Title
Selecting tile shape for minimal execution time

Authors
Hogstedt, Karin
Carter, Larry
Ferrante, Jeanne

Publication Date
1999-05-20

Peer reviewed

Selecting tile shape for minimal execution time

Karin Högstedt [*], Larry Carter[†], Jeanne Ferrante[‡]
{hogstedt, carter, ferrante}@cs.ucsd.edu
Department of Computer Science and Engineering, UCSD
9500 Gilman Drive, La Jolla, CA 92093-0114

Abstract

Many computationally-intensive programs, such as those for differential equations, spatial interpolation, and dynamic programming, spend a large portion of their execution time in multiply-nested loops which have a regular stencil of data dependences. Tiling is a well-known optimization that improves performance on such loops, particularly for computers with a multi-levelled hierarchy of parallelism and memory.

Most previous work on tiling restricts the tile shape to be rectangular. Our previous work and its extension by Desprez, Dongarra, Rastello and Robert showed that for doubly nested loops, using parallelograms can improve parallel execution time by decreasing the *idle time*, the time that a processor spends waiting for data or synchronization. In this technical report, we extend that work to more deeply nested loops, as well as to more complex loop bounds. We introduce a model which allows us to demonstrate the equivalence in complexity of linear programming and determining the execution time of a tiling in the model. We then identify a sub-class of these loops that constitute *rectilinear iteration spaces* for which we derive a closed form formula for their execution time. This formula can be used by a compiler to predict the execution time of a loop nest. We then derive the tile shape that minimizes this formula.

Using the duality property of linear programming, we also study how the longest path of dependent tiles within a rectilinear iteration space changes with the tile shape. Finally, we observe that the execution time of a rectilinear iteration space depends on the slope of only four of the facets defining the iteration space, independent of its dimensionality.

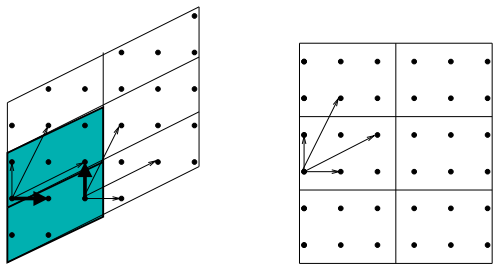
1 Introduction

Many computationally intensive programs spend a high percentage of their execution time in nested loops that have a regular stencil of data dependences between the iterations. Obtaining high performance on such loops requires both data locality and efficient use of parallel resources. *Tiling* [Wol87, RAP87, IT88, Wol89, RS91] is a well-known optimization that is effective at achieving both goals. Conceptually, given a loop nest, tiling operates on the *iteration space* defined by that loop nest. For a loop nest of depth K, the iteration space is a polytope in K-dimensional space. The tilings we consider partition this space into a collection of *parallelepipeds* which are then scheduled for atomic execution. Tiling enables data locality when the tile size ensures that the necessary data fit in a given level of the memory hierarchy.

Tiling also enables parallelization since independent tiles can simultaneously be executed on different processors. The data dependences [Wol96] between the iterations impose dependences between the tiles. If the tile shape[1] is chosen such that the angle between the extreme vectors of the iteration dependences [IT88] are included within the angle of the tile (see Figure 1), then we are guaranteed no dependence cycles between the tiles [IT88], and the tiling is hence legal.

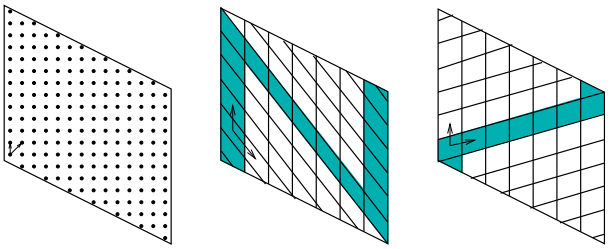
Apart from affecting the legality of the tiling, the shape also affects the execution time. Consider the iteration space in Figure 2(a). The iteration dependences are such that both the tilings in Figure 2(b) and (c) are legal

[*]Partly supported by the Hans Werthén Foundation, Swedish Royal Academy of Engineering Sciences.
[†]Also at San Diego Supercomputing Center.
[‡]This work was partly supported by NSF grant CCR-9504150 and IBM.
[1] "Shape" is an informal notion referring to the coefficients of the the linear constraints that define the tile boundaries.



(a) (b)

Figure 1: *Examples of (a) an illegal and (b) a legal tiling. The tiling in (a) is illegal since there is a dependence cycle between the two shaded tiles, due to the dependence vectors in bold. In (b) the slope of the tiling is chosen small enough so that the angle between the extreme vectors of the iteration dependences are included within the angle of the tile. As shown in [IT88], there are therefore no cycles between the tiles.*



(a) (b) (c)

Figure 2: *Examples of two legal tilings ((b) and (c)) of the iteration space in (a). The iteration dependence stencil is shown in (a) and repeated across all the iterations. The iteration dependences result in the tile dependences shown in (b) and (c). Assume that the execution time of a full tile, and the amount of communication necessary between processors, are the same in the two tilings. Then the longest paths of dependent tiles (shaded tiles) in (b) is much longer than in (c). Assuming the model described in Section 3, the tiling in (c) therefore yields a better parallel execution time than the one in (b).*

tilings. In the latter however, the longest path of dependent tiles is much shorter, and therefore executes in a shorter amount of time.[2] In this technical report, we explore the relationship between the shape of the tiles and the execution time. A summary of the contributions of this technical report follows.

- We introduce the notions of *initial* and *final synchronization points*. Intuitively these points are the end points of the longest path of dependent tiles within the iteration space, and therefore a measure of the execution time of the iteration space.
- We introduce a new model for which we show that the complexity of determining the execution time of an iteration space is equivalent to linear programming.
- We identify *rectilinear iteration spaces*, a sub-class of convex iteration spaces.
- For rectilinear iteration spaces, we find the coordinates of the initial and final synchronization points and are thus able to derive a simple formula for the execution time.

[2] under the model described in Section 3.

- We study how the longest path of dependent tiles within a rectilinear iteration space varies with the slope of the tile shape.
- We determine the tile shape that minimizes the execution time of a rectilinear iteration space.
- We observe that the execution time depends on the slope of *four* of the facets defining a rectilinear iteration space, independent of its dimensionality.

Rectilinear iteration spaces include rectangular iteration spaces of any dimension that have been skewed in one dimension. A particular example is the abstraction of hyperbolic code (SOR) used originally by Wolf and Lam [WL91a] and in [CFH95a]. Our previous work could not handle this code as tiled in [CFH95a].

In Section 2 we discuss related work in the area, and Section 3 contains a description of the assumptions and the model used in this technical report. In Section 4 we show that the complexity of determining the execution time of a convex iteration space under our model is equivalent to general linear programming. In Section 5 we introduce *rectilinear iteration spaces*, a sub-class of convex iteration spaces, and in Section 6 we use the duality property of linear programming to determine the location of the initial and final synchronization points in a rectilinear iteration space. In Section 7 we then study how the longest path of dependent tiles within the iteration space changes with the tile shape, and determine the tiling that optimizes the execution time under our model. Finally, in Section 8 we present a closed-form execution time formula for a K-dimensional rectilinear iteration space, as well as for some common iteration space shapes.

2 Background

As previously mentioned, traditional tiling [Wol87, RAP87, IT88, Wol89, RS91] partitions the iteration space of a nested loop with a regular stencil of dependences into regular size and shape pieces. Tiling has been used to achieve both locality and parallelism, and its beneficial effects are well established [WL91a, WL91b, CK92, RS91, CFH95a, CFH95b]. Unimodular transformations [Ban90, WL91b] (which include loop skewing) can increase the applicability of tiling.

Most of this previous work on tiling considered rectangular tiles, or tiles of the same slope as the iteration space.[3] For such tiles, only tile size selection is of concern [CM95, Sar97, ARY98]. Our previous work [HCF97] recognizes the importance of tile shape selection for minimizing the idle time of parallel execution. Both that work and [DDRR97] considered parallelepiped-shaped tiles for two dimensional iteration spaces, and determined simple formulas for idle time and finishing time. Allowing more general tile shapes is particularly useful when tiling is applied in the context of multiple levels of parallelism [MCFH97, MHCF98], such as in hierarchical tiling [CFH95a, CFH95b]. The difference between our previous work and [DDRR97] is that the latter used a slightly different model of partial tiles, resulting in simpler formulas and proofs. This technical report directly extends [HCF97] from 2-dimensional iteration spaces to K-dimensional iteration spaces of a much more general shape. [DKR91] uses linear programming to determine the execution time of a convex iteration space under the assumptions that the iteration space is large enough compared to the size of a tile to be able to ignore the effects of partial tiles. Our model extends this work by incorporating the effect of partial tiles.

3 Model and assumptions

The model of an iteration space we consider in this technical report is a convex, K-dimensional polytope, that is, the set of K-tuples of real numbers that satisfy a given set of linear inequalities. Convex polytopes model nested loops of the form:

```
for x_K = 0 to N
 for x_{K-1} =max[f_1^{K-1}, f_2^{K-1}, ...] to min[g_1^{K-1}, g_2^{K-1}, ...]
  ...
   for x_j =max[f_i^j, f_2^j, ...] to min[g_i^j, g_2^j, ...]
    ...
```

[3] An exception is [RAP87], which considered the use of hexagonal tiles.

```
for x1 =max[f1^1, f2^1, ...] to min[g1^1, g2^1, ...]
 <loop body>
```

where f_i^j and g_i^j are linear functions of less deeply nested loop index variables. Each such function corresponds to a linear inequality that bounds the convex polytope, and *vice versa*.

Our model resides in real (as opposed to integer) space, even though the iterations of a loop are more naturally modeled by a lattice of discrete points. We approximate the number of iterations in a region of space by the volume of the region. This allows us to avoid messy details about the alignment of a region with respect to the lattice of iterations. For large regions, the volume approaches the number of iteration points in the region.

In general, a tiling of a nested loop induces a partitioning of the model's K-dimensional real space into parallelepipeds by K families of parallel hyper-planes [HCF97]. The angles of the hyper-planes are chosen so that the angle between the extreme vectors of the iteration dependences are included within the angle of the parallelepiped. As shown in [IT88], the resulting tiling is hence legal.

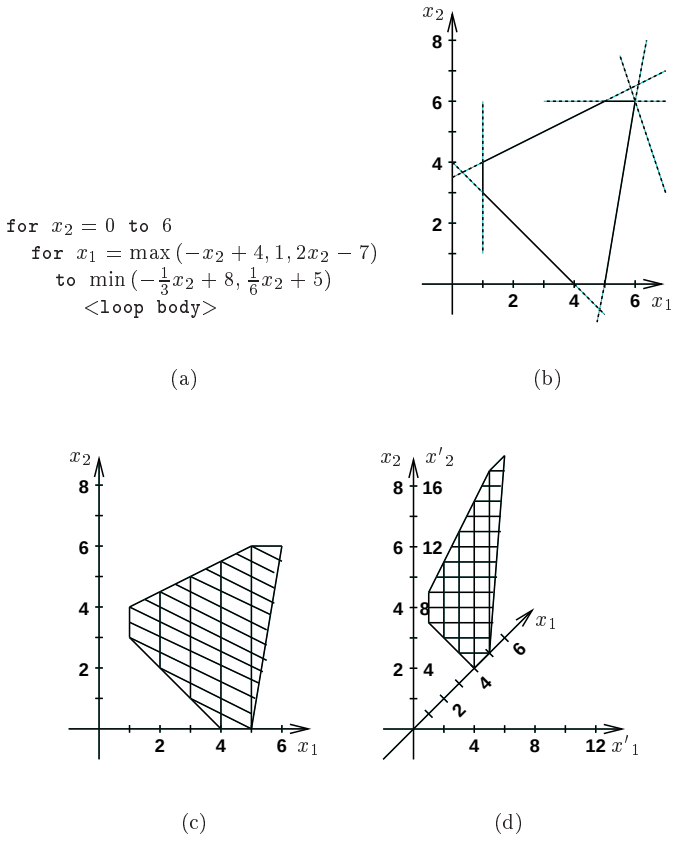


Figure 3: *Transformation of a sample loop nest to an iteration space. The loop nest is shown in (a), and the corresponding iteration space in (b). In (c) a tiling has been chosen and in (d) the necessary skewing and scaling has been performed to achieve unit cube tiles.*

In this technical report, we skew and scale the parallelepipeds into unit cubes.[4] This transformation is quite straight-forward and illustrated in Figure 3.[5] The intersection of a unit cube with the iteration space is called a *tile*. If the cube is entirely contained in the iteration space, it is a *full* tile; if it contains both points inside the iteration space and points outside, it is a *partial* tile; otherwise it is an *empty* tile.

[4]This is not a transformation of the loop, but a conceptual transformation of the iteration space.

[5]Given a facet of a tiled iteration space, let i_d be the original slope of the facet in dimension d, and let t_d be the slope of the tiling in dimension d and let w_d and w_K be the width of the parallelepipeds in dimension d and K, respectively. The slope in dimension d of the same facet after the skewing and scaling, r_d, is then given by $r_d = (i_d - t_d)\frac{w_d}{w_K}$.

As in [HCF97, DDRR97], we assume that in each dimension, there are data dependences and synchronization constraints between the tiles so that no point in tile T may be executed before all points in its predecessor[6] tiles in all dimensions have completed execution. If in some dimension d there are no dependences the problem is embarrassingly parallel in that dimension and reduces to a lower dimensional problem.

The last (i.e. the K-th) dimension is considered the vertical dimension. A *stack* of tiles is the set of all non-empty tiles that differ only in the last coordinate; and the *bottom* (or *top*) tile in a stack is the tile with minimal (or maximal) last coordinate. Similarly, the bottom (or top) of the iteration space is the set of bottom (or top) tiles in all stacks. The *left* (*right*) sides of an iteration space are any tiles that in a particular dimension d do not have any predecessors (successors) in that dimension *and* that constitute neither the bottom nor the top of the iteration space.

This technical report is concerned with the execution of an iteration space by a parallel computer. We consider *block distribution* of tiles to processors, in which each stack of tiles is assigned to its own unique processor. Because of the dependences, a processor must execute its stack from bottom to top. In all sections but Section 4, we also make the following technical assumption.

Assumption 1 *We assume that in each stack either 1) there exists at least one full tile that has an immediate successor tile that is full, or 2) no tile in the stack has non-empty successors.*

At time zero, each processor begins executing the bottom tile T in its first stack, provided T has no non-empty predecessors. Otherwise, the processor is idle. Thereafter, a processor starts executing the next tile in its current stack when all predecessor tiles are completed. In our model, we assume the time required to execute a tile is proportional to its volume, with the time for a full tile being a constant, notated by E_t. Given a tiling with tiles of a certain shape and size, we assume E_t to be given, maybe determined experimentally. E_t includes overheads such as cache miss penalties, loop overheads, and communication latency [CKP+96], as well as the pure computation cost of the tile. It is future work to study the effects the choice of tile parameters has on those overheads. In this technical report we concentrate on minimizing the number of tiles on the longest path of dependent tiles.

We want our model for the execution time of partial tiles to have the following properties.

1. The L_1-norm distance[7] between two vertices of the iteration space polytope (i.e. extreme points of the polytope) should be equal to the sum of the volume of the tiles on the longest path of dependent tiles between those points. Note that this distance is not necessarily an integer due to the partial tiles.

2. The volume of each stack, and thus the whole iteration space, should be the same as in the original iteration space.

The first property permits us to determine the execution time of a tiled iteration space using linear programming. This is a peculiar property since we want a linear distance to be equal to a volume. For a single tile, a unit cube, this can be achieved if we ensure that the extreme points selected by the linear programming algorithm always occur at the midpoint of the top and bottom surface of a tile. For this to happen we choose the set of constraints for the linear programming problem as shown in Figure 4. This choice gives the desired property not only for single tiles, but also for paths of dependent tiles.

The second property is desirable since we assume that the execution time of a tile is proportional of the volume of the tile. If this property does not hold and the difference in volumes can be large, it is likely that the difference between the execution time determined using the model and the actual execution time also is large.

The models used in previous research do not have these properties. The volume of the iteration space using the model in [HCF97] exactly corresponds to the volume of the original iteration space, but the model does not fulfill the first requirement, since the facets bounding the iteration space can intersect at any point in a stack. Using the model in [DDRR97] the volume of a stack might differ by an arbitrarily large amount compared to the original iteration space.

We introduce a new model that fulfills the above two properties. The necessary transformation is illustrated in Figure 4, and a comparison to other models are given in Figure 5. Note that after this transformation the slopes of the different facets and even the number of facets might change.

[6] A tile T_1 is a predecessor of T_2 if the tile dependences are such that T_1 needs to be executed before T_2 can start. If T_1 is a predecessor of T_2, then T_2 is a successor of T_1.

[7] L_1-norm distance between two points $\vec{x}$ and $\vec{y}$ is defined as $\sum_i |y_i - x_i|$.

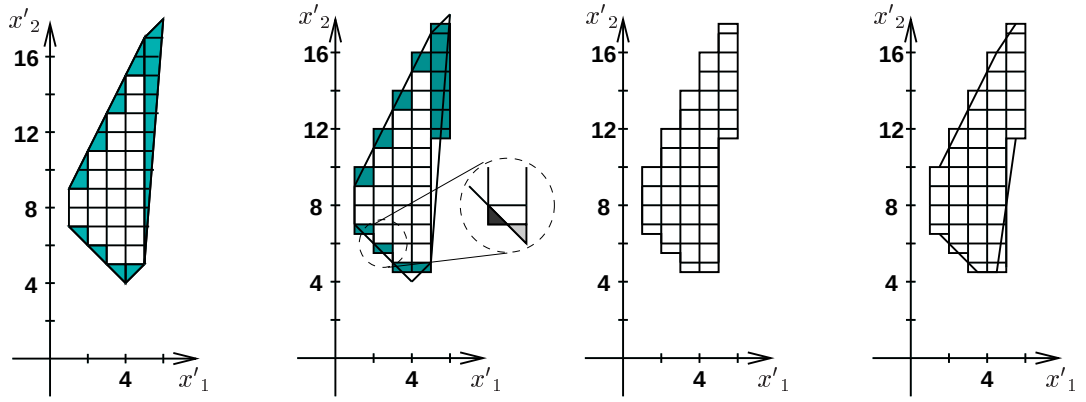


Figure 4: *Example to illustrate our model. (a) shows the same sample iteration space as in Figure 3(d) (this corresponds to the model used in [HCF97]) with the partial tiles shaded, and (b) shows how the partial tiles are transformed to our model. In our model the bottom and top of a stack is always horizontal. The height of a stack is determined by making a horizontal cut so that the volume cut off is equal to the volume added by the cut, i.e in the blow-up in (b), the area of the dark shading is equal to the area of the light shading. Note that the total area shaded in (a) is equal to the total area shaded in (b), i.e. this transformation preserves the volume of the stacks. In (c) we show the transformed iteration space. In (d) we show how the constraints of the linear programming problem are determined. For all dimensions the slope between two stacks is equal to the slope of the line between the mid-points of the two stacks in that dimension.*

Apart from the number of tiles on the longest path of dependent tiles, the execution time also depends on the additional time it takes to communicate data between processors. To model the communication time, we define a parameter c_d such that $c_d E_t$ is the amount of time that is needed to communicate a full tile surface between two neighboring processors in dimension d, *and* that can be overlapped with computation of subsequent tiles on the same processor. This corresponds to the latency part of communication in the LogP model [CKP+96]. The part of the communication time that *cannot* be overlapped, i.e. the overhead in the LogP model, is included in E_t. E.g. c_d is equal to zero when the architecture does not allow any overlap of communication by computation. This does not mean that the communication cost is 0, merely that the processor is busy communicating and the communication cost is therefore included in the execution time of the tile. For example, $c_d = 0.25$ if the architecture allows for $0.25E_t$ seconds of the communication of one full tile surface to be overlapped by computation. In this case, immediately after a tile is completed the execution of the next tile in the same stack may begin, but the dependent tile in the adjacent stack has to wait for $0.25E_t$ seconds. The reason the value of c varies with the dimension is that the amount of data needed to be communicated varies from dimension to dimension. In particular, since all tiles in each stack are executed by the same processor no communication is needed in the K-th dimension and we have $c_K = 0$.

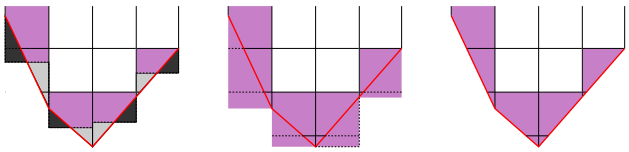
Let p_d and q_d be two coordinates in dimension d such that $p_d \leq q_d$. Due to property 1 of our model, there are then $q_d - p_d$ tiles and $q_d - p_d$ surfaces to be communicated on the path between p_d and q_d. This observation leads to the following definition of the execution time of the tiles on the longest path of dependent tiles between $\vec{p}$ and $\vec{q}$.

Definition 1 *Let $\vec{p} = (p_1, \ldots, p_K)$ and $\vec{q} = (q_1, \ldots, q_K)$ be two points within an iteration space such that $\vec{p}$ precedes[8] $\vec{q}$. We define $E(\vec{p}, \vec{q})$, the execution time of the path from $\vec{p}$ to $\vec{q}$, as follows.*

$E(\vec{p}, \vec{q}) = \sum_{d=1}^{K} (q_d - p_d)(1 + c_d)E_t$

Lemma 1 proves a lower and upper bound on the difference in the execution time under our model and the model in [HCF97]. The proof of this lemma consists of two parts. Let I and $\hat{I}$ denote an iteration space that

[8]Given two points $\vec{p}$ and $\vec{q}$ in the iteration space, we say $\vec{p}$ *precedes* $\vec{q}$ if for all dimensions d, $p_d \leq q_d$. If $\vec{p}$ doesn't precede $\vec{q}$, the longest path of dependent tiles between them is undefined.



(a) (b) (c)

Figure 5: *The bottom of a 2-dimensional, rectilinear iteration space. (a) shows our model of partial tiles in this technical report. The height of each stack is chosen such that the area of the light shading is equal to the area of the dark shading. (b) and (c) show the models used in [DDRR97] and [HCF97] respectively. The model in (a) allow us to elegantly write the problem of finding the length of a path of dependent tiles as a linear programming problem. In [DDRR97], the execution time of a 2-dimensional partial tile is a rectangular tile of size $p * f$, where f is the length of a full tile in the first dimension and p is the length of the partial tile in the second and last dimension.*

has been modeled using our model and the model in [HCF97], respectively. Let p and $\hat{p}$ be the longest path of dependent tiles through I and $\hat{I}$, respectively. First we notice that for all paths in I there exist a path in $\hat{I}$ of equal or greater length. p can therefore not be longer than $\hat{p}$. See Figure 6 for an illustration of this argument. Furthermore, let $\hat{q}$ be a path through $\hat{I}$ that does not go through (non-empty) tiles that are empty in I (unless it also goes through all such tiles in that same stack). Then we know that for each $\hat{q}$ there exists a path in I of equal length. If $\hat{p}$ is of this type, it can therefore not be longer than p. This argument holds since $\hat{p}$ then goes through either all or no partial tiles in a stack, and all stacks have the same volume in the two models.

Second, we determine an upper bound on the total volume of the tiles that can lie on $\hat{p}$ but are empty in I. At either end of $\hat{p}$ there can be at most $K-1$ of them and it can be shown that the largest volume the i-th of those tiles can have is a $\frac{1}{(i+1)!}$-th of a full tile volume. See Figure 7 for a 2 and 3-dimensional example. Adding the volume of all those tiles together, we get an upper bound on the difference in volume of the tiles on $\hat{p}$ and p of $2V \sum_{d=1}^{K-1} \frac{1}{(d+1)!} \leq 2V$ where V is the volume of a full tile. The same reasoning is applied to prove a limit on the amount of communication time needed in between those tiles.

Lemma 1 *Let $c = \max_d [c_d]$, and let E_i and $\hat{E}_i$ be the execution time of a rectilinear iteration space using the model of partial tiles used in this technical report and [HCF97], respectively.*

$$0 \leq \hat{E}_i - E_i \leq 2(2c+1)E_t$$

Proof of Lemma 1: Let $\hat{I}$ be an iteration space modeled with the model used in [HCF97] and let $\hat{p}$ be the longest path of dependent tiles through $\hat{I}$. Now consider the same iteration space modeled with the model used in this paper, and let us call the resulting iteration space I. Let p be the longest path of dependent tiles through I.

$\hat{E}_i - E_i \geq 0$**:** This direction is trivial once we have proved Sublemma 1. If for each path in I, there exists a path in $\hat{I}$ of equal or greater length, then p cannot be longer than $\hat{p}$. This argument is illustrated in Figure 6.

Sublemma 1 *For all paths in I there exists a path in $\hat{I}$ of equal or greater length.*

Proof of Sublemma 1: Let q be a path in I, and let T_1 and T_2 be the first and last tile on q that corresponds to a full tile in $\hat{I}$. Either q is such that all the tiles below T_1 (or above T_2) in that same stack are on q, or it is not.

Assume q is of the former kind. Then we know there exists a path in $\hat{I}$ with the same volume as q, since each stack in the original iteration space has the same volume in I and $\hat{I}$.

Assume q is of the latter kind. Then we know either the beginning end of q has gone through a tile in a different stack before going through T_1, or the ending end of q has gone through a tile in another stack after having gone through T_2. Due to Assumption 1 we know that there exists a path identical to q except that it goes between the same stacks as q through only full tiles. That path is then either of the same or of a greater volume than q, and is of the former kind.

End of Proof of Sublemma 1 □

$\hat{E}_i - E_i \leq 2E_t(1+2c)$**:**

Let us look at two possible characteristics $\hat{p}$ might have at each end of the path. Let T^l and T^u be the first and last full tile on $\hat{p}$, respectively . Due to Assumption 1, we know both T^l and T^u exist (they might coincide). Before $\hat{p}$ goes through T^l, it either only goes through partial tiles in the same stack as T^l, or it does not. Similarly, after $\hat{p}$ goes through T^u it either only goes through partial tiles in the same stack as T^u, or it does not. In dimensions where either the beginning or the end of $\hat{p}$ only goes through partial tiles from the same stack as T^l or T^u, we say that end of $\hat{p}$ belongs to Case 1. Otherwise it belongs to Case 2.

Consider the case in which both the beginning and end of $\hat{p}$ belongs to Case 1. In this case, $\hat{p}$ either goes through all or no partial tiles of the top or bottom of a given stack. Since all stacks have the same volume in the two models, we then know there exists a path in I of the same length as $\hat{p}$. We also know this path is p, using Sublemma 1. Hence, dimensions in which both ends of $\hat{p}$ belong to Case 1 do not contribute to the difference between the two models. The largest difference in execution time between the two models thus occurs when both ends of $\hat{p}$ belong to Case 2 in all dimensions. We consider this worst-case scenario next.

Assume T^l is located at the coordinate $(t_1^l, \ldots, t_K^l)$ and let S_K^l be the stack that contains T^l. Symmetrically, assume T^u is located at the coordinate $(t_1^u, \ldots, t_K^u)$ and let S_K^u be the stack that contains T^u. We first note that the tile just below T^l (above T^u), $(t_1, \ldots, t_{K-1}, t_K - 1)$ $((t_1^u, \ldots, t_{K-1}^u, t_K^u + 1))$, is the lowest (highest) tile in S_K^l (S_K^u) to have predecessors (successors) in any other dimension than K.[9] We also notice that $\hat{p}$ can not cross more than one stack boundary in each dimension before (after) going through T^l (T^u), since such a path would be shorter than the one that crossed those stack boundaries at a larger (smaller) coordinate in the K-th dimension (see Figure 7). These two observations lead to the conclusion that the tiles preceding (succeeding) the tile at $(t_1, \ldots, t_{K-1}, t_K - 1)$ $((t_1, \ldots, t_{K-1}, t_K + 1))$ in any dimension d have no predecessors (successors) in dimension d.

$\forall d : 1 \leq d \leq K-1$ let T_d^l be the tile located at $(t_1^l - 1, \ldots, t_d^l - 1, t_{d+1}^l, \ldots, t_{K-1}^l, t_K^l - 1)$, and let S_d^l be the stack that contains T_d^l. $\forall d : 1 \leq d \leq K-1$ let T_d^u be the tile located at $(t_1^u - 1, \ldots, t_d^u - 1, t_{d+1}^u, \ldots, t_{K-1}^u, t_K^u + 1)$, and let S_d^u be the stack that contains T_d^u. Let q be the path that goes through all the tiles below T^l and above T^u in I. The largest difference in execution time between $\hat{p}$ and q therefore occurs when $\hat{p}$ goes through T_d^l and T_d^u $\forall d$, and when T^l and T^u are the first and last tiles on q, respectively. The size of the difference is less or equal to the sum of the volumes of T_d^l and T_d^u $\forall d$ plus the sum of all the tile surfaces communicated between those tiles. This is also an upper bound on the difference between the execution times of $\hat{p}$ and p since p can not be shorter than q. We now determine the volume of those tiles and tile surfaces in Sublemma 3 and 4, respectively. But first we calculate an integral in Sublemma 2 that we need in the proofs.

Sublemma 2

$$\int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{d-1} x_i} 1 - \sum_{i=1}^{d} x_i \quad dx_d \ldots dx_2 = \frac{1}{d!}(1-x_1)^d$$

Proof of Sublemma 2:

Let D and d be positive integers such that $2 \leq D \leq d$.

Induction Hypothesis:

$$\int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-1} x_i} \frac{1}{(d-(D-1))!}(1 - \sum_{i=1}^{D} x_i)^{d-(D-1)} \quad dx_D \ldots dx_2 = \frac{1}{d!}(1-x_1)^d$$

[9] We know tiles $(t_1, \ldots, t_{K-1}, t_K - 1)$ and $(t_1^u, \ldots, t_{K-1}^u, t_K^u + 1)$ are non-empty since otherwise $\hat{p}$ would belong to Case 1.

Base Case ($d = 2$):

When $d = 2$, the only possible value of D is 2.

$$\int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-1} x_i} \frac{1}{(d-(D-1))!}(1-\sum_{i=1}^{D} x_i)^{d-(D-1)} \quad dx_D \ldots dx_2 =$$

$$= \int_0^{1-x_1} (1-x_1-x_2) \quad dx_2 \qquad (D = d = 2)$$

$$= \left[(1-x_1)x_2 - \frac{x_2^2}{2}\right]_0^{1-x_1}$$

$$= (1-x_1)(1-x_1) - \frac{(1-x_1)^2}{2}$$

$$= \frac{(1-x_1)^2}{2}$$

$$= \frac{1}{d!}(1-x_1)^d \qquad (d = 2)$$

Induction Step: Suppose the induction hypothesis true for $d-1$. We now show it is true for d.

Base Case ($D = 2$):

$$\int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-1} x_i} \frac{1}{(d-(D-1))!}(1-\sum_{i=1}^{D} x_i)^{d-(D-1)} \quad dx_D \ldots dx_2 =$$

$$= \int_0^{1-x_1} \frac{1}{(d-1)!}(1-x_1-x_2)^{d-1} \quad dx_2 \qquad (D = 2)$$

$$= \frac{1}{(d-1)!}\left[-\frac{(1-x_1-x_2)^d}{d}\right]_0^{1-x_1}$$

$$= \frac{1}{(d-1)!}(-\frac{(1-x_1-(1-x_1))^d}{d} + \frac{(1-x_1-0)^d}{d})$$

$$= \frac{1}{d!}(1-x_1)^d$$

Induction Step: Suppose the induction hypothesis true for $D-1$. We now show it is true for D.

$$\int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-1} x_i} \frac{1}{(d-(D-1))!}(1-\sum_{i=1}^{D} x_i)^{d-(D-1)} \quad dx_D \ldots dx_2 =$$

$$= \int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-2} x_i} \frac{1}{(d-(D-1))!}\left[-\frac{(1-\sum_{i=1}^{D} x_i)^{d-(D-2)}}{d-(D-2)}\right]_0^{1-\sum_{i=1}^{D-1} x_i} dx_{D-1} \ldots dx_2$$

$$= \int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-2} x_i} \frac{1}{(d-(D-2))!}\left[-(1-\sum_{i=1}^{D-1} x_i - x_D)^{d-(D-2)}\right]_0^{1-\sum_{i=1}^{D-1} x_i} dx_{D-1} \ldots dx_2$$

$$= \int_0^{1-x_1} \cdots \int_0^{1-\sum_{i=1}^{D-2} x_i} \frac{(1-\sum_{i=1}^{D-1} x_i)^{d-(D-2)}}{(d-(D-2))!} \quad dx_{D-1} \ldots dx_2$$

$$= \frac{1}{d!}(1-x_1)^d \qquad \text{(Ind. Hyp.)}$$

We have now proved the induction hypothesis for any values of d and D such that $2 \leq D \leq d$. In particular it is true for $d = D$, and we have proven the Sublemma.

End of Proof of Sublemma 2 □

Sublemma 3 *The largest possible volume that T_d^l and T_d^u can have is a $\frac{1}{(d+1)!}$-th of the volume of a full tile.*

Proof of Sublemma 3: Let us start by rotating each T_d^u upside-down and both T_d^l and T_d^u for all d's, such that their bottom surfaces can be described by the equations $x_K = \sum_{i=1}^{d} a_i^l x_i$ and $x_K = \sum_{i=1}^{d} a_i^u x_i$, respectively. Since T_d^l doesn't have any predecessors and T_d^u does not have any successors for any values of d, we know that $\forall i : 1 \leq i \leq d$ ($a_i^l \geq 1$ and $a_i^u \geq 1$). The volumes of T_d^l and T_d^u are therefore going to be smaller or equal to the volume of a tile that has its bottom surface described by the equation $x_K = \sum_{i=1}^{d} x_i$. We call this volume V_d. Integrating over this tile we get the following derivation.

$$
\begin{aligned}
V_d = & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-1} x_i} \int_0^1 \dots \int_0^1 \int_{\sum_{i=1}^{d} x_i}^1 dx_K \dots dx_1 \\
= & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-1} x_i} \int_0^1 \dots \int_0^1 1 - \sum_{i=1}^{d} x_i \quad dx_{K-1} \dots dx_1 \\
= & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-1} x_i} 1 - \sum_{i=1}^{d} x_i \quad dx_d \dots dx_1 \\
= & \int_0^1 \frac{1}{d!}(1-x_1)^d \quad dx_1 \qquad \text{(Sublemma 2)} \\
= & \frac{1}{d!}\left[-\frac{(1-x_1)^{d+1}}{d+1}\right]_0^1 \\
= & \frac{1}{(d+1)!}
\end{aligned}
$$

End of Proof of Sublemma 3 □

Sublemma 4 *The largest possible tile surface that needs to be communicated between T_d^l and T_{d-1}^l or between T_{d-1}^u and T_d^u is a $\frac{1}{d!}$ of a full tile surface.*

Proof of Sublemma 4: As stated in Sublemma 3, the volumes of T_d^l and T_d^u are going to be smaller or equal to the volume of a tile that has its bottom surface described by the equation $x_K = \sum_{i=1}^{d} x_i$. Let us call this tile T. Let V_d be the volume of the tile surface communicated between T_d^l and T_{d-1}^l or between T_{d-1}^u and T_d^u. V_d is smaller or equal to the integral over T in all dimensions but d, with $x_d = 0$. Integrating over T in this manner we get the following derivation for the upper bound on the communication surface between T_d^l and T_{d-1}^l, and T_{d-1}^u and T_d^u.

$$
\begin{aligned}
V_d = & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-2} x_i} \int_0^1 \dots \int_0^1 \int_{\sum_{i=1}^{d-1} x_i}^1 dx_K \dots dx_{d+1} dx_{d-1} \dots dx_1 \\
= & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-2} x_i} \int_0^1 \dots \int_0^1 1 - \sum_{i=1}^{d-1} x_i \quad dx_{K-1} \dots dx_{d+1} dx_{d-1} \dots dx_1 \\
= & \int_0^1 \int_0^{1-x_1} \dots \int_0^{1-\sum_{i=1}^{d-2} x_i} 1 - \sum_{i=1}^{d-1} x_i \quad dx_{d-1} \dots dx_1 \\
= & \int_0^1 \frac{1}{(d-1)!}(1-x_1)^{d-1} \quad dx_1 \qquad \text{(Sublemma 2)} \\
= & \frac{1}{(d-1)!}\left[-\frac{(1-x_1)^d}{d}\right]_0^1 \\
= & \frac{1}{d!}
\end{aligned}
$$

End of Proof of Sublemma 4 □

As stated above, the biggest difference in the execution times using the two models occurs when $\forall d$ with $1 \leq d \leq K-1$ T_d^l and T_d^u lie on $\hat{p}$ and T^l and T^u are the first and last tiles on p respectively. We therefore get the following derivation for $\hat{E}_i - E_i$.

$$
\begin{aligned}
\hat{E}_i - E_i = \;& 2E_t \sum_{d=1}^{K-1} (\frac{1}{(d+1)!} + c_d \frac{1}{d!}) && \text{(Sublemma 3, 4)} \\
\leq \;& 2E_t \sum_{d=1}^{K-1} (\frac{1}{(d+1)!} + c \frac{1}{d!}) && \text{(if } c = \max_d [c_d]) \\
\leq \;& 2E_t (\sum_{d=1}^{K-1} \frac{1}{(d+1)!} + c \sum_{d=1}^{K-1} \frac{1}{d!}) \\
\leq \;& 2E_t (\sum_{d=2}^{K} \frac{1}{d!} + c \sum_{d=1}^{K-1} \frac{1}{d!}) \\
\leq \;& 2E_t (\sum_{d=2}^{\infty} \frac{1}{d!} + c \sum_{d=1}^{\infty} \frac{1}{d!}) \\
\leq \;& 2E_t (\sum_{d=0}^{\infty} \frac{1}{d!} - 2 + c(\sum_{d=0}^{\infty} \frac{1}{d!} - 1)) \\
\leq \;& 2E_t (e - 2 + c(e-1)) \\
\leq \;& 2E_t (1 + 2c)
\end{aligned}
$$

where $c = \max_d [c_d]$.

End of Proof of Lemma 1 □

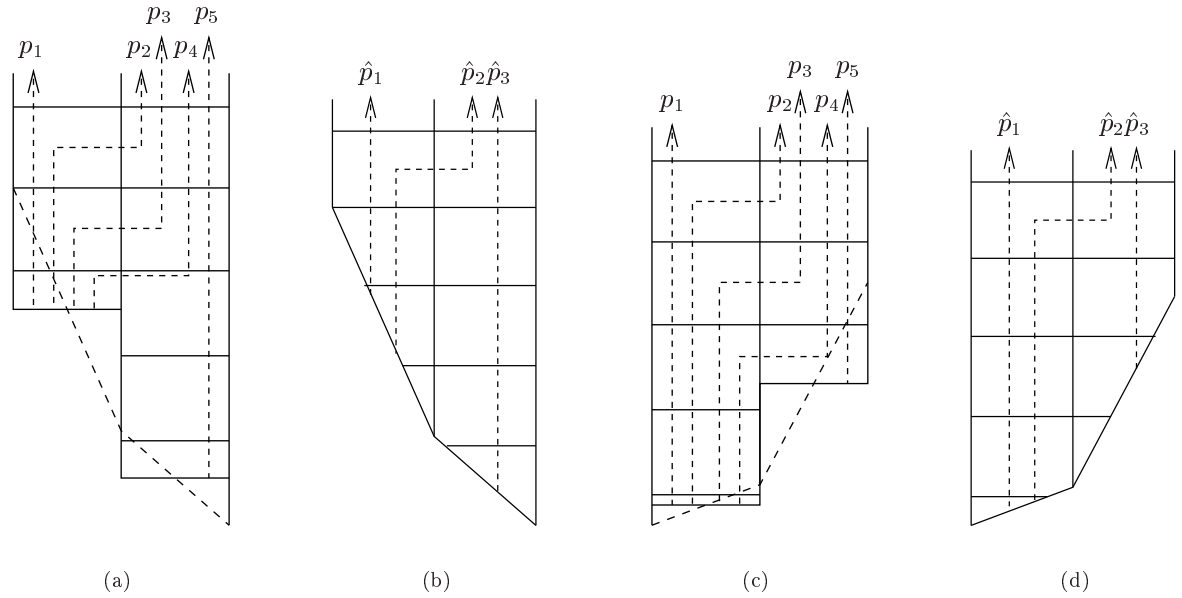


Figure 6: *Paths in a portion of an iteration space modeled ((a) and (c)) using the model presented in this paper, and ((b) and (d)) using the model in [HCF97]. We see that for each path in (a) and (c) there exists a path in (b) and (d) with equal or greater length. Comparing (a) and (b) we see that* $\hat{p}_1$ *has the same length as* p_1*, and* $\hat{p}_3$ *as* p_5*, and* $\hat{p}_2$ *is as long as or longer than* p_2*,* p_3*, and* p_4*. Same is true when comparing (c) and (d) ;* $\hat{p}_1$ *has the same length as* p_1*,* $\hat{p}_3$ *as* p_5*, and* $\hat{p}_2$ *is as long as or longer than* p_2*,* p_3*, and* p_4*. The longest path of dependent tiles in (a) and (c) can therefore not be longer than the longest path of dependent tiles in (b) and (d).*

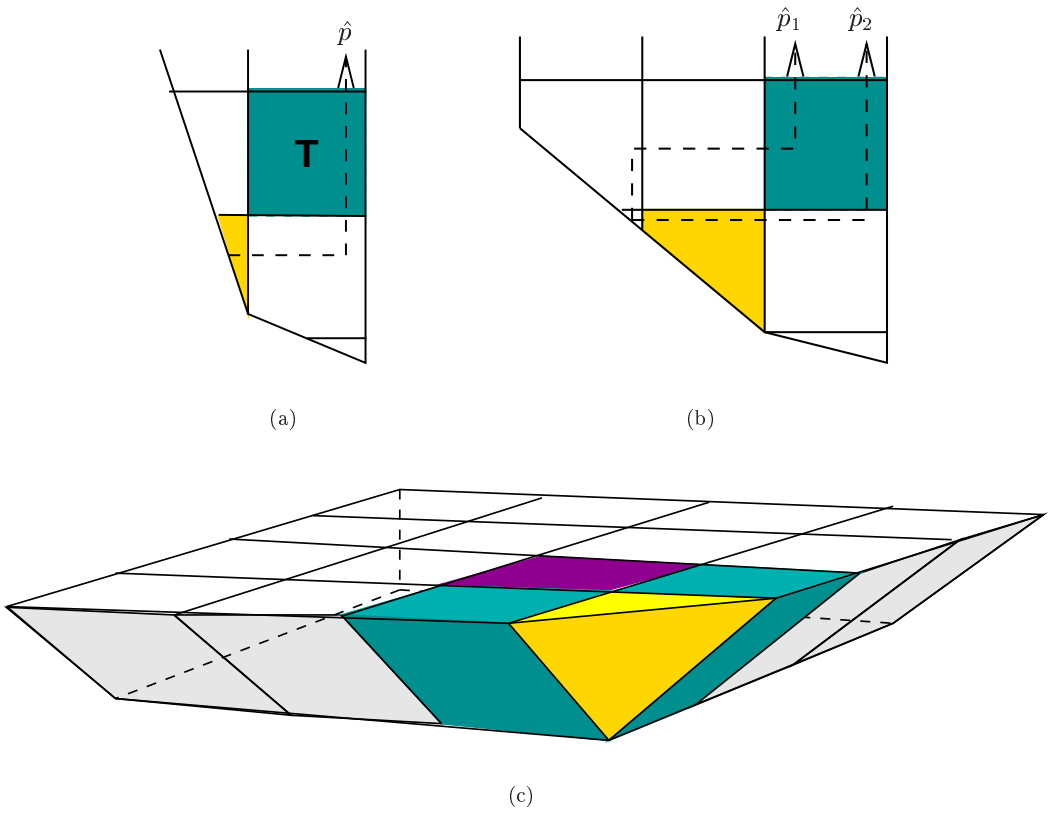


Figure 7: *In (a) we show a 2-dimensional iteration space with $\hat{p}$, the longest path of dependent tiles, and T, the first full tile on $\hat{p}$, marked. The largest possible volume the lightly shaded tile, i.e. the last partial tile on $\hat{p}$ not in the same stack as T, can have is half of the volume of a full tile. If it would have had a larger volume as depicted in (b) we see that $\hat{p}_1$ is longer than $\hat{p}_2$. $\hat{p}_2$ can therefore not be $\hat{p}$, and the dark tile in (b) is therefore not the first full tile on $\hat{p}$ anymore, and the lightly shaded tile is not the last partial tile on $\hat{p}$ not in the same stack as T. (c) shows a 3-dimensional iteration space. The first full tile on $\hat{p}$ is the darkest tile, and all the shaded partial tiles are of maximum size. The lightest shaded tile has a volume of a $\frac{1}{3!}$-th of a full tile and the medium shaded tile has a volume of a $\frac{1}{2!}$-th of a full tile.*

4 Execution time of a convex iteration space

In this section we show that the problem of determining the execution time of a general convex iteration space is equally hard as general linear programming. We start by showing that we can solve it by linear programming. This direction was shown in [DKR91] using a different model. In that model all tiles were assumed to be full, and to reduce the effect of the error of this assumption, the iteration spaces were assumed to be sufficiently large.
Direction 1 - Determining the execution time is no harder than linear programming: The execution time of an iteration space is given by the execution time of the tiles on the path within the iteration space that takes the longest to execute. We define the *initial* and *final* synchronization points to be the end points on this path.

Definition 2 *Let $\vec{z}^{\,i}$ and $\vec{z}^{f}$ be two points within a polytope such that $E(\vec{z}^{\,i}, \vec{z}^{f}) = \max_{\vec{p},\vec{q}}(E(\vec{p},\vec{q}))$. We call $\vec{z}^{\,i}$ and $\vec{z}^{f}$ the* initial *and* final synchronization points *of that polytope.*

The problem of determining the $E(\vec{z}^{\,i}, \vec{z}^{f})$ for a general convex iteration space bounded by M_{tot} constraints can then be formulated as the following linear programming problem.

Constraints:

// First M_{tot} constraints ensure the starting point,
// $(x_1, \ldots, x_K)$, of the longest path of dependent tiles
// lies within iteration space.

$$\forall i : 1 \leq i \leq M_b$$
$$(\textstyle\sum_{d=1}^{K-1} s_d^i x_d - x_K \quad \leq \quad -n_i)$$
$$\forall i : M_b + 1 \leq i \leq M_b + M_t$$
$$(-\textstyle\sum_{d=1}^{K-1} s_d^i x_d + x_K \quad \leq \quad n_i)$$
$$\forall i : M_b + M_t + 1 \leq i \leq M_b + M_t + M_l$$
$$(-x_i \quad \leq \quad -n_i)$$
$$\forall i : M_b + M_t + M_l + 1 \leq i \leq M_{tot}$$
$$(x_i \quad \leq \quad n_i)$$

// Next M_{tot} constraints ensure end point, $(x_{K+1}, \ldots, x_{2K})$,
// of longest path of dependent tiles lies within iteration space.

$$\forall i : M_{tot} + 1 \leq i \leq M_{tot} + M_b$$
$$(\textstyle\sum_{d=1}^{K-1} s_d^{i-M_{tot}} x_{K+d} - x_{2K} \quad \leq \quad -n_{i-M_{tot}})$$
$$\forall i : M_{tot} + M_b + 1 \leq i \leq M_{tot} + M_b + M_t$$
$$(-\textstyle\sum_{d=1}^{K-1} s_d^{i-M_{tot}} x_{K+d} + x_{2K} \quad \leq \quad n_{i-M_{tot}})$$
$$\forall i : M_{tot} + M_b + M_t + 1 \leq i \leq 2M_{tot} - M_r$$
$$(-x_{K+i-M_{tot}-M_b-M_t} \quad \leq \quad -n_{i-M_{tot}})$$
$$\forall i : 2M_{tot} - M_r + 1 \leq i \leq 2M_{tot}$$
$$(x_{K+i-M_{tot}-M_b-M_t-M_l} \quad \leq \quad n_{i-M_{tot}})$$

// Last K constraints ensure tile dependences are obeyed.

$$\forall i : 2M_{tot} + 1 \leq i \leq 2M_{tot} + K$$
$$(-x_{K+i-2M_{tot}} + x_{i-2M_{tot}} \quad \leq \quad 0)$$

Objective:

Maximize $\sum_{d=1}^{K} (x_{K+d} - x_d)(1 + c_d)$

where M_b, M_t, M_l, and M_r denote the number of facets defining the bottom, top, left and right side of the iteration space, respectively, and $M_{tot} = M_b + M_t + M_l + M_r$. Here and in the rest of this technical report we use $\sum_{d=1}^{K-1} (-s_d^i) x_d + x_K = n_i$ to denote the equation describing facet i.

Definition 3 *$\forall d : 1 \leq d \leq K-1$, let s_d^i be the slope in dimension d of facet i. We define $\sum_{d=1}^{K-1} (-s_d^i) x_d + x_K = n_i$ to be the equation of facet i.*

The objective function ensures that the solution is indeed the coordinates for the synchronization points, as defined in Definition 1 and 2. Note that we are maximizing $\sum_{d=1}^{K} (x_{K+d} - x_d)(1 + c_d)$ rather than E_t times this quantity (as in Definition 1) since E_t is a positive constant with respect to the tile shape, and does not affect the solution.

We have now shown that the problem of determining the execution time of a general planar convex iteration space can be solved with linear programming, and go on to showing the other direction.

Direction 2 - Determining the execution time is at least as hard as linear programming: Let A be an algorithm that determines the execution time of a given convex iteration space under our model, and let L be a general linear programming problem that we want to solve. We know that finding the solution to L is equally hard as determining whether there exists a solution to L [Kar91]. This means that the objective function of L is irrelevant and we can replace it with any other objective function without changing the answer of whether there exists a solution to L or not. We then transform L into a linear programming problem L' such that the solution space is finite [Chv83]. Now we give the convex polytope described by the constraints of L' as input to A. If A outputs a non-zero execution time, then we know there exist a solution to L' (and thus to L), otherwise there doesn't.

We have now shown that the problem of determining the execution time of a general planar convex iteration space is equally hard as general linear programming. For a general convex iteration space it is therefore not likely that we can derive a closed-form formula for the execution time. A formula would however be very useful both

for predicting the execution time given a tiling and also to find the tile shape that minimizes the execution time. In the next section we introduce *rectilinear iteration spaces*, a sub-class of convex iteration spaces, for which we can derive a closed-form formula for the execution time.

5 Rectilinear iteration spaces

The execution time of a tiled, convex iteration space is given by the execution time of the tiles on the path within the iteration space that takes the longest to execute (see Definition 1). The problem of finding this path can be formulated as a linear programming problem (see Section 4) and we can thus find the answer in polynomial time. This however only gives us a numeric value and not a closed-form, symbolic formula for the execution time. A closed-form solution directly allow us to determine the tile shape that optimizes the execution time. It can also be used to predict the execution time of a given tiling. Since finding a symbolic solution for a general convex iteration space corresponds to finding a symbolic solution to a general linear programming problem (see Section 4), we must further restrict the kind of iteration spaces we consider.

Rectilinear iteration spaces will be a sub-class of convex iteration spaces. For rectilinear iteration spaces we can use the duality property of a linear programming formulation to derive a closed-form, symbolic formula for the execution time. The key property that makes it easier to analyze rectilinear iteration spaces is the fact that the slopes of two adjacent facets differ only in one dimension. All 2-dimensional, convex iteration spaces are therefore rectilinear. See Figure 8 for an example of a 3-dimensional rectilinear iteration space.

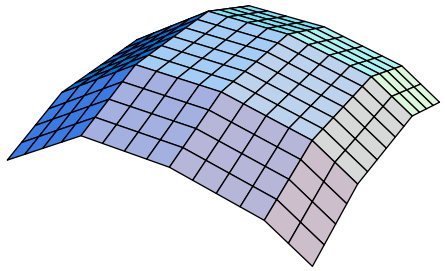


Figure 8: *The top of a 3-dimensional, rectilinear iteration space.*

A rectilinear iteration space is defined as follows.

Definition 4 *Let $T(x_1, \ldots, x_{K-1})$ and $B(x_1, \ldots, x_{K-1})$ be the functions describing the top and bottom of the iteration space respectively. A convex iteration space is rectilinear if*

$\forall i : 1 \leq i \leq K-1$ $(\frac{\partial B(x_1,\ldots,x_{K-1})}{\partial x_i} = g_i(x_i))$

and

$\forall i : 1 \leq i \leq K-1$ $(\frac{\partial T(x_1,\ldots,x_{K-1})}{\partial x_i} = h_i(x_i))$

where $g_i(x_i)$ and $h_i(x_i)$ are arbitrary[10] *functions of x_i.*

In other words, a K-dimensional, convex iteration space is rectilinear if the following condition holds.

> The intersection of any two facets defining the boundaries of the iteration space is either parallel or perpendicular to all coordinate axes.

The loop nests that generate convex iteration spaces with planar surfaces are all loop nests with loops of the form

$$\texttt{for } x_j \texttt{ = } \max_{\texttt{i}}(\texttt{f}_\texttt{i}) \texttt{ to } \min_{\texttt{k}}(\texttt{g}_\texttt{k})$$

where $\texttt{f}_\texttt{i}$ and $\texttt{g}_\texttt{k}$ are linear functions of less deeply nested iteration variables. For example, this includes all rectangular iteration spaces with any amount of skewing. For rectilinear iteration spaces, however, there are additional requirements that we describe next.

[10] By the convexity of the iteration space they must be monotone

From the loop bounds of a loop nest of depth K of the form given in Section 3, we get equations $x_K \geq m$, $x_K \leq n$, $x_{K-1} \geq f_1^{K-1}$, $x_{K-1} \geq f_2^{K-1}$, $\ldots$ $x_{K-1} \leq g_1^{K-1}$, $x_{K-1} \leq g_2^{K-1}$, $\ldots$, $x_1 \geq f_1^1$, $x_1 \geq f_2^1$, $\ldots$, $x_1 \leq g_1^1$, $x_1 \leq g_2^1$, $\ldots$. We get a system of equations of the following form.

$$-i_1^1 x_1 - i_2^1 x_2 - \ldots - i_{K-1}^1 x_{K-1} + x_K \geq n_1$$
$$-i_1^2 x_1 - i_2^2 x_2 - \ldots - i_{K-1}^2 x_{K-1} + x_K \geq n_2$$
$$\ldots$$
$$-i_1^{M_b} x_1 - i_2^{M_b} x_2 - \ldots - i_{K-1}^{M_b} x_{K-1} + x_K \geq n_{M_b}$$

$$-i_1^{M_b+1} x_1 - i_2^{M_b+1} x_2 - \ldots - i_{K-1}^{M_b+1} x_{K-1} + x_K \leq n_{M_b+1}$$
$$-i_1^{M_b+2} x_1 - i_2^{M_b+2} x_2 - \ldots - i_{K-1}^{M_b+2} x_{K-1} + x_K \leq n_{M_b+2}$$
$$\ldots$$
$$-i_1^{M_b+M_t} x_1 - i_2^{M_b+M_t} x_2 - \ldots - i_{K-1}^{M_b+M_t} x_{K-1} + x_K \leq n_{M_b+M_t}$$

$$x_1 \geq n_{M_b+M_t+1}$$
$$x_2 \geq n_{M_b+M_t+2}$$
$$\ldots$$
$$x_{K-1} \geq n_{M_b+M_t+K}$$

$$x_1 \leq n_{M_b+M_t+K}$$
$$x_2 \leq n_{M_b+M_t+K+1}$$
$$\ldots$$
$$x_{K-1} \leq n_{M_b+M_t+2K-2}$$

where M_b, and M_t are the number of facets defining the bottom, and top of the iteration space respectively. The M_b first equations are the equations describing the bottom of the iteration space. The following M_t equations describe the top, the next K the left side and the final K the right side. Note that there could be less than, but no more than $K-1$ equations defining the left and right sides.

Using this notation, a loop nest of depth K corresponds to a rectilinear iteration space if the following four conditions are true:

1. There exists a placement of equations 1 through M_b at grid-points in an contiguous, $(K-1)$-dimensional grid, such that no two neighboring equations i and j, have the property $\exists l, m : (i_l^i \neq i_l^j$ and $i_m^i \neq i_m^j)$.

2. There exists a placement of equations $M_b + 1$ through $M_b + M_t$ at grid-points in an contiguous, $(K-1)$-dimensional grid, such that no two neighboring equations i and j have the property $\exists l, m : (i_l^i \neq i_l^j$ and $i_m^i \neq i_m^j)$.

3. Consider the same grids as in condition 1 and 2. Let i and $i+1$ be any two neighboring equations at coordinate l and $l+1$, respectively, in dimension d, and let j and $j+1$ be any other two neighboring equations at coordinates l and $l+1$, respectively, in the same dimension d. The third condition is that $\forall i, j, l, d$ $(n_i - n_{i+1} = n_j - n_{j+1})$.

These conditions ensure that the slope only changes in one dimension at a time between any two neighboring surfaces. Note that these conditions are sufficient but not necessary for an iteration space to be rectilinear. For example, the conditions might not be met and the resulting iteration space still be rectilinear if one or more of the $M_b + M_t$ first equations are superfluous and do not contribute to the actual iteration space boundary.

6 Longest path of dependent tiles

According to Definition 1, to determine the execution time of an iteration space we need to determine the location of the initial and final synchronization points (Def. 2) of that iteration space. For general iteration space shapes we can locate the synchronization points by solving a linear programming problem (see Section 4). In this section we show how to find the synchronization points to a rectilinear iteration space without running a

linear programming algorithm. This result allows us to derive a *closed-form* formula for the execution time of the iteration space (Section 8). It also gives us an understanding as to how the longest path of dependent tiles changes with the shapes of the iteration space and the tiles. This is discussed in Section 7, where we also determine the tile shape that minimizes the length of this path.

Apart from the synchronization points, there are two other points that turn out to be of importance: the *bottom and top corner* of the iteration space. These are the coordinate points at which in all dimensions d the slope of the bordering facets is less than $-(1+c_d)$ on one side and greater or equal to $-(1+c_d)$ on the other side (see Figure 9). We define these points, $\vec{q}^{\,b}$ and $\vec{q}^{\,t}$, as follows.

Definition 5 *For a rectilinear iteration space, the* bottom corner, $\vec{q}^{\,b}$, *is the coordinate point such that* $\forall d$

1. *there exists no facet of the bottom surface of the iteration space at a smaller coordinate than* q_d^b *that has slope* $s_d \geq -(1+c_d)$, *and*
2. *there exists no facet of the bottom surface of the iteration space at a larger coordinate than* q_d^b *that has slope* $s_d < -(1+c_d)$.

Similarly, the top corner, $\vec{q}^{\,t}$, *is the coordinate point such that* $\forall d$

1. *there exists no facet of the top surface of the iteration space at a smaller coordinate than* q_d^t *that has slope* $s_d < -(1+c_d)$, *and*
2. *there exists no facet of the top surface of the iteration space at a larger coordinate than* q_d^t *that has slope* $s_d \geq -(1+c_d)$.

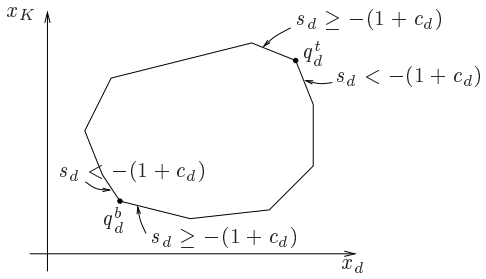


Figure 9: *A 2-dimensional projection of a K-dimensional iteration space. $\forall d : 1 \leq d \leq K-1$, the bottom corner (see Definition 5), q_d^b, is defined as the point along the bottom of the iteration space for which all facets to its left have slopes less than $-(1+c_d)$ and all facets to its right have slopes greater or equal to $-(1+c_d)$. The top corner, q_d^t, is defined similarly for all $d : 1 \leq d \leq K-1$. s_d refers to the slope in dimension d of the corresponding facet.*

The location of the synchronization points in each dimension changes if the bottom corner precedes the top corner in that dimension. In Theorem 1 we show that in all dimensions d where the tile slope is such that $q_d^b \leq q_d^t$, we have $z_d^i = q_d^b$ and $z_d^f = q_d^t$, and in all dimensions d such that $q_d^b > q_d^t$, z_d^i and z_d^f are the end points of the longest vertical path in that dimension (see Figure 10). To distinguish between these two kinds of dimensions, we introduce the following terminology.

Definition 6 $\forall d : 1 \leq d \leq K-1$, *if* $q_d^b \leq q_d^t$ *we call d a* forward *dimension.* $\forall d : 1 \leq d \leq K-1$, *if* $q_d^b > q_d^t$ *we call d a* backward *dimension. We call the set of all forward dimensions $\mathcal{F}$ and the set of all backward dimensions $\mathcal{B}$.*

We prove Theorem 1 by using the duality property of linear programming on the linear programming formulation from Section 4. As always, finding the solution to the dual problem is tricky since we have to "guess" the values of the dual variables. For rectilinear iteration spaces the two solution points to the primal problem are surrounded by 2^{K-1} facets, which limits the problem to finding the dual variables corresponding to those constraints. (The dual variables corresponding to the other constraints are all set to zero.) It turns out that K

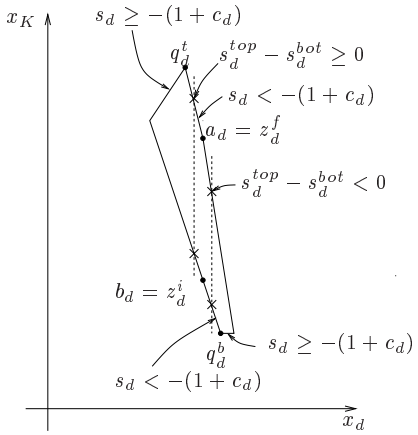


Figure 10: *A 2-dimensional projection of a K-dimensional iteration space. When the tile slope in dimension d is small enough so that d is a forward dimension (as in Figure 9), then the initial and final synchronization points (see Definition 2) are equal to q_d^b and q_d^t, respectively. When d is a backward dimension, as in this picture, then z_d^i and z_d^f are defined such that the path between them is the longest vertical path in the projection.*

of the 2^{K-1} variables should be picked to be a telescoping series that sums to 1. To determine which those K variables are, we need to order the constraints to which they correspond. The ordering is determined by the value of a certain measure that we provide. The variables are ordered so that the constraint corresponding to the i-th variable only differs in slope in dimension d compared to the constraint corresponding to the $(i-1)$-th variable, where d is the dimension in which the measure is the i-th largest for all dimensions. This will define a unique ordering of K variables (constraints). The rest of the 2^{K-1} variables of the dual problem are set to zero. Once we have "guessed" this solution to the dual problem, we prove that the objective function of the dual problem at that point is equal to the objective function of the primal problem at the synchronization points.

Theorem 1 *Let $\vec{i}$ and $\vec{j}$ be any two coordinate points on the top and bottom surface of the iteration space respectively, such that $\forall d : 1 \leq d \leq K-1$ $(i_d = j_d)$. Let $\vec{a}$ and $\vec{b}$ be the two points on the top and bottom of the iteration space respectively, such that $a_K - b_K = \max_{\vec{i},\vec{j}}[i_K - j_K]$.*

The location of the two synchronization points are then given by

$$\forall d \in \mathcal{F} \ (z_d^i = q_d^b, z_d^f = q_d^t)$$
$$\forall d \in \mathcal{B} \ (z_d^i = b_d = z_d^f = a_d)$$

Proof of Theorem 1: We first prove the theorem in Lemma 2 for the kinds of iteration spaces in which the bottom and top corners do not lie at neither the maximum or minimum coordinate in any dimension other than the vertical. An example of an iteration space not covered by Lemma 2 is one that does not have any top or bottom surfaces such that $s_d < -(1+c_d)$, e.g. a rectangle. Then we show that the location of the synchronization points are the same for those kinds of iteration spaces.

Lemma 2 *Consider an iteration space where $\forall d : 1 \leq d \leq K-1$, neither q_d^b nor q_d^t are equal to the smallest or greatest coordinate possible in dimension d. Let $\vec{i}$ and $\vec{j}$ be any two coordinate points on the top and bottom surface of the iteration space respectively, such that $\forall d : 1 \leq d \leq K-1$ $(i_d = j_d)$. Let $\vec{a}$ and $\vec{b}$ be the two points on the top and bottom of the iteration space respectively, such that $a_K - b_K = \max_{\vec{i},\vec{j}}[i_K - j_K]$.*

The locations of the two synchronization points are then given by

$$\forall d \in \mathcal{F} \ (z_d^i = q_d^b, z_d^f = q_d^t)$$
$$\forall d \in \mathcal{B} \ (z_d^i = b_d = z_d^f = a_d)$$

as long as a_d, b_d, q_d^b and q_d^t do not to lie at neither the maximum nor the minimum coordinate in any dimension d.

Proof of Lemma 2: To prove the lemma we use the duality property of a linear programming problem. We formulate the dual problem and guess a solution $\vec{v}$. We then need to prove four things, 1) that $\vec{z}^{\,i}$ and $\vec{z}^{f}$ satisfy the constraints of the primal problem, 2) that $\vec{v}$ satisfies the constraints of the dual problem, 3) that the objective function of the primal problem at $(z_1^i, \ldots, z_K^i, z_1^f, \ldots, z_K^f)$ is equal to the objective function of the dual problem at $\vec{v}$, and 4) that $\forall i : 1 \leq i \leq 2M_{tot} + K$ $(v_i \geq 0)$.[11] But first we need to introduce some terminology and notation.

Recall that the synchronization points in a rectilinear iteration space are each defined by the intersection of 2^{K-1} facets. For a given dimension d, there are only two slopes used in those facets. Using this fact, we define the four *defining facets* of an iteration space as follows.[12]

Definition 7 *Consider all the 2^{K-1} facets intersecting in the initial synchronization point, $\vec{z}^{\,i}$. For all dimensions d, let s_d^{low} and s_d^{high} be the two slopes in dimension d used in those facets such that $s_d^{low} \leq s_d^{high}$.*

The first defining facet, Facet 1, is the facet with equation $-\sum_{d=1}^{K-1} s_d^{low} x_d + x_K = n_1$.

$\forall d : 1 \leq d \leq K-1$, the slope in dimension d of Facet 1, s_d^1, is thus equal to s_d^{low}.

The second defining facet, Facet K, is the facet with equation $-\sum_{d=1}^{K-1} s_d^{high} x_d + x_K = n_K$.

$\forall d : 1 \leq d \leq K-1$, the slope in dimension d of Facet K, s_d^K, is thus equal to s_d^{high}.

Definition 8 *Consider all the 2^{K-1} facets intersecting in the final synchronization point, $\vec{z}^{f}$. For all dimensions d, let s_d^{low} and s_d^{high} be the two slopes in dimension d used in those facets such that $s_d^{low} \leq s_d^{high}$.*

The third defining facet, Facet $M_b + 1$, is the facet with equation $\sum_{d=1}^{K-1} s_d^{high} x_d - x_K = n_{M_b+1}$.

$\forall d : 1 \leq d \leq K-1$, the slope in dimension d of Facet $M_b + 1$, $s_d^{M_b+1}$, is thus equal to s_d^{high}.

The fourth defining facet, Facet $M_b + K$, is the facet with equation $\sum_{d=1}^{K-1} s_d^{low} x_d - x_K = n_{M_b+K}$.

$\forall d : 1 \leq d \leq K-1$, the slope in dimension d of Facet $M_b + K$, $s_d^{M_b+K}$, is thus equal to s_d^{low}.

Together with the definition of the synchronization points in the statement of this lemma, Definition 7 and 8 let us make the following observations.

Observation 1 $\forall d : 1 \leq d \leq K-1$ $(s_d^1 \leq s_d^K)$

Observation 2 $\forall d : 1 \leq d \leq K-1$ $(s_d^{M_b+1} \geq s_d^{M_b+K})$

Observation 3 $\forall d : 1 \leq d \leq K-1$ $(s_d^{M_b+1} - s_d^1 \geq 0)$

Observation 4 $\forall d : 1 \leq d \leq K-1$ $(s_d^{M_b+K} - s_d^K \leq 0)$

Observation 1 and 2 are true for the bottom and top of any convex iteration space, and Observation 3 and 4 follow from the construction of $\vec{a}$ and $\vec{b}$ in the formulation of the lemma. Remember that by Definitions 7 and 8, for each dimension d, s_d^1, s_d^K, $s_d^{M_b+1}$ and $s_d^{M_b+K}$ are the slopes surrounding $z_d^i = b_d$ and $z_d^f = a_d$ respectively.

Observation 5 $\forall d : 1 \leq d \leq K-1$ $(s_d^1 + 1 + c_d \leq 0)$

Observation 6 $\forall d \in \mathcal{F}$ $(s_d^K + 1 + c_d \geq 0)$

Observation 7 $\forall d \in \mathcal{F}$ $(s_d^{M_b+1} + 1 + c_d \geq 0)$

Observation 8 $\forall d : 1 \leq d \leq K-1$ $(s_d^{M_b+K} + 1 + c_d \leq 0)$

[11] $\forall d : 1 \leq d \leq K$ $(z_d^i \geq 0, z_d^f \geq 0)$ by construction.

[12] Note that these two definitions are identical to Definition 12 and 13, but we restate them here to avoid circular dependences.

The last four observations all follow from the construction of the initial and final synchronization points in the formulation of the Lemma, together with Definition 5, 7, 8. The observations regarding backward dimensions also use Definition 6 and the construction of $\vec{a}$ and $\vec{b}$.

We now formulate the dual problem as follows

Constraints:

$$\begin{aligned}
&\forall d : 1 \leq d \leq K-1 \\
&\quad (\textstyle\sum_{i=1}^{M_b} s_d^i y_i - \sum_{i=M_b+1}^{M_b+M_t} s_d^i y_i - \sum_{i=M_b+M_t+1}^{M_b+M_t+M_l} y_i + \\
&\qquad\qquad + \sum_{i=M_b+M_t+M_l+1}^{M_{tot}} y_i + y_{2M_{tot}+d} \quad \geq \quad -(1+c_d)) \\
&\qquad\qquad - \sum_{i=1}^{M_b} y_i + \sum_{i=M_b+1}^{M_b+M_t} y_i + y_{2M_{tot}+K} \quad \geq \quad -1 \\
&\forall d : 1 \leq d \leq K-1 \\
&\quad (\textstyle\sum_{i=M_{tot}+1}^{M_{tot}+M_b} s_d^{i-M_{tot}} y_i - \sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+M_t} s_d^{i-M_{tot}} y_i - \\
&\qquad - \sum_{i=M_{tot}+M_b+M_t+1}^{M_{tot}+M_b+M_t+M_l} y_i + \sum_{i=M_{tot}+M_b+M_t+M_l+1}^{2M_{tot}} y_i - y_{2M_{tot}+d} \quad \geq \quad 1+c_d) \\
&\qquad\qquad - \sum_{i=M_{tot}+1}^{M_{tot}+M_b} y_i + \sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+M_t} y_i - y_{2M_{tot}+K} \quad \geq \quad 1
\end{aligned}$$

Objective:

Minimize $-\sum_{i=1}^{M_b} n_i(y_i + y_{M_{tot}+i}) + \sum_{i=M_b+1}^{M_b+M_t} n_i(y_i + y_{M_{tot}+i}) - \sum_{i=M_b+M_t+1}^{M_b+M_t+M_l} n_i(y_i + y_{M_{tot}+i})$
$+ \sum_{i=M_b+M_t+M_l+1}^{M_{tot}} n_i(y_i + y_{M_{tot}+i})$ subject to the constraints.

and guess the following solution to this problem.[13]

$$\begin{aligned}
v_1 &= 1 - \tfrac{s^1_{\sigma(1)}+1+c_{\sigma(1)}+v_{2M_{tot}+\sigma(1)}}{s^1_{\sigma(1)}-s^K_{\sigma(1)}} \\
v_l &= \tfrac{s^1_{\sigma(l-1)}+1+c_{\sigma(l-1)}+v_{2M_{tot}+\sigma(l-1)}}{s^1_{\sigma(l-1)}-s^K_{\sigma(l-1)}} \\
&\quad - \tfrac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}} && (\forall l : 2 \leq l \leq K-1) \\
v_K &= \tfrac{s^1_{\sigma(K-1)}+1+c_{\sigma(K-1)}+v_{2M_{tot}+\sigma(K-1)}}{s^1_{\sigma(K-1)}-s^K_{\sigma(K-1)}} \\
v_i &= 0 && (\forall i : K+1 \leq i \leq M_{tot}+M_b) \\
v_{M_{tot}+M_b+1} &= 1 - \tfrac{s^{M_b+1}_{\pi(1)}+1+c_{\pi(1)}+v_{2M_{tot}+\pi(1)}}{s^{M_b+1}_{\pi(1)}-s^{M_b+K}_{\pi(1)}} \\
v_{M_{tot}+M_b+l} &= \tfrac{s^{M_b+1}_{\pi(l-1)}+1+c_{\pi(l-1)}+v_{2M_{tot}+\pi(l-1)}}{s^{M_b+1}_{\pi(l-1)}-s^{M_b+K}_{\pi(l-1)}} \\
&\quad - \tfrac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}} && (\forall l : 2 \leq l \leq K-1) \\
v_{M_{tot}+M_b+K} &= \tfrac{s^{M_b+1}_{\pi(K-1)}+1+c_{\pi(K-1)}+v_{2M_{tot}+\pi(K-1)}}{s^{M_b+1}_{\pi(K-1)}-s^{M_b+K}_{\pi(K-1)}} \\
v_i &= 0 && (\forall i : M_{tot}+M_b+K+1 \leq i \leq 2M_{tot}) \\
v_{2M_{tot}+d} &= \min\left[-(1+c_d+s^1_d), -(1+c_d+s^{M_b+K}_d)\right] && (\forall d \in \mathcal{B}) \\
v_{2M_{tot}+d} &= 0 && (\forall d \in \mathcal{F}) \\
v_{2M_{tot}+K} &= 0
\end{aligned}$$

where $\sigma : [1,\ldots,K] \rightarrow [1,\ldots,K]$ and $\pi : [1,\ldots,K] \rightarrow [1,\ldots,K]$ are the following two one-to-one ordering functions.

Definition 9 *$\forall l : 1 \leq l \leq K-1$ $(\sigma(l) = d)$ where d is the dimension with the l-th largest value of the fraction* $\frac{s^1_d+1+c_d+v_{2M_{tot}+d}}{s^1_d-s^K_d}$.

Definition 10 *$\forall l : 1 \leq l \leq K-1$ $(\pi(l) = d)$ where d is the dimension with the l-th largest value of the fraction* $\frac{s^{M_b+1}_d+1+c_d+v_{2M_{tot}+d}}{s^{M_b+1}_d-s^{M_b+K}_d}$.

[13] To reduce the amount of notation needed, we define $\forall l : s^1_{\sigma(l)} = s^K_{\sigma(l)}$ $(\frac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}} = 0)$ and $\forall l : s^{M_b+1}_{\pi(l)} = s^{M_b+K}_{\pi(l)}$ $(\frac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}} = 0)$, since in those dimensions both the numerator and the denominator evaluate to zero. The numerator is equal to zero when the corresponding denominator is equal to zero since then we know l is a backward dimension and by Observation 3 and 4.

We make the following six observations about the solution to the dual problem.

Observation 9 $\sum_{i=1}^{K} v_i = 1$

Observation 10 $\sum_{i=1}^{K} v_{M_{tot}+M_b+i} = 1$

Observation 11 $\sum_{i=1}^{l} v_i = 1 - \frac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}}$

Observation 12 $\sum_{i=1}^{l} v_{M_{tot}+M_b+i} = 1 - \frac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}}$

Observation 13 $\frac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}} \leq \frac{s^1_{\sigma(l-1)}+1+c_{\sigma(l-1)}+v_{2M_{tot}+\sigma(l-1)}}{s^1_{\sigma(l-1)}-s^K_{\sigma(l-1)}}$

Observation 14 $\frac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}} \leq \frac{s^{M_b+1}_{\pi(l-1)}+1+c_{\pi(l-1)}+v_{2M_{tot}+\pi(l-1)}}{s^{M_b+1}_{\pi(l-1)}-s^{M_b+K}_{\pi(l-1)}}$

Now we are almost ready. We only need four more observations. Consider using σ to order the facets such that $\forall j : 1 \leq j \leq K$ Facet j is the facet with equation $\sum_{l=1}^{j-1} s^K_{\sigma(l)} x_{\sigma(l)} + \sum_{l=j}^{K-1} s^1_{\sigma(l)} x_{\sigma(l)} - x_K = -n_j$, and π to order the facets such that $\forall j : 1 \leq j \leq K$ Facet $M_b + j$ is the facet with equation $-\sum_{l=1}^{j-1} s^{M_b+K}_{\pi(l)} x_{\pi(l)} - \sum_{l=j}^{K-1} s^{M_b+1}_{\pi(l)} x_{\pi(l)} + x_K = n_{M_b+j}$. We know such orderings are possible by Definition 7 and 8.

Observation 15 *$\exists$ an ordering of K of the 2^{K-1} facets surrounding the initial synchronization point such that $\forall j : 1 \leq j \leq K$ the equation for Facet j is $\sum_{l=1}^{j-1} s^K_{\sigma(l)} x_{\sigma(l)} + \sum_{l=j}^{K-1} s^1_{\sigma(l)} x_{\sigma(l)} - x_K = -n_j$.*

Similarly, $\exists$ an ordering of K of the 2^{K-1} facets surrounding the final synchronization point such that $\forall j : 1 \leq j \leq K$ the equation for Facet $M_b + j$ is $-\sum_{l=1}^{j-1} s^{M_b+K}_{\pi(l)} x_{\pi(l)} - \sum_{l=j}^{K-1} s^{M_b+1}_{\pi(l)} x_{\pi(l)} + x_K = n_{M_b+j}$.

But we could also pick the $2K$ constraints in the following way.

Observation 16 *$\exists$ an ordering of K of the 2^{K-1} facets surrounding the initial synchronization point such that $\forall j : 1 \leq j \leq K$ the equation for Facet j is $\sum_{d=1}^{j-1} s^K_d x_d + \sum_{d=j}^{K-1} s^1_d x_d - x_K = -n_j$.*

Similarly, $\exists$ an ordering of K of the 2^{K-1} facets surrounding the final synchronization point such that $\forall j : 1 \leq j \leq K$ the equation for Facet $M_b + j$ is $-\sum_{d=1}^{j-1} s^{M_b+K}_d x_d - \sum_{d=j}^{K-1} s^{M_b+1}_d x_d + x_K = n_{M_b+j}$.

Using the latter ordering, subtracting the equation for Facet d from the equation for Facet $d+1$, and the equation for Facet $M_b + d$ from the equation for Facet $M_b + d + 1$, gives us the following coordinates for the initial and final synchronization point. (The coordinates for z^i_K and z^f_K come from solving the equations of Facet 1 and Facet $M_b + 1$, respectively.)[14]

Observation 17

$\forall d : 1 \leq d \leq K-1$, $z^i_d = \frac{n_{d+1}-n_d}{s^1_d - s^K_d}$

$\forall d : 1 \leq d \leq K-1$, $z^f_d = \frac{n_{M_b+d+1}-n_{M_b+d}}{s^{M_b+1}_d - s^{M_b+K}_d}$

$z^i_K = \sum_{d=1}^{K-1} s^1_d z^i_d + n_1$

$z^f_K = \sum_{d=1}^{K-1} s^{M_b+1}_d z^f_d + n_{M_b+1}$

On the other hand, subtracting the equation for Facet l from the equation for Facet $l+1$ given in Observation 15, we get $\forall l : 1 \leq l \leq K-1$, $z^i_{\sigma(l)} = \frac{n_{l+1}-n_l}{s^1_{\sigma(l)} - s^K_{\sigma(l)}}$. This can also be written as $\forall d : 1 \leq d \leq K-1$, $z^i_d = \frac{n_{\sigma^{-1}(d)+1}-n_{\sigma^{-1}(d)}}{s^1_d - s^K_d}$, since $\forall d : 1 \leq d \leq K-1$ $\exists l : \sigma(l) = d$. Combining this expression with Observation 17, we make the following observation.

[14] Note that this observation is identical to Observation 20, but we repeat it here to avoid circular dependences.

Observation 18

$\forall d : 1 \leq d \leq K-1 \quad \frac{n_{d+1}-n_d}{s_d^1 - s_d^K} = \frac{n_{\sigma^{-1}(d)+1}-n_{\sigma^{-1}(d)}}{s_d^1 - s_d^K}$

$\forall d : 1 \leq d \leq K-1 \quad \frac{n_{M_b+d+1}-n_{M_b+d}}{s_d^{M_b+1} - s_d^{M_b+K}} = \frac{n_{M_b+\pi^{-1}(d)+1}-n_{M_b+\pi^{-1}(d)}}{s_d^{M_b+1} - s_d^{M_b+K}}$

Let us now go on by proving the the four steps.

Step 1: True by construction.

Step 2:

Constraint d $\forall d : 1 \leq d \leq K-1$**:**

$\sum_{i=1}^{M_b} s_d^i y_i - \sum_{i=M_b+1}^{M_b+M_t} s_d^i y_i - \sum_{i=M_b+M_t+1}^{M_b+M_t+M_l} y_i +$
$+ \sum_{i=M_b+M_t+M_l+1}^{M_{tot}} y_i + y_{2M_{tot}+d} =$

$= \sum_{i=1}^{K} s_d^i v_i + v_{2M_{tot}+d}$ (Def. of $\vec{v}$)

$= \sum_{i=1}^{K} s_{\sigma(l)}^i v_i + v_{2M_{tot}+\sigma(l)}$ ($\forall d : 1 \leq d < K$, $\exists l : (\sigma(l) = d)$)

$= \sum_{i=1}^{l} s_{\sigma(l)}^1 v_i + \sum_{i=l+1}^{K} s_{\sigma(l)}^K v_i + v_{2M_{tot}+\sigma(l)}$ (Obs. 15)

$= s_{\sigma(l)}^1 \sum_{i=1}^{l} v_i + s_{\sigma(l)}^K \sum_{i=l+1}^{K} v_i + v_{2M_{tot}+\sigma(l)}$

$= s_{\sigma(l)}^1 \sum_{i=1}^{l} v_i + s_{\sigma(l)}^K (1 - \sum_{i=1}^{l} v_i) + v_{2M_{tot}+\sigma(l)}$ (Obs. 9)

$= (s_{\sigma(l)}^1 - s_{\sigma(l)}^K) \sum_{i=1}^{l} v_i + s_{\sigma(l)}^K + v_{2M_{tot}+\sigma(l)}$

$= (s_{\sigma(l)}^1 - s_{\sigma(l)}^K)(1 - \frac{s_{\sigma(l)}^1 + 1 + c_{\sigma(l)} + v_{2M_{tot}+\sigma(l)}}{s_{\sigma(l)}^1 - s_{\sigma(l)}^K}) + s_{\sigma(l)}^K + v_{2M_{tot}+\sigma(l)}$ (Obs. 11)

$= (s_{\sigma(l)}^1 - s_{\sigma(l)}^K) \frac{-(s_{\sigma(l)}^K + 1 + c_{\sigma(l)} + v_{2M_{tot}+\sigma(l)})}{s_{\sigma(l)}^1 - s_{\sigma(l)}^K} + s_{\sigma(l)}^K + v_{2M_{tot}+\sigma(l)}$

$= -(s_{\sigma(l)}^K + 1 + c_{\sigma(l)} + v_{2M_{tot}+\sigma(l)}) + s_{\sigma(l)}^K + v_{2M_{tot}+\sigma(l)}$

$= -(1 + c_{\sigma(l)})$

$\geq -(1 + c_d)$

Constraint K**:**

$-\sum_{i=1}^{M_b} y_i + \sum_{i=M_b+1}^{M_b+M_t} y_i + y_{2M_{tot}+K} = -\sum_{i=1}^{K} v_i$ (Def. of $\vec{v}$)

≥ -1 (Obs. 9)

Constraint d $\forall d : K+1 \leq d \leq 2K-1$**:**

$\sum_{i=M_{tot}+1}^{M_{tot}+M_b} s_d^{i-M_{tot}} y_i - \sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+M_t} s_d^{i-M_{tot}} y_i - \sum_{i=M_{tot}+M_b+M_t+1}^{M_{tot}+M_b+M_t+M_l} y_i +$
$+ \sum_{i=M_{tot}+M_b+M_t+M_l+1}^{2M_{tot}} y_i - y_{2M_{tot}+d} =$

$= -\sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+K} s_d^{i-M_{tot}} v_i - v_{2M_{tot}+d}$ (Def. of $\vec{v}$)

$= -\sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+K} s_{\pi(l)}^{i-M_{tot}} v_i - v_{2M_{tot}+\pi(l)}$ ($\forall d : 1 \leq d < K$, $\exists l : (\pi(l) = d)$)

$= -\sum_{i=1}^{l} s_{\pi(l)}^{M_b+1} v_{M_{tot}+M_b+i} - \sum_{i=l+1}^{K} s_{\pi(l)}^{M_b+K} v_{M_{tot}+M_b+i} - v_{2M_{tot}+\pi(l)}$ (Obs. 15)

$= -s_{\pi(l)}^{M_b+1} \sum_{i=1}^{l} v_{M_{tot}+M_b+i} - s_{\pi(l)}^{M_b+K} \sum_{i=l+1}^{K} v_{M_{tot}+M_b+i} - v_{2M_{tot}+\pi(l)}$

$= -s_{\pi(l)}^{M_b+1} \sum_{i=1}^{l} v_{M_{tot}+M_b+i} - s_{\pi(l)}^{M_b+K} (1 - \sum_{i=1}^{l} v_{M_{tot}+M_b+i}) - v_{2M_{tot}+\pi(l)}$ (Obs. 10)

$= (s_{\pi(l)}^{M_b+K} - s_{\pi(l)}^{M_b+1}) \sum_{i=1}^{l} v_{M_{tot}+M_b+i} - s_{\pi(l)}^{M_b+K} - v_{2M_{tot}+\pi(l)}$

$= (s_{\pi(l)}^{M_b+K} - s_{\pi(l)}^{M_b+1})(1 - \frac{s_{\pi(l)}^{M_b+1} + 1 + c_{\pi(l)} + v_{2M_{tot}+\pi(l)}}{s_{\pi(l)}^{M_b+1} - s_{\pi(l)}^{M_b+K}}) - s_{\pi(l)}^{M_b+K} - v_{2M_{tot}+\pi(l)}$ (Obs. 12)

$= (s_{\pi(l)}^{M_b+K} - s_{\pi(l)}^{M_b+1}) \frac{-(s_{\pi(l)}^{M_b+K} + 1 + c_{\pi(l)} + v_{2M_{tot}+\pi(l)})}{s_{\pi(l)}^{M_b+1} - s_{\pi(l)}^{M_b+K}} - s_{\pi(l)}^{M_b+K} - v_{2M_{tot}+\pi(l)}$

$= s_{\pi(l)}^{M_b+K} + 1 + c_{\pi(l)} + v_{2M_{tot}+\pi(l)} - s_{\pi(l)}^{M_b+K} - v_{2M_{tot}+\pi(l)}$

$= 1 + c_{\pi(l)}$

$\geq 1 + c_d$

Constraint $2K$**:**

$-\sum_{i=M_{tot}+1}^{M_{tot}+M_b} y_i + \sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+M_t} y_i - y_{2M_{tot}+K} = \sum_{i=M_{tot}+M_b+1}^{M_{tot}+M_b+K} v_i$ (Def. of $\vec{v}$)

≥ 1 (Obs. 10)

Step 3: To compare the two objective functions, we first calculate the objective function for the dual problem and then for the primal.

Dual objective function:

$$
\begin{aligned}
&-\textstyle\sum_{i=1}^{M_b} n_i(v_i + v_{M_{tot}+i}) + \sum_{i=M_b+1}^{M_b+M_t} n_i(v_i + v_{M_{tot}+i}) - \sum_{i=M_b+M_t+1}^{M_b+M_t+M_l} n_i(v_i + v_{M_{tot}+i}) \\
&+\textstyle\sum_{i=M_b+M_t+M_l+1}^{M_{tot}} n_i(v_i + v_{M_{tot}+i}) = \\
=\ & -\textstyle\sum_{i=1}^{K} n_i v_i + \sum_{i=1}^{K} n_{M_b+i} v_{M_{tot}+M_b+i} && \text{(Def. of } \vec{v}) \\
=\ & -(1 - \tfrac{s^1_{\sigma(1)}+1+c_{\sigma(1)}+v_{2M_{tot}+\sigma(1)}}{s^1_{\sigma(1)}-s^K_{\sigma(1)}})n_1 \\
& -\textstyle\sum_{l=2}^{K-1} (\tfrac{s^1_{\sigma(l-1)}+1+c_{\sigma(l-1)}+v_{2M_{tot}+\sigma(l-1)}}{s^1_{\sigma(l-1)}-s^K_{\sigma(l-1)}} - \tfrac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}})n_l \\
& -\tfrac{s^1_{\sigma(K-1)}+1+c_{\sigma(K-1)}+v_{2M_{tot}+\sigma(K-1)}}{s^1_{\sigma(K-1)}-s^K_{\sigma(K-1)}} n_K + (1 - \tfrac{s^{M_b+1}_{\pi(1)}+1+c_{\pi(1)}+v_{2M_{tot}+\pi(1)}}{s^{M_b+1}_{\pi(1)}-s^{M_b+K}_{\pi(1)}})n_{M_b+1} \\
& +\textstyle\sum_{l=2}^{K-1} (\tfrac{s^{M_b+1}_{\pi(l-1)}+1+c_{\pi(l-1)}+v_{2M_{tot}+\pi(l-1)}}{s^{M_b+1}_{\pi(l-1)}-s^{M_b+K}_{\pi(l-1)}} - \tfrac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}})n_{M_b+l} \\
& +\tfrac{s^{M_b+1}_{\pi(K-1)}+1+c_{\pi(K-1)}+v_{2M_{tot}+\pi(K-1)}}{s^{M_b+1}_{\pi(K-1)}-s^{M_b+K}_{\pi(K-1)}} n_{M_b+K} && \text{(Def. of } \vec{v})
\end{aligned}
$$

Primal objective function:

$$
\begin{aligned}
&\textstyle\sum_{d=1}^{K} (x_{K+d} - x_d)(1 + c_d) = \\
=\ & z^f_K - z^i_K + \textstyle\sum_{d=1}^{K-1} (z^f_d - z^i_d)(1 + c_d) && (c_K = 0) \\
=\ & \textstyle\sum_{d=1}^{K-1} (1 + c_d + s^{M_b+1}_d) z^f_d + n_{M_b+1} - \sum_{d=1}^{K-1} (1 + c_d + s^1_d) z^i_d - n_1 && \text{(Obs 17)} \\
=\ & \textstyle\sum_{d=1}^{K-1} (1 + c_d + s^{M_b+1}_d + v_{2M_{tot}+d}) z^f_d + n_{M_b+1} \\
& -\textstyle\sum_{d=1}^{K-1} (1 + c_d + s^1_d + v_{2M_{tot}+d}) z^i_d - n_1 && (\forall d \in \mathcal{B}\ (z^i_d = z^f_d)) \\
=\ & \textstyle\sum_{d=1}^{K-1} (1 + c_d + s^{M_b+1}_d + v_{2M_{tot}+d}) \tfrac{n_{M_b+d+1} - n_{M_b+d}}{s^{M_b+1}_d - s^{M_b+K}_d} + n_{M_b+1} \\
& -\textstyle\sum_{d=1}^{K-1} (1 + c_d + s^1_d + v_{2M_{tot}+d}) \tfrac{n_{d+1} - n_d}{s^1_d - s^K_d} - n_1 && \text{(Obs. 17)} \\
=\ & \textstyle\sum_{d=1}^{K-1} (1 + c_d + s^{M_b+1}_d + v_{2M_{tot}+d}) \tfrac{n_{M_b+\pi^{-1}(d)+1} - n_{M_b+\pi^{-1}(d)}}{s^{M_b+1}_d - s^{M_b+K}_d} + n_{M_b+1} \\
& -\textstyle\sum_{d=1}^{K-1} (1 + c_d + s^1_d + v_{2M_{tot}+d}) \tfrac{n_{\sigma^{-1}(d)+1} - n_{\sigma^{-1}(d)}}{s^1_d - s^K_d} - n_1 && \text{(Obs. 18)} \\
=\ & \textstyle\sum_{d=1}^{K-1} (1 + c_d + s^{M_b+1}_d + v_{2M_{tot}+d}) \tfrac{n_{M_b+\pi^{-1}(d)+1} - n_{M_b+\pi^{-1}(d)}}{s^{M_b+1}_d - s^{M_b+K}_d} + n_{M_b+1} \\
& -\textstyle\sum_{l=1}^{K-1} (1 + c_{\sigma(l)} + s^1_{\sigma(l)} + v_{2M_{tot}+\sigma(l)}) \tfrac{n_{l+1} - n_l}{s^1_{\sigma(l)} - s^K_{\sigma(l)}} - n_1 && (\forall d : 1 \le d < K, \\
& && \exists l : (\sigma(l) = d)) \\
=\ & \textstyle\sum_{l=1}^{K-1} (1 + c_{\pi(l)} + s^{M_b+1}_{\pi(l)} + v_{2M_{tot}+\pi(l)}) \tfrac{n_{M_b+l+1} - n_{M_b+l}}{s^{M_b+1}_{\pi(l)} - s^{M_b+K}_{\pi(l)}} + n_{M_b+1} \\
& -\textstyle\sum_{l=1}^{K-1} (1 + c_{\sigma(l)} + s^1_{\sigma(l)} + v_{2M_{tot}+\sigma(l)}) \tfrac{n_{l+1} - n_l}{s^1_{\sigma(l)} - s^K_{\sigma(l)}} - n_1 && (\forall d : 1 \le d < K, \\
& && \exists l : (\pi(l) = d)) \\
=\ & -(1 - \tfrac{s^1_{\sigma(1)}+1+c_{\sigma(1)}+v_{2M_{tot}+\sigma(1)}}{s^1_{\sigma(1)}-s^K_{\sigma(1)}})n_1 \\
& -\textstyle\sum_{l=2}^{K-1} (\tfrac{s^1_{\sigma(l-1)}+1+c_{\sigma(l-1)}+v_{2M_{tot}+\sigma(l-1)}}{s^1_{\sigma(l-1)}-s^K_{\sigma(l-1)}} - \tfrac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}})n_l \\
& -\tfrac{s^1_{\sigma(K-1)}+1+c_{\sigma(K-1)}+v_{2M_{tot}+\sigma(K-1)}}{s^1_{\sigma(K-1)}-s^K_{\sigma(K-1)}} n_K + (1 - \tfrac{s^{M_b+1}_{\pi(1)}+1+c_{\pi(1)}+v_{2M_{tot}+\pi(1)}}{s^{M_b+1}_{\pi(1)}-s^{M_b+K}_{\pi(1)}})n_{M_b+1} \\
& +\textstyle\sum_{l=2}^{K-1} (\tfrac{s^{M_b+1}_{\pi(l-1)}+1+c_{\pi(l-1)}+v_{2M_{tot}+\pi(l-1)}}{s^{M_b+1}_{\pi(l-1)}-s^{M_b+K}_{\pi(l-1)}} - \tfrac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}})n_{M_b+l} \\
& +\tfrac{s^{M_b+1}_{\pi(K-1)}+1+c_{\pi(K-1)}+v_{2M_{tot}+\pi(K-1)}}{s^{M_b+1}_{\pi(K-1)}-s^{M_b+K}_{\pi(K-1)}} n_{M_b+K}
\end{aligned}
$$

And we see that the values of the two objective functions are equal at the two solutions $(z^i_1, \ldots, z^i_K, z^f_1, \ldots, z^f_K)$ and $\vec{v}$.

Step 4: As the last step of the lemma we show that all elements of $\vec{v}$ are non-negative. Let's start with $v_{2M_{tot}+d}$ $\forall d : 1 \leq d \leq K-1$.

Case 1 $(\forall d : 1 \leq d \leq K-1 \ (v_{2M_{tot}+d}))$**:** If d is a forward dimension $v_{2M_{tot}+d} = 0$ and trivially non-negative. If d is a backward dimension, $v_{2M_{tot}+d}$ is either equal to $-(1 + c_d + s_d^1)$ or $-(1 + c_d + s_d^{M_b+K})$. We know that both $1 + c_d + s_d^1$ and $1 + c_d + s_d^{M_b+K}$ are less or equal to 0 (Obs. 5 and 8), and therefore that $v_{2M_{tot}+d} \geq 0$.

Case 2 (v_1)**:**

From the definition of v_1 we have $v_1 = 1 - \frac{s^1_{\sigma(1)}+1+c_{\sigma(1)}+v_{2M_{tot}+\sigma(1)}}{s^1_{\sigma(1)}-s^K_{\sigma(1)}} = -\frac{s^K_{\sigma(1)}+1+c_{\sigma(1)}+v_{2M_{tot}+\sigma(1)}}{s^1_{\sigma(1)}-s^K_{\sigma(1)}}$. Since $s^1_{\sigma(1)} - s^K_{\sigma(1)} \leq 0$ (Obs. 1), we need to prove that $s^K_{\sigma(1)} + 1 + c_{\sigma(1)} + v_{2M_{tot}+\sigma(1)} \geq 0$. If $\sigma(1)$ is a forward dimension this is true since $v_{2M_{tot}+\sigma(1)} = 0$ and $s^K_{\sigma(1)} + 1 + c_{\sigma(1)} \geq 0$ (Obs. 6). If $\sigma(1)$ is a backward dimension however we have two cases. If $(v_{2M_{tot}+\sigma(1)} = -(1 + c_{\sigma(1)} + s^1_{\sigma(1)}))$, $s^K_{\sigma(1)} + 1 + c_{\sigma(1)} + v_{2M_{tot}+\sigma(1)} = s^K_{\sigma(1)} - s^1_{\sigma(1)} \geq 0$ (Obs. 1). If $(v_{2M_{tot}+\sigma(1)} = -(1 + c_{\sigma(1)} + s^{M_b+K}_{\sigma(1)}))$, $s^K_{\sigma(1)} + 1 + c_{\sigma(1)} + v_{2M_{tot}+\sigma(1)} = s^K_{\sigma(1)} - s^{M_b+K}_{\sigma(1)} \geq 0$ (Obs. 4).

Case 3 $(\forall l : 2 \leq l \leq K-1 \ (v_l))$**:**

$\forall l : 2 \leq l \leq K-1$, $v_l = \frac{s^1_{\sigma(l-1)}+1+c_{\sigma(l-1)}+v_{2M_{tot}+\sigma(l-1)}}{s^1_{\sigma(l-1)}-s^K_{\sigma(l-1)}} - \frac{s^1_{\sigma(l)}+1+c_{\sigma(l)}+v_{2M_{tot}+\sigma(l)}}{s^1_{\sigma(l)}-s^K_{\sigma(l)}}$, which is non-negative by Observation 13.

Case 4 (v_K)**:**

From the definition of v_K we have $v_K = \frac{s^1_{\sigma(K-1)}+1+c_{\sigma(K-1)}+v_{2M_{tot}+\sigma(K-1)}}{s^1_{\sigma(K-1)}-s^K_{\sigma(K-1)}}$. Since $s^1_{\sigma(K-1)} - s^K_{\sigma(K-1)} \leq 0$ (Obs. 1), we need to show that $s^1_{\sigma(K-1)} + 1 + c_{\sigma(K-1)} + v_{2M_{tot}+\sigma(K-1)} \leq 0$. If $\sigma(K-1)$ is a forward dimension, it follows from the fact that $v_{2M_{tot}+\sigma(K-1)} = 0$ and $s^1_{\sigma(K-1)} + 1 + c_{\sigma(K-1)} \leq 0$ (Obs. 5). If $\sigma(K-1)$ is a backward dimension and $v_{2M_{tot}+\sigma(K-1)} = -(1 + c_{\sigma(K-1)} + s^1_{\sigma(K-1)})$, $s^1_{\sigma(K-1)} + 1 + c_{\sigma(K-1)} + v_{2M_{tot}+\sigma(K-1)} = 0$ and we are done. If $v_{2M_{tot}+\sigma(K-1)} = -(1 + c_{\sigma(K-1)} + s^{M_b+K}_{\sigma(K-1)})$, $s^1_{\sigma(K-1)} + 1 + c_{\sigma(K-1)} + v_{2M_{tot}+\sigma(K-1)} = s^1_{\sigma(K-1)} - s^{M_b+K}_{\sigma(K-1)}$, which is less or equal to zero since $s^{M_b+K}_{\sigma(K-1)} \geq s^1_{\sigma(K-1)}$ by the construction of $v_{2M_{tot}+\sigma(K-1)}$.

Case 5 $(\forall i : K+1 \leq i \leq M_{tot}+M_b, M_{tot}+M_b+K+1 \leq i \leq 2M_{tot}, i = 2M_{tot}+K \ (v_i))$**:**

Trivially true since $\forall i : K+1 \leq i \leq M_{tot}+M_b, M_{tot}+M_b+K+1 \leq i \leq 2M_{tot}, i = 2M_{tot}+K \ (v_i = 0)$.

Case 6 $(v_{M_{tot}+M_b+1})$**:**

From the definition of $v_{M_{tot}+M_b+1}$ we have $v_{M_{tot}+M_b+1} = 1 - \frac{s^{M_b+1}_{\pi(1)}+1+c_{\pi(1)}+v_{2M_{tot}+\pi(1)}}{s^{M_b+1}_{\pi(1)}-s^{M_b+K}_{\pi(1)}} = -\frac{s^{M_b+K}_{\pi(1)}+1+c_{\pi(1)}+v_{2M_{tot}+\pi(1)}}{s^{M_b+1}_{\pi(1)}-s^{M_b+K}_{\pi(1)}}$. Since $s^{M_b+1}_{\pi(1)} - s^{M_b+K}_{\pi(1)} \geq 0$ (Obs. 2), we need to prove that $s^{M_b+K}_{\pi(1)} + 1 + c_{\pi(1)} + v_{2M_{tot}+\pi(1)} \leq 0$. If $\pi(1)$ is a forward dimension this is true since $v_{2M_{tot}+\pi(1)} = 0$ and $s^{M_b+K}_{\pi(1)} + 1 + c_{\pi(1)} \leq 0$ (Obs. 8). If $\pi(1)$ is a backward dimension however we have two cases. If $(v_{2M_{tot}+\pi(1)} = -(1 + c_{\pi(1)} + s^1_{\pi(1)}))$, $s^{M_b+K}_{\pi(1)} + 1 + c_{\pi(1)} + v_{2M_{tot}+\pi(1)} = s^{M_b+K}_{\pi(1)} - s^1_{\pi(1)}$, which is less or equal to zero since $s^1_{\pi(1)} \geq s^{M_b+K}_{\sigma(1)}$ by the construction of $v_{2M_{tot}+\sigma(K-1)}$. If $(v_{2M_{tot}+\pi(1)} = -(1 + c_{\pi(1)} + s^{M_b+K}_{\pi(1)}))$, $s^{M_b+K}_{\pi(1)} + 1 + c_{\pi(1)} + v_{2M_{tot}+\pi(1)} = 0$.

Case 7 $(\forall l : 2 \leq l \leq K-1 \ (v_{M_{tot}+M_b+l}))$**:**

$\forall l : 2 \leq l \leq K-1$, $v_{M_{tot}+M_b+l} = \frac{s^{M_b+1}_{\pi(l-1)}+1+c_{\pi(l-1)}+v_{2M_{tot}+\pi(l-1)}}{s^{M_b+1}_{\pi(l-1)}-s^{M_b+K}_{\pi(l-1)}} - \frac{s^{M_b+1}_{\pi(l)}+1+c_{\pi(l)}+v_{2M_{tot}+\pi(l)}}{s^{M_b+1}_{\pi(l)}-s^{M_b+K}_{\pi(l)}}$, which is non-negative by Observation 14.

Case 8 $(v_{M_{tot}+M_b+K})$**:**

From the definition of $v_{M_{tot}+M_b+K}$ we have $v_{M_{tot}+M_b+K} = \frac{s^{M_b+1}_{\pi(K-1)}+1+c_{\pi(K-1)}+v_{2M_{tot}+\pi(K-1)}}{s^{M_b+1}_{\pi(K-1)}-s^{M_b+K}_{\pi(K-1)}}$. Since $s^{M_b+1}_{\pi(K-1)} - s^{M_b+K}_{\pi(K-1)} \geq 0$ (Obs. 2), we need to show that $s^{M_b+1}_{\pi(K-1)} + 1 + c_{\pi(K-1)} + v_{2M_{tot}+\pi(K-1)} \geq 0$. If $\pi(K-1)$ is a forward dimension, it follows from the fact that $v_{2M_{tot}+\pi(K-1)} = 0$ and $s^{M_b+1}_{\pi(K-1)} + 1 + c_{\pi(K-1)} \geq 0$ (Obs. 7). If $\pi(K-1)$ is a backward dimension and $v_{2M_{tot}+\pi(K-1)} = -(1 + c_{\pi(K-1)} + s^1_{\pi(K-1)})$, $s^{M_b+1}_{\pi(K-1)} + 1 + c_{\pi(K-1)} + v_{2M_{tot}+\pi(K-1)} = s^{M_b+1}_{\pi(K-1)} - s^1_{\pi(K-1)} \geq 0$ (Obs. 3). If $v_{2M_{tot}+\pi(K-1)} = -(1 + c_{\pi(K-1)} + s^{M_b+K}_{\pi(K-1)})$, $s^{M_b+1}_{\pi(K-1)} + 1 + c_{\pi(K-1)} + v_{2M_{tot}+\pi(K-1)} = s^{M_b+1}_{\pi(K-1)} - s^{M_b+K}_{\pi(K-1)} \geq 0$ (Obs. 4).

End of Proof of Lemma 2 □

Lemma 2 does not cover iteration spaces where the location of the bottom and top corners is at either the minimum or maximum coordinate in any dimension. The intuition behind the proof in these cases is that extra stacks can be added to an iteration space without altering the longest path of dependent tiles.

Let I be an iteration space such that $\exists d$ such that either $q_d^b = x_d^{min}$, $q_d^b = x_d^{max}$, $q_d^t = x_d^{min}$, or $q_d^t = x_d^{max}$. Let I' be an iteration space that we construct in the following way. First, construct I' so it is an identical copy of I. Second, pull the top of I' a distance of ϵ apart from the bottom in the vertical dimension. This ensures that there are vertical sides in all dimensions of I'. (We will later refer to this iteration space as I'".) Third, $\forall d : (q_d^b = x_d^{max}$ or $q_d^t = x_d^{max})$ add some stacks to the coordinates greater than x_d^{max}, and $\forall d : (q_d^b = x_d^{min}$ or $q_d^t = x_d^{min})$ add some stacks to the coordinates smaller than x_d^{min}. The shape of the added stacks should be such that the bottom and top corners undefined in I are equal to x_d^{min} or x_d^{max} in I'. This completes the construction of I'.

Observation 19 *$\nexists d$ in I' such that q_d^b or q_d^t lie within the stacks that were added to I' in dimension d.*

Now, use Lemma 2 to calculate the location of the synchronization points of I' and thus the longest path of dependent tiles through I'. Due to Observation 19 we know that the longest path of dependent tiles through I' can not go though any of the stacks added to I'. We therefore know that we can remove these stacks without affecting the longest path of dependent tiles. If there were a longer path after removing the extra stacks there would have been a longer path before the removal as well. If the longest path of dependent tiles were shortened after removing the extra stacks, the longest path before the removal would also have been shorter. We therefore know that the longest path of dependent tiles in I'' is the same as in I', and the location of the synchronization points must therefore also be identical. Letting ϵ approach zero and thus letting I'' approach I, we see that this claim is true for I as well.

End of Proof of Theorem 1 □

7 Tile shape selection

In this section we discuss the implications Theorem 1 has on the preferred choice of tile shape. We first study how the longest path of dependent tiles changes with the tile shape and then determine the tile shape that minimizes the length of this path. In past practice, tiles were chosen either to be rectangular, or of the same shape in the case of parallelepiped iteration spaces, as the iteration space. Our results do not necessarily support either of these choices.

The height of the tallest stack in an iteration space is a lower bound on $E(\vec{z}^{\,i}, \vec{z}^{f})$, since all vertical paths are paths of dependent tiles. We therefore know that for any two points $\vec{e}$ and $\vec{f}$ that lie on the bottom and top of a single stack, $E(\vec{e}, \vec{f})$ can never be less than $f_K - e_K$. Due to the rectilinear property of the iteration space we know that $E(\vec{z}^{\,i}, \vec{z}^{f})$ decreases as the number of dimensions d for which $z_d^i = z_d^f$ increases.

The tile shape should therefore be chosen such that the coordinates for the initial and final synchronization points are *the same* in as many dimensions as possible. If that is not possible then they should lie *as close as possible*. Increasing the tile slope in a given dimension decreases the distance between the synchronization points in that dimension. At the point when q_d^b first becomes greater than q_d^t, there is no longer a benefit in increasing the slope further. The longest path of dependent tiles is at that point already vertical and can not become shorter. This is illustrated in Figure 11, and we see that the longest path of dependent tiles between the synchronization points in (f) is indeed shorter than when we use rectangular tiles in (e).

We state the following corollary to Theorem 1.

Corollary 1 *The execution time of a tiling is minimized when, for all dimensions $d : 1 \leq d \leq K-1$, the slope of the tiling in dimension d is chosen as small as possible such that $q_d^b \geq q_d^t$. If this is not possible due to legality, the slope should be chosen as large as possible without violating the legality of the tiling.*

8 Execution time of a rectilinear iteration space

In this section we first derive a closed form formula for the execution time of a rectilinear iteration space. We then give the execution time for some common iteration space shapes in Table 1.

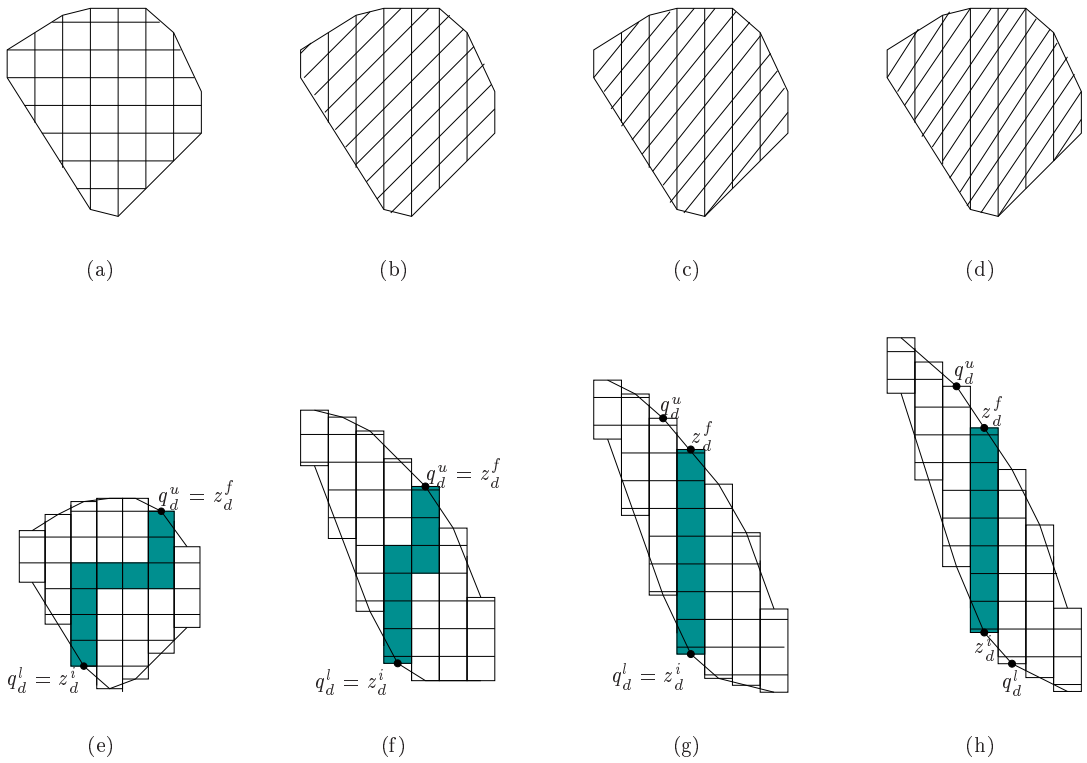


Figure 11: *The same projection through the same iteration space tiled with four different tile slopes, increasing from (a) to (d). In all tilings the tile size is the same. Figures (e)-(h) show the iteration space after transformations. The example illustrates how the location of both the bottom and top corner as well as the two synchronization points changes with the tile slope. The shaded tiles constitute the longest path of dependent tiles. Increasing the tile slope pushes the bottom and top corners of the iteration space to the right and left respectively. Also the synchronization points move but only to the point where $z_d^i = z_d^f$. At that point they specify the coordinate in dimension d for the tallest stack within the iteration space. Since the execution time of a stack is a lower bound on the execution time of the iteration space, we would like for as many dimensions as possible to achieve this lower bound, i.e. be backward dimensions.*

As it turns out, the execution time only depends on the slope of four of the facets on the top and bottom surfaces, independently of the dimensionality of the iteration space. All other facets can change slope without affecting the execution time, as long as the iteration space remains convex and rectilinear. We call these four facets the *defining facets* of an iteration space. Before we define these facets we need to introduce some terminology.

Definition 11 *Let x_d^{min} and x_d^{max} be the lowest and highest coordinates that exist within the iteration space in dimension d.*
We define $\mathcal{C}^i$ to be the set of dimensions d such that ($z_d^i = x_d^{min}$ or $z_d^i = x_d^{max}$) by Theorem 1. We define $\mathcal{C}^f$ to be the set of dimensions d such that ($z_d^f = x_d^{min}$ or $z_d^f = x_d^{max}$) by Theorem 1.
We also define $G^i = K - |\mathcal{C}^i|$, and $G^f = K - |\mathcal{C}^f|$.

Recall that the initial and final synchronization points in a rectilinear iteration space are defined by the intersection of 2^{G^i-1} and 2^{G^f-1} facets of bounding the bottom and top surfaces, respectively. For a dimension $d \in \mathcal{C}^i$ (or $\in \mathcal{C}^f$) there are only one slope used in those facets, and for a dimension $d \notin \mathcal{C}^i$ (or $\notin \mathcal{C}^f$) there are two slopes. Using this fact, we define the four defining facets as follows (see Figure 12).

Definition 12 *Consider all the 2^{G^i-1} facets bounding the bottom surface and intersecting in the initial synchronization point, $\vec{z}^{\,i}$. For all dimensions $d \notin \mathcal{C}^i$, let s_d^{low} and s_d^{high} be the two slopes in dimension d used in those facets such that $s_d^{low} \leq s_d^{high}$. For all dimensions $d \in \mathcal{C}^i$ we call the only slope used in those 2^{G^i-1} facets s_d^{only}.*

The first defining facet, Facet 1*, is the facet with equation*

$$-\sum_{\forall d \notin \mathcal{C}^i} s_d^{low} x_d - \sum_{\forall d \in \mathcal{C}^i} s_d^{only} x_d + x_K = n_1.$$

$\forall d \notin \mathcal{C}^i$*, the slope in dimension* d *of Facet* 1*,* s_d^1*, is thus equal to* s_d^{low}*.*
The second defining facet , Facet G^i*, is the facet with equation*

$$-\sum_{\forall d \notin \mathcal{C}^i} s_d^{high} x_d - \sum_{\forall d \in \mathcal{C}^i} s_d^{only} x_d + x_K = n_{G^i}.$$

$\forall d \notin \mathcal{C}^i$*, the slope in dimension* d *of Facet* G^i*,* $s_d^{G^i}$*, is thus equal to* s_d^{high}*.*
$\forall d \in \mathcal{C}^i$*,* $s_d^{G^i} = s_d^1 = s_d^{only}$*.*

Definition 13 *Consider all the* 2^{G^f-1} *facets bounding the top surface and intersecting in the final synchronization point,* $\vec{z}^f$*. For all dimensions* $d \notin \mathcal{C}^f$*, let* s_d^{low} *and* s_d^{high} *be the two slopes in dimension* d *used in those facets such that* $s_d^{low} \leq s_d^{high}$*. For all dimensions* $d \in \mathcal{C}^f$ *we call the only slope used in those* 2^{G^f-1} *facets* s_d^{only}*.*
The third defining facet, Facet $M_b + 1$*, is the facet with equation*

$$\sum_{\forall d \notin \mathcal{C}^f} s_d^{high} x_d + \sum_{\forall d \in \mathcal{C}^f} s_d^{only} x_d - x_K = n_{M_b+1}.$$

$\forall d \notin \mathcal{C}^f$*, the slope in dimension* d *of Facet* $M_b + 1$*,* $s_d^{M_b+1}$*, is thus equal to* s_d^{high}*.*
The fourth defining facet, Facet $M_b + G^f$*, is the facet with equation*

$$\sum_{\forall d \notin \mathcal{C}^f} s_d^{low} x_d + \sum_{\forall d \in \mathcal{C}^f} s_d^{only} x_d - x_K = n_{M_b+G^f}.$$

$\forall d \notin \mathcal{C}^f$*, the slope in dimension* d *of Facet* $M_b + G^f$*,* $s_d^{M_b+G^f}$*, is thus equal to* s_d^{low}*.*
$\forall d \in \mathcal{C}^f$*,* $s_d^{M_b+G^f} = s_d^{M_b+1} = s_d^{only}$*.*

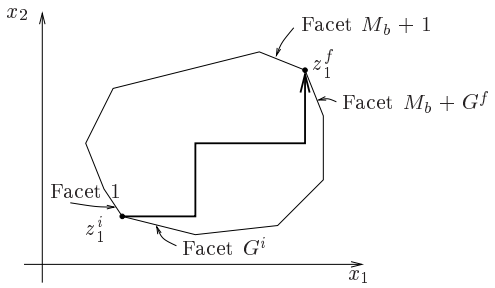


Figure 12: *In a* 2*-dimensional iteration space the defining facets are the facets intersecting at the synchronization points. We number the defining facets* 1*,* G^i*,* $M_b + 1$ *and* $M_b + G^f$*, as shown.*

Knowing that there are only $2(G^i - 1) + |\mathcal{C}^i|$ (and $2(G^f - 1) + |\mathcal{C}^f|$) different slopes in the 2^{G^i-1} (and 2^{G^f-1}) facets intersecting at the two synchronization points, helps us solve the system of linear equations from Section 4. We present the resulting coordinates in Observation 20.

We start by selecting the following G^i equations from the equations of the facets bounding the iterations space. Due to the way they are chosen, they all intersect in the initial synchronization point. We let $\forall d \in \mathcal{C}^i$ s_d^1 denote the only slope used in both Facet 1 and G^i.

$$\begin{bmatrix} s_1^1 & s_2^1 & s_3^1 & \dots & s_{G^i-2}^1 & s_{G^i-1}^1 & -1 \\ s_1^{G^i} & s_2^1 & s_3^1 & \dots & s_{G^i-2}^1 & s_{G^i-1}^1 & -1 \\ s_1^{G^i} & s_2^{G^i} & s_3^1 & \dots & s_{G^i-2}^1 & s_{G^i-1}^1 & -1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_1^{G^i} & s_2^{G^i} & s_3^{G^i} & \dots & s_{G^i-2}^{G^i} & s_{G^i-1}^1 & -1 \\ s_1^{G^i} & s_2^{G^i} & s_3^{G^i} & \dots & s_{G^i-2}^{G^i} & s_{G^i-1}^{G^i} & -1 \end{bmatrix} \begin{bmatrix} z_1^i \\ z_2^i \\ z_3^i \\ \dots \\ z_{G^i-1}^i \\ z_{G^i}^i \end{bmatrix} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ \dots \\ n_{G^i-1} \\ n_{G^i} \end{bmatrix}$$

To determine z_d^i for all $d : 1 \leq d \leq G^i - 1$ we then subtract equation d and $d+1$ from each other. For all dimension $d \in \mathcal{C}^i$, we know by Theorem 1 that z_d^i is equal to either x_d^{min} or x_d^{max}. Similarly we get the coordinates for $\vec{z}^f$ by solving the following system of linear equations.

$$\begin{bmatrix} s_1^{M_b+1} & s_2^{M_b+1} & s_3^{M_b+1} & \dots & s_{G^f-2}^{M_b+1} & s_{G^f-1}^{M_b+1} & -1 \\ s_1^{M_b+G^f} & s_2^{M_b+1} & s_3^{M_b+1} & \dots & s_{G^f-2}^{M_b+1} & s_{G^f-1}^{M_b+1} & -1 \\ s_1^{M_b+G^f} & s_2^{M_b+G^f} & s_3^{M_b+1} & \dots & s_{G^f-2}^{M_b+1} & s_{G^f-1}^{M_b+1} & -1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_1^{M_b+G^f} & s_2^{M_b+G^f} & s_3^{M_b+G^f} & \dots & s_{G^f-2}^{M_b+G^f} & s_{G^f-1}^{M_b+1} & -1 \\ s_1^{M_b+G^f} & s_2^{M_b+G^f} & s_3^{M_b+G^f} & \dots & s_{G^f-2}^{M_b+G^f} & s_{G^f-1}^{M_b+G^f} & -1 \end{bmatrix} \begin{bmatrix} z_1^f \\ z_2^f \\ z_3^f \\ \dots \\ z_{G^f-1}^f \\ z_{G^f}^f \end{bmatrix} = \begin{bmatrix} n_{M_b+1} \\ n_{M_b+2} \\ n_{M_b+3} \\ \dots \\ n_{M_b+G^f-1} \\ n_{M_b+G^f} \end{bmatrix}$$

Observation 20 *In the last dimension, K, z_K^i and z_K^f are given by*

$$z_K^i = \sum_{d=1}^{K-1} s_d^1 z_d^i + n_1$$
$$z_K^f = \sum_{d=1}^{K-1} s_d^{M_b+1} z_d^f + n_{M_b+1}$$

For all dimensions $d \notin \mathcal{C}^i$, z_d^i is given by

$$z_d^i = \begin{cases} \frac{n_{d+1}-n_d}{s_d^1 - s_d^{G^i}} & (\forall d : (s_d^1 \neq s_d^{G^i})) \\ z_d^f & (\forall d : (s_d^1 = s_d^{G^i})) \end{cases}$$

For all dimensions $d \notin \mathcal{C}^f$, z_d^f is given by

$$z_d^f = \begin{cases} \frac{n_{M_b+d+1}-n_{M_b+d}}{s_d^{M_b+1} - s_d^{M_b+G^f}} & (\forall d : (s_d^{M_b+1} \neq s_d^{M_b+G^f})) \\ z_d^i & (\forall d : (s_d^{M_b+1} = s_d^{M_b+G^f})) \end{cases}$$

where $\forall i : 1 \leq i \leq K-1$ the facets are ordered such that there is only one dimension that changes between facet i and facet $i+1$.[15,16]

Using these coordinates we can now calculate the execution time of the iteration space as defined in Definition 1. Given the execution time of a tile a compiler can use this formula to predict the execution time of the loop for a particular tile shape. In Theorem 2, the execution time is given as a function of the parameter T. As illustrated in Figure 13, T is the number of (possibly empty) tiles between Facet 1 and Facet $M_b + 1$ at coordinate $\vec{z}^{\,i}$. The quantity $T + \sum_{\forall d \in \mathcal{F}} s_d^{M_b+1}(z_d^f - z_d^i)$ therefore represents the vertical distance (in units of tiles) between the top surface at coordinate $\vec{z}^f$ and the bottom surface at $\vec{z}^{\,i}$. In each dimension $d \in \mathcal{F}$ the quantity $\sum_{\forall d \in \mathcal{F}}(1 + c_d)(z_d^f - z_d^i)$ is the number of tiles (stacks) plus communication cost between the processors along the horizontal path in that dimension d from $\vec{z}^{\,i}$ to $\vec{z}^f$. Table 1 shows the execution time for some common iteration space shapes.

Theorem 2 *The execution time of a rectilinear iteration space, E_i, is given by the following expression.*

$$E_i = E_t \sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^{M_b+1})(z_d^f - z_d^i) + E_t T$$

where $T = \sum_{d=1}^{K-1}(s_d^{M_b+1} - s_d^1) z_d^i + n_{M_b+1} - n_1$ and $\forall d : 1 \leq d \leq K-1$ z_d^i and z_d^f are given by Observation 20.

[15] We know that such an ordering exists since the iteration space is rectilinear.

[16] We know by Theorem 1 that $\nexists d : (d \notin \mathcal{C}^i$ and $d \notin \mathcal{C}^f)$ where $(s_d^1 = s_d^{G^i}$ and $s_d^{M_b+1} = s_d^{M_b+G^f})$.

Proof of Theorem 2:

$$
\begin{aligned}
E_i &= E(\vec{z}^{\,i}, \vec{z}^{f}) && \text{(Def. 2, 1)}\\
&= E_t((z_K^f - z_K^i) + \textstyle\sum_{d=1}^{K-1} (z_d^f - z_d^i)(1 + c_d)) && \text{(Def. 1)}\\
&= E_t(\sum_{d=1}^{K-1} (1 + c_d + s_d^{M_b+1}) z_d^f \\
&\quad - \sum_{d=1}^{K-1} (1 + c_d + s_d^1) z_d^i + n_{M_b+1} - n_1) && \text{(Obs. 20)}\\
&= E_t(\sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^{M_b+1}) z_d^f - \sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^1) z_d^i \\
&\quad + \sum_{\forall d \in \mathcal{B}} (s_d^{M_b+1} - s_d^1) z_d^i + n_{M_b+1} - n_1) && (\forall d \in \mathcal{B}\ (z_d^i = z_d^f))\\
&= E_t(n_{M_b+1} - n_1 + \sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^{M_b+1})(z_d^f - z_d^i) \\
&\quad + \sum_{\forall d \in \mathcal{F}} (s_d^{M_b+1} - s_d^1) z_d^i + \sum_{\forall d \in \mathcal{B}} (s_d^{M_b+1} - s_d^1) z_d^i) \\
&= E_t(\sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^{M_b+1})(z_d^f - z_d^i) \\
&\quad + \sum_{d=1}^{K-1} (s_d^{M_b+1} - s_d^1) z_d^i + n_{M_b+1} - n_1) \\
&= E_t \sum_{\forall d \in \mathcal{F}} (1 + c_d + s_d^{M_b+1})(z_d^f - z_d^i) + E_t T
\end{aligned}
$$

where $T = \sum_{d=1}^{K-1} (s_d^{M_b+1} - s_d^1) z_d^i + n_{M_b+1} - n_1$.

End of Proof of Theorem 2 □

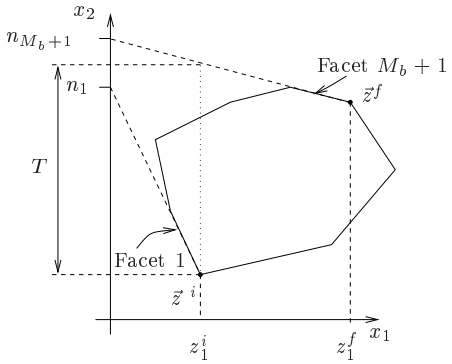


Figure 13: *Example 2-dimensional iteration space. The quantity* $\sum_{d=1}^{K-1} (s_d^{M_b+1} - s_d^1) z_d^i + n_{M_b+1} - n_1$ *that we call* T *is the distance between Facet* 1 *and Facet* $M_b + 1$ *at coordinate* $\vec{z}^{\,i}$*. Since tiles are unit cubes,* T *is equal to the number of tiles in the stack located at* $\vec{z}^{\,i}$*. Note that these tiles might be empty. Since they also might be partial,* T *is not necessarily an integer.*

References

[ARY98] Rumen Andonov, Sanjay Rajopadhye, and Nicola Yanev. Optimal orthogonal tiling. In *Europar'98*, pages 480–490, September 1998.

[Ban90] Utpal Banerjee. Unimodular transformations of double loops. In *Workshop on Programming Languages and Compilers for Parallel Computing*, Irvine, CA, August 1990.

	dim	*I.S. Shape*	$\mathcal{F}$ *or* $\mathcal{B}$	*Execution time formula*
a	2	*Parallelogram w/ vertical sides*	$\mathcal{B}$	$E_t T$
			$\mathcal{F}$	$E_t(1 + c_1 + s_1^{M_b+1})(P_1 - 1) + E_t T$
b		*Parallelogram*	$\mathcal{B}$	$E_t T$
			$\mathcal{F}$	$E_t(1 + c_1 + s_1^{M_b+1})(z_1^f - z_1^i) + E_t T$
c		*General*	$\mathcal{B}$	$E_t T$
			$\mathcal{F}$	$E_t(1 + c_1 + s_1^{M_b+1})(z_1^f - z_1^i) + E_t T$
d	3	*Polytope w/ vertical sides & only one top&bottom facet*	$\mathcal{B}, \mathcal{B}$	$E_t T$
			$\mathcal{F}, \mathcal{B}$	$E_t((1 + c_1 + s_1^{M_b+1})(P_1 - 1) + T)$
			$\mathcal{F}, \mathcal{F}$	$E_t(\sum_{d=1}^{2}(1 + c_d + s_d^{M_b+1})(P_d - 1) + T)$
e	K	*Parallelepiped*	$j \in \mathcal{B}$	$E_t(\sum_{\forall d \in \mathcal{F}}(1 + c_d + s_d^{M_b+1})(P_d - 1) + T)$
			$j \in \mathcal{F}$	$E_t(\sum_{\forall d \in \mathcal{F}, d \neq j}(1 + c_d + s_d^{M_b+1})(P_d - 1) + (1 + c_j + s_j^{M_b+1})(z_j^f - z_j^i) + T)$

Table 1: *The execution time formula from Theorem 2 for some common iteration space shapes.* **Case a:** *In a 2-dimensional parallelogram with vertical sides, the longest path of dependent tiles goes straight up each stack if the first dimension is a backward dimension (Theorem 1). The execution time is thus T times the execution time of a tile. If the first dimension is a forward dimension the longest path of dependent tiles goes horizontally through all P stacks as well as up to the top. Along the horizontal component of the path, $(P_1 - 1)$ tiles are executed and $(P_1 - 1)$ surfaces communicated. Along the vertical component we execute the number of tiles between z_K^i and z_K^f, i.e. $T + (P_1 - 1)s^{M_b+1}$. Due to the fact that only the defining facets affect the execution time, there is no difference in the execution time formula between the* **Case b** *and* **c**. **Case e:** *The only K-dimensional parallelepiped that is rectilinear, is the parallelepiped skewed in at most one dimension other than the vertical. In this table we call this dimension j. The table shows the formula for when $j \in \mathcal{F}$ and $j \in \mathcal{B}$. P_d is the number of processors along dimension d in the processor grid, i.e. the total number of processors is equal to $\prod_{d=1}^{K-1} P_d$. T is equal to $\sum_{d=1}^{K-1}(s_d^{M_b+1} - s_d^1)z_d^i + n_{M_b+1} - n_1$, and is explained in Figure 13.*

[CFH95a] Larry Carter, Jeanne Ferrante, and S. Flynn Hummel. Efficient parallelism via hierarchical tiling. In *SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[CFH95b] Larry Carter, Jeanne Ferrante, and S. Flynn Hummel. Hierarchical tiling for improved superscalar perfomance. In *International Parallel Processing Symposium*, April 1995.

[Chv83] Vašek Chvátal. *Linear Programming.* W. H. Freeman and Company, 1983.

[CK92] Steve Carr and Ken Kennedy. Compiler blockability of numerical algorithms. *The Journal of Supercomputing*, pages 114–124, November 1992.

[CKP+96] David Culler, Richard Karp, David Patterson, Abhijit Sahay, Eunice Santos, Klaus Erik Schauser, Rarnesh Subramonian, and Thorsten von Eicken. LogP: A practical model of parallel computation. *Communications of the ACM*, 39(11):78–85, November 1996.

[CM95] Stephanie Coleman and Kathryn S. McKinley. Tile size selection using cache organization and data layout. In *Programming Language Design and Implementation*, June 1995.

[DDRR97] Frederic Desprez, Jack Dongarra, Fabrice Rastello, and Yves Robert. Determining the idle time of a tiling: New results. In *International Conference on Parallel Architectures and Compilation Techniques (PACT'97)*, November 1997.

[DKR91] Alain Darte, Leonid Khachiyan, and Yves Robert. Linear scheduling is nearly optimal. *Parallel Processing Letters*, 1(2):73–81, 1991.

[HCF97] Karin Högstedt, Larry Carter, and Jeanne Ferrante. Determining the idle time of a tiling. In *Symposium on Principles of Programming Languages*, January 1997.

[IT88] François Irigoin and Rémi Triolet. Supernode partitioning. In *Symposium on Principles of Programming Languages*, pages 319–328, January 1988.

[Kar91] Howard Karloff. *Linear Programming*. Progress in Theoretical Computer Science. Birkhäuser, 1991.

[MCFH97] Nicholas Mitchell, Larry Carter, Jeanne Ferrante, and Karin Högstedt. Quantifying the multi-level nature of tiling interactions. In *Workshop on Languages and Compilers for Parallel Computing*, 1997. Published in Springer-Verlag Lecture Notes in Computer Science Volume 1366.

[MHCF98] N. Mitchell, K. Högstedt, L. Carter, and J. Ferrante. Quantifying the multi-level nature of tiling interactions. *International Journal of Parallel Programming*, 26(6):641–670, 1998.

[RAP87] D. A. Reed, L. M. Adams, and M. L. Patrick. Stencils and problem partitionings: Their influence on the performance of multiple processor systems. *IEEE Transactions on Computers*, C-36(7):845–858, July 1987.

[RS91] J. Ramanujam and P. Sadayappan. Tiling multidimensional iteration spaces for nonshared memory machines. In *Supercomputing*, November 1991.

[Sar97] Vivek Sarkar. Automatic selection of high-order transformations in the IBM XL FORTRAN compilers. *IBM Journal of Research and Development*, 41(3):233–264, 1997.

[WL91a] Michael E. Wolf and Monica S. Lam. A data locality optimizing algorithm. In *Symposium on Programming Language Design and Implementation*, 1991.

[WL91b] Michael E. Wolf and Monica S. Lam. A loop transformation theory and an algorithm to maximize parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, 1991.

[Wol87] Michael J. Wolfe. Iteration space tiling for memory hierarchies. In *Parallel Processing for Scientific Computing*, pages 357–361, 1987.

[Wol89] Michael J. Wolfe. More iteration space tiling. In *Supercomputing*, pages 655–664, 1989.

[Wol96] Michael J. Wolfe. *High Performance Compilers for Parallel Computing*. Addison-Wesley, 1996.