electronics

MDPI

Article

Research on Parallel Reading and Drawing Techniques for Chemical Mechanical Polishing Simulation Data Based on Multi-Thread

Zhenyu Ji [1,2], Lan Chen [1,*], Yan Sun [1] and Hong Cai [1]

[1] The EDA Center, Institute of Microelectronics, Chinese Academy of Sciences, Beijing 100029, China; jizhenyu@ime.ac.cn (Z.J.); sunyan1@ime.ac.cn (Y.S.); caihong@ime.ac.cn (H.C.)
[2] The School of Integrated Circuits, University of Chinese Academy of Sciences, Beijing 100049, China
* Correspondence: chenlan@ime.ac.cn

Abstract: In advanced integrated circuit manufacturing, the quality of chemical mechanical polishing (CMP) is a key factor affecting chip performance and yield. Designers need to use CMP simulation tools to locate and analyze the defects in the layout after the CMP process. However, the advancement of process nodes and the increase in data volume presents a great challenge to the speed of graphical display of CMP simulation data. To solve this issue, we propose a solution that uses multi-threading technology to optimize both data reading and drawing. In the process of data reading, we employ OpenMP and memory mapping (Mmap) technology to achieve parallel reading of file segmentation and propose a fast-string conversion algorithm based on the properties of simulation data. In the process of data drawing, we propose an adaptive downsampling method for data graphical display that combines multi-threading and double buffering technology to enable the parallel drawing of layouts. The effectiveness of this method is verified by testing CMP simulation data of various scales. Compared to traditional methods, this approach improves reading efficiency by over 8 times and drawing efficiency by more than 10 times. Furthermore, it enhances the smoothness of interaction with the CMP simulation tool.

Keywords: chemical mechanical polishing; OpenMP; memory mapping; double buffering

check for updates

Citation: Ji, Z.; Chen, L.; Sun, Y.; Cai, H. Research on Parallel Reading and Drawing Techniques for Chemical Mechanical Polishing Simulation Data Based on Multi-Thread. *Electronics* **2024**, *13*, 706. https://doi.org/10.3390/electronics13040706

Academic Editor: Giulio D'Emilia

Received: 12 December 2023
Revised: 5 February 2024
Accepted: 7 February 2024
Published: 9 February 2024

Correction Statement: This article has been republished with a minor change. The change does not affect the scientific content of the article and further details are available within the backmatter of the website version of this article.

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Chemical mechanical polishing (CMP) is a super-precision polishing technology that combines both chemical and mechanical actions. It is widely employed to achieve global and local flatness on wafer surfaces. During the CMP process, the wafer rotates in relation to the polishing pad, applying a specific pressure and utilizing a polishing slurry. The removal of excess material from the wafer surface occurs through the mechanical grinding of abrasive particles and the corrosive action of chemical oxidants [1,2]. Due to the increasing complexity of integrated circuit manufacturing and stringent requirements for process deviation, the CMP process faces significant challenges. The chip surface morphology after the CMP process mainly depends on the layout characteristics of the chip. During the CMP process, especially in the copper interconnection process, the metal density distribution on the wafer surface is uneven, and the hardness of metal copper, diffusion barrier and dielectric is different, so that the chip surface is not completely flat during and after CMP. After CMP, dishing and erosion defects will appear on the wafer surface [3], as shown in Figure 1. Dishing defect refers to the difference between the thickness of the dielectric layer and the thickness of the metal layer in the graphics area; erosion defect refers to the difference between the thickness of the dielectric layer in the area without graphics and that in the area with graphics. These topologies will affect the subsequent lithography process, as well as seriously affect the RC parameters of the interconnects, ultimately damaging the production yield and electrical performance of the chip [4–6].

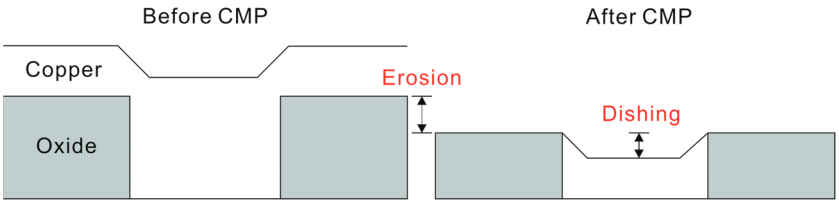


Figure 1. Dishing and erosion in CMP process.

In order to improve the chip yield and reduce the production cost, the CMP model is usually used to predict the surface morphology of the polished chip layout in advance [7]. The Art series CMP simulation tool, developed by the EDA Center of the Institute of Microelectronics, Chinese Academy of Sciences, can effectively simulate the CMP process and provide graphical representations of the simulation results. This tool greatly assists designers in conveniently and intuitively analyzing and identifying defects of layout surface morphology after CMP. The complete morphology prediction process of the CMP simulation tool can be divided into four steps: layout division, feature parameter extraction, CMP simulation and result output. The layout division is to divide the layout into a series of continuous grids. Due to the different surface height in different areas of the chip layout, the grid size needs to be appropriately set. The general principle for setting the grid size is to choose a size not smaller than the equivalent planarization length (EPL) of the layout [8]. The EPL refers to the minimum size of the impact of the flatness between the different patterns obtained from CMP experimental tests of the chip manufacturing process. In simple terms, it can be considered that within the region of the planarization length as the dimension, the layout patterns are approximately flat. In order to ensure the accuracy of the extracted layout pattern features, the grid division size is preferably smaller than or equal to the EPL of the chip layout. Although there are significant differences in the line width and density distribution of the interconnected patterns in the actual layouts compared to the overall chip layout size of millimeters or even centimeters, the grid size can range from 1 μm to 40 μm, with a preferred value of 20 μm. Feature parameter extraction is to extract the feature parameters of the layout structure in the grid, such as equivalent density, equivalent line width, equivalent spacing, and perimeter; CMP simulation is to use the extracted layout feature parameters as the input of the CMP model for the CMP process simulation; the result output means that the layout morphology information obtained by CMP simulation is saved in a text file, and then the CMP simulation results are further displayed graphically. The CMP simulation tool also supports hotspot detection in the layout and allows the hotspot detection results to be imported into other layout analysis software for subsequent processing and analysis, so that the analysis results feedback to the real CMP process to control the surface morphology of the chip after the real CMP process within an acceptable range. There are three different hotspot rules available for configuration:

1. The absolute value of dishing is greater than a certain value. The default value is 130 Å.
2. The absolute value of the difference between the thickness of Cu and the average thickness of Cu is greater than a certain value, and the default value is 100 Å.
3. The difference between the surface height and the average height is within a certain range. The default value is 130, which means that it is within ±130 Å.

Currently, parallel computing techniques are primarily utilized for large-scale data processing [9,10], and there are numerous ways to implement parallel computing in shared memory mode. The most straightforward approach involves the usage of multi-threaded programming [11,12]. This approach achieves parallel computation by assigning tasks and controlling calculations for each thread. However, its main drawback is its complexity and inflexibility, as it requires high programming skills. Another simpler method involves implementing parallel computing based on OpenMP [13–15], which is an application programming interface (API) for shared memory parallelization. OpenMP provides a set of compiler directives, environment variables, and runtime library functions for thread creation,

management, and synchronization. It achieves parallelism through the fork-join programming model. One advantage of OpenMP is that it allows users to gradually parallelize existing serial programs according to the standard. Therefore, we parallelized the serial program by utilizing OpenMP to achieve multi-threaded parallel reading and drawing.

The primary approach for improving the efficiency of processing large data files is to employ memory mapping (Mmap) technology. This method reduces input/output (I/O) operations by mapping all or part of the file content into the virtual memory of the process, which enables applications to directly access file data located on the disk through memory. Currently, research on Mmap primarily focuses on two aspects. One approach is to use Mmap technology in conjunction with corresponding algorithms to enhance the processing capacity of massive data [16,17]. The other approach is to utilize specific indexing methods to enhance the efficiency of Mmap in solving specific problems [18,19]. It is evident that previous research did not fully exploit the multi-core performance of computers.

Double buffering technology is commonly employed to address problems such as image flickering and tearing caused by frequent refreshing [20–22]. The basic concept involves not directly drawing images in the display memory (front buffer), but first drawing the images in a memory (back buffer), and subsequently copying the already-drawn images in the back buffer to the display memory. This not only significantly enhances the responsiveness of the graphical interface, but also facilitates multi-threaded drawing of CMP simulation data.

Based on the above analysis, this paper proposes a new method for parallel reading and drawing of CMP simulation data. The method is based on OpenMP technology and combines Mmap and double buffering technology. This approach optimizes string conversion and data storage performance during the reading process, and further enhances drawing efficiency through a uniform hierarchical downsampling method while ensuring display accuracy. Through testing and analysis of CMP simulation result files of different sizes, this method not only substantially reduces the time required for initial reading and drawing, but also improves the smoothness of interactive operations in the CMP simulation tool.

The subsequent sections of this paper are organized as follows. Section 2 analyzes the tasks of reading and drawing. Section 3 introduces the design of parallel reading and drawing. Section 4 presents the experimental results and discusses the findings. Section 5 concludes this paper.

2. Reading and Drawing Task Analysis

2.1. Reading Task Analysis

The CMP simulation process typically predicts the topography based on a grid; the result file also uses the grid as the unit to store the morphology information after CMP. A complete CMP simulation result file includes the post-CMP information for one or more layout layers, with the "BEGIN" and "END" fields marking the start and end of each layer of data. Each grid data is placed on a new line, where the first two columns indicate the coordinates of the grid, the third and fourth columns represent the high and low steps, and the last four columns provide the corresponding topography information for the grid, such as density, copper height, surface height, and oxide height. The format of the CMP simulation result file is illustrated in Figure 2. Each grid data is independent and does not rely on other data, allowing for high parallelization.

```
BEGIN metal1
0 0 -0.361616 0.000000 100.000000 0.126400 0.000000 0.126400
0 20000 -0.361616 0.000000 100.000000 0.126400 0.000000 0.126400
……
1000000 960000 -0.590404 0.000000 2.250000 0.074900 0.074800 0.081100
1000000 980000 -0.646970 0.000000 0.000000 0.069900 0.069900 0.000000
END metal1
BEGIN metal2
……
END metal2
```

Figure 2. Format of CMP simulation result file.

The process of reading the CMP simulation result file can be divided into four steps: (1) reading the string data; (2) splitting the string; (3) converting the split string into numerical values using string conversion functions; and (4) storing the values in memory. To optimize the reading process more effectively, we analyzed the time-consuming nature of these four steps. The test was conducted on a CMP simulation result file with a size of 423.7 MB, containing 6,000,384 grids. The test results, depicted in Figure 3, indicate that the majority of time is spent on converting floating-point values during the reading of CMP simulation result files. Therefore, it is necessary to further optimize the conversion of floating-point values, which is described in detail in Section 3.1.3.

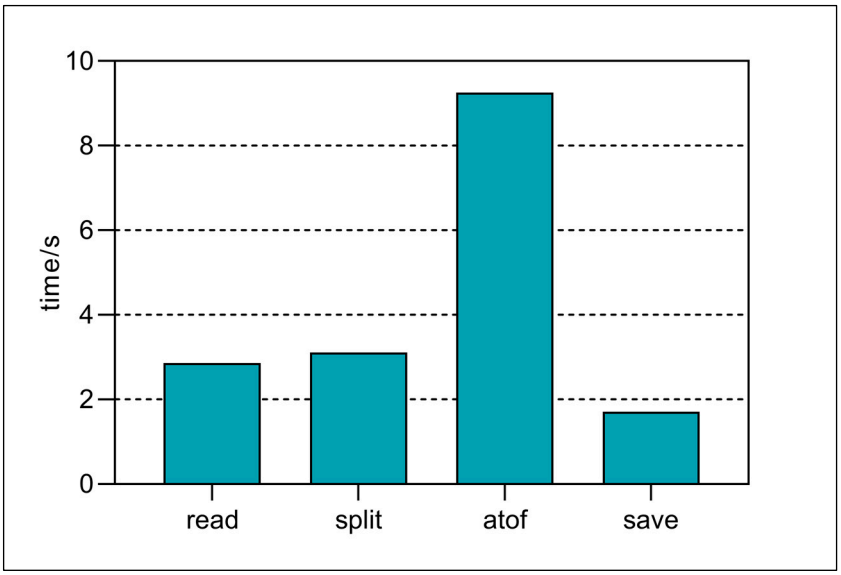


Figure 3. Time consumed in each step of the reading process.

2.2. Reading Task Analysis

The graphical display of CMP simulation data includes multiple layers of various topographical information, such as density, copper height, surface height, and oxide height. The drawing process begins at the coordinate (0,0) of the grid. By applying pre-specified numerical values and color mapping rules, the RGB color data corresponding to this grid can be obtained. The first column of grid data is then drawn from bottom to top, followed by a column-by-column drawing from left to right, thus achieving the complete color mapping of the entire topographical information. Throughout the drawing process, each grid data and each layout layer and each morphology information are independent of one another.

By default, the CMP simulation tool displays the surface height of the first layout layer. However, when users wish to view other morphology information, the simulation tool must redraw the corresponding layout morphology. This can lead to issues such as interface lag and graphics flickering. Additionally, excessively long redraw waiting times can seriously affect the user's interactive experience.

3. Optimization Scheme

3.1. Reading Optimization

We proposed the following read optimization scheme. Initially, the CMP simulation result file is mapped to the virtual memory of the application process using the Mmap technology provided by the QT platform. The number of file segments is determined based on the number of threads, so that each thread corresponds to a specific segment of the file. The OpenMP multi-thread parallel segmentation processing method is employed to enhance the efficiency of reading the CMP simulation result file. Additionally, a simplified string conversion function is designed taking into consideration the data characteristics of the CMP simulation result file. Finally, the data is stored in a designated fixed memory space.

3.1.1. Memory Mapping

Traditional file operations utilize the page cache mechanism [23] to enhance reading and writing efficiency and protect the disk. Consequently, when reading files, two data copying processes occur, firstly, the file is copied from the disk to the page cache, and then from the page cache to the memory. In contrast, Mmap only require a single data copying process from the disk to the user's memory, as illustrated in Figure 4. Particularly for large data files, Mmap technology exhibits higher efficiency.

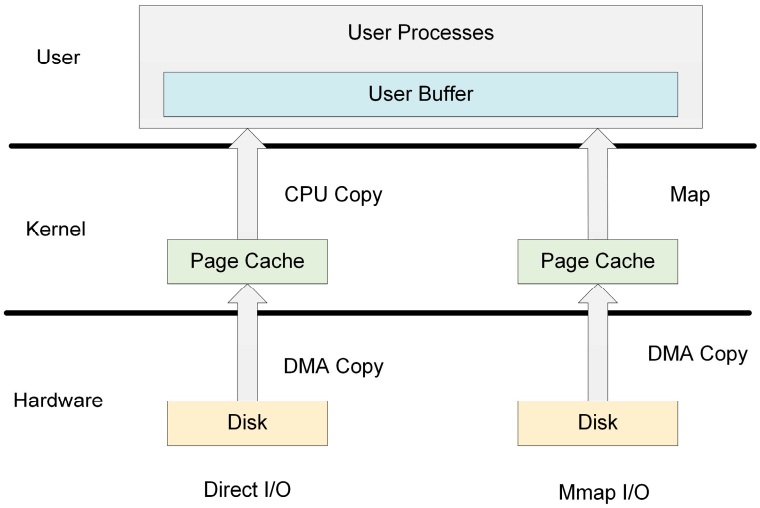


Figure 4. Comparison between traditional file reading and Mmap reading.

There are five key functions involved in using Mmap in Qt:

1. QFile::QFile(const Qstring &filename), used to create a file object.
2. Bool QFile::open(OpenMode mode), used to open the file.
3. Uchar *QfileDevice::map(qint64 offset,qint64 size, MemoryMapFlags flags = NoOptions), used to map the file. It is important to highlight that the "offset" parameter indicates the offset value of the data segment to be mapped in memory relative to the starting position of the file. The "size" parameter indicates the size of the data block to be mapped. File segmentation mapping can be achieved by setting different values for both the "offset" and "size" parameters.
4. Bool QfileDevice::unmap (uchar *address), used to unmap the file.
5. Void QfileDevice::close( ), used to close the file.

3.1.2. Multi-Threaded Reading

A good scheduling strategy can not only ensure load balancing between different CPUs, but also increase the utilization and parallel efficiency of system resources. OpenMP provides four different scheduling strategies: static, dynamic, guided, and runtime. The runtime scheduling strategy allows the specification of one of the other three scheduling strategies through environment variables during program execution. Therefore, this paper focuses on discussing the first three scheduling strategies. Based on their implementation principles, static scheduling reduces scheduling overhead, dynamic scheduling alleviates load imbalance, and guided scheduling aims to strike a balance between reducing scheduling overhead and alleviating load imbalance.

In CMP simulation result files, the grid serves as the minimum unit of partition. During reading and drawing tasks, the task size for each iteration remains constant. Therefore, we adopted static task scheduling to distribute N grids evenly among threads to achieve load balancing. Since the CMP simulation result file is a text file that undergoes file segmentation processing in terms of bytes, any byte error may lead to data reading errors. To ensure accurate data reading at the segment junction, the common segmentation approach involves making the character on the end point of each file segmentation the newline character

("\n"). The first segment begins at position 0, and the end is initially set to the average reading length. However, if the last character in the block data is not a newline character ("\n"), it indicates that the final grid data of the block is incomplete. In such cases, the file pointer is moved backwards until it points to the newline character ("\n"), obtaining the end position of the first file segment, also known as the start position of the second file segment. This process is repeated until all data in the file are read.

However, for subsequent segmentation drawing processes, it is crucial to ensure both data integrity and that the size of data contained in each file segment corresponds to an integer column of grid data. Consequently, we proposed a new file segmentation method.

1. Utilize Mmap to pre-read the CMP simulation result file, counting the number of newline characters ("\n") and recording the position of each newline character ("\n") in the file.
2. Find the number of key characters "begin" or "end" to get the number of layout layers, so as to get the total number of grids in the file. Extract the total number of grid rows by reading the first column of grid data, and then calculate the total number of grid columns.
3. According to the configured number of threads and the total number of grid columns, the grid data is allocated to each thread with the column as the smallest unit. Then, each thread calculates the starting position of the segment file based on the counted number of grids and the recorded newline characters positions. The end position of the previous file segment is the starting position of the next file segment.

While this file segmentation method introduces additional reading overhead, it eliminates the need for conditional checks during segmentation drawing. Furthermore, in files with multi-layer layouts, it ensures that the layout data of the same layer remains within a single thread. Most importantly, the memory size of grid data can be determined according to the number of grids obtained by file pre-reading. The data can be stored by allocating fixed memory, and the memory leakage problem can be avoided by releasing unnecessary memory space in time, so as to avoid performance issues caused by vector dynamic memory allocation [24].

3.1.3. String Conversion Performance Optimization

The string conversion functions in the C and C++ standard libraries are inefficient, resulting in a slow reading speed when reading massive layout and morphology data. The research indicates that these functions prioritize supporting various input formats and sacrificing efficiency to ensure data compatibility. However, the data format of the CMP simulation result file remains consistent; there is no need to spend a significant amount of time evaluating all possible scenarios for large amounts of single-type data. Therefore, we proposed a string conversion mechanism based on the characteristics of CMP simulation result file data. During the process of scanning the CMP simulation result text file byte by byte, the floating-point numbers are efficiently calculated by accumulating the integer and decimal parts, so as to complete the reading of the CMP simulation result file. The specific process is as follows.

1. Build a lookup table. Based on the data characteristics of the CMP simulation result file, we established a lookup table consisting of 10 rows and 7 columns denoted as $M[r][c]$, where r represents the row and c represents the column. The lookup table is shown in Table 1.
2. Scan the floating-point number string. By scanning the floating-point number string, denoted as S, we can identify the positive/negative sign, integer part, and decimal part. In this representation, S = 'sdndn-1...d1.x1x2...xm', where the positive/negative sign 's' can be empty or a minus sign, 'd1d2...dn' represents the integer part, 'x1x2...xm' represents the decimal part, and the decimal point separates the integer and decimal parts.
3. According to the positive/negative sign, integer part and decimal part of the floating-point string, combined with the lookup table, add it for the calculation to obtain the calculated value of the floating-point values. When the positive/negative sign 's' is empty or positive,

let $sign = 1$; when the positive/negative sign 's' is a negative sign, let $sign = -1$. Combined with reading the value of each element M in the comparison table, the integer part and decimal part of the floating-point string are added by the lookup table.

The integer part $T1$ is calculated as shown in the following equation:

$$\sum_{i=1}^{n} M_{(di+1),i} \tag{1}$$

The decimal part $T2$ is calculated as shown in the following equation:

$$\sum_{i=1}^{M} M_{(xi+1),(8-i)} \tag{2}$$

The calculated value of the floating-point number coordinate T is calculated as shown in the following equation:

$$T = sign \times \left(T1 + T2/10^7\right) \tag{3}$$

Table 1. Lookup table.

	1	**2**	**3**	**4**	**5**	**6**	**7**
1	0	0	0	0	0	0	0
2	1	10	1×10^2	1×10^3	1×10^4	1×10^5	1×10^6
3	2	20	2×10^2	2×10^3	2×10^4	2×10^5	2×10^6
4	3	30	3×10^2	3×10^3	3×10^4	3×10^5	3×10^6
5	4	40	4×10^2	4×10^3	4×10^4	4×10^5	4×10^6
6	5	50	5×10^2	5×10^3	5×10^4	5×10^5	5×10^6
7	6	60	6×10^2	6×10^3	6×10^4	6×10^5	6×10^6
8	7	70	7×10^2	7×10^3	7×10^4	7×10^5	7×10^6
9	8	80	8×10^2	8×10^3	8×10^4	8×10^5	8×10^6
10	9	90	9×10^2	9×10^3	9×10^4	9×10^5	9×10^6

The entire process primarily consists of addition operations and does not involve any function calls. This approach reduces the memory resources utilized during runtime and significantly enhances the efficiency of reading extensive topographic data. To convert the topography data, comprising of 6,000,384 grids, both the atof function and the customized function are employed. The time-consuming situation is shown in Figure 5. The test indicates that utilizing the self-developed simplified string conversion function results in a conversion efficiency improvement of more than 10 times.

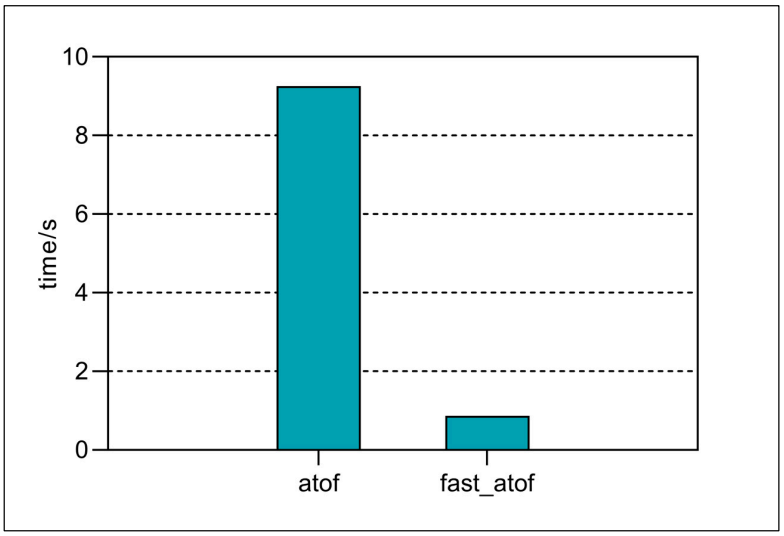


Figure 5. Comparison of efficiency between atof and fast_atof function.

3.2. Drawing Optimization

Currently, there are two main approaches to rapidly generate large-scale graphics. The first approach involves analyzing and simplifying the data to reduce the display scale [25–27]. The second approach utilizes computer software and hardware technology to enhance the computer's display capabilities [28–31]. We proposed a fast-drawing scheme for layout topography. Firstly, the layout surface topography data of various sizes are downsampled according to the display separation rate, thereby reducing the display size. Additionally, the efficiency of the drawing process is improved by employing a multi-threaded parallel drawing technique in conjunction with the double buffer mechanism.

3.2.1. Uniform Hierarchical Downsampling Method

Since the resolution of the computer display is limited, when the number of grids exceeds the number of pixels on the screen, attempting to draw large scale layout morphology graphics will result in a situation where less than one pixel of the display area corresponds to a grid. Therefore, it is necessary to use downsampling technology to simplify the graphics of the layout morphology. Downsampling techniques can be divided into uniform downsampling and non-uniform downsampling. By analyzing grid data characteristics, it is observed that all grids are uniformly distributed, and due to the grid size often being less than or equal to the EPL, the difference in surface morphology information between a certain grid in the layout and its adjacent grids is minimal. Displaying only one grid can effectively represent the morphology characteristics of the local area where this grid is located. Therefore, we used a uniform downsampling method to reduce the display scale. The key to uniform downsampling lies in determining the downsampling ratio. If the downsampling ratio is too small, the improvement in drawing efficiency will not be significant. However, if the downsampling ratio is set too large, it leads to a decrease in the accuracy of the display of the layout surface morphology graphics. Figure 6 shows the comparison of the layout morphology before and after uniform downsampling for a simulation result file of CMP with a file size of 423.7 MB, containing 6,000,834 grids. The downsampling ratio in Figure 6a is 1:1, which is the original layout surface morphology without downsampling. The downsampling ratio in Figure 6b is 9:1, the downsampling ratio in Figure 6c is 25:1, and the downsampling ratio in Figure 6d is 49:1. It can be observed that the characteristics of the layout surface morphology are well preserved after downsampling at a ratio of 9:1, and it is even difficult to notice any difference. However, when the downsampling ratio is 49:1, the overall characteristics of the layout surface morphology after downsampling are basically consistent, the display accuracy is significantly reduced, and many details are lost. Therefore, it is crucial to choose a reasonable downsampling ratio.

When users want to observe the details of a local area, they need to zoom in on the layout morphology graphics. Figure 7 shows the detailed comparison of the layout morphology before and after downsampling at a downsampling ratio of 9:1. It can be observed that even if the layout morphology graphics are downsampled at a small ratio, after being zoomed in to a certain degree, the detailed features will be lost, and the display accuracy will be significantly reduced.

Based on the above analysis, we proposed a hierarchical uniform downsampling method, which can take into account the efficiency of drawing and the display accuracy of graphics. The specific process is as follows:

1. Determine the initial downsampling level. Start by calculating the number of pixels N in the graphic display area based on the screen resolution. This will serve as the dividing criterion to create different intervals: [0, N], [N, 4N], [4N, 9N], …, $[(n-1)^2N, n^2N]$. Each interval corresponds to a downsampling level, which we label as L1, L2, L3, …, Ln. For each grid data segment at level Ln, begin from the bottom left corner of the topographic data and replace the $n \times n$ grid data with its central grid data. Repeat this process until all grids have been reduced.
2. Adjust the downsampling level according to the zoom level. Firstly, determine the maximum and minimum values of the zoom level. Then, divide the range of the scaling

level into n consecutive intervals, with each interval corresponding to a specific scaling level (S1, S2, S3, . . ., Sn). For the initial downsampling level Ln, no changes are made when the zoom level is S1. However, as the user zooms in, the downsampling level will decrease by one level for each increment in the zoom level, as depicted in Figure 8.

3. Obtain the segmented data by downsampling. Start by determining the number of grids after downsampling at each level, from initial level Ln to L1. Pre-allocate memory of the appropriate size for each level and obtain the segmented grid data using the corresponding relationships from Ln to L1.

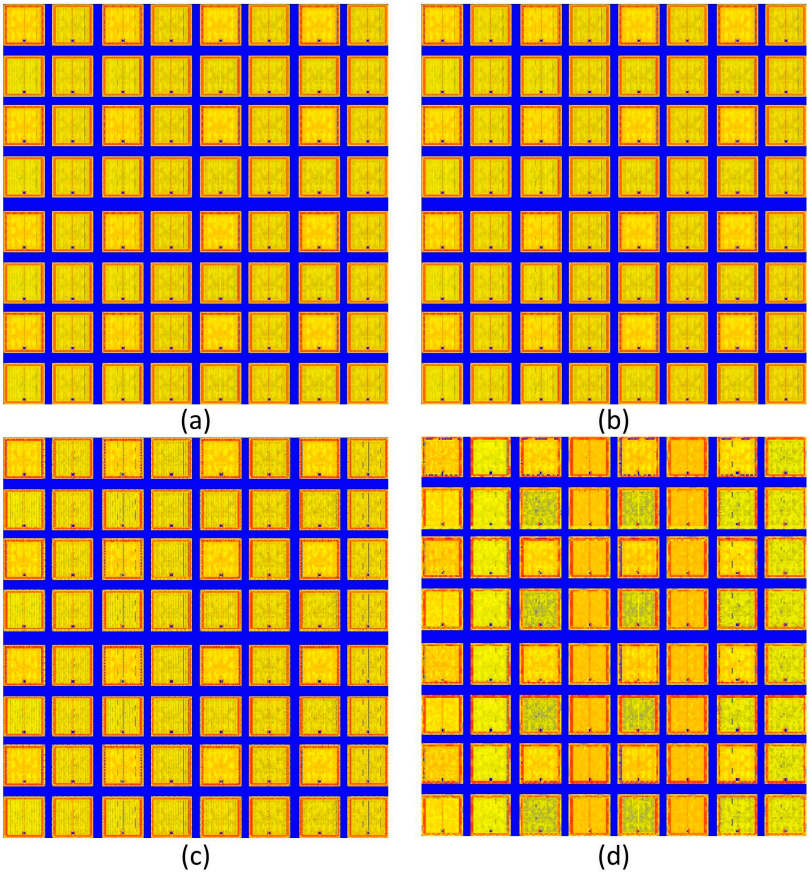


Figure 6. Comparison of layout morphology graphics at different downsampling ratios. (**a**) 1:1, (**b**) 9:1, (**c**) 25:1, (**d**) 49:1.

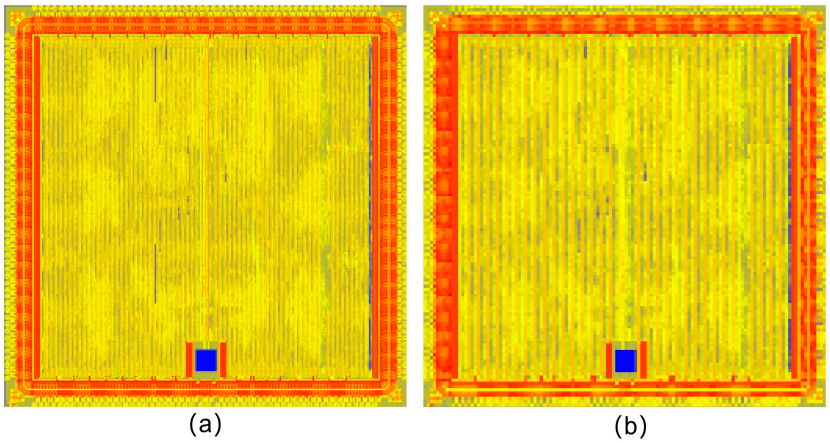


Figure 7. Comparison of local layout morphology graphic details before and after downsampling at ratio of 9:1. (**a**) Before downsampling, (**b**) after downsampling.

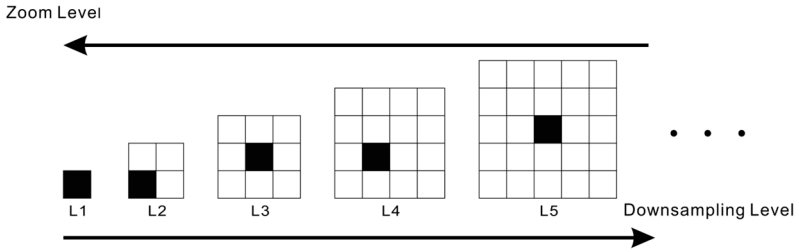


Figure 8. Downsampling rule.

3.2.2. Multi-Threaded Drawing

In Qt applications with a Graphical User Interface (GUI), the GUI thread serves as the main thread and the only thread capable of performing GUI-related operations. Qt provides two canvas class interfaces, QImage and QPixmap, each with its own set of functions. QImage is specifically designed and optimized for I/O, allowing for direct pixel access and manipulation. It can also be executed in non-GUI threads. When combined with a double buffering mechanism, QImage can significantly enhance the responsiveness of the graphical interface. QPixmap has made optimization in the realm of screen display imagery, making it a more suitable choice for graphic output due to its superior display effects. Furthermore, both the GUI thread and non-GUI threads are able to communicate with one another through the Qt signal and slot mechanism.

According to the analysis of the Qt drawing engine's characteristics and the drawing process of layout morphology described in Section 2.2, we proposed a fast parallel drawing method.

1. Parallelization of Drawing Threads. Each block of a complete morphology graph corresponds to a drawing thread. Each drawing thread primarily performs parallel operations, including data processing, g coordinate mapping, and drawing. When all threads finish drawing their block, the graphical composition thread combines them to form a complete image, marking the completion of a one-time drawing. This process is repeated until all graphical appearances are rendered.
2. Output of the graphics. When the drawing working thread completes the drawing process on the QImage drawing memory, it transfers the QImage memory block to the GUI thread. Then, it converts the QImage object into a QPixmap object. Finally, it utilizes the API function of the QT drawing engine to output the graphics. The process of graphics output occurs in two stages. Firstly, to enhance the efficiency of the initial display, the graphics of the surface height of the first layer of layout under the maximum downsampling level are first drawn during the memory drawing process. After the drawing is complete, the graphics are then transmitted to the GUI thread for output display. Secondly, when all the layout morphology graphics of different layers and morphology information under all downsampling levels are drawn, they are bundled together and delivered to the GUI thread. This approach enables quicker switching between different display contents or zooming by retrieving the corresponding QImage image and converting it to QPixmap format for display output. Notably, the conversion time is shorter than the time required for graphics drawing, resulting in a significant improvement in the smoothness of human–computer interaction. It should be noted that there is a time difference in the output of these two graphics. When the user switches to display graphics before the second graphics output, it can cause display failure or display errors. Therefore, we used the signal and slot mechanism to take the GUI thread receiving the drawn graphics set as the signal and clicking the switch display button as the slot to avoid this display error.

3.3. Overall Design

According to the above analysis, the overall design for parallel reading and drawing of CMP simulation data based on multi-threading, as proposed in this paper, is shown in Figure 9.

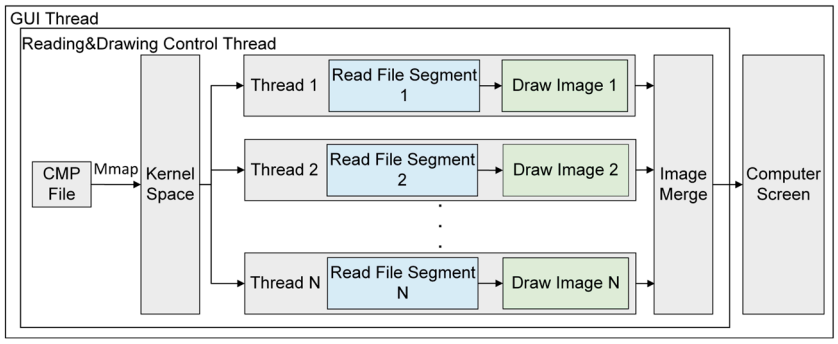


Figure 9. Parallel reading and drawing overall design for CMP simulation data.

Firstly, the values of the 'offset' and 'size' memory mapped file segmentation parameters are obtained through pre-reading. Each thread is then assigned the responsibility of reading data and drawing a file segmentation using the OpenMP parallel framework. Performance optimization is carried out for each step in the reading process. This involves reducing the number of file I/O operations by utilizing Mmap, improving the efficiency of string conversion through the use of a simplified string conversion function, and achieving data storage by opening up a fixed memory. In the drawing process, the data is adaptively downscaled to reduce the display scale. Subsequently, the background drawing of different downsampling level block shape graphics is accomplished by utilizing the double buffer mechanism. The control thread utilizes signal and slot mechanisms to control the process of multi-threaded reading and drawing, avoiding thread conflicts. For CMP result files with layout layers of m and a downsampling level of Ln, it is necessary to draw 4 × m × n layout morphology graphics. The graphics of the surface height of the first layer layout under the Ln level are drawn first to enhance the initial display speed. Once the background drawing of other morphology graphics is completed, the display can be switched according to the user's commands.

4. Discussion

4.1. Simulation Environment

This study was conducted using a workstation equipped with two Intel Core 12-core e5-2650 V4 2.2 GHz processors, 128 GB of memory, a display resolution of 1920 × 1080, the qt 5.6.1 drawing engine, Qt creator 4.3.0 as the programming environment, and C++ as the programming language. The simulation files used for data analysis had varying sizes, specifically 160.3 MB, 423.7 MB, 867.7 MB, and 3.4 GB. Table 2 shows the grid size, the number of horizontal and vertical grids and the grid number corresponding to four different sizes of files.

Table 2. Grid size and number information for different sizes of files.

File Size	Grid Size/μm^2	Number of Horizontal and Vertical Grids	Grid Number
160.3 MB	20 × 20	1503 × 1524	2,290,572
423.7 MB	10 × 10	2404 × 2496	6,000,384
867.7 MB	2 × 2	5751 × 2221	12,772,971
3.4 GB	1 × 1	5775 × 8653	49,971,075

4.2. The Optimal Number of Threads

Under the simulation environment described above, CMP simulation result files of sizes 160.3 MB, 423.7 MB, 867.7 MB, and 3.4 GB were tested with 1-48 test threads. The files were read and drawn using the method proposed in this paper. Figure 10 displays the test results of reading and drawing times for each file under different numbers of threads. The drawing time represents the time taken to draw all layout morphology graphics, including the various layout layers and morphology information across all downsampling levels. From the figure, it is evident that the change trend of reading and drawing times for files of different sizes under different threads is largely similar. Initially, the reading and drawing time decreases rapidly as the number of threads increases. However, after 12 threads, no further acceleration is observed. This is because, with an increasing number of threads, the time required for block data merging and image block merging also increases, offsetting the acceleration effect brought about by segmented reading and block drawing. Thus, the optimal number of threads for the reading and drawing scheme is determined to be 12.

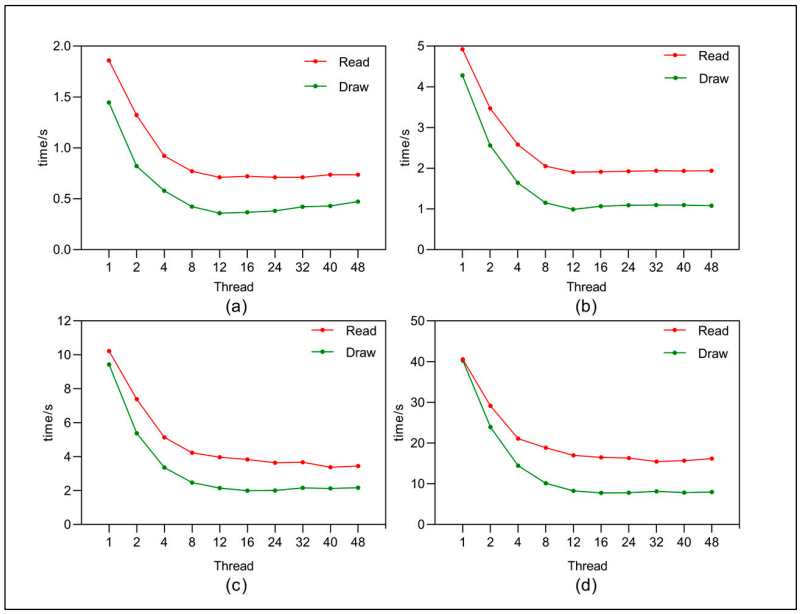


Figure 10. The reading and drawing time of files of different sizes under different thread numbers. (**a**) 160.3 MB, (**b**) 423.7 MB, (**c**) 867.7 MB, and (**d**) 3.4 GB.

4.3. Comparison of Reading Efficiency

Four different reading methods, including single thread, single threaded Mmap, single threaded Mmap and optimized string conversion function and data storage performance, multi-threaded Mmap and optimized string conversion function and data storage performance, were used to sequentially read files. The number of threads was set to 12, and the time taken to read the files of four different sizes using each method was recorded and is presented in Table 3 and Figure 11. The timing started at the QFile::open of the pre-read process and ended when each thread completed reading the segmented file and combined the data.

Table 3. Reading time of files of different sizes for the four proposed methods.

File Size	Reading Time/s *			
	Single Thread	Single Thread + Mmap	Single Thread + Mmap + Optimize	Multi-Thread + Mmap + Optimize
160.3 MB	6.36	4.89	1.61	0.68
423.7 MB	16.72	12.01	4.23	1.82
867.7 MB	33.52	21.79	9.51	4.50
3.4 GB	133.09	80.12	38.95	17.04

* The reading times are the average values measured from 10 tests.

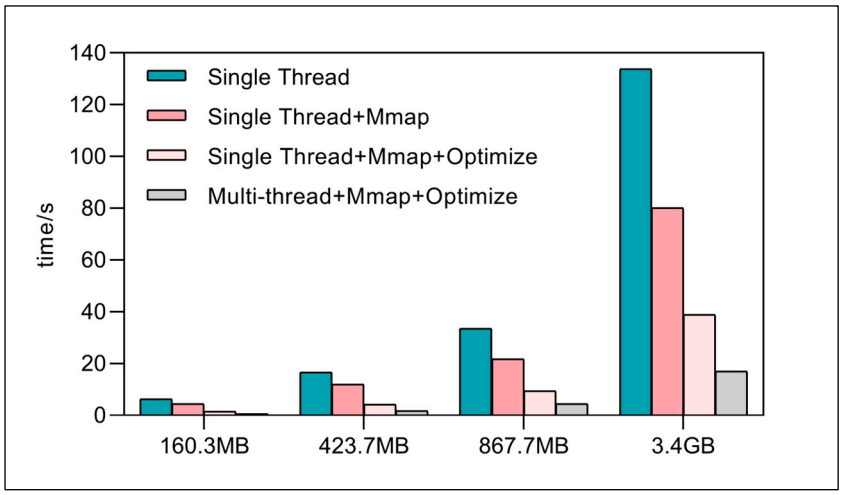


Figure 11. Reading time of files of different sizes for the four proposed methods.

First, by comparing the reading time of ordinary single thread and single threaded Mmap, it can be observed that the use of Mmap technology can effectively improve the file reading speed. The reading efficiency of four different sizes of test files using Mmap is increased by 30%, 39%, 52% and 67%, respectively. It can be found that the larger the file, the greater the performance improvement brought by Mmap technology. This is because the Mmap reading process eliminates one copy of data compared with the traditional reading process. The larger the file, the greater the efficiency improvement brought by reducing one copy of data. After optimizing the string conversion function and data storage performance, it can be found that the efficiency of the optimized single-threaded Mmap is further improved compared to the unoptimized single-threaded Mmap, with an efficiency increase of more than 100%. On this basis, using multi-threading technology increases the reading efficiency by more than twice compared to the optimized single-threaded Mmap. Finally, by comparing the ordinary single-threaded and the optimized multi-threaded Mmap schemes, the effectiveness of the proposed integrated parallel reading optimization scheme is validated. When reading large size CMP simulation result files, the reading scheme proposed in this paper, which combines Mmap technology, multi-threading technology and optimized string conversion and data storage technology, achieves an efficiency improvement of more than 8 times compared to the traditional single-threaded method.

4.4. Comparison of Drawing Efficiency

Sequential drawing of files using three different drawing methods, single-threaded drawing, multi-threaded drawing, and multi-threaded drawing after data downsampling, with the number of threads set to 12. The time taken to draw a layout surface topography pattern of four different sizes using each method was recorded and presented in Table 4 and Figure 12. The timing started with setting the downsampling level and ended with refreshing the surface height of the first layer of the layout on the screen.

Table 4. Drawing time of files of different sizes for the three proposed methods.

File Size	Drawing Time/s *		
	Single Thread	Single Thread + Downsample	Multi-Thread + Downsample
160.3 MB	2.77	0.76	0.23
423.7 MB	7.34	1.03	0.67
867.6 MB	16.32	1.66	0.96
3.4 GB	56.89	1.97	1.08

* The drawing times are the average values measured from 10 tests.

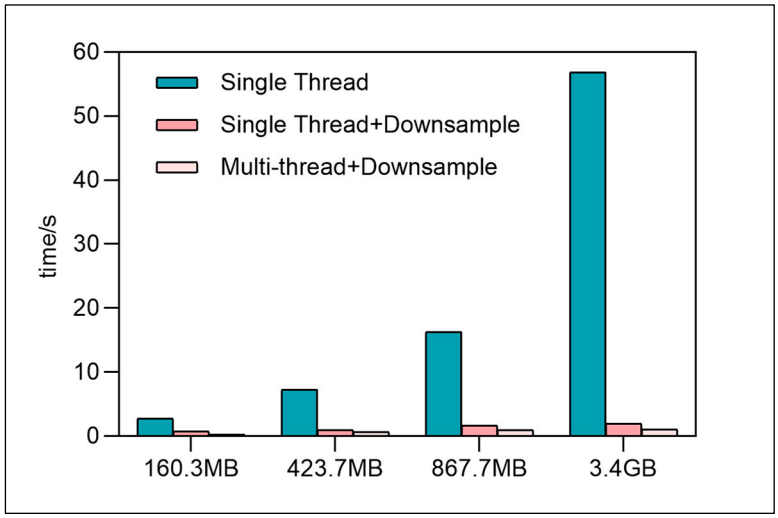


Figure 12. Drawing time of files of different sizes for the three proposed methods.

Comparing the drawing time of single thread and single thread+downsampling, it is found that downsampling technology can significantly improve the efficiency of drawing. The drawing efficiency of four test files with different sizes after downsampling is increased by 2 times, 6 times, 8 times and 27 times, respectively. The larger the file size, the greater the efficiency of downsampling drawing methods. This is because according to the set downsampling rules, the downsampling levels of the four files are L2, L3, L4 and L8, respectively, that is, the number of grids to be drawn after downsampling the four files is reduced to 1/4, 1/9, 1/16 and 1/64 of the original. The larger the file size, the greater the downsampling ratio, and the greater the performance improvement brought by downsampling. The number of grids drawn after downsampling is close to the number of screen pixels, and the drawing time tends to be the same. On the basis of downsampling, utilizing multi-threaded technology to draw the surface morphology of the layout further improves the drawing efficiency. Based on the above analysis, the drawing scheme in this paper combines downsampling and multi-threading technology. Compared with the traditional single thread drawing, the efficiency is improved by at least 10 times, and with the increase in file size, the drawing efficiency will be far greater than 10 times.

5. Conclusions

This paper presents a solution to address the issue of slow reading and drawing speed when dealing with large-scale layout CMP simulation data. The proposed method entails a fast parallel reading and drawing approach for CMP simulation data. In the reading process, multi-threading technology and Mmap technology are utilized to segment the file. Additionally, the string conversion function and data storage performance are further optimized, combined with the data characteristics of CMP simulation results. For the drawing process, a double buffered drawing mechanism is employed to achieve complete separation between foreground display and background drawing. Building on this, the background drawing time is further reduced through display scale reduction and the adoption of a multi-threaded block drawing technique. The experimental results demonstrate that compared to traditional methods, the reading and drawing methods proposed in this paper achieve a reading efficiency improvement of over 8 times and a drawing efficiency improvement of over 10 times. Furthermore, the smoothness of operations when users switch between displaying and scaling different morphology information is also optimized.

Author Contributions: Conceptualization, Z.J.; methodology, Z.J. and L.C.; software, Z.J.; validation, Z.J., Y.S. and H.C.; formal analysis, Z.J.; investigation, Z.J. and H.C.; writing—original draft preparation, Z.J.; writing—review and editing, Y.S. and L.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program of China, grant number 2021YFB3600301.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tsujimura, M. The way to zeros: The future of semiconductor device and chemical mechanical polishing technologies. *Jpn. J. Appl. Phys.* **2016**, *55*, 6S3. [CrossRef]
2. Ghulghazaryan, R.; Wilson, J.; Abouzeid, A. FEOL CMP modeling: Progress and challenges. In Proceedings of the 2015 International Conference on Planarization/CMP Technology, Chandler, AZ, USA, 30 September–2 October 2015; pp. 1–4.
3. Zhang, J.H.; Huang, H.; Greene, A.M.; Xie, R.; Seo, S.C.; Montanini, P.; Tseng, T.W.; Tsai, S.; Malley, M.; Fang, Q.; et al. CMP challenges for advanced technology nodes. *MRS Adv.* **2017**, *2*, 2361–2372. [CrossRef]
4. Xie, X. Physical Understanding and Modeling of Chemical Mechanical Planarization in Dielectric Materials. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2007.
5. Ma, T.; Chen, L.; Fang, J. Study of Optimal Dummy Fill Modes in Chemical–Mechanical Polishing Process. *IEEE Trans. Compon. Packag. Manuf. Technol.* **2012**, *2*, 1043–1047. [CrossRef]
6. Tugbawa, T.E.; Park, T.H.; Boning, D.S. Integrated chip-scale simulation of pattern dependencies in copper electroplating and copper chemical mechanical polishing processes. In Proceedings of the IEEE 2002 International Interconnect Technology Conference (Cat. No.02EX519), Burlingame, CA, USA, 3–5 June 2002.
7. Ruan, W.; Chen, L.; Ma, T.; Fang, J.; Zhang, H.; Ye, T. Optimization of a Cu CMP process modeling parameters of nanometer integrated circuits. *J. Semicond.* **2012**, *33*, 086001. [CrossRef]
8. Bao, H.; Chen, L.; Ren, B. A Study on the Pattern Effects of Chemical Mechanical Planarization with CNN-Based Models. *Electronics* **2020**, *9*, 1158. [CrossRef]
9. Chen, X.; Wang, Y.; Yang, H. Parallel Circuit Simulation on Multi/Many-core Systems. In Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, Shanghai, China, 21–25 May 2012; pp. 2530–2533.
10. Ye, X.; Dong, W.; Li, P.; Nassif, S. Maps: Multi-algorithm parallel circuit simulation. In Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, USA, 10–13 November 2008; pp. 73–78.
11. Berger, E.D.; Yang, T.; Liu, T.; Novark, G. Grace: Safe multithreaded programming for C/C++. In Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, New York, NY, USA, 25–29 October 2009; pp. 81–96.
12. Sharif, K.H.; Zeebaree, S.R.M.; Haji, L.M.; Zebari, R.R. Performance measurement of processes and threads controlling, tracking and monitoring based on shared-memory parallel processing approach. In Proceedings of the 2020 3rd International Conference on Engineering Technology and its Applications (IICETA), Najaf, Iraq, 6–7 September 2020; pp. 62–67.
13. Gepner, P.; Kowalik, M.F. Multi-core processors: New way to achieve high system performance. In Proceedings of the International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, 13–17 September 2006; pp. 9–13.
14. Bucker, H.M.; Lang, B.; Rasch, A.; Bischof, C.H.; Mey, D. Explicit loop scheduling in OpenMP for parallel automatic differentiation. In Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications, Moncton, NB, Canada, 16–19 June 2002; pp. 121–126.
15. Li, J.; Shu, J.; Chen, Y.; Wang, D. Analysis of factors affecting execution performance of openMP programs. *Tsinghua Sci. Technol.* **2005**, *10*, 304–308. [CrossRef]
16. Lin, Z.; Kahng, M.; Sabrin, K.M.; Chau, D.H.P.; Lee, H.; Kang, U. Mmap: Fast billion-scale graph computation on a pc via memory mapping. In Proceedings of the 2014 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 27–30 October 2014; pp. 159–164.
17. Van Essen, B.; Hsieh, H.; Ames, S.; Pearce, R.; Gokhale, M. DI-MMAP—A scalable memory-map runtime for out-of-core data-intensive applications. *Cluster Comput.* **2015**, *18*, 15–28. [CrossRef]
18. Choi, J.; Kim, J.; Han, H. Efficient Memory Mapped File {I/O} for {In-Memory} File Systems. In Proceedings of the 9th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage'17), Santa Clara, CA, USA, 12–17 July 2017; p. 5.
19. Song, N.Y.; Yu, Y.J.; Shin, W.; Eom, H.; Yeom, H.Y. Low-latency memory-mapped i/o for data-intensive applications on fast storage devices. In Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, UT, USA, 10–16 November 2012; pp. 766–770.
20. Lu, H.; Cai, X.; Zhang, Y.; Fei, X. Visualize traffic data using double buffer graphics. In Proceedings of the 2012 2nd International Conference on Remote Sensing, Environment and Transportation Engineering, Nanjing, China, 1–3 June 2012; pp. 1–4.
21. Sheeparamatti, R.B.; Sheeparamatti, B.G.; Bharamagoudar, M.; Ambali, N. Simulink model for double buffering. In Proceedings of the 32nd Annual Conference on IEEE Industrial Electronics (IECON 2006), Paris, France, 6–10 November 2006; pp. 4593–4597.
22. Seng, D.; Wang, H. Realistic real-time rendering of 3D terrain scenes based on OpenGL. In Proceedings of the 2009 First International Conference on Information Science and Engineering, Nanjing, China, 26–28 December; pp. 2121–2124.

23. Rajgarhia, A.; Gehani, A. Performance and extension of user space file systems. In Proceedings of the 2010 ACM Symposium on Applied Computing, New York, NY, USA, 16–22 March 2010; pp. 206–213.
24. Katajainen, J.; Simonsen, B. Adaptable component frameworks: Using vector from the C++ standard library as an example. In Proceedings of the 2009 ACM SIGPLAN Workshop on Generic Programming, Edinburgh, UK, 31 August–2 September 2009; pp. 13–24.
25. Li, S.; Marsaglia, N.; Garth, C.; Woodring, J.; Clyne, J.; Childs, H. Data reduction techniques for simulation, visualization and data analysis. *Comput. Graph. Forum* **2018**, *37*, 422–447. [CrossRef]
26. Heok, T.K.; Daman, D. A Review on Level of Detail. In Proceedings of the International Conference on Computer Graphics, Imaging and Visualization (CGIV 2004), Penang, Malaysia, 26–29 July 2004; pp. 70–75.
27. Agrawal, R.; Kadadi, A.; Dai, X.; Andres, F. Challenges and opportunities with big data visualization. In Proceedings of the 7th International Conference on Management of computational and collective intElligence in Digital EcoSystems, New York, NY, USA, 25–29 October 2015; pp. 169–173.
28. Ma, K.L.; Parker, S. Massively parallel software rendering for visualizing large-scale data sets. *IEEE Comput Graph Appl.* **2001**, *21*, 72–83. [CrossRef]
29. Yu, H.; Ma, K.L.; Welling, J. A parallel visualization pipeline for terascale earthquake simulations. In Proceedings of the 2004 ACM/IEEE Conference on Supercomputing (SC'04), Pittsburgh, PA, USA, 6–12 November 2004; p. 49.
30. Lindstrom, P.; Pascucci, V. Visualization of large terrains made easy. In Proceedings of the Visualization, San Diego, CA, USA, 21–26 October 2001; pp. 363–574.
31. Ahrens, J.; Brislawn, K.; Martin, K.; Geveci, B.; Law, C.C.; Papka, M. Large-scale data visualization using parallel data streaming. *IEEE Comput Graph Appl.* **2001**, *21*, 34–41. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.