Energy and Communication Efficient Partitioning for Large-scale Finite Element Computations

Milinda Fernando, Dmitry Duplyakin , Hari Sundar

Motivation

- When moving into Exascale (million way parallelism), computation cost is dominated by communication cost which makes it necessity to have efficient partitioning schemes.
- Space Filling Curve (SFC) based partitioning schemes are widely used in adaptive scientific computations.
- In order to reduce **overall communication cost imbalances** as well as **overall energy consumption** in Finite Element Method (FEM) computations we introduce **user defined load imbalance (*tolerance or slack*)** into SFC based partitioning schemes.

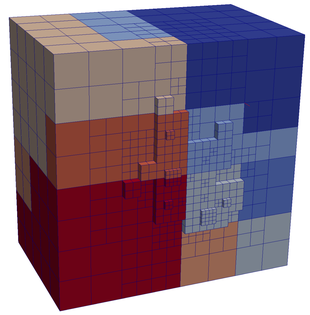


Figure 1: Octree partitioned using space filling curve, color coded by the process id (MPI rank).

Methodology

Space Filling Curves (SFC)

SFC is a surjective mapping between the one dimensional space to higher dimensional space. Hence using SFCs we can define a linear ordering of higher dimensional data which can be used in data partitioning and many other fields.

Finite element computations and MATVEC

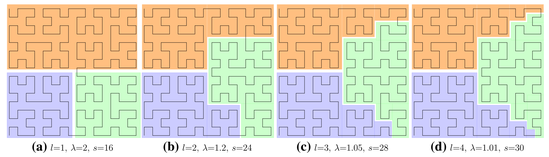
- Since our evaluation of the quality of the partition will be based on the evaluation of a MATVEC with the global finite element stiffness matrix, please refer to qr for additional details on performing distributed finite element computations using octree meshes.
- The MATVEC refers to a function that takes a vector and returns another vector, the result of applying the discretized PDE operator to the the input vector. As the building block for most large-scale PDE solvers,the cost, efficiency and scalability of the MATVEC operation will determine the efficiency and scalability of the overall system.

Flexibility in SFC based partitioning schemes

In this work we focus on how MORTON and HILBERT curve based partition schemes behave with a load flexibility (see Eq.1), in terms of ,

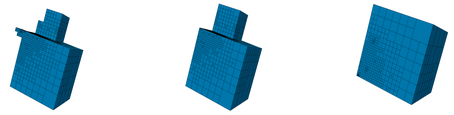
- MATVEC execution time
- Overall communication cost
- Energy consumption

$$\text{flex\_part} = \text{ideal\_part} \pm \text{tol} \times \text{ideal\_part} \quad (1)$$



(a) l=1, λ=2, s=16 **(b)** l=2, λ=1.2, s=24 **(c)** l=3, λ=1.05, s=28 **(d)** l=4, λ=1.01, s=30

Figure 2: Illustration of the increase in communication costs with low tolerance (ideal load balancing). Partitions for the case of $p = 3$ are drawn with the boundary of the partition (s) and the load-imbalance (λ) given along with the level (l) at which the partition is defined. At each level, the orange partition (■) gets the extra load that is progressively reduced. The green partition (■) gets the largest boundary that progressively increases.



(a) $tol = 0.001$ **(b)** $tol = 0.1$ **(c)** $tol = 0.5$

Figure 3: Octree partition which assigned to mpi rank 3 with varying tolerance. Note that the partition get smoother (reduced number of boundary surfaces) with higher tolerance value.

Octree generation, balancing (2:1) and meshing

In order to perform distributed memory FEM(MATVEC) operation we need complete following steps.

- **Octree generation**: Generates an adaptive octree for a given set of in a normal distribution.
- **Octree balancing**: We enforce $2:1$ balance constraint over the generated octree. Which means for a given octant all it's neighbours can be one level refined or coarsen.
- **Mesh generation**: Simply refers to building an an efficient data structure, which can efficiently retrieve neighbour node ids for a given octant (node).

Ghost octants & communication matrix

- Since we are performing distributed MATVEC operation, at the time of the meshing we exchange the octants (nodes) resides on other processes (know as ghost octants, see Fig. 4) which has read only access by others.
- Due to existence of ghost octants, each process need to communicate with subset of processes while performing MATVEC operation. Hence for the entire MATVEC operation we can define a communication matrix $\mathcal{M}$, where

$$\mathcal{M} = \begin{cases} m_{ij} & \text{if rank } i \text{ exchange } m_{ij} \text{ data with rank } j \\ 0 & \text{if rank } i \text{ need not to communicate with rank } j \end{cases} \quad (2)$$

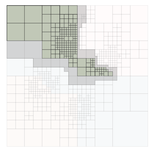


Figure 4: Demontration of ghost octants for a given partition in an adaptive octree. Assuming that the green octants belongs to partition a and all the grey octant would act as ghost octants for that partition which needs to be accessed when performing the distributed MATVEC operation.

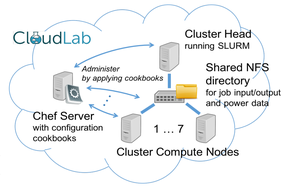


Figure 5: High level diagram showing main comphonents and their conectivity to measure the energy consumption of MATVEC operaton.

Energy & MATVEC

We provision resources and configure an 8-node cluster of physical machines on CloudLab, as shown in Fig. 5. While a set of selected 8-node jobs is running (scheduled using SLURM), we collect power draw measurements (obtained from on-board IPMI sensors) for every node every second. After job completion, we use the instantaneous draw samples (in W) to obtain per-job energy consumption estimates (in J), for both entire jobs and the MATVEC computation. **Hardware specs**: 2 Intel E5-2630 v3 8-core Haswell CPUs (2.40 GHz), 128GB ECC Memory, 10Gb Ethernet.

Results

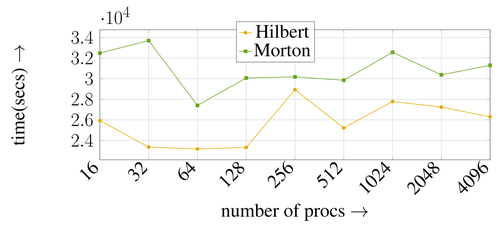
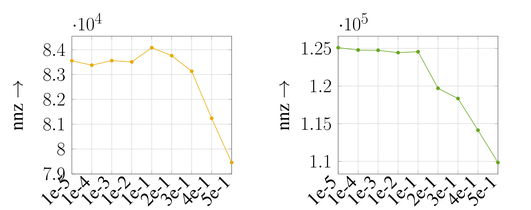
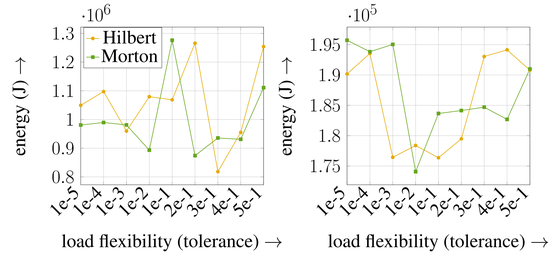


Figure 6: Maximum MATVEC execution time across all processes for weak scaling experiment with a grain size of $100K$ octants per process on Titan. HILBERT is faster for all cases. The fluctuations are due to the MATVEC code overlapping communication with computation.



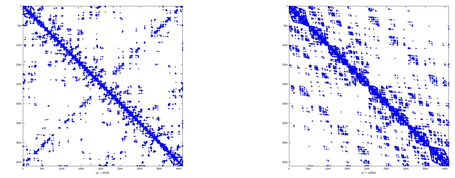
(a) Hilbert curve **(b)** Morton curve

Figure 7: Comparison for number of non-zeros (nnz) elements in the communication matrix corresponding to perform MATVEC operation based on HILBERT and MORTON based partitioning schemes for a mesh size of 1B nodes with 4096 mpi tasks with varying tolerance values in TACC's Stampede. Note that the scale difference between the axes in the plots, and for both partitioning schemes we can reduce the nnz (overall communication cost) by increasing the tolerance value.



(a) mpi tasks=256 number of cores=8 **(b)** mpi tasks=8 number of cores=8

Figure 8: Comparison for MATVEC energy consumption based on HILBERT and MORTON based partitioning schemes for a mesh size of 95M & 2M nodes with 256 mpi tasks & 8 mpi tasks with varying tolerance values in CloudLab. Note that the two scenarios depicts the effect of running 32 tasks in a single node Vs. 1 task in a single node.



(a) HILBERT **(b)** MORTON

Figure 9: Sparsity structure of the communication matrices of HILBERT and MORTON based partition schemes for a total mesh size of 1B nodes with 4096 mpi tasks ran on TACC's Stampedewith a tolerance value of 0.3. Note that the two matrices have different sparsity structures and HILBERT is more sparse compared to MORTON communication matrix.

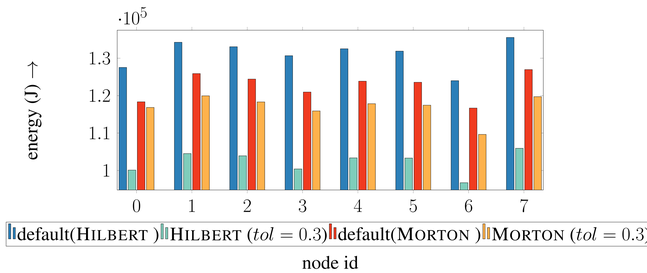


Figure 10: Energy consumed by each node while performing MATVEC operation , with ideal load balancing (for both HILBERT and MORTON) Vs. flexible load balancing with a tolerance of 0.3 for 95M mesh nodes with 256 mpi tasks in CloudLab8 node cluster.

Conclusions

- Flexible SFC-based partitioning schemes can reduce overall communication cost imbalances and energy consumption, by tolerating user specified load imbalance.
- Hilbert ordering is better than Morton ordering in terms of preserving the geometric locality of objects resulting less communication cost imbalances as well as overall communication cost.

More information and animations can be found via this qr.