Estimating the upper bound on arithmetic intensity for a stencil algorithm

Sergey Khilkov
khilkov.s@gmail.com
Hipercone Ltd.
Jerusalem, Israel

Abstract

For a stencil algorithm, arithmetic intensity is closely related to performance. We want to find an upper bound for arithmetic intensity for a given stencil algorithm with the help of geometric inequalities. This article presents a conjecture that for large problems arithmetic intensity I is bound by the cache size D_{cache} to the power of $1/n$, $I \leq C \cdot \sqrt[n]{D_{\text{cache}}}$, where n is the number of space dimensions for the algorithm lattice. The coefficient C depends only on the algorithm parameters and does not depend on the parameters of a computer system.

The conjecture works for every implementation of the algorithm as long as the arithmetic operations stay the same. Therefore it helps to understand the limits of the optimization techniques like polyhedral optimization.

To estimate the bound C for a stencil algorithm, we introduce a geometric locality model. The model works in a continuous vector space and is expected to yield asymptotic estimations of the bound C for large tile sizes. It also produces other interesting results related to tiling validity.

We discuss several ways to generalize the geometric locality model. The space transformation, which makes the stencil convex, is of particular importance, since it may also prove to be useful in the polyhedral model.

Keywords: arithmetic intensity, geometric inequalities, LRnLA, metric embeddings, polyhedral model, stencil algorithms

1 Introduction

Stencil computations represent an important class of data processing solutions. They are common in engineering and scientific codes. For instance, many explicit numerical methods for computational physics may be considered stencil algorithms.

A stencil algorithm is defined by a data array and an update pattern which is called stencil. The algorithm performs the update of each array cell according to the stencil. In this article, we consider the data array to be a uniform grid with n dimensions, and the stencil to be a finite set of shifts in that grid. Using a physics analogy, we say that the grid represents space and update iterations represent time.

IMPACT 2025, Barcelona, Spain

For implementations of stencil algorithms, the performance is often constrained by the memory or cache bandwidth due to insufficient data locality. Performance constraints may be described in the terms of Roofline model [23]. It defines arithmetic intensity as a measure of data locality. The *arithmetic intensity* is the average number of arithmetic operations per byte of traffic to cache or memory. We want to emphasize two points about arithmetic intensity. Firstly, it depends on the algorithm implementation and the computation system. Secondly, there is more than one arithmetic intensity for a given algorithm and a machine. We can count the traffic from each level of memory hierarchy, and each level will give us a different intensity value. According to Roofline model, if the problem is memory bound, then the performance is proportional to arithmetic intensity. Hence increasing arithmetic intensity increases the performance.

There is a number of tricks and techniques one can use to increase arithmetic intensity of the code, e.g. temporal blocking and loops skewing. The polyhedral model [6] provides a generalization of most of these techniques in a single mathematical theory. The core of the approach is based on the polyhedral representation, which describes a program as a set of integer polyhedra. Affine transformations for these polyhedra allow one to rearrange computation order to improve some aspects of the program. The affine transformation is found as a result of an integer linear programming problem in which a cost function is optimized. Originally the cost function was related to delays and the goal was to reduce the processor wait time in parallel programs [4, 5]. Nowadays it is far more common to use the cost functions related to data locality [2], which results in improving arithmetic intensity.

In the end, all mentioned techniques increase the arithmetic intensity by changing the order of the arithmetic operations. At this point a question comes to mind: "What are the bounds of arithmetic intensity for a given algorithm? Is there an upper limit?" Answering this question for the case of stencil algorithms is the topic of this paper.

Our idea is based on Locally Recursive non-Locally Asynchronous (LRnLA) algorithms [17] for stencil computations. The LRnLA algorithms employ geometric approach to tiling creation and allow to create state-of-the-art codes for various problems [14–16, 21, 24]. The approach mainly focuses on data locality and may be applied for GPU, CPU and even

heterogeneous codes. Note that tilings described by LRnLA are similar to tilings produced by polyhedral optimizers.

Geometric approach allows us to realize that the number of data transfers between tiles behaves like the tile surface area while the number of arithmetic operations corresponds to the tile volume. Then arithmetic intensity becomes the volume to area ratio. The limit of the volume to area ratio in a metric space is given by isoperimetric inequality [20]. For Euclidean space the inequality saturates only for a ball, which proves that ball has largest volume for a given surface area. The space our algorithm lives in is not Euclidean, but we can use similar geometric inequalities to estimate the upper bound on arithmetic intensity.

In this article we outline the approach. Rigorous mathematical proofs will be presented in followup articles.

The rest of the paper is structured as follows. Section 2 explains how geometric inequalities put a constraint on arithmetic intensity. Section 3 outlines a geometric locality model for stencil algorithms and how it helps to obtain the intensity bound. The model in Section 3 imposes restrictions on the stencil. Ways to lift these restrictions are discussed in the Section 4. Section 5 specifies how one can apply results of geometric locality model in polyhedral model. Related works and discussion presented in the Section 6. The last Section 7 contain results and conclusions.

2 Geometric inequalities and arithmetic intensity

In this section we discuss the relations between arithmetic intensity and some geometric inequalities. For arithmetic intensity we count only data loads. The writes may be counted in a similar way. We also assume that the grid of the stencil algorithm is large enough to neglect effects caused by the grid boundaries.

The data layout is deliberately left unspecified. It allows us to apply the estimation results to a broader variety of implementations, but makes it hard to count any cache misses except capacity misses. For the same reason cache is assumed to be fully–associative.

We divide the explanation into two parts. In the first we use a simplified cache model and in the second we consider a more realistic one. Both cache models have large memory and a single level of cache.

2.1 Simplified cache model

Consider an implementation of a stencil algorithm which has operations grouped into a tile. Each tile calculation is an atomic operation. In our simplified model, cache simultaneously holds all the data a tile needs during calculations, and discards all the data before loading a new tile.

Discarding the loaded data in our model may be explained if the next tile for a thread is always far from the current. It may happen in a sufficiently large OpenMP “parallel for” loop of tiles and thread affinity set to “close”, processed by a machine with a large number of computational units and no cache shared between the units.

We also use an example problem to illustrate our approach.

Example 2.1. The dependence graph for the example is given on the Figure 1. In this example stencil corresponds to the three arrows going into a point. If we describe it in terms of dependence vectors, stencil becomes a set of three points $S = \{(-1,-1),(0,-2),(1,-1)\}$. For convenience, we further assume that every stencil also contains vector $(0,0)$ for the point we’re updating.

The best solution for the Example 2.1 is given by a well known diamond tiling. You can find an experimental comparison of this technique to some others at Orozco et al. [19]. So the question before us is, “How to prove the diamond tiling optimality from a theoretical standpoint?”

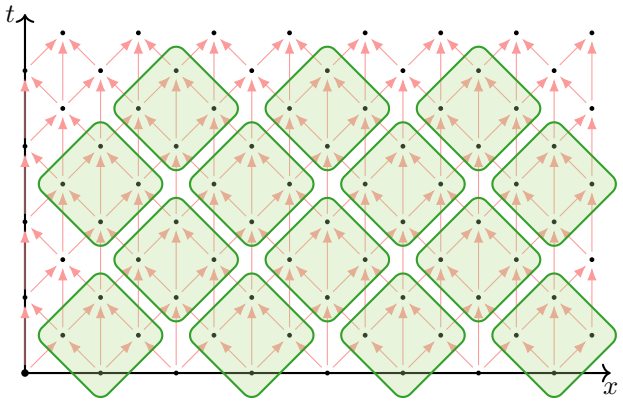


Figure 1. The dependence graph for a finite volume method in 1D which is decomposed with the diamond tiling. Arrows represent data transfers. Points correspond to arithmetic operations. For simplicity, each arrow transfers D_{scheme} bytes of data and each point corresponds to O_{scheme} arithmetic operations.

2.1.1 Arithmetic intensity in geometric terms.

To calculate dependencies of a tile, it is convenient to use Minkowski sum.

Definition 2.2. Let A and B be arbitrary sets of vectors. The set composed of the sums $a+b$ for all the pairs where $a \in A$ and $b \in B$ is called Minkowski sum $A+B$ of sets A and B, $A+B = \{a+b | a \in A, b \in B\}$.

If T is the set of lattice points in a tile, then the set of its outgoing dependencies is given by $(T+S) \setminus T$. Note that $T \subset T+S$ because we added zero vector into a stencil. Let us assume that tile data can fit into the cache. The simplified cache model ensures that load from memory is performed only for the loads across the tile border. All loads within the tile are cached. Then cardinality $|T|$ gives us the number of

points in a tile. Multiplying it by the number of arithmetic operations O_{scheme} corresponding to a single point we get the number of arithmetic operations corresponding to the tile T. Each tile requires $|(T+S)\setminus T| = |T+S| - |T|$ data loads, which, multiplied by the data size of a single load D_{scheme}, gives us total size of data loads for the tile T. Since in this section we are counting only data load parts of the arithmetic intensity I we can calculate it with the formula

$$I = I_{\text{scheme}} \frac{|T|}{|T+S| - |T|}, \tag{1}$$

where $I_{\text{scheme}} = O_{\text{scheme}}/D_{\text{scheme}}$.

Note that every data point is used more than once. If it is loaded from memory we count it only once assuming that all subsequent loads are cached. Hence we calculate only the data loads form memory which are absolutely necessary. The real size of the memory loads may be larger which would lead to a lower arithmetic intensity.

To get the intensity bound, we need to use an inequality that allows us to estimate $|T+S|$. Since we are describing a discrete case, the inequality has to be a discrete one.

2.1.2 Discrete geometric inequalities.

We would like to use an analog of the isoperimetric inequality for the discrete case. The isoperimetric inequality is derived from Brunn–Minkowski inequality. The derivation may be found in [20]. Fortunately Brunn–Minkowski inequality has lattice versions [8].

Brunn-Minkowski inequalities. Before going to the discrete case, we describe the classic version of the Brunn–Minkowski inequality.

Theorem 2.3 (Brunn-Minkowski)**.** *Let $\mathcal{L}$ be the Lebesgue measure in $\mathbb{R}^n$. If A and B are nonempty measurable subsets of $\mathbb{R}^n$, then*

$$\mathcal{L}(A+B)^{\frac{1}{n}} \geq \mathcal{L}(A)^{\frac{1}{n}} + \mathcal{L}(B)^{\frac{1}{n}}. \tag{2}$$

The proof is quite technical and may be found in the textbook [7]. It is important to us that Brunn–Minkowski inequality (2) connects the volume of the Minkowski sum with the volumes of the summands. If we look at the arithmetic intensity formula (1), this is exactly what we need.

Lattice versions of the inequality (2) are usually formulated for finite sets. Hence instead of volumes they use cardinalities of the sets. However the form (2) is not valid for them. For example if one uses a single point as the set B the form (2) would yield $|A|^{\frac{1}{n}} \geq |A|^{\frac{1}{n}} + 1$, which is false.

There are different ways to deal with this problem. Most of them either involve enlarging the set on the left hand side, or appending some terms to the right hand side. In this work we prefer the latter:

$$|A+B|^{\frac{1}{n}} \geq |A|^{\frac{1}{n}} + \frac{1}{n!^{\frac{1}{n}}} (|B|-n)^{\frac{1}{n}}. \tag{3}$$

This inequality was derived in the article [8]. Note that it requires $\dim(B) = n$. This means that the set B should have full dimension. Since a simplex, which is the smallest full dimension set in $\mathbb{R}^n$, has $n+1$ points, we can assume that $|B| - n > 0$.

The upper bound for arithmetic intensity. Let us show how to derive the upper bound for arithmetic intensity using lattice Brunn–Minkowski inequality. We can raise the inequality (3) to the n-th power:

$$|T+S| \geq |T|^{\frac{n}{n}} + n|T|^{\frac{n-1}{n}} \frac{1}{n!^{\frac{1}{n}}} (|S|-n)^{\frac{1}{n}} + \sum_{k=2}^{n} \binom{n}{k} |T|^{n-k} \frac{1}{n!^{\frac{k}{n}}} (|S|-n)^{\frac{k}{n}}. \tag{4}$$

Since all the terms in the right hand side are non-negative we can discard all binomial terms after the second. We leave two leading terms of the expansion in the powers of $|T|$. That's why our estimation will work best for large tiles T. After discarding binomial terms we can construct the arithmetic intensity (1) in the right hand side of the inequality (4):

$$\frac{n!}{n^n} \frac{1}{|S|-n} \geq \frac{|T|^{n-1}}{(|T+S|-|T|)^n} = \frac{1}{|T+S|-|T|} \left(\frac{I}{I_{\text{scheme}}}\right)^{n-1}. \tag{5}$$

This inequality is general enough to give a constraint on arithmetic intensity for any stencil algorithm. However it works only for a simplified cache model, where the tile data is discarded from cache right after the tile has been computed. This cache model holds all tile data in cache simultaneously, which means that cache size is greater than $D_{\text{scheme}}(|T+S|-|T|)$. But for a more realistic cache model we can load data when we need it which makes it possible to use this tiling for smaller cache sizes. We consider a more realistic cache model in the next subsection.

Another point to take away: we can asymptotically compare any uniform tilings by comparing their "isoperimetric" ratios $(|T+S|-|T|)^n/|T|^{n-1}$, and the lower is better.

Let us return to the Example 2.1. The space has only one dimension, but the tiling is in the space–time. Therefore $n = 2$. The stencil S has four points (including zero). Substituting this values into inequality (3) we get

$$\frac{1}{4} \geq \frac{|T|}{(|T+S|-|T|)^2}. \tag{6}$$

The same ratio for the diamond tile with the side k is given by:

$$\frac{k^2}{(2k+1)^2} \xrightarrow[k\to\infty]{} \frac{1}{4} \tag{7}$$

This makes the diamond tiling asymptotically optimal, but only for the simplified cache model.

2.2 More realistic cache model

We got the inequality (5) for the simplified cache model. How does it change for a more realistic one?

Let us start by describing the new cache model. We still have a single cache level and the memory. This time cache is not discarded after a tile calculation. It will be reused if possible. We also don't need to hold all the data required by a tile in the cache, as we can load it one part at a time.

Realistic cache model assumptions make it possible to cache cross tile data transfers. This time instead of tiles we consider a joined sequence of tiles computed by the same thread. We call this union of tiles a tower.

Let us assume that a tower can be divided into sequential chunks with size less than cache size. We also require that every chunk depends only on the previous one. We can consider these chunks to be cache snapshots at a certain point of computations. If we can find such a division, the tower can be calculated in such a way that only cross–boundary tower dependencies are loaded from memory. In this subsection we derive the limitations for such towers.

Example 2.4. Let us consider the dependence graph from Example 2.1 again. On Figure 2 we depict a tower and the chunks of data for a single thread tiling resembling CATS [22].

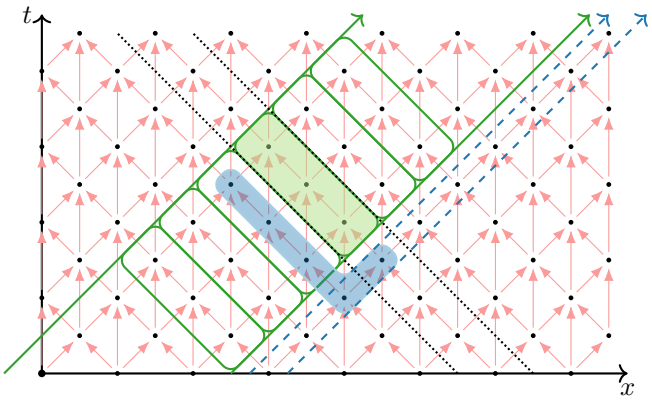


Figure 2. The tower points are between solid slanted arrows. The tower is divided into chunks which are outlined with solid lines. For the shaded chunk, there is a dependence set depicted with thin shadow. The tower dependencies reside between dashed arrows. The cross–section corresponds to the area between dotted lines.

On Figure 2 a single chunk is encircled with a thick line. It corresponds to a cross–section bounded by the dotted lines. Assuming that the chunk size can't be larger then a cache size we can derive the arithmetic intensity for this case:

$$I = \frac{D_{\text{cache}} O_{\text{scheme}}}{{D_{\text{scheme}}}^2}. \tag{8}$$

If we compare it with the inequality (6) we find that our estimation exceeds the previously found boundary. In the Example 2.4 the intensity limit may be estimated using the isoperimetric type inequality in the cross–section rather than the whole space. We will return to this idea after considering the general case.

In all examples, we describe a problem in such a way that loads from memory only occur across the boundary. The general case should not be an exception. Consider a program with a specified order of calculations. If there is a memory load we mark it with a line which crosses the dependence corresponding to a load. Those lines outline a shape similar to a tower in the Example 2.4. This shape is essentially the trace the cache leaves while making its way through the space–time (Fig. 3). As it is seen on the picture, the tower–like shape is technically inside the cache trace, but their borders are quite close. We want to find restrictions which the cache size applies to this shape.

A load from memory occurs when we try to load data which is not currently cached. In addition there are only two ways to put a value in the cache: either calculate a new one or load an already calculated one from memory. To see how the border behaves, let us examine the cache contents at a certain timestamp.

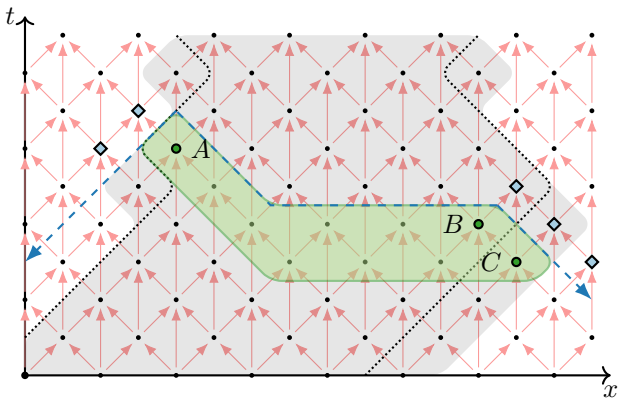


Figure 3. Continuous shape in cache is shaded and outlined with a solid line. Dashed lines represent the dependence cone of the shape. The points A, B and C which are marked with thick circles we denote as the "border points". The "cache trace" example is outlined with a light shade, while memory load boundaries are represented by dotted lines.

We can divide the cache contents into shapes which are continuous in space–time. We will call these shapes continuous cache fragments. Keep in mind that continuous cache fragments are not necessarily sequential in memory. An example of such a shape is given on the Figure 3. The points in the shape which lie on the side border of the dependence cone for a continuous cache fragment shape we call the "border points". These border points reside near the memory load lines we are drawing. At some point in the future when we calculate one of the diamond shaped points (see Fig. 3), we

would move the side borders of the cone and add memory load lines near the newly calculated point. Not all the "border points" are going to be near the memory load lines. But in our 2D example at least one "border point" on each side of the cone should reside near the memory load line.

This reasoning allows us to notice that each continuous cache fragment at every time point touches boundaries of the tower-like shape which describes memory loads. Moreover, in order to avoid additional loads between border points, we need the continuous cache fragment to be thick enough. Note that the required thickness can be described in a similar way to the dependencies of a tile, $|T + S| - |T|$.

In short, the cache fragments which are continuous in a space–time are similar to a cross–section of a tower. And the cache size limits the area of this cross–section. This notion may help to formulate a general form of the bound on the arithmetic intensity.

It is well known that isoperimetric inequality also works on submanifolds and hyperplanes in particular. See for example [20]. The dimension of the inequality in these cases equals to the dimension of the submanifold rather than that of the ambient space. We can assume that similar inequalities may be found for discrete cross-sections. This gives us an evidence for a universal bound on the arithmetic intensity. This bound should limit the arithmetic intensity in every possible implementation.

Conjecture 2.5. *For any stencil algorithm in n-dimensions plus time the arithmetic intensity I is bounded by:*

$$I \le C \cdot \sqrt[n]{D_{\text{cache}}} \tag{9}$$

The constant C here depends on the stencil and algorithm parameters like D_{scheme} and O_{scheme}. However C does not depend on any machine parameters, which makes this constant a property of the algorithm.

Estimation of the constant C for an arbitrary case of the stencil algorithm is an interesting problem. We outline a way to obtain these estimations in the next two sections.

3 Geometric locality model

For the diamond tiling Example 2.1 and the simplified cache model, the resulting inequality (6) saturates only asymptotically for large tile sizes. We believe that asymptotic nature is common for similar inequalities. That's why we consider deriving our geometric inequalities in the asymptotic limit of the big tiles.

From one point of view, increasing the tile size is similar to reducing step size of the lattice. Hence, in the continuous limit the number of points becomes Jordan measure. With some limitations it allows us to use classic Brunn–Minkowski inequality to derive a constraint for the arithmetic intensity.

Let us sketch a plan to find the upper bound on arithmetic intensity for a stencil algorithm:

1. describe a model of continuous tilings,
2. show that every discrete tiling corresponds to at least one continuous tiling,
3. use the continuous model to obtain an intensity limit in the form (9).

Now we take a closer look at each step.

3.1 Continuous dependence model

Please note that from now on "tiling" is used as a synonym to a general tessellation or honeycomb rather than a loop optimization method. Tiles do not have to be of the same shape. Also it does not imply any specific order of calculations for the tiles beyond the dependence relation on them. Tiling only provides a way to split the whole task into smaller ones described by tiles.

What makes a tiling valid? Usually a tiling is called valid if there is an order of calculations in which every dependency is satisfied and each tile may be considered an atomic operation. We call a valid tiling atomic. Then our goal is to describe every atomic tiling.

For continuous tilings, we will assume that lattice points are always lying in the interior of tiles. Hence every point of a discrete lattice has a single tile containing it.

Before discussing atomic tilings any further we need to define dependencies between points and between tiles. There are quite a few ways to represent dependencies in an algorithm. We have a generalized dependence graph [4], a transitive closure of the dependence binary relation [12] and simpler models like dependence vectors [11]. In this article we study stencil algorithms with uniform stencil. Hence we can use a simple representation like dependence vectors. We also don't need to know how operations correspond to program statements. This information is critical for making loop transformations, but for a locality study it is not necessary.

We begin the description of the model from a vector space with a norm. Dependencies of a point are described by a cone Cone_d, which has a metric ball as a base. Note that the relation between a metric ball and the dependence cone Cone_d means that the norm in the vector space depends on the stencil. This representation imposes tight restrictions on the stencil, namely convexity of the stencil and central symmetry of its space part. These restrictions are necessary for a metric ball in a normed space [13]. Since in our representation the stencil essentially corresponds to a metric ball, similar restrictions should be applied to it. We will explain how one can lift these restrictions in the next section.

Dependencies. We can use cones to describe dependencies between tiles (Fig. 4). We will define dependence cone for a tile A as a union of all dependence cones for its points. Dependence cone for a tile A may be represented in terms of a Minkowski sum as $A + \text{Cone}_d$.

Definition 3.1. If the interior of tile B intersects with the dependence cone of tile A, then A depends on B.

Note that another transitive closure is necessary to make tile dependencies transitive (e.g. tile D on Fig. 4).

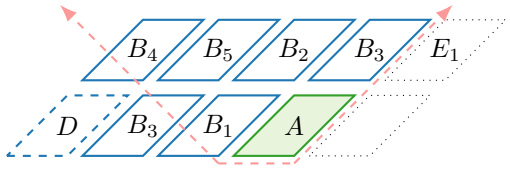


Figure 4. The dashed line depicts the influence cone $A + (-\text{Cone}_d)$ of the tile A, which is opposite to the dependence cone. All B_i tiles depend on tile A. Tile D depends on tile B_3 which depends on A. In order to make dependence transitive D should depend on A. Finally tile E_1 and tile A are independent of each other.

Conoids. For convenience we introduce a new class of shapes.

Definition 3.2 (Conoid)**.** We will call a set A conoid iff $A = \overline{A + \text{Int}\,(-\text{Cone}_d) \cap A + \text{Int}\,(\text{Cone}_d)}$.

Here $\text{Int}\,(X)$ means the interior of the set X and $\overline{X}$ denotes the closure of the set X. Plus corresponds to Minkowski addition, while the unary minus multiplies every vector of a set by minus one.

Figure 5 gives us several examples of conoids. More examples of handcrafted tilings based on conoids may be found in [16] and [17]. Note that conoids are not necessary convex (Fig. 5b) or even connected.

(a) Convex conoid. **(b)** Non-convex conoid.

Figure 5. Examples of conoids. The dependence cone and the influence cone for the depicted conoid are shown with dashed lines.

In an atomic tiling, all dependencies between tiles should be unidirectional. If there are two tiles dependent on each other, then the dependence relation contains a loop. We cannot consider a tile in a loop to be an atomic operation. Hence a tiling with a dependence loop is not atomic.

The requirement for unidirectional dependencies allows us to impose a restriction on the tile shapes in an atomic tiling.

Theorem 3.3. *Every atomic tiling consists of conoids.*

The theorem is a consequence of a fact that for any given shape other than a conoid, there is other tile which simultaneously depends and is dependent on the given tile.

Atomic tilings. Unfortunately not every tiling which consists of conoids is atomic. In order to describe every atomic tiling we can use two additional propositions.

Proposition 3.4. *Tiling with exactly two conoids is atomic. We call such tilings* ***binary****.*

Proposition 3.5. *Intersection of two atomic tilings is atomic.*

With these propositions we can formulate a theorem.

Theorem 3.6. *Tiling is atomic iff it can be produced by an intersection of binary tilings.*

We can provide a sketch of the proof. For the if case propositions 3.4 and 3.5 provide all necessary details. Every binary tiling is atomic, and the intersection of two atomic tilings is atomic. Simple induction proves the case.

For the only if case we need to remember the definition of the atomic tiling. In an atomic tiling all the tiles are computable as atomic operations. Hence there is an order in which these tiles may be computed one by one without breaking any dependence requirements. In this order, for every two tiles, we can tell which is computed earlier.

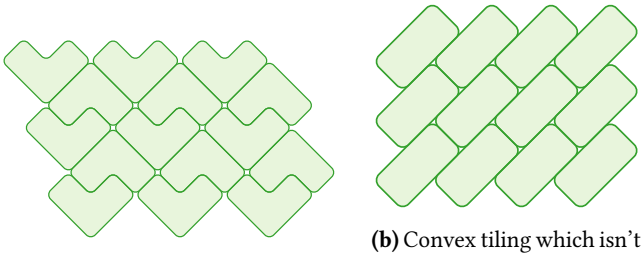
For any tile T we can split the tiling into two sets:

1. $T_<$ contains every tile computed before T,
2. $T_\geq$ contains the tile T and every tile computed after it.

Unions of all tiles in each of these sets represent tiling with exactly two tiles. This tiling with two tiles is atomic since we can compute $T_<$ before $T_\geq$ without breaking any dependence requirements. Finally the atomic tiling consists of conoids, which makes the tiling in question binary.

Intersecting all such splits for all tiles T, we get the initial atomic tiling. This proves the only if case.

The Theorem 3.6 concludes the description of atomic tilings within the geometric locality model. It is worth mentioning that the atomic tilings in this model may include non-edge-to-edge tilings and non-convex tilings (e.g. Fig.6). The latter is particular hard to describe in the polyhedral model.



(a) Edge-to-edge non-convex tiling. **(b)** Convex tiling which isn't edge-to-edge.

Figure 6. Notable atomic tilings examples.

3.2 The correspondence between discrete and continuous atomic tilings

We reached the second step of the plan. Now we need to prove that at least one continuous tiling corresponds to a given discrete tiling.

We lay out the prove in three propositions.

1. There is an atomic tiling such that
   a. each tile in it contains no more than one point of the lattice,
   b. dependencies between the tiles containing lattice points coincide with dependencies between lattice points.
2. It is possible to join tiles in atomic tiling to get another atomic tiling. The set of rules governing such joins is the same as for joining tiles in a valid discrete tiling to get a valid discrete tiling.
3. A valid discrete tiling can be constructed from points using valid joining rules. The same joins may be performed for the atomic tiling in the step 1. It gives us the tiling corresponding to the valid discrete tiling in question.

It is worth mentioning that in some tilings from step 1 we may have tiles without a single lattice point in their interior. It does not affect the correspondence proposition validity.

3.3 Obtaining the limit

We reached the last point of the plan. Now we can estimate the constant C. In continuous limit we can use Lebesgue measure instead of the number of points. It imposes some restrictions on tiles, but we can obtain geometric inequalities like we did in Subsection 2.1.2 using classic Brunn–Minkowski inequality (3). In many cases this approach gives us a sensible upper bound for the arithmetic intensity. For example form the Section 2 it gives the inequality

$$I \le I_{\text{scheme}} \frac{D_{\text{cache}}}{D_{\text{scheme}}}, \tag{10}$$

which corresponds to the value of the arithmetic intensity (8) we found for Example 2.4.

4 Generalizations of geometric locality model

In this section we discuss the way to generalize the geometric locality model.

General case of stencil algorithms. In Subsection 3.1 we imposed hard constraints on the stencil, namely stencil convexity and central symmetry for its space projection. Now let us discuss how to lift them. Suppose we have a non-convex stencil. Instead of changing the model to include the case we can produce a nonlinear transformation for our lattice which will make the stencil in question convex. Let us consider an example to illustrate this approach.

Example 4.1. On Figure 7 one can see the space projection of the cross stencil for 4-th order finite difference scheme. This scheme may be employed to solve scalar wave equation in 2D. The stencil is not convex but it is symmetrical in space.

The lattice transformation for this case is shown on Figure 8. Let us describe this transformation in details. In the first step of our construction we break down the lattice into four subgrids. In each subgrid, every lattice point has the same coordinates modulo two. Subgrids are shown on Figure 7. We denote each subgrid with corresponding pair of $(x \bmod 2, y \bmod 2)$.

The transformation appends a new periodic coordinate $z \in \{-1, 0, 1, 2\}$ to coordinates of each point, keeping x and y values unchanged. The value of coordinate z is the same for every point in a subgrid. Different subgrids have different values of coordinate z. Since coordinate z is periodic, the surfaces $z = -2$ and $z = 2$ are identified. Transformation does not affect the axis t in any way.

Here is the description of the transformations for each subgrid:

1. for circles subgrid $(0, 0)$, we add $z = 0$,
2. for diamonds subgrid $(1, 0)$, we add $z = -1$,
3. for triangles subgrid $(0, 1)$, we add $z = 1$,
4. for stars subgrid $(1, 1)$, we add $z = 2$.

The transformation on Figure 8 maps the stencil into an octahedron. Since octahedron is a convex body, we have made the stencil convex. Note that octahedron is also symmetrical, which allows us to use it as a ball in normed space. This norm corresponds to Manhattan distance L_1. Strictly speaking, the described transformation maps a lattice into a

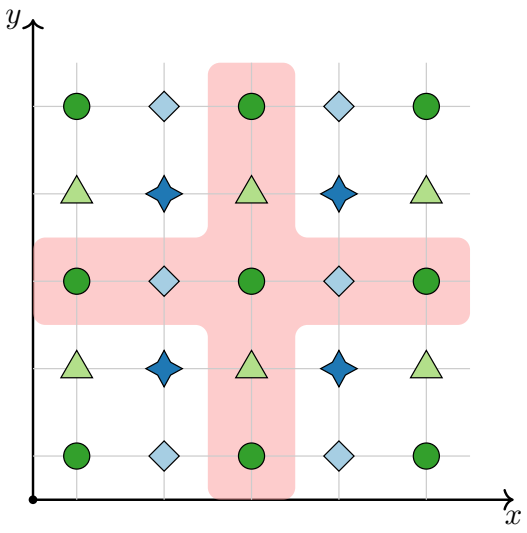


Figure 7. The shade represents space projection of the non-convex stencil. We introduce four subgrids each of which has its own marker for a point, namely: circle, diamond, triangle and a star.

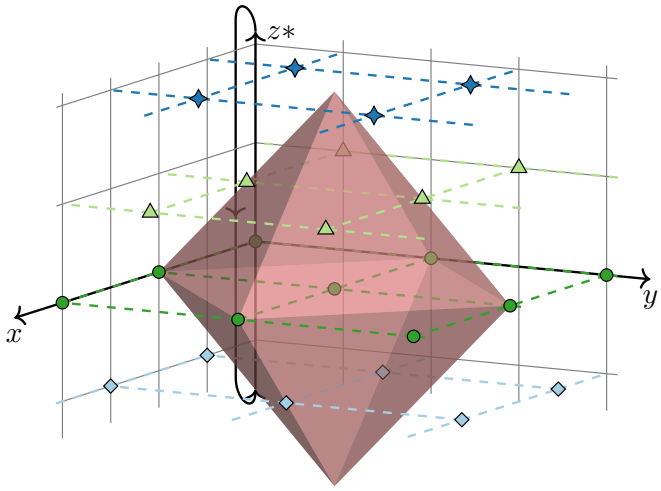


Figure 8. The points of the subgrids are denoted in the same way as on Figure 7. Transformation maps the stencil space projection into octahedron. Note that axis z is periodic.

manifold. However it can be easily embedded into a higher dimension vector space.

It is worth mentioning that points of different subgrids have different stencils after the transformation. Nevertheless octahedron perfectly represents each of these new stencils.

In the space on Figure 8, dependence cone of a point is a convex cone. A more common way to make a stencil convex is to take a convex hull. Unfortunately this approach will add phantom dependencies for every point. Basically, it forces us to load some data points, even though we may not need them. The described transformation lacks that drawback. The dependence cone we got in the transformed space does not contain any point of the discrete grid which is not a real dependency for the apex of the cone. From one point of view, the transformation makes the representation of dependencies from Section 3 exact.

Getting an exact representation of dependencies is important, but it also can be done by employing the generalized dependence graph from [4]. The dependence cone convexity has another consequence. If we describe the tiling with hyperplanes each hyperplane is required to lay outside dependence and influence cones. It is the very same requirement from [11] described in terms of cones. Non-convex dependence cone means we can't use a cone face as the tiling hyperplane. After the transformation we can do it.

Let us return to the example in question. The transformation from Figure 8 may be described as the code transformation from List. 1 to List. 2.

An interesting fact about the transformed code in List. 2 is that it allows all transformations allowed by original code (List. 1). We can choose to ignore the two innermost loops, which reduces the code to the original except the overhead on additional control statements. However we can also use the whole loop nest for optimization. For example we can use tiling surfaces $(i+p+2t,\ j+q+2t,\ 2t-i-j-p-q)$. These surfaces produce tilings which are not convex in terms of the original loop nest.

```
for(int t=0; t<T-2; t++) {
   for(int i=2; i<N-3; i++) {
      for(int j=2; j<M-3; j++) {
         A[t+1][i][j] = f(A[t][i-2][j], A[t][i-1][j],
            A[t][i][j], A[t][i+1][j], A[t][i+2][j],
            A[t][i][j-2], A[t][i][j-1], A[t][i][j+1],
            A[t][i][j+2]);
      }
   }
}
```

Listing 1. Computational kernel corresponding to the stencil form Fig. 7.

```
for(int t=0; t<T-2; t++) {
   for(int i=2; i<N-3; i++) {
      for(int j=2; j<M-3; j++) {
         for(int p=0; p<2; p++){
            for(int q=0; q<2; q++){
               if ((i%2==p) && (j%2==q)) {
                  A[t+1][i][j] = f(A[t][i-2][j],
                     A[t][i-1][j], A[t][i][j],
                     A[t][i+1][j], A[t][i+2][j],
                     A[t][i][j-2], A[t][i][j-1],
                     A[t][i][j+1], A[t][i][j+2]);
               }
            }
         }
      }
   }
}
```

Listing 2. Computational kernel corresponding to the space after the transformation similar to one on Fig. 8.

Unfortunately the transformed code on Listing 2 has an "if" clause with a non-linear condition. Usually such conditions are not supported by polyhedral code optimizers. If kernel from Listing 2 used as is, it may lead to incorrect dependence detection. There are several ways to use such transformations correctly within a polyhedral framework. We can calculate dependencies along with the transformation or we can add support for non-singular increments in the loops. No matter which way we choose, such transformations allow us to extend polyhedral framework to some non-convex tilings.

By applying the described transformation we can extend our geometric locality model to this particular non-convex stencil. Now the only question left is how to construct similar transformations for an arbitrary stencil.

For a stencil algorithm with a uniform stencil, it is possible to explicitly construct such transformations. The construction employs two steps:

1. define the distance between each pair of points with the help of dependence graph,
2. use the defined distance to construct a metric embedding.

This approach allows us to convert the transformation search problem into a metric embedding problem. An overview of metric embedding theory may be found in the article [1]. This article mentions a conjecture implying the existence of a relation between intrinsic dimension of the graph and embedding dimension, i.e. dimension of the space we embed the graph into. This conjecture gives us hope that embeddings we are looking for in the step 2 of the construction exist for cases which are more complex than uniform stencil algorithms.

Threads and synchronizations model. Up until now we did not mention synchronizations and threads in the geometric locality model. Our intention is to incorporate threads in the described framework. We need to build a geometric threading and synchronizations model, which would make it convenient to describe locality. Let us emphasize that the program code doesn't provide a good way to describe it. It is especially true for parallel programs, where performance and arithmetic intensity may vary dramatically between different runs. For certain codes it is possible to make predictions [9],[10], but accuracy of these estimations depends on the code in question.

We will describe a threading extension to geometric locality model in future works.

5 Applying results of the geometric locality model to the polyhedral model

Many ideas and results from our work may be also applied to the polyhedral model.

Consider the definition 3.2 of a conoid. If a conoid is a convex polyhedron, the definition 3.2 implies the same restrictions on normals to its faces as the ones the polyhedral model imposes on the tiling hyperplanes. For stencil algorithms geometric locality model describes more valid tilings than the polyhedral model. In our opinion, knowing what is missing may prove valuable in attempts to generalize the polyhedral model.

Theorem 3.6 allows us to easily prove validity of a tiling, e.g. Theorem 1 from [18] may be considered as its particular case.

Finally, the transformations analogous to one from the Section 4 may be employed to obtain non-convex tilings. If we have a direct and an inverse transformations, we can describe a convex tiling in the image space and get the non-convex tiling in the source space by applying the inverse transformation. The space tiling on Figure 9 was obtained by using this approach for the transformation in Example 4.1.

If non-linear transformations from the Section 4 is represented in terms of loop transformations, it will allow polyhedral model to find some non-convex tilings.

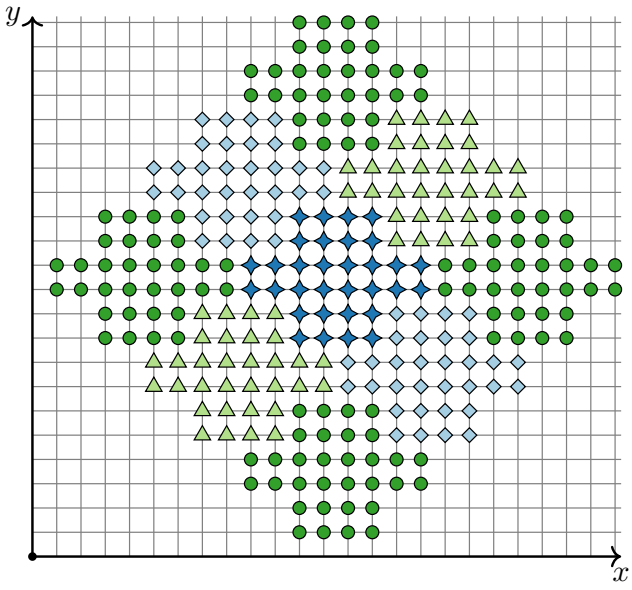


Figure 9. Non-convex tiling in space. The depicted tiling corresponds to a convex tiling with hyperplanes in the image space of the transformation on Figure 8.

6 Related works and discussion

The idea to describe locality problems in terms of isoperimetric problems is not entirely new. For example, article [3] uses the same concept. However there is a number of difference between our approaches.

Boulet at al. solve the problem only for a specific type of tile, namely a parallelotope. They have a way to find the corresponding best tiling. Since the article had been written before Roofline model appeared, it does not use the arithmetic intensity as a performance indicator and is vague on the role of machine parameters in its performance analysis. Finally, Boulet at al. optimize the communication volume for a fixed computation volume, which corresponds to the simplified cache model from Subsection 2.1.

We use an isoperimetric-type inequality which is universal enough to cover any shape of tiles. On the other hand, our approach is not constructive. We do not have a way to build a tile which saturates the inequality. We cannot even be sure if it exists. We also consider a more realistic cache model giving a different inequality, for which we only outlined a possible proof. From a certain point of view, our work may be considered as a continuation of the aforementioned work with a slightly different angle to the approach.

7 Results and conclusions

In this paper we have outlined how to use geometric inequalities to estimate the upper bound on arithmetic intensity

of a stencil algorithm. We have formulated Conjecture 2.5, which stated that arithmetic intensity for a stencil algorithm has a bound of the form $I \leq C\sqrt[n]{D_{\text{cache}}}$, where n is a number of space dimensions for the algorithm. This inequality limits the impact of the polyhedral optimization on a stencil algorithm performance.

The geometric locality model has been introduced to simplify asymptotic estimation of the constant C for the case of large tiles. The constraints on tile shapes and valid tilings were described in terms of the geometric locality model. These constraints allow a number of valid tilings which cannot be described by the polyhedral model. These results of geometric locality model may prove useful for understanding the polyhedral model.

The non-linear transformations discussed in Section 4 are especially notable. With the aid of these transformations polyhedral model may be extended to valid nonconvex tilings.

The geometric locality model admits a rigorous mathematical description which will be given in future work.

The presented approach described only stencil algorithms. Generalizing the approach for an arbitrary algorithm presents an interesting research problem.

Acknowledgments

We gratefully commend anonymous reviewers of IMPACT 2025 for helpful comments and literature suggestions. We thank Vadim Levchenko and Anastasia Perepelkina for formulating this problem and useful discussions. We also express gratitude to Ksenia Bulycheva for support and reviewing all aspects of this work.

References

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. 2011. Advances in metric embedding theory. *Advances in Mathematics* 228, 6 (2011), 3026–3126. doi:10.1016/j.aim.2011.08.003

[2] Uday Bondhugula, Muthu Baskaran, Sriram Krishnamoorthy, J. Ramanujam, Atanas Rountev, and P. Sadayappan. 2008. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 132–146. doi:10.1007/978-3-540-78791-4_9

[3] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. 1994. (Pen)-ultimate tiling? *Integration* 17, 1 (1994), 33–51. doi:10.1016/0167-9260(94)90019-1

[4] Paul Feautrier. 1992. Some efficient solutions to the affine scheduling problem. I. One-dimensional time. *International journal of parallel programming* 21, 5 (1992), 313–347.

[5] Paul Feautrier. 1992. Some efficient solutions to the affine scheduling problem. Part II. Multidimensional time. *International journal of parallel programming* 21, 6 (1992), 389–420.

[6] Paul Feautrier and Christian Lengauer. 2011. *Polyhedron Model*. Springer US, Boston, MA, 1581–1592. doi:10.1007/978-0-387-09766-4_502

[7] Herbert Federer. 1996. *Geometric measure theory*. Springer.

[8] Richard Gardner and Paolo Gronchi. 2001. A Brunn-Minkowski inequality for the integer lattice. *Trans. Amer. Math. Soc.* 353, 10 (2001), 3995–4024.

[9] Pieter Ghysels and Wim Vanroose. 2015. Modeling the Performance of Geometric Multigrid Stencils on Multicore Computer Architectures. *SIAM Journal on Scientific Computing* 37, 2 (jan 2015), C194–C216. doi:10.1137/130935781

[10] Julian Hammer, Jan Eitzinger, Georg Hager, and Gerhard Wellein. 2017. Kerncraft: A Tool for Analytic Performance Modeling of Loop Kernels. (13 Jan. 2017). doi:10.1007/978-3-319-56702-0_1 arXiv:1702.04653 [cs.PF]

[11] Francois Irigoin and Remi Triolet. 1988. Supernode Partitioning. In *Symposium on Principles of Programming Languages (POPL'88)*. San Diego, CA, 319–328. http://ssh.cri.ensmp.fr/classement/doc/A-179.pdf

[12] Wayne Kelly, William Pugh, Evan Rosser, and Tatiana Shpeisman. 1996. Transitive Closure of Infinite Graphs and Its Applications. *International Journal of Parallel Programming* 24, 6, 579–598. doi:10.1007/bf03356760

[13] A. Kolmogoroff. 1934. Zur Normierbarkeit eines allgemeinen topologischen linearen Raumes. *Studia Mathematica* 5, 1 (1934), 29–33. http://eudml.org/doc/218127

[14] B. A. Korneev and V. D. Levchenko. 2016. Effective solving of three-dimensional gas dynamics problems with the Runge-Kutta discontinuous Galerkin method. *Computational Mathematics and Mathematical Physics* 56, 3 (March 2016), 460–469. doi:10.1134/S0965542516030118

[15] Vadim Levchenko and Anastasia Perepelkina. 2023. Heterogeneous LBM Simulation Code with LRnLA Algorithms. *Commun. Comput. Phys* 33 (2023), 214–244.

[16] Vadim Levchenko, Anastasia Perepelkina, and Andrey Zakirov. 2016. DiamondTorre Algorithm for High-Performance Wave Modeling. *Computation* 4, 3 (Aug. 2016), 29. doi:10.3390/computation4030029

[17] V. D. Levchenko and A. Y. Perepelkina. 2018. Locally Recursive Non-Locally Asynchronous Algorithms for Stencil Computation. *Lobachevskii Journal of Mathematics* 39, 4 (5 2018), 552–561. doi:10.1134/s1995080218040108

[18] Ravi Teja Mullapudi and Uday Bondhugula. 2014. Tiling for Dynamic Scheduling. In *Proceedings of the 4th International Workshop on Polyhedral Compilation Techniques*, Sanjay Rajopadhye and Sven Verdoolaege (Eds.). Vienna, Austria. http://impact.gforge.inria.fr/impact2014/papers/impact2014-mullapudi.pdf

[19] Daniel Orozco, Elkin Garcia, and Guang Gao. 2011. Locality Optimization of Stencil Applications Using Data Dependency Graphs. In *LCPC 2010 (LNCS, Vol. 6548)*, K. Cooper, J. Mellor-Crummey, and V. Sarkar (Eds.). Springer, 77–91. doi:10.1007/978-3-642-19595-2_6

[20] Robert Osserman. 1978. The isoperimetric inequality. *Bull. Amer. Math. Soc.* 84, 6 (1978), 1182–1238.

[21] A Yu Perepelkina, V D Levchenko, and I A Goryachev. 2014. Implementation of the Kinetic Plasma Code with Locally Recursive non-Locally Asynchronous Algorithms. *Journal of Physics: Conference Series* 510, 1 (may 2014), 012042. doi:10.1088/1742-6596/510/1/012042

[22] Robert Strzodka, Mohammed Shaheen, Dawid Pajak, and Hans-Peter Seidel. 2011. Cache Accurate Time Skewing in Iterative Stencil Computations. In *2011 International Conference on Parallel Processing*. IEEE, 571–581. doi:10.1109/icpp.2011.47

[23] Samuel Williams, Andrew Waterman, and David Patterson. 2009. Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures. *Commun. ACM* 52, 4, 65–76. doi:10.1145/1498765.1498785

[24] Andrey Zakirov, Sergei Belousov, Ilya Valuev, Vadim Levchenko, Anastasia Perepelkina, and Yasunari Zempo. 2017. Using memory-efficient algorithm for large-scale time-domain modeling of surface plasmon polaritons propagation in organic light emitting diodes. *Journal of Physics: Conference Series* 905, 1 (oct 2017), 012030. doi:10.1088/1742-6596/905/1/012030