Lossless Compression Techniques for Maskless Lithography Data

Vito Dai* and Avideh Zakhor

Video and Image Processing Lab
Department of Electrical Engineering and Computer Science
Univ. of California/Berkeley

ABSTRACT

Future lithography systems must produce more dense chips with smaller feature sizes, while maintaining the throughput of one wafer per sixty seconds per layer achieved by today's optical lithography systems. To achieve this throughput with a direct-write maskless lithography system, using 25 nm pixels for 50 nm feature sizes, requires data rates of about 10 Tb/s. In a previous paper, we presented an architecture which achieves this data rate contingent on consistent 25 to 1 compression of lithography data, and on implementation of a decoder-writer chip with a real-time decompressor fabricated on the same chip as the massively parallel array of lithography writers. In this paper, we examine the compression efficiency of a spectrum of techniques suitable for lithography data, including two industry standards JBIG and JPEG-LS, a wavelet based technique SPIHT, general file compression techniques ZIP and BZIP2, our own 2D-LZ technique, and a simple list-of-rectangles representation RECT. Layouts rasterized both to black-and-white pixels, and to 32 level gray pixels are considered. Based on compression efficiency, JBIG, ZIP, 2D-LZ, and BZIP2 are found to be strong candidates for application to maskless lithography data, in many cases far exceeding the required compression ratio of 25. To demonstrate the feasibility of implementing the decoder-writer chip, we consider the design of a hardware decoder based on ZIP, the simplest of the four candidate techniques. The basic algorithm behind ZIP compression is Lempel-Ziv 1977 (LZ77), and the design parameters of LZ77 decompression are optimized to minimize circuit usage while maintaining compression efficiency.

Keywords: maskless, lithography, compression

1. INTRODUCTION

Future lithography systems must produce more dense chips with smaller feature sizes, while maintaining the throughput of one wafer per sixty seconds per layer achieved by today's optical lithography systems. To achieve this throughput with a direct-write maskless lithography system, using 25 nm pixels for 50 nm feature sizes, requires data rates of about 10 Tb/s. The basic system design of a data processing system capable of delivering tera-pixel data rates necessary to achieve next-generation maskless lithography is shown in Figure 1. This design, presented in a previous paper [1], consists of storage disks, a processor board with DRAM memory, and a decoder-writer chip with data-decoding circuitry fabricated together with a massive array of pixel writers. Layout data for all layers of a single chip is compressed off-line and stored on the disks. Before the writing process begins, only a single compressed layer is transferred from disks to the processor board DRAM memory and stored there. As the writers write a stripe across the wafer, compressed data is streamed from the processor board to the decoder-writer chip in real-time as needed. The on-chip decoding circuitry, in real-time, expands the compressed data stream into the data signals necessary to control the writers.

There are several challenges that must be met for the design of Figure 1 to be feasible. The transfer of data from the processor board to the writer-decoder chip is bandwidth limited by the capability of chip to board communications. Although it is well known that this bandwidth does not increase with the same rate as Moore's law, nonetheless projections indicate that bandwidths around 400 Gb/s may be available by the later half of the decade either in the form of a large number of slow communication channels, e.g. 1000 pins operating at 400 MHz, or a few number of fast communication channels, e.g. 40 pins operating at 10 GHz [2]. This is the first challenge, maximizing the input data rate available to the decoder-writer chip.

* Correspondence: Email: vdai@eecs.berkeley.edu; WWW: http://www-video.eecs.berkeley.edu; Telephone: 510 643 1587

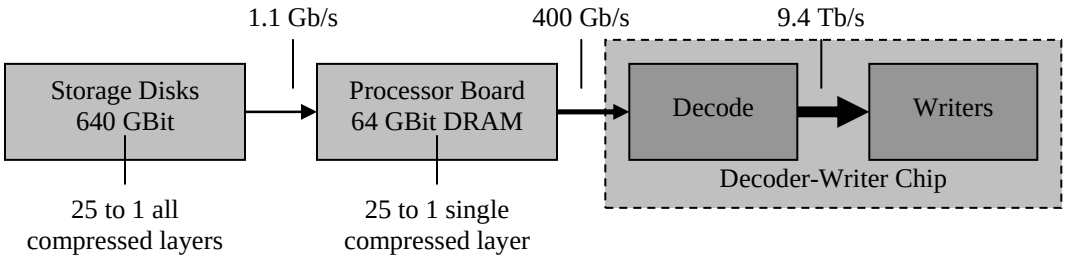

Figure 1. System architecture of a data-delivery system for maskless lithography

On the other hand, the decoder-writer chip is required to image 1.6 Tpixel/s to the wafer to meet the production throughput target of one wafer per layer per minute achieved by today's mask based lithography writers [1]. Assuming each pixel can take a gray value from 0 to 31, and a standard 5-bit binary representation, the effective output data rate of the decoder-writer chip is about 10 Tb/s. The shortfall between the input data rate and the output data rate is reconciled through the use of data compression, and a quick division, 10 Tb/s ÷ 400 Gb/s, yields the required average compression ratio of 25. This is the second challenge, i.e. developing lossless compressed representations of lithography data over 25 times smaller than the 5-bit gray pixel representation.

The third challenge involves the feasibility of building a decoder circuitry, a powerful data processing system in its own right, capable of decoding an input data rate of 400 Gb/s to an output data rate of 10 Tb/s. These data rates are many times larger than that achieved by any single chip decoding circuitry in use today. Moreover, this is not just a challenge to the creativity and competence of the hardware circuit designer. Depending on the compression algorithm used, the decoding circuitry has different buffering, arithmetic, and control requirements, and in general, higher compression ratios can be achieved at a cost of greater amount of hardware resources and longer decoding times, both of which are limited in this application. The decoder circuitry must share physical chip space with the writers, and it must operate fast enough meet the extremely high input/output data rates.

These observations are intended to establish the groundwork for discussion of feasibility and tradeoffs in the construction of a maskless lithography data delivery system, as well as approximate targets for research into meeting the three challenges outlined in this section. The first challenge, though important, is answered by the evolution of chip I/O technologies of the computer industry, and for the remainder of this paper we assume it is a challenge that can be met [9]. Section 2 answers the second challenge by presenting and evaluating the compression ratio achieved on modern industry lithography data by a spectrum of techniques: industry standard image compression techniques such as JBIG [11] and JPEG-LS [14]; wavelet techniques such as SPIHT [15]; general byte stream compression techniques such as Lempel-Ziv 1977 (LZ77) [12] as implemented by ZIP, and Burrows-Wheeler Transform (BWT) [13] as implemented by BZIP2; our own 2D-LZ [1], a two-dimensional extension of LZ77 matching algorithm; and RECT, an inherently compressed representation of layout as a list of rectangles. JBIG, ZIP, 2D-LZ, and BZIP2 are found to be strong candidates for application to maskless lithography data. Section 3 demonstrates the feasibility of implementing the decoder-writer chip, by considering the design of a high-speed decoder based on ZIP, the simplest of the four candidate techniques. The basic algorithm behind ZIP compression is LZ77, and the parameters of LZ77 decompression are optimized to minimize circuit usage while maintaining compression efficiency. Conclusions and directions for future research are presented in Section 4.

2. LOSSLESS LAYOUT COMPRESSION

To identify and develop a compression technique able to achieve a compression ratio of 25 on layout data, we apply a multitude of techniques to various industry layout data. Sections 2.1 and 2.2 describe the pre-processing of GDS-2 layout data before compression is applied, i.e. hierarchical flattening and rasterization. Section 2.3 describes the spectrum of compression techniques that are applied to the data. Section 2.4 presents the results of compression techniques applied to layout data, rasterized to a binary image with black-and-white pixels. Section 2.5 presents the results of compression techniques applied to more modern layout data, which requires rasterization to an image with

gray pixels. Several compression techniques achieve a compression ratio of 25, but the algorithms have vastly different implementation requirements that are difficult to compare directly. Section 2.6 provides a preliminary understanding of the relative complexity between the various algorithms based on buffer size.

2.1 Hierarchical flattening and rasterization

In a GDS-2 data file, layout data consists of a hierarchy of cells, each of which contains rectangles, polygons, and references to other cells. On the other hand, we assume the writer array in Figure 1 is an array of pixel writers, and therefore the control signals accepted by the writers are pixel intensities. Converting GDS-2 data into pixel intensity values requires flattening of the hierarchy, which converts cell references into rectangles and polygons, and rasterization, which converts rectangles and polygons into an image, a 2D array of pixel intensity values. The GDS-2 data file is in fact a very compact representation of layout, and which can even be further compressed [3]. This immediately raises the question as to whether the GDS-2 representation can be used as a possible candidate for the compression scheme need in Figure 1. Closer examination though reveals that this is not a feasible option. Specifically, a GDS-2 type representation stored on disk, would require the decoder-writer chip to perform both hierarchical flattening and rasterization in real-time. We believe that performing these operations, traditionally done by powerful multi-processor systems over many hours, with a single decoder-writer chip is impractical.

The alternative approach adopted in this paper is to perform both hierarchical flattening and rasterization off-line, and then apply compression algorithms to the pixel intensity data. Doing so offers some advantages. First, the decoder only needs to perform decompression, greatly simplifying its design. Second, any necessary signal processing, such as proximity correction or adjusting for resist sensitivity, can be computed off-line and incorporated into the pixel-intensity data before compression.

It is possible to adopt an approach in-between the two extremes, in which a fraction of the flattening and rasterization operation is performed off-line, and the remainder is performed in real-time by decoder-writer chip. Alternatives include adopting a more limited hierarchical representation that involves only simple arrays or cell references, or organizing rectangle and polygon information into a form that is easily rasterizable. Nonetheless it is unclear whether such representations offer either higher compression ratios or simpler decoding than the compressed pixel representation. As an example, a naïve list-of-rectangles representation, described shortly in Section 2.3 as RECT, does not necessarily offer more compression for the layouts tested, as shown in Table 3 of Section 2.4.

2.2 Rasterization parameters

The goal of rasterization is to convert the layout data into pixel intensity values which can be directly used to control the pixel-based writers themselves. Therefore, parameters of rasterization, such as the pixel-size and the number of gray-values, are specified by the lithography writer. Ideally, these parameters would be independent of the layout data itself, but such is not the case. For example, it is possible to write 300 nm minimum feature data with a state-of-the-art 50 nm maskless lithography writer using 25nm pixels, but doing so is extremely cost inefficient. Realistically speaking, lithography data is designed with some writer specification in mind, though this is not explicitly stated in the GDS-2 file. Hence, compression results should be reported with this target pixel size in mind.

What is troublesome about this situation is that compression ratios are data dependent, and it is entirely possible to report inflated compression ratios by artificially rasterizing the same GDS-2 data to a grid finer than the target pixel size the designers have in mind. However, because the target pixel size is not explicitly stated, it is difficult to ascertain whether this is or is not the case. For the layouts which we report compression results on, the writer specification is obtained from the layout owner, or is deduced from the GDS-2 file by measuring the minimum feature of the data. In all cases, we verify that in each layout image, there exists some feature which is two-pixels wide, corresponding to the two-pixels per minimum feature rule-of-thumb for pixel-based lithography writers.

GDS-2 files specify all of their polygons and structures on a 1 nm grid rather than the pixel grid described above. However, most layouts are built on a coarser address-grid that determines exact edge placement, in addition to the pixel grid defined by the minimum feature described above. When the edge-placement grid is the same as the pixel grid, then

the resulting rasterized layout image is a black-and-white binary image. When the edge-placement grid is finer than the pixel grid, each pixel adopts a gray value in order to achieve sub-pixel accuracy on edge placement. The entire rasterization process shown in Figure 2. To compute the actual gray value, our software rasterizes the layout on the finer edge-placement grid, generating a binary image. An $n \times n$ box filter is then applied to the image, where n denotes the ratio of the coarse pixel grid to the fine address grid. The values are then downsampled by n in both dimensions to generate intensity values on the coarse pixel grid. The resulting intensity values are then uniformly quantized to a minimum of n+1 gray values to preserve edge placement accuracy, but not corner accuracy. In general, however, the number of quantization levels is determined by the hardware writers and can be higher. For the example in Figure 2, n = 3 and the number of quantization levels is 4. As mentioned previously in Section 2.1, the filtering and quantization steps may be modified to adjust for proximity, resist sensitivity, and other effects.

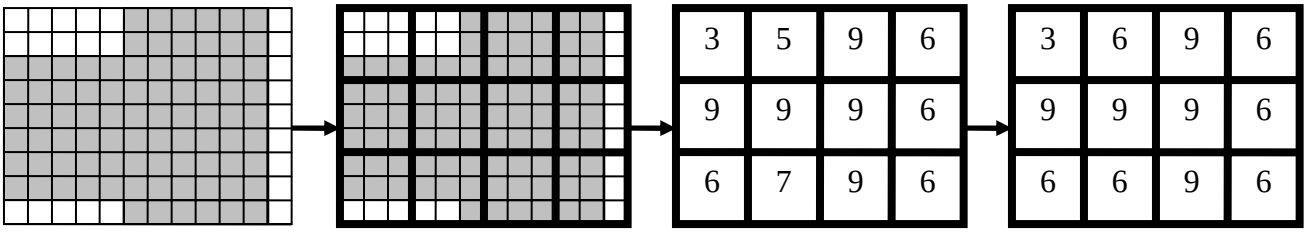

Figure 2. Rasterization process for generating gray pixels when the fine grid is 3 times smaller than the pixel grid

2.3 Spectrum of compression techniques: JBIG, SPIHT, JPEG-LS, ZIP, 2D-LZ, BZIP2, RECT

In this section, we provide a brief overview of each of the above compression techniques. JBIG [11] is a standard for lossless compression of binary images, developed jointly by the CCITT and ISO international standards bodies. JBIG uses a ten-pixel context to estimate the probability of the next pixel being white or black. It then encodes the next pixel with an arithmetic coder based on that probability estimate. Assuming the probability estimate is reasonably accurate and heavily biased toward one color, the arithmetic coder can reduce the data rate to far below one bit per pixel. The more heavily biased toward one color, the more the rate can be reduced below one bit per pixel, and the greater the compression ratio.

The lossless version of Set Partitioning in Hierarchical Trees (SPIHT) [15] is based on an integer multi-resolution transform similar to wavelet transformation designed for compression of natural images. Compression is achieved by taking advantage of correlations between transform coefficients. JPEG-LS [14] , an ISO/ITU-T international standard for lossless compression of grayscale images, adopts a different approach, using local gradient estimates as context to predict the pixel values, achieving good compression when the prediction errors are consistently small. Both techniques represent the state-of-the-art in lossless natural image compression.

ZIP is an implementation of the LZ77 [12] compression method used in a variety of compression utilities commonly available on today's computers, such as PKZip™, gzip, and WinZip™. It is highly optimized in terms of both speed and compression efficiency. The ZIP algorithm treats all types of data as a continuous stream of bytes. For binary images each byte is a block of 8 pixels sequenced in raster scan order, and for gray-pixel images each byte is one pixel value, again sequenced in raster scan order. To encode the next few bytes, it searches a window of up to 32 KB of previously encoded bytes to find the longest match to the next few bytes. If a match of sufficient length is found, the match position and length are recorded; otherwise, a literal byte is recorded. A sequence of control bits specifies whether a match or literal has occurred. Literals and match lengths are encoded together using one Huffman code, and the match position is encoded using another Huffman code. Although the LZ77 algorithm was originally developed with text compression in mind, where recurring byte sequences represent recurring words, applied to image compression it can compress recurring sequences of pixels. In general, longer matches and frequent repetitions increase the compression ratio.

2D-LZ is an extension of the basic LZ77 algorithm used by ZIP to two dimensions [1], capturing the inherent two-dimensional nature of layout data. Both binary and gray pixels can be encoded using raster-scan order. However, the linear search window, which appears in LZ77, is replaced with a rectangular search region of previously coded pixels. A match is now a rectangular *region*, specified with four coordinates: a pair of coordinates, *x* and *y*, specify the match position, and another pair of integers, *width* and *height*, specify the match. If a match of minimum size cannot be found, then a *literal* is outputted representing a vertical column of binary pixels, or a single gray pixel. A sequence of control bits is also stored so the decoder can determine whether the output is a literal or a match. To further compress the output, five Huffman codes are used: one for each of the match coordinates *x*, *y*, *width*, *height*, and one for the *literal*. In order to find the largest match region, we exhaustively test each pixel in the search region *(x,y)*. When a match at a particular position *(x,y)* is found, the *width* is increased as much as possible, while still ensuring a match; then the *height* is increased as much as possible. This procedure guarantees the widest possible match size for a given match position. The match position that results in the largest match size is chosen and stored as the match region. In general, larger match regions, and frequent matches increase the compression ratio.

	sort key			sort key
c	ompression	sort rows by key →	n	compressio
o	mpressionc		r	essioncomp
m	pressionco		s	ioncompres
p	ressioncom		o	mpressionc
r	essioncomp		o	ncompressi
e	ssioncompr		c	ompression
s	sioncompre		i	oncompress
s	ioncompres		m	pressionco
i	oncompress		p	ressioncom
o	ncompressi		s	sioncompre
n	compressio		e	ssioncompr

Figure 3. Block sorting of "compression" results in "nrsoocimpse"

	sort key
:	
t	ion (x,y) …
s	ion of th …
s	ion ratio …
g	ion speci …
s	ions, cap …
g	ions, fre …
:	

Figure 4. Effect of block sorting a paragraph

BZIP2 is an implementation of the Burrows-Wheeler Transform [13] which utilizes a technique called block-sorting to permute the sequence of bytes to an order that is easier to compress. For binary images, each byte is a block of 8 pixels sequenced in raster scan order, and for gray-pixel images, each byte is one pixel value, again sequenced in raster scan order. The permutation which occurs is in fact a sort, where the sort key of a byte is the string of bytes immediately following it. For example, in Figure 3, the characters of the string "compression" are block-sorted. The sort key for "c", is "ompression", the sort key for "o" is "mpressionc", etc. Since "ompression" comes sixth in lexicographical order, "c" is sixth letter of the permuted string; "mpressionc" comes fourth, so "o" is the fourth letter; etc. The block sorting result is the permuted string "nrsoocimpse", which is in fact, not any easier to compress than "compression". For block sorting to be effective, it must be applied to very long strings to produce an appreciable effect. Using, for example, the previous paragraph of this paper as a string, Figure 4 illustrates the effect of block sorting. Because the sub-strings "gion", "sion", and "tion" occur frequently, the sort keys beginning with "ion…" groups "g", "s", and "t" together. The resulting permuted string "… tssgsg …" is easy to compress using a simple adaptive technique called move-to-front coding [13]. In general, the longer the block of bytes, the more effective the block-sorting operation will be at grouping, and the greater the compression ratio.

RECT is not a compression algorithm, but simply an inherently compressed representation. Each layout image is generated from the rasterization of a collection of rectangles. If each rectangle is stored as a four 32-bit integers *(x, y, width, height)*, along with the necessary rasterization parameters, then this, in effect, is a compressed representation of the image data. As stated in Section 2.1, the drawback of this approach is that decoding this representation involves the complex process of rasterization in real-time. We have tested the possibility of further compression of the *(x, y, width, height)* representation by merging overlapping rectangles into Manhattan polygons, differentially encoding the vertices, and Huffman coding these

differences. This technique, known as Differential Vertex (DV) encoding achieves a compression ratio of about 4 to 6 over the RECT representation, as shown in Table 1. Decoding DV encoded rectangles requires both decompression and rasterization to be performed in real-time by the decoder-writer chip, making it by far the most difficult of the spectrum of compression algorithms to implement.

Layer #	RECT	DV Encoding	Compression Ratio
1	2.7 MB	550 KB	4.9
2	8.5 MB	1.8 MB	4.7
3	9.7 MB	1.6 MB	6.1
4	700 KB	150 KB	4.7
5	4.0 MB	900 KB	4.4

Table 1. Compression results for differential vertex encoding of rectangle data

2.4 Compression results for layout image data with binary pixels

In our previous work [1], the compression capability of JBIG, ZIP, 2D-LZ, and a cascade of a 2D-LZ encoding followed by ZIP encoding, are tested on several types and various layers of layout data rasterized from industry GDS-2 files by our own software system. Subsequently, a few improvements to the 2D-LZ algorithm [7] have been made to greatly increase its compression performance, beyond that of the 2D-LZ/ZIP cascade. The GDS-2 layout data, on which these algorithms are tested, consists of rectangles with a minimum feature of 600 nm, aligned to a coarse 300 nm edge-placement grid. Using the methodology described in Section 2.2, this data is rasterized to a 300 nm pixel grid, producing a black-and-white binary image of the layout, which is denoted in shorthand as "300 nm, 1 bpp (bits-per-pixel)." It is to this binary image which the compression algorithms are applied, and the compression ratios are reported. To these previously published results [7], we add the compression results of the Burrows-Wheeler Transform as implemented by BZIP2, and we present the overall result in Table 2.

Type	Layer	JBIG	ZIP	2D-LZ	BZIP2
Memory Cells	Metal 2	58.7	88.0	233	**260**
	Metal 1	9.77	47.9	**79.1**	55.5
	Poly	12.4	50.7	**120**	82.5
Control Logic	Metal 2	**47.0**	22.1	25.5	32.4
	Metal 1	**20.0**	10.9	11.2	10.6
	Poly	**41.6**	18.9	20.4	23.2
Mixed	Metal 2	**51.3**	28.3	34.4	39.4
	Metal 1	**21.2**	11.9	12.6	12.1
	Poly	**41.3**	22.9	27.2	27.8
Large Area	Metal 1	35.5	26.3	48.2	**48.5**

Table 2. Compression ratios for JBIG, ZIP, 2D-LZ and BZIP2 on 300 nm, 1 bpp data

The layout image data used in the first nine rows of Table 2 are 2048-pixel wide and 2048-pixel tall, which corresponds to a 0.61 mm by 0.61 mm section of the chip, covering about 0.1% of the chip area. The large area image is a 8192-pixel wide by 8192-pixel tall, which corresponds to a 2.4 mm by 2.4 mm section of a chip, covering about 1% of the chip area.

A detailed analysis of the results presented in Table 2 for JBIG, ZIP, and 2D-LZ has already been presented [7], and as such we focus on the compression performance of BZIP2 relative to those three techniques. In general the compression ratio of BZIP2 exceeds that of ZIP compression, and parallels the performance of 2D-LZ. Similar to 2D-LZ, BZIP2 achieves high compression efficiency for regular repetitive layouts such as memory cells, as well as the large area layout. For irregular layouts such as control logic or mixed logic, the efficiency of BZIP2 is significantly lower than that of JBIG.

Although for several layouts none of the compression techniques yield a compression ratio of 25, most notably metal 1 control logic and metal 1 mixed logic, it is worth noting that these compression results are reported for an image with black-

and-white pixels, whereas the compression ratio requirement of 25 is determined for layout with gray pixels. Certainly, if a particular layout with 50 nm minimum feature size could be rasterized using 25 nm pixels with black-and-white pixels, so that all edges are aligned on a 25 nm grid, for that layout, the required output data rate can be reduced by a factor of 5. This can potentially reduce the compression ratio requirement to 5, which is easily achieved by all the techniques tested in Table 2. All four compression techniques remain strong candidates for application to maskless lithography. To better extrapolate compression ratio achievable on future lithography data, we next consider the compression results of more modern layout data.

2.5 Compression results for layout image data with gray pixels

More recently, we have obtained the use of a GDS-2 file containing the active layer data of a piece of modern industry layout with the understanding that this layer would be fabricated using state-of-the-art lithography tools capable of printing 150 nm feature sizes, with an edge placement accurate to approximately 5 nm. Again applying the methodology presented in Section 2.2, portions of the layer are rasterized on a 5 nm grid to form a binary image, which is then filtered with a 15 × 15 box filter, and downsampled by 15 in each dimension. The resulting 226 intensity values are then quantized to 32 gray levels, to form a image with 75 nm pixels and 5 bits-per-pixel, denoted in shorthand as "75nm, 5bpp". To this data we apply SPIHT image compression, JPEG-LS image compression, 2D-LZ image compression, ZIP byte stream compression, BZIP2 byte stream compression, and the RECT compressed representation. The results are presented in Table 3.

Layout	SPIHT	JPEG-LS	RECT	ZIP	2D-LZ	BZIP2 (100K)	BZIP2 (900K)
active_a	8.44	9.27	33.9	45.7	111	227	**227**
active_b	9.69	9.76	61.1	61.1	144	497	**800**
active_c	5	5.31	18.7	46.4	328	296	**518**
active_d	7.44	8.45	24.5	60.1	273	319	**409**
active_e	9.37	11.3	72.8	47.3	145	189	**195**
Decoder buffering requirements				4 KB	200 KB	200 KB	1800 KB

Table 3. Compression ratios for SPIHT, JPEG-LS, RECT, ZIP, 2D-LZ, BZIP2, on 75 nm, 5 bpp data

The first column of Table 3 names the layout image to which the compression techniques are applied. *Active_a* is an image 1000-pixel wide and 1000-pixel tall, corresponding to a 75 µm × 75 µm square randomly selected from the center area of the chip. *Active_b* is an image 2000-pixel wide and 2000-pixel tall, corresponding to a 150 µm × 150 µm square selected from the same area as *active_a*, and includes *active_a*. *Active_c* through *active_e* are images 2000-pixel wide and 2000-pixel tall, corresponding to a 150 µm × 150 µm squares randomly selected from different parts of the chip, covering approximately 0.01% of a chip. Each sample is visually inspected to ensure that it is not mostly black.

The numbers in the next seven columns are compression ratios achieved on these images by the compression algorithms SPIHT, JPEG-LS, RECT, ZIP, 2D-LZ, and BZIP2 respectively, with each row corresponding to one of the *active* images described in the paragraph above. Comparing the second and third columns, JPEG-LS achieves slightly better compression than SPIHT, however, ZIP, 2D-LZ, and BZIP2 outperform them by more than a factor of 4. Although SPIHT and JPEG-LS are state-of-the-art lossless natural image compressors, they cannot take full advantage of the highly structured nature of images generated from lithography layout.

The fourth column RECT, is an "effective compression ratio" achieved if we had simply not performed rasterization at all, and stored the relevant rectangles in a list. A rectangle is considered relevant if any portion of it exists in the area being rasterized. The size of the RECT representation increases linearly with the number of rectangles inside the rasterization region, so a large "compression ratio" indicates few rectangles in the region, and a small "compression ratio" indicates more rectangles in the region. The "compression ratio" can therefore be interpreted as an inverse measure of rectangle density, which varies dramatically in different areas of the layout. Although at times RECT performs similarly to ZIP for *active_b* and *active_e*, it achieves less than half the compression ratio of ZIP for *active_c* and *active_d*, though as noted in Section 2.3, these compression ratios can be improved significantly, using geometry compression techniques such as DV Encoding.

In the last four columns, compression ratios for ZIP, 2D-LZ, and BZIP2 are listed. For BZIP2 there are two results corresponding to two different block sizes used for block sorting, 100KB blocks and 900KB blocks. All three techniques are flexible enough to be used for both 300 nm, 1 bpp data and 75 nm, 5 bpp data. Impressively, all three techniques exceed the target compression ratio of 25 with considerable margins. In general, the compression ratio of 2D-LZ exceeds that of ZIP, and the compression of ratio of BZIP2 exceeds that of 2D-LZ. The only exception is *active_c* where BZIP2 using a 100KB block has slightly less compression efficiency than 2D-LZ. All three data compression techniques remain strong candidates for application to maskless lithography. It is difficult to compare the compression results for ZIP, 2D-LZ, and BZIP2 between Tables 2 and 3, having been applied to distinctly different layout data.

It is worth noting that, although *active_a* and *active_b* are rasterized from the same area of layout, and thus share considerable similarity, their compression ratios differ so greatly, with *active_b* consistently achieving a higher compression ratio than *active_a*. This observation can be explained by the fact that *active_b* is a much larger image which covers an area of the chip 4 times larger than *active_a*. Intutively, *active_a* is a ¼ of *active_b*, so a compression algorithm operating on *active_b* can exploit the correlation between *active_a* and the other ¾ of *active_b* to achieve better compression. The same algorithm, given just *active_a* to work with, cannot possibly know of the other ¾ of *active_b*. In general, if an area is homogenous statistically, it is advantageous to compress as large a block as possible; and conversely, artificially breaking up a large homogenous area into smaller sub-blocks, and compressing each independently is typically detrimental to the compression ratio. The latter frequently happens because of implementation issues, for example, using multiple independent decoders to improve overall decoding speed, as discussed in Section 3.

2.6 Buffering requirements

The last row in Table 3 shows a comparison of the internal buffer size used by the three compression algorithms ZIP, 2D-LZ, and BZIP2. Since the compression algorithm can only work with data in its internal buffer, the buffer size is a measure of the maximum separation in the data stream, measured in bytes, when correlations between pieces of data can be found, and exploited for compression. For example, in ZIP and 2D-LZ, the buffer size determines how far back these algorithms can look to find an identical block of pixels to copy. For BZIP2, the internal buffer size is double the size of the block to sort, so BZIP2(100K) uses 200KB of buffer, while BZIP2(900K) uses 1800KB of buffer. The doubling is necessary for the BZIP2 decoder to reverse the effects of block-sorting. The internal buffer size of ZIP is by far the smallest at 4 KB, followed by 200 KB for 2D-LZ, 200 KB for BZIP2(100K), and 1800 KB for BZIP2(900K).

In general, the internal buffer used by each of the three techniques can be increased to improve compression efficiency at the expense of using more memory, or decreased to conserve memory, at the expense of compression efficiency. The last two columns of Table 3 illustrate this fact: when the buffer size used by BZIP2 increase from 200KB to 1800KB, the compression ratio increases by a factor of 1.0 to 1.6. In general the relationship between the compression ratio and buffer size is highly non-linear and data dependent, and the tradeoff between compression efficiency and buffer size only works over a limited range. There exist operating points for each of the three techniques beyond which increasing the internal buffer size no longer results in significant improvement in compression efficiency. At the same time, decreasing buffer size beyond a certain limit completely eliminates compression efficiency. The actual 4 KB buffer size used by ZIP, and the 200 KB to 1800 KB range of buffer sizes used by BZIP2, are widely recognized as “reasonably good” for compressing most data, when implemented on a PC where memory is plentiful and the main concern is compression efficiency. Similarly the buffer size of our implementation of 2D-LZ has been adjusted to a point where increasing the buffer size has little effect on improving the compression ratio.

However, for implementation on a decoder-writer chip, where chip space is a premium, the size of the internal buffer becomes a critical issue. In Section 3, we consider the design of a LZ77 decoder, which is the core compression technique behind ZIP compression and uses by far the smallest buffer among ZIP, 2D-LZ, and BZIP2. Parameters of LZ77 decoding, including the buffer size, are optimized to minimize circuit usage while maintaining compression efficiency.

2.7 Other performance comparisons

Another testimony to the speed and simplicity of decoding ZIP, relative to decoding BZIP2, can be inferred from the run-time performance of the decoders on a workstation. As an example, the authors in [10] present the decoding speed of ZIP and BZIP2 when applied to compressed text data on a 170 MHz Sun SPARCstation 5. Based on their experiment, ZIP decodes 4 MB of text data in 1.2 s, whereas BZIP2 decodes the same data in 8.0 s, so ZIP decoding is about 7 times faster than BZIP2 decoding. In general, one should be skeptical about interpreting speed measurements such as these; this is because not only

is lithography data significantly different from text data, but, more importantly, the architecture for implementing these decoders on a chip is extremely different from the architecture of a general-purpose microprocessor, such as that used by a SPARCstation. Nonetheless, this experimental result serves as a crude indication of the simplicity of decoding ZIP, relative to decoding BZIP2.

3. DESIGN OF A HIGH SPEED LZ77 DECODER

The primary difficulty of using compression to solve data transmission problems is that in order to be effective, the compressed representation must be decoded in a timely manner with reasonable computational and memory resources. Although we have established some relative measure of complexity between ZIP, 2D-LZ, and BZIP2 in Section 2, the fundamental question has not been answered. Is it even remotely feasible to implement a decoder-writer chip capable of decoding compressed data at a rate of 400 Gb/s and into 10 Tb/s of output data? To address this important question, we consider the design of a high-speed hardware decoder based on ZIP, which is much simpler in implementation than both 2D-LZ and BZIP2. The basic algorithm behind ZIP compression is Lempel-Ziv 1977 (LZ77), and the design parameters of a LZ77 decoder are optimized to minimize circuit usage while maintaining compression efficiency.

3.1 Fast LZ77 decoding architecture

The design of a fast LZ77 decoder, shown in Figure 5, consists of two decoding blocks, a Huffman decoder, and a match decoder. Because each decoder block has the potential to expand the data, and because the expansion factor varies in time with the compression ratio of the data, buffers are necessary to smooth both input and output data rates. In general, this would require three buffers, one in front of the Huffman decoder, one after the match decoder, and one between the two decoders. To reduce the buffering requirements, we choose an implementation of the Huffman decoding algorithm which observes a constant input rate, and we choose an implementation of the LZ77 decoding algorithm which observes a constant output rate. Consequently the variability in data rate is completely absorbed by a single buffer, capable of both variable input and output data rates.

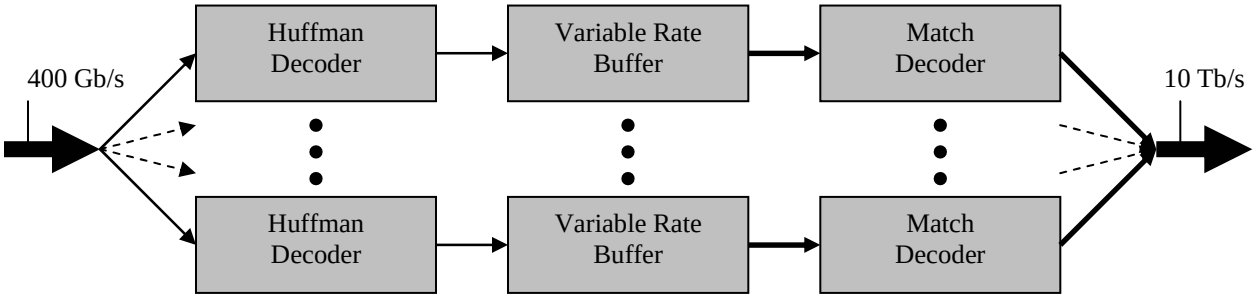

Figure 5. Multiple LZ77 decoders operating in parallel

For simplicity, a Huffman decoder with an input data rate of 1 bit per cycle is chosen. With this input rate, a single Huffman decoder must operate at 400 GHz to achieve an input throughput of 400 Gb/s. Again for simplicity, a match decoder with an constant output rate of one pixel per cycle is chosen, which corresponds to an output rate of 5 bits per cycle for 32 gray level data. A single match decoder must operate at 2 THz to achieve an output throughput of 10 Tb/s. To reduce these operating frequencies to more reasonable values, one approach is to implement Huffman algorithms that can decode *m*-bits per cycle or match decoders that can output *n*-pixels per cycle, with a corresponding increase in hardware complexity. The simpler alternative which we adopt is to use *k* independent LZ77 decoders to multiply both input and output data rate by a factor of *k*. To accomplish this, the layout data must be divided into blocks and compressed independently, though this may come at some cost to compression efficiency as discussed in Section 2.5. Estimating the actual number of LZ decompressors needed requires an estimate of the maximum operating frequency of the Huffman decoder and match decoder.

Using high level layout synthesis tools based on the Synopsis™ Module Compiler, the authors in [4] constructed a custom Huffman decoder, and a match decoder based on a systolic design [5]. Simulation results show that a Huffman decoder implemented in today's 0.25 µm technology can operate at 150 MHz, and a match decoder implemented in the same

technology can operate at 432 MHz. Based on these results, the output data rate of the match decoder is the bottleneck, and approximately 2 THz / 432 MHz = 4600 match decoders are needed to achieve the 10 Tb/s output rate. In contrast, the Huffman decoder is less of a bottleneck, and approximately 400 GHz / 150 MHz = 2700 Huffman decoders are needed to achieve the 400 Gb/s input rate. Based on chip area estimates derived from the simulation, the resulting system is about 10 times larger than can be economically fabricated today. Nonetheless, we do not believe the issues are intractable. A combination of better design and technology scaling works in our favor, both to speed up the decoders so that less are needed, and to reduce the size of the decoders so that more can fit on a chip. A more thorough discussion on the effects of technology scaling on the speed, size, and power, as well as layout of such a LZ77 decoder-writer chip, can be found in [2].

3.2 LZ77 match decoder – history buffer size and maximum match length

An important concern is the size of the history buffer used by the match decoder, which determines how far back the LZ77 algorithm can look to find matches. If the history buffer is too small, potential matches may be lost leading to a decrease in compression efficiency. If the history buffer is too large, the size of the match decoder circuit, which is directly proportional to the size of the history buffer, becomes excessively large. The size of the history buffer used by ZIP implementation of LZ77 in software is 4096 bytes, assuming each piece of data is one byte wide. To determine the optimal size of the history buffer to use for a hardware LZ77 implementation, we examine the compression ratio achieved by the match process only, without Huffman encoding, using history buffers of varying size.

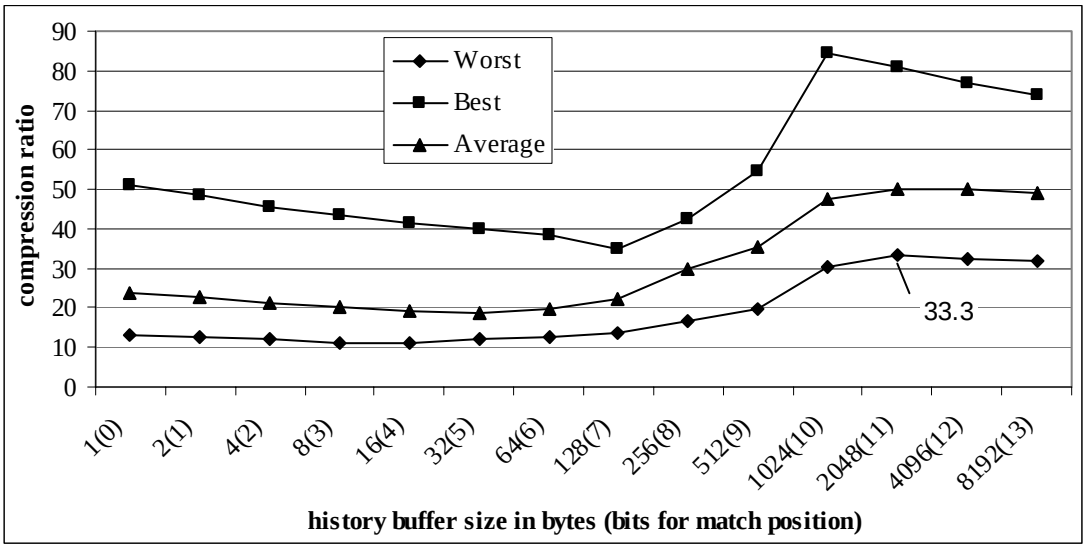

Figure 6. Effect of varying history buffer size on compression performance for uncompressed 16 KB with maximum match length of 256

For the experiment in Figure 6, the history buffer size is varied from 1 to 8192 bytes while holding the match length constant at 256. The horizontal axis indicates the length of the history buffer, as well as the number of bits required to encode the match position. The vertical axis indicates the compression ratio. Several portions of layouts, all 16 KB, rasterized from the active layer data described in Section 2.5 are tested, and the best-case, worst-case and average compression ratios are displayed. For application to the decoder-writer chip, the worst case results are the most important. As expected, best compression ratios are achieved by larger buffer sizes. For the layouts tested, the worst case compression ratio is maximized with a value of 33.3, when the size of the history buffer is 2048 bytes, and match positions are represented with 11 bits. Trading off compression efficiency for reduced buffer size, the size of the history buffer can be halved to 1024 bytes and the worse case compression ratio is still a reasonable 30.4. Further reducing the history buffer size causes worst case compression ratio to drop below 20.

A secondary concern is the maximum match length allowed by the match decoder. If the maximum match length is too short, potentially long matches will be broken up into smaller matches, causing a loss of compression efficiency. If the maximum match length is too long, some compression efficiency is lost to allow for extremely long matches that never occur. In this case, however, the decoder-writer hardware is only concerned with the number of bits needed to represent the match length,

which increases slowly as the base 2 logarithm of the maximum match length, and therefore not as important a concern. To find the optimal match length, we fix the size of the history buffer at 2048 bytes and vary the maximum match length, plotting the result in Figure 7.

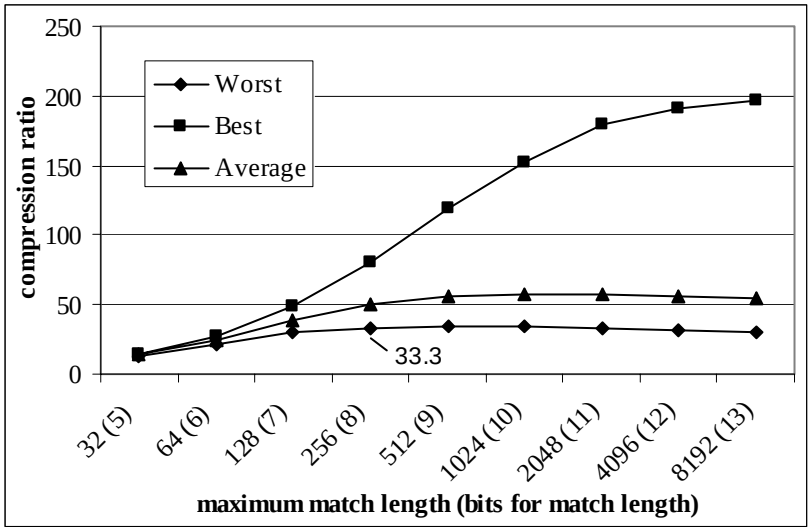

Figure 7. Effect of varying maximum match length on compression performance for 16 KB uncompressed data with history buffer length 2048

The horizontal axis of Figure 7 indicates the maximum match length, as well as the number of bits required to encode the match length. The vertical axis of Figure 7 indicates the resulting compression ratio. For data, the same 16 KB active layer layouts are used as in the previous experiment. The worst case compression ratio is maximized with a value of 34.6, when the maximum match length is set to 512, and match lengths are represented with 9 bits. This is a marginal improvement over the 33.3 compression ratio achieved in the previous experiment, when the maximum match length was fixed at 256.

4. SUMMARY, CONCLUSION AND FUTURE WORK

We have presented a spectrum of compression techniques suitable for application to maskless lithography data, and evaluated their compression performance on industry layout data. In each case, layout data is extracted from GDS-2 files through a process of hierarchical flattening and rasterization according to the specification of the designer or inferred from the GDS-2 file itself. In all cases, a two-pixel per minimum feature rule is observed and verified through visual inspection after rasterization. For layout data rasterized to 300 nm black-and-white pixels, JBIG, ZIP, 2D-LZ, and BZIP2 compression are applied, and compression ratios tabulated. For all cases, at least one of the techniques achieves a compression ratio of 20, and all four techniques remain strong candidates for application to lithography data with black-and-white pixels.

For layout data rasterized to 75 nm, 32 gray level pixels, SPIHT, JPEG-LS, RECT, ZIP, 2D-LZ and BZIP2 compression are applied, and compression ratios tabulated. SPIHT and JPEG-LS, state-of-the-art natural image compression algorithms, are ill-suited to layout images, achieving compression ratios of 5 to 11. RECT, a naturally compressed representation of layout as a list of rectangles, achieves compression ratios ranging from 19 to 73 depending on the rectangle density of the layout. The drawback of using RECT is that rasterization must be performed in real time. ZIP, 2D-LZ, and BZIP consistently exceed the compression requirement of 25, making them strong candidates for application to lithography data with gray pixels. ZIP, the simplest of the three algorithms, achieves compression ratios from 46 to 61 using a 4 KB internal buffer. 2D-LZ, achieves compression ratios from 111 to 328 using a 200 KB internal buffer. BZIP2, the most complex, achieves compression ratios from 189 to 497 using a 200 KB internal buffer, and 195 to 800 using a 1800KB internal buffer.

To test the feasibility of implementing the decoder-writer chip of the architecture presented in Figure 1, we discuss the design of a decoder-writer chip based on high-speed LZ77 decoding. Our choice of LZ77 is driven by the fact that it is the basis of ZIP compression, which is by far the simplest of the techniques, and applicable to both layout binary and gray pixel layout data. To optimize circuit space on the chip, a study finds that the compression efficiency of LZ77 is maximized with a history

buffer size of 2 KB. The history buffer size can be reduced to 1 KB with little loss in compression efficiency, but further reduction causes a much larger drop in efficiency. Based on a simple multiple-parallel decoder design, performance numbers are extracted with high-level synthesis tools using today's technology. Although the design is about 10 times larger than can be practically fabricated today, better design and technology scaling should bring it to the realm of feasibility in the future.

The work completed thus far demonstrates the potential of building a data-handling system for use with a pixel-based maskless lithography system for use in the 50 nm device generation. While LZ77 remains an attractive option from both an implementation and compression efficiency point of view, it is a very general algorithm. The possibility remains of developing a compression technique specifically engineered to take advantage of the highly structured nature of layout image data, and still preserve low implementation complexity.

ACKNOWLEDGEMENT

This research is conducted under the Research Network for Advanced Lithography, supported jointly by the funding of the Semiconductor Research Corporation (01-MC-460) and the Defense Advanced Research Project Agency (MDA972-01-1-0021).

REFERENCES

1. V. Dai and A. Zakhor, "Lossless Layout Compression for Maskless Lithography Systems", *Emerging Lithographic Technologies IV*, Elizabeth A. Dobisz, Editor, **3997**, pp. 467-477, SPIE, 2000.
2. B. Wild, *Data Handling Circuitry for Maskless Lithography Systems*, Master Thesis, UC Berkeley, 2001.
3. I. Ashida, Y. Sato, and H. Kawahira, "Proposal of new layout data format for LSI patterns", *Photomask and X-Ray Mask Technology VI*, **3748**, 205-213, SPIE, 1999.
4. V. Dai, M. Freed, and Y. Shroff, *Decompression Circuit Implementation Issues for Direct-Write Lithography Systems*, Project Report EE225c, UC Berkeley, 2000.
5. C. Chen and C. Wei, "VLSI design for high-speed LZ-based data compression", *IEEE Proceedings – Circuits Devices System*, **146** (5), pp. 268-277, IEEE, 1999.
6. Y. Shroff, Y. Chen, and W. Oldham, "Fabrication of parallel-plate nanomirror arrays for extreme ultraviolet maskless lithography", *Journal of Vacuum Science Technology*, **B 19** (6), 2001.
7. V. Dai, *Binary Lossless Layout Compression Algorithms and Architectures for Direct-write Lithography Systems*, Masters Thesis, U.C. Berkeley, 2000.
8. *The National Technology Roadmap for Semiconductors, 1997 Edition*, Semiconductor Industry Association, San Jose, CA, 1997.
9. E. H. Laine, and P. M. O'Leary, "IBM Chip Packaging Roadmap", *International Packaging Strategy Symposium*, SEMICON West, 1999.
10. I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes, Second Edition*, Academic Press, 1999.
11. CCITT, ITU-T Rec. T.82 & ISO/IEC 11544:1993, Information Technology – Coded Representation of Picture and Audio Information – Progress Bi-Level Image Compression, 1993.
12. J. Ziv, and A. Lempel, "A universal algorithm for sequential data compression", *IEEE Transasctions on Information Theory*, **IT-23** (3), pp.337-43, 1977.
13. M. Burrows, and D. J. Wheeler, "A block-sorting lossless data compression algorithm", Technical report 124, Digital Equipment Corporation, Palo Alto CA, 1994.
14. M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS", *IEEE Transactions on Image Processing*, **9** (8), pp.1309-24, 2000.
15. Amir Said and William A. Pearlman, "A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees", *IEEE Transactions on Circuits and Systems for Video Technology*, **6**, pp. 243-250, 1996.