High Performance Computing for Weather and Climate

Unstructured vs. structured grids

Group 10: Tat Chi Wong, Chiara Ghielmini, Miku Nakamura

Supervisor: Oliver Fuhrer

Submission date: 31st August 2024

1 Introduction

Climate and weather models require discretization of partial differential equations (PDEs) on a grid, of which the options include the structured and the unstructured grids. The most commonly used type of structured grid is the longitude-latitude grid, in which the Earth's surface is divided by a mesh of cells formed by lines of constant longitude and latitude drawn on a particular map projection. The advantage of this type of grid is that it is quite simple to generate; moreover, the discretization of PDEs can be done by spectral patterns or finite differences. The problem, however, is that in polar regions the points are not well delineated (Chukkapalli et al., 1999). Besides, convergence of meridians at the poles causes bottlenecks in parallel mass calculation. As a result, although most models currently used for forecast and research are based on structured grids, there is renewed interest in quasi-uniform alternatives, which is possible only through an unstructured approach (Staniforth and Thuburn, 2012).

In an unstructured grid, the points are not arranged on a regular matrix, and hence it is necessary to define a relationship between the individual points and their respective positions (Thompson and Weatherill, 2020). This relationship is often defined through a lookup table, which is not universally determined, but there may be several options. This makes the arrangement of grid points more flexible, but also more complex to handle with. With unstructured grids is it possible to avoid poleward crowding and resulting problems through the construction of isotropic gridpoints. Moreover, they are efficiently handled by spatial discretization to the finite elements. The disadvantage, however, is the computational overhead and the lack of flexibility in partitioning the computational domain (Chukkapalli et al., 1999).

According to Zängl et al. (2015), there is a trend in abandoning regular grids and spectral models as they impose limits on scalability and do not have a trivial way to obtain local mass conservation and mass-consistent tracer transport. According to the author, most of the newly developed models are based on icosahedral grids, centroidal Voronoi grids or cubed-sphere grids, but the possibilities are numerous, for example, Chukkapalli et al. (1999) has developed an unstructured grid based on a spherical spiral. An example of icosahedral grid, which means the grids are generated by bisecting the edges of the triangles on an icosahedral, is the Icosahedral Nonhydrostatic Weather and Climate Model (ICON) (Giorgetta et al., 2018). This model has also recently been employed by the Swiss Meteorological Service. One reason it is considered better than the previous model is that, because it is based on an unstructured grid, the cells do not differ in size from each other, allowing more reliable results (Meteoswiss, n.d.).

Since the unstructured grid works by indirectly addressing gridpoints through the lookup table, the way to construct the lookup table may affect the code performance due to data locality. One possible way to optimize data locality is the space-filling curves. A space-filling curve is a continuous line that links every grid in the multi-dimensional space into the 1-D space, never crossing paths (Mokbel et al., n.d.). There are different types of space-filling curves, but according to (Jagadish, 1990a), the Hilbert curve performs on average better than the others because the points that are close to each other in the 1-D space stay close also in the multi dimensional space. This allows to preserve data locality and thus improve cache performance.

The goal of this project is to use Fortran to implement an unstructured version of the stendil2d code through the integration of a lookup table and to compare its performance with the original structured grid. Secondly, we are also interested in finding other ways to improve the runtime of the unstructured version of the program. Therefore, a code that addresses the gridpoints by means of a space-filling curve and one that addresses them randomly will also be implemented. The performances of these different codes are analysed and an attempt is made to explain the different execution times and cache performance.

2 Methods

While in a structured grid, the neighbors can be accessed using simple array indexing, in an unstructured grid, a lookup table is necessary to know the irregular relations between the grid points. Our implementation of the unstructured approach only applies to the x and y dimensions, and the z dimension is kept structured. To evaluate the differences in processing time for various procedures of producing unstructured grids, we employed multiple methodologies to identify the most efficient approach.

2.1 Initial Condition

We generate the initial condition in a structured grid where the dimensions are powers of two, specifically of the form $2^n \times 2^n$ ensuring that each dataset is a perfect square. This requirement is due to the limitations of the Hilbert curve mapping method, described in the conversion techniques section (Section 2.4). In these data, the values are predominantly zeros, except for a distinct central square region filled with ones. This central region spans from the first quarter to the third quarter along both the x-axis and y-axis, creating a defined area of ones surrounded by zeros. The initial condition is then mapped to the unstructured grid. This setup provides a precise and symmetric pattern ideal for testing and simulations.

2.2 Fortran Implementation

Several new functions are implemented for unstructured grid based on `stencil2d-base.F90`. In order to adapt to the new functions, some parameters in the original script need to be modified, and some new parameters are added (Table 1).

In this project, we use three different methods to convert structured grid to unstructured grid, which are sequential mapping, random mapping, and Hilbert Curve mapping. We introduce each of these methods in detail in Section 2.4. For each method, we create a separate version of the stencil2d program, which are `stencil2d-sequential.F90`, `stencil2d-random.F90` and `stencil2d-hilbert.F90`. Here is the list of major subroutines that are essential for unstructured grid implementation.

- **`mapping`**: This is responsible for mapping the unstructured grid. This is the only subroutine that has different codes in all the versions
- **`inverse_mapping`**: This is responsible for converting the unstructured grid back to structured grid. The purpose is to output the results in structured grid for easier plotting and visualization.
- **`find_neighbors`**: This is responsible for constructing a lookup table for finding the neighboring indices of each grid cell and filling the neighbors parameter.

In the stencil computation, the necessary grid cells are retrieved with indirect addressing involving the variables `xPoint`, `yPoint` and `neighbors`. The halo update in the original script is removed as the periodic boundary condition is already incorporated in the `find_neighbors` subroutine in our implementation. Besides, in `stencil2d-hilbert.F90`, the code for the generation of the Hilbert curve is adopted from Justin (2023).

2.3 Lookup Table Construction

A lookup table, in the form of a 2D array `neighbors` (Table 1), is constructed to facilitate the conversion to unstructured grids. This table includes the i-coordinates, j-coordinates and cell values.

Parameter	Data type	Dimension	Description
`nPoint`	integer	1	The number of gridpoints in the unstructured grid. Essentially `nx` $\times$ `ny`.
`in_field`	real	(`nPoint`, `nz`)	A 2D array storing the initial condition in an unstructured grid. Only the x and y dimensions are employed with unstructured approach, and the z dimension keeps structured
`out_field`	real	(`nPoint`, `nz`)	A 2D array storing the results in an unstructured grid.
`in_field_structured` & `out_field_structured`	real	(`nx`, `ny`, `nz`)	The 3D array storing the initial condition and the output field respectively in a structured grid. This is to facilitate the generation of the initial condition and the plotting of the output data.
`xPoint`	integer	(`nPoint`)	The x-coordinate for each gridpoint in unstructured grid. Important for indirect addressing.
`yPoint`	integer	(`nPoint`)	Similar to `xPoint` but for y-coordinates.
`gridindex`	integer	(`nx`, `ny`)	Obtain the position of a gridpoint in the unstructured grid based on (x, y) in the structured grid.
`neighbors`	integer	(4, `nPoint`)	A 2D array that stores the positions of the 4 neighboring cells (up,down,left,right) of each grid point. We will examine whether the code will be faster if we rearrange the dimension to (`nPoint`, 4), with further explanation in Section 2.3

Table 1: List of parameters that are modified from the original script or newly added to adapt to the functions implemented for unstructured grid

2.3.1 Indexing Comparison

We exchange the index order of the `neighbors` parameter, and experiment which index order is faster. The experiment is done for all conversion techniques described below and domain size is set to 128x128.

Our first hypothesis is that using the parameter `neighbors(4,nPoint)` in the script stencil2d-sequential.F90 makes the code faster than `neighbors(nPoint,4)`. In fact, Fortran saves data in column order, which means that elements in the same column are stored next to each other in the memory. This implies that by applying the `neighbors(4,nPoint)`, the access to the neighbour grid would be easier because they are stored sequentially and it is hence cache-friendly. On the other hand, by applying the `neighbors(nPoint,4)` to access the neighbouring grid, it is necessary to jump across the memory and the efficiency is then reduced. The second hypothesis is that the result would be reversed using the Hilbert curve as space filling curve. This may be due to the fact that the Hilbert curve ensures that elements close to each other in the plane are still adjacent even in the 1D sequence. With this configuration, using `neighbors(nPoint,4)` allows to access the data in a more localized way and in consequence to improve the cache performance. Finally, we don't expect big differences in the random case since the look up table is accessed in a random way.

2.4 Conversion Techniques

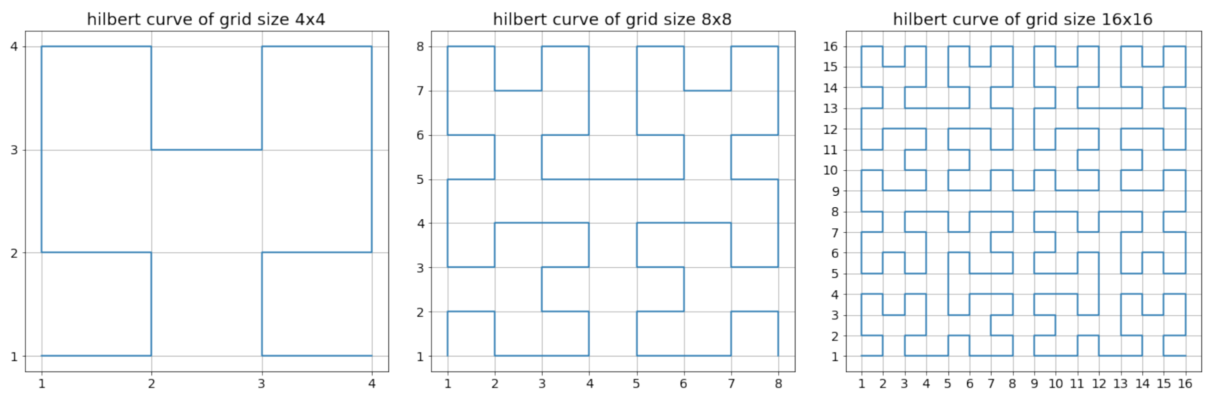
We implement three distinct techniques: random, sequential, and Hilbert curve to convert from structured into unstructured grid. The runtime for each technique is measured and recorded for comparison.

- **Random Mapping:** The grid cells are mapped to unstructured grids randomly by shuffling the indices using Fisher–Yates shuffle (Eberl, 2016). This approach does not follow any specific pattern or sequence, leading to potentially varied performance based on the randomness of the order.

- **Sequential Mapping:** The grid cells are converted in a straightforward sequential order. Two settings are used for this method: row-by-row and column-by-column. In the row-by-row setting, the outer loop iterates over the rows (`j` loop), and the inner loop iterates over the columns (`i` loop). Conversely, in the column-by-column setting, the outer loop iterates over the columns (`i` loop), and the inner loop iterates over the rows (`j` loop). Both settings are compared in a 128×128 dataset to evaluate their performance and efficiency. This systematic approach ensures a predictable and orderly conversion process.

- **Hilbert Curve Mapping:** The position of the grid cells in an unstructured grid are computed to follow the Hilbert curve. According to (Abel and Mark, 1990) and (Jagadish, 1990b), the Hilbert curve is the best space filling curve that achieves the best clustering and hence we decided to apply it to our code. The Hilbert Curve is a continuous fractal space-filling curve that preserves spatial locality, meaning that cells that are close in the structured grid remain close in the unstructured grid. It is known to enhance performance by improving data locality and cache usage (Araújo et al., 2021). The Hilbert Curve is visualised for 4x4, 8x8, and 16x16 domain sizes (Fig. 1).

To determine which method is the fastest, we run the program of the three versions (`sequential`, `random` and `hilbert`) in domain sizes increasing from 16x16 to 1024x1024. The domain size is consistently increasing with a power of 2 (2^n) and the domain is always a square matrix because these are the geometrical requirement for the Hilbert curve. The run time for each domain size and each method is compared and analyzed.


hilbert curve of grid size 4x4
hilbert curve of grid size 8x8
hilbert curve of grid size 16x16

(a) Domain size: 4x4 (b) Domain size: 8x8 (c) Domain size: 16x16

Figure 1: Visualization of the Hilbert curve of domain sizes of (a) 4x4, (b) 8x8, and (c) 16x16. The curve starts from the bottom left grid

Our hypothesis is that the random mapping is the slowest and the Hilbert curve mapping is the fastest. This is because in random mapping, all the indices are shuffled and the neighboring grid cells do not necessarily have neighboring memory locations. This reduces data locality and thus the code performance. On the other hand, Hilbert curve optimizes the locality of data reference. Access to cache would be more efficient and hence the runtime would be faster.

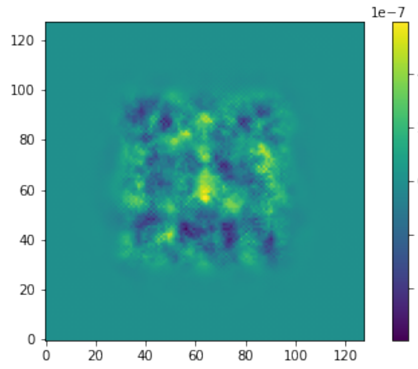
3 Results and discussions

3.1 Result Validation

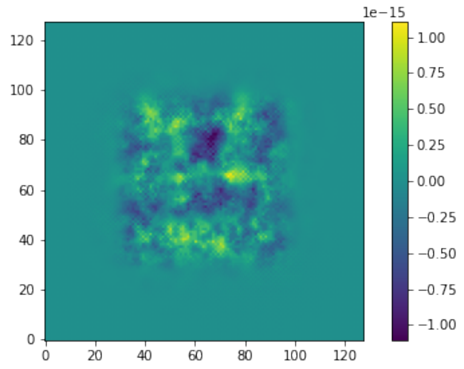
After implementing the code described in Section 2.2, we first validate the results by running the program with a domain size of 128x128x64 by 1024 iterations. The initial condition is a cube in the center of the domain initialized with ones, and all the surrounding grid cells are initialized with zeros. The results are validated in the following ways:

1. We first plot out both the initial condition and the final result, to check if the patterns are correct.
2. To ensure the results in the unstructured grid are the same as that in the structured grid, we subtract the results of the base version of the stencil2d program by the unstructured versions, and check if the results are the same.
3. To ensure the results computed in all the three versions of the unstructured grid stencil2d programs are the same, we subtract the outputs and check if any differences exist.

We confirm that the outputs from all three mapping methods are correct and that they are exactly the same. However, we observe a very subtle difference in magnitude of -7 between the results of the structured grid and the unstructured grid (Fig. 2a). Since the output data uses a data type of 4-byte floating point, the precision has only around 7 decimal digits. This suggests that the difference arises from bitwise precision error. Indeed, when we switch the data type to 8-byte floating point number and compare the outputs, the difference decreases to a magnitude of -15 (Fig. 2b). Unfortunately, we are not sure why this occurs. However, this bitwise difference in the final result is negligible. This means that our implementation of the unstructured grid version of the stencil program is valid.



(a) 4-byte precision



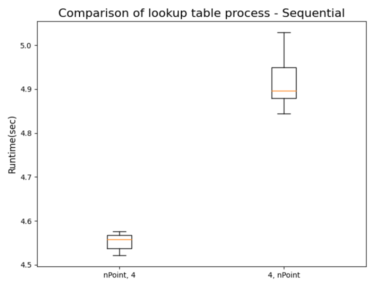
(b) 8-byte precision

Figure 2: Color contour shows the difference between the output of `stencil2d-base.F90` and `stencil2d-sequential.F90`, with the output data having (a) 4-byte and (b) 8-byte precision.

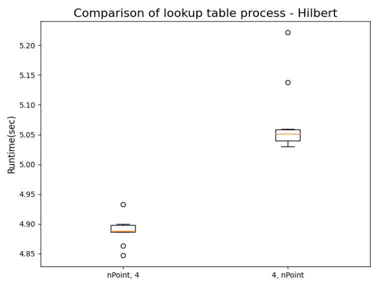
3.2 Lookup Table Comparison

The results of the experiment shown in Figure 3 compare the runtimes of `neighbors(4,nPoi nt)` function against `neighbors(nPoint,4)` for each conversion technique: sequential, Hilbert, and random. They are all applied to a 128x128x64 initial condition. Each method was executed ten times to ensure reliability and consistency in the results. Contrary to our hypothesis, the box plot illustrates that `neighbors(nPoint,4)` generally exhibits shorter runtimes compared to `neighbors (4,nPoint)`. The median runtime for the row-by-row method is visibly lower, suggesting a performance advantage in this configuration.

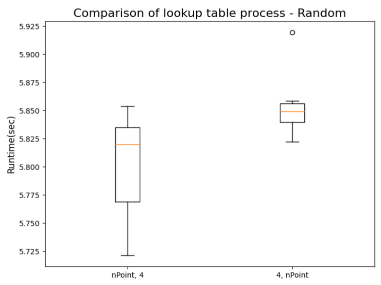
Due to the small sample size, the Mann-Whitney U test was employed to assess the statistical significance of the observed differences in runtimes. This non-parametric test is appropriate for comparing two independent samples without assuming a normal distribution. According to the Mann-Whitney test, the sequential method yielded a p-value of 0.0002, the Hilbert method also had a p-value of 0.0002, and the random method produced a p-value of 0.01. All these values satisfy the significance level of 0.05, indicating that the observed differences are statistically significant. Therefore, we conclude that using `neighbors(nPoint,4)` yields better performance.



(a) Sequential



(b) Hilbert



(c) Random

Figure 3: Visualization of the lookup table procedure, `nPoint,4` and `4, nPoint` of (a) Sequential, (b) Hilbert, and (c) Random conversion techniques.Each method is executed ten times, and the runtimes were measured in seconds.

3.3 i and j loop in the sequential method

The results of the experiment shown in Figure 4 compare the runtimes of the row-by-row and column-by-column sequential mapping methods applied to a 128x128x64 initial condition. Each method is executed 10 times to ensure reliability and consistency. The box plot illustrates that the row-by-row method generally exhibits shorter runtimes compared to the column-by-column method. The median runtime for the row-by-row method is visibly lower, suggesting a performance advantage in this configuration.

Similar to the experiment in Section 3.2, we employed the Mann-Whitney U test to assess the statistical significance of the observed differences in runtimes. The test yielded a p-value of 0.0889, indicating a marginal statistical difference between the two methods. These results highlight that while there is a tendency towards better performance with the row-by-row approach, it is impossible to conclude that the row-by-row is more efficient since the difference does not satisfy the significance level of 0.05. However, for experimental purposes, we employed the row-by-row setting for sequential mapping to evaluate conversion techniques and compare domain sizes.

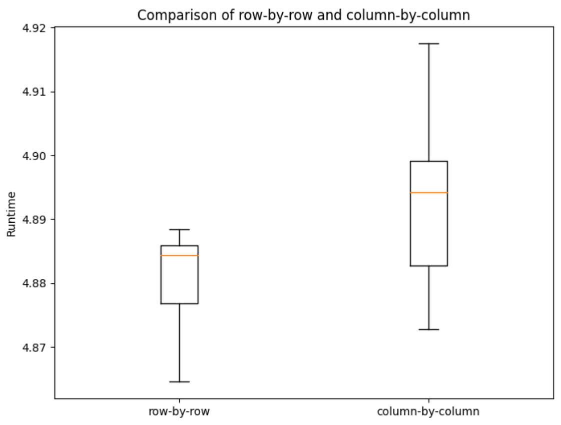


Figure 4: Box plot comparison of runtime for row-by-row and column-by-column sequential mapping methods in a 128×128 dataset. Each method is executed ten times, and the runtimes were measured in seconds.

3.4 Comparison of conversion techniques across domain sizes

We compare the runtimes of the structured grid and the unstructured grid of the three mapping methods across different domain sizes (Fig. 5). For the structured grid, we use the base version of the stencil-2d program. The domain size in x and y is set to increase exponentially from 16x16 to 1024x1024, with the z-dimension fixed at 64. In general, a larger domain size means a longer runtime per gridpoint for both the structured and unstructured grids. This occurs because, with larger domain sizes, the fastest cache memory cannot hold as much of the domain's data as possible. Access to cache of other levels or even the main memory may be required for computations involving gridpoints that are not in neighboring memory locations. Since the program runtime is mainly constrained by the memory access, the lower bandwidth of the slower cache and the main memory increases the runtime.

Comparing the performance of the structured and the unstructured grid versions of the program, the structured grid in general has a faster runtime than the unstructured grid across all domain sizes. This is probably because indirect addressing in an unstructured grid requires two layers of memory access to retrieve the necessary data. The program has to first search for the indices for the neighboring gridpoints, and then access the data based on those indices. In contrast, direct addressing in a structured grid only requires one layer of memory access to retrieve the data.

Among the different mapping methods of the unstructured grid, the runtime for domain sizes of 16x16, 32x32 and 64x64 are similar, suggesting that access to the fastest cache is possible at these domain sizes. For larger domain sizes, random mapping has the slowest runtime among all mapping methods. This is expected because the allocation of the gridpoints in the array is shuffled, and the neighboring gridpoints are not guaranteed to be located in adjacent memory locations. Comparing sequential mapping and Hilbert curve mapping, the Hilbert curve has a slightly slower runtime. This does not match our hypothesis that Hilbert curve would be faster than sequential mapping. To explain this phenomenon, we will investigate the cache statistics for these two methods in the following section.

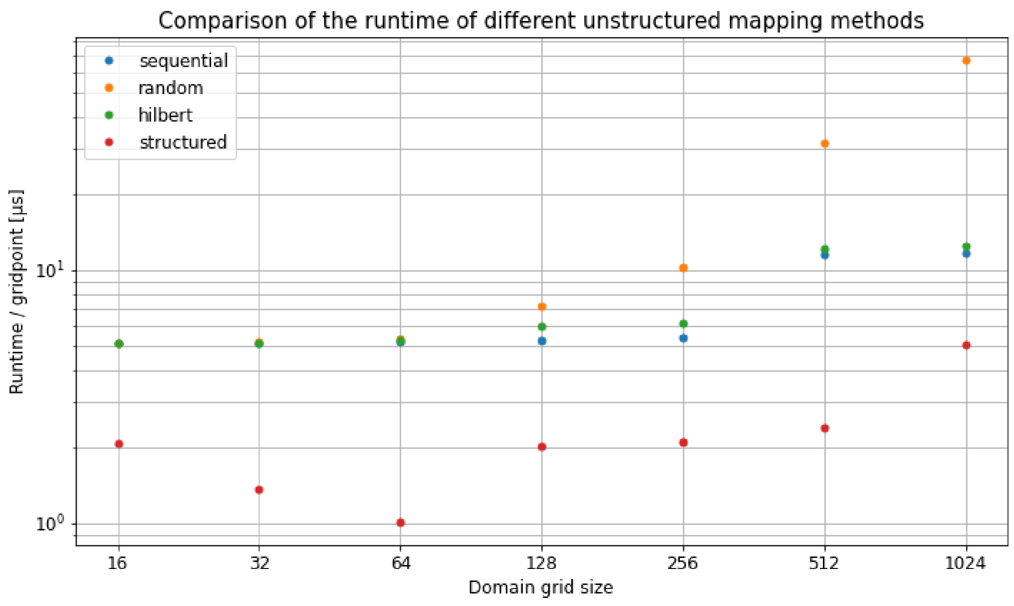


Figure 5: Scatter plot showing the runtime per gridpoint (μs) for both, structured grid and unstructured grid (sequential, random, and Hilbert curve mapping) as a function of the working set size (MB).

3.4.1 Cache hit statistics for sequential mapping and Hilbert curve mapping

The sequential mapping method has faster runtime than the Hilbert curve mapping method for domain sizes larger than or equal to 128x128 (Fig. 5). The number of cache hits determines how efficient data transfer is and consequently the overall runtime of the program. Therefore, we examine the cache hit rates of these two mapping methods at a domain size of 128x128. From the results (Table 2), the sequential method has a higher hit ratios and also greater utilization for both D1 and D2 caches than the Hilbert curve method. For the D1+D2 cache hit ratio, the sequential method obtained a 99.9% of hits, which is 0.1% higher than that of the Hilbert curve method. The sequential method also has a higher D1+D2 cache utilization (1,631.75 refs/miss) than the Hilbert curve method (466.82 refs/miss). At the same time, the sequential method has a lower D2 to D1 bandwidth than the Hilbert curve method, probably because the sequential method has a lower reliance on L2 cache, reducing the frequency of data transfers from the L2 cache. Overall, the faster runtime of the sequential method can be explained by the better utilization of cache, especially the fastest L1 cache.

4 Conclusion

In this project, we implemented different unstructured grids versions of the stencil2d program. The results highlighted several key insights into the performance of structured grids versus unstructured

	Sequential	**Hilbert curve**
D1 cache hit,miss ratios	96.0% hits 4.0% misses	95.8% hits 4.2% misses
D1 cache utilization (misses)	25.28 refs/miss 3.16 avg hits	23.90 refs/miss 2.99 avg hits
D2 cache hit,miss ratio	98.5% hits 1.5% misses	94.9% hits 5.1% misses
D1+D2 cache hit,miss ratio	99.9% hits 0.1% misses	99.8% 0.2% misses
D1+D2 cache utilization	1,631.75 refs/miss 203.97 avg hits	466.82 refs/miss 58.35 avg hits
D2 to D1 bandwidth	0.345 GiB/sec 2,037,710,955 bytes	1.187GiB/sec 8,050,458,736 bytes

Table 2: Cache hit statistics of sequential mapping and Hilbert curve mapping in the domain size of 128x128

grids and the impact of different mapping methods.

Our analysis showed that structured grids generally outperform unstructured grids in terms of runtime across various domain sizes. This was primarily due to the simplicity of direct addressing in structured grids, which allows for more efficient memory access than the indirect addressing required by unstructured grids. As the domain size increased, the runtime per grid point also increased due to the limitations in cache memory, which cannot accommodate the entire dataset.

Among the unstructured grid mapping techniques, the random mapping method consistently demonstrated the slowest runtime, especially as the domain size increased. This was likely due to the lack of spatial locality in data access, resulting in inefficient cache utilization. In contrast, both sequential and Hilbert curve mappings showed better performance, with sequential mapping slightly outperforming Hilbert curve mapping for larger domain sizes. In addition, we tested the differences in runtimes by applying the lookup table in different ways, line by line or column by column. As expected, no significant differences could be observed in the case of the random code. However, in the remaining two cases, the index order in the lookup table may yield better different performances.

Analyzing cache statistics revealed that sequential mapping achieves higher cache hit rates and utilization compared to Hilbert curve mapping. This finding contradicts our initial hypothesis that Hilbert curve mapping would be more efficient due to its spatial locality properties. The higher D1 and D2 cache hit ratios in sequential mapping suggested that this method was better optimized for data locality, leading to faster runtime performance.

Our experiments had two main limitations: time constraints and inflexibility in grid types. We ran our code 10 times to compare the runtime results; however, more iterations would have been desirable to obtain more significant results if more time had been available. Additionally, our choice of conversion technique, the Hilbert curve, imposed a crucial constraint on the grid size, as it requires a square grid. Although our experiments provided an insight in how different methods of constructing unstructured grids may change the code performance, there are still many possibilities to explore.

Further investigation is warranted to explore the reasons behind the unexpected performance differences between the Hilbert curve and sequential mapping methods. Additionally, exploring alternative space-filling curves or hybrid approaches could potentially enhance the efficiency of unstructured grids. A deeper understanding of cache performance and optimization techniques may also yield significant improvements in runtime efficiency for both structured and unstructured grids.

In summary, while structured grids currently offer better performance in terms of runtime, unstructured grids present promising flexibility and scalability opportunities. Continued research and development in unstructured grid mapping techniques are essential for advancing the capabilities of climate and weather modelling in high-performance computing environments.

References

Abel, D. J. and Mark, D. M. (1990). A comparative analysis of some two-dimensional orderings, *International Journal of Geographical Information Systems* **4**(1): 21–31.

Araújo, R. R., Gross, L. and de Souza, S. X. (2021). Boosting memory access locality of the Spectral Element Method with Hilbert space-filling curves, *Computers & Geosciences* **157**: 104938. doi: `10.1016/j.cageo.2021.104938`.

Chukkapalli, G., Karpik, S. R. and Ethier, C. E. (1999). A Scheme for Generating Unstructured Grids on Spheres with Application to Parallel Computation, *Journal of Computational Physics* **149**(1): 114–127. doi: `10.1006/jcph.1998.6146`.

Eberl, M. (2016). Fisher–yates shuffle, *Archive of Formal Proofs* . `https://isa-afp.org/entries/Fisher_Yates.html`, Formal proof development.

Giorgetta, M. A., Brokopf, R., Crueger, T., Esch, M., Fiedler, S., Helmert, J., Hohenegger, C., Kornblueh, L., Köhler, M., Manzini, E., Mauritsen, T., Nam, C., Raddatz, T., Rast, S., Reinert, D., Sakradzija, M., Schmidt, H., Schneck, R., Schnur, R., Silvers, L., Wan, H., Zängl, G. and Stevens, B. (2018). ICON-A, the Atmosphere Component of the ICON Earth System Model: I. Model Description, *Journal of Advances in Modeling Earth Systems* **10**(7): 1613–1637. doi: `10.1029/2017MS001242`.

Jagadish, H. V. (1990a). Linear clustering of objects with multiple attributes, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’90, Association for Computing Machinery, New York, NY, USA, p. 332–342. doi: `10.1145/93597.98742`.

Jagadish, H. V. (1990b). Linear clustering of objects with multiple attributes, *SIGMOD Rec.* **19**(2): 332–342. doi: `10.1145/93605.98742`.

Justin, S. M. (2023). spacefill [source code]. `https://github.com/program--/spacefill/tree/main` [Accessed: 10 July 2024].

Meteoswiss (n.d.). ICON-22 - MeteoSwiss. `https://www.meteoswiss.admin.ch/about-us/research-and-cooperation/projects/2023/icon-22.html` [Accessed: 01 July 2024].

Mokbel, M. F., Aref, W. G. and Kamel, I. (n.d.). Analysis of Multi-Dimensional Space-Filling Curves, **7**(3): 179–209. doi: `10.1023/A:1025196714293`.

Staniforth, A. and Thuburn, J. (2012). Horizontal grids for global weather and climate prediction models: a review, *Quarterly Journal of the Royal Meteorological Society* **138**(662): 1–26. doi: `10.1002/qj.958`.

Thompson, J. F. and Weatherill, N. P. (2020). Structured and unstructured grid generation, *High-Performance Computing in Biomedical Research* pp. 63–111.

Zängl, G., Reinert, D., Rípodas, P. and Baldauf, M. (2015). The ICON (ICOsahedral non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core, *Quarterly Journal of the Royal Meteorological Society* **141**(687): 563–579. doi: `10.1002/qj.2378`.

A Code Availability

This project is available on github. Please read `README.md` for further descriptions of the scripts and the steps of how the results are generated.