Title
Parallel Performance Optimizations on Unstructured Mesh-based Simulations

Journal
Procedia Computer Science, 51(1)

ISSN
1877-0509

Authors
Sarje, Abhinav
Song, Sukhyun
Jacobsen, Douglas
et al.

Publication Date
2015

DOI
10.1016/j.procs.2015.05.466

Peer reviewed

Parallel Performance Optimizations on Unstructured Mesh-Based Simulations

Abhinav Sarje[1], Sukhyun Song[2], Douglas Jacobsen[3], Kevin Huck[4], Jeffrey Hollingsworth[2], Allen Malony[4], Samuel Williams[1], and Leonid Oliker[1]

[1] Lawrence Berkeley National Laboratory {asarje,swwilliams,loliker}@lbl.gov
[2] University of Maryland {shsong,hollings}@cs.umd.edu
[3] Los Alamos National Laboratory {douglasj}@lanl.gov
[4] University of Oregon {khuck,malony}@cs.uoregon.edu

Abstract

This paper addresses two key parallelization challenges the unstructured mesh-based ocean modeling code, MPAS-Ocean, which uses a mesh based on Voronoi tessellations: (1) load imbalance across processes, and (2) unstructured data access patterns, that inhibit intra- and inter-node performance. Our work analyzes the load imbalance due to naive partitioning of the mesh, and develops methods to generate mesh partitioning with better load balance and reduced communication. Furthermore, we present methods that minimize both inter- and intra-node data movement and maximize data reuse. Our techniques include predictive ordering of data elements for higher cache efficiency, as well as communication reduction approaches. We present detailed performance data when running on thousands of cores using the Cray XC30 supercomputer and show that our optimization strategies can exceed the original performance by over 2×. Additionally, many of these solutions can be broadly applied to a wide variety of unstructured grid-based computations.

Keywords: Unstructured Mesh, Ocean Modeling, Parallel Performance Optimization

1 Introduction

Iteration-based computational simulations require an abstract representation of the real-world state. In most cases, the data is represented as a collection of multidimensional data points where some of the data points define spatial coordinates. In such applications, the domain is discretized into *cells*, forming a *mesh* or *grid*. In a distributed parallel environment, these grids often provide a convenient framework for decomposition of the data into independent, groups of cells as *partitions*.

Structured meshes have regular cells and connectivity. For example, 2D structured meshes are typically represented with rectilinear or curvilinear quadrangles which can often be decomposed into partitions in a straightforward way. Structured meshes are appropriate when they are sufficient to define regular domains. One problem with structured meshes in defining certain domains, such as surface of spheres, is that they get distorted in order to enclose areas such as the poles of spheres, resulting in multiple grid points sharing the same location creating a *coordinate singularity* where the tangent space is undefined.

On the other hand, *unstructured* or irregular meshes have arbitrary polygon cells with irregular connectivity. As such, partitioning

of these meshes is often not obvious and generally driven by complex algorithms. These meshes are useful in applications requiring either multiscale and/or variable resolution, and they map better to curved surfaces, such as surfaces of spheres, avoiding coordinate singularities and major distortions when mapped to 2D [16]. Applications where unstructured meshes are applicable include adaptive mesh refinement, multi-resolution domains, and finite element methods on an irregular domain.

Often domain decomposition for distributed memory parallel computations require the use of *ghost* or *halo* cells from neighboring partitions when cells on the boundary of a partition require input from neighboring cells assigned to a different partition. Furthermore, decomposition of unstructured meshes that are not fully connected results in irregular partitions, potentially with high variability in the numbers of halo cells for each partition. This leads to severe load imbalance, increasing synchronization wait times and decreasing computational throughput. Deeper halo regions further exacerbate this situation.

The on-node memory access patterns of structured and unstructured codes tend to be very different. Structured meshes are convenient in that their layout in memory is amenable to optimizations such as blocking, tiling, good cache reuse and vectorization because the cells are often represented as multidimentional rectilinear arrays. Because of their regular shape and layout, they have a predictable access patterns. In contrast, unstructured meshes are a performance challenge as they are frequently represented as connected graphs with non-contiguous memory layouts. Iterating over the "neighbors" of an unstructured mesh cell can result in poor cache reuse due to pointer chasing since the neighbors may not necessarily be localized in memory.

In this paper, we address these two particular challenges put forth by unstructured meshes – load imbalance and unstructured data. The issue of load imbalance is discussed and addressed in Section 3, and the issue of unstructured data is addressed in Section 4. We present our results and solutions in the context of an ocean modeling code, MPAS-Ocean [16], which we briefly describe next.

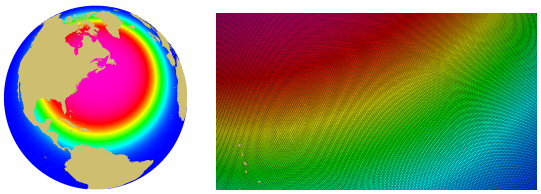

Figure 1: (Left) Global multiscale ocean mesh with refinement over the North Atlantic. The coloring represents variable resolutions with magenta corresponding to finest resolution to blue for coarsest. (Right) Detail of a region from the global mesh. The actual mesh is overlaid to show the smooth transitions of the mesh resolutions.

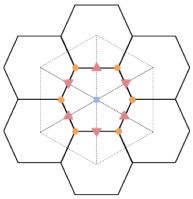

Figure 2: Diagram showing the MPAS-O horizontal staggering with variables stored within cells and on edges and vertices. Blue squares represent scalar data (mass, temperature, etc.), orange circles represent vorticity related quantities, and red triangles represent vector quantities (velocities).

2 Background

2.1 Ocean Modeling with MPAS

The Model for Prediction Across Scales (MPAS) is a modeling framework collaboratively developed between Los Alamos National Laboratory and the National Center for Atmospheric Research [16]. The MPAS framework is built on top of unstructured mesh data structures that make heavy use of indirect addressing. In this paper, we present our optimization efforts on unstructured meshes in the context of the ocean modeling core, MPAS-Ocean.

MPAS-O is a next generation global ocean

model, built on top of spherical centroidal Voronoi tessellations (SCVT). The resultant meshes are composed of arbitrarily shaped polygons which bring unique challenges to parallel load balancing and data locality. These meshes can be generated using a density function to produce static mesh refinement in areas of interest (see Fig. 1). A major advantage of such mesh is that it provides smooth resolution transition regions, while allowing for drastically different resolutions in different regions of the mesh. It also eliminates the need for "hanging node" issues that are common in regular structured variable resolution meshes, while still providing refinement in areas of interest. Building off of this unstructured mesh, MPAS-O enumerates its data on a staggered horizontal Arakawa C-Grid, visualized in Fig. 2. While the horizontal data structure is staggered and unstructured, this mesh has a vertical (ocean depth) data structure that is regular and structured.

Global climate modeling is considered a grand challenge problem due to the spatial and temporal scales required to accurately simulate the phenomena. Temporal scales for climate models are on the order of centuries, while spatial scales are on the order of tens of kilometers. The requirements for performing global climate model simulations require models such as MPAS-O to simulate on large number of horizontal degrees of freedom of $O(10^6)$ to $O(10^7)$. In addition, MPAS-O performs simulations using generally 100 vertical levels per horizontal degree of freedom. These imply MPAS-O requires efficient execution on high concurrency systems to drive ocean analysis simulations.

2.2 Load Balance

Load balancing across processing units in parallel applications is a widely researched topic due to its importance in gaining higher parallel efficiency. Load imbalance leads to underutilization of computational resources. Achieving an efficient load balance is particularly challenging with unstructured meshes because their balanced decomposition is not obvious [3, 12]. As unstructured meshes are easily representable as graphs, many of the decomposition strategies are based on the graph partitioning algorithms. Although the graph partitioning problem is NP-complete, a number of heuristic-based partitioning algorithms exist [9, 15, 17, 21].

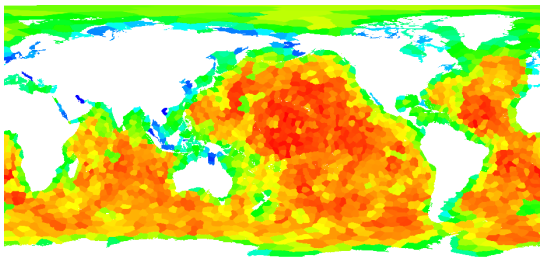

Figure 3: An example visualization of the load imbalance in an execution of MPAS-Ocean when the mesh is partitioned using the Metis tool. The grid used here consists of 1.8 million cells decomposed into 2048 partitions, one per process. Blue parts have low load and red parts have high load. In this example, highest load process performs about 3× as much work as the lowest load process.

MPAS-O uses a popular graph partitioning tool, Metis [13]. As will be discussed later, using this approach directly causes significant imbalance across processes. In addition to the unstructured nature of the mesh, a primary factor causing the imbalance is the use of deep halo regions in this application. The cell counts in halo regions of different mesh partitions is hence highly variable. A visualization showing variability in computation and communication loads across partitions in a sample execution of MPAS-O is shown in Fig. 3.

It is known that graphs do not model parallel communication volume well. Hypergraphs, on the other hand, are able to model the communication volume accurately [5, 10, 11]. Hence, in our work to improve the mesh partitioning, we build a partitioner using hypergraph partitioning [6, 10, 18, 19]. We analyze and address this problem of load balance in Section 3.

2.3 Data Locality

Flops are free is the mantra of today's large-scale parallel systems. This refers to the observation that the primary bottleneck in nearly all scientific applications today is data movement and not computations. This gap in the cost between computation and data movement is only going to grow wider in future. Optimizing data movement (both volume and bandwidth) is, hence, essential in obtaining high performance. To affect this, the main strategies are (1) maximizing reuse of data once it has been loaded (through reordering data traversal, and "communication-avoiding" algorithms), and, (2) maximizing bandwidth via streaming access patterns amenable to acceleration by hardware stream pre-fetchers.

To improve intra-node performance, it is essential that the memory hierarchy (caches) is used efficiently in data accesses. Data locality, both spatial and temporal, is an important factor in maximizing efficiency. When operating on unstructured meshes, indirect and semi-random access to data presents a number of challenges to maximizing cache reuse and attaining high bandwidth. In this paper, we increase data locality by reordering the input mesh data to make sure that spatially nearby cells are stored nearby in the memory as well. Researchers have previously used various strategies for improving cache performance by introducing locality into structured meshes [2]. In [20], although the authors consider unstructured data, they work with meshes with high-degree of regularity. Here, we address this problem for generic unstructured and multiscale meshes in Section 4.

2.4 Computational Environment

The experiments described in this paper were executed on *Edison*, a Cray XC30 system at NERSC. Each Edison node consists of two 12-core Intel Ivy Bridge sockets. Each socket is attached to 32 GB of DDR3 DRAM via a memory controller providing about 44 GB/s of bandwidth. Nodes are connected via Cray's high-performance Aries (dragonfly) network.

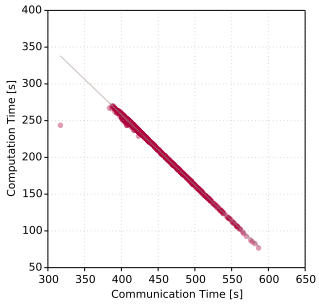

Figure 4: The variation in the computation versus communication times for all processes, represented by a dot, during an example simulation using a partitioning obtained from Metis. This simulation consists of 1,536 processes, each owning a mesh partition. Communication time varies by about 42% and computation by 75% showing a large imbalance.

In all the experiments presented in this paper, we run with 12 MPI processes per socket.

The mesh used is a 15 km resolution world ocean mesh, consisting of about 1.85M cells and 5.5M edges. Each cell has 3 to 40 vertical levels, representing a normalized vertical depth of the ocean at that cell.

3 Mesh Partitioning

3.1 Load Imbalance Analysis

As previously mentioned, MPAS-O uses Metis to decompose an input mesh into partitions. Each partition may then be assigned to a different processors during simulations. Deep halo regions are created for each partition which represent the data that needs to be communicated from the neighboring partitions, performing "halo-exchanges". To demonstrate the computational and communication load imbalance due to a given Metis partitioning, across the different processes during an execution, we show the variation in the computation and communication time for each of the process for a sample run in Fig. 4. It can be observed that in this example the computation and communication times vary by 75% and 42%, respectively, showing a high imbalance.

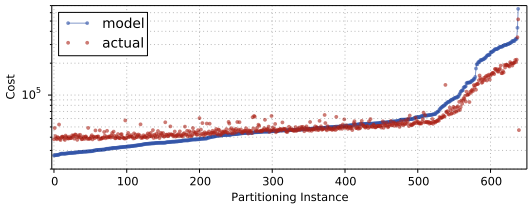

Figure 5: Total cost predicted using our model compared with actual run times with respect to 700 different mesh partitioning obtained by using PaToH with different parameter configurations.

Available partitioners generate a partitioning by (1) balancing the number of cells in all partitions, and (2) minimizing the total number of edge cuts. Based on our analysis of load-imbalance due to a straightforward Metis partitioning, it is clear that a different approach is required to achieve better load balancing, which would effectively balance the computation as well as the actual communication distribution. Therefore, this partitioner should balance the partitions with respect to total cells including the halo cells, and minimize the communication volume. We turn to partitioning via hypergraphs using the PaToH partitioner [7], which promises to model the communication overhead more accurately.

The communication overhead between processes is due to halo exchanges, and because of the unstructured and non-contiguous nature of meshes under consideration, each partition has different number of halo cells, with high variability across all partitions. The effect of halo cells is magnified when deep halos are used. MPAS-O typically requires halos with depth of three cells. Since graph/hypergraph partitioning algorithms do not take these halo cells into account, a different partitioning strategy is necessary for a better balance. In order to develop a *halo-aware* partitioning scheme, we first construct a cost model for a given partitioning to accurately represent computation and communication costs, and then use this model to design our partitioning algorithm.

3.2 Cost Modeling

The computational cost for a processor includes the cost due to computations on the local cells and on the halo cells. The cost may be different for a local cell and a halo cell. For p total partitions, let a represent the ratio between computation performed on a halo and local cells. Given a partition k, we model its computational cost, C_α as:

$$C_\alpha = \frac{1}{F(p)} \left(\sum_{i}^{i \in N_k} w_i + a \sum_{i}^{i \in H_k} w_i \right) \quad (1)$$

where N_k is the set of local cell for partition k, H_k is the set of halo cells for partition k, and w_i is weight of a cell i. $F(p)$ is a function of p, the number of partitions, and represents the fact that the cost decreases with increasing p.

The communication volume depends on the number of halo cells for each partition, and the number of neighbors each processor needs to communicate with. We model the communication cost for a partition k as:

$$C_\beta = \frac{1}{F(p)} \frac{h_k}{b_k} + \left(\max_{i \in [1,p]} (c_i) - c_k \right) \quad (2)$$

where $h_k = |H_k|$, b_k = number of neighbors of partition k, c_k is the total computational cost of partition k. Hence, the first term represents the average communication load to each neighbor, and the second term represents the wait time for processor owning partition k.

We perform extensive experiments with the actual cost of representative simulations with varying concurrency and input meshes, and use the obtained performance data to perform a least-squares line fit in order to find the computation-communication ratio, and the value of $F(p)$. From the data collected from these experiments, we obtain the fitted values $a = 0.7$ and $F(p) \approx \log(4p)$. We confirm the correctness of our cost model for any given mesh partitioning as shown in Fig. 5.

3.3 Halo-aware Partitioning

To model deep halos, we could naively use BFS or DFS starting from each cell and identify all

the l distance neighbors to construct l-depth halos. However, given that graphs can be represented as sparse matrices, a more efficient approach is to compute the l-th power of a matrix, through sparse matrix-matrix multiplication (SpMM), which identifies all nodes in the corresponding graph that are at a distance l and connects them with an edge. Therefore, given A as the sparse matrix representation of the input mesh, computing A^l determines the l depth halo cells. We utilize the CombBLAS package [4] to perform the matrix power operations. Aggregating the edges from $A, A^2, ..., A^l$ identifies all the halo cells for any given partitioning.

However, because the halo cells cannot be identified until a partitioning has been performed, their cost cannot be added for balancing during the partitioning process. Therefore, our new partitioning strategy follows an iterative approach. The strategy is designed as a Monte-Carlo based partitioning scheme that iteratively refines the partitioning using total cost estimated for previous iteration's partitioning. Each iteration, thus, includes a call to the hypergraph partitioning tool PaToH. This scheme is as follows for an input mesh $\mathcal{G}$:

procedure HALOAWAREPARTITION($\mathcal{G}$)
  Construct sparse matrix A representing $\mathcal{G}$
  Compute $A^2, \cdots A^l$
  Compute $A_{1\cdots l} = \sum^{l} A^i$
  Construct hypergraph $\mathcal{H}_0$ for $A_{1\cdots l}$
  while not converged **do**
    Construct partitioning P_i of $\mathcal{H}_i$
    Construct halos for P_i
    Compute cost prediction for each partition k and assign corresponding weights to the cells (distributing the cost of halo cells among partition cells), and hence compute partition weights W_k
    Compute imbalance, $f_i = \left(1 - \frac{\min_k(W_k)}{\max_k(W_k)}\right)$
    if $f_i < f_{i-1}$ **then**
      $m = 1$
    else
      $m = \min\left(1, e^{(f_{i-1}-f_i)/2}\right)$ ▷ (*Metropolis-Hastings* probability)
    end if
    Accept P_i with probability m
    if P_i is accepted **then**
      Update $\mathcal{H}_i$ with the new cell weights to construct $\mathcal{H}_{(i+1)}$
    else
      Reject P_i by $P_i = P_{i-1}$ and $f_i = f_{i-1}$
    end if
  end while
  Output last accepted partitioning as result
end procedure

The above procedure ensures that the cost due to halos are taken into account by assigning weights to the cells and refining the partitioning. Experimental results show that the convergence is reached quickly, generally within 10 iterations.

3.4 Variable Cell Weights

In general a mesh may have variable cell weights. For MPAS-O the cells in the input mesh have variable depths, representing the ocean depth at the corresponding location. This introduces another opportunity for optimization, since the computational cost of a cell is proportional to the number of depth layers it defines (varying between 3–40). To take this into account, variable weights corresponding to the cell depth are assigned to each cell while calculating the cost using the performance model. The next subsection presents the impact of our depth and halo-aware partitioning strategies compared with assigning the original partitioning.

3.5 Performance Results

We perform experiments on the Edison system (described in Section 2.4), using a 15 km resolution mesh to simulate 10 days, with the following four partitioning strategies: (1) *Original* Metis partitioning. (2) *Hypergraph* partitioning from PaToH. (3) *Halo-aware* partitioning. (4) *Depth and halo-aware* partitioning. Each partitioning used in the following experiments was selected from a sample size of 10 obtained from the corresponding partitioning scheme, on the basis of best performance.

The performance as communication and computation times of each run is shown in Figure 6. These plots capture the maximum to minimum ranges for all processes in a run. Observe that the communication time drops significantly for the partitions obtained from our new partitioners, with a small increase in the computation time. This is because

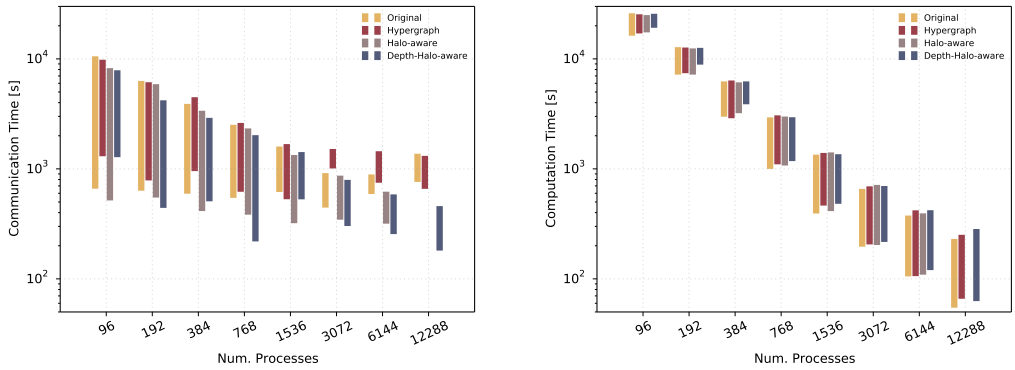

Figure 6: (Left) The ranges of time involved in communication related tasks among all processes in each run. (Right) The ranges of time spent in computations among all processes in each run. Comparison of the four types of partitioning is depicted for varying number of processes. The number of partitions is equal to number of processors, and each partition is assigned to a different processor.

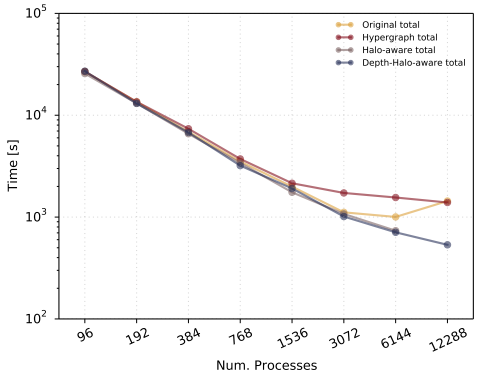

Figure 7: Total application execution times with varying number of processes are shown for each of the four types of partitioning. The number of partitions is equal to number of processors, and each is assigned to a different processor. Corresponding communication time is also shown in lighter color for reference.

the new partitioner attempt to combine the computation and communication cost, and better overall cost may be achieved by increasing the maximum compute time and decreasing the maximum communication time. It should be noted that these plots do not show the correspondence between the minimum and maximum of computation and communication time for each process, and total time taken is not simply the sum of the two. The total execution time taken by each run is shown in Fig. 7. It can be seen that at 12,288 cores, performance improvement of our scheme is more than 2× over the base graph partitioning. Lower concurrencies also show performance improvements, although less dramatic. Note that due to the significant reduction in communication costs, our partitioning scheme improves performance as well as scalability.

4 Ordering Data

An unstructured mesh implicitly means that the data layout is also unstructured, causing the involved data structures to be mapped to the memory in no particular order. Taken as is, there is nearly no locality, leading to inefficient use of the memory hierarchies. In our case, the meshes have no regularity, and no assumptions regarding the data ordering can be made. Since maintaining locality is essential for high performance, particularly in presence of memory hierarchies, we need to introduce spatial data locality by re-arranging data so that the data with temporal locality reside nearby in the memory as much as possible. Currently, since the MPAS-O mesh can be directly represented as a sparse matrix, to create locality, its

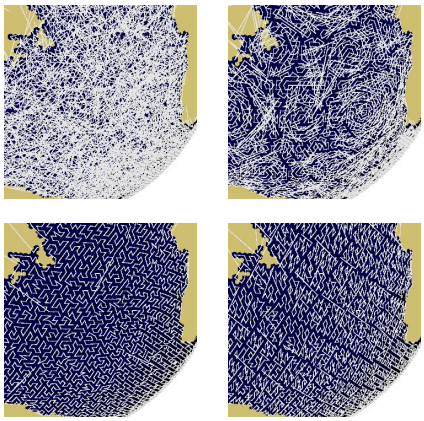

Figure 8: A part of the mesh covering Earth's oceans is shown for each of the four cell orderings: random (top-left), original (top-right), Hilbert (bottom-left) and Morton (bottom-right). The ordering of the cells is shown using white lines connecting the cells.

generation process follows a cell ordering called the *reverse Cuthill-McKee* [8] (RCM) ordering, which is an algorithm for permuting a symmetric sparse matrix to convert it into a band matrix with a small bandwidth. We will show below that even though it follows some locality pattern, its performance can get nearly as low as a complete random ordering.

4.1 Space Filling Curves

Space Filling Curves (SFCs) [1, 14] are generally used to map a higher dimensional data on to a one-dimensional line. A number of SFCs have been developed and have proven to be widely applicable for solving scientific problems as well as for improving performance of applications. In our case, SFCs are an obvious choice to explore for the introduction of data locality, but due to the lack of any regularity in the underlying mesh, they cannot be directly applied since SFCs are defined for regular and/or repeating grid patterns. Fortunately, the SCVT mesh under consideration exists in 3D space, which allows overlaying a structured grid on top of the unstructured mesh for the purposes of data reordering.

Given an unstructured mesh defined in a 3-dimensional space, we represent each cell as a single point at its centroid. We then define an octree-based rectangular grid enclosing all these points. This grid constructed by recursive decomposition such that each point resides in a separate leaf node. Once we have this representation, we simply order the octree leaves according to an SFC of choice. This leads to an ordering of the corresponding points representing the unstructured mesh. This ordering is then mapped onto the corresponding mesh cells to reorder the data.

Due to the mesh definition on a spherical surface and the discontinuities, the application of the above strategy to the complete mesh at once causes a number of "jumps" between cells along the ordering, which are not spatially close but appear next to each other in the ordering. Hence, instead of reordering the full mesh, we apply ordering to each mesh partition independently. This minimizes the number of jumps and allows for better data locality. We considered several SFCs to improve simulation performance, and present two top approaches: Hilbert curve, and Morton ordering (or Z-SFC). Examples of the ordering of the mesh cells produced by these curves is shown in Fig. 8. In addition to ordering the cells, we also follow the same scheme to reorder the vertices and edges in the mesh.

4.2 Performance Impact

We perform experiments to analyze the performance due to each ordering. In Fig. 9, the results are summarized as the total application execution time taken using each of the following four mesh orderings: (1) *Random* ordering, for reference. (2) *Original* RCM ordering. (3) *Morton* SFC ordering. (4) *Hilbert* SFC ordering. The on node performance improvement observed for the Hilbert and Morton orderings is as high as 40% with respect to the original (RCM) and random orderings. All these experiments have been performed using our depth and halo-aware partitioning, hence, the observed benefits are on top of the previously

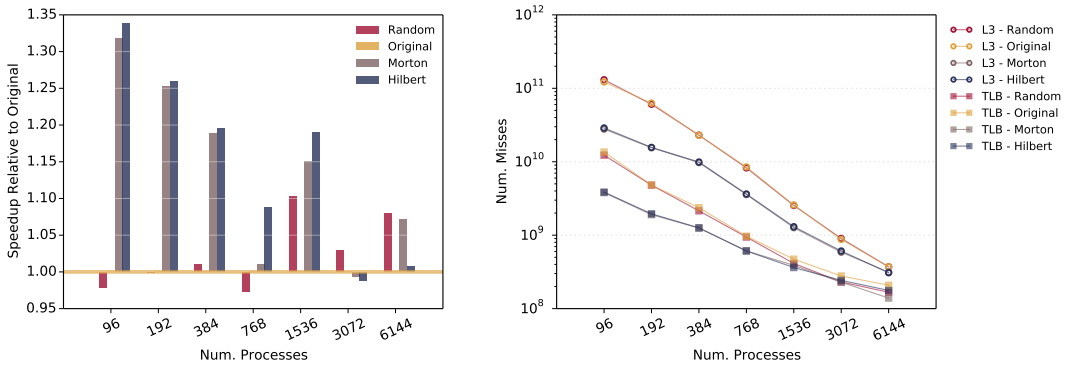

Figure 9: (Left) Speedup of the total application execution time, relative to the original (RCM) ordering, for the various mesh data orderings is shown. (Right) Total number misses for L1, L2, L3 caches and TLB during the application execution with the four mesh data orderings is shown. Optimized depth and halo-aware partitioning as obtained from previous section are used for all cases.

presented improvements.

The benefit of introducing data locality into the mesh by using the two SFCs are clearly observed with smaller number of partitions, and the performance gain gradually diminishes. With strong scaling, this is expected because as the partition sizes grow smaller, more of the mesh data can fit into the caches, which makes their ordering not as important. Nevertheless, the benefits are more profound on higher resolution meshes as they scale to even larger number of processors.

Further, to analyze how the cache behavior is affected by the reordering, we collect cache statistics using hardware counters. The total number of cache misses for L1, L2, L3 caches and TLB, during the execution of the applications are shown in Fig. 9. These show that ordering the data is definitely beneficial for improving cache performance. Apart from reducing the execution time with reduced cache misses, the minimization of data movement also leads to lower power consumption.

5 Conclusions

In this work we address two of the primary performance challenges, load balance and data locality, when faced with computations over unstructured meshes. We demonstrated that distributing a work load across processors with an unstructured mesh partitioning, it is necessary to consider the computation and communication costs due to any halos used in order to reduce overall communication and gain higher performance. We therefore devised a new hypergraph-based depth- and halo-aware partitioning strategy, which allowed significant improvement for MPAS-O at scaling. We additionally applied an SFC-based ordering to the unstructured mesh by mapping it to a 3D regular domain, showing that ordering is essential to reduce data movement when the data size is significantly larger than the system cache size.

Overall our optimizations attained up to 2.2× speedup compared with the original implementation, allowing additional MPAS short simulations that improve statistics via increased ensembles. Additionally, our improved scaling enables increased resolution and throughput of higher resolution meshes thus, allowing the optimized MPAS version to answer novel science question of importance to the climate modeling community.

Acknowledgments

The authors thank Aydin Buluc and Umit Catalyurek for discussions on graph partitioning. Authors from LBNL were supported by

DOE Office of ASCR under contract number DE-AC02-05CH11231. Authors from Univ. of Oregon were supported by DOE SciDAC grant DE-SC0006723. Douglas Jacobsen was supported by DOE Office of BER. We used resources at NERSC, which is supported by Office of Science of US DOE under Contract No. DE-AC02-05CH11231.

References

[1] M. Bader. *Space-Filling Curves*. An Introduction With Applications in Scientific Computing. Springer, Oct. 2012.

[2] M. A. Bender, B. C. Kuszmaul, S.-H. Teng, and K. Wang. Optimal Cache-Oblivious Mesh Layouts. *Theory of Computing Systems*, 48(2):269–296, Feb. 2011.

[3] M. Berzins. A new metric for dynamic load balancing. *Applied Mathematical Modelling*, 2000.

[4] A. Buluc and J. R. Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *International Journal of High Performance Computing Applications*, 2011.

[5] Ü. Catalyürek. *Hypergraph models for sparse matrix partitioning and reordering*. PhD thesis, 1999.

[6] Ü. Catalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *Parallel and Distributed Systems, IEEE Transactions on*, 10(7):673–693, 1999.

[7] U. Catalyurek and C. Aykanat. *PaToH: Partitioning Tool for Hypergraphs*, March 2011.

[8] E. Cuthill and J. McKee. Reducing the Bandwidth of Sparse Symmetric Matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, 1969.

[9] J. M. Dennis. Inverse space-filling curve partitioning of a global ocean model. In *International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–10. IEEE, 2007.

[10] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and Ü. V. Catalyürek. Parallel hypergraph partitioning for scientific computing. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10 pp. IEEE, 2006.

[11] O. Fortmeier, H. M. Bücker, B. O. F. Auer, and R. H. Bisseling. A new metric enabling an exact hypergraph model for the communication volume in distributed-memory parallel applications. *Parallel Computing*, 39(8):319–335, Aug. 2013.

[12] Y. F. Hu and R. J. Blake. Load balancing for unstructured mesh applications. *Parallel and Distributed Computing Practices*, 1999.

[13] G. Karypis. *METIS: A Software Package for Partitioning UnstructuredGraphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices*, Mar. 2013.

[14] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[15] I. Moulitsas and G. Karypis. Architecture Aware Partitioning Algorithms. In *ICA3PP '08: Proceedings of the 8th international conference on Algorithms and Architectures for Parallel Processing*. Springer-Verlag, June 2008.

[16] T. Ringler, M. Petersen, R. L. Higdon, D. Jacobsen, P. W. Jones, and M. Maltrud. A multi-resolution approach to global ocean modeling. *Ocean Modeling*, 69(C):211–232, Sept. 2013.

[17] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multi-constraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14(3):219–240, 2002.

[18] N. Selvakkumaran and G. Karypis. Multi-Objective Hypergraph Partitioning Algorithms for Cut and Maximum Subdomain Degree Minimization. *IEEE Transactions on Computer Aided Design*, pages 1–14, Apr. 2005.

[19] B. Uçar and C. Aykanat. Revisiting hypergraph models for sparse matrix partitioning. *SIAM review*, 49(4):595–603, 2007.

[20] H. T. Vo, C. T. Silva, L. F. Scheidegger, and V. Pascucci. Simple and Efficient Mesh Layout with Space-Filling Curves. *Journal of Graphics Tools*, 16(1):25–39, 2012.

[21] C. Walshaw, M. Cross, and M. G. Everett. Dynamic mesh partitioning: A unified optimisation and load-balancing algorithm. 1995.