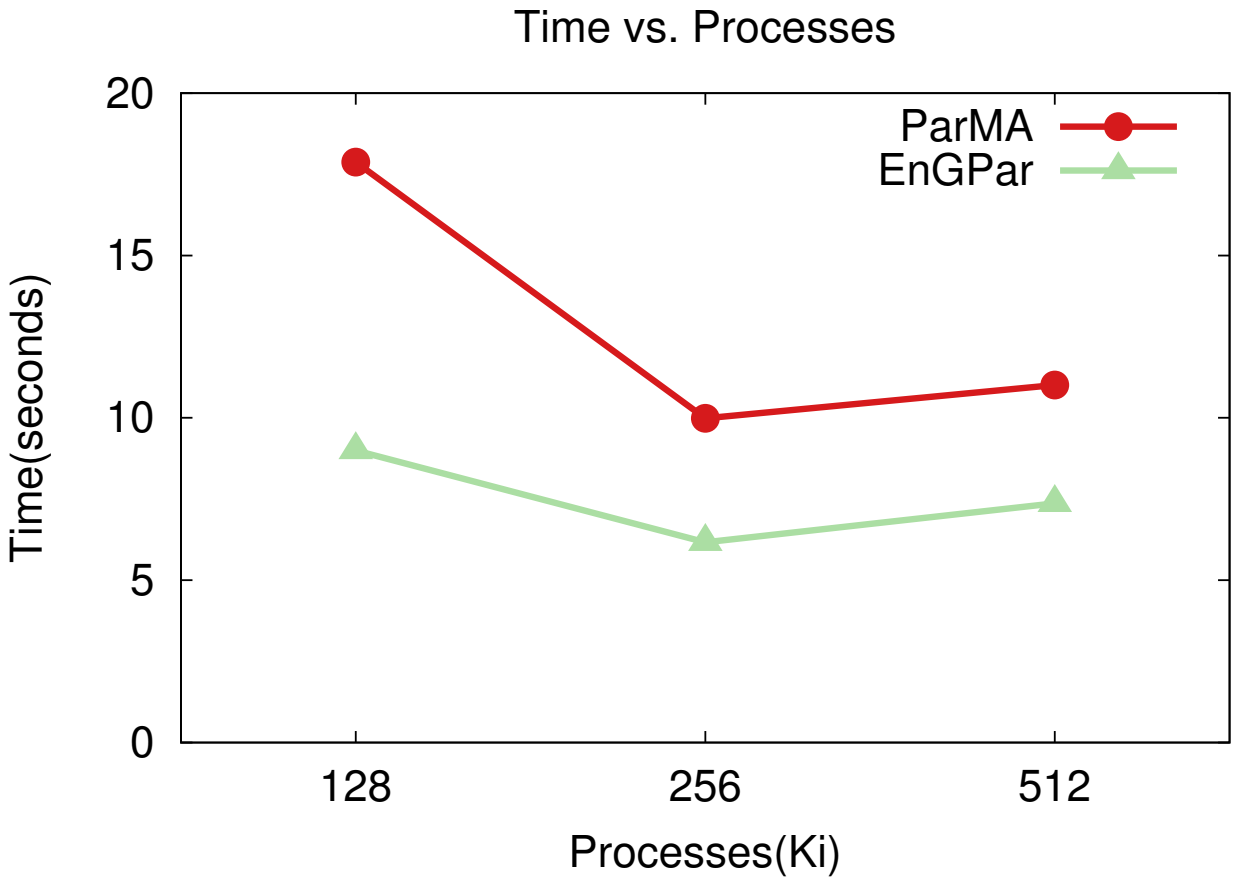
Mesh Vertex Imbalance



Mesh Element Imbalance



RunTime Comparison



Local ParMETIS creates the partition from 8Ki to 512Ki in 16 seconds followed by a PUMI migration that takes 131 seconds.

Closing Remarks

EnGPar builds off of ParMA by:

- generalizing the diffusive load balancing algorithm for applications without an element-based unstructured mesh.
- improving the algorithm of key portions to improve overall runtime.

The experiments show:

- EnGPar reduces the high mesh vertex imbalance from the 512Ki mesh from 1.53 to 1.13.
- EnGPar maintains the mesh element imbalance at the tolerance.
- EnGPar runs around 33% faster than ParMA.

Future Work

Expanding the capabilities of EnGPar:

- Improve partitioning at very high part count.
- Improve performance of the balancing procedures using accelerators.

Applying EnGPar to other applications:

- CODES - a discrete event simulation for communication on supercomputer networks. `press3.mcs.anl.gov/codes/`
- FUN3D - a computational fluid dynamic simulation using a vertex-based partitioned mesh. `fun3d.larc.nasa.gov`
- PHASTA - massively parallel computational fluid dynamics. `github.com/PHASTA/phasta`

Thank you.