MESH PARTITIONING

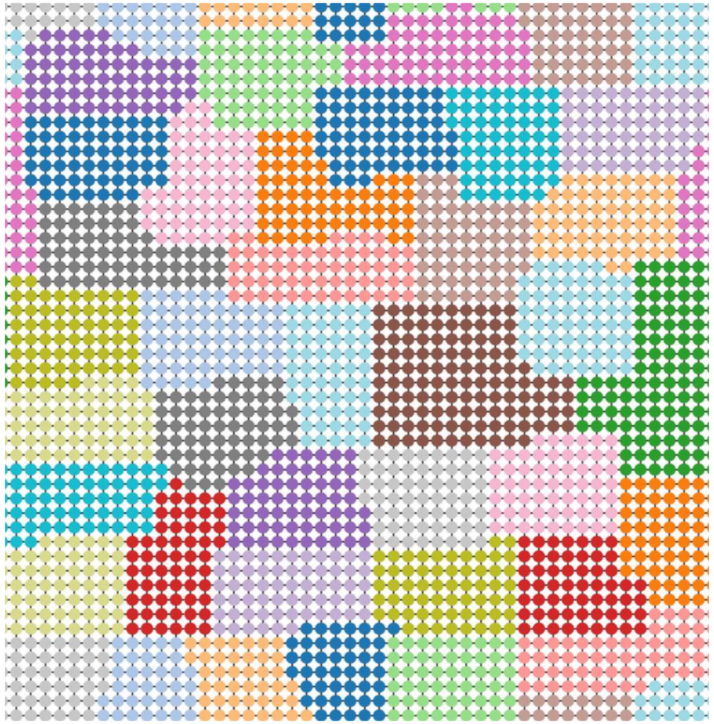
Harvey Kwong, MS CSE, University at Buffalo

University at Buffalo The State University of New York

What is mesh partitioning?

A mesh is a collection of simple geometric elements (e.g., triangles) connected to form some physical domain. Partitioning involves dividing the mesh into subdomains.

1. Balance partition sizes.
2. Minimize communication between parts (shared boundaries).



https://www.researchgate.net/figure/Graph-partitioning-using-Metis-Each-node-represents-a-component-of-the-Hamiltonian_fig2_349205240

Types of mesh partitioning Pt 1

	Graph-based (METIS)	Geometric (SFC)
Basis:	Connectivity graph of mesh	Physical coordinates of elements
Goal:	Minimize boundaries	Preserve spatial locality
Quality:	High-quality partitions	Fast, simple, perfect balance, not great boundaries
Computation cost:	Heavy preprocessing (graph construction)	Lightweight and parallelizable
Scalability:	Good but costly at scale	Excellent scalability

Types of mesh partitioning Pt 2

Graph Partitioners (METIS):

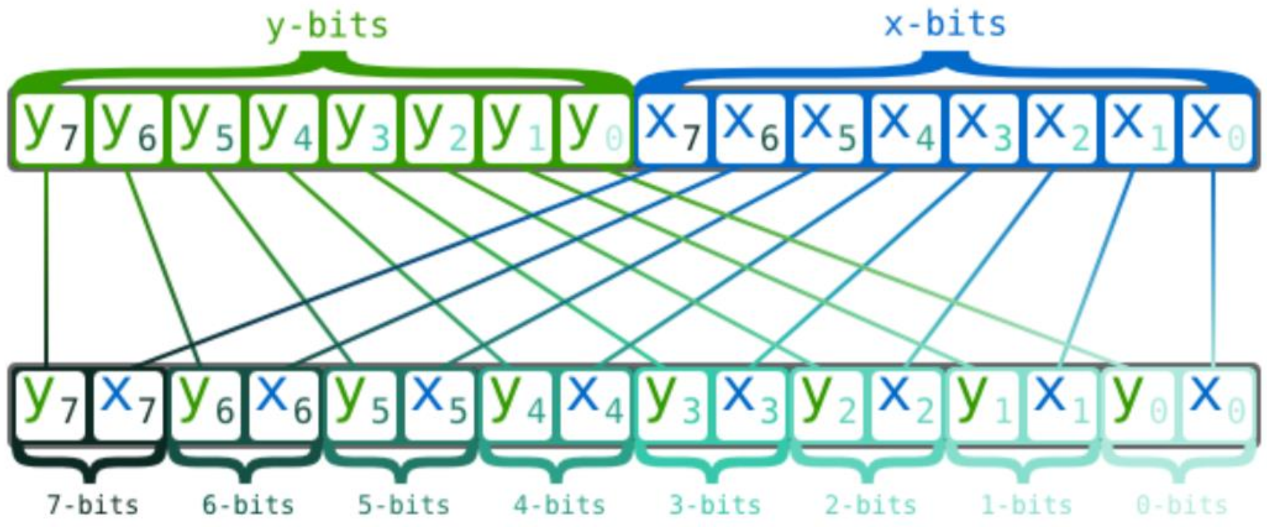
- Finite element method solvers where connectivity accuracy matters (stress analysis).
- Adaptive meshes that require fine control of boundary edges.

Geometric Partitioners (SFC):

- Particle simulations or fluid dynamics where spatial proximity is more relevant.
- Applications that require partitioning to be done very often/quickly.

The Morton Code (SFC)

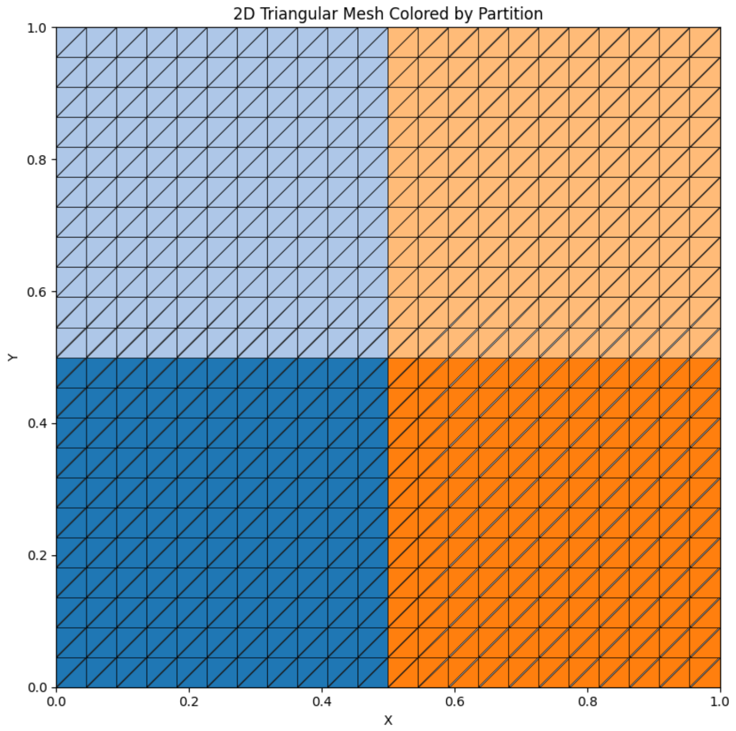
- The Morton code (or Z-order curve) converts each element's 2D or 3D coordinates into a 1D key by interleaving the bits of its x, y (and z in 3 dimensions) coordinates.
- Elements are then sorted by their Morton codes, producing an order that follows the Z-shaped traversal of space.
- This ordering ensures that spatially close elements have similar codes, preserving locality.
- To partition: simply divide the sorted list into equal-sized chunks. Each chunk forms a partition.



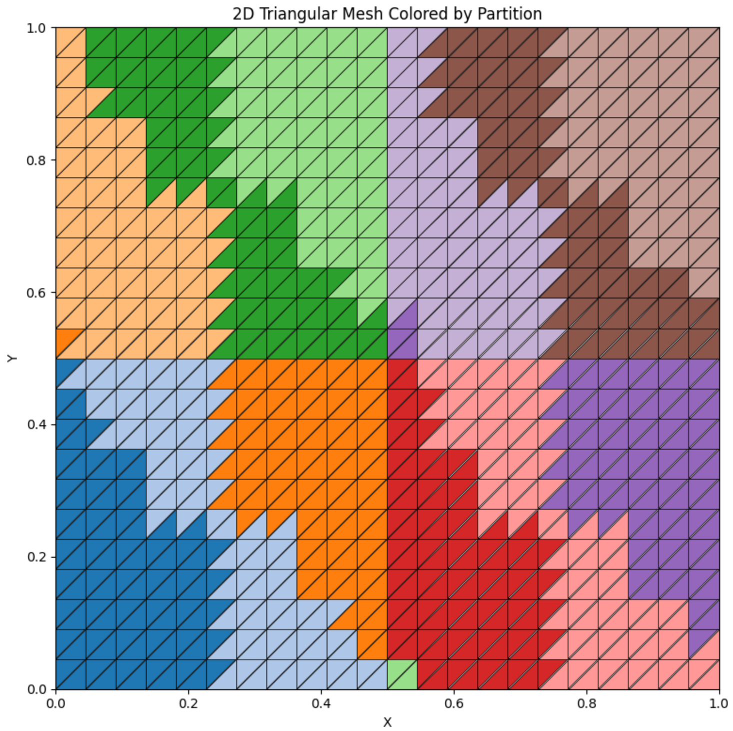
https://ashtl.sourceforge.net/morton_overview.html

My Morton Code Partitioning

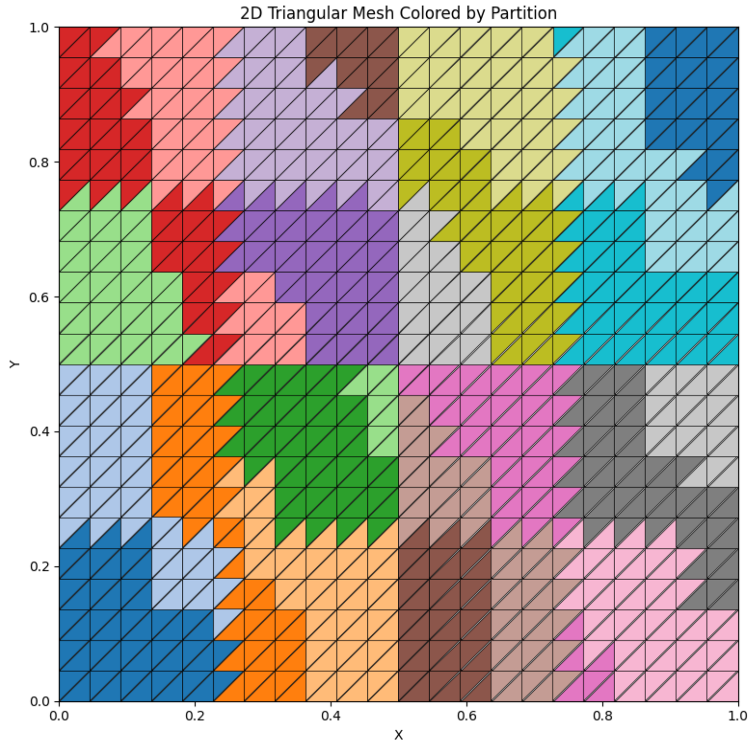
My implementation of morton code partitioning, applied on a regular triangular mesh of 1000 elements:



4 Parts



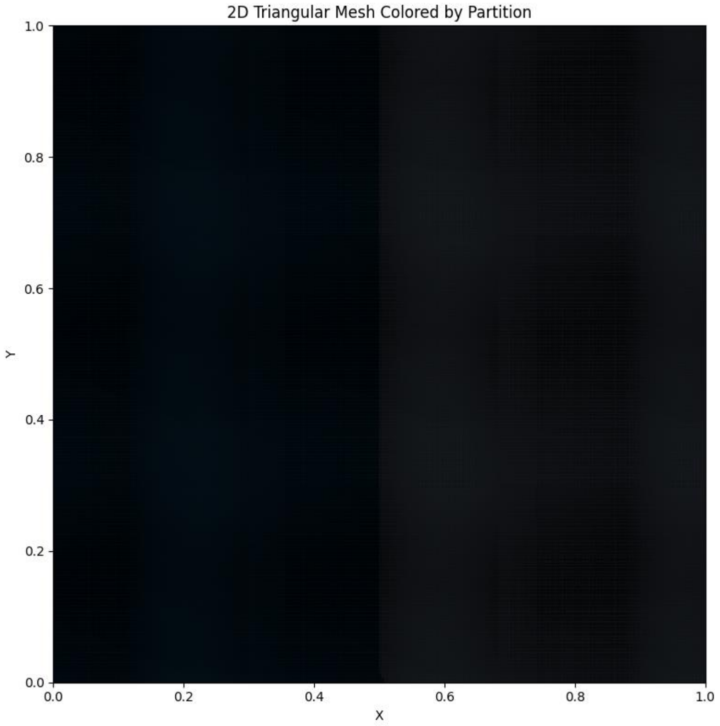
12 Parts



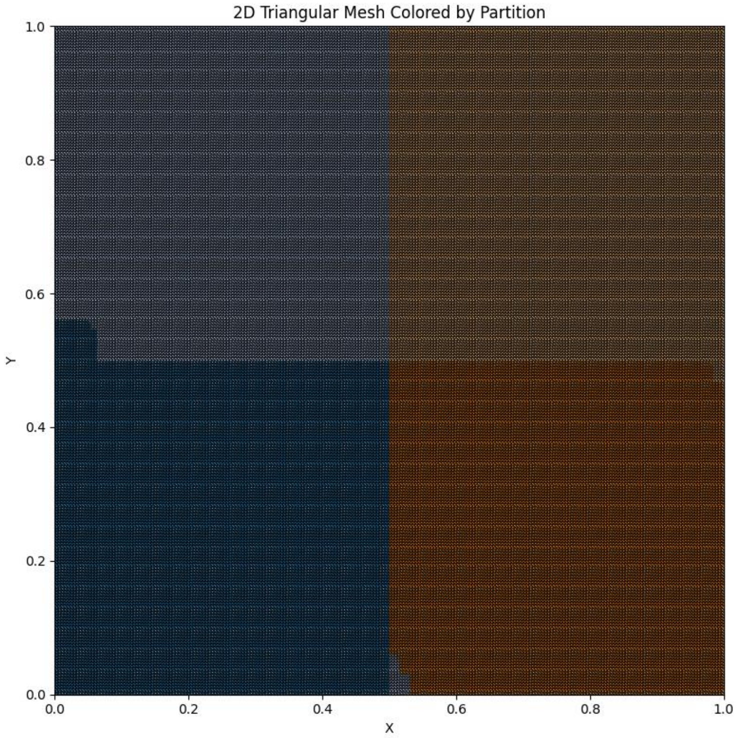
21 Parts

My Morton Code Partitioning (PARALLEL)

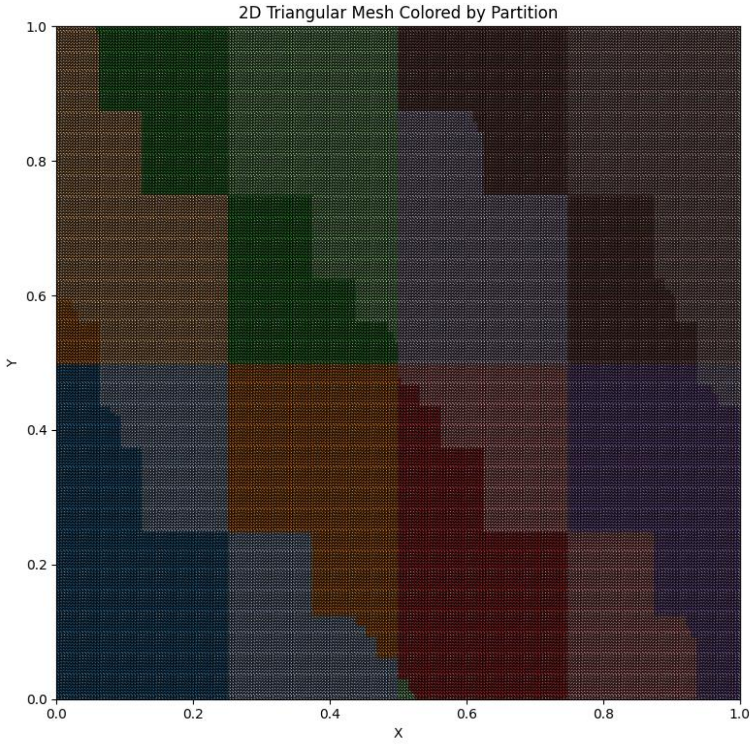
Parallel Morton code partitioning, applied on a regular triangular meshes



2 processors, 1 million triangles



4 processors, 100 thousand triangles



12 processors, 100 thousand triangles

How does the Parallel Implementation work?

In the sequential implementation, we computed the morton code per triangle, then sorted the codes, and distributed the sorted code list into equal parts for partitions.

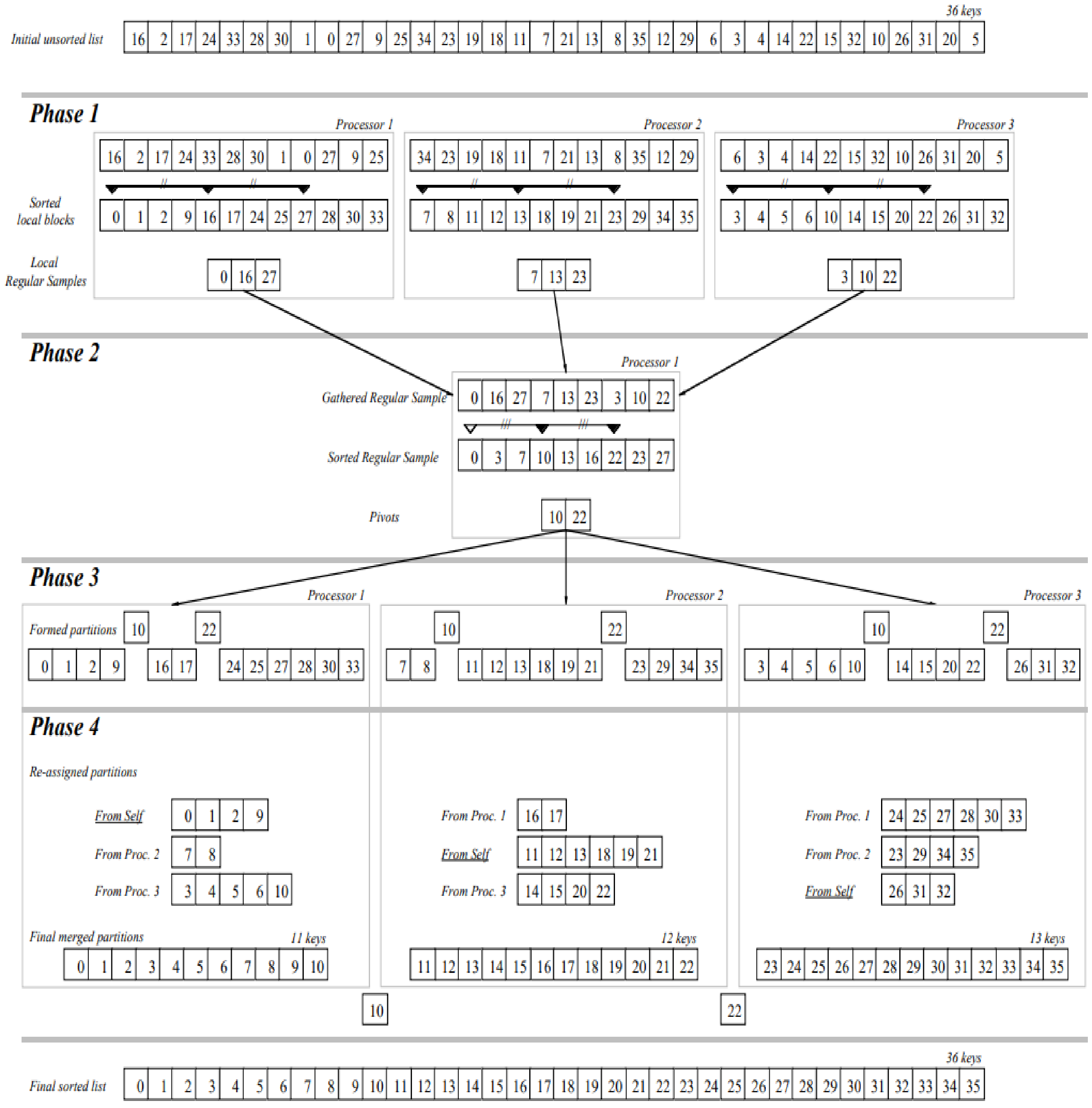
The parallel implementation works the same but with distributed data:

TWO VERSIONS:

1. Distributed mesh points across all processors
2. Distributed mesh points AND vertex-coordinate list across all processors

Both use parallel sample sort for sorting and one side communication for vertex-coordinate list.

Parallel Sample Sort



On the Versatility of Parallel Sorting by Regular Sampling

Mesh Partitioning Timings (Seconds)

Processors	10M	20M	40M	80M	160M
1	2.0148	4.3520	9.6266	19.152	41.078
2	1.5734	3.5140	7.2644	15.686	33.028
4	0.7349	1.3314	3.0937	6.5184	13.806
8	0.3800	0.8980	1.6649	3.3185	7.4129
16	0.2211	0.5075	0.9959	2.0103	4.4158
32	0.1852	0.3822	0.7125	1.3528	3.7757

Mesh Partitioning Speedup

Processors	10M	20M	40M	80M	160M
1	1	1	1	1	1
2	1.28	1.238	1.325	1.221	1.244
4	2.742	3.269	3.112	2.938	2.975
8	5.302	4.846	5.782	5.771	5.541
16	9.112	8.574	9.666	9.527	9.302
32	10.874	11.387	13.509	14.157	10.88

*Ideally, to scale strongly, the speedup is close to the processor count

Mesh Partitioning Efficiency

Processors	10M	20M	40M	80M	160M
1	1	1	1	1	1
2	0.64	0.619	0.663	0.61	0.622
4	0.685	0.76	0.778	0.735	0.744
8	0.663	0.606	0.723	0.721	0.693
16	0.57	0.536	0.604	0.595	0.581
32	0.34	0.356	0.422	0.442	0.34

*Ideally, for good weak scaling, the efficiency for fixed problem-size/processor-count ratio should be the same