A Parallel Chain Mail Approach for Scalable Spatial Data Interpolation

Albert Asratyan
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
asratyan@kth.se

Sina Sheikholeslami
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
sinash@kth.se

Vladimir Vlassov
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
vladv@kth.se

***Abstract*—Deteriorating air quality is a growing concern that has been linked to many health-related issues. Its monitoring is a good first step to understanding the problem. However, it is not always possible to collect air quality data from every location. Various data interpolation techniques are used to assist with populating sparse maps with more context, but many of these algorithms are computationally expensive. This work introduces a three-step Chain Mail algorithm that uses kriging (without any modifications to the base algorithm) and achieves up to $\times$100 execution time improvement with minimal accuracy loss (relative RMSE of 3%) by running concurrent interpolation executions. This approach can be described as a multiple-step parallel interpolation algorithm that includes specific regional border data manipulation for achieving greater accuracy. It does so by interpolating geographically defined data chunks in parallel and sharing the results with their neighboring nodes to provide context and compensate for lack of knowledge of the surrounding areas. Combined with a serverless cloud architecture, this approach opens doors to interpolating large data sets in a matter of minutes while remaining cost-efficient. The effectiveness of the three-step Chain Mail approach depends on the equal point distribution among all nodes and the resolution of the parallel configuration. In general, it offers a good balance between execution speed and accuracy.**

***Index Terms*—Distributed Computing, Parallel Execution, Data Interpolation, Kriging, Geostatistics, Air Quality**

I. Introduction

In recent years, the Internet of Things (IoT) has gained lots of traction, and there are no signs of its growth slowing down. One of the main IoT strengths is ubiquitous data gathering possibilities. Some of the standard civic applications of IoT data collection could include traffic monitoring or water management [1]. A particularly interesting health-related area to look at is air quality, considering the ever-growing global industrialization and production volumes affecting the environment around us. Even though recently there has been made some progress with reducing exposure to unhealthy air in more developed areas, global pollution is still rising, sometimes reaching 99% of the population living in areas with dangerous levels, primarily in the Southern, Eastern, and South-Eastern regions of Asia [2]. The first step in tackling this problem is understanding its scale and raising awareness about the health risks associated with unhealthy air.

One of the most important and used variables that characterizes air quality is particulate matter (PM) that consists of liquid droplets and solid particles [3]. The smaller the particle, the higher the health risk it introduces. Some of them could get into the bloodstream, and, subsequently, into the lungs, increasing probabilities of nonfatal heart attacks, aggravated asthma, or decreased lung functionality [4]. From the environmental perspective, high PM values have been linked to making streams and lakes more acidic, changing nutrient balance in coastal waters and large river basins, depleting soil nutrients, and damaging crops [4].

Collected air quality data can and will be sparse in some geographic regions. This problem can be addressed by interpolating the existing values to the neighboring unknown areas. However, being a computationally expensive task, execution times of spatial interpolation are important to consider. For systems handling large data sets, parallelization should be considered as a means of improving performance. Otherwise, computations may take hours because execution time will grow exponentially with linear growth in the number of points.

This research aims to improve real-world spatial interpolation performance by introducing a parallel approach to interpolation. The algorithms in use will be the same as in the conventional single process interpolation cases. However, parallel execution has its own issues. For example, individual threads will not know the context of their neighbors, which will result in regional edges having colliding values. Consider Figure 1. The four regions have been interpolated in parallel, but since each thread does not know about the existence of its neighbors, there can be a great value difference at the neighboring edge coordinates. It would be desirable to have a simple solution to this edge value collision problem without any modifications to the original interpolation algorithms by focusing on the distributed configuration instead.

Contributions. We present the design and development of a Chain Mail parallel interpolation algorithm that greatly improves execution speeds of standard data interpolation algorithms (specifically, kriging) without any modifications to the data interpolation itself and allows to use multiple threads for a more efficient resource utilization. This work provides a perspective on what direction the development of distributed data interpolation algorithms could take.

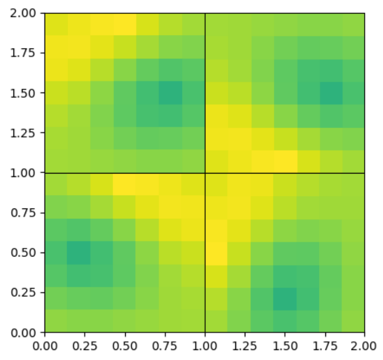


Fig. 1. Parallel interpolation edge collision example

II. Spatial Interpolation and Kriging

Spatial interpolation is the process of using known points to estimate values at unknown locations. Extensive data collection can be an expensive task, if at all possible. Because of this, it makes sense to collect data only from the most significant regions. Then it is up to the interpolation algorithms to fill the gaps. There are multiple existing algorithms and some of the most popular are proximity interpolation, Inverse Distance Weighted (IDW) interpolation, and **Kriging** interpolation [5].

In geostatistics, Kriging is a well-known algorithm for spatial interpolation, first published in 1951 and named after its inventor, Danie G. Krige. The technique is also known as Gaussian process regression. As the name suggests, the method is modeled by a Gaussian process, which is governed by prior covariances. Given a set of known points s and values at their locations $Z(s)$, the estimation at an unknown location $Z^*(s_0)$ is a weighted mean that would be equal to:

$$Z^*(s_0) = \sum_{i=1}^{N} \lambda_i Z(s_i) \tag{1}$$

where N is the size of s, and λ is the array of weights.

Kriging is different from other interpolation methods because it uses spatial correlation between known points to interpolate the values in the spatial field. Kriging is also capable of generating an estimate of uncertainties around every interpolated value [6]. It is important to highlight that Kriging will be less effective if there are no spatial correlations between points [6]. In this work, we focus on *Universal Kriging*. The difference between Universal Kriging and Ordinary Kriging is that the universal one relaxes the stationarity requirement by allowing the mean of the values to differ in a deterministic way for different locations. This type is more suitable for environmental applications [6]. It is also worth noting that kriging time complexity is rather high, being $O(N^4)$ for N points of input data [7].

III. Related Work

Not much publicly available relevant work has been done in the domain of parallelization aspect of data interpolation. However, there has been put a lot of effort into improving interpolation techniques by focusing on different approaches, mainly algorithm optimizations. A study performed by Armstrong and Marciano investigated parallel data interpolation on mainframe-grade computers in late 90s [8]. They have used the IDW algorithm on multiple processors, and only the execution time was measured. The study has shown that there was almost a linear decrease in execution time with a linear increase in the number of processors. However, the performance gains would slow down with more added hardware. The study has also not addressed the accuracy for any of the tests.

A study on the efficiency of kriging for real-time spatio-temporal interpolation was done by Srinivasan et al. [7], who have attempted to reduce the kriging time complexity to $O(N^3)$. The study has used atmospheric data, and the authors state that substantial performance increases were achieved, with some test execution times improving from 2 days to under 7 minutes. The kriging algorithm has been changed by introducing iterative solvers like conjugate gradient, and then GPUs were used for further performance improvement.

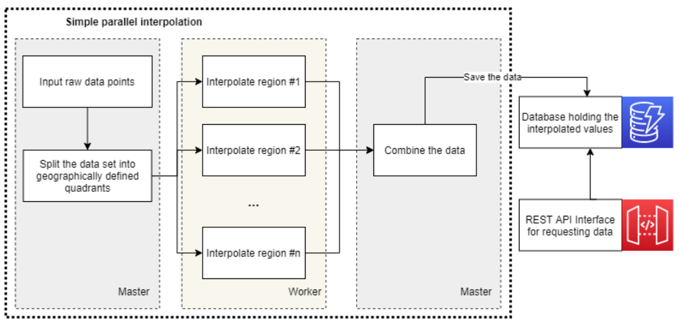
Henneböhl et al. [9] have compared CPU and GPU performance differences for the IDW algorithm. The study has found that, for small known data to prediction location size ratios, GPU implementation outperforms the CPU one by a factor of 2. However, with the data/location size ratios going up, GPU execution times start growing exponentially, falling behind CPU times by a lot. The study concludes by suggesting that the ideal solution should be a hybrid CPU/GPU system that is able to combine the strengths of both of the approaches.

Some of these works focus on algorithm optimizations, whereas other resort to hardware solutions (GPU). But as Henneböhl et al. have mentioned, the answer lies somewhere in the middle where all of these techniques are combined. This work will put focus on improving execution times by the means of parallelization with consideration of interpolated edges together with global accuracy.

IV. Method

We consider two approaches to running the Kriging algorithm in parallel, namely (1) the naive approach of splitting the data to geographical blocks of equal size and executing the interpolations in parallel; (2) a Chain Mail approach that improves the accuracy of the former method.

A. Naive Parallel Approach



Simple parallel interpolation
Input raw data points
Split the data set into geographically defined quadrants
Master
Interpolate region #1
Interpolate region #2
...
Interpolate region #n
Worker
Combine the data
Master
Save the data
Database holding the interpolated values
REST API Interface for requesting data

Fig. 2. Naive parallel kriging execution system diagram

A system diagram for a naive parallel kriging interpolation is show in Figure 2. To compare the performance of this approach with single process Kriging interpolation, consider the following example. A set of 1000 points is split to a 2×2 grid of 4 equal-sized regions. Assuming equal point distribution, each region will have 250 points. Thus, each region will take $250^4 \approx 4{\times}10^9$ units of time to compute.

Compared to the single process kriging interpolation (with a time complexity of $O(N^4)$ for N points of input data), this approach yields the following factor of theoretical execution time performance improvement:

$$1000^4/(4 \times 10^9) = (10^{12}/4 \times 10^9) = 250$$

However, as we will see in the Results section, the naive parallel approach is 250 times faster for the data set size of 1000 points only in theory. The real-world execution performance gap is smaller but still indicative of the advantage of the parallel approach. However, the biggest shortcoming is the fact that individual regions do not have any knowledge about their neighbors. Each parallel worker process would generate its own interpolated surface for its assigned region. Each of those workers would rely only on the data assigned to its grid region, which means that the values around the regional edges would be drastically different, when compared to what a single interpolation algorithm would produce.

B. Chain Mail Approach

Consider an example surface that is the result of the 2×2 naive parallel interpolation (Figure 3) from a synthetically generated surface with 10000 points in which 200 points of them were given as input for interpolation, and focus on the top two quadrants with their shared border that lies on the line between points $(0, 0)$ and $(0, 2)$. In basic terms, the main issue is the lack of smooth transition from one region to the other. In data interpolation context, the issue is that the corner areas do not have any reference data to consider that is located outside of the initially assigned grid cell coordinates, since the area is finite and each worker can only cover so much area before the benefits of parallel execution disappear. As a result, each of the workers interpolates only with regards to the data assigned to it. This issue also breaks the isotropy requirement for Kriging. Imagine the edge values for each of the regions are known; then the problem would be simplified to running the interpolation algorithm in parallel on each region. Therefore, the main task can be changed to acquiring the edge values.

Our solution is a *Chain Mail* approach to improve the accuracy of brute force parallel interpolation while also retaining the benefits of faster execution times. The name originates from chain mail – a type of medieval armor made from small metal rings linked together in a pattern that forms a mesh.

The core idea adopted from chain mail armor is the reliance of each ring only on its immediate neighbors to support it. Similarly, each grid cell would be able to know about four of its neighbors and their data. In turn, each of those neighbors would be able to pick up context from four of their neighbors, and so on. As a result, each cell would be able

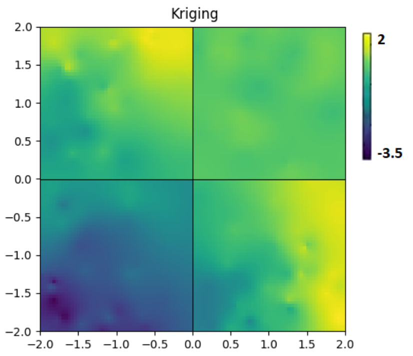


Fig. 3. A synthetic surface generated by the naive parallel approach

to consider data from its immediate neighbors and provide a ”smoother” estimation, resembling the results of a single process interpolation more. Repeating the process multiple times allows getting the edge data context for each of the workers that was unknown before.

Quick Overview: consider Figure 4. Let the combined gray area designate the initial area for the single process interpolation approach. Same as before, it can be split into multiple (for this example – four) regions and interpolated in parallel. However, only a small subset of the results form this step is important. Specifically, it is the data from the overlapping edges that is required for further processing.

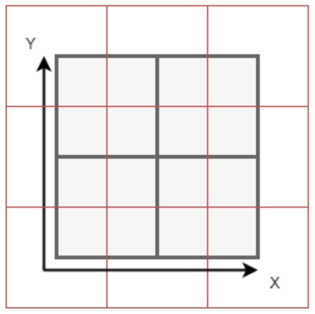


Fig. 4. Chain mail interpolation regions

Once the edge data is collected, the second step begins where new regions are formed, represented by the red quadrants in Figure 4. Each red region is essentially a spatial translation of a corresponding gray region by half of its width and height in the x and y directions, respectively. The edge data collected from the first step (gray lines in the figure) can be then used for the second round of interpolation to give more context about what those borders should look like.

Step-by-Step Description: consider Figure 5a for a 2×2 interpolation, in which each small black dot represents a data point from the raw data set, and each quadrant represents an area that is going to be interpolated by each parallel worker. After the first round of interpolations, each parallel worker will generate a surface for its respective region. Only edge data points should be collected to provide about neighboring regions. Moreover, not all of the edges should be added, but only a small subset at a specific step value. This ensures that the subsequent interpolation steps' performance will not suffer from greater data set sizes. The newly generated edges are displayed as red dots in Figure 5a.

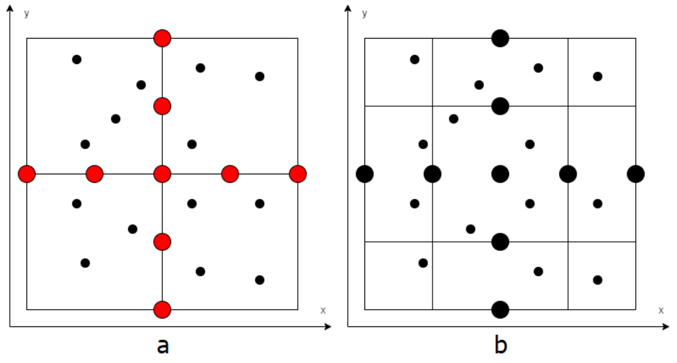


Fig. 5. Chain Mail interpolation: (a) from initial state (black) to generating edges (red); (b) resolving overlaps and forming new regions.

It can be noticed that some regions will have overlapping interpolated edges with potentially different values. All of these conflicting points should be resolved in some way. There are multiple approaches to merging the overlapping edge data. The simplest solution would be to calculate an average of the values of the two points with the same set of coordinates. Other approaches could include using weighted or inversely weighted averages, where the weight is calculated based on the number of points in each of the neighboring regions.

Once the conflicting data points have been merged, they can be added to the initial data set and treated as an equal part of the data. In the meantime, new geographical regions should be generated in preparation for the second round of interpolation. New regional boundaries are formed by translating the old boundaries by half of the width/height value of the old boundary in both x/y directions, thus resembling a chain mail. Figure 5b demonstrates the new boundaries together with the merged edge data displayed as big black dots. After the second round of interpolation, there are two alternatives:

1) Retrieve the edge data and continue with the third round by repeating the merging and appending processes. This results in a three-step Chain Mail interpolation.
2) Retrieve the generated surfaces as the final result. This results in a two-step Chain Mail interpolation. Only half of the required edge data will be known, so this produces less accurate results, albeit it is faster.

Assuming that the first option is chosen after the second round, some of the regional borders will have overlapping edge data points again. However, they will overlap not just with the other new edge points but also with the data points appended to the initial data set at the end of the first interpolation step. These points will have to be merged in the same fashion.

After resolving the conflicting points once again, they can be appended to the initial data set. The next step is to create new geographical boundaries. This time, the boundaries are the same as in the first step, but now each of the regions has some context about the data on its edges and is able to interpolate the regions with some consideration of neighboring values. The surfaces generated from the third round of parallel interpolations can be considered to be the final result. The following section will present a system diagram for the three-step chain mail interpolation algorithm.

Time Complexity Estimation: compared to the two-step architecture, the three-step just repeats the first step, thus increasing execution time by one third. Thus, for the example of 1000 points, the total time spent would be approximately 12×10^9 units of time (4×10^9 for each of the interpolation steps). With this number in mind, the theoretical performance is still expected to improve by a factor of ≈ 80.

Theoretical Correctness: the major advantage of the three-step interpolation (compared to the two-step one) is that it calculates all of the border data at the end of the second step. When it is fed into the third iteration, all of the edge values are known, so the final interpolation step can be completed without the edge collision issue present in both the naive parallel approach and the two-step Chain Mail approach.

V. Implementation

Here we present implementation of the three-step Chain Mail interpolation that follows the same architecture as the two-step one plus an extra interpolation step. Three-step chain mail interpolation system diagram is shown in Figure 6.

A. *Splitting Data*

The bottom left and top right points are the key to splitting the data into regions of the same size. Grid dimensions can be defined by those two points, and the algorithm could split it into equally sized grid cells. The number of grid cells is restricted by (and can not exceed) the maximum amount of parallel processes available for the second step of the algorithm, as it requires the most computing power.Splitting the data is handled by the master node. First, the raw data should be sorted by the x coordinate value and then split into vertical strips along the x axis. Then, for every vertical strip, each data point should be sorted by the y coordinate value and then split into the same amount of horizontal strips along the y axis. As a result, the processed strips will not have to be squared in absolute shape with regards to the initial area boundaries but rather split the initial area into an N×N grid where N is the provided step value for a grid cell of a constant size dynamically calculated from the area boundaries (acquired from the bottom left and top right points).

Time Complexity: it is largely defined by sorting the strips. Python in-built $sort()$ function uses Timsort, a sorting algorithm, which was developed specifically for Python with the time complexity of $O(N \times logN)$ [10]. With the total of two sorts required, the total time spent would still follow $O(N \times logN)$, which is significantly lower than $O(N^4)$.

B. *Retrieving Edges*

Once the intermediate interpolation steps are complete, edge data points need to be retrieved. However, the amount of collected edge points is important to consider for performance

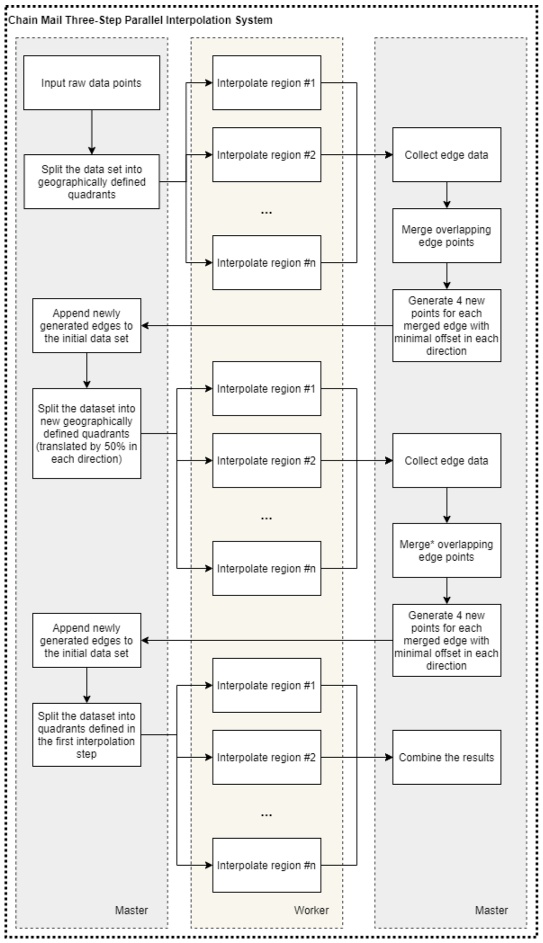

Chain Mail Three-Step Parallel Interpolation System
Input raw data points
Split the data set into geographically defined quadrants
Interpolate region #1
Interpolate region #2
...
Interpolate region #n
Collect edge data
Merge overlapping edge points
Generate 4 new points for each merged edge with minimal offset in each direction
Append newly generated edges to the initial data set
Split the dataset into new geographically defined quadrants (translated by 50% in each direction)
Interpolate region #1
Interpolate region #2
...
Interpolate region #n
Collect edge data
Merge* overlapping edge points
Generate 4 new points for each merged edge with minimal offset in each direction
Append newly generated edges to the initial data set
Split the dataset into quadrants defined in the first interpolation step
Interpolate region #1
Interpolate region #2
...
Interpolate region #n
Combine the results
Master
Worker
Master

Fig. 6. Three-step chain mail interpolation system diagram

reasons. Every collected edge is an extra point added to the data set. This is not a problem for grids with small resolutions. However, for loosely populated grids with high resolution, this will introduce a massive performance slowdown because the number of collected edges may vastly exceed the number of initial data points. This will result in a considerable performance drop. A simple solution to this problem is to take the four corner points and a certain percentage of the remaining edge points. This can be controlled by using a step variable. The following Python pseudo-code proposes an example of how this can be done (Listing 1). For example, with a step value of 2, the function will collect 50% of the edge points, with a step value of 4 — 25% of the points, etc. Only a small number of edge points should be collected, because it must be enough to represent a general behavior of the edge values, but also keep the extra load on the interpolation process low.

```
def get_edges(data, step=2):
  # data is a 2D arr of interpolated values
  edge_points = []
    for row in data:
      # if row is first or last, take all
      # data points at certain step value
      if row.index == 0 or
      row.index == (data.size - 1):
        for point in row:
          if point.index % step == 0:
            edge_points.append(point)
      # or if the row index is at
      # the step value:
      elif row.index % step == 0:
        # take the first and last points
        edge_points.append(row.first_point)
        edge_points.append(row.last_point)
  return edge_points
```

Listing 1: Pseudocode for edge collection behavior

C. *Merging Edges*

When regions are interpolated and edge data points are collected, the next step becomes to resolve the points that have the same coordinates but different values. A prime example of such a situation would be two neighboring regions interpolating a common edge, and ending up with two completely different values for the same coordinate, purely due to them having different inputs. There are two ways of approaching this problem. The first is to simply average the value of the two points with the same coordinates. The second approach is to use a weighted (or inversely weighted) average, depending on some other metric. This work assumes the first approach.

VI. Evaluation and Results

In this section we report and discuss the results of the experimental evaluation of our proposed approach. All local experiments have been performed using the following hardware: 8-core (16 threads) Intel Xeon CPU E5-2667 V4 3.20 GHz, 64 GB of DDR4 RAM. The tests were run using Apache Ray framework on Windows and Windows Subsystem for Linux (WSL). Two data sets were used: a synthetic one generated specifically for testing edge data consistency (interpolation results visualized in Figure 7a) and a proprietary air quality one (interpolation results visualized in Figure 8a).

A. *EXP1: Three-Step Kriging (Synthetic)*

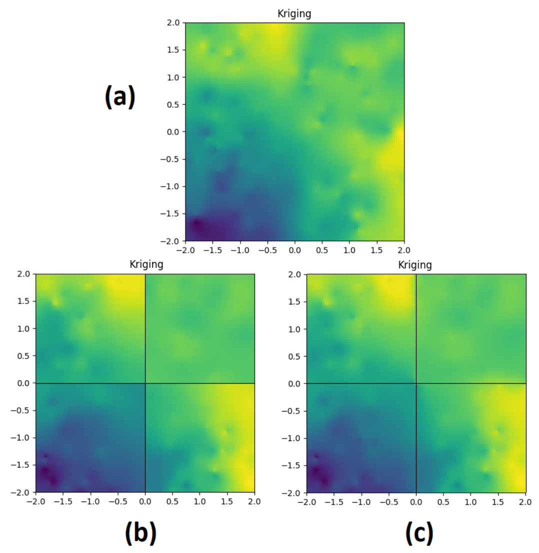


Fig. 7. Comparison of single process (a), naive parallel (b), and three-step Chain Mail (c) approaches

Figure 7 combines the results of all three approaches: single process interpolation (a), naive parallel interpolation (b), and three-step Chain Mail interpolation (c). Visually, (b) has the issue of colliding edge values, which is especially noticeable around the $(0, 0)$ coordinate where there is a lack of ”smoothness” when moving over from one region to another. (c) does not suffer from this problem, and the regional border transitions resemble the single parallel approach more.

For accuracy evaluation, we use Relative Root Mean Squared Error (RRMSE) results for naive parallel vs Chain Mail approaches for different metrics. The accuracy baseline for RRMSE is the surface generated by the single process interpolation. Thus, the approach with smaller RRMSE will be the one closer to the single process interpolation in terms of accuracy. The tested variables are: (i) Grid resolution, which is the desired resolution of the interpolated surface; (ii) Edge step size, which represents the number of edges used for the second and third interpolation steps of the Chain Mail approach; (iii) Data set size, defined as the total number of points given as input to the interpolation algorithms; and (iv) Grid configuration size, as the total number of parallel workers in a grid (e.g., 2×2 or 3×3).

Grid Resolution. Table I presents RRMSE values for naive parallel and three-step Chain Mail approaches relative to the single process interpolation at different grid resolutions (bold font highlights the best result). The show that the RRMSE stays relatively constant with the resolution increase for both of the parallel approaches, but the error of the three-step Chain Mail approach is smaller by 0.7% (or 20% in relative terms) than that of the naive parallel approach.

TABLE I
RRMSE FOR PARALLEL APPROACHES RELATIVE TO SINGLE PROCESS INTERPOLATION (100 POINTS, 20% OF EDGES TAKEN)

Method	Grid resolution (at 20% edge values)			
	30×30	50×50	100×100	150×150
naive parallel 2×2	3.34%	3.25%	3.19%	3.17%
Chain Mail 2×2	**2.78%**	**2.58%**	**2.46%**	**2.49%**

Edge Step Size. Table II compares RRMSE values between the two parallel approaches (for 2×2 grids) in regards to the edge step size. At 100×100 initial grid resolution, each of the 4 parallel workers would interpolate an area with the resolution of 50×50. The edge steps of 2, 5, 10, 15, 25, and 50 represent 50%, 20%, 10%, 7%, 4% and 2% of edge values, respectively.

TABLE II
RRMSE FOR DIFFERENT EDGE STEP SIZES

Method	Edge step (at 100×100 grid res.)					
	50	25	15	10	5	2
naive parallel	3.19%					
Chain Mail	**2.62%**	**2.57%**	**2.64%**	**2.46%**	**2.56%**	**2.44%**

The smaller the step, the more edges will be taken in – and vice versa. However, it can be noticed that even by introducing the 2% of the edge data values (which, in this case, is just the four vertices of each of the quadrants), the error is lowered by 0.58% (or approximately 30% in relative terms). The accuracy differences between various numbers of added edge points is rather small, so there is little reason to take more than 20% of the edge points (or step value of 5 for this specific example).

Data Set Size. Table III presents RRMSE values for different data set sizes at 100×100 grid resolution and 20% of edge data points added. We can see that the greater the sample size, the smaller the accuracy improvement over the naive parallel approach. This can be attributed to the fact that the more points that are present, the higher the chance of those points lying close to the edges. The accuracy improvement of the Chain Mail approach over the naive parallel approach slowly drops from 0.73% to 0.26% (or 23% to 14% in relative terms) after tripling the number of initial data points.

TABLE III
RRMSE FOR DIFFERENT DATA SET SIZES

Method	# of raw points		
	100	200	300
naive parallel 2×2	3.19%	2.11%	2.02%
Chain Mail 2×2	**2.46%**	**1.89%**	**1.74%**

Parallel Grid Configuration. Table IV presents RRMSE values for different worker configurations. We can see that the higher the amount of grid cells, the greater the error grows. With a constant number of raw data points, each grid cell will have fewer and fewer points to interpolate from, reducing the amount of context that each cell can gather from the area. The Chain Mail approach starts off with small error improvements (for example, 0.6% difference for 2×2 grid), but the difference becomes much greater (almost by a factor of 2) with the increase in the grid configuration size.

TABLE IV
RRMSE FOR DIFFERENT PARALLEL GRID CONFIGURATIONS

Method	Parallel configuration (at 100x100 grid res.)			
	2×2	3×3	4×4	5×5
naive parallel	3.19%	3.46%	5.99%	8.08%*
Chain Mail	**2.46%**	**3.27%**	**3.42%**	**4.76%**

The number for the naive parallel approach at 5×5 grid configuration includes only the RRMSE values of the grid cells that have managed to interpolate data successfully. Some of the grid cells did not have enough raw data for a successful interpolation, highlighting another disadvantage of the naive parallel approach. Therefore, technically, the RRMSE of 5×5 for the naive parallel approach can not be calculated since some of the data is missing completely. The Chain Mail approach is able to fix these empty regions, as long as the neighboring regions have sufficient data.

B. EXP2: Air Quality Data Interpolation

The proprietary industrial air quality data set consists of PM2.5, PM10, and CO. The figures presented in this section are based only on the PM2.5 values. The interpolated area is the area of Greater London, defined by the area between the two coordinate points $(-0.77083, 51.22977)$ and $(0.66632, 51.77933)$, where the first value shows longitude, and the second – latitude. Figure 8 shows the results of different interpolations of the data set. The range of values is between $2.4\mu g/m^3$ and $10.1\mu g/m^3$. Higher (or brighter) values represent worse air quality and vice versa.

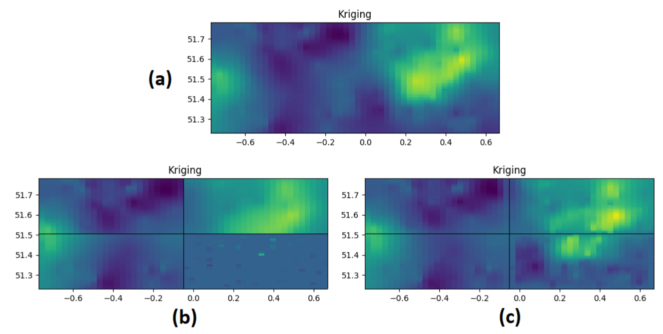


Fig. 8. Comparison of single process kriging (a), naive parallel approach (b), and three-step Chain Mail approach (c) for air quality data in Greater London

Note how for the naive parallel approach (b) the bottom right quadrant did not have enough data to generate a meaningful interpolated surface. As a result, the surface looks uniform with rare points sticking out. On the other side, the Chain Mail approach (c) had enough context to work with because of the additional data from neighbors that was generated after the first and second interpolation steps. Combined with the initial data points, that intermediary data allowed to reconstruct a rough representation of the quadrant, where the result would resemble the surface from the single process interpolation (a).

C. EXP3: Execution Time Evaluation

Here, we evaluate the execution times of the three-step Chain Mail approach compared to the naive parallel and the single process ones. Two variables are tested: edge step and data set size. The results of the first test determine the optimal edge step value that should be used for the second test.

Edge Step Size. The test is performed on the synthetic data set at 100×100 interpolation resolution in a 2×2 three-step Chain Mail configuration. This means that each worker will interpolate a grid cell with a resolution of 50×50, and the edge value percentages are the same as in the RRMSE evaluation in for different step sizes of EXP1. The results (Table V) show that using 50% of edge data points reduces the performance of the algorithm drastically, no matter the data set size, and using 20% of edge data has a small performance decrease. As for the remaining values, there is little to no performance difference for using between 2% and 10% of edge data points.

Combined with the accuracy evaluation for different edge steps from the Table II (that concluded that the optimal RRMSE is between 10% and 20%), it can be concluded that 10% is the most optimal amount of edge data points to include in the intermediate data sets. With this knowledge in mind, we evaluate Chain Mail performance versus other approaches.

TABLE V
EXECUTION TIME (SECONDS) FOR DIFFERENT DATA SET SIZES AT DIFFERENT EDGE STEP VALUES

Data set size	% of edge data points added						
	2%	4%	5%	7%	10%	20%	50%
100	**1.401**	1.415	1.450	1.474	1.502	2.577	17.911
200	**1.515**	1.582	1.610	1.633	1.651	2.860	20.211
300	**1.469**	1.601	1.644	1.670	1.797	3.415	22.167
400	**1.686**	1.724	1.771	1.966	2.105	4.042	23.029
500	**1.835**	2.030	2.075	2.162	2.351	4.432	27.695

Data Set Size. Here, we compare execution times for different data set sizes for the single process interpolation, naive parallel, and both Chain Mail two-step and three-step approaches in the 2×2 and 3×3 configurations. Single process interpolation has been tested on both native Windows and WSL configurations since all of the parallel approaches use WSL. However, WSL had shown to have virtualization overhead (5× slower than native Windows), so only the single process Windows results will be shown in the figures. The single process WSL results are still present in the tables.

The test is performed on the synthetic data set at 100×100 interpolation. For the Chain Mail approaches, 10% of the edge data points have been taken in each step (step value of 10 for the 2×2 configuration and 7 for the 3×3 configuration). Results are demonstrated in Table VI.

TABLE VI
EXECUTION TIME (SEC) OF INTERPOLATION APPROACHES

Method	Data set size (# of points)				
	200	400	600	800	1000
single (WSL)	26.36	154.569	419.103	776.013	1189.365
single (Windows)	3.554	26.304	82.378	151.795	239.492
naive 2×2	1.334	1.205	1.365	1.625	2.213
chain 2×2 (2 step)	**1.223**	1.409	1.898	2.396	3.420
chain 2×2 (3 step)	1.771	2.053	2.824	3.84	5.837
naive 3×3	1.259	**1.019**	**1.041**	**1.076**	**1.160**
chain 3×3 (2 step)	1.557	1.533	1.733	1.727	1.849
chain 3×3 (3 step)	1.588	1.651	1.748	2.000	2.244

Figure 9 shows the data from this table (with WSL single process interpolation excluded). As can be seen from the figure, all parallel algorithms vastly outperform the single process interpolation. Supported by the theoretical time complexity example, real-world performance indicates that the Chain Mail approach is slower than the naive parallel one of the same grid size. It can be noted that performance for Chain Mail of a higher degree (e.g., 3×3) is on par with the naive parallel approach of a lower degree (2×2).

D. EXP4: Cloud (AWS) Performance

Now we discuss the performance of the system in the cloud. The evaluation is conducted on the synthetic data set at 50×50 interpolation resolution with 10% of edge values. Six different grid configurations have been tested: 2×2 (4 workers), 5×5 (25 workers), 10×10 (100 workers), 15×15 (225 workers), 20×20 (400 workers), and 25×25 (625 workers). 300 points

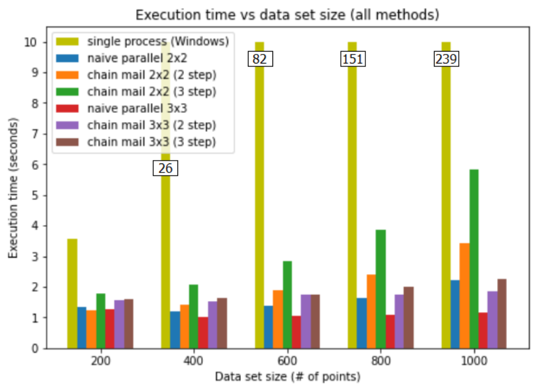


Fig. 9. Comparison of execution times (in seconds) for different methods

per worker were chosen as a benchmark baseline. Table VII shows the execution times for different data set sizes.

TABLE VII
CLOUD SCALABILITY TEST

Serverless Function (MB RAM)	Execution time for a given data set size (seconds)					
	1200	7500	30000	67500	120000	187500
512 (cold)	59.631	59.351	57.328	59.707	63.295	70.199
512	40.217	46.172	46.313	49.483	54.009	63.509
1792 (cold)	16.385	17.773	21.615	22.982	28.018	37.899
1792	**11.603**	**14.011**	**17.507**	**18.696**	**20.007**	**26.471**

There are two key findings based on this experiment. First, there is a performance gap between local execution versus cloud. There are a couple of factors potentially affecting cloud performance. The problem of *cold start* may be present where a serverless function is not active until its first execution [11].

Second, with the initial cloud overhead out of the way, the data shows little change with the increase of amount of parallel worker functions. For example, 25×25 grid spawns 625 workers compared to 4 of the 2×2 grid, but it took only 22% longer to execute for a much greater data set size (187500 vs 1200 points). The increased execution time can be attributed to the data processing steps of the master node, as well as sending HTTP requests to control cloud infrastructure.

To sum up, if the system is able to balance the data set size with the right amount of workers, then the execution times can stay similar to that of the smaller data set sizes.

E. Performance Summary

Table VIII summarizes the comparison of the three approaches. Compared to the single process and naive parallel approaches, Chain Mail presents performance/accuracy tradeoff. Single process interpolation, which is the baseline for all of the comparisons, offers the best accuracy of the bunch because it has the most context due to having all of the data set available to it. However, it runs extremely slowly when compared to the parallel approaches. The naive parallel approach is the fastest of the three because it has no extra steps, like the Chain Mail one, and focuses purely on parallelization. As a result, the accuracy suffers a bit and the edge collision issue is introduced. The Chain Mail approach is much faster than the single process interpolation and it is in line with the performance of the naive parallel approach, albeit slower by a factor of approximately 2.5. Regarding accuracy, its RRMSE is approximately 25% lower than that of the naive parallel approach, and it also eliminates the issue of colliding edges around regional borders when looking at the visualized results.

TABLE VIII
HIGH-LEVEL INTERPOLATION APPROACH COMPARISON

Method	Speed	Accuracy
single process interpolation	Slowest (base)	Most accurate (base)
naive parallel approach	Fastest (×108 for 2×2, ×206 for 3×3)	Less accurate, has edge collisions (3.3% off)
Chain Mail approach	Faster (×41 for 2×2, ×106 for 3×3)	More accurate, no edge collisions (2.4% off)

Parallel Approach Advantages. The most apparent advantage of the Chain Mail approach is that it offers a good middle ground between speed and accuracy, including a more pleasant visualization result compared to the naive parallel approach.

Also, as can be seen in Figure 8, if there is not enough data in a grid cell, the naive parallel approach may fail to generate a surface. The Chain Mail algorithm is able to provide some more data to sparsely populated grid cells during the intermediate interpolation steps, generating a surface.

Parallel Approach Downsides. There are two main points to be considered when using the Chain Mail algorithm. First, if the edges of the neighboring regions have enough data when partitioning the data set into a grid, the naive parallel approach may be accurate enough and the edge collision issue may not be present. Second, the Chain Mail algorithm can only get enough neighboring data from one grid cell in each direction. This means that if the initial surface is split into too many cells, at some point the accuracy drops considerably. On the other side, not splitting the initial area into enough cells may leave free execution time improvements on the table. This entails careful consideration of the granularity of the grid.

VII. CONCLUSION AND FUTURE WORK

We present and analyze a system capable of distributed data interpolation that improves execution time and also solves (or reduces to an acceptable level) the issue of colliding edges. The implemented three-step Chain Mail system has visually demonstrated to remove the edge value collisions present in images for the naive parallel approach, as well as to lower RRMSE values at the same time. Compared to the naive parallel implementation, the Chain Mail approach is slower but within an order of magnitude. However, if compared to the standard single process interpolation, the chain mail approach vastly outperforms it, allowing to interpolate data sets of sizes that would not be feasible with the single process interpolation. All in all, the presented three-step Chain Mail algorithm tries to offer a good middle ground between speed and accuracy, heavily focusing on speed improvement. As the next step, we would like to investigate the performance of the Chain Mail approach on other interpolation algorithms, such as IDW.

References

[1] A. Asratyan and M. Joshi, “Iot framework for water monitoring using the m-bus interface,” p. 97. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1334333/FULLTEXT01.pdf

[2] G. Shaddick, M. L. Thomas, P. Mudu, G. Ruggeri, and S. Gumy, “Half the world’s population are exposed to increasing air pollution,” vol. 3, no. 1, pp. 1–5, number: 1 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/s41612-020-0124-2

[3] US EPA, “Particulate matter (pm) basics,” 2018.

[4] US EPA, “Health and environmental effects of particulate matter (pm),” 2018.

[5] D. W. Wong, L. Yuan, and S. A. Perlin, “Comparison of spatial interpolation methods for the estimation of air quality data,” vol. 14, no. 5, pp. 404–415, number: 5 Publisher: Nature Publishing Group. [Online]. Available: https://www.nature.com/articles/7500338

[6] Kriging interpolation explanation | columbia public health. [Online]. Available: https://www.publichealth.columbia.edu/research/population-health-methods/kriging-interpolation

[7] B. V. Srinivasan, R. Duraiswami, and R. Murtugudde, “Efficient kriging for real-time spatio-temporal interpolation,” in *Proceedings of the 20th Conference on Probability and Statistics in the Atmospheric Sciences*. American Meteorological Society Atlanta GA, 2010, pp. 228–235.

[8] M. P. Armstrong and R. Marciano, “Parallel spatial interpolation,” in *AutoCarto Conference*, 1993, pp. 414–414.

[9] K. Henneböhl, M. Appel, and E. Pebesma, “Spatial interpolation in massively parallel computing environments,” in *Proc. of the 14th AGILE International Conference on Geographic Information Science (AGILE 2011)*. Citeseer, 2011.

[10] R. Bauermeister. Understanding timsort. [Online]. Available: https://medium.com/@rylanbauermeister/understanding-timsort-191c758a42f3

[11] M. Shilkov. Cold starts in AWS lambda. [Online]. Available: https://mikhail.io/serverless/coldstarts/aws/