

國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering  
College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

頻譜診斷與微顯影平行計算

Spectrum Diagnosis and Parallel Optical Simulation



指導教授：陳中平 博士

Advisor: Charlie Chung-Ping Chen, Ph.D.

中華民國九十九年一月

January, 2010

國立臺灣大學碩士學位論文  
口試委員會審定書  
頻譜診斷與微顯影平行計算  
Spectrum Diagnosis and Parallel Optical Simulation

本論文係黃冠儒君 (R96943021) 在國立臺灣大學電子工程學系、  
所完成之碩士學位論文，於民國 99 年 1 月 27 日承下列考試委員審查  
通過及口試及格，特此證明

口試委員：

許叮  
(指導教授)

洪士毅

王倫

蔡錫河

系主任、所長

Y. S. E

# Spectrum Diagnosis and Parallel Optical Simulation


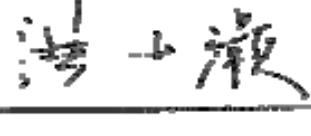
By  
Kuan-Lu Huang

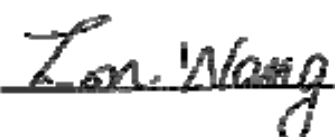

## THESIS

Submitted in partial fulfillment of the requirement  
for the degree of Master of Science in Electronics Engineering  
at National Taiwan University  
Taipei, Taiwan, R.O.C.

Jan. 2010

Approved by :

Advised by :



Approved by Director :



# 誌謝

回首這兩年半來的碩士生活，感謝指導教授查理博士的細心教導讓我得以一窺 EDA 領域的深奧，並不時的指點我給與正確的方向。在博理 405 實驗室裡共同的生活點滴，學術上的討論或者是茶餘飯後的消遣，感謝眾多學長姐的共同砥礪，才有這樣多采多姿的生活。感謝 Tin 學長、Lulu 學姊、劉繼蔚學長、斯鐸學長、忠杰學長、宗昱、盟竣、仕杰學弟，有你們的幫忙，本論文才能完成。還有陪我走過無數個假日的青年公園幫：石頭、達賴、康康、冠瑜、阿多、老大、阿諾、興仔，與你們一起打球的日子才使我更有動力往前走。以及大學同學：大頭、小曼、跳蛋，有你們在大學時期的互相砥礪，才得以順利考取研究所。表哥政佑的經驗分享與陪伴讓我受益匪淺。女朋友晴馨在背後的默默支持，體諒、包容更是我繼續往前走的動力。最後，僅以此論文獻給我的雙親與姊姊。



# 摘要

為了保證奈米壓印製造光柵的品質，光散射(optical scatterometry) 是一個有效率和有效的方法來診斷實際光柵的幾何形狀。為了方便診斷的過程，一個有效率針對大型資料庫的匹配演算法是非常重要的。在本篇論文中，我們提出一個有效的演算法利用最小誤差(MSE)的方式用來比對大型的頻譜資料庫，藉此反推原始的幾何組態。我們利用奇異值分解(Singular Value Decomposition)對大型的資料庫作壓縮並使用分層的動差(Moment)匹配方式來執行匹配演算法。我們的搜尋和診斷演算法是非常快速且精確的。跟傳統的最小誤差比起來，快上了 3000 倍以上且精確度在 0.1%以內。



第二部分是介紹使用平行計算的方式來加快微顯影中的成像生成。隨者超大型積體電路技術的特徵尺寸(feature size)迅速縮小，已小於曝光光的波長，光的繞射效應使得曝光後的圖像明顯偏離了原本設計的光罩。因此，微顯影結果的品質，在超大型積體電路(VLSI)的製造過程中是非常重要的。但是往往花費了相當多的時間來產生成像。在論文中，我們使用 CUDA 技術，它是一個通用的平行計算架構，充分利用在 NVIDIA 繪圖晶片(GPU)中的平行計算引擎，用來加快微顯影中的圖像生成。

**關鍵字:** 光散射、奇異值分解、動差匹配、阿貝成像方法、繪圖處理器平行運算

# Abstract

To ensure the quality of the nanoprint fabricated optical gratings, optical scatterometry (OS) is an efficient and effective mean to diagnose the actual fabricated geometry. To facilitate the diagnosis process, efficient pattern matching algorithms over a huge database are of great importance. In this thesis, we propose an efficient algorithm using minimum error square approach used to matching in a huge simulated spectrum database in order to obtain the original geometric configuration inversely. We use Singular Value Decomposition to do compression on large database and the use of hierarchical moment to perform matching algorithm; our searching and diagnosis algorithm is extremely fast and accurate. It is over 3000x faster than a exhausted searching algorithm within 0.1% accuracy.

The second part is to introduce the use of parallel computing in the imaging of microlithography for acceleration. As the VLSI technology feature sizes quickly shrink smaller than the wavelength of exposure light sources, the diffraction effects have made the exposed patterns significantly deviated from the original intended mask pattern. Therefore, the quality of microlithography simulation is an important part of the VLSI manufacturing process. However, it takes considerable time to produce image. In the thesis, we use CUDA, which is a general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA graphics processing units (GPUs) to speed up the image generation in Microlithography simulation.

**Keywords:** *Optical Scatterometry, Singular Value Decomposition, Moment Matching, Abbe's method, CUD*

## Content

中文口試委員審定書 .....	i
英文口試委員審定書 .....	ii
致謝 .....	iii
摘要 .....	iv
Abstract .....	v
Content .....	1
List of Figures.....	4
List of Tables .....	6
Chapter 1. Efficient Ways for Optical Scatterometry Diagnosis.....	7
1.1. Introduction and Background.....	7
1.2. Previous Work .....	8
1.3. Motivation .....	9
1.4. Spectrum Diagnosis Scheme .....	10
1.4.1. Database Construction.....	11
1.4.2. Problem Formulation.....	12
1.4.3. Eigenvalue Decomposition.....	13
1.4.4. Compaction by Singular Value Decomposition.....	15
1.4.5. Classification.....	18
1.4.6. Segmented Moment Matching.....	21
1.4.7. Grid-based Searching .....	27
1.5. Approach to Arbitrary Segment Spectrum Range .....	32
1.6. Simulation Results.....	34
1.7. Runtime Comparison.....	39

1.8.	Summary.....	40
1.9.	Feature Work .....	40
Chapter 2.	Parallel Optical Simulation Using Graphic Processor Unit.....	41
2.1.	Introduction and Background.....	41
2.2.	Motivation .....	41
2.3.	Preliminary .....	42
2.4.	Overview of Image Generation .....	43
2.4.1.	Imaging Equation.....	43
2.4.2.	The Abbe's Method.....	45
2.4.3.	Mask Decomposition.....	48
2.4.4.	Lookup Table.....	49
2.4.5.	Image Generation Flow.....	53
2.4.6.	Mask Pattern Partition.....	54
2.5.	Parallel Optical Simulation.....	55
2.5.1.	Code Analysis.....	56
2.5.2.	The Evolution of Microprocessor.....	58
2.5.3.	Multi-core Background Overview .....	58
2.5.4.	NVIDIA CUDA.....	60
2.5.5.	Architecture of Tesla C1060.....	61
2.5.6.	Software Model .....	64
2.5.7.	Approach to Parallel Lookup Table .....	67
2.5.8.	Kernel Execution Flow .....	68
2.5.9.	The Optimization on Our Work.....	69
2.6.	Performance Comparison .....	72



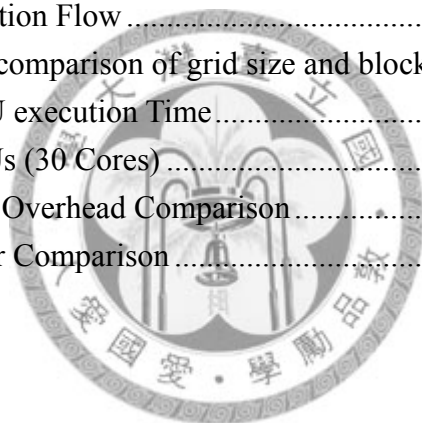
2.6.1. Grid size and Block size .....	73
2.6.2. Execution Time Comparison.....	74
2.6.3. Transmission Overhead.....	77
2.6.4. Relative Error .....	78
2.7. Summary.....	79
2.8. Feature work .....	80
Bibliography .....	81
Appendix A.....	85



## List of Figures

Figure 1-1 Optical Scatterometry Experimental Step.....	9
Figure 1-2 The spectrum diagnosis scheme.....	10
Figure 1-3 Three kinds of grating shapes in our simulation.....	11
Figure 1-4 The parameter of the grating.....	11
Figure 1-5 Database construction by R-Soft simulation .....	12
Figure 1-6 Problem formulation .....	13
Figure 1-7 Problem formulation .....	14
Figure 1-8 Singular Value Decomposition .....	15
Figure 1-9 Database compaction .....	16
Figure 1-10 Singular value distribution of simulated reflected spectrum .....	17
Figure 1-11 The meanings of moments .....	18
Figure 1-12 Variance varying according to shape variation .....	19
Figure 1-13 Skewness varying according to shape variation .....	20
Figure 1-14 Segmented Spectrums with moment computations .....	20
Figure 1-15 Grid storage structure in moment spaces.....	21
Figure 1-16 Grid storage structure cell in 3D moment spaces .....	21
Figure 1-17 Coded grid.....	22
Figure 1-18 Distribution of V space .....	24
Figure 1-19 Disperse the data uniformly according to distribution of V space.....	25
Figure 1-20 Perform SVD to 3D grid structure.....	26
Figure 1-21 Schematic diagram of the distance between grid.....	27
Figure 1-22 None-homogeneous grid.....	29
Figure 1-23 Relative to the grid structure of the correspond tree.....	30
Figure 1-24 The upper bound from observation points.....	31
Figure 1-25 Compensation for a particular spectrum.....	33
Figure 1-26 Direct Searching (MSE) without compact database .....	35
Figure 1-27 Direct searching (MSE) in compact database.....	36
Figure 1-28 Grid-based searching in compact database.....	37
Figure 1-29 The input reflected spectrum with 0.5% Gaussian noise on grid-based searching	38
Figure 1-30 Run time comparison Direct search VS Grid Search.....	39
Figure 2-1 Coherent illumination .....	43
Figure 2-2 Discretization of a conventional partially coherent illumination system	46

Figure 2-3 Mask pattern Decomposition .....	49
Figure 2-4 Convolution by pre-computed lookup table .....	51
Figure 2-5 A delta function convolutions with kernel function.....	52
Figure 2-6 Example for the center of convolution .....	53
Figure 2-7 Image Generation Flow .....	54
Figure 2-8 Mask Partition.....	55
Figure 2-9 Even-bit CSG circuit of 32-bit adder .....	56
Figure 2-9 Even-bit CSG circuit of 32-bit adder .....	57
Figure 2-10 GPU throughput exceeded the CPUS .....	59
Figure 2-11 The GPU devotes more transistors to data processing.....	61
Figure 2-11 The CUDA hardware model .....	62
Figure 2-13 The CUDA software model .....	65
Figure 2-14 The CUDA warp scheduler .....	66
Figure 2-15 The memory hierarchical on CUDA.....	66
Figure 2-16 Parallel computing on look-up table.....	68
Figure 2-17 Kernel Execution Flow .....	69
Figure 2-18 Performance comparison of grid size and block size .....	74
Figure 2-19 GPU VS CPU execution Time.....	75
Figure 2-20 CPU VS CPUs (30 Cores) .....	76
Figure 2-21 Transmission Overhead Comparison.....	78
Figure 2-22 Relative Error Comparison.....	79



## List of Tables

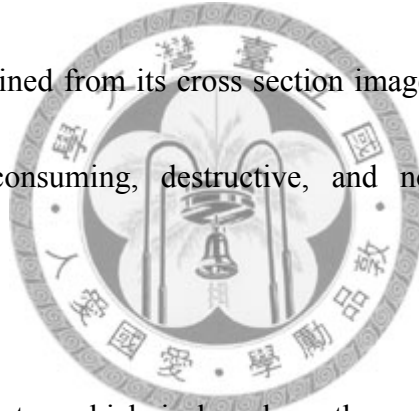
Table 1.1 The SPEC of simulation step .....	35
Table 1.2 Searching time comparison.....	38
Table 2.1 The test case of Even-bit CSG of 32-bit Adder .....	56
Table 2.2 The profile output of optical simulation .....	57
Table 2.3 Hardware information of Tesla C1060 .....	63
Table 2.4 Memory addressing of CUDA.....	67
Table 2.6 Circuit size of our simulation .....	75
Table 2.7 Performance comparison between GPUs and CPUs .....	76
Table 2.8 GPU overhead comparison .....	78



# Chapter 1. Efficient Ways for Optical Scatterometry Diagnosis

## 1.1. Introduction and Background

Since the linewidths of most holographic gratings in modern applications are less than 100nm, few inspection tools are available. Conventionally, the scanning electron microscope (SEM) is the powerful tools to characterize such microstructures. With SEM, which is regarded as the most intuitive means of microstructure measurement, the grating profile can be determined from its cross section images. However, this method is local, expensive, time consuming, destructive, and not extendable to online monitoring.



In recent years, scatterometry which is based on the measurement of diffraction efficiencies or polarization responses and encompasses reflectometry, ellipsometry, and diffractometry, has become popular and widely accepted for accurate grating topography extraction [1-5]. Such kinds of techniques aim to solve inverse diffraction problems. From a practical point of view, the relationships between grating profiles and their diffraction responses can be found. Using the well developed grating theory, the curve of any diffraction response versus grating parameters can be calculated. For the

inverse grating problem, an analytical function is unavailable for data got from complicated numerical calculations. The functions are nonlinear, and the variables are complexly related from each other. The approaches have been developed the finite element method to solve the inverse grating problem. Since this approach fundamentally differs from the usually scatterometric approach, we do not discuss it.

## 1.2. Previous Work

There're mainly two approaches to solving inverse grating problems: the look-up table method [2] and the nonlinear regression method [6-8]. The minimum square error method, which uses huge library with specialized searching algorithms, is a powerful tool for multi-parameter grating profile measurement. Niu et al. [1] used this method to measure CDs of integrated circuit with pseudoperiodic structures. In 1993, Krukar et al. [6] used an artificial neural network (ANN) model to simulate the relationship between the reflection at a fixed wavelength and the CDs of gratings with a trapezoidal profile, but the method is not economical for large scale fabrications because it is difficult to construct model. Although the look-up table method wastes the huge cost in computing time and storage space. However, it's higher accuracy than ANN.

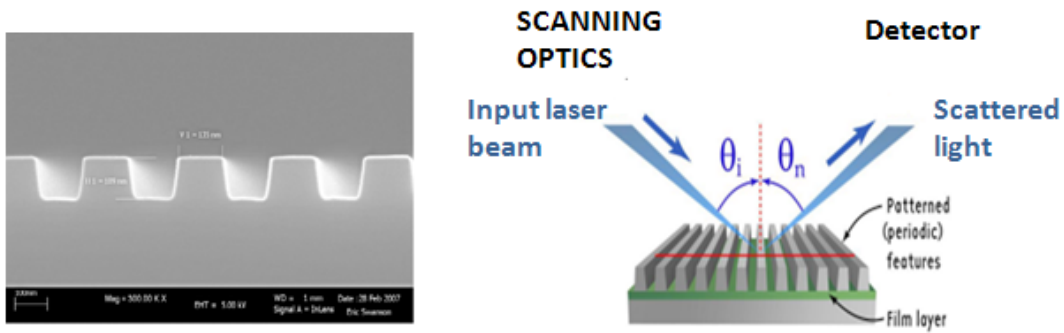


Figure 1-1 Optical Scatterometry Experimental

### 1.3. Motivation

In our work, we focus on huge library. We improve the minimum square error method [1]. First, the compact database of reflected spectrums needs to be built. Second, we have to reduce searching time to find the match geometry of grating. We proposed an efficient Grid-based searching with almost  $O(1)$  time complexity. Reserve only 1% data size by Segmented Moment Hierarchical SVD. It's over 3000x faster than a exhausted searching within 0.1% accuracy.

## 1.4. Spectrum Diagnosis Scheme

The spectrum diagnosis scheme involves three phases (Figure 1-2). The first step is to collect all reflected spectrums by R-Soft simulation to establish database. The second step is to perform Singular Value Decomposition (SVD) to compact the huge database and segment the spectrum of zones. Third, feature extraction, the moments distributions are different between square and trapezoid gratings. Hence, we take advantage of this feature as classification. Finally, given an unknown reflection spectrum as an input with the Grid-based searching, we can find the most possible grating. In this work, I devote to the SVD Compaction and Grid-Based Searching.

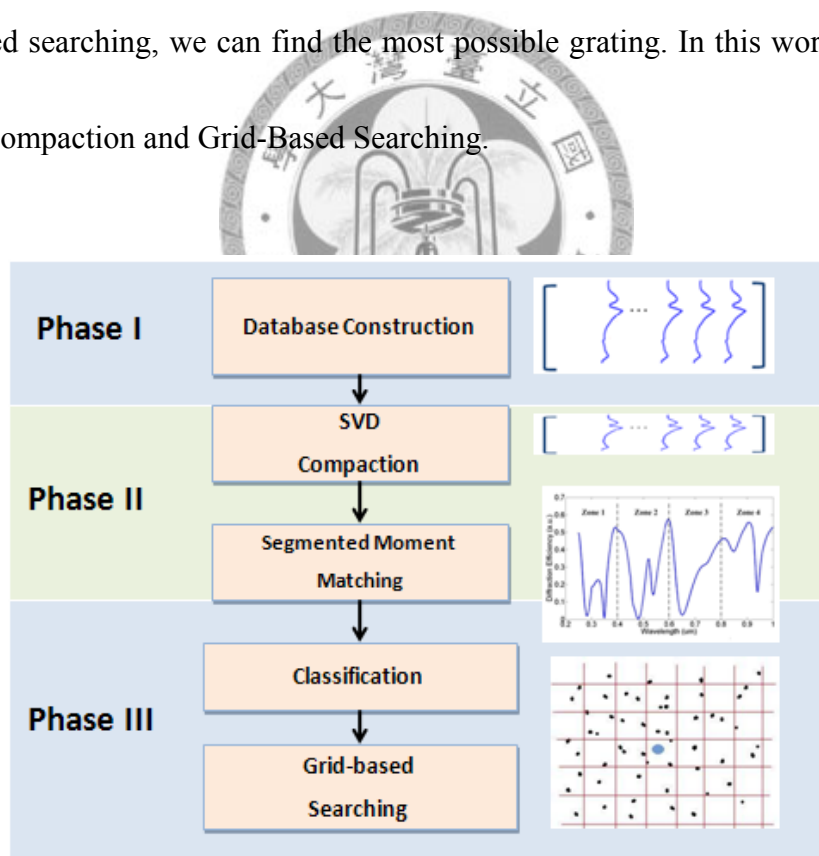


Figure 1-2 The spectrum diagnosis scheme



### 1.4.1. Database Construction

There are three kinds of grating shapes in our simulation: Rectangle, Upside down Trapezoid and Trapezoid (Figure 1-3).

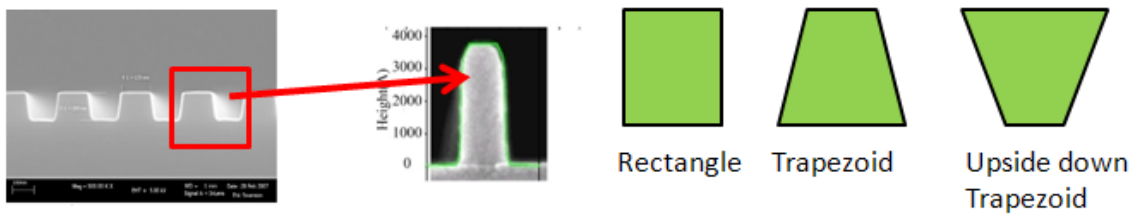


Figure 1-3 Three kinds of grating shapes in our

We use the R-Soft simulator [9] to generate a large amount of spectrum by different optical gratings which are depended on UPPER CD, LOWER CD and PR HEIGHT (Figure 1-4).

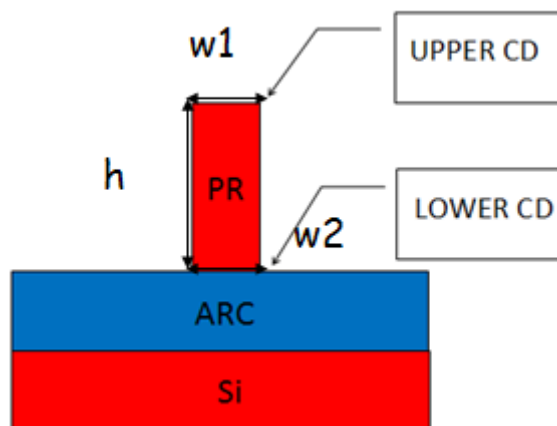


Figure 1-4 The parameter of the grating

However, the space complexity of our database is  $O(M \times N)$ , where  $M$  is the range of reflected spectrum,  $N$  is the total number of spectrum in the database. Typically, the data size of matrix  $A$  is  $225 \times 10^6$ . Therefore, we can perform Principle Component Analysis for compaction.

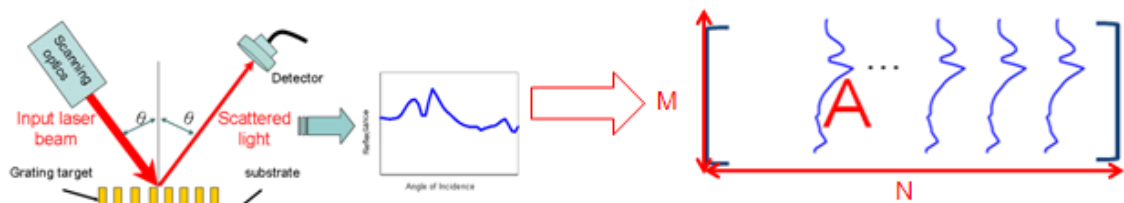
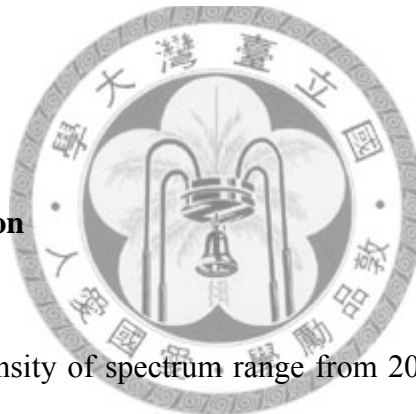


Figure 1-5 Database construction by R-Soft simulation



#### 1.4.2. Problem Formulation

The light reflectance intensity of spectrum range from 200nm to 1000nm or more, searching the most possible pattern of optical gratings is similarity to look for minimum error least square. Therefore, we can formulate the optical grating pattern recognition to the following simple equations:  $Ax=b$ , where  $A$  is a database of reflection spectrum of different grating shapes,  $x$  is a selection spectrum, and  $b$  is observed spectrum (Figure 1-6).

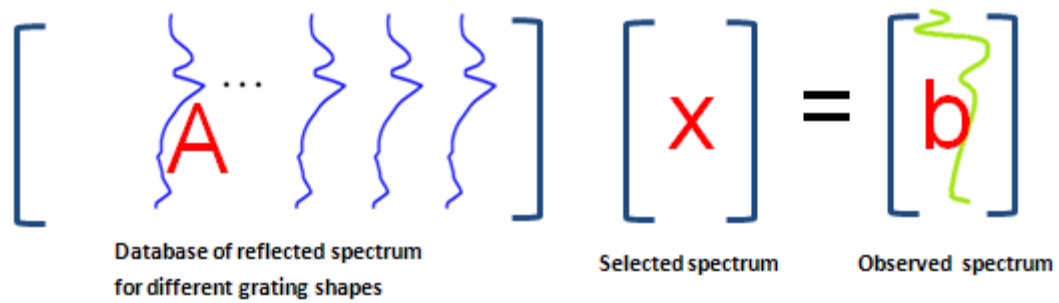


Figure 1-6 Problem formulation

With a measure spectrum  $b$ , searching process can be fundamentally mapped to a least square fitting problem in the following, where  $a_i$  is the  $i$ -th column vector of matrix

A:



$$\|b - a_i\|$$

(1.1)

### 1.4.3. Eigenvalue Decomposition

Suppose  $A$  is a square matrix of  $n$ -by- $n$ , let  $\lambda_1, \lambda_2, \dots, \lambda_n$  be the eigenvalues of a matrix  $A$ , let  $x^1, x^2, \dots, x^n$  be a set of corresponding eigenvectors, then:  $A = XDX^{-1}$  (Figure 1-7), where  $D$  is the  $n$ -by- $n$  diagonal matrix with  $\lambda_1, \lambda_2, \dots, \lambda_n$ ,  $X$  is the  $n$ -by- $n$  matrix whose  $i$ -th column is  $x^i$ ; if eigenvectors are orthonormal, then:

$$A = \lambda_1 x^1 (x^1)^T + \lambda_2 x^2 (x^2)^T + \dots + \lambda_n x^n (x^n)^T \quad (1.2)$$

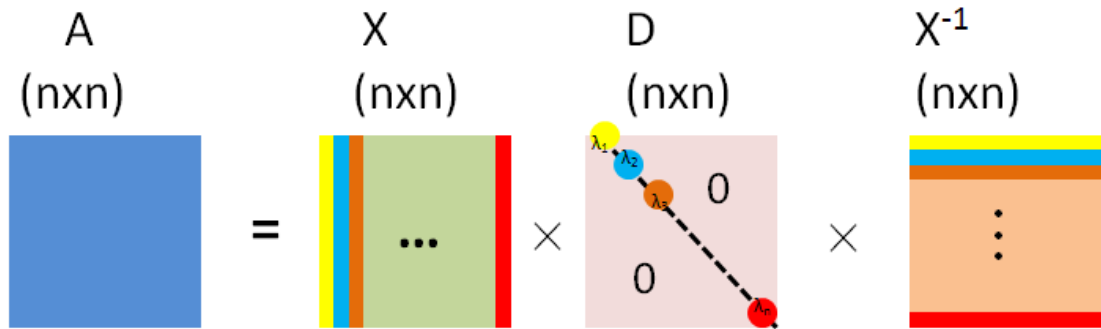


Figure 1-7 Problem formulation

We can perform the Principle Component Analysis for matrix A to simplify the express: (suppose  $\lambda_i \doteq 0$  if  $i > 2$ )

$$A \cong \lambda_1 x^1 (x^1)^T + \lambda_2 x^2 (x^2)^T \quad (1.3)$$

If the matrix A is not a square matrix, we can perform Singular Value Decomposition, we will introduce in the next section.

#### 1.4.4. Compaction by Singular Value Decomposition

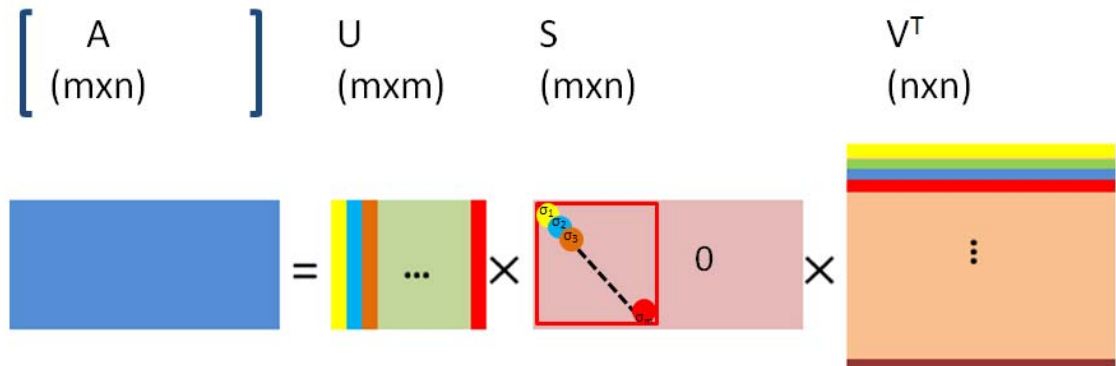


Figure 1-8 Singular Value Decomposition

Suppose  $A$  is an  $m$ -by- $n$  matrix, then there exists a factorization of the form  $A=USV^T$ . We perform the Singular Value Decomposition (SVD) (Figure 1-8) operation on  $A$  to decompose it into the product of three matrices,  $A=USV^T$ , where the size of  $U$ ,  $S$ ,  $V$  are  $m \times m$ ,  $m \times n$  and  $n \times n$ , respectively. The matrix  $U$  is  $m$ -by- $m$  and  $V$  is  $n$ -by- $n$ . Both matrix  $U$  and  $V$  are orthonormal. Here  $S$  is a matrix the same size as  $A$  that is zero except possibly on its main diagonal where each entry (singular value) in diagonal denoted the importance of the corresponding columns and rows of  $U$  and  $V$ , respectively. In this work, each reflected spectrum (column) of  $A$  can be represented by smaller linear combinations of more compact set of basic kernels. The matrix  $U$  is an orthonormal matrix that each column (orthonormal) is consisted of those kernels.  $SV^T$

are those weighting corresponding to each column of A. A simple concept of SVD can be illustrated in the following diagram (Figure 1-9).

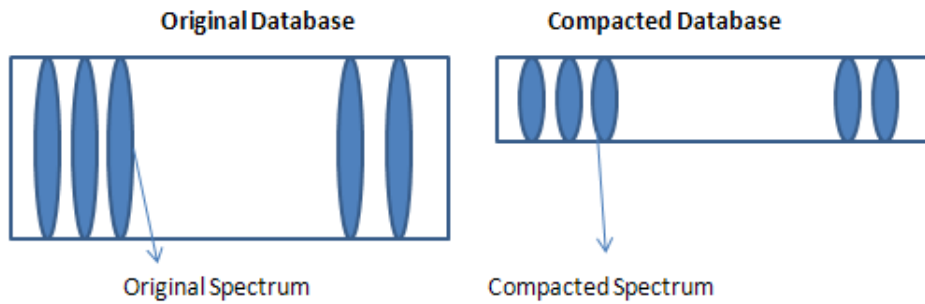


Figure 1-9 Database compaction

Depends on the decreasing speed of the singular values in S (Figure 1-10), we can effectively compact A into smaller sizes. Define the ratio of singular values; if the largest k components are reserved, the space complexity will be reduced from  $O(M \times N)$  to  $O(k \times N)$ , where M is the range of reflected spectrum and N is the total number of spectrum in the database. According to the singular value distribution, we can get large p with small k:

$$\frac{\sigma_1 + \sigma_2 + \dots + \sigma_k}{\sigma_1 + \sigma_2 + \dots + \sigma_k + \dots + \sigma_N} \cong p\% \quad (1.4)$$

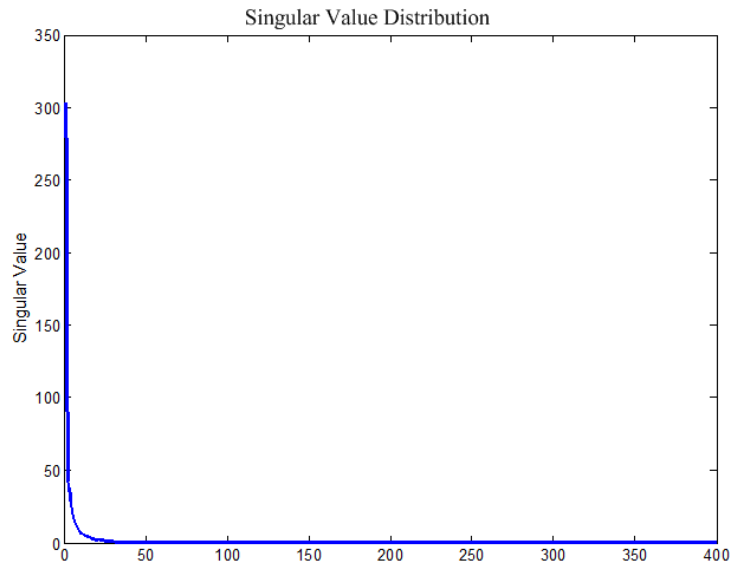
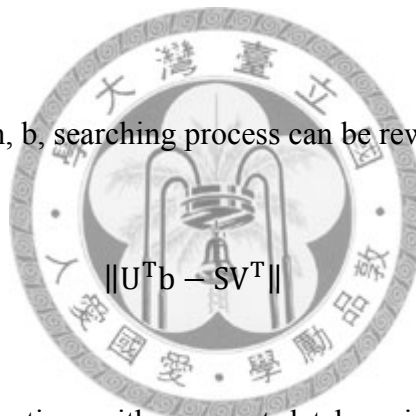


Figure 1-10 Singular value distribution of simulated reflected spectrum

With a measured spectrum,  $b$ , searching process can be rewritten as follow:



$$\|U^T b - SV^T\|$$

(1.5)

The speed up of searching time with compact database is 53X faster than original database. However, it always tries to find the match spectrum in the database exhausted. When  $A$  is getting huge, the time complexity is  $O(N)$ , where  $N$  is the total number reflected spectrum in the database. To solve this problem, it is crucial to sort the data into different bins according to its similarity. We will introduce it in the next section.

### 1.4.5. Classification

If we can group the data according to its similarity, then it's efficient to search the data. The quest is how to easily compute the similarity between reflected spectrums.

The first thought come into our mind is statistical moments [16], that is: mean, variation, skewness and kurtosis (Figure 1-11). The definition of the k-th moment can be represented as follows :

$$m_k = E(x - \mu)^k = \frac{1}{N} \sum_{i=1}^N x_i^k \quad (1.6)$$

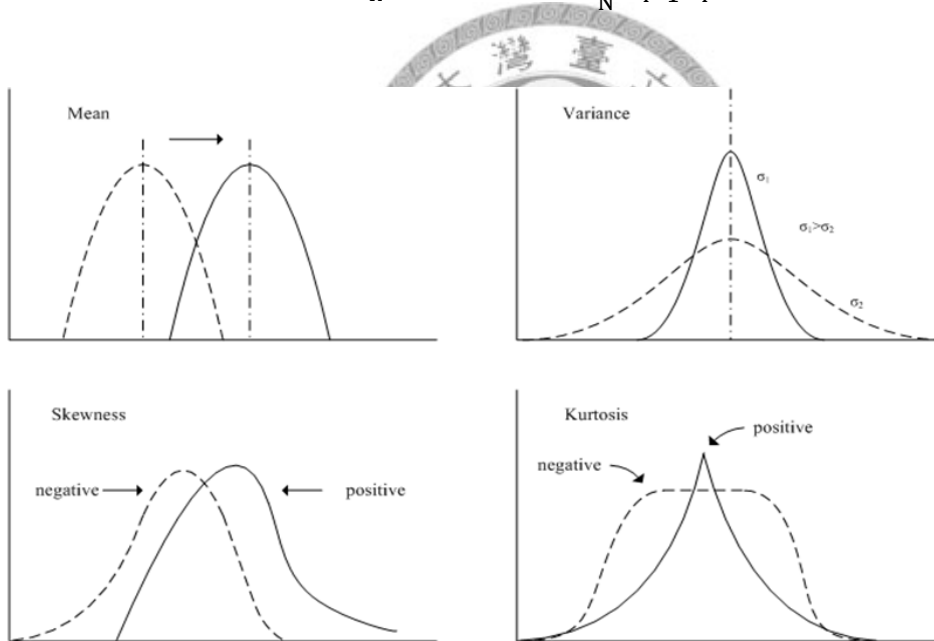


Figure 1-11 The meanings of moments



With the different shape of different moment distribution (Figure 1-12, 1-13), one can use this feature to classify the sub-database. Given a spectrum database A, we segment each spectrum into a few zones (Figure 1-14), and calculate the moments such as mean, variance, skewness and kurtosis in each zone. Once the moments of each zone are computed, there is a need to sort the data according to the moments.

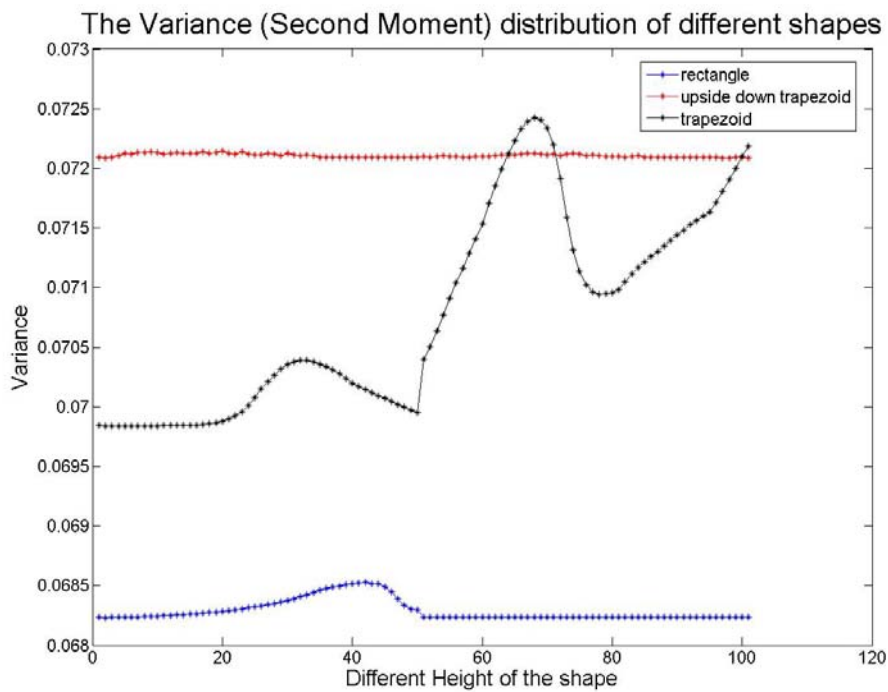


Figure 1-12 Variance varying according to shape variation

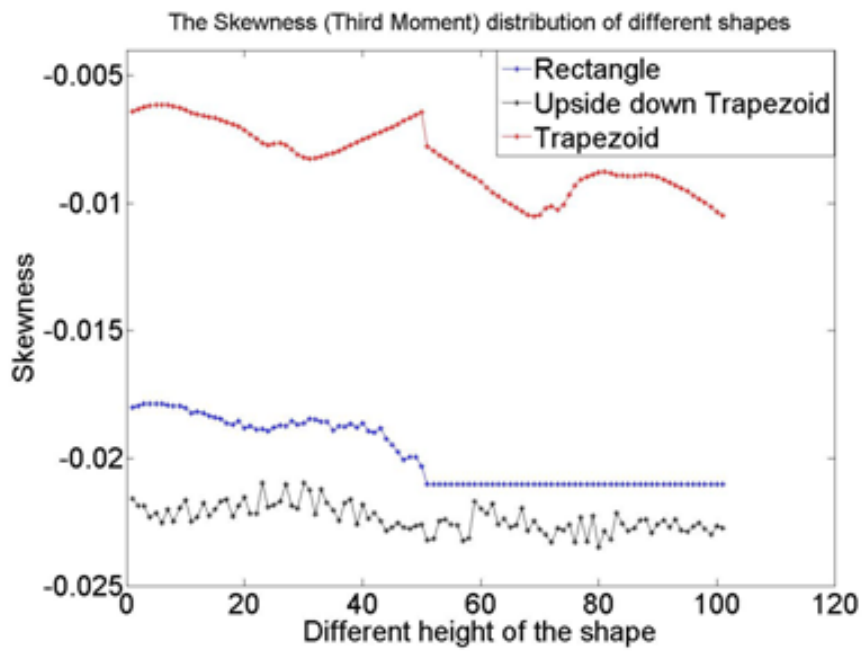


Figure 1-13 Skewness varying according to shape variation

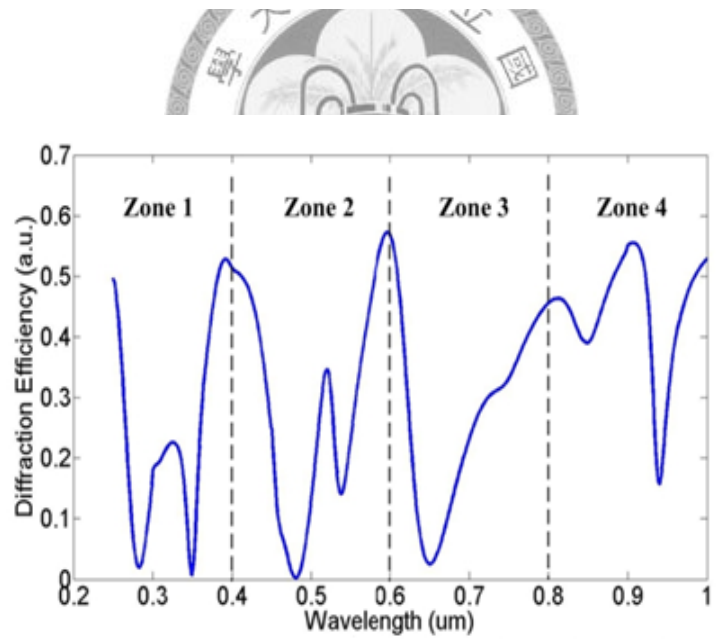


Figure 1-14 Segmented Spectrums with moment computations

### 1.4.6. Segmented Moment Matching

With the segmented moments, our searching process is to first compute its segmented moments as well and try to search the nearest spectrum with close moments.

To facilitate the process, we can map the segmented moments to a grid structure to store the spectrum according to the segmented moments. The concept is illustrated in

following diagram (Figure 1-15, 16):

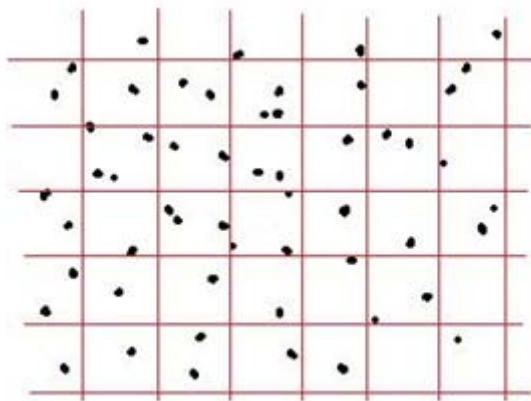


Figure 1-15 Grid storage structure in moment

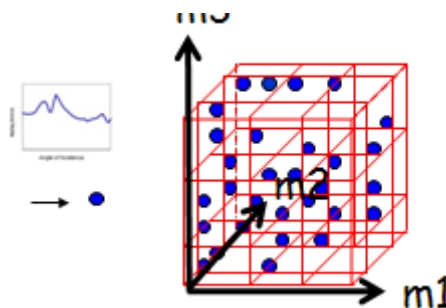


Figure 1-16 Grid storage structure cell in 3D moment spaces

It's similar to the idea of space partitioning and locality sensitive hashing. It's straightforward that the overall space could be separated into grids as the following graph (Figure 1-17). The grids act as a index for the grouping. Along each axial direction, it could be given a code for each grid. Thus, every grid could be represented as a coded string.

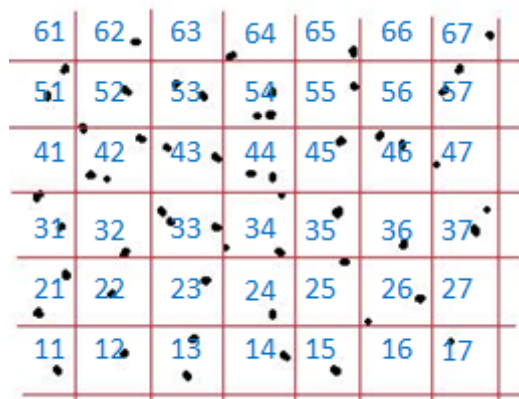


Figure 1-17 Coded grid

It's apparently that to minimize the penalty from the dimension. The effect could be reduced by reducing the dimension. Singular value decomposition is a powerful tool to construct a compact representation. With the segmented moment, each spectrum can be arranged into a moment vector in the following form, where  $m_{ij}$  is  $i$ -th zone of  $j$ -th moment:

$$[m_{11} \quad \cdots \quad m_{14} \quad m_{21} \quad \cdots \quad m_{24} \quad \cdots \quad m_{ij}] \quad (1.7)$$

then, the M matrix consists of each moment vector:

$$M = \begin{bmatrix} m_{11}^1 & m_{11}^2 & \cdots & m_{11}^n \\ m_{12}^1 & m_{12}^2 & \cdots & m_{12}^n \\ \vdots & \vdots & \cdots & \vdots \\ m_{m4}^1 & m_{m4}^2 & \cdots & m_{m4}^n \end{bmatrix}_{m \times n} \quad (1.8)$$

The first column is moment vector of spectrum of the first case and the last column is the moment vector of spectrum of the last case, etc. Then, we perform the Singular Value Decomposition on matrix M for dimension reduction in the following form:

$$M = USV^T \quad (1.9)$$

There's another critical problem that the data may be intensively clustered in some particular grid. This could not be avoided in a general sense. However, the SVD operation illustrates another view for how it is distributed. First, we can rewrite the SVD operation to M matrix as follow:

$$U^T M = S V^T \quad (1.10)$$

The standard deviation of each row in  $V^T$  is:

$$\sqrt{\frac{1}{N} \sum (v_i - \mu)^2} = \sqrt{\frac{1}{N} (\sum v_i^2 - N\mu^2)} = \frac{1}{\sqrt{N}} \quad (1.11)$$

That is, the V space might disperse roughly in the same trend for each row.

Furthermore, combining the singular value, the standard deviation of each row is :

$$\frac{\sigma_r}{\sqrt{N}} \tag{1.12}$$

This illustrates that  $V^T$  is related to the root of the size of the database. Also, for every row that is normalized by the singular value ( $\sigma_r$ ) will then have equal standard deviation. The distribution is similar to normal distribution in the following graph:

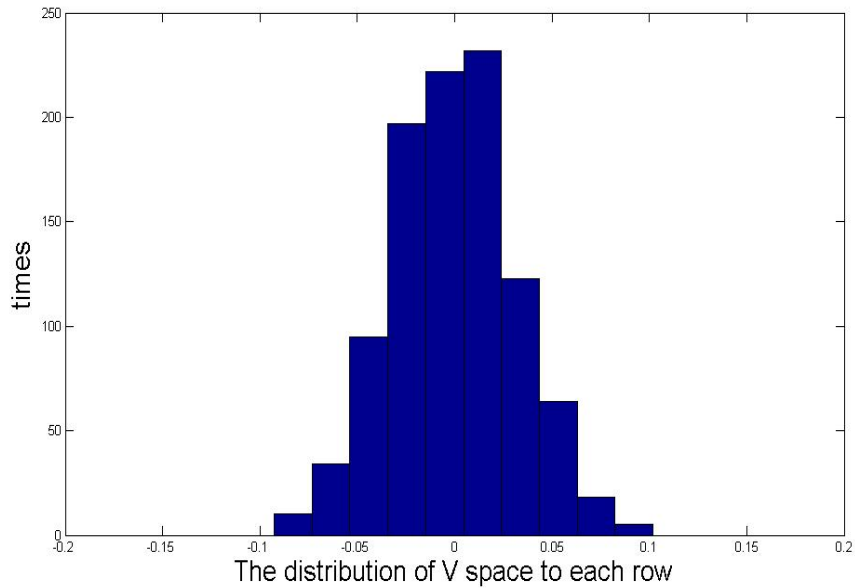


Figure 1-18 Distribution of V space

If  $\mu_i$  is and  $\sigma_i$  are the mean and the standard deviation of i-th row in V matrix. Then for any real number  $k > 0$ , by Chebyshev's inequality:

$$P(|x - \mu| \geq k\sigma) \leq \frac{1}{k^2} \quad (1.13)$$

Therefore, we can disperse the data uniformly according to the distribution; then, we can segment zones to each dimension; the following graph illustrates this concept:

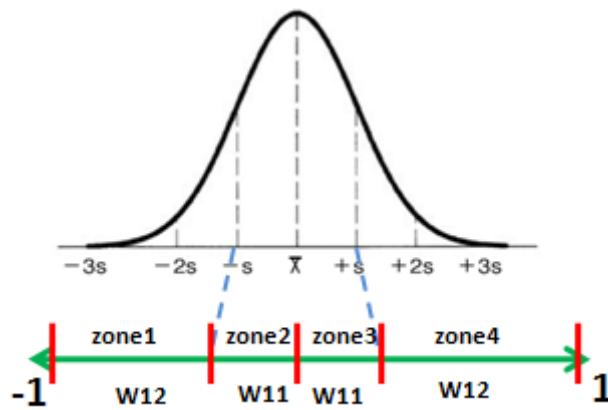


Figure 1-19 Disperse the data uniformly according to distribution of V space

With an input spectrum  $b$ , we can use the following algorithm to calculate the code locates on the grid.

**Algorithm Enode** ( $b, p, M, z$ )

**Input:**  $b$  (input spectrum),  $p$  (dimension of moment space),  $M$  (Moment matrix),  $z$  (zones)

**Output:**  $C$  (code)

**begin**

do  $M=USV^T, U^T b$ ;

**for** row one to  $p$ ; do

**for** zone one to  $z$

define  $:(U^T b)_p$  is the  $p$ -th row of  $U^T b$

observe the  $(U^T b)_p$  locates in which zone;

mark the index of the zone;

**end**

**end**

$C :=$  the set of marked index;

**end**



Finally, when the number of the data in one grid is still too large, we again perform SVD method on the moment spaces to find out the best match. The concept is illustrated in following diagram:

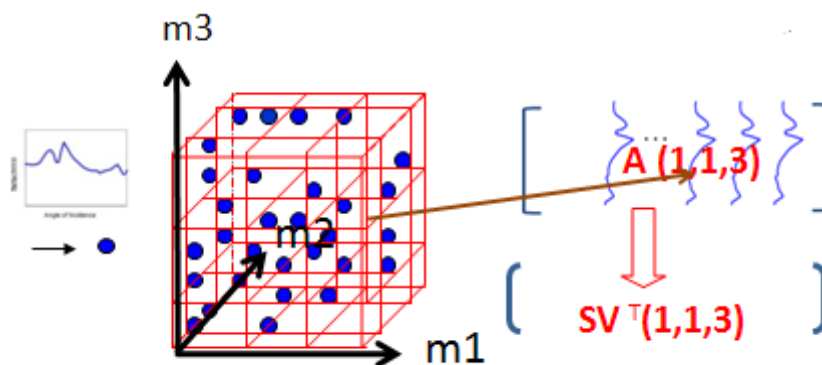


Figure 1-20 Perform SVD to 3D grid structure



### 1.4.7. Grid-based Searching

However the grid itself is not a perfect replacement to the distance. That is, for any possible sample in grid  $G_1$  and  $G_2$ , there is loose distance range, and is roughly the error tolerance of the grid. Suppose two samples  $s_1$  and  $s_2$  are located in grid  $G_1$  and  $G_2$ . The codes of  $G_1$  and  $G_2$  are:  $[C_{11} \ \dots \ C_{1p}]$  and  $[C_{21} \ \dots \ C_{2p}]$  respectively. See figure 1-21 for a 2-D illustration.

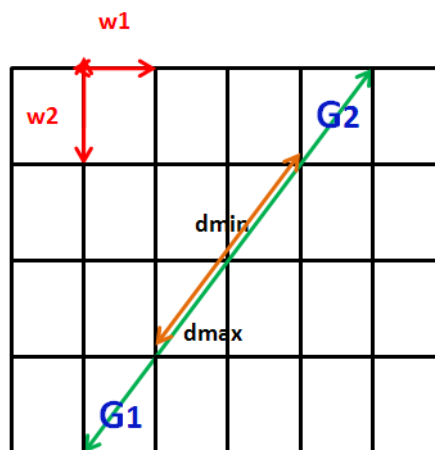


Figure 1-21 Schematic diagram of the distance between grid

The orange line in figure 1-21 is the least possible distance while the green one refers to the farthest distance. If the  $w_{pi}$  is the width of the grid in  $p$ 'th dimension with code  $i$ . Suppose  $w_{pi}$  is homogenous, that is:

$\forall i, w_{pi} = w_p$  , then:

the difference vector are:

$$d_{\min} = \begin{bmatrix} w_1 \times \max\{|C_{11} - C_{21}| - 1, 0\} \\ \vdots \\ w_p \times \max\{|C_{1p} - C_{2p}| - 1, 0\} \end{bmatrix}^T \quad (1.14)$$

and

$$d_{\max} = \begin{bmatrix} w_1 \times \max\{|C_{11} - C_{21}| + 1, 0\} \\ \vdots \\ w_p \times \max\{|C_{1p} - C_{2p}| + 1, 0\} \end{bmatrix}^T \quad (1.15)$$

then:

$$|d_{\min}| \leq |s_1 - s_2| \leq |d_{\max}| \quad (1.16)$$

If isotropic property holds, that is:

$\forall i, w_1 = \dots = w_p = w$  , then:

$$d_{\min} = w \times \begin{bmatrix} \max\{|C_{11} - C_{21}| - 1, 0\} \\ \vdots \\ \max\{|C_{1p} - C_{2p}| - 1, 0\} \end{bmatrix}^T \quad (1.17)$$

$$d_{\max} = w \times \begin{bmatrix} \max\{|C_{11} - C_{21}| + 1, 0\} \\ \vdots \\ \max\{|C_{1p} - C_{2p}| + 1, 0\} \end{bmatrix}^T \quad (1.18)$$

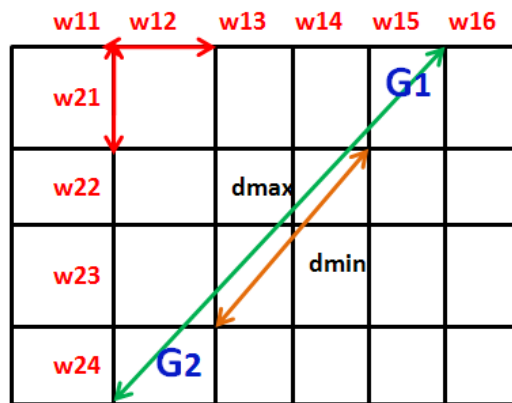


Figure 1-22 None-homogeneous grid



If the width  $i$  is not homogeneous (figure 1-22), that is

$\exists i, w_{pi} \neq w_p$ , then, we can denote an accumulated function as:

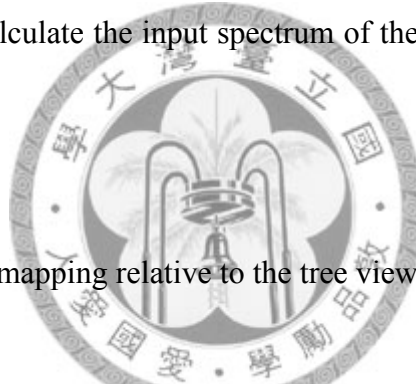
$$A_p(i) = \sum_{k=-\infty}^i w_{pk} \quad (1.19)$$

, then:

$$d_{\min} = \begin{bmatrix} A_1(\max\{C_{11}, C_{21}\} - 1) - A_1(\min\{C_{11}, C_{21}\}) \\ \vdots \\ A_p(\max\{C_{1p}, C_{2p}\} - 1) - A_p(\min\{C_{1p}, C_{2p}\}) \end{bmatrix}^T \quad (1.20)$$

$$d_{\max} = \begin{bmatrix} A_1(\max\{C_{11}, C_{21}\}) - A_1(\max\{\min\{C_{11}, C_{21}\} - 1, 0\}) \\ \vdots \\ A_p(\max\{C_{1p}, C_{2p}\}) - A_p(\max\{\min\{C_{1p}, C_{2p}\} - 1, 0\}) \end{bmatrix}^T \quad (1.21)$$

So far, we introduced how to calculate the distance between each grid. Therefore, the code calculated input spectrum, we can do so far from near to the visit. We still have a problem is that when we calculate the input spectrum of the encoded, we call on how far the margin?



First, we process the grid mapping relative to the tree view, show in figure 1-23.

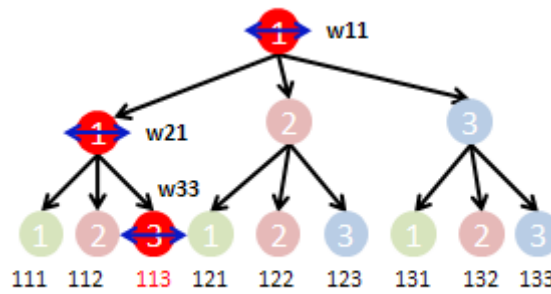


Figure 1-23 Relative to the grid structure of the correspond tree

If the  $w_{pi}$  is the width of  $p$ 'th dimension zone with code  $i$ , the upper bound of each

level to reflected spectrum is:

$$\left(\frac{(U^T b)_i}{\sigma_i} - V_i^T\right)^2 \leq w_{pi}^2 \quad (1.22)$$

With the searching processing of red path, the upper bound distance ( $u_b$ ) is:

$$0 \leq D_{113}^2 \leq \sum_{i=1}^p w_{pi}^2 = u_b \quad (1.23)$$

From near to far to visit, until the center of each grid with minimum distance greater than the upper bound (Figure 1-24):

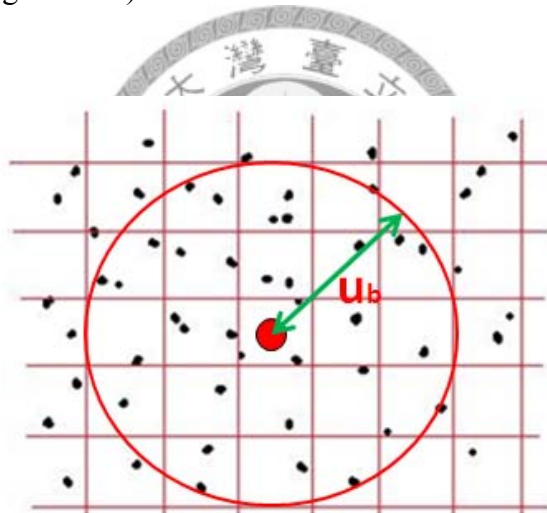


Figure 1-24 The upper bound from observation points

So far, we introduced how to calculate the distance between the grid and how to identify the input spectrum of the upper bound. Therefore, with the input spectrum  $b$ , we can use the following algorithm to find the nearest grating.

1. Calculate its code and upper bound of measured spectrum  $b$
2. Searching the grid with the same code
3. Calculate the minimum error square with data in the grid with the same code
4. Drop by the grid from nearest to the farthest
5. Repeat 1-4 until the grid with the observation points greater than the upper bound of the minimum distance
6. From each visited the smallest error in the grid, find the smallest. And to identify the corresponding grating



### **1.5. Approach to Arbitrary Segment Spectrum Range**

Sometimes, with a measured spectrum  $b$ , the range of wavelength is not fixed. For example, we usually measure 250nm to 1000nm as standard spectrum range. However, the range of spectrum probably is located at 400nm to 750nm, or 600nm to 650nm, etc.

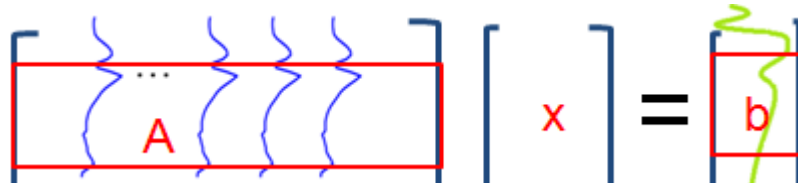


Figure 1-25 Compensation for a particular spectrum

Let's see the form of input spectrum, if it was located at 400nm to 600nm.

$$b_{400-600} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b_{400} \\ \vdots \\ b_{600} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (1.24)$$

In this case, we have only to be directed against the range of 400nm to 600nm in database. In order to reduce the error, we have to fill up the b vector. The following is the form:

$$\left\| U^T \left( \begin{bmatrix} 0 \\ b_{start} \\ \vdots \\ b_{end} \\ 0 \end{bmatrix} + \begin{bmatrix} A \\ 0 \\ \vdots \\ 0 \\ A \end{bmatrix} \right) - SV^T \right\| = \left\| U^T \left( \begin{bmatrix} 0 \\ b_{start} \\ \vdots \\ b_{end} \\ 0 \end{bmatrix} + \begin{bmatrix} USV^T \\ 0 \\ \vdots \\ 0 \\ USV^T \end{bmatrix} \right) - SV^T \right\| \quad (1.25)$$

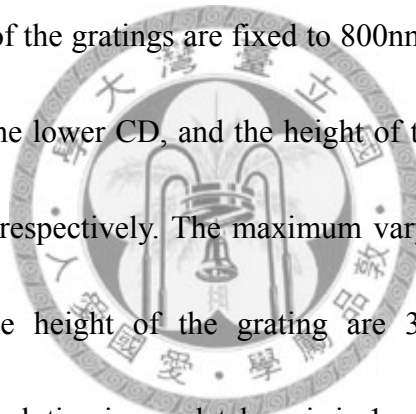
where,  $a_i$  is the  $i$ -th column vector of matrix  $A$ ,  $b_i$  is the  $i$ -th element of input spectrum  $b$ .

Therefore, we can derive the correct result by compensating the out-side range spectrum range.

## 1.6. Simulation Results

We use R-soft to simulate our experimental setup according to the following table at PC with Intel Core(TM)2 Duo CPUT7300 @ 2.00GHZ RAM:2015MB OS:VISTA(32-bit).

The pitch and arc height of the gratings are fixed to 800nm and 165nm respectively. The upper CD (figure 1-4), the lower CD, and the height of the grating are centered at 250nm, 250nm, and 350nm, respectively. The maximum varying amount of the upper CD, the lower CD, and the height of the grating are 30nm, 30nm, and 50nm, respectively. With total the resolution in our database is in 1nm scale.



The SPEC of our simulation setup	
Parameter	Value
Pitch	800 nm
Upper CD	250+/- 30 nm
Steps Increments	1 nm



Lower CD	250+/- 30 nm
Steps Increments	1 nm
PR Height	350+/-50 nm
Steps Increments	1 nm
ARC Height	165 nm

Table 1.1 The SPEC of simulation step

The search results by direct searching (MSE) and grid-based searching are show in the following diagrams; At first, we compare the direct search (MSE) without compact database. The searching time is almost 24 minutes (Figure 1-26).

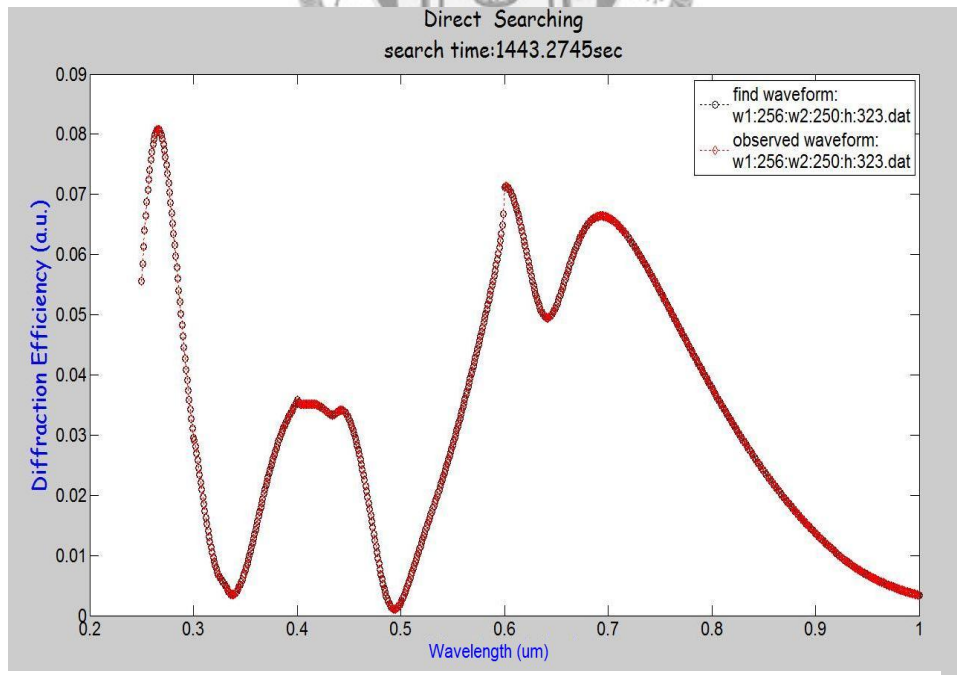


Figure 1-26 Direct Searching (MSE) without compact database

Second, we use direct searching for compact database, the total searching time about 26.92 seconds. The speed up is about 53 X than direct searching without compacting (Figure 1-27).

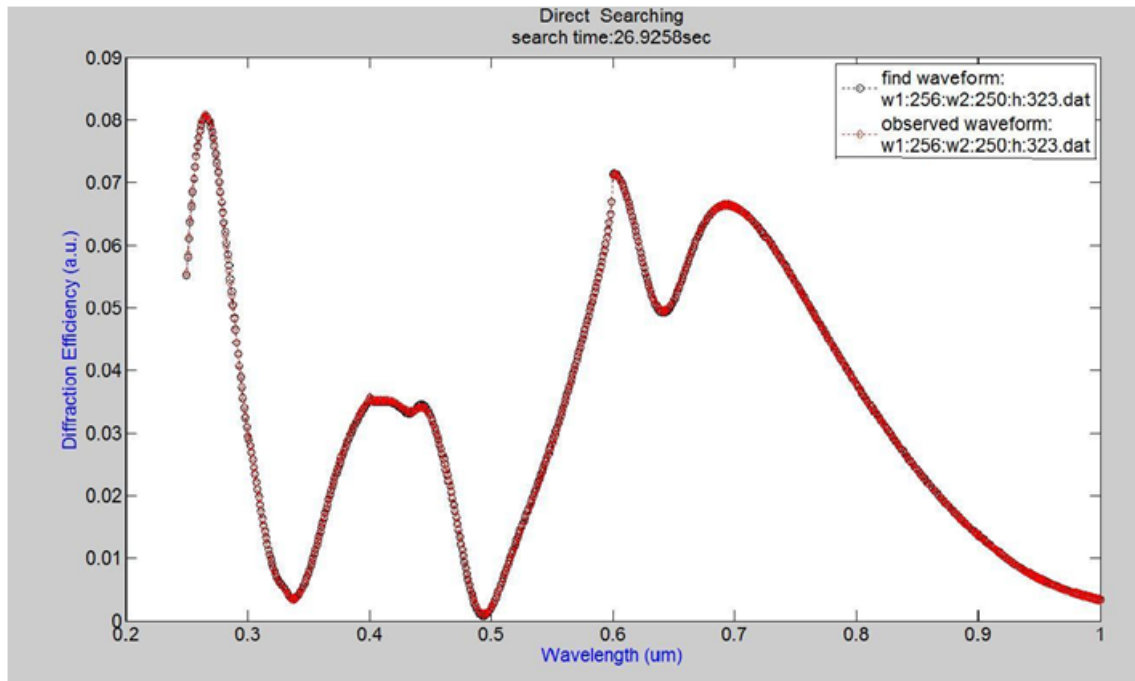


Figure 1-27 Direct searching (MSE) in compact database

Third, we use grid-based searching (Figure 1-28) for compact database; the searching time is about 0.393 seconds. The speed up is about 68 X speed up than direct searching (MSE) on entire compact database. Besides, it's almost 3600 x speed up than direct searching without compacting.

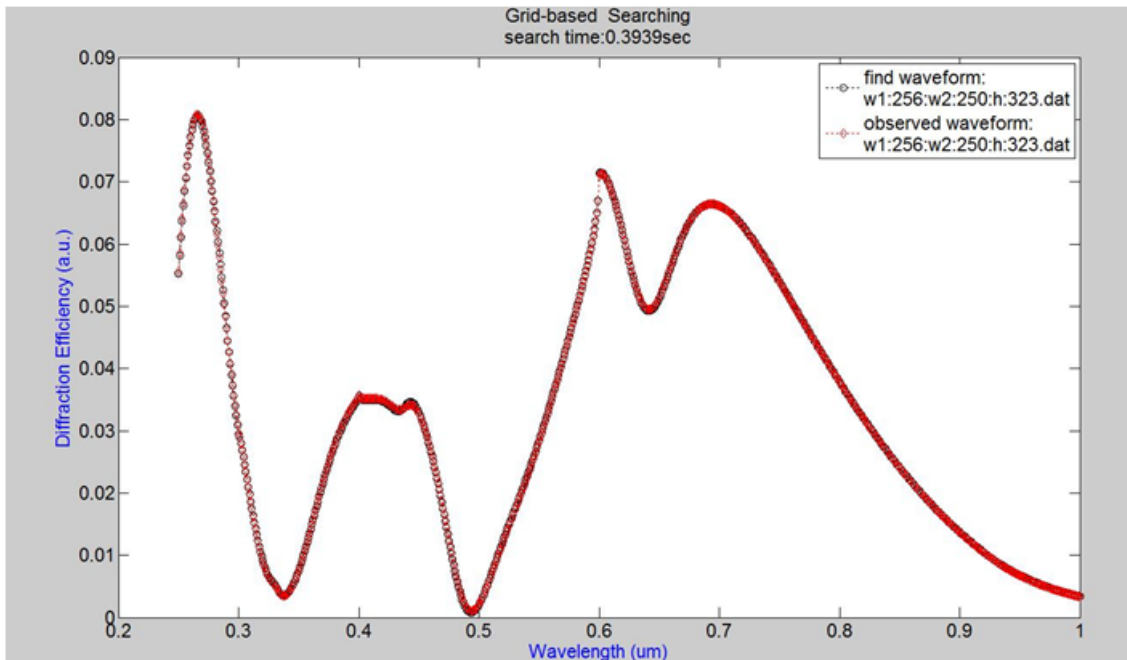
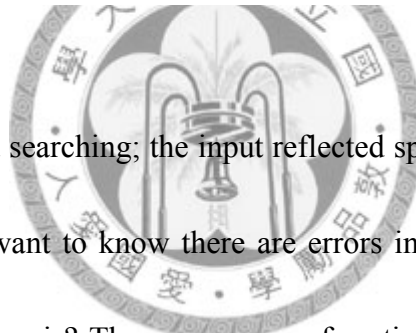


Figure 1-28 Grid-based searching in compact database



Fourth, we use grid-based searching; the input reflected spectrum with the Gaussian noise of 0.5%. Because we want to know there are errors in the input conditions, the accuracy of the search database is? The error range of grating on each dimensions are less than 1nm; the searching time is about 0.448 seconds. The speed up is about 3221x than direct searching (MSE) without compact database.

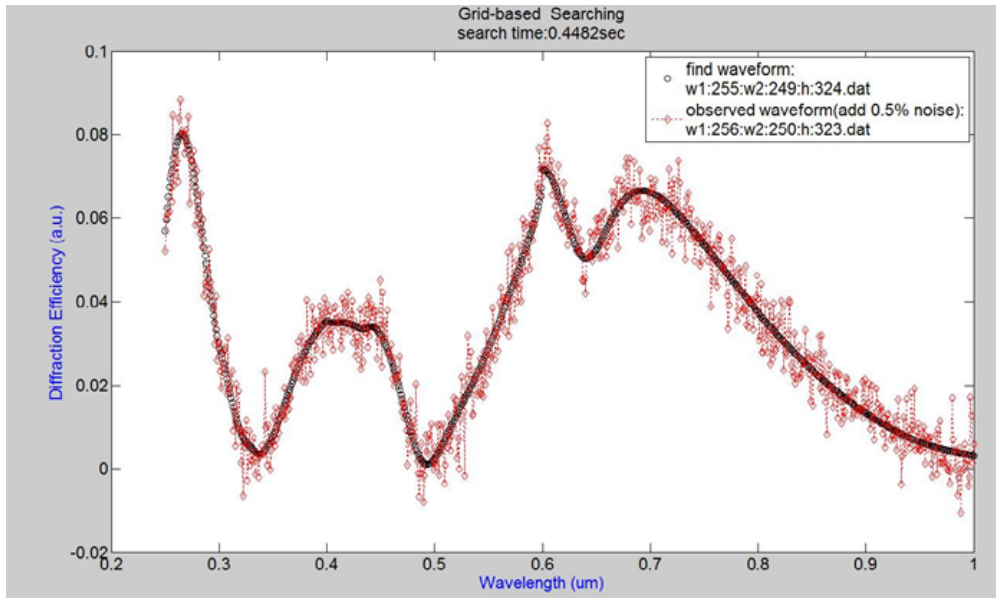


Figure 1-29 The input reflected spectrum with 0.5% Gaussian noise on grid-based searching

The following table is at the speed up is relative to the direct searching:

Searching way	Database	Time (s)	Speed up	Result
Direct searching	Original	1443.27	1	Exact
Direct searching	Compact	26.92	53	Exact
Grid-based searching	Compact	0.393	3672	Exact
Grid-based +0.5% Gaussian	Compact	0.448	3221	Less than 1nm

Table 1.2 Searching time comparison

## 1.7. Runtime Comparison

From the following graph, we can find that the direct searching method searches the entire database, so the runtime is linearly proportional to the data size. The runtime for grid-based searching is almost constant. It can see that our algorithm is significantly faster than MSE searching algorithm and the runtime remain flat when the database size increasing. The capability of our algorithm is still expandable.

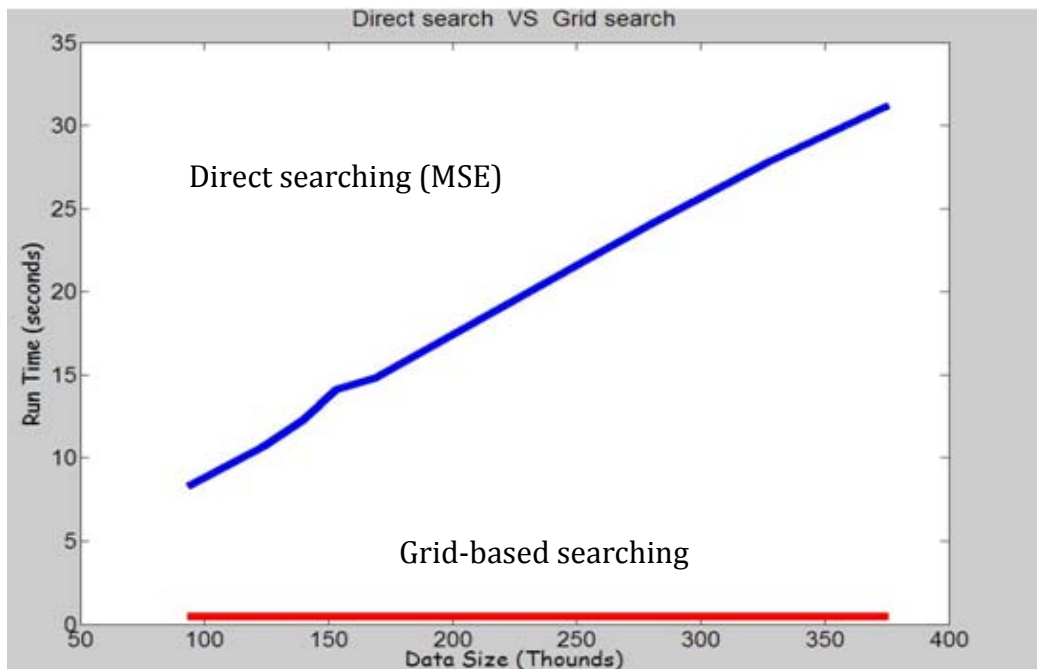
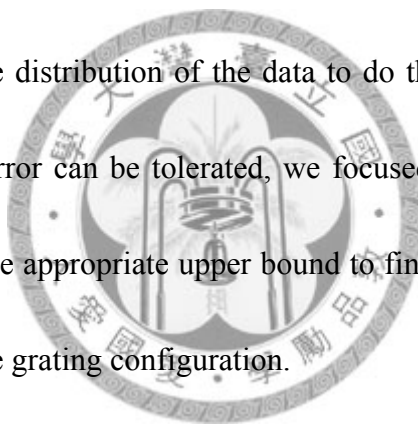


Figure 1-30 Run time comparison Direct search VS Grid Search

## 1.8. Summary

In this work, we are mainly aimed at large scale library, so we have chosen to improve the original MSE approach to search to find the grating inversely. We know that MSE drawback is that large database storage and searching time is too long. For the former, we use Singular Value Decomposition to compress our database. And use segmented moment matching for classification of database to form grid structure. Once again, we perform SVD of the characteristics of the moment spaces dimension reduction and based on the distribution of the data to do the cutting evenly dispersed. Finally in order to input error can be tolerated, we focused the search on the grid for visit from near to far. At the appropriate upper bound to find out the smallest minimum error square and the inverse grating configuration.



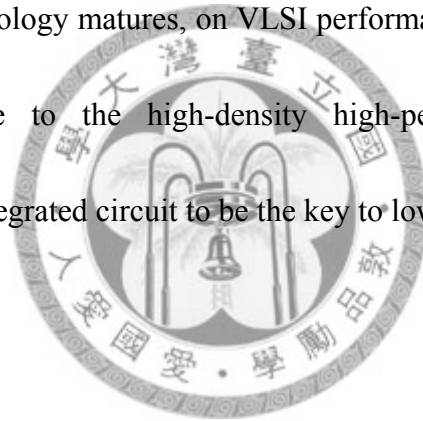
## 1.9. Feature Work

With the reflected spectrum  $b$ , the grid-based searching drops by from nearest to farthest and calculates the least square error to each grid to find the most possible result. In our work, it's suitable for parallel computing. We can use the multi-thread programming to visit each grid in the same time and compute the minimum error square to each spectrum in the same grid.

# Chapter 2. Parallel Optical Simulation Using Graphic Processor Unit

## 2.1. Introduction and Background

Integrated Circuits in a variety of application areas such as biomedical electronics, multimedia communications, consumer electronics and other products and services, the development and mass production has always occupied an important position, but with the electronic circuit technology matures, on VLSI performance and cost of the demand is growing. In response to the high-density high-performance circuit design, microlithography is the integrated circuit to be the key to low-cost mass production



## 2.2. Motivation

With the evolution of the chip manufacturing process as well as sub-naometer generation of high-performance components demand mask pattern of the size of microlithography is challenging the exposure light source wavelength resolution limits (CD: critical dimension), which makes a variety of optical diffraction, and other physical phenomena must be taken into account to predict and, while a variety of resolution enhancement technology is also widely been proposed in order to compensate

for circuit design and chip as much as possible entities the gap between to maintain the advanced integrated circuit production yield and functional correctness. To evaluate the microlithography quality, massive images are often generated for careful inspection using applications such as OPC.

In recent years, due to multi-core advances, more and more technology used to speed up parallel to a large number of complex computational problems. NVIDIA developed CUDA general-purpose parallel computing architecture that allows graphics processors have adequate capacity to solve complex computing problems. If we can use CUDA for the imaging in microlithography, then we can reduce a lot of computing time.



### **2.3. Preliminary**

The first part we will give an introduction on the image simulation in microlithography. The second part, we will introduce the General-purpose computing on graphics processing (GPGPU). We will apply this technology in parallel to produce the images. Finally, we will compare the performance of GPU and CPU on the differences and analyze it.



## 2.4. Overview of Image Generation

There are two approaches for imaging, one is Abbe's method and the other is Hopkins's theory. Hopkins's theory of the light source and system functions as a representative of the interaction of micro-imaging system, each conversion coefficient (TCC: transmission cross coefficient), while the Abbe's theory of malpractices light source is approximately the same tonal equivalent discrete light source (effective discrete source points) substituted into the formula into a number of imaging the convolution kernel to get the result of mask pattern.

### 2.4.1. Imaging Equation

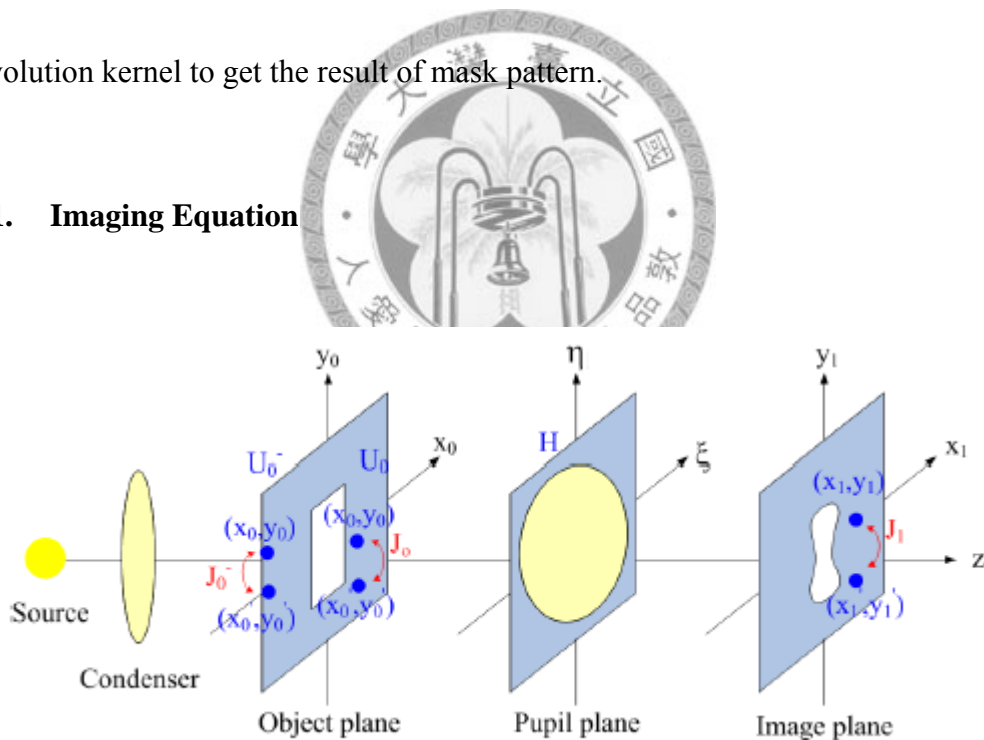


Figure 2-1 Coherent illumination

Initially, we define the cording systems as shown in Figure 2-1. Points in the object plane, pupil and image plane are specified by  $(x_0, y_0)$ ,  $(\xi, \eta)$ , and  $(x_1, y_1)$  respectively [11].

The image intensity  $I(x_1, y_1)$  at the point  $(x_1, y_1)$  on the image plane can be obtained Hopkins formula, which can be approximated subsequently by a summation

$$I(x_1, y_1) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} J_0^-(x_0 - x'_0, y_0 - y'_0) O(x_0, y_0) O^*(x'_0, y'_0) \times H(x_1 - x_0, y_1 - y_0) H^*(x_1 - x_0, y_1 - y_0) dx_0 dy_0 dx'_0 dy'_0 \quad (2.1)$$

Where points  $(x_0, y_0)$  and  $(x'_0, y'_0)$  are two arbitrary points on the object plane.  $O(x_0, y_0)$  is an appropriate transmission of the object, which is the mask function consists of complex number or with value 0 and 1 for Binary Intensity Mask. For convenience, we neglect the time and temporal frequency, so the frequency component is omitted. This simplified quantity is called the mutual intensity of the light and is given by  $J_0^-(x_0, y_0, x'_0, y'_0)$  [10].

Such expression is possible because source points the light source are mutually incoherent. The quantity delimited by the absolute value is the electric field, arising from a coherent source point of unit strength located at  $(f, g)$ . If we denote the square of this quantity by  $I_{con}$ , we can rewrite the imaging Eq. (2.1) as:

$$I(x_1, y_1) = a_{image} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} J(f, g) I_{con}(f, g) df dg \quad (2.2)$$

where

$$I_{con}(f, g) = \left| \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} H(f + f', g + g') O(f', g') e^{-i2\pi(f'x + g'y)} df' dg' \right|^2 \quad (2.3)$$

There are two approaches for imaging, one is Abbe's approach and the other is Hopkins's approach. In Abbe's approach is based on a spatial discretization of the source into discrete point sources. Hopkins's imaging requires calculating the transmission cross-correlation matrix (TCC) of the illuminating pattern with the pupil and its complex conjugate. In our work, we focus on Abbe's method, and we will introduce it in the next section.

#### 2.4.2. The Abbe's Method

Abbe's approach [11], which is also called integration approach of the source points, models imaging with such illuminators [12]. This approach is based on a spatial discretization of the source into discrete point sources as illustrated in the following:

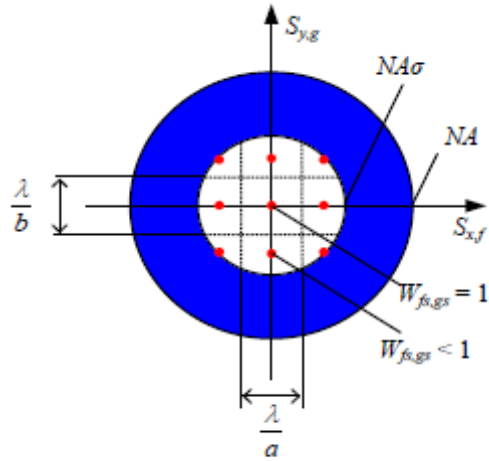


Figure 2-2 Discretization of a conventional partially coherent illumination system

Imaging system regards the light intensity  $I(x, y)$  as the output and the mask function  $O(x, y)$  as the input passing through the transfer function  $K$ . So we may rewrite the imaging equation (2.2) as:

$$I(x, y) = a_{image} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} J(f, g) I_{con}(x, y) df dg \quad (2.4)$$

where

$$I_{con}(f, g) = |H(x, y) e^{-i2\pi[(f-f'')x + (g'-g''y)]} * O(x, y)|^2 \quad (2.5)$$

The Abbe's method first approximates the effective source function  $J$  by a finite number of point sources:

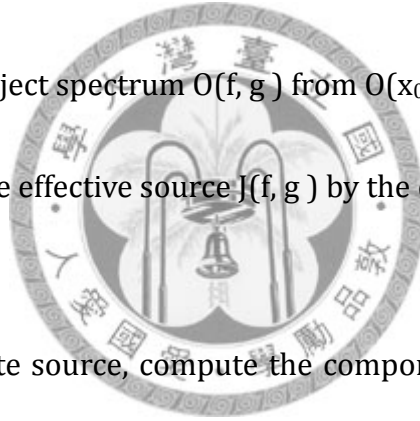
$$J(f, g) \cong J_{\Delta}(f, g) = \sum_s a_s \delta(f - f_s, g - g_s) \quad (2.6)$$

where  $a_s$  is the effective strength of the discretized point source located at  $\delta(f-f_s, g-g_s)$ . The aerial image is then obtained by an incoherent superposition of all the contributions and becomes

$$I(x, y) \cong a_{image} \sum_s a_s I_{con}(f_s, g_s) \quad (2.7)$$

The computation procedure for the Abbe's approach is given as below:

1. Calculate the object spectrum  $O(f, g)$  from  $O(x_0, y_0)$
2. Approximate the effective source  $J(f, g)$  by the discretized source  $J_{\Delta}(f, g)$  by Eq. (2.6).
3. For each discrete source, compute the component image according to Eq.(2.5)
4. Sum component images to obtain total intensity by Eq.(2.7)



For an effective source of area  $A_s$ , and an object of area  $A_m$ , the computation time scales according to

$$t_{compute} \propto A_s \cdot A_m \quad (2.8)$$

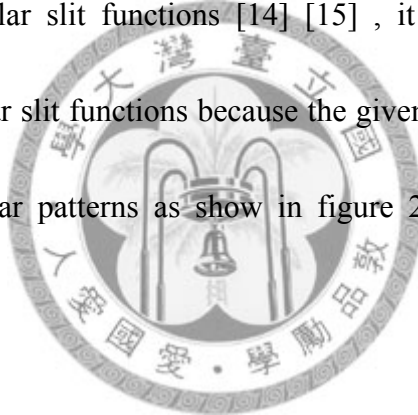
We rewrite the imaging equation in a convolution form at spatial domain as

$$I(x, y) \cong a \sum_s^{source} a_s |K_s * O(x, y)|^2 \quad (2.9)$$

Where we call  $K_s$  is a kernel function at  $(f_s, g_s)$ .

### 2.4.3. Mask Decomposition

Mathematically, if  $O(x, y)$  is the mask pattern function, it could be represented as a summation of  $N$  rectangular slit functions [14] [15], it could be represented as a summation of  $N$  rectangular slit functions because the given pattern with any shape can be composed of rectangular patterns as show in figure 2-3. We define the pattern function  $f(x, y)$  to be



$$O(x, y) = \sum_{i=1}^N f_i(x, y) \quad (2.10)$$

where

$$f(x, y) = \begin{cases} 1, & \text{if } x_i^0 \leq x \leq x_i^1, y_i^0 \leq y \leq y_i^1 \\ 0, & \text{others} \end{cases} \quad (2.11)$$

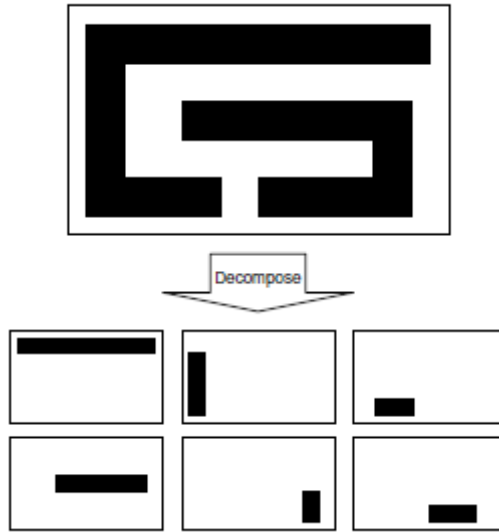


Figure 2-3 Mask pattern Decomposition

#### 2.4.4. Lookup Table



According to the **Eq.** and **Eq.** and we can get [14] [15]

$$\begin{aligned}
 I(x, y) &\cong a_{image} \sum_s^{source} a_s \left| K_s * \sum_{i=1}^N f_i(x, y) \right|^2 \\
 &= a_{image} \sum_s^{source} a_s \left| \sum_{i=1}^N K_s * f_i(x, y) \right|^2
 \end{aligned}
 \tag{2.12}$$

Therefore, we expand it to integral form as:

$$I(x, y) \cong a_{image} \sum_s^{source} a_s \left| \sum_{i=1}^N \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} K_s(x - x', y - y') f_i(x', y') dx' dy' \right|^2
 \tag{2.13}$$

Then, we concern the Binary Intensity Mask (BIM), the infinity limit can be replaced and the equation will be:

$$I(x, y) \cong a_{image} \sum_S^{source} a_s \left| \sum_{i=1}^N \int_{y_i^0}^{y_i^1} \int_{x_i^0}^{x_i^1} K_s(x - x', y - y') dx' dy' \right|^2 \quad (2.14)$$

Trough the mask of the decomposition, we have built a Rectangle based lookup table, for each rectangle to do look-up table to calculate the intensity. To consider an area A of the rectangle for an observation point (x, y) of the intensity contributions, convolution operators, according to the reciprocal theorem, we apply change of variables and let  $u=x_1-x_0$  and  $v=y_1-y_0$ . Substituting these variables into Eq.(2.14)

$$I(x, y) \cong a_{image} \sum_S^{source} a_s \left| \sum_{i=1}^N \int_{y-y_i^0}^{y-y_i^1} \int_{x-x_i^0}^{x-x_i^1} K_s(u, v) dudv \right|^2 \quad (2.15)$$

We build lookup table for Abbe's compact kernel [14] [15] without resolution restriction. Convolution result can be obtained by overlap area as show in figure 2-4. This means that our Table to record only those in lower left corner of the rectangle of the convolution results. All areas of the results can be obtained from the table look-up.



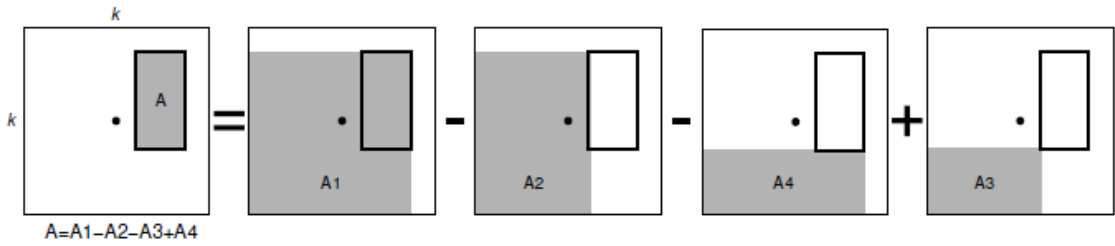


Figure 2-4 Convolution by pre-computed lookup table

In this way, we define the mask function  $O$ , which is given by the superposition of rectangle indicators as  $f_r(x, y)$ :

$$O(x, y) = \sum_{r=1}^N a_r \cdot f_r(x|y) \quad (2.16)$$

Here  $N$  is the number of decomposed rectangles,  $a_r = \pm 1$  for edges included all the edges or none of slit and  $a_r = -1$  for edges included only one edges of slit. The  $f_r(x, y)$  means the region of area ranges from  $(x_0, y_0)$  to  $(x_1, y_1)$ . Apply the linearity of convolution, it allows the output of a linear system with kernel  $K(x, y)$  to be expressed as a summation of contributions from the individual areas via the generated squares:

$$I(x, y) \cong a_{image} \sum_s^{source} a_s |\sum_{r=1}^N K_s * a_r f_r(x, y)|^2 \quad (2.17)$$

According to multiplicative identity of the convolution, most collections of functions can be consisted of several data distributions which allowed that delta distributions individually convolve with kernel function. Specifically,

$$K * \delta = K \tag{2.18}$$

Where  $\delta$  is the delta function and  $K$  is the kernel function as show in figure 2-5. One property is that the convolution distribution of the delta function and the kernel function is only within  $k \times k$  block. In other words, the range of its center of convolution result is  $k/2$ .

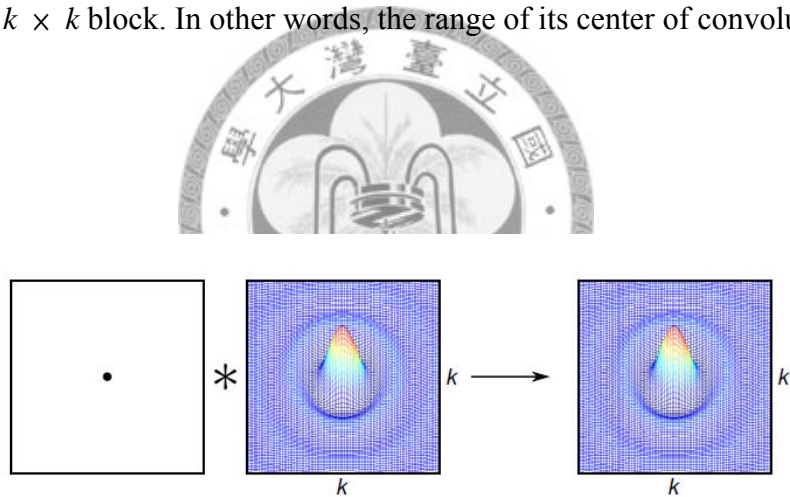


Figure 2-5 A delta function convolutions with kernel function

Apply this property; we only build a look up table whose elements represent the center of convolution function. For any shapes of rectangles, the center of convolution results only concerns with the shapes within  $k/2$  range. For example, there are three cases in

figure 2-6, although the rectangles are the gray ones, only the black rectangles contribute to the center of the convolution result.

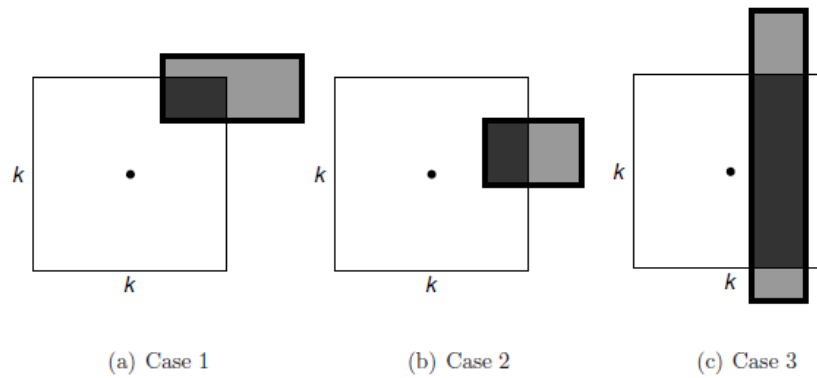


Figure 2-6 Example for the center of convolution



Therefore, we will build exact one look up table of  $k \times k$  size, for every black rectangle.

#### 2.4.5. Image Generation Flow

The image is calculated by convolution of mask patterns and kernel tables. We generated look up tables for the convolution operation. From the look up tables, we can derive the intensity of each observed point. The following figure 2-7 shows a brief image generation flow.

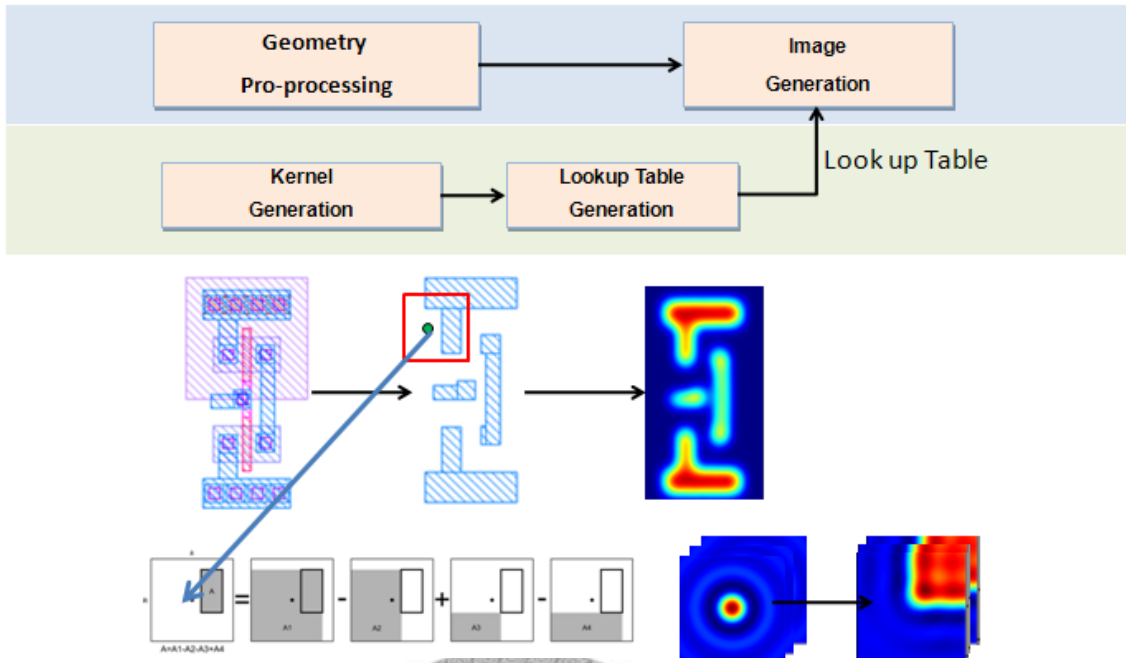


Figure 2-7 Image Generation Flow

#### 2.4.6. Mask Pattern Partition



We can partition the mask first[15], which has several advantages, usually a circuit area is quite large, and this will not only target specific areas to do the calculation of imaging, and can also be used to deal with parallel-oriented individual partition. Considering the influence scope of a point light source, we must make an appropriate extension for the adjacent regions. According to the size of the lookup table, we let the extension width is a half of the table width. So each partition region and its adjacent regions have overlap. The following is the graph of mask partition:

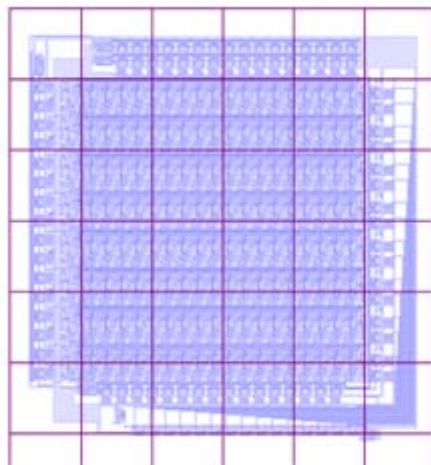


Figure 2-8 Mask Partition

## 2.5. Parallel Optical Simulation

So far, we have introduced how to generate the images in the OPC as a quality assessment, however, time is often generated images need to be considered. The time complexity of image simulation is  $O((n/r)^2)$ , where  $r$  is the resolution, and  $n$  is mask size. For example,  $n=10^4$ (nm),  $r=10$  (nm), then the total size of simulation is almost  $10^6$ , that means we need to calculate about  $10^6$  points in image simulation. Usually this will take a period of about six minutes. Taking into account the image simulation is actually a mask on the right to do look-up table for each point action. The look-up table operations are mutually independent. It's similar to do vector addition for times. Besides, the flow control is few used in look-up table. Therefore, it's suitable for

parallel-oriented operations. Next we have to verify that the look-up table accounts for high percentage in image simulation. We must use the profiling tool to observe it, we will introduce in the next section.

### 2.5.1. Code Analysis

To determine which function would be rewrite, we have to know what portion the total running time they take. Therefore, we can use a profiling tool to analysis it. The GNU profiler tool [18], GNU gprof which was used to gather the information during program run time that include the percentage of the total execution time and total number of times the function was called. The following table and graph is our test case:

Test Case:	
Circuit	Even-bit CSG of 32-bit Adder
Size	10740nm × 8210 nm
Layer	METAL 1
Resolution	50(nm)
Level	5

Table 2.1 The test case of Even-bit CSG of 32-bit Adder

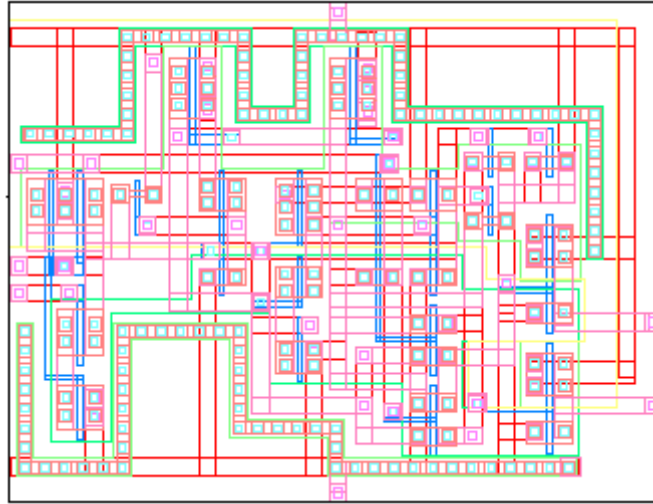


Figure 2-9 Even-bit CSG circuit of 32-bit adder

The following table is the show of gprof output:

The profile from gprof:			
Function	%	times(seconds)	function calls
Lookup_table:table_look_up()	98.3	16.48	439538656
XY:XY()	0.72	0.12	3595213
Polygon:add_point()	0.18	0.01	400142
GDSii:Record::read_data()	0.06	0.01	60304

Table 2.2 The profile output of optical simulation

From the profile output, the look-up table dominate the great majority of total execute time. Therefore, we would parallel the look-up table in order to speed up. To take advantage of multi-core hardware environments coupled with the software interface to do parallelism. Today, however, there are many choices of platforms to do. The next section, we will gradually introduce multi-core development history.

### **2.5.2. The Evolution of Microprocessor**

The field of microprocessor design is approaching a problem: the physical limitations. In the traditional, in order to promote performance, increasing the clock frequency of microprocessor is a basic method. However, the limitations of the power consumption became more and more ascended. Therefore, the architecture of multi-core processors was born. However an efficient way to take advantage of multi-core processors is difficult. Many algorithms cannot split up to parallel completely. Several designs of multi-core have been researched in order to apply the wide computational applications.

### **2.5.3. Multi-core Background Overview**

In the year 2004, IBM introduced the POWER5 processor. [23] It is a dual-core processor with support for simultaneous multithreading with two threads, so it



implements four logical processor. Instead, AMD introduced its first multi-core Opterons in the year 2005. The Intel Core architecture unveiled in the year 2006. Since the advantage of multi-core on multi-tasks, the multi-processor generation is coming. However, in the recent year, not only graphic card being used in entertainment and 3D imaging but can help us to calculate the computational intensive work. The graphics processing units contain multiple processing elements which are alike multi-core processors being used to operate simultaneously on streams of data. And in early 2003, the GPU throughput exceeded the CPUs in following graph [17]:

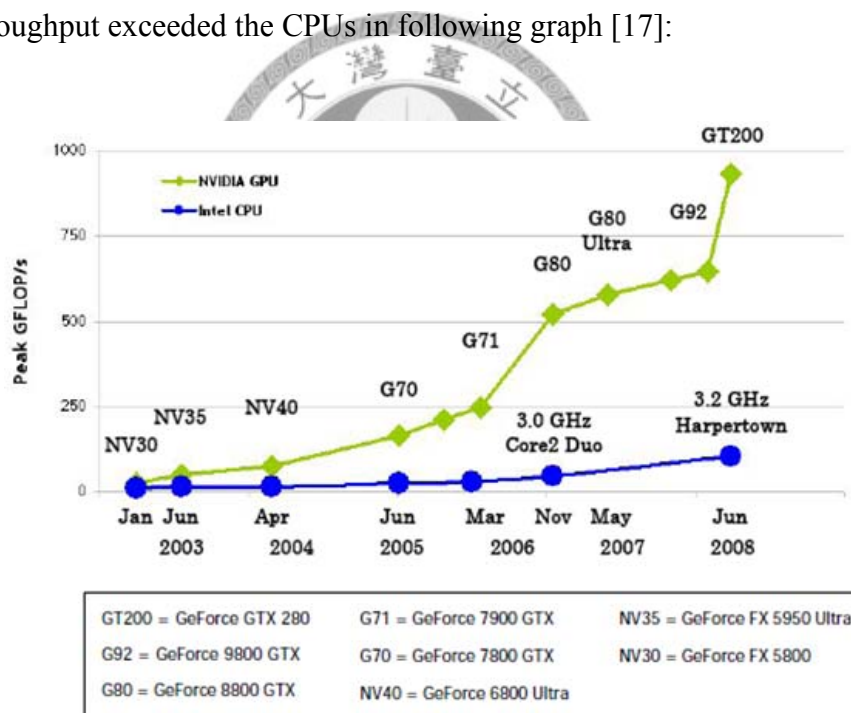
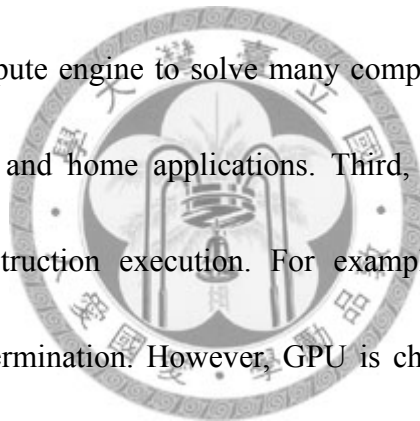


Figure 2-10 GPU throughput exceeded the CPUS

#### 2.5.4. NVIDIA CUDA

The following is the reason why we choose the graphics processing units: At first, the architecture of graphics processing units is specialized for compute-intensive which means more transistors can be devoted to data processing rather data caching and flow control (Figure 2-11). Second, the programming interface is similarly to C code, which is called “CUDA”. [17] The application programming interface “CUDA”, developed by NVIDIA corporation. CUDA stands for Compute Unified Device Architecture that leverages the parallel compute engine to solve many complex computational problems in variety of professional and home applications. Third, the architecture of CPU is designed for efficient instruction execution. For example, branch prediction, data dependence and logic determination. However, GPU is characterized by dealing with the same type of data-intensive computing without dependence. Its advantage is that no logical relationship of data computing. Under CUDA programming, we usually called GPU as device instead of CPU is host.



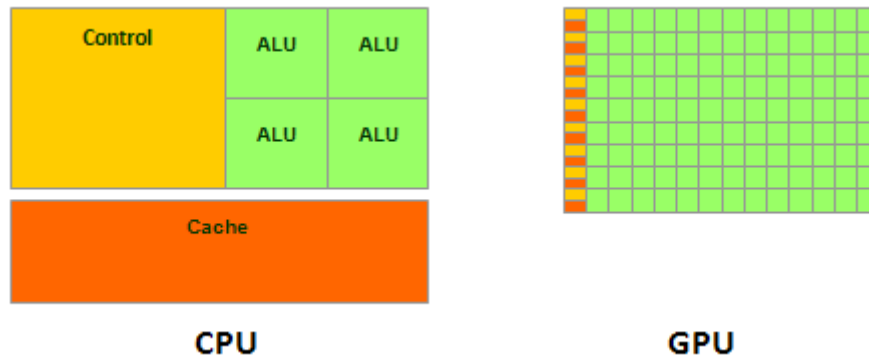


Figure 2-11 The GPU devotes more transistors to data processing

### 2.5.5. Architecture of Tesla C1060

Unlike CPUs that are designed for high sequential performance, GPUs are designed for computational data-parallel work [17][21]. A GPU is implemented as an aggregation of multiple that was called Multiprocessors which contains a number of Streaming Processor. A Streaming Processor is a SIMD ALU. Single Instruction Multiple Data (SIMD) means every Streaming Processor within a Multiprocessor executes the same instruction at the same time but the data may vary. Figure 2-11 shows the execution hierarchical level on CUDA.

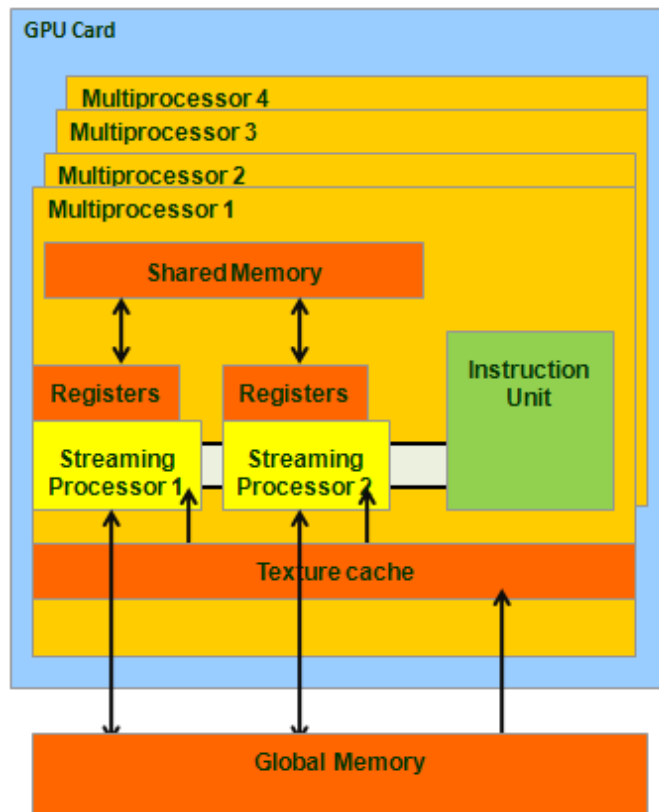


Figure 2-11 The CUDA hardware model



Each Stream processor accesses the local registers. The Multiprocessor has the shared memory that is available to the Streaming Processors on the same Multiprocessor. The Device memory is available to all Streaming Processors for read and write. Besides, the texture and constant caches are available on each Multiprocessor.

In this work, we use the NVIDIA Tesla C1060 on CUDA programming. There are 30 Multiprocessors in Tesla C1060, and each Multiprocessor contains 8 streaming Multiprocessors. Therefore, the total Stream Multiprocessors in Tesla C1060 are 240.

The total global memory is 4GB and memory bandwidth reaches 102GB/sec. Besides, each stream multiprocessor has shared memory, constant memory and texture memory. However, it is only read for constant memory and texture memory. The following table is the detail hardware information in Tesla C1060 [19]:

Device Name	Tesla C1060
# of multiprocessor	30
# of streaming processor cores	240
Frequency of processor cores	1.3GHz
Total Dedicated Memory	4GB DDR3
Memory Speed	800MHz
Memory Interface	512-bit
Memory Bandwidth	102GB/sec
System Interface	PCIE x 16
Compute capability	1.3

Table 2.3 Hardware information of Tesla C1060

However, the compute capability that means the standard of instructions, size of grid, size of block, memory per block and register. The minor revision number corresponds to an incremental architecture, possibly including new features. Therefore it's also another estimation way to evaluate the compute ability. In our work, the Tesla C1060 is the

version of 1.3 on compute capability. The detail can reference the appendix A.

### 2.5.6. Software Model

When we run a CUDA program, the CPU will distribute the data to GPU. In this process, the task on GPU is called kernel. Such a kernel is executed in the SIMD model. A grid is included of thread blocks and each block mapped to one Multiprocessors. Multiple thread blocks can be mapped onto the same block. [17][21][22] However, the resources, such as registers and shared memory will limit the maximum blocks. A thread block is a batch of threads that can cooperate with each other. Moreover, the thread blocks are grouped called warps. Warps are executed by scheduling them on the Streaming Processors of a Multiprocessor. The current available warp size is 32. Therefore, at least 4 clock cycles to execute an instruction by 8 Streaming processors. A Multiprocessor executes the warp in one block (Figure 2-13) .

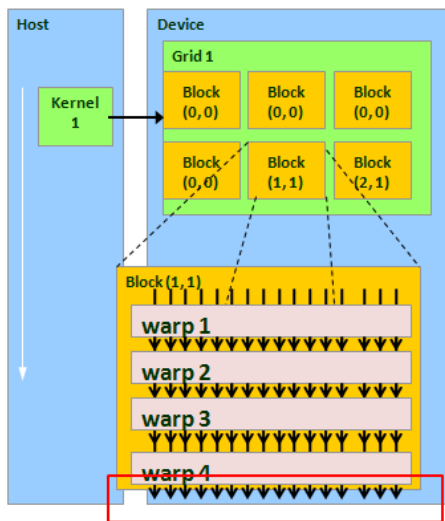
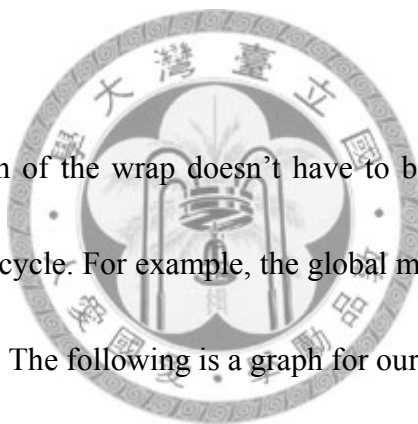


Figure 2-13 The CUDA software model



However, an instruction of the wrap doesn't have to be done at a time. When the wrap was need wait a long cycle. For example, the global memory access from the warp. It will switch another wrap. The following is a graph for our concept (Figure 2-14):

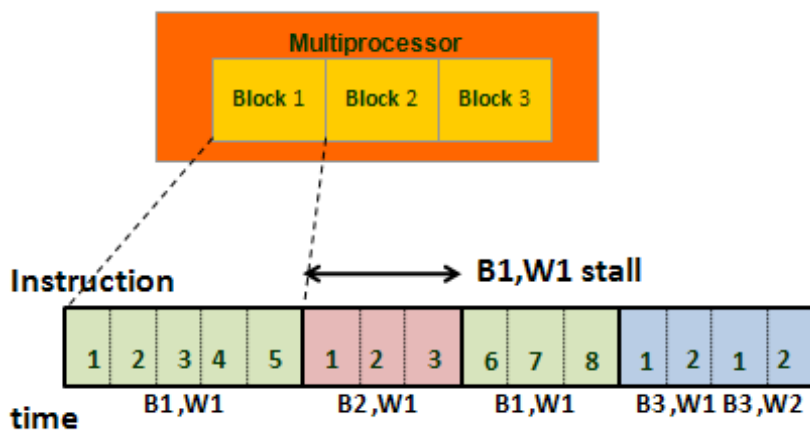


Figure 2-14 The CUDA warp scheduler

There are many memory addresses on CUDA as following: Registers, shared memory, constant memory, texture memory and global memory (figure 2-15).

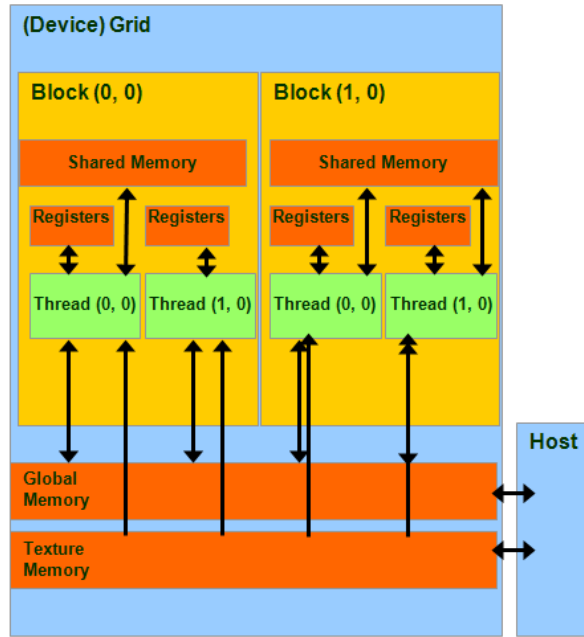


Figure 2-15 The memory hierarchical on CUDA

The following table is the feature of address:

Memory Type	Read/Write	Speed	Usage	Data lifetime
Register	R/W	faster	Thread	Thread lifetime
Shared memory	R/W	faster	Block	Block lifetime



Constant memory	R	faster	Grid	From allocation to deallocation
Texture memory	R	faster	oGrid	From allocation to deallocation
Global memory	R/W	slower	Grid	From allocation to deallocation

Table 2.4 Memory addressing of CUDA

Threads may access data from multiple memory spaces during their execution time. Each thread has the private local memory and registers. Each block has shared memory to all threads that are in the same block. Finally they can access to the global memory. Besides, there are two read-only memory spaces accessible by all threads: the constant and texture memory. They are optimized for memory usages by the data they only to read. For example, in our work the table is suitable for it.

### 2.5.7. Approach to Parallel Lookup Table

From the profile output, the look-up table dominates the great majority of total execution time. According to the operation of look-up table, it's similar to the vector

addition. Taking into account the impact of kernel, each mask of the partition has to calculate the impact respectively. Therefore, a table look-up is actually a kernel, the corresponding rectangle to make a addition. For the two-dimensional mask of the look-up table by the operator that we can flat into a loop to do thread synchronization of one-dimensional look-up table calculation.

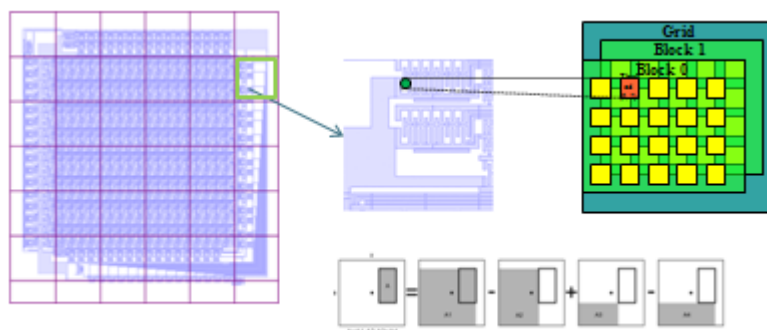


Figure 2-16 Parallel computing on look-up table

### 2.5.8. Kernel Execution Flow

There are five steps to run kernel function [20] in our work. At first, we have to dispose the host and device memory. Second, using the function `cudaBindTexture ( )` to bind global memory to texture reference. Therefore, we can take advantage of texture cache in stream multiprocessor and copy look-up table to it. Third, using function `cudaMemcpy( )` to transfer data from host memory to global memory. Fourth, using function `Run_GPU( )` to initiate the kernel and to calculate. Fifth, the result from GPU

must be written back to host memory.

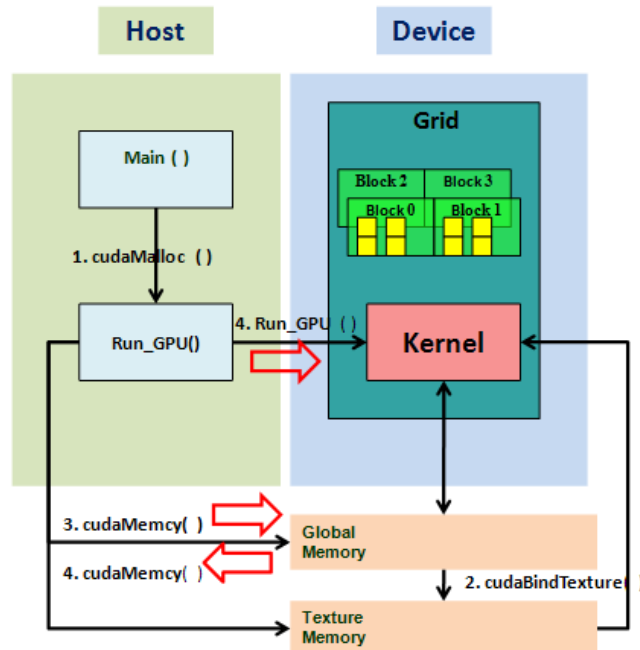


Figure 2-17 Kernel Execution Flow

### 2.5.9. The Optimization on Our Work

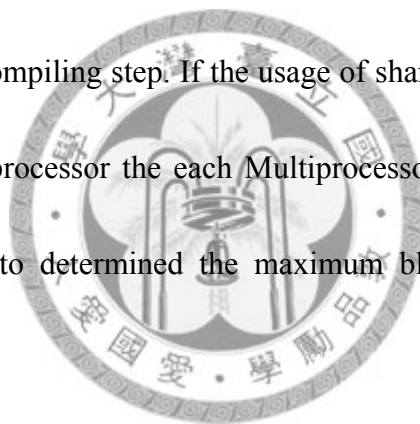


It's important to promote performance on CUDA [17][18][21]. Therefore, there are several basic concepts for optimization on CUDA.

At first, the clock cycle of accessing data from global memory is almost 500. It will reduce the performance under I/O time. Therefore, we can use the shared memory or texture memory than global memory. The look-up table records the results of a variety of rectangular convolution, and we do not need to write back to it. Therefore, it's very

suitable for texture cache to access.

Second, we have to arrange grid size and block size appropriately. The resources consumption of switching wrap is an important part on CUDA. We have to use up the Multiprocessors rather than idle some. Notice that there is a limit to distribute the registers to thread. If we exceed the rule, it will run error message in program. The maximum size of block per Multiprocessor are determined some reasons, the important part is the shared memory usage in our program. To observe it, we can use the flag `-ptx-options=-v` in compiling step. If the usage of shared memory greater than the half shared in each Multiprocessor the each Multiprocessor map one block only. The following formula is the to determined the maximum block per Multiprocessor by shared memory:



$$\begin{aligned} & \text{If shared memory/block} > 8\text{KB, block}=1 \\ & \text{else block} = \text{FLOOR}\left(\frac{\text{LimitTotalSharedMoemory}}{\text{MyShared}}\right) \end{aligned} \tag{2.19}$$

The register affects the block size per Multiprocessor, too. Use the flag `-maxrregcount` to limit the size of registers per thread and `-ptx-options=-v` to observe the registers usage per thread. Finally, we can use the following formula to

calculate the maximum blocks per Multiprocessor approximately:

$$\frac{R}{B \times \text{ceil}(T, 32)}$$

(2.20)

where,

1. R is the total number of registers per Multiprocessor
2. B is the number of active blocks per Multiprocessor
3. T is the number of threads per Block
4. Ceil(T,32) means the T rounded up to nearest of 32

Finally, the maximum of blocks per Multiprocessor determined the minimum of the limited by registers and limited by shared memory.

Third, the data transfer path between host and device is slow. By means of limit in PCI-Express bus of 8GB/sec is slower than the GPU memory bandwidth 102GB/sec (Tesla C1060). Therefore, the reduction transmission between host and device transmission is important. The mask should be observed for the purpose of calculation, do not always pass so many partition to the device.

Fourth, the branch condition just like while ( ), if ( ) and switch ( ) cause the divergence on multi-threads. The reason is that the switch on warp must have to be synchronization until the next cycle. It will spend twice time to execute the instruction. Therefore, in the program, we should avoid excessive writing to determine process control.

## 2.6. Performance Comparison

The environment of host PC is the following table and the information of GPU was mentioned about in section 2.5.5

CPU	AMD Athlon(tm) 64 X2 Dual Core Processor 5600+
Memory	4GB
VGA Card	Nvidia Tesla C1060
VGA Memory	4GB
Operating System	SUSE Linux
Compiler	GNU gcc 4.3.1 Nvidia CUDA NVCC Compiler

Table 2.5 The hardware environment

### 2.6.1. Grid size and Block size

The set up for block size and grid size is an important part to run the kernel program. The shared memory used in our work is less than the half of the Multiprocessor, we don't worry the grid size will limit one Multiprocessor that mapped into the one block. However, the compute capability 1.3 has the rule that the maximum of the active block per Multiprocessor is 8. That means at a time, the Multiprocessor just to manage the 8 block only. In our work, the shard memory usage per block about 4288bytes, and the Tesla C1060 have the shared memory 16384bytes per Multiprocessor. Therefore, the maximum of active blocks per Multiprocessor is 3. However, the registers of usage per thread also will limit the active block per Multiprocessor. In our work, registers per thread is 22, the limited of register by Eq.(2.20). If the block size is 512, then the maximum of active blocks per Multiprocessor is 1. Instead, the block size is 256, the maximum of active blocks per Multiprocessor is 2. However, the exceeding usage block size or registers will reduce the performance because the local memory will simulate the register for our program. The figure 2-18 is the performance between the different block size and grid size. The test circuit is Even-bit CSG which size is 2500nm × 4699nm in a 32-bit adder. Its entire layout shows in figure 2-9.

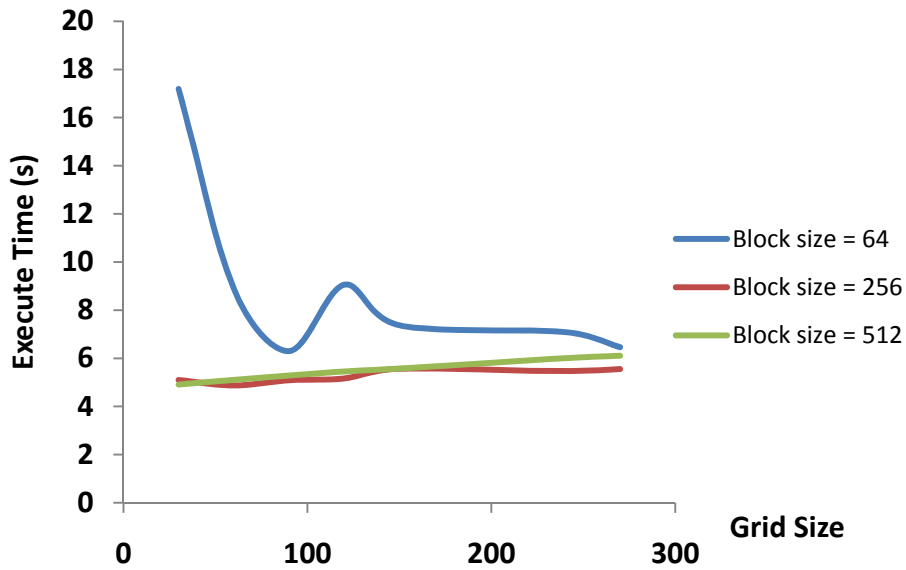


Figure 2-18 Performance comparison of grid size and block size



From the result, the best choice for our work is block size equal to 256 and grid size equal to 60. However, the block size equal to 512 and grid size equal 30 is another acceptable choice.

### 2.6.2. Execution Time Comparison

At first, let's see the total execution time on CUDA that include the I/O time and compare the execution time on CPU. The following graph compares the total time of GPU versus the CPU for different circuit size of layer METAL 1 of 32-bit adder tested.



Circuit	CMS	CMD	Latch_	EvenCSG_	OddCSG_
Size × 10 <sup>4</sup> (nm <sup>2</sup> )	2538	3058	6554	8817	10939

Table 2.6 Circuit size of our simulation

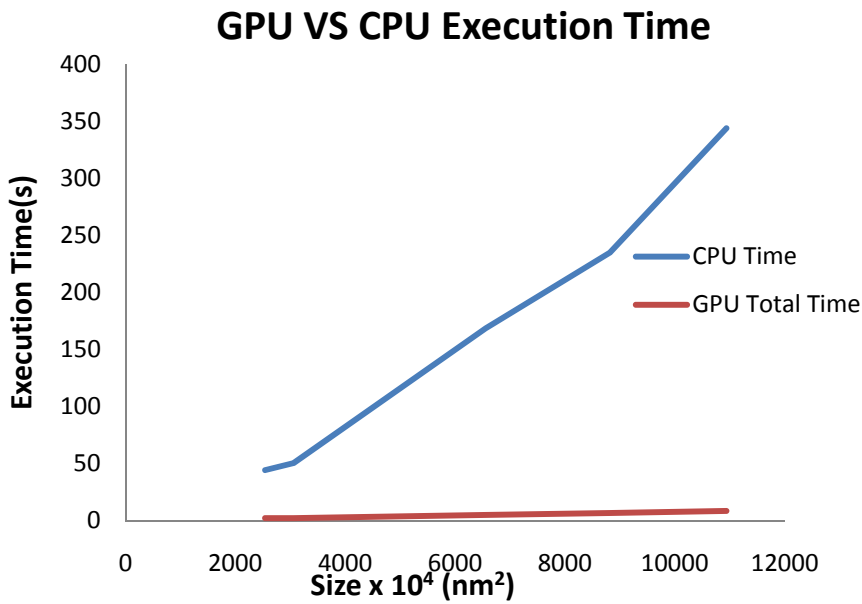


Figure 2-19 GPU VS CPU execution Time

For all layout size tested, the GPU finishes before CPU does. There are roughly 40 accelerations on maximum layout size. If we rule out the I/O time, the roughly speed up is 45. These results tell us the multi-thread programming on CUDA is useful to promote performance. However, we also have to compare the multi-cores of CPU between GPU.

The following is graph is the result:

### GPU VS. CPUs Performance Comparison

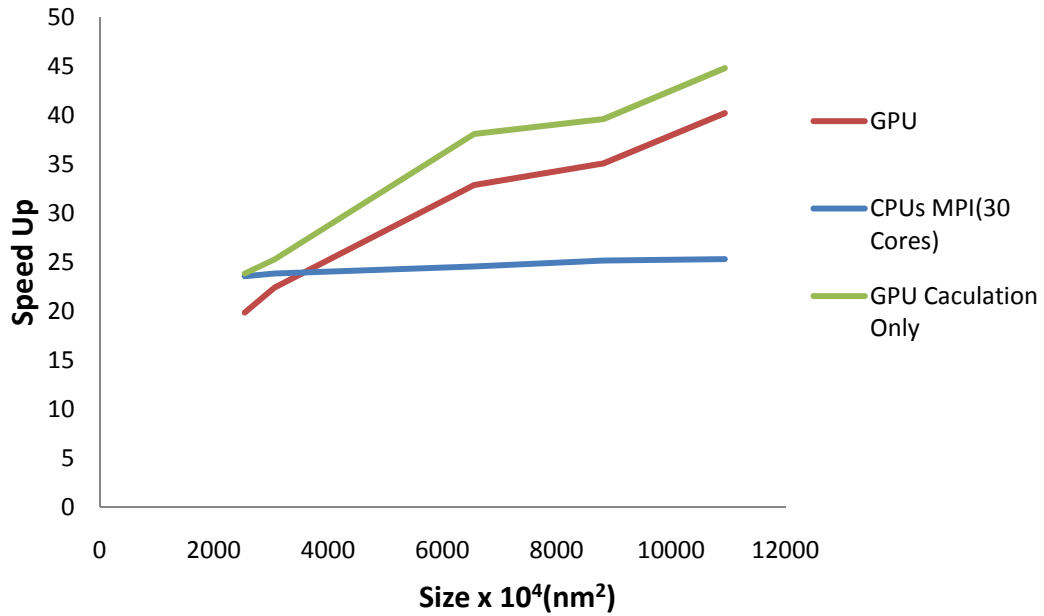


Figure 2-20 CPU VS CPUs (30 Cores)

Size × 10 <sup>4</sup> (nm <sup>2</sup> )	2538	3058	6554	8817	10939
GPU Only Calculation (s)	23.87	25.30	38.12	39.63	44.84
GPU Total (s)	19.88	22.43	32.62	35.90	40.24
CPU (30 Cores) (s)	23.6	23.88	24.6	25.2	25.35

Table 2.7 Performance comparison between GPUs and CPUs

Although, the multi-cores of CPU have a good speed up roughly 28 and one can

estimate that it's direct proportion to the cores. That means if someone want to promote the performance on multi-threading programming. The cost must high rather than just buy a GPU card only. Therefore, we can claim that use CUDA to speed up the parallel code on multi-threading programming is a wise choice.

### 2.6.3. Transmission Overhead

The I/O overhead is an important part on performance. From the following graph, we observe the overhead percent overhead is high on small case. The reason is that the calculation time under CUDA almost equal to the transfer time from host to device. These results provide us useful information that CUDA as possible as promote the device time that means more calculation under CUDA or large layout size will reduce the ratio of transfer time. The following formula is the percentage of transmission overhead:

$$\frac{\text{Transfer Time} + \text{Write Back Time}}{\text{GPU Total Time}} \quad (2.21)$$

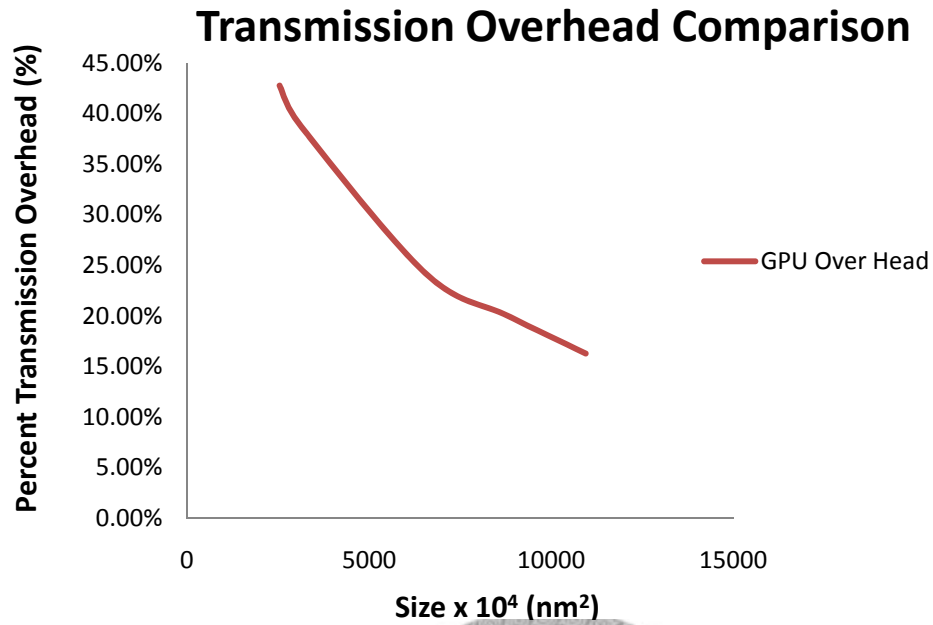
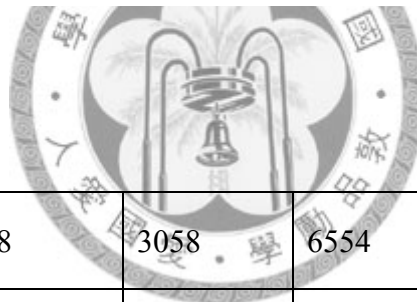


Figure 2-21 Transmission Overhead Comparison



Size x 10 <sup>4</sup> (nm <sup>2</sup> )	2538	3058	6554	8817	10939
GPU Overhead (%)	42.74%	39.05%	24.15%	19.96%	16.26%

Table 2.8 GPU overhead comparison

#### 2.6.4. Relative Error

Finally, we have to sure that the result from GPU is correct. Therefore, we can use the relative error to examine the total error. To compare the two arrays which are used to

storing the optic intensity. First array is generated by CUDA and the other is host PC.

The following formula is the relative error between two arrays and the following graph

is the comparison between different layout sizes:

$$\sqrt{\frac{\sum_{i=1}^N (\text{image}_{\text{CUDA}}[i] - \text{image}_{\text{Host}}[i])^2}{\sum_{i=1}^N \text{image}_{\text{CUDA}}^2[i]}} \quad (2.22)$$

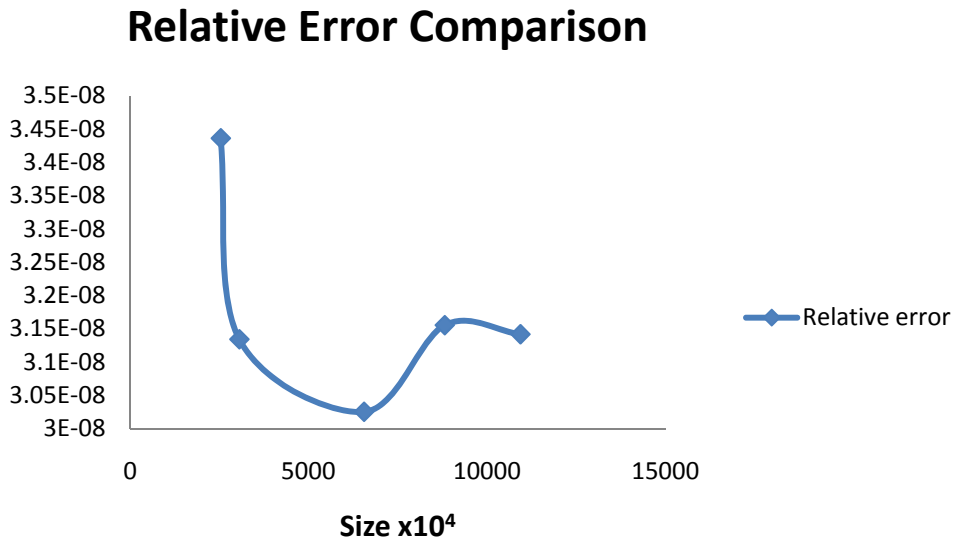


Figure 2-22 Relative Error Comparison

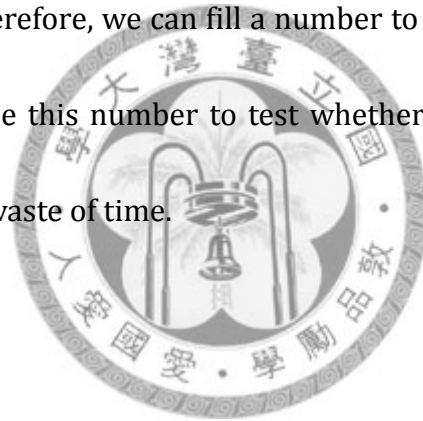
## 2.7. Summary

In this chapter, we described how to speed up for the optical simulation image generation. We use CUDA to apply parallel computing in look-up table method. It is faster 40x than a single CPU in order to save the quality evaluation time in OPC. In

today's , when the physical limitations of CPU to exercise restraint when the clock has become the key factors of continuous growth. Oriented parallel processing architecture has become the current trend.

## 2.8. Feature work

The boundary of the mask will determine whether to do look-up table. However, for each thread, the judge needs twice times to run the flow control. (if and else will be over-doing it) Therefore, we can fill a number to the most peripheral in the mask. In this way can use this number to test whether the boundary in order to reduce the unnecessary waste of time.



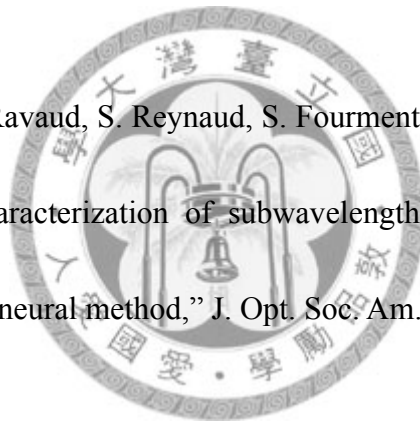
## Bibliography

- [1] X. Niu, N. Jakatdar, J. Bao, and C. J. Spanos, "Specular spectroscopic scatterometry," *IEEE Trans. Semicond. Manuf.* 14, 97–111 (2001).
- [2] R. Antos, J. Pistora, I. Ohlidal, K. Postava, J. Mistrik, T. Yamaguchi, S. Visnovsky, and M. Horie, "Specular spectroscopic ellipsometry for the critical dimension monitoring of gratings fabricated on a thick transparent plate," *J. Appl. Phys.* 97, 053107 (2005).
- [3] H. Huang and F. L. Terry, "Spectroscopic ellipsometry and reflectometry from gratings (scatterometry) for critical dimension measurement and in situ, real-time process monitoring," *Thin Solid Films* 455, 828–836 (2004).
- [4] C. J. Raymond, M. R. Murnane, S. S. H. Naqvi, and J. R. McNeil, "Metrology of subwavelength photoresist gratings using optical scatterometry," *J. Vac. Sci. Technol. B* 13, 1484–1495 (1995).
- [5] J. Garnaes, P. E. Hansen, N. Agersnap, J. Holm, F. Borsetto, and A. Kühle, "Profiles of a high-aspect-ratio grating determined by spectroscopic scatterometry and atomic-force microscopy," *Appl. Opt.* 45, 3201–3212 (2006).

[6] R. H. Krukar, S. L. Prins, D. M. Krukar, G. A. Petersen, S. M. GasparGaspar, J. R. McNeil, S. S. H. Naqvi, and D. R. Hush, "Using scattered light modeling for semiconductor critical dimension metrology and calibration," Proc. SPIE 1926, 60–71 (1993).

[7] I. Kallioniemi, J. Saarinen, and E. Oja, "Optical scatterometry of subwavelength diffraction gratings: neural network approach," Appl. Opt. 37, 5830–5835 (1998).

[8] S. Robert, A. M. Ravaud, S. Reynaud, S. Fourment, F. Carcenac, and P. Arguel, "Experimental characterization of subwavelength diffraction gratings by an inverse-scattering neural method," J. Opt. Soc. Am. A 19, 2394–2402 (2002)



[9] <http://www.rsoftdesign.com>

[10] Max Born and Emil Wolf. *Principle of Optics*. Press Syndicate of The University of Cambridge, 7 edition, 2005. pp.569-572.

[11] Max Born and Emil Wolf. *Principle of Optics*. Press Syndicate of The University of Cambridge, 7 edition, 2005.

[12] A. K. Wong. *Optical Imaging in Projection Microlithography*. SPIE Press, 2005

[13] H.H. Hopkins. "The Concept of Partial Coherence in Optics." *In Proc. Royal Soc.London,A208:263-277,1951*



- [14] Ming-Feng Tsai "Abbe-PCA: Compact Abbe's Kernel Generation for Microlithography Aerial Image Simulation using Principal Components Analysis", National Taiwan University
- [15] Szu-Kai Lin "An Efficient Contour Generation Algorithm for Microlithography Aerial Image", National Taiwan University
- [16] Jui-Hsiang Liu, Kuan-Lu Huang, Tsung-Yu Li, Meng-Chun Chiu, Charlie Chung-Ping Chen, Chih-Sheng Jao, Lon Wang "Efficient and Accurate Optical Scatterometry Diagnosis of Grating Variation Based on Segmented Moment Matching and Singular Value Decomposition Method", Graduate Institute of Photonics and Optoelectronics Engineering, National Taiwan University
- [17] NVIDIA(2008). "CUDA Programming Guide v2.0"
- [18] Steven Cook (2007). "Examining Suitability of Multicore Processor Architectures for Solving Realistic Computationally Intensive Problems by Simulating Synaptic Behavior in Large Neural Networks".
- [19] Wikipedia(2009). Tesla, [http://en.wikipedia.org/wiki/NVIDIA\\_Tesla](http://en.wikipedia.org/wiki/NVIDIA_Tesla)

[20] 林壽佑(2008) A Preliminary Study of Using Graphic Processors on Discrete Element Method Computation 國立台灣科技大學營建工程系

[21] J. Breitbart *CuPP -- A framework for easy CUDA integration*, HiPS 2009 workshop with IPDPS 2009, Rome, Italy, May 2009..

[22] J. Breitbart: *Case studies on GPU usage and data structure design*, Master Thesis, University of Kassel, 2008.

[23] [http://en.wikipedia.org/wiki/Central\\_processing\\_unit#History](http://en.wikipedia.org/wiki/Central_processing_unit#History)



## Appendix A

The following table is the compute capability 1.3:

The maximum number of threads per block is	512
The maximum sizes of the x-, y-, and z-dimension of a thread block are	(512,512,64)
The warp size is	32
The number of registers per multiprocessor is	16384
The amount of shared memory available per multiprocessor is	16KB
The total amount of constant memory	64KB
The maximum number of active blocks per multiprocessor	8
The maximum number of active warps per multiprocessor	32
The maximum number of active threads per multiprocessor	1024
The cache working set for constant memory per multiprocessor	8KB
The cache working set for texture memory per multiprocessor	6KB to 8KB