Data Distribution Equivalence for Data Intensive Interconnection Networks

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

Xiaolin Li - Member
Professor, Department of Applied Mathematics and Statistics

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Data Distribution Equivalence for Data Intensive Interconnection Networks

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

List of Tables

Acknowledgements

Thank you to Professor Thomas G. Robertazzi who gave me strong academic and emotional support during hard times.
Thank you to Professor Esther M. Arkin, Professor Joseph S.B. Mitchell and Professor Xiaolin Li, who give me lots of paper modification suggestions and joined my thesis defense committee. Thank you to Professor Jim Jiao, who gives me lots of suggestions about Ph.D. life time management.
Thank you to Christine Rota, who helped me to process the internship application, visa issue, and so on.
Thanks to Professor David Gu, who taught me a great deal of optimal mass transport theory and computational conformal geometry.
Thanks to my co-authors, especially Ph.D. candidate Liang Mi, I learn a great deal of medical image knowledge from him.
Thanks to Ocoos, eBay, BNL and Uber, I found four perfect internship mentors. They gave me intensive industry project training and a good work attitude.

Chapter 1

Introduction

Four chapters are organized as follows :

- A problem introduction, notation and definitions are shown in Chapter II.
- Chapter III considers the mesh network. We solve the processor equivalence problem with virtual cut-through scenario and store and forward situation. Sensitivity analysis is another topic. In considering the multi-source workload assignment problem, we propose a reduced Voronoi diagram assignment.
- Chapter IV investigates the hypercube network situation. The front-end and without front-end assumption are discussed. Sensitivity analysis is examined and the multi-source assignment in hypercube environment is also an interesting problem.
- Chapter V discuss the conclusion and future work.

Chapter 2

Problem Description

2.1 Problem Background

In large-scale data intensive problems with geographically distributed resources, load is generated from multiple sources [2] for a class of problems. It is assumed that the problem representation can be arbitrarily divided amongst the processors. Thus the problem representation is said to be "divisible". 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.
In this chapter, the closed-form processor equivalence [3] [4] problem in the grid networks of mesh network and toroidal rectangle network is discussed. Also, the multi-source workload assignment is also taken into account.

In this thesis, we investigate two problems. One is the processor equivalence problem. The other one is scheduling divisible workloads from multiple sources in mesh networks Fig. 2.1, toroidal rectangle networks Fig. 2.2.

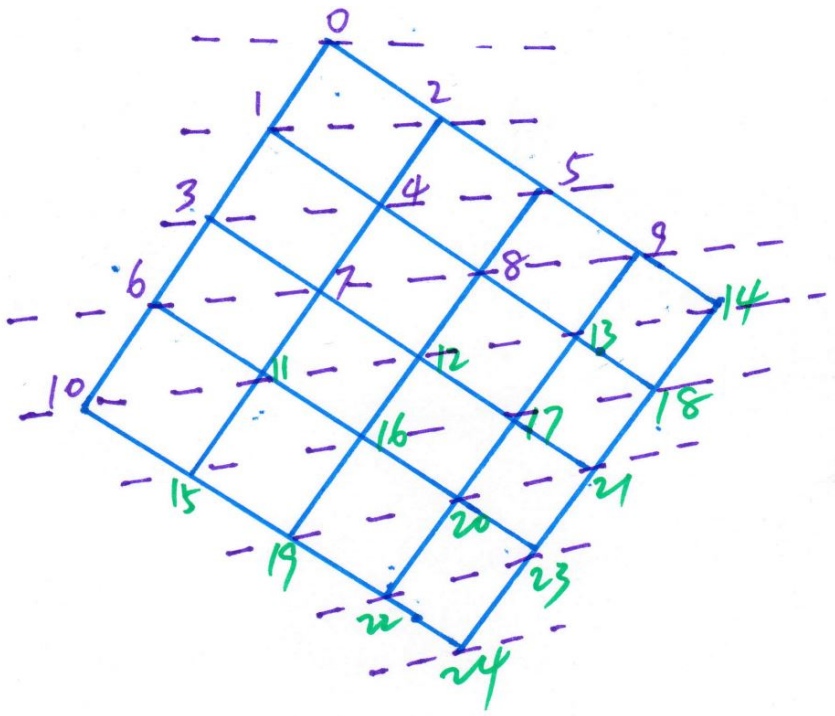

Figure 2.1: A m*n mesh network(m = 5, n = 5)

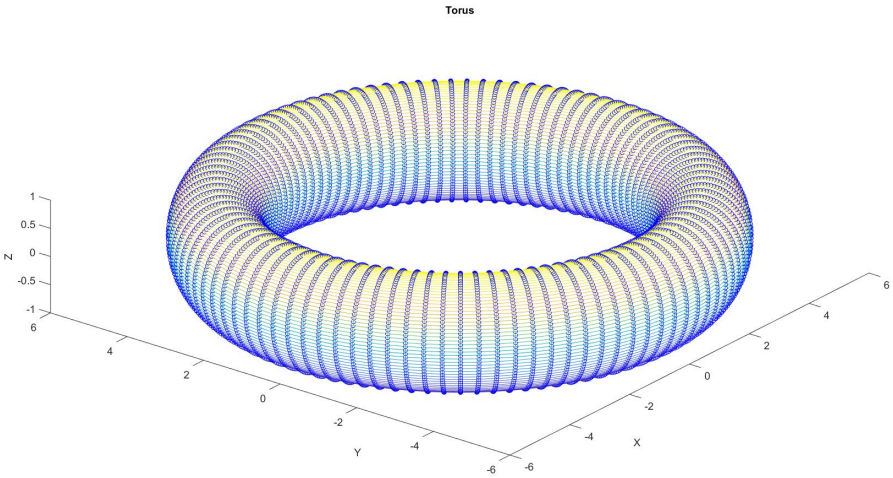

Figure 2.2: A toroidal rectangle network with grid unit cores

2.2 Definitions and Assumption

Definition 2.2.1. *Equal Computation*
Equal computation is a technique, which considers combining a cluster of processors as one whole processor to process a unit 1 *workload.*

The following assumptions are used throughout the paper:

- Virtual cut-through [5] switching and store and forward switching is used to transmit the assigned workload between processors.
  - Under virtual cut-through switching, a node can relay the beginning bit of a message (packet) before the entire message is received.
  - Under store and forward switching, a message must be completely received by a node before it can be relayed to the next node along their transmission path.
- For simplicity, return communication is not considered.
- The communication delays are taken into consideration.
- The time costs of computation and communication are assumed to be linear function of the data size.
- 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 5, .

The optimization objective functions are as follows :

- Equal computation : the problem's objective function is how to partition and schedule the workloads amongst the processors to obtain the minimum makespan (finish time).

- Multi-source assignment : how all processors can finish processing a unit 1 workload at the same time utilizing fewer processors.

To achieve the minimum time 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 processors to idle ones [6] [7].

2.2.1 Notions

The following notations and definitions are utilized:

- P_i: The ith processor. $0 \leq i \leq m * n - 1$.

- L_i: The ith work load. $1 \leq i \leq k$.

- D_i: The minimum number of hops from the processor P_i to the data load injection site L.

- $level_i$: The processors have i minimum Manhattan distance to the data injection node.

- 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 cluster of processors.

- z_i: The inverse link speed on the ith link.
- T_{cp}: Computing intensity constant. The entire load is processed in time $\omega_i T_{cp}$ seconds on the ith processor.
- T_{cm}: Communication intensity constant. The entire load is transmitted in time $z_i T_{cm}$ seconds over the ith link.
- $T_{f,n}$: The finish time of the whole processor 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] [8].
- In multi-source situation, $\sum_{i=1}^{k} L_i = 1$
- $\sum_{i=0}^{m*n-1} \alpha_i = 1$
- $Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

Chapter 3

Mesh Network

3.1 Virtual Cut-through Switching Scenario

In the virtual cut-through environment, a node can begin relaying the first part of a message (packet) along a transmission path as soon as it starts to arrive at the node , that is, it doesn't have to wait to receive the entire message before it can begin forwarding the message.

First we consider about the $2*2$ mesh network, $2*n$ mesh network. After, we analyze a more general case $m*n$ mesh network 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 are also discussed.

3.1.1 Data Injection on The Corner Processor

2*2 Mesh Network

The L is assigned on the corner processor P_0 Fig. 3.1. The whole task is tackled by four processors P_0, P_1, P_2, P_3 together.

The processor P_0, P_1 and P_2 start to process its respective fraction at the same time. The processor P_3 starts to work when the α_1 and α_2 are completed

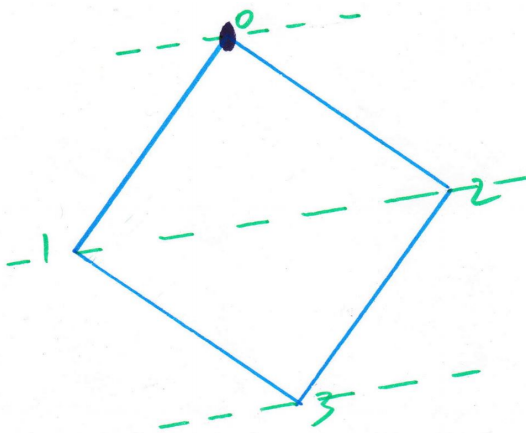

Figure 3.1: The 2*2 mesh network and the root processor is P_0

transmission. That is, the link $0-1$ and $0-2$ are occupied transmitting load to processor 1 and 2, respectively and only transmission to 3 when that is finished.

According to the divisible load theory [9], we obtain the timing diagram Fig. 3.2.

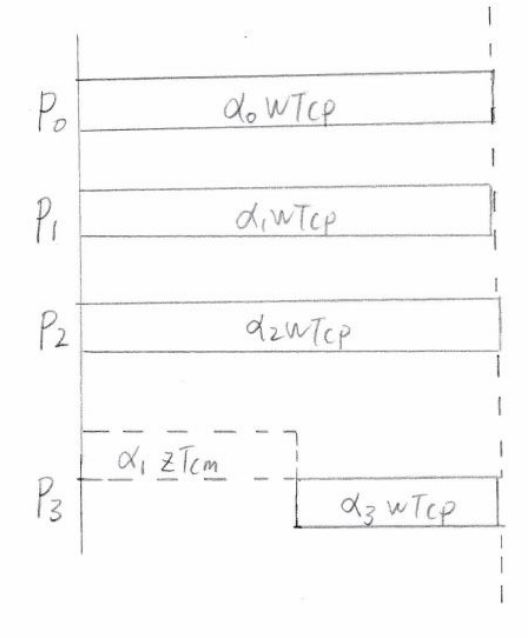

Figure 3.2: The timing diagram for 2*2 mesh network and the root processor is P_0

Here in the Gantt-like timing diagram communication appears above each axis and computations appears below the each axis. Let's assume that all processors stop computing at the same time in order to minimize the makespan [7].

Based on the timing diagram, we obtain a group of linear equations to find the fraction workload assigned to each processor α_i :

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (3.1) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (3.2) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (3.3) \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.4) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (3.5) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.6) \\ 0 < \sigma < 1 & (3.7) \\ 0 < \alpha_0 \leq 1 & (3.8) \\ 0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 & (3.9) \end{cases}$$

The group of equations are represented by the matrix form:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.10)$$

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

Finally, the explicit solution is:

$$\begin{cases} \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (3.11) \\ \alpha_0 = \dfrac{1}{4-\sigma} & (3.12) \\ \alpha_1 = \dfrac{1}{4-\sigma} & (3.13) \\ \alpha_3 = \dfrac{1-\sigma}{4-\sigma} & (3.14) \end{cases}$$

The simulation result is illustrated:

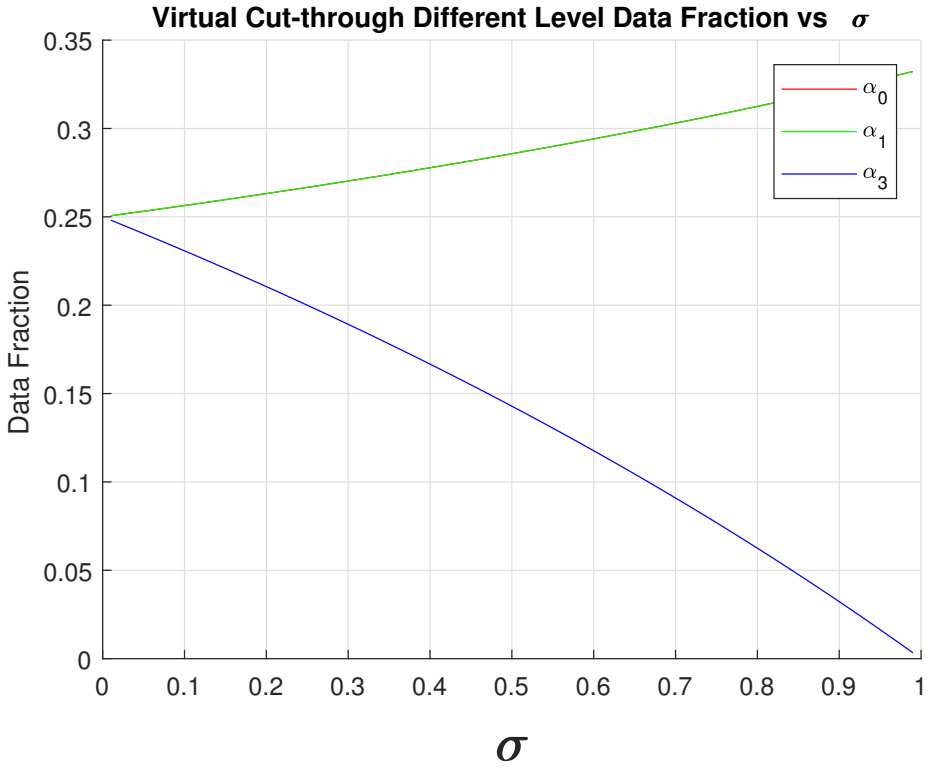

Figure 3.3: 2*2 mesh network. α_0, α_1, α_2, α_3 value curve

In Fig. 3.3, the three processors P_0, P_1, P_2 three processors have the same data fraction workload, so the curve of α_0 and α_1 coincide.

Also from symmetry $\alpha_1 = \alpha_2$, the figure says that as σ grows, the value α_3 drops. In other words, as the communication capacity decreases, there is less data workload assigned to P_3. Further, it means it will be economical to keep the load local on P_0 P_1 P_2 and not distribute it, to other processors.

The speedup is:

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

2*3 Mesh Network

In Fig. 3.4 mesh network, L originates on processor P_0. There are 6 processors in this network.

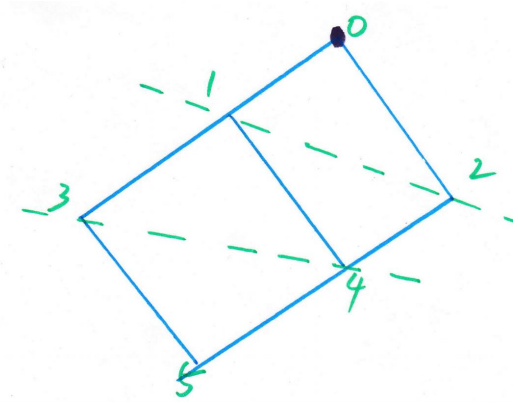

Figure 3.4: The 2*3 mesh network and the data injection happens on corner processor P_0

Here P_0, P_1 and P_2 start processing at the same time. Processor P_3 and P_4 start to work when they receive the data from processor P_1, P_2, after the $0-1$ and $0-2$ link become available to transfer load to P_3 and P_4
The last processor P_5 doesn't start 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 [9], we obtain the timing diagram Fig. 3.5.

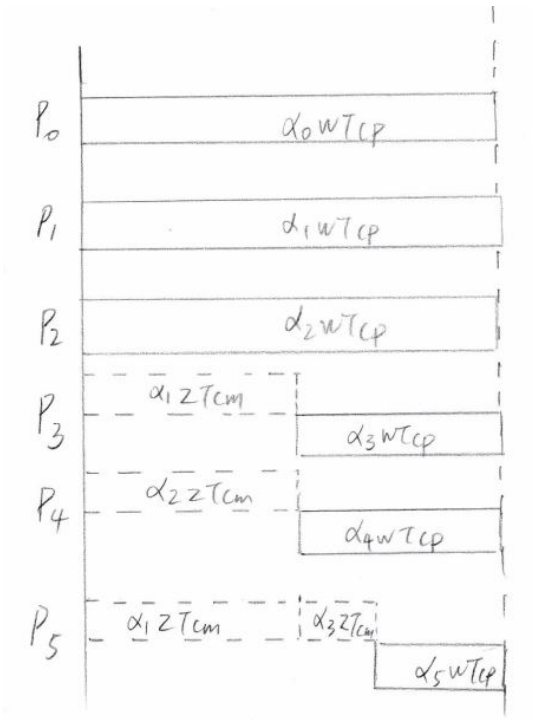

Figure 3.5: The timing diagram for a 2*3 mesh network and the data injection happens on processor P_0

The timing diagram equations as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.15) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (3.16) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (3.17) \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.18) \\
\alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.19) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.20) \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (3.21) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.22) \\
0 < \sigma < 1 & (3.23) \\
0 < \alpha_0 \leq 1 & (3.24) \\
0 \leq \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 < 1 & (3.25)
\end{cases}
$$

The flow matrix closed-form formula is:

$$
\begin{bmatrix}
1 & 2 & 2 & 1 \\
1 & -1 & 0 & 0 \\
0 & \sigma - 1 & 1 & 0 \\
0 & \sigma - 1 & \sigma & 1
\end{bmatrix}
\times
\begin{bmatrix}
\alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5
\end{bmatrix}
=
\begin{bmatrix}
1 \\ 0 \\ 0 \\ 0
\end{bmatrix}
\tag{3.26}
$$

The explicit solution is:

$$\begin{cases} \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (3.27) \\ \alpha_0 = \dfrac{1}{\sigma^2 - 4 \times \sigma + 6} & (3.28) \\ \alpha_1 = \dfrac{1}{\sigma^2 - 4 \times \sigma + 6} & (3.29) \\ \alpha_3 = \dfrac{1 - \sigma}{\sigma^2 - 4 \times \sigma + 6} & (3.30) \\ \alpha_5 = \dfrac{\sigma^2 - 2 \times \sigma + 1}{\sigma^2 - 4 \times \sigma + 6} & (3.31) \end{cases}$$

The α calculation result are shown in Fig. 3.6. P_0, P_1 have the same fraction so the curve of α_0 and α_1 coincide. Because of symmetry, $\alpha_1 = \alpha_2$ and $\alpha_3 = \alpha_4$, if the communication time is zero, each of the six processors get like $\frac{1}{6}$ of the load which shows in the figure. If the communication delay equals computation time $\frac{1}{3}$ of the load goes to each of P_0 P_1 and P_2.

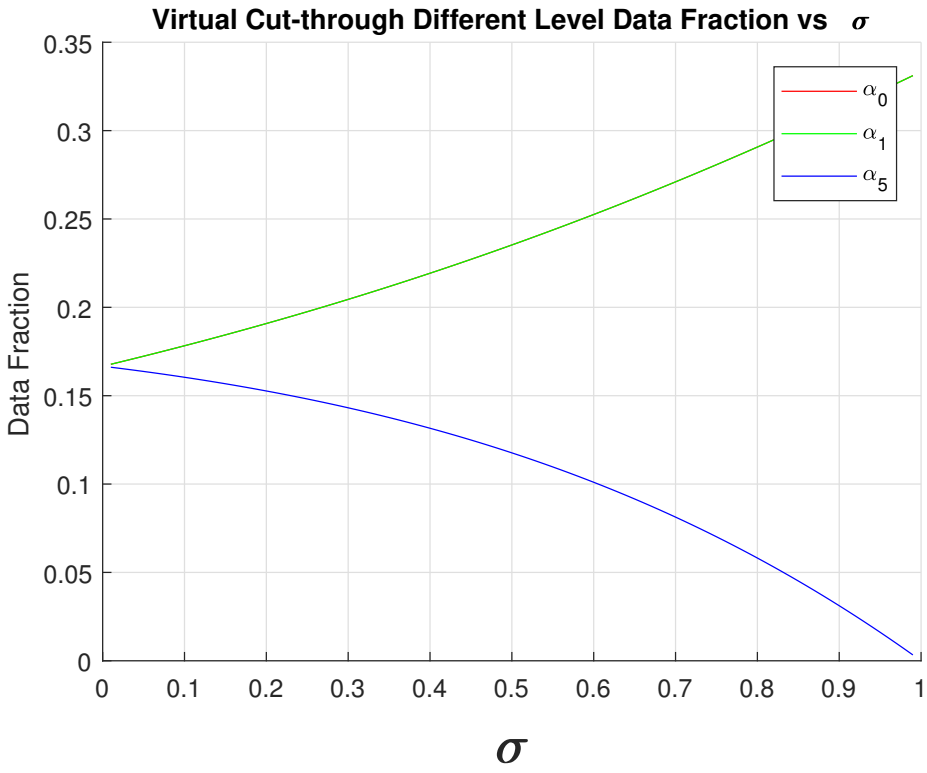

Figure 3.6: 2*3 mesh network. α_0, α_1, α_3, α_5 value curve.

2*N Mesh Network

The $2 * n$ Fig. 3.7 homogeneous mesh network process load L and L originates P_0.

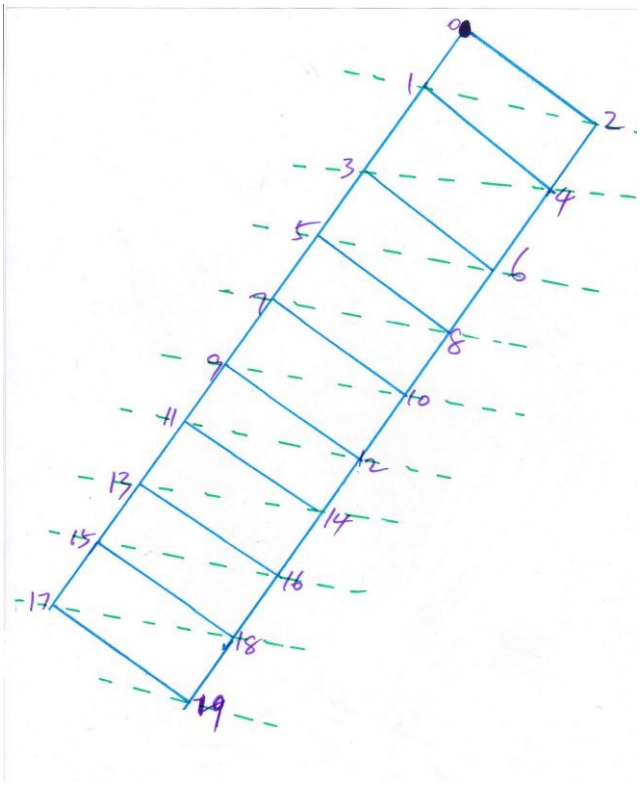

Figure 3.7: 2*n (n = 10) mesh network and the workload happens on P_0

Load a distribution from P_0 to P_1 and P_2 via virtual cut-through. After P_1 and P_2 finish receiving load from link $0-1$ and $0-2$, they will be used to forward load to P_3 and P_4 and so on.

Similarly to the analysis of Fig. 3.2 and Fig. 3.5, the timing diagram for Fig. 3.7 is shown in Fig. 3.8

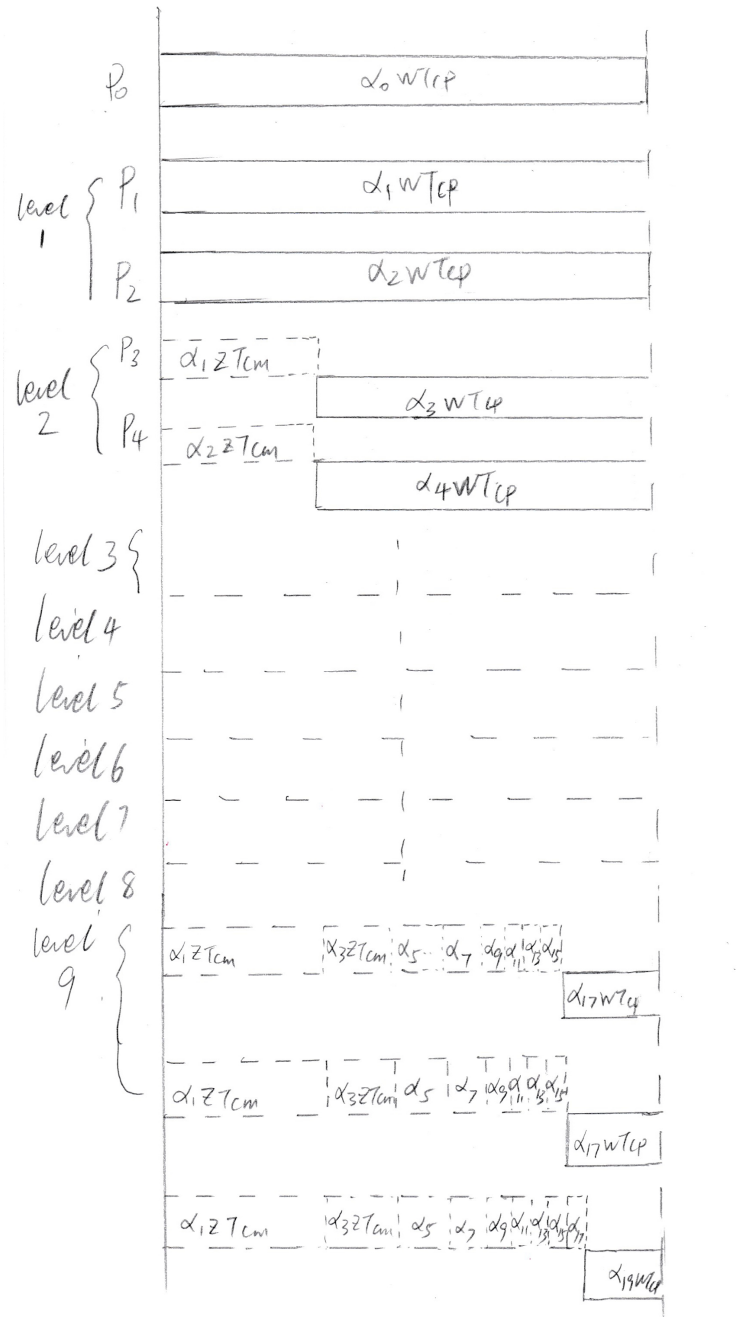

Figure 3.8: The timing diagram for 2*10 mesh network and the data injection happens on P_0

The equations are presented as:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.32) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (3.33) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (3.34) \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.35) \\
\alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.36) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.37) \\
\vdots & (3.38) \\
(\alpha_1 + \alpha_3 + \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (3.39) \\
\alpha_0 + \cdots + \alpha_{2\times n-1} = 1 & (3.40) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.41) \\
0 < \sigma < 1 & (3.42) \\
0 < \alpha_0 \leq 1 & (3.43) \\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (3.44)
\end{cases}
$$

The flow matrix closed-form is shown:

$$
\begin{bmatrix}
1 & 2 & 2 & \cdots & 2 & 2 & 1 \\
1 & -1 & 0 & \cdots & 0 & 0 & 0 \\
0 & \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\
0 & \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\
0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\
0 & \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1
\end{bmatrix}
\times
\begin{bmatrix}
\alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1}
\end{bmatrix}
=
\begin{bmatrix}
1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0
\end{bmatrix}
\tag{3.45}
$$

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

$$\begin{cases} \alpha_i = \left| \dfrac{\det A_i^\star}{\det A} \right| \end{cases} \tag{3.46}$$

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

Especially,

$$A_0^\star = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{3.47}$$

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

$$\det A_0^\star = -1$$

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 $\det A \neq 0$.

$$C = \begin{bmatrix} -1 & 0 & \cdots & 0 & 0 & 0 \\ \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{3.48}$$

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.

After a series of column reduction and row reduction actions, we get

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

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

Considering the matrix $\hat{C}$

$$\hat{C} = \begin{bmatrix} -3 & -2 & \cdots & -2 & -2 & -1 \\ \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{3.49}$$

, which is still column linear independence. Considering $0 < \sigma < 1$, the flow matrix is full rank. So $\det A \neq 0$.

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

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

m*n Mesh Network

Considering a general $m * n$ mesh network, such as Fig. 3.9 Fig. 2.1.

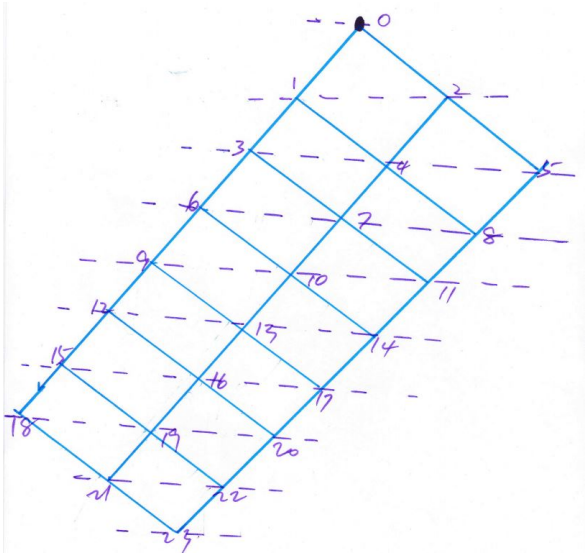

Figure 3.9: 3*8 mesh network. The data injection position is P_0

Utilizing the previous methodology, we obtain the closed-form flow matrix equations for Fig. 3.9:

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_9 \\ \alpha_{12} \\ \alpha_{15} \\ \alpha_{18} \\ \alpha_{21} \\ \alpha_{23} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.50}$$

Also, the flow matrix equations for Fig. 2.1:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.51}$$

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

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

.

3.1.2 Data Injection On The Boundary Processor

After the corner scenario, we extend the methodology to boundary processor load origination.

A single data injection root-node on the boundary processor, as shown in Fig. 3.10.

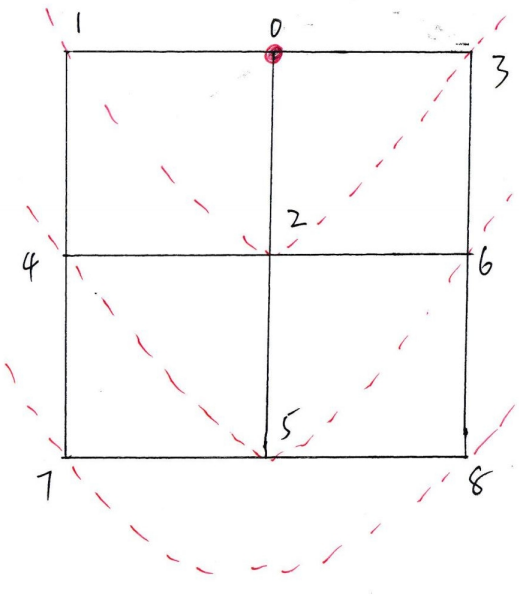

Figure 3.10: The 3*3 mesh network and the root processor is P_0

The timing diagram is Fig. 3.11:

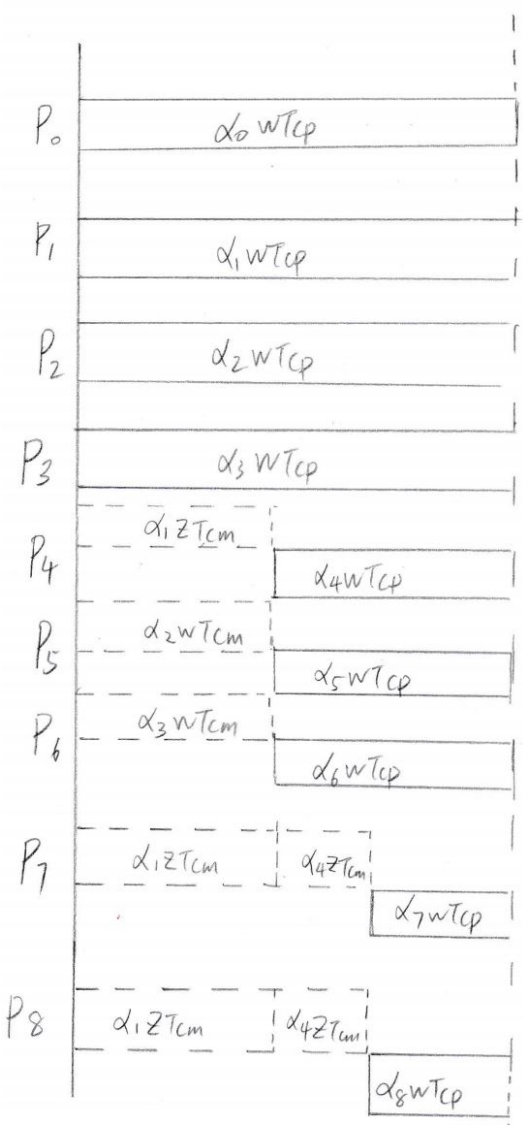

Figure 3.11: The timing diagram for 3*3 mesh network and the data injection occurs on P_0

P_0 P_1 P_2 and P_3 commence processing at $t = 0$ axis timing diagram. P_1 P_2 and P_3 receive load in virtual cut-through mode from P_0. Once that is finished, link $0-1$, $0-2$ and $0-3$ is used to forward load to nodes P_4, P_5 and P_6 in virtual cut-through mode. Once that is finished, the link that have been

so far are used to forward load to P_7 and P_8 in virtual cut-through mode. The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.52) \\
\alpha_1 \omega T_{cp} = T_{f,m} & (3.53) \\
\alpha_2 \omega T_{cp} = T_{f,m} & (3.54) \\
\alpha_3 \omega T_{cp} = T_{f,m} & (3.55) \\
\alpha_1 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.56) \\
\alpha_2 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.57) \\
\alpha_3 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (3.58) \\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (3.59) \\
(\alpha_2 + \alpha_5) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (3.60) \\
\alpha_0 + \cdots + \alpha_8 = 1 & (3.61) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.62) \\
0 < \sigma < 1 & (3.63) \\
0 < \alpha_0 \leq 1 & (3.64) \\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_8 < 1 & (3.65) \\
 & (3.66)
\end{cases}
$$

And the flow matrix form is :

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_4 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.67}$$

The explicit solution is:

$$\begin{cases} \alpha_0 = \dfrac{1}{9-7\times\sigma+2\times\sigma^2} & (3.68) \\ \alpha_1 = \dfrac{1}{9-7\times\sigma+2\times\sigma^2} & (3.69) \\ \alpha_4 = \dfrac{1-\sigma}{9-7\times\sigma+2\times\sigma^2} & (3.70) \\ \alpha_7 = \dfrac{(1-\sigma)^2}{9-7\times\sigma+2\times\sigma^2} & (3.71) \\ & (3.72) \end{cases}$$

The simulation result is shown: The curve α_0 is identified to the curve for

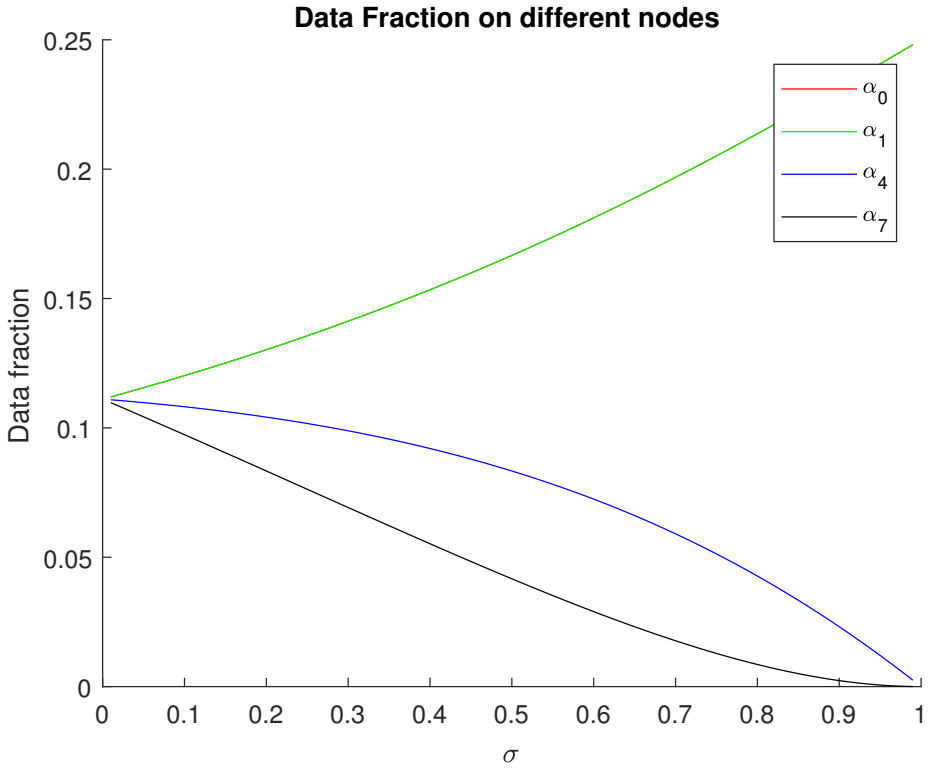

Figure 3.12: The data fraction simulation result of 3*3 mesh network and the data injection happens on the boundary P_0

α_1, α_2 and α_3. The curve α_4 is identified to the curves of α_5 and α_6. The curve α_7 is identified to the curve for α_8. If communication time is zero (extremely fast communication), each processor get $\frac{1}{9}$ of the load. If communication time equals computation time, each of the four start stage nodes (P_0,P_1,P_2,P_3) get $\frac{1}{4}$ of the load fraction.

3.1.3 Data Injection On The Inner Grid Processor

Fig. 3.13 shows that L load on the inner grid processor P_0, P_0, P_1, P_2, P_3 and P_4 commence processing at time = 0 through virtual cut-through

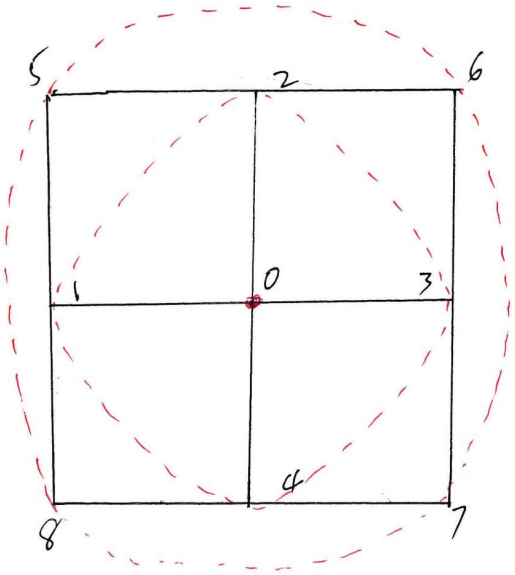

Figure 3.13: 3*3 mesh network. The data injection position is inner grid point P_0

switching. Once this is finished links $0-1$, $0-2$, $0-3$ and $0-4$ are released to forward load to P_5, P_6, P_7 and P_8 in virtual cut-through mode.

The timing diagram for this user case is illustrated as Fig. 3.14:

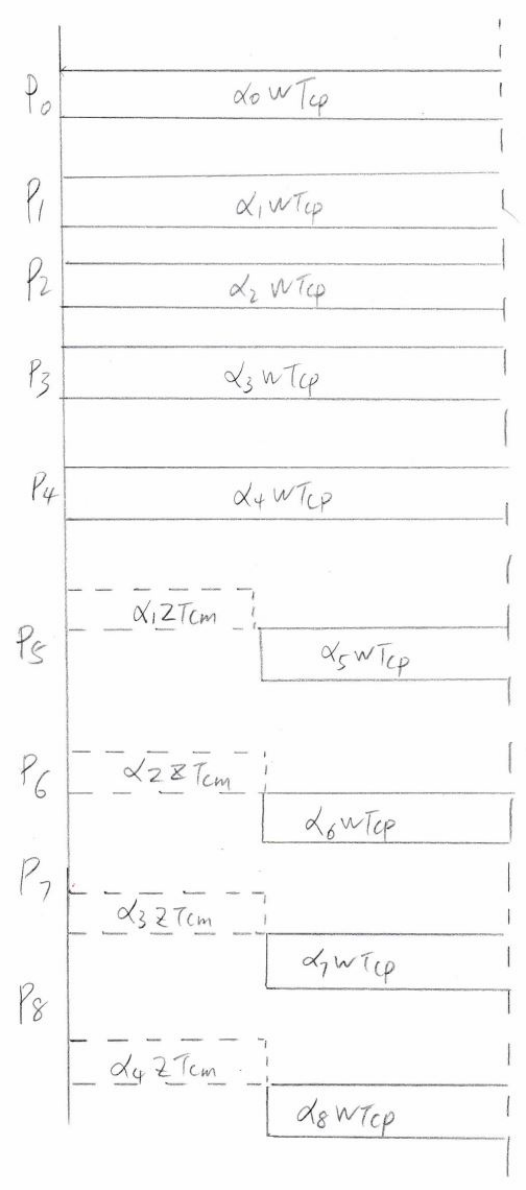

Figure 3.14: The timing diagram for 3*3 mesh network and the data injection is inner grid P_0

The group of equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.73)\\
\alpha_1 \omega T_{cp} = T_{f,m} & (3.74)\\
\alpha_2 \omega T_{cp} = T_{f,m} & (3.75)\\
\alpha_3 \omega T_{cp} = T_{f,m} & (3.76)\\
\alpha_4 \omega T_{cp} = T_{f,m} & (3.77)\\
\alpha_1 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.78)\\
\alpha_1 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (3.79)\\
\alpha_1 z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (3.80)\\
\alpha_1 z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (3.81)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.82)\\
0 < \sigma < 1 & (3.83)\\
0 < \alpha_0 \leq 1 & (3.84)\\
0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_8 < 1 & (3.85)
\end{cases}
$$

The flow matrix form is :

$$
\begin{bmatrix} 1 & 4 & 4 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.86}
$$

The simulation result is Fig. 3.15: P_0 and P_1 have the same α value, so the curve of α_0 and α_1 coincide.

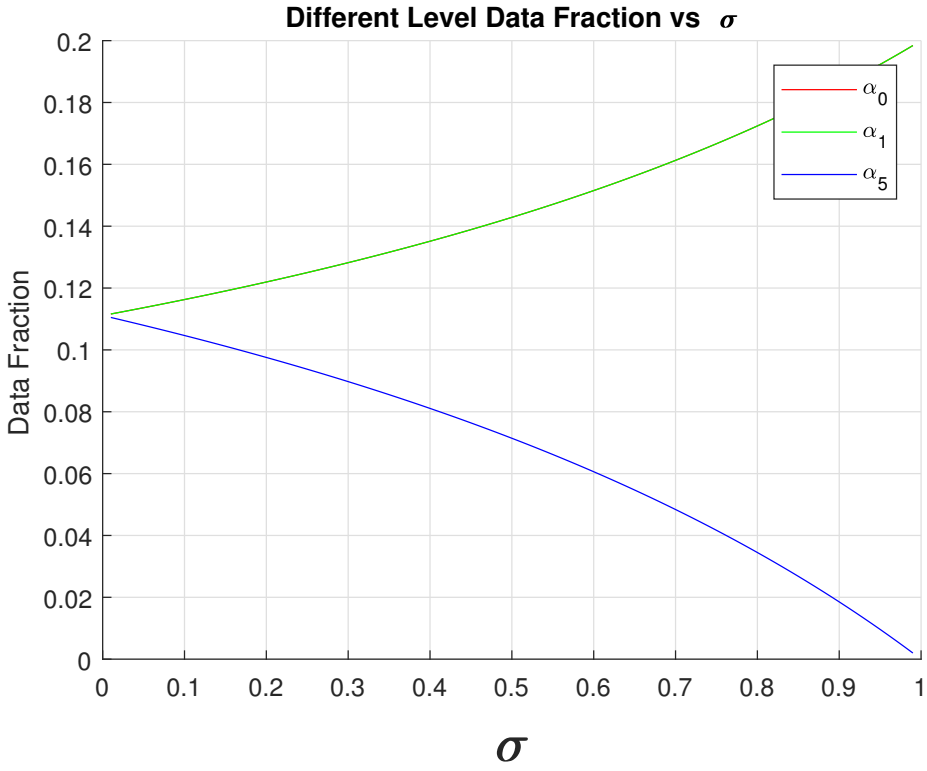

Figure 3.15: 3*3 mesh network. The data injection position is inner grid point P_0

The explicit solution is:

$$\begin{cases} \alpha_0 = \dfrac{1}{9 - 4 \times \sigma} & (3.87) \\ \alpha_1 = \dfrac{1}{9 - 4 \times \sigma} & (3.88) \\ \alpha_5 = \dfrac{1 - \sigma}{9 - 4 \times \sigma} & (3.89) \end{cases}$$

If the communication time is zero (very fast communication), each of nine

processors get $\frac{1}{9}$ of the load. If communication time equals computation time, each of processors P_0, P_1, P_2, P_3 and P_4 get $\frac{1}{5}$ of the load.

3.1.4 Sensitivity Analysis Virtual Cut-through Switching Scenario

From Chapter 2, we know the speedup is :

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

.

Data Injection On The Corner Processor

The simulation result of sensitivity analysis of $2*n$ mesh network Fig. 3.7 is as follows: One can see speedup increases with an increasing number of

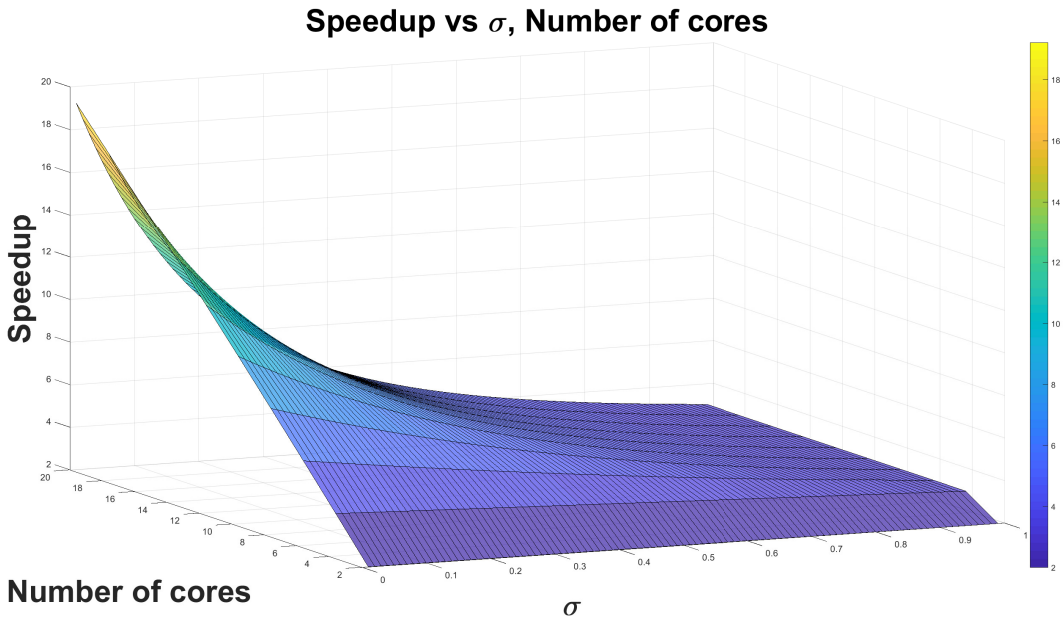

Figure 3.16: Sensitivity analysis result of 2*n mesh network (where $n = 1 \cdots 10$)

core (i.e. processor) and also increases with decreasing σ (that is increasing

communication speed relative to computation speed).

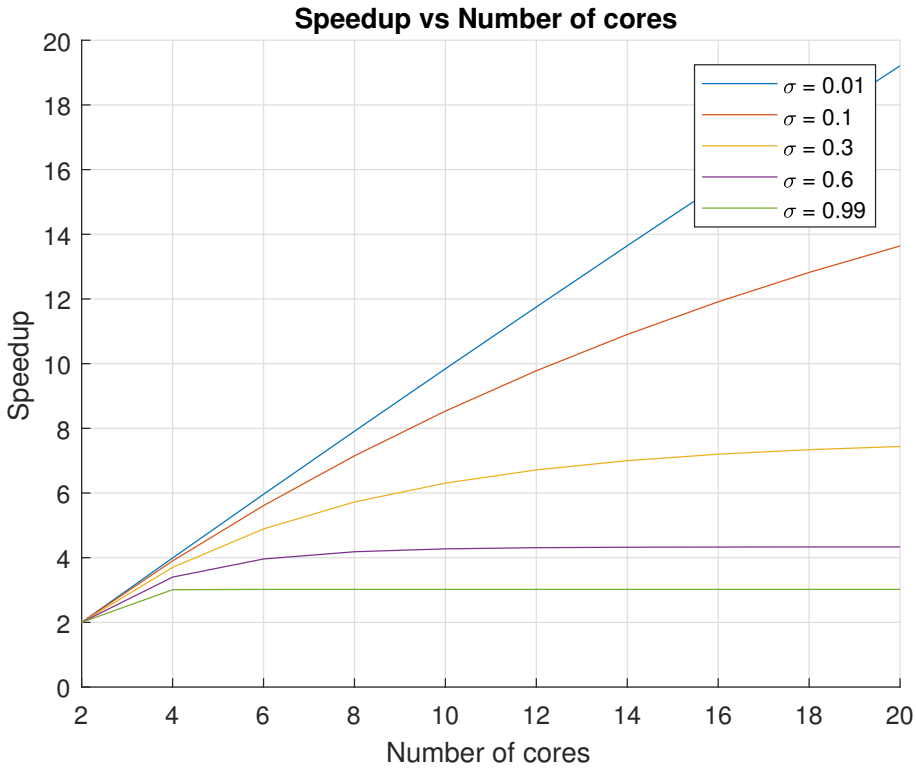

Figure 3.17: Speedup curves of 2*n mesh network (where $n = 1 \cdots 10$)

For a large number of cores and σ close to one, speedup is three times. P_0 and two adjacent neighbor processor do most of the processing.

Data Injection On The Boundary Processor

For a $3*n$ mesh network Fig. 3.9, the data injection happens on boundary processor P_2.

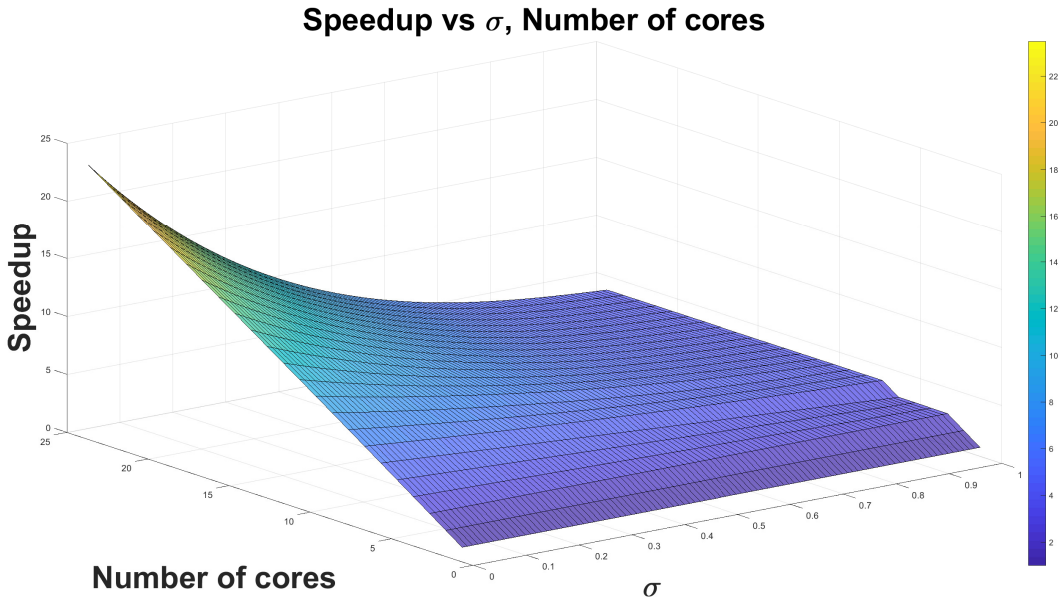

Figure 3.18: Sensitivity analysis result of 3*n mesh network and the injection position on boundary processor P_2 (where $n = 1 \cdots 8$)

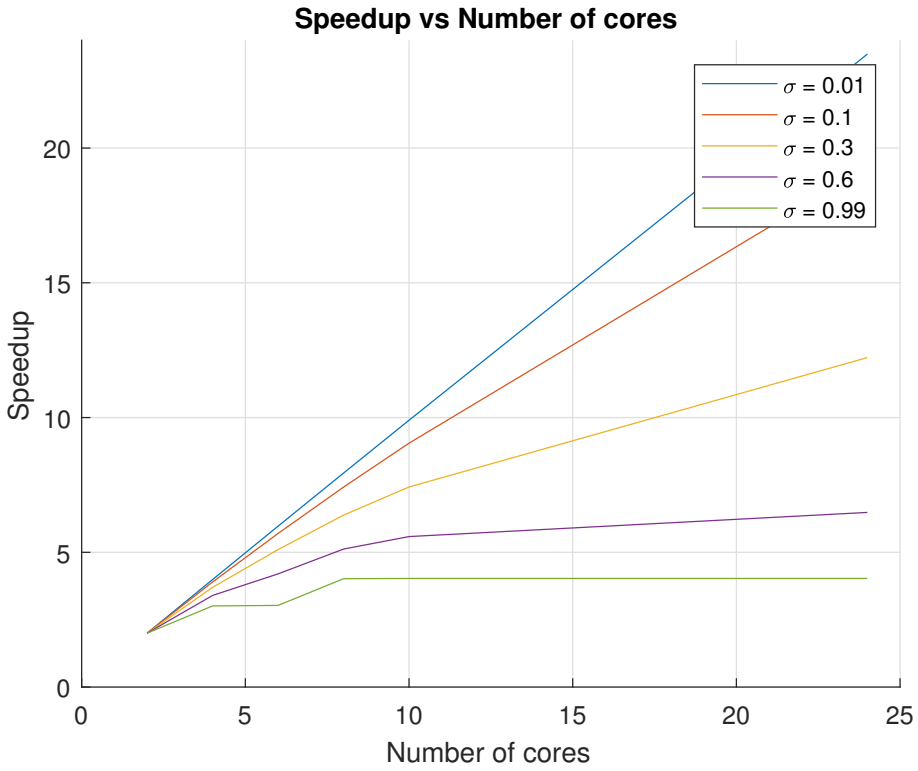

Figure 3.19: Speedup curve of 3*n mesh network (where $n = 1 \cdots 8$)

Fig. 3.18 and Fig. 3.19 show that if the value $\sigma > 0.2$, the speedup simulation effect increases rapidly. If the value $\sigma < 0.1$, the number of cores has linear impact on the speedup performance. If the number of cores is for σ close to one, the speedup is about four since only the three processors close to P_0 and P_0 do almost all the processing.

Data Injection On The Inner Grid Processor

For a $5 * n$ mesh network is in Fig. 2.1, L originate on the inner grid P_{12} and the simulation result says:

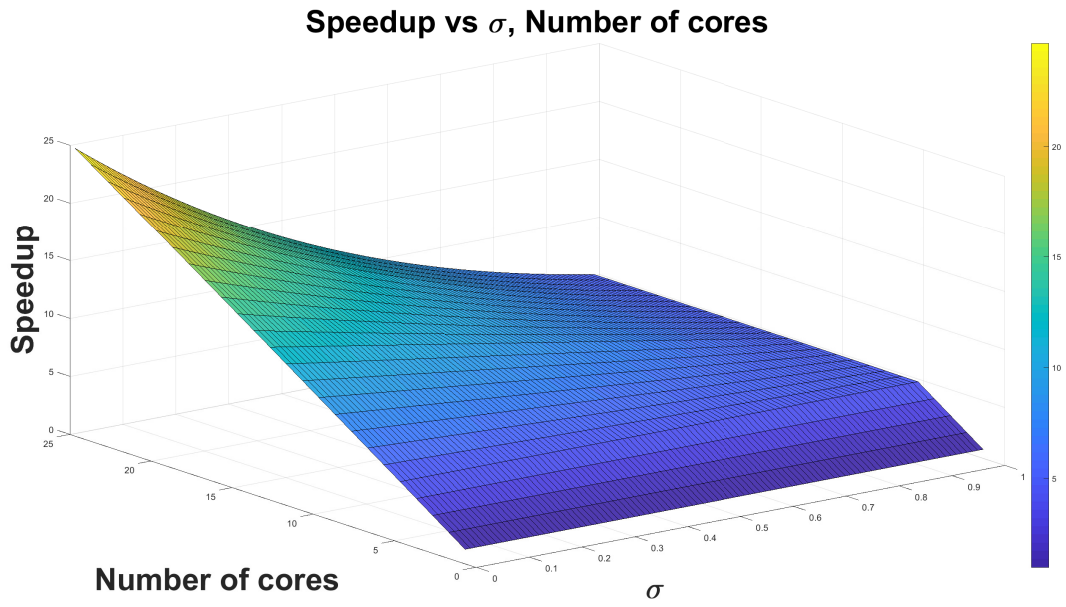

Figure 3.20: Sensitivity analysis result of $5 * n$ mesh network and the injection position on inner processor P_{12} (where $n = 1 \cdots 5$)

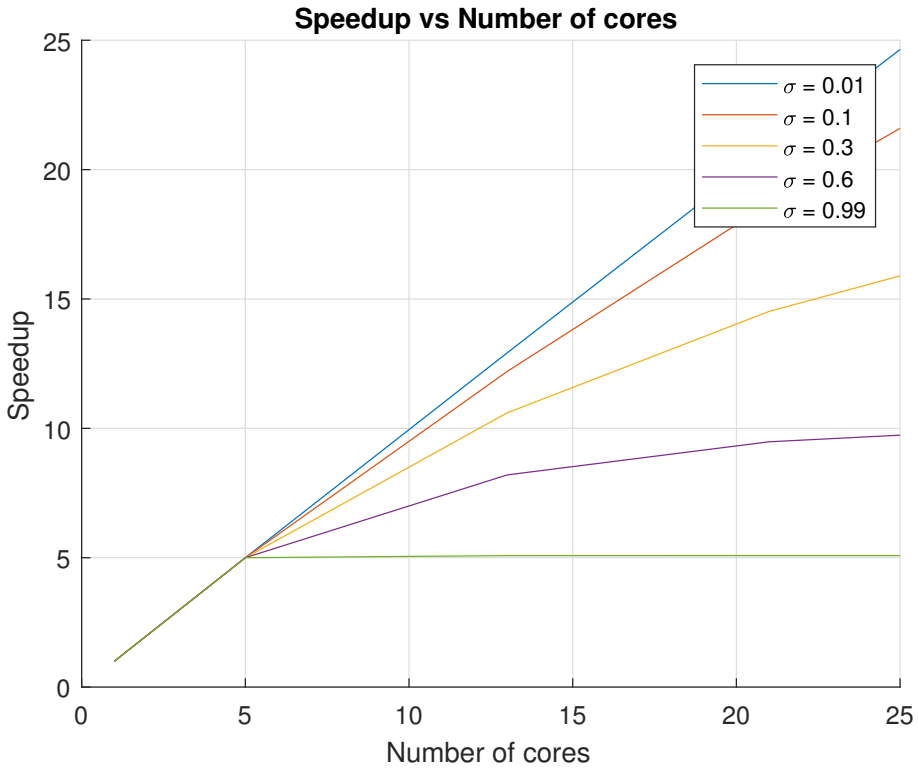

Figure 3.21: Speedup curves of 5*n mesh network Fig. 2.1 (where $n = 1 \cdots 5$)

If the number of processor > 5, the cluster equivalence computation ability is at least 5 time speedup. There are 5 processors in the first stage. For small σ speedup growth is linear. For large σ and a large number of cores, speedup is about five as only P_0 and four adjacent processors do almost all the processing.

3.1.5 Multi-source Uniform Data Fraction

We extend the single source assignment to multi-source assignment problem [10] [4]. According to each processor, we concentrate on the processors' geographical location P_i, data fraction assigned α_i.

Assume the data fraction is specially uniform. For example, the workload is unit 1 and there are k different data injection options. So each data injection is assigned $\frac{1}{k}$ workload.

From the data injection position relationship we consider three different situations :

- Data injection positions consist of a connected subgraph G_L of G.
- Data injection processors don't connect with each other.
- Some data injection positions consist of some connected subgraphs and some are individual injection.

Situation I

If the data injection positions consist of a connected subgraph of G, we use G_L to present it.

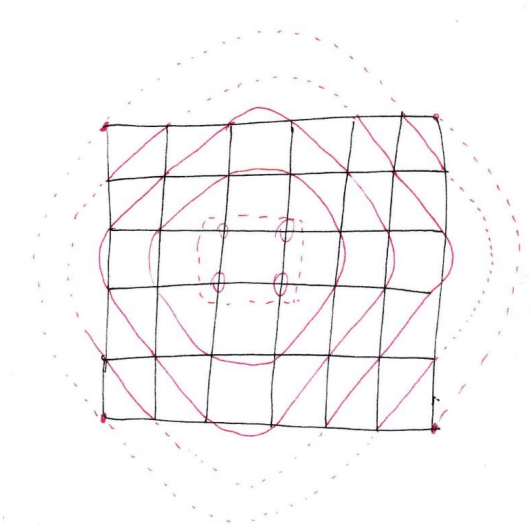

Figure 3.22: Data injection consists of a connected subgraph of G

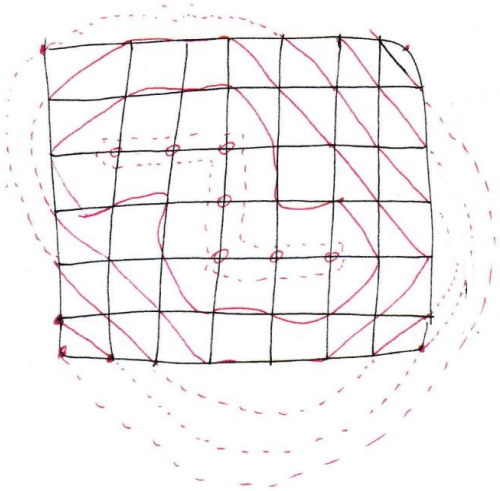

Figure 3.23: Data injection consists of a connected subgraph of G

Connected subgraph Fig. 3.22 and connected subgraph Fig. 3.23 illustrate two examples that the data injections consist of connected subgraphs of G.

Our objective is to propose a general algorithm framework to tackle this situation and give quantitative model analysis. This algorithm is named as ***Equivalence Processor Scheduling Algorithm (EPSA)***.

Algorithm 1 Equivalence Processor Scheduling Algorithm (EPSA)

Input: n data injection positions

Output: $m * n$ processor data fractions

Collapse the data injection processors into one "big" equivalent processor [3].

Calculate $m * n$ processor's D_i.

Obtain the flow matrix A_i.

Calculate $m * n$ processors data fraction α_i.

- The time complexity of calculating the determinant is $O(r^3)$ with Gaussian elimination or LU decomposition. r is the rank of flow matrix and r is $O(\max(m, n))$.
- The time complexity of calculating the flow matrix A_i is $O(k * m * n)$. k is the number of data injection.
- The total time complexity is $O(k * \max(m, n)^3)$.

For example, connected subgraph Fig. 3.22's ***flow matrix*** is :

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.90}$$

The simulation result illustrates as follows:

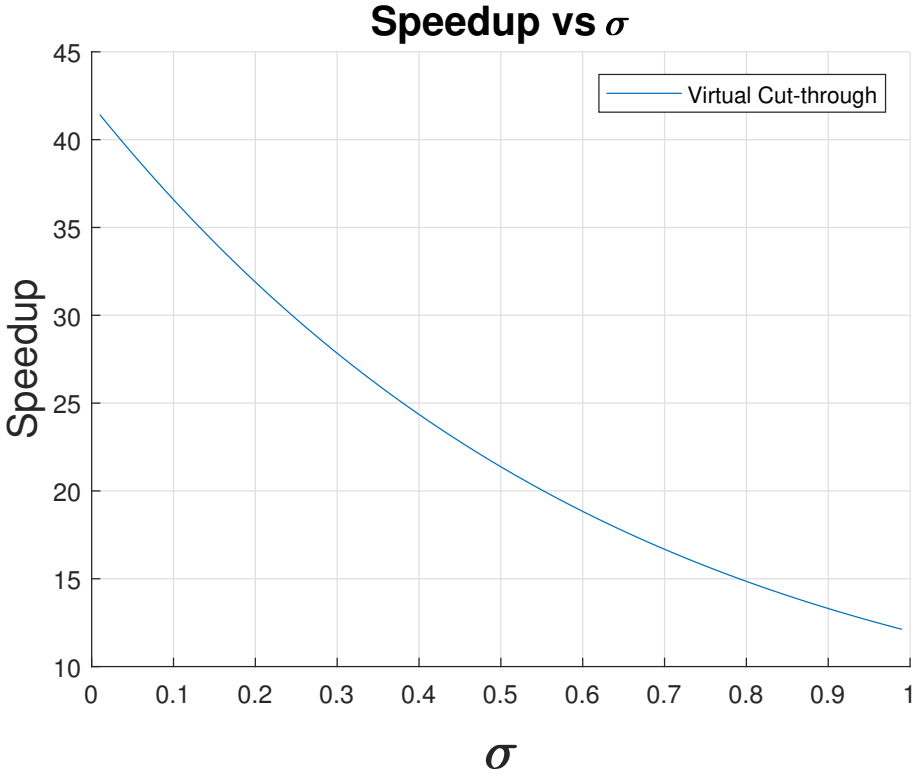

Figure 3.24: Connected subgraph Fig. 3.22 speedup vs σ

This simulation says the best performance happens on value $\sigma \leq 0.05$, which hits about 40 times speedup. There are about 12 processors at the start

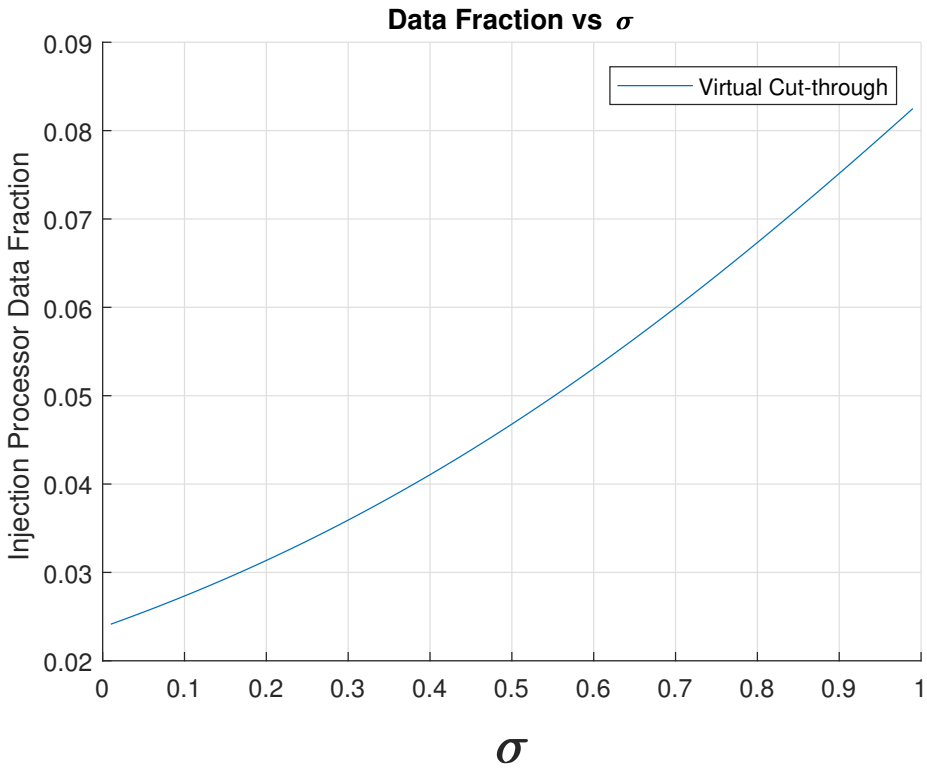

Figure 3.25: Injection Processor Data Fraction vs σ

stage. So if $\sigma \approx 1$, that is, communication time equals computation time, the network achieves almost 12 times speedup performance.

For example, connected subgraph Fig. 3.23's ***flow matrix*** is :

$$\begin{bmatrix} 7 & 14 & 15 & 10 & 6 & 3 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.91}$$

The simulation result illustrates as follows:

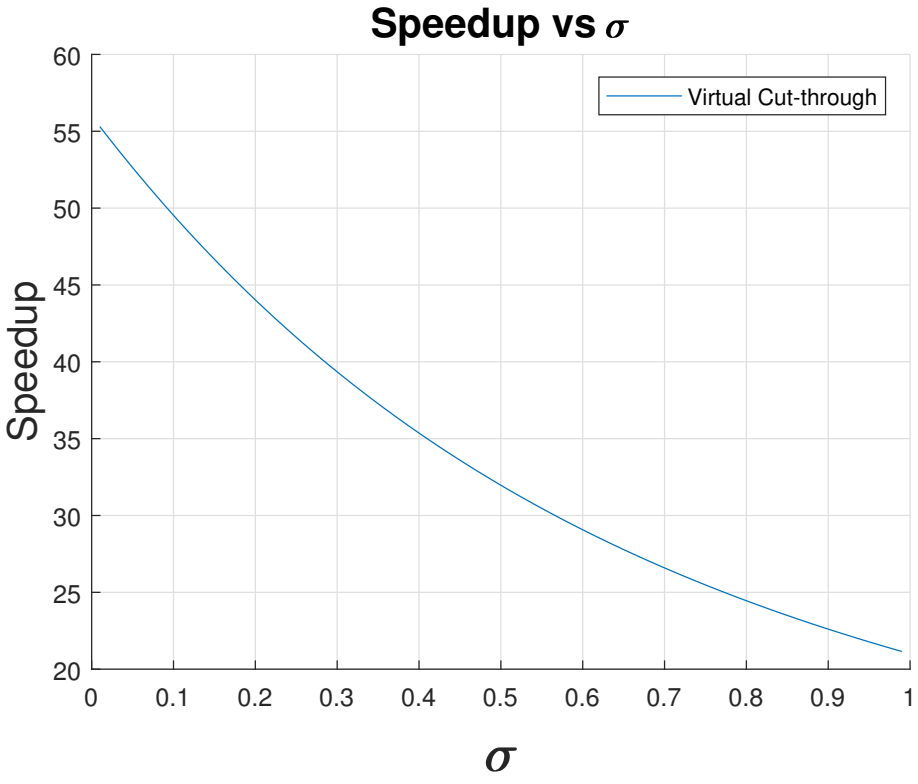

Figure 3.26: Connected subgraph Fig. 3.23 speedup vs σ

This simulation says the best performance happens on value $\sigma \leq 0.05$,

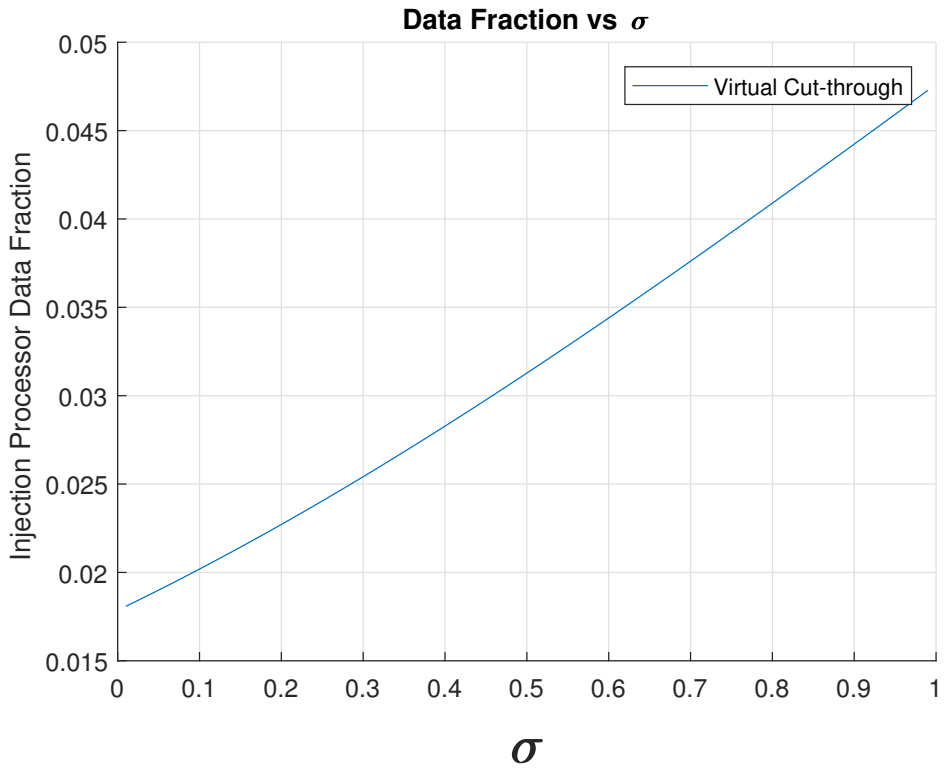

Figure 3.27: Injection Processor Data Fraction vs σ

which hits about 55 times speedup. There are about 21 processors in the start stage. So if $\sigma \approx 1$, the network achieves almost 21 times speedup performance.

Situation II

Now the data injections don't consist of a connected subgraph of G. Jia [11] proposes a generic algorithm to utilize the nearest data injection principle to tackle this scenario. In the dividable load intricate applications, for example, big file transmission, HPC scientific computation, like BlueGene [12] or Hadoop job.

Our objective is to propose a general algorithm framework to utilize less processors and achieve the same finish time. This is a critical issue for networks on chip (NOC) where the number of active processor is closely related to the chip power computation chip. Power computation is a limiting factor to scalability in such NOC.

Voronoi diagram division Fig. 3.28 shows 10 Voronoi cells division and Voronoi diagram division speedup curves Fig. 3.29 figure out the speedup of each cells.

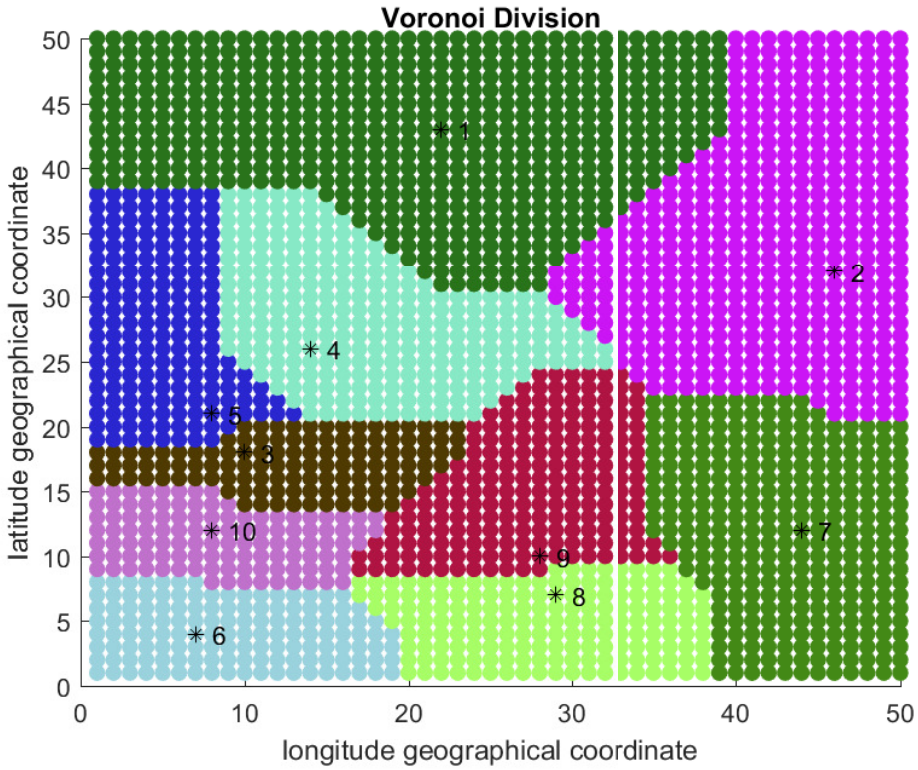

Figure 3.28: 10 Manhattan distance Voronoi Cells

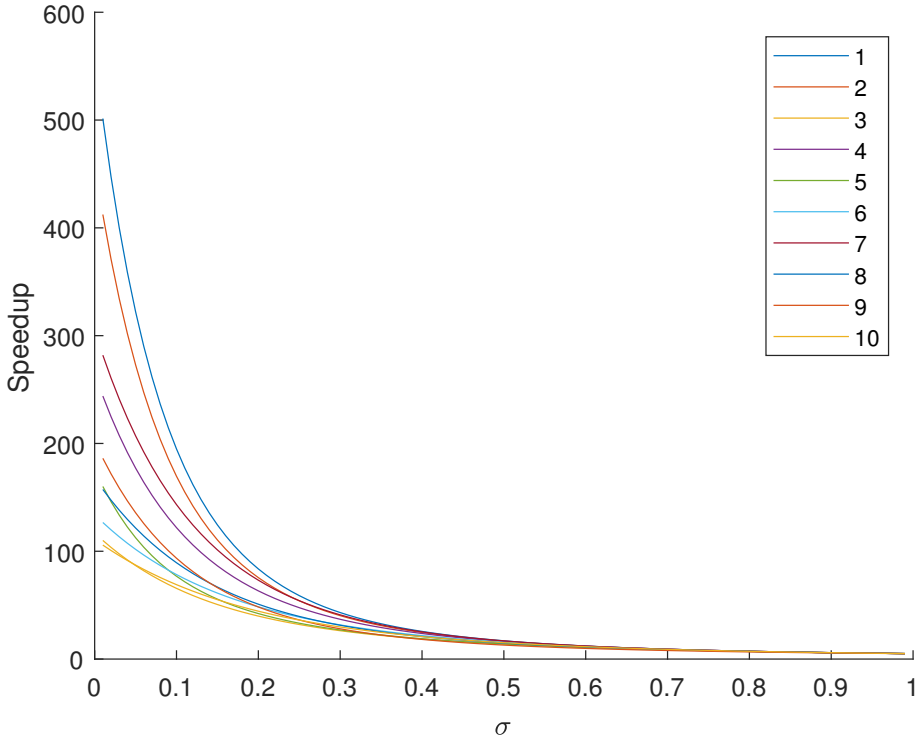

Figure 3.29: 10 Manhattan distance Voronoi Cells speedup curves

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

Algorithm 2 Reduced Voronoi Diagram Algorithm (RVDA)

Calculate k Voronoi cells with Manhattan distance.

Calculate k Voronoi cells' radius R_i.

Calculate k flow matrix A_i.

$depth_{min} = \min(Speedup_i)$'s R_i.

Calculate the reduced Voronoi cells by setting the $depth_i = depth_{min}$ in each Voronoi cell.

Calculate reduced Voronoi cell's flow matrix $\hat{A}_i$.

Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves.

- The time complexity of Manhattan distance Voronoi cells is $O(k*m*n)$;
- The time complexity of flow matrix determinant is $O(r^3)$. r is the rank of flow matrix.
- So the total time complexity is $O(k * \max(m, n)^3)$.

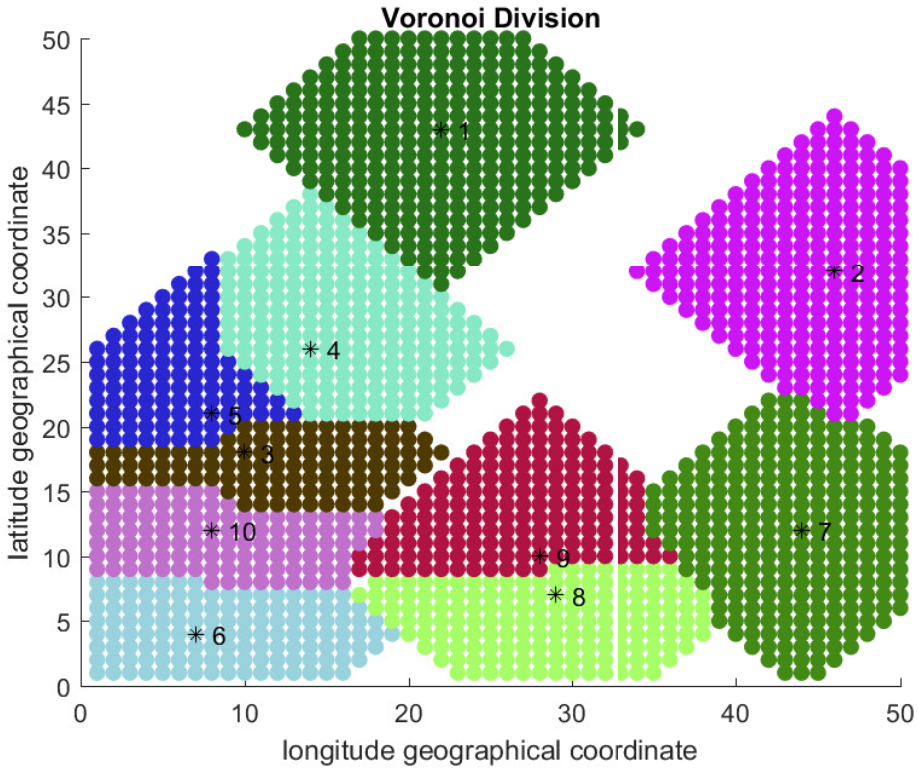

Figure 3.30: 10 reduced Voronoi cells

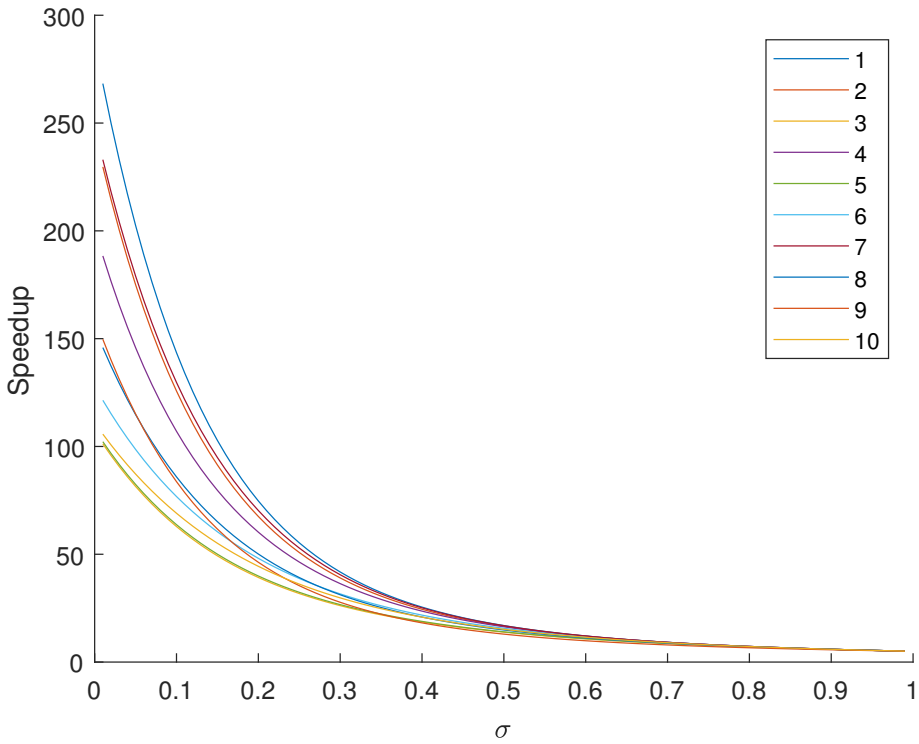

Figure 3.31: 10 reduced Voronoi cells's speedup curves

Reduced Voronoi cells Fig. 3.30 and reduced Voronoi speedup curves Fig. 3.31 show this algorithm obtains the same running time yet saves about 30% processors.

- Voronoi speedup curves Fig. 3.29 show that $\sigma < 0.2$, the ratio $\max(\frac{maxspeedup}{minspeedup}) = \frac{500}{100} = 5$
- Reduced Voronoi speedup curves Fig. 3.31 show that $\sigma < 0.2$, the ratio is $\max(\frac{maxspeedup}{minspeedup}) = \frac{270}{100} = 2.7$.

It displays that 10 cell's equivalence computation is more balanced than initial setting, and the whole cluster finishes load within the same time by less processors. After 1000 round random sampling experiments, we obtain the average saved processors ratio in Fig. 3.32.

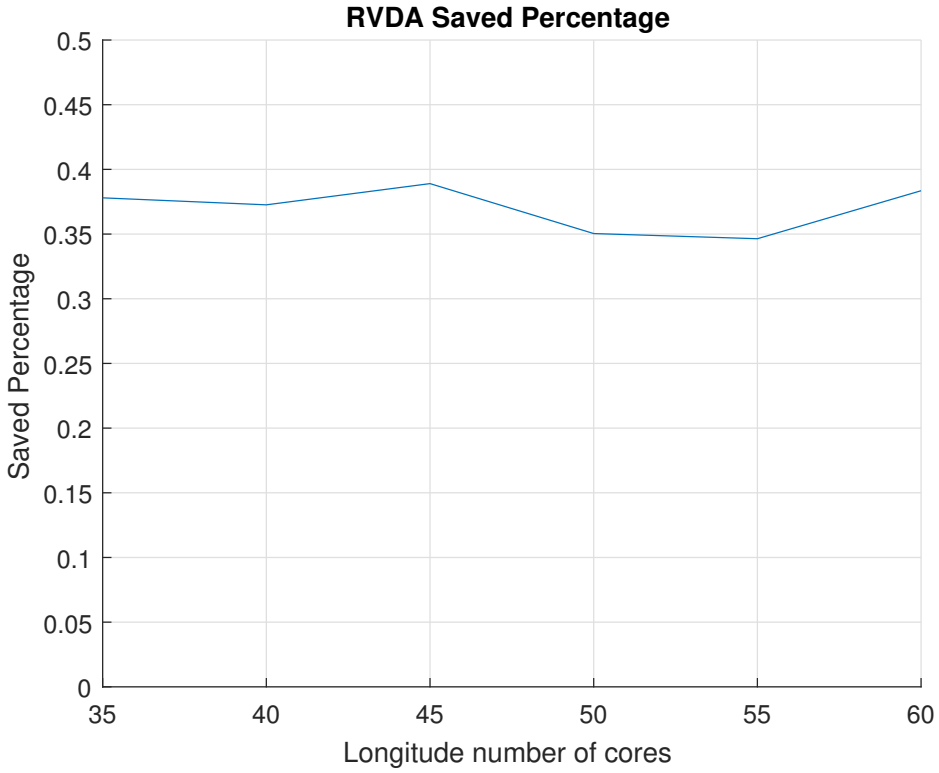

Figure 3.32: Reduced Voronoi division algorithm average saved processors' percentage

From the average saved processors ratio Fig. 3.32, it shows the average

percentage of saved processor is about 35%.

Situation III

If there are some nodes consisting some connected subgraphs and there are some nodes individual processors, our objective to finish the load in the same time and save processors. We process an Reduced Constrained Voronoi Diagram Algorithm (RCVDA) to tackle this situation.

Algorithm 3 Reduced Constrained Voronoi Diagram Algorithm (RCVDA)

Collapse the connected data injection processors into some "big" equivalent processors, individually.

Calculate k constrained Manhattan distance Voronoi cells [13].

Calculate k constrained Voronoi cells' radius R_i.

Calculate k constrained Voronoi cells' flow matrix A_i.

Calculate k constrained Voronoi cells' $Speedup_i$.

Set the $depth_{min} = \min(Speedup_i)$'s radius.

Calculate k reduced constrained Voronoi cells by setting the $depth_i = depth_{min}$ in each Voronoi cell.

Calculate Voronoi cell's flow matrix $\hat{A}_i$.

Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves

- The time complexity of Manhattan distance constrain Voronoi cells is $O(k * m * n)$. k is the number of data injection.
- The time complexity of flow matrix determinant is $O(r^3)$. r is the rank of flow matrix.
- So the total time complexity is $O(k * \max(m, n)^3)$.

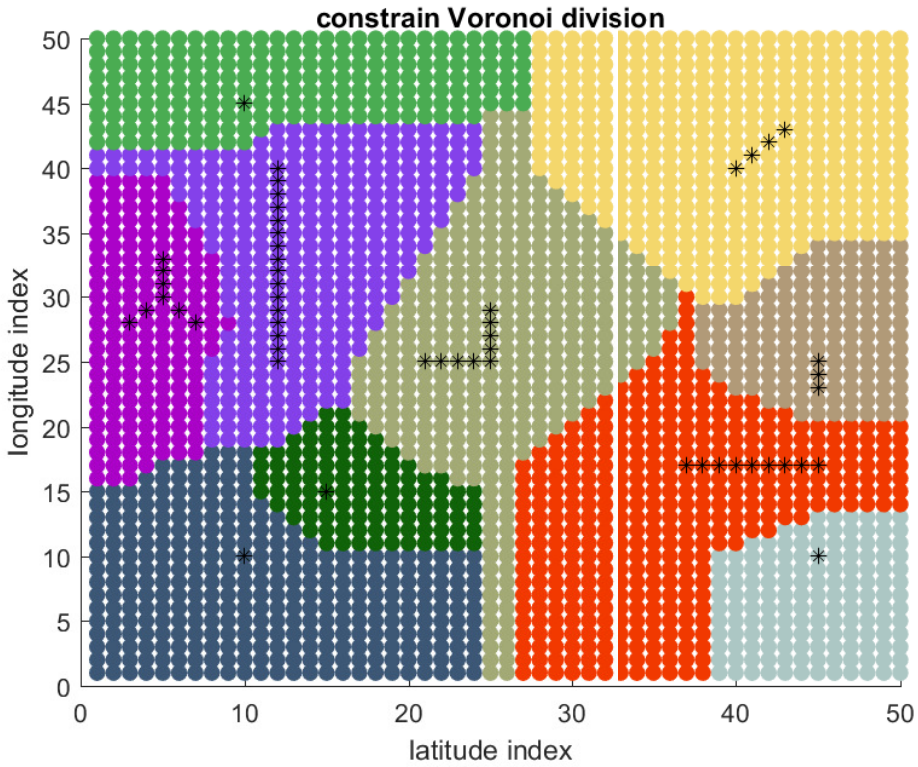

Figure 3.33: Connected subgraphs and individual data injections on mesh network

In this user case, we save about $294/2500 \approx 11.7\%$ processors.

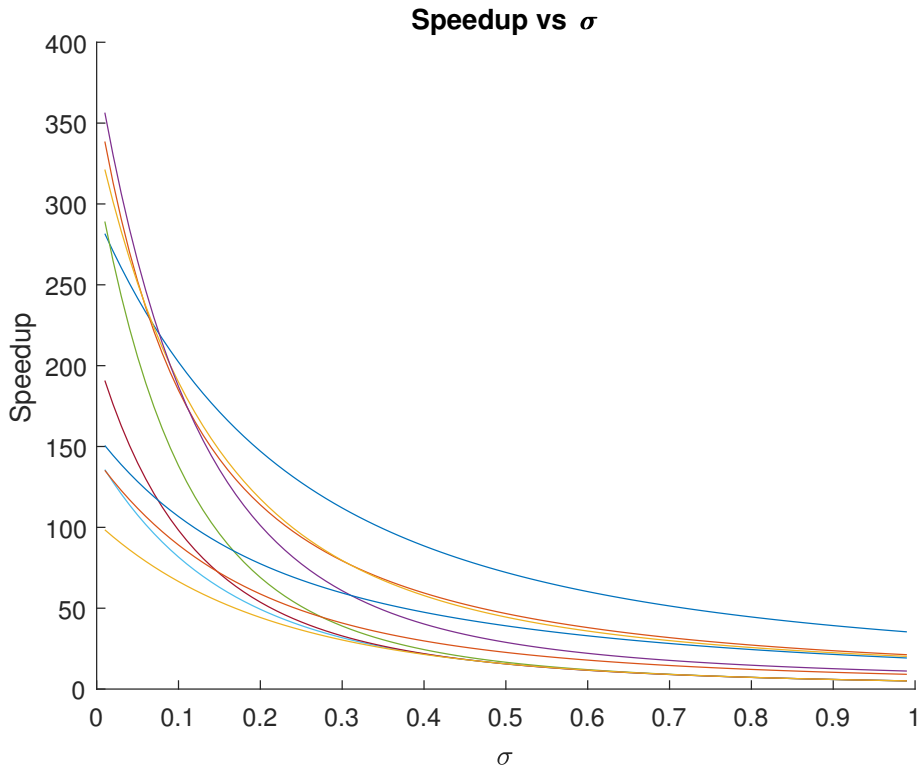

Figure 3.34: Speedup curves for Situation III

3.1.6 Multi-source Non-uniform Data Injection

From the data injection position relationship, we consider three different situations :

- Data injection positions consist of a subgraph G_L of G.
- Data injection processors don't connect with each other.
- Some data injection positions consist of some connected subgraphs and some are individual injection.

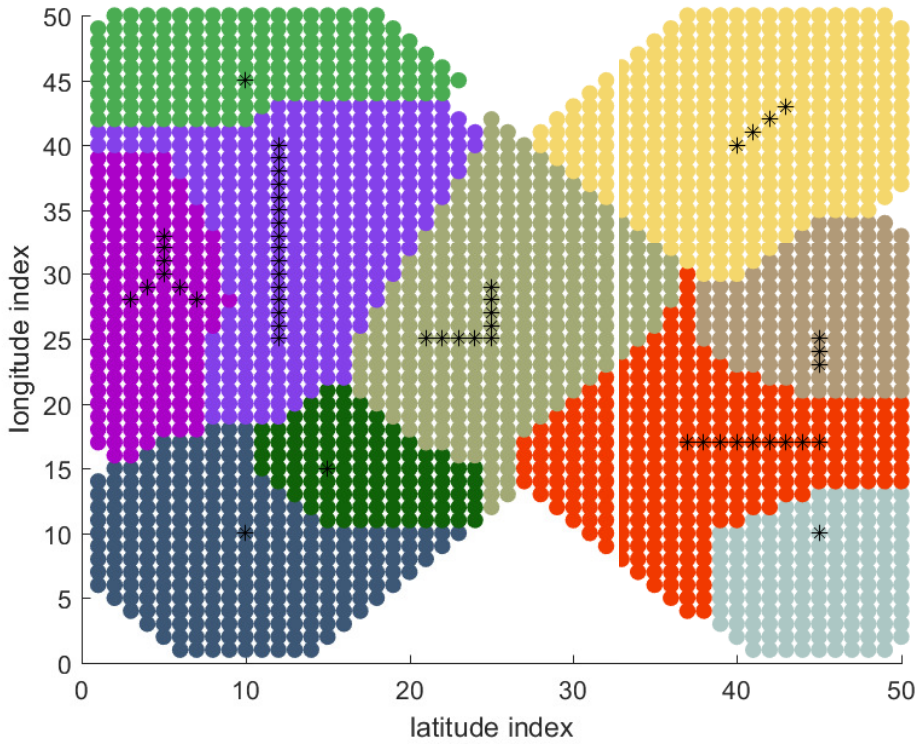

Figure 3.35: Reduced constrained Voronoi diagram for connected subgraphs and individual data injections on mesh network

Situation I

We propose a suboptimal algorithm framework to address this kind of problem. The main idea is re-balancing the data fraction between the nodes in connected subgraph first. Then, run Equivalence Processor Scheduling Algorithm.

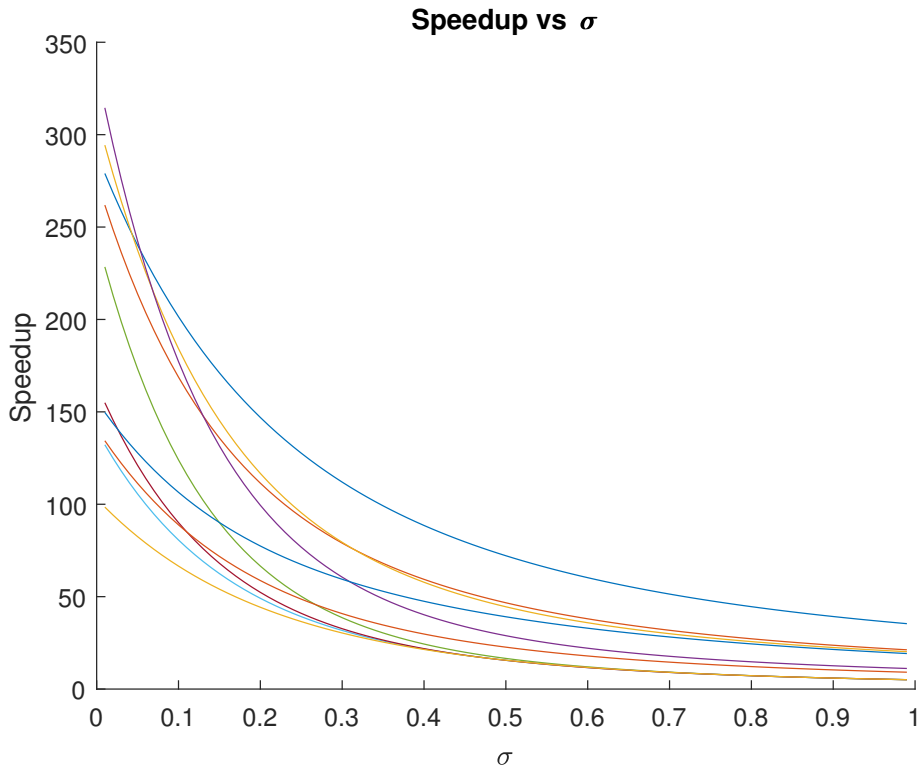

Figure 3.36: Speedup curves of connected subgraphs and individual data injection on mesh network

In $local_s$ stage, our target is from a data fraction distribution to a uniform distribution and the minimum cost is called Earth Mover Distance [14].

- The time complexity of calculating Voronoi diagram is $O(k * m * n)$. k is the number of data injection.
- The time complexity of calculating determinant is $O(r^3)$ with Gaussian elimination or LU decomposition. r is the rank of flow matrix.
- The total time complexity is $O(k * \max(m, n)^3)$.

Algorithm 4 Improved Equivalence Processor Scheduling Algorithm (IEPSA)

$local_s$:

Re-balance the connected subgraph G_L node's fraction distribution to be uniform.

$global_s$:

Collapse the data injection processors into one "big" equivalent processor [3].

Calculate $m * n$ processor's D_i

Obtain the flow matrix A_i.

Calculate $m * n$ processors data fraction α_i.

Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves.

Situation II

We utilize the Reduced Voronoi Diagram Algorithm (RVDA) to address this problem.

Situation III

Algorithm 5 Improved Reduced Constrained Voronoi Diagram Algorithm (IRCVDA)

$local_s$:

Collapse the connected data injection processors into some "big" equivalent processors, individually.

Redistribute the data fraction to uniform fraction within each connected subgraph processors.

$global_s$:

Calculate k constrained Manhattan distance Voronoi cells [13].

Calculate k constrained Voronoi cells' radius R_i.

Calculate k constrained Voronoi cells' flow matrix A_i.

Calculate k constrained Voronoi cells' $Speedup_i$.

Set the $depth_{min} = \min(Speedup_i)$'s radius.

Calculate the reduced constrained Voronoi cells by setting the $depth_i = depth_{min}$ in each Voronoi cell.

Calculate Voronoi cell's flow matrix $\hat{A}_i$.

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells' speedup curves

3.2 Store and Forward Switching Scenario

In the store and forward scenario, the processors solely start to process as soon as each processor receives its entire load assignment [15].

This subsection concerns the processors store and forward switching. Because of store and forward switching, the processors simultaneously receive the data and solely start to process it as soon as each processor receives its entire load assignment.

3.2.1 Data Injection on The Corner Processor

2*2 Mesh Network

The timing diagram of Fig. 3.1 is shown:

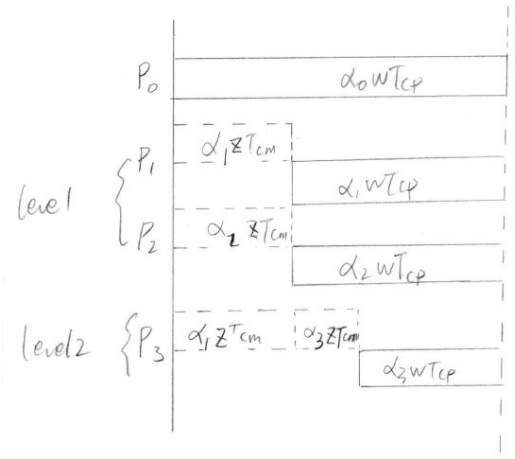

Figure 3.37: The timing diagram for 2*2 mesh network without front-end.

The corresponding group of equations are as follows:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (3.92) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.93) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.94) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.95) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.96) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (3.97) \\ 0 < \sigma < 1 & (3.98) \\ 0 < \alpha_0 \leq 1 & (3.99) \\ 0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 & (3.100) \end{cases}$$

The matrix closed-from is presented as:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.101)$$

The explicit solution is:

$$\begin{cases} \alpha_0 = (\dfrac{\sigma+1}{\sigma+2})^2 & (3.102) \\ \alpha_1 = \dfrac{\sigma+1}{(\sigma+2)^2} & (3.103) \\ \alpha_3 = \dfrac{1}{(\sigma+2)^2} & (3.104) \end{cases}$$

The simulation result for Fig. 3.1 is provided in Fig. 3.38:

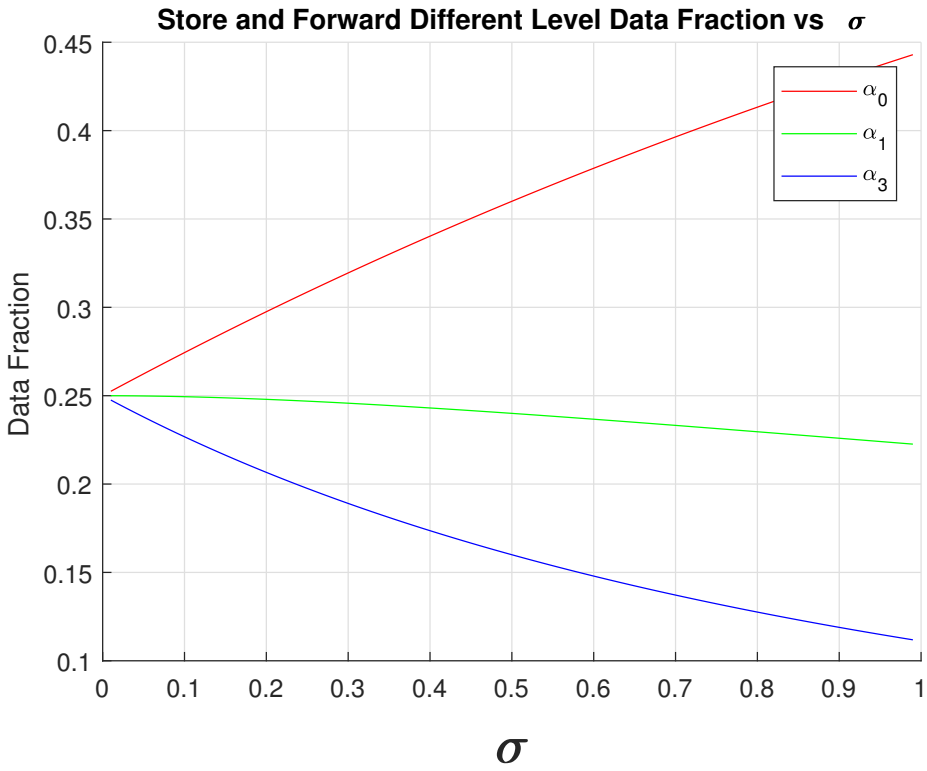

Figure 3.38: Data fraction curves vs σ

A 2∗2 mesh network Fig. 3.1's data fraction curves Fig. 3.38 explains that as the value σ grows up, the fraction assigned to P_0 increases, the fractions distributed to $level_1$ and $level_2$ reduces. 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 equaling to the processor computation capacity, there is solely 11% data is deployed to the $level_2$. In addition, if the $\sigma > 1$, it means that the 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 Mesh Network

P_0 starts to process the assigned workload and it starts to transfer the data. P_1 and P_2 are idle until they get their data fraction from P_0 through the link $0-1$ and $0-2$.

According to the $level_2$, the similar situation happens to P_3 and P_4. They both start to transmit the α_5 after they totally receive the appropriate workload. In other words, P_5 has to wait until the previous two layers processors obtain their own data.

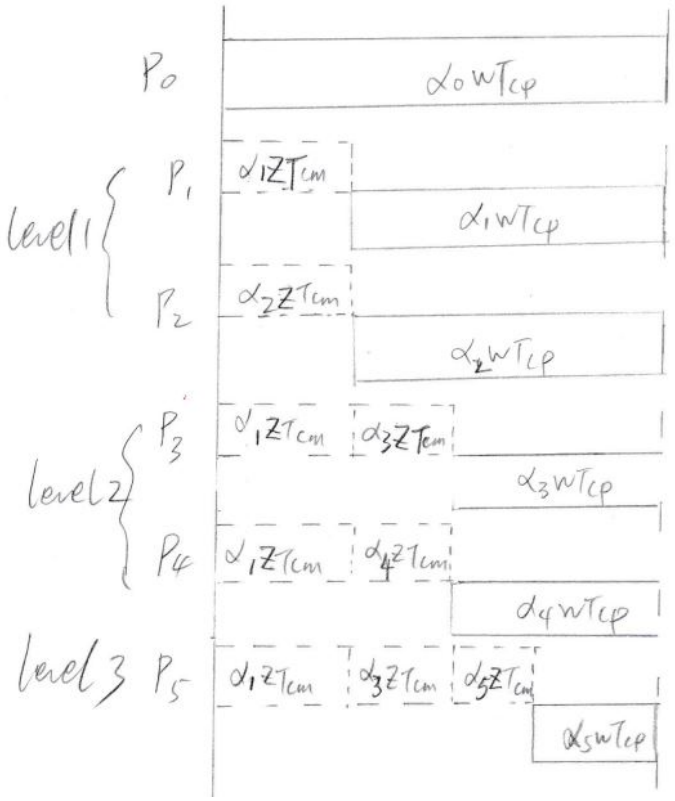

Figure 3.39: The timing diagram for store and forward switching 2*3 mesh network.

In addition, the group of equations are as follows:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (3.105) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.106) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.107) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.108) \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.109) \\ (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.110) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.111) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (3.112) \\ 0 < \alpha_0 \leq 1 & (3.113) \\ 0 \leq \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 < 1 & (3.114) \end{cases}$$

The flow matrix is :

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.115)$$

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 explicit solution is:

$$\begin{cases} \alpha_0 = (\dfrac{\sigma+1}{\sigma+2})^3 & (3.116) \\ \alpha_1 = \dfrac{(\sigma+1)^2}{(\sigma+2)^3} & (3.117) \\ \alpha_3 = \dfrac{\sigma+1}{(\sigma+2)^3} & (3.118) \\ \alpha_5 = \dfrac{1}{(\sigma+2)^3} & (3.119) \end{cases}$$

The simulation result is:

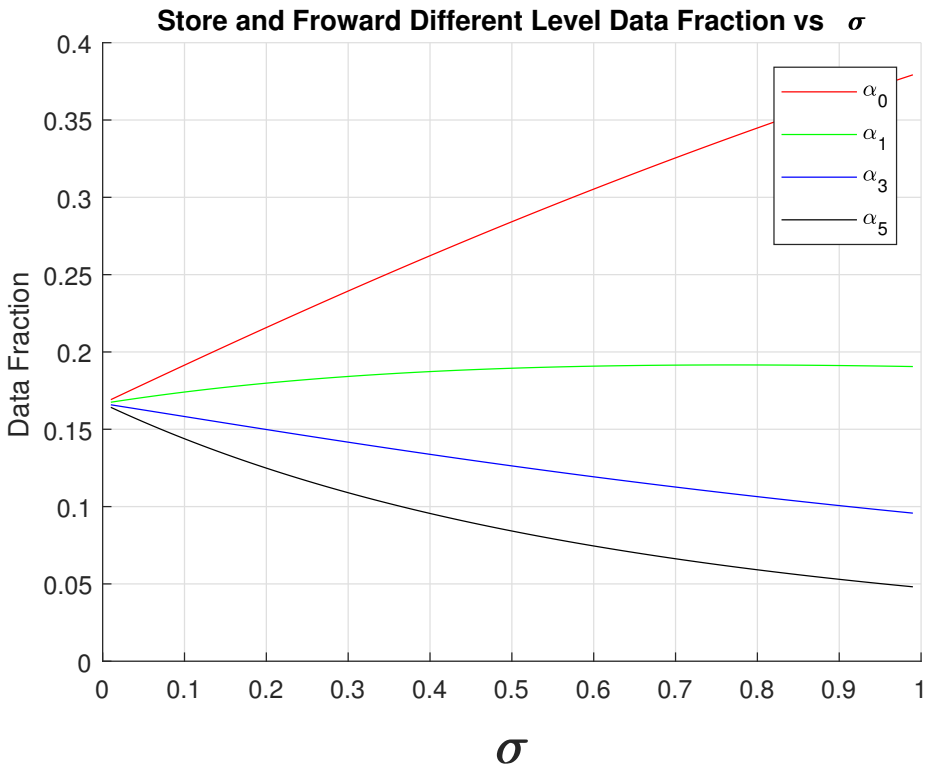

Figure 3.40: Data fraction curve vs σ

2*n Mesh Network

Considering a $2 * 10$ mesh network Fig. 3.7, the equations are demonstrated as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.120) \\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.121) \\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.122) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.123) \\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.124) \\
(\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.125) \\
\vdots & (3.126) \\
(\alpha_1 + \alpha_3 + \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (3.127) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.128) \\
0 < \sigma < 1 & (3.129) \\
0 < \alpha_0 \leq 1 & (3.130) \\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (3.131)
\end{cases}
$$

Utilize $\sigma^\star$ to present $-(\sigma+1)$ and then the flow matrix form for the group of equations is :

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & \sigma^\star & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{3.132}$$

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

$$\begin{cases} \alpha_i = \left| \dfrac{\det A_i^\star}{\det A} \right| \end{cases} \tag{3.133}$$

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

We use $-\sigma - 2 = \epsilon$ and $\sigma^\star - 2 = \beta$.

After a series of column reduction and row reduction, the flow matrix changes as follows :

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

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

We define

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

$0 < \sigma < 1$, then $-2 < \sigma^\star < -1$, which means C is column linear

independent, after column and row reduction.

Further, we define

$$\hat{C} = \begin{bmatrix} \beta & -2 & \cdots & -2 & -2 & -1 \\ \epsilon & \beta & \cdots & 0 & 0 & 0 \\ \epsilon & \epsilon & \beta & 0 & \cdots & 0 \\ \epsilon & \epsilon & \epsilon & \beta & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \epsilon & \epsilon & \cdots & \epsilon & \epsilon & \beta \end{bmatrix}$$

$\hat{C}$ is also full rank.

So the flow matrix A is full rank, that is, $\det A \neq 0$ and $\det A^* \neq 0$. 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 Mesh Network

Referring to a $5 * 5$ mesh network Fig. 2.1, we utilize $\sigma^\star$ to present the $-(\sigma + 1)$.

The flow matrix closed-form is:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.134)$$

3.2.2 Data Injection on The Boundary Processor

A $3 * 3$ mesh network Fig. 3.10 shows an example of boundary processor P_0 receiving L.

The timing diagram for Fig. 3.10 is Fig. 3.41.

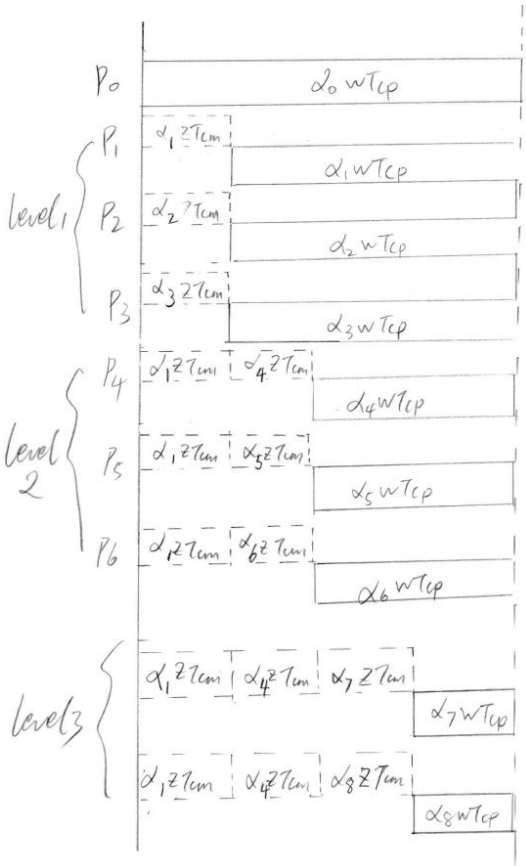

Figure 3.41: The timing diagram for 3*3 boundary data injection on P_0

The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (3.135)\\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.136)\\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.137)\\
\alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.138)\\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.139)\\
(\alpha_2 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.140)\\
(\alpha_3 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (3.141)\\
(\alpha_1 + \alpha_4 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (3.142)\\
(\alpha_1 + \alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (3.143)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.144)\\
0 < \sigma < 1 & (3.145)\\
0 < \alpha_0 \leq 1 & (3.146)\\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 & (3.147)
\end{cases}
$$

The flow matrix is :

$$
\begin{bmatrix}
1 & 3 & 3 & 2\\
1 & -(\sigma+1) & 0 & 0\\
1 & -\sigma & -(\sigma+1) & 0\\
1 & -\sigma & -\sigma & -(\sigma+1)
\end{bmatrix}
\times
\begin{bmatrix}
\alpha_0\\ \alpha_1\\ \alpha_4\\ \alpha_7
\end{bmatrix}
=
\begin{bmatrix}
1\\ 0\\ 0\\ 0
\end{bmatrix}
\qquad (3.148)
$$

The simulation result is shown in :

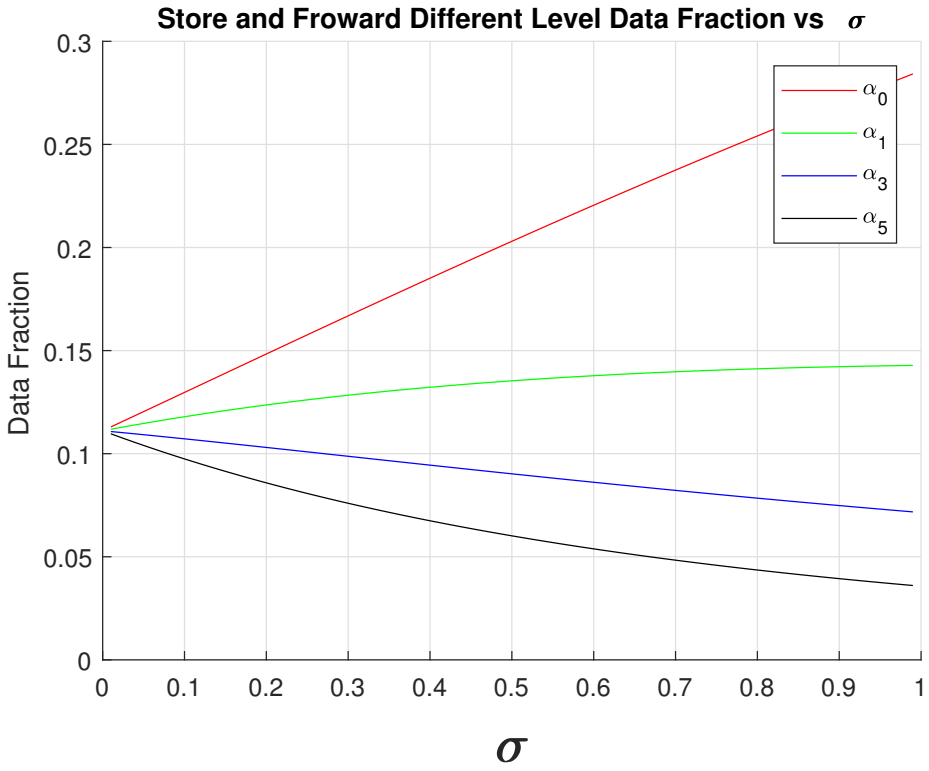

Figure 3.42: The fraction curve for 3*3 boundary data injection on P_0

A $3 * 3$ mesh network Fig. 3.10's data fraction curves Fig. 3.42 explains that as the value σ grows up, the fraction assigned to P_0 increases and the fractions distributed to $level_2$ and $level_3$ reduces. 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 equaling to the processor computation capacity, there is solely 4% data is deployed to the $level_3$. In addition, if the $\sigma > 1$, it means that the transmitting power is less than the processor's processing ability. In this scenario, keeping the data locally is more economical than transmitting it.

3.2.3 Data Injection on The Inner Grid Processor

The equations are:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (3.149) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.150) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.151) \\ \alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.152) \\ \alpha_4 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.153) \\ (\alpha_1 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.154) \\ (\alpha_2 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (3.155) \\ (\alpha_3 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (3.156) \\ (\alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (3.157) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (3.158) \\ 0 < \sigma < 1 & (3.159) \\ 0 < \alpha_0 \leq 1 & (3.160) \\ 0 \leq \quad \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 & (3.161) \end{cases}$$

The flow matrix closed-form is:

$$\begin{bmatrix} 1 & 4 & 4 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.162)$$

The simulation result shows:

A $3 * 3$ mesh network Fig. 3.13's data fraction curves Fig. 3.43 explains that as the value σ grows up, the fraction assigned to P_0 increases and the fractions distributed to $level_2$ reduces. In other words, if the communication

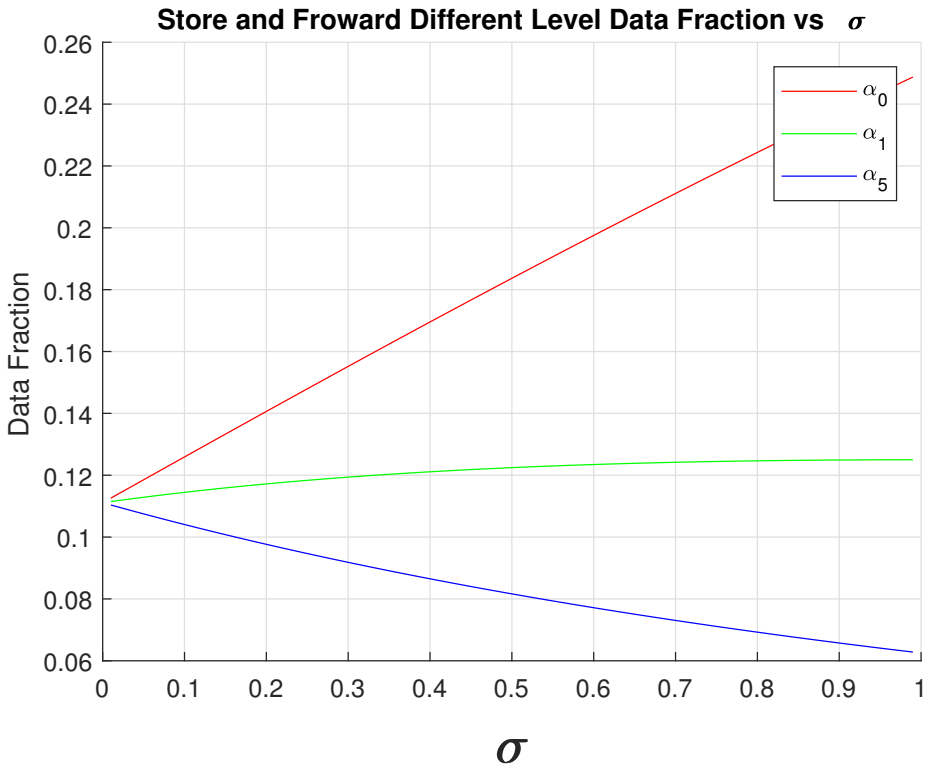

Figure 3.43: The timing diagram for 3*3 inner grid injection P_0

capability decreases, there are more data processed locally, which is reasonable.

If the ability of the link degrades, asymptotically equaling to the processor computation capacity, there is solely 6% data is deployed to the $level_2$.

3.2.4 Sensitivity Analysis Store and Forward Scenario

Data Injection on The Corner Processor

The simulation result of sensitivity analysis of $2*n$ mesh network Fig. 3.7 is as follows:

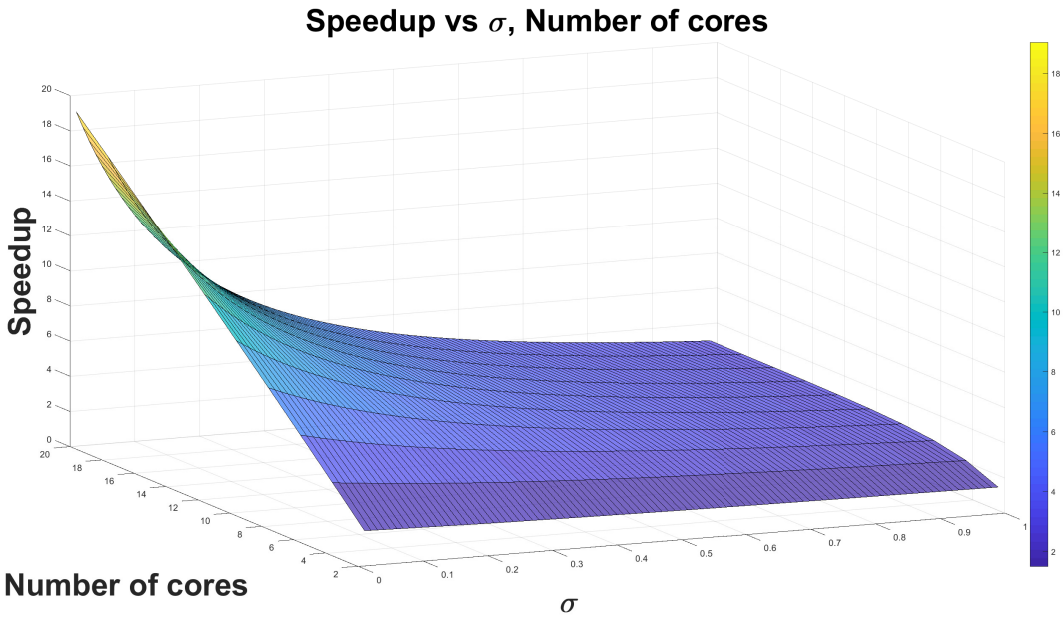

Figure 3.44: Sensitivity analysis result of 2*10 mesh network result

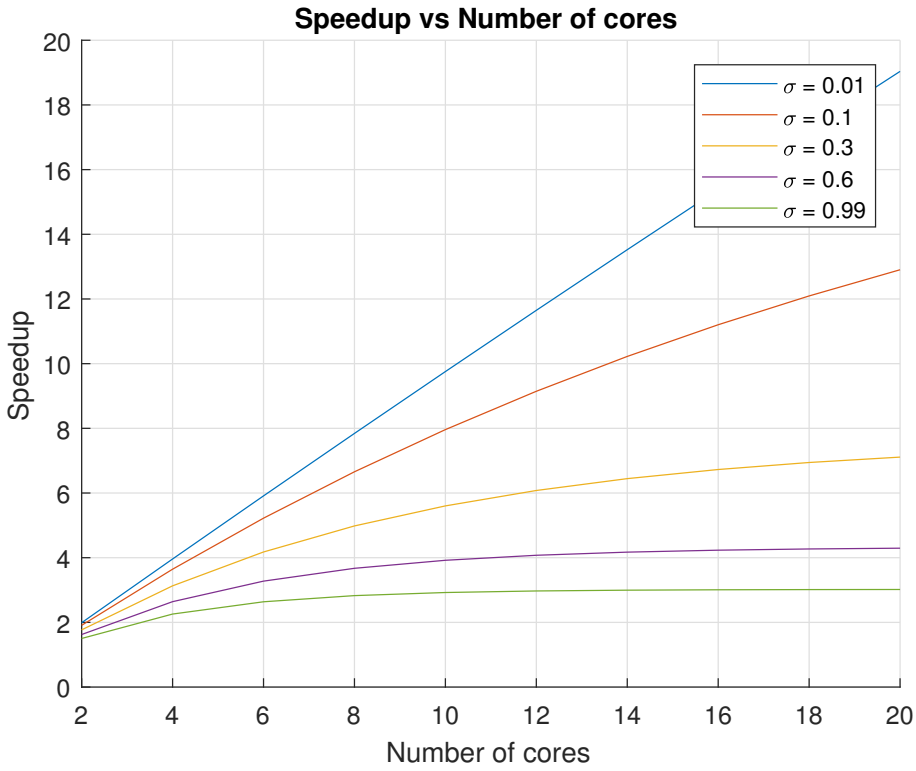

Figure 3.45: Speedup curve of 2*10 mesh network result

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 factor core's number dominates the efficiency. If the $\sigma > 0.3$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation.

Data Injection on The Boundary Processor

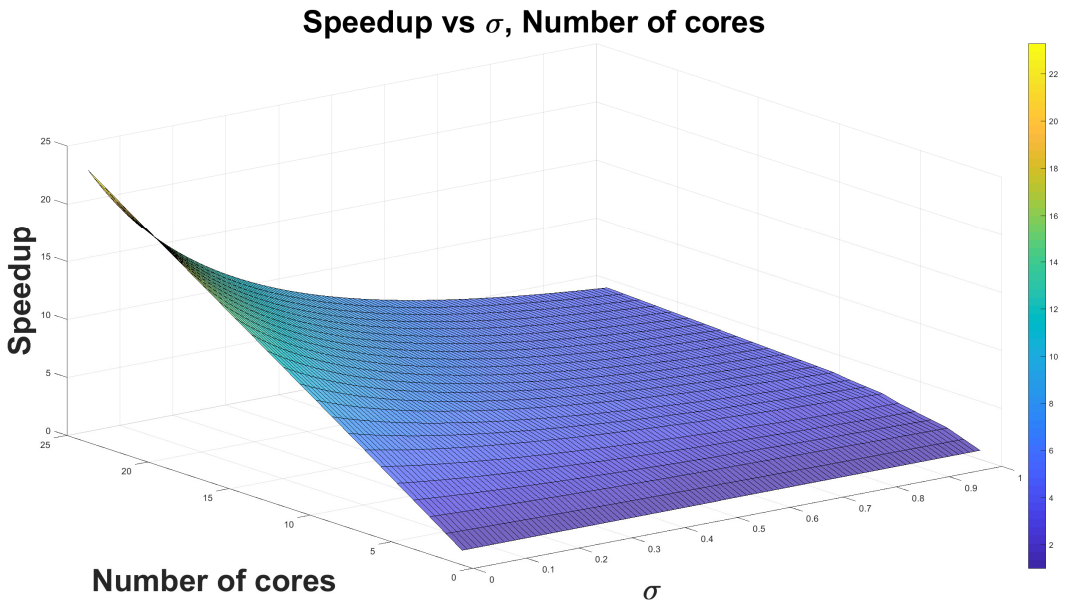

Figure 3.46: Sensitivity analysis result of 3*8 mesh network result

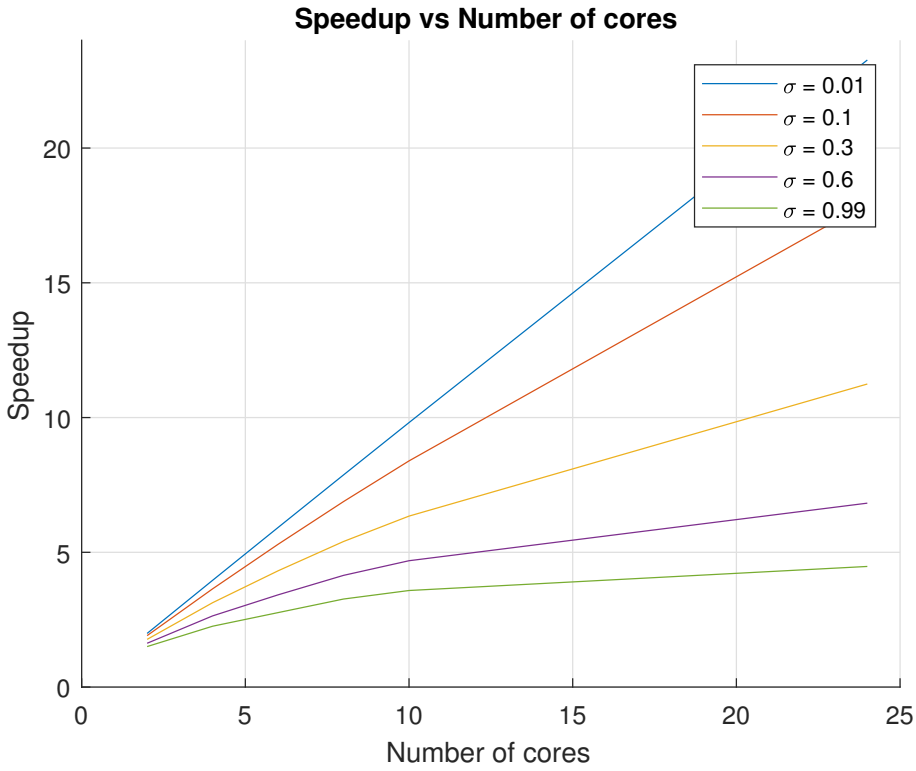

Figure 3.47: Speedup curve of 3*8 mesh network result

Sensitivity analysis result Fig. 3.46 and speedup curve Fig. 3.47 tell the speedup efficiency for the data injection on the boundary scenario.

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 factor core's number dominates the efficiency. If the $\sigma > 0.3$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation.

Data Injection on The Inner Grid Processor

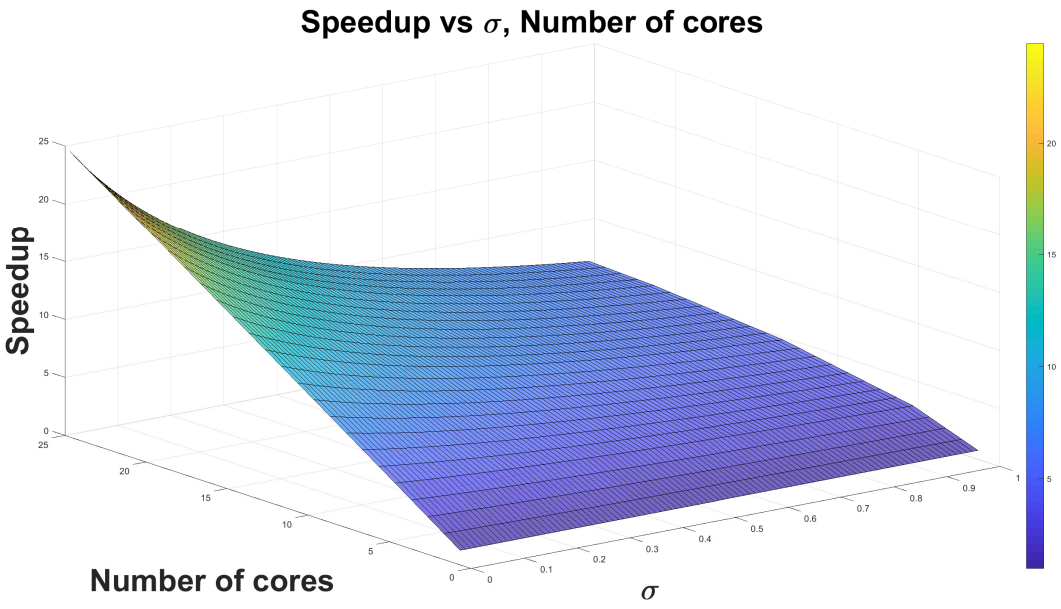

Figure 3.48: Sensitivity analysis result of data injection position on inner grid processor

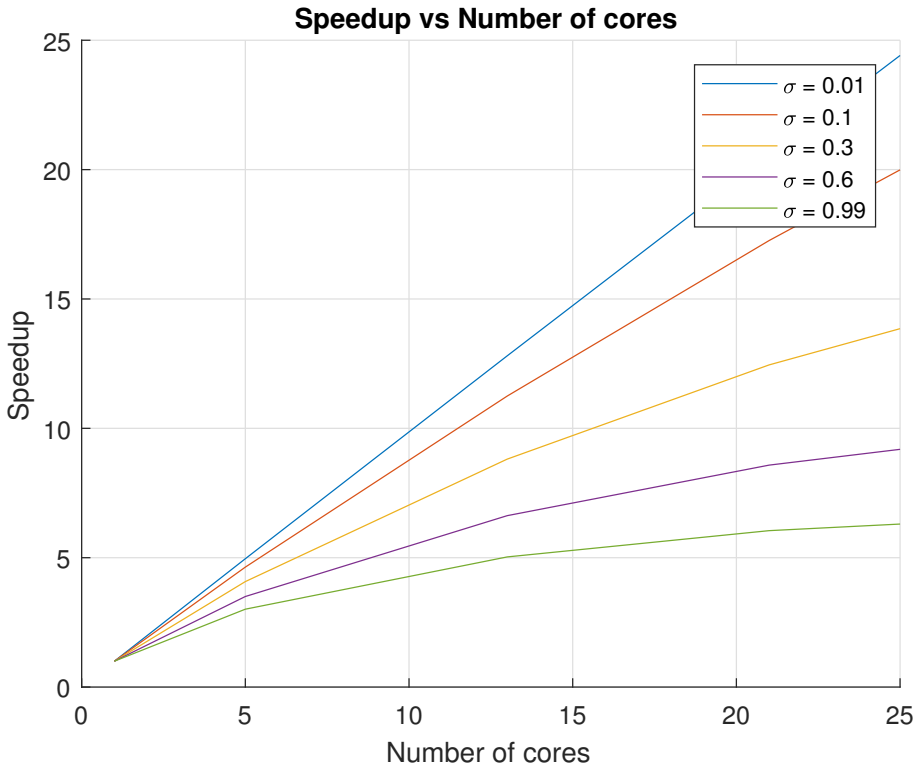

Figure 3.49: Speedup curve of 5*5 mesh network result

Sensitivity analysis Fig. 3.48 and speedup curves Fig. 3.49 display the simulation result for the data injection position P_{12}. If $\sigma < 0.1$, the speedup linear grows up and the best speedup is 24.

3.2.5 Multi-source Uniform Data Fraction

We utilize the similar methodology and algorithms in virtual cut-through situation to deal with these problems. The critical point is that the flow matrix formula is different, which is replaced by the store and forward flow matrix formula.

Situation I

We utilize the Equivalence Processor Scheduling Algorithm to address this problem.

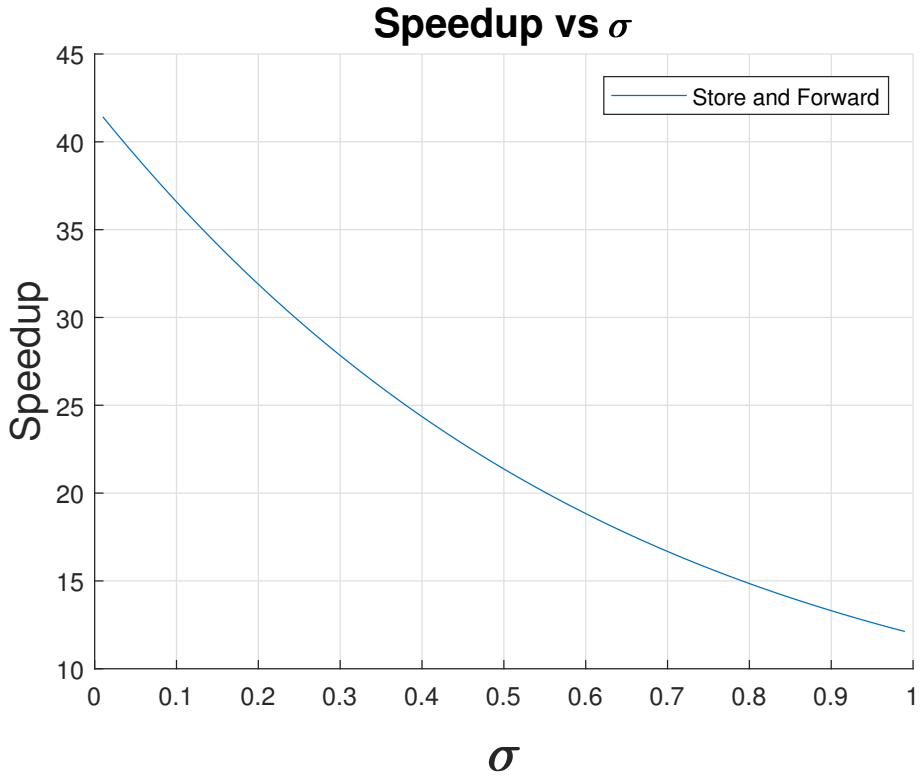

Figure 3.50: Connected subgraph Fig. 3.22 speedup vs σ

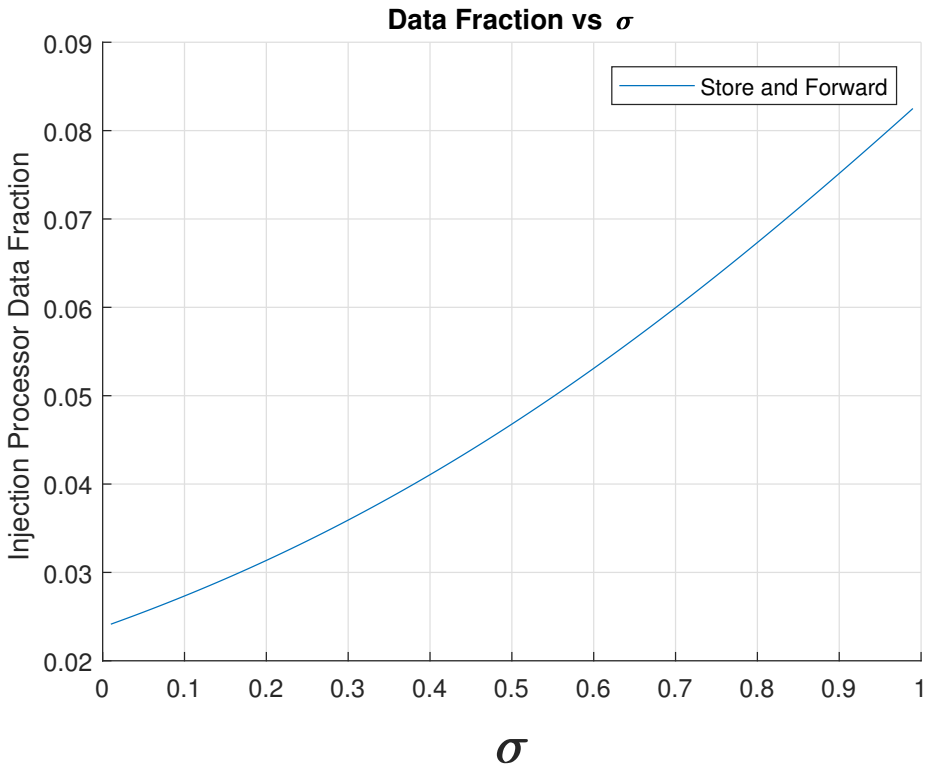

Figure 3.51: Connected subgraph Fig. 3.22's data injection processor workload fraction vs σ

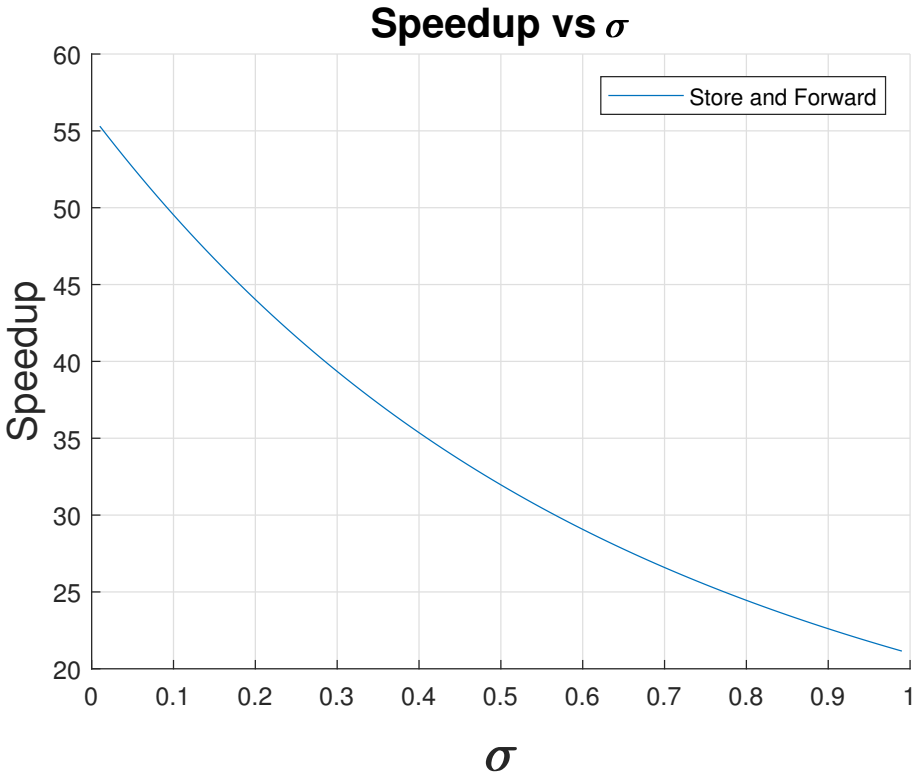

Figure 3.52: Connected subgraph Fig. 3.23 speedup vs σ

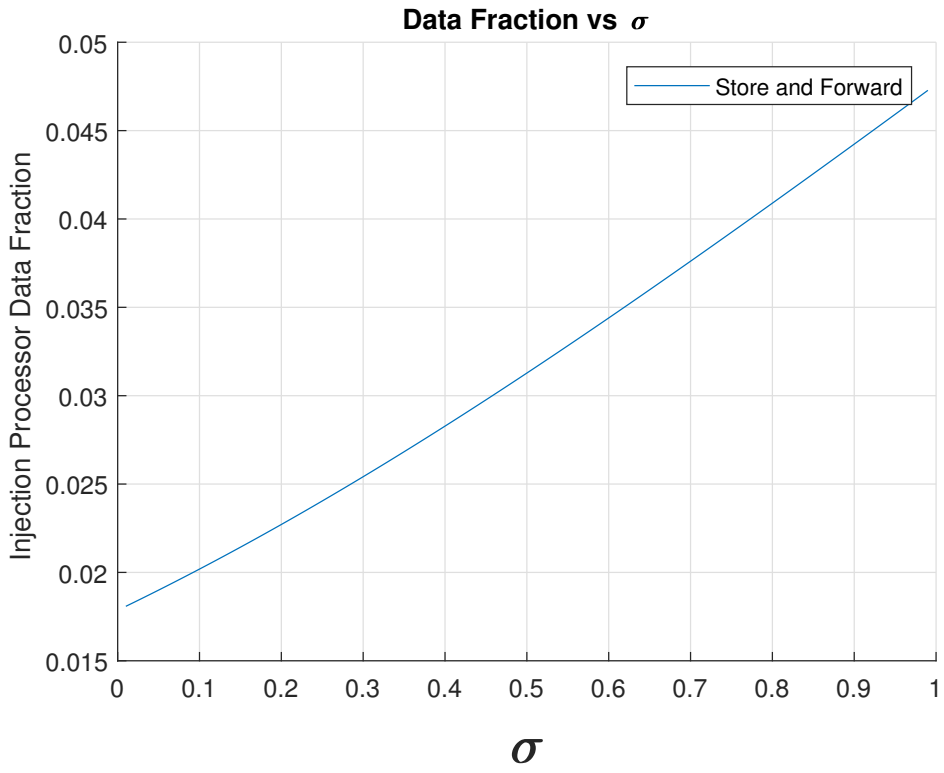

Figure 3.53: Connected subgraph Fig. 3.23's data injection processor workload fraction vs σ

Situation II

We utilize the Reduced Voronoi Diagram Algorithm to address this problem. The simulation result is as follows :

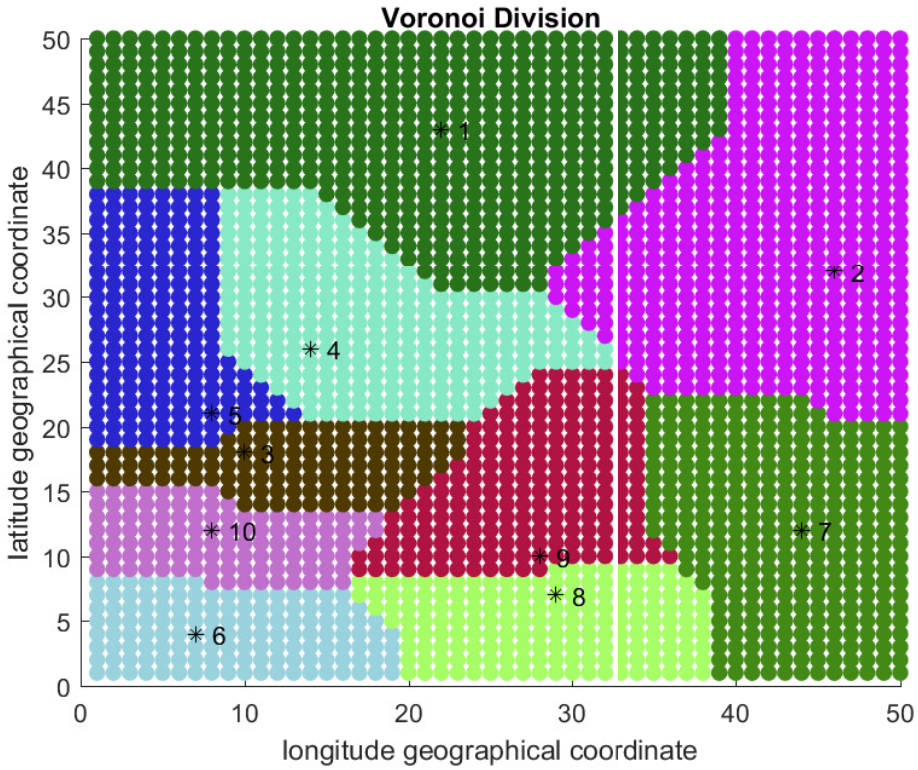

Figure 3.54: 10 Manhattan distance Voronoi Cells

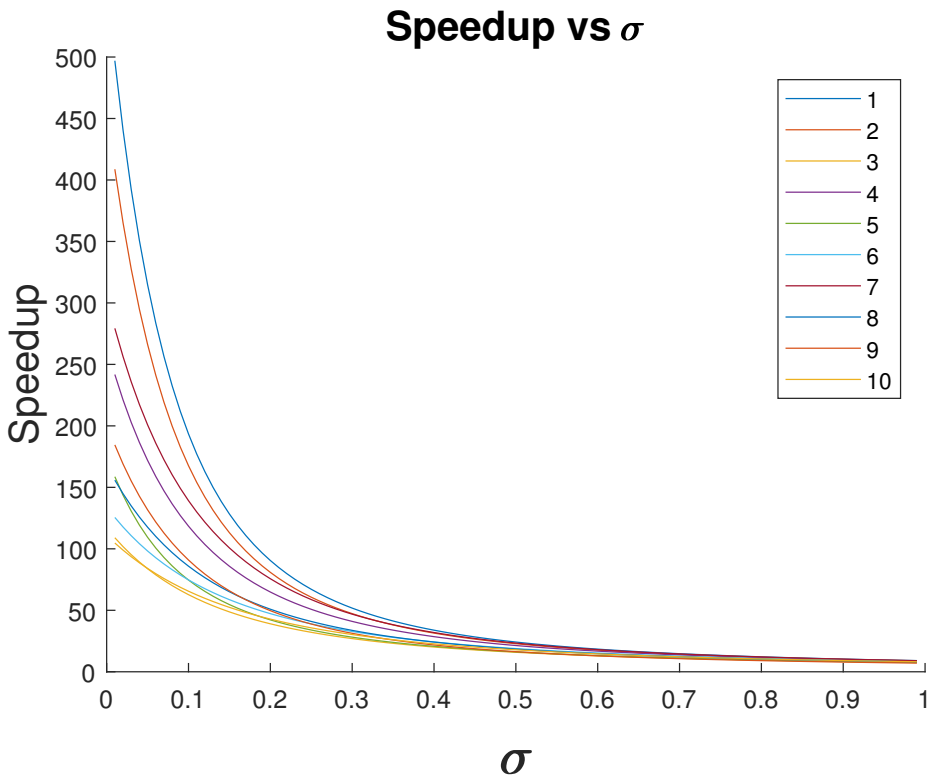

Figure 3.55: 10 Manhattan distance Voronoi cells store and forward switching speedup curves

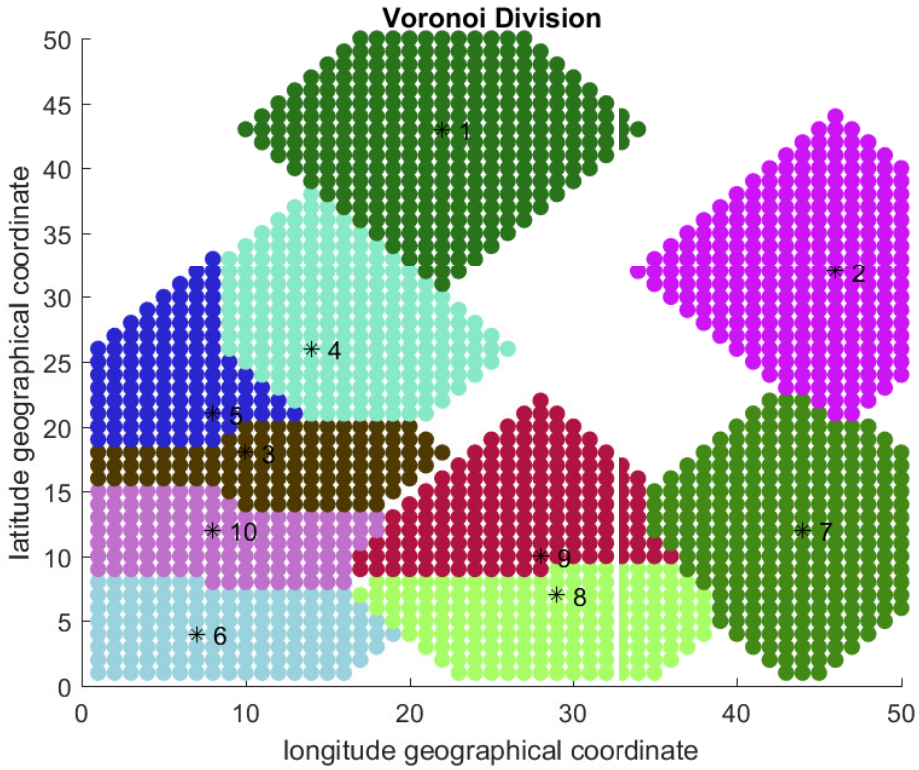

Figure 3.56: 10 reduced Manhattan distance store and forward switching Voronoi cells

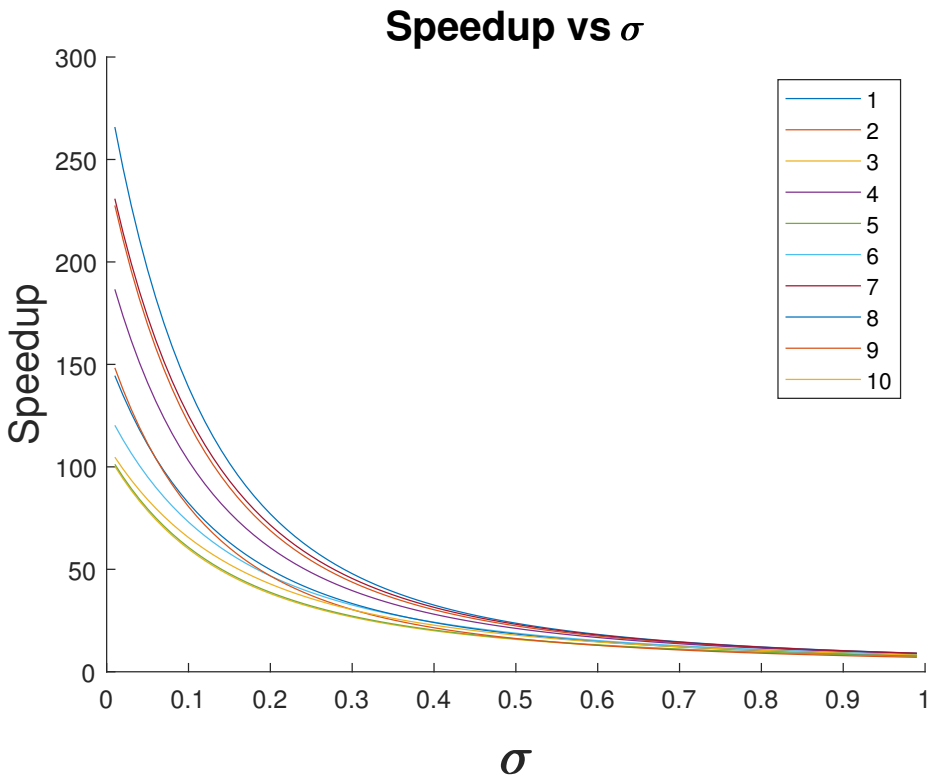

Figure 3.57: 10 reduced Voronoi cells' store and forward switching speedup curves

We save about 37% processors and finish the task within the same time.

Situation III

We utilize the Reduced Constrained Voronoi Diagram Algorithm to address this problem.

3.2.6 Multi-source Non-uniform Data Injection

We utilize the similar methodology and algorithms in virtual cut-through situation to deal with these problem.

Chapter 4

Toroidal Rectangle Network

4.1 Toroidal Rectangle Network Notions

The toroidal network is 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. According to the paper [16], there are three different torus mesh network. In this paper, our intent is not to propose one model to "fit all" problems but rather to indicate one normal case.
Considering the toroidal rectangle network Fig. 4.1 and Fig. 4.2.

- m : There are m processors on the longitude.
- n : There are n processors on the latitude.
- L : The load injection.
- L_x : The L's longitude coordinate.
- L_y : The L's latitude coordinate.
- D_k : P_k's shortest Manhattan distance to L.
- $D_{k,x}$: P_k's longitude shortest Manhattan distance to L.

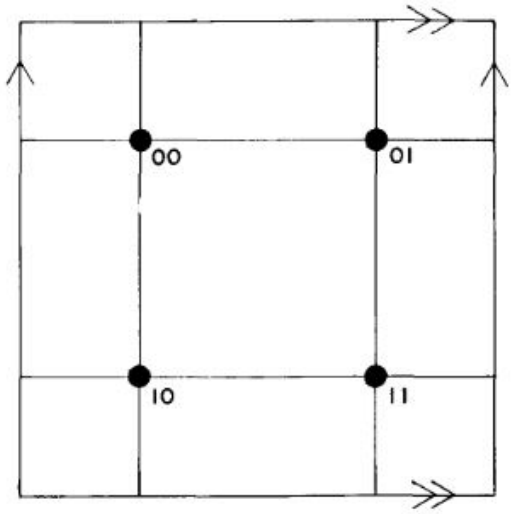

Figure 4.1: The rectangular toroidal network

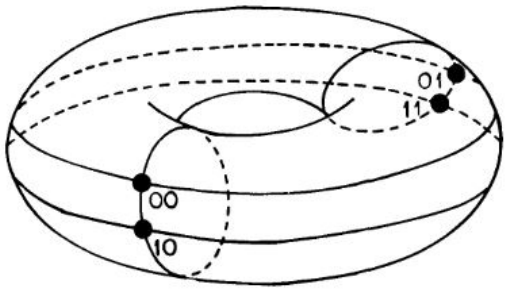

Figure 4.2: The rectangular toroidal network

- $D_{k,y}$: P_k's altitude shortest Manhattan distance to L.

$$\begin{cases} D_k = D_{k,x} + D_{k,y} & (4.1) \\ D_{k,x} = \min\{\|D_{k,x} - L_x\|, m - \|D_{k,x} - L_x\|\} & (4.2) \\ D_{k,y} = \min\{\|D_{k,y} - L_y\|, n - \|D_{k,y} - L_y\|\} & (4.3) \end{cases}$$

4.2 Virtual Cut-through Switching Scenario

4.2.1 Data Injection On The Grid Processor

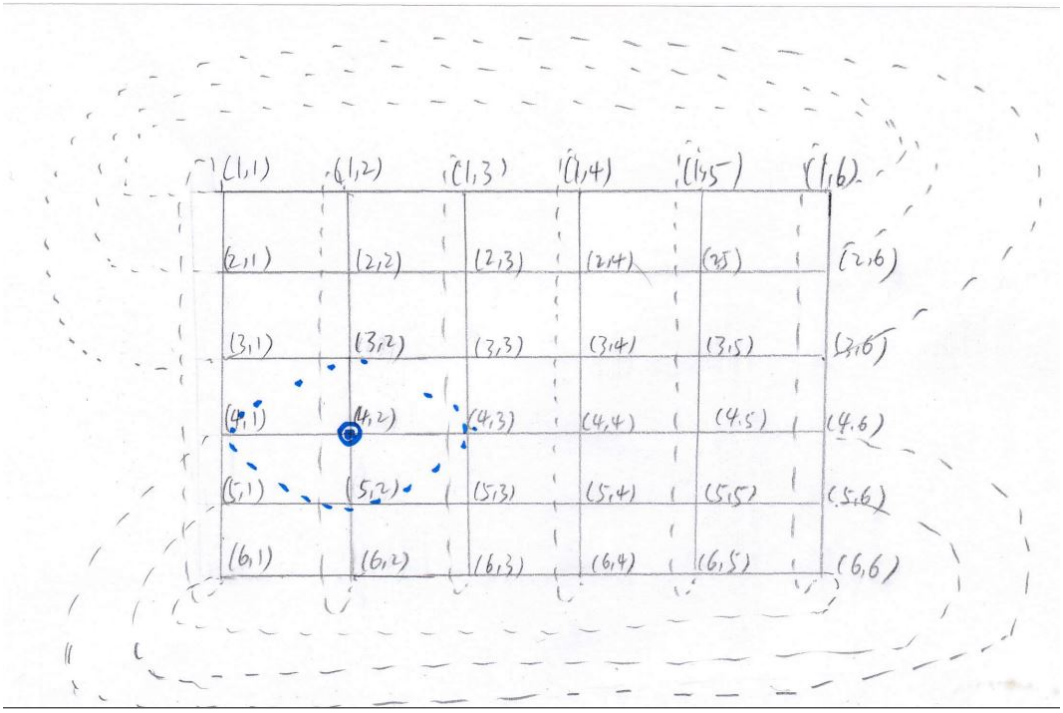

Figure 4.3: The m*n toroidal rectangle network and the data injection is $P_{4,2}$

In $m * n(m = 6, n = 6)$ Fig. 4.3 toroidal rectangle network, L happens on grid position $(4, 2)$. We calculate the $D_{k,i}$ table Table 4.1 by breadth first search(***BFS***) algorithm.

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

Table 4.1: D_i vs Number

The flow matrix closed-form is :

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.4}$$

The simulation result is :

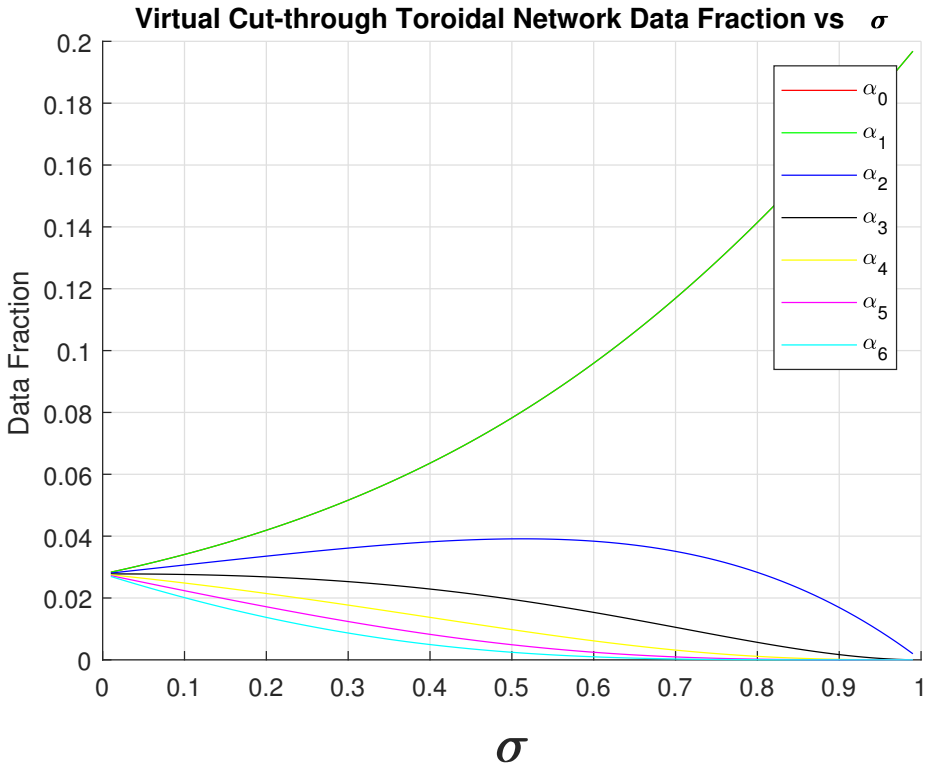

Figure 4.4: Data fraction curve of toroidal rectangle network Fig. 4.3

From data fraction curves Fig. 4.4, we see that as the value σ grows, more and more workload is assigned to the $P_{4,2}$ and its one hop neighbors. That is, as the communication ability decades, the economical method is to locally process the job.

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

Table 4.2: D_i vs Number of processor on each D_i

4.2.2 Sensitivity Analysis of Toroidal Rectangle Network

Considering a $5 * 5$ toroidal rectangle network, the $level_i$ table shows Table 4.2.

So the simulation result illustrates in Fig. 4.5

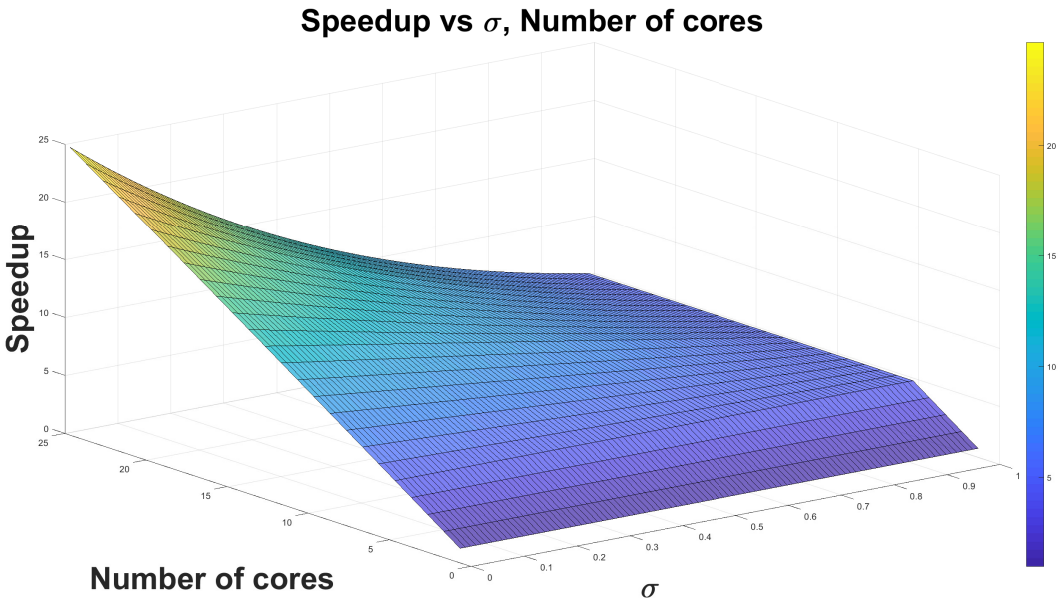

Figure 4.5: Sensitivity analysis of 5*5 torus network

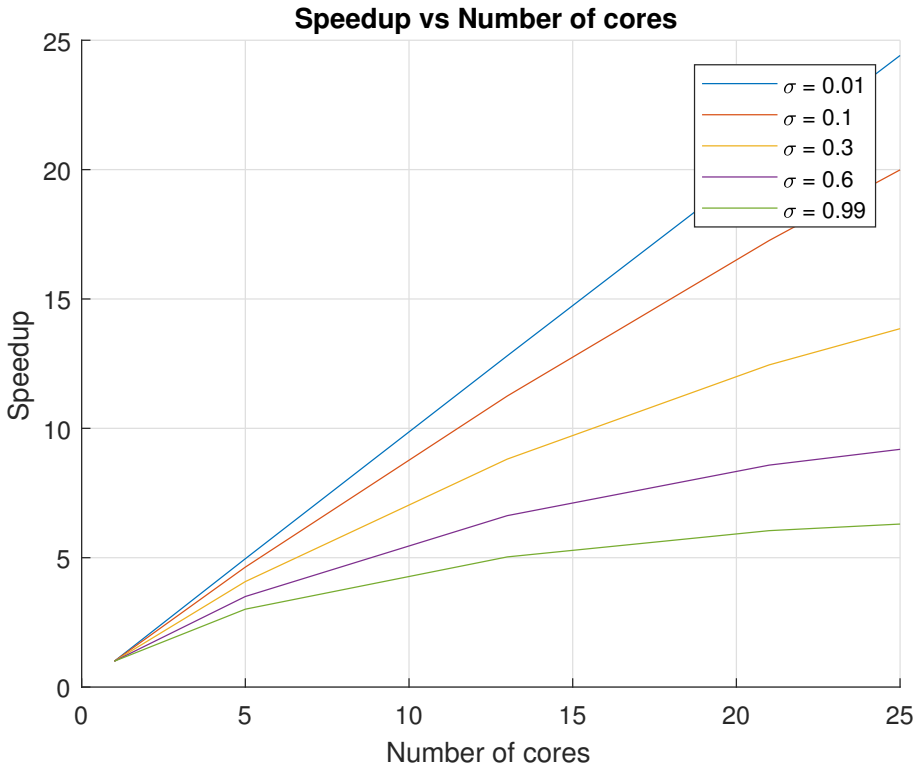

Figure 4.6: Data fraction curves of a 5*5 torus network on different σ

Fig. 4.6 illustrates that as the value σ decreases, the speedup efficiency increases. The best performance happens on the $\sigma \approx 1$ and the efficiency speedup is almost 25. The bottom speedup situation is about 5.

4.2.3 Multi-source Uniform Data Fraction

We extend the single source assignment to multi-source assignment problem [10] [4]. According to each processor, we concentrate on the processors' geographical location P_i, data fraction assigned α_i.

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

From the data injection position relationship we consider three different situations :

- Data injection positions consist of a subgraph G_L of G.
- Data injection processor doesn't connect with each other.
- Data injection positions consist of some connected subgraphs and individual injection.

Situation I

We choose the Equivalence Processor Scheduling Algorithm to address the problem.

Torus connected subgraph data injection Fig. 4.7 Fig. 4.9 show a connected subgraph data injection situation.

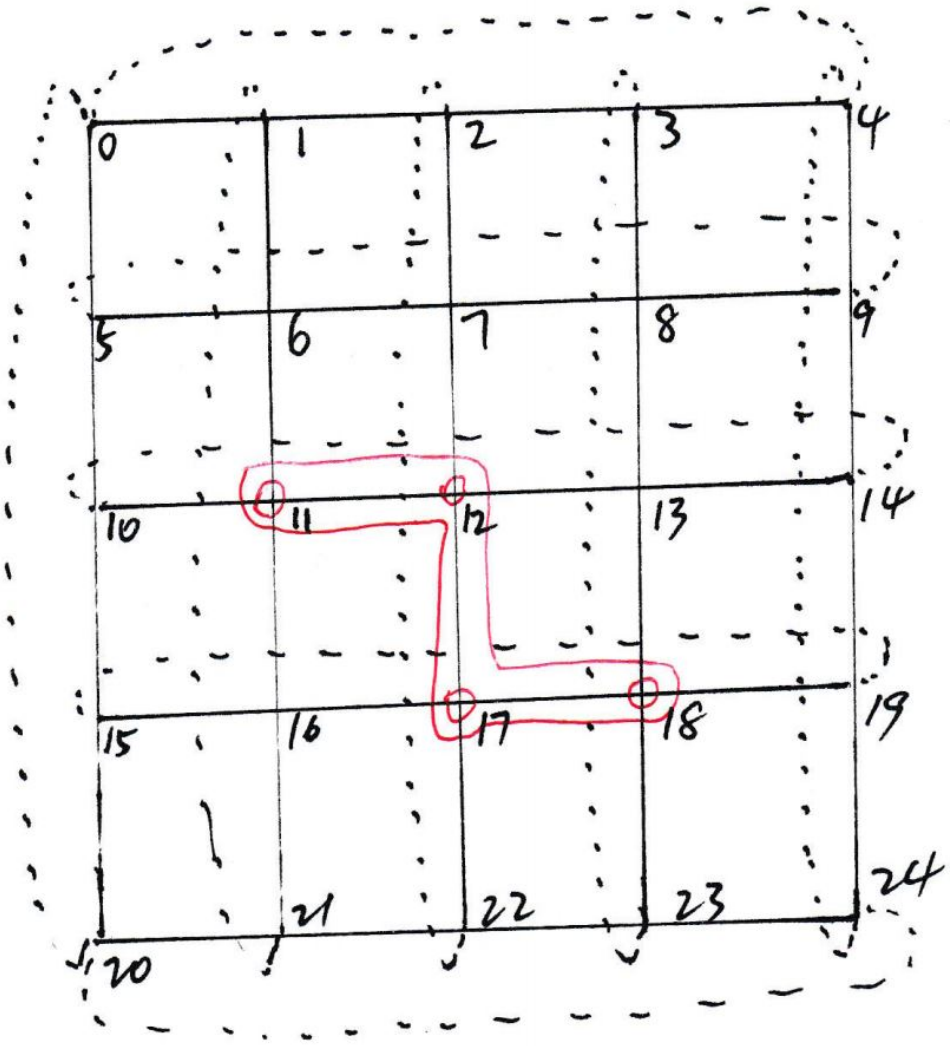

Figure 4.7: Toroidal mesh network subgraph data injection

A connected subgraph Fig. 4.7's flow matrix equation is

$$\begin{bmatrix} 4 & 8 & 9 & 4 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.5}$$

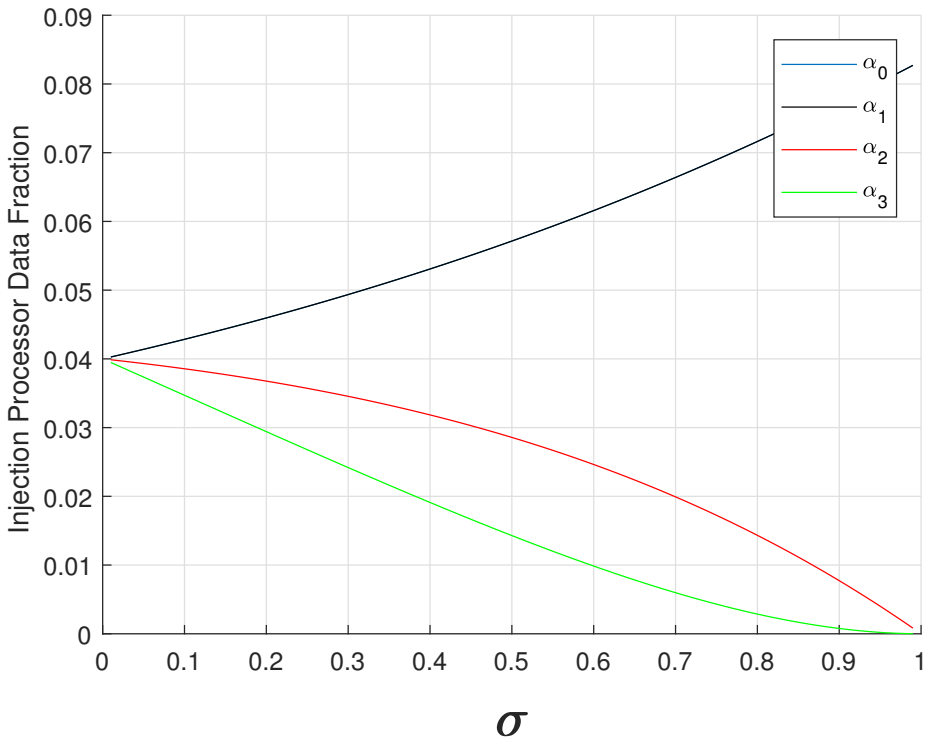

Figure 4.8: Data fraction curves of connected subgraph Fig. 4.7

Fig. 4.9 shows another connected subgraph data injection example.

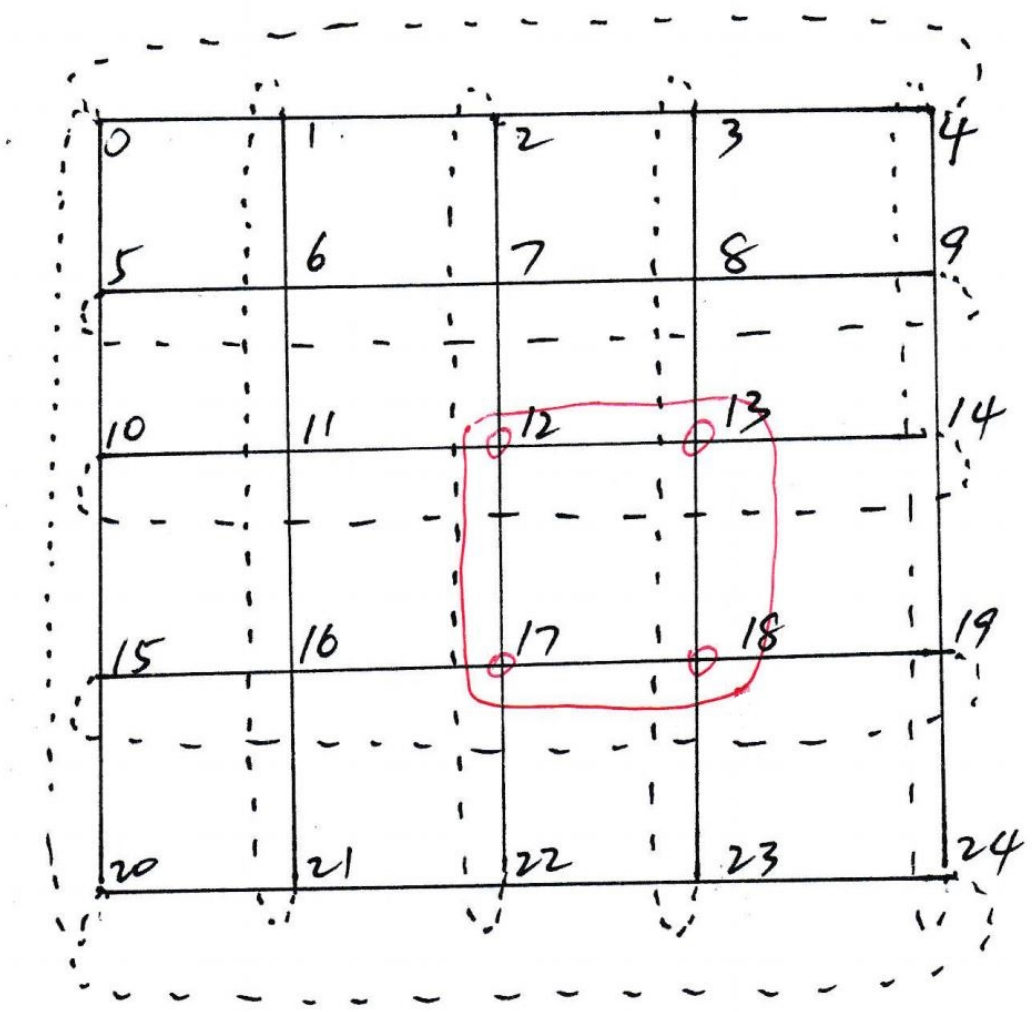

Figure 4.9: Toroidal mesh network subgraph data injection

Fig. 4.9's flow matrix equation is :

$$\begin{bmatrix} 4 & 8 & 9 & 4 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.6}$$

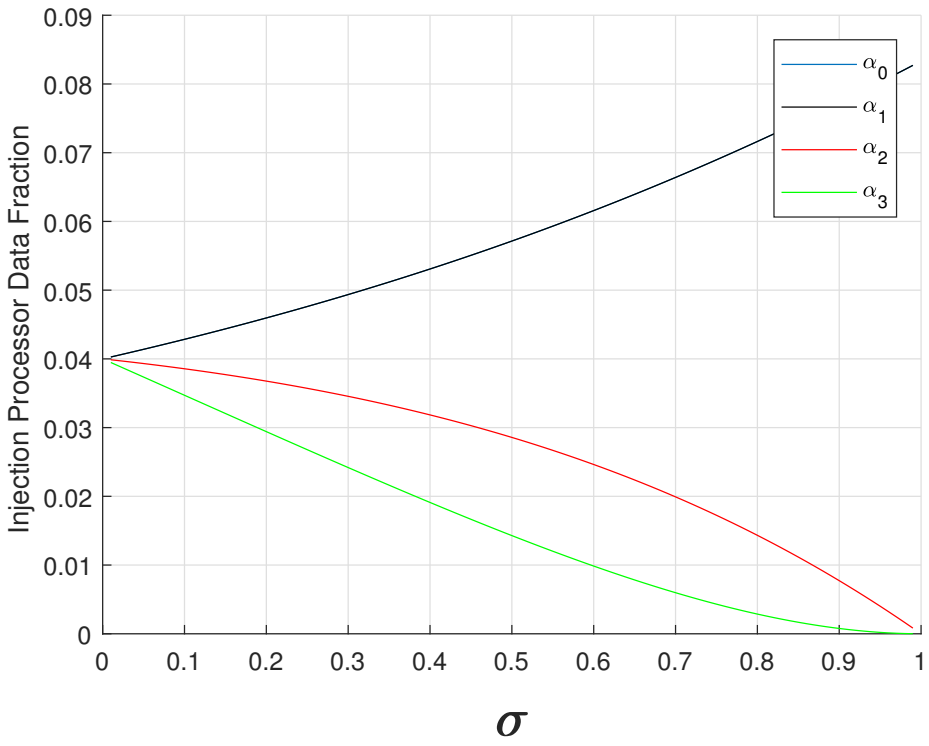

Figure 4.10: Data fraction on different type processors for Fig. 4.9

Situation II

We choose the Reduced Voronoi Diagram Algorithm to address the problem.

Fig. 4.11 [17] provides a torus Voronoi method, which extends the original domain to 8 copy and calculate the Voronoi Diagram as planner algorithm [18]. Then the corner part is the torus Voronoi diagram.

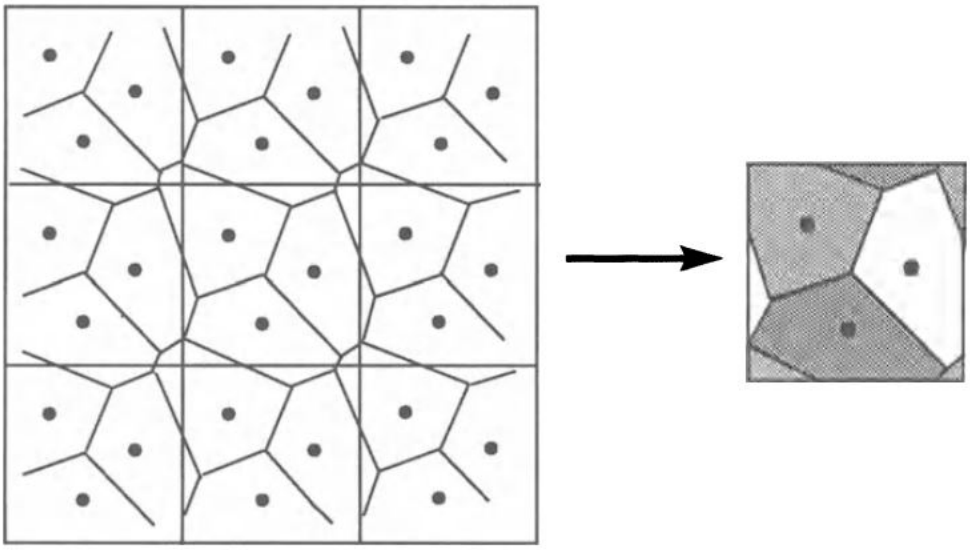

Figure 4.11: How to calculate torus Voronoi Diagram

Fig. 4.12 says the initial Voronoi division of a $50 * 50$ mesh network. The base is the data injection and the processor Voronoi cell tackle base's workload together. The user case contains 10 data injections and each cell's speedup curve shows in Fig. 4.15.

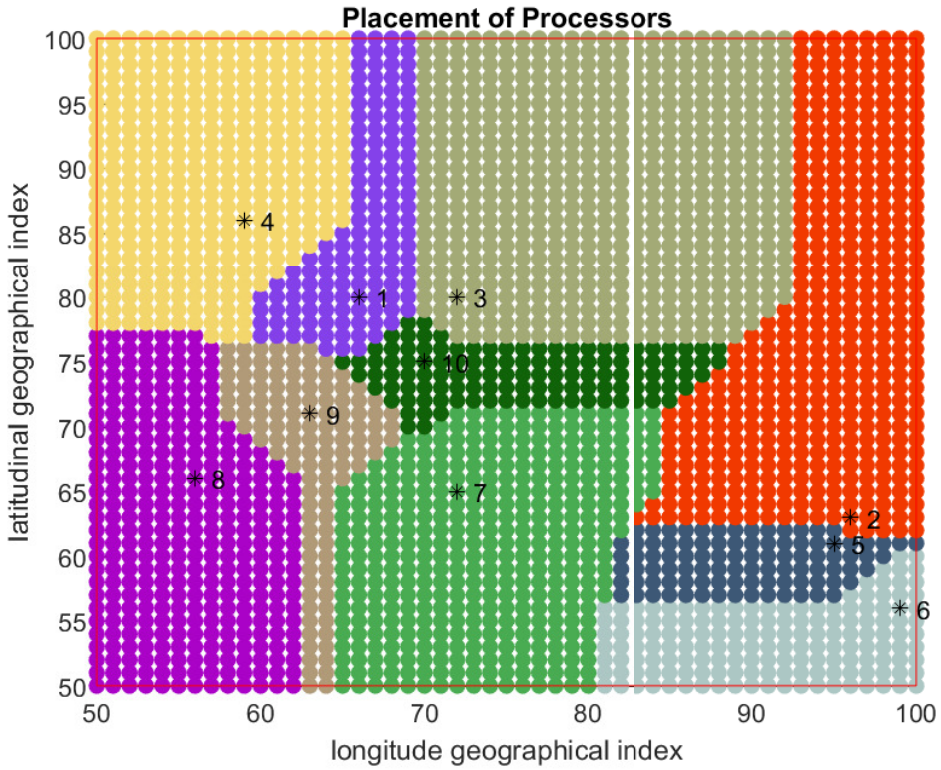

Figure 4.12: Initial Manhattan distance Voronoi Digram

Torus Manhattan distance Voronoi diagram Fig. 4.13 shows a toroidal Voronoi division result. The same color parts connect with each other and consist of a whole division in toroidal rectangle network. For example, the four gray corners consist of a whole area and the injection processor index is 6.

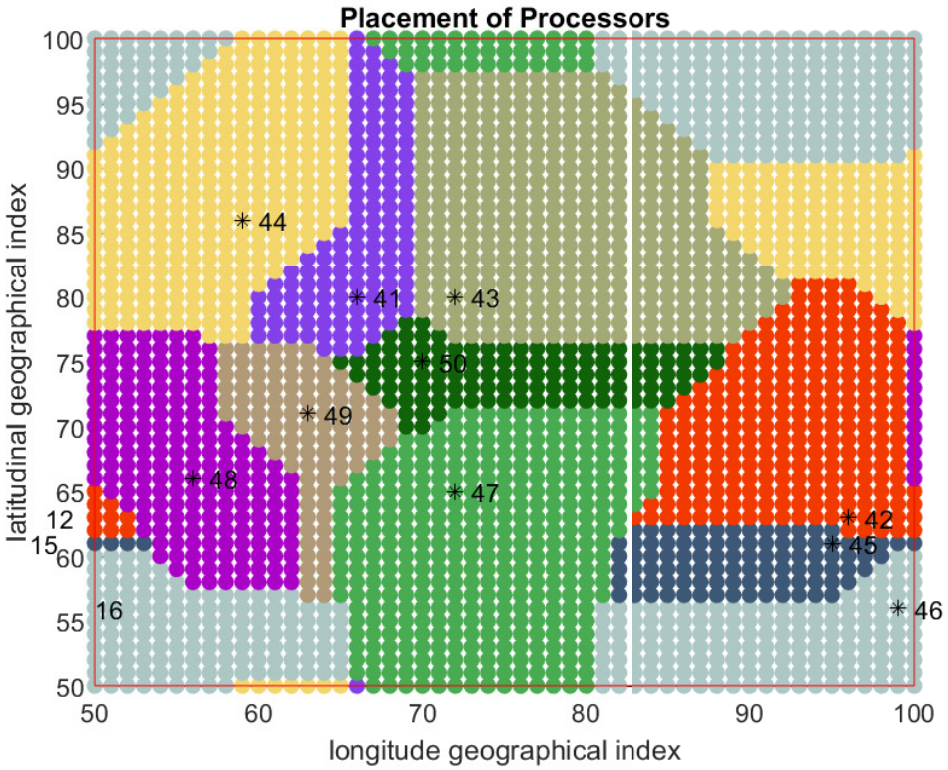

Figure 4.13: Torus Manhattan distance Voronoi diagram

Fig. 4.14 shows the Fig. 4.13 casts to the toroidal rectangle network result.

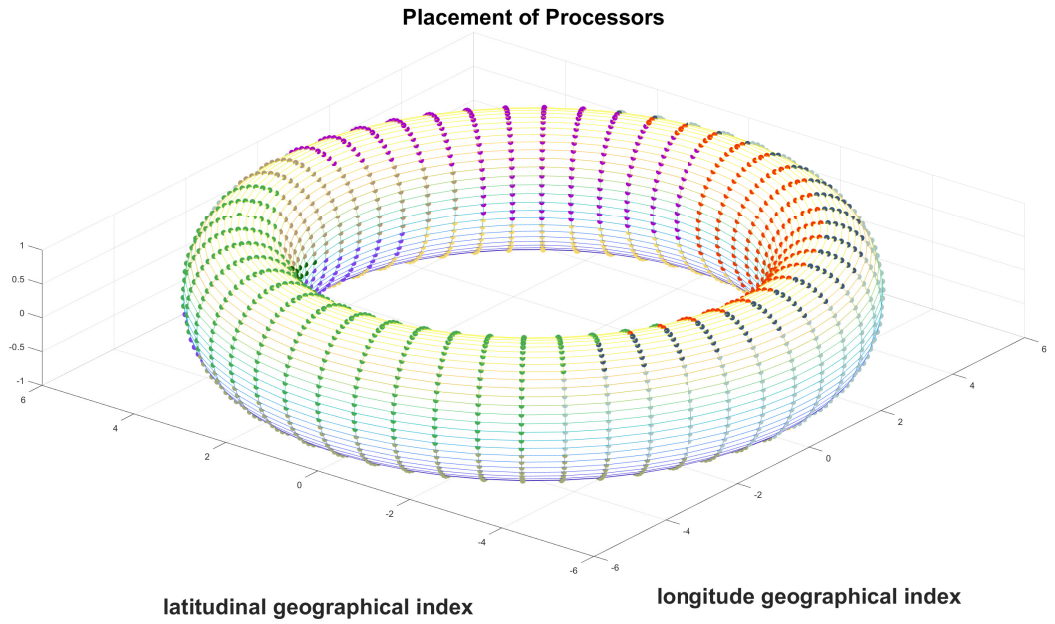

Figure 4.14: Voronoi diagram casting to the torus

In the reduced model Fig. 4.17 and Fig. 4.18, reduced toroidal Voronoi diagram save 27% processors hit the same processing capacity.

The ratio speedup is about $\frac{speedup_{max}}{speedup_{min}} = \frac{490}{98} = 5$. After the reduced stage, the ratio is $\frac{290}{98} \approx 2.96$. That is the reduced heuristic algorithm obtaining more balanced computation capacity distribution.

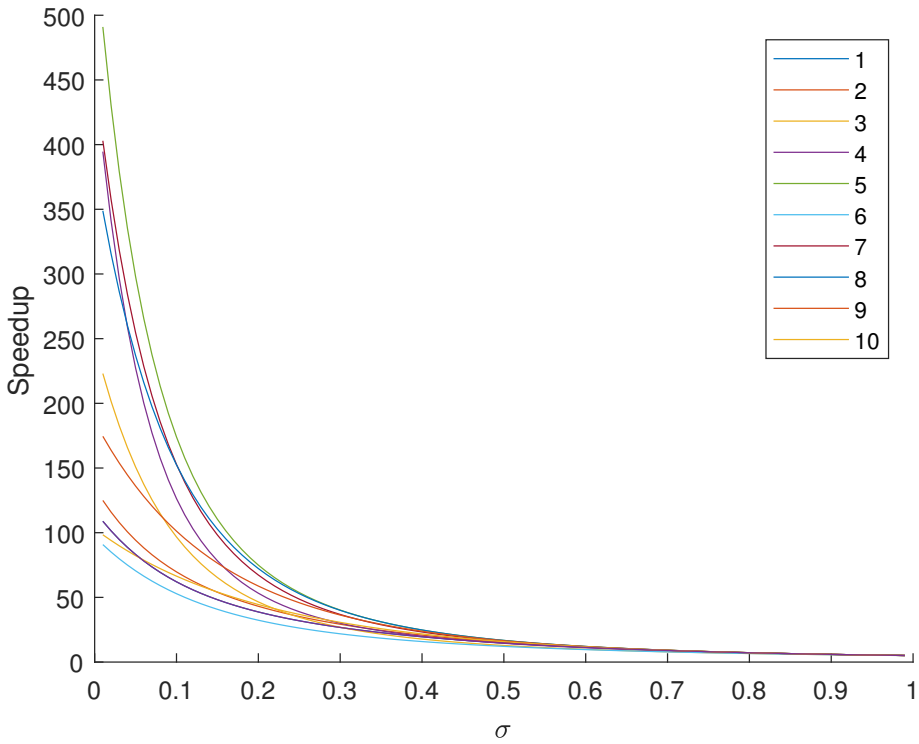

Figure 4.15: Toroidal Voronoi cells' speedup curves

Situation III

We utilize the Reduced Constrained Voronoi Diagram Algorithm to address this kind of question.

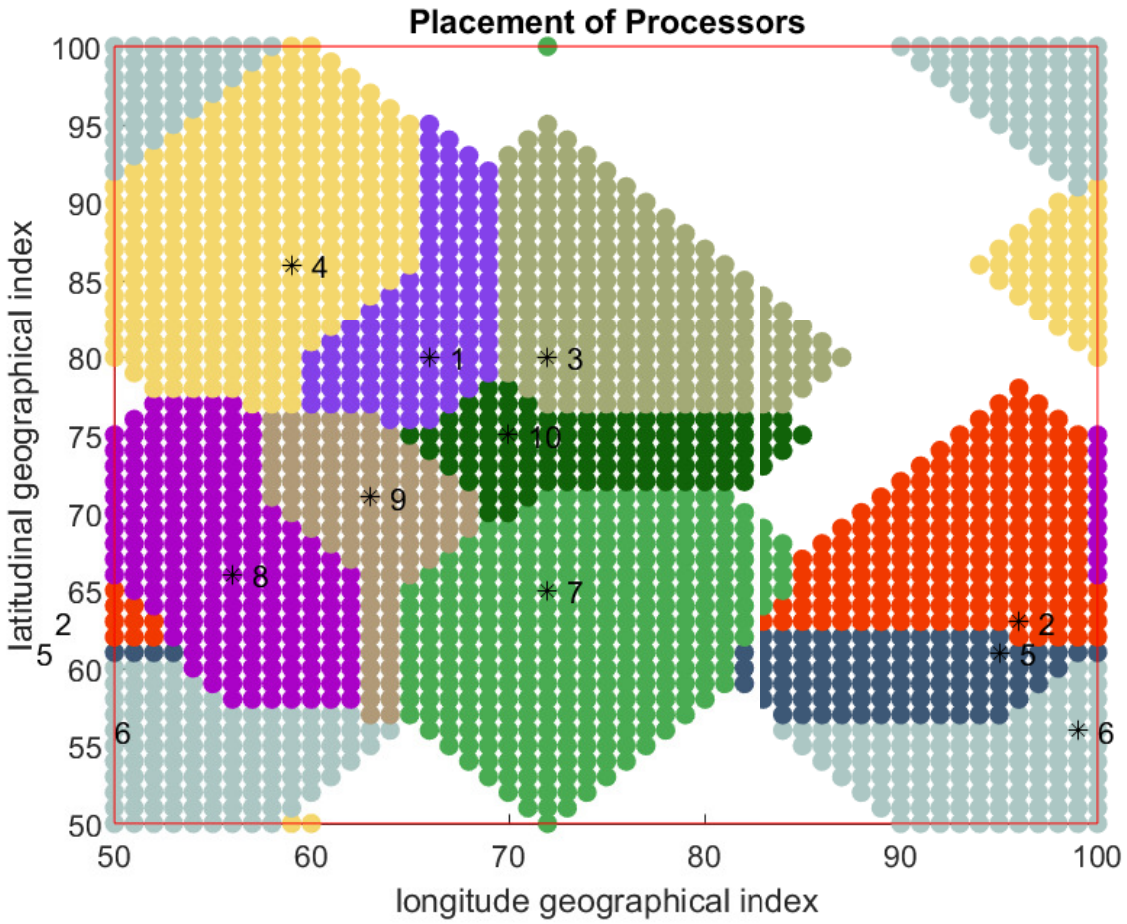

Figure 4.16: Torus reduced Voronoi diagram

Save about $436/2500 \approx 17\%$ processors.

4.2.4 Multi-source Non-uniform Data Injection

We utilize the same methodology and algorithm in the mesh network.

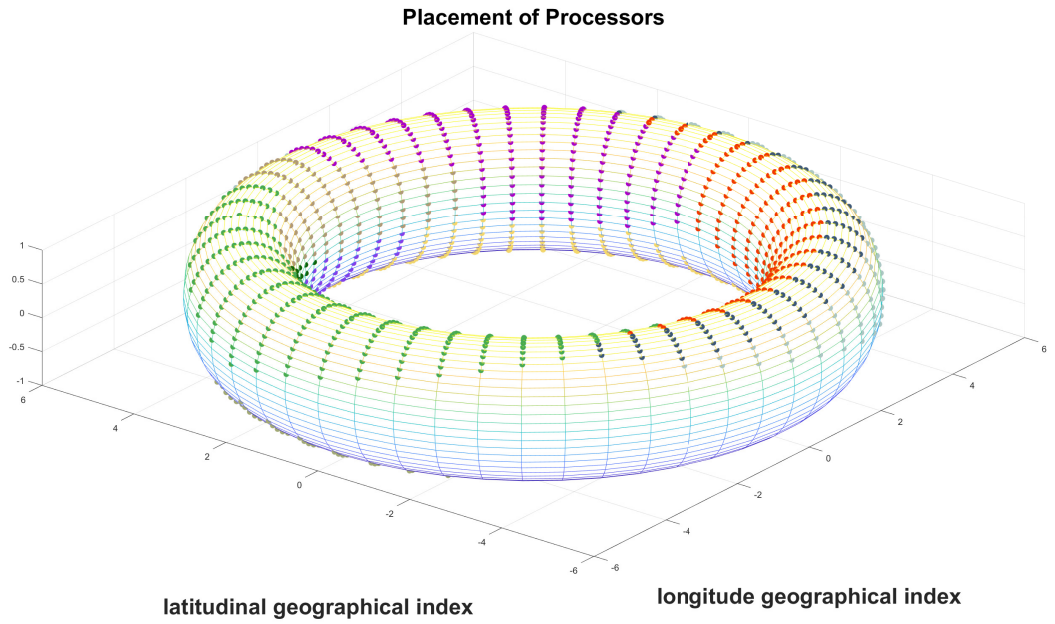

Figure 4.17: Torus reduced Voronoi diagram casting to a toroidal rectangle network

4.3 Store and Forward Switching Scenario

4.3.1 Data Injection On The Grid Processor

We utilize the $\sigma^\star$ to present $-(\sigma+1)$. The flow matrix closed-form of Fig. 4.3 is:

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.7}$$

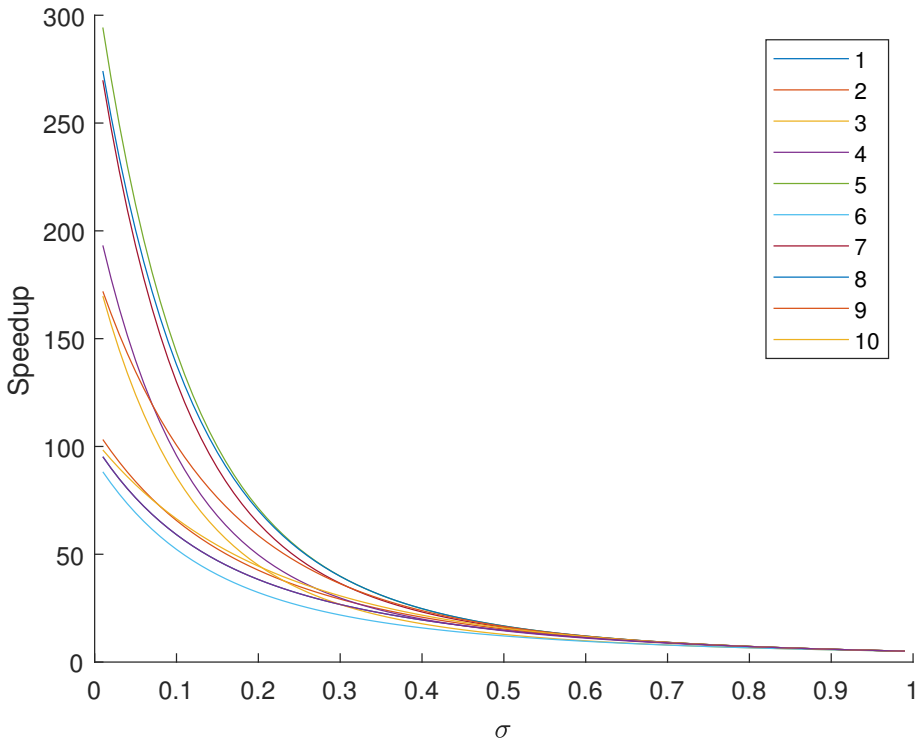

Figure 4.18: Torus reduced Voronoi diagram speedup curves

The fraction curve result is :

From Fig. 4.24, we see that as the value σ grows, more and more workload is assigned to the $P_{4,2}$ and its one hop neighbors. That is, as the communication ability decades, the economical method is to locally process the job.

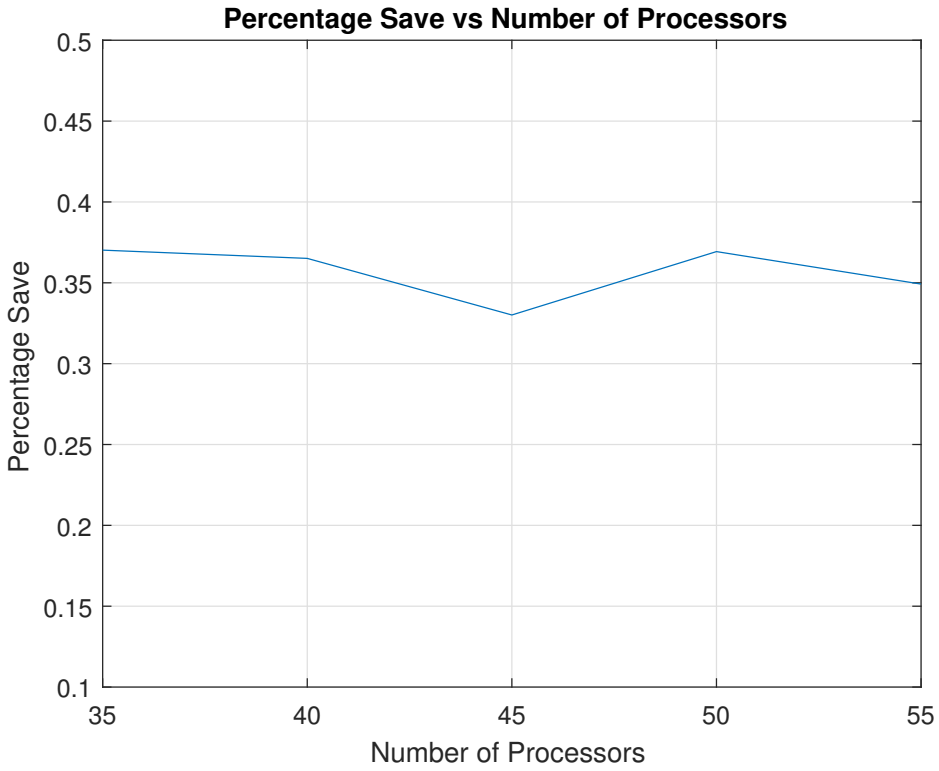

Figure 4.19: Torus reduced Voronoi diagram average saved processors

4.3.2 Sensitivity Analysis of Toroidal Rectangle Network

Considering a $5 * 5$ toroidal rectangle network, the $level_i$ table shows Table 4.3:

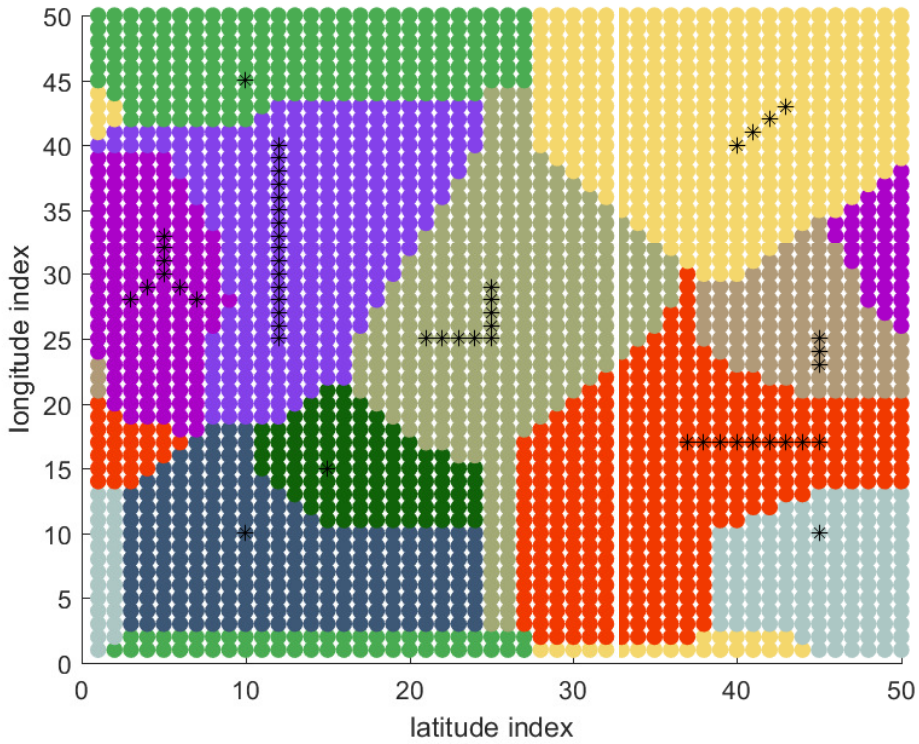

Figure 4.20: Subgraph and individual data injection on torus network

So the simulation result illustrates in Fig. 4.25

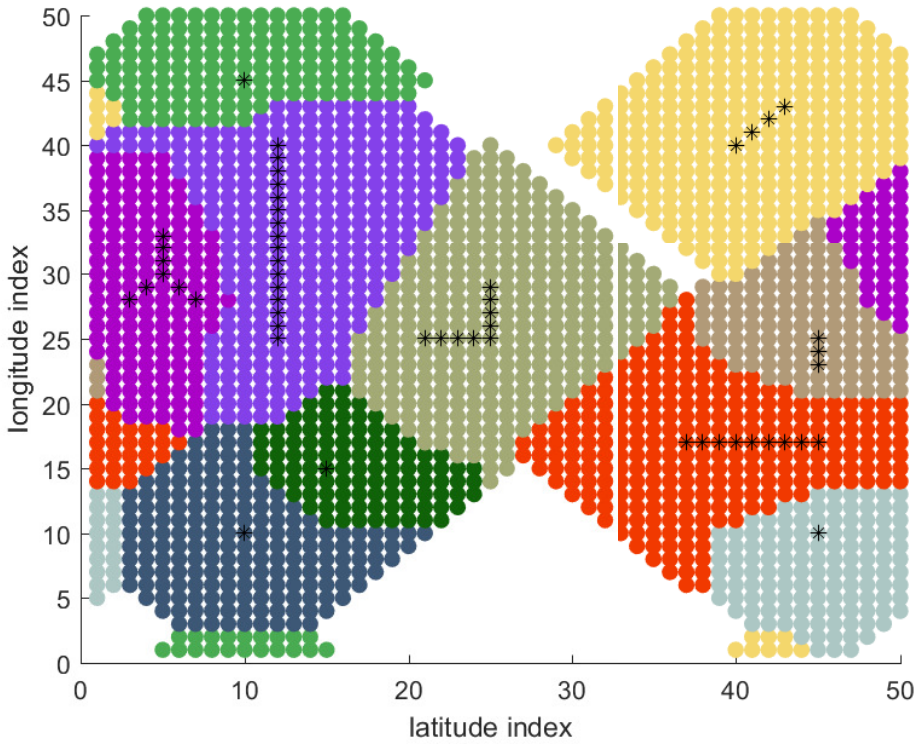

Figure 4.21: Subgraph and individual data injection on torus network

4.3.3 Multi-source Uniform Data Injection

We choose the same algorithm in virtual cut-through scenario to deal with this problem.

4.3.4 Multi-source Non-uniform Data Injection

We choose the same algorithm in virtual cut-through scenario to deal with this problem.

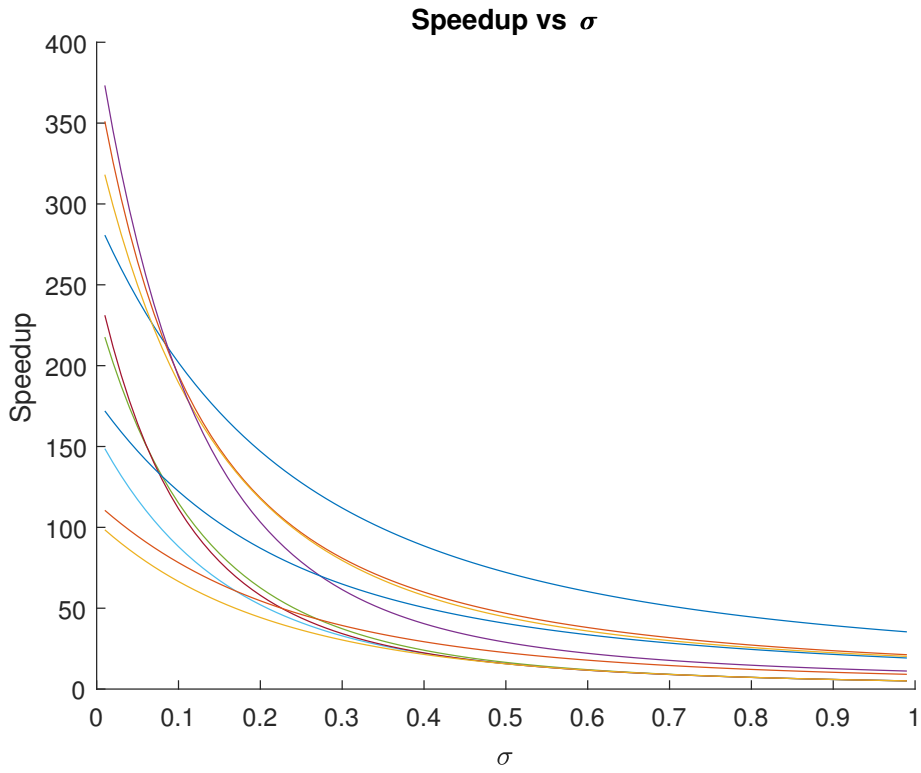

Figure 4.22: Subgraph and individual data injection on torus network

4.4 Comparison Result

4.4.1 Comparison Result Between Mesh Network and Toroidal With Same Number of Processors

Considering a $5 * 5$ mesh network Fig. 2.1, the best position for data injection is P_{12}. Other positions, for example P_8, P_{13}, they don't have the same speedup efficiency. Yet, for a toroidal $5 * 5$ regular mesh network, each position's efficiency is equal. Fig. 4.26 explores the comparison result between the toroidal and corner scenario difference.

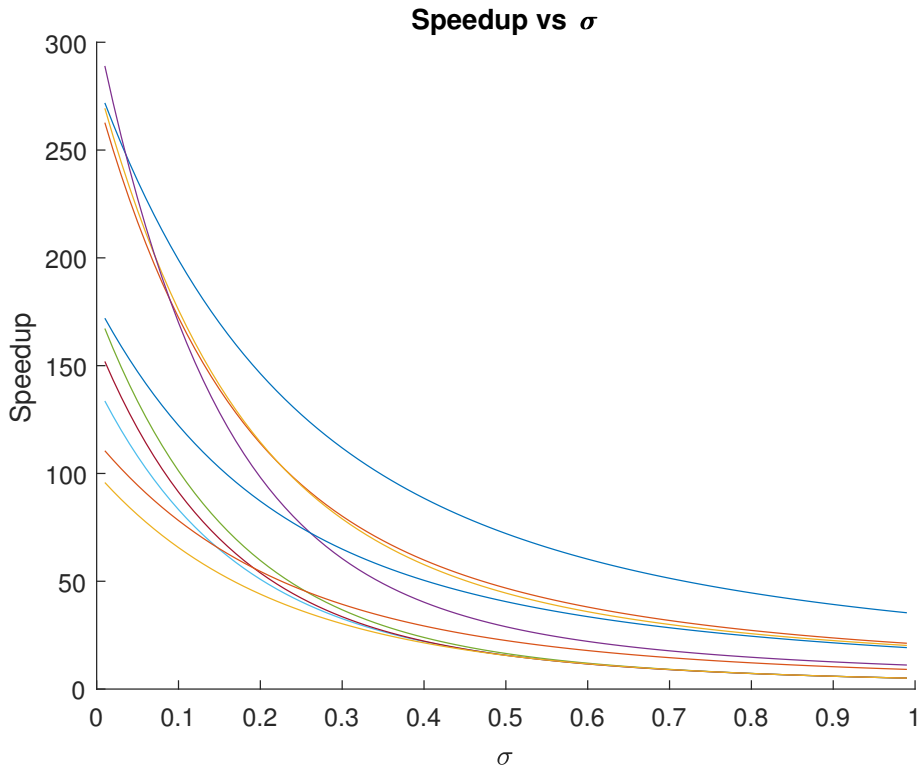

Figure 4.23: Subgraph and individual data injection on torus network

4.4.2 Comparison Result With Corner Processor and Inner Grid Processor

For a $5 * 5$ mesh network, the inner grid position is P_{12} and the corner data injection position is P_0.

The comparing result is Fig. 4.26.

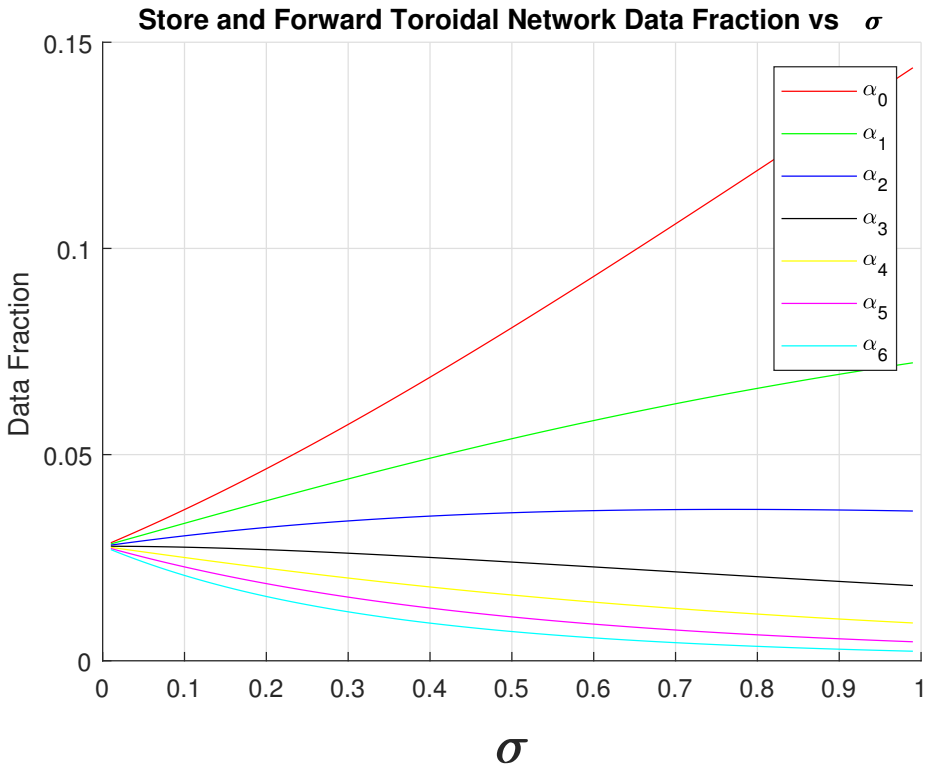

Figure 4.24: The data fraction curve of torus network Fig. 4.3

Generally speaking, Fig. 4.26 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.

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

Table 4.3: D_i vs Number of processor on D_i

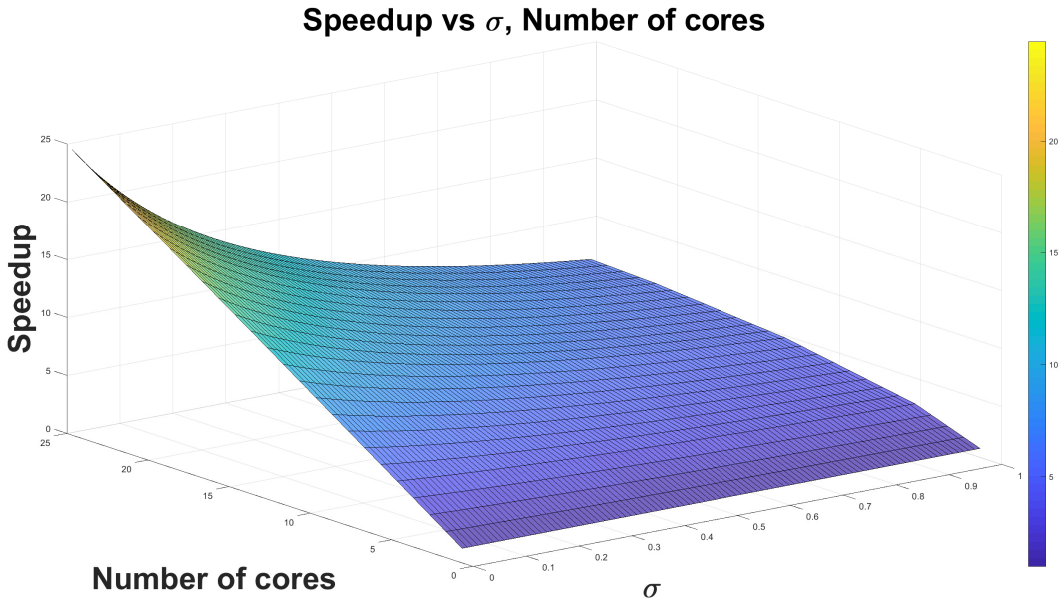

Figure 4.25: Sensitivity analysis result of torus

4.4.3 Comparison Result Between Virtual Cut-through Switching Scenario and Store and Forward Scenario

In the legend of figures, we use

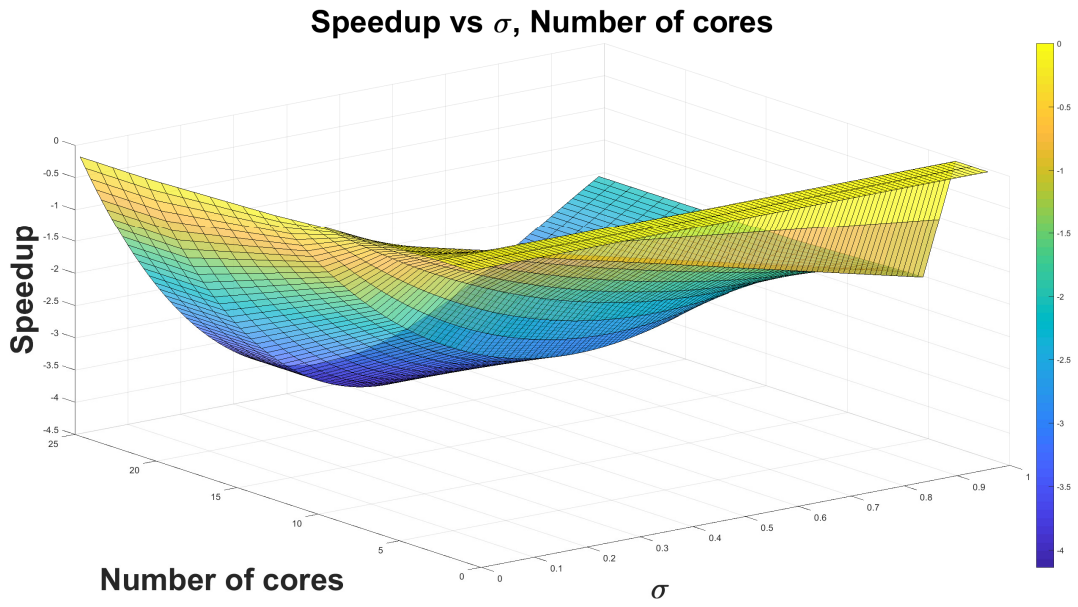

Figure 4.26: Speedup difference between corner injection and inner grid injection

- F presents virtual cut-though switching.
- NF presents store and forward switching.
- $F\alpha_0$ means the data fraction of P_0 in virtual cut-through switching scenario.
- $NF\alpha_0$ means the data fraction of P_0 in store and forward switching scenario.

Data Injection On the Corner Processor

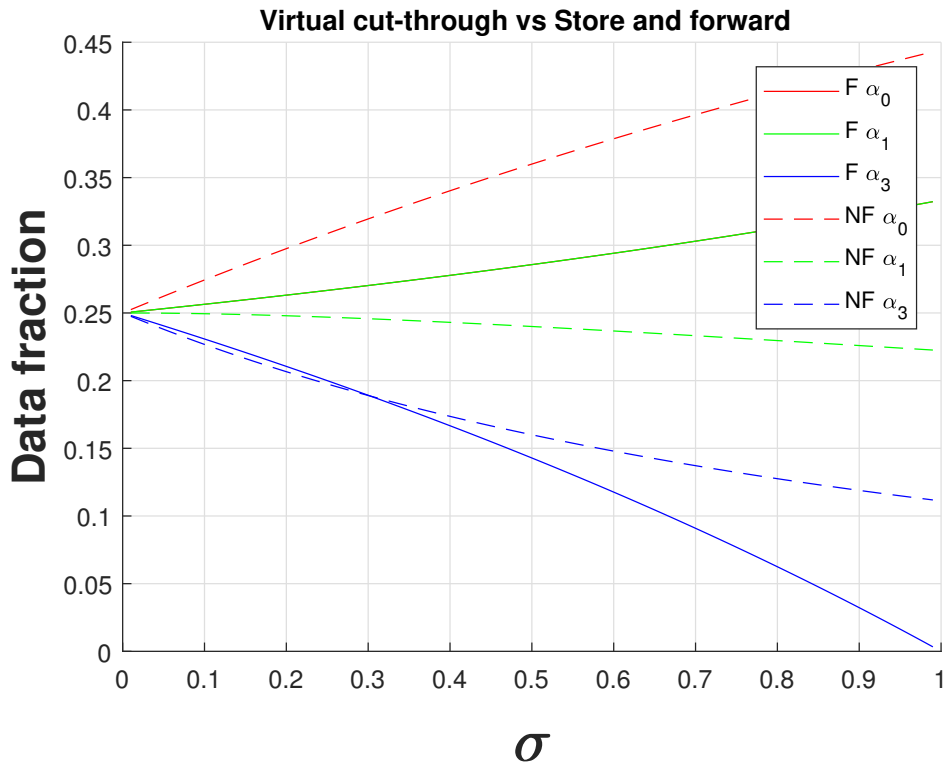

Figure 4.27: Comparing result between virtual cut-through switching and store and forward switching in $2 * 2$ mesh network

Fig. 4.27 says that P_0 takes more assigned task in without virtual cut-through switching scenario than virtual cut-through switching processor situation. As the value σ value goes up, the fractions are deployed to the deeper layers decreases. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to P_3 in the virtual cut-through switching assumption, yet in the without virtual cut-through switching situation, there is still about 10% data fraction are communicated to P_3.

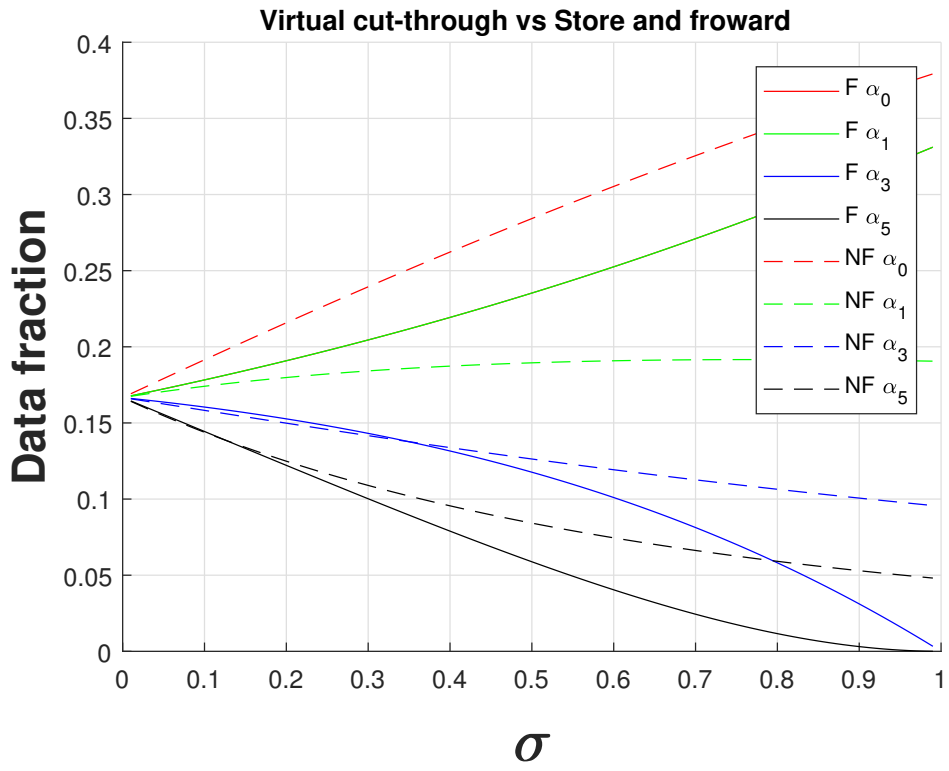

Figure 4.28: Comparing result between virtual cut-through switching and store and forward switching in $2 * 3$ mesh network

Fig. 4.28 says that P_0 takes more assigned task in without virtual cut-through switching scenario than virtual cut-through switching processor situation. As the σ value goes up, the fractions are deployed to the deeper levels degrades. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_3$, that is, P_5 in the virtual cut-through switching assumption. Yet in the without virtual cut-through switching situation, there is still about 5% data fraction is communicated to P_5.

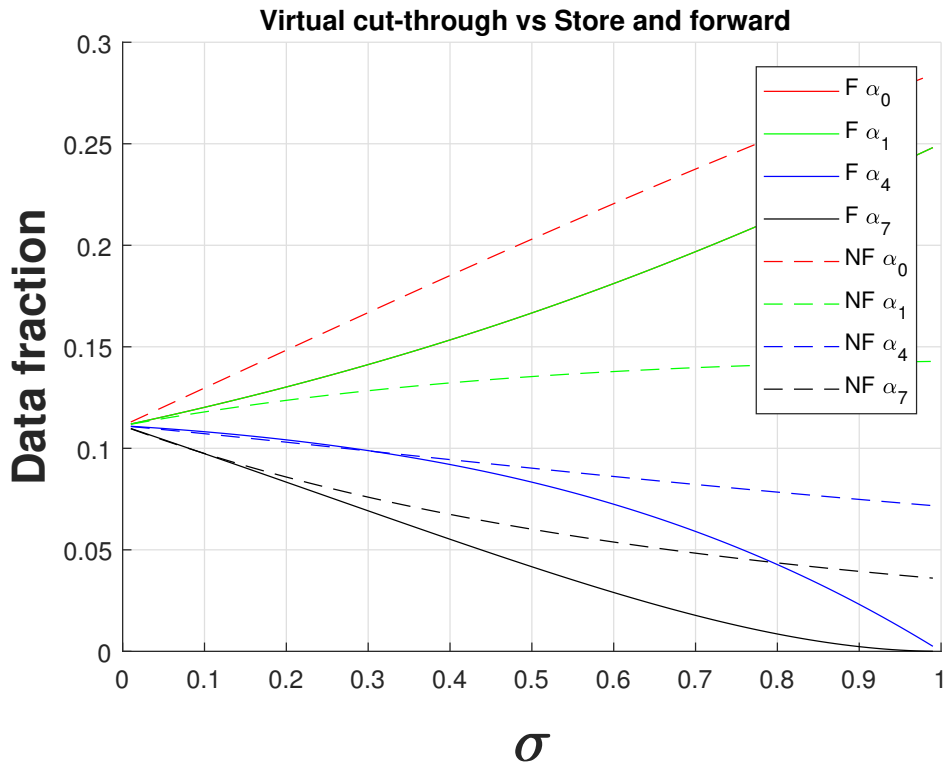

Figure 4.29: Comparing result between virtual cut-through switching scenario and store and forward switching scenario. The data injection is boundary processor.

Fig. 4.29 says that P_0 takes more assigned task in without virtual cut-through switching scenario than virtual cut-through switching processor situation. As the σ value goes up, the fractions are deployed to the deeper levels decreases. 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 virtual cut-through switching assumption. Yet in the without virtual cut-through switching situation, there is still about 5% data fraction is communicated to P_7 and P_8.

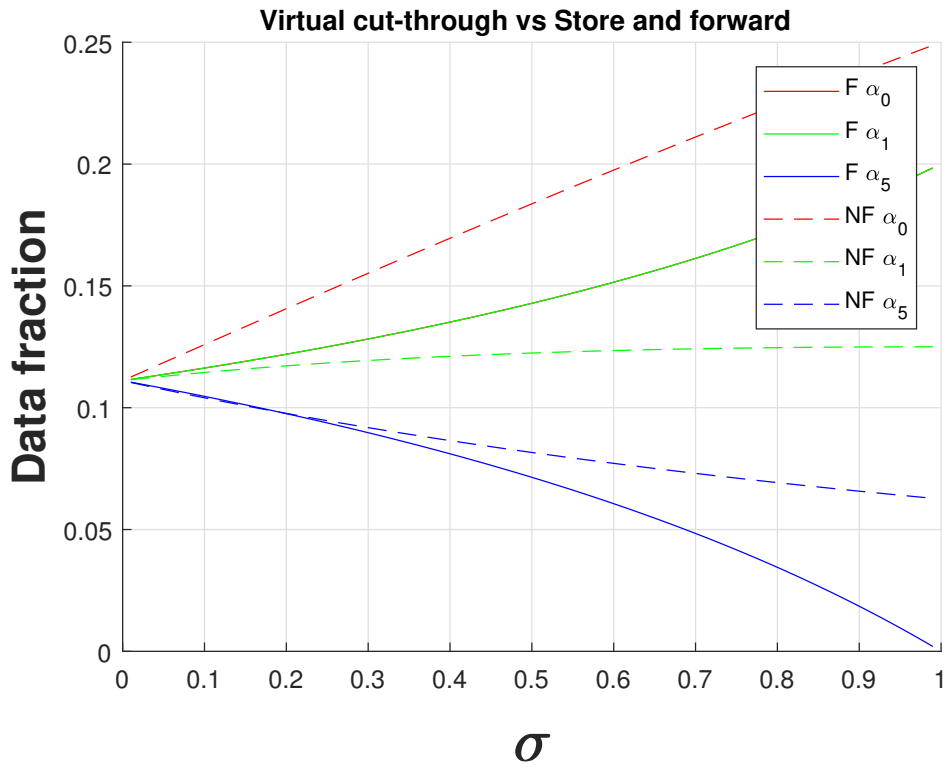

Figure 4.30: The comparing result between virtual cut-through switching processor with without virtual cut-through switching processor in 3*3 mesh network injection on inner grid processor

Fig. 4.30 says that P_0 takes more assigned task in without virtual cut-through switching scenario than virtual cut-through switching 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 P_8 in the virtual cut-through switching assumption. Yet in the without virtual cut-through switching situation, there is still about 5% data fraction is communicated to P_5, P_6, P_7 and P_8.

Comparing with Fig. 4.29, 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 $level_1$ on the boundary.

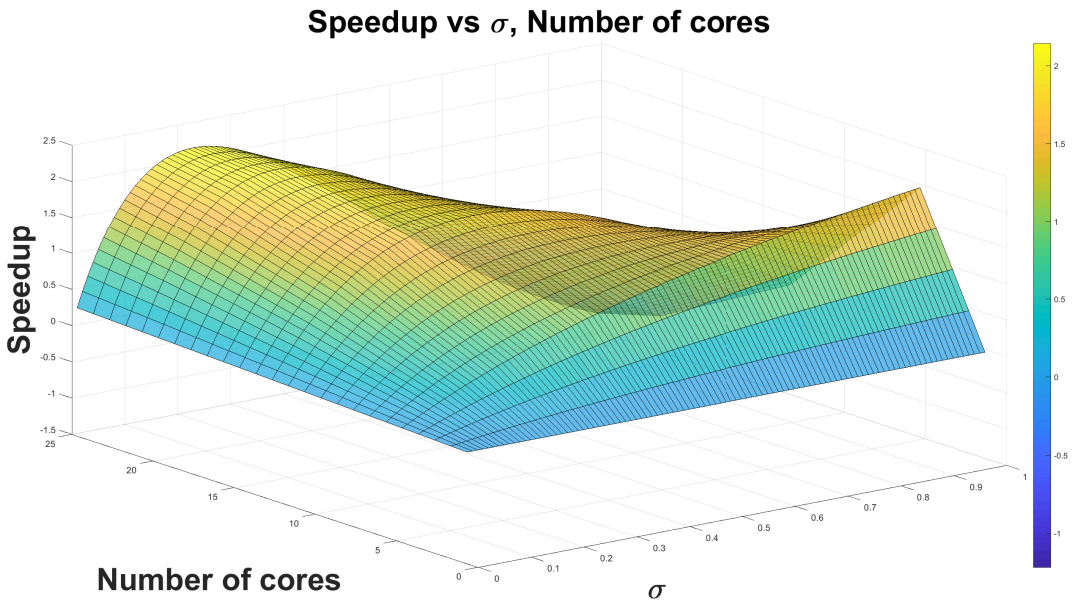

Figure 4.31: Speedup difference between virtual cut-through switching and store and forward switching in $5 * 5$ mesh network

Fig. 4.31 shows the speedup difference between the virtual cut-through switching situation and store and forward switching scenario.

Chapter 5

Hypercube Network

5.1 Virtual Cut-through Switching Scenario

The hypercube topology has two nodes along each dimension and $\log_2 n$ dimensions. The construction of a hypercube goes as follows, in general a $d-$dimensional hypercube is constructed by connecting corresponding nodes of two $(d-1)$ dimensional hypercubes Fig. 5.1.

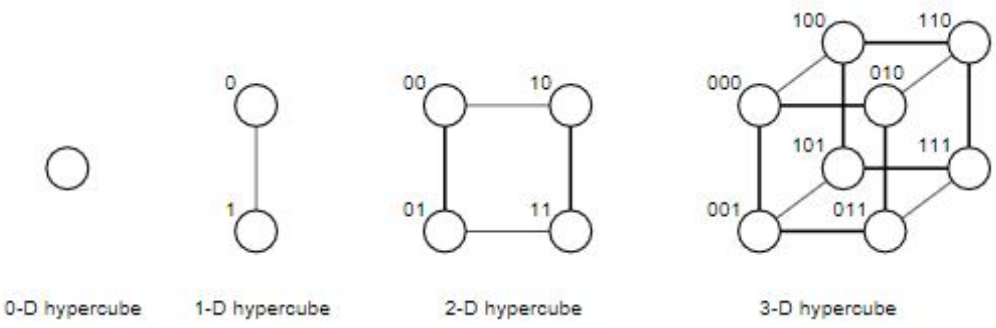

Figure 5.1: Hypercube in 0, 1, 2, 3 dimension. [1]

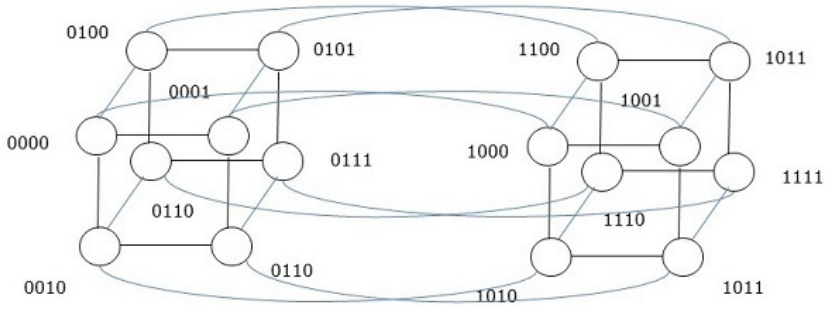

Figure 5.2: A hypercube network

In this work, we have two tasks as follows :

- One data injection, we propose a method of finding an optimal distribution of a divisible job among a cluster of processors connected by communication links and forming a hypercube network. The methodology we apply is similar to the flow matrix technique.

- Sensitivity analysis of the hypercube structure by adding more processors or dimensions.

- The multi-source sub-optimal algorithms to speedup job execution.

  - The data injection fractions are even.
  - The data injection fractions are different with each other.

5.1.1 Data Injection On The Grid Processor

For the hypercube of dimension d there are 2^d processor in the system. Each of the processors has direct links to d neighbors. A method of naming

the processors is to use label consisting of a binary string $d-$ position long. Further, the label of a processor is a binary number from the interval $[0, 2^d - 1]$

To address the qualitative model of computation, the critical problem is to calculate the number of processor on each D_i. Each node is connected by link and the hamming distance of their's label is 1. According to the lemma of [19],

Lemma 5.1.1. *In each layer i of $d-$dimensional hypercube, there are $\binom{n}{i}$ processors each of which can be accessed through i communications links and is capable of transmitting to $d-i$ still idle processors.*

According to a $2-D$ hypercube Fig. 3.1, the flow matrix is

$$A = \begin{bmatrix} \binom{2}{0} & \binom{2}{1} & \binom{2}{2} \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \tag{5.1}$$

, which is investigated in Regular Network Chapter.

According to a $3-D$ hypercube, the flow matrix is

$$A = \begin{bmatrix} \binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} \\ 1 & -1 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 \\ 0 & \sigma - 1 & \sigma & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 \\ 0 & \sigma - 1 & \sigma & 1 \end{bmatrix} \tag{5.2}$$

The speedup is $|-\det A|$.

A general case, $D-$dimension network, the flow matrix is :

$$A = \begin{bmatrix} \binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \cdots & \binom{n}{n-2} & \binom{n}{n-1} & \binom{n}{n} \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix}$$

5.1.2 Sensitivity Analysis Virtual Cut-through Switching Scenario

5.1.3 Multi-source Uniform Data Fraction

5.1.4 Multi-source Non-uniform Data Injection

5.2 Store and Forward Switching Scenario

In this work, we have two tasks as follows :

- One data injection, we propose a method of finding an optimal distribution of a divisible job among a cluster of processors connected by communication links and forming a hypercube network. The methodology we apply is similar to the flow matrix technique.

- Sensitivity analysis of the hypercube structure by adding more processors or dimensions.

- The multi-source sub-optimal algorithms to speedup job execution.
  - The data injection fractions are even.
  - The data injection fractions are different with each other.

5.2.1 Data Injection On The Grid Processor

According to a $2-D$ hypercube Fig. 3.1, the flow matrix is

$$A = \begin{bmatrix} \binom{2}{0} & \binom{2}{1} & \binom{2}{2} \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \quad (5.3)$$

, which is investigated in Regular Network Chapter.

According to a $3-D$ hypercube, the flow matrix is

$$A = \begin{bmatrix} \binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \tag{5.4}$$

The speedup is $|-\det A|$.

A general case, $D-$dimension network, the flow matrix is : We use $\sigma^\star$ to represents $-(\sigma+1)$.

$$A = \begin{bmatrix} \binom{n}{0} & \binom{n}{1} & \binom{n}{2} & \cdots & \binom{n}{n-2} & \binom{n}{n-1} & \binom{n}{n} \\ 1 & \sigma^\star & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & \sigma^\star \end{bmatrix}$$

5.2.2 Sensitivity Analysis Store and Forward Switching Scenario

5.2.3 Multi-source Uniform Data Fraction

5.2.4 Multi-source Non-uniform Data Injection

Chapter 6

General Network

6.1 General Case Virtual Cut-through Switching Scenario

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.1}$$

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:

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

.

6.2 General Case Store and Forward Switching Scenario

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -(\sigma+1) & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -(\sigma+1) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.2}$$

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.

The speedup is

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

Chapter 7

Conclusion and Future work

7.1 Conclusion

7.2 Future Work

Bibliography

[1] D. R.-P. Mundan, "Parallel programmingexercise sheet 2: Topologies." `https://www5.in.tum.de/lehre/vorlesungen/parhpp/materialien/sheet02.pdf`. Issued at 10/05/07, to be discussed at 15/05/07.

[2] 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.

[3] 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.

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

[5] 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.

[6] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi, *Scheduling divisible loads in parallel and distributed systems*, vol. 8. John Wiley & Sons, 1996.

[7] J. Sohn and T. G. Robertazzi, "Optimal load sharing for a divisible job on a bus network," tech. rep., Stony Brook, NY: State University of New York at Stony Brook, College of Engineering., 1992.

[8] 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.

[9] 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.

[10] 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.

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

[12] E. Krevat, J. G. Castaños, and J. E. Moreira, "Job scheduling for the bluegene/l system," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 38–54, Springer, 2002.

[13] F. Chin and C. A. Wang, "Finding the constrained delaunay triangulation and constrained voronoi diagram of a simple polygon in linear time," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 471–486, 1998.

[14] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[15] 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.

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

[17] 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.

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

[19] J. Błazewicz and M. Drozdowski, "Scheduling divisible jobs on hypercubes," *Parallel computing*, vol. 21, no. 12, pp. 1945–1956, 1995.