Title
Determining the idle time of a tiling

Authors
Hogstedt, Karin
Carter, Larry
Ferrante, Jean

Publication Date
1999-07-06

Peer reviewed

Determining the Idle Time of a Tiling

Karin Högstedt, Larry Carter[*], Jeanne Ferrante[†]
Department of Computer Science and Engineering, UCSD
9500 Gilman Drive, La Jolla, CA 92093-0114

Abstract

This paper investigates the *idle time* associated with a parallel computation, that is, the time that processors are idle because they are either waiting for data from other processors or waiting to synchronize with other processors. We study doubly-nested loops corresponding to parallelogram- or trapezoidal-shaped iteration spaces that have been parallelized by the well-known tiling transformation. We introduce the notion of *rise* r, which relates the shape of the iteration space to that of the tiles. For parallelogram- shaped iteration spaces, we show that when $r \leq -2$, the idle time is linear in P, the number of processors, but when $r \geq -1$, it is quadratic in P. In the context of hierarchical tiling, where multiple levels of tiling are used, a good choice of rise can lead to less idle time and better performance. While idle time is not the only cost that should be considered in evaluating a tiling strategy, current architectural trends (of deeper memory hierarchies and multiple levels of parallelism) suggest it has increasing importance.

1 Introduction

Tiling is a well-known transformation that has been used for parallelism and to obtain better locality in the memory hierarchy.

Tiling starts with the iteration space determined by a perfectly nested loop in the program, and partitions the iteration space into tiles of chosen size and shape. The iterations represented by each tile are executed in proximity: on a given processor, all the iterations of one tile are executed before any iterations of the next tile.

In the parallel context, different cost functions have been used to evaluate different tiling choices. For example, [RS91] minimizes communication costs; [WL91] maximizes the granularity of parallelism. None of this previous work considers minimizing *idle time*, that is, the time a processor is idle because it is waiting for data from another processor or to synchronize with other processors.

Current architectural trends toward multiple levels of parallelism and deeper memory hierarchies warrant a closer look at idle time. Handling these multiple levels entails recursively partitioning tiles into subtiles multiple times; at each subdivision, the number of subtiles per tile is roughly the ratio of the larger to the smaller memory capacity. Because of the trend towards *more* levels, this ratio is *decreasing*. Thus the number of subtiles per tile is decreasing. This in turn means the idle time can be a larger fraction of the total and should not be ignored.

In hierarchical tiling [CFH95a, CFH95b], tiling is done for each level of memory hierarchy and parallelism. Typically, tiling for a given level corresponds to introducing one to three levels of loop nesting. This partitions the iteration space of the larger level into smaller tiles. Thus, the tiles at a given level become the iteration space at the next level, which is itself tiled. The tiles are typically parallelograms, triangles, or trapezoids (or their higher-dimensional equivalents), and their size and shape is chosen by the tiling transformation. This gives rise to a need to examine the relationship of the size and shape of the iteration space and tiles to the overall execution costs, and particularly to the idle time.

In this paper, we define the notion of *rise*, which relates the shape of the iteration space to that of the tiles. We show that the idle time depends on the rise, and that large values of rise lead to larger idle time.

In hierarchical tiling, choosing the tile shape and size for one level affects the possible values of the rise at the next finer level of granularity. The interaction of the choices at different levels must guide the best overall choice. To evaluate the global solution it is important to have a reasonably accurate and efficiently computed estimation of idle time. In the paper, we

- Give simple formulas that bound the idle time of two-dimensional tilings of parallelogram-shaped iteration spaces. When the rise r is greater than or equal to -1, the idle time is roughly proportional to the square of the number processors P. The case of $r \leq -2$ is more favorable, with the idle time being smaller and proportional to P. These formulas hold for both block and block cyclic distribution of

[*]Also at San Diego Supercomputing Center.

[†]email: {hogstedt, carter, ferrante}@cs.ucsd.edu. This work was partly supported by NSF grant CCR-9504150.

iterations to processors. (We don't have a simple formula for the case of $-2 < r < -1$).

- Show our formula is exact when $r \geq -1$ for "sufficiently high" iteration spaces.
- Derive bounds for the more general trapezoidal- or triangular-shaped iteration spaces.
- Illustrate that a good choice of r can lead to better performance.

To illustrate the idea behind the rise, consider the three tilings of the same iteration space in Figure 5. In this example, all tiles have identical height, width, and communication times, but different values of rise. The result is vastly different parallel execution times due to different idle times. The best execution times corresponds to the tiling with negative rise, and the worst to positive rise.

Because a given tiling choice determines the rise, we can use our formula to advantage for multi-level tiling. In particular, at a level with small number of processors, P (most notably $P = 1$), we can with only a small penalty choose a tiling which slopes downward with respect to the dependences, even though this results in a (locally unfavorable) positive rise. Since this downwardly sloping tile is itself the iteration space for the next level of tiling, it will allow a negative rise for its subtiles, resulting in reduced idle time for this level. This is illustrated in Figure 7.

2 Related Work

Iteration space tiling is an important and well-known technique [JM86, GJG88, IT88, W87, RAP87, W89, HA90, L90, LRW91, RS91, WL91, WL91b, B92, KM92] used previously to achieve both parallelism and locality for a given loop nest. Tiling partitions the iteration space of a loop nest into uniform tiles of a given size and shape (except at the boundaries of the iteration space) which tessellate the iteration space. Tiling can often be achieved by first strip mining and then interchanging loops [W89].

Unimodular transformations [IT88, B92, WL91b] are often used before tiling, for instance, to change the original loop order. In two dimensions, unimodular transformations can skew the iteration space so that only rectangular (as opposed to parallelogram-shaped) tiles need be considered. Our approach considers parallelepiped shapes (the generalization of 2-dimensional parallelograms).

The cost measures used by previous work on tiling for parallelism all differ from idle time. For instance, [RS91] attempts to minimize communication time, and [WL91] considers maximizing the granularity of parallel loops. [KP93] shows how to estimate performance factors such as parallelism granularity and number of cache misses for transformations such as tiling. It does not explicitly consider the number of wasted cycles.

3 Nomenclature and Definitions

3.1 Tilings, Tiles and Iteration Spaces

In this paper, we give some non-standard but roughly equivalent definitions to the optimizing compiler literature [IT88]. In our formulation, tiles are subsets of $\Re^K$, K-dimensional real space. This allows us to use the tile's volume as a measure of execution time, and thereby to avoid unimportant complications having to do with there being a variable number of lattice points per tile, depending on the alignment of the tile with respect to the lattice points. Furthermore, it is easier to make geometric arguments about the volume of partial tiles than to compute the number of iterations involved.

Definition 1 *Let $\vec{n}$ be a vector in $\Re^K$. The* **j-th half-open hyperswath with normal $\vec{n}$,** *denoted* $\boldsymbol{H_j(\vec{n})}$, *is the set of points*
$\{\vec{x} \mid j - 1 \leq \frac{\vec{x} \cdot \vec{n}}{\vec{n} \cdot \vec{n}} < j\}$.

Thus, $H_j(\vec{n})$ is bounded by two hyperplanes orthogonal to $\vec{n}$.

Observe that the family $\mathcal{F}(\vec{n})$ of half-open hyperswaths $H_j(\vec{n})$ for integral j partitions $\Re^K$ into disjoint pieces.

Definition 2 *Let $\vec{n}_1, \vec{n}_2, ..., \vec{n}_K$ be a set linearly independent vectors in $\Re^K$. A* **tiling** $\mathcal{T}$ *is the K families of hyperswaths $\mathcal{F}(\vec{n}_i), 1 \leq i \leq K$.*

In the standard approach to tiling [W87, W89], the set of iterations that a nested loop executes, and the data dependences between them, are represented by an *iteration space graph* (ISG). For the purposes of this paper, the *iteration space* is simply a subset of $\Re^K$, typically one bounded by a polygonal region. The intuition behind our representation is that the lattice points of the iteration space (those points with all coordinates integral) are the iterations of the ISG. However, none of our theorems are concerned with these lattice points; instead in our model, the amount of computation is the volume of the iteration space.

We assume the program's data dependences are identical at each iteration, and can be modeled by a finite set of *dependence vectors* in $\Re^K$. These vectors provide constraints on the allowable order of execution. Furthermore, they dictate where communication is needed.

Definition 3 *Given an iteration space I, a tiling $\mathcal{F}(\vec{n}_1)$,...,$\mathcal{F}(\vec{n}_K)$, and a sequence of K integers $i_1, i_2, ..., i_K$, the* **tile** $\boldsymbol{T_{i_1,i_2,...,i_K}}$ *is $I \cap H_{i_1}(\vec{n}_1) \cap ... \cap H_{i_K}(\vec{n}_K)$.*

When $H_{i_1}(\vec{n}_1) \cap ... \cap H_{i_K}(\vec{n}_K)$ is a subset of I, then $T_{i_1,i_2,...,i_K}$ is called a **full tile**, *otherwise it is either a* **partial** *or* **empty** *(or* **non-existing***) tile.*

Figure 1 shows a tile in 2-dimensional space.

The full tiles are identical parallelepipeds, and it is (relatively) easy to generate code that iterates through the lattice points contained in the tile. Parallelepiped-shaped tiles are sufficiently general to implement the tiling methodologies presented in [IT88] and [WL91b],

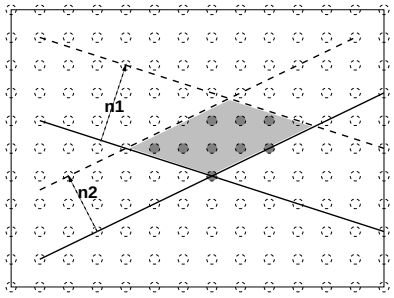

Figure 1: A tile which is the intersection of the iteration space (shown as a rectangle) and half-open hyperswaths with normals given by $\vec{n}_1$ and $\vec{n}_2$. The lattice points correspond to iterations; those included in the tile are shown as filled circles. Note that lattice points on the upper and the right-hand lines are *not* included in the tile.

which consider unimodular transformations with skewing. Some papers give a more general definition of a tiling, including hexagonal-shaped tiles [RAP87] or tiles of varying sizes. In particular, Flynn Hummel [H93] uses tiles of decreasing sizes to improve load balancing.

Given a tiling, there are certain dependences between tiles inherited from the program. We say there is a *tile dependence* from tile T_1 to tile T_2 if there is a path of dependences from an iteration in T_1 to an iteration in T_2.

The data input to and output from the iterations represented by a tile, referred to as the *surface* of the tile, may need to be moved to or from the processor executing the tile. The *volume* of the tile intuitively represents the computational work of the tile. It is desirable to have a small surface to volume ratio so that the computation time dominates the communication overhead, and so that data movement can be overlapped with computation.

In the remainder of this paper we will consider in detail a common instance of tiling. We assume the original program had the form:

```
for jj = 1 to n
    for kk = a1*jj+a2 to a1*jj+a2+m
        ... loop body ...
```

where `n, m, a1,` and `a2` are constants.

Note that the iteration space is a two-dimensional parallelogram. Later in the paper we will also consider triangular and trapezoidal iteration spaces, which arise when the lines representing the upper and lower bound on the second loop index (`kk`) have different slopes.

A tiling of the parallelogram will partition it into tiles $\{T_{j,k}\}$, which are smaller parallelograms (plus some partial tiles). It is well known that to ensure a *legal* tiling (one that allows tiles to be executed atomically in an order that satisfies the dependences), it is sufficient that $\vec{n} \cdot \vec{d} \geq 0$ for all hyperswath normals $\vec{n}$ and dependence vectors $\vec{d}$ [IT88].[1] Pictorially, a tiling is certain to be *legal* if the lower left-hand angle of a tile includes all dependence vectors. We will investigate the desirability of using tiles with wider angles. To rule out the uninteresting case of "embarrassingly parallel" problems, we assume there is a tile dependences from $T_{j-1,k}$ to $T_{j,k}$.

In our pictures (such as Figure 2), we draw the `kk` axis vertically. We consider tiling where the first family of hyperswaths are parallel to this axis. Let w be the width of these swaths. Each swath will be assigned to a processor. We consider two common ways of doing this assignment: a *block* distribution, where there are exactly P vertical swaths (one for each processor), and a *block cyclic* distribution, where there are bP swaths assigned P processors in a cyclic fashion (that is, swath j is assigned to processor j modulo P).

Each swath is partitioned into tiles via a second family of swaths, each intersecting the first family h apart as shown in Figure 2. Each full tile has area hw. In our figures, the vertical columns appear as a *stack of tiles*.

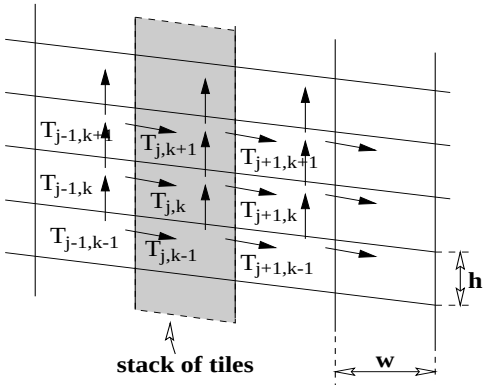

Figure 2: A typical picture of a 2-dimensional iteration space. The regions between the vertical hyperplanes, are called *stacks of tiles*, and are all of width w. The distance between the intersections of the crossing hyperplanes and the vertical hyperplanes is called the height h of a tile. The area of all full tiles is equal to hw. The arrows in the picture corresponds to either the execution order or to tile dependences between the tiles. Each tile is always dependent on the tile to its left, and we execute the tiles in a stack from bottom to top.

Both the block distribution and the block cyclic distribution are very advantageous scheduling algorithms. They reduce the number of memory accesses needed, since consecutive tiles executed on a processor share a surface (which increases locality), and limit the interprocessor communication to the dependences that cross the vertical cuts. They also allow pipelined execution of tiles, and are furthermore simple to implement.

3.2 Idle time

The cost function for a tiling we use in this paper is the total number of idle processor cycles that occur during the execution of a program on P processors. This idle time can occur for two different reasons: (a) *communication idle time* when a processor is waiting for data from another processor, or (b) *synchronization idle time* where it has finished all of the tiles assigned to it, and is waiting at a barrier for all other processors to finish their tiles.

We model communication by assuming that a processor sends its output surface as soon as the sender has

[1] This condition is not a necessary condition.

finished computing it, and the data is queued by the receiving processor. This strategy is desirable (though not optimal [VSM96]) in that the receiver will never unnecessarily wait for data, and also the sender can occupy itself with useful work while it itself is waiting for data. The time spent both sending and receiving the data are considered to be necessary work and not counted as idle time.

Idle time of type (a) occurs when a sender has not initiated sending the data early enough to eliminate all waiting time by the receiver. In a typical computer system, in order to achieve the most efficient communication, it is necessary for the sender to initiate the send operation some number L of cycles before the receiver has completed the previous tile. We will call $\boldsymbol{L}$ the *lead time* of the communication system. One can think of L as the amount of time it takes the first bit of a message to travel from the sender to the receiver, but in fact, there are many more complicated factors (such as operating system overheads and the ability of a system to overlap communication with computation) that contribute to L, and L is usually a function of the message length.

We define the parameter $\boldsymbol{c}= L/hw$. Thus, c is the lead time amortized over the computation of a full tile. Generally c is between 0 and 1, and we will assume[2] that $c > 0$.

Another delicate modeling question concerns the lead time needed to prevent idle cycles when the data corresponds to a partial tile. In this paper, we make the simplifying assumption that the lead time needed for a partial tile is proportional to the height of the tile's output surface (i.e., to the amount of data being transferred). Although beyond the scope of this paper, modifying this assumption would have little effect on our results.

Our assumption that computation time is proportional to the area of the tile is also inaccurate for several reasons: first, it doesn't accurately account for the communication overheads of partial tiles; second, there may be more computation overhead for the irregularly shaped partial tiles; third, the first and last stack of tiles have reduced communication costs; and fourth, there may be varying numbers of iterations in a given area, depending on the alignment of lattice points. Nevertheless our approximation is relatively accurate for large tiles.

Idle time is not the only cost that must be considered in evaluating a choice of tile size and shape. Other important factors are the total number of messages, and the total size of all the messages. These costs have been considered by many other papers that use a latency-bandwidth model of communication cost. But for fixed width w and height h tiles, both the number of messages and their total length will be constant. This allows us to isolate the effect of tile *shape* (as given by the direction of the second family of swaths) on idle time.

3.3 Notation

We now define the notation that will be used in the rest of the paper.

[2] In fact, it is possible to design a system that has negative c, though we know of no such real system.

The set of tiles is $\{T_{j,k} | 1 \leq j \leq bP\}$. The first non-empty tile in the first stack is numbered $T_{1,1}$. Depending on the relationship between the slope of the iteration space and the slope of the tiles, the lowest non-empty tile of other stacks might have any integer as its second subscript. Let $\boldsymbol{l_j}$ be the smallest integer such that T_{j,l_j} is non-empty.

For each (j,k) pair, we define $\boldsymbol{S_{j,k}}$ to be the union of of the tiles $T_{j,m}$ with $m \leq k$, that is, all tiles up to the k-th in the j-th stack.

Let $\boldsymbol{A_{T_{j,k}}}$ denote the area, which represents the *execution time*, of tile $T_{j,k}$. If $T_{j,k}$ is a full tile of width w and height h, then $A_{T_{j,k}} = hw$.

Similarly, we define $\boldsymbol{A_{S_{j,k}}}$ to be the area of $S_{j,k}$, and $\boldsymbol{A_{S_j}}$ to be the total area of the j-th stack.

$\boldsymbol{F_{T_{j,k}}}$ will denote the *finishing time* of tile $T_{j,k}$, that is, the elapsed time from the start of execution of $T_{1,1}$ to the end of execution of tile $T_{j,k}$ (including sending its output surface) by the processor assigned to it.

Let $\boldsymbol{y_{T_{j,k}}}$ denote the height of the right-hand output surface of $T_{j,k}$. As described in the discussion of lead time, we assume that tile $T_{j+1,k}$ cannot begin execution until $cwy_{T_{j,k}}$ after tile $T_{j,k}$ is completed.

Tile $T_{j,k}$ must wait for tile $T_{j-1,k}$ to its left, and tile $T_{j,k-1}$ below. Any other tile dependences will be from tiles that finished earlier than these two, and so will not further delay the finishing time of $T_{j,k}$. This gives the following formula:

Formula 1

$$\begin{aligned} F_{T_{j,k}} &= A_{T_{j,k}} + max(F_{T_{j,k-1}}, F_{T_{j-1,k}} + cwy_{T_{j-1,k}}) \\ F_{T_{j,k}} &= 0 \qquad \text{if } j < 1 \text{ or } k < l_j \end{aligned}$$

The next observation follows from the monotonicity of plus and max.

Observation 1 *Decreasing the finishing times of any number of tiles can never increase the finishing time of any other tile.*

The *execution time* of an application, $\boldsymbol{E}$, is the maximum finishing time of all tiles.

The *idle time of processor* p_j, denoted $\boldsymbol{I_{p_j}}$, is E minus the the sum of $A_{T_{m,k}}$ for all tiles $T_{m,k}$ that are executed on processor p_j. If $I_{p_1} = ... = I_{p_P}$ we refer to the processor idle time by $\boldsymbol{I_p}$. The definition of idle time includes both types of idle time, the *communication idle time* $\boldsymbol{I^c_{p_j}}$ that p_j spends waiting for data from other processors, and the *synchronization idle time* $\boldsymbol{I^s_{p_j}}$ after it has finished its last tile and before the application has finished.

It will turn out that when a processor is executing a stack of tiles, it may have idle time at the lowest tiles in the stack and additional idle time for the highest tiles in the stack, but if the stack is sufficiently high (in a technical sense defined later), the tiles in between will be executed in a "wavefront" with no additional idle time. We will write $\boldsymbol{I^c_{p_j}} = \boldsymbol{I^c_{p_j\,bottom}} + \boldsymbol{I^c_{p_j\,top}}$ to differentiate the communication idle time incurred by processor p_j at the stacks' bottoms from that at the stacks' tops.

Definition 4 *The* idle time of an application, I_a, *is the sum of the idle times of the individual processors.*

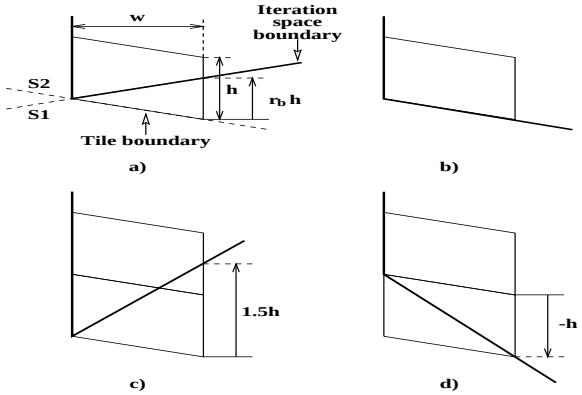

Figure 3: a) Illustration of the rise $r_b = \frac{w}{h}(s_1 - s_2)$, where s_1 is the slope of the iteration space and s_2 is the slope of the tile. Geometrically r_b is the number of tile heights that the iteration space boundary rises in the width of one tile. The rise turns out to greatly affect the processor idle time. b) $r_b = 0$. c) $r_b = 1.5$. d) $r_b = -1$.

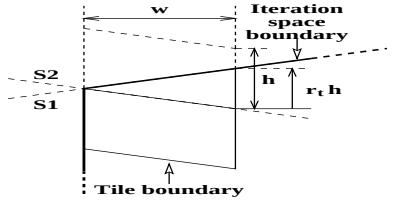

Figure 4: Rise r_t at the top of the iteration space.

$P = 2$	Execution Time
$r = 1$	$(14 + 5c)hw$
$r = 0$	$max(13 + c, 9 + 5c)hw$
$r = -1$	$max(12 + c, 4 + 4c)hw$

$P = 3$	Execution Time
$r = 1$	$(14 + 5c)hw$
$r = 0$	$max(10 + 2c, 9 + 5c)hw$
$r = -1$	$max(8 + 2c, 4 + 4c)hw$

$P = 6$	Execution Time
$r = 1$	$(14 + 5c)hw$
$r = 0$	$(9 + 5c)hw$
$r = -1$	$(4 + 4c)hw$

Table 1: Total execution times for varying values of r. The rise affects the execution time up to a factor of 3, depending on the values of c (chw is the finishing time of one full surface of length h) and P (the number of processors).

An important concept relating the iteration space to a tiling is the *rise*.

Definition 5 *The* **rise** r *is* $\frac{w}{h}(s_1 - s_2)$, *where* w *and* h *are the dimensions of the tile, and* s_1 *and* s_2 *are the slopes of the iteration space and the tile respectively. If the top and bottom slope of an iteration space are not the same, we distinguish between the bottom rise* r_b *(Figure 3) and the top rise* r_t *(Figure 4).*

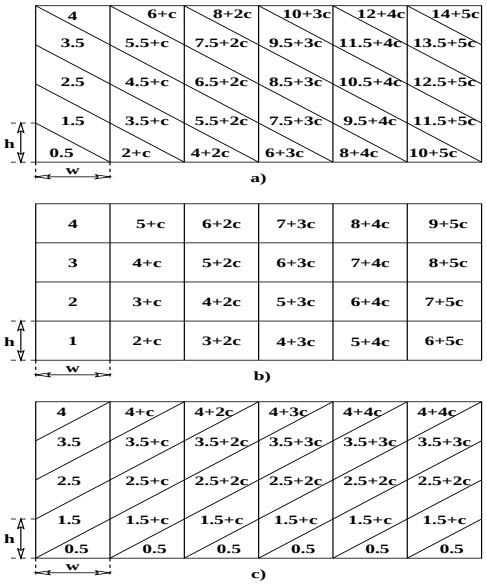

Figure 5: This picture illustrates the importance of choosing a tiling with a good value of r. The numbers inside the tiles are the finishing times for a normalized area $= hw = 1$ and $P = 6$. The iteration space and tiles in all three examples have the same area. Each tile depends on the tiles below it and to the left of it. In (a) $r = 1$, (b) $r = 0$, and (c) $r = -1$. Table 1 has the execution times for other values of P.

Figures 3 and 4 gives some examples of different values of rise. Geometrically r is the number of tile heights that the iteration space boundary rises in the width of one tile.

The following two observations, easy to see geometrically, are central to our proofs.

Observation 2 *When* $r_b \geq -1$, $A_{T_{j,k}} \leq A_{T_{j-1,k+1}}$

Observation 3 *When* $r_b \leq -1$, $A_{T_{j,k}} \geq A_{T_{j-1,k+1}}$

4 Preview of Results

It turns out that the execution time varies significantly with the rise, as illustrated in Figure 5(a), (b), and (c), where the rise is equal to 1, 0, and -1 respectively. The number written inside each tile is the finishing time, given at least six processors. Table 1 contains the execution times for 2, 3, and 6 processors.

The first processor in Figure 5(a) for example, can execute the five tiles in its first stack (two half tiles, and three whole) in 4 time steps. The second processor on the other hand, has to wait for the first processor, both before starting executing its first tile and also later, before executing its second tile, in order to receive the output values sent from the first processor.

For typical values of c in our example with $P = 6$, the total execution time for the tiling in Figure 5(a) is

Infinite iteration spaces	
$r_b \geq -1$	$I^c_{p_j\,bottom} = (j-1)(1+r_b+c)hw$
$r_b \leq -2$	$I^c_{p_j\,bottom} \leq max((c - \frac{1-r_b^2}{2r_b})hw, 0)$

Parallelogram-shaped iteration spaces Block distribution	
$r \geq -1$	$I_a = P(P-1)(1+r+c)hw$
$r \leq -2$	$I_a \leq max((c - \frac{1-r^2}{2r})Phw, 0) - \frac{rPhw}{2}$

Parallelogram-shaped iteration spaces Block cyclic distribution	
$r \geq -1$ $(A_{S_j} \geq Phw(1+r+c))$	$I_a = P(P-1)(1+r+c)hw$
$r \leq -2$	$I_a \leq max((c - \frac{1}{2r})bPhw, \frac{-rbPhw}{2})$

Trapezoidal iteration spaces $r_b < r_t$ Block distribution	
$r_b \geq -1$	$I_a = P(P-1)(1 + \frac{r_t+r_b}{2} + c)hw$
$r_b \leq -2 < -1 \leq r_t$	$I_a \leq max(c - \frac{1-r^2}{2r})Phw, 0) + P(P-1)\frac{r_t+r_b}{2}hw$
$r_b < r_t < -1, r_b \leq -2$	$I_a \leq max(c - \frac{1-r^2}{2r})Phw, 0) + P(P-1)\frac{r_t+r_b}{2}hw - \frac{r_tPhw}{2}$

Trapezoidal iteration spaces $r_t < r_b$ Block distribution	
$-1 \leq r_t < r_b$	$I_a = P(P-1)(1 + \frac{r_t+r_b}{2} + c)hw$
$-(1+c) \leq r_t < -1 \leq r_b$	$I_a \leq P(P-1)(1 + \frac{r_t+r_b}{2} + c)hw - \frac{r_tPhw}{2}$
$r_t \leq -(1+c) < -1 \leq r_b$	$I_a \leq -P(P-1)\frac{r_t-r_b}{2}hw - \frac{r_tPhw}{2}$
$r_t < r_b \leq -2$	$I_a \leq max(c - \frac{1-r^2}{2r})Phw, 0) - P(P-1)\frac{r_t-r_b}{2}hw - \frac{r_tPhw}{2}$

Table 2: Summary of results.

about three times longer than for the tiling shown in Figure 5(c). This is because in 5(a), each processor needs to wait while two tiles are executed at the previous processor, whereas in 5(c) all processors can start at the same time. Furthermore we see in Figure 5(b) that a tiling using rectangular tiles ($r = 0$) results in an execution time that is about twice as long as when $r = -1$.

Table 2 is a summary of the results that will be presented in the rest of the paper. We look at five tiling scenarios as given by the table headings. The case of "infinite iteration space" examines the idle time $I^c_{p_j\,bottom}$ that occurs at the bottom of P sufficiently tall stacks. It will be shown that, although there may be idle time for several tiles at the bottom of each stack, once a processor reaches a row that contains only full tiles, it will incur no additional idle time (until the stack top is encountered). The next two cases are for parallelogram-shaped iteration spaces with block and block cyclic distributions. The last cases examine trapezoidal (which includes triangular) iteration spaces with block distribution, with different cases depending on whether successive stacks are becoming taller or shorter.

Each of these cases are further divided into subcases, depending on the rise(s). For parallelograms, the reason there are two intervals of the rise is that when $r_b \leq -2$, the only possible communication idle time for a stack of tiles is waiting for the first two messages at the bottom of the stack, whereas in the case when $r_b \geq -1$, messages never arrive ahead of when they are needed. When $-2 < r_b < -1$, neither statement is necessarily true, but depend on the where the bottom edge of the iteration space intersects the stack. This complicates the formulas considerably and we will not cover this interval of r_b in this paper.

It is convenient, because of this different behavior in the two cases, to define a constant v that in some cases gives the slope of the "wavefront" of parallel execution of tiles.

Definition 6 *We define* $\boldsymbol{v}$ *to be* $-(1+c)\frac{h}{w}$ *when* $r_b \geq -1$ *and* $v = r_b\frac{h}{w}$ *when* $r_b \leq -2$.

Section 5 examines infinitely high iteration spaces, parallelograms are covered in Section 6.1, and trapezoids and triangles are covered in Section 6.2. Because of space limitations, only the proofs of the first lemmas are included here. The full set of proofs can be found in the technical report [HCF96].

5 Infinite iteration spaces

5.1 Total processor idle time for $r_b \geq -1$

Assume that we extend the iteration space to infinity towards the top edge of the original iteration space. We will start by obtaining a formula for the finishing time for any tile $T_{j,k}$ in a row where all tiles to its left are full tiles.

Since it isn't well defined what a block cyclic distribution for an infinite iteration space is, we are only covering block distribution in this section.

Lemma 1 *When $r_b \geq -1$ and $T_{j,k}$ is a tile in row k (for $j,k \geq 1$) where all tiles $T_{l,k}$ are full (for $1 \leq l \leq j$), then*

$$F_{T_{j,k}} = A_{S_{1,k}} + (j-1)(c+1)hw$$

Proof: *See appendix.* □

The lemma shows that after the k-th row, the finishing time of the tiles in the j-th stack are a constant larger than the corresponding area of the first stack. Since all tiles are full above the k-th row, this means both the work and the finishing time of tiles in the j-th stack are increasing by hw. Thus no additional idle time is introduced above the k-th row. This justifies the earlier discussion of the definition of $I^c_{p_j\,bottom}$ where "sufficiently high" means "above row k". We have:

Corollary 1 *When $r_b \geq -1$,*

$$I^c_{p_j\,bottom} = (j-1)(1+r_b+c)hw$$

Proof: *See appendix.* □

5.2 Total processor idle time for $r_b \leq -2$

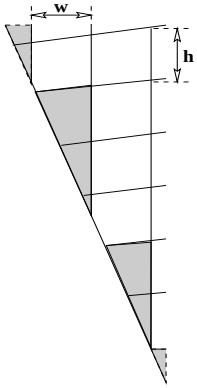

Figure 6: Iteration space for $r_b \leq -2$. The shaded area correspond to the computation that can be executed immediately without waiting for data from another processor.

When the rise is less than -2 we see in Figure 6 that all processors can start executing at the same time; not only their first tile but also the second and, depending on the rise, maybe more. (The shaded areas in Figure 6 correspond to these tiles.) This changes the formula considerably, since now more of the time that otherwise would be spent waiting for earlier processors is being overlapped by computation. We will show that the total idle time for an arbitrary stack up to and including its first full tile is given by the following formula in an infinite iteration space.

$$I^c_{p_j\,bottom} \leq max((c - \frac{1-r_b^2}{2r_b})hw, 0)$$

Lemma 2 *When $r_b \leq -2$, $j \geq 2$, and $T_{j,k}$ is a full tile, then*

$$F_{T_{j,k}} \leq A_{S_{j,k}} + max((c - \tfrac{1-r_b^2}{2r_b})hw, 0)$$

Proof: *See appendix.* □

Corollary 2 *When $r_b \leq -2$ and $j \geq 2$,*

$$I^c_{p_j\,bottom} \leq max((c - \frac{1-r_b^2}{2r_b})hw, 0)$$

Proof: *See appendix.* □

It can be shown that this approximation is off by at most $\frac{hw}{4}$.

6 Finite iteration spaces

We have so far derived formulas for the idle time in infinite iteration spaces, but we need now to derive similar formulas for finite iteration spaces. Removing computation from the infinite iteration space to make it finite will in some cases introduce extra idle time at the top of each stack.

The idle time that occur at the top of the l-th stack in an iteration space can occur for two reasons. The processor executing the l-th stack might be idle while it still have computation left to do in the l-th stack. We call this idle time $I^c_{l\,top}$. If l is the last stack to be executed by processor p_j, then there might also be idle time due to synchronization after l has been executed. The following two lemmas state the formulas for these two idle times. (The proofs for these lemmas, and all subsequent lemmas and theorems can be found in [HCF96].)

Lemma 3

$$I^c_{l\,top} \begin{cases} = 0 & (for\ r_t \geq -1) \\ \leq \frac{-r_t hw}{2} & (for\ r_t < -1) \end{cases}$$

Lemma 4

$$I^s_{p_j} = \begin{cases} hw(r_t - v\frac{w}{h})(P-j) & (for\ r_t \geq v\frac{w}{h}) \\ -hw(r_t - v\frac{w}{h})(j-1) & (for\ r_t \leq v\frac{w}{h}) \end{cases}$$

where r_t is the top rise and v is the wavefront constant.

We are now ready to derive formulas for the idle time. These formulas all come from the definition of $I_{p_j} = I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j}$, where $I^c_{p_j\,top} = \sum_{l=j \bmod P} I^c_{l\,top}$. The formulas concerning block distribution are then all obtained straightforwardly using lemmas 3, 4 and Corollary 1 or 2. In the case of block cyclic distribution, $I^c_{p_j\,bottom}$ might also differ from the formulas in Corollary 1 and 2 (see proofs of Lemma 7 and 8).

6.1 Parallelogram shaped iteration spaces

When the iteration space is a parallelogram, all stacks have the same height, and assuming that they also have the same width, all processors will therefore perform the same amount of computation. According to the definition of processor idle time we can conclude that $I_{p_1} = ... = I_{p_P} = I_p$. The total idle time, I_a, therefore becomes equal to PI_p.

6.1.1 Block Distribution for $r \geq -1$

We claim that the idle time for the processor executing the j-th stack is given by the following lemma.

Lemma 5 *Let $r \geq -1$ and assume that there exists a row k, such that for all stacks, there is a full tile in k.*[3] *The total processor idle time for the processor executing the j-th stack is then given by the following formula.*

$$I_p \quad = \quad (P-1)hw(1+r+c)$$

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r is the rise.

The next theorem follows from this lemma using the fact that the idle time of an application is the sum of the idle times of the individual processors.

Theorem 1 *Let $r \geq -1$. If there are P stacks distributed one per processor, and there exists a row which includes full tiles in every stack, then the total idle time will be*

$$I_a \quad = \quad P(P-1)hw(1+r+c)$$

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r is the rise.

6.1.2 Block Distribution for $r \leq -2$

The total idle time for an arbitrary stack is given by the following lemma.

Lemma 6 *If $r \leq -2$ and each processor only executes one stack, then the following formula is an upper bound on the total idle time for one processor.*

$$I_p \quad \leq \quad max((c - \frac{1}{2r})hw, \frac{-rhw}{2})$$

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, and r is the rise.

As before, this lemma gives the following theorem.

Theorem 2 *If $r \leq -2$ and there are P stacks, then*

$$I_a \quad \leq \quad max((c - \frac{1}{2r})Phw, \frac{-rPhw}{2})$$

is an upper bound on the total idle time.

[3] Here and in subsequent sections we assume that if the stacks are so short that there is no row with a full tile in each stack, then we will assume that the stacks are higher and the results that we derive will then be an upper bound on the idle time. This follows from Observation 1.

6.1.3 Block Cyclic Distribution for $r \geq -1$

In the case of block cyclic distribution, we assume there are P processors executing bP stacks, such that p_j executes stack numbers j mod P. As before we assume the stacks to be "sufficiently high" (there is at least one row of full tiles), but in the block cyclic case we also need the assumption that they have an area of at least $Phw(1+r+c)$. This last assumption turns out to be sufficient to assure that a processor can start executing the next stack immediately after the finishing its current stack.

We claim that the idle time is given by the following lemma.

Lemma 7 *Let $r \geq -1$ and p_j be the processor that executes stack numbers $j, j+P, j+2P, ..., j+(b-1)P$. Let all stacks have an area of at least $Phw(1+r+c)$, and let there exist a row in which all tiles are full.*

$$I_p \quad = \quad (P-1)(1+r+c)hw$$

Theorem 3 *If $r \geq -1$ and there are bP stacks as described in Lemma 7, then the total idle time is given by*

$$I_a \quad = \quad P(P-1)(1+r+c)hw$$

6.1.4 Block Cyclic Distribution for $r \leq -2$

When $r \leq -2$ we do not need to make any assumptions on the height of the stacks.

Lemma 8 *Let $r \leq -2$ and p_j be the processor that executes stack numbers $j, j+P, j+2P, ..., j+(b-1)P$.*

$$I_p \quad \leq \quad max((c - \frac{1}{2r})bhw, \frac{-rbhw}{2})$$

Theorem 4 *If $r \leq -2$ and there are bP stacks, then the total idle time is given by*

$$I_a \quad \leq \quad max((c - \frac{1}{2r})bPhw, \frac{-rbPhw}{2})$$

6.2 Trapezoidal iteration spaces

6.2.1 Block Distribution for $-1 \leq r_b < r_t$

We claim that the idle time for the processor executing the j-th stack is given by the following lemma.

Lemma 9 *Let $r_t > r_b \geq -1$ and assume that there exists a row k such that for all stacks there is a full tile in k. The total processor idle time for the processor executing the j-th stack is then given by the following formula.*

$$I_{p_j} = hw((j-1)(1+r_b+c) + (r_t+1+c)(P-j))$$

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r is the rise.

Theorem 5 *If there are P stacks distributed one per processor, and there exist a row which includes full tiles in every stack, then the total idle time for $-1 \leq r_b < r_t$ will be*

$$I_a = P(P-1)hw(1+c+\frac{r_t+r_b}{2})$$

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r_b and r_t are the bottom and top rise.

6.2.2 Block Distribution for $r_b \leq -2$, $r_b < r_t$

The total idle time for the processor executing the j-th stack is given by the following lemma.

Lemma 10 *If $r_b \leq -2$ and each processor only executes one stack, then the following formula is a upper bound on the total idle time for one processor.*
For $r_b \leq -2 < -1 \leq r_t$**:**

$$I_{p_j} \leq max((c-\frac{1-r_b^2}{2r_b})hw, 0) + hw(r_t - r_b)(P-j)$$

For $r_b < r_t < -1, r_b \leq -2$**:**

$$I_{p_j} \leq max((c-\frac{1-r_b^2}{2r_b})hw, 0) + hw(r_t - r_b)(P-j) - \frac{r_t hw}{2}$$

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, r_b is the rise at the bottom of the iteration space and r_t is the rise at the top.

Theorem 6 *If $r_b \leq -2$, $r_b < r_t$ and there are P stacks, then*
For $r_b \leq -2 < -1 \leq r_t$**:**

$$I_a \leq max(P(c-\frac{1-r_b^2}{2r_b})hw, 0) + P(P-1)hw\frac{r_t - r_b}{2}$$

For $r_b < r_t < -1, r_b \leq -2$**:**

$$I_a \leq max(P(c-\frac{1-r_b^2}{2r_b})hw, 0) + P(P-1)hw\frac{r_t - r_b}{2} - \frac{r_t hwP}{2}$$

is an upper bound on the total idle time.

6.2.3 Block Distribution for $r_b > r_t$, $r_b \geq -1$

We claim that the idle time for the processor executing the j-th stack is given by the following lemma.

Lemma 11 *Let $r_b \geq -1$ and assume that there exists a row k, such that for all stacks, there is a full tile in k. The total processor idle time for the processor executing the j-th stack is then given by the following formula.*
For $-1 \leq r_t < r_b$**:**

$$I_{p_j} = hw((j-1)(1+r_b+c) + (P-j)(1+r_t+c))$$

For $-(1+c) \leq r_t < -1 \leq r_b$**:**

$$I_{p_j} \leq hw((j-1)(1+r_b+c) + (P-j)(1+r_t+c) - \frac{r_t}{2})$$

For $r_t \leq -(1+c) < -1 \leq r_b$**:**

$$I_{p_j} \leq -hw((j-1)(r_t - r_b) + \frac{r_t}{2})$$

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r_b is the rise at the bottom of the iteration space and r_t is the rise at the top.

Theorem 7 *Let $r_b \geq -1$. If there are P stacks distributed one per processor, and there exist a row which includes full tiles in every stack, then the total idle time will be*
For $-1 \leq r_t < r_b$**:**

$$I_a = P(P-1)hw(1+c+\frac{r_t+r_b}{2})$$

For $-(1+c) \leq r_t < -1 \leq r_b$**:**

$$I_a \leq P(P-1)hw(1+c+\frac{r_t+r_b}{2}) - \frac{r_t Phw}{2}$$

For $r_t \leq -(1+c) < -1 \leq r_b$**:**

$$I_a \leq -P(P-1)hw(\frac{r_t - r_b}{2}) - \frac{r_t Phw}{2}$$

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r_b and r_t are the bottom and top rise.

6.2.4 Block Distribution for $r_t < r_b \leq -2$

The total idle time for the processor executing the j-th stack is given by the following lemma.

Lemma 12 *If $r_t < r_b \leq -2$ and each processor only executes one stack, then the following formula is an upper bound on the total idle time for one processor.*

$$I_{p_j} \leq max((c-\frac{1-r_b^2}{2r_b})hw, 0) - hw(r_t - r_b)(j-1) - \frac{r_t hw}{2}$$

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, and r_b and r_t are the bottom and top rise.

Theorem 8 *If $r_t < r_b \leq -2$ and there are P stacks, then*

$$I_a \leq max(P(c-\frac{1-r_b^2}{2r_b})hw, 0) - P(P-1)hw(\frac{r_t - r_b}{2}) - \frac{Pr_t hw}{2}$$

7 Idle time tradeoffs for multilevel tiling

This section will illustrate how paying attention to idle time can be important in the context of tiling for multiple levels of memory and parallelism hierarchy.

In Figure 7 an iteration space with vertical and horizontal dependences is partitioned into tiles and these tiles are again partitioned into subtiles. We will assume that the original iteration space and the subtiles are rectangular and consider the effect of the shape of the intermediate tiles. In Figure 7a) we use a square intermediate tile whereas in b) we use a sloping parallelogram. Conventional tiling techniques would choose the tiling of Figure 7a). We argue that in some circumstances the tiling in b) is more desirable.

Suppose for example the intermediate tiles are brought in from disk to main memory and the subtiles are executed on individual processors. Since there is only a single main memory, i.e. $P = 1$ for the coarse granularity tiles, there is no idle time at this level. However,

the idle time to execute each subtile using multiple processors is significant. In the case of square tiles in Figure 7a) *each tile* incur a substantial idle time penalty. However for the sloping tiles, even though there is a similar penalty for the partial tiles, the full tiles will be able to execute without idle time.

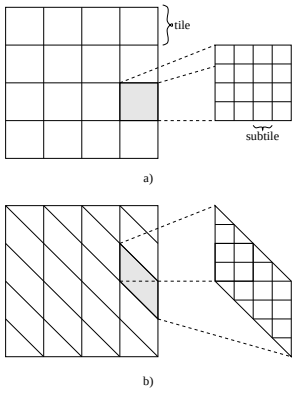

Figure 7: This figure shows two different two level tilings of an iteration space. The first has rise of zero in both cases, whereas the second has negative rise for subtiles and positive for tiles. Which of the two is more efficient depends on the degree of parallel execution at the two levels.

We have observed the importance of idle time in actual programs [CFHAG96]. The first example is a PDE-like nested loop, tiled for both cache and registers on the IBM RS/6000. Conventional tiling would produce rectangular cache tiles and an unrolled loop (which can be considered to be a very small rectangular "register tile"). Hierarchical tiling would also choose rectangular cache tiles; however, because of the pipelined functional unit of depth 2 in the RS/6000, which is modeled as two-way parallelism, register tiles are chosen to be parallelograms with rise of +1 with respect to the cache tiles. Since there is only one set of registers per cache, having a positive rise doesn't lead to any idle time. However, each register tile is further partitioned into "instruction tiles". The choice of register tile shape makes the instruction tiles have rise of -1, which gives greater instruction level parallelism. The code produced by hierarchical tiling ran 64% faster than the conventionally tiled code on the RS/6000, taking an average of 3.23 cycles per iteration point compared to 5.31. The second example is a protein matching code with a similar set of considerations. In this example on the IBM SP2, the inner loop code with rectangular register tiles ran at 8.0 cycles per iteration, whereas a register tile a parallelogram with rise of -1 ran at 6.37 cycles per iteration, i.e., 26% faster. The difference in performance on these two codes illustrates that idle time can have a significant effect on real performance.

References

[B92] Banerjee, U., "Unimodular Transformations of Double Loops", in *Advances in Languages and Compilers for Parallel Processing*, A. Nicolau and D. Padua, editors, MIT Press, 1992.

[CFH95a] Carter, L., J. Ferrante and S. Flynn Hummel, "Efficient Parallelism via Hierarchical Tiling," Proc. of SIAM Conference on Parallel Processing for Scientific Computing, (February, 1995).

[CFH95b] Carter, L., J. Ferrante and S. Flynn Hummel, "Hierarchical Tiling for Improved Superscalar Performance," *Ninth International Parallel Processing Symposium*, Santa Barbara, CA, April 1995.

[CFHAG96] Carter, L., J. Ferrante, S. Flynn Hummel, B. Alpern, K. S. Gatlin, "Hierarchical Tiling: A Methodology for High Performance," UCSD Computer Science and Engineering Technical Report, available at http://www-cse.ucsd.edu/users/carter/ppbib.html.

[CK92] Carr, S. and K. Kennedy, "Compiler Blockability of Numerical Algorithms," *Proc. of Supercomputing*, November, 1992.

[GJG88] Gannon, D., W. Jalby and K. Gallivan, "Strategies for Cache and Local Memory Management by Global Program Transformation," *Journal of Parallel and Distributed Computing*, Vol. 5, No. 5, October 1988, pp. 587-616.

[HCF96] Högstedt, K., L. Carter and J. Ferrante, "Determining the Idle Time of a Tiling," UCSD Computer Science and Engineering Technical Report, no. CS96-489, 1996, available at http://www-cse.ucsd.edu/users/carter/ppbib.html.

[HA90] Hudak, D. E. and S. G. Abraham, "Compiler Techniques for Data Partitioning of Sequentially Iterated Parallel Loops," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2 No. 3, July, 1991, pp. 318-328.

[H93] Hummel, S. F., "Fractiling: A Method for Scheduling Parallel Loops on NUMA Machines", IBM RC 18958, June 1993.

[IT88] Irigoin, F. and R. Triolet, "Supernode Partitioning," *Proc. 15th ACM Symp. on Principles of Programming Languages*, January 1988, pp. 319-328.

[JM86] Jalby, W. and U. Meier, "Optimizing Matrix Operations on a Parallel Multiprocessor with a Hierarchical Memory System," *Proc. International Conference on Parallel Processing*, August 1986, pp. 429-432.

[KM92] Kennedy, K. and K. S. Mc Kinley, "Optimizing for Parallelism and Data Locality," *Proc. International Conference on Supercomputing*, July 1992, pp. 323-334.

[KP93] Kelly, W. and W. Pugh, "Determining Schedules based on Performance Estimation," *Technical report, Dept. of Computer Science, University of Maryland, College Park* UMIACS-TR-93-67, December 1993.

[LRW91] Lam, M, E. Rothberg, and M. Wolf, "The Cache Performance and Optimization of Blocked Algorithms," *Proc. ASPLOS*, April 1991, pp. 63-74.

[L90] Lee, F. F., "Partitioning of Regular Computation on Multiprocessor Systems," *Journal of Parallel and Distributed Computing*, Vol. 9, 1990, pp. 312-317.

[RS91] Ramanujam, J. and P. Sadayappan, "Tiling Multidimensional Iteration Spaces for Nonshared Memory Machines," *Proceedings of Supercomputing*, November, 1991, pp. 111-120.

[RAP87] Reed, D. A., L. M. Adams and M. L. Patrick, "Stencil and Problem Partitionings: Their Influence on the Performance of Multiple Processor Systems," *IEEE Transactions on Computers*, July, 1987, pp. 845-858.

[VSM96] Van der Wijngaart, R.F., S.R. Sarukkai, and P. Mehra, "The Effects of Interrupts on Software Pipeline Execution on Message-passing Architectures," *International Conference on Supercomputing*, May, 1996.

[WL91] Wolf, M. E. and M. S. Lam, "A Data Locality Optimizing Algorithm," *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, June, 1991, pp. 30-44.

[WL91b] Wolf, M. E. and M. S. Lam, "A Loop Transformation Theory and an Algorithm to Maximize Parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 4, October, 1991, pp. 452-471.

[W87] Wolfe, M., "Iteration Space Tiling for Memory Hierarchies," *Parallel Processing for Scientific Computing*, G. Rodrigue (Ed), SIAM 1987, pp. 357-361.

[W89] Wolfe, M., "More Iteration Space Tiling ," *Proceedings of Supercomputing*, November, 1989, pp. 655-664.

Appendix

Infinite iteration spaces

Lemma 1 *When $r_b \geq -1$, $j,k \geq 1$ and $T_{j,k}$ is a tile in row k where all tiles $T_{l,k}$, $l \leq j$ are full, then*
$F_{T_{j,k}} = A_{S_{1,k}} + (j-1)(c+1)hw$

Proof of Lemma 1: We first need two additional lemmas (13 and 14).

Lemma 13 *Let $T_{j,k}$ be a tile in row k such that $T_{l,k}, l \leq j$ are full tiles.*
$F_{T_{j,k}} \leq F_{T_{j-1,k+1}} + chw$ *(for $j \geq 2$)*

Proof of Lemma 13:
Induction Hypothesis:
$F_{T_{j,k}} \leq F_{T_{j-1,k+1}} + chw$ (for $j \geq 2$)
for all tiles within the iteration space boundaries. $F_{T_{j,k}} = 0$ otherwise.
Base Case ($j+k=2$):
$A_{T_{j,k}} = 0$**:**
Formula 1 states that the finishing time of a non-existing tile is equal to 0, and the induction hypothesis is therefore trivially true in this case.
$A_{T_{j,k}} \neq 0$**:**
$F_{T_{j,k}} = max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}}$ (Form. 1)
Now, Formula 1 says that the finishing time for non-existing tiles is equal to 0, and when $r_b \geq -1$ we know that both $T_{j-1,k}$ and $T_{j,k-1}$ do not exist. We therefore have that

$$\begin{array}{lll} F_{T_{j,k}} \leq & max(0+0,0) + A_{T_{j,k}} & (r_b \geq -1, \text{Form. 1}) \\ \leq & max(0+0,0) + A_{T_{j-1,k+1}} & (\text{Obs. 2}) \\ \leq & max(F_{T_{j-2,k+1}} + cwy_{T_{j-2,k+1}}, F_{T_{j-1,k}}) + A_{T_{j-1,k+1}} & \\ \leq & F_{T_{j-1,k+1}} & (\text{Form. 1}) \\ \leq & F_{T_{j-1,k+1}} + chw & \end{array}$$

for $j+k=2$.
Induction Step: Suppose the induction hypothesis holds for $j+k=x-1$, show it also holds for $j+k=x$.

$$\begin{array}{lll} F_{T_{j,k}} = & max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}} & (\text{Form. 1}) \\ \leq & max(F_{T_{j-1,k}} + chw, F_{T_{j,k-1}}) + A_{T_{j,k}} & \\ \leq & max(F_{T_{j-1,k}} + chw, F_{T_{j-1,k}} + chw) + A_{T_{j,k}} & (\text{Induction hypothesis}) \\ \leq & F_{T_{j-1,k}} + chw + A_{T_{j,k}} & \\ \leq & F_{T_{j-1,k}} + chw + A_{T_{j-1,k+1}} & (\text{Obs. 2}) \\ \leq & F_{T_{j-1,k+1}} + chw & \end{array}$$

since $F_{T_{j-1,k+1}} \geq F_{T_{j-1,k}} + A_{T_{j-1,k+1}}$ by the definition of finishing time (Form 1). We have now proved the induction hypothesis and lemma.
End of Proof of Lemma 13 □

Lemma 14 *Let $T_{j,k}$ be a tile in row k such that $T_{l,k}, l \leq j$ are full tiles.*
$F_{T_{j,k}} = F_{T_{j-1,k}} + (c+1)hw$ *(for $j \geq 2$)*

Proof of Lemma 14

$$\begin{array}{lll} F_{T_{j,k}} = & max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}} & (\text{Form. 1}) \\ = & max(F_{T_{j-1,k}} + chw, F_{T_{j,k-1}}) + A_{T_{j,k}} & (A_{T_{j-1,k}} = hw) \\ = & F_{T_{j-1,k}} + chw + A_{T_{j,k}} & (\text{Lemma 13}) \\ = & F_{T_{j-1,k}} + chw + hw & (A_{T_{j,k}} = hw) \\ = & F_{T_{j-1,k}} + (c+1)hw & \end{array}$$

End of Proof of Lemma 14 □
We are now ready to complete the proof of Lemma 1.

$$\begin{array}{llll} F_{T_{j,k}} = & F_{T_{j-1,k}} + (c+1)hw & (\text{for } j \geq 2) & (\text{Lemma 14}) \\ = & F_{T_{1,k}} + (j-1)(c+1)hw & (\text{for } j \geq 1) & (\text{by simple induction}) \\ = & A_{S_{1,k}} + (j-1)(c+1)hw & & (\text{by simple induction}) \end{array}$$

End of Proof of Lemma 1 □

Corollary 1

$$I^c_{p_j\,bottom} = (j-1)(1+r_b+c)hw$$

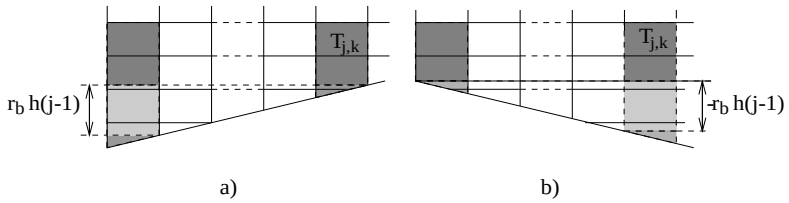

Figure 8: Iteration spaces showing that the difference between the area of stack 1 and stack j up to a certain level always differs with $r_b h(j-1)w$. This fact is used in the proof of Corollary 1.

when $r_b \geq -1$.

Proof of Corollary 1: The idle time for p_j in an infinite iteration space is given by the finishing time of a tile $T_{j,k}$ in row k such that $T_{l,k}, l \leq j$ are full tiles, minus the amount of computation p_j has executed before $F_{T_{j,k}}$

Using the notation in Figure 8, and Lemma 1, we have that

$$\begin{aligned} I^c_{p_j\,bottom} &= F_{T_{j,k}} - A_{S_{j,k}} && \text{(for } j \geq 1\text{)} \\ &= A_{S_{1,k}} + (j-1)(c+1)hw - A_{S_{j,k}} && \text{(Lemma 1)} \\ &= (A_{S_{1,k}} - A_{S_{j,k}}) + (j-1)(c+1)hw \\ &= r_b h(j-1)w + (j-1)(c+1)hw && \text{(Figure 8)} \\ &= (j-1)(1+r_b+c)hw \end{aligned}$$

where $A_{S_{j,k}}$ is the area of the j-th stack up to, and including row k.

End of Proof of Corollary 1 □

Lemma 2 *When* $r_b \leq -2$, $j \geq 2$ *and* $T_{j,k}$ *is a full tile, then*

$$F_{T_{j,k}} \leq A_{S_{j,k}} + max((c + \tfrac{1-r_b^2}{2r_b})hw, 0)$$

Proof of Lemma 2: We first need to prove the following lemma.

Lemma 15 $F_{T_{j,k}} \geq F_{T_{j-1,k+1}} + chw$ *(for* $j \geq 2, k \geq f_j$*)*
where f_j *is the value of* k *such that* $T_{j,k}$ *is the first full tile in stack* j.

Proof of Lemma 15:
Induction Hypothesis:

$$F_{T_{j,k}} \geq F_{T_{j-1,k+1}} + cwy_{T_{j-1,k+1}} \quad \text{(for } j \geq 1\text{)}$$

for all tiles within the iteration space boundaries. $F_{T_{j,k}} = 0$ otherwise.

Base Case ($j + k = l_{bP} + 1$, where l_{bP} is the lowest value of k such that $T_{bP,k}$ exists.)**:**

$A_{T_{j,k}} = 0$**:**

Formula 1 states that the finishing time of a non-existing tile is equal to 0, and the induction hypothesis is therefore trivially true in this case.

$A_{T_{j,k}} \neq 0$**:**

$$F_{T_{j,k}} = max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}} \quad \text{(Form. 1)}$$

Now, Formula 1 says that the finishing time for non-existing tiles is equal to 0, and when $r_b \leq -2$ we know that both $T_{j-1,k}$ and $T_{j,k-1}$ do not exist when $j + k = l_{bP} + 1$. We therefore have that

$$\begin{aligned} F_{T_{j,k}} &= max(0+0,0) + A_{T_{j,k}} && (r_b \leq -2, \text{Form. 1}) \\ &\geq max(0+0,0) + A_{T_{j-1,k+1}} && \text{(Obs. 3)} \\ &\geq max(F_{T_{j-2,k+1}} + cwy_{T_{j-2,k+1}}, F_{T_{j-1,k}}) + A_{T_{j-1,k+1}} && (A_{T_{j-2,k+1}} = A_{T_{j-1,k}} = 0) \\ &\geq F_{T_{j-1,k+1}} && \text{(Form. 1)} \\ &\geq F_{T_{j-1,k+1}} + cwy_{T_{j-1,k+1}} && (A_{T_{j-1,k+1}} = 0) \end{aligned}$$

Induction Step: Suppose the induction hypothesis holds for $j + k = x - 1$, show it also holds for $j + k = x$.

$$\begin{aligned} F_{T_{j,k}} &= max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}} && \text{(Form. 1)} \\ &\geq max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j-1,k}} + cwy_{T_{j-1,k}}) + A_{T_{j,k}} && \text{(Induction hypothesis)} \\ &\geq F_{T_{j-1,k}} + cwy_{T_{j-1,k}} + A_{T_{j,k}} \\ &\geq F_{T_{j-1,k}} + cwy_{T_{j-1,k}} + A_{T_{j-1,k+1}} && \text{(Obs. 3)} \\ &\geq max(F_{T_{j-2,k+1}} + cwy_{T_{j-2,k+1}}, F_{T_{j-1,k}}) + cwy_{T_{j-1,k}} + A_{T_{j-1,k+1}} && \text{(Induction hypothesis)} \\ &\geq F_{T_{j-1,k+1}} + cwy_{T_{j-1,k}} \end{aligned}$$

Now we have shown that $F_{T_{j,k}} \geq F_{T_{j-1,k}} + cwy_{T_{j-1,k}}$ for all tiles. We want to show $F_{T_{j,k}} \geq F_{T_{j-1,k}} + cwh$ for $k \geq f_j$ and this is trivially true since when $T_{j,k}$ is full, $y_{T_{j-1,k}} = h$, and we have now proved the lemma.

End of Proof of Lemma 15 □

Using Lemma 15 we now have

$$\begin{aligned} F_{T_{j,k}} &= max(F_{T_{j-1,k}} + cwy_{T_{j-1,k}}, F_{T_{j,k-1}}) + A_{T_{j,k}} && && \text{(Form. 1)} \\ &= F_{T_{j,k-1}} + A_{T_{j,k}} && \text{(for } k > f_j\text{)} && \text{(Lemma 15)} \end{aligned} \quad (1)$$

To calculate the finishing time for $k > f_j$ we now need to calculate the finishing time for $T_{j,k}$, the first full tile in stack j.

Lemma 16 *Let T_{j,f_j} be the first full tile in stack j. Then*

$$F_{T_{j,f_j}} \leq max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + A_{S_{j,f_j}} \quad \textit{(for } j \geq 2\textit{)}$$

Proof of Lemma 16 Let us first calculate the finishing time for the first full tiles neighboring tiles.

$$\begin{aligned} F_{T_{j-1,f_j}} &= max(F_{T_{j-2,f_j}} + cwy_{T_{j-2,f_j}}, F_{T_{j-1,f_j-1}}) + A_{T_{j-1,f_j}} && \text{(Form. 1)} \\ &= max(F_{T_{j-2,f_j}} + cwy_{T_{j-2,f_j}}, A_{T_{j-1,f_j-1}}) + A_{T_{j-1,f_j}} && (A_{T_{j-2,f_j-1}} = A_{T_{j-1,f_j-2}} = 0 \text{ when } r_b \leq -2) \\ &= max(0 + 0, A_{T_{j-1,f_j-1}}) + A_{T_{j-1,f_j}} && (A_{T_{j-2,f_j}} = 0) \\ &= A_{T_{j-1,f_j-1}} + A_{T_{j-1,f_j}} \\ &= A_{S_{j-1,f_j}} \end{aligned} \quad (2)$$

$$\begin{aligned} F_{T_{j,f_j-1}} &= max(F_{T_{j-1,f_j-1}} + cwy_{T_{j-1,f_j-1}}, F_{T_{j,f_j-2}}) + A_{T_{j,f_j-1}} && \text{(Form. 1)} \\ &= max(F_{T_{j-1,f_j-1}} + cwy_{T_{j-1,f_j-1}}, A_{S_{j,f_j-2}}) + A_{T_{j,f_j-1}} && (l_{j-1} > f_j - 2 \text{ when} r_b \leq -2) \\ &= max(A_{T_{j-1,f_j-1}} + cwy_{T_{j-1,f_j-1}}, A_{S_{j,f_j-2}}) + A_{T_{j,f_j-1}} && (A_{T_{j-2,f_j-1}} = A_{T_{j-1,f_j-2}} = 0 \text{ when } r_b \leq -2) \\ &= max(cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0) + A_{S_{j,f_j-2}} + A_{T_{j,f_j-1}} \\ &= max(cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0) + A_{S_{j,f_j-1}} \end{aligned} \quad (3)$$

Now we can calculate the finishing time for the first full tile $T_{j,k}$ in stack j using these two formulas.

$$\begin{aligned} F_{T_{j,f_j}} &= max(F_{T_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, F_{T_{j,f_j-1}}) + A_{T_{j,f_j}} && \text{(Form. 1)} \\ &= max(A_{S_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, F_{T_{j,f_j-1}}) + A_{T_{j,f_j}} && \text{(eqn. 2)} \\ &= max(A_{S_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, \\ &\qquad max(cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0) + A_{S_{j,f_j-1}}) + A_{T_{j,f_j}} && \text{(eqn. 3)} \\ &= max(cwy_{T_{j-1,f_j}} - (A_{S_{j,f_j-1}} - A_{S_{j-1,f_j}}), \\ &\qquad max(cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0)) + A_{S_{j,f_j-1}} + A_{T_{j,f_j}} \\ &= max(cwy_{T_{j-1,f_j}} - (A_{S_{j,f_j-1}} - A_{S_{j-1,f_j}}), cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0) + A_{S_{j,f_j}} \\ &= max(chw - (A_{S_{j,f_j-1}} - A_{S_{j-1,f_j}}), cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}}), 0) + A_{S_{j,f_j}} \\ &= max(D_1, D_2, 0) + A_{S_{j,f_j}} \end{aligned} \quad (4)$$

where $D_1 = chw - (A_{S_{j,f_j-1}} - A_{S_{j-1,f_j}})$ and $D_2 = cwy_{T_{j-1,f_j-1}} - (A_{S_{j,f_j-2}} - A_{T_{j-1,f_j-1}})$.

Now let us write out the formulas for D_1 and D_2. Using the notation in Figure 9a) and 9b) respectively we get the following expressions for $j \geq 2$.

$$\begin{aligned} D_1 &= chw - (A_2 + A_3 - A_1) \\ &= chw - (A_2) \\ &= chw - ((y+v)w - \tfrac{vx}{2}) \\ &= chw - ((y+v)w - \tfrac{vwv}{-2r_bh}) && (\tfrac{x}{w} = \tfrac{v}{-r_bh}) \\ &= chw - ((y + (-r_bh - z))w - \tfrac{w(-r_bh-z)^2}{-2r_bh}) && (v = -r_bh - z) \\ &= chw - ((y + (-r_bh - h - y))w - \tfrac{w(-r_bh-h-y)^2}{-2r_bh}) && (z = h + y) \\ &= chw - ((-r_bh - h)w + \tfrac{w(r_bh+h+y)^2}{2r_bh}) \\ &= chw - \tfrac{w}{2r_bh}((r_bh + h + y)^2 - 2r_b^2h^2 - 2r_bh^2) \end{aligned}$$

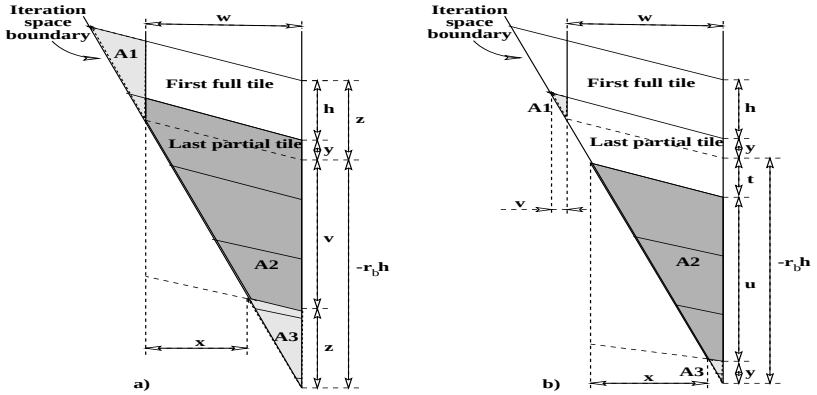

Figure 9: Iteration space for $r_b \leq -2$.

$$
\begin{aligned}
D_2 &= cwy_{T_{j-1,f_j-1}} - (A_2 - A_1) \\
&= cwy - (\tfrac{ux}{2} - \tfrac{yv}{2}) \\
&= cwy - (\tfrac{uwu}{-2r_b h} - \tfrac{yv}{2}) && (\tfrac{x}{w} = \tfrac{u}{-r_b h}) \\
&= cwy - (\tfrac{wu^2}{-2r_b h} - \tfrac{ywy}{-2r_b h}) && (\tfrac{v}{w} = \tfrac{y}{-r_b h}) \\
&= cwy - (\tfrac{wy^2}{2r_b h} - \tfrac{wu^2}{2r_b h}) \\
&= cwy - (\tfrac{wy^2}{2r_b h} - \tfrac{w(-r_b h - t)^2}{2r_b h}) && (u = -r_b h - t) \\
&= cwy - (\tfrac{wy^2}{2r_b h} - \tfrac{w(-r_b h - (h-y))^2}{2r_b h}) && (t = h - y) \\
&= cwy - (\tfrac{w}{2r_b h}(y^2 - (-r_b h - (h-y))^2)) \\
&= cwy - \tfrac{w}{2r_b h}(y^2 - (-r_b h - h + y)^2)
\end{aligned}
$$

We see that D_2 is a linear function of y, and since $0 \leq y \leq h$, we know that the maximum value of D_2 is $max(D_2[y=h], D_2[y=0])$. D_1 on the other hand is a parabola, but taking the second derivative with respect to y, we see it is concave and the maximum value of D_1 is therefore bounded by $max(D_1[y=h], D_1[y=0])$.

Going back to the expression for $F_{T_{j,f_j}}$ we have for $j \geq 2$

$$
\begin{aligned}
F_{T_{j,f_j}} &= max(D_1, D_2, 0) + A_{S_{j,f_j}} && \text{(eqn. 4)} \\
&\leq max(D_1[y=h], D_1[y=0], D_2[y=h], D_2[y=0], 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{w}{2r_b h}((r_b h + h + h)^2 - 2r_b^2 h^2 - 2r_b h^2), \\
&\quad chw - \tfrac{w}{2r_b h}((r_b h + h)^2 - 2r_b^2 h^2 - 2r_b h^2), \\
&\quad chw - \tfrac{w}{2r_b h}(h^2 - (-r_b h - h + h)^2), -\tfrac{w}{2r_b h}(-(-r_b h - h)^2), 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{hw}{2r_b}((r_b+2)^2 - 2r_b^2 - 2r_b), chw - \tfrac{hw}{2r_b}((r_b+1)^2 - 2r_b^2 - 2r_b), \\
&\quad chw - \tfrac{hw}{2r_b}(1 - r_b^2), \tfrac{hw}{2r_b}(r_b+1)^2, 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{hw}{2r_b}(r_b^2 + 4r_b + 4 - 2r_b^2 - 2r_b), chw - \tfrac{hw}{2r_b}(1 - r_b^2), \\
&\quad chw - \tfrac{(1-r_b^2)}{2r_b}hw, \tfrac{(r_b+1)^2}{2r_b}hw, 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{hw}{2r_b}(1 - r_b^2 + 2r_b + 3), chw - \tfrac{(1-r_b^2)}{2r_b}hw, \tfrac{(r_b+1)^2}{2r_b}hw, 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{1-r_b^2}{2r_b}hw - \tfrac{2r_b+3}{2r_b}hw, chw - \tfrac{1-r_b^2}{2r_b}hw, \tfrac{(r_b+1)^2}{2r_b}hw, 0) + A_{S_{j,f_j}} \\
&\leq max(chw - \tfrac{1-r_b^2}{2r_b}hw, \tfrac{(r_b+1)^2}{2r_b}hw, 0) + A_{S_{j,f_j}} && (-\tfrac{2r_b+3}{2r_b} < 0) \\
&\leq max(chw - \tfrac{1-r_b^2}{2r_b}hw, 0) + A_{S_{j,f_j}} && (\tfrac{(r_b+1)^2}{2r_b} < 0)
\end{aligned}
$$

End of Proof of Lemma 16 □

We are now ready to complete the proof of Lemma 2.

$$
\begin{aligned}
F_{T_{j,k}} &= F_{T_{j,k-1}} + A_{T_{j,k}} && (\text{for } k > f_j, j \geq 2) && \text{(eqn. 1)} \\
&= F_{T_{j,f_j}} + (k - 1 - f_j)hw + A_{T_{j,k}} && && \text{(by simple induction)} \\
&\leq max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + A_{S_{j,f_j}} + (k - 1 - f_j)hw + A_{T_{j,k}} && && \text{(Lemma 16)} \\
&\leq max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + A_{S_{j,k}}
\end{aligned}
$$

and we have now proven the lemma.

End of Proof of Lemma 2 □

Corollary 2

$$I^c_{p_j\,bottom} \leq max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) \quad (for\ j \geq 2)$$
when $r_b \leq -2$.

Proof of Corollary 2: The idle time for p_j in an infinite iteration space is given by the finishing time of a full tile $T_{j,k}$ in stack j, minus the amount of computation p_j has executed before $F_{T_{j,k}}$. In other words,

$$\begin{aligned} I^c_{p_j\,bottom} &= F_{T_{j,k}} - A_{S_{j,k}} \\ &\leq A_{S_{j,k}} + max((c + \tfrac{1-r_b^2}{2r_b})hw, 0) - A_{S_{j,k}} \quad \text{(for } j \geq 2\text{)} \quad \text{(Lemma 2)} \\ &\leq max((c + \tfrac{1-r_b^2}{2r_b})hw, 0) \end{aligned}$$

End of Proof of Corollary 2 □

Finite parallelogram-shaped iteration spaces

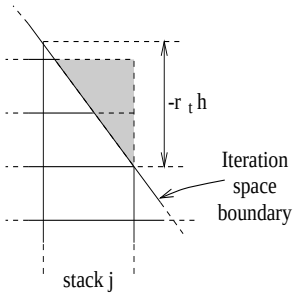

Figure 10: Iteration space for $r_t \leq -2$. Light shaded areas at each stack correspond to added idle time if that stack finishes last.

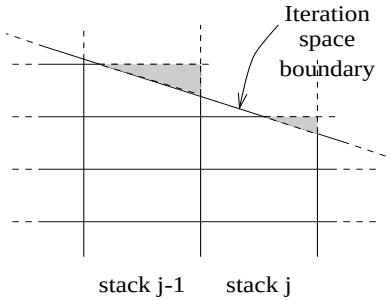

Figure 11: Upper right corner of iteration space with $r_t \geq -1$. The shaded area correspond to the computation that is removed from the first partial tile in each stack when making the infinite iteration space finite.

Lemma 3

$$I^c_l \begin{cases} \leq \frac{-r_t hw}{2} & (for\ r_t < -1) \\ = 0 & (for\ r_t \geq -1) \end{cases}$$

Proof of Lemma 3: If $r_t < -1$, the amount of time a processor can be forced to be idle even though it still has computation left to do, is less than or equal to the shaded area in Figure 10, j being an arbitrary stack. Since this area varies in size from processor to processor, the processors will not finish at the same time. The maximum amount of area that will be added to the execution time of a single processor[4] is equal to $\frac{-r_t hw}{2}$.

If on the other hand $r_t \geq -1$, the j-th stack will not have to wait for the $(j-1)$-th processor since the amount of computation removed from the j-th stack always will be smaller or equal to the amount of computation removed in the $(j-1)$-th stack. (See the shaded areas in Figure 11.) The $(j-1)$-th processor will therefore always have the data ready for p_j at the time p_j needs it, and $I^c_{p_j}$ will be equal to 0.

End of Proof of Lemma 3 □

[4]The processor that finishes last will not necessarily be the P-th processor.

Lemma 4

$$I^s_{p_j} = \begin{cases} hw(r_t - v\frac{w}{h})(P-j) & (for\ r_t \geq v\frac{w}{h}) \\ -hw(r_t - v\frac{w}{h})(j-1) & (for\ r_t \leq v\frac{w}{h}) \end{cases}$$

where r_t is the top rise and v is the wavefront constant.

Proof of Lemma 4: The amount of time p_j will be idle after it has finished all its computation is given by the following cases in the case of block distribution.

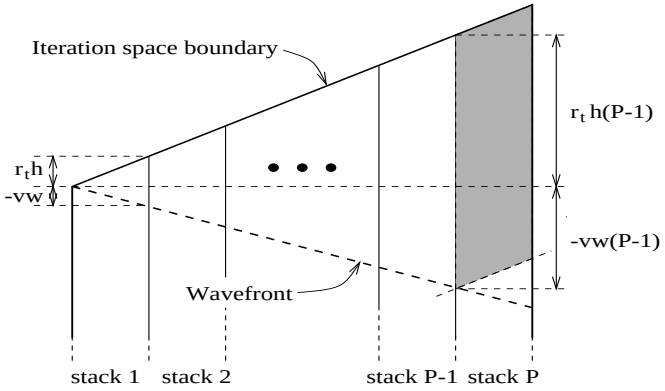

Figure 12: Top of iteration space. The shaded area correspond to the computation that p_1 is going to be idle after having executed its stack.

Case 1 ($v\frac{w}{h} \leq r_t$): After p_1 has finished executing, it will be idle during the time p_P executes the shaded area in Figure 12 (follows from the definition of wavefront), i.e. during $(-vw(P-1) + r_t h(P-1))w$ time. The j-th processor, will similarly be idle during $(-vw(P-j) + r_t h(P-j))w$ time.

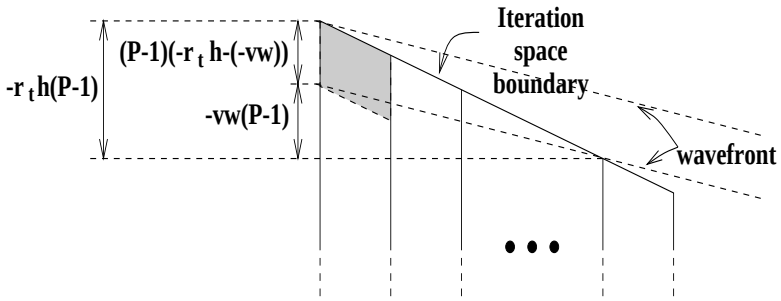

Figure 13: Top of iteration space. The shaded area correspond to the computation that p_P is going to be idle after having executed its stack.

Case 2 ($r_t \leq v\frac{w}{h}$): After p_P has finished executing, it will be idle during the time p_1 executes the shaded area in Figure 13 (follows from the definition of wavefront), i.e. during $(P-1)(-r_t h - (-vw))w$ time. The j-th processor, will similarly be idle during $(j-1)(-r_t h - (-vw))w$ time.

The additional idle time after p_j has finished computing its stack is thus equal to

$$\begin{aligned} (-vw(P-j) + r_t h(P-j))w &= hw(r_t - v\tfrac{w}{h})(P-j) \quad (\text{for } v\tfrac{w}{h} \leq r_t) \\ (j-1)(-r_t h - (-vw))w &= -hw(r_t - v\tfrac{w}{h})(j-1) \quad (\text{for } r_t \leq v\tfrac{w}{h}) \end{aligned}$$

In the case of block cyclic distribution we need to note that the only synchronization idle time for p_j occurs after p_j has executed all its stacks. Since the amount of time p_j needs to wait at this point will be the same as in the block distribution case, we have proven the lemma also for block cyclic distribution.

End of Proof of Lemma 4 □

Lemma 5 *Let $r \geq -1$ and assume that there exists a row k, such that for all stacks, there is a full tile in k. The total processor idle time for the processor executing the j-th stack is then given by the following formula.*

$$I_p = (P-1)hw(1+r+c)$$

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r is the rise.

Proof of Lemma 5:

$$\begin{aligned}
I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)}\\
&= (j-1)(1+r_b+c)hw + \textstyle\sum_{l=j \bmod P} I^c_{l\,top} + (P-j)(r_t - v\frac{w}{h})hw && \text{(by definition)}\\
&= (j-1)(1+r_b+c)hw + 0 + (P-j)(r_t - v\tfrac{w}{h})hw && \text{(Cor. 1, Lemma 3 and 4)}\\
&= (j-1)(1+r+c)hw + (P-j)(r-(-(1+c)))hw && (v\tfrac{w}{h} = -(1+c))\\
&= (j-1)(1+r+c)hw + (P-j)(1+r+c)hw\\
&= (P-1)(1+r+c)hw
\end{aligned}$$

Since $(P-1)(1+r+c)hw$ is independent on j, $I_p = I_{p_j} = (P-1)(1+r+c)hw$, and we have proven the lemma.
End of Proof of Lemma 5 □

Theorem 1 *Let $r \geq -1$. If there are P stacks distributed one per processor, and there exists a row which includes full tiles in every stack, then the total idle time will be*

$$I_a = P(P-1)hw(1+r+c)$$

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r is the rise.

Proof of Theorem 1: Follows directly from Lemma 5
End of Proof of Theorem 1 □

Lemma 6 *If $r \leq -2$ and each processor only executes one stack, then the following formula is an upper bound on the total idle time for one processor.*

$$I_p \leq max((c - \frac{1}{2r})hw, \frac{-rhw}{2})$$

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, and r is the rise.

Proof of Lemma 6:

$$\begin{aligned}
I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom} + I^c_{p_j\,top} - (j-1)(r_t - v\tfrac{w}{h})hw && \text{(Lemma 4)}\\
&= I^c_{p_j\,bottom} + I^c_{p_j\,top} - 0 && (v\tfrac{w}{h} = r_b = r_t)\\
&= I^c_{p_j\,bottom} + \textstyle\sum_{l=j \bmod P} I^c_{l\,top} && \text{(by definition)}\\
&\leq max((c - \tfrac{1-r_b}{2r_b})hw, 0) + \tfrac{-r_t hw}{2} && \text{(Cor. 2 and Lemma 3)}\\
&\leq max((c - \tfrac{1-r}{2r})hw, 0) + \tfrac{-rhw}{2}\\
&\leq max((c - \tfrac{1}{2r})hw, \tfrac{-rhw}{2})
\end{aligned}$$

Since $max((c - \frac{1}{2r})hw, \frac{-rhw}{2})$ is independent on j, $I_p = I_{p_j} = max((c - \frac{1}{2r})hw, \frac{-rhw}{2})$, and we have proven the lemma.
End of Proof of Lemma 6 □

Theorem 2 *If $r \leq -2$ and there are P stacks, then*

$$I_a \leq max((c - \frac{1}{2r})Phw, \frac{-rPhw}{2})$$

is an upper bound on the total idle time.

Proof of Theorem 2: Follows directly from Lemma 6.
End of Proof of Theorem 2 □

Lemma 7

Let $r_b \geq -1$ and p_j be the processor that executes stack numbers j mod P. Let all stacks have an area of at least $Phw(1+r+c)$, and let there exist a row in which all tiles are full.

$I_p = (P-1)(1+r+c)hw$

Proof of Lemma 7:

Let us first note that

$$\begin{aligned} I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)} \\ &= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)} \\ &= I^c_{p_j\,bottom} + \textstyle\sum l = j \bmod P\, I^c_{l\,top} + I^s_{p_j} && \text{(by definition)} \end{aligned}$$

$I^c_{l\,top}$ and $I^s_{p_j}$ are given by Lemma 3 and 4, but we need to derive a formula for $I^c_{p_j\,bottom}$ in the case of block cyclic distribution.

Let $I^c_{j\,bottom}$ be the amount of time the processor executing S_j would be idle before starting executing S_j if S_j was the first stack to be executed on that processor. (This amount of idle time is given by Corollary 1.)

Since S_{j-1} and S_j are executed in lock step and all stacks take the same amount of time to execute since they are all of the same area, S_{j+P-1} and S_{j+P} also execute in lock step. Data calculated in S_{j+P-1} will therefore not delay the execution of S_{j+P} any more than already accounted for in $I^c_{j+P_{bottom}}$.

An very course estimation of $I^c_{p_j\,bottom}$ would be to say it is the sum of all $I^c_{l\,bottom}$ where l is a stack executed by p_j. But if p_j can overlap $I^c_{l\,bottom}$ for some stack S_l with useful computation, then $I^c_{l\,bottom}$ will not be part of $I^c_{p_j\,bttom}$. Overlapping can occur if

$$I^c_{j+P_{bottom}} \leq F_{S_j}$$

where F_{S_j} is the finishing time of the j-th stack.

So if we can show that $I^c_{j+P_{bottom}} \leq F_{S_j}$ then we know that $I^c_{j+P_{bottom}}$ can be fully overlapped by the computation of S_j and will therefore not be added to $I^c_{p_j\,bottom}$. Inductive reasoning can then be used to show that $I^c_{p_j\,bottom}$ will infact only consist of $I^c_{j\,bottom}$.

$$\begin{aligned} F_{S_j} &= I^c_{j\,bottom} + I^c_{j\,top} + A_{S_j} && \text{(by definition)} \\ &= (j-1)(1+r+c)hw + 0 + A_{S_j} && \text{(Cor. 1 and Lemma 3)} \\ &\geq (j-1)(1+r+c)hw + Phw(1+r+c) && \text{(by assumption)} \\ &\geq (j+P-1)(1+r+c)hw \\ &\geq I^c_{j+P_{bottom}} && \text{(Cor. 1)} \end{aligned}$$

So we have

$$\begin{aligned} I^c_{p_j\,bottom} &= I^c_{j\,bottom} \\ &= (j-1)(1+r+c)hw \quad \text{(Cor. 1)} \end{aligned}$$

$$\begin{aligned} I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)} \\ &= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)} \\ &= I^c_{p_j\,bottom} + 0 + hw(r_t - v\tfrac{w}{h})(P-j) && \text{(Lemma 3 and 4)} \\ &= I^c_{p_j\,bottom} + hw(r-(-(1+c)))(P-j) && (v\tfrac{w}{h} = -(1+c)) \\ &= (j-1)(1+r+c)hw + hw(1+r+c)(P-j) && \text{(from above)} \\ &= (P-1)(1+r+c)hw \end{aligned}$$

and we have proven the lemma.

End of Proof of Lemma 7: □

Theorem 3 *If $r \geq -1$ and there are bP stacks, then the total idle time is given by*

$$I_a = P(P-1)(1+r+c)hw$$

Proof of Theorem 3: Follows directly from Lemma 7.

End of Proof of Theorem 3 □

Lemma 8

Let $r_b \leq -2$ and p_j be the processor that executes stack numbers $j, j+P, j+2P, ..., j+(b-1)P$.

$I_p \leq max((c - \frac{1}{2r})bhw, \frac{-rbhw}{2})$

Proof of Lemma 8: For p_j $I^c_{p_j\,top}$ will be $b\frac{-rhw}{2}$ (Lemma 3), since p_j will need to wait for communication at the top of all stacks. $I^s_{p_j}$ is by Lemma 4 equal to 0, since $r_t = r_b = v\frac{w}{h}$. $I^c_{p_j\,bottom}$ will be less or equal to $bmax((c-\frac{1-r_b}{2r_b})hw, 0)$ (Corollary 2).[5]

We therefore have

$$\begin{aligned}
I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)}\\
&\le bmax((c-\tfrac{1-r_b^2}{2r_b})hw, 0) + b\tfrac{-r_thw}{2} + 0 && \text{(Cor. 2, Lemma 3 and 4 } (v\tfrac{w}{h} = r_b = r_t)\\
&\le bmax((c-\tfrac{1-r^2}{2r})hw, 0) + b\tfrac{-rhw}{2} && (v\tfrac{w}{h} = r_b = r_t)\\
&\le max((c-\tfrac{1-r^2}{2r})bhw, 0) - \tfrac{rbhw}{2}\\
&\le max((c-\tfrac{1}{2r})bhw, \tfrac{-rbhw}{2})
\end{aligned}$$

End of Proof of Lemma 8 □

Theorem 4 *If $r \le -2$ and there are bP stacks, then the total idle time is given by*

$$I_a \le max((c-\frac{1}{2r})bPhw, \frac{-rbPhw}{2})$$

Proof of Theorem 4:

$$\begin{aligned}
I_a &\le \textstyle\sum_{j=1}^{P} max((c-\frac{1}{2r})hw, \frac{-rbhw}{2}) && \text{(Lemma 8)}\\
&\le Pmax((c-\tfrac{1}{2r})hw, \tfrac{-rbhw}{2})\\
&\le max((c-\tfrac{1}{2r})bPhw, \tfrac{-rbPhw}{2})
\end{aligned}$$

End of Proof of Theorem 4 □

Finite trapezoidal iteration spaces

Lemma 9 *Let $r_b \ge -1$ and assume that there exists a row k, such that for all stacks, there is a full tile in k.*
The total processor idle time for the processor executing the j-th stack is then given by the following formula.

$I_{p_j} = hw((j-1)(1+r_b+c) + (r_t+1+c)(P-j))$ *(for $-1 \le r_b < r_t$)*

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r is the rise.

Proof of Lemma 9:

$$\begin{aligned}
I_{p_j} &= I^c_{p_j} + I^s_{p_j} && \text{(by definition)}\\
I_{p_j} &= I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} && \text{(by definition)}\\
I_{p_j} &= I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(r_t - v\tfrac{w}{h})hw && \text{(Lemma. 4)}\\
I_{p_j} &= I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(1+r_t+c)hw && (v\tfrac{w}{h} = -(1+c))\\
I_{p_j} &= (j-1)(1+r_b+c)hw + I^c_{p_j\,top} + (P-j)(1+r_t+c)hw && \text{(Cor. 1)}\\
I_{p_j} &= (j-1)(1+r_b+c)hw + 0 + (P-j)(1+r_t+c)hw && \text{(Lemma 3)}\\
I_{p_j} &= (j-1)(1+r_b+c)hw + (P-j)(1+r_t+c)hw
\end{aligned}$$

End of Proof of Lemma 9 □

Theorem 5 *If there are P stacks distributed one per processor, and there exist a row which includes full tiles in every stack, then the total idle time will be*

$I_a = P(P-1)hw(1+c+\frac{r_t+r_b}{2})$ *(for $-1 \le r_b < r_t$)*

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r_b and r_t are the bottom and top rise.

Proof of Theorem 5: Let $-1 \le r_b < r_t$.

[5] This step is really only true for $j > 2$ since the first processor will not be delayed waiting for communication from an earlier processor, whereas the second processor will. But even though p_1 does not have to wait at the beginning of the execution of the first stack, it has to wait the same amount of time after the execution of its last stack. This means that we can, without losing accuracy in our calculations, assume that p_1 is in fact being idle at the beginning of the execution of the first stack.

$$
\begin{aligned}
I_a &= \textstyle\sum_{j=1}^{P} hw((j-1)(1+r_b+c)+(r_t+1+c)(P-j)) && \text{(Lemma 9)}\\
&= hw(1+r_b+c)\textstyle\sum_{j=1}^{P}(j-1)+hw(r_t+1+c)\sum_{j=1}^{P}(P-j)\\
&= hw(1+r_b+c)\tfrac{P(P-1)}{2}+hw(r_t+1+c)\tfrac{P(P-1)}{2}\\
&= hw(2+r_t+r_b+2c)\tfrac{P(P-1)}{2}\\
&= P(P-1)hw(1+c+\tfrac{r_t+r_b}{2})
\end{aligned}
$$

End of Proof of Theorem 5 □

Lemma 10 *If $r_b \leq -2$ and each processor only executes one stack, then the following formula is a upper bound on the total idle time for one processor.*

$I_{p_j} \leq max((c-\frac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)(P-j)$ *(for $r_b \leq -2 < -1 \leq r_t$)*

$I_{p_j} \leq max((c-\frac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)(P-j)-\frac{r_t hw}{2}$ *(for $r_b < r_t < -1$, $r_b \leq -2$)*

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, r_b is the rise at the bottom of the iteration space and r_t is the rise at the top.

Proof of Lemma 10:

For $r_b \leq -2 < -1 \leq r_t$:

$$
\begin{aligned}
I_{p_j} &= I^c_{p_j}+I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+(P-j)(r_t-v\tfrac{w}{h})hw && \text{(Lemma 4)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+(P-j)(r_t-r_b)hw && (v\tfrac{w}{h}=r_b)\\
&= I^c_{p_j\,bottom}+0+(P-j)(r_t-r_b)hw && \text{(Lemma 3)}\\
&\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+(P-j)(r_t-r_b)hw && \text{(Cor. 2)}
\end{aligned}
$$

For $r_b < r_t < -1$, $r_b \leq -2$:

$$
\begin{aligned}
I_{p_j} &= I^c_{p_j}+I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+I^s_{p_j} && \text{(by definition)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+(P-j)(r_t-v\tfrac{w}{h})hw && \text{(Lemma 4)}\\
&= I^c_{p_j\,bottom}+I^c_{p_j\,top}+(P-j)(r_t-r_b)hw && (v\tfrac{w}{h}=r_b)\\
&\leq I^c_{p_j\,bottom}+\tfrac{-r_t hw}{2}+(P-j)(r_t-r_b)hw && \text{(Lemma 3)}\\
&\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)-\tfrac{r_t hw}{2}+(P-j)(r_t-r_b)hw && \text{(Cor. 2)}
\end{aligned}
$$

End of Proof of Lemma 10 □

Theorem 6 *If $r_b \leq -2$, $r_b < r_t$ and there are P stacks, then*

$I_a \leq max(P(c-\frac{1-r_b^2}{2r_b})hw,0)+P(P-1)hw(\frac{r_t-r_b}{2})$ *(for $r_b \leq -2 < -1 \leq r_t$)*

$I_a \leq max(P(c-\frac{1-r_b^2}{2r_b})hw,0)+P(P-1)hw(\frac{r_t-r_b}{2})-\frac{r_t hwP}{2}$ *(for $r_b < r_t < -1, r_b \leq -2$)*

is an upper bound on the total idle time.

Proof of Theorem 6:

$$
\begin{aligned}
I_a &\leq \textstyle\sum_{j=1}^{P}(max((c-\frac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)(P-j)) && \text{(for } r_b \leq -2 < -1 \leq r_t\text{)}\\
& && \text{(Lemma 10)}\\
&\leq Pmax((c-\tfrac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)\textstyle\sum_{j=1}^{P}(P-j)\\
&\leq Pmax((c-\tfrac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)\tfrac{P(P-1)}{2}\\
&\leq max(P(c-\tfrac{1-r_b^2}{2r_b})hw,0)+P(P-1)hw(\tfrac{r_t-r_b}{2})\\
I_a &\leq \textstyle\sum_{j=1}^{P}(max((c-\frac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)(P-j)-\frac{r_t hw}{2}) && \text{(for } r_b < r_t < -1, r_b \leq -2\text{)}\\
& && \text{(Lemma 10)}\\
&\leq Pmax((c-\tfrac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)\textstyle\sum_{j=1}^{P}(P-j)-\frac{r_t hwP}{2})\\
&\leq max(P(c-\tfrac{1-r_b^2}{2r_b})hw,0)+hw(r_t-r_b)\tfrac{P(P-1)}{2}-\tfrac{r_t hwP}{2}\\
&\leq max(P(c-\tfrac{1-r_b^2}{2r_b})hw,0)+P(P-1)hw(\tfrac{r_t-r_b}{2})-\tfrac{r_t hwP}{2}
\end{aligned}
$$

End of Proof of Theorem 6 □

Lemma 11 *Let $r_b \geq -1$ and assume that there exists a row k, such that for all stacks, there is a full tile in k. The total processor idle time for the processor executing the j-th stack is then given by the following formula.*

$$\begin{array}{lcll} I_{p_j} & = & hw((j-1)(1+r_b+c)+(P-j)(1+r_t+c)) & \textit{(for } -1 \leq r_t < r_b\textit{)} \\ I_{p_j} & \leq & hw((j-1)(1+r_b+c)+(P-j)(1+r_t+c)-\frac{r_t}{2}) & \textit{(for } -(1+c) \leq r_t < -1 \leq r_b\textit{)} \\ I_{p_j} & \leq & -hw((j-1)(r_t-r_b)+\frac{r_t}{2}) & \textit{(for } r_t \leq -(1+c) < -1 \leq r_b\textit{)} \end{array}$$

where h and w are the height and width of a tile, chw is the time is takes to communicate one tile surface of height h, and r_b is the rise at the bottom of the iteration space and r_t is the rise at the top.

Proof of Lemma 11:

For $-1 \leq r_t < r_b$:

$$\begin{array}{lcll} I_{p_j} & = & I^c_{p_j} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(r_t - v\frac{w}{h})hw & \text{(Lemma 4)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(1+r_t+c)hw & (v\frac{w}{h} = -(1+c)) \\ & = & I^c_{p_j\,bottom} + 0 + (P-j)(1+r_t+c)hw & \text{(Lemma 3)} \\ & = & (j-1)(1+r_b+c)hw + (P-j)(1+r_t+c)hw & \text{(Cor. 1)} \end{array}$$

For $-(1+c) \leq r_t < -1 \leq r_b$:

$$\begin{array}{lcll} I_{p_j} & = & I^c_{p_j} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(r_t - v\frac{w}{h})hw & \text{(Lemma 4)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + (P-j)(1+r_t+c)hw & (v\frac{w}{h} = -(1+c)) \\ & \leq & I^c_{p_j\,bottom} + \frac{-r_t hw}{2} + (P-j)(1+r_t+c)hw & \text{(Lemma 3)} \\ & \leq & (j-1)(1+r_b+c)hw - \frac{r_t hw}{2} + (P-j)(1+r_t+c)hw & \text{(Cor. 1)} \end{array}$$

For $r_t \leq -(1+c) < -1 \leq r_b$:

$$\begin{array}{lcll} I_{p_j} & = & I^c_{p_j} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} + I^s_{p_j} & \text{(by definition)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} - (j-1)(r_t - v\frac{w}{h})hw & \text{(Lemma 4)} \\ & = & I^c_{p_j\,bottom} + I^c_{p_j\,top} - (j-1)(1+r_t+c)hw & (v\frac{w}{h} = -(1+c)) \\ & \leq & I^c_{p_j\,bottom} + \frac{-r_t hw}{2} - (j-1)(1+r_t+c)hw & \text{(Lemma 3)} \\ & \leq & (j-1)(1+r_b+c)hw - \frac{r_t hw}{2} - (j-1)(1+r_t+c)hw & \text{(Cor. 1)} \\ & \leq & -(j-1)\frac{r_t - r_b}{2}hw - \frac{r_t hw}{2} & \end{array}$$

End of Proof of Lemma 11 □

Theorem 7 *Let $r_b \geq -1$. If there are P stacks distributed one per processor, and there exist a row which includes full tiles in every stack, then the total idle time will be*

$$\begin{array}{lcll} I_a & = & P(P-1)hw(1+c+\frac{r_t+r_b}{2}) & \textit{(for } -1 \leq r_t < r_b\textit{)} \\ I_a & \leq & P(P-1)hw(1+c+\frac{r_t+r_b}{2}) - \frac{r_t Phw}{2} & \textit{(for } -(1+c) \leq r_t < -1 \leq r_b\textit{)} \\ I_a & \leq & -P(P-1)hw(\frac{r_t-r_b}{2}) - \frac{r_t Phw}{2} & \textit{(for } r_t \leq -(1+c) < -1 \leq r_b\textit{)} \end{array}$$

where h is the height and w the width of the tiles, chw is the communication time for one tile surface of length h, and r_b and r_t are the bottom and top rise.

Proof of Theorem 7:

$$
\begin{array}{rcll}
I_a & = & \sum_{j=1}^{P} (hw((j-1)(1+r_b+c)+(P-j)(1+r_t+c))) & (\text{for } -1 \le r_t < r_b) \quad (\text{Lemma 11}) \\
 & = & hw(1+r_b+c)\sum_{j=1}^{P}(j-1)+ & \\
 & & +hw(1+r_t+c)\sum_{j=1}^{P}(P-j) & \\
 & = & hw(1+r_b+c)\frac{P(P-1)}{2}+hw(1+r_t+c)\frac{P(P-1)}{2} & \\
 & = & P(P-1)hw(1+c+\frac{r_t+r_b}{2}) & \\
I_a & \le & \sum_{j=1}^{P} (hw((j-1)(1+r_b+c)+(P-j)(1+r_t+c)-\frac{r_t}{2})) & (\text{for } -(1+c) \le r_t < -1 \le r_b) \quad (\text{Lemma 11}) \\
 & \le & hw(1+r_b+c)\sum_{j=1}^{P}(j-1)+ & \\
 & & +hw(1+r_t+c)\sum_{j=1}^{P}(P-j)-\frac{Pr_thw}{2} & \\
 & \le & hw(1+r_b+c)\frac{P(P-1)}{2}+hw(1+r_t+c)\frac{P(P-1)}{2}-\frac{Pr_thw}{2} & \\
 & \le & P(P-1)hw(1+c+\frac{r_t+r_b}{2})-\frac{r_tPhw}{2} & \\
I_a & \le & \sum_{j=1}^{P} (-hw((j-1)(r_t-r_b)+\frac{r_t}{2})) & (\text{for } r_t \le -(1+c) < -1 \le r_b) \quad (\text{Lemma 11}) \\
 & \le & -hw(r_t-r_b)\sum_{j=1}^{P}(j-1)-\frac{Pr_thw}{2} & \\
 & \le & -hw(r_t-r_b)\frac{P(P-1)}{2}-\frac{Pr_thw}{2} & \\
 & \le & -P(P-1)hw(\frac{r_t-r_b}{2})-\frac{Pr_thw}{2} &
\end{array}
$$

End of Proof of Theorem 7 □

Lemma 12 *If $r_t < r_b \le -2$ and each processor only executes one stack, then the following formula is an upper bound on the total idle time for one processor.*

$$I_{p_j} \le max((c-\frac{1-r_b^2}{2r_b})hw,0)-hw(r_t-r_b)(j-1)-\frac{r_thw}{2}$$

where w is the width and h the height of the tile, chw is the time it takes to communicate one tile surface of length h, and r is the rise.

Proof of Lemma 12:

$$
\begin{array}{rcll}
I_{p_j} & = & I^c_{p_j}+I^s_{p_j} & (\text{by definition}) \\
 & = & I^c_{p_j\,bottom}+I^c_{p_j\,top}+I^s_{p_j} & (\text{by definition}) \\
 & = & I^c_{p_j\,bottom}+I^c_{p_j\,top}-(j-1)(r_t-v\frac{w}{h})hw & (\text{Lemma 4}) \\
 & = & I^c_{p_j\,bottom}+I^c_{p_j\,top}-(j-1)(r_t-r_b)hw & (v\frac{w}{h}=r_b) \\
 & \le & I^c_{p_j\,bottom}+\frac{-r_thw}{2}-(j-1)(r_t-r_b)hw & (\text{Lemma 3}) \\
 & \le & max((c-\frac{1-r_b^2}{2r_b})hw,0)-\frac{r_thw}{2}-(j-1)(r_t-r_b)hw & (\text{Cor. 2})
\end{array}
$$

End of Proof of Lemma 12 □

Theorem 8 *If $r_t < r_b \le -2$ and there are P stacks, then*

$$I_a \le max(P(c-\frac{1-r_b^2}{2r_b})hw,0)-P(P-1)hw(\frac{r_t-r_b}{2})-\frac{Pr_thw}{2}$$

is a tight upper bound on the total idle time.

Proof of Theorem 8:

$$
\begin{array}{rcll}
I_a & \le & \sum_{j=1}^{P}(max((c-\frac{1-r_b^2}{2r_b})hw,0)-hw(r_t-r_b)(j-1)-\frac{r_thw}{2}) & (\text{Lemma 12}) \\
 & \le & Pmax((c-\frac{1-r_b^2}{2r_b})hw,0)-hw(r_t-r_b)\sum_{j=1}^{P}(j-1)-\frac{Pr_thw}{2} & \\
 & \le & max(P(c-\frac{1-r_b^2}{2r_b})hw,0)-hw(r_t-r_b)\frac{P(P-1)}{2}-\frac{Pr_thw}{2} & \\
 & \le & max(P(c-\frac{1-r_b^2}{2r_b})hw,0)-P(P-1)hw(\frac{r_t-r_b}{2})-\frac{Pr_thw}{2} &
\end{array}
$$

End of Proof of Theorem 8 □

New proofs

Lemma 17 is to replace Lemma 2. Changes need to be propagated to all places where Lemma 2 is directly or indirectly used, namely in Corollary 2, Lemma 6, 8, 10, 12, Theorem 2, 4, 6 and 8.

Lemma 17 *When $r_b \leq -2$, $j \geq 2$ and $k \geq f_2$, then*

$$F_{T_{j,k}} \leq A_{S_{j,k}} + max((c + \frac{1-r_b^2}{2r_b})hw, 0) + max((j-2)(1 + r_b + c)hw, 0)$$

Proof of Lemma 17: We first need to prove a couple of lemmas (18, 19, 20 and 21). Each lemma determines the finishing time for tiles in each of the shaded areas in Figure 14.

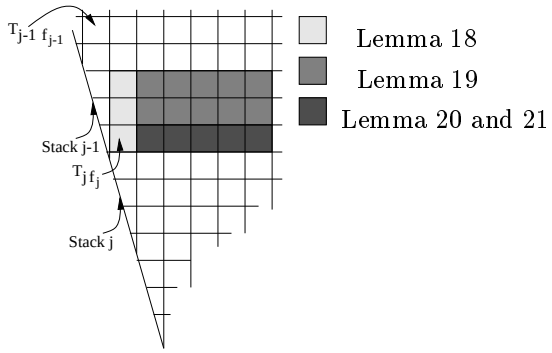

Figure 14: The tiles in the shaded areas are the tiles for which the corresponding lemmas determine the finishing time.

Lemma 18 *Let f_j be the value of k such that $T_{j,k}$ is the first full tile in stack j.*

$$F_{T_{j,k}} = F_{T_{j,f_j}} + (k - f_j)hw \quad \textit{(for } j \geq 1, f_j \leq k < f_{j-1} - 1\textit{)}$$

Proof of Lemma 18:
Induction Hypothesis:

$$F_{T_{j,k}} = F_{T_{j,f_j}} + (k - f_j)hw \quad \text{(for } j \geq 1, f_j \leq k < f_{j-1} - 1\text{)}$$

Base Case ($k = f_j$)**:** Trivially true.
Induction Step: Suppose the induction hypothesis holds for k, show it also holds for $k+1$.

$$\begin{aligned}
F_{T_{j,k+1}} &= max(F_{T_{j-1,k+1}} + cwy_{T_{j-1,k+1}}, F_{T_{j,k}}) + A_{T_{j,k+1}} && \text{(Form. 1)} \\
&= max(F_{T_{j-1,k+1}} + chw, F_{T_{j,k}}) + hw && (A_{T_{j,k+1}} = hw \text{ when } k+1 \geq f_j) \\
&= max(max(F_{T_{j-2,k+1}} + cwy_{T_{j-2,k+1}}, F_{T_{j-1,k}}) + A_{T_{j-1,k+1}} + chw, \\
&\quad F_{T_{j,k}}) + hw && \text{(Form. 1)} \\
&= max(F_{T_{j-1,k}} + A_{T_{j-1,k+1}} + chw, F_{T_{j,k}}) + hw && (A_{T_{j-2,k+1}} = 0 \text{ when } k+1 < f_{j-1}) \\
&= max(F_{T_{j-1,k}} + A_{T_{j-1,k+1}} + chw, F_{T_{j,f_j}} + (k - f_j)hw) + hw && \text{(Induction hypothesis)}
\end{aligned}$$

We know that $F_{T_{j-1,k}} < F_{T_{j-1,f_j}} + (k - f_j)hw$ since the j-th stack consists of partial tiles between row f_j and f_{j-1} and $k < f_{j-1} - 1$. We also know that $F_{T_{j-1,f_j}} + chw + hw < F_{T_{j,f_j}}$ from Formula 1. So we have that $F_{T_{j-1,k}} < F_{T_{j,f_j}} + (k - f_j)hw - hw - chw$ and therefore $F_{T_{j-1,k}} + A_{T_{j-1,k+1}} + chw < F_{T_{j,f_j}} + (k - f_j)hw$. This is used to complete the proof.

$$\begin{aligned}
F_{T_{j,k}} &= max(F_{T_{j-1,k}} + A_{T_{j-1,k+1}} + chw, F_{T_{j,f_j}} + (k - f_j)hw) + hw \\
&= F_{T_{j,f_j}} + (k - f_j)hw + hw && \text{(Reasoning above)} \\
&= F_{T_{j,f_j}} + (k + 1 - f_j)hw
\end{aligned}$$

End of Proof of Lemma 18 □

Lemma 19 *Let f_j be the value of k such that $T_{j,k}$ is the first full tile in stack j.*

$$F_{T_{l,k}} = F_{T_{l,f_j}} + (k - f_j)hw \quad \textit{(for } j \geq 1, l \geq j, f_j \leq k < f_{j-1} - 1\textit{)}$$

Proof of Lemma 19:
Induction Hypothesis:

$$F_{T_{l,k}} = F_{T_{l,f_j}} + (k - f_j)hw \quad \text{(for } j \geq 1, l \geq j, f_j \leq k < f_{j-1} - 1\text{)}$$

Base Case ($l = j$)**:**

$$F_{T_{j,k}} = F_{T_{j,f_j}} + (k - f_j)hw \quad \text{(Lemma 18)}$$

Induction Step: Suppose the induction hypothesis holds for l, show it also holds for $l+1$.
Induction Hypothesis:

$F_{T_{l+1,k}} = F_{T_{l+1,f_j}} + (k - f_j)hw$ (for $j \geq 1, l+1 \geq j, f_j \leq k < f_{j-1} - 1$)

Base Case ($k = f_j$)**:** Trivially true.
Induction Step: Suppose the induction hypothesis holds for k, show it also holds for $k+1$.

$$\begin{aligned}
F_{T_{l+1,k+1}} &= max(F_{T_{l,k+1}} + cwy_{T_{l,k+1}}, \\
&\quad F_{T_{l+1,k}}) + A_{T_{l+1,k+1}} && \text{(Form. 1)} \\
&= max(F_{T_{l,k+1}} + chw, \\
&\quad F_{T_{l+1,k}}) + hw && (A_{T_{l+1,k+1}} = hw) \\
&= max(F_{T_{l,f_j}} + (k+1-f_j)hw + chw, \\
&\quad F_{T_{l+1,k}}) + hw && \text{(1st induction hypothesis)} \\
&= max(F_{T_{l,f_j}} + (k+1-f_j)hw + chw, \\
&\quad F_{T_{l+1,f_j}} + (k-f_j)hw) + hw && \text{(2nd induction hypothesis)} \\
&= max(F_{T_{l,f_j}} + (k+1-f_j)hw + hw + chw, \\
&\quad F_{T_{l+1,f_j}} + (k+1-f_j)hw) \\
&= F_{T_{l+1,f_j}} + (k+1-f_j)hw && \text{(Form. 1)}
\end{aligned}$$

End of Proof of Lemma 19 □

Lemma 20 *Let f_j be the value of k such that $T_{j,k}$ is the first full tile in stack j.*

$F_{T_{l,f_j}} = max(F_{T_{j,f_j}} + (l-j)(c+1)hw, F_{T_{l,f_{j+1}}} + (f_j - f_{j+1})hw)$ *(for $j \geq 1, l > j$)*

Proof of Lemma 20:
Induction Hypothesis:

$F_{T_{l,f_j}} = max(F_{T_{j,f_j}} + (l-j)(c+1)hw, F_{T_{l,f_{j+1}}} + (f_j - f_{j+1})hw)$ (for $j \geq 1, l > j$)

Base Case ($l = j+1$)**:**

$$\begin{aligned}
F_{T_{j+1,f_j}} &= max(F_{T_{j,f_j}} + cwy_{T_{j,f_j}}, F_{T_{j+1,f_j-1}}) + A_{T_{j+1,f_j}} && \text{(Form. 1)} \\
&= max(F_{T_{j,f_j}} + chw, F_{T_{j+1,f_j-1}}) + hw && (A_{T_{j+1,f_j}} = hw) \\
&= max(max(F_{T_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, F_{T_{j,f_j-1}}) + A_{T_{j,f_j}} + chw, \\
&\quad F_{T_{j+1,f_j-1}}) + hw && \text{(Form. 1)} \\
&= max(max(F_{T_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, F_{T_{j,f_j-1}}) + A_{T_{j,f_j}} + chw, \\
&\qquad max(F_{T_{j,f_j-1}} + cwy_{T_{j,f_j-1}}, F_{T_{j+1,f_j-2}}) + A_{T_{j+1,f_j-1}}) + hw && \text{(Form. 1)} \\
&= max(max(F_{T_{j-1,f_j}} + cwy_{T_{j-1,f_j}}, F_{T_{j,f_j-1}}) + A_{T_{j,f_j}} + chw, \\
&\qquad F_{T_{j+1,f_j-2}} + hw) + hw && (A_{T_{j+1,f_j-1}} = A_{T_{j,f_j}} = hw) \\
&= max(F_{T_{j,f_j}} + chw, F_{T_{j+1,f_j-2}} + hw) + hw && \text{(Form. 1)} \\
&= max(F_{T_{j,f_j}} + chw, F_{T_{j+1,f_{j+1}}} + (f_j - 2 - f_{j+1})hw + hw) + hw && \text{(Lemma 18)} \\
&= max(F_{T_{j,f_j}} + (j+1-j)(c+1)hw, F_{T_{j+1,f_{j+1}}} + (f_j - f_{j+1})hw)
\end{aligned}$$

Induction Step: Suppose the induction hypothesis holds for l, show it also holds for $l+1$.

$$\begin{aligned}
F_{T_{l+1,f_j}} &= max(F_{T_{l,f_j}} + cwy_{T_{l,f_j}}, F_{T_{l+1,f_j-1}}) + A_{T_{l+1,f_j}} && \text{(Form. 1)} \\
&= max(F_{T_{l,f_j}} + cwy_{T_{l,f_j}}, \\
&\qquad max(F_{T_{l,f_j-1}} + cwy_{T_{l,f_j-1}}, F_{T_{l+1,f_j-2}}) + A_{T_{l+1,f_j-1}}) + A_{T_{l+1,f_j}} && \text{(Form. 1)} \\
&= max(max(F_{T_{l-1,f_j}} + cwy_{T_{l-1,f_j}}, F_{T_{l,f_j-1}}) + A_{T_{l,f_j}} + cwy_{T_{l,f_j}}, \\
&\qquad max(F_{T_{l,f_j-1}} + cwy_{T_{l,f_j-1}}, F_{T_{l+1,f_j-2}}) + A_{T_{l+1,f_j-1}}) + A_{T_{l+1,f_j}} && \text{(Form. 1)} \\
&= max(max(F_{T_{l-1,f_j}} + cwy_{T_{l-1,f_j}}, F_{T_{l,f_j-1}}) + A_{T_{l,f_j}} + cwy_{T_{l,f_j}}, \\
&\qquad F_{T_{l+1,f_j-2}} + A_{T_{l+1,f_j-1}}) + A_{T_{l+1,f_j}} && (A_{T_{l,f_j}} = A_{T_{l+1,f_j-1}} = hw) \\
&= max(F_{T_{l,f_j}} + cwy_{T_{l,f_j}}, F_{T_{l+1,f_j-2}} + A_{T_{l+1,f_j-1}}) + A_{T_{l+1,f_j}} && \text{(Form. 1)} \\
&= max(F_{T_{l,f_j}} + chw, F_{T_{l+1,f_j-2}} + hw) + hw && (A_{T_{l+1,f_j}} = A_{T_{l+1,f_j-1}} = hw) \\
&= max(max(F_{T_{j,f_j}} + (l-j)(c+1)hw, F_{T_{l,f_{j+1}}} + (f_j - f_{j+1})hw) \\
&\qquad +chw, F_{T_{l+1,f_j-2}} + hw) + hw && \text{(Induction hypothesis)} \\
&= max(max(F_{T_{j,f_j}} + (l-j)(c+1)hw, F_{T_{l,f_{j+1}}} + (f_j - f_{j+1})hw) \\
&\qquad +chw, F_{T_{l+1,f_{j+1}}} + (f_j - 2 - f_{j+1})hw + hw) + hw && \text{(Lemma 19)} \\
&= max(F_{T_{j,f_j}} + (l+1-j)(c+1)hw, F_{T_{l,f_{j+1}}} + (f_j - f_{j+1})hw \\
&\qquad +(c+1)hw, F_{T_{l+1,f_{j+1}}} + (f_j - f_{j+1})hw) \\
&= max(F_{T_{j,f_j}} + (l+1-j)(c+1)hw, F_{T_{l+1,f_{j+1}}} + (f_j - f_{j+1})hw) && \text{(Form. 1)}
\end{aligned}$$

End of Proof of Lemma 20 □

Lemma 21 *Let f_j be the value of k such that $T_{j,k}$ is the first full tile in stack j.*

$$F_{T_{l,f_j}} \leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(l-j)hw,0)+A_{S_{l,f_j}} \quad \textit{(for } j\geq 1, l\geq j\textit{)}$$

Proof of Lemma 21:
Induction Hypothesis:

$$F_{T_{l,f_j}} \leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(l-j)hw,0)+A_{S_{l,f_j}} \quad (\text{for } j\geq 1, l\geq j)$$

Base Case ($j = J$ where J is the last stack): When $j = J$ we also know $l = j = J$, since $l \geq j$ and J is the maximum value of both l and j.

$$\begin{aligned} F_{T_{J,f_J}} &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{J,f_J}} && \text{(Lemma 16)}\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(J-J)hw,0)+A_{S_{J,f_J}} \end{aligned}$$

Induction Step: Suppose the induction hypothesis holds for $j+1$ show it also holds for j.
Case $l = j$:

$$\begin{aligned} F_{T_{j,f_j}} &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{j,f_j}} && \text{(Lemma 16)}\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(j-j)hw,0)+A_{S_{j,f_j}} \end{aligned}$$

Case $l > j$:

$$\begin{aligned} F_{T_{l,f_j}} &= max(F_{T_{j,f_j}}+(l-j)(c+1)hw, F_{T_{l,f_{j+1}}}+(f_j-f_{j+1})hw) && \text{(Lemma 20)}\\ &\leq max(max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{j,f_j}}+(l-j)(c+1)hw,\\ &\quad F_{T_{l,f_{j+1}}}+(f_j-f_{j+1})hw) && \text{(Lemma 16)}\\ &\leq max(max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+(l-j)(c+1)hw+A_{S_{l,f_j}}\\ &\quad +(l-j)r_bhw, F_{T_{l,f_{j+1}}}+(f_j-f_{j+1})hw) && (A_{S_{j,f_j}}-A_{S_{l,f_j}}=(l-j)r_bhw)\\ &\leq max(max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+(l-j)(c+1)hw+A_{S_{l,f_j}}\\ &\quad +(l-j)r_bhw, max((c-\tfrac{1-r_b^2}{2r_b})hw,0)\\ &\quad +max((1+r_b+c)(l-j-1)hw,0)+A_{S_{l,f_{j+1}}}+(f_j-f_{j+1})hw) && \text{(Induction hypothesis)}\\ &\leq max(max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+(l-j)(1+r_b+c)hw,\\ &\quad max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(l-j-1)hw,0)\\ &\quad -(f_j-f_{j+1})hw+(f_j-f_{j+1})hw)+A_{S_{l,f_j}} && (A_{S_{l,f_j}}-A_{S_{l,f_{j+1}}}=(f_j-f_{j+1})hw)\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{l,f_j}}\\ &\quad +max((l-j)(1+r_b+c)hw,(1+r_b+c)(l-j-1)hw,0)\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{l,f_j}}+max((l-j)(1+r_b+c)hw,0) && (l-j-1\geq 0) \end{aligned}$$

End of Proof of Lemma 21 □

Now we can continue to prove Lemma 17, using a doubly nested induction proof.
Induction Hypothesis:

$$F_{T_{j,k}} \leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(j-2)hw,0)+A_{S_{j,k}} \quad (\text{for } j\geq 2, k\geq f_2)$$

Base Case ($k = f_2$):

$$F_{T_{j,f_2}} \leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{j,f_2}}+max((j-2)(1+r_b+c)hw,0) \quad \text{(Lemma 21)}$$

Induction Step: Suppose the induction hypothesis holds for $k-1$, show it also holds for k.
Induction Hypothesis:

$$F_{T_{j,k}} \leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((1+r_b+c)(j-2)hw,0)+A_{S_{j,k}} \quad (\text{for } j\geq 2, k> f_2)$$

Base Case ($j = 2$):

$$\begin{aligned} F_{T_{2,k}} &= max(F_{T_{1,k}}+chw, F_{T_{2,k-1}})+hw && \text{(Form. 1)}\\ &= max(A_{S_{1,k}}+(c+1)hw, F_{T_{2,k-1}}+hw)\\ &\leq max(A_{S_{1,k}}+(c+1)hw, max((c-\tfrac{1-r_b^2}{2r_b})hw,0)\\ &\quad +max((1+r_b+c)(2-2)hw,0)+A_{S_{2,k-1}}+hw) && \text{(Induction hypothesis)}\\ &\leq max(A_{S_{1,k}}+(c+1)hw, max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{2,k}}) && (A_{S_{2,k}}=A_{S_{2,k-1}}+hw)\\ &\leq max((1+r_b+c)hw, max((c-\tfrac{1-r_b^2}{2r_b})hw,0))+A_{S_{2,k}} && (A_{S_{1,k}}-A_{S_{2,k}}=r_bhw)\\ &\leq max((1+r_b+c)hw,(c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{2,k}}\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+A_{S_{2,k}} && (1+r_b+c\leq c-\tfrac{1-r_b^2}{2r_b})\\ &\leq max((c-\tfrac{1-r_b^2}{2r_b})hw,0)+max((2-2)(1+r_b+c)hw,0)+A_{S_{2,k}} \end{aligned}$$

Induction Step: Suppose the induction hypothesis holds for $j-1$, show it also holds for j.

$$
\begin{aligned}
F_{T_{j,k}} &= max(F_{T_{j-1,k}} + chw, F_{T_{j,k-1}}) + hw && \text{(Form. 1)} \\
&\leq max(max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-2-1)hw, 0) \\
&\quad + A_{S_{j-1,k}} + chw, F_{T_{j,k-1}}) + hw && \text{(2nd induction hypothesis)} \\
&\leq max(max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-2-1)hw, 0) \\
&\quad + A_{S_{j-1,k}} + chw, max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) \\
&\quad + max((1 + r_b + c)(j-2)hw, 0) + A_{S_{j,k-1}}) + hw && \text{(1st induction hypothesis)} \\
&\leq max(max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-3)hw, 0) \\
&\quad + A_{S_{j-1,k}} + (c+1)hw, max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) \\
&\quad + max((1 + r_b + c)(j-2)hw, 0) + A_{S_{j,k}}) \\
&\leq max(max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-3)hw, 0) \\
&\quad + (1 + r_b + c)hw, max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) \\
&\quad + max((1 + r_b + c)(j-2)hw, 0)) + A_{S_{j,k}} && (A_{S_{j-1,k}} - A_{S_{j,k}} = r_b hw) \\
&\leq max(max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-2)hw, \\
&\quad (1 + r_b + c)hw), max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) \\
&\quad + max((1 + r_b + c)(j-2)hw, 0)) + A_{S_{j,k}} \\
&\leq max((c - \tfrac{1-r_b^2}{2r_b})hw, 0) + max((1 + r_b + c)(j-2)hw, 0) + A_{S_{j,k}}
\end{aligned}
$$

End of Proof of Lemma 17 □