Surface-to-volume ratios of space-filling curves on 2:1 balanced adaptive grids

Student Name: Leith Osborne

Supervisor Name: Dr Tobias Weinzierl

Submitted as part of the degree of BSc Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

Abstract —

Context: Locality preserving properties of space-filling curves have made space-filling curves a popular tool for partitioning multi-dimensional grids in parallelisation. The quality of partitions induced by space-filling curves on regular Cartesian grids has analytically been proven to be near optimal besides a constant. Comparatively, the explosion of partition opportunities available on adaptive grids has made the quality of adaptive grid partitions difficult to quantify.

Aims: The aim for this project is to assess the quality of partitions induced by the Hilbert curve on 2:1 balanced two-dimensional adaptive grids. By measuring surface-to-volume ratios of partitions on multitudes of adaptive grids in a brute-force manner, we aim to compare our results to partitions of optimal parallel efficiency, introduced by (Zumbusch 2000) and derive meaningful estimates for the quality of partitions induced by the Hilbert curve on adaptive grids.

Method: Comprised of an infinite number of adaptive grids, we model the search space of the problem as a tree-like structure and propose a brute-force algorithm that measures the quality of partitions induced by the Hilbert curve through a breadth-first search. We compute the quality of partitions using surface-to-volume ratios and exploit properties of the Hilbert curve such as symmetry and recursion to reduce computation time.

Results: The quality of partitions induced by space-filling curves on regular grids are of optimal parallel efficiency, with (Zumbusch 2000) providing 7 as the constant part for the Hilbert curve. Our results infer that the average surface-to-volume ratios on adaptive grids are also quasi-optimal, with the constant part as low as 5.02 for the first three depth levels. We also find that the worst-case surface length s for volume v to be $s = 3v + 1$.

Conclusion: The brute-force solution is able to compute the surface-to-volume ratios for all partitions induced by the Hilbert curve on 2:1 balanced adaptive grid structures on a depth level basis. Results from the first three depth levels suggest average surface-to-volume ratios of partitions on adaptive grids are quasi-optimal.

Keywords — space-filling curves, adaptive grid partitions, quality of partitions, optimal parallel efficiency, surface-to-volume ratios, parallelisation, the Hilbert curve,

I INTRODUCTION

Space-filling curves present a method of mapping multi-dimensional spaces to a one-dimensional domain and are a popular tool for domain decomposition in scientific computing. Their ability

to provide compact partitions of uniform size has made their applications prevalent in parallelisation. Ranging within solvers for shallow water equations (Xia & Liang 2016) to numerical simulations of detonation (Wang et al. 2015), space-filling curves are commonly used as a tool for partitioning regular and adaptive Cartesian grids.

Numerous load-balancing packages exploiting partitions induced by space-filling curves also feature in a vast range of applications. The p4est software library (Burstedde et al. 2011) provides scalable algorithms for parallel adaptive mesh refinement and has been used to model mantle convection (Rudi et al. 2015) and computational fluid dynamics (Drui et al. 2016). Likewise, the Peano software provides a framework that handles parallel adaptive grid traversals internally for PDE solvers (Weinzierl 2019). ExaHype (An Exascale Hyperbolic PDE Engine) (Reinarz et al. 2020), a high-performance engine able to simulate physical phenomena, uses the Peano framework for cache-efficient adaptive mesh refinement.

A Locality properties of space-filling curves

With applications in other scientific fields such as image compression (Liang et al. 2008), database indexing (Psomadaki et al. 2016) and neural networks (Corcoran et al. 2018), various metrics have been used to quantify the locality properties of space-filling curves on regular grids. Relative to parallelisation, (Zumbusch 2000) analyses the quality of partitions produced by space-filling curves in search for an efficient PDE solver (Griebel & Zumbusch 2001). In summary, the paper measures the quality of partitions using surface-to-volume ratios and considers estimates of type

$$s \leq C_{part} \cdot v^{\frac{d-1}{d}} \tag{1}$$

with low constants for C_{part} as optimal. Importantly, (Zumbusch 2000) proves partitions induced by space-filling curves on regular grids are quasi-optimal:

Lemma I.1. *Given a connected discrete space-filling curve F on a domain $[1, ..., k]^d$ and a partition $F([j, ..., j + v - 1])$ of v nodes, the surface s of the partition is bounded by Equation 1. The constant C_{part} depends on the curve.*

For the two-dimensional Hilbert curve mapped onto regular grids, (Zumbusch 2000) unveils an upper bound for C_{part} equal to 7. Due to the explosion of partition opportunities available on adaptive grids, limited work has been made to quantify the quality of partitions on adaptive grids.

B Objectives and aims

The aim for this project is to assess the quality of partitions induced by space-filling curves on adaptive grids. We are interested in finding out whether the quality of adaptive grid partitions induced by space-filling curves are similar to the quasi-optimal partitions space-filling curves provide on regular grids. If the quality of adaptive grid partitions conforms with Equation 1, we aim to derive tight bounds and averages for C_{part}.

To tackle the objectives of the project, we propose a brute-force algorithm that calculates surface-to-volume ratios for all partitions induced by the Hilbert curve on two-dimensional 2:1 balanced adaptive grids on a depth level basis. By collecting substantial amounts of data relating to the surface-to-volume ratios of partitions, our results from the first three depth levels infer that the average quality of partitions conform with Equation 1.

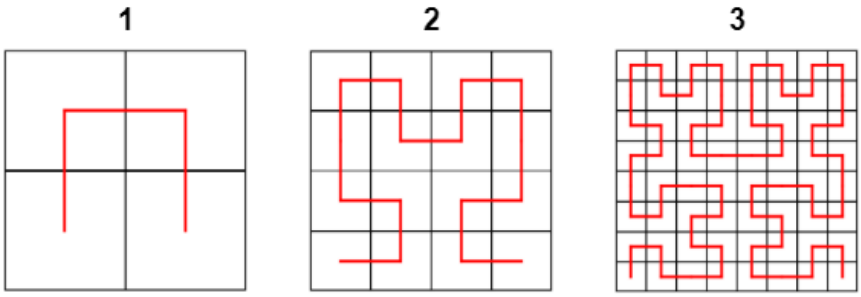


Figure 1: The first three iterations of the Hilbert curve.

The remainder of the report is organised as follows: Section II analyses the properties that make space-filling curves effective in parallelisation and uses the work of (Zumbusch 2000) to outline a method of assessing the quality of partitions induced by space-filling curves. In Section III, we explore the various two-dimensional adaptive grids that exist within the search space and propose a brute-force algorithm that calculates surface-to-volume ratios for all possible partitions that exist within each grid structure. Operating on a depth level basis, the major contribution of this project is in the methods we use to reduce the computational cost of the algorithm. During the grid generation process, we use recursive and reflective properties of the Hilbert curve to lower the number of grid structures we consider in each depth level. Additionally, we introduce a pattern dictionary that exploits the similarities of partitions across various grid structures. Section IV provides the runtime of the algorithm across the first three depth levels and analyses the data obtained. Our results infer that the average surface-to-volume ratios on adaptive grids are of optimal parallel efficiency, with c_{part} equal to 5.02 for the first three depth levels. We also find that the worst-case surface length s for the volume v to be $s = 3v + 1$. We discuss the strengths and limitations of the brute-force solution in Section V, before concluding the report with an overview of the project's major findings.

Depth level - The maximum number of refinements made on any cell in a given grid structure.

Partitioning - The process of breaking down a grid structure into a subgrid. Each subgrid is known as a partition.

II RELATED WORK

Is it possible to obtain a continuous surjective mapping from $[0, 1]$ onto $[0, 1]^2$? In his article, (Peano 1890) solved this question by producing such a curve in mathematical terms. The following year, (Hilbert 1891) found a variant of the solution and constructed a geometric representation of the curve, shown in Figure 1. Today, these curves are categorised as space-filling and conform to the following (Bader 2012, p17):

Definition II.1 (Space-filling curve)**.** *Given a curve $f_*(I)$, where $I \in \mathbb{R}$ and the corresponding mapping $f : I \to \mathbb{R}^n$, then $f_*(I)$ is called a space-filling curve, if $f_*(I)$ has a strictly positive Jordan content (area, volume, ...).*

Implicitly, space-filling curves can therefore index hypercubes. Looking at the two-dimensional case, (Quinqueton & Berthod 1981) design an adaptive space-filling scanning algorithm able to

sequentialise Cartesian grids following a quadtree structure. Noticing the influx of applications using space-filling traversals, (Gotsman & Lindenbaum 1996) analyses the properties of space-filling curves and find that they preserve locality, with the Hilbert curve achieving near optimal locality. These two properties that space-filling curves hold are highly desirable for parallelisation algorithms.

The computational efficiency of a parallel algorithm can largely be impacted by the load distribution and communication time between processors. By partitioning the workloads between processors as uniformly as possible, we reduce the time processors spend idle, waiting for other processors to finish. As space-filling curves can sequentialise multidimensional domains, it is a trivial task to partition the sequence into equally sized segments. To lower communication time, (Bader 2012, p145) expresses the importance of shortening the boundaries of the partitions. As space-filling curves preserve locality, neighbour relations are retained between the index space and image space. Partitions combining neighbouring points in the index space will therefore correspond to a local point set in the image space too.

A Quality of partitions induced by space-filling curves

The execution time of a parallel program is dependent on the parallel computing time and the amount of communication between processors. (Zumbusch 2000) introduces a method of measuring the parallel efficiency of a program using surface-to-volume ratios of partitions. We calculate the parallel computing time of an algorithm from the partition of n data onto p processors (the volume $\frac{n}{p} = v$) and model the communication time by the amount of data needing to be transferred between processors, which is proportional to the surface s of the partition. Together, (Zumbusch 2000) derives the total runtime t as

$$t = C_1 \frac{n}{p} + C_2(t_{startup} + s \cdot t_{bandwidth}) \tag{2}$$

Accordingly, the paper suggests that the surface-to-volume ratio $\frac{s}{v}$ of the partition must be minimised to achieve a high parallel efficiency of

$$\text{efficiency} = 1/(1 + \frac{C_2}{C_1}(\frac{1}{v} t_{startup} + \frac{s}{v} \cdot t_{bandwidth})) \tag{3}$$

Explaining that the n-sphere holds the lowest continuous surface to volume ratio, (Zumbusch 2000) simplifies this to Equation 1.

Definition II.2 (Optimal parallel efficiency)**.** *Two-dimensional partitions of surface length s and volume v are considered to be of optimal parallel efficiency if*

$$s \leq C_{part} \cdot v^{\frac{1}{2}} \tag{4}$$

where C_{part} is a low constant.

Definition II.3 (Hölder continuous)**.** *A mapping $f : I \to \mathbb{R}$ is called Hölder continuous with exponent r on the interval I, if there is a constant $C > 0$ for all parameters $x, y \in I$, such that:*

$$\|f(x) - f(y)\|_2 \leq C|x - y|^r \tag{5}$$

(Zumbusch 2000) derives Lemma I.1 through a proof featuring Hölder continuity and the connectedness of the partition. For space-filling curves, the Hölder continuity presents a relation between the distance of the indices and the distance of the image points. (Bader 2012, p168) provides a detailed proof based on the three-dimensional Hilbert curve which confirms that all connected, recursive space-filling curves are Hölder continuous with exponent d^{-1}. By discretizing the mapping of the three-dimensional Hilbert curve to the image of the interval $[x, y]$ (such that the curve is parameterised by volume), (Bader 2012, p169-170) shows that Hölder continuity holds for discrete space-filling curves, where the length of the interval $|x - y|$ is equal to the volume of the image $f[x, y]$ and where $\|f(x) - f(y)\|_2$ represent the distance between the two corresponding image points.

Locality measures relating the volume of a discrete space-filling curve to the distance of the image points of the interval boundaries that define the partition were first introduced by (Gotsman & Lindenbaum 1996). Here, it is shown that the locality of a d-dimensional Hilbert curve is indeed Hölder continuous and held an upper bound of $(d+3)^{\frac{d}{2}}2d$. However, as (Bader 2012, p178) states, if space-filling curves define the partitions on a computational grid for parallelisation, it is more fitting to measure the amount of data to be communicated between partitions using surface-to-volume ratios. (Zumbusch 2000) explains that the quasi-optimality of the Hölder continuity does transfer to the surface-to-volume ratio of partitions (Lemma I.1). Hence, partitions based on space-filling curves provide a speedup for large problems of

$$\text{efficiency} = 1/(1 + (\frac{C_2 C_{part} t_{bandwidth}}{C_1} \cdot \frac{p}{n^{\frac{1}{d}}})) \tag{6}$$

which suggests that optimal parallel efficiency is achieved as $n \to \infty$.

If Equation 1 were to hold true for partitions induced by space-filling curves on adaptive grids, space-filling curves on adaptive grids would also provide partitions that are of optimal parallel efficiency.

B Construction of the Hilbert curve

In order to calculate surface-to-volume ratios on a multitude of adaptive grids, an adaptive grid generation method is implemented. (Bader 2012, p32) expresses how iterations of the Hilbert curve can be defined by a context-free grammar. Using a set of non-terminals $\{H, A, B, C\}$ that represent basic patterns, a set of terminals $\{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ that describe the transitions between the basic patterns and the following set of production rules,

$$H \longleftarrow A \uparrow H \rightarrow H \downarrow B$$
$$A \longleftarrow H \rightarrow A \uparrow A \leftarrow C$$
$$B \longleftarrow C \leftarrow B \downarrow B \rightarrow H$$
$$C \longleftarrow B \downarrow C \leftarrow C \rightarrow A$$

any Hilbert curve on a regular Cartesian grid can be described and produced. To ensure the grid remains regular between refinements, (Bader 2012, p32) introduces an additional rule that insists all non-terminal symbols must be replaced at once in a derivation step. Figure 2 shows the refining effects the production rules have on the basic patterns and Figure 3 demonstrates how the first three iterations of the Hilbert curve can be constructed.

By making a few tweaks that are explained in Section III.A, a modified algorithm can be used to describe and produce 2:1 balanced adaptive grids.

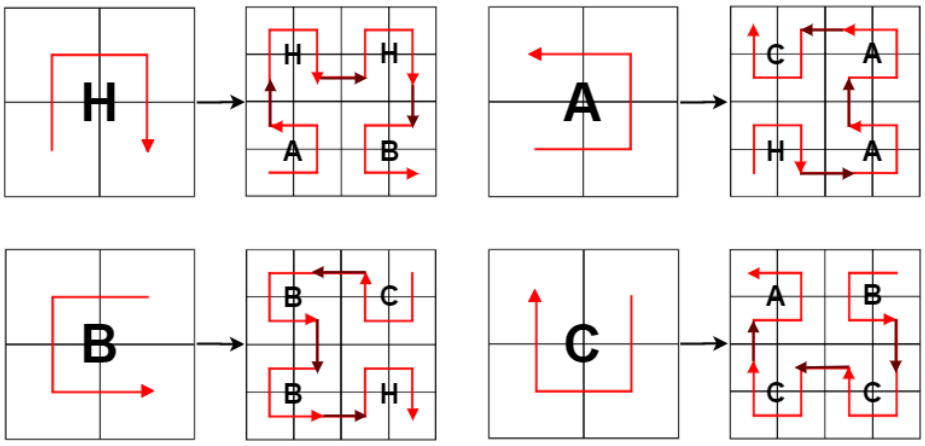


Figure 2: Replacement schemes for the basic patterns that the non-terminal symbols represent.

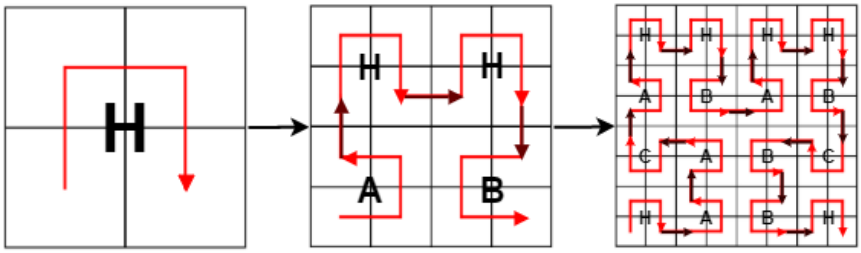


Figure 3: Production of the first three iterations of the Hilbert curve.

Definition II.4 (2:1 balanced adaptive grid)**.** *An adaptive grid is 2:1 balanced if and only if all neighbouring cells within the grid are of at most a 2:1 difference in size.*

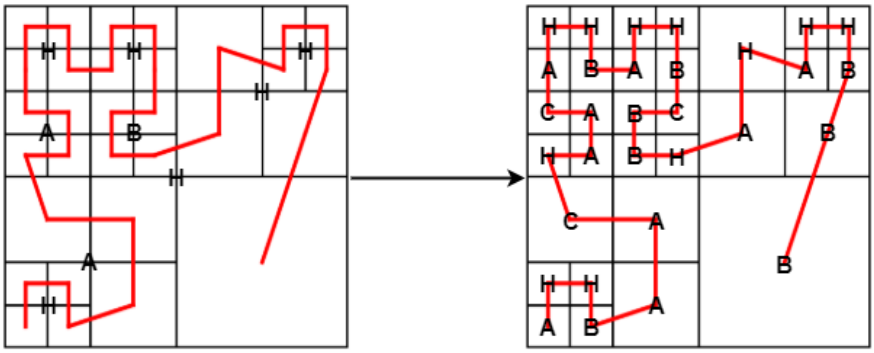
First introduced as a 1-irregular rule, (Bank et al. 1983) define and discuss the advantages of using 2:1 balanced adaptive grids in finite element mesh generation. With many octree-related applications requiring the 2:1 balance condition since, (Sundar et al. 2008) provide an efficient algorithm for enforcing 2:1 balance refinement in parallel. To simplify the grid generation process in Section III, this constraint also features in our brute-force algorithm.

III SOLUTION

A Production of the Hilbert curve on adaptive grids

To describe the Hilbert curve on adaptive grids, we modify the grammar defined by (Bader 2012, p17). Instead of representing the non-terminals as four basic patterns, every cell in the grid is assigned a non-terminal symbol dependent on the pattern of its next refinement. In other words, each basic pattern from Bader's grammar is now described by four non-terminals rather than one. For instance, the initial Hilbert curve previously described as H is now described as $[A \uparrow H \rightarrow H \downarrow B]$. It is also essential to remove the additional derivation rule found in (Bader 2012, p17), else the new grammar would remain limited to constructing regular grids only. By assigning each non-terminal an additional integer representing the depth level of the basic pattern, any two-dimensional Hilbert curve mapped onto an adaptive grid can be described

and constructed. An example output of the proposed grammar can be found in Figure 4, which also highlights the limitations of using Bader's grammar on adaptive grids.



Depth Array = [3,3,3,3,2,2,2,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,2,2,3,3,3,3,2,1]

Production Array = [A,H,H,B,A,A,C,H,A,A,C,A,H,H,B,A,H,H,B,C,B,B,H,A,H,A,H,H,B,B,B]

Figure 4: Example of the transition from using the grammar defined by Bader (excluding the additional derivation rule, left) and the proposed grammar (right) on an adaptive grid.

B Quantifying the search space of the problem

By limiting the brute-force algorithm to only traverse adaptive grids that are 2:1 balanced, we can model the search space of the problem as a tree-like structure (Figure 5); all grid generations can be produced solely from refining grid structures from the depth level above.

In the case of regular grids, it is trivial that only one regular grid exists within each depth level

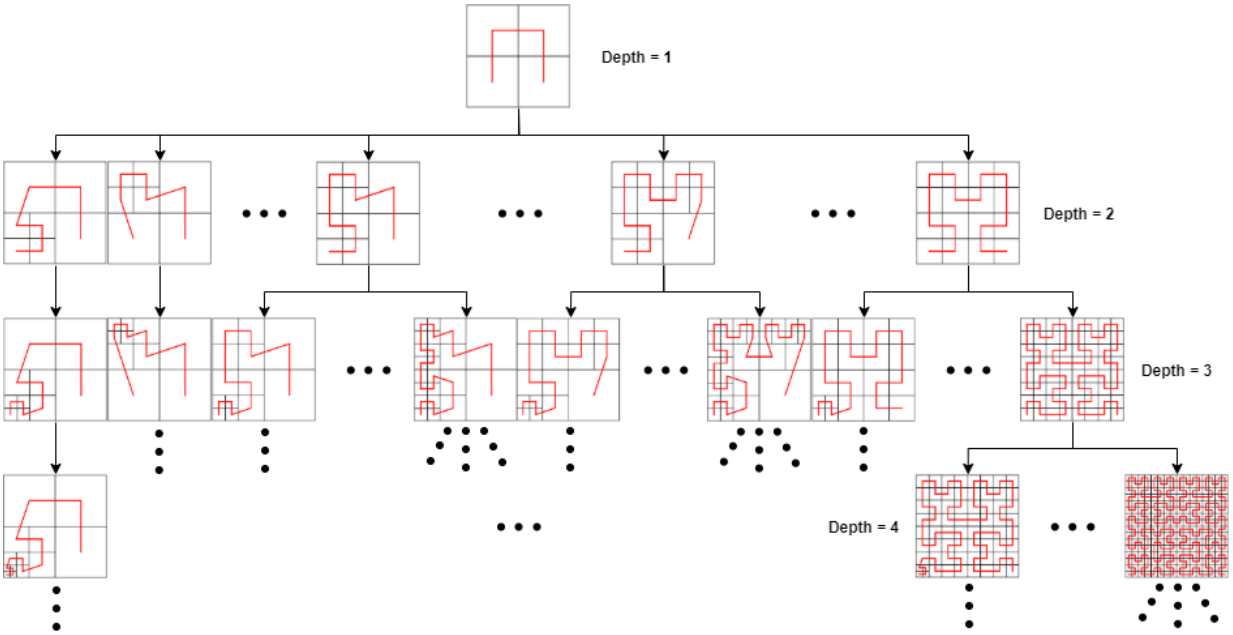


Figure 5: Search space of the brute-force algorithm.

since all cells within a regular grid must be uniform in size. In comparison, the quantity of 2:1

balanced adaptive grids that exists within each depth level explodes. As the Hilbert curve from the first depth level maps over four cells of uniform size, all four cells are refinable. Refining any combination of these four cells outputs a 2:1 balanced adaptive grid that can be traversed by the Hilbert curve. The total number of adaptive grids that exist in the second depth level is therefore $\binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 2^4 - 1 = 15$. Likewise, the regular grid in the second depth level (item 15 in Figure 6) is able to generate $\binom{16}{1} + \binom{16}{2} + \binom{16}{3} + \cdots + \binom{16}{16} = 2^{16} - 1 = 65535$ different 2:1 balanced adaptive grids. Figure 5 demonstrates that in many cases, adaptive grids can generate an explosion of partitions too. By iterating through the grids in the second depth level, a total of $4(2^1 - 1) + (4(2^4 - 1) + 2(2^2 - 1)) + 4(2^8 - 1) + (2^{16} - 1) = 66625$ adaptive grids exist in the third depth level. Efficiency of the brute-force algorithm is therefore key towards discovering surface-to-volume ratios of partitions within structures and is addressed in Section III.D.

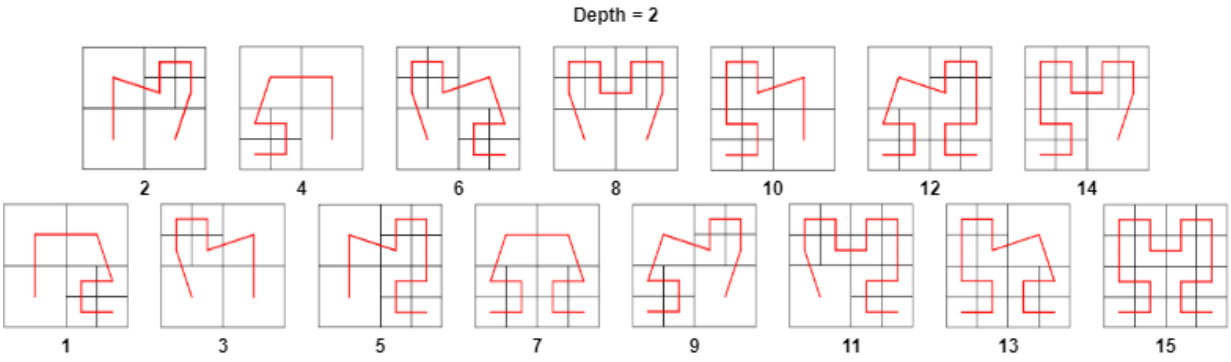


Figure 6: All adaptive grids that exist in the second depth level.

C Measuring the surface-to-volume ratio of a partition

To calculate the average surface-to-volume ratio of Hilbert curve partitions on a depth level basis, we consider all possible partitions from every adaptive grid in each depth level. By traversing the search tree described in Section III.A in a breadth-first manner, we can execute and gather results from each depth level sequentially.

To calculate the average surface-to-volume ratio of an individual grid structure, every possible partition of the Hilbert curve is inspected.

Theorem III.1. *Given an adaptive grid G consisting of c cells, the total number of possible partitions induced by a space-filling curve is*

$$\binom{c+1}{2} = \frac{c(c+1)}{2} \tag{7}$$

Proof. Consider a space-filling curve discretized by c cells on an adaptive grid. The total number of possible partitions to the problem can be modelled using the stars and bars scheme. We represent the c cells by c adjacent stars and consider inserting two bars to determine the volume v of the partition and the position of the partition x_1, x_2 within the grid. This gives us $v+x_1+x_2 = c$ satisfying $v \geq 1$. Thus, with $c-1$ stars and 2 bars, the stars and bars scheme gives us a total of $\binom{(c-1)+2}{2} = \binom{c+1}{2}$ partition combinations. □

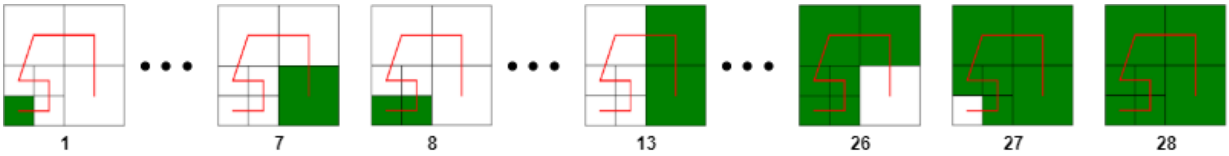


Figure 7: All the possible partitions induced by the Hilbert curve on an adaptive grid consisting of 7 cells.

The proposed algorithm considers the volume of a partition to be the number of cells within the partition and the surface to be the total length of the volume's edges. As intended, the bounding box therefore contributes towards the surface's value. For example, the surface of partition 8 in Figure 7 is 6 and its volume is 2. By dividing the surface of each partition within a grid by its respective volume and taking the mean, an average surface-to-volume ratio is made for each grid structure. The surface-to-volume ratio of partition 8 in Figure 7 is therefore $\frac{6}{2} = 3$. In cases where a cell in a partition has an edge neighbouring two cells e.g. partition 7 in Figure 7, the length is set to 1, making its total surface length 4.

D Exploiting the similarities between partitions

Evidently, brute-forcing through all the possible partitions within each depth level of the search space is a computationally expensive task. With the aim to reduce computational time of the brute-force algorithm, the major contribution of this work is in the exploitation of similarities between the various partitions in the search space. Whenever the surface of a partition is calculated, the result is stored in a pattern dictionary. This pattern dictionary is a nested dictionary that iterates between depths and non-terminal keys, storing the values of surfaces after non-terminal keys. An example structure of the pattern dictionary can be found in Figure 8.

In many cases, the surface-to-volume ratio of different partitions are equal. For example,

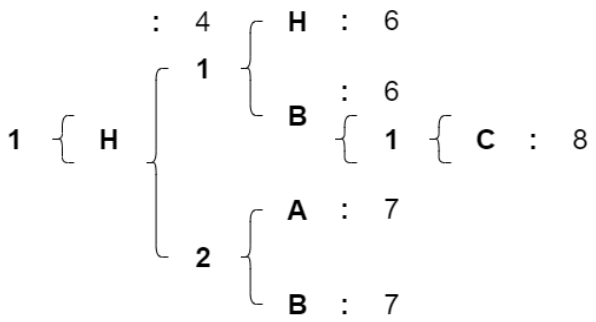


Figure 8: Example structure of the pattern dictionary.

partitions A and B in Figure 9 both output a ratio of $\frac{6}{3} = 2$ and partitions C and D both output a ratio of $\frac{8}{3}$.

Before measuring a partition through brute-force, the pattern of non-terminals corresponding to the partition is mapped such that the first non-terminal in the partition is H. The substitution

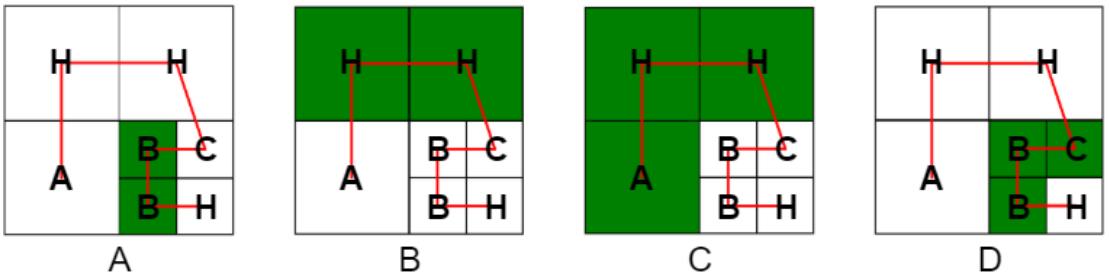


Figure 9: Examples of similarities between different partitions.

of the non-terminals is determined by the first non-terminal ϕ in the path:

$\phi = H$	$\phi = A$	$\phi = B$	$\phi = C$
$H \leftarrow H$	$H \leftarrow A$	$H \leftarrow B$	$H \leftarrow C$
$A \leftarrow A$	$A \leftarrow H$	$A \leftarrow C$	$A \leftarrow B$
$B \leftarrow B$	$B \leftarrow C$	$B \leftarrow H$	$B \leftarrow A$
$C \leftarrow C$	$C \leftarrow B$	$C \leftarrow A$	$C \leftarrow H$

The list of depths corresponding to the partition is also adjusted dependent on the lowest depth of any cell. If the lowest depth within the partition is greater than one, all the depths within the list are scaled up to the point where the lowest depth becomes one. By scaling up and substituting partitions, multiple surface-to-volume ratios of partitions will avoid brute-force calculations as their surface length can already be found in the pattern dictionary. For example, the path in partition D can be replaced by $[2 \rightarrow H \rightarrow 2 \rightarrow A \rightarrow 2 \rightarrow A]$ using the substitution rules and then can be upscaled to become $[1 \rightarrow H \rightarrow 1 \rightarrow A \rightarrow 1 \rightarrow A]$. This path is identical to the edited path of partition C. It is therefore unnecessary to calculate partition D through brute-force as its surface-to-volume ratio can be calculated from the pattern dictionary.

In many cases, entire grids output identical surface-to-volume ratios.

Lemma III.2. *For every grid x in the search tree Ω, such that $x \in \Omega$, the reflection $f(x) = y$ along the central vertical line of x also exists in the search tree; $f(x) \in \Omega$.*

Proof. Consider a grid $x \in \Omega$ and reflection $f(x) = y$ along the central vertical line of x. If $y \notin \Omega$, y must not be a 2:1 balanced adaptive grid. However, reflection is an isometric transformation, meaning the shape and size of y is the same as x. This means grid x is not 2:1 balanced, which is a contradiction as all grids in Ω are 2:1 balanced. □

For symmetrical cases, x and y point to the same grid, whereas for non-symmetrical cases, x and y represent different grids.

Theorem III.3. *Consider the Hilbert curve x' mapped onto an adaptive grid $x \in \Omega$ and the reflection $f(x) = y$ along the central vertical line of x. The Hilbert curve y' mapped onto y is also a reflection of x'.*

Proof. The refinement of a cell in grid x does not alter the path of the Hilbert curve outside of the cell to be refined (see production rules in Section III.A). Let g_1 be an adaptive grid made up of the left section of grid x and the right section of grid y. Let g_2 be an adaptive grid made up of the left section of grid y and the right section of grid x. The mapped path of the Hilbert curve

in grid g_1 is therefore identical to the paths in the left and right sections of the grids x and y. Likewise, this applies for g_2. If the Hilbert curves mapped onto grids g_1 and g_2 are symmetrical along the central vertical line for all grids g_1 and g_2, x' and y' must always be reflections of one another.

Since grids x and y are reflections of one another, grids g_1 and g_2 are both symmetrical along the central vertical line. We must therefore prove that the mapping of the Hilbert curve g' is symmetric for all grids symmetrical along the central vertical line:

Consider the basic patterns H, A, B, C of the Hilbert curve (Bader 2012, p32). H and C are both symmetrical, along with their refinements. Comparatively, A and B are reflections of one another, along with their respective refinements. For an adaptive grid to be symmetrical, refinements of H must be mirrored by a refinement of H in its reflective position, refinements of C must be mirrored by a refinement of C in its reflective position and refinements of A and B must be mirrored by a refinement of the other in their reflective positions.

As all grids in the search tree stem from the initial grid structure, we consider the initial mapping of the Hilbert curve. Consisting of 4 cells labelled $[A, H, H, B]$, the initial Hilbert curve is symmetric. Here, H is mirrored by H and A is mirrored by B. Hence, the path of the Hilbert curve following any number of refinements on one side of the grid can be mirrored by making the same reflective refinements on the other. For g' to be non-symmetric, grid x must therefore be non-symmetric. However, as grids x and y are reflections of one another, grids g_1 and g_2 are both symmetrical; a contradiction. □

Importantly, Theorem III.3 proves that all partitions existing in grid x also exist in grid y, and all partitions existing in grid y exist in grid x. The surface-to-volume ratios for any pair of grids that are reflections of one another along the central vertical line are therefore identical. Note that this property does not hold for reflections along the central horizontal line, as shown by the second iteration of the Hilbert curve on a regular grid.

E Iterating through the search space

The reflective properties of the Hilbert curve mentioned in Section III.D are exploited during the grid generation process. To find all possible refinement combinations for each grid in the previous depth level, we select each curve from the depth level above in turn. All the refinement locations $X = [x_1, x_2, ..., x_n]$ for each grid are passed in from the previous iteration. An iterator i is set to iterate from 1 to n. In general, all possible $\binom{n}{i}$ refinement location combinations are appended to the list of refinements next to be processed. Figure 10 presents the order in which the combinations are processed.

To ensure the search algorithm does not traverse the reflections of grids, we treat symmetrical grids from the previous depth level as a special case. Instead of iterating through the $\binom{n}{i}$ combinations, the refinement locations $X_l, X_r \in X$ are split by the central vertical line and an iterator r is set to iterate from 0 to $\frac{i}{2}$. To avoid traversing the reflections of the non-symmetric of grids, this iterator is used to limit the maximum number of refinements on the left of the grid. For each c_r in $\binom{n}{i-r}$ refinement location combinations selected from X_r, we iterate through $\binom{n}{r}$ refinement location combinations c_l selected from X_l. We add all pairings (c_l, c_r) to the list of refinements until the combinations c_l and c_r mirror each other in the grid. Once found, we also add this symmetrical refinement to the list of refinements and increment c_r. All refinements in the list of refinements that produce non-symmetric grids are tagged to have their results duplicated.

By limiting the maximum number of cells on the left to the number of cells selected on the right, reflections of all grids that have more refinements on the right than on the left will not be reproduced in the search. This is intended, as if refinements are not evenly balanced around the midpoint, the grid cannot be symmetrical. Figure 10 shows an example of the iteration process. If i is not incremented upon the discovery of the symmetrical refinement, selection '27' becomes a reflection of selection 25, selection '28' becomes a reflection of selection 24 etc.

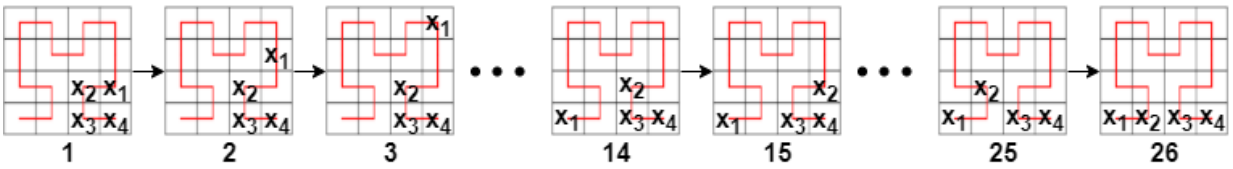


Figure 10: The order in which $\binom{n}{k}$ refinement combinations are iterated through over an example grid structure, consisting of $n = 16$ cells and the selection k = 4.

F Processing the surface-to-volume ratios of refined adaptive grids

For each refinement combination that exists in the list of refinements, we create a curve object that inherits the properties of the curve we are refining. To refine and calculate the surface-to-volume ratios of partitions within the grid structure, we apply the following process:

1. Constructing a new path for the Hilbert curve.

   As described in Section III.A, Hilbert curves on adaptive grids can be produced from a list of depths and a list of non-terminals each representing the basic pattern of a cell's next refinement. By inheriting the production and depth lists from the grid we are refining, we are able to list slice and insert new production rules in these lists where necessary.

2. Determining the cell neighbourhood.

   We also inherit the cell neighbourhood from the grid we are refining. This dictionary stores the neighbouring relations of every cell in the grid structure. Based on the location of the cells we are refining, we shift keys and values in this dictionary where necessary. Using the production and depths list we generate in step 1, we find the new neighbours of all the refined cells in the grid structure. A visual demonstration of this process can be found in Figure 11.

3. Iterating through all partitions induced by the Hilbert curve.

   Having updated the path of the Hilbert curve and neighbour relations in the neighbourhood dictionary, we are able to identify all partitions induced by the Hilbert curve on the grid structure and compute surface-to-volume ratios through brute-force for each partition if necessary. We use a ratio list to store all surface-to-volume ratios within the grid structure. Following the surface-to-volume measurements specified in Section III.C, it is trivial that the surface-to-volume ratio is always 4 for partitions consisting one cell. Hence, we initialise the ratio list with n values of 4, where n is the number of cells within the grid.

To iterate through the remaining partitions within the grid structure, we loop through all possible starting positions for the partition $s = 0, 1..., n$ and consider volumes $v = 2, ..., (n - s)$ for each starting position. We initialise the depths and path of the partition to the depth level of the cell in the starting position and H. Upon incrementing v, we apply the substitution rules we describe in Section III.D to the cell we are appending to the partition and upscale the depths if necessary.

4. Calculating the surface-to-volume ratios.

   Upon substituting the path and depths of the pattern, we search for this partition in the pattern dictionary. If found, we use the surface stored in the pattern dictionary to calculate and add the surface-to-volume ratio of the partition to the ratio list. Otherwise, we compute the surface-to-volume ratio of the partition using the neighbourhood dictionary. Instead of computing the surface length from scratch, we use the surface length from the previous partition or 4 if $v = 2$. Next, we lower this value based on the number of surfaces in the previous partition that are connected to the cell we are appending to the partition. This value is then raised based on the neighbours the cell has that are not in the partition, which gives us the total surface length of the new partition. We store the surface length in the pattern dictionary and the surface-to-volume ratio in the ratio list.

5. Finding the refinable cells in the grid structure.

   We also use the neighbourhood dictionary to identify the locations of all refinable cells that can generate new 2:1 balanced adaptive grids. Cells that are of maximum depth in the current depth level and are neighboured by cells only of the same depth are listed as refinable. For grid generation purposes mentioned in Section III.E, the refinable cells are split into two lists relative to which side of the midpoint in the curve they are on. We store these refinable cells along with other properties relevant to future refinements of the grid structure (i.e. production list, depth list, etc) using a curve class, which is in turn written to the hard disk in the form of a Python pickle file to reduce memory consumption. When iterating through the next depth level of the algorithm, we fetch properties of the curves from the various pickle files.

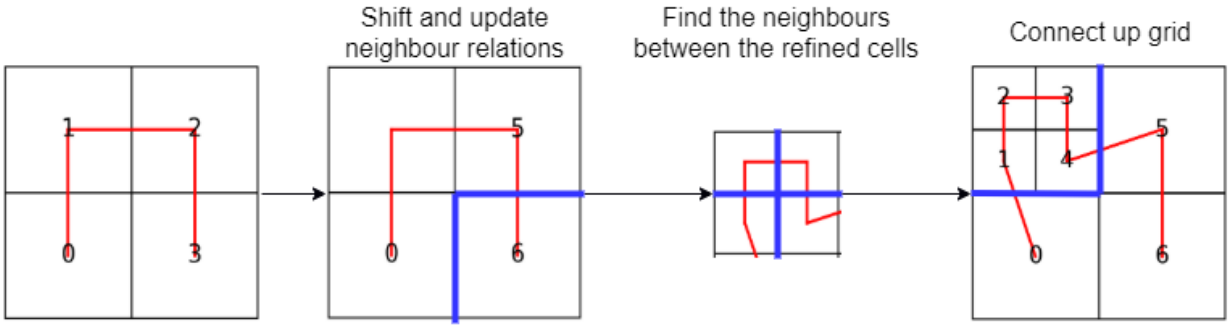


Figure 11: The process used to update the cell neighbourhood upon new refinements.

IV RESULTS

In Section II, we define the optimal parallel efficiency of two-dimensional partitions using Equation 1, introduced by (Zumbusch 2000). In Section III, we model the search space of the problem as a tree-like structure and provide an overview of the brute-force algorithm used to traverse the search tree breadth-first. For each grid structure traversed, the proposed solution computes the surface-to-volume ratios for all partitions within the grid. With (Zumbusch 2000) proving the quality of partitions induced by space-filling curves on regular grids to be of optimal parallel efficiency, we run the proposed solution over a multitude of grids in the search tree and compare the surface-to-volume ratios with Zumbusch's finding. We expect the quality of partitions induced by the Hilbert curve on two-dimensional 2:1 balanced adaptive grids to have an upper bound and to be similar to the surface-to-volume ratios of regular grid partitions.

The brute-force solution is implemented in Python 3. All experiments were performed on a Lenovo V110-15ISK, which features 64 bit Intel Core i5 (6th Gen) 6220U dual-core processing units with L1, L2 and L3 cache sizes of 128KB, 512KB and 3.0MB and 4GB DDR4-2133 memory.

A Surface-to-volume ratios of adaptive grid partitions

Having run the proposed solution over the first three depth levels, a total of 1,161,696 surface-to-volume ratios were computed out of a possible 55,271,264 partitions. Of which, 33,452 different grids in the search tree were inspected. The surface-to-volume ratios computed on these 33,452 grids represent the 66,641 well-balanced adaptive grids that exist with the first three depth levels of the search tree (Section III.B). Each of the 33,452 different grids are stored in individual pickle files and take up a total storage space of 32.3 MB.

The minimum and maximum values within each depth level of Table 1 represent the grids

Table 1: Surface-to-volume ratio metrics.

	Surface-to-volume ratio			
Depth	Minimum	Maximum	Average Grid	Average Partition
1	3.23333	3.23333	3.23333	3.23333
2	2.24764	2.88019	2.60218	2.51172
3	1.38314	2.66020	1.68171	1.65006

structures holding the minimum and maximum surface-to-volume ratio average. Ratios in the 'average grid' column represent the average surface-to-volume ratio of all grid structures within a depth level and is an average of averages. Comparatively, the 'average partition' column in Table 1 represents the average surface-to-volume ratio across all partitions within each depth level.

For all metrics, the values decrease per depth level. Assuming this trend continues, the quality of partitions the Hilbert curve provides on adaptive grids improves as the problem increases in size, similar to space-filling partitions on regular grids (Griebel & Zumbusch 2001). By plotting the average surface-to-volume ratio (represented by the blue dots) for every grid structure in the first three depth levels of the search tree (Figure 12a), we witness a strong negative correlation between the surface-to-volume ratio and the depth level the grid features in. Additionally,

Figure 12b shows a monotonically decreasing relationship between the worst-case surface-to-volume average and the number of cells within the grid structure.

Due to the exponential time complexity of iterating through each depth level, we pruned

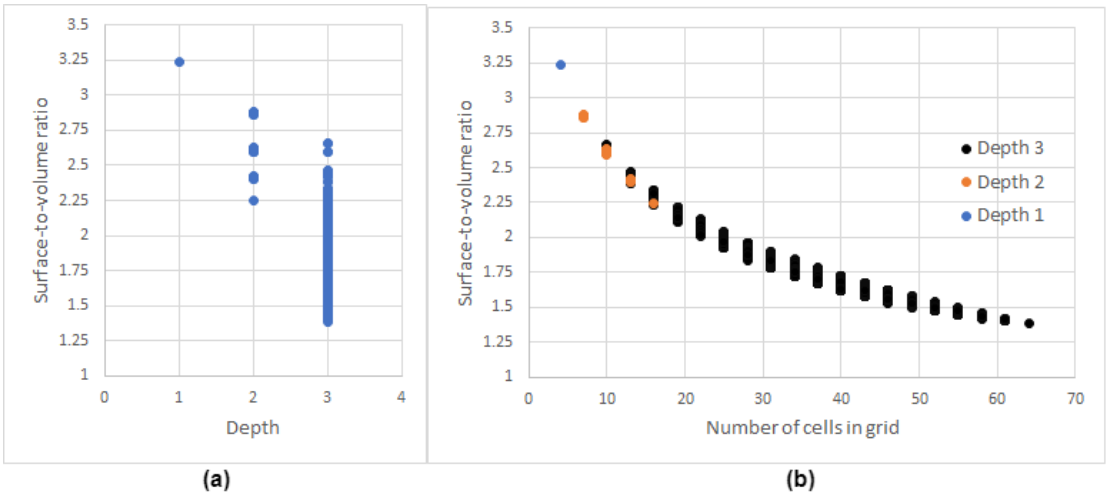


Figure 12: Surface-to-volume ratios induced by the Hilbert curve for all adaptive grid structures in the first three depth levels.

the search tree to gather samples of surface-to-volume ratios in deeper depth levels. Figure 13 presents the results for three different pruned traversal paths; Traversal A and Traversal B are the recursive processes of refining the bottom-leftmost cell and the upper-leftmost cell within each grid structure, starting from the initial grid. The traversal paths can be visualised as the left-most items in the search tree from Figure 5. Traversal C is the traversal of all the adaptive grids produced from the second to the fifth depth level stemming from grid 5 in Figure 6. Overall, results from Figure 12 and Figure 13 indicate the existence of an upper bound for the quality of partitions induced by space-filling curves on 2:1 balanced adaptive grids.

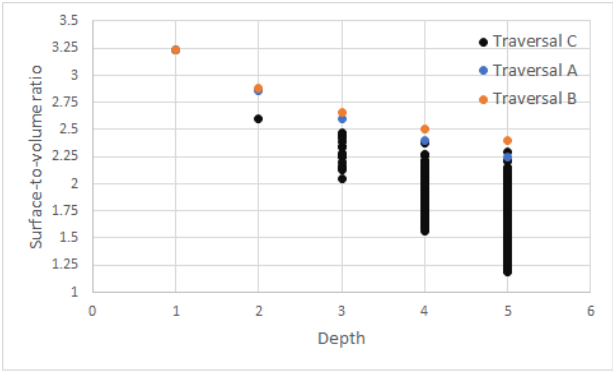


Figure 13: Surface-to-volume ratios induced by the Hilbert curve on adaptive grid structures following traversal paths pruned from the search tree.

B Quality of adaptive grid partitions

To compare the results with Equation 1, the upper bound for the quality of partitions induced by space-filling curves on regular grids proven by (Zumbusch 2000), we consider the maximum and average surface-to-volume ratio of partitions for all depth levels. Results from Figure 14a suggest that the worst-case partitions for all depth levels are of $s = 3v + 1$. This partition exists in the initial grid and exists in the recursive refinement of the upper-leftmost cell from previous depth level. The first v cells within each refined grid are then partitioned, where v is the depth level the grid structure exists in. Figure 15 presents the worst-case partitions for $v = 1, 2, 3, 4$.

On the contrary, the average surface-to-volume ratios of partitions induced by Hilbert curve appears to be of optimal parallel efficiency. For the first three depth levels, the averages are bounded by $\sqrt{v}$ where the supremum for C_{part} is 5.02 to two decimal places (Figure 14b).

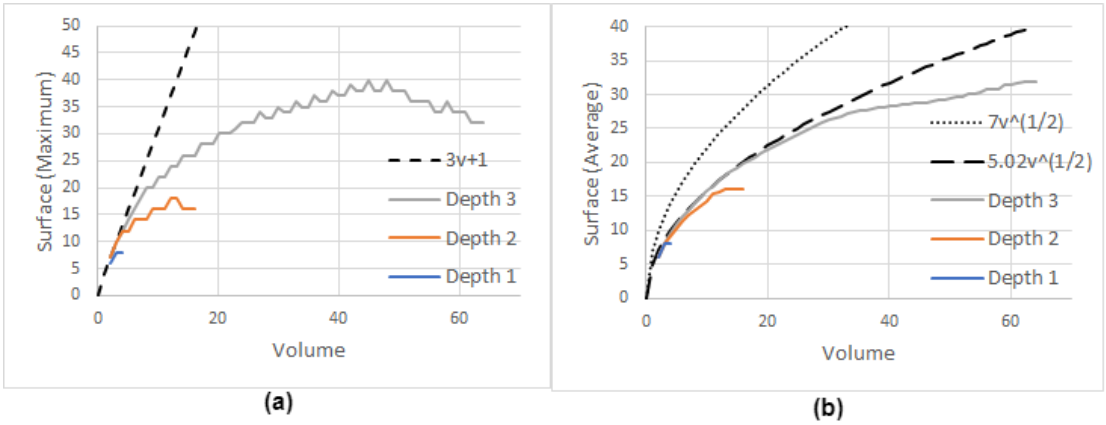


Figure 14: The quality of individual partitions induced by the Hilbert curve on adaptive grids.

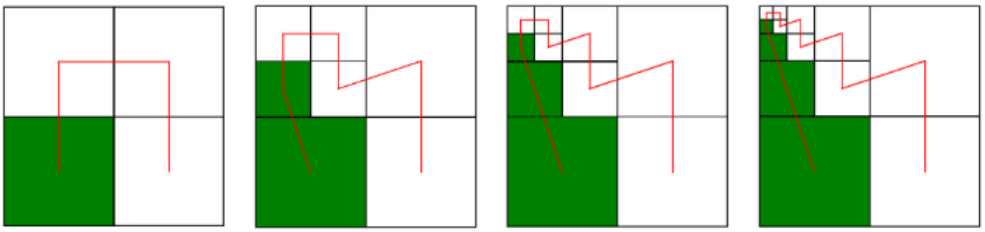


Figure 15: Worst-case partitions that exist within each depth level ($s = 3v + 1$).

C Time complexity of the brute-force algorithm

Without the use of the pattern dictionary in Section III.D, the time complexity of refining the previous and processing surface-to-volume ratios for all partitions within a grid structure is O(n^3). This is because grids consisting of more cells contain more partitions which are of larger sizes. Whereas the refinement of cells has a time complexity of O(n^2), the multiplication of Theorem III.1 with the linear time complexity of iterating through the cell neighbourhoods gives the computation of surface-to-volume ratios a time complexity of O(n^3). Figure 16 shows the

polynomial relationship between the number of cells within each grid structures and computation time. Although runtime is reduced with the introduction of the pattern dictionary, many partitions within grid structures require brute-force calculations. As a result, the total computation time for calculating the surface-to-volume ratios across partitions within the first three depth levels of the search tree took 7 minutes 30 seconds. Of which, 7 minutes 8 seconds were spent refining and processing surface-to-volume ratios for all partitions within a grid structure.

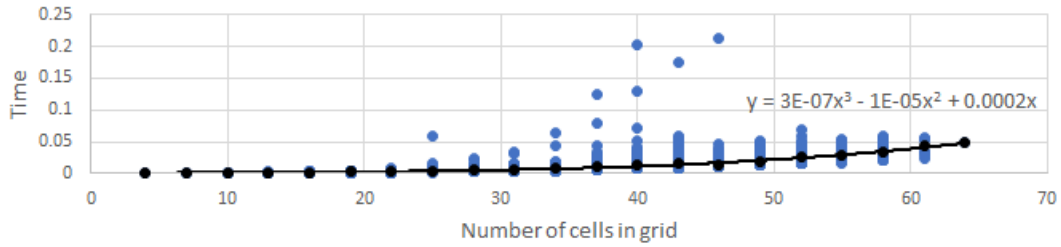


Figure 16: The computational time for calculating all surface-to-volume ratios of partitions within grid structures.

D Validation

To validate the number of grids we iterate through represents total number of grids that exist in each depth level, a quick check on the number of symmetrical cases can be made. Following the grid construction steps in Section III.E, all non-symmetric grids represent two grids. By duplicating all non-symmetrical cases and taking the sum away from the total number of grids traversed, $(33,453 \cdot 2) - 66,641 = 265$ symmetric grids exist within the first three depth levels. In the first and second depth levels, there are a total of 4 symmetrical grids (the initial grid and items 7, 8, 15 in Figure 6). To produce symmetric grids in the third depth level, the grids must be refinements of symmetric grids in the second depth level. As any refinement on the right of the midpoint in the curve can be mirrored by a refinement to the left in symmetric grids, items 7 and 8 in Figure 6 are each able to produce $2^2 - 1 = 3$ symmetrical grids and item 15 is able to produce $2^8 - 1 = 255$. The total number of symmetrical grids in the first three depth levels is therefore $255 + (2 \cdot 3) + 4 = 265$, meaning all the necessary symmetric (and non-symmetric) grids have been considered.

V EVALUATION

With the first working implementation of a basic brute-force algorithm taking over two days to execute the first three depth levels, drastic improvements to the efficiency of the code were made. By exploiting reflective properties of the Hilbert curve in conjunction with the inheritance of cell neighbourhoods from the previous grid structures, we reduced the runtime to 30 minutes 14 seconds. By introducing the pattern dictionary in Section III.D, the final runtime of our brute-force algorithm took 7 minutes 30 seconds to compute results for the first three depth levels.

A Strengths and limitations

The objective of this project is to assess the quality of partitions induced by the Hilbert curve. Having computed the surface-to-volume ratios of all partitions across the first three depth levels in sufficient time, our brute-force algorithm obtains enough data to identify trends between the surface-to-volume ratios we collect. We find that the worst-case adaptive grid partitions are of the form $s = 3v + 1$ and find that our averages comply with Equation 1. Ideally, to confirm that the average quality of partitions is of optimal parallel efficiency, traversing an additional depth level would be appropriate. Although our algorithm successfully finds the supremum for C_{part} to be 5.02 across the first three depth levels, it is unlikely that this upper bound holds for higher depth levels. With a trajectory more similar to C_{part} as 7 (Figure 14), results from an additional two depth levels could estimate tighter bounds for C_{part}, for surface-to-volume ratios across all depth levels.

A lower bound for the number of grid structures existing within each depth level can be computed by considering all refinements of the regular grid from the previous depth level (Table 2). By introducing the reflective properties of the Hilbert curve in Section III.D, our solution traverses approximately half of the grids within each depth level. Whilst providing a significant speedup across the first few depth levels, its effectiveness diminishes in greater depth levels. No matter how computationally efficient the algorithm is at calculating surface-to-volume ratios within a grid structure, our brute-force algorithm traverses over 2^{63} grid structures to assess the quality of partitions within the fourth depth level.

Overall, our solution is therefore suitable for identifying that the average quality of partitions induced by the Hilbert curve on 2:1 balanced adaptive grids are of optimal parallel efficiency. However it is unsuitable for deriving tight bounds for C_{part} in Equation 1.

Table 2: The number of refinement combinations existing within regular grid structures

Depth of regular grid	Cells within regular grid	Number of refinement combinations
1	4	$2^1 - 1 = 1$
2	16	$2^4 - 1 = 15$
3	64	$2^{16} - 1 = 65535$
4	256	$2^{64} - 1 = 18446744073709551616$

B Approach

From the beginning of the project, it was decided that a brute-force approach would be used to measure the surface-to-volume ratios of partitions induced by space-filling curves. By using the spiral model, we were able to implement a basic breadth-first traversal algorithm that computes surface-to-volume ratios in a brute-force manner early in the project life cycle. Having eliminated the possibility of an incomplete solution, our aim shifted towards reducing the time spent computing surface-to-volume ratios for all partitions in each depth level. The risk-driven process behind the spiral model allows us to set new objectives that contribute towards reducing the runtime of the algorithm whilst keeping our global aims constant. Upon determining an objective such as introducing a pattern dictionary, we identify the sections of the previous prototype

that need changing before developing and testing the new additions we make to the algorithm. Through testing, we can assess whether the algorithms function correctly, whether the additions are effective and identify new objectives for the next iteration of the spiral model.

VI CONCLUSIONS

In this project, we propose a brute-force algorithm that computes surface-to-volume ratios of all partitions within adaptive grid structures on a depth level basis. All adaptive grid structures within the search space are 2:1 balanced and two-dimensional. To improve the efficiency of the brute-force algorithm, we exploit the similarities between the various partitions induced by the Hilbert curve on both the same and different grid structures. Reflective properties of the Hilbert curve are used to approximately halve the total number of grids traversed by the algorithm for each depth level. Whenever the surface length of a partition is calculated through brute-force, its value is stored in a pattern dictionary. Prior to calculating the surface-to-volume ratio of any partition, the partition is substituted as described in Section III.D and searched for in the pattern dictionary. If found, we can bypass brute-force process for the given partition.

Having generated surface-to-volume ratios for all grids existing within the first three depth levels in under 8 minutes, we compare our results with the quality of partitions induced by regular grids. Introduced by (Zumbusch 2000), we use Equation 1 to evaluate the parallel efficiency for partitions induced by the Hilbert curve on two-dimensional 2:1 balanced adaptive grids. We find that quality of partitions are bounded by $3v + 1$ and that average surface-to-volume ratios amongst the first three depth levels infer optimal parallel efficiency, with the supremum of C_{part} equal to 5.02.

In general, we modelled the problem with lots of constraints. Our brute-force algorithm assesses the quality of partitions induced by the Hilbert curve on two-dimensional 2:1 balanced adaptive grids. To see if the quality of partitions induced by other space-filling curves are of optimal parallel efficiency, a reimplementation of the algorithm featuring a new grammar would need to be introduced. Future work may also involve the investigation of new methods for exploiting properties of space-filling curves and may possibly need to deviate from a brute-force approach to retrieve data for all surface-to-volume ratios of greater depth levels.

References

Bader, M. (2012), *Space-filling curves: an introduction with applications in scientific computing*, Vol. 9, Springer Science & Business Media.

Bank, R. E., Sherman, A. H. & Weiser, A. (1983), 'Some refinement algorithms and data structures for regular local mesh refinement', *Scientific Computing, Applications of Mathematics and Computing to the Physical Sciences* **1**, 3–17.

Burstedde, C., Wilcox, L. C. & Ghattas, O. (2011), 'p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees', *SIAM Journal on Scientific Computing* **33**(3), 1103–1133.

Corcoran, T., Zamora-Resendiz, R., Liu, X. & Crivelli, S. (2018), 'A spatial mapping algorithm with applications in deep learning-based structure classification', *arXiv preprint arXiv:1802.02532* .

Drui, F., Fikl, A., Kestener, P., Kokh, S., Larat, A., Le Chenadec, V. & Massot, M. (2016), 'Experimenting with the p4est library for amr simulations of two-phase flows', *ESAIM: Proceedings and Surveys* **53**, 232–247.

Gotsman, C. & Lindenbaum, M. (1996), 'On the metric properties of discrete space-filling curves', *IEEE Transactions on Image Processing* **5**(5), 794–797.

Griebel, M. & Zumbusch, G. (2001), Hash based adaptive parallel multilevel methods with space-filling curves, *in* 'NIC Symposium', Vol. 9, pp. 479–492.

Hilbert, D. (1891), 'Über die stetige abbildung einer linie aufein flächenstück', *Mathematische Annalen* **38**, 459–460.

Liang, J.-Y., Chen, C.-S., Huang, C.-H. & Liu, L. (2008), 'Lossless compression of medical images using hilbert space-filling curves', *Computerized Medical Imaging and Graphics* **32**(3), 174–182.

Peano, G. (1890), 'Sur une courbe, qui remplit toute une aire plane', *Mathematische Annalen* **36**(1), 157–160.

Psomadaki, S., Van Oosterom, P., Tijssen, T. & Baart, F. (2016), 'Using a space filling curve approach for the management of dynamic point clouds', *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, IV-2 W* **1**, 107–118.

Quinqueton, J. & Berthod, M. (1981), 'A locally adaptive peano scanning algorithm', *IEEE Transactions on Pattern Analysis and Machine Intelligence* (4), 403–412.

Reinarz, A., Charrier, D. E., Bader, M., Bovard, L., Dumbser, M., Duru, K., Fambri, F., Gabriel, A.-A., Gallard, J.-M., Köppel, S. et al. (2020), 'Exahype: an engine for parallel dynamically adaptive simulations of wave problems', *Computer Physics Communications* p. 107251.

Rudi, J., Malossi, A. C. I., Isaac, T., Stadler, G., Gurnis, M., Staar, P. W., Ineichen, Y., Bekas, C., Curioni, A. & Ghattas, O. (2015), An extreme-scale implicit solver for complex pdes: highly heterogeneous flow in earth's mantle, *in* 'Proceedings of the international conference for high performance computing, networking, storage and analysis', pp. 1–12.

Sundar, H., Sampath, R. S. & Biros, G. (2008), 'Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel', *SIAM Journal on Scientific Computing* **30**(5), 2675–2708.

Wang, C., Dong, X. & Shu, C.-W. (2015), 'Parallel adaptive mesh refinement method based on weno finite difference scheme for the simulation of multi-dimensional detonation', *Journal of Computational Physics* **298**, 161–175.

Weinzierl, T. (2019), 'The peano software—parallel, automaton-based, dynamically adaptive grid traversals', *ACM Transactions on Mathematical Software (TOMS)* **45**(2), 1–41.

Xia, X. & Liang, Q. (2016), 'A gpu-accelerated smoothed particle hydrodynamics (sph) model for the shallow water equations', *Environmental Modelling & Software* **75**, 28–43.

Zumbusch, G. (2000), *On the quality of space-filling curve induced partitions*, Sonderforschungsbereich 256.