Lossless Compression Algorithm for REBL Direct-Write E-Beam Lithography System

George Cramer, Hsin-I Liu, and Avideh Zakhor

Dept. of Electrical Engineering and Computer Sciences,
University of California, Berkeley
{cramerg, hsil, avz}@eecs.berkeley.edu

ABSTRACT

Future lithography systems must produce microchips with smaller feature sizes, while maintaining throughputs comparable to those of today's optical lithography systems. This places stringent constraints on the effective data throughput of any maskless lithography system. In recent years, we have developed a datapath architecture for direct-write lithography systems, and have shown that compression plays a key role in reducing throughput requirements of such systems. Our approach integrates a low complexity hardware-based decoder with the writers, in order to decompress a compressed data layer in real time on the fly. In doing so, we have developed a spectrum of lossless compression algorithms for integrated circuit layout data to provide a tradeoff between compression efficiency and hardware complexity, the latest of which is Block Golomb Context Copy Coding (Block GC3). In this paper, we present a modified version of Block GC3 called Block RGC3, specifically tailored to the REBL direct-write E-beam lithography system. Two characteristic features of the REBL system are a rotary stage and E-beam corrections prior to writing the data. The former results in arbitrarily-rotated layout imagery to be compressed, and as such, presents significant challenges to the lossless compression algorithms, including Block GC3. We characterize the performance of Block RGC3 in terms of compression efficiency and encoding complexity on a number of rotated layouts at various angles, and show that it outperforms existing lossless compression algorithms.

Keywords: GC3, C4, Lossless Compression, Segmentation, Data path, REBL, Direct-Write, Maskless, Lithography.

1. INTRODUCTION

Future lithography systems must produce chips with smaller feature sizes, while maintaining throughputs comparable to today's optical lithography systems. This places stringent data handling requirements on the design of any direct-write maskless system. Optical projection systems use a mask to project the entire chip pattern in one flash. An entire wafer can then be written in a few hundreds of such flashes. To be competitive with today's optical lithography systems, direct-write maskless lithography needs to achieve throughput of one wafer layer per minute. In addition, to achieve the required 1nm edge placement with 22 nm pixels in 45 nm technology, a 5-bit per pixel data representation is needed. Thus, the data rate requirement for a maskless lithography system is about 12 Tb/s [4]. To achieve such a data rate, we have recently proposed a data path architecture shown in Fig. 1[1]. In this architecture, rasterized, flattened layouts of an integrated circuit (IC) are compressed and stored in a mass storage system. The compressed layouts are then transferred to the processor board with enough memory to store one layer at a time. This board then transfers the compressed layout to the writer chip, composed of a large number of decoders and actual writing elements. The outputs of the decoders correspond to uncompressed layout data, and are fed into D/A converters driving the writing elements such as a micro-mirror array or E-beam writers. In this architecture, the writer system is independent of the data-delivery path, and as such, the path and the compression algorithm can be applied to arbitrary direct-write lithography systems.

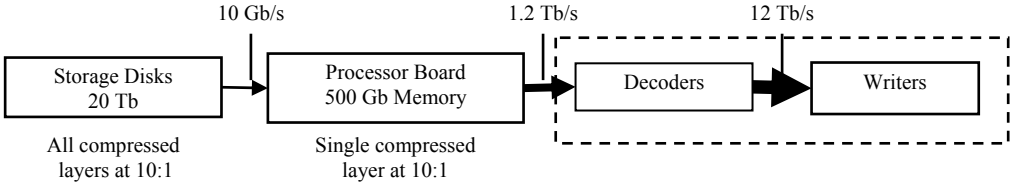


Fig. 1. The data-delivery path of the direct-write systems.

In the proposed data-delivery path, compression is needed to minimize the transfer rate between the processor board and the writer chip, and also to minimize the required disk space to store the layout. Since there are a large number of decoders operating in parallel on the writer chip, an important requirement for any compression algorithm is to have an extremely low decode complexity. To this end, we have developed a class of lossless layout compression algorithms for flattened, rasterized data [1-4], the latest of which, Block Golomb Context Copy Coding (Block GC3), has been shown to outperform all existing techniques such as BZIP2, 2D-LZ, and LZ77 in terms of compression efficiency, especially with limited decoder buffer size and hardware complexity, as required for hardware implementation[2][4].

A new writing system, called Reflective Electron Beam Lithography (REBL), is currently under development at KLA-Tencor[5]. In this system, the layout patterns are written on a rotary writing stage, resulting in layout data which is rotated at arbitrary angles with respect to the pixel grid. Moreover, the data is subjected to E-beam proximity correction effects. We have empirically found that applying the Block GC3 algorithm [4] to E-beam proximity corrected and rotated layout data results in poor compression efficiency far below those obtained on Manhattan geometry and without E-beam proximity correction. Consequently, Block GC3 needs to be modified to accommodate the characteristics of REBL data while maintaining a low-complexity decoder for the hardware implementation. In this paper, we modify Block GC3 in a number of ways in order to make it applicable to the REBL system; we refer to this new algorithm as Block Rotated Golomb Context Copy Coding (Block RGC3).

The outline of this paper is as follows. Section 2 provides a brief overview of the Block GC3 algorithm. Section 3 introduces the data-delivery path of the REBL system and the requirements it imposes on the compression technique. Section 4 describes the modifications that form Block RGC3. These include an alternate copy algorithm, which better suits layout patterns rotated with respect to the pixel grid, as well as a modified Block GC3 encoder, which introduces spatial coherence among neighboring blocks of Block GC3. Section 5 characterizes the additional encoding complexity required to implement our proposed changes to Block GC3. Section 6 presents results for different layout data with different parameters. Conclusions and future work are presented in Section 7.

2. REVIEW OF BLOCK GC3

The Block GC3 architecture is shown in Fig. 2, and is detailed in [4].

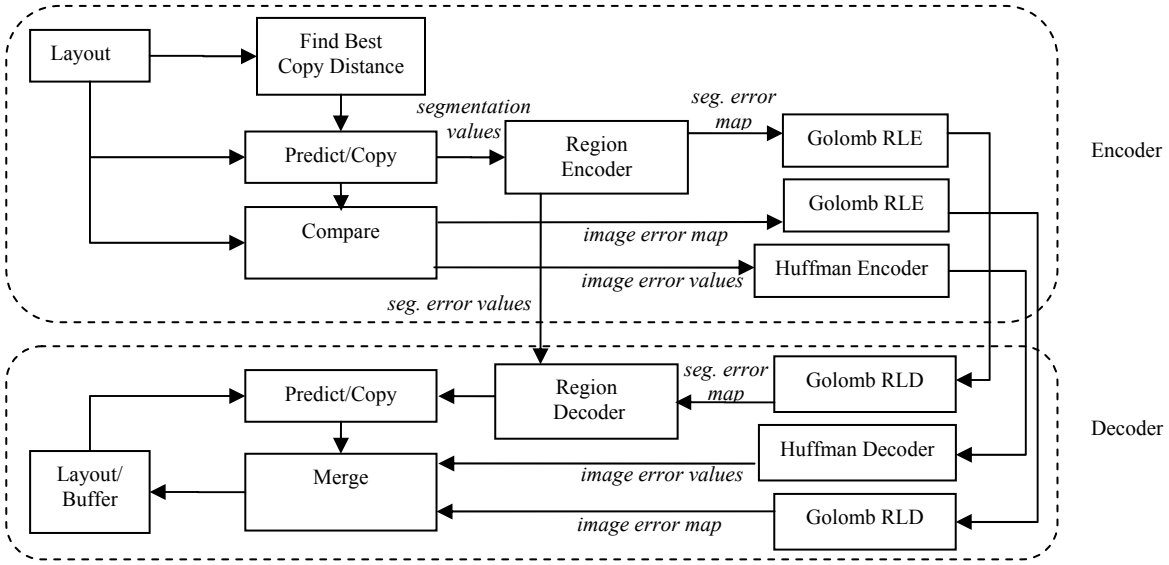


Fig. 2. The encoder/decoder architecture of Block GC3.

A summary of the algorithm is as follows. Each non-overlapping $H \times W$ section of pixels is grouped together as a "block". Each block is assigned a "segmentation value," which takes on either (a) a copy distance selecting an already-decoded $H \times W$ image section to copy from in LZ77 fashion, or (b) a copy distance of "0", which signals the decoder to predict the pixel values based on the properties of neighboring pixels. The set of segmentation values for the blocks forms its own pixel grid called the segmentation map, which is compressed by the region encoder block using a similar prediction method. Copy distances are strictly aligned either above or to the left of the block being encoded; diagonal copying is

not allowed, in order to minimize encoding complexity. In essence, Block GC3 attempts to exploit the benefits of both LZ-style copying and local context-based prediction. However, neither the segmentation values nor the image pixels are likely to be encoded free of errors. Specifically, to achieve higher compression efficiency, we allow errors to be introduced within the copy/prediction operation for the image and within the prediction operation for the segmentation map. For both the image and the segmentation map, error correction is performed by separating the bitstream into two sections: (a) an error map, in which a "zero" ("one") pixel signifies a correctly (incorrectly) predicted value for the corresponding pixel, and (b) a list of error values. Both the image and segmentation error maps are compressed using Golomb run-length codes. The image error values are compressed using Huffman codes, while the segmentation error values are transmitted uncompressed.

Each Block GC3 copy distance points to a location in a history buffer, which stores the most recently decoded pixels. A larger buffer increases the range of possible copy distances, and tends to improve compression efficiency at the expense of higher encoding complexity. In our previous work [4], limited decoder layout area led to the choice of a 1.7 KB buffer, which did not greatly degrade compression efficiency for non-rotated image patterns. However, in the REBL system, layout area considerations allow for a maximum buffer of 40 KB; this extra buffer is potentially valuable, since finding repetition is more challenging for arbitrarily-rotated image patterns.

3. DATA PATH FOR REBL SYSTEM

The REBL system is visualized in Fig. 3(a), and detailed in [5][6]. REBL's goal is to produce the high resolution of electron-beam lithography while maintaining throughputs comparable to those of today's optical lithography systems. The Digital Pattern Generator (DPG) uses reflective electron optics to constantly shape the electron beam as it scans across the wafers, which are located on a rotary stage shown in Fig. 3(b). This paper focuses on the data delivery path of the REBL system, which constrains the compression hardware implementation. As shown in Fig. 4, the compressed layout data is decoded by Block GC3 decoders in parallel, and then fed into the DPG, which can be located on the same chip as the decoder. In order to meet the required minimum wafer layer throughput of the REBL system, namely 5-7 wafer layers per hour (WPH), given the data rate of the available optical input data link of about 10Gb/s/link, a required minimum compression ratio of 5 is projected.

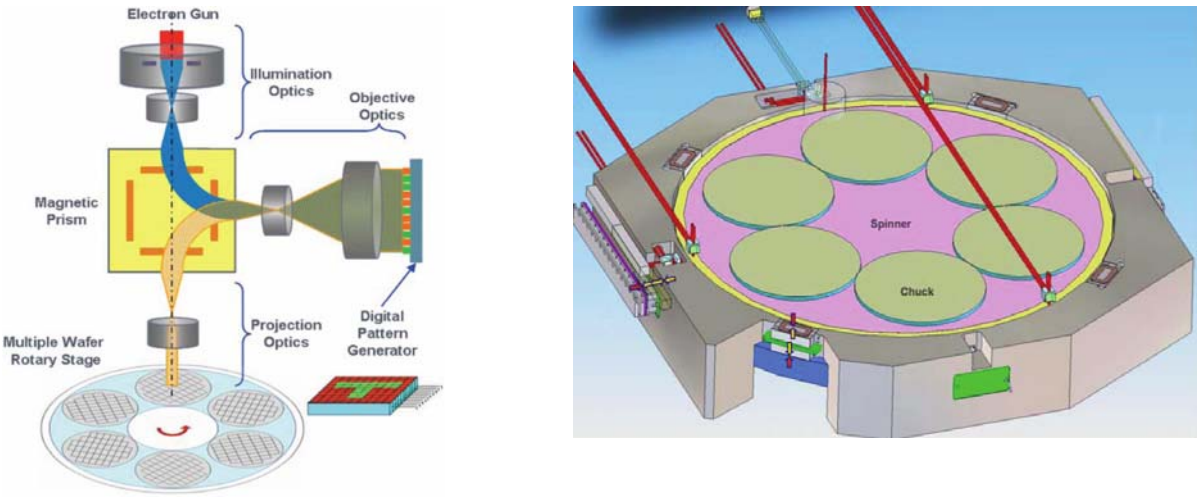


Fig. 3. (a) Block diagram of the REBL Nanowriter; (b) detailed view of the rotary stage [5].

In the REBL system architecture, similar to the architecture presented in [1], every data path can be handled independently, with its own input data link. Moreover, in the REBL system, the DPG reads layout patterns from the decoders in a column-by-column fashion. Every decoder provides data for a set number of pixel rows: either 64 or 256 rows. The number of columns in each compressed 64- or 256-row "image" effectively can be thought of as being infinite, since the writing system runs continuously until the entire wafer is written. For testing purposes, we restrict the number of columns to either 1024 or 2048. Properties of the test layout images for this paper are listed in Table 1.

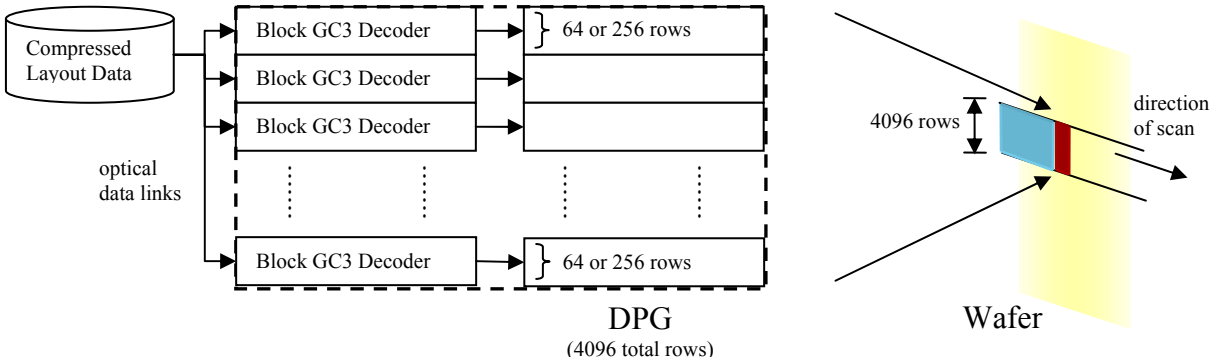


Fig. 4. The data-delivery path of the REBL system.

Table 1. Properties of the test layout images.

Image Size	64×1024, 64×2048, 256×1024, 256×2048
Pixel Value	0-31 (5 bit)
Tilting Angle	25, 35, 38

Each image pixel can take on one of 32 gray levels, in order to guarantee a 1 nm edge placement. In addition, due to the unique rotating writing stage of the REBL system, shown in Fig. 3, the layout images are rotated at arbitrary angles, ranging from 15° to 75°. In our testing set, we have collected layout images of three angles, as listed in Table 1. All the images have undergone E-beam proximity correction (EPC) compatible with the REBL system.

4. ADAPTING BLOCK RGC3 TO REBL DATA

In this section, we discuss the modifications that distinguish Block RGC3 from Block GC3. To ensure a feasible hardware implementation for the decoder, modifications have been added mainly to the encoding process, while keeping the decoding process as simple as possible. As shown in Sections 4.1 and 4.2, a diagonal copying algorithm and a smaller block size allow repetition in rotated layouts to be better exploited. A region-growing technique described in Section 4.3 adds spatial coherence to the segmentation values, increasing the compression efficiency.

4.1 Modifying the copy algorithm

Fig. 5(a) shows the Block GC3 encoding process as it progresses from left to right. A history buffer stores the most recently decoded pixels, as shown in the dashed region. The current block is encoded and decoded using a strictly horizontal or vertical copy distance. Fig. 6(a) shows an example of a 25°-rotated REBL layout image. Notice that repetition does not occur in either the horizontal or vertical direction for rotated layout images. Therefore, we need to modify the copy method to allow the decoder to copy from anywhere within the buffer range, at any arbitrary direction and distance, as shown in Fig. 5(b). This facilitates repetition discovery regardless of the layout's angle of rotation. Note that the buffered image area does not increase for diagonal copying; however, the number of possible copy distances to choose from has increased, thereby increasing the encode complexity, as discussed in Section 5.

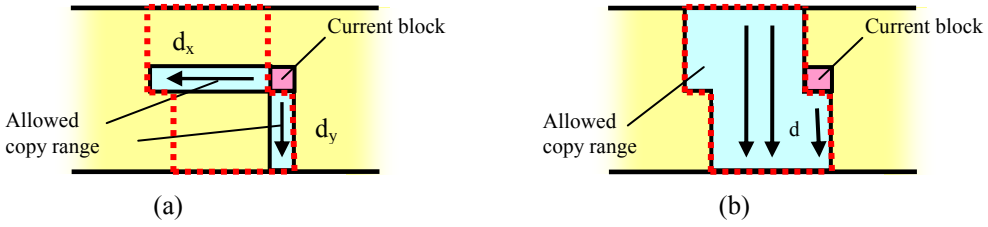
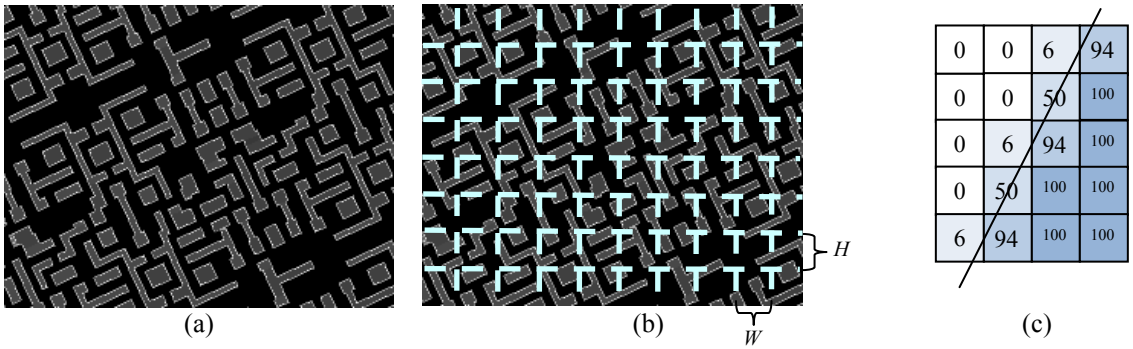


Fig. 5. Two copying methods: (a) Block GC3: only horizontal/vertical copying is allowed; (b) Block RGC3: blocks may be copied from anywhere within the search range. In both cases, the dashed areas must be stored in the history buffer.



(a) (b) (c)

Fig. 6. Layout image of the REBL system: (a) Original layout image; (b) fitting the image to an $H \times W$ block grid; (c) an example of image repetition, given an edge oriented at $\tan^{-1}(2)$ with respect to the pixel grid.

All results in Section 4 refer to 25°-oriented Metal 1 layout images, which are the most challenging to compress; a 40 KB buffer is assumed unless otherwise stated. A performance comparison between diagonal copying and the original horizontal/vertical copying is shown in Table 2. As seen, for a fixed buffer size, diagonal copying improves compression efficiency by 25-70%. As compared to horizontal/vertical copying, diagonal copying decreases image errors from 17.8% to 12.1% for a 1.7 KB buffer, or from 15.9% to 6.2% for a 40 KB buffer. In other words, diagonal copying significantly improves the ability to find repetition for arbitrarily-rotated layout images.

Table 2. Average compression efficiency comparison of two copy methods.

Image size	1.7 KB Buffer		40 KB Buffer	
	Hor. / Ver. Copying	Diagonal Copying	Hor. / Ver. Copying	Diagonal Copying
64×1024	3.12	3.89	3.35	4.91
64×2048	3.13	3.91	3.44	5.22
256×1024	3.19	3.96	3.36	5.60
256×2048	3.19	3.97	3.37	5.71

4.2 Decreasing the block size

Fig. 6(a) shows a typical rotated rasterized layout image of the REBL system. Although the image is visually repetitive, copy blocks closely approximating each $H \times W$ image block shown in Fig. 6(b) may not be found if the block size is too large. Thus, the block size should be sufficiently small to ensure that repetition in the image is fully exploited. However, as the block size decreases, the number of copy regions in the image tends to increase, requiring more segmentation information to be transmitted to the decoder. Specifically, reducing the block size from 8×8 to 4×4, using a 40 KB buffer, reduces the image error rate from 6.2% to 2.2%,[1] while increasing the number of segmentation errors by a factor of 2.6.

This latter effect is aggravated by the fact that rotated layout images introduce more copy regions than 0°-rotated layout images, thus decreasing the effectiveness of the segmentation prediction method of [4]. Fig. 6(c) shows a simple example of repetition in a rotated layout image. Each pixel value represents the percentage of the pixel that lies to the right of the diagonal "edge", which in this case is rotated by exactly $\tan^{-1}(2)$ with respect to the pixel grid. Note that all boundary pixels can be correctly copied using a (d_x, d_y) copy distance of (1, 2). Ignoring second-order EPC effects, the angle of rotation can generally be approximated as $\tan^{-1}(b/a)$, where integers a and b are as small as possible, given the approximation remains valid; this likely leads to $(d_x, d_y) = (a, b)$. If a or b are too large, however, implementing such a large copy distance may be infeasible, due to finite buffer size, finite image width, or simply a change in the local layout feature pattern; in this case, the best copy distance may change from one block to the next. This phenomenon becomes more pronounced as the block size is reduced. Fig. 7 shows a typical segmentation image for a 25°-rotated Metal 1

1 Note that a smaller block size also implicitly requires more buffers for the region decoder shown in Fig. 2, which stores segmentation information from the previous row of blocks in order to decode the compressed segmentation values [4]. A small block size leads to more blocks per row, which increases the required buffer size. However, compared with the size of the layout image's history buffer, especially under the 40 KB constraint, this change is negligible.

layout, where each pixel represents a 4×4 block and each copy distance is randomly assigned a different color. For each block, the first-discovered copy distance resulting in minimal image errors is chosen. This segmentation map looks fairly random, making it hard to compress. In particular, after applying the segmentation prediction method in [4], only 35% of the segmentation values in Fig. 7 are correctly predicted.

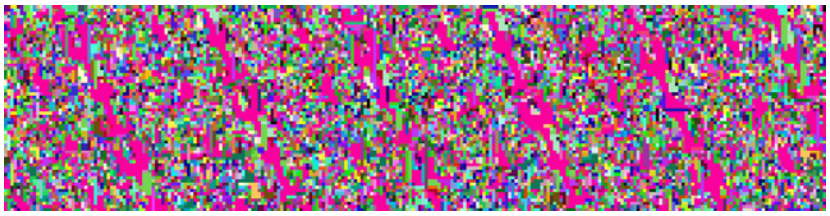


Fig. 7. Segmentation map of a 256×1024, 25°-oriented image.

In general, encoding the segmentation map is perhaps the most challenging part of compressing REBL data using Block RGC3, especially if a small block size such as 4×4 is used. Table 3 shows the percentage of each data stream after compression, using a 4×4 block size in Block GC3. As the buffer size increases, the number of image errors decreases; however, the higher number of possible copy distances for each block results in an increase of segmentation regions. Notice that segmentation information contributes up to 76% of the total compressed data, for a 40KB buffer size. The next subsection describes a more compression-efficient way of encoding this segmentation information. With this method in place, and assuming a square block size, we have empirically found that a 4×4 block size optimizes compression efficiency.

Table 3. Bit allocation of the Block GC3 compressed streams, using diagonal copying.

Buffer size	Image Error Map	Image Error Values	Seg. Error Map	Seg. Error Values
1.7 KB	28.7%	22.9%	5.2%	43.1%
20 KB	16.3%	10.9%	5.9%	66.8%
40 KB	14.5%	9.4%	5.9%	70.2%

4.3 Growing one-dimensional copy regions

As detailed in [13], we have empirically found that for all rotated layouts tested, the "copy" function consistently generates fewer image errors per block than the "predict" function. Thus, for the remainder of this paper, we assume all image blocks to be copied from previously-encoded blocks. In other words, each segmentation value represents a (d_x, d_y) copy distance. As implied by Table 3, maximizing compression efficiency involves minimizing both the number of image errors and the number of copy distance errors. The latter can be reduced by enforcing spatial coherence among the copy distances of neighboring blocks, as explained shortly. Unfortunately, these two metrics are not independent: minimizing image errors likely reduces the potential for spatial coherence, while enforcing maximum coherence increases image errors.

Enforcing spatial coherence in the segmentation map is made possible by the fact that multiple copy distances often lead to the minimum number of image errors for a given image block. This flexibility can be utilized by creating "regions" consisting of one or more adjacent blocks, each assigned the same copy distance. Regions can be grown in any preferred 2-dimensional way. The optimal region-growing metric is to minimize the total number of 2-D regions, assuming a known fixed number of image errors for each block. However, this problem is NP-complete, even if the number of image errors per block is already known, as we have shown in the Appendix. An alternative is to grow 1-D regions after first assuming each block contains minimal image errors. We have empirically determined that one-dimensional regions of size $1 \times N$ blocks result in higher compression efficiencies than polynomial-time 2-D region-growing heuristics. We refer to dimension N as the "length" of the region, which may be oriented either in the horizontal or vertical direction. Since the images associated with the REBL system may require a height as small as 64 pixels, we have chosen to grow regions horizontally in order to maximize N.

The following region-growing heuristic shown in Fig. 8 is solvable in polynomial time, and is proven to minimize the number of 1-D regions, if the number of image errors per block is first minimized [12]. Starting at the left of each image row, the longest possible region is assigned. This is iterated, region by region, until the entire row has been assigned:

Let M = image width / block width. Let $k = 1$.

For each non-overlapping $1 \times M$ block image row C_j
  For each block $B_i \in C_j$
    Determine S_i, the set of all copy distances which generates the minimum # image errors
  While ($k \leq M$)
    Find/assign the longest region in C_j with a common copy distance in $\{S_k,...,S_{k+N-1}\}$, $B_k \in C_j$
    $k = k + N$.

Fig. 8. Region-growing algorithm.

To encode the 1-D horizontal segmentation regions, we propose a new prediction method shown in Fig. 9(a), whereby each block's copy distance is predicted to be equal to that of the block to its left. This ensures that the (d_x, d_y) copy distance associated with a given horizontally-oriented region is only sent once. Specifically, only the left-most block in a region is predicted incorrectly. The prediction method for Block GC3 is shown in Fig. 9(b).

(a) (b)

Fig. 9. The block-based prediction method of (a) Block RGC3, (b) Block GC3.

4.4 Determining the optimal block shape

In Section 4.2, we discussed the tradeoffs of changing the block size; in this section, we discuss the optimal block aspect ratio. We assume that layout features are not preferentially oriented in either a horizontal or vertical direction, regardless of layout rotation. Regardless of the region-growing method utilized, we expect square-shaped segmentation regions to maximize the average region size, and therefore improve compression efficiency. This is because the longer dimension of an oblong segmentation region is likely to infringe on neighboring features, thus limiting repetition discovery and compression efficiency. In effect, square segmentation regions minimize the effect of this limitation.

Block GC3 uses square blocks [4] and grows regions in either a horizontal or vertical direction, thus resulting in approximately square-shaped segmentation regions. In contrast, Block RGC3 clearly grows regions using a directional preference, by using identical copy distances for neighboring blocks along the horizontal direction only. For a square block size, the average Block RGC3 region has a greater horizontal width than a vertical height. For example, assuming a 4×4 block size, the average Metal 1 region contains 2.7 blocks, which corresponds to a region size of 4×10.8. Since square-shaped segmentation regions are likely to optimize compression efficiency, one approach is to choose block shapes resulting in more or less square regions. Since Block RGC3 grows segmentation regions in a horizontal direction, this would mean that the block shape resulting in near-square segmentation regions is likely to have smaller width than height.

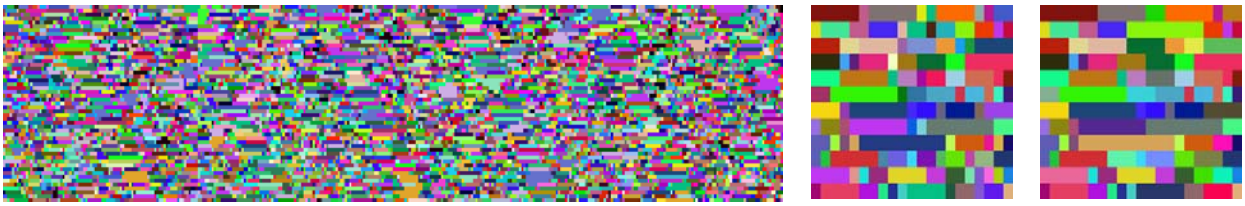
Table 4 compares compression and encoding time results for various block sizes, using the final Block RGC3 algorithm which includes the region-absorbing method discussed in Section 4.5. Two factors improve compression efficiency. First, the image error rate decreases as the block size decreases, as justified in Section 4.2; the resulting increase in blocks per image area also increases the encoding time. Second, a tall, narrow block aspect ratio such as 7×1 likely yields a more square-shaped average region, which, in turn, increases the average region size. As verified in Table 4, block shapes resulting in more square-shaped average regions are likely to have a larger average region size, and hence fewer regions, and hence a higher compression ratio.

For our final design, we have decided to use a 5×3 block size, due to its high compression ratio and low encoding time. Even though 7×1 and 6×1 block sizes slightly improve compression efficiency in the example shown, the encoding times for these are nearly twice as much as for 5×3; furthermore, the compression gains are lost if the buffer size is reduced or if a Via layer is compressed instead of Metal 1.

Table 4. Performance comparison of various block sizes, using a 1024×256 25°-oriented Metal 1 image and a 40 KB buffer.

Block Size	4×4	5×3	3×5	7x2	2x7	6×2	2x6	4x3	3x4	7x1	6x1
Average region shape (pixels)	4×10.8	5×9.3	3×12.6	7×7.1	2×15.3	6×7.9	2×15.0	4×10.5	3×12.4	7×6.6	6×7.5
Average region size (pixels)	43.2	46.5	37.8	49.7	30.6	47.4	30.0	42.0	37.2	46.2	45.0
Number of regions	6065	5572	6878	5185	8555	5412	8706	6198	7020	5557	5761
Image error rate (%)	2.67	2.60	2.64	2.70	2.83	2.41	2.55	2.32	2.37	2.08	1.84
Compression ratio	6.79	7.12	6.32	7.24	5.46	7.32	5.50	6.93	6.40	7.39	7.43
Encoding time (seconds)	182	180	189	191	213	209	230	202	226	295	334

Assuming a 4×4 block size and comparing with Section 4.1's results, growing regions as per Section 4.3 increases the number of correctly-predicted copy distances from 35% to 55%, while the image error rate remains unchanged at 2.18%. Now using a 5×3 block size, 59% of the blocks' copy distances are correctly predicted, while the image error rate is reduced to 2.09%. Thus, growing regions and utilizing a 5×3 block size both improve compression efficiency, as shown in Table 5. A sample of 25° Metal 1 copy regions is shown in Fig. 10(a) and (b), where each copy distance is randomly assigned a different color. The effectiveness of region-growing is loosely proportional to the number of candidate copy distances per block which generate minimal image errors. A cumulative distribution function of this is shown in Fig. 11; notice that 71% of blocks in the given layout have at least ten optimal copy distances, assuming a 40KB buffer size. As this number increases, the probability of finding a common copy distance among neighboring blocks also increases, thereby resulting in larger 1-D regions.



(a) (b) (c)

Fig. 10. (a) Segmentation map, utilizing 1-D region-growing; (b) Zoomed in portion of (a); (c) Regions after applying the region-absorbing method.

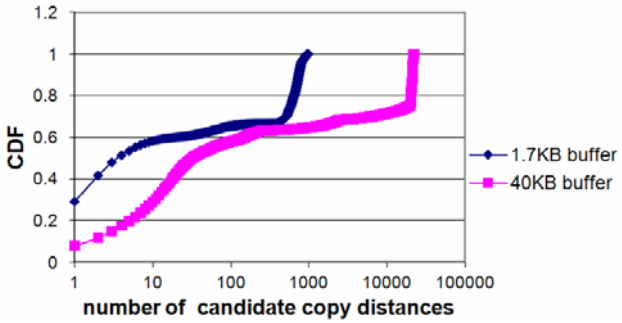


Fig. 11. Cumulative distribution function (cdf) of the number of copy distances per block which generate minimal image errors.

4.5 Combining neighboring regions

As mentioned in Section 4.3, maximizing the compression ratio requires a tradeoff between minimizing image errors and maximizing spatial coherence of copy distances. However, our 1-D region-growing method minimizes image errors as much as possible, at the expense of reducing spatial coherence. This constraint can be alleviated by adding a post-processing step, which combines neighboring regions while accepting a small penalty of image errors. The encoder judges this tradeoff using two parameters: α_1, the marginal decrease of transmitted bits resulting from one fewer

segmentation error, and α_2, the marginal increase of transmitted bits resulting from one additional image error. α_1 and α_2 are empirically calibrated, since these marginal changes depend on how effectively Huffman codes and Golomb run-length codes compress the segmentation/image error maps and values. For each image row, we apply the following heuristic to optimally combine the first five regions by finding combinations which maximize the expression: $\alpha_1 \times$ [# *regions absorbed*] $- \alpha_2 \times$ [# *image errors incurred*]; if no beneficial combinations are found, the set of five observed regions is shifted by three. We iterate on this until the entire row has been inspected. Region absorbing is only effective if $\alpha_1 > \alpha_2$; otherwise, a single additional image error more than negates the benefit of combining two regions. Empirically, we find $\alpha_1 = 14$ and $\alpha_2 = 10$ to optimize compression efficiency for 25°-rotated Metal 1, with a 40 KB buffer. By optimizing *five* regions and shifting by *three*, the new set of five regions includes two regions from the previous set; this allows nearly all beneficial absorptions of one or two regions to be carried out, including those straddling the boundary of each five-region set. Fig. 12 shows the pseudo-code for our proposed region-absorbing algorithm.

```
Let M = image width / block width.    Let k = 1.

For each non-overlapping 1×M block image row C_j, containing regions R_j,1 through R_j,m
    While (k < m)
        For the sub-row D_j,k = {R_j,k,…,R_j,k+4}
            For each region R_j,k+n ∈ D_j,k
                For each "optimal" copy distance for R_j,k+n, as determined in Fig. 8
                    List costs associated with absorbing one or more regions from D_j,k into R_j,k+n
                        - break if the number of image errors exceeds a known threshold
            If the "best" cost is beneficial, re-assign regions accordingly.
            Else, k = k + 3.
```

Fig. 12. Region-absorbing algorithm.

Fig. 10(c) shows how the regions shown in Fig. 10(b) have been modified by the region-absorbing algorithm. Specifically, the number of correctly-predicted copy distances has increased from 59% to 68%, but the image error rate has increased from 2.09% to 2.60%. This tradeoff has improved the compression efficiency overall, as shown in the right column of Table 5.

Table 5. Compression efficiency of Block RGC3, compared with other lossless compression algorithms.

	w/ 1-D region-growing, 4×4 Section 4.3		w/ 1-D region-growing, 5×3 Section 4.4		w/ region-absorbing, 5x3 Section 4.5	
Image size	Avg.	Min.	Avg.	Min.	Avg.	Min.
64×1024	5.73	5.53	5.87	5.70	6.19	6.00
64×2048	6.13	5.98	6.28	6.16	6.60	6.46
256×1024	6.51	6.50	6.76	6.75	7.12	7.12
256×2048	6.67	6.67	6.91	6.91	7.29	7.29

5. ENCODING COMPLEXITY

The total encoding time required to compress a given image is dominated by three main factors: (1) determining a list of optimal copy distances for each block, allowing diagonal copies, (2) choosing a copy distance for each block from this list such that the total number of regions is minimized, and (3) absorbing neighboring regions.

Contribution #1 essentially requires each image pixel to be compared with the pixel associated with each available copy distance. Thus, this portion of the encoding time is independent of the input image pattern. The image is both encoded and decoded in a column-by-column fashion. For Block RGC3, the number of possible copy distances per block is $d_{x,max} \times d_{y,max}$, where $d_{y,max}$ typically equals the height of the image and $d_{x,max}$ = buffer_size/$d_{y,max}$; in contrast, for Block GC3's horizontal/vertical copying, the copy candidate range is $d_{x,max} + d_{y,max}$. Due to extra computational overhead which is inversely proportional to the block size, we have empirically found encoding time to vary with $1/\beta + 1/(H \times W)$, where $H \times W$ represents the block size and $\beta \approx 10$. The β-dependent and block size-dependent factors equally affect encoding time when $H \times W = \beta$. Thus, Block RGC3 encoding time for a given image area is proportional to

$$O\left(d_{x,\max}d_{y,\max}\left(\frac{1}{\beta}+\frac{1}{HW}\right)\right)=O\left(buffer\_size\left(\frac{1}{\beta}+\frac{1}{HW}\right)\right).$$

Contribution #2 depends on $\overline{d_{matches,block}}$, the average number of copy distances per block resulting in minimal image errors. Contribution #3 depends on $\overline{N}$, the average number of blocks per region, and $\overline{d_{matches,region}}$, the average number of copy distances common to all the blocks in an entire region which result in minimal image errors. These three parameters depend on the layout image and the buffer size. Regions are grown by searching through each block's list of copy distances until the longest region is found. Similarly, regions are absorbed by applying each region's list of copy distances to its neighboring regions, to find the best possible absorptions. Encoding time increases as the block size decreases, since each block corresponds to a smaller image area. Thus, for a given image area, Contribution #2's encoding complexity is proportional to

$$O\left(\frac{\overline{d_{matches,block}}}{HW}\right),$$

while Contribution #3's encoding complexity is proportional to

$$O\left(\frac{\overline{d_{matches,region}}}{\overline{N}HW}\right).$$

Note that encoding time increases substantially if $\overline{d_{matches,block}}$ is very high; this typically occurs in images containing large amounts of background area, such as the Via layer.

Table 6 shows examples of software encoding times for various layouts and encoding parameters, using a 2.66GHz Intel Xeon processor with 2.75GB RAM. As expected, encoding times are proportional to image size and inversely proportional to block size. Notice that the easily-compressible via layer, composed of 96% background pixels, requires more time to encode because of its high $\overline{d_{matches,block}}$. Additionally, larger buffer sizes increase the number of possible copy distances per block; this increases $\overline{d_{matches,block}}$ and $\overline{d_{matches,region}}$, resulting in larger encoding times.

The encoding and decoding times for Block GC3 are shown in Table 7 as a comparison. The vast increase in Block RGC3 encoding times is due to the use of diagonal copying, which greatly increases the allowed copy range, as shown in Fig. 5. In the worst case, if $d_{y,max} = d_{x,max}$ = image height, diagonal copying increases the allowed copy range by a factor of approximately half the image height, in pixels, e.g., 32 or 128 for an image height of 64 or 256, respectively.

Table 6. Block RGC3 encoding times in seconds for various layouts, image sizes, block sizes, and buffer sizes.

			Contribution #1		Contribution #2		Contribution #3		Total encoding time	
Image size	Block size	Buffer size	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°
64×2048	5×3	20KB	35.3	35.4	1.6	5.4	0.2	0.6	37.0	41.4
256×1024	5×3	20KB	87.7	92.9	3.9	15.1	0.4	1.2	92.1	109.2
64×2048	5×3	40KB	63.6	66.8	2.4	11.0	0.2	0.9	66.2	78.7
256×1024	5×3	40KB	166.3	184.0	7.0	40.9	0.5	2.0	173.9	226.9
64×2048	8×8	20KB	28.2	28.4	0.2	2.5	0.2	0.8	28.6	31.7
256×1024	8×8	20KB	66.9	62.8	0.4	4.9	0.4	2.1	67.7	69.8
64×2048	8×8	40KB	49.1	49.1	0.3	4.3	0.2	1.2	49.7	54.6
256×1024	8×8	40KB	124.5	116.2	0.7	9.6	0.5	3.3	125.7	129.1

Table 7. Encoding and Decoding Times: Comparison between Block RGC3 and Block GC3.

			Encoding Time (sec) (Block GC3)		Encoding Time (sec) (Block RGC3)		Decoding Time (ms) (Block GC3)		Decoding Time (ms) (Block RGC3)	
Image size	Block size	Buffer size	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°	Metal 1 25°	Via 25°
64×2048	5×3	20KB	0.58	0.55	37.0	41.4	6.7	5.5	6.3	4.7
256×1024	5×3	20KB	0.69	0.63	92.1	109.2	16.0	11.8	12.5	12.7
64×2048	5×3	40KB	1.00	0.92	66.2	78.7	6.1	5.0	6.3	5.3
256×1024	5×3	40KB	0.92	0.88	173.9	226.9	13.7	11.2	12.0	10.8
64×2048	8×8	20KB	0.52	0.45	28.6	31.7	6.8	4.8	6.1	4.8
256×1024	8×8	20KB	0.53	0.47	67.7	69.8	13.8	10.2	11.7	10.8
64×2048	8×8	40KB	0.859	0.734	49.7	54.6	6.8	4.5	5.8	5.5
256×1024	8×8	40KB	0.766	0.672	125.7	129.1	14.7	10.8	11.6	10.6

6. COMPRESSION RESULTS

6.1 Block RGC3 Results

Table 8 shows the average compression efficiency for several layout samples of different buffer and image sizes. In total, three block sizes, three buffer sizes, three image sizes, and over four different layers were tested. Different wafer layers have significantly different compression ratios, ranging from 18.6 for the via layer, to 13.2 for the poly layer, to 7.3 for Metal 1. From most significant to least significant, the factors affecting compression efficiency are: wafer layer, buffer size, block size, image size, and angle of rotation.

Table 8. Average Block RGC3 compression efficiency for various layouts, image sizes, block sizes, and buffer sizes.

Image size	Block size	Buffer size	Metal 1			Metal 1b[2]	Poly	Via	
			25°	35°	38°	25°	35°	25°	35°
64×2048	5×3	1.7KB	4.92	5.37	5.14	--	8.49	13.14	12.67
256×1024	5×3	1.7KB	5.09	5.43	5.33	8.55	8.47	13.58	13.17
256×2048	5×3	1.7KB	5.10	5.45	5.35	--	8.51	13.62	13.22
64×2048	5×3	40KB	6.60	6.79	6.71	--	11.91	15.86	16.11
256×1024	5×3	40KB	7.12	7.23	7.34	14.87	12.80	17.05	17.27
256×2048	5×3	40KB	7.29	7.41	7.50	--	13.17	17.45	17.79
64×2048	4×4	1.7KB	4.93	5.38	5.22	--	8.64	13.21	12.88
256×1024	4×4	1.7KB	4.99	5.32	5.22	8.40	8.24	13.46	13.02
256×2048	4×4	1.7KB	5.01	5.33	5.23	--	8.29	13.54	13.10
64×2048	4×4	40KB	6.40	6.69	6.60	--	11.43	15.84	16.15
256×1024	4×4	40KB	6.81	6.98	7.05	14.29	12.09	16.76	16.94
256×2048	4×4	40KB	6.97	7.14	7.19	--	12.44	17.16	17.43
64×2048	8×8	1.7KB	3.97	4.55	4.47	--	7.61	13.54	13.24
256×1024	8×8	1.7KB	4.02	4.51	4.5	7.71	7.34	13.92	13.41
256×2048	8×8	1.7KB	4.03	4.52	4.51	--	7.37	13.96	13.48
64×2048	8×8	40KB	5.38	5.96	5.91	--	10.59	16.94	17.36
256×1024	8×8	40KB	5.75	6.26	6.30	13.66	11.52	18.09	18.48
256×2048	8×8	40KB	5.88	6.39	6.43	--	11.86	18.55	19.08

2 Only one 720×256 image is available for Metal 1b. The "Metal 1" and "Metal 1b" image samples are taken from different locations of the Metal 1 layer.

The 5×3 block size consistently outperforms the 4×4 block size, independent of the buffer size, the layout, or its angle of rotation. Note the 8×8 block size actually performs best for the via layer, because of its sparse pattern density.

The 38° Metal 1 layout has been tested in an effort to minimize pixel repetition, as per the $\tan^{-1}(b/a)$ approximation discussed in Section 4.2. Since $38° \approx \tan^{-1}(25/32)$, we expect the integers in this ratio to be too large for a copy distance of $(d_x, d_y) = (32, 25)$ to be utilized. However, due to REBL's E-beam proximity correction, repetition is often found using $(d_x, d_y) = (9, 7)$, where $\tan^{-1}(7/9)$ is a less accurate approximation for 38°. As a result, 38° compression results are not significantly different from results using rotations of 25° and 35°, which can be approximated by $\tan^{-1}(7/15)$ and $\tan^{-1}(7/10)$, respectively.

256×1024 images outperform 64×2048 images, partially due to their larger size. In addition, the buffer is more square-shaped for 256×1024 images, assuming the buffer size is greater than 10KB. This increases the overall likelihood of finding repetition at an arbitrary diagonal direction. While encoding the first blocks of an image, very few copy distances are available to choose from; this detrimental effect dies out as the history buffer becomes "full", and is less prominent for larger image sizes. Thus, 256×2048 images perform still better than 256×1024 images. In the REBL system, the expected image width approaches infinity, resulting in a further increase in compression efficiency.

6.2 Performance Comparison: Block RGC3 vs. Other Compression Algorithms

Table 9. Avg. compression efficiency for various layouts, image sizes, block sizes, buffer sizes, compression algorithms.

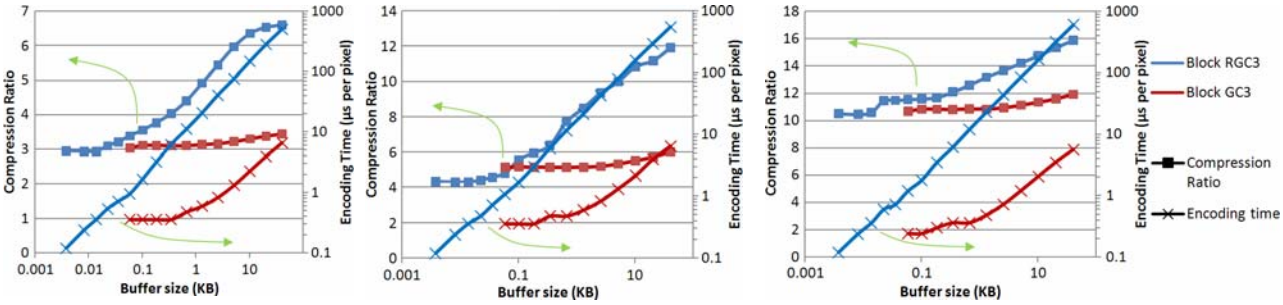
Image size	Block size	Buffer size	Metal 1, 25°					Via, 25°				
			Block RGC3	Block GC3	ZIP (30 KB)	BZIP2 (900 KB)	JPEG-LS (2.2 KB)	Block RGC3	Block GC3	ZIP (30 KB)	BZIP2 (900 KB)	JPEG-LS (2.2 KB)
64×2048	5×3	1.7KB	4.92	3.04	3.23	3.95	0.95	13.14	10.11	10.64	14.24	3.91
256×2048	5×3	1.7KB	5.10	3.05	3.43	4.68	0.97	13.62	9.88	11.68	15.98	4.03
64×2048	5×3	40KB	6.60	3.54	3.23	3.95	0.95	15.86	11.00	10.64	14.24	3.91
256×2048	5×3	40KB	7.29	3.42	3.43	4.68	0.97	17.45	10.26	11.68	15.98	4.03
64×2048	8×8	1.7KB	3.97	3.13	3.23	3.95	0.95	13.54	10.81	10.64	14.24	3.91
256×2048	8×8	1.7KB	4.03	3.19	3.43	4.68	0.97	13.96	10.79	11.68	15.98	4.03
64×2048	8×8	40KB	5.38	3.44	3.23	3.95	0.95	16.94	11.93	10.64	14.24	3.91
256×2048	8×8	40KB	5.88	3.37	3.43	4.68	0.97	18.55	11.34	11.68	15.98	4.03

Table 9 compares the compression efficiency of Block RGC3 with that of Block GC3, ZIP, BZIP2, and JPEG-LS, for both Metal 1 and Via layers [7][8][10][11]. Block RGC3 and Block GC3 are tested using buffer sizes of 1.7 KB and 40 KB, while ZIP, BZIP2, and JPEG-LS have constant buffer sizes of 30 KB, 900 KB, and 2.2 KB, respectively. In terms of compression efficiency, Block RGC3 consistently outperforms Block GC3, ZIP, and JPEG-LS, even using a 1.7 KB buffer size. BZIP2 outperforms Block RGC3 only when compressing the via layer using a 1.7 KB buffer. However, impractical hardware implementation and high buffer requirements prevent BZIP2 from being a practical solution.

Block RGC3 and Block GC3 compression efficiency and encoding complexity, as a function of buffer size, are compared in Fig. 13. Even though higher buffer sizes result in steadily higher compression ratios, this comes at a high price. First, larger buffer sizes result in memory occupying more decoder chip area, which is likely severely constrained to begin with. Second, Block RGC3 encoding time is linearly proportional to the buffer size. Although the encoding process may be completed offline in software, there are practical limits to how long this process may take. For example, using Block RGC3 with a 40KB buffer to compress a 20mm × 10mm 25° via layer in 64×2048 pixel segments would require 7.9 CPU years, assuming each pixel corresponds to a 22nm × 22nm area. If a 1.7KB buffer is used instead, computing time is reduced to 110 CPU days; this is clearly practical if several multiple-core processors are used in parallel.

As shown in Fig. 13, Block RGC3 is capable of producing significantly higher compression efficiencies than Block GC3. Also, for a given compression ratio, Block RGC3 requires less encoding time and buffering in almost every

instance. The only exception occurs when compressing Via and Poly using a 60 byte buffer size, at which Block RGC3 and Block GC3 have an identical 64×1-pixel copy range. At such a small copy range, region-growing is somewhat ineffective; thus, the main design distinction is Block GC3's 8×8 block size, which yields similar compression efficiency but requires four times lower encoding time than Block RGC3's 5×3 block size.



(a) (b) (c)

Fig. 13. Avg. compression ratios and encoding times vs. buffer size for 64×2048 (a) 25° Metal 1; (b) 35° Poly; (c) 25° Via.

The largest Block RGC3 buffer size shown in Fig. 13, 40 KB, allows each block to be assigned one of 2^{16} copy distances; the smallest buffer size represents the trivial case in which each pixel is always copied from the pixel immediately below it. For a given compression ratio, we have found that the encoding time can be reduced by roughly a factor of two with respect to Fig. 13 if the copy range is not searched thoroughly. Most of this reduction results from, whenever possible, blindly picking the first discovered copy distance resulting in zero image errors. Further reduction results from checking, at most, a small percentage of the available copy range, specifically those copy distances in close proximity to the copy range from Fig. 5(a), but rotated to match the wafer pattern's current angle of orientation.

6.3 Removing Huffman Coding from Block RGC3

If the complexity of the decoder's hardware is too high, because of area constraints or other factors, it may become necessary to simplify the compression algorithm. Block GC3 typically Huffman codes the image error values, as shown in Fig. 2 [9]. However, as shown in Table 10, we have found that eliminating Huffman coding reduces compression efficiency only slightly, i.e. 0-2%. Accordingly, removing the Huffman decoder from the decoder hardware reduces the area by 4.4% [4].

Table 10. Average Block RGC3 compression ratios, assuming a 40KB buffer, both with and without Huffman coding.

Image size	Block size	Use Huffman?	Metal 1			Metal 1b[2]	Poly	Via	
			25°	35°	38°	25°	35°	25°	35°
64×2048	5×3	Yes	6.60	6.79	6.71	--	11.91	15.86	16.11
256×1024	5×3	Yes	7.12	7.23	7.34	14.87	12.80	17.05	17.27
256×2048	5×3	Yes	7.29	7.41	7.50	--	13.17	17.45	17.79
64×2048	5×3	No	6.51	6.68	6.59	--	11.66	15.90	16.21
256×1024	5×3	No	7.02	7.11	7.20	14.71	12.48	17.05	17.33
256×2048	5×3	No	7.18	7.28	7.36	--	12.84	17.43	17.82

7. CONCLUSIONS AND FUTURE WORK

This paper has described Block RGC3, a lossless compression algorithm which optimizes compression efficiency for layouts compatible with the REBL direct-write E-beam lithography system. Layouts are assumed to be rotated at an arbitrary angle with respect to the pixel grid, and have undergone E-beam proximity correction compatible with the REBL system. Three main modifications to the original Block GC3 algorithm have been described. First, diagonal copying allows repetition of image data to be exploited, regardless of the layout's rotation. Second, a region-growing

method assigns equal segmentation values to neighboring blocks, reducing the impact of segmentation information on compression efficiency. The effect of block size and shape on the number of regions, image errors, and encoding time has been explored. Third, pixel-based prediction is removed, thus simplifying the decoder's hardware implementation without degrading performance. On average, Block RGC3 has compressed Metal 1 images by a factor of 7.3, assuming a worst-case rotation of 25°. This compares with a compression ratio of 3.3, if the original Block GC3 algorithm is used. The drawback of Block RGC3 is increased encoding time, which is roughly proportional to the allowed copy range.

For future research, it would be worthwhile to test our algorithm on a larger, more diverse set of layout images, in order to better characterize both compression efficiency and encoding complexity. In particular, Section 5 notes that portions of the encoding complexity are image-dependent. In order to bound the maximum encoding time, it may be necessary to set a hard limit on the size of $d_{matches,block}$.

In a practical implementation, controlling the position of the writer chip relative to the wafer to within 1 nm accuracy at all times may not be feasible. If so, this uncertainty dictates that layout data must be compressed in real-time through hardware, instead of being compressed in advance through software. Given the significant encoding complexity of Block RGC3, the process of exhaustively searching through the history buffer to find optimal copy distances for each image block likely must be restricted.

APPENDIX: NP-COMPLETENESS PROOF OF 2-D REGION-GROWING

Assume that each block is given a list of copy distances which yield no more than some pre-determined number of image errors, as described in Section 4.3. In this Appendix, we show that the process of choosing one copy distance from each block's list such that the total number of regions is minimized is NP-complete. We define a "region" as a group of 4-connected blocks each having the same copy distance. Let X_1 be an image with n optimal copy distances $D(p) = \{d_{1(p)}, d_{2(p)}, \ldots d_{n(p)}\}$ for all blocks $p \in X_1$. Our goal is to minimize the number of regions in X_1, such that each block in a given region has at least one copy distance in common. More formally, the following uniformity predicate holds:

$$U_1(X_1) = \text{ true iff } \forall p \in X_{1,i}, \exists\, a \text{ s.t. } a \in D(p),$$

where $X_{1,i}$ is any region in X_1. As proven by MIN2DSEG in [12], minimizing the number of 2-D regions in an image X_2 is an NP-complete problem, assuming the uniformity predicate

$$U_2(X_2) = \text{ true iff } \forall\, p,\, q \in X_{2,i},\ |I(p) - I(q)| \le 1,$$

where $I(p)$ are the block values for all blocks $p \in X_2$. For X_2, let $D(p) = \{I(p), I(p) + 1\}$. $U_1(X_2)$ and $U_2(X_2)$ are equivalent; this proves the reduction from MIN2DSEG in [12] to our 2-D region-growing method, which is thus NP-complete.

ACKNOWLEDGEMENT

This research was conducted under the Research Network for Advanced Lithography, supported by MICRO, KLA Tencor, and California State Micro award #07-027. We would also like to acknowledge Allen Carroll and Andrea Trave at KLA-Tencor for offering assistance with the REBL system and providing EPC layouts.

REFERENCES

[1] V. Dai and A. Zakhor, "Lossless Compression Techniques for Maskless Lithography Data", Emerging Lithographic Technologies VI, Proc. of the SPIE Vol. 4688, pp. 583–594, 2002.

[2] V. Dai and A. Zakhor, "Advanced Low-complexity Compression for Maskless Lithography Data", Emerging Lithographic Technologies VIII, Proc. of the SPIE Vol. 5374, pp. 610–618, 2004.

[3] V. Dai. "Data Compression for Maskless Lithography Systems: Architecture, Algorithms and Implementation," Ph.D. Dissertation, UC Berkeley, 2008.

[4] H. Liu, V. Dai, A. Zakhor, B. Nikolic, "Reduced Complexity Compression Algorithms for Direct-Write Maskless Lithography Systems," SPIE Journal of Microlithography, MEMS, and MOEMS (JM3), Vol. 6, 013007, Feb. 2, 2007.

[5] P. Petric et. al, "REBL Nanowriter: A Novel Approach to High Speed Maskless Electron Beam Direct Write Lithography," International conference on Electron, Ion, and Photon Beam Technology and Nanofabrication (EIPBN), 2008.

[6] P. Petric et. al, " Reflective electron-beam lithography (REBL)," Alternative Lithographic Technologies, Proc. of SPIE Vol. 7271, 2009.

[7] J. Ziv, and A. Lempel, "A universal algorithm for sequential data compression," IEEE Trans. on Information Theory, IT-23 (3), pp. 337–43, 1977.

[8] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," Technical report 124, Digital Equipment Corporation, Palo Alto CA, 1994.

[9] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," Proceedings of the IRE, 40(9), pp. 1098-1101, September 1952.

[10] 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. 1309–1324, 2000.

[11] M. E. Papadonikolakis, A. P. Kakarountas, and C. E. Coutis," Efficient high-performance implementation of JPEG-LS encoder," Journal of Real-Time Image Processing, 3, pp. 303-310, 2008.

[12] M. C. Cooper. “The Tractability of Segmentation and Scene Analysis”. International Journal of Computer Vision 30(1), pp. 27-42, 1998.

[13] G. Cramer, “Lossless Compression Algorithms for the REBL Direct-Write E-Beam Lithography System,” M.S. Thesis, UC Berkeley, 2009, http://www-video.eecs.berkeley.edu/papers/cramerg/ms-thesis.pdf.