International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 3, Number 1, pages 75–97, January 2013

Pipelined Execution of Windowed Image Computations

Ramachandran Vaidyanathan
School of Electrical Engineering and Computer Science
Louisiana State University, Baton Rouge, LA 70803, USA

Phaneendra Vinukonda
Tata Consultancy Services
Microsoft Corporation, Redmond, WA, USA

and

Alyssa C. Lessing
School of Electrical Engineering and Computer Science
Louisiana State University, Baton Rouge, LA 70803, USA

Received: July 25, 2012
Revised: October 26, 2012
Accepted: December 4, 2012
Communicated by Akihiro Fujiwara

Abstract

Many image processing operations manipulate an individual pixel using the values of other pixels in the neighborhood. Such operations are called windowed operations. The size of the windowed operation is a measure of the size of the given pixel's neighborhood. A windowed computation applies a windowed operation on all pixels of the image. An image processing application is typically a sequence of windowed computations. While windowed computations admit high parallelism, the cost of inputting and outputting the image often restricts the computation to a few computational units.

In this paper we analytically study the running of a sequence of z windowed computations, each of size w, on a z-stage pipelined computational model. For an $N \times N$ image and $n \times n$ input/output bandwidth per stage, we show that the sequence of windowed computations can be run in at most $\frac{N^2}{n^2}(1+\delta)$ steps, where $\delta = \left(\frac{n}{N} + \frac{6n^3}{wN^2} + \frac{zw}{N} + \frac{zn^2}{N^2}\right)$. This produces a speed-up of $\frac{z}{1+\delta}$ over a single stage; δ, the overhead is quite small. We also show that the memory requirement per stage is $O(wN + n^2)$. With values of N, n and w that reflect the current state-of-the-art, over 20 pipeline stages can be sustained with less than 5% overhead for a 10M-pixel image. Each of these stages would require less than 128 Kbytes of storage.

Keywords: image processing, pipelining, windowed computation

1 Introduction

Image processing algorithms are used in a wide variety of applications areas ranging from meteorology, military (satellite information analysis), security (face recognition, baggage scan) to gaming, videos and photography [4]. Some of these applications, particularly security applications, also require a real time response.

Many operations in image processing manipulate an image pixel using the values of other pixels around it. For example, one could use the average of values around each pixel to blur the image. In general, a rectangular window that extends to a width of w around the pixel is used to define the neighborhood of interest and to perform an operation on the pixel. When extended to all pixels of the image, we will call it a windowed computation of size w. Examples of such windowed computations include filtering techniques, and morphological operations [4, 9].

Many image processing applications include a series of windowed computations, that lend themselves well to running on a pipelined environment. For example, the scale invariant feature transform (SIFT) [7], one of the most widely used techniques for image feature extraction, has several stages of Gaussian filtering and a stage of descriptor generation, all of which are windowed computations. Generally speaking, image processing algorithms have large data sizes, admit high parallelism and require limited data sharing across individual computations (captured by the overlap of windows of nearby pixels). Notwithstanding this potential for a parallel (and pipelined) execution of these algorithms, the high data bandwidth demanded by a large image restricts the computation to remain within a single chip or module. This adversely impacts performance and cost, particularly in real-time applications.

In this paper we will model an image processing algorithm as a series of z windowed computations, each of size w on an $N \times N$ image. A z-stage pipeline will be used to run the algorithm, each stage dedicated to a windowed computation. Note that a set of relatively simple windowed computations could be clubbed together into a single windowed computation running on a stage. To capture the input/output bandwidth restriction between stages we will use a parameter $n \ll N$; only an $n \times n$ "tile" can be moved between stages at a time. As we show later, the data bandwidth requirement generally reduces for windowed computations as we move down stages of the pipeline. Thus, n^2 can be viewed as the the unit of input bandwidth.

An ideal pipeline should have matched input, output and computational speeds. Therefore, the quantity n^2 is also used as a measure of a unit-time computation by a stage. In practice, one could achieve this balance by selectively speeding up bottlenecked stages or slowing down (or even hibernating) stages that are unnecessarily fast.

The scheduling of a pipelined computation is a very well studied problem (see Benoit *et al.* [3] for a comprehensive survey). However, the particular case of windowed computation has not received much attention in an analytical framework. On the other hand there is a lot of work on implementing image and windowed computations on particular platforms (notably FPGAs and GPUs) [1, 2, 5, 8, 11, 12]. To our knowledge, ours is the first work to analytically study windowed computations and their execution on a pipelined platform of computation.

The main result of this paper is to show that the z windowed computations can be run on the pipeline in about $\frac{N^2}{n^2}(1+\delta)$ time units, where $\delta = \left(\frac{n}{N} + \frac{6n^3}{wN^2} + \frac{zw}{N} + \frac{zn^2}{N^2}\right) = A + Bz$ (say). With n^2 operations per unit time, the entire computation takes $\frac{zN^2}{n^2}$ time on a single stage. The speed-up achieved with z stages is $\frac{z}{1+\delta}$.

Deep pipelines generally increase speed-up. A deep pipeline also allows the computation to be spread across more processing units and this presents greater opportunities for reducing thermal hot-spots and for tailoring and tuning pipeline elements to particular tasks. A large value of N reduces the overhead δ and pushes the speed-up towards the ideal. In general, $N > n^2 > z > w$, so the constants A and B in $\delta = A + Bz$ are quite small. Under current technology, over 30 stages can be supported with an overhead of less than 5%. Put differently, if the computation was spread across 30 stages, the speed-up would drop to only about $0.96 \times 30 = 28.8$.

We show that if $z \leq \frac{n}{w}$ then the computation proceeds even more efficiently. Newer technologies such as optical and 3-D interconnects [13, 15, 16] will support a larger value n. On one hand,

this will admit a larger value of z for efficient operation. This, coupled with increasing processing capabilities, would allow for larger images to be handled without significant impact on processing time. On the other hand, a larger value of n (without a change in N) would improve the total time, albeit at a slightly lower efficiency.

We show that each stage of the pipeline requires $O(wN + n^2)$ space. With numbers reflecting current technology, processing a 10M-pixel image would require less that 128k bytes of space per stage.

This work also derives two broad results for computational pipelines that may have independent interest outside windowed computations (see Section 5).

The remainder of this paper is organized as follows. In the next section we formally define a windowed computation. In Section 3 we describe the computational pipeline, and define a template for executing a windowed computation on it. Section 4 is devoted to deriving the time for each stage and tile in the pipeline. In Section 5 we derive the running time for two broad scenarios on the pipeline and we put these intermediate results together in Section 6. In Section 7 we derive the memory requirement per stage of the pipeline. Finally in Section 8 , we make some concluding remarks.

2 Windowed Computations

In this section we introduce some definitions. These definitions are for the non-negative integer plane $\{(x, y) \ : \ \text{integer } x, y \geq 0\}$ using the L_1 norm (Manhattan space) in which all lines and distances are horizontal or vertical.

Let $P = [p_{x,y}]$ be an $N \times N$ image, where $0 \leq x, y < N$ and $p_{x,y}$ is the image pixel at row x and column y of the image.

Definition 1 *A w-neighbor of $p_{x,y}$ is a point whose horizontal or vertical distance does not exceed w. Formally, $p_{u,v}$ is a w-neighbor of $p_{x,y}$ iff $|x - u|, |y - v| \leq w$.* □

Definition 2 *A w-window (or window of size w) for pixel $p_{x,y}$ is the set*

$$N_w(x, y) = \{p_{x+u,y+v} \ : \ -w \leq u, v \leq w\}$$

of all w-neighbors of $p_{x,y}$. □

Figure 1 shows a w window for a pixel.

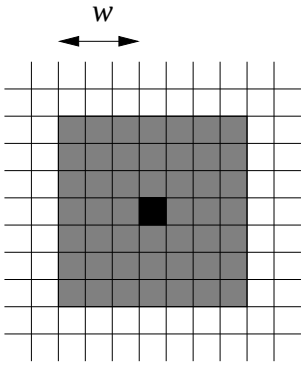

Figure 1: A window (shaded) of size $w = 3$ for a pixel (dark)

Definition 3 *A windowed computation on pixel $p_{x,y}$ refers to computing a function $f_w(x, y)$ whose input is the set $N_w(x, y)$ and whose output is associated with point (x, y). A windowed computation on an entire image applies f_w to each image pixel.* □

Several image processing operations can be described as windowed computations. For example, one form of median filtering uses the median of $N_w(x, y)$ as $f_w(x, y)$. Other filtering techniques use a $(2w+1) \times (2w+1)$ matrix $[m_{u,v}]$ and compute $f_w(x, y) = \sum_{u=-w}^{w} \sum_{v=-w}^{w} m_{u,v} \cdot q_{x+u,y+v}$. Morphological operators [4] for image processing are also windowed computations.

Definition 4 *Let $\mathcal{A}$ be an area (subset of the plane) of any shape. A w-contraction of $\mathcal{A}$ is the largest subset $\gamma_w(\mathcal{A}) \subseteq \mathcal{A}$ such that for every point in $\gamma_w(\mathcal{A})$, its w-neighbor is in $\mathcal{A}$.* □

Remarks: Let $\mathcal{A}$ be a subset of the image available to a processor. Then, the largest subset on which the processor can perform a windowed computation of size w is $\gamma_w(\mathcal{A})$. Where there is no danger of confusion, we will drop the w-subscript and denote a w-contraction simply as $\gamma(\cdot)$. In general, the set $\gamma(\mathcal{A})$ is obtained by excluding all elements of $\mathcal{A}$ that are within a horizontal or vertical distance of w from the perimeter of $\mathcal{A}$. However, for the purpose of Definition 4, we will allow $\gamma(\mathcal{A})$ to coincide with $\mathcal{A}$ for any portions that touch a border of the plane (or image) as the portion of the window outside the image can be constructed using portions within the image.

In Figure 1, the central pixel is the w contraction of the shaded window. Figures 6, 7 and 8 also illustrate w-contractions; all but the darkest shaded regions form a w-contraction of the entire shaded region in these figures. In these figures, however, the left, top, and in one case, right edges of the set are on the image border; the w-contraction does not recede by w pixels from these borders.

Generally speaking, image processing algorithms have large data sizes, admit high parallelism and require limited data sharing across individual computations (captured by the overlap of windows of nearby pixels). These algorithms can be viewed as series of windowed computations. For example, the scale invariant feature transform (SIFT) [7] has several stages of Gaussian filtering with windows of size 2 or 3, a stage of descriptor generation that can be viewed as a windowed computation with w around 8 and a few steps that are local to pixels (window size 0)[1]. In this paper we consider a series of z windowed computations on an $N \times N$ image. Each windowed computation uses a window of size w. The entire computation is run on a z-stage pipeline (see Section 3). The aim of this work is to develop core ideas that describe how this computation can be run on the pipeline.

3 The Pipelined Model of Computation

We use a z-stage pipeline with stages $S_0, S_1, \cdots S_{z-1}$, through which the entire image passes (see Figure 2). Stage S_0 is assumed to be the source of the image (possibly a camera). The input to

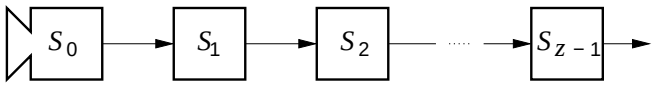

Figure 2: A z-stage pipeline

Stage S_1 is the original image (from Stage S_0). The input to Stage S_i (for $i > 0$) is the output of Stage S_{i-1}. The output of Stage S_{z-1} is the output of the algorithm. The setting is that of a streaming algorithm, in which a stage receives an input from the previous stage only once, and must save any part of the input that it requires for future use.

Each stage in the pipeline has two autonomous units: (a) input buffers that can receive inputs from the previous stage and (b) a computation/output unit that can process input data and output

[1] In this paper we assume that the window size is at least 1.

to the next stage. Since these units work independently, no major coordination is required between them. Note also that for the pipeline on the whole, since the output of Stage S_ℓ is concurrent with the input to Stage $S_{\ell+1}$, the time needed for a stage to handle a unit of data can be viewed as the sum of the computing and output times.

3.1 Tiles

The entire image is typically too large to move between stages in one "iteration." We will therefore partition the image into small "tiles" that can move more easily between stages. Tiles also act as atomic objects of the computation whose processing details can be abstracted away from, thereby allowing for a more general analysis of windowed computations. The size of tiles, though primarily indicative of the input/output bandwidth, will also reflect the computation speed of the stages. We elaborate on this further later.

For some $1 \leq w \leq n < N$, let $\xi = \frac{N}{n}$ and let $\rho = \frac{n}{w}$. (The assumptions that n divides N, $w \leq n$ and w divides n are only for ease of explanation.) The entire image is typically too large to move between stages in one "iteration." Partition the $N \times N$ input image into $n \times n$ *tiles* that are sequentially output in row-major order[2] from Stage S_0 to Stage S_1. Call these tiles $\tau_{k,0}$ for $0 \leq k < \xi^2 = \frac{N^2}{n^2}$. Figure 3 shows the tiles for an example with $\xi = 5$.

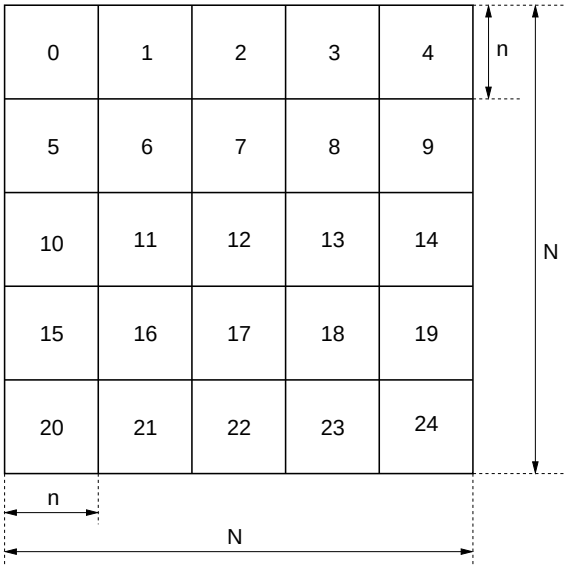

Figure 3: Partition of the image into tiles with $\xi = \frac{N}{n} = 5$. Numbers in the tiles indicate the value of k in Tiles $\tau_{k,0}$.

Consider a stage that has received (through a sequence of tiles) a subset $\mathcal{A}$ of the image. As observed earlier, the stage can perform a windowed computation on only a w-contraction $\gamma(\mathcal{A}) \subseteq \mathcal{A}$ of pixels. Given that it may have already processed some parts of $\gamma(\mathcal{A})$, the stage may currently process only a subset $\mathcal{A}' \subseteq \gamma(\mathcal{A})$.

Let us now consider this in the context of tiles received by Stage S_1 from S_0 (see Figure 4). On receiving the first $n \times n$ Tile $\tau_{0,0}$ from the northwest corner of the image, S_1 can process only $\gamma(\tau_{0,0})$, an $(n-w) \times (n-w)$ image subset (Figure 4 shows $\tau_{0,0}$ as the $n \times n$ square on the top-left corner of the image and the $(n-w) \times (n-w)$ subset as a medium shaded square within $\tau_{0,0}$). Call this $(n-w) \times (n-w)$ subset Tile $\tau_{0,1}$. Tile $\tau_{0,1}$ is output by S_1 to S_2. Next consider Tile $\tau_{1,0}$ (shown as another $n \times n$ square to the right of $\tau_{0,0}$). On receiving $\tau_{1,0}$ from S_0, Stage S_1 so far holds the subset $\mathcal{A} = \tau_{0,0} \cup \tau_{1,0}$ of the image. It can process subset $\gamma(\mathcal{A})$, but needs to process and pass on to

[2]The index of element (x, y) of an $X \times Y$ array enumerated in *row-major order* is $xY + y$, where $0 \leq x < X$ and $0 \leq y < Y$ (also see Figure 3).

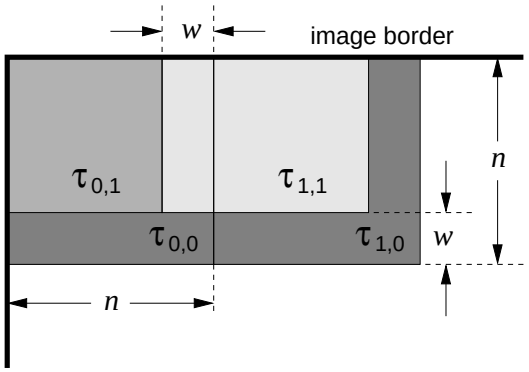

Figure 4: Movement of the first two tiles through stages S_0, S_1.

Stage S_2 only $\tau_{1,1} = \gamma(\mathcal{A}) - \tau_{0,1}$ (as $\tau_{0,1}$ has already been processed). This tile, $\tau_{1,1}$ is shown as a lightly shaded $(n - w) \times n$ rectangle.

In general, in "iteration" k, Stage S_ℓ receives $\tau_{k,\ell-1}$ as input and constructs

$$\tau_{k,\ell} = \gamma\left(\bigcup_{i=0}^{k} \tau_{i,\ell-1}\right) - \bigcup_{i=0}^{k-1} \tau_{i,\ell}$$

In subsequent discussion, Tile $\tau_{k,\ell}$ will be called the k^{th} tile of S_ℓ. The term *Tile k* or the notation τ_k refers to the collection of tiles $\tau_{k,\ell}$ for all $0 \leq \ell < z$. The period of time during which S_ℓ produces $\tau_{k,\ell}$ will be called its k^{th} *iteration.*

3.2 Running Times for a Stage

Let $|\tau_{k,\ell}|$ denote the area of (number of pixels in) Tile $\tau_{k,\ell}$.

In each iteration k, Stage S_ℓ does the following: (a) receives Tile $\tau_{k,\ell-1}$ from the previous Stage $S_{\ell-1}$, (b) generates Tile $\tau_{k,\ell}$ and (c) sends $\tau_{k,\ell}$ to the next Stage $S_{\ell+1}$. As noted earlier, part (a) above (tile input) is concurrent with the tile output by the previous stage. Thus, the time for Stage S_ℓ to handle a tile in iteration k is the sum of those needed for parts (b) and (c) above.

The time for Stage S_ℓ to output $\tau_{k,\ell}$ is clearly proportional to $|\tau_{k,\ell}|$, the number of pixels in the tile. The time to perform a windowed computation of size w, given Tile $\tau_{k,\ell-1}$, depends on w, $|\tau_{k,\ell}|$ and the function f_w. Since we have assumed w to be relatively small, we will also assume the total time for Stage S_ℓ to complete iteration k to be proportional to $|\tau_{k,\ell}|$. This assumption is reasonable, as in a specific instance of an image algorithm, the pipeline would be expected to be tuned to balance the computation and input/output times. The size $n \times n$ of input tiles $\tau_{k,0}$, that reflects the amount of data the camera can transmit in "unit time," can be used as a "step" in terms of which other times are expressed. Thus, we have the following assumption.

Assumption 1 *For any* $1 \leq \ell < z$ *and any* $0 \leq k < \xi^2$*, the time needed for Stage* S_ℓ *to complete the* k^{th} *iteration is* $t_{k,\ell} = \dfrac{|\tau_{k,\ell}|}{n^2}$. □

Remark: The actual times could differ from that assumed above by some constant amount that takes into account speeds of the stages (relative to the camera) and the nature of the computation required for f_w. However, the overall ideas of this paper would still be a good starting point to optimize for these differences.

To capture the notion of a "step" that transacts n^2 elements, we will say that the pipeline has a *bandwidth* of n^2.

4 Iteration Times (Tile Sizes)

In Section 3 we established that the time for a stage S_ℓ to complete iteration k is proportional to the size of Tile $\tau_{k,\ell}$ output during that iteration. In this section we first determine the size $|\tau_{k,\ell}|$ of that tile, which, in turn, would give the "raw processing times" of the tile. Subsequently, we will rearrange elements of tiles to obtain the "modified processing times."

4.1 Raw Iteration Times

We first introduce some alternate notation for a tile that will be useful in this section. Observe that for $0 \leq k < \xi^2 = \frac{N^2}{n^2}$, the $n \times n$ tiles $\tau_{k,0}$ partition the $N \times N$ image into a $\xi \times \xi$ array of tiles (see Figure 3). In this array, Tile $\tau_{k,0}$ is in row x and column y where $x = \left\lfloor \frac{k}{\xi} \right\rfloor$ and $y = k \pmod{\xi}$. Using this one-to-one correspondence between k and (x, y) we will write $\tau_{k,\ell}$ and $\tau_{(x,y),\ell}$ interchangeably. We will refer to an $a \times b$ tile as having *height* a and *width* b.

Since the image is initially tiled into $n \times n$ tiles $\tau_{k,0} = \tau_{(x,y),0}$ as shown in Figure 3, we have the following result.

Lemma 1 *For each $0 \leq x, y < \xi$, Tile $\tau_{(x,y),0}$ is an $n \times n$.* ■

Figure 4 showed $\tau_{0,0}$ and $\tau_{1,0}$ to be $n \times n$ tiles (consistent with Lemma 1.) It also showed $\tau_{0,1}$ to be an $(n-w) \times (n-w)$ tile and $\tau_{1,1}$ to be an $(n-w) \times n$ tile. Figure 5 further illustrates the pattern of sizes of Tiles $\tau_{k,0}$ (as dotted squares marked k), Tiles $\tau_{k,1}$ (as dotted rectangles marked k) and Tiles $\tau_{k,2}$ as solid rectangles marked k.

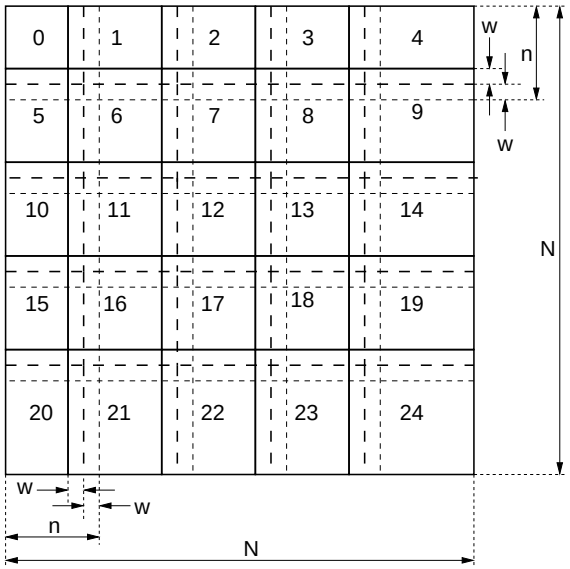

Figure 5: Tiles output by Stage S_2 (solid), by Stage S_1 (dashed) and Stage S_0 (dotted). Numbers indicate the value of k in Tiles $\tau_{k,\ell}$, for $\ell \in \{0, 1, 2\}$.

The following sequence of lemmas examine the size of $\tau_{k,\ell}$ for $\ell > 0$ and lead to Theorem 8, the main result of this section. Before we proceed, recall that $\gamma(\mathcal{A})$ is a w-contraction of set $\mathcal{A}$ and that $\frac{n}{w} = \rho$ is an integer. The results of this section hold for up to $\rho + 1$ stages. Therefore, the maximum stage index ℓ is ρ.

We consider the following six (non-disjoint) cases for Tiles $\tau_{(x,y),\ell}$: (a) $x = 0$, (b) $1 \leq x < \xi - 1$, (c) $x = \xi - 1$, (d) $y = 0$, (e) $1 \leq y < \xi - 1$, and (f) $y = \xi - 1$.

Case $x = 0$: Here Tile $\tau_{(0,y),\ell}$ is one of the top row of tiles and its top edge coincides with the image border.

Lemma 2 *For $0 \leq \ell \leq \rho$ and $0 \leq y < \xi$, Tile $\tau_{y,\ell}$ has a height of $n - w\ell$.*

Proof: For $0 \leq \ell \leq \rho$ and $0 \leq y < \xi$, let tile $\tau_{y,\ell}$ correspond to integer $g = \xi\ell + y$. That is, $0 \leq g < (\rho+1)\xi$ enumerates tiles in the following order.

$$\underbrace{\tau_{0,0}, \tau_{1,0}, \cdots, \tau_{\xi-1,0}}_{\tau_{y,0}}, \underbrace{\tau_{0,1}, \tau_{1,1}, \cdots, \tau_{\xi-1,1}}_{\tau_{y,1}}, \cdots, \underbrace{\tau_{0,\ell}, \tau_{1,\ell}, \cdots, \tau_{\xi-1,\ell}}_{\tau_{y,\ell}}, \cdots, \underbrace{\tau_{0,\rho}, \tau_{1,\rho}, \cdots, \tau_{\xi-1,\rho}}_{\tau_{y,\rho}}$$

We proceed by induction on $g \geq 0$. By Lemma 1, $\tau_{0,0}$ has height $n = n - w \cdot 0$ (base case). Assuming the lemma to hold for any $0 \leq g < (\rho+1)\xi - 1$, consider the case of $g+1$ that corresponds to tile $\tau_{y,\ell}$ (say). Figure 6 illustrates the situation. By the induction hypothesis, all

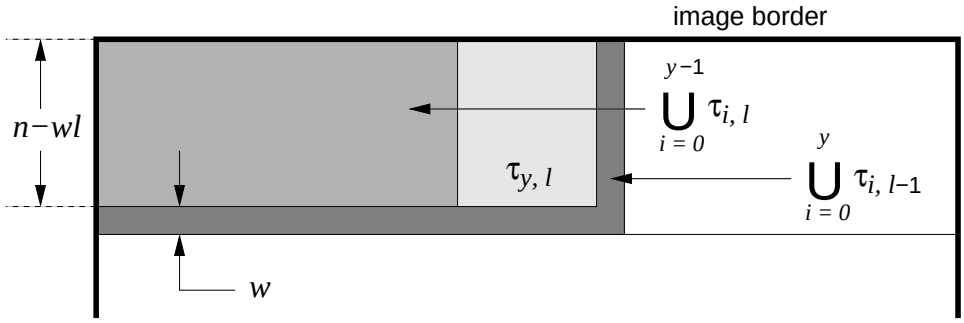

Figure 6: An illustration of the proof of Lemma 2.

tiles $\tau_{i,\ell-1}$ (if they exist) have a height of $n - w(\ell-1)$. Consider the contraction $\gamma\left(\bigcup_{i=0}^{y} \tau_{i,\ell-1}\right)$. Since the top of $\bigcup_{i=0}^{y} \tau_{i,\ell-1}$ touches the image boundary, the contraction reduces its height by w only on the bottom edge. Thus, $\gamma\left(\bigcup_{i=0}^{y} \tau_{i,\ell-1}\right)$ has a height of $n - w(\ell-1) - w = n - w\ell$. Again by the induction hypothesis, tiles $\tau_{i,\ell}$ (for $0 \leq i < y$) have a height of $n - w\ell$ as well. Since, $\tau_{y,\ell} = \gamma\left(\bigcup_{i=0}^{y} \tau_{i,\ell-1}\right) - \bigcup_{i=0}^{y-1} \tau_{i,\ell}$, the tile corresponding to $g+1$, namely $\tau_{y,\ell}$, has a height of $n - w\ell$. ■

Case $1 \leq x < \xi - 1$: Consider any tile $\tau_{(x,y),\ell}$, with $x > 1$. Observe that tile $\tau_{(x-1,y),\ell}$ (directly above $\tau_{(x,y),\ell}$) has already been received by Stage S_ℓ.

Lemma 3 *For $0 \leq \ell \leq \rho$, $1 \leq x < \xi - 1$, and $0 \leq y < \xi$, Tile $\tau_{(x,y),\ell}$ has a height of n and spans rows $xn - w\ell$ and $(x+1)n - w\ell - 1$.*

Proof: Consider any $\tau_{k,\ell} = \tau_{(x,y),\ell}$ with $x \geq 1$. Let $\tau_{(x,y),\ell}$ correspond to an integer $g = (\rho+1)\xi(x-1) + \xi\ell + y$; it can be verified that $0 \leq g < (\rho+1)\xi(\xi-2)$. We will proceed by induction on g.

The base case with $g = 0$ corresponds to the tile $\tau_{(1,0),0}$. Since $\ell = 0$, the height of the tile is n (by Lemma 1). Also since Tile $\tau_{(0,0),0}$ (that lies directly above $\tau_{(1,0),0}$) spans rows 0 to $n - w\ell - 1 = n - 1$ (Lemma 2), Tile $\tau_{(1,0),0}$ spans rows n and $2n - 1$, as required.

Assume the lemma to hold for any $0 \leq g < (\rho+1)\xi(\xi-2) - 1$ and consider the case of $g+1$. Let this value of g correspond to tile $\tau_{k,\ell} = \tau_{(x,y),\ell}$. Let $\tau_{k',\ell} = \tau_{(x-1,y),\ell}$ be the tile directly above $\tau_{k,\ell}$. Figure 7 illustrates this case.

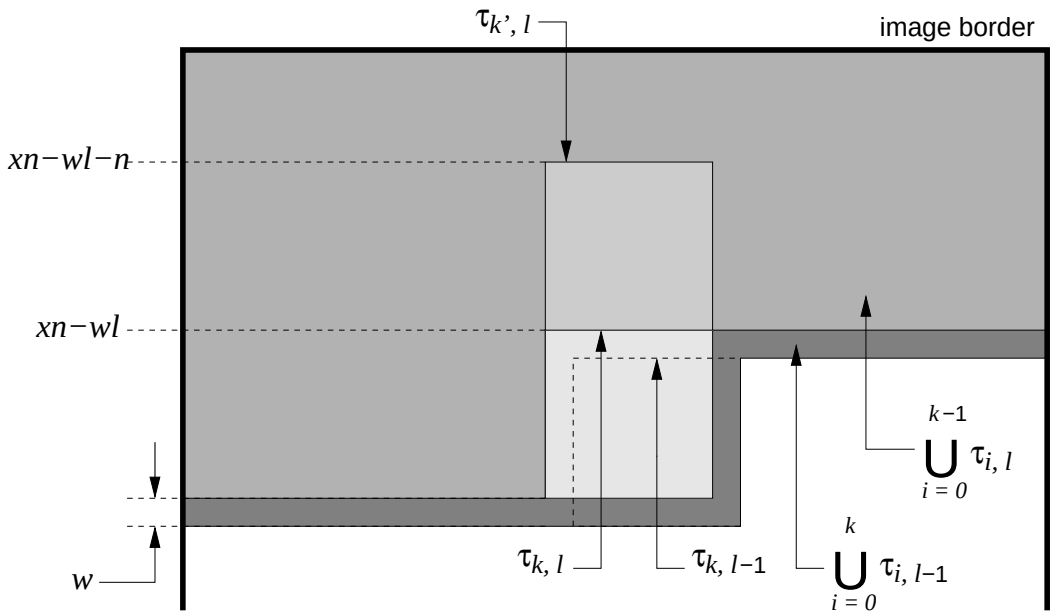

Figure 7: An illustration of the proof of Lemma 3.

Again $\tau_{k,\ell} = \gamma\left(\bigcup_{i=0}^{k} \tau_{i,\ell-1}\right) - \bigcup_{i=0}^{k-1} \tau_{i,\ell}$. The height of $\tau_{k,\ell}$ is bounded above by the lower border of $\tau_{k',\ell}$. By the induction hypothesis, $\tau_{k',\ell}$ spans $xn - w\ell - 1$. Therefore $\tau_{k,\ell}$ must have an upper border on row $xn - w\ell$ as required. As a result of a w-contraction, the lower border of $\tau_{k,\ell}$ is w units above the lower border of $\bigcup_{i=0}^{k} \tau_{i,\ell-1}$; specifically, it must be w above the lower border of those $\tau_{i,\ell-1}$'s that have the form $\tau_{(x,y'),\ell-1}$. By the induction hypothesis, these tiles have a lower border at $(x+1)n - w(\ell-1) - 1$. Consequently, the lower border of $\tau_{k,\ell}$ must be at $(x+1)n - w(\ell-1) - 1 - w = (x+1)n - w\ell - 1$.

In summary $\tau_{k,\ell}$ spans n rows from $xn - w\ell$ to $(x+1)n - w\ell - 1$, as required to complete the proof. ■

Case $x = \xi - 1$: This includes the last row of tiles of the image.

Lemma 4 *For $0 \le \ell \le \rho$ and $0 \le y < \xi$, Tile $\tau_{(\xi-1,y),\ell}$ has a height of $n + w\ell$.*

Proof: Since the tiles are the last row of tiles, their lower border coincides with the image border. Consequently, the contraction does not reduce the height. Thus, each of these tiles $\tau_{(\xi-1,y),\ell}$ ends at the last row of the image, namely $N - 1 = n\xi - 1$, and starts at row $(\xi-1)n - w\ell$ (immediately after tile $\tau_{(\xi-2,y),\ell}$ (see Lemma 3)). Thus the height of tile $\tau_{(\xi-1,y),\ell}$ is $n\xi - 1 - ((\xi-1)n - w\ell) + 1 = n + w\ell$. ■

Cases $y = 0$, $1 \le y < \xi - 1$ and $y = \xi - 1$: These cases are analogous to the first three cases. The situation of the $y = 0$ case is shown in Figure 8. Notice that the hatched portion of this figure is not relevant in determining the width of $\tau_{k,\ell}$, which make this analogous to the $x = 0$ case shown in Figure 6. The $1 \le y < \xi - 1$ and $y = \xi - 1$ cases similarly correspond to the $1 \le x < \xi - 1$ and $x = \xi - 1$ cases. Therefore, we have the following results that correspond to Lemmas 2, 3 and 4.

Lemma 5 *For $0 \le \ell \le \rho$ and $0 \le x < \xi$, Tile $\tau_{(x,0),\ell}$ has a width of $n - w\ell$.* ■

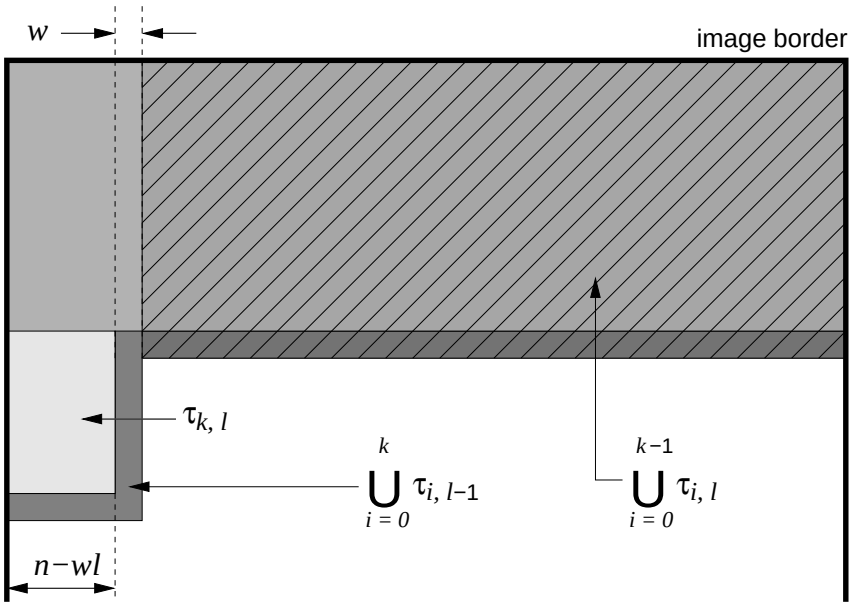

Figure 8: An illustration of the $y = 0$ case.

Lemma 6 *For $0 \leq \ell \leq \rho$, $0 \leq x < \xi$, and $1 \leq y < \xi - 1$, Tile $\tau_{(x,y),\ell}$ has a width of n and spans columns $yn - w\ell$ and $(y+1)n - w\ell - 1$.* ■

Lemma 7 *For $0 \leq \ell \leq \rho$ and $0 \leq x < \xi$, Tile $\tau_{(x,\xi-1),\ell}$ has a width of $n + w\ell$.* ■

The following theorem which gives the tile sizes, is a direct consequence of all the results developed in this section so far.

Theorem 8 *For any $0 \leq \ell \leq \rho$ and any $0 \leq x, y < \xi$, Tile $\tau_{(x,y),\ell}$ is an $(n + \alpha\ell) \times (n + \beta\ell)$ array, where*

- $\alpha = -w$, *for* $x = 0$
- $\alpha = w$, *for* $x = \xi - 1$
- $\beta = -w$, *for* $y = 0$
- $\beta = w$, *for* $y = \xi - 1$
- $\alpha = \beta = 0$, *for all* $0 < x, y < \xi - 1$

■

From Theorem 8 and Assumption 1 the "raw time" needed for Stage S_ℓ to complete various iterations is shown in the table of Figure 9. For example, Theorem 8 states that $\tau_{(0,0),\ell}$ is an $(n - w\ell) \times (n - w\ell)$ tile. So $|\tau_{(0,0),\ell}| = (n - w\ell)^2 = n^2 \left(1 - \frac{w\ell}{n}\right)^2 = n^2 \left(1 - \frac{\ell}{\rho}\right)^2$. By Assumption 1, the raw time needed to process $\tau_{(0,0),\ell}$ is $\left(1 - \frac{\ell}{\rho}\right)^2$. It is this quantity that is shown in entry $(0, 0)$ of the table in Figure 9.

4.2 Modified Iteration Times

An ideal pipeline has the same delay in each stage. This is clearly not the case for a windowed computation (see Figure 9). To better balance out the processing times across stages, one could reapportion elements across adjacent tiles within a stage.

	$y=0$	1		$\xi-2$	$\xi-1$
$x=0$	$\left(1-\frac{\ell}{\rho}\right)^2$	$1-\frac{\ell}{\rho}$	$\cdots$	$1-\frac{\ell}{\rho}$	$1-\left(\frac{\ell}{\rho}\right)^2$
1	$1-\frac{\ell}{\rho}$	1	$\cdots$	1	$1+\frac{\ell}{\rho}$
2	$1-\frac{\ell}{\rho}$	1	$\cdots$	1	$1+\frac{\ell}{\rho}$
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$\xi-2$	$1-\frac{\ell}{\rho}$	1	$\cdots$	1	$1+\frac{\ell}{\rho}$
$\xi-1$	$1-\left(\frac{\ell}{\rho}\right)^2$	$1+\frac{\ell}{\rho}$	$\cdots$	$1+\frac{\ell}{\rho}$	$\left(1+\frac{\ell}{\rho}\right)^2$

Figure 9: Raw iterations times for Stage S_ℓ. Entry (x, y) represents the time for S_ℓ to generate $\tau_{(x,y),\ell}$. Entries in red indicate that a portion of the tile is processed with the next tile (shown in green).

> For any Stage S_ℓ, and any tile index $0 \le k < \xi^2 - 1$, consider tiles $\tau_{k,\ell}$ and $\tau_{k+1,\ell}$ of sizes s_1 and s_2, respectively. The processing of $q \le s_1$ elements of $\tau_{k,\ell}$ can be deferred to the next tile $\tau_{k+1,\ell}$ with the use of q additional memory. The new sizes of $\tau_{k,\ell}$ and $\tau_{k+1,\ell}$ are $s_1 - q$ and $s_2 + q$, respectively.

Consider Tiles $\tau_{(x,\xi-1),\ell}$ (where $1 \le x < \xi - 1$) of Stage S_ℓ (indicated in red in Figure 9). Defer $\frac{n^2\ell}{\rho}$ elements of each of these tiles to the next tiles (indicated in green in Figure 9). The additional memory of $\frac{n\ell}{\rho} \le n$ required for this is quite modest considering that n^2 space is needed for the initial tile. The new "modified iteration times" are shown in Figure 10.

Initially as we consider the $z \le \rho$ case in Section 6.1, we will use the times from Figure 10. Later when we consider the $z \ge \rho$ case in Section 6.2, we will make further modifications to stage S_ρ.

5 Pipelining Time

So far, we have developed results for the amount of time each stage takes to individually process a tile. In this section we develop some preliminary results for the time needed to run the image through the pipeline. The results of this section are general and apply to any pipeline; therefore, they may be of independent interest. Subsequently in Section 6, we will apply the results of this section to the image pipeline.

Recall that $t_{k,\ell}$ denotes the time for Stage S_ℓ to complete iteration k. By Assumption 1, $t_{k,0} = 1$, for all k. Let $T_{k,\ell}$ be the earliest time when S_ℓ can begin iteration k. Clearly $T_{k,0} = \sum_{u=0}^{k-1} t_{u,0} = k$ as the the first stage S_0 does not wait on any other stage. Similarly, since iteration 0 of S_ℓ can start immediately after iteration 0 of $S_{\ell-1}$ has been completed, we have $T_{0,\ell} = \sum_{v=0}^{\ell-1} t_{0,v}$. For $k, \ell > 0$, Stage S_ℓ can start on iteration k after (a) S_ℓ has completed iteration $k-1$ and (b) after $S_{\ell-1}$ has sent it $\tau_{k,\ell-1}$. Thus,

$$T_{k,\ell} = \max\left\{T_{k-1,\ell} + t_{k-1,\ell}\ ,\ \ T_{k,\ell-1} + t_{k,\ell-1}\right\} \tag{1}$$

	$y=0$	1		$\xi-2$	$\xi-1$
$x=0$	$\left(1-\frac{\ell}{\rho}\right)^2$	$1-\frac{\ell}{\rho}$	$\cdots$	$1-\frac{\ell}{\rho}$	$1-\left(\frac{\ell}{\rho}\right)^2$
1	$1-\frac{\ell}{\rho}$	1	$\cdots$	1	1
2	1	1	$\cdots$	1	1
$\vdots$	$\vdots$	$\vdots$	$\cdots$	$\vdots$	$\vdots$
$\xi-2$	1	1	$\cdots$	1	1
$\xi-1$	$1+\frac{\ell}{\rho}-\left(\frac{\ell}{\rho}\right)^2$	$1+\frac{\ell}{\rho}$	$\cdots$	$1+\frac{\ell}{\rho}$	$\left(1+\frac{\ell}{\rho}\right)^2$

Figure 10: Modified iterations times for Stage S_ℓ. Entry (x, y) represents the time for S_ℓ to generate $\tau_{(x,y),\ell}$.

Special cases of this recurrence (that are useful for the problem at hand) can be solved. We need a few definitions.

Observe that at any given point in time if S_ℓ is in iteration k, then nominally Stage $S_{\ell+1}$ is in iteration $k-1$ (if it exists).

Definition 5 For any interval $[k_1, k_2]$ of iterations and Stage S_ℓ, we say that S_ℓ *dominates* $S_{\ell+1}$ (or $S_\ell \succeq S_{\ell+1}$) in $[k_1, k_2]$ iff for all iterations $k_1 < k \leq k_2$ we have $t_{k,\ell} \geq t_{k-1,\ell+1}$. Similarly, S_ℓ is dominated by $S_{\ell+1}$ (also written as $S_\ell \preceq S_{\ell+1}$ or $S_{\ell+1} \succeq S_\ell$) in $[k_1, k_2]$ iff for all iterations $k_1 < k \leq k_2$ we have $t_{k,\ell} \leq t_{k-1,\ell+1}$ □

Remarks: (a) When S_ℓ is handling $\tau_{k,\ell}$, Stage $S_{\ell+1}$ is handling Tile $\tau_{k-1,\ell+1}$. The conditions $t_{k,\ell} \geq t_{k-1,\ell+1}$ and $t_{k,\ell} \leq t_{k-1,\ell+1}$ are simply specifying which of the two stages (if any) stalls the pipeline.
(b) Note that the definitions bound k so that both $k, k-1 \in [k_1, k_2]$. Thus, even if $t_{k_1,\ell} < t_{k_1-1,\ell+1}$ was true, it would not violate the condition for $S_\ell \succeq S_{\ell+1}$ in $[k_1, k_2]$. Therefore, in asserting $S_\ell \succeq S_{\ell+1}$ or $S_{\ell+1} \succeq S_\ell$ in $[k_1, k_2]$, we may assume any convenient value for $t_{k_1-1,\ell+1}$ and $t_{k_2+1,\ell-1}$.

Recall that $T_{k,\ell}$ is the time when Stage S_ℓ starts on Tile $\tau_{k,\ell}$. This starting time assumes that all ξ^2 tiles will traverse the stage. In the following results we will consider running a subset of tiles from interval $[k_1, k_2]$ on S_ℓ. To capture this idea we use the notation $T^{k_1}_{k,\ell}$ to indicate that the time starts from tile $\tau_{k_1,\ell}$ (rather than $\tau_{0,\ell}$). In this notation $T_{k,\ell} = T^0_{k,\ell}$.

We now develop some results for processing subsets of tiles when the pipeline stages have dominance relationships. In the following, we assume the pipeline to have z stages, $S_0, S_1, \cdots, , S_{z-1}$.

Lemma 9 *Let $S_v \succeq S_{v+1}$ in $[k_1, k_2]$, for all $0 \leq v < z-1$ and $0 \leq k_1 \leq k_2 < \xi^2$. Then for $k_1 \leq k \leq k_2$ and $0 \leq \ell < z$,*

$$T^{k_1}_{k,\ell} = \sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k,v}$$

Proof: Because of the given dominance relationship, we have $t_{k,\ell} \geq t_{k-1,\ell+1}$. We proceed by induction on $k+\ell \geq k_1$. For the base case we observe that $T^{k_1}_{k_1,0} = 0 = \sum_{u=k_1}^{k_1-1} t_{u,0} + \sum_{v=0}^{-1} t_{k_1,v}$. Assuming

the lemma to hold for $k+\ell=n$, consider the case where $k+\ell=n+1$. By Equation (1) we have

$$T^{k_1}_{k,\ell} = \max\left\{T^{k_1}_{k-1,\ell} + t_{k-1,\ell} \quad , \quad T^{k_1}_{k,\ell-1} + t_{k,\ell-1}\right\}$$

Applying the induction hypothesis to $T^{k_1}_{k-1,\ell}$ and $T^{k_1}_{k,\ell-1}$ we have

$$\begin{aligned}
T^{k_1}_{k,\ell} &= \max\left\{T^{k_1}_{k-1,\ell} + t_{k-1,\ell} \quad , \quad T^{k_1}_{k,\ell-1} + t_{k,\ell-1}\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-2} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k-1,v} + t_{k-1,\ell} \quad , \quad \sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-2} t_{k,v} + t_{k,\ell-1}\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=1}^{\ell} t_{k-1,v} \quad , \quad \sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k,v}\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k-1,v+1} \quad , \quad \sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k,v}\right\} \\
&= \sum_{u=k_1}^{k-1} t_{u,0} + \sum_{v=0}^{\ell-1} t_{k,v}
\end{aligned}$$

The final step uses the dominance relationship to note that $t_{k,v} \geq t_{k-1,v+1}$. ■

This leads to the following result.

Corollary 10 *Let $S_v \succeq S_{v+1}$ in $[k_1,k_2]$, for all $0 \leq v < z-1$ and $0 \leq k_1 \leq k_2 < \xi^2$. Then the time to run all iterations $k \in [k_1,k_2]$ through a z-stage pipeline is $\sum_{u=k_1}^{k_2} t_{u,0} + \sum_{v=1}^{z-1} t_{k_2,v}$.*

Proof: The time T to run all the tiles is the time to start the last tile and run it; that is $T = T_{k_2,z-1} + t_{k_2,z-1}$. By Lemma 9 we have $T = \sum_{u=k_1}^{k_2-1} t_{u,0} + \sum_{v=0}^{z-2} t_{k_2,v} + t_{k_2,z-1} = \sum_{u=k_1}^{k_2} t_{u,0} + \sum_{v=1}^{z-1} t_{k_2,v}$. ■

Remark: Observe that because of the dominance relationship, earlier stages take more time than later stages, and later stages have to wait for earlier stages to complete before they can proceed. Thus, the overall time is the sum of the time needed to run all tiles through S_0 (the most dominant stage) and the time to run the very last tile through the remaining stages after it has traversed S_0.

Analogous results exist for a dominance relationship in the reverse direction.

Lemma 11 *Let $S_v \preceq S_{v+1}$ in $[k_1,k_2]$, for all $0 \leq v < z-1$ and $0 \leq k_1 \leq k_2 < \xi^2$. Then for $0 \leq k_1 \leq k \leq k_2 < \xi^2$ and $0 \leq \ell < z$,*

$$T^{k_1}_{k,\ell} = \sum_{u=k_1}^{k-1} t_{u,\ell} + \sum_{v=0}^{\ell-1} t_{k_1,v}$$

Proof: Because of the given dominance relationship, we have $t_{k,v} \geq t_{k+1,v-1}$. As before, we proceed by induction on $k+\ell \geq k_1$. The base case is the same as in Lemma 9. For the induction step, consider the case where $k+\ell = n+1$. As before, applying the induction hypothesis to $T^{k_1}_{k-1,\ell}$ and

$T^{k_1}_{k,\ell-1}$ we have

$$\begin{aligned}
T^{k_1}_{k,\ell} &= \max\left\{T^{k_1}_{k-1,\ell} + t_{k-1,\ell} \;\;,\;\; T^{k_1}_{k,\ell-1} + t_{k,\ell-1}\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-2} t_{u,\ell} + \sum_{v=0}^{\ell-1} t_{k_1,v} + t_{k-1,\ell} \;\;,\;\; \sum_{u=k_1}^{k-1} t_{u,\ell-1} + \sum_{v=0}^{\ell-2} t_{k_1,v} + t_{k,\ell-1}\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-1} t_{u,\ell} + \sum_{v=0}^{\ell-1} t_{k_1,v} \;\;,\;\; \left(\sum_{u=k_1+1}^{k-1} t_{u,\ell-1} + t_{k,\ell-1}\right) + \left(\sum_{v=0}^{\ell-2} t_{k_1,v} + t_{k_1,\ell-1}\right)\right\} \\
&= \max\left\{\sum_{u=k_1}^{k-1} t_{u,\ell} + \sum_{v=0}^{\ell-1} t_{k_1,v} \;\;,\;\; \sum_{u=k_1}^{k-1} t_{u+1,\ell-1} + \sum_{v=0}^{\ell-1} t_{k_1,v}\right\} \\
&= \sum_{u=k_1}^{k-1} t_{u,\ell} + \sum_{v=0}^{\ell-1} t_{k_1,v}
\end{aligned}$$

■

Again, this translates to the following result for running tiles through all z stages of the pipeline.

Corollary 12 *Let $S_v \preceq S_{v+1}$ in $[k_1, k_2]$, for all $0 \le v < z-1$ and $0 \le k_1 \le k_2 < \xi^2$. Then, the time to run all iterations $k \in [k_1, k_2]$ through the z-stage pipeline is $\sum_{u=k_1}^{k_2} t_{u,z-1} + \sum_{v=0}^{z-2} t_{k_1,v}$.*

Proof: As before, the time T to run all tiles is $T = T_{k_2,z-1} + t_{k_2,z-1} = \sum_{u=k_1}^{k_2-1} t_{u,z-1} + \sum_{v=0}^{z-2} t_{k_1,v} + t_{k_2,z-1}$ $= \sum_{u=k_1}^{k_2} t_{u,z-1} + \sum_{v=0}^{z-2} t_{k_1,v}$. ■

6 Windowed Computation Time on Pipeline

In this section we put the results of the previous section together to determine the total time needed to perform the z-stage windowed computation on the $N \times N$ image. We begin with the case where $z \le \rho = \frac{n}{w}$. Then we show how the $z \ge \rho$ case can be expressed in terms of the $z < \rho$ case.

6.1 Pipelines with $z \le \rho$ Stages

We develop dominance relationships between stages using the iteration times in Figure 10 and then use Corollaries 10 and 12 to derive running times.

Lemma 13 *If $z \le \rho = \frac{n}{w}$, then $S_\ell \succeq S_{\ell+1}$ in $[0, \xi-1]$, for $0 \le \ell < z-1$.*

Proof: We need to prove that for $0 < k \le \xi - 1$, $t_{k,\ell} \ge t_{k-1,\ell+1}$, or $t_{k,\ell} - t_{k-1,\ell+1} \ge 0$. Note that since $0 \le \ell < z \le \rho \le n$, we have $0 \le \frac{\ell+1}{n} \le 1$. We examine three cases.

Observe from Figure 10 that $t_{1,\ell} - t_{0,\ell+1} = \left(1 - \frac{\ell}{\rho}\right) - \left(1 - \frac{\ell+1}{\rho}\right)^2 = \frac{1}{\rho} + \left(1 - \frac{\ell+1}{\rho}\right)\left(\frac{\ell+1}{\rho}\right) > 0$.

For $2 \le k \le \xi - 2$, $t_{k,\ell} - t_{k-1,\ell+1} = \left(1 - \frac{\ell}{\rho}\right) - \left(1 - \frac{\ell+1}{\rho}\right) = \frac{1}{\rho} > 0$.

For $k = \xi - 1$, we have $t_{\xi-1,\ell} - t_{\xi-2,\ell+1} = \left(1 - \left(\frac{\ell}{\rho}\right)^2\right) - \left(1 - \frac{\ell+1}{\rho}\right) = \frac{1}{\rho} + \frac{\ell}{\rho}\left(1 - \frac{\ell}{\rho}\right) > 0$. ■

Lemma 14 *If $z \le \rho = \frac{n}{w}$, then $S_\ell \succeq S_{\ell+1}$ in $[\xi, \xi^2 - \xi]$, for $0 \le \ell < z-1$.*

Proof: Again we need to show that $t_{k,\ell} - t_{k-1,\ell+1} \geq 0$.

For $k = \xi + 1$, we have $t_{k,\ell} - t_{k-1,\ell+1} = 1 - \left(1 - \frac{\ell+1}{\rho}\right) = \frac{\ell+1}{\rho} > 0$.

For $\xi + 1 < k < \xi^2 - \xi$, we have $t_{k,\ell} - t_{k-1,\ell+1} = 1 - 1 = 0 \geq 0$.

For $k = \xi^2 - \xi$, we have $t_{k,\ell} - t_{k-1,\ell+1} = \left(1 + \frac{\ell}{\rho} - \left(\frac{\ell}{\rho}\right)^2\right) - 1 = \frac{\ell}{\rho}\left(1 - \frac{\ell}{\rho}\right) \geq 0$. ■

Lemma 15 *If $z \leq \rho = \frac{n}{w}$, then $S_\ell \preceq S_{\ell+1}$ in $[\xi^2 - \xi + 1, \xi^2 - 2]$, for $0 \leq \ell < z - 1$.*

Proof: Here we need to show that $t_{k,\ell+1} - t_{k+1,\ell} \geq 0$. For $\xi^2 - \xi + 1 \leq k \leq \xi^2 - 2$, we have $t_{k,\ell+1} - t_{k+1,\ell} = \left(1 + \frac{\ell+1}{\rho}\right) - \left(1 + \frac{\ell}{\rho}\right) = \frac{1}{\rho} > 0$. ■

The last tile, $\tau_{\xi^2-1,\ell}$, does not fall in a range with useful dominance properties for the stages.

We now use the above dominance properties with Corollaries 10 and 12 to determine the running times for tile segments. The following quantities that recur in the derivations in this section are denoted as shown below:

$$x_1 = \sum_{v=1}^{z-1} \frac{v}{\rho} = \frac{z(z-1)}{2\rho} \tag{2}$$

$$x_2 = \sum_{v=1}^{z-1} \frac{v^2}{\rho^2} = \frac{z(z-1)(2z-1)}{6\rho^2} \tag{3}$$

Corollary 16 *If $z \leq \rho$, then the time needed for tiles τ_k (where $0 \leq k < \xi$) to traverse all z stages is*

$$T_1 = \xi + z - 1 - x_2.$$

Proof: From Lemma 13 and Corollary 10, we have the time to be

$$T_1 = \sum_{u=0}^{\xi-1} t_{u,0} + \sum_{v=1}^{z-1} t_{\xi-1,v} = \sum_{u=0}^{\xi-1} 1 + \sum_{v=1}^{z-1} 1 - \frac{v^2}{\rho^2} = \xi + z - 1 - \sum_{v=1}^{z-1} \frac{v^2}{\rho^2} = \xi + z - 1 - x_2$$

■

With Lemma 14 and Corollary 10, we have the following result.

Corollary 17 *If $z \leq \rho$, then the time needed for tiles τ_k (where $\xi \leq k \leq \xi^2 - \xi$) to traverse all z stages is*

$$T_2 = \xi^2 - 2\xi + z + x_1 - x_2.$$

Proof: Here

$$T_2 = \sum_{u=\xi}^{\xi^2-\xi} t_{u,0} + \sum_{v=1}^{z-1} t_{\xi^2-\xi,v} = \xi^2 - 2\xi + 1 + \sum_{v=1}^{z-1} \left(1 + \frac{\ell}{\rho} - \frac{\ell^2}{\rho^2}\right) = \xi^2 - 2\xi + z + x_1 - x_2$$

■

With Lemma 15 and Corollary 12, we have the following result.

Corollary 18 *If $z \leq \rho$, then the time needed for tiles τ_k (where $\xi^2 - \xi < k \leq \xi^2 - 2$) to traverse all z stages is*

$$T_3 = \xi - 3 + \frac{(\xi - 3)(z-1)}{\rho} + z + x_1.$$

Proof: Here

$$T_3 = \sum_{u=\xi^2-\xi+1}^{\xi^2-2} t_{u,z-1} + \sum_{v=0}^{z-2} t_{\xi^2-\xi+1,v} = \sum_{u=\xi^2-\xi+1}^{\xi^2-2} \left(1+\frac{z-1}{\rho}\right) + \sum_{v=0}^{z-2}\left(1+\frac{v}{\rho}\right)$$
$$= (\xi-3)\left(1+\frac{z-1}{\rho}\right) + \sum_{v=0}^{z-1}\left(1+\frac{v}{\rho}\right) = \xi - 3 + \frac{(\xi-3)(z-1)}{\rho} + z + x_1$$

■

Lemma 19 *The time needed for the last tile to run through all z tiles is*

$$T_4 = z + 2x_1 + x_2.$$

Proof: Here we have

$$T_4 = \sum_{v=0}^{z-1} t_{\xi^2-1,v} = \sum_{v=0}^{z-1}\left(1+\frac{\ell}{\rho}\right)^2 = z + 2x_1 + x_2$$

■

Putting the results of Corollaries 16, 17, 18 and Lemma 19 together, we have the following result.

Theorem 20 *Let integers z, n, w satisfy $1 \le z \le \frac{n}{w}$. Then a sequence of z windowed computations of size w for an $N \times N$ image can be run on a z-stage, n^2-bandwidth pipeline in at most*

$$\frac{N^2}{n^2}\left[1+\delta_0\right] \quad \textit{steps,}$$

where

$$\delta_0 = \frac{(4n^2+wN)z + 2wnz^2}{N^2}.$$

Proof: The total time is the sum of the times in Corollaries 16, 17, 18 and Lemma 19. That is, $T = T_1 + T_2 + T_3 + T_4 = \xi^2 + 4(z-1) + \frac{(\xi-3)(z-1)}{\rho} + 4x_1 - x_2$. Substituting the values for x_1 and x_2 from Equations (2) and (3) and simplifying we have

$$T = \xi^2 + \left(\frac{z-1}{\rho}\right)\left(4\rho + (\xi-3) + 2z - \frac{z(2z-1)}{6\rho}\right) \le \xi^2 + \left(\frac{z}{\rho}\right)(4\rho + \xi + 2z).$$

By substituting $\xi = \frac{N}{n}$ and $\rho = \frac{n}{w}$ we have

$$T \le \xi^2 + \left(\frac{z}{\rho}\right)(4\rho + \xi + 2z) = \frac{N^2}{n^2}\left[1 + \frac{(4n^2+wN)z + 2wnz^2}{N^2}\right] = \frac{N^2}{n^2}\left[1+\delta_0\right]$$

■

Remarks: Since each input Tile $\tau_{k,0}$ requires unit time and there are z windowed computations per tile, the sequential time (on a 1-stage pipeline) needed for the computation is $\frac{zN^2}{n^2}$. Clearly, a lower bound for the computation time on a z-stage pipeline is $\frac{N^2}{n^2}$, the total number of input tiles. The pipeline time of $\frac{N^2}{n^2}(1+\delta_0)$ implies a speedup of $\frac{z}{1+\delta_0}$ with an overhead of δ_0. Typically N is quite large (around 3000 for a 10 Mpixel image that is quite routine in digital cameras). A large modern chip like an FPGA has a few hundred high-speed I/O pins. With 625 pins and $n = \sqrt{625} = 25$, assume that 625 pixels can be input in unit time. The window-size w is typically quite small. For example, the Scale Invariant Feature Transform algorithm as originally proposed by Lowe [7] uses $w \le 3$ in most stages. With $N = 3000$, $n = 25$ and $w = 3$, we have $\delta_0 \approx 0.0012z + 0.00017z^2$. With $z \approx \frac{n}{w} = 3$, the overhead $\delta_0 = 0.4\%$ is quite small.

By substituting $z = \rho$ in Theorem 20 we have the following result.

Corollary 21 *All ξ^2 tiles can be processed in a ρ-stage pipeline in at most*

$$\xi^2 + \xi + 6\rho \quad \textit{steps.}$$

■

6.2 Pipelines with $z > \rho$ Stages

We now address the case where the pipeline has $z > \rho$ stages. The last stage in a $(\rho + 1)$-stage pipeline is Stage S_ρ. By substituting $\ell = \rho$ in Figure 10 we have the table shown in Figure 11(a). Observe first that this table shows the sizes of tiles available (or their processing times) at successive

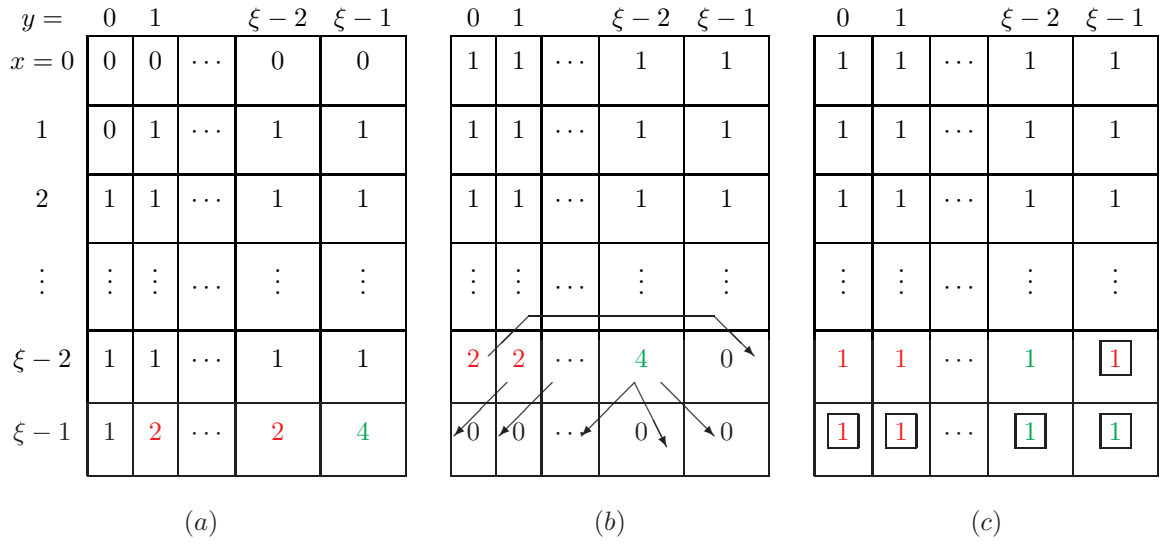

Figure 11: Running tiles on Stage S_ρ. Part (a) shows the modified iteration times for S_ρ. Part (b) shows iteration times assuming a $\xi + 1$ delay. Part (c) shows delayed iteration times with some tile times reapportioned across iterations. The arrows in part (b) show where partial tiles are reassigned. The numbers within boxes show the times corresponding to the reassigned partial tiles.

iterations. If these entries are shifted back by Δ positions in row-major order, then the new table will represent the sizes of tiles available after a delay of Δ. Figure 11(b) shows the same table with a delay of $\xi + 1$; the colors indicate corresponding entries.

As noted in Section 4.2, if a tile is available at a particular time, then it can be considered to be available at a future time, provided there is sufficient space to save the information until needed. Now consider, the first $\xi - 1$ tiles in row $\xi - 2$ of Figure 11(b). Each of the first $\xi - 2$ of these has a size of 2 (shown in red) and the last one has size 4 (shown in green). For each of the "red tiles," move one unit to an unoccupied position of Figure 11(b) as indicated by the arrows. For the "green tile," move three units to positions indicated by the three arrows. These movements entail an increase in the storage requirement of S_ρ by about 2ξ. As we show in Section 7, this requirement is quite small compared to the requirement for the normal operation of the pipeline.

The movement of tile segments shown in Figure 11(b) results in the tile sizes shown in Figure 11(c); the boxed entries denote the units moved along the arrows of Figure 11(b). Clearly, the table in Figure 11(c) (in which all tiles have unit size) is the same as the table for $\tau_{k,0}$ (see Figure 3).

Thus at stage S_ρ, the tiles are identical to those of S_0 except that there is a $\xi + 1$ delay before stage S_ρ starts. In general Stages S_ℓ and $S_{\rho+\ell}$ have identical distribution of tiles, except that stage $S_{\rho+\ell}$ incurs an additional $\xi + 1$ delay, compared to Stage S_ℓ.

Now we consider the problem of running the computation on $z > \rho$ stages. Let $z = q\rho + r$, for $0 \le r < \rho$. The first tile, τ_0, will run normally for the first ρ stages, requiring $\sum_{v=0}^{\rho-1} \left(1 - \frac{v}{\rho}\right) = \frac{\rho+1}{2}$ steps. Then after an additional $\xi + 1$ delay, the process repeats in the next ρ stages. The last set of r stages completes as indicated in Theorem 20 and its time is upper bounded by Corollary 21.

Thus assuming $\rho \geq 3$, the overall time is at most

$$q\left(\frac{(\rho+1)}{2}+(\xi+1)\right)+\xi^2+\xi+6\rho \leq \frac{z}{\rho}(\rho+\xi)+\xi^2+\xi+6\rho \leq \xi^2\left(1+\left(\frac{1}{\xi}+\frac{6\rho}{\xi^2}+\frac{z}{\xi^2}+\frac{z}{\rho\xi}\right)\right)$$

Putting this all together and substituting for $\rho = \frac{n}{w}$ and $\xi = \frac{N}{n}$ we have the following theorem.

Theorem 22 *A sequence of z windowed computations of size w for an $N \times N$ image can be run on a z-stage, n^2-bandwidth pipeline in*

$$\frac{N^2}{n^2}\left(1+\left(\frac{n}{N}+\frac{6n^3}{wN^2}+\frac{zw}{N}+\frac{zn^2}{N^2}\right)\right)$$

time. ■

Let $\delta = \left(\frac{n}{N}+\frac{6n^3}{wN^2}+\frac{zw}{N}+\frac{zn^2}{N^2}\right)$. So the total time is $\frac{N^2}{n^2}(1+\delta)$. As noted earlier, the pipeline time of $\frac{N^2}{n^2}(1+\delta)$ implies a speedup of $\frac{z}{1+\delta}$ with an overhead of δ.

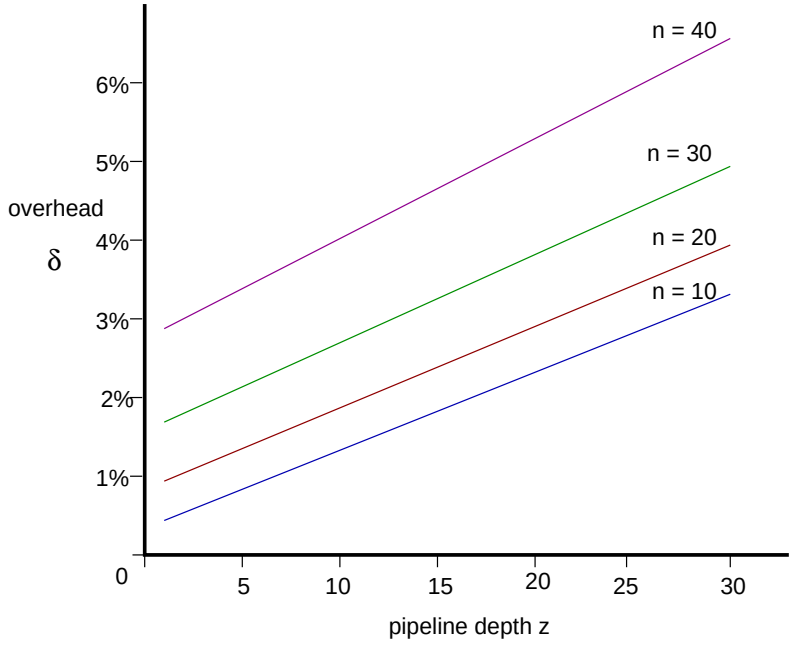

Figure 12: Overhead δ as a function of pipeline depth z. The graph assumes a tile size of $n = 25$ and illustrates several values of image size N.

The expression of Theorem 22 is a function of N, n, w, z. We now discuss the impact of these problem parameters. The quantity w indicates the window size of the image processing algorithm in question; each pixel uses a $(2w+1) \times (2w+1)$ window around it. The value of w is fixed by the algorithm and is typically quite small (around 3). As noted earlier, the values $N = 3000$ and $n = 25$ are quite reasonable under current technology. In the following discussion, we will often fix N and n at these values. With $w = 3$ as discussed above, the expression for the overhead $\delta = \left(\frac{n}{N}+\frac{2n^3}{N^2}+\frac{2z}{N}+\frac{zn^2}{N^2}\right)$ can be written as follows:

$$\delta = n\left(\frac{1}{N}+2\left(\frac{n}{N}\right)^2\right)+z\left(\frac{3}{N}+\left(\frac{n}{N}\right)^2\right) < (n+z)\left(\frac{3}{N}+2\left(\frac{n}{N}\right)^2\right)$$

Clearly for a fixed n and N, the overhead δ grows linearly with z. If $n \approx \sqrt{\frac{3N}{2}}$ (that is, $\frac{3}{N} \approx 2\left(\frac{n}{N}\right)^2$), then $\delta \approx \frac{6(n+z)}{N}$. With $N \gg 6$, this implies a relatively weak dependence of δ on z. That is, the overhead does not increase much for deep pipelines. On the other hand, if $n \gg \sqrt{\frac{3N}{2}}$, then $\delta \approx 2(n+z)\left(\frac{n}{N}\right)^2$, which could be somewhat large for small N or large n. Figures 12 and 13 illustrates this.

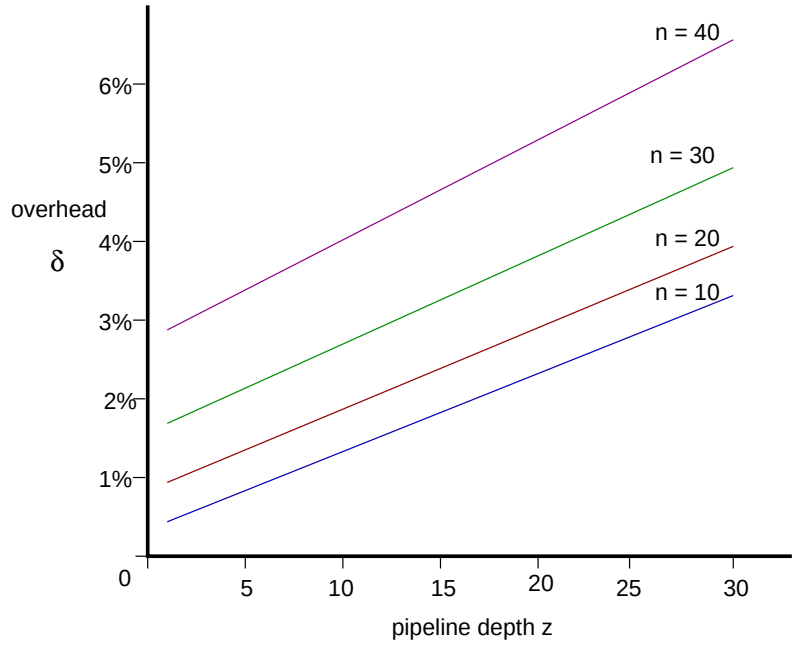

Figure 13: Overhead δ as a function of pipeline depth z. The graph assumes an image size of $N = 3000$ and illustrates several values of tile size n.

The number of stages z in the pipeline, to some extent, depends on the algorithm and the operation granularity. For example the SIFT algorithm [7] has four main stages, some of which can be further decomposed into "substages." For instance, the "extrema detection" phase of SIFT consists of a sequence of Gaussian blurrings (each of which is a windowed computation). Similarly other stages can be decomposed into substages. Thus the value of z used for SIFT could range from 1 (non-pipelined) to around 20 (with somewhat fine-grained operations). The decomposition of the algorithm into stages must balance processing times across stages to keep the computation flowing smoothly through the pipeline. For a given algorithm, clearly, a deep pipeline (with a large value of z) reduces the processing time of each stage and tends to produce a higher speedup. However, large values of z could increase inefficiencies caused by stalls between pipeline stages. Under current technology, $N \gg n > w$. As noted earlier, the values $N = 3000$, $n = 25$ and $w = 3$, are quite reasonable under current technology. For these values, $\delta \approx 0.012 + 0.0012z$. This sustains over 30 stages with an overhead of less than 5%. Put differently, with $z = 30$, the speedup drops from the ideal of 30 to only about $0.96 \times 30 = 28.8$. Figures 14 and 15 illustrate the drop in efficiency as z increases.

From Figures 12 and 14 it is clear that the pipeline is more efficient (smaller overhead) as the image size N increases (relative to tile size n). Figures 13 and 15 indicate that a larger tile size (relative to image size N) decreases efficiency. This seems to favor small tile sizes for a fixed image size. However, the absolute time needed to process the image is $\frac{N^2}{n^2}(1+\delta)$, so a large tile size has the benefit of smaller pipeline time (albeit with higher overheads). To get a better sense of the effect

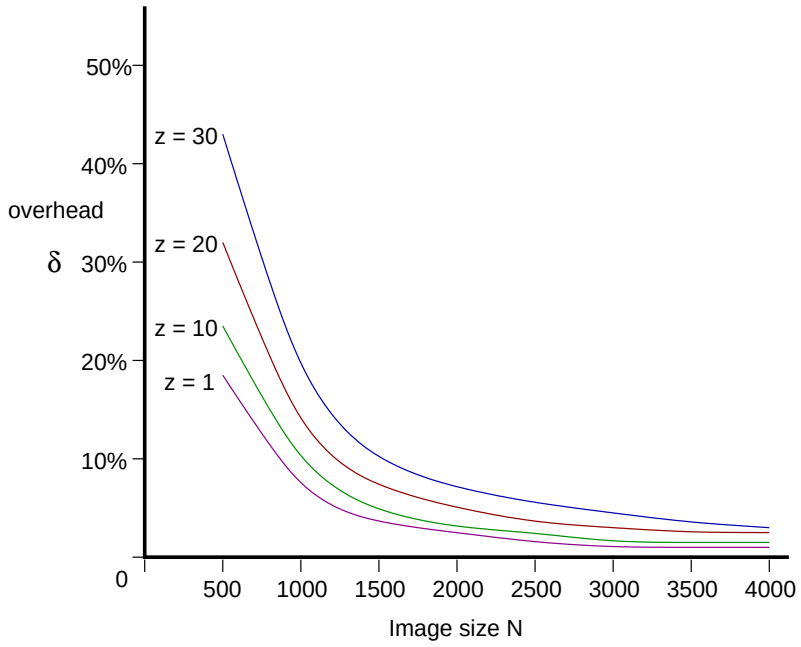

Figure 14: Overhead δ as a function of the image size N. The graph assumes a tile size of $n = 25$ and illustrates several values of pipeline depth z.

of tile size we look at the overhead δ in terms of $\xi = \frac{N}{n}$. Observe that

$$\delta = \frac{1}{\xi}\left(1 + \frac{zw}{n}\right) + \frac{1}{\xi^2}\left(\frac{6n}{w} + z\right)$$

In general, $N \gg n$ and so $\xi \gg 1$. Therefore, the coefficient of n in the above expression is much smaller than that of $\frac{1}{n}$. Consequently, δ decreases with n (at small values of n) at a much faster rate than it increases with n (at higher values of n). Figure 16 illustrates this. Observe that the overhead is nearly constant for large values of n, particularly for large values of ξ. That is, the small loss of efficiency in increasing tile size (by increasing I/O bandwidth) is more than amply compensated by the increase in speed.

Finally, we observe that our assumption of unit time for n^2 operations requires the computation in each stage to be suitably small, which requires z to be large. Our results show that a large value of z can be supported quite efficiently.

7 Memory Requirement

In this section we briefly analyze the memory requirement per stage. Tiles have a maximum size of around $4n^2 = O(n^2)$. Figure 17 shows the amount of space required by a stage to hold data needed for the current and future iterations. The medium shaded region represents the output tile and the dark regions represent the portion of the image received, but not yet processed. These must be saved in memory.

The total size of the memory needed is

$$wq + (w + b)(w + a) + w(N - q - w - b) = wN + wa + ab = \Theta(wN + n^2);$$

we have used the fact that $ab = O(n^2)$. The additional memory needed for reapportioning tiles (see Section 4.2 and 6.2 across iterations is $O\left(\frac{\ell}{\rho} + \xi\right) = O(\xi) = O\left(\frac{N}{n}\right)$, which is insignificant compared to the $\Theta(wN + n^2)$ space needed for normal operation.

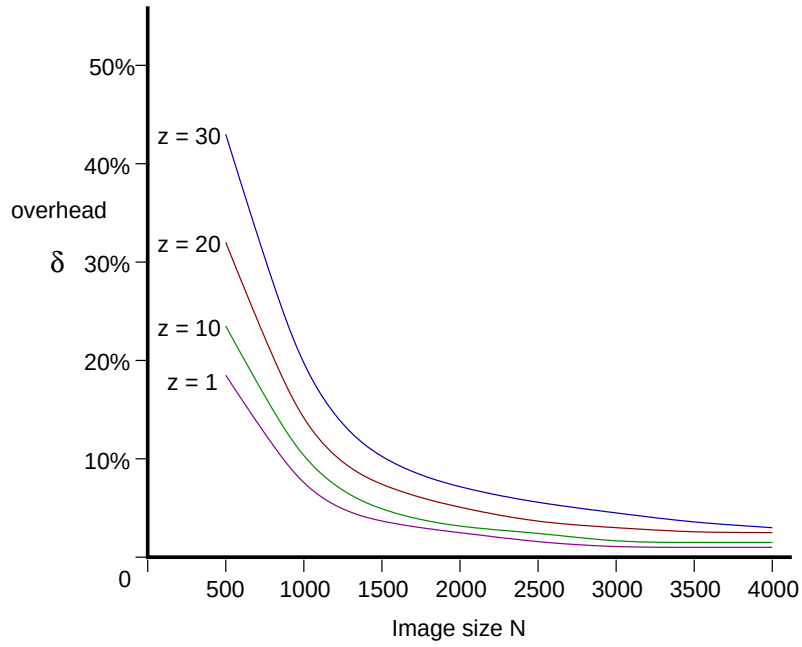

Figure 15: Overhead δ as a function of tile size n. The graph assumes an image size of $N = 3000$ and illustrates several values of pipeline depth z.

With $N = 4$K, $w = 3$, $n = 32$, and with the assumption that each pixel conservatively requires 8 bytes, the memory requirement per stage is less than 107K bytes.

8 Concluding Remarks

In this paper we have analyzed the running time for a sequence of windowed computations on a pipeline and shown that quite a deep pipeline is feasible under current technology. While particular application instances may differ from the assumptions made for our analysis, this work provides a good basis for optimizing these instances.

We have primarily dealt with the time and memory analysis of the pipeline. These could also be starting points for pipeline tuning. The stages could be designed and clocked independently to support a uniformity in computing and communication speeds across stages. For example, one stage could use a multicore chip whereas another, representing a less compute-intensive stage, may be implemented on a slower platform (possibly a lower clock rate or a uniprocessor chip). We have assumed a row-major enumeration of tiles. It can be shown that our results also hold for the snake-like row-major ordering. (It is important for adjacent tiles in the order to be spatially proximate. This facilitates output of these tiles from the camera through a scan path.) Is it possible for other input orders (such as a space-filling curve) to allow better performance? The input/output protocol we have assumed allows for tile sizes to vary across stages and over the image. This lack of uniformity could have disruptive effects in an implementation. One way is to fix input/output sizes and wait to produce an output tile until all necessary input pixels have been received. Some of these issues have been addressed in Phaneendra [10].

References

[1] S. Asano, T. Maruyama and Y. Yamaguchi, "Performance Comparison of FPGA, GPU and CPU in Image Processing," *Proc. Field Programmable Logic and Applications (FPL)*, 2009, pp. 126–

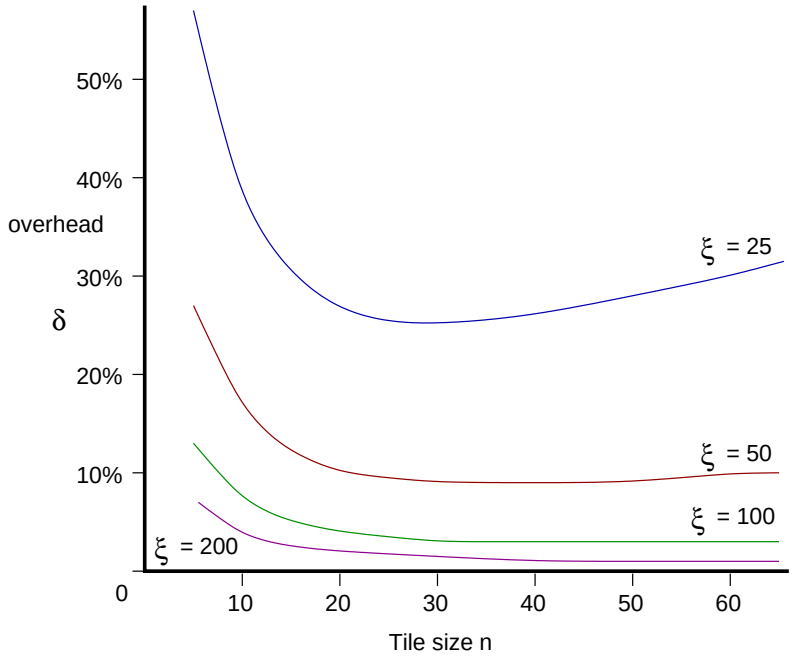

Figure 16: Overhead δ as a function of tile size n. The graph assumes an a pipeline size of $z = 20$ and illustrates several values of $\xi = \frac{N}{n}$.

131.

[2] P. M. Athanas and A. L. Abbott, "Real-Time Image Processing on a Custom Computing Platform," *IEEE Computer*, 1995, pp. 16–24.

[3] A. Benoit, U. V. Catalyurek, Y. Robert and E. Saule, "A Survey of Pipelined Workflow Scheduling: Models and Algorithms," LIP Research Report RR-LIP-2010-28, École Normale Supérieure de Lyon, France, 2010.

[4] A. C. Bovik, *The Essential Guide to Image Processing*, Academic Press (Elsevier), San Diego, 2009.

[5] B. A. Draper, J. R. Beveridge, A. P. W. Böhm and M. Chawathe, "Accelerated Image Processing on FPGAs," *IEEE Trans. Image Processing*, vol. 12, no. 12, Dec. 2003, pp. 1543–1551.

[6] M. Jordan, "A Configurable Decoder for Pin Limited Applications," Master's Thesis, Dept. of Electrical and Computer Eng., Louisiana State University, 2006.
`http://etd.lsu.edu/docs/available/etd-08192006-134649/`

[7] D. G. Lowe, "Object recognition from local scale-invariant features," *Proc. 7th Int. Conf. Computer Vision*, vol. 2, pp. 1150–1157, 1999.

[8] I. K. Park, N. Singhal, M. H. Lee, S. Cho and C. W. Kim, "Design and Performance Evaluation of Image Processing Algorithms on GPUs," *IEEE Trans. Parallel and Distributed Systems*, vol. 22, issue 1, pp. 91–104.

[9] J. R. Parker, *Algorithms for Image Processing and Computer Vision*, (2nd edition), Wiley Publishing inc. Indianapolis, 2011.

[10] P. Vinukonda, "A Study of the Scale Invariant Feature Transform on a Parallel Pipeline," Master's thesis, Electrical and Computer Engg. LSU, Baton Rouge.
`http://etd.lsu.edu/docs/available/etd-04272011-105721/`

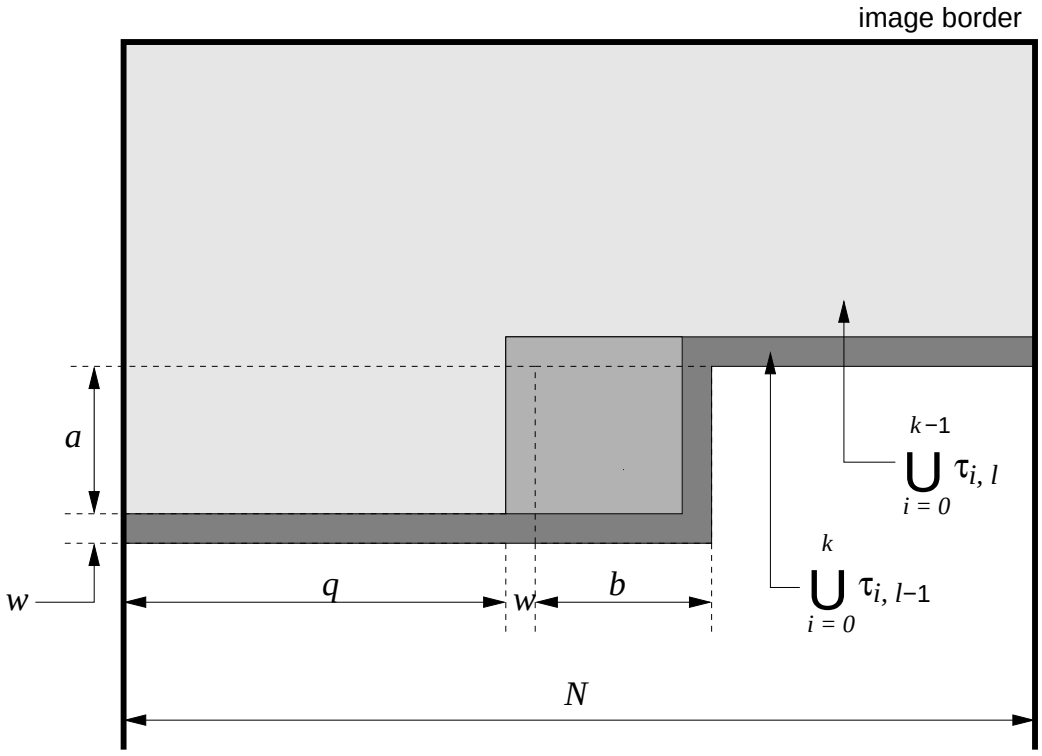

Figure 17: Memory requirement in processing an $a \times b$ tile. There must be enough memory to hold the portions shaded dark and medium.

[11] M.-S. Seok, I.-S. Song, S. Jin and J. W. Jeon, "A Real-time Window-based Image Processing Architecture using a Mapping Table," *Proc Control Automation and Systems*, 2010, pp. 1678–1681.

[12] N. Srivastava, J. L. Trahan, R. Vaidyanathan and S. Rai, "Adaptive Image Filtering using Run-Time Reconfiguration," *Proc. Reconfigurable Architectures Workshop, Int. parallel and Distributed Processing Symposium*, 2003. DOI: 10.1109/IPDPS.2003.1213332

[13] R. Weerasekera, D. Pamunuwa, L.-R. Zheng and H. Tenhunen, "Two-Dimensional and Three-Dimensional Integration of Heterogeneous Electronic Systems Under Cost, Performance, and Technological Constraints," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, issue 8, pp. 1237–1250, August 2009.

[14] Xilinx Inc., "Virtex-5 User Guide," http://direct.xilinx.com/bvdocs/userguides/ug190.pdf.

[15] S. W. Yoon, D. W. Yang, J. H. Koo, M. Padmanabhan and F. Carson, "3D TSV Processes and its Assembly/Packaging Technology," *Proc. Int. Conf. 3D System Integration*, pp. 1–5, 2009.

[16] I. A. Young, E. Mohammed, J. T. S. Liao, A. M. Kern, S. Palermo, B. A. Block, M. R. Reshotko and P. L. D. Chang, "Optical I/O Technology for Tera-Scale Computing ," *IEEE J. Solid-State Circuits*, vol. 45, issue 1, pp. 235–248, 2009.