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

Milinda Fernando, Dmitry Duplyakin , Hari Sundar

CONTRIBUTIONS

Our main contribution is the design of optimal $\mathcal{O}(N)$ sequential and $\mathcal{O}(N/p + \log p)$ parallel SFC-based partitioning algorithms that produce high-quality partitions.

- **Method:** Since SFCs are defined in a recursive manner, they converge progressively to the ideal work-load balanced partition, but increase the partition surface in the process. By using a performance model to determine the tradeoff between work-load imbalance and communication costs, we can terminate this recursion early and obtain a better partition.
- **Experimental Evaluation:** We conduct experiments to demonstrate the efficiency and scalability of our algorithm on ORNL's Titan upto 262,144 cores. We also include energy measurements for resulting MATVEC operations on CloudLab and demonstrate up to 22% savings. We will also release our code on github. (`https://github.com/orgs/paralab`)

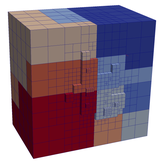


Figure 1: Octree partitioned using space filling curve, color coded by the process id (MPI rank).

METHODOLOGY

Modified SFC Ordering

Problem: Given that we are interested in generating a space-traversal, more so of application specific coordinates or regions.

Approach (TREESORT): It is efficient to consider this problem as one of generating a quadtree or an octree, in 2D and 3D respectively. Specifically, we construct the tree in a top-down fashion, one level at a time, arranging the octants based on the recurrence rules for the specific SFC, say Hilbert.

Key advantage

- If the SFC can dictate the ordering based on the level, as in the case of Hilbert, then these are applied to the ordering at this level with an O(1) cost. **Hence the run time is independent from the SFC curve (i.e HILBERT , MORTON) being used.**

Distributed TREESORT

- Unlike the sequential TREESORT, we have to traverse the tree in a breadth first fashion, as the data needs to be distributed across processors.
- Note that at each level, we split each octant 8 times (for 3D), so in $log_8(p)$ steps we will have p buckets. A reduction provides us with the global ranks of these p buckets.
- Using the optimal ranks $(n/p \pm tol)$ at which the data needs to be partitioned, we selectively partition the buckets to obtain the correct partitioning of the local data.

The expected running time (T_p) for the staged distributed TREESORT is,

$$T_p = t_c\frac{N}{p} + (t_s + t_w k)\log p + t_w\frac{N}{p}. \quad (1)$$

p	number of MPI tasks in $comm$
N	global number of elements in A
α	constant that depends on the application
W_{max}	max. of work assigned to an individual processor
C_{max}	max. of data communicated
t_w	interconnect slowness (1/bandwidth)
t_s	interconnect latency
t_c	intranode memory slowness (1/ RAM bandwidth)
k	Number of stages in MPI data exchange

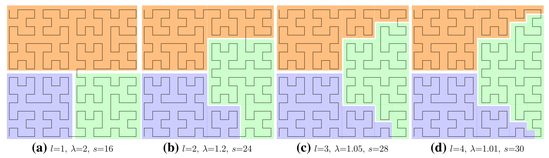
Table 1: *Here we summarize the notation used in this poster.*

Architecture Optimal Partitioning

Desirable qualities of any partitioning strategy are

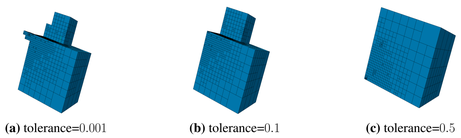
- load balancing
- minimization of overlap between the processor domains.

Why simple partitions perform better ? SFC-based partitioning does a very good job in load balancing but do not permit an explicit control on the level of overlap. Simple partitions, such as those producing relatively cuboid partitions have a smaller overlap compared with more irregular partitions (see Fig. 2 & Fig. 3), as might be produced by a SFC-based partitioning algorithm.



(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) tolerance=0.001 **(b)** tolerance=0.1 **(c)** tolerance=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.

Energy & MATVEC

- We provision resources and configure an 32-node cluster of physical machines on CloudLab.
- We collect power draw measurements (obtained from on-board IPMI sensors) for every node every second.
- **CloudLab node specs:** 2 Intel E5-2630 v3 8-core Haswell CPUs (2.40 GHz), 128GB ECC Memory, 10Gb Ethernet.

Can we predict the optimal tolerance value ?

- We would like to automatically determine the optimum tolerance based on the data and the machine characteristics.
- Computational time of MATVEC operation will be dominated by the processor that has the maximum load (W_{max}) or has to communicate the maximum (C_{max}) amount.
- We can try predict the execution time of distributed TREESORT using following model.

$$T_p = \alpha t_c W_{max} + t_w C_{max}. \quad (2)$$

RESULTS

Weak scaling in Titan

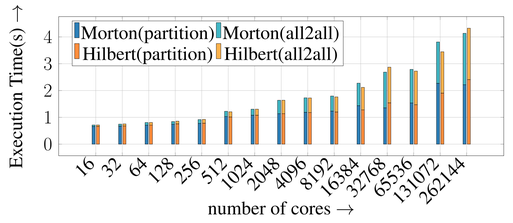


Figure 4: Total execution time for HILBERT & MORTON curve based partitioning scheme with, 10^6 grain size (minimum problem size of $80M$ & maximum problem size of $1.3T$ points), in ORNL's Titan varying number of cores from 16 to 262144. The total time is divided into time for computing the partition (partition) and the cost of actually exchanging data (all2all).

Strong scaling in Titan

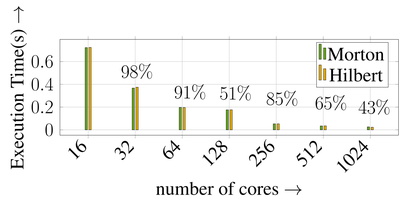


Figure 5: Strong scaling results for HILBERT & MORTON curve based partitioning scheme with, problem size of 16×10^6 points, in ORNL's Titan varying number of cores from 16 to 1024. The parallel efficiency for each case (rounded) is listed above the bars.

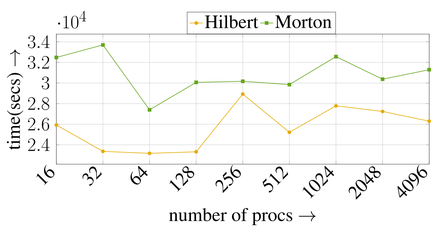


Figure 6: Maximum MATVEC execution time across all processes for weak scaling experiment with a grain size of $100K$ octants per process on Titan. HILBERT is faster for all cases.

Energy consumed by MATVEC

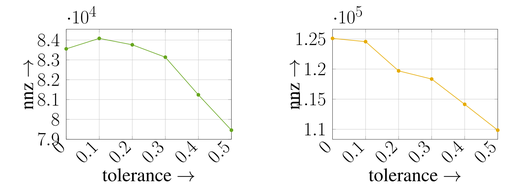


Figure 7: Comparison for number of non-zeros (nnz) elements in the communication matrix corresponding to perform MATVECoperation based on HILBERT and MORTON based partitioning schemes for a mesh size of 1B nodes with 4096 cores with varying tolerance values in TACC's Stampede . Note that the scale difference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

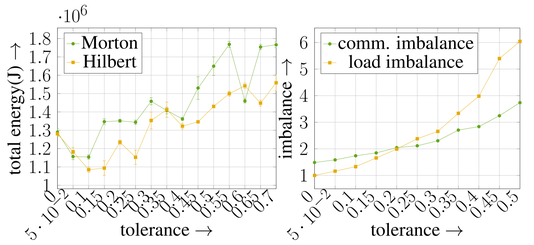


Figure 8: (left) Total energy consumption for 100 iterations of MATVEC(distributed) operations and (right) Load imbalance (work_max/work_min) and communication imbalance (bdy_max/bdy_min) plots for HILBERT curve based partitioning. In both cases we use an initial point grain size with 10^5 with maximum depth (of octree) of 30 across 1792 MPI tasks on the Clemson CloudLab cluster.

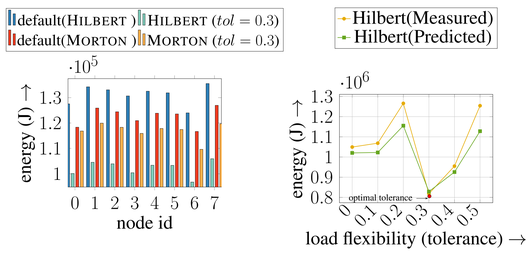


Figure 9: Energy consumed by each node while performing MATVECoperation , with ideal load balancing (for both HILBERT and MORTON) Vs. flexible load balancing with a tolerance of 0.3 for 95M mesh nodes with 256 cores in CloudLab8 node cluster.

Figure 10: Total energy consumed by the 100 MATVECoperations with 256 cores in Wisconsin CloudLab cluster and the interpolated energy values for HILBERT and MORTON based partitioning, using the model $\alpha t_c W_{max} + t_w * C_{max}$.

CONCLUSIONS

- We presented a new partitioning algorithm that by being architecture and application aware is able to reduce parallel runtime as well as overall energy consumption (22%).
- The key idea is to assign unequal work to processes in order to reduce overall communication costs.

More information and animations can be found via this qr.