Closed-form Equal Computation Capability And Scheduling Divisible Workloads From Multiple Sources In Regular And Torus Mesh

A Dissertation Presented

by

Junwei Zhang

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2018

Stony Brook University

The Graduate School

Junwei Zhang

We, the dissertation committee for the above candidate for the
Doctor of Philosophy degree, hereby recommend
acceptance of this dissertation.

Thomas G. Robertazzi - Dissertation Advisor
Professor, Department of Electrical and Computer Engineering

Joseph S.B. Mitchell - Chairperson of Defense
Professor, Department of Applied Mathematics and Statistics

Esther M. Arkin - Member
Professor, Department of Applied Mathematics and Statistics

Yue Zhao - Member
Professor, Department of Electrical and Computer Engineering

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Closed-form Equal Computation Capability And Scheduling Divisible Workloads From Multiple Sources In Regular And Torus Mesh

by

Junwei Zhang

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2018

Abstract Here

Key Words:

To my Parents and all loving ones

Table of Contents

List of Figures . . . . . **xi**

List of Tables . . . . . **xii**

Acknowledgements . . . . . **xiii**

1 Introduction **1**

2 Closed-form Processor Equivalence **2**

2.1 Problem Description 3

2.2 With Front End Scenario 8

2.2.1 Regular Mesh 8

2.2.2 Torus mesh 36

2.2.3 General Case 41

2.3 Without Front End Scenario 42

2.3.1 Regular Mesh 42

2.3.2 Toroidal Mesh 59

2.3.3 General Case 59

2.4 Comparison Result Between Front End Processor and Without Front End Processor 61
2.4.1 Regular Mesh 61
2.4.2 Toroidal 65
2.5 Store and Froward Switching 66

3 Sensitivity Analysis 67
3.1 Front End Scenario 67
3.1.1 Data Injection on Corner 67
3.1.2 Data Injection on Boundary Processor 68
3.1.3 Data Injection on Inner Grid Processor 69
3.1.4 Toroidal 70
3.2 Without Front End Scenario 71
3.2.1 Data Injection on Corner Processor 71
3.2.2 Data Injection on Boundary Processor 73
3.2.3 Data Injection on Inner Grid Processor 74
3.3 Comparison Result With Corner Processor and Inner Grid Processor 75
3.4 Comparison Result With Front End and Without Front End . 76
3.5 Comparison Result Between Regular Mesh and Toroidal . . . 77

4 Multi-source Assignment Utilizing Voronoi Diagram 78
4.1 Even Data Fraction 78
4.1.1 Subgraph Data Injection 79
4.1.2 Individual Data Injection Positions 81

4.1.3 Toroidal 85

5 Multi-source Re-assignment Using Optimal Mass Transportation 93

5.1 Optimal Mass Transport Theory 93

5.1.1 Optimal Mass Transport Map Algorithm 95

5.2 Monte Carlo Method Implementation of Optimal Mass Transport 96

5.2.1 Divisible Load OMT Experiment 96

6 Conclusions and Future Works 104

Bibliography 105

List of Figures

2.1 A m*n regular mesh(m = 5, n = 5) 4

2.2 A toroidal mesh with grid unit cores 4

2.3 The 2*2 regular mesh and the data injection is on P_0 8

2.4 The timing diagram for 2*2 regular mesh and the data injection position is P_0 9

2.5 2*2 regular mesh. α_0, α_1, α_2, α_3 value 12

2.6 The 2*3 regular mesh and the data injection happens on corner processor P_0 13

2.7 The timing diagram for a 2*3 regular mesh and the data injection happens on processor P_0 14

2.8 2*2 regular mesh. α_0, α_1, α_3, α_5 data fraction value 17

2.9 The 2*n (n = 10) regular mesh and the workload happens on P_0 18

2.10 The timing diagram for 2*10 regular mesh and the data injection happens on P_0 19

2.11 3*8 regular mesh. The data injection position is P_0 25

2.12 5*5 regular mesh. The data injection position is processor P_0 . . 25

2.13 The 3*3 regular mesh and the data injection position is P_0 . . . 27

2.14 The timing diagram for 3*3 regular mesh and the data injection occurs on the boundary processor P_0 28
2.15 The data fraction simulation result of 3*3 regular mesh and the data injection happens on the boundary P_0 31
2.16 3*3 regular mesh. The data injection position is inner grid point P_0 32
2.17 The timing diagram for 3*3 regular mesh and the data injection is inner grid P_0 33
2.18 3*3 regular mesh. The data injection position is inner grid point P_0 35
2.19 The rectangular toroidal network 36
2.20 The rectangular toroidal network 37
2.21 The m*n toroidal network and the data injection is $P_{4,2}$ 38
2.22 The data fraction deployed based on the radius value 40
2.23 The timing diagram for 2*2 regular mesh without front end. . . 43
2.24 The data fraction deployed based on the radius value 45
2.25 The timing diagram for 2*3 regular mesh without front end. . . 47
2.26 The data fraction deployed based on the radius value 49
2.27 The timing diagram for 3*3 boundary data injection on P_0 . . . 54
2.28 The fraction curve for 3*3 boundary data injection on P_0 . . . 56
2.29 The timing diagram for 3*3 inner grid injection P_0 58
2.30 The data fraction deployed based on the radius value 60
2.31 The comparing result between front-end processor with without front-end processor in 2*2 regular mesh 62
2.32 The comparing result between front-end processor with without front-end processor in 2*4 regular mesh 63

2.33 The comparing result between front-end processor with without front-end processor in 3*3 regular mesh injection on boundary processor 64

2.34 The comparing result between front-end processor with without front-end processor in 3*3 regular mesh injection on inner grid processor 65

2.35 The comparing result between front-end processor with without front-end processor in 6*6 regular mesh injection on inner grid processor 66

3.1 Sensitivity analysis result of 2*10 regular mesh result 68

3.2 Sensitivity analysis result of 3*8 regular mesh and the injection position on boundary processor P_2 69

3.3 Sensitivity analysis result of data injection position on inner grid processor 70

3.4 Sensitivity analysis result of data injection position on inner grid processor 71

3.5 Sensitivity analysis result of 2*10 regular mesh result 72

3.6 Sensitivity analysis result of 3*8 regular mesh result 73

3.7 Sensitivity analysis result of data injection position on inner grid processor 74

3.8 Speedup difference between corner injection and inner grid injection 75

3.9 Speedup difference between front end and without front end in 5*5 regular mesh 76

4.1 Data injection consists of a subgraph of G 79
4.2 Data injection consists of a subgraph of G 79
4.3 Speedup vs σ 81
4.4 10 Voronoi Cells 82
4.5 10 Voronoi Cells's speedup curves 83
4.6 10 Voronoi Cells 85
4.7 10 Voronoi Cells's speedup curves 86
4.8 Reduced Voronoi Division Algorithm average processors' percentage 87
4.9 How to calculate torus Voronoi Diagram 88
4.10 Initial Voronoi Digram 89
4.11 Torus Voronoi Diagram 90
4.12 Voronoi Diagram Casting to the torus model 90
4.13 Voronoi Diagram Casting to the torus model 91
4.14 Torus Reduced Voronoi Diagram Casting to the Torus Model . . 91
4.15 Torus Reduced Voronoi Diagram 92

5.1 OMT theory relationship 96
5.2 Initial Voronoi Cell 98
5.3 OMT Division Cell 99
5.4 OMT Division Cell base relocation 100
5.5 Initial Voronoi Cell Speedup Curve 101
5.6 OMT Division Cell Speedup Curve 102
5.7 OMT Division Cell base relocation Speedup Curve 103

List of Tables

2.1 The processor number of various D_i 39

3.1 The processor number of various D_i 71

Acknowledgements

Thank you for Professor Thomas G. Robertazzi who gives me strong academic and emotional support in the past hard time period.

Chapter 1

Introduction

[?, ?]

Chapter 2

Closed-form Processor Equivalence

In large-scale data intensive problems with geographically distributed resources, load is generated from multiple sources[1]. It is assumed that the problem representation can be divided amongst the processors. Thus the problem representation is said to be "divisible". This chapter we discuss the closed-form processor equivalence[2][3] problem in the grid network of regular mesh, toroidal rectangle mesh.

This chapter is organized as follows. A problem introduction, notation and definitions are shown in Section I.

Section II considers about the unit core with front-end scenario. In the front-end situation, the regular mesh and torus rectangle mesh closed-form processor equivalence formula is investigated. Considering some various closed-form formula, the data injection position occurs on the corner processor, the boundary unit core or the inner grid processor.

Section III presents the unit core without front-end situation. In addition, the regular mesh and toroidal are discussed, respectively.

In section IV we give a comprehensive comparison result considering the different factors, for example the ratio between the communication ability to the computation capacity, the data injection position and the number of processors.

2.1 Problem Description

The processing of massive amounts of data on distributed and parallel networks is becoming more and more common. The problem of minimizing the processing time of extensive loads originating from a multiplicity of sources and being processed on a multiplicity of nodes presents a challenge.
The research user case involves the regular mesh Fig. 2.12 or toroidal mesh Fig. 2.2 of m*n processors. We refer to the $m * n$ processor as P_0, P_1, P_2, $\cdots$, P_{m*n-1}.

The workloads are denoted as $\mathbf{L_1}, \mathbf{L_2}, \cdots, \mathbf{L_k}$. We consider two situations, such as the processor contains the front-end and without front-end. In order to simply the question model, we have the following assumptions:

- We choose virtual cut through [4] switching technique to transmit the assigned workload between processors.
- We do not consider return communications.
- Communication delays are taken into consideration.
- The time costs of computation and communication are assumed to be linear function of the data size.

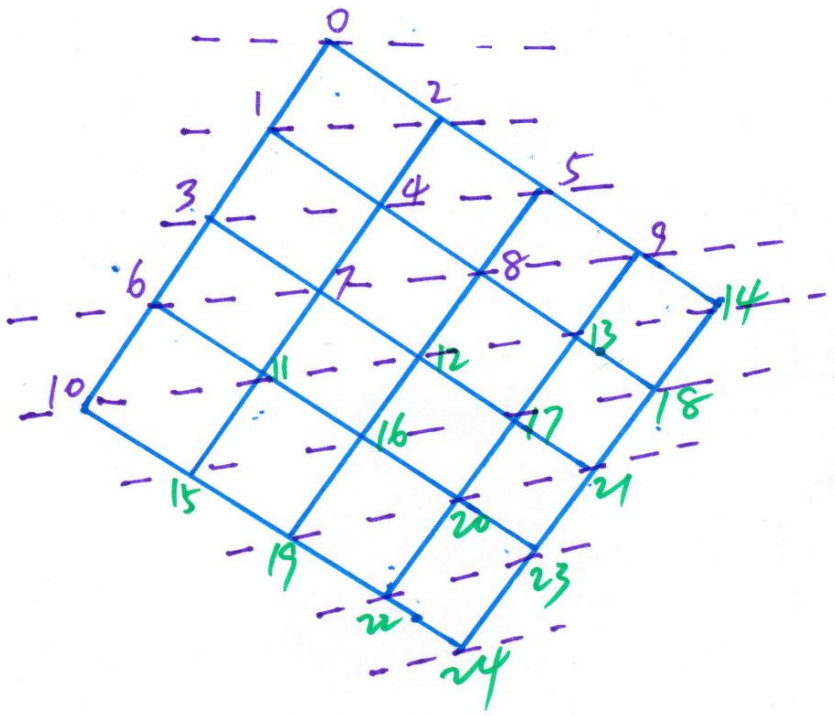

Figure 2.1: A m*n regular mesh(m = 5, n = 5)

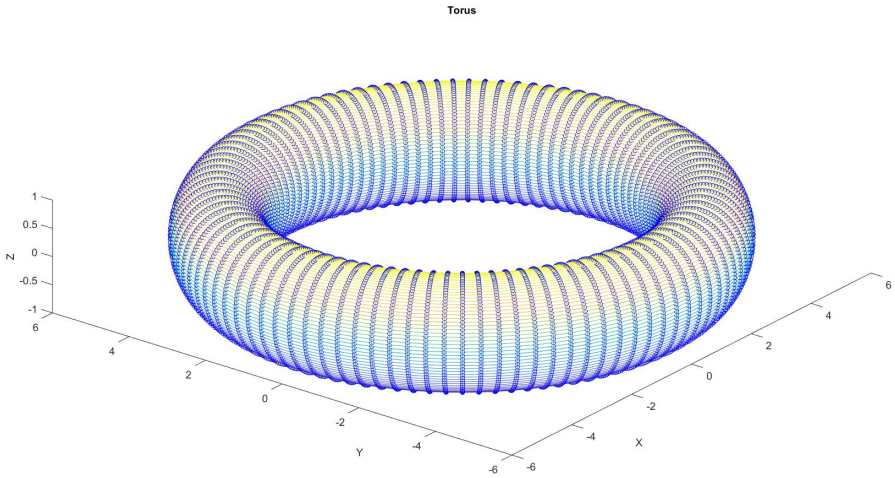

Figure 2.2: A toroidal mesh with grid unit cores

- The network environment is homogeneous, that is, all the processors have the same computation capacity.
- The link speeds between any two unit cores are identical.
- The number of outgoing ports in each processor is limited. In NOC(network on chip), the port number is fixed 4 or 5.
- The general graph's grid node's in-degree and out-degree is 4 or 5.

In the front-end assumption, the communication and the computation can be executed simultaneously. That is, upon receiving their respective load fractions, the processors start processing their own workload and rely all the other fractions to the next processor at the same time.

In the without front-end scenario, unlike the previous protocol, the processors now simultaneously receive the data and solely start to process it as soon as each processor receives its entire load assignment[5].

Thus, the problem becomes how to partition and schedule the workloads amongst the processors to get the minimum finish time. This study will concentrate on the speedup metric which is defined as the ratio of computation time on one processor to the computation time on the entire $m * n$ processors. To achieve the minimum solution is obtained by forcing the processors over a network to stop processing simultaneously. Intuitively, this is because the solution could be improved by transfer load from some busy processor to idle ones. The following notations and definitions are utilized:

Notations and Definitions

- D_i: The minimum number of hops from the node P_i to the nearest data load injection L.
- $level_i$: The Manhattan distance to the nearest data injection
- P_i: The ith processor. i should be from 0 to $m * n - 1$.
- L_i: The ith work load. i should be from 1 to k.
- α_0: The load fraction assigned to the root processor.
- α_i: The load fraction assigned to the ith processor.
- ω_i: The inverse computing speed on the ith processor.
- ω_{eq}: The inverse computing speed on an equivalent node collapsed from a regular mesh or toroidal.
- z_i: The inverse link speed on the ith link.
- T_{cp}: Computing intensity constant. The entire load can be processed in $\omega_i T_{cp}$ on the ith processor.
- T_{cm}: Communication intensity constant. The entire load can be transmitted in $z_i T_{cm}$ seconds over the ith link.
- $T_{f,n}$: The finish time of the whole regular network. Here $T_{f,n}$ is equal to $\omega_{eq} T_{cp}$.
- $T_{f,0}$: The finish time for the entire divisible load solved on the root processor. Here $T_{f,0}$ is equal to $1 \times \omega_0 T_{cp}$, that is $\omega_0 T_{cp}$.

- $\sigma = \frac{zT_{cm}}{\omega T_{cp}}$: The ratio between the communication speed to the computation speed, $0 < \sigma < 1$ [6] [7].
- $\sum_{i=1}^{k} L_i = 1$
- $\sum_{i=0}^{m*n-1} \alpha_i = 1$
- $Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

2.2 With Front End Scenario

The processors compute and transfer the data load simultaneously. First we consider about the $2*2$ regular mesh, $2*n$ regular mesh. After, we analyze a more general case $m*n$ regular mesh and obtain a general closed-form matrix presentation. Finally, we give a key principle to address this type of question. In addition, different data injection position, such as the corner, boundary and inner grid point are also discussed.

2.2.1 Regular Mesh

Data Injection on The Corner Processor

2*2 Regular Mesh

The L_1 is assigned on the corner unit core P_0 Fig. 2.3. The whole task are tackled by four processors P_0, P_1, P_2, P_3 together.

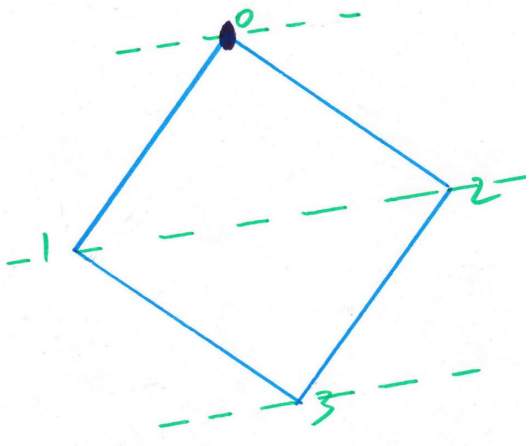

Figure 2.3: The 2*2 regular mesh and the data injection is on P_0

The processor P_0, P_1 and P_2 start to process its respective fraction at the same time. The port number of P_0 is limited, so the processor P_3 handles until the α_1 and α_2 are assigned completed to P_1 and P_2.

According to the divisible load theory[8], we can obtain the timing diagram Fig. 2.4.

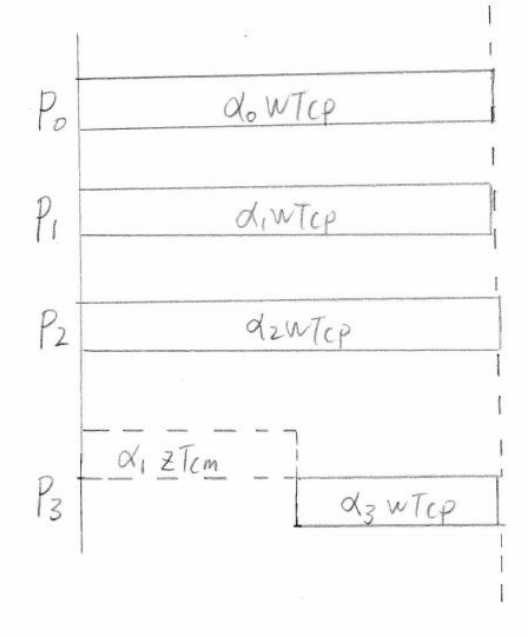

Figure 2.4: The timing diagram for 2*2 regular mesh and the data injection position is P_0

Based on the timing diagram, we can set the equations for the finish time of each processor to get ht optimal fragment size of loads:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.1) \\
\alpha_0 \omega T_{cp} = T_{f,m} & (2.2) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (2.3) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (2.4) \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.5) \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (2.6) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.7) \\
0 < \sigma < 1 & (2.8) \\
0 < \alpha_0 \leq 1 & (2.9) \\
0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 & (2.10)
\end{cases}
$$

We can obtain the

$$\alpha_1 = \alpha_2$$

The group of equations are represented by the matrix form:

$$
\begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.11}
$$

The matrix is represented as $A \times \alpha = b$. A is named as ***flow matrix***.

So the explicit solution is:

$$\begin{cases} \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (2.12) \\ \alpha_0 = \dfrac{1}{4-\sigma} & (2.13) \\ \alpha_1 = \dfrac{1}{4-\sigma} & (2.14) \\ \alpha_3 = \dfrac{1-\sigma}{4-\sigma} & (2.15) \end{cases}$$

The simulation result is illustrated:

In Fig. 2.5, P_0, P_1, P_2 three processors have the same data fraction workload, so the curve of α_0 and α_1 coincide. The figure says that the σ value grows and the α_3 drops simultaneously. In other words, the communication capacity drops down and there is less data workload assigned to P_3. Further, it means it will be economical to keep the load local nor distribute to other processor.

The speedup is:

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = 4 - \sigma$$

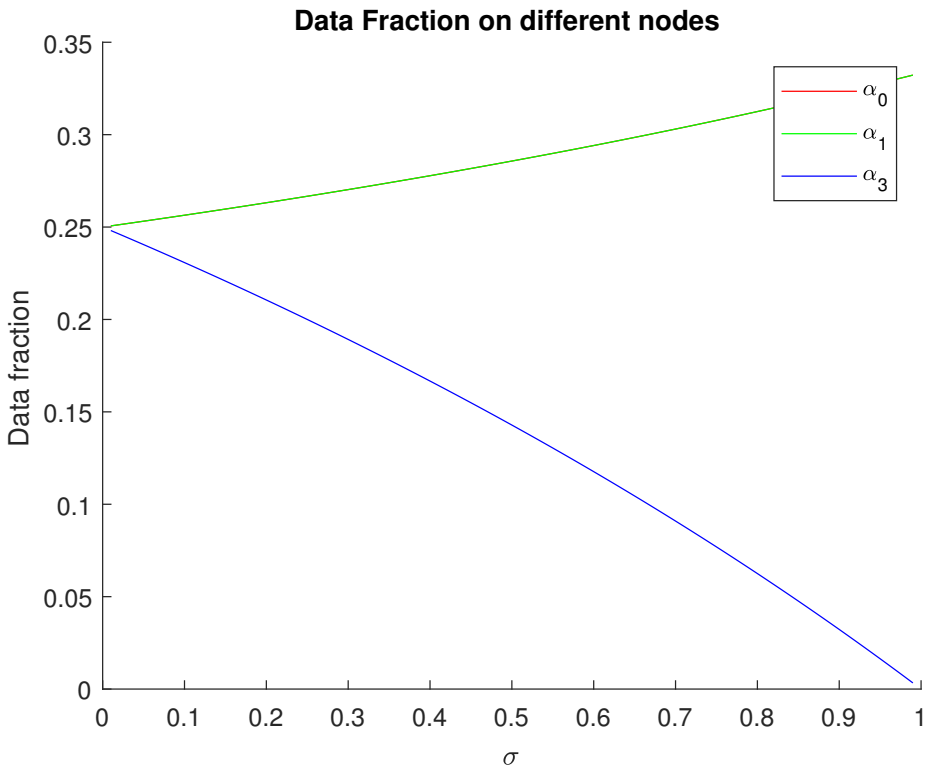

Figure 2.5: 2*2 regular mesh. α_0, α_1, α_2, α_3 value

2*3 Regular Mesh

In the $2*3$ Fig. 2.6 regular mesh, L_1 happens on processor P_0. There are 6 processors to take the responsibility.

P_0, P_1 and P_2 start processing the load at the same time. According the processor P_3 and P_4 start to work until they get the data from their parent processor P_1, P_2. Yet, the number of port of each processor is 4 or 5. So the fraction of α_3, α_4 are queued in the L_1. So the P_3 and P_4 have to wait the fraction of α_1 and α_2 are transmitted completely and then they will obtain their respective workload task fraction.

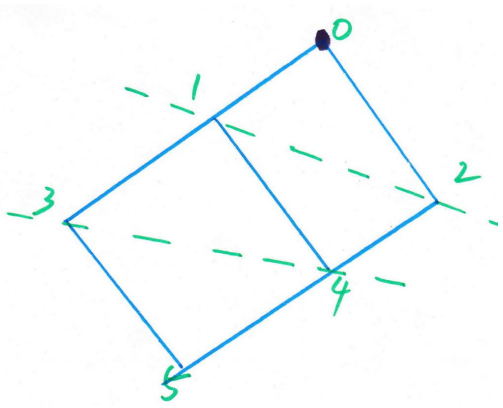

Figure 2.6: The 2*3 regular mesh and the data injection happens on corner processor P_0

The last unit core P_5 starts to execute until the work load fraction α_0, α_1, α_2, α_3, α_4 are transmitted completed. According to the divisible load theory[8], we can obtain the timing diagram Fig. 2.7.

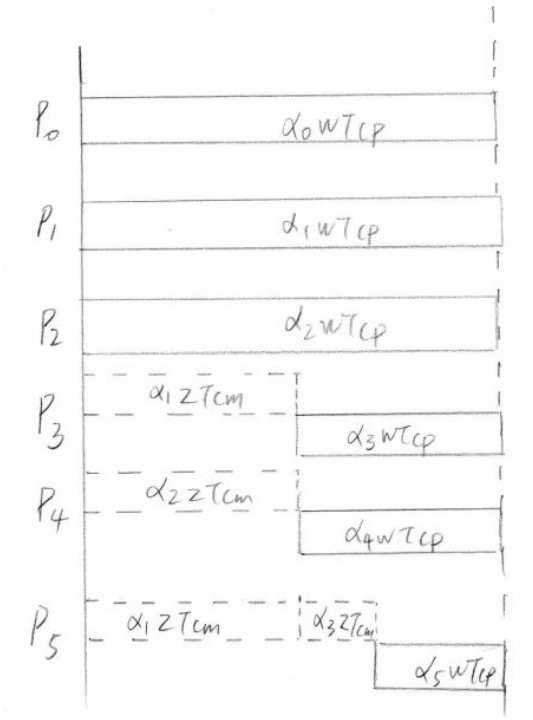

Figure 2.7: The timing diagram for a 2*3 regular mesh and the data injection happens on processor P_0

The equations as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.16) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (2.17) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (2.18) \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.19) \\
\alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.20) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.21) \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (2.22) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.23) \\
0 < \sigma < 1 & (2.24) \\
0 < \alpha_0 \leq 1 & (2.25) \\
0 \leq \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 < 1 & (2.26)
\end{cases}
$$

The flow matrix closed-form formula is:

$$
\begin{bmatrix}
1 & 2 & 2 & 1 \\
1 & -1 & 0 & 0 \\
0 & \sigma - 1 & 1 & 0 \\
0 & \sigma - 1 & \sigma & 1
\end{bmatrix}
\times
\begin{bmatrix}
\alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5
\end{bmatrix}
=
\begin{bmatrix}
1 \\ 0 \\ 0 \\ 0
\end{bmatrix}
\tag{2.27}
$$

The explicit solution is:

$$\begin{cases} \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (2.28) \\ \alpha_0 = \dfrac{1}{\sigma^2 - 4 \times \sigma + 6} & (2.29) \\ \alpha_1 = \dfrac{1}{\sigma^2 - 4 \times \sigma + 6} & (2.30) \\ \alpha_3 = \dfrac{1 - \sigma}{\sigma^2 - 4 \times \sigma + 6} & (2.31) \\ \alpha_5 = \dfrac{\sigma^2 - 2 \times \sigma + 1}{\sigma^2 - 4 \times \sigma + 6} & (2.32) \end{cases}$$

The simulation result are shown in Fig. 2.8. P_0, P_1 have the same fraction so their curves of α_0 and α_1 coincide.

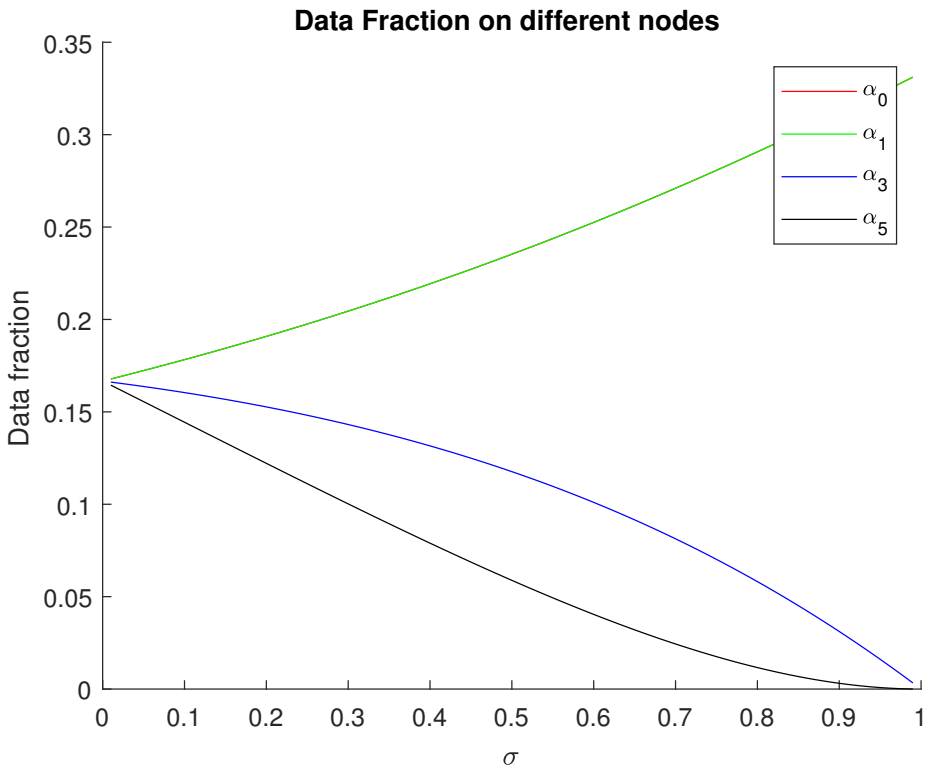

Figure 2.8: 2*2 regular mesh. α_0, α_1, α_3, α_5 data fraction value

2*n Regular Mesh

The $2*n$ Fig. 2.9 homogeneous regular mesh address L_1 at the same time and L_1 happens on P_0.

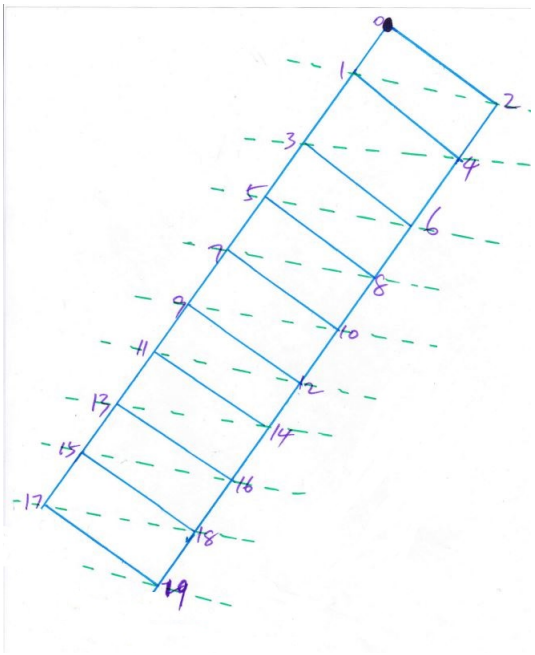

Figure 2.9: The 2*n (n = 10) regular mesh and the workload happens on P_0

Similarly to the analysis of Fig. 2.4 and Fig. 2.7, the timing diagram for Fig. 2.9 is shown in Fig. 2.10

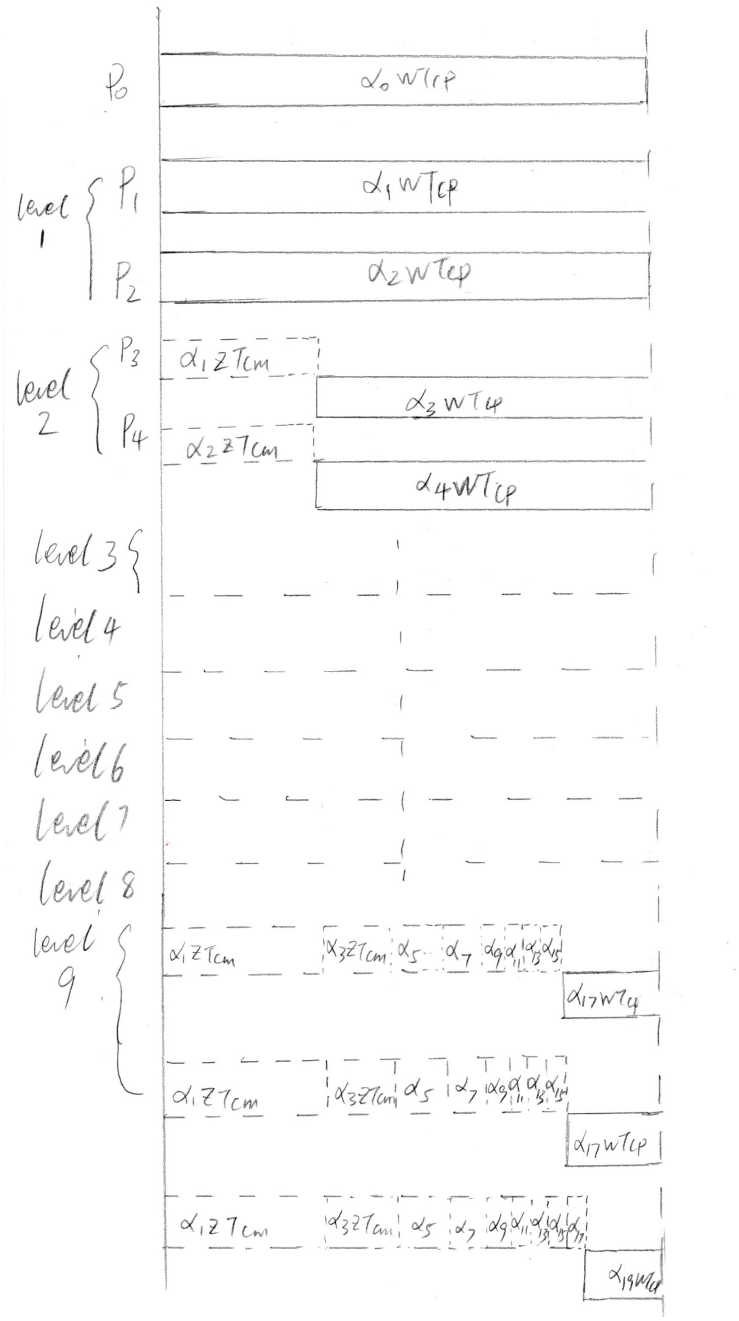

Figure 2.10: The timing diagram for 2*10 regular mesh and the data injection happens on P_0

The equations are presented as:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.33) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (2.34) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (2.35) \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.36) \\
\alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.37) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.38) \\
\vdots & (2.39) \\
(\alpha_1 + \alpha_3 + \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (2.40) \\
\alpha_0 + \cdots + \alpha_{2\times n-1} = 1 & (2.41) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.42) \\
0 < \sigma < 1 & (2.43) \\
0 < \alpha_0 \leq 1 & (2.44) \\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (2.45)
\end{cases}
$$

The flow matrix closed-form is shown:

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{2.46}$$

According to the ***Cramer's rule***,the explicit solution for the group of equations is:

$$\begin{cases} \alpha_i = \dfrac{\det A_i^\star}{\det A} \end{cases} \tag{2.47}$$

where $A_i^\star$ is the matrix formed by replacing the i-th column of A by the column vector b.

$$\alpha_0 = \frac{\det A_0^\star}{\det A}$$

$$A_0^\star = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{2.48}$$

$$\det A_0^\star = \frac{-1}{\det A}$$

Finally, the speedup is:

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

.

Further, we prove the matrix $\det A \neq 0$.

$$C = \begin{bmatrix} -1 & 0 & \cdots & 0 & 0 & 0 \\ \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{2.49}$$

C is a lower triangular matrix and the diagonal elements are not 0. So C is non-degenerate, that is, the matrix is column linear independence.

$$A = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \xrightarrow[\text{Reduction}]{\text{Column}} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & -3 & -2 & \cdots & -2 & -2 & -1 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix}$$

$$\xrightarrow[\text{Reduction}]{\text{Row}} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & -3 & -2 & \cdots & -2 & -2 & -1 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix}$$

$$C^{\star} = \begin{bmatrix} -3 & -2 & \cdots & -2 & -2 & -1 \\ \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{2.50}$$

$C^{\star}$ is still column linear independence.Considering $0 < \sigma < 1$, the flow

matrix is full rank. So $\det A \neq 0$.

m*n Regular Mesh

Considering a general $m * n$ regular mesh, such as Fig. 2.11 Fig. 2.12.

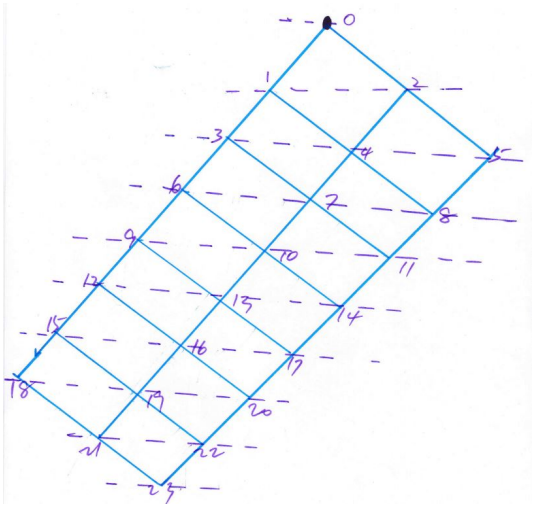

Figure 2.11: 3*8 regular mesh. The data injection position is P_0

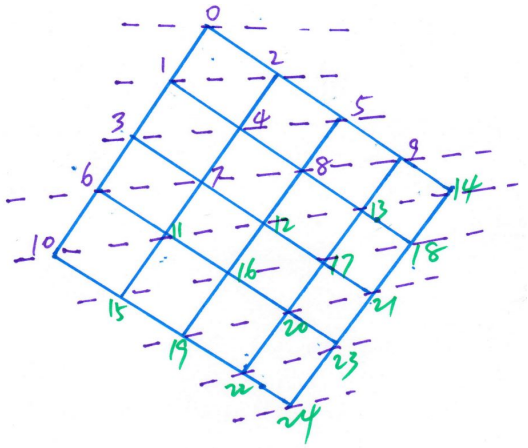

Figure 2.12: 5*5 regular mesh. The data injection position is processor P_0

After three user cases' investigation, we find a crucial rule:

$$\forall D_i = D_j, \quad then \quad \alpha_i = \alpha_j, \quad 0 \leq i, j \leq m * n - 1$$

Utilizing the rule, we obtain the closed-form matrix equations for Fig. 2.11:

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_9 \\ \alpha_{12} \\ \alpha_{15} \\ \alpha_{18} \\ \alpha_{21} \\ \alpha_{23} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.51}$$

Also, the matrix equation for Fig. 2.12:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.52}$$

We use the similar method to prove $\det A \neq 0$,so the speedup is:

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = -\det A$$

.

Data Injection On the Boundary Processor

After explain the corner scenario, we extend the rule to boundary processor condition.

If the single data injection roots on the boundary processor, for example Fig. 2.13.

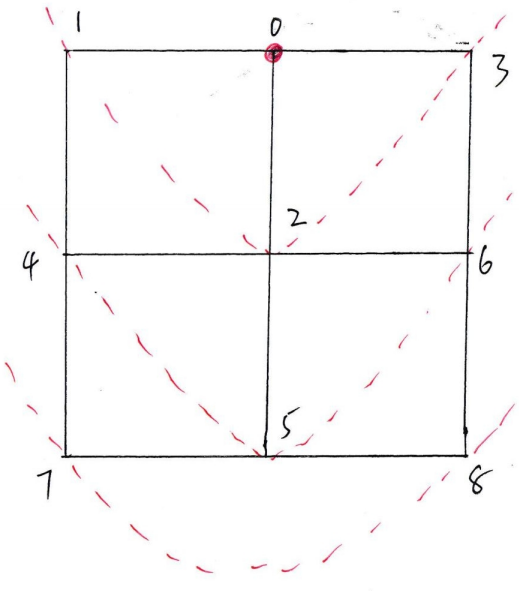

Figure 2.13: The 3*3 regular mesh and the data injection position is P_0

The timing diagram is Fig. 2.14:

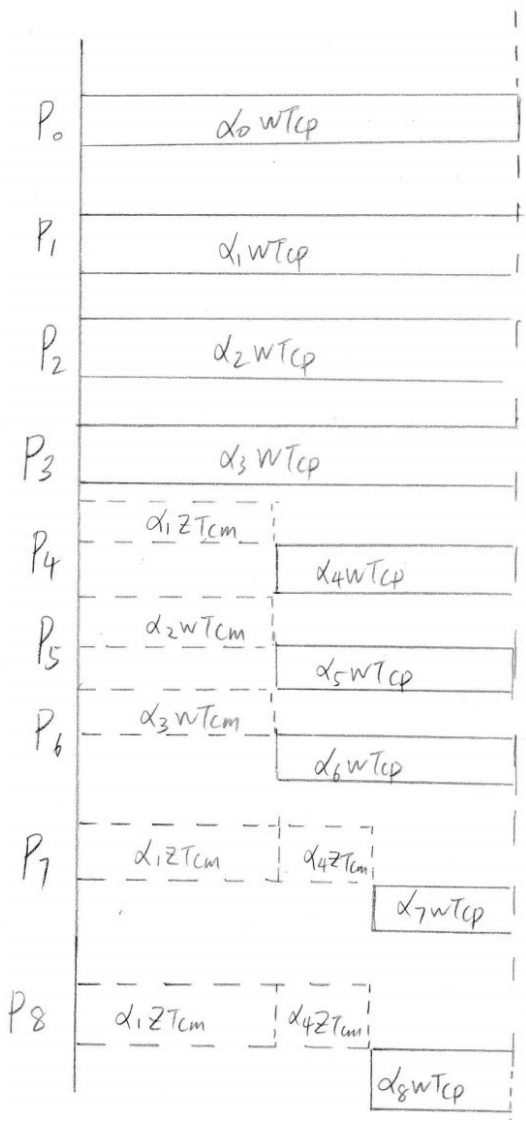

Figure 2.14: The timing diagram for 3*3 regular mesh and the data injection occurs on the boundary processor P_0

The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.53)\\
\alpha_1 \omega T_{cp} = T_{f,m} & (2.54)\\
\alpha_2 \omega T_{cp} = T_{f,m} & (2.55)\\
\alpha_3 \omega T_{cp} = T_{f,m} & (2.56)\\
\alpha_1 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.57)\\
\alpha_2 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.58)\\
\alpha_3 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.59)\\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.60)\\
(\alpha_2 + \alpha_5) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.61)\\
\alpha_0 + \cdots + \alpha_8 = 1 & (2.62)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.63)\\
0 < \sigma < 1 & (2.64)\\
0 < \alpha_0 \leq 1 & (2.65)\\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_8 < 1 & (2.66)\\
& (2.67)
\end{cases}
$$

And the matrix form is :

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_4 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.68}$$

The explicit solution is:

$$\begin{cases} \alpha_0 = \dfrac{1}{9-7\times\sigma+2\times\sigma^2} & (2.69) \\ \alpha_1 = \dfrac{1}{9-7\times\sigma+2\times\sigma^2} & (2.70) \\ \alpha_4 = \dfrac{1-\sigma}{9-7\times\sigma+2\times\sigma^2} & (2.71) \\ \alpha_7 = \dfrac{(1-\sigma)^2}{9-7\times\sigma+2\times\sigma^2} & (2.72) \\ & (2.73) \end{cases}$$

The simulation result is shown: P_0 and P_1 have the same α, so the curve of α_0 and α_1 coincide.

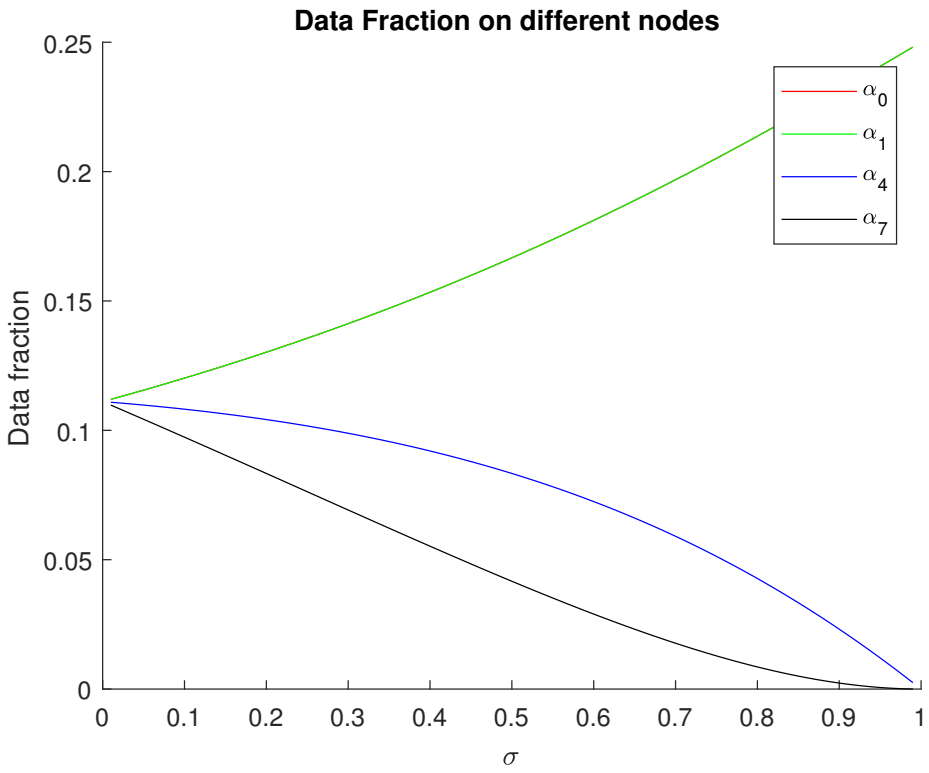

Figure 2.15: The data fraction simulation result of 3*3 regular mesh and the data injection happens on the boundary P_0

Data Injection On The Inner Grid Processor

L_1 is on the inner grid processor P_0Fig. 2.16,

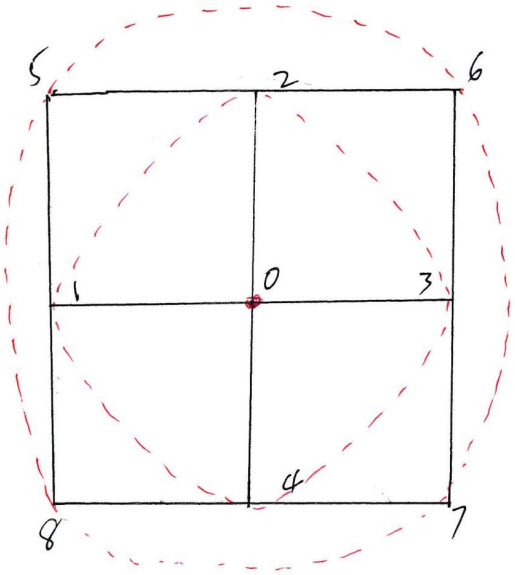

Figure 2.16: 3*3 regular mesh. The data injection position is inner grid point P_0

The timing diagram for this user case is illustrated as Fig. 2.17:

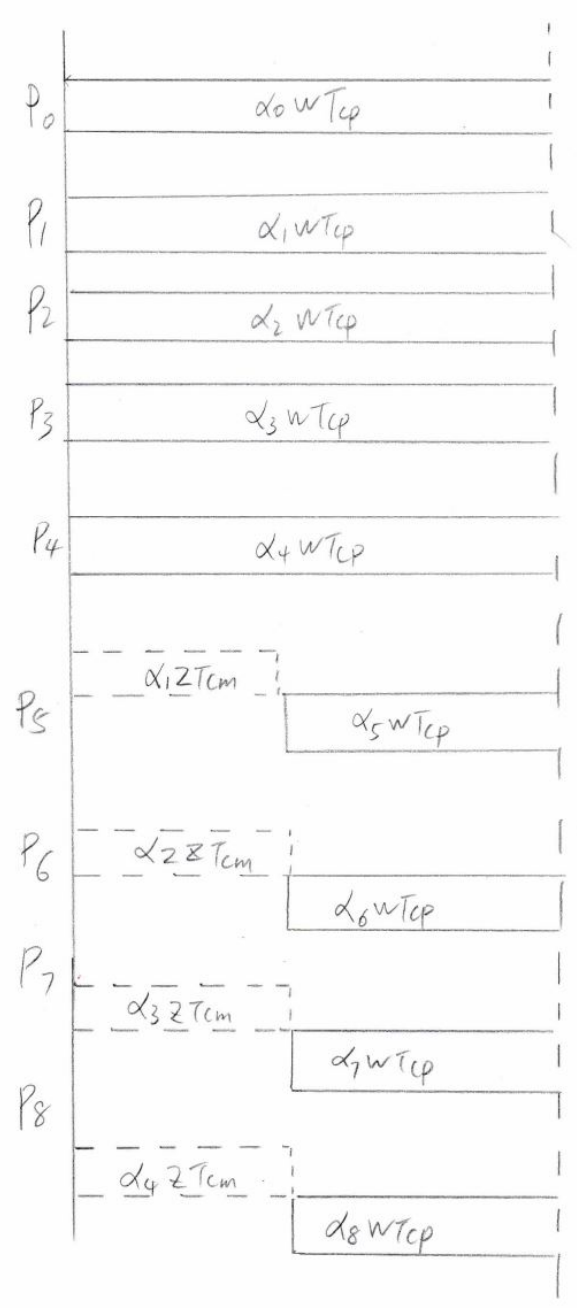

Figure 2.17: The timing diagram for 3*3 regular mesh and the data injection is inner grid P_0

The group of equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.74) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (2.75) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (2.76) \\
\alpha_3 \omega T_{cp} = T_{f,m} & (2.77) \\
\alpha_4 \omega T_{cp} = T_{f,m} & (2.78) \\
\alpha_1 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.79) \\
\alpha_1 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.80) \\
\alpha_1 z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.81) \\
\alpha_1 z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.82) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.83) \\
0 < \sigma < 1 & (2.84) \\
0 < \alpha_0 \leq 1 & (2.85) \\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_8 < 1 & (2.86)
\end{cases}
$$

The matrix form is :

$$
\begin{bmatrix} 1 & 4 & 4 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.87}
$$

The simulation result for the inner grid is Fig. 2.18: P_0 and P_1 have the same α value, so the curve of α_0 and α_1 coincide.

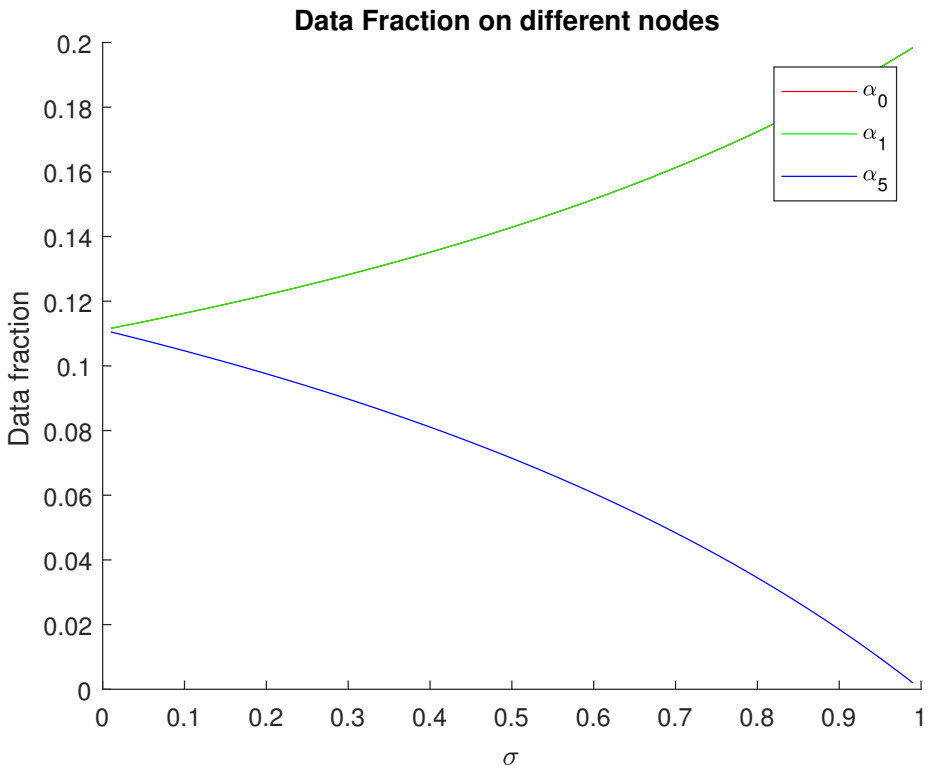

Figure 2.18: 3*3 regular mesh. The data injection position is inner grid point P_0

2.2.2 Torus mesh

According to the paper[9], there are three different torus mesh. In this paper, out intent is not to propose one model to "fit all" problems but rather to indicate one normal case. Other situation can be extend use the same rule. The toroidal network is also a regular complex in that each polygonal face has the same number of links and each node is connected to the same number of links.

We consider the toroidal network is Fig. 2.19 and Fig. 2.20.

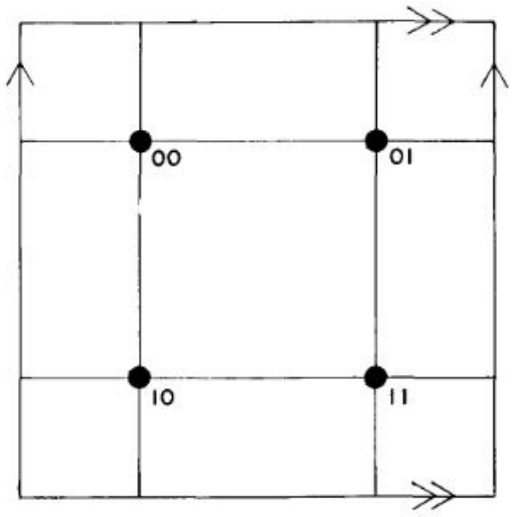

Figure 2.19: The rectangular toroidal network

Based on the rule, this question is transfered to that given a D_i and I find the number of instance.

- We assume the toroidal processor number is $m * n$, in other words, the width is m and the height is n.
- L to present the load injection.

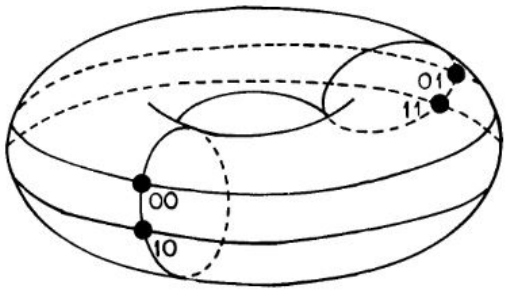

Figure 2.20: The rectangular toroidal network

- L_x is the L's X coordinate.
- L_y is the L's Y coordinate.
- D_i to present the shortest Manhattan distance between the grid node i and L.
- D_x is the X coordinate shortest Manhattan distance.
- D_y is the Y coordinate shortest Manhattan distance.
- N_x is the node i's X coordinate.
- N_y is the node i's Y coordinate.

$$\begin{cases} D = D_x + D_y & (2.88) \\ D_x = \min\{\|N_x - L_x\|, m - \|N_x - L_x\|\} & (2.89) \\ D_y = \min\{\|N_y - L_y\|, n - \|N_y - L_y\|\} & (2.90) \end{cases}$$

In the $m*n(m=6,n=6)$ toroidal networks, L happens on grid position $(4,2)$. We calculate the D_i matrix Table 2.1 by breadth first search(***BFS***) algorithm.

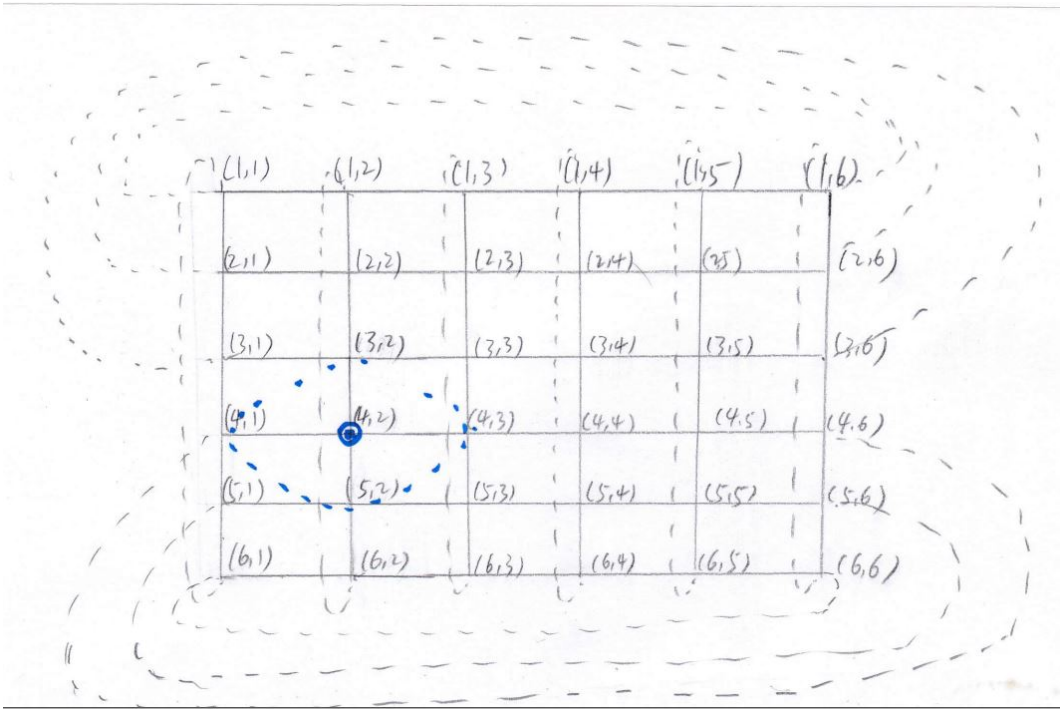

Figure 2.21: The m*n toroidal network and the data injection is $P_{4,2}$

D_i	Number
0	1
1	4
2	8
3	10
4	8
5	4
6	1

Table 2.1: The processor number of various D_i

And the D_i matrix as follow in table Table 2.1

The matrix closed-form is

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.91}$$

The simulation result is :

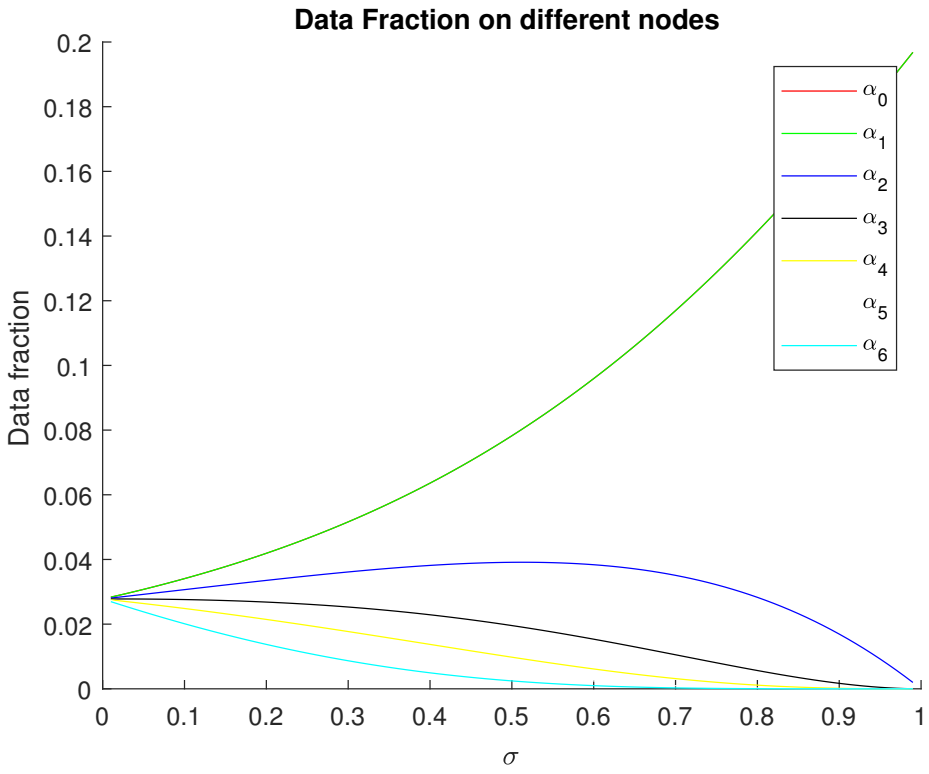

Figure 2.22: The data fraction deployed based on the radius value

From Fig. 2.22, we see as the σ value grows, more and more workload are assigned to the $P_{4,2}$ and its one hop neighbors. That is, as the communication ability goes down, the economical method is to process the data locally.

2.2.3 General Case

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.92}$$

The m_1, m_2, $\cdots$, m_n are the number of processors on the $level_1$, $level_2$, $\cdots$, $level_n$. Also, the α_{l0}, α_{l1}, $\cdots$, α_{ln} are corresponding workload fraction.

Finally, the speedup is:

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = -\det A$$

.

2.3 Without Front End Scenario

2.3.1 Regular Mesh

Data Injection on Corner Processor

2*2 Regular Mesh

This subsection concerns the processors without front-end. Because of without front-end, the processors simultaneously receive the data and solely start to process it as soon as each processor receives its entire load assignment. We consider the timing diagram for Fig. 2.3, Fig. 2.6, Fig. 2.9 and so on. In addition, we also give the new closed-form matrix equations for the previous user cases.

Also, the rule also plays a dominate role in establish the mathematics model. The timing diagram of Fig. 2.3 is shown:

P_0 starts to process the assigned workload and it starts to transfer the α_1, α_2 and α_3 fraction workload after it totally receive its α_0 task. That is, P_1 and P_2 are idle until the L_1 finish its data injection to P_0.The similar situation happens to P_1 and P_2 and they both starts to transmit the α_3 after they totaly receive the appropriate workload. In other words, P_3 has to wait until the previous two level processors obtains their own data.

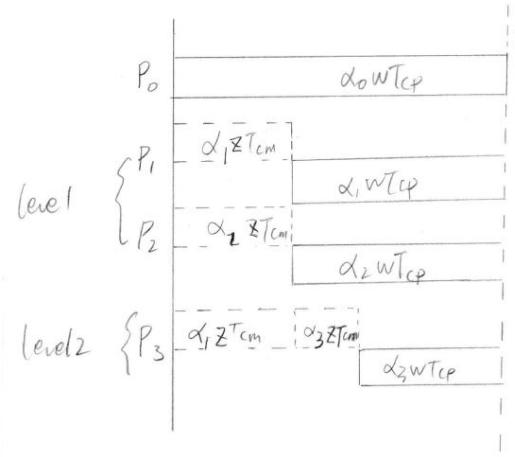

Figure 2.23: The timing diagram for 2*2 regular mesh without front end.

The corresponding group of equations are as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.93) \\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (2.94) \\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (2.95) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.96) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.97) \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (2.98) \\
0 < \sigma < 1 & (2.99) \\
0 < \alpha_0 \leq 1 & (2.100) \\
0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 & (2.101)
\end{cases}
$$

The group of equations imply that

$$\alpha_1 = \alpha_2$$

Further, the equations test and verify the rule again.

The matrix closed-from is presented as:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.102}$$

The simulation result for Fig. 2.3 is provided in Fig. 2.24:

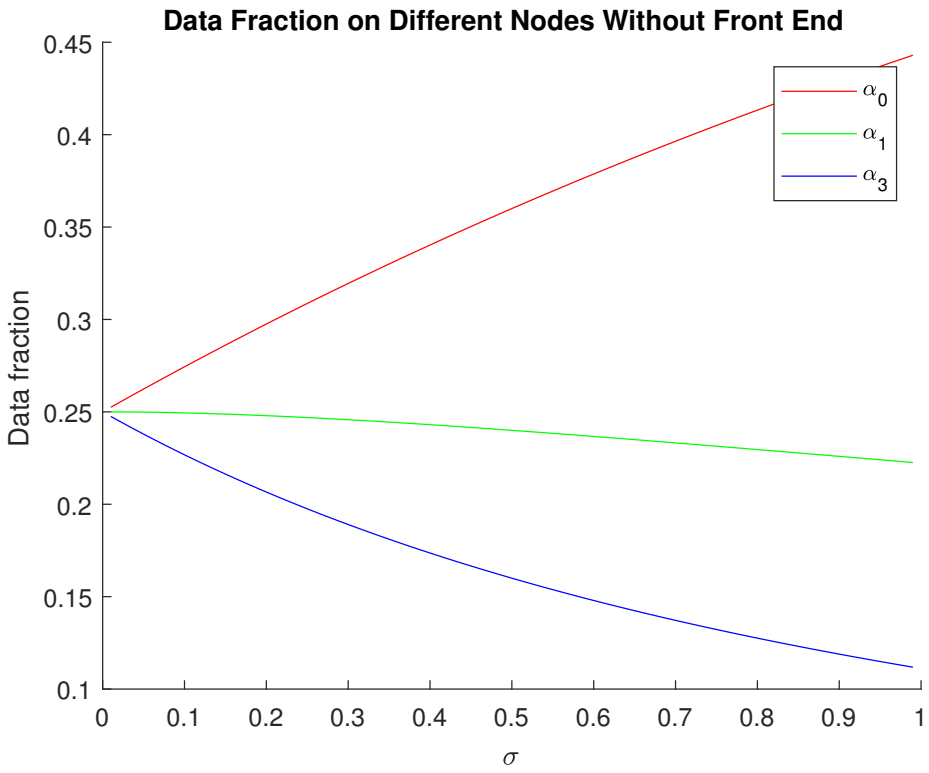

Figure 2.24: The data fraction deployed based on the radius value

Fig. 2.24 explains that as the σ value grows up, the fraction is assigned to P_0 increases and the fraction are distributed to $level_1$ and $level_2$ reduce. In other words, if the communication capability decreases, there are more data processed locally, which is reasonable. If the ability of the link degrades asymptotically equal to the processor computation capacity, there is solely 10% data is deployed to the $level_2$. In addition, if the $\sigma > 1$, it means the link transmitting power is less than the processor's processing ability. In this scenario, keeping the data locally is more economical than transmitting it.

2*3 Regular Mesh

P_0 starts to process the assigned workload and it starts to transfer the α_1, α_2 , α_3, α_5, α_5 fraction workload after it totally receive its α_0 task. That is, P_1 and P_2 are idle until the L_1 finish its data injection to P_0.$Accordingtothelevel$1The similar situation happens to P_1 and P_2 and they both starts to transmit the α_3 after they totally receive the appropriate workload. In other words, P_3 has to wait until the previous three levels, $level_0$, $level_1$ and $level_2$ processors obtain their own data.

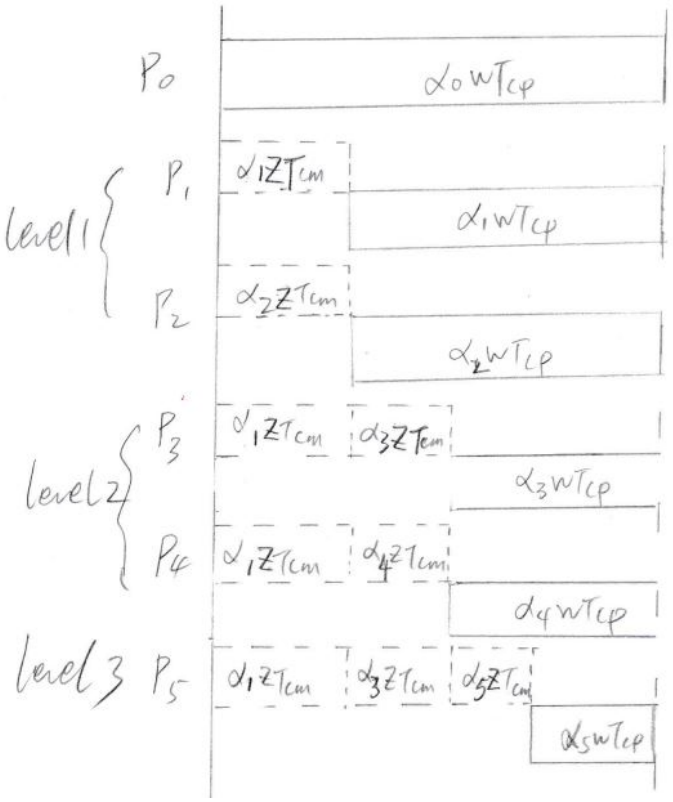

Figure 2.25: The timing diagram for 2*3 regular mesh without front end.

In addition, the group of equations are as follows:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (2.103) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (2.104) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (2.105) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.106) \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.107) \\ (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.108) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.109) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (2.110) \\ 0 < \alpha_0 \leq 1 & (2.111) \\ 0 \leq \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 < 1 & (2.112) \end{cases}$$

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.113)$$

The speedup is

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

The simulation result is:

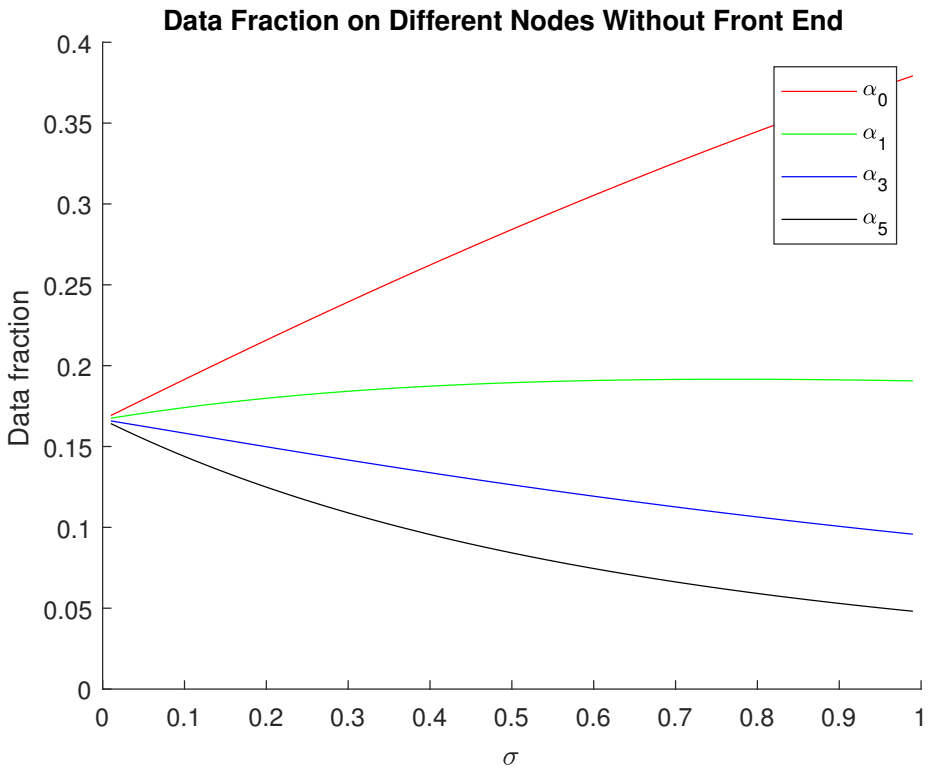

Figure 2.26: The data fraction deployed based on the radius value

2*n Regular Mesh

Considering Fig. 2.9, the equations are demonstrated as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.114) \\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (2.115) \\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (2.116) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.117) \\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.118) \\
(\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.119) \\
\vdots & (2.120) \\
(\alpha_1 + \alpha_3 + \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (2.121) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.122) \\
0 < \sigma < 1 & (2.123) \\
0 < \alpha_0 \leq 1 & (2.124) \\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (2.125)
\end{cases}
$$

The matrix form for the group of equations are:

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -(\sigma+1) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.126}$$

$$-\sigma - 2 = \epsilon$$

$$\sigma^\star - 2 = \beta$$

$$A = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \xrightarrow[\text{Reduction}]{\text{Column}} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & \beta & -2 & \cdots & -2 & -2 & -1 \\ 1 & \epsilon & \beta & \cdots & 0 & 0 & 0 \\ 1 & \epsilon & \epsilon & \beta & 0 & \cdots & 0 \\ 1 & \epsilon & \epsilon & \epsilon & \beta & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \beta \end{bmatrix}$$

$$\xrightarrow[\text{Reduction}]{\text{Row}} \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \beta & -2 & \cdots & -2 & -2 & -1 \\ 0 & \epsilon & \beta & \cdots & 0 & 0 & 0 \\ 0 & \epsilon & \epsilon & \beta & 0 & \cdots & 0 \\ 0 & \epsilon & \epsilon & \epsilon & \beta & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \beta \end{bmatrix}$$

$$C = \begin{bmatrix} \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 \\ -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix}$$

$0 < \sigma < 1$, then $-2 < \sigma^\star < -1$, which means C is column linear independent, after column and row reduction. $C^\star$ is still full rank. So the matrix is full rank, that is, $\det A \neq 0$ and $\det A^* \neq 0$.

So the speedup is

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = \frac{\det A}{\det A^\star} = \left| \frac{\det A}{(\sigma^\star)^{n-1}} \right|$$

m*n Regular Mesh

Referring to Fig. 2.12, we utilize $\sigma^\star$ to present the $-(\sigma+1)$. The matrix closed-form is:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.127)$$

Data Injection on Boundary Processor

Fig. 2.13 shows an example of boundary processor P_0 receiving the L_1 The timing diagram for Fig. 2.13 is Fig. 2.27.

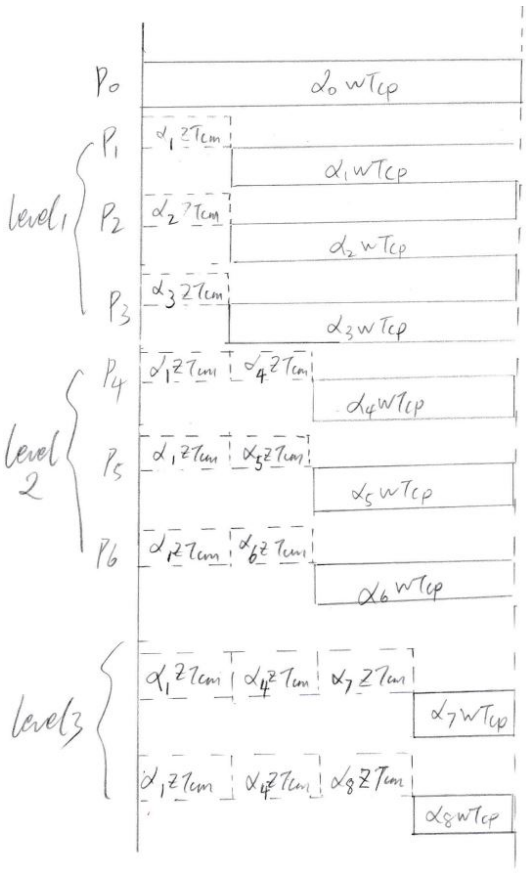

Figure 2.27: The timing diagram for 3*3 boundary data injection on P_0

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.128) \\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (2.129) \\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (2.130) \\
\alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.131) \\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.132) \\
(\alpha_2 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.133) \\
(\alpha_3 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.134) \\
(\alpha_1 + \alpha_4 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.135) \\
(\alpha_1 + \alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.136)
\end{cases}
$$

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_4 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.141}$$

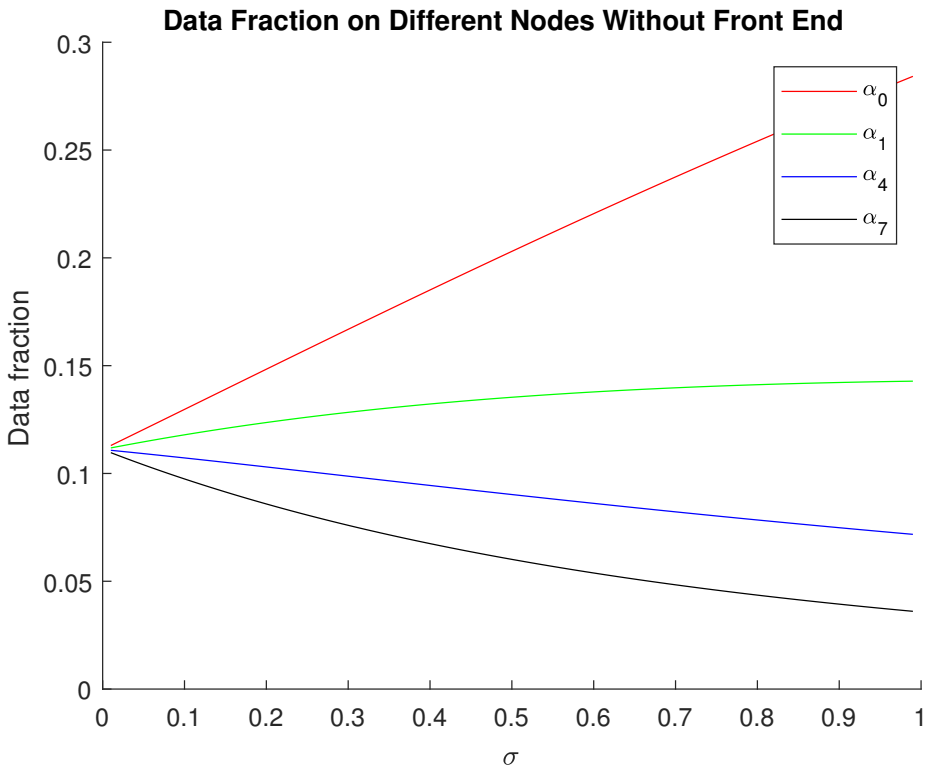

Figure 2.28: The fraction curve for 3*3 boundary data injection on P_0

Data Injection on The Inner Grid Processor

The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (2.142)\\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (2.143)\\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (2.144)\\
\alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.145)\\
\alpha_4 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.146)\\
(\alpha_1 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.147)\\
(\alpha_2 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.148)\\
(\alpha_3 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.149)\\
(\alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.150)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (2.151)\\
0 < \sigma < 1 & (2.152)\\
0 < \alpha_0 \leq 1 & (2.153)\\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 & (2.154)
\end{cases}
$$

The matrix closed-form is:

$$
\begin{bmatrix} 1 & 4 & 4 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad (2.155)
$$

The simulation result shows:

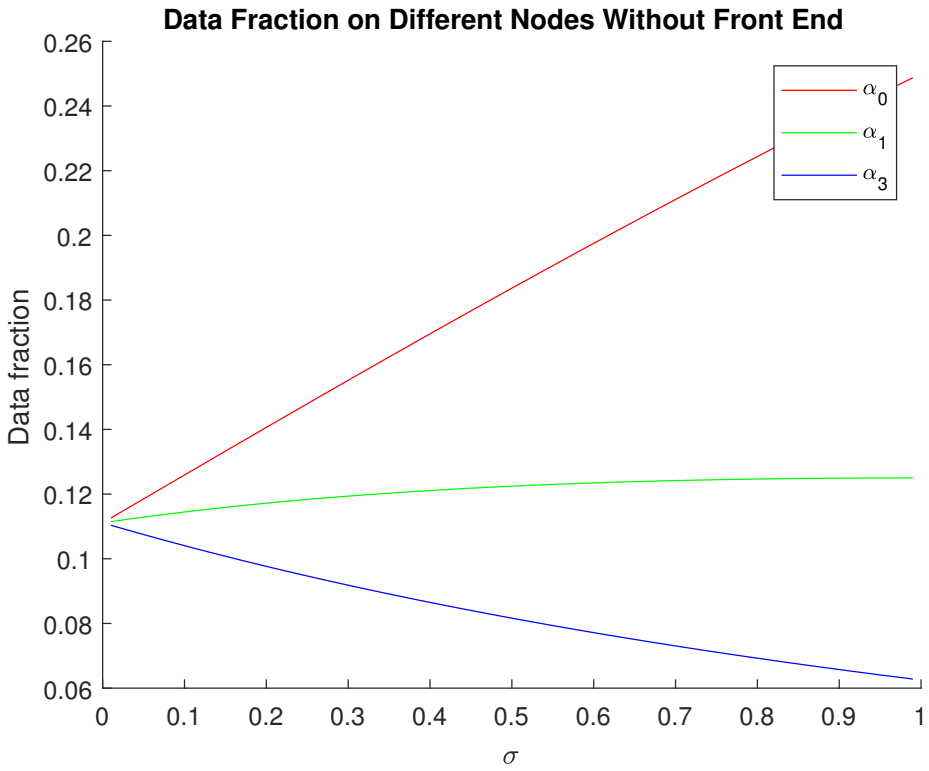

Figure 2.29: The timing diagram for 3*3 inner grid injection P_0

2.3.2 Toroidal Mesh

We utilize the $\sigma^\star$ to present $-(\sigma+1)$. The matrix closed-form is:

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.156}$$

The simulation result is :

2.3.3 General Case

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -(\sigma+1) & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -(\sigma+1) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.157}$$

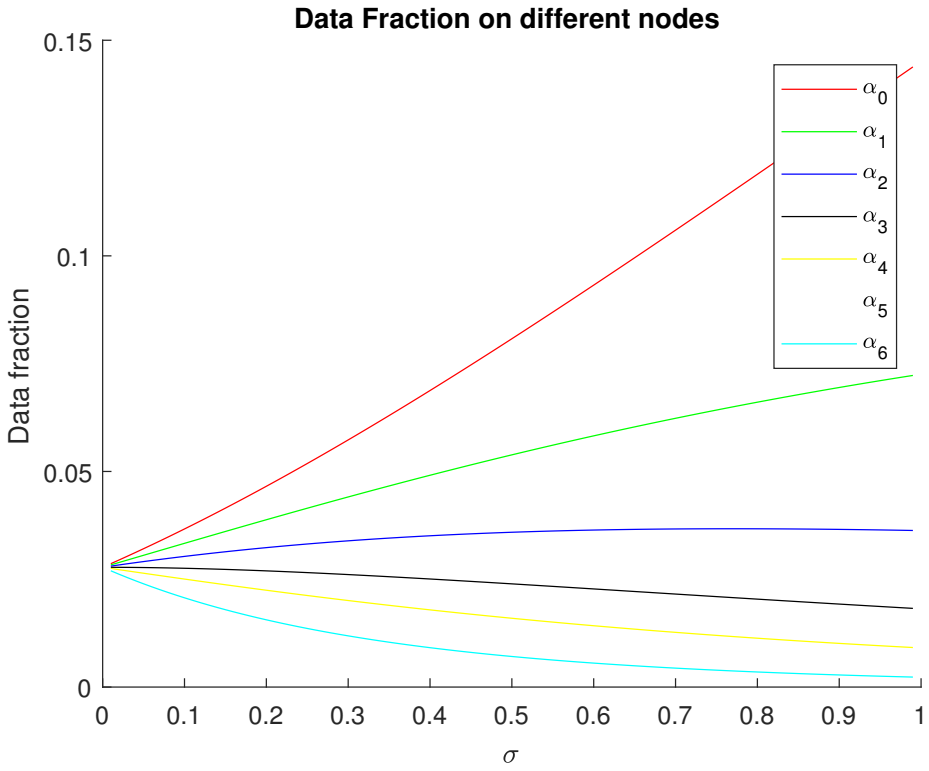

Figure 2.30: The data fraction deployed based on the radius value

The m_1, m_2, $\cdots$, m_n are the number of processors on the $level_1$, $level_2$, $\cdots$, $level_n$. Also, the α_{l0}, α_{l1}, $\cdots$, α_{ln} are corresponding workload fraction.

2.4 Comparison Result Between Front End Processor and Without Front End Processor

In the legend of figures, we use F to present the front-end situation and use NF to presents the without front-end processors setting, for example $F\alpha_0$ means the α_0 data fraction deployed to P_0, if the processor has front-end. $NF\alpha_0$ means the α_0 data fraction deployed to P_0, if the processor is without front-end setting.

2.4.1 Regular Mesh

On the Corner Processor

Fig. 2.31 says that P_0 takes more assigned task in without front-end scenario than front-end processor situation. As the σ value goes up, the fractions are deployed to the deeper levels dropping down. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to P_3 in the front-end assumption, yet in the without front-end situation, there is still about 10% data fraction are communicated to P_3.

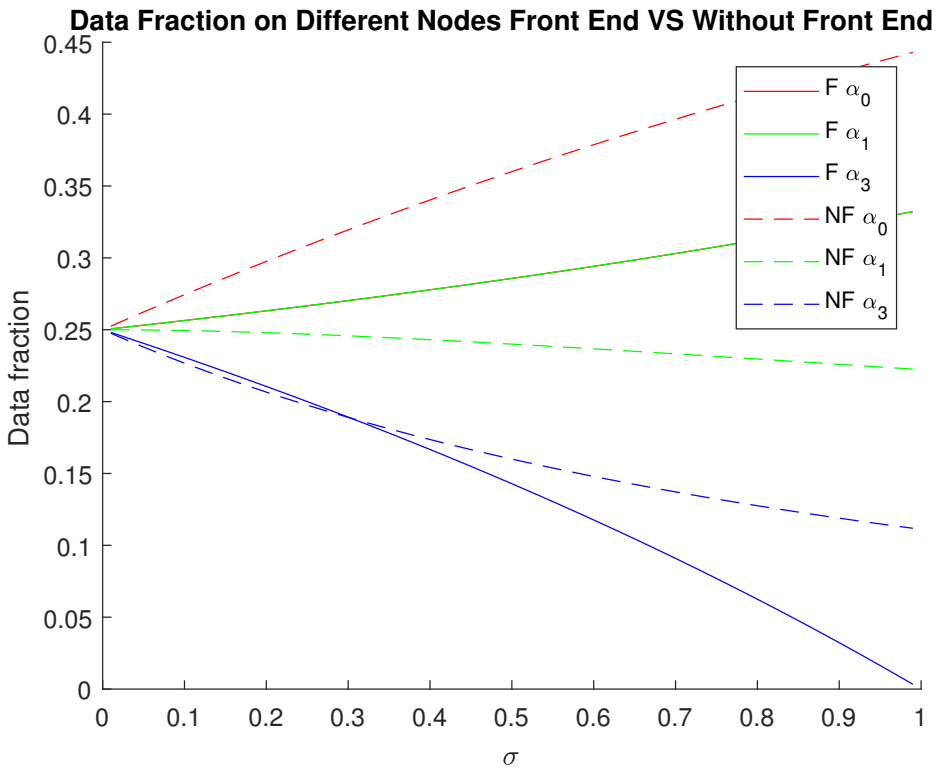

Figure 2.31: The comparing result between front-end processor with without front-end processor in 2*2 regular mesh

Fig. 2.32 says that P_0 takes more assigned task in without front-end scenario than front-end processor situation. As the σ value goes up, the fractions are deployed to the deeper levels degrading. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_3$, that is, P_5 in the front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to P_5.

Fig. 2.33 says that P_0 takes more assigned task in without front-end scenario than front-end processor situation. As the σ value goes up, the fractions

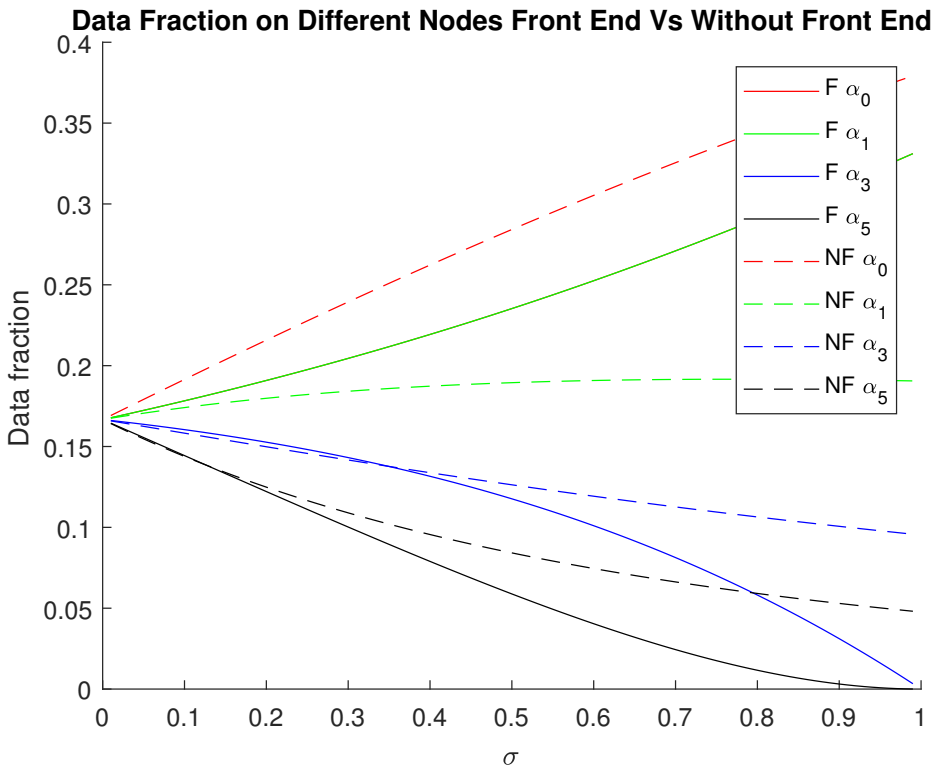

Figure 2.32: The comparing result between front-end processor with without front-end processor in 2*4 regular mesh

are deployed to the deeper levels dropping down. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_3$, that is, P_7 and P_8 in the front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to P_7 and P_8.

Fig. 2.34 says that P_0 takes more assigned task in without front-end scenario than front-end processor situation. As the σ value goes up, the fractions are deployed to the deeper levels dropping down. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_2$, that is, P_5, P_6, P_7 and

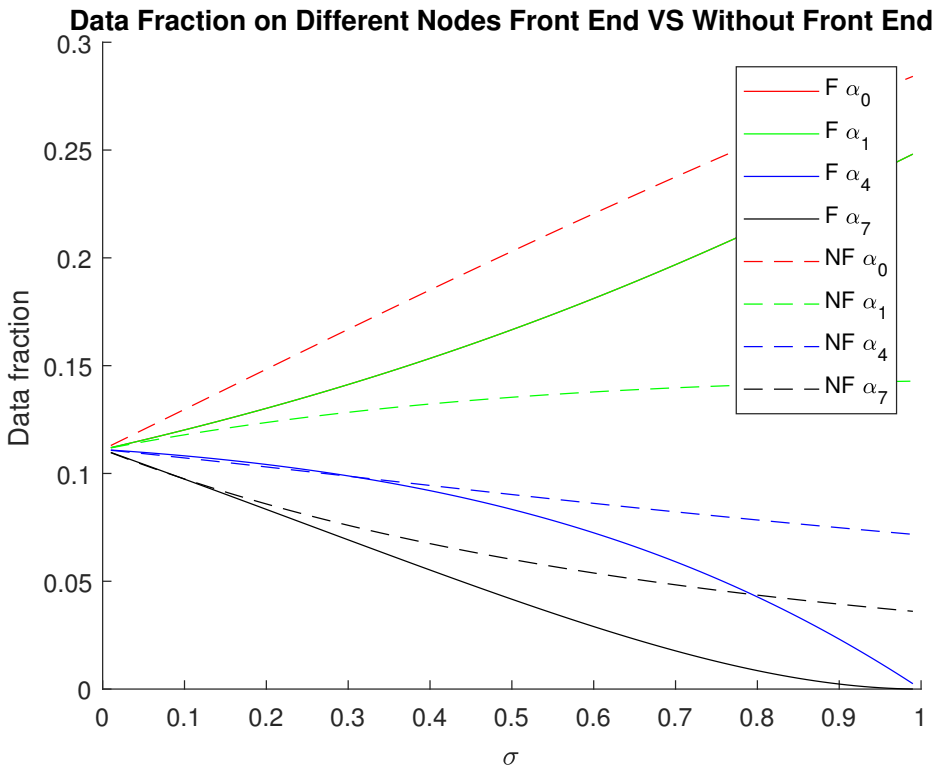

Figure 2.33: The comparing result between front-end processor with without front-end processor in 3*3 regular mesh injection on boundary processor

P_8 in the front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to P_5, P_6, P_7 and P_8.

Comparing with Fig. 2.33, P_0 takes less workload in inner grid position than boundary data injection. The reason is there are 4 neighbor processors on the $level_1$, yet there is solely three processors on $level_1$ in boundary plan. In other words, the $level_1$ plays a critical role in data deployed.

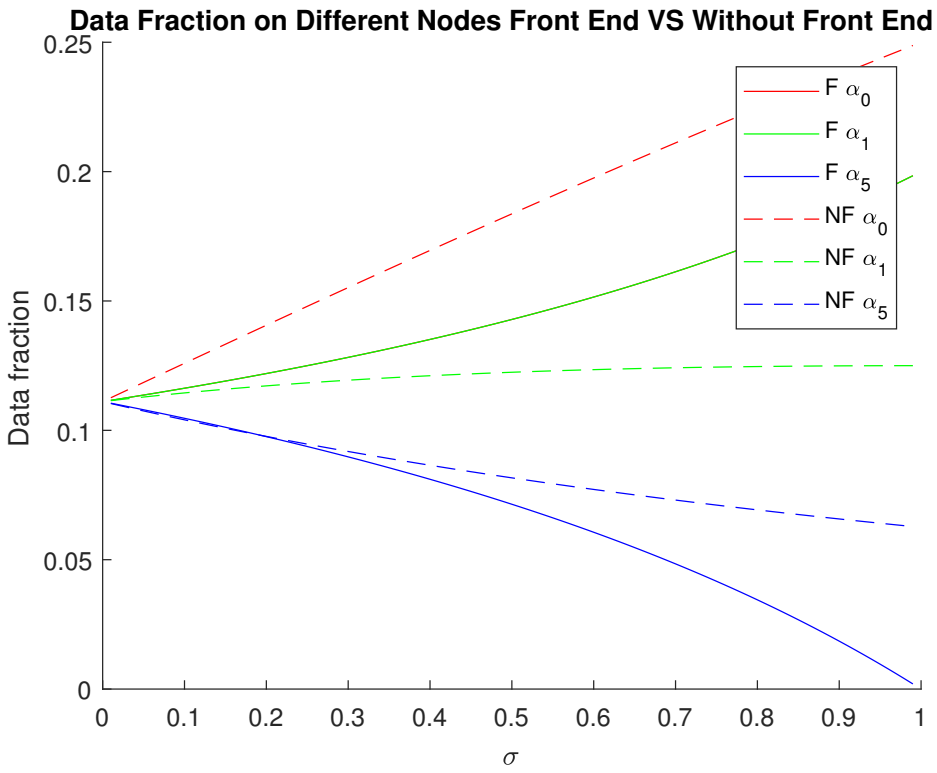

Figure 2.34: The comparing result between front-end processor with without front-end processor in 3*3 regular mesh injection on inner grid processor

2.4.2 Toroidal

This figure says the simulation difference result happens between front-end processors and without front-end processors.

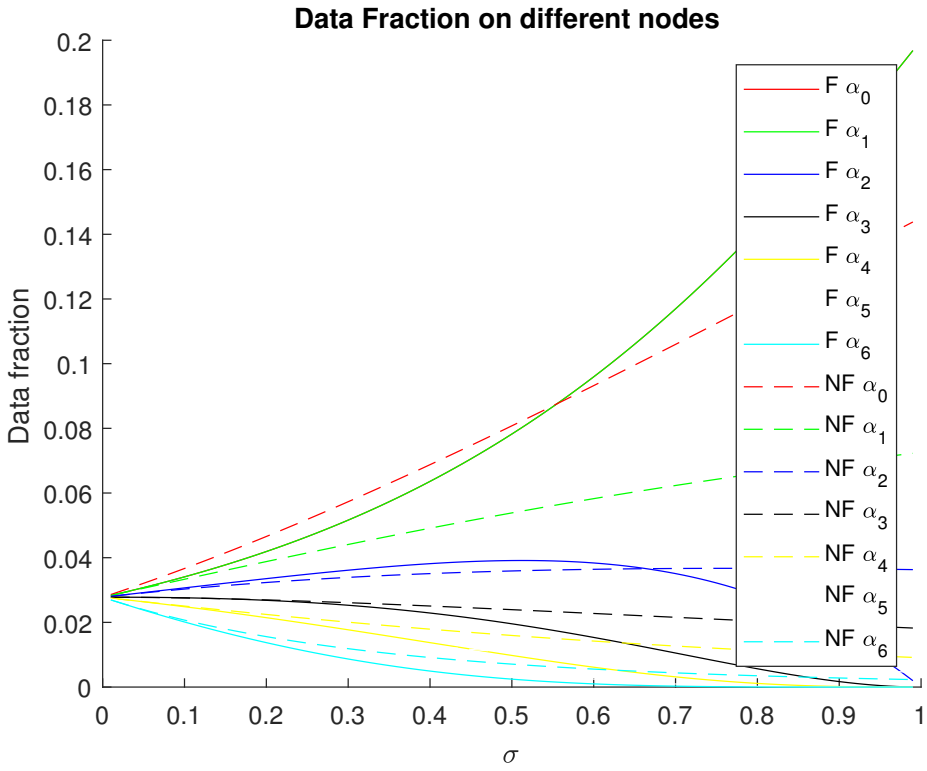

Figure 2.35: The comparing result between front-end processor with without front-end processor in 6*6 regular mesh injection on inner grid processor

2.5 Store and Froward Switching

In this chapter, we mainly discuss the virtual cut through [4] switching. In addition, the store and froward [10] schema is a mature data processing technique. In future works, we will consider this switching.

Chapter 3

Sensitivity Analysis

3.1 Front End Scenario

3.1.1 Data Injection on Corner

From Chapter 2, we know the speedup is

$$Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = -\det A$$

The simulation result of sensitivity analysis of $2 * n$ regular mesh Fig. 2.9 is as follows:

The figure illustrates that if $\sigma < 0.1$, the number of cores grow up, the speedup efficiency is likely linear increasing. Alternatively speaking, if $\sigma < 0.1$, the number of cores dominate the efficiency. If the $\sigma > 0.2$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation. This important investigation benefit the multi-source assignment problem. In addition, if the number of cores is bigger than 4, the bottom speedup effect is about 3 time.

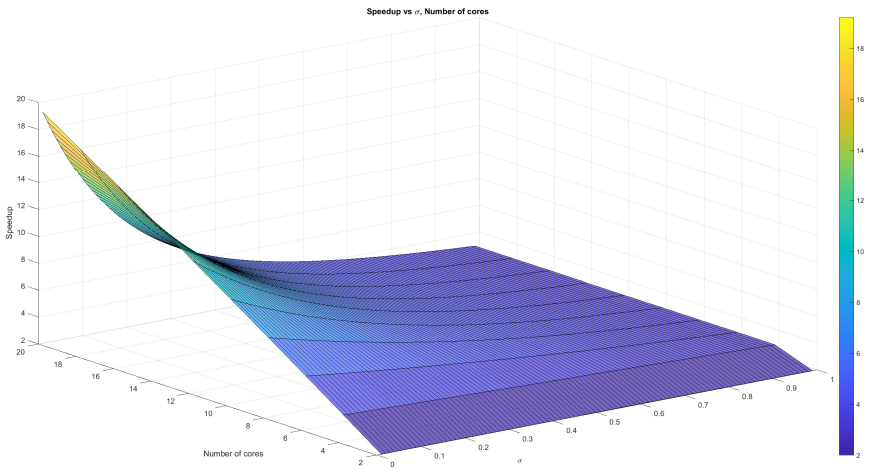

Figure 3.1: Sensitivity analysis result of 2*10 regular mesh result

3.1.2 Data Injection on Boundary Processor

Talking to Fig. 2.11, the data injection happens on boundary processor P_2.

Fig. 3.2 shows that $\sigma > 0.2$ the speedup simulation effect is impacted obviously. If $\sigma < 0.1$, the number of cores has linear impact on the speedup performance. If the number of cores is bigger than 5, the bottom effect with front end, the cluster at least get about 4 times speedup. It is because, a boundary processor has 3 $level_1$ processors.

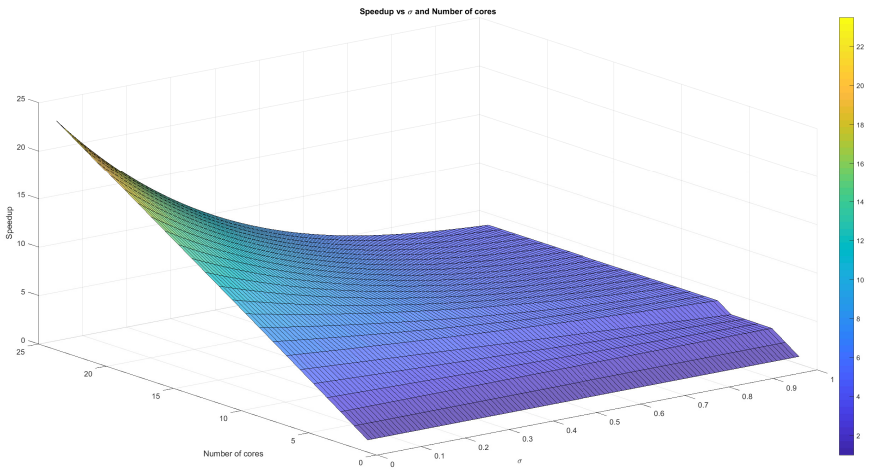

Figure 3.2: Sensitivity analysis result of 3*8 regular mesh and the injection position on boundary processor P_2

3.1.3 Data Injection on Inner Grid Processor

Scaling Fig. 2.16, to 25 processors, the simulation result says:

If the number of processor large than 5, the cluster equivalence computation ability is at least 5 time speedup. It is the inner grid has 4 $level_1$ neighbor processors.

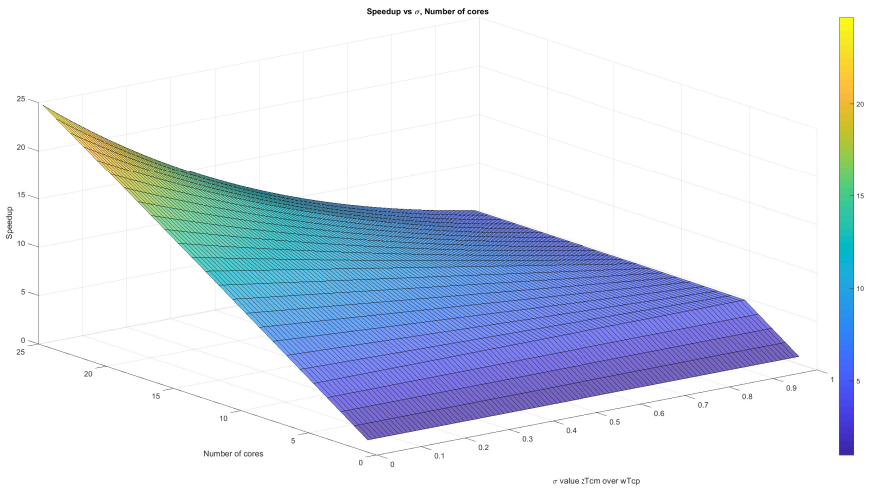

Figure 3.3: Sensitivity analysis result of data injection position on inner grid processor

3.1.4 Toroidal

The $level_i$ table shows Table 3.1:

So the simulation result illustrates in Fig. 3.4

D_i	Number
0	1
1	4
2	8
3	8
4	4

Table 3.1: The processor number of various D_i

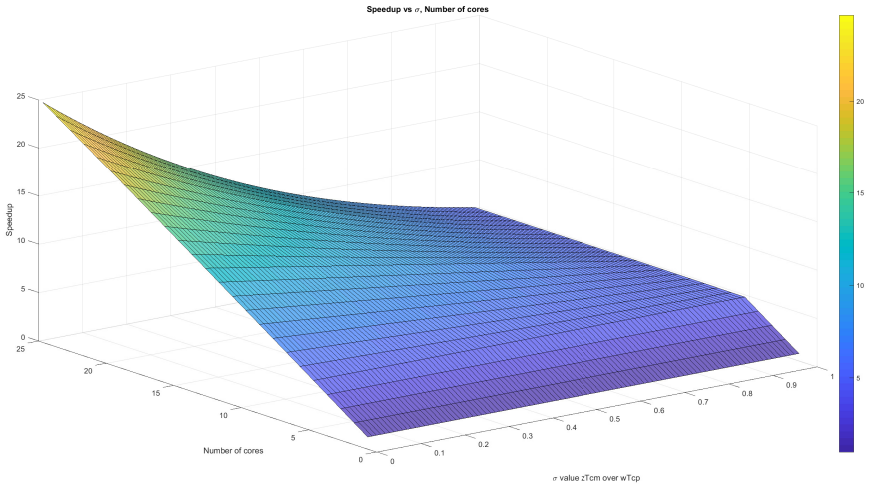

Figure 3.4: Sensitivity analysis result of data injection position on inner grid processor

3.2 Without Front End Scenario

3.2.1 Data Injection on Corner Processor

The simulation result of sensitivity analysis of $2 * n$ regular mesh Fig. 2.9 is as follows:

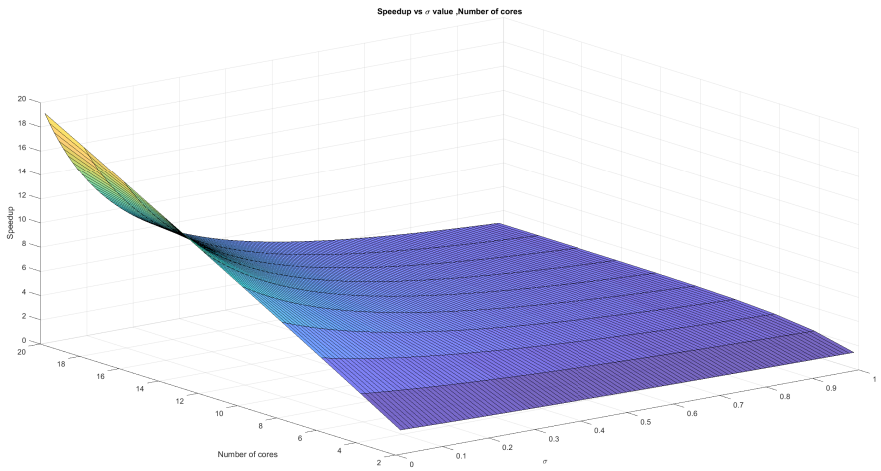

Figure 3.5: Sensitivity analysis result of 2*10 regular mesh result

The figure illustrates that if $\sigma < 0.1$, the number of cores grow up, the speedup efficiency is likely linear increasing. Alternatively speaking, if $\sigma < 0.1$, the number of cores dominate the efficiency. If the $\sigma > 0.2$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation. This important investigation benefit the multi-source assignment problem. In addition, if the number of cores is bigger than 4, the bottom speedup effect is about 3 time.

3.2.2 Data Injection on Boundary Processor

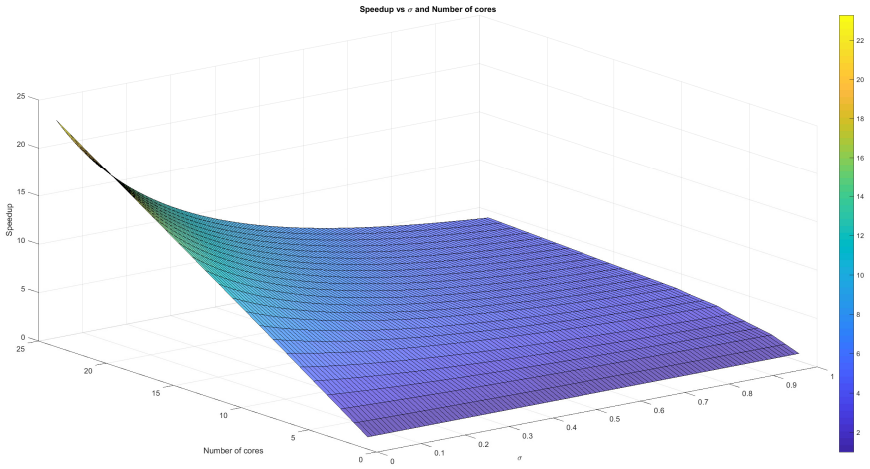

Figure 3.6: Sensitivity analysis result of 3*8 regular mesh result

Fig. 3.6 tells the simulation result for the data injection on the boundary.

3.2.3 Data Injection on Inner Grid Processor

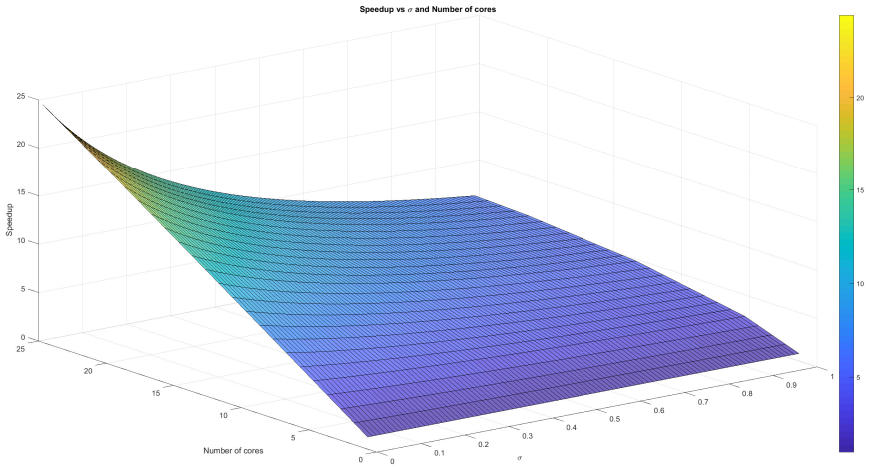

Figure 3.7: Sensitivity analysis result of data injection position on inner grid processor

Fig. 3.7 displays the simulation result for the data injection position P_{12}. The best speedup is 24, which happens on the $\sigma < 0.1$. If $\sigma < 0.1$, the speedup linear grows up.

3.3 Comparison Result With Corner Processor and Inner Grid Processor

For a $5 * 5$ regular mesh, the inner grid position is P_{12} and the corner data injection position is P_0. The comparing result is Fig. 3.8.

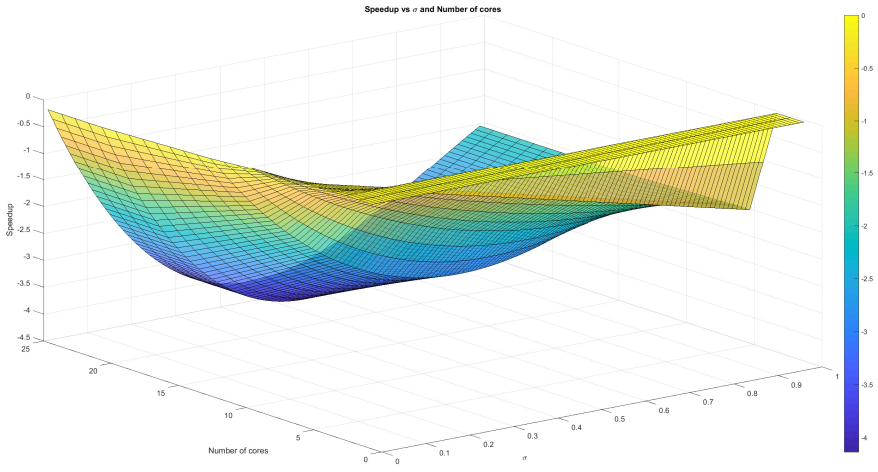

Figure 3.8: Speedup difference between corner injection and inner grid injection

Generally speaking, Fig. 3.8 says the inner grid position scenario has better performance than the corner injection option. If the grid node is 25 and $\sigma = 0.5$, the speedup difference is largest, which is 4.

3.4 Comparison Result With Front End and Without Front End

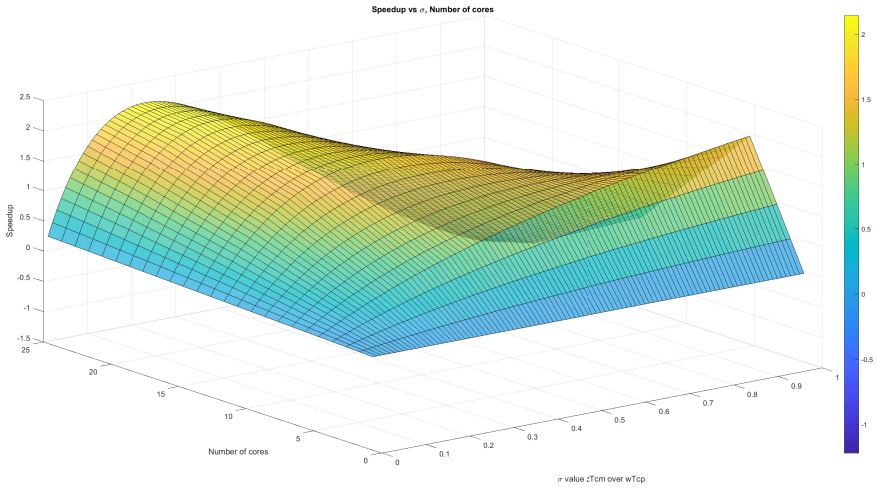

Figure 3.9: Speedup difference between front end and without front end in 5*5 regular mesh

Fig. 3.9 shows the speedup difference between the front end situation and without front end scenario.

3.5 Comparison Result Between Regular Mesh and Toroidal

Considering a regular mesh Fig. 2.12, the best position for inner grid data injection is P_{12}. Other data injection position, for example P_8, P_{13} they don't the same high speedup impact. Yet, for toroidal $5 * 5$ regular mesh, each position is equal, which means each position hit the best choice for regular mesh. Fig. 3.8 explores the comparison result between the toroidal and corner scenario difference.

Chapter 4

Multi-source Assignment Utilizing Voronoi Diagram

Chapter 2 and Chapter 3 address the equal computation capability problem. Further, we extend the single source assignment to multi-source assignment problem[11].

According to each processor, we concentrate on the processors' geographical location, data fraction assigned.

- The processors are front end.
- The processors execute the cut through switching.
- The network is homogeneous.

4.1 Even Data Fraction

We assume the data fraction is even. For example, the workload is unit 1 and there are n different data injection options. So each data injection is assigned $\frac{1}{n}$ workload.

4.1.1 Subgraph Data Injection

If the data injection consists of a subgraph of the whole graph, we use G_{data} to present it.

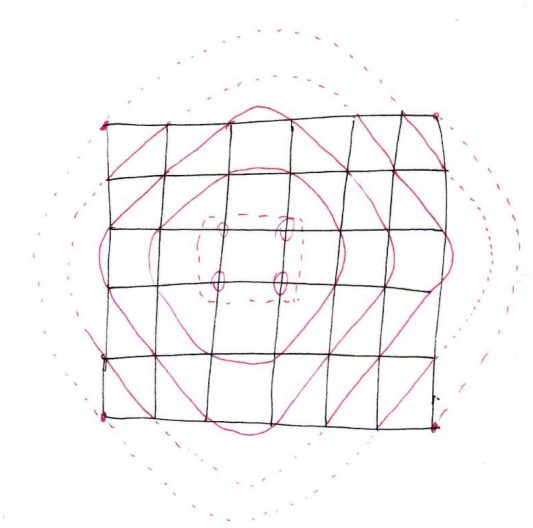

Figure 4.1: Data injection consists of a subgraph of G

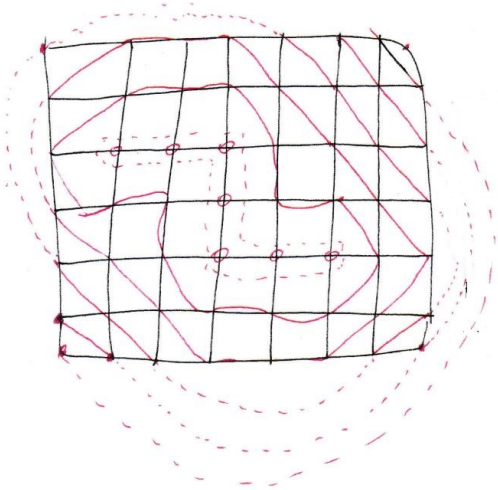

Figure 4.2: Data injection consists of a subgraph of G

Fig. 4.1 and Fig. 4.2 illustrate two situations that the data injections consist of subgraph G_{data} of the whole graph G.

The heuristic algorithm tackling this scenario is divided into two stages, $global_s$ and $local_s$ stages.

- $global_s$: Considering the cluster of data injection is a whole data source. Utilize the equal computation technique in Chapter 2.
- $local_s$: Re-distributing workload between the data injection processors.

For example, Fig. 4.1's ***flow matrix*** is :

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.1}$$

The simulation result illustrates as follows:

This simulation says the best performance happens on $\sigma \leq 0.05$, it hits about 40 times speedup. If $\sigma \approx 1$, the next work achieve almost 12 times performance.

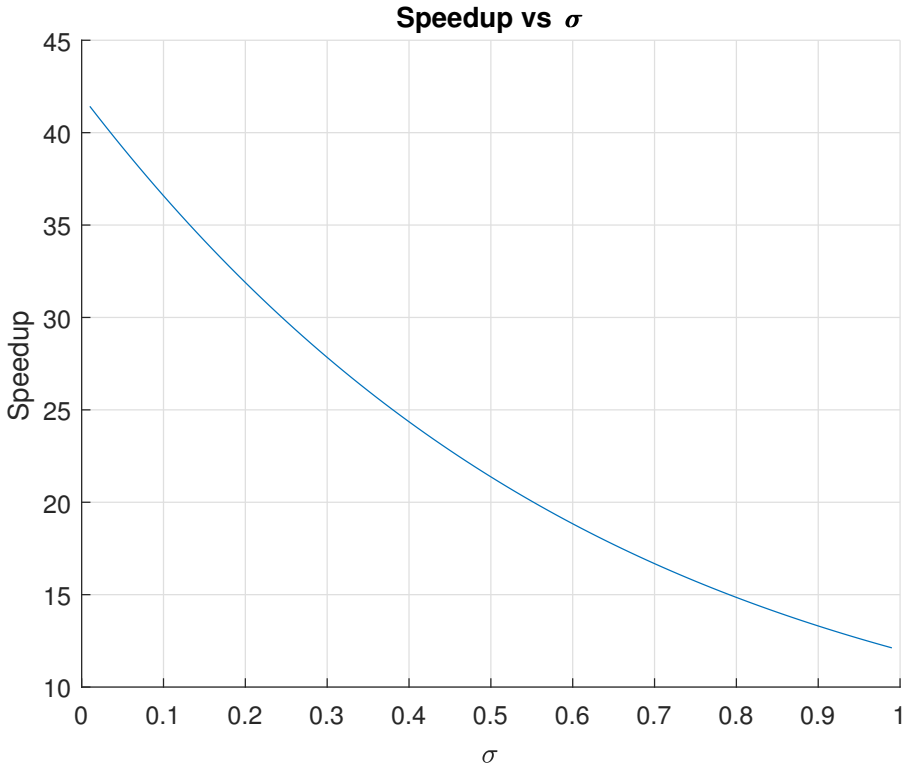

Figure 4.3: Speedup vs σ

4.1.2 Individual Data Injection Positions

Considering another situation, the data injection doesn't consist of a subgraph of G. Jia's paper [12] inspire me to utilize Voronoi tool to tackle this problem. In the dividable load application, for example, big file transmission, GPU scientific computation or Hadoop job, the total job time depends on the last fraction finish time. So in this section, our goal is to use less processors to obtain the same or similar finish time. The critical idea comes from the investigation from Chapter 2, which means, if $\sigma > 0.3$, the speedup performance drops quickly.

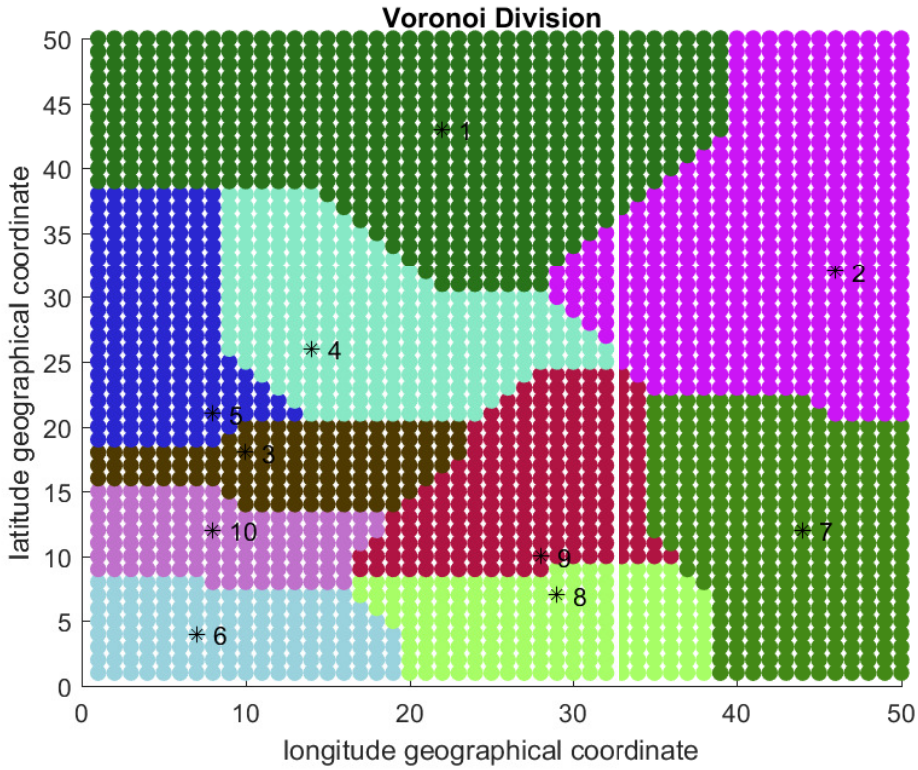

Figure 4.4: 10 Voronoi Cells

Fig. 4.4 shows 10 voronoi cells division and Fig. 4.5 figures out the speedup of each cells.

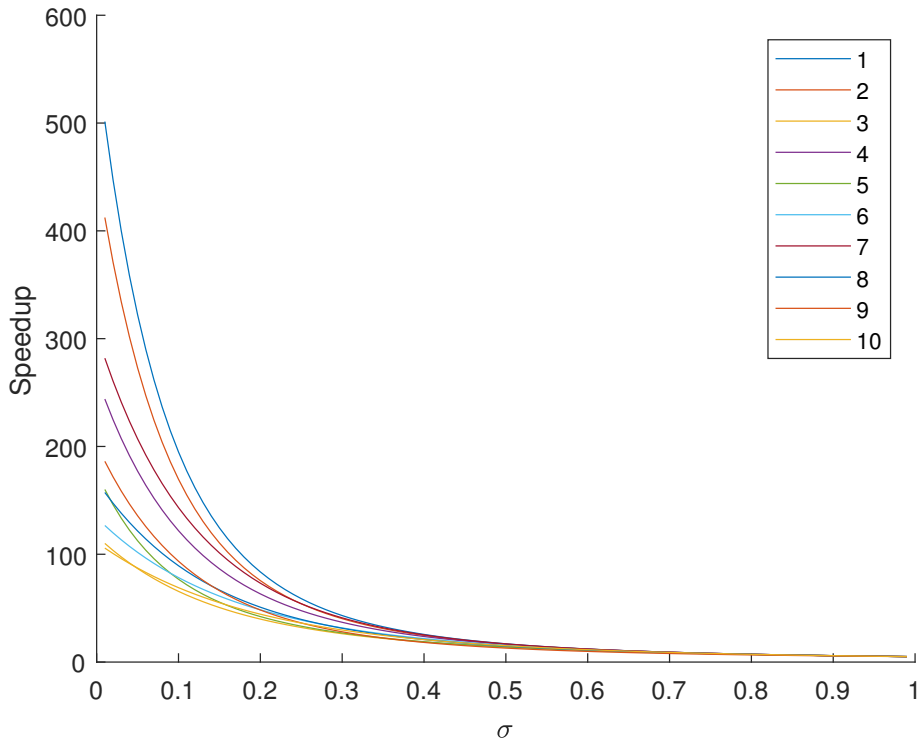

Figure 4.5: 10 Voronoi Cells's speedup curves

The heuristic algorithm is named as ***Reduced Voronoi Division Algorithm***:

Fig. 4.6 and Fig. 4.7 show the algorithm's result and speedup curve. That is, we obtain the same running time yet occupy 30% processors.

Fig. 4.5 shows the ratio $\frac{maxspeedup}{minspeedup} = \frac{500}{100} = 5$ and Fig. 4.7 shows the ratio is $\frac{maxspeedup}{minspeedup} = \frac{270}{100} = 2.7$. It displays the computation power is more balanced, further, the cluster finish the task with the same time by less processors.

After 1000 round random sampling the data injection position, we obtain the average saved processors result.

Algorithm 1 Reduced Voronoi Division Algorithm(RVDA)

Calculate n Voronoi cells with Mahattan distance

Calculate n radius R_i in n Vornoi cells

Set $R_{min} = \min R_i$

while $0 < i \leq n$ **do**

Calculate the Reduced Vornoi cells by setting the $R_i = R_{min}$ in each cell.

Calculate Voronoi cell's flow matrix A_i.

$i = i + 1$

end while

Display each Reduced Voronoi Cells

Illustrate each Reduced Voronoi Cells' Speedup curves

From Fig. 4.8, it shows the average percentage of saved processor is about 35%.

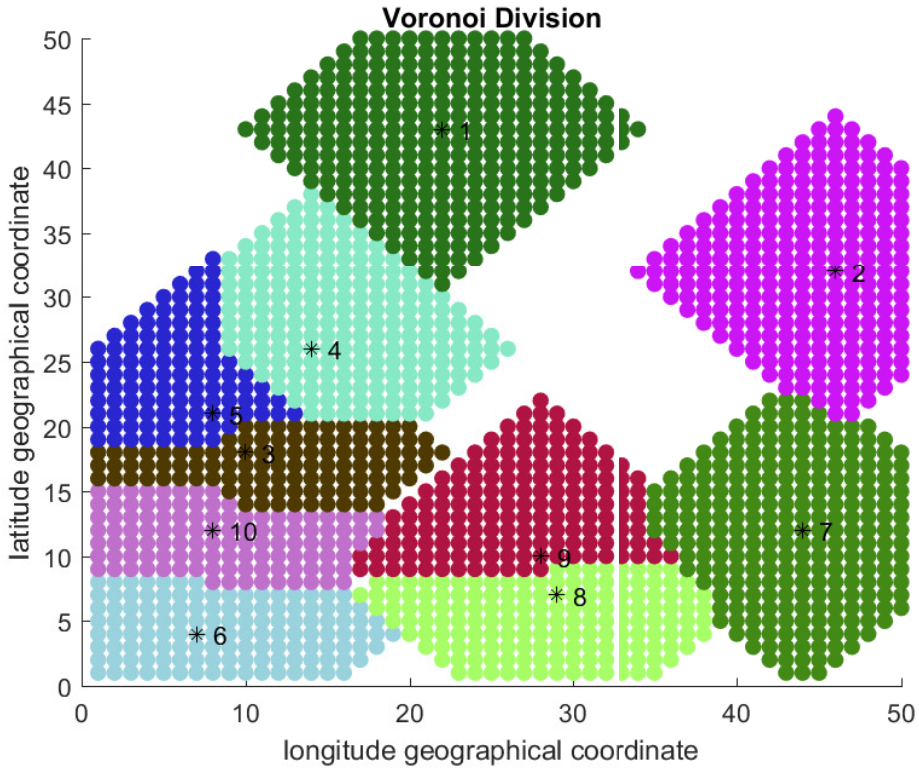

Figure 4.6: 10 Voronoi Cells

In real case, it usually combine the subgraph situation and sparse scenario. We will discuss it in future work.

4.1.3 Toroidal

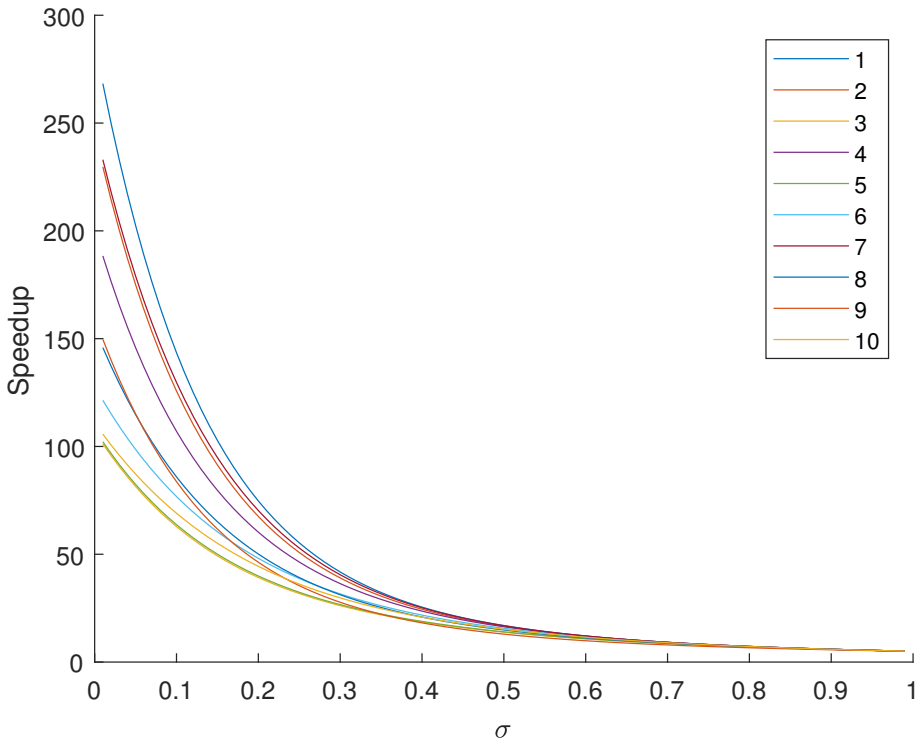

Figure 4.7: 10 Voronoi Cells's speedup curves

Fig. 4.9 [13] provides a torus Voronoi method, which extends the original domain to 8 copy and calculate the Voronoi Diagram as planner algorithm [14]. Then the corner part is the torus Voronoi diagram.

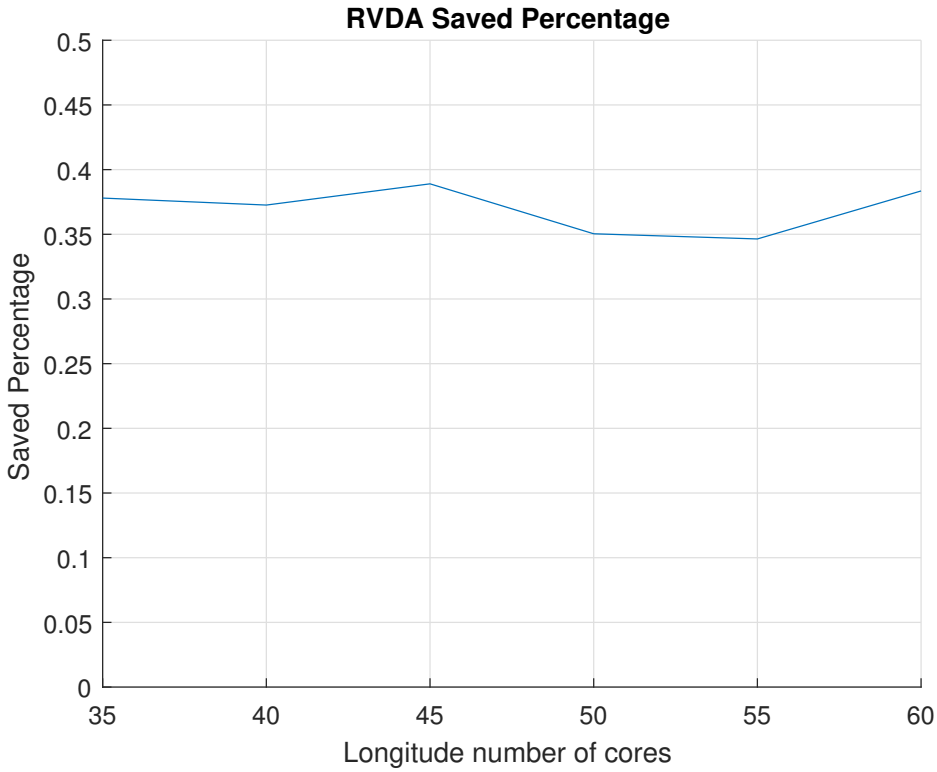

Figure 4.8: Reduced Voronoi Division Algorithm average processors' percentage

In the reduced model, reduced toroidal Voronoi diagram save 27% processors hit the same processing capacity. The ratio of original method is about $\frac{490}{98} = 5$, after the reduced action, the ratio is $\frac{290}{98} \approx 2.96$. That is the reduced heuristic algorithm obtaining more balanced computation capacity distribution.

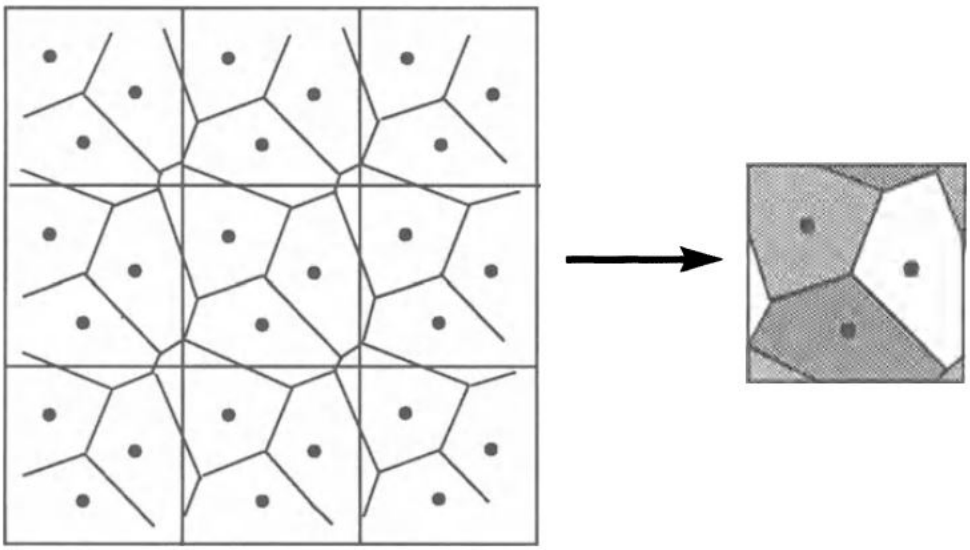

Figure 4.9: How to calculate torus Voronoi Diagram

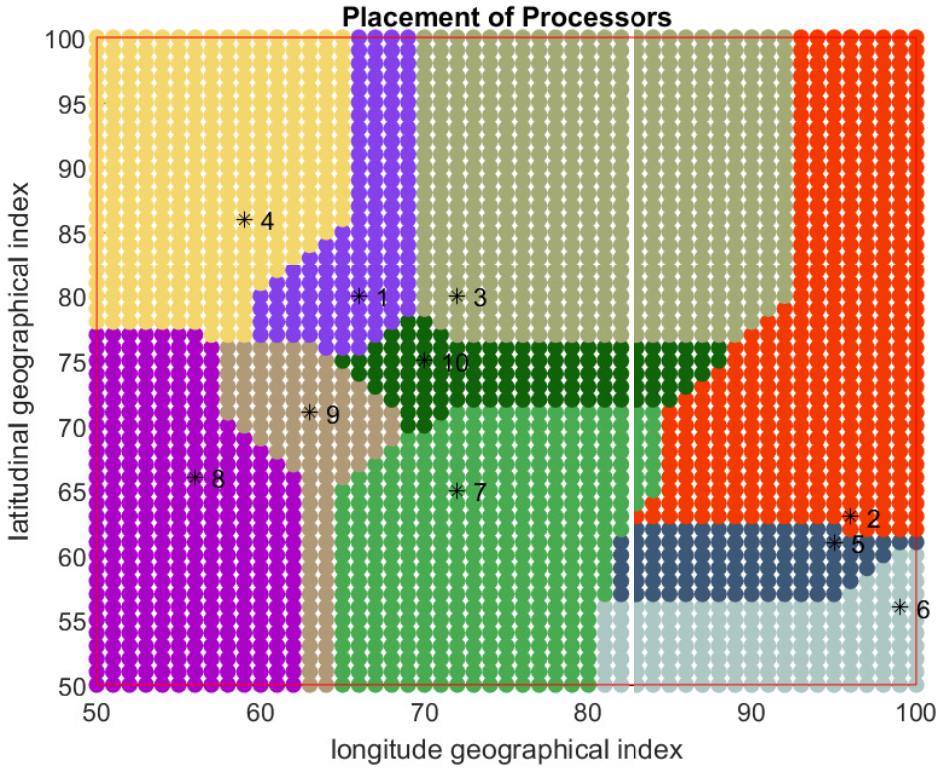

Figure 4.10: Initial Voronoi Digram

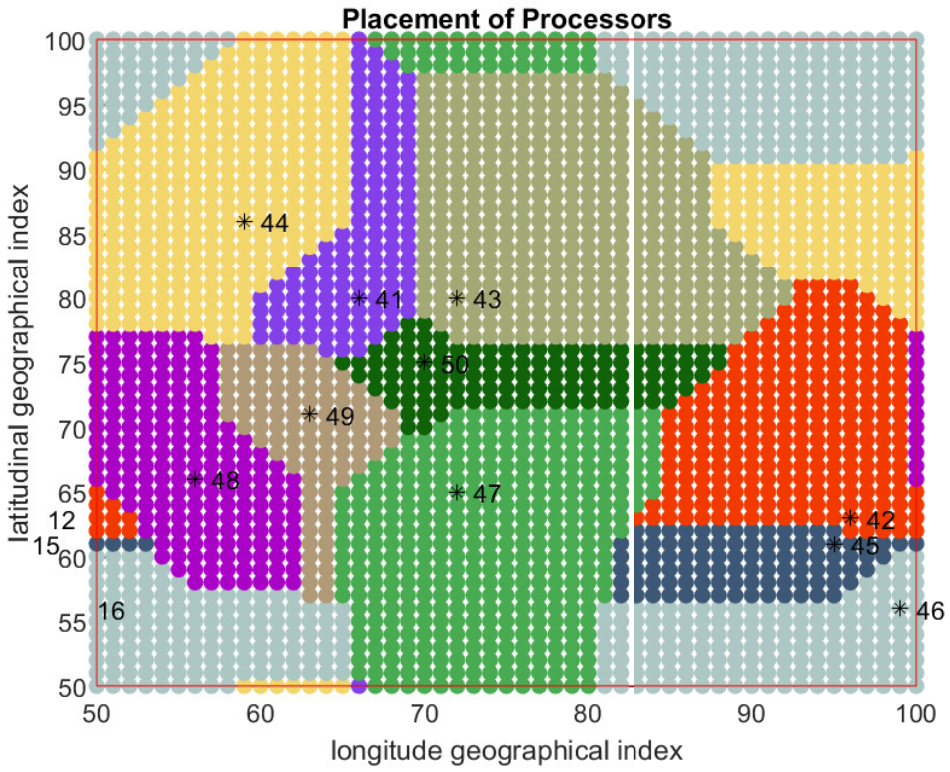

Figure 4.11: Torus Voronoi Diagram

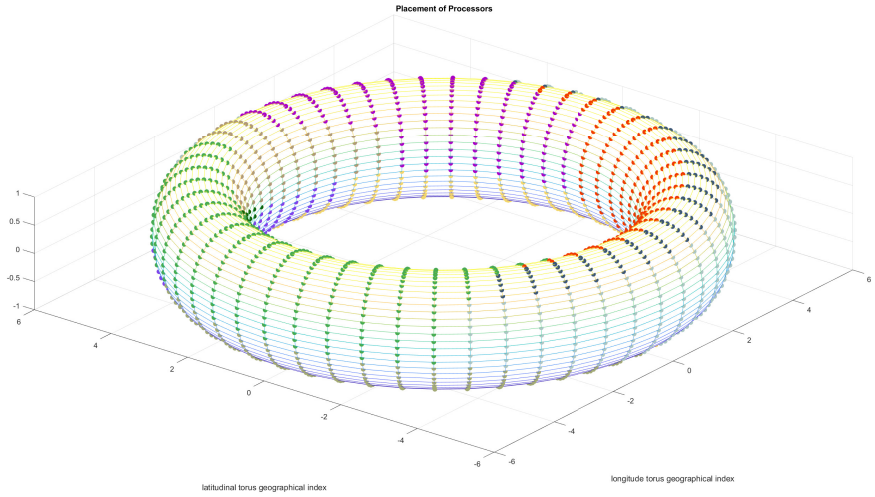

Figure 4.12: Voronoi Diagram Casting to the torus model

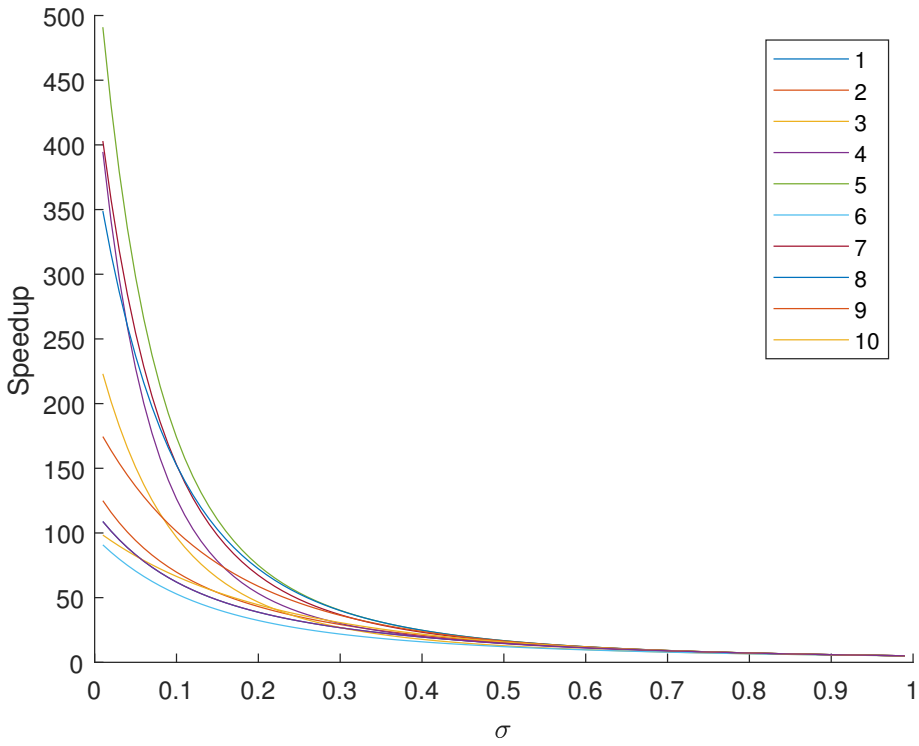

Figure 4.13: Voronoi Diagram Casting to the torus model

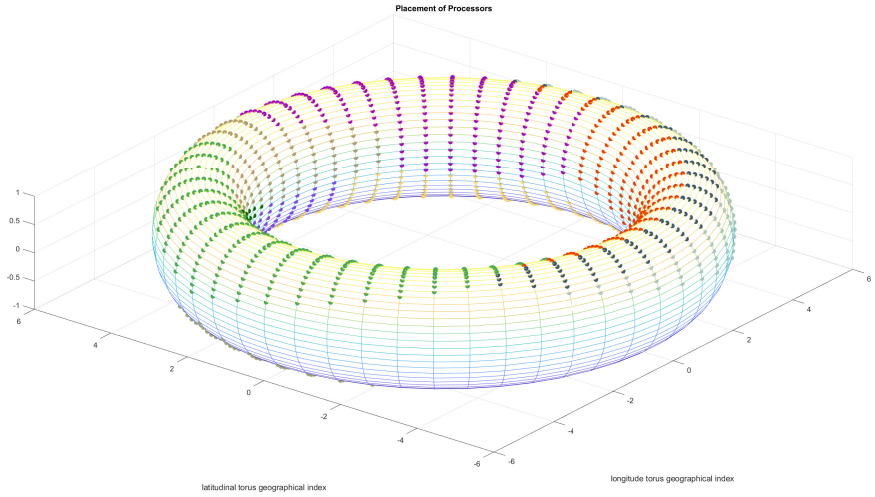

Figure 4.14: Torus Reduced Voronoi Diagram Casting to the Torus Model

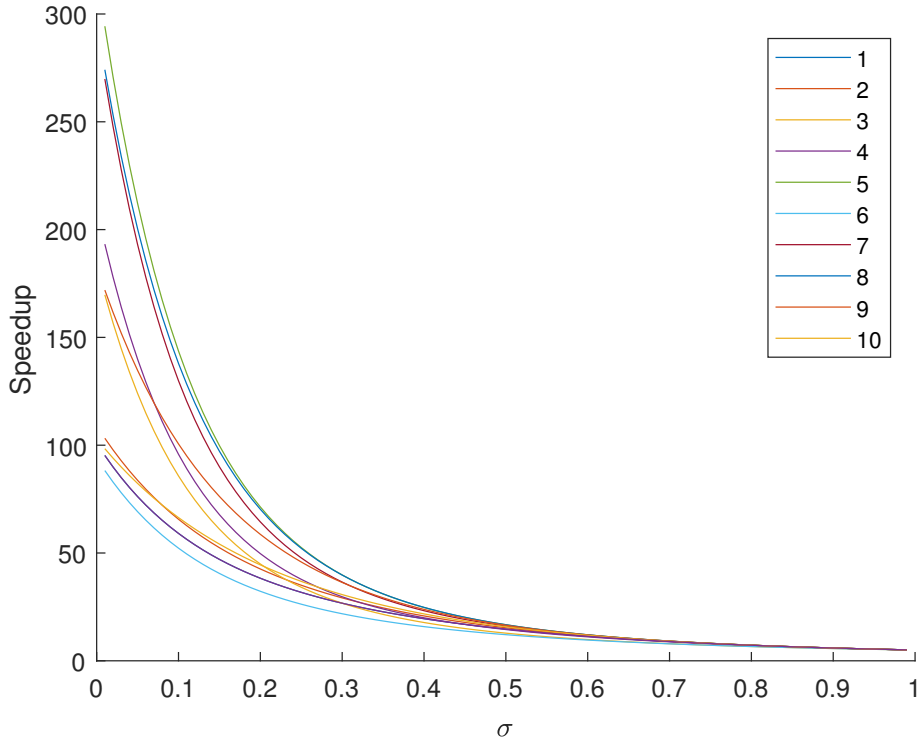

Figure 4.15: Torus Reduced Voronoi Diagram

Chapter 5

Multi-source Re-assignment Using Optimal Mass Transportation

5.1 Optimal Mass Transport Theory

Optimal Mass Transport Problem (OMT) is introduced by Monge in 18th century[15], which is presented as a problem minimizing the inter-domain transportation cost while preserving measure quantities.

Kantorovich[16] utilized the linear programming approach to prove the existence and uniqueness of the relaxed optimal mass transport(OMT) problem and the space complexity is $O(n^2)$. In order to increase the time efficiency, Professor Gu introduces a practical discrete optimal mass transport(OMT) map based on the Brenier's approach[17] and this algorithm framework's space complexity is $O(n)$[18]. Professor Jie's group [19] implement an application in mobile networks.

We first implement it utilizing CGAL [20] library, yet we find this method has strong library dependency and in $3D$ version there are unstable because of numerical errors cases. The algorithm framework critical solution is n dimension convex hull, yet this method is hard to implement in real simulation.

So we implement it through Monte Carlo Method. The benefit is that this method's accuracy and executing time based on the sampling point and GPU power. As **Moore's law**, this method is more easily to extend to $3D$ and high dimension. Also, during the implementation of Monte Carlo version, it doesn't depend on the geometry structure, so it is friendly to implement.

Nowdays, the optimal mass transport theory and the wasserstein distance plays a critical role in GAN model[21].

The discrete version is tackled [22]. In our case, we consider about the continuous to discrete measure.

- We consider the continuous domain to discrete measure.
- The cost function is $cost(x, f(x)) = |x - f(x)|^2$.
- The object function is minimizing the cost $\sum_{n=0}^{m*n-1} cost(x, f(x))$

5.1.1 Optimal Mass Transport Map Algorithm

Assume Ω is a convex planar domain with measure density μ, $P = \{p_1, ..., p_k\}$ is a point set with measure $\nu = \{\nu_1, ..., \nu_k\}$, such that $\int_\Omega \mu(x)dx = \sum_{i=1}^{k} \nu_i$. In practice, the energy can be optimized using Newton's method, with the help of the computation of the energy gradient$\nabla E(\mathbf{h}) = (w_1(\mathbf{h}) - \nu_1), ..., w_k(\mathbf{h}) - \nu_k)^T$. The Hessian of $E(\mathbf{h})$ is given as following:

$$\frac{\partial^2 E(\mathbf{h})}{\partial h_i \partial h_j} = \begin{cases} \frac{\int_{e_{ij}} \mu(x)dx}{|y_j - y_i|} & W_i(\mathbf{h}) \cap W_j(\mathbf{h}) \cap \Omega \neq \emptyset \\ 0 & otherwise \end{cases}$$

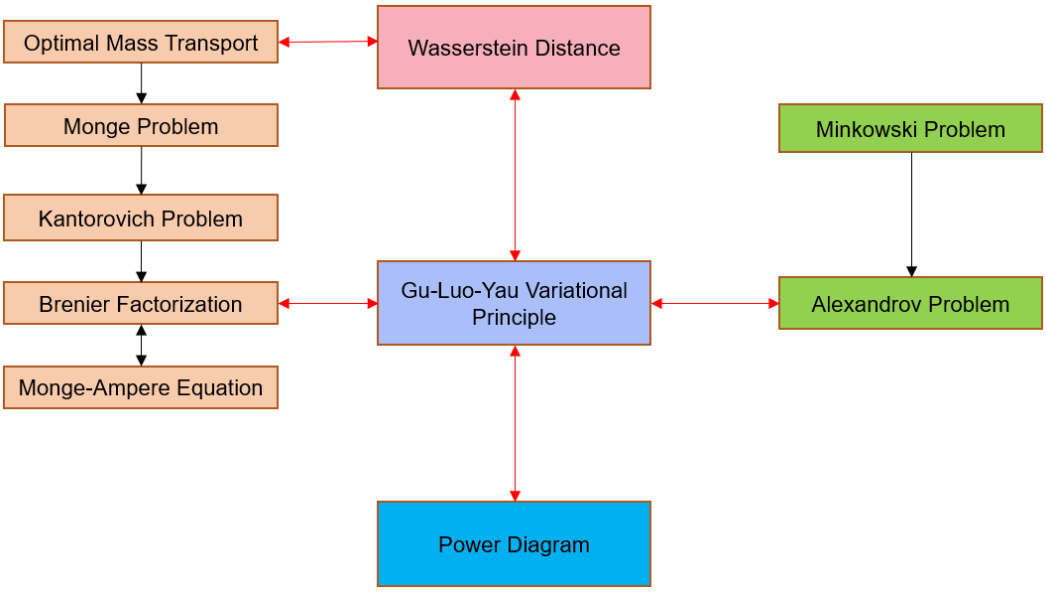

Figure 5.1: OMT theory relationship

5.2 Monte Carlo Method Implementation of Optimal Mass Transport

In our demo, we consider the convex domain is square and this theory is available extended to n Dimension space. Professor Allen [23] [24] propose a multi-resolution Fluid mechanics GPU optimal mass transport algorithm framework.

5.2.1 Divisible Load OMT Experiment

We implement this algorithm in GPU cuda 9.0 and the experiment result is as follows:

Algorithm 2 Optimal Mass Transport Map

Require: Input:

1. A convex planar domain with measure (Ω, μ);

2. A planar point set with measure $(P, \nu), \nu_i > 0, \int_\Omega u(x)dx = \sum_{i=1}^{k} \nu_i$

Output: The unique discrete OMT-Map $f : (\Omega, \mu) \to (P, \nu)$

Scale and translate P, such that $P \subset \Omega$
$\mathbf{h} \leftarrow (0, 0, ..., 0)$
Compute the power diagram $D(\mathbf{h})$
Compute the dual power Delaunay triangulation $T(\mathbf{h})$
Compute the cell areas $\mathbf{w}(\mathbf{h}) = (w_1(\mathbf{h}), ..., w_k(\mathbf{h}))$
while $||\nabla E|| < \epsilon$ **do**
    Compute ∇E
    Compute the Hessian matrix
    $\lambda \leftarrow 1$
    $\mathbf{h} \leftarrow \mathbf{h} - \lambda H^{-1} \nabla E(\mathbf{h})$
    Compute $D(\mathbf{h})$, $T(\mathbf{h})$, and $\mathbf{w}(\mathbf{h})$
    while $\exists w_i(\mathbf{h}) == 0$ **do**
        $\mathbf{h} \leftarrow \mathbf{h} + \lambda H^{-1} \nabla E(\mathbf{h})$
        $\lambda \leftarrow \frac{1}{2}\lambda$
        $\mathbf{h} \leftarrow \mathbf{h} - \lambda H^{-1} \nabla E(\mathbf{h})$
        Compute $D(\mathbf{h})$, $T(\mathbf{h})$, and $\mathbf{w}(\mathbf{h})$
    end while
end while
Output the result mapping $f : \Omega \to P, W_i(\mathbf{h}) \to p_i,\ i = 1, 2, ..., k.$

OMT Division

The initial cell situation:

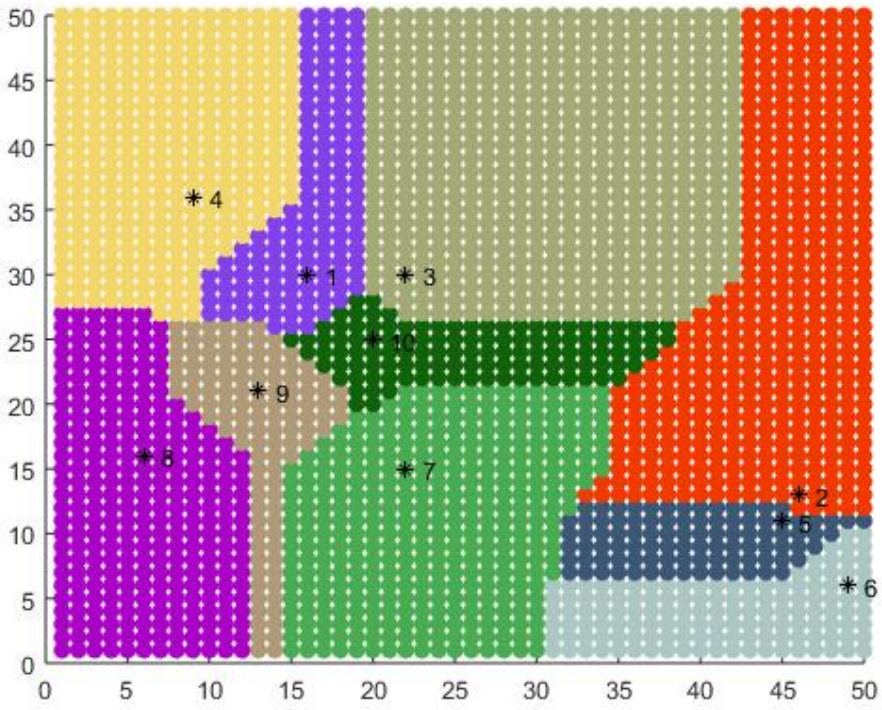

Figure 5.2: Initial Voronoi Cell

The minimal cost of OMT cell division.

The relocation of base

OMT Division Speedup Curve

The initial speedup curve:

The minimal cost of OMT cell division.

The relocation of base

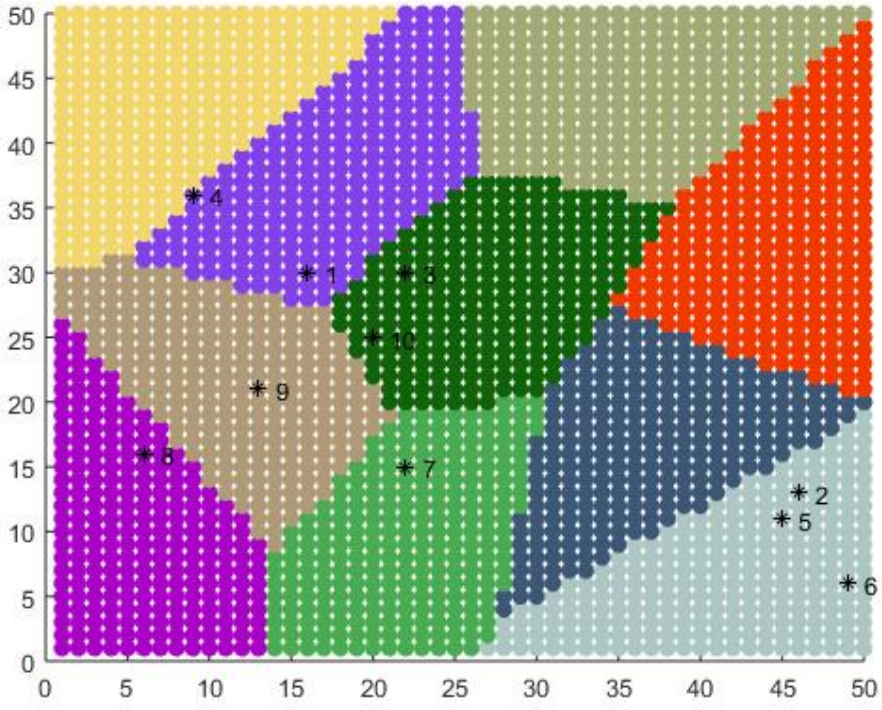

Figure 5.3: OMT Division Cell

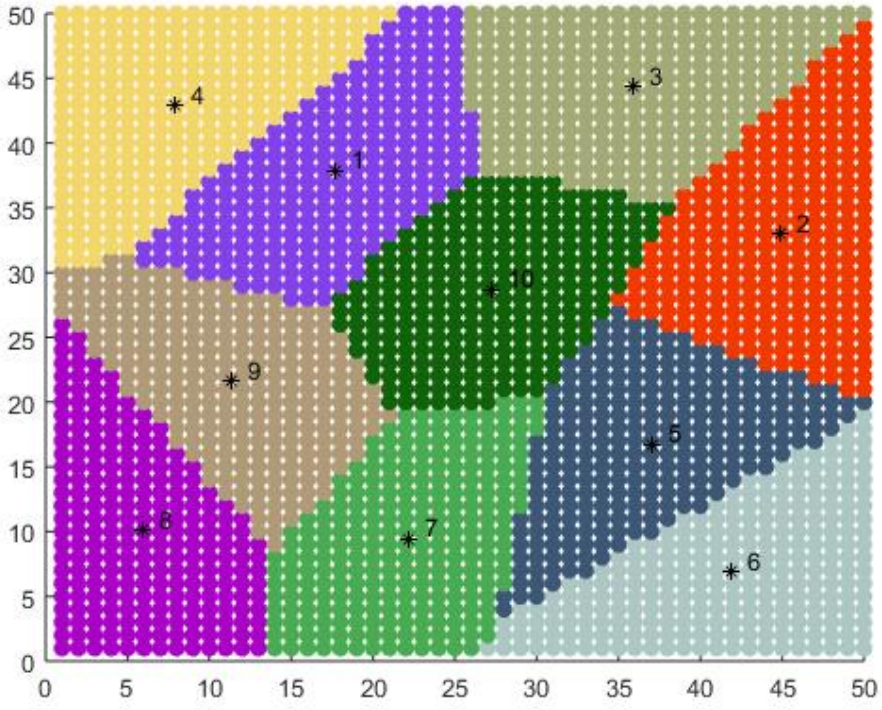

Figure 5.4: OMT Division Cell base relocation

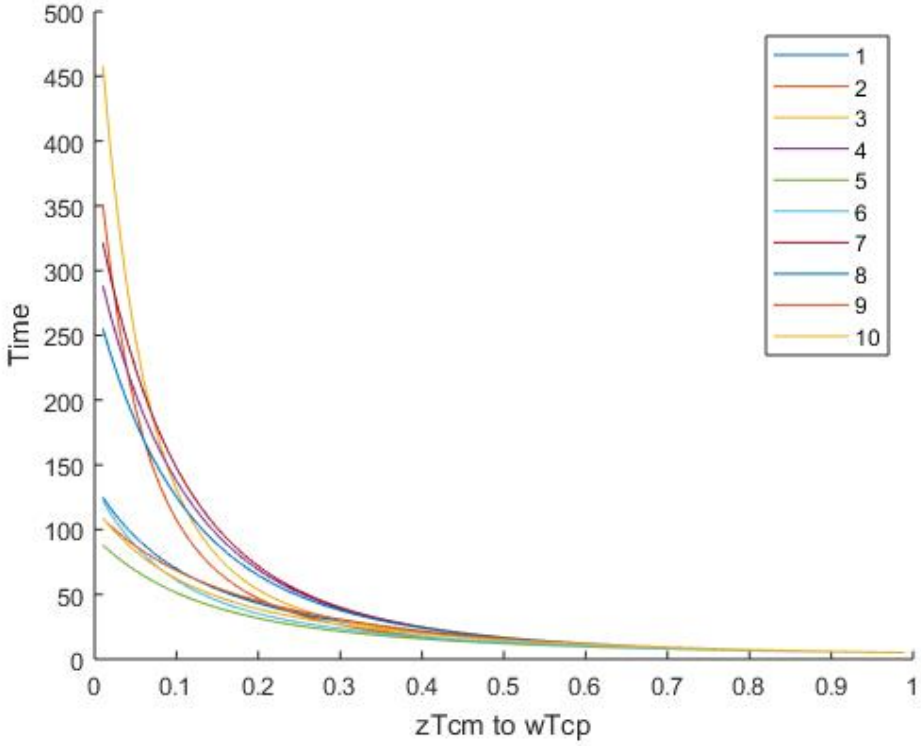

Figure 5.5: Initial Voronoi Cell Speedup Curve

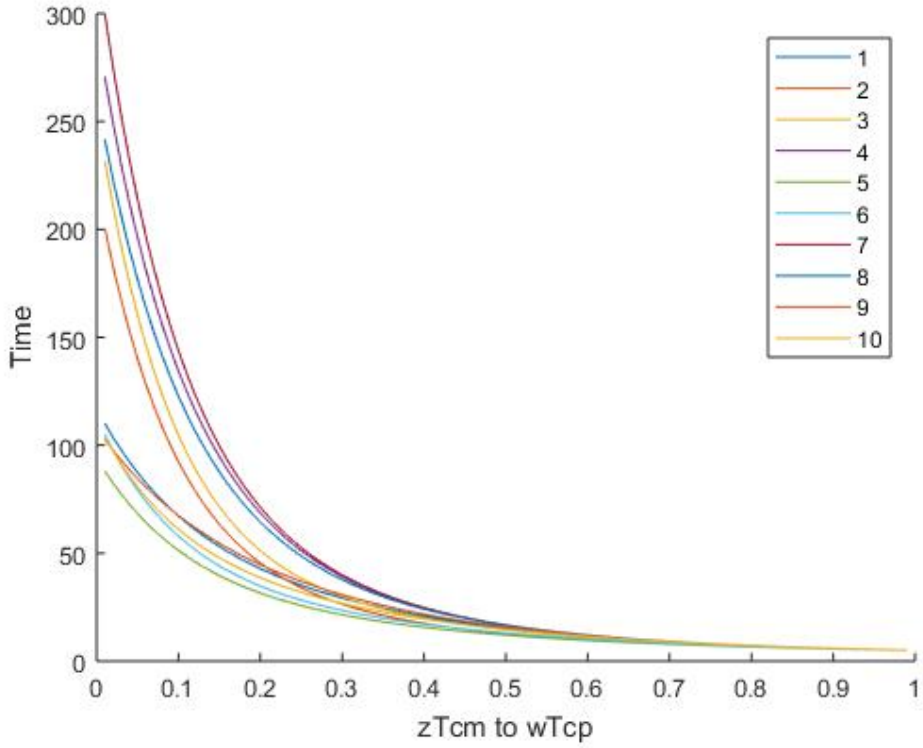

Figure 5.6: OMT Division Cell Speedup Curve

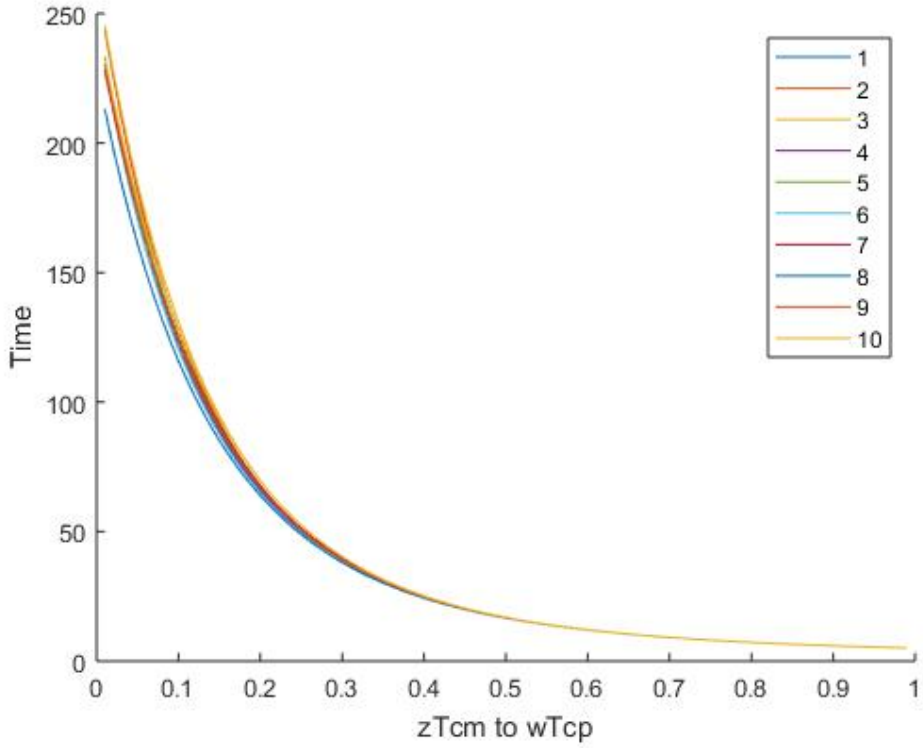

Figure 5.7: OMT Division Cell base relocation Speedup Curve

Chapter 6

Conclusions and Future Works

- Slice OMT
- Different Data Fraction Voronoi
- Different Data Fraction OMT

Bibliography

[1] M. Moges, D. Yu, and T. G. Robertazzi, "Grid scheduling divisible loads from two sources," *Computers & Mathematics with Applications*, vol. 58, no. 6, pp. 1081–1092, 2009.

[2] T. G. Robertazzi, "Processor equivalence for daisy chain load sharing processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 29, no. 4, pp. 1216–1221, 1993.

[3] X. Liu, H. Zhao, and X. Li, "1scheduling divisible workloads from multiple sources in linear daisy chain networks."

[4] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks (1976)*, vol. 3, no. 4, pp. 267–286, 1979.

[5] C. F. Gamboa and T. Robertazzi, "Simple performance bounds for multicore and parallel channel systems," *Parallel Processing Letters*, vol. 21, no. 04, pp. 439–460, 2011.

[6] V. Bharadwaj, *Scheduling divisible loads in parallel and distributed systems*, vol. 8. John Wiley & Sons, 1996.

[7] J. T. Hung and T. G. Robertazzi, "Switching in sequential tree networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 968–982, 2004.

[8] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, "Divisible load theory: A new paradigm for load scheduling in distributed systems," *Cluster Computing*, vol. 6, no. 1, pp. 7–17, 2003.

[9] T. G. Robertazzi, "Toroidal networks," *IEEE Communications Magazine*, vol. 26, no. 6, pp. 45–50, 1988.

[10] G. R. Kanthraj, C. R. Srinivas, *et al.*, "Store and forward teledermatology," *Indian Journal of Dermatology, Venereology, and Leprology*, vol. 73, no. 1, p. 5, 2007.

[11] J. Jia, B. Veeravalli, and J. Weissman, "Scheduling multisource divisible loads on arbitrary networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 21, no. 4, pp. 520–531, 2010.

[12] S. Fortune, "Voronoi diagrams and delaunay triangulations," in *Computing in Euclidean geometry*, pp. 225–265, World Scientific, 1995.

[13] C. I. Grima and A. Márquez, *Computational Geometry on Surfaces: Performing Computational Geometry on the Cylinder, the Sphere, the Torus, and the Cone*. Springer Science & Business Media, 2013.

[14] S. Fortune, "A sweepline algorithm for voronoi diagrams," *Algorithmica*, vol. 2, no. 1-4, p. 153, 1987.

[15] G. Monge, *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale, 1781.

[16] L. V. Kantorovich, "On the transfer of masses," in *Dokl. Akad. Nauk. SSSR*, vol. 37, pp. 227–229, 1942.

[17] Y. Brenier, "Polar factorization and monotone rearrangement of vector-valued functions," *Communications on pure and applied mathematics*, vol. 44, no. 4, pp. 375–417, 1991.

[18] X. Gu, F. Luo, J. Sun, and S.-T. Yau, "Variational principles for minkowski type problems, discrete optimal transport, and discrete monge-ampere equations," *arXiv preprint arXiv:1302.5472*, 2013.

[19] C.-C. Ni, Z. Su, J. Gao, and X. D. Gu, "Capacitated kinetic clustering in mobile networks by optimal transportation theory," in *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pp. 1–9, IEEE, 2016.

[20] M. Hemmer, "Algebraic foundations," in *CGAL User and Reference Manual*, CGAL Editorial Board, 4.12 ed., 2018.

[21] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[22] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 2, no. 1-2, pp. 83–97, 1955.

[23] G. D. Pryor, J. Melonakos, A. R. Tannenbaum, *et al.*, "Multi-resolution 3d nonrigid registration via optimal mass transport on the gpu," Georgia Institute of Technology, 2007.

[24] T. ur Rehman, E. Haber, G. Pryor, J. Melonakos, and A. Tannenbaum, "3d nonrigid registration via optimal mass transport on the gpu," *Medical image analysis*, vol. 13, no. 6, pp. 931–940, 2009.