

Abstract of the Dissertation

**Closed-form Processor Equivalence And  
Scheduling Divisible Workloads From Multiple  
Sources In Regular, Toroidal And Hypercube  
Network**

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

<b>List of Figures</b> . . . . .	<b>xiii</b>
<b>List of Tables</b> . . . . .	<b>xiv</b>
<b>Acknowledgements</b> . . . . .	<b>xv</b>
<b>1</b> <b>Introduction</b> . . . . .	<b>1</b>
<b>2</b> <b>Problem Description</b> . . . . .	<b>3</b>
2.1 Problem Background . . . . .	3
2.2 Definitions and Assumption . . . . .	5
2.2.1 Notions . . . . .	7
<b>3</b> <b>Regular network</b> . . . . .	<b>9</b>
3.1 With Front-end Scenario . . . . .	9
3.1.1 Data Injection on The Corner Processor . . . . .	10
3.1.2 Data Injection On The Boundary Processor . . . . .	28
3.1.3 Data Injection On The Inner Grid Processor . . . . .	32
3.1.4 Sensitivity Analysis With Front-end Processors . . . . .	36
3.1.5 Multi-source Even Data Fraction . . . . .	41

3.1.6	Multi-source Different Data Fraction . . . . .	56
3.2	Without Front-end Scenario . . . . .	56
3.2.1	Data Injection on The Corner Processor . . . . .	57
3.2.2	Data Injection on The Boundary Processor . . . . .	69
3.2.3	Data Injection on The Inner Grid Processor . . . . .	72
3.2.4	Sensitivity Analysis Without Front-end Processors . . .	74
3.2.5	Multi-source Even Data Fraction . . . . .	77
3.2.6	Multi-source Different Data Fraction . . . . .	77
<b>4</b>	<b>Toroidal Rectangle Network . . . . .</b>	<b>80</b>
4.1	Toroidal Rectangle Network Notions . . . . .	80
4.2	With Front-end Scenario . . . . .	82
4.2.1	Data Injection On The Grid Processor . . . . .	82
4.2.2	Sensitivity Analysis of Toroidal Rectangle Network . .	85
4.2.3	Multi-source Even Data Injection . . . . .	85
4.2.4	Multi-source Different Data Injection . . . . .	87
4.3	Without Front-end Scenario . . . . .	88
4.3.1	Data Injection On The Grid Processor . . . . .	88
4.3.2	Sensitivity Analysis of Toroidal Rectangle Network . .	90
4.3.3	Multi-source Even Data Injection . . . . .	91
4.3.4	Multi-source Different Data Injection . . . . .	92
4.4	Comparison Result . . . . .	93
4.4.1	Comparison Result Between Regular Mesh and Toroidal With Same Processors Number . . . . .	93

4.4.2	Comparison Result With Corner Processor and Inner Grid Processor . . . . .	94
4.4.3	Comparison Result Between Front-end Processor and Without Front-end Processor . . . . .	96
4.5	Store and Froward Switching . . . . .	102
<b>5</b>	<b>Hypercube Network . . . . .</b>	<b>103</b>
5.1	With Front-end Scenario . . . . .	103
5.1.1	Data Injection On The Grid Processor . . . . .	104
5.1.2	Sensitivity Analysis With Front-end Processors . . . .	106
5.1.3	Multi-source Even Data Fraction . . . . .	107
5.1.4	Multi-source Different Data Fraction . . . . .	108
5.2	Without Front-end Scenario . . . . .	109
5.2.1	Data Injection On The Grid Processor . . . . .	109
5.2.2	Sensitivity Analysis Without Front-end Processors . .	111
5.2.3	Multi-source Even Data Fraction . . . . .	112
5.2.4	Multi-source Different Data Fraction . . . . .	113
<b>6</b>	<b>General Network . . . . .</b>	<b>114</b>
6.1	General Case With Front-end Processors . . . . .	114
6.2	General Case Without Front-end Processors . . . . .	116
<b>7</b>	<b>Conclusion and Future work . . . . .</b>	<b>117</b>
7.1	Conclusion . . . . .	117
7.2	Future Work . . . . .	118

<b>Bibliography</b>	.....	<b>119</b>
---------------------	-------	------------

## List of Figures

2.1	A m*n regular network(m = 5, n = 5) . . . . .	4
2.2	A toroidal rectangle network with grid unit cores . . . . .	5
2.3	A hypercube network . . . . .	5
3.1	The 2*2 regular network and the root processor is $P_0$ . . . . .	10
3.2	The timing diagram for 2*2 regular network and the root processor is $P_0$ . . . . .	11
3.3	2*2 regular network. $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ value curve . . . . .	13
3.4	The 2*3 regular network and the data injection happens on corner processor $P_0$ . . . . .	14
3.5	The timing diagram for a 2*3 regular network and the data injection happens on processor $P_0$ . . . . .	15
3.6	2*2 regular network. $\alpha_0, \alpha_1, \alpha_3, \alpha_5$ data fraction value . . . . .	18
3.7	The 2*n (n = 10) regular network and the workload happens on $P_0$ .	19
3.8	The timing diagram for 2*10 regular network and the data injection happens on $P_0$ . . . . .	20
3.9	3*8 regular network. The data injection position is $P_0$ . . . . .	26
3.10	The 3*3 regular network and the root processor is $P_0$ . . . . .	28

3.11	The timing diagram for 3*3 regular network and the data injection occurs on $P_0$ . . . . .	29
3.12	The data fraction simulation result of 3*3 regular network and the data injection happens on the boundary $P_0$ . . . . .	32
3.13	3*3 regular network. The data injection position is inner grid point $P_0$ . . . . .	33
3.14	The timing diagram for 3*3 regular network and the data injection is inner grid $P_0$ . . . . .	34
3.15	3*3 regular network. The data injection position is inner grid point $P_0$ . . . . .	36
3.16	Sensitivity analysis result of 2*10 regular network result . . . . .	37
3.17	Speedup curves of 2*10 regular network . . . . .	38
3.18	Sensitivity analysis result of 3*8 regular network and the injection position on boundary processor $P_2$ . . . . .	39
3.19	Speedup curve of 3*8 regular network . . . . .	40
3.20	Sensitivity analysis result of data injection position on inner grid processor . . . . .	41
3.21	Speedup curves of 5*5 regular network Fig. 2.1 . . . . .	42
3.22	Data injection consists of a subgraph of $G$ . . . . .	43
3.23	Data injection consists of a subgraph of $G$ . . . . .	43
3.24	Speedup vs $\sigma$ . . . . .	45
3.25	Injection Processor Data Fraction vs $\sigma$ . . . . .	46
3.26	Speedup vs $\sigma$ . . . . .	47
3.27	Injection Processor Data Fraction vs $\sigma$ . . . . .	48

3.28	10 Voronoi Cells . . . . .	50
3.29	10 Voronoi Cells speedup curves . . . . .	52
3.30	10 reduced Voronoi cells . . . . .	53
3.31	10 reduced Voronoi cells's speedup curves . . . . .	54
3.32	Reduced Voronoi Division Algorithm average processors' percentage	56
3.33	The timing diagram for 2*2 regular network without front-end.	57
3.34	The data fraction deployed based on the radius value . . . . .	59
3.35	The timing diagram for 2*3 regular network without front-end.	61
3.36	The data fraction deployed based on the radius value . . . . .	63
3.37	The timing diagram for 3*3 boundary data injection on $P_0$ . . .	69
3.38	The fraction curve for 3*3 boundary data injection on $P_0$ . . .	71
3.39	The timing diagram for 3*3 inner grid injection $P_0$ . . . . .	73
3.40	Sensitivity analysis result of 2*10 regular network result . . . . .	74
3.41	Speedup curve of 2*10 regular network result . . . . .	75
3.42	Sensitivity analysis result of 3*8 regular network result . . . . .	76
3.43	Speedup curve of 3*8 regular network result . . . . .	77
3.44	Sensitivity analysis result of data injection position on inner grid processor . . . . .	78
3.45	Speedup curve of 5*5 regular network result . . . . .	79
4.1	The rectangular toroidal network . . . . .	81
4.2	The rectangular toroidal network . . . . .	81
4.3	The $m^*n$ toroidal rectangle network and the data injection is $P_{4,2}$	83
4.4	The data fraction curve of Fig. 4.3 . . . . .	85

4.5	Sensitivity analysis result of data injection position on inner grid processor . . . . .	86
4.6	How to calculate torus Voronoi Diagram . . . . .	87
4.7	Initial Voronoi Digram . . . . .	88
4.8	Torus Voronoi Diagram . . . . .	89
4.9	Voronoi Diagram Casting to the torus model . . . . .	90
4.10	Voronoi Diagram Casting to the torus model . . . . .	91
4.11	Torus Reduced Voronoi Diagram Casting to the Torus Model . .	92
4.12	Torus Reduced Voronoi Diagram . . . . .	93
4.13	The data fraction curve of Fig. 4.3 . . . . .	94
4.14	Sensitivity analysis result of data injection position on inner grid processor . . . . .	95
4.15	Speedup difference between corner injection and inner grid injection	95
4.16	The comparing result between front-end processor with without front-end processor in 2*2 regular network . . . . .	97
4.17	The comparing result between front-end processor with without front-end processor in 2*3 regular network . . . . .	98
4.18	The comparing result between front-end processor with without front-end processor in 3*3 regular network injection on boundary processor . . . . .	99
4.19	The comparing result between front-end processor with without front-end processor in 3*3 regular network injection on inner grid processor . . . . .	100

4.20	Speedup difference between front-end and without front-end in 5*5 regular network . . . . .	101
5.1	Hypercube in 0, 1, 2, 3 dimension. [1] . . . . .	103

## List of Tables

4.1	$D_i$ vs Number . . . . .	84
4.2	The processor number of various $D_i$ . . . . .	86
4.3	The processor number of various $D_i$ . . . . .	90

## Acknowledgements

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

Thank you for Professor Esther M. Arkin, Professor Joseph S.B. Mitchell and Professor Yue Zhao, who give me lots of paper modification suggestions and join my thesis defense committee. Thank you for Professor Jim Jiao, who gives me lots of suggestions about Ph.D. life time management.

Thank you for Christine Rota, who helps me to process the internship application, visa issue, and so on.

Thank you for Professor David Gu, who guides me about the optimal mass transport theory and computational conformal geometry.

Thank you for my co-authors, especially Ph.D. Liang Mi, I learn lots of medical image knowledge from him.

Thank you for Ocoos, eBay, BNL and Uber, I came across four perfect internship mentors. They gave me massive industry project training and a right 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 about the regular network. We take into account of the processor equivalence problem with front-end scenario and without front-end situation. The sensitivity analysis is another topic. Considering about the multi-source workload assignment problem, we propose a reduced Voronoi diagram assignment.
- Chapter IV presents the toroidal rectangle network situation. We consider the closed-form of processor equivalence in unique load injection, in different injection positions, corner processor, boundary processor and inner grid processor, respectively. The sensitivity analysis and multi-source assignment problem are also referred.
- Chapter V investigates the hypercube network situation. The front-end

and without front-end assumption are discussed. The sensitivity analysis is referred and the multi-source assignment in hypercube environment is also an interesting problem.

- Chapter VI figures out 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 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 network of regular 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. The other one is scheduling divisible workloads from multiple sources in regular network Fig. 2.1, toroidal rectangle network Fig. 2.2 and hypercube

network Fig. 2.3.

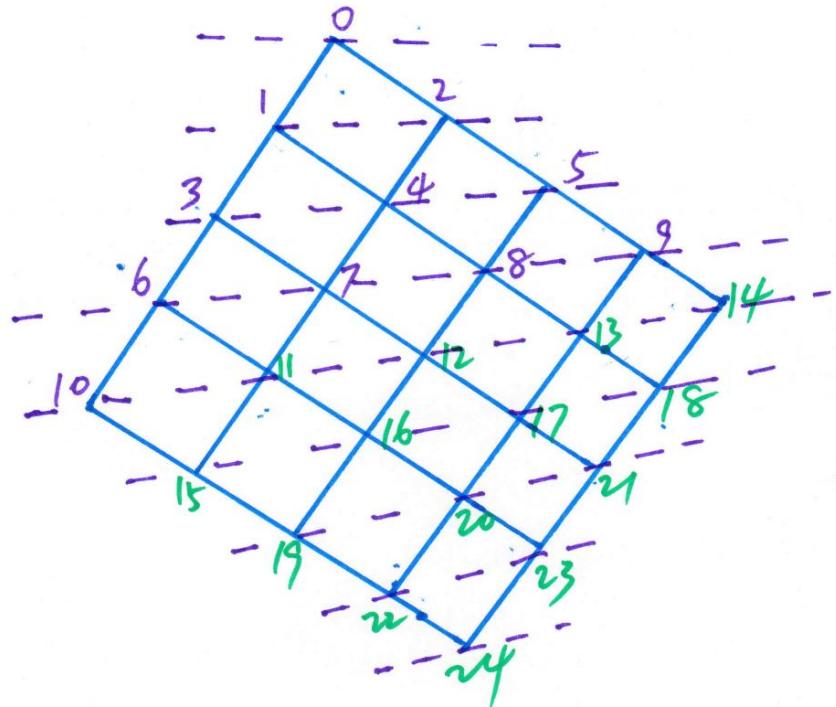


Figure 2.1: A  $m \times n$  regular network ( $m = 5, n = 5$ )

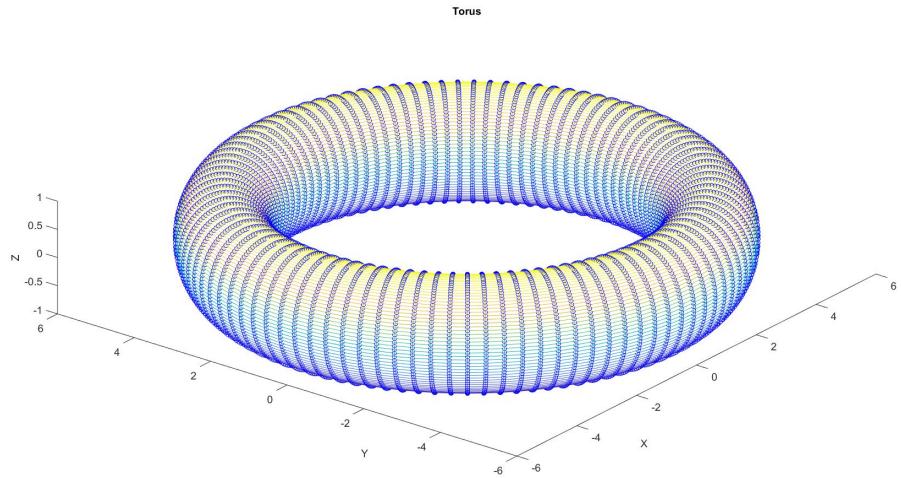


Figure 2.2: A toroidal rectangle network with grid unit cores

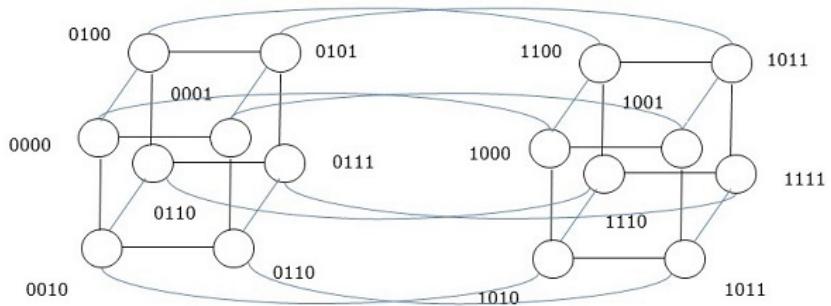


Figure 2.3: A hypercube network

## 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 the unit 1 workload.*

The following assumptions are used throughout the paper:

- The virtual cut through [5] switching is used to transmit the assigned workload between processors.
- For simplicity, we do not consider return communications.
- 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 4 or 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 get the minimum finish time.
- Multi-source assignment : how to finish the unit 1 workload at the same time utilizing smaller processor.

To achieve the minimum solution is obtained by forcing the processors over a network to stop processing simultaneously. Intuitively, this is because the solution could be improved by transfer load from some busy processor to idle ones.

### 2.2.1 Notions

The following notations and definitions are utilized:

- $P_i$ : The  $i$ th processor.  $0 \leq i \leq m * n - 1$ .
- $L_i$ : The  $i$ th work load.  $1 \leq i \leq k$ .
- $D_i$ : The minimum number of hops from the processor  $P_i$  to the data load injection  $L$ .
- $level_i$ : The processors have  $i$  minimum Manhattan distance to the data injection.
- $\alpha_0$ : The load fraction assigned to the root processor.
- $\alpha_i$ : The load fraction assigned to the  $i$ th processor.
- $\omega_i$ : The inverse computing speed on the  $i$ th processor.
- $\omega_{eq}$ : The inverse computing speed on an equivalent node collapsed from a cluster of processors.
- $z_i$ : The inverse link speed on the  $i$ th link.

- $T_{cp}$ : Computing intensity constant. The entire load is processed in  $\omega_i T_{cp}$  on the  $i$ th processor.
- $T_{cm}$ : Communication intensity constant. The entire load is transmitted in  $z_i T_{cm}$  seconds over the  $i$ th 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  $\omega_0 T_{cp}$ .
- $\sigma = \frac{z T_{cm}}{\omega T_{cp}}$ : The ratio between the communication speed to the computation speed,  $0 < \sigma < 1$  [6] [7].
- 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**

## **Regular network**

### **3.1 With Front-end Scenario**

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

First we consider about the  $2 * 2$  regular network,  $2 * n$  regular network. After, we analyze a more general case  $m * n$  regular 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 Regular 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.

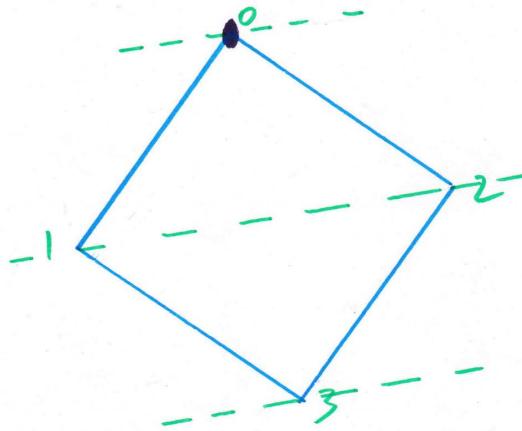


Figure 3.1: The 2\*2 regular network and the root processor is  $P_0$

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 until the  $\alpha_1$  and  $\alpha_2$  are completed transmission.

According to the divisible load theory[8], we obtain the timing diagram Fig. 3.2.

Based on the timing diagram, we get a group of equations to deploy the fraction workload:

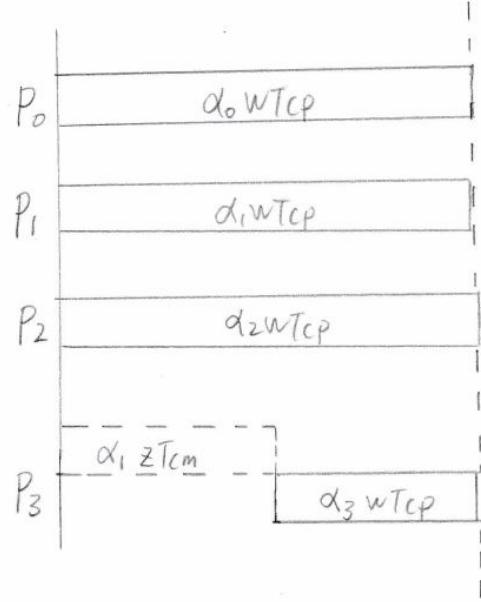


Figure 3.2: The timing diagram for  $2^*2$  regular network and the root processor is  $P_0$

$$\left\{ \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 \end{array} \right. \quad \begin{array}{l} (3.1) \\ (3.2) \\ (3.3) \\ (3.4) \\ (3.5) \\ (3.6) \\ (3.7) \\ (3.8) \\ (3.9) \end{array}$$

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 ***flow matrix***.

Finally, the explicit solution is:

$$\sigma = \frac{zT_{cm}}{\omega T_{cp}} \quad (3.11)$$

$$\alpha_0 = \frac{1}{4 - \sigma} \quad (3.12)$$

$$\alpha_1 = \frac{1}{4 - \sigma} \quad (3.13)$$

$$\alpha_3 = \frac{1 - \sigma}{4 - \sigma} \quad (3.14)$$

The simulation result is illustrated:

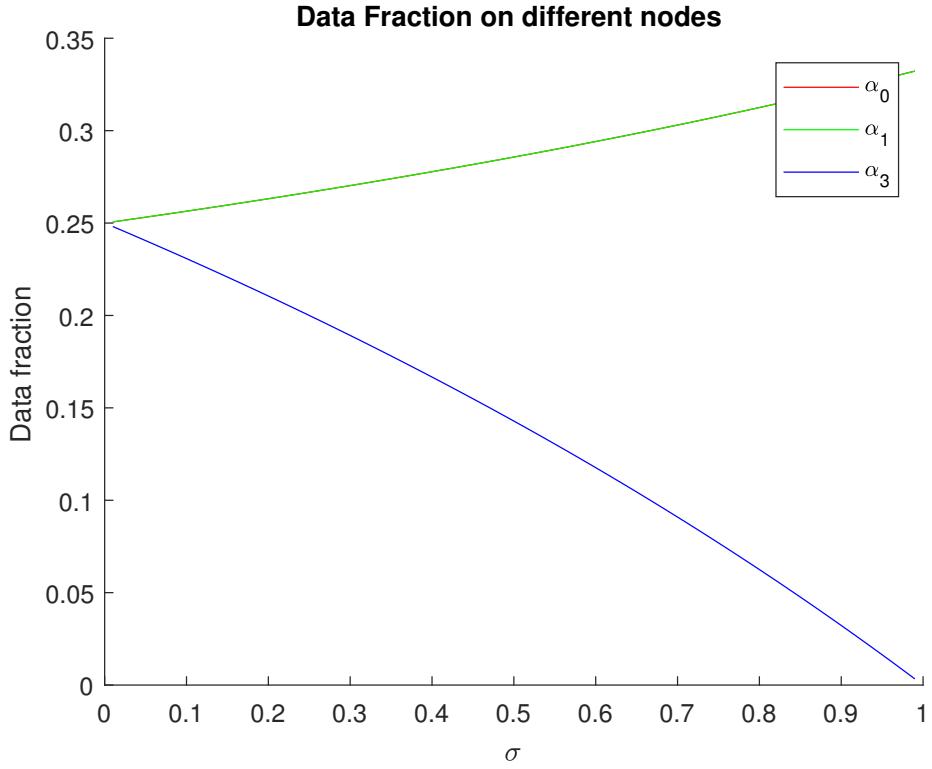


Figure 3.3:  $2^*2$  regular network.  $\alpha_0, \alpha_1, \alpha_2, \alpha_3$  value curve

In Fig. 3.3,  $P_0, P_1, P_2$  three processors have the same data fraction workload, so the curve of  $\alpha_0$  and  $\alpha_1$  coincide. The figure says that as  $\sigma$  grows, the value  $\alpha_3$  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  $P_0$  nor 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 Regular Network

In Fig. 3.4 regular network,  $L$  happens on processor  $P_0$ . There are 6 processors to be processing.

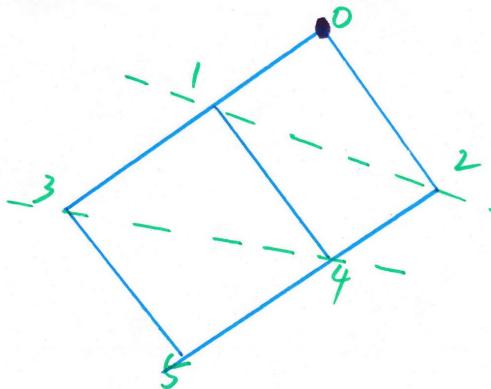


Figure 3.4: The 2\*3 regular network and the data injection happens on corner processor  $P_0$

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  $P_1$ ,  $P_2$ . That is,  $P_3$  and  $P_4$  have to wait the fraction of  $\alpha_1$  and  $\alpha_2$  are transmitted completely. The last processor  $P_5$  starts to execute until the work load fraction  $\alpha_0$ ,  $\alpha_1$ ,  $\alpha_2$ ,  $\alpha_3$ ,  $\alpha_4$  are transmitted completed. According to the divisible load theory[8], we obtain the timing diagram Fig. 3.5.

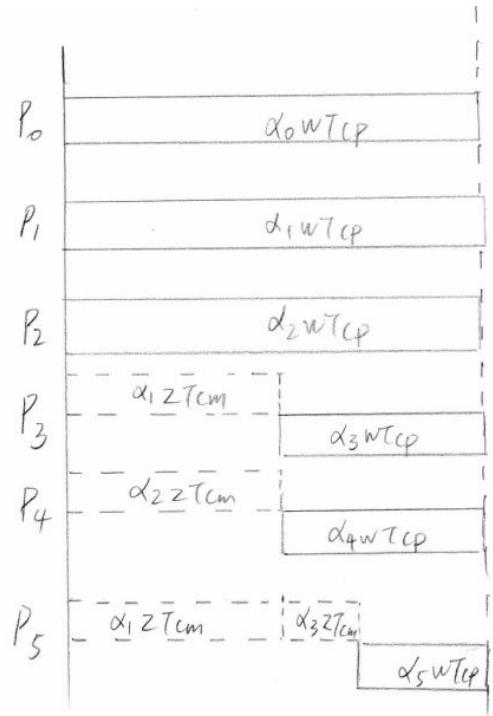


Figure 3.5: The timing diagram for a  $2^*3$  regular network and the data injection happens on processor  $P_0$

The equations as follows:

$$\left. \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ \alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 < 1 \end{array} \right\} \quad (3.15)$$

$$\alpha_1 \omega T_{cp} = T_{f,m} \quad (3.16)$$

$$\alpha_2 \omega T_{cp} = T_{f,m} \quad (3.17)$$

$$\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \quad (3.18)$$

$$\alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \quad (3.19)$$

$$(\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \quad (3.20)$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \quad (3.21)$$

$$\sigma = \frac{z T_{cm}}{\omega T_{cp}} \quad (3.22)$$

$$0 < \sigma < 1 \quad (3.23)$$

$$0 < \alpha_0 \leq 1 \quad (3.24)$$

$$0 \leq \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 < 1 \quad (3.25)$$

The flow matrix closed-form formula is:

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

The explicit solution is:

$$\left\{ \begin{array}{l} \sigma = \frac{zT_{cm}}{\omega T_{cp}} \end{array} \right. \quad (3.27)$$

$$\left\{ \begin{array}{l} \alpha_0 = \frac{1}{\sigma^2 - 4 \times \sigma + 6} \end{array} \right. \quad (3.28)$$

$$\left\{ \begin{array}{l} \alpha_1 = \frac{1}{\sigma^2 - 4 \times \sigma + 6} \end{array} \right. \quad (3.29)$$

$$\left\{ \begin{array}{l} \alpha_3 = \frac{1 - \sigma}{\sigma^2 - 4 \times \sigma + 6} \end{array} \right. \quad (3.30)$$

$$\left\{ \begin{array}{l} \alpha_5 = \frac{\sigma^2 - 2 \times \sigma + 1}{\sigma^2 - 4 \times \sigma + 6} \end{array} \right. \quad (3.31)$$

The  $\alpha$  calculation result are shown in Fig. 3.6.  $P_0$ ,  $P_1$  have the same fraction so the curve of  $\alpha_0$  and  $\alpha_1$  coincide.

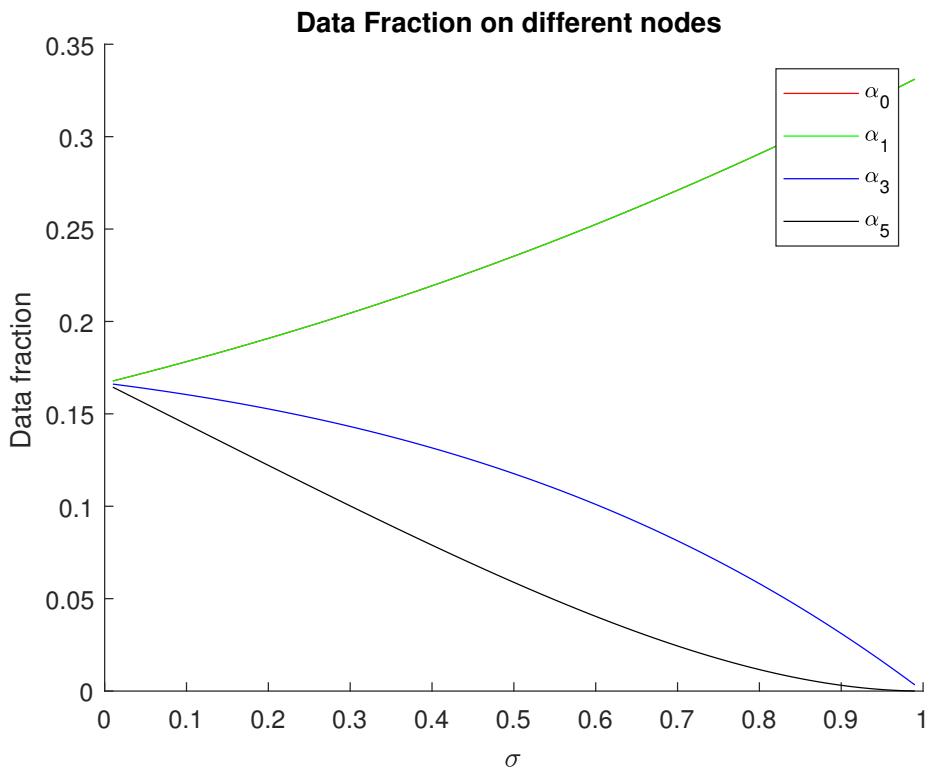


Figure 3.6:  $2^*2$  regular network.  $\alpha_0, \alpha_1, \alpha_3, \alpha_5$  data fraction value

### 2\*N Regular Network

The  $2 * n$  Fig. 3.7 homogeneous regular network address  $L_1$  at the same time and  $L$  happens on  $P_0$ .

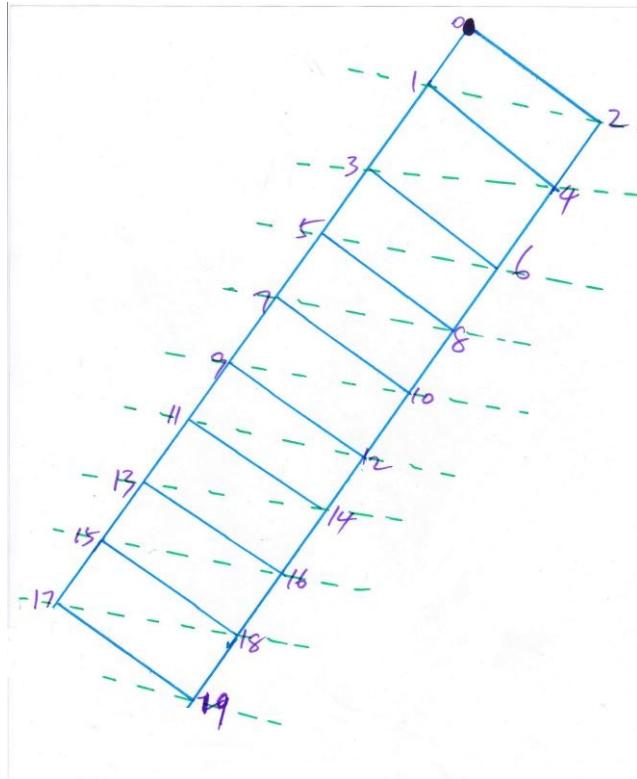


Figure 3.7: The  $2^*n$  ( $n = 10$ ) regular network and the workload happens on  $P_0$

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

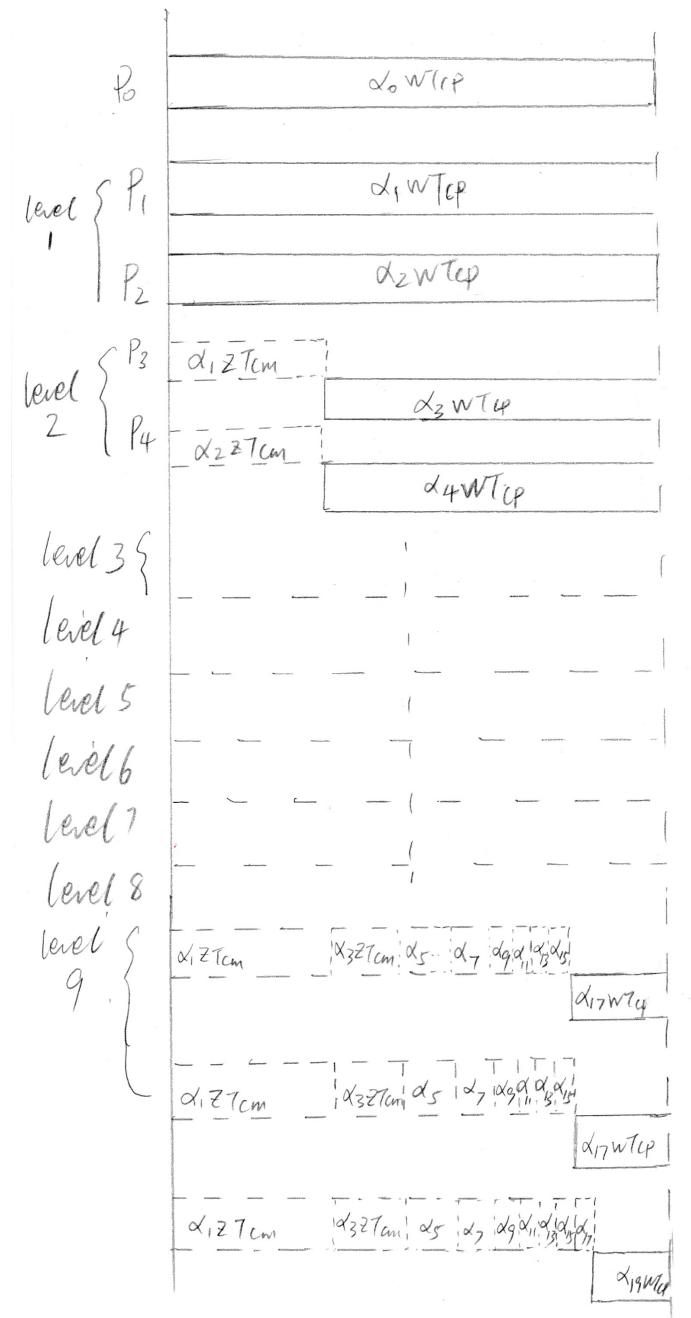


Figure 3.8: The timing diagram for  $2 \times 10$  regular network and the data injection happens on  $P_0$

The equations are presented as:

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

The flow matrix closed-form is shown:

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

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

$$\left\{ \alpha_i = \frac{\det A_i^*}{\det A} \right\} \quad (3.46)$$

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

Especially,

$$A_0^* = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \quad (3.47)$$

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

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

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  $\det A \neq 0$ .

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

$C$  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

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

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

Considering the matrix  $\hat{C}$

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

, 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 rule:

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

## $m*n$ Regular Network

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

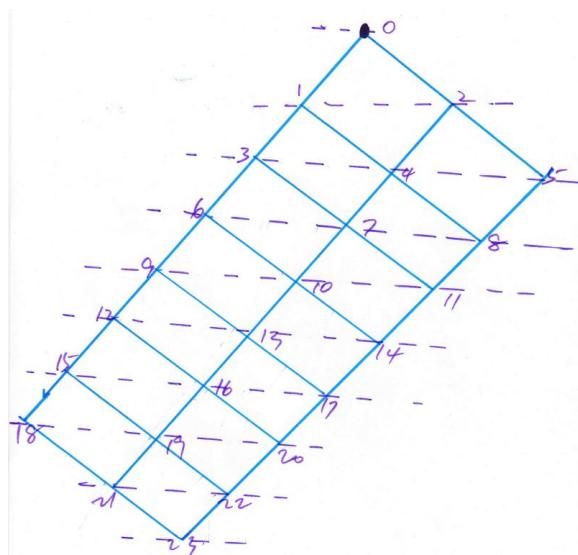


Figure 3.9:  $3*8$  regular network. The data injection position is  $P_0$

Utilizing the rule, we obtain the closed-form flow matrix equations for

Fig. 3.9:

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

Also, the flow matrix equations for Fig. 2.1:

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

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

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

### 3.1.2 Data Injection On The Boundary Processor

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

If the single data injection roots on the boundary processor, for example Fig. 3.10.

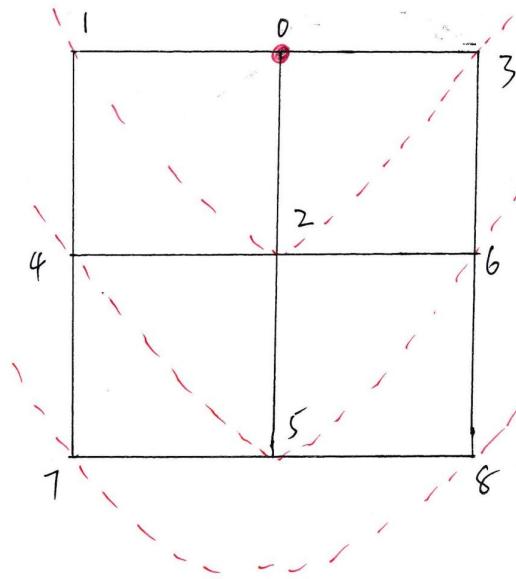


Figure 3.10: The  $3 \times 3$  regular network and the root processor is  $P_0$

The timing diagram is Fig. 3.11:

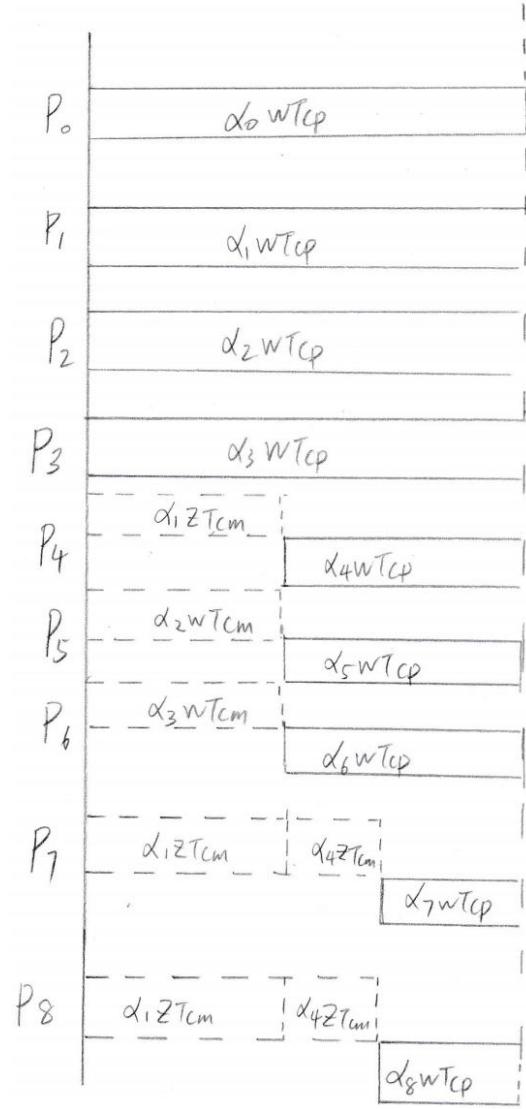


Figure 3.11: The timing diagram for  $3 \times 3$  regular network and the data injection occurs on  $P_0$

The equations are:

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

And the flow matrix form is :

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

The explicit solution is:

$$\left\{ \begin{array}{l} \alpha_0 = \frac{1}{9 - 7 \times \sigma + 2 \times \sigma^2} \end{array} \right. \quad (3.68)$$

$$\left\{ \begin{array}{l} \alpha_1 = \frac{1}{9 - 7 \times \sigma + 2 \times \sigma^2} \end{array} \right. \quad (3.69)$$

$$\left\{ \begin{array}{l} \alpha_4 = \frac{1 - \sigma}{9 - 7 \times \sigma + 2 \times \sigma^2} \end{array} \right. \quad (3.70)$$

$$\left\{ \begin{array}{l} \alpha_7 = \frac{(1 - \sigma)^2}{9 - 7 \times \sigma + 2 \times \sigma^2} \end{array} \right. \quad (3.71)$$

$$\left\{ \begin{array}{l} \alpha_0 = \frac{1}{9 - 7 \times \sigma + 2 \times \sigma^2} \\ \alpha_1 = \frac{1}{9 - 7 \times \sigma + 2 \times \sigma^2} \\ \alpha_4 = \frac{1 - \sigma}{9 - 7 \times \sigma + 2 \times \sigma^2} \\ \alpha_7 = \frac{(1 - \sigma)^2}{9 - 7 \times \sigma + 2 \times \sigma^2} \end{array} \right. \quad (3.72)$$

The simulation result is shown:  $P_0$  and  $P_1$  have the same  $\alpha$ , so the curve of  $\alpha_0$  and  $\alpha_1$  coincide.

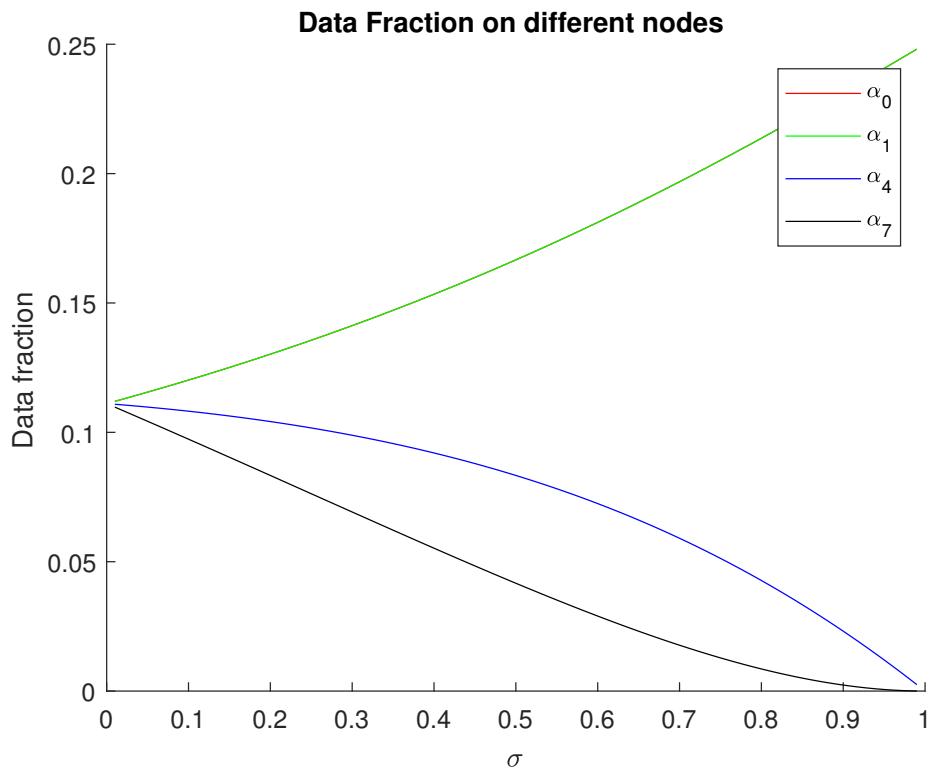


Figure 3.12: The data fraction simulation result of  $3 \times 3$  regular network and the data injection happens on the boundary  $P_0$

### 3.1.3 Data Injection On The Inner Grid Processor

Fig. 3.13 shows that  $L$  loads on the inner grid processor  $P_0$ ,

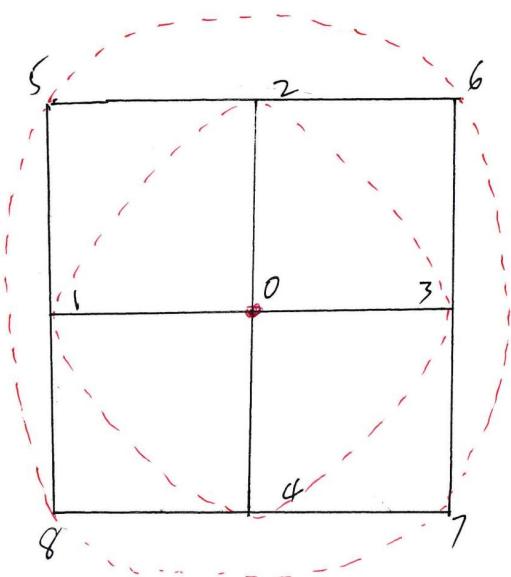


Figure 3.13:  $3 \times 3$  regular network. The data injection position is inner grid point  $P_0$

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

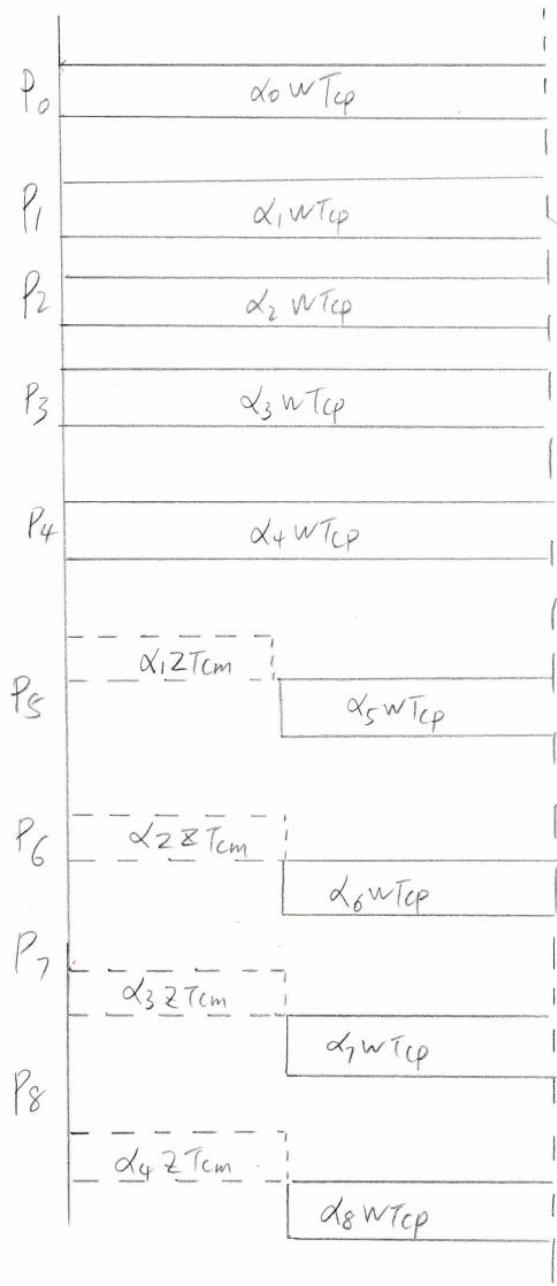


Figure 3.14: The timing diagram for  $3 \times 3$  regular network and the data injection is inner grid  $P_0$

The group of equations are:

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

The flow matrix form is :

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

The simulation result is Fig. 3.15:  $P_0$  and  $P_1$  have the same  $\alpha$  value, so the curve of  $\alpha_0$  and  $\alpha_1$  coincide.

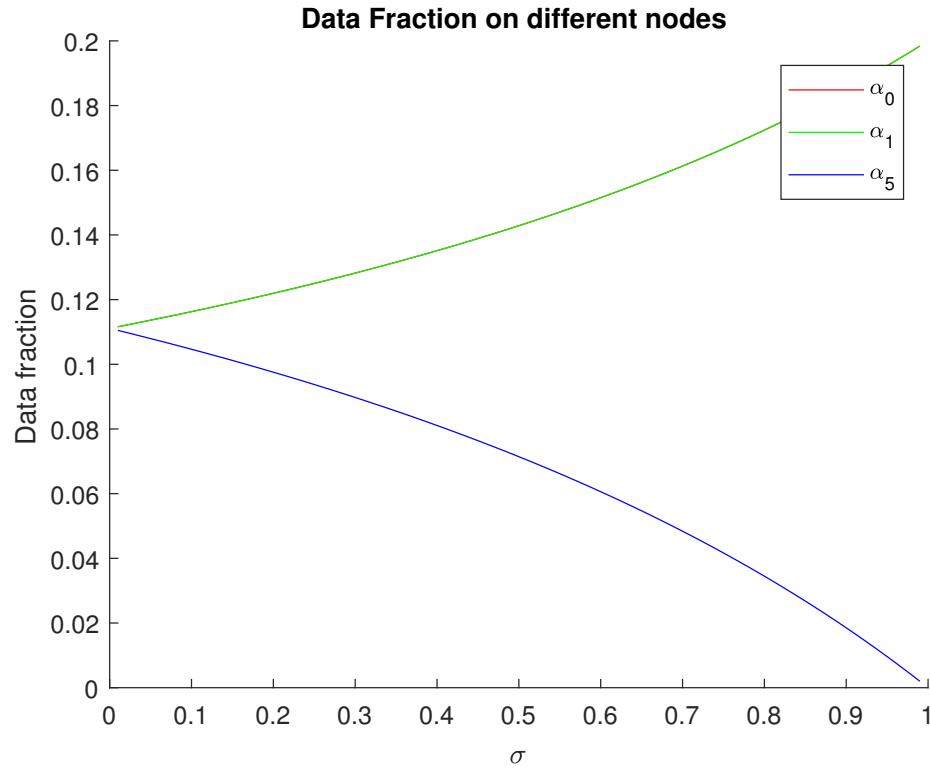


Figure 3.15: 3\*3 regular network. The data injection position is inner grid point  $P_0$

### 3.1.4 Sensitivity Analysis With Front-end Processors

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  $2 * n$  regular network Fig. 3.7 is as follows:

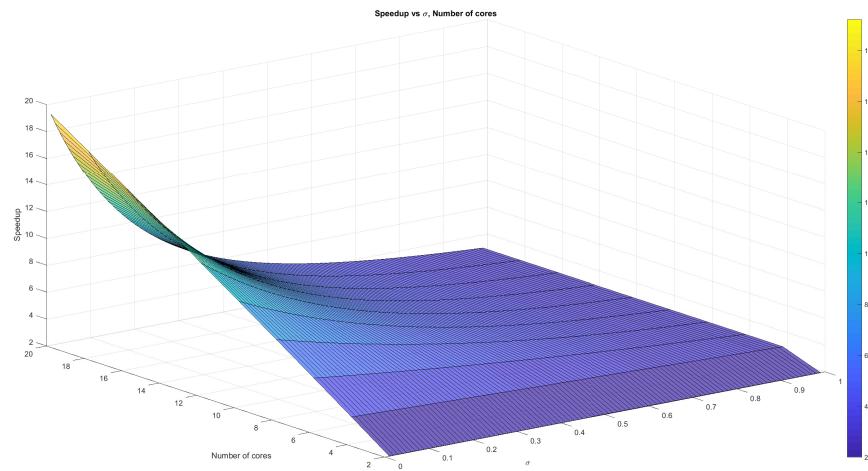


Figure 3.16: Sensitivity analysis result of  $2*10$  regular network result

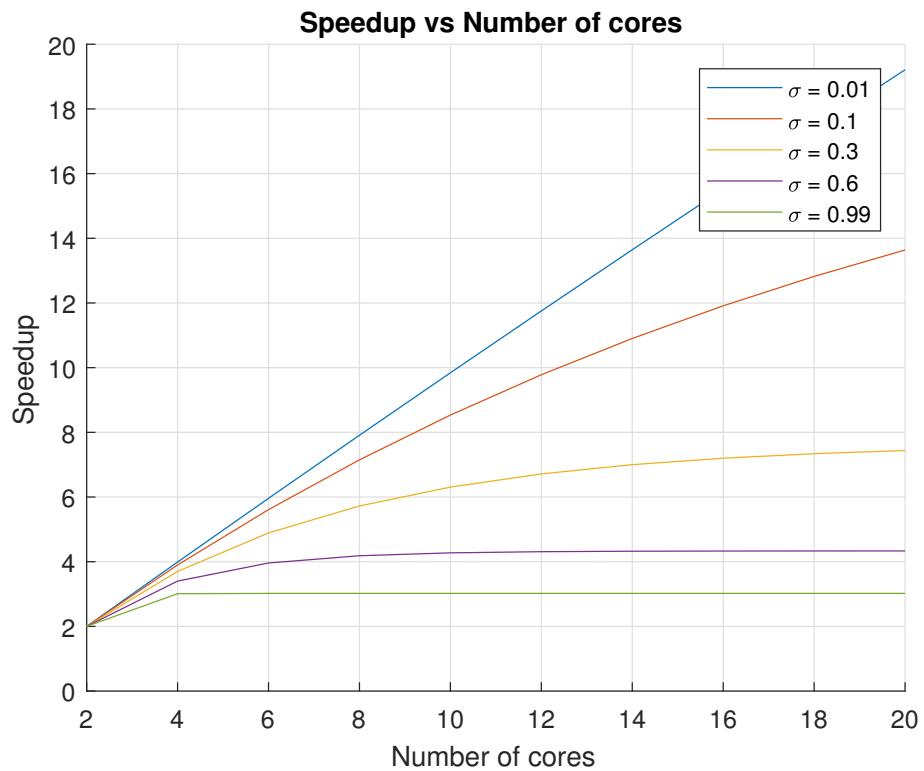


Figure 3.17: Speedup curves of  $2 \times 10$  regular network

### Data Injection On The Boundary Processor

Talking to Fig. 3.9, the data injection happens on boundary processor

$P_2$ .

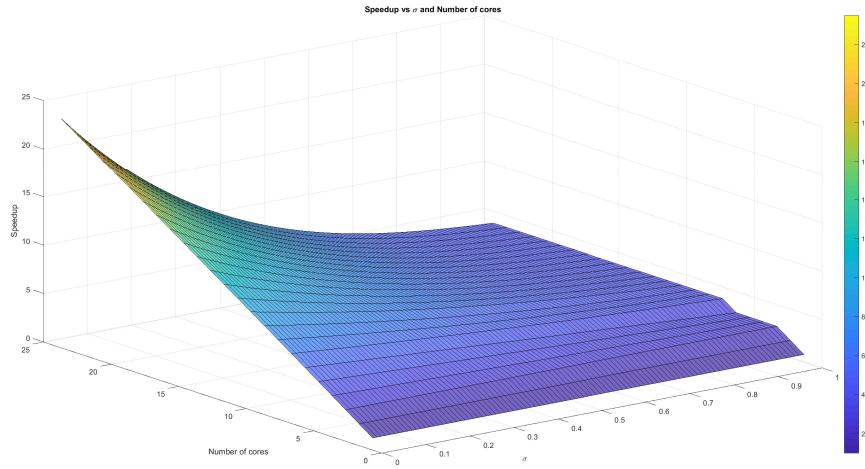


Figure 3.18: Sensitivity analysis result of  $3 \times 8$  regular network and the injection position on boundary processor  $P_2$

Fig. 3.18 and Fig. 3.19 show that if the value  $\sigma > 0.2$ , the speedup simulation effect is obvious. If the value  $\sigma < 0.1$ , the number of cores has linear impact on the speedup performance. If the number of cores  $> 5$ , the bottom effect with front-end, the cluster at least get about 4 times speedup. The reason is  $P_0$  has 3  $level_1$  processors.

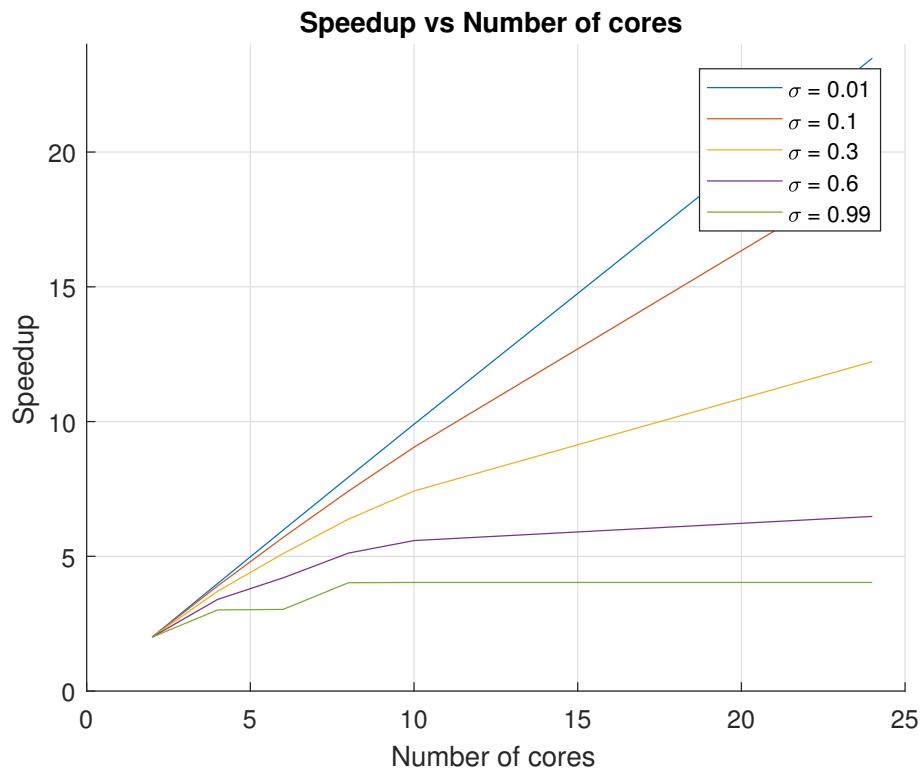


Figure 3.19: Speedup curve of  $3 \times 8$  regular network

### Data Injection On The Inner Grid Processor

Fig. 2.1,  $L$  incurs on  $P_{12}$  and the simulation result says:

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

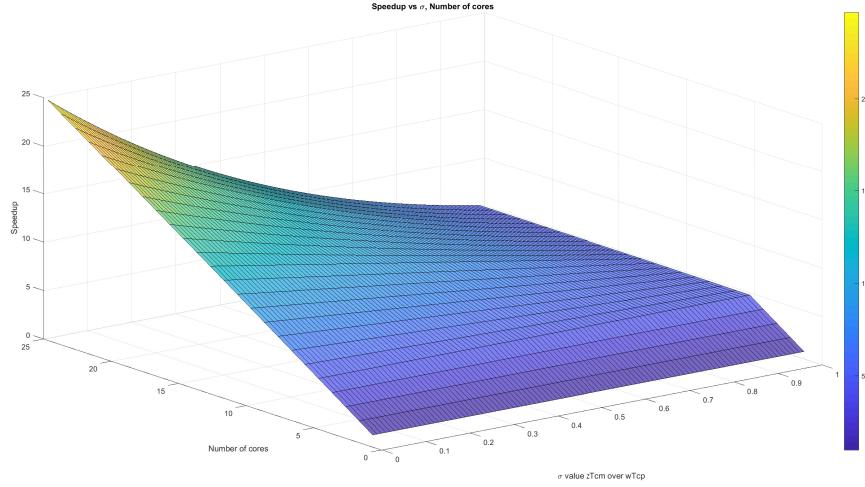


Figure 3.20: Sensitivity analysis result of data injection position on inner grid processor

### 3.1.5 Multi-source Even Data Fraction

We extend the single source assignment to multi-source assignment problem[9] [4]. According to each processor, we concentrate on the processors' geographical location  $P_i$ , data fraction assigned  $\alpha_i$ .

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.
- Other situations.

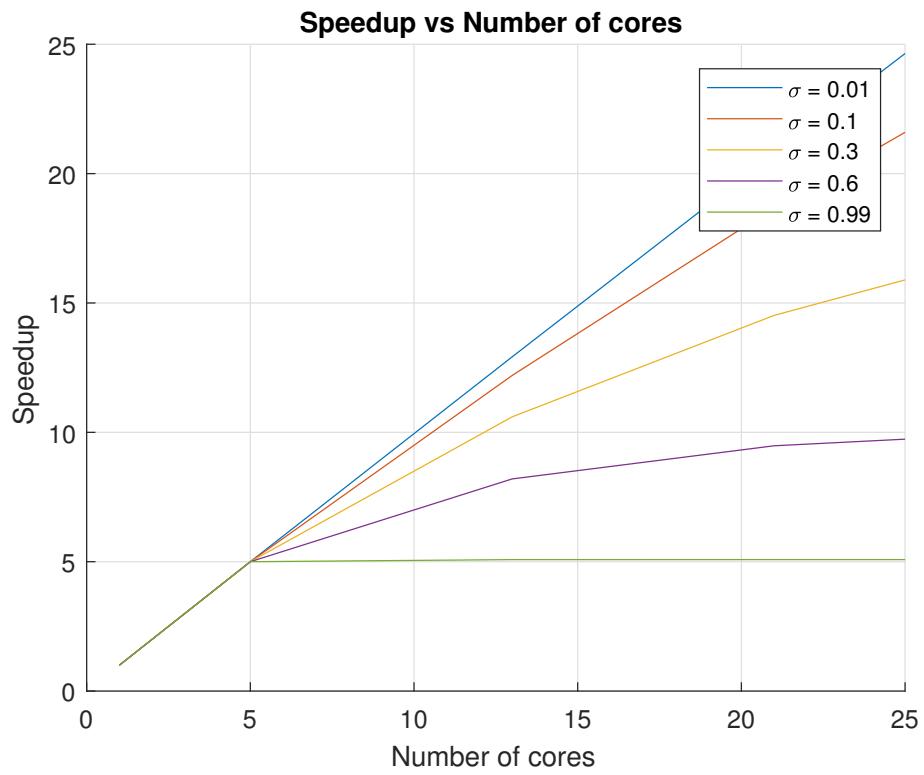


Figure 3.21: Speedup curves of  $5 \times 5$  regular network Fig. 2.1

### Situation I

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

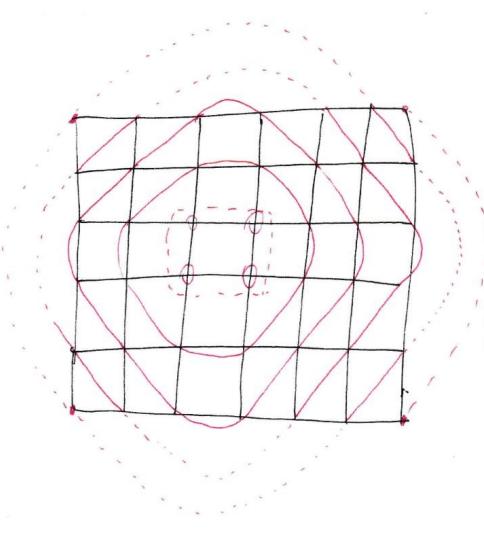


Figure 3.22: Data injection consists of a subgraph of  $G$

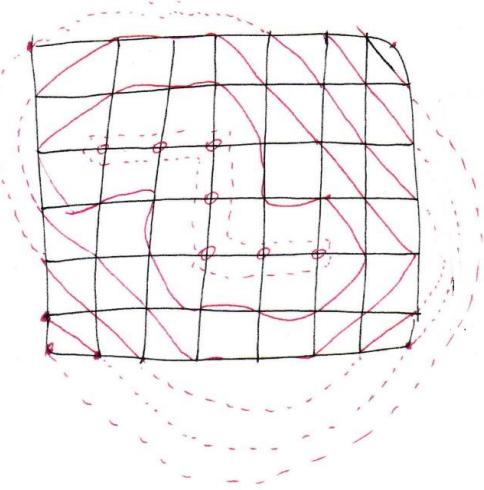


Figure 3.23: Data injection consists of a subgraph of  $G$

Fig. 3.22 and Fig. 3.23 illustrate two examples that the data injections consist of subgraph  $G_L$  of  $G$ .

Our objective is to propose a general algorithm framework to tackle this situation and give quantity model analysis. This algorithm is named as ***Two Phase Scheduling Algorithm (TPSA)***.

---

**Algorithm 1** Two Phase Scheduling Algorithm (TPSA)

---

*global*<sub>s</sub> :

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  $\alpha_i$

*local*<sub>s</sub> :

Re-distributing workload between the data injection processors.

---

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

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.87)$$

The simulation result illustrates as follows:

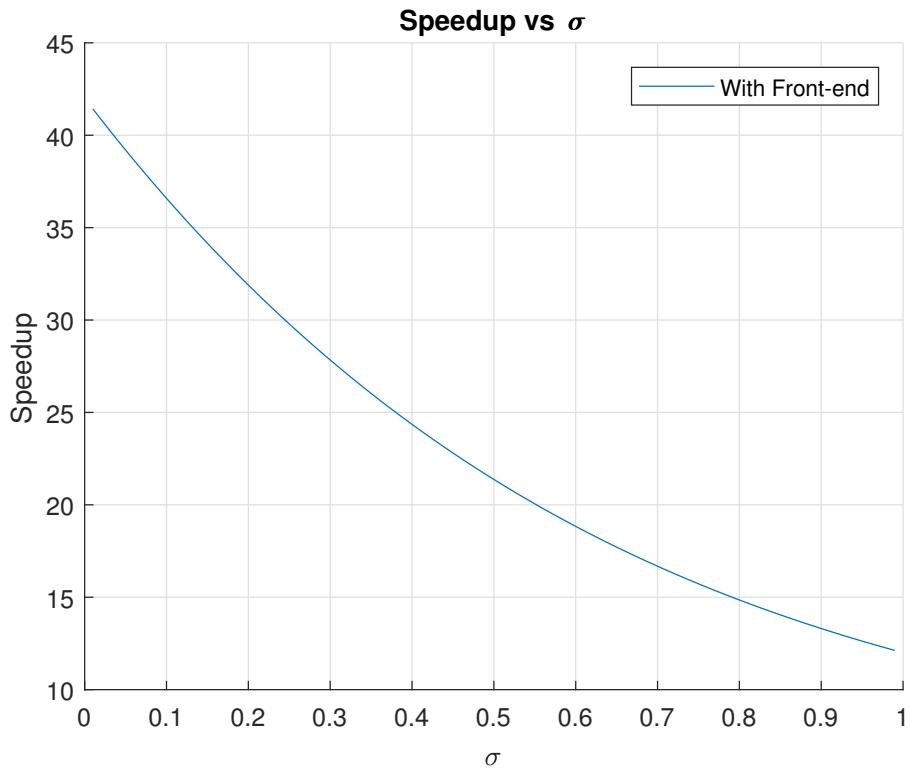


Figure 3.24: Speedup vs  $\sigma$

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

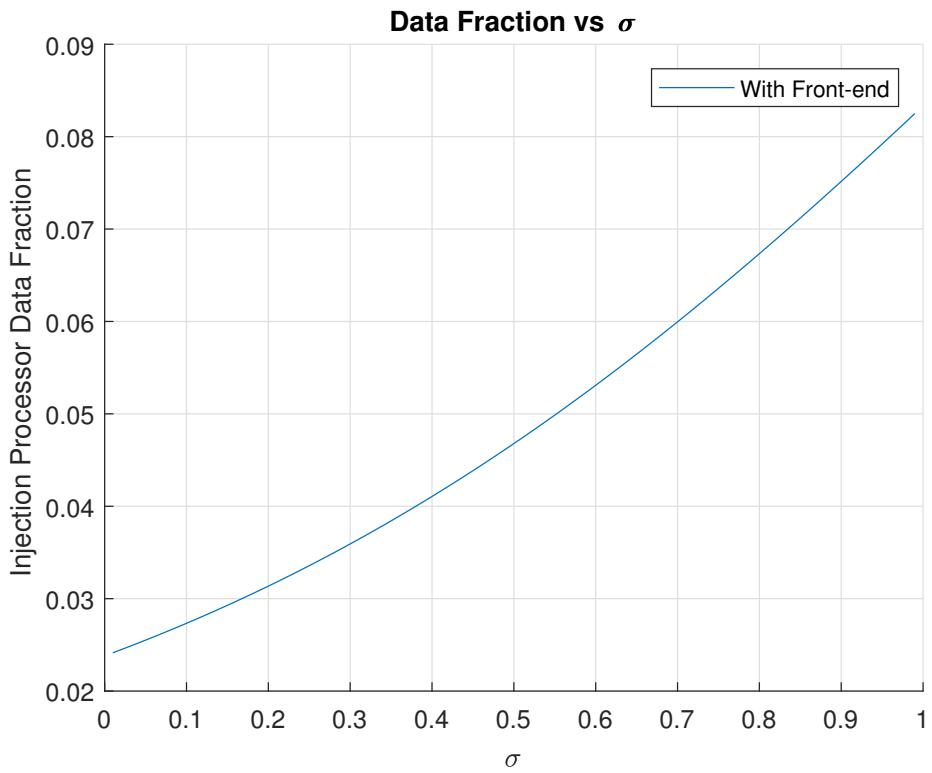


Figure 3.25: Injection Processor Data Fraction vs  $\sigma$

which hits about 40 times speedup. If  $\sigma \approx 1$ , the network achieves almost 12 times performance. In addition, the data fraction in each injection processors are the same, that is, there is no need to transfer data between local processors.

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

$$\begin{bmatrix} 7 & 14 & 15 & 10 & 6 & 3 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.88)$$

The simulation result illustrates as follows:

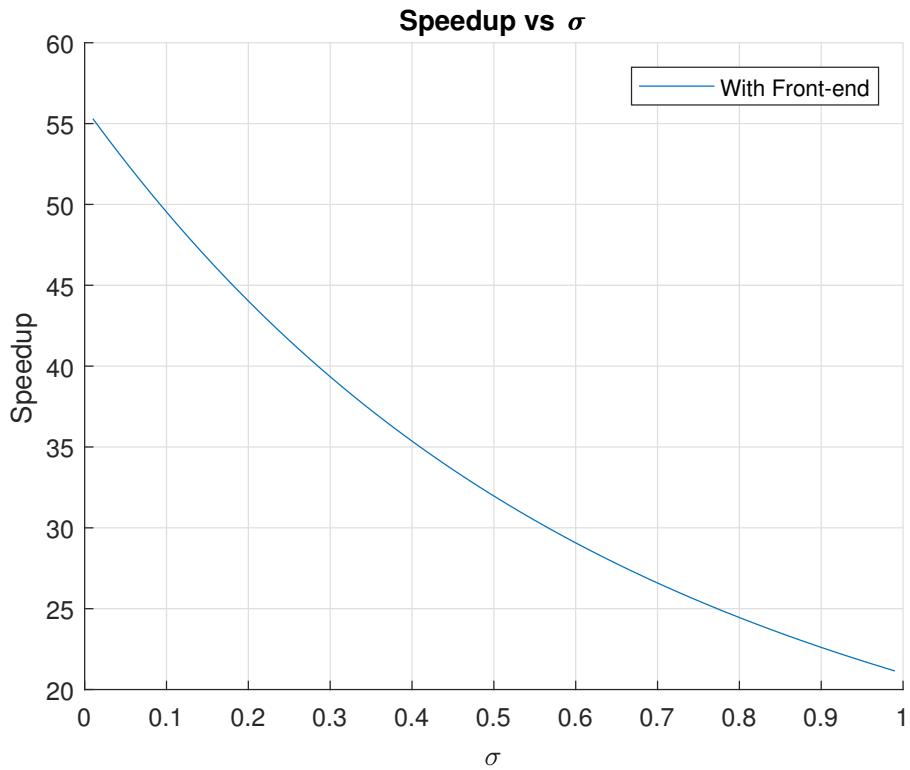


Figure 3.26: Speedup vs  $\sigma$

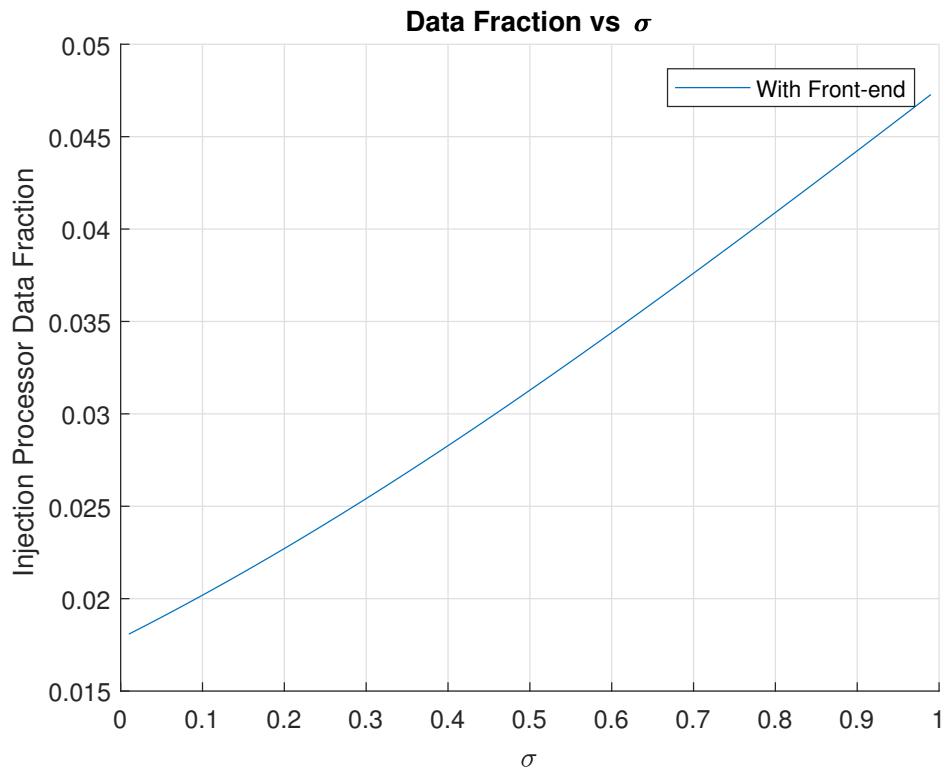


Figure 3.27: Injection Processor Data Fraction vs  $\sigma$

This simulation says the best performance happens on value  $\sigma \leq 0.05$ , which hits about 55 times speedup. If  $\sigma \approx 1$ , the network achieves almost 21 times performance. In addition, the data fraction in each injection processors are the same, that is, there is no need to transfer data between local processors.

## Situation II

The data injections don't consist of a subgraph of  $G$ . Jia [10] proposes an 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 [11] or Hadoop job, the total finish time depends on the last piece ending time-stamp.

Our objective is to propose a general algorithm framework to utilize less processors and achieve the same finish time.

Fig. 3.28 shows 10 Voronoi cells division and Fig. 3.29 figures out the speedup of each cells.

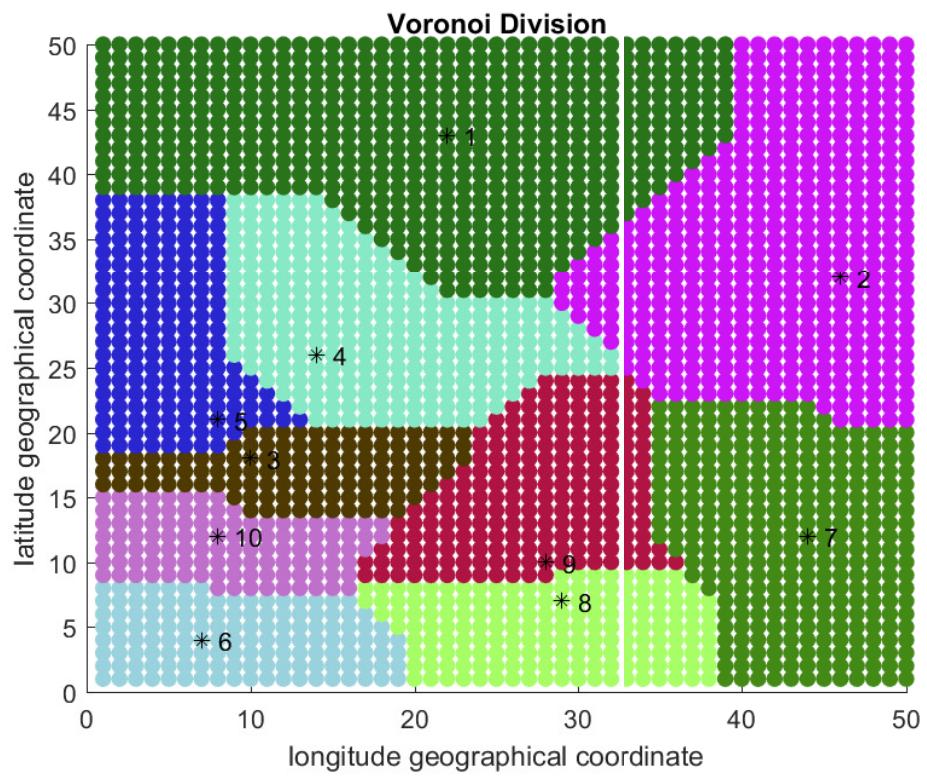


Figure 3.28: 10 Voronoi Cells

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

---

**Algorithm 2** Reduced Voronoi Diagram Algorithm(RVDA)

---

*global<sub>s</sub>* :

Define the cell's flow matrix column's number is *depth*.

Calculate *n* Voronoi cells with Manhattan distance.

Calculate *n* radius *R<sub>i</sub>* of *n* Voronoi cells.

Set *depth<sub>min</sub>* = min *R<sub>i</sub>*

**while**  $1 \leq i \leq n$  **do**

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

Calculate Voronoi cell's flow matrix *A<sub>i</sub>*.

$i = i + 1$

**end while**

*local<sub>s</sub>* :

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells' speedup curves

---

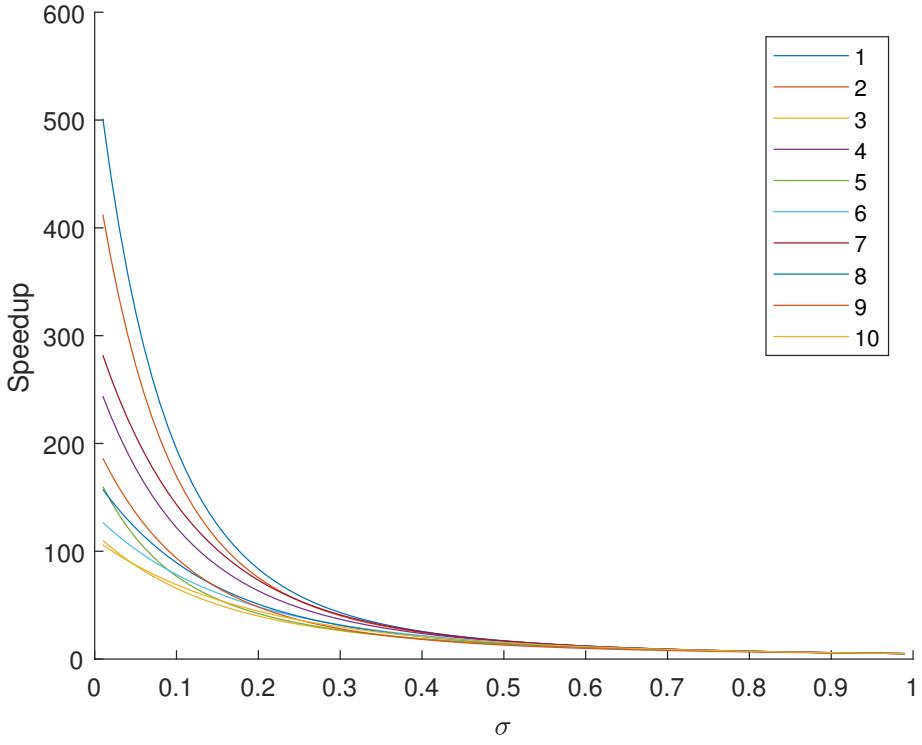


Figure 3.29: 10 Voronoi Cells speedup curves

Fig. 3.30 and Fig. 3.31 show the algorithm's result and speedup curves.

The algorithm obtains the same running time yet saves about 30% processors.

Fig. 3.29 shows  $\sigma < 0.2$ , the ratio  $\max\left(\frac{\text{maxspeedup}}{\text{minspeedup}}\right) = \frac{500}{100} = 5$  and Fig. 3.31 shows the ratio is  $\max\left(\frac{\text{maxspeedup}}{\text{minspeedup}}\right) = \frac{270}{100} = 2.7$ .

It displays that 10 pieces of processor cluster's equal computation is more balanced than initial setting, and the whole cluster finishes tasks with the same time by less processors.

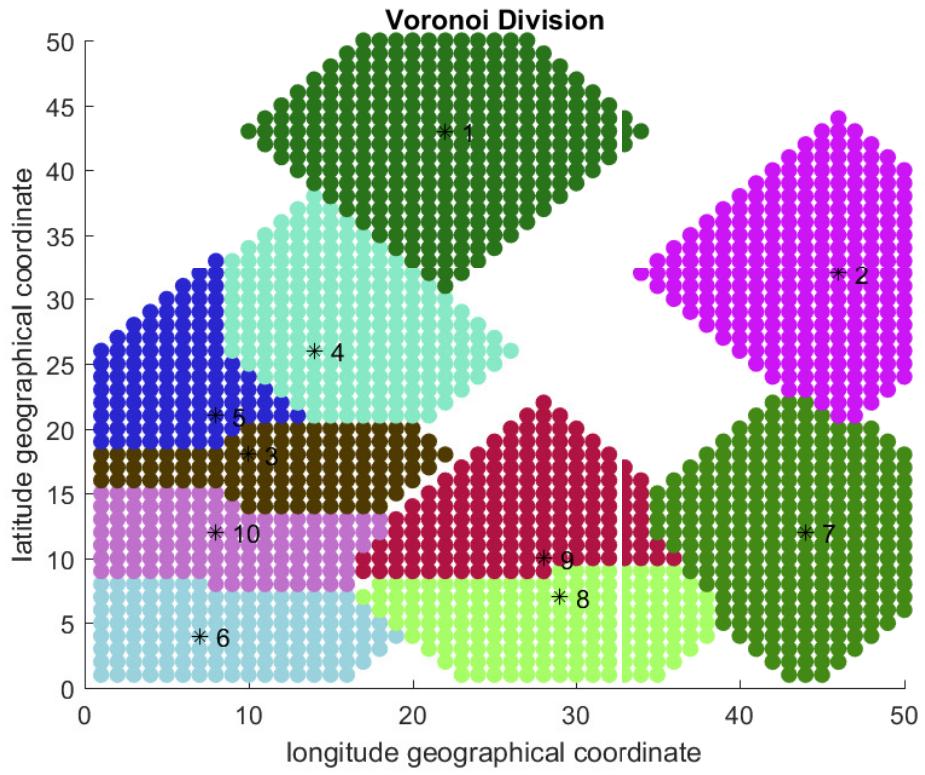


Figure 3.30: 10 reduced Voronoi cells

After 1000 round random sampling experiments, we obtain the average saved processors ratio in Fig. 3.32. From Fig. 3.32, it shows the average percentage of saved processor is about 35%.

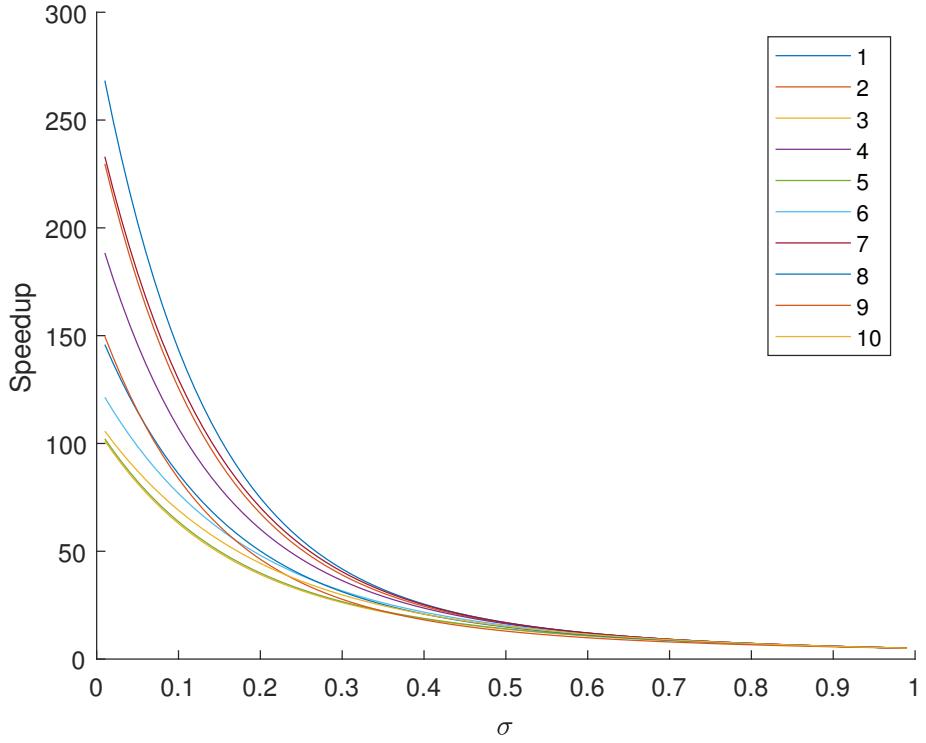


Figure 3.31: 10 reduced Voronoi cells's speedup curves

### Situation III

If there are some nodes consisting a subgraph and there are some nodes individual, our objective to finish the whole project in the same time and save processors and give a quantity analysis model to this data injection situation.

We process an improved RVDA algorithm to tackle this situation.

---

**Algorithm 3** Improved Reduced Voronoi Diagram Algorithm(IRVDA)

---

*global<sub>s</sub>* :

Define the cell's flow matrix column's number is *depth*.

Collapse the data injection processors into one “big” equivalent processor.

Calculate *k* constrained Voronoi cells[12] with Manhattan distance and get *k* cells.

Calculate *k* radius  $R_i$  of *k* Voronoi cells.

**while**  $1 \leq i \leq k$  **do**

Calculate the speedup  $Speedup_i$  with flow matrix  $A_i$ .

$i = i + 1$

**end while**

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

**while**  $1 \leq i \leq n$  **do**

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

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

$i = i + 1$

**end while**

*local<sub>s</sub>* :

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells' speedup curves

---

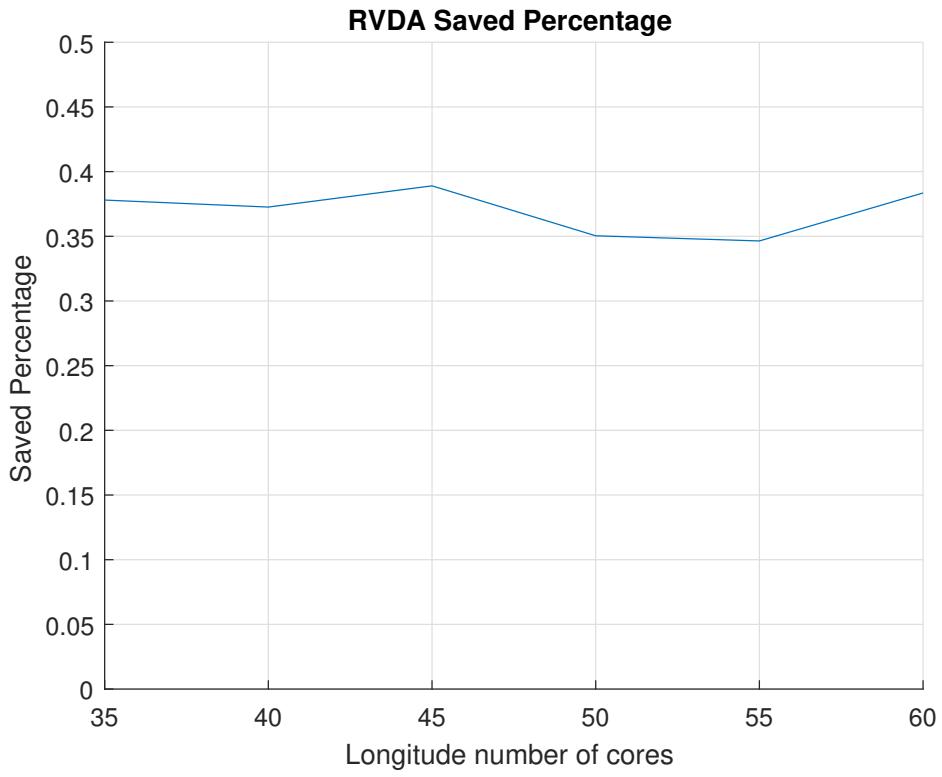


Figure 3.32: Reduced Voronoi Division Algorithm average processors' percentage

### 3.1.6 Multi-source Different Data Fraction

## 3.2 Without Front-end Scenario

In the without front-end scenario, the processors solely start to process as soon as each processor receives its entire load assignment[13].

This subsection concerns the processors without front-end. Because of without front-end, the processors simultaneously receive the data and solely start to process it as soon as each processor receives its entire load assignment.

We consider the timing diagram for Fig. 3.1, Fig. 3.4, Fig. 3.7 and so on. In addition, we also give the new closed-form matrix equations for the previous user cases.

The rule also plays a dominate role in establish the mathematics model.

### 3.2.1 Data Injection on The Corner Processor

#### 2\*2 Regular Network

The timing diagram of Fig. 3.1 is shown:

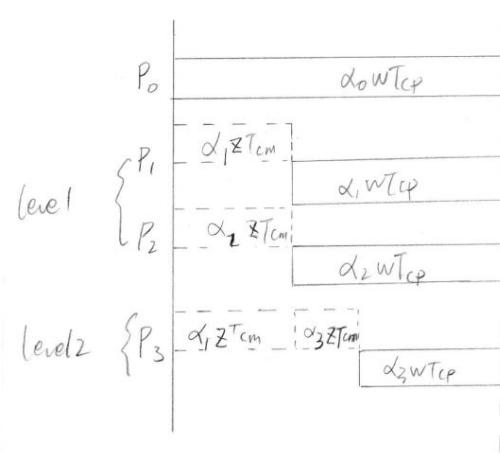


Figure 3.33: The timing diagram for 2\*2 regular network without front-end.

$P_0$  starts to process the assigned workload and it starts to transfer the  $\alpha_1$ ,  $\alpha_2$  and  $\alpha_3$  fraction workload after it totally receive its  $\alpha_0$  task. That is,  $P_1$  and  $P_2$  are idle until the  $L$  finish its data injection to  $P_0$ . The similar situation happens to  $P_1$  and  $P_2$  and they both start to transmit the  $\alpha_3$  after they totally receive the appropriate workload. In other words,  $P_3$  has to wait until the previous two layer processors obtain their own data.

The corresponding group of equations are as follows:

$$\left. \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 \end{array} \right\} \quad (3.89)$$

$$(3.90)$$

$$(3.91)$$

$$(3.92)$$

$$(3.93)$$

$$(3.94)$$

$$(3.95)$$

$$(3.96)$$

$$(3.97)$$

The matrix closed-form is presented as:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma + 1) & 0 \\ 1 & -\sigma & -(\sigma + 1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.98)$$

The explicit solution is:

$$\left\{ \begin{array}{l} \alpha_0 = \left( \frac{\sigma + 1}{\sigma + 2} \right)^2 \\ \alpha_1 = \frac{\sigma + 1}{(\sigma + 2)^2} \end{array} \right. \quad (3.99)$$

$$\left\{ \begin{array}{l} \alpha_0 = \left( \frac{\sigma + 1}{\sigma + 2} \right)^2 \\ \alpha_1 = \frac{\sigma + 1}{(\sigma + 2)^2} \\ \alpha_3 = \frac{1}{(\sigma + 2)^2} \end{array} \right. \quad (3.100)$$

$$\left\{ \begin{array}{l} \alpha_0 = \left( \frac{\sigma + 1}{\sigma + 2} \right)^2 \\ \alpha_1 = \frac{\sigma + 1}{(\sigma + 2)^2} \\ \alpha_3 = \frac{1}{(\sigma + 2)^2} \end{array} \right. \quad (3.101)$$

The simulation result for Fig. 3.1 is provided in Fig. 3.34:

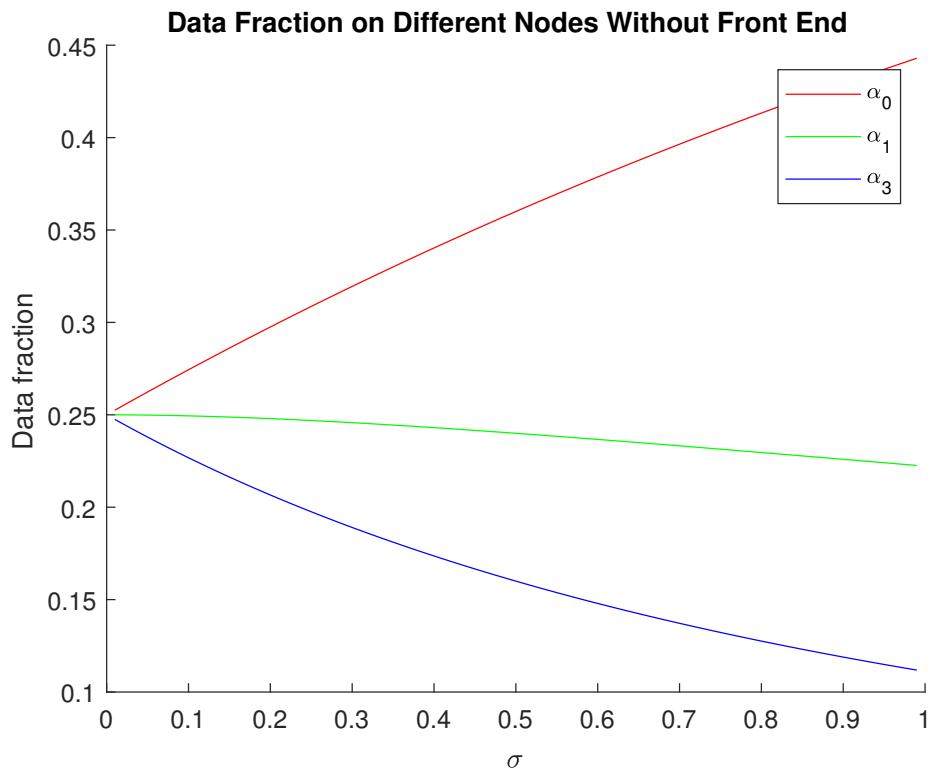


Figure 3.34: The data fraction deployed based on the radius value

Fig. 3.34 explains that as the value  $\sigma$  grows up, as 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 Regular Network

$P_0$  starts to process the assigned workload and it starts to transfer the  $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$  fraction workload after it totally receive its  $\alpha_0$  task. That is,  $P_1$  and  $P_2$  are idle until the  $L$  finishes its data injection to  $P_0$ . According to the  $level_1$ , the similar situation happens to  $P_1$  and  $P_2$  and they both start to transmit the  $\alpha_3$  after they totally receive the appropriate workload. In other words,  $P_3$  has to wait until the previous three layers,  $level_0$ ,  $level_1$  and  $level_2$  processors obtain their own data.

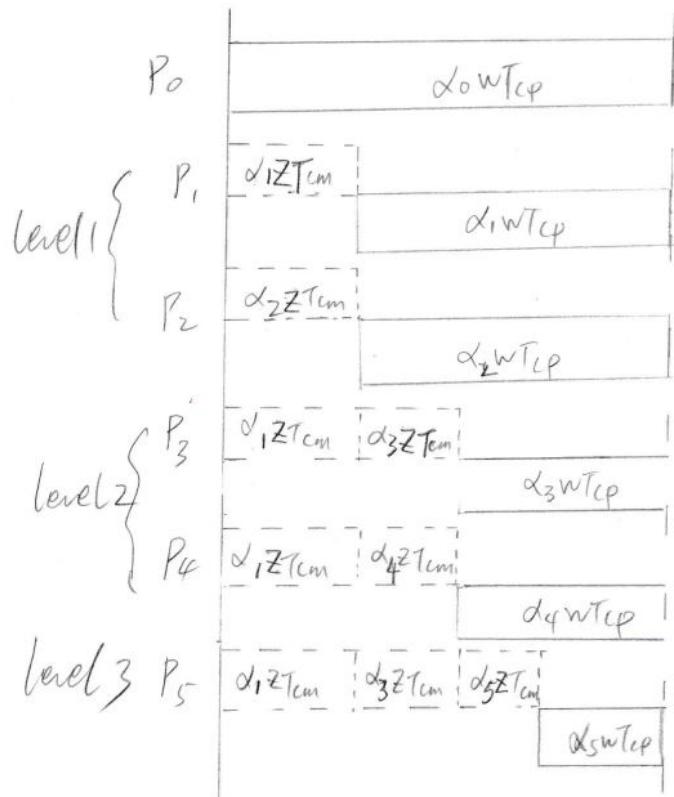


Figure 3.35: The timing diagram for 2\*3 regular network without front-end.

In addition, the group of equations are as follows:

$$\left. \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 < 1 \end{array} \right\} \quad (3.102)$$

$$\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} \quad (3.103)$$

$$\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} \quad (3.104)$$

$$(\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \quad (3.105)$$

$$(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \quad (3.106)$$

$$(\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \quad (3.107)$$

$$\sigma = \frac{z T_{cm}}{\omega T_{cp}} \quad (3.108)$$

$$\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 \quad (3.109)$$

$$0 < \alpha_0 \leq 1 \quad (3.110)$$

$$0 \leq \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 < 1 \quad (3.111)$$

The flow matrix is :

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma + 1) & 0 & 0 \\ 1 & -\sigma & -(\sigma + 1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma + 1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.112)$$

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:

$$\left\{ \begin{array}{l} \alpha_0 = \left( \frac{\sigma + 1}{\sigma + 2} \right)^3 \\ \alpha_1 = \frac{(\sigma + 1)^2}{(\sigma + 2)^3} \end{array} \right. \quad (3.113)$$

$$\left\{ \begin{array}{l} \alpha_3 = \frac{\sigma + 1}{(\sigma + 2)^3} \\ \alpha_5 = \frac{1}{(\sigma + 2)^3} \end{array} \right. \quad (3.114)$$

$$\left\{ \begin{array}{l} \alpha_3 = \frac{\sigma + 1}{(\sigma + 2)^3} \\ \alpha_5 = \frac{1}{(\sigma + 2)^3} \end{array} \right. \quad (3.115)$$

$$\left\{ \begin{array}{l} \alpha_5 = \frac{1}{(\sigma + 2)^3} \end{array} \right. \quad (3.116)$$

The simulation result is:

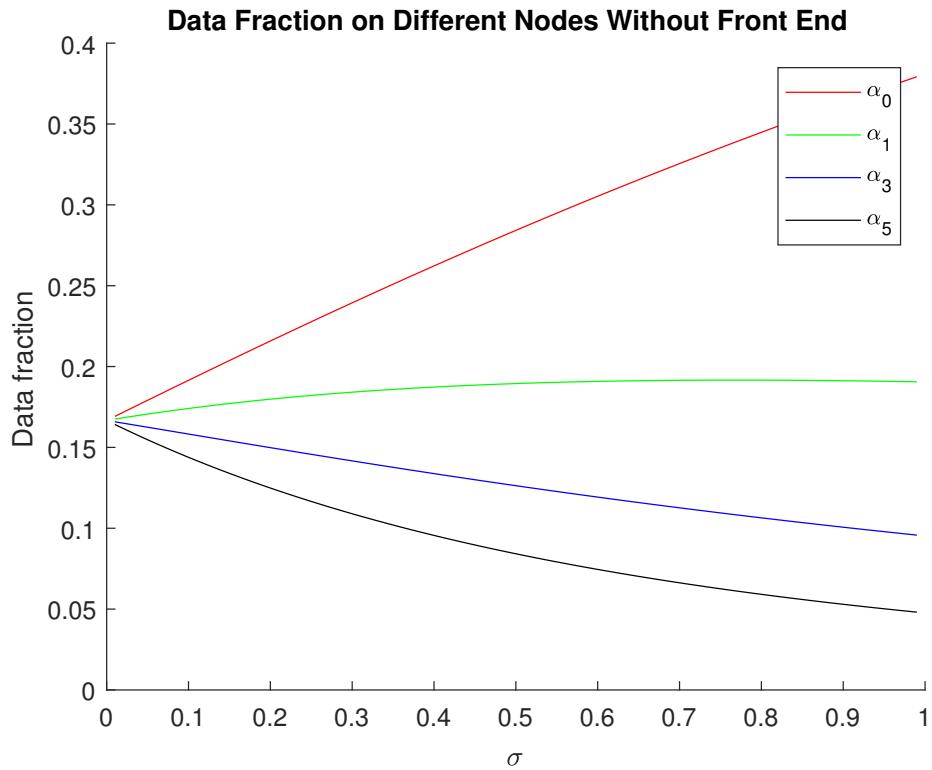


Figure 3.36: The data fraction deployed based on the radius value

## 2\*n Regular Network

Considering Fig. 3.7, the equations are demonstrated as follows:

$$\left\{
 \begin{array}{ll}
 \alpha_0 \omega T_{cp} = T_{f,m} & (3.117) \\
 \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (3.118) \\
 \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (3.119) \\
 (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (3.120) \\
 (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (3.121) \\
 (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (3.122) \\
 \vdots & (3.123) \\
 (\alpha_1 + \alpha_3 + \cdots + \alpha_{2 \times n - 1}) z T_{cm} + \alpha_{2 \times n - 1} \omega T_{cp} = T_{f,m} & (3.124) \\
 \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (3.125) \\
 0 < \sigma < 1 & (3.126) \\
 0 < \alpha_0 \leq 1 & (3.127) \\
 0 \leq \alpha_1 \alpha_3 \cdots \alpha_{2 \times n - 1} < 1 & (3.128)
 \end{array}
 \right.$$

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

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & \sigma^* & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^* & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^* & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^* & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & \sigma^* \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2 \times n-3} \\ \alpha_{2 \times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (3.129)$$

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

$$\left\{ \alpha_i = \left| \frac{\det A_i^*}{\det A} \right| \right. \quad (3.130)$$

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

We use  $-\sigma - 2 = \epsilon$  and  $\sigma^* - 2 = \beta$ . After a series of column reduction and row reduction, the flow matrix changes as follows :

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

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

We define

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

$0 < \sigma < 1$ , then  $-2 < \sigma^* < -1$ , which means  $C$  is column linear independent, after column and row reduction.

Further, we define

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

$C^*$  is full rank.

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

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

## **m\*n Regular Network**

Referring to Fig. 2.1, we utilize  $\sigma^*$  to present the  $-(\sigma + 1)$ . The matrix closed-form is:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & \sigma^* & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^* & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^* & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^* & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^* & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^* & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^* & 0 \\ 1 & -\sigma & \sigma^* \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.131)$$

### 3.2.2 Data Injection on The Boundary Processor

Fig. 3.10 shows an example of boundary processor  $P_0$  receiving  $L$ . The timing diagram for Fig. 3.10 is Fig. 3.37.

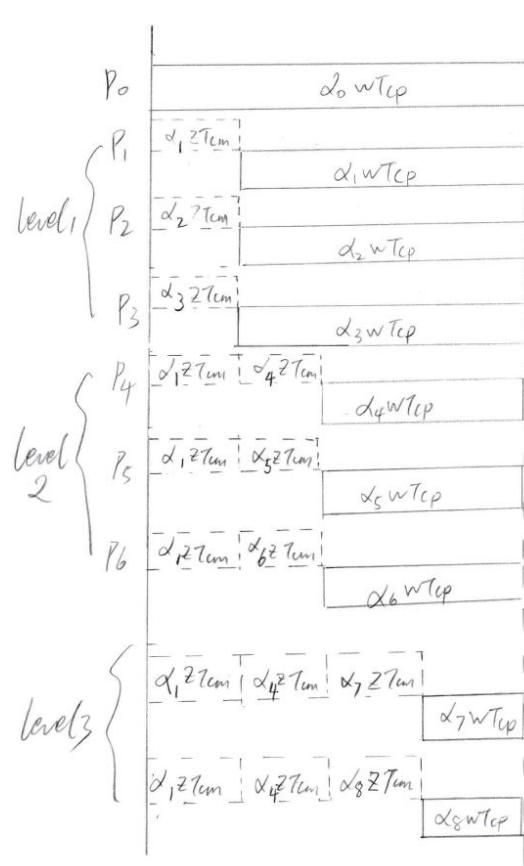


Figure 3.37: The timing diagram for  $3 \times 3$  boundary data injection on  $P_0$

The equations are:

$$\left\{ \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} \\ \alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \\ (\alpha_2 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \\ (\alpha_3 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_4 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1 \alpha_3 \alpha_4 \alpha_5 \alpha_6 \alpha_7 \alpha_8 < 1 \end{array} \right. \quad \begin{array}{l} (3.132) \\ (3.133) \\ (3.134) \\ (3.135) \\ (3.136) \\ (3.137) \\ (3.138) \\ (3.139) \\ (3.140) \\ (3.141) \\ (3.142) \\ (3.143) \\ (3.144) \end{array}$$

The flow matrix is :

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 1 & -(\sigma + 1) & 0 & 0 \\ 1 & -\sigma & -(\sigma + 1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma + 1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_4 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.145)$$

The simulation result is shown in :

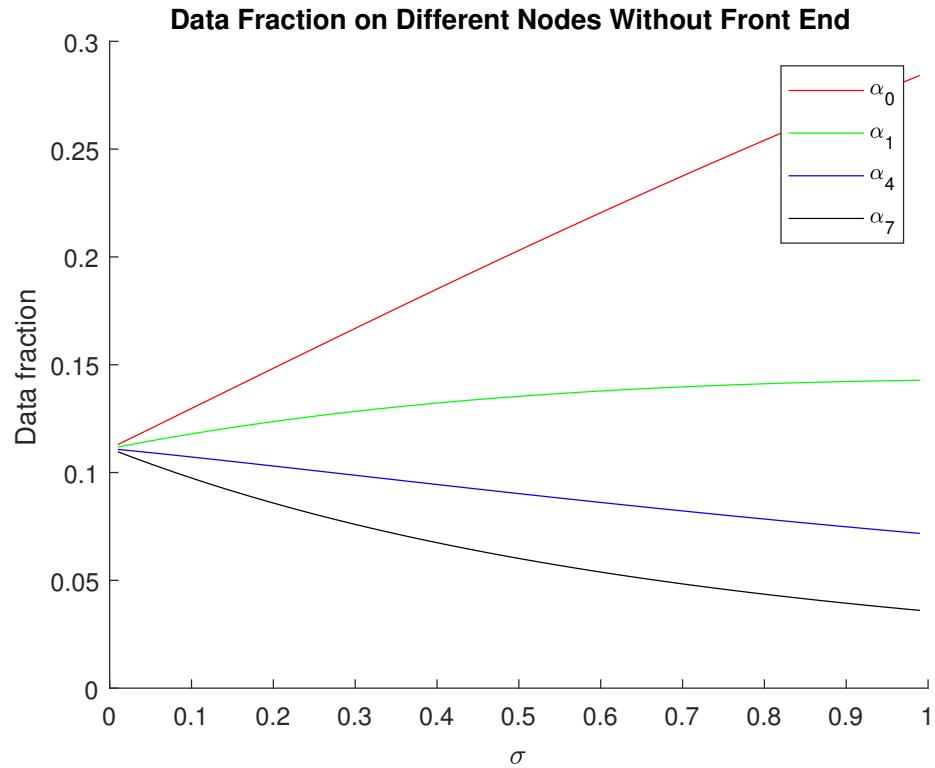


Figure 3.38: The fraction curve for 3\*3 boundary data injection on  $P_0$

### 3.2.3 Data Injection on The Inner Grid Processor

The equations are:

$$\left\{ \begin{array}{l} \alpha_0 \omega T_{cp} = T_{f,m} \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} \\ \alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} \\ \alpha_4 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} \\ (\alpha_1 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} \\ (\alpha_2 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} \\ (\alpha_3 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} \\ (\alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} \\ 0 < \sigma < 1 \\ 0 < \alpha_0 \leq 1 \\ 0 \leq \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 \end{array} \right. \begin{array}{l} (3.146) \\ (3.147) \\ (3.148) \\ (3.149) \\ (3.150) \\ (3.151) \\ (3.152) \\ (3.153) \\ (3.154) \\ (3.155) \\ (3.156) \\ (3.157) \\ (3.158) \end{array}$$

The flow matrix closed-form is:

$$\begin{bmatrix} 1 & 4 & 4 \\ 1 & -(\sigma + 1) & 0 \\ 1 & -\sigma & -(\sigma + 1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (3.159)$$

The simulation result shows:

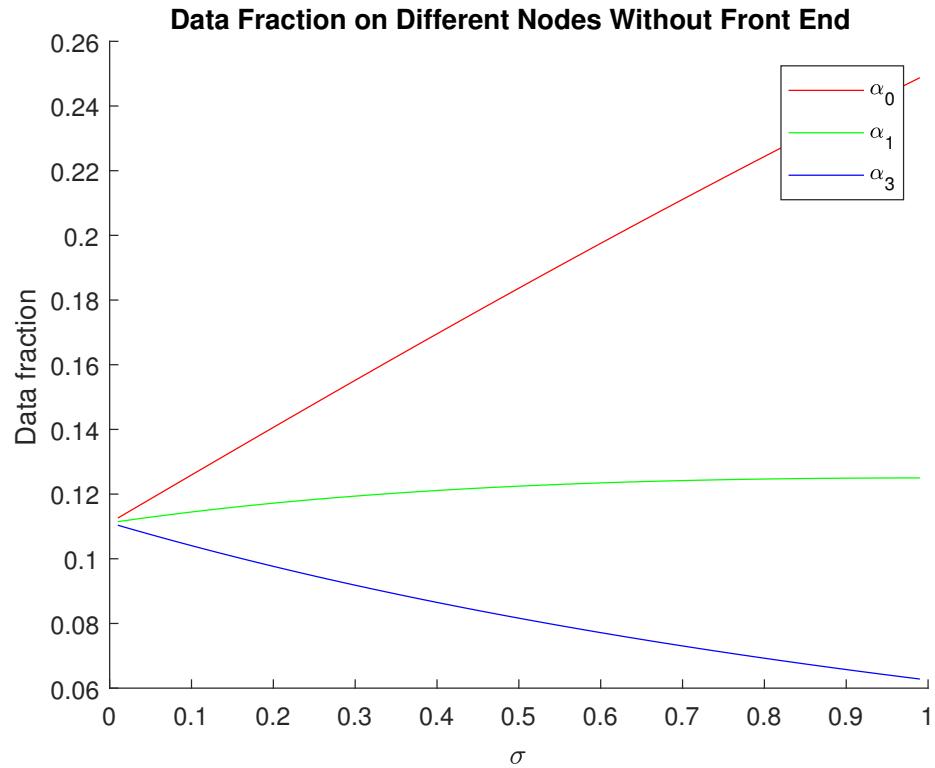


Figure 3.39: The timing diagram for 3\*3 inner grid injection  $P_0$

### 3.2.4 Sensitivity Analysis Without Front-end Processors

#### Data Injection on The Corner Processor

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

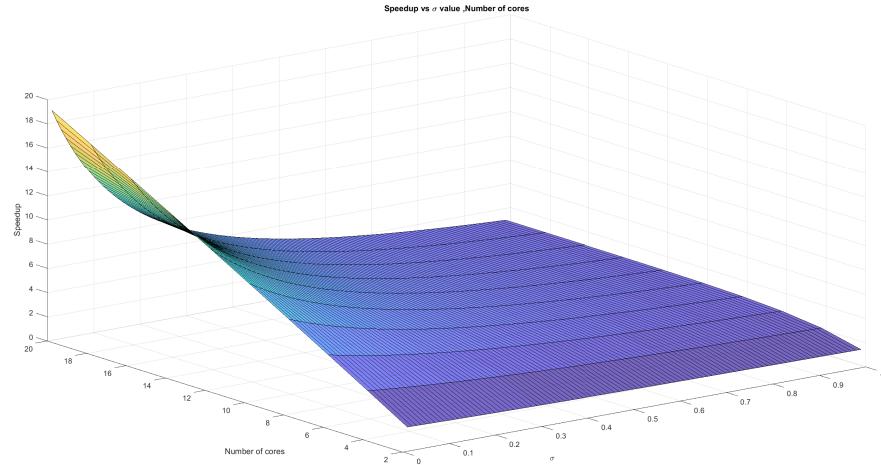


Figure 3.40: Sensitivity analysis result of  $2*10$  regular network result

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 number of cores dominate the efficiency. If the  $\sigma > 0.2$ , the efficiency drops dramatically. That is, the  $\sigma$  value plays more critical role in the speedup simulation. This important investigation benefit the multi-source assignment problem. In addition, if the number of cores is bigger than 4, the bottom speedup effect is about 3 time.

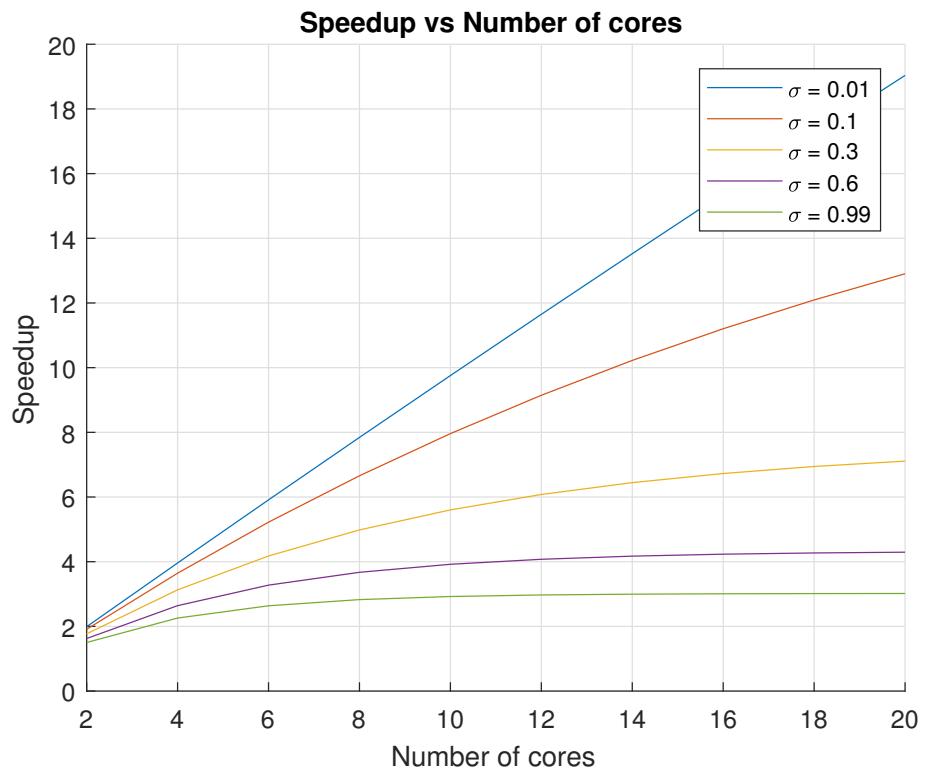


Figure 3.41: Speedup curve of  $2 \times 10$  regular network result

### Data Injection on The Boundary Processor

Fig. 3.42 and Fig. 3.43 tell the speedup efficiency for the data injection on the boundary.

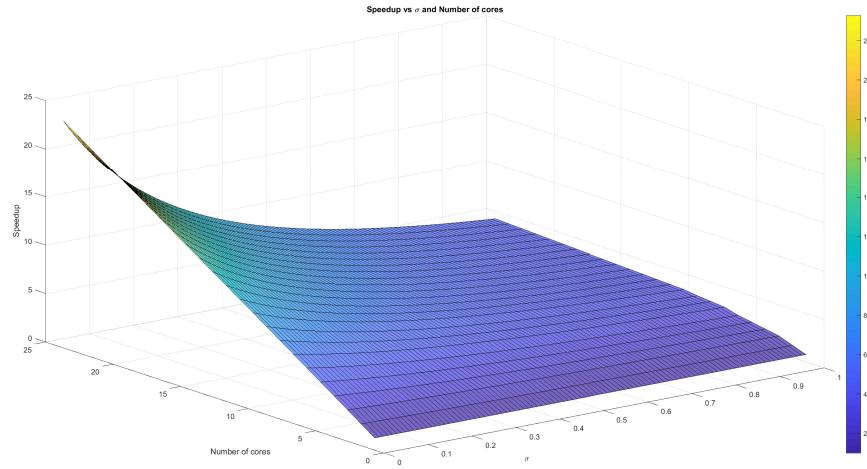


Figure 3.42: Sensitivity analysis result of  $3 \times 8$  regular network result

### Data Injection on The Inner Grid Processor

Fig. 3.44 and Fig. 3.45 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, which happens on the  $\sigma < 0.1$ .

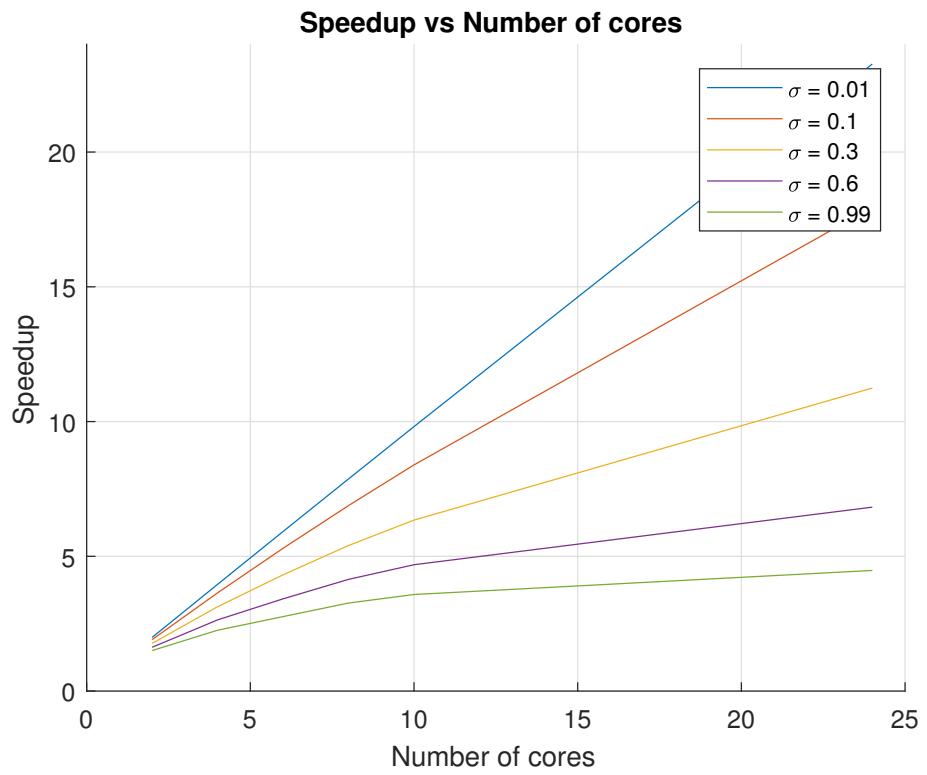


Figure 3.43: Speedup curve of 3\*8 regular network result

### 3.2.5 Multi-source Even Data Fraction

### 3.2.6 Multi-source Different Data Fraction

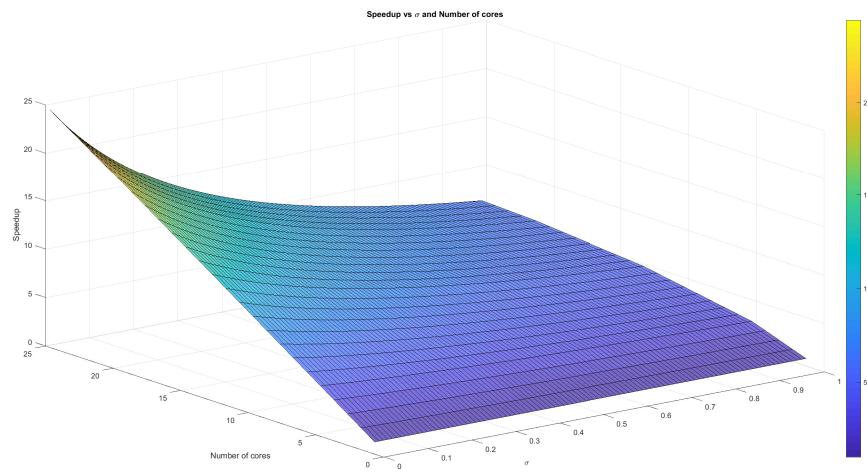


Figure 3.44: Sensitivity analysis result of data injection position on inner grid processor

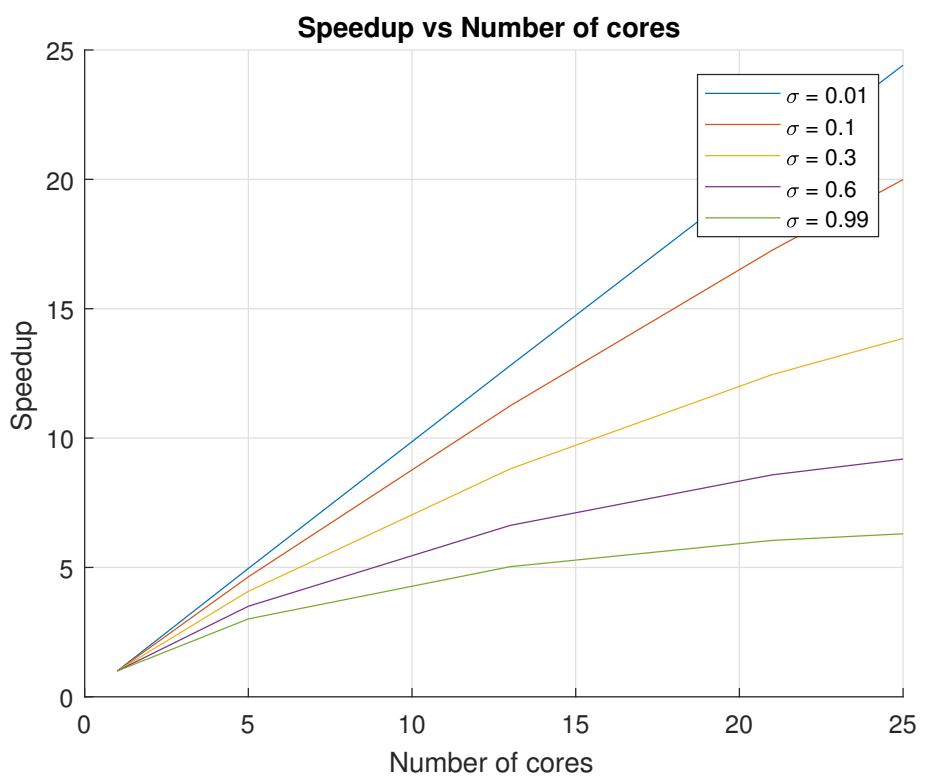


Figure 3.45: Speedup curve of 5\*5 regular network result

# 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[14], there are three different torus mesh. 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.

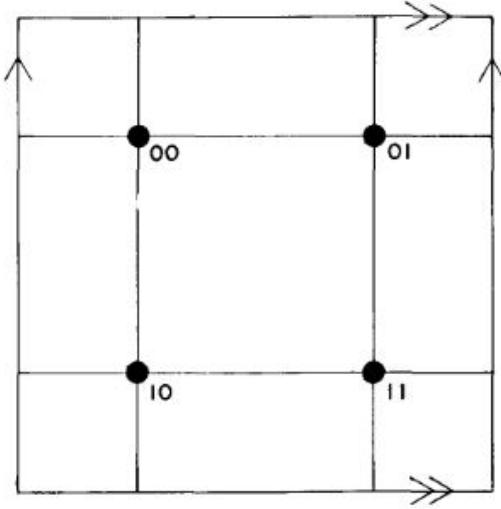


Figure 4.1: The rectangular toroidal network

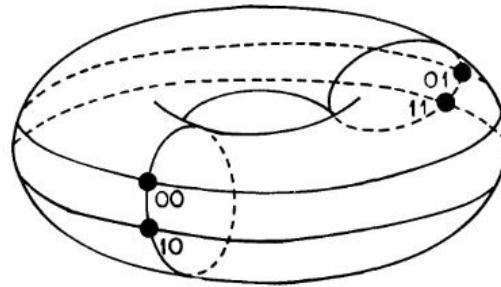


Figure 4.2: The rectangular toroidal network

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

$$\begin{cases} D_k = D_{k,x} + D_{k,y} \end{cases} \quad (4.1)$$

$$\begin{cases} D_{k,x} = \min\{\|D_{k,x} - L_x\|, m - \|D_{k,x} - L_x\|\} \end{cases} \quad (4.2)$$

$$\begin{cases} D_{k,y} = \min\{\|D_{k,y} - L_y\|, n - \|D_{k,y} - L_y\|\} \end{cases} \quad (4.3)$$

## 4.2 With Front-end Scenario

### 4.2.1 Data Injection On The Grid Processor

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.

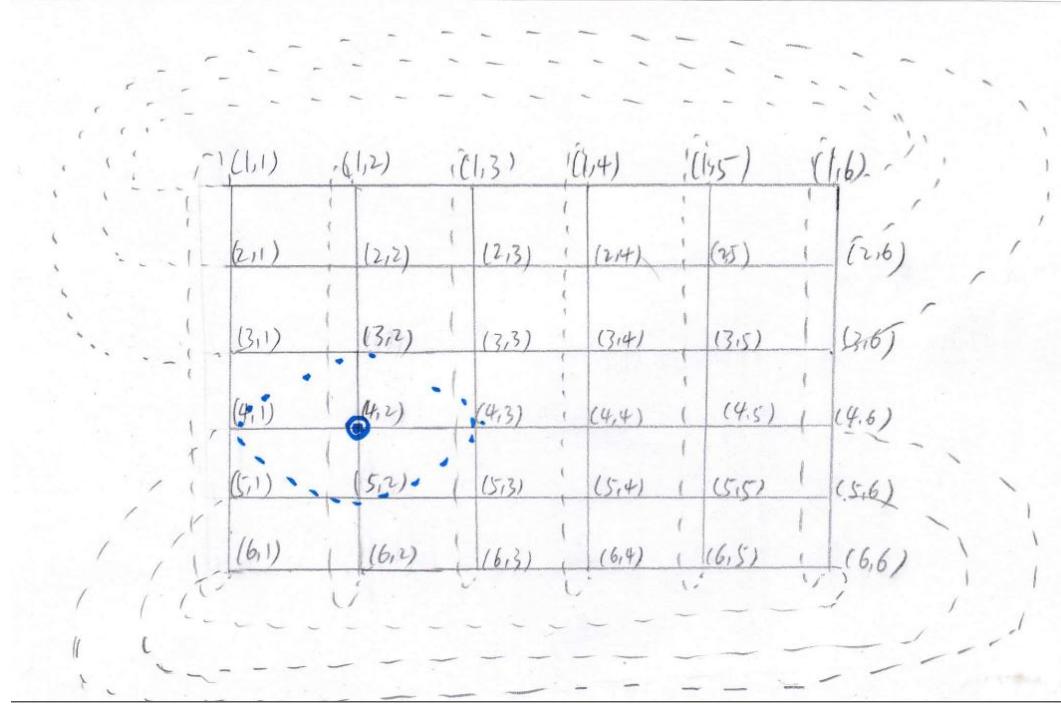


Figure 4.3: The  $m^*n$  toroidal rectangle network and the data injection is  $P_{4,2}$

The  $D_{k,i}$  table is as follow in table Table 4.1

The flow matrix closed-form is

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

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

Table 4.1:  $D_i$  vs Number

The simulation result is :

From Fig. 4.4, we see that as the value  $\sigma$  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.

The figure illustrates that if  $\sigma < 0.3$ , the number of processors grow up, the speedup efficiency is likely linear increasing. Alternatively speaking, if  $\sigma < 0.3$ , the number of cores dominate the efficiency. If the  $\sigma > 0.3$ , the efficiency drops dramatically. That is, the  $\sigma$  value plays more critical role in the speedup simulation. This important investigation benefit the multi-source assignment problem. In addition, if the number of cores is bigger than 4, the bottom speedup effect is about 3 time.

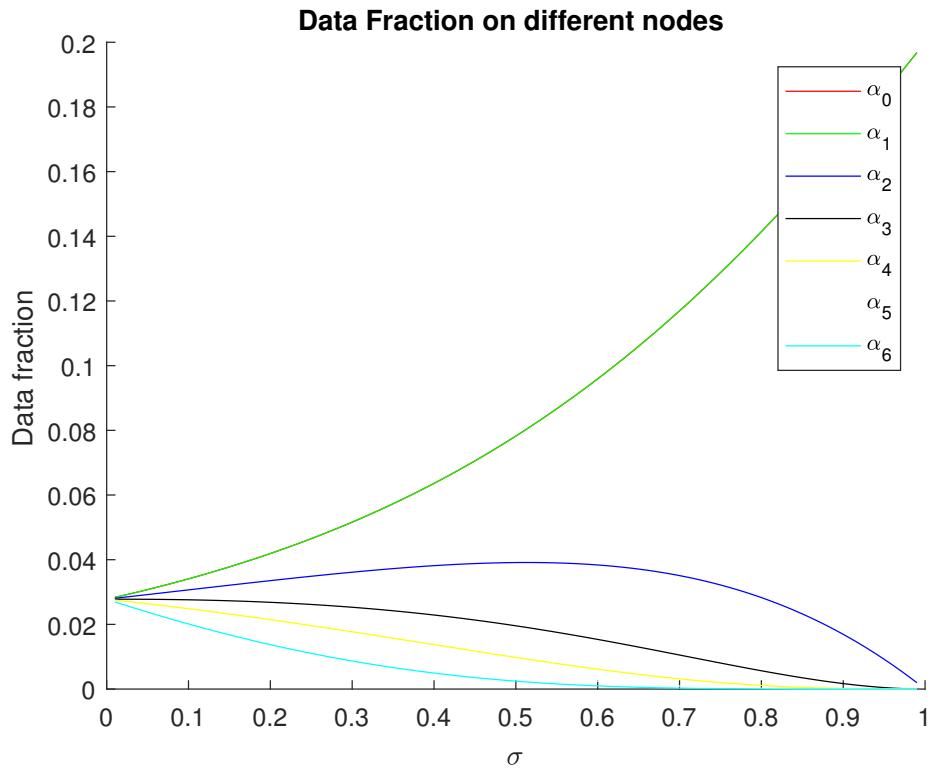


Figure 4.4: The data fraction curve of Fig. 4.3

#### 4.2.2 Sensitivity Analysis of Toroidal Rectangle Network

Considering a  $5 \times 5$  toroidal rectangle network, the  $level_i$  table shows Table 4.2:

So the simulation result illustrates in Fig. 4.5

#### 4.2.3 Multi-source Even Data Injection

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

Table 4.2: The processor number of various  $D_i$

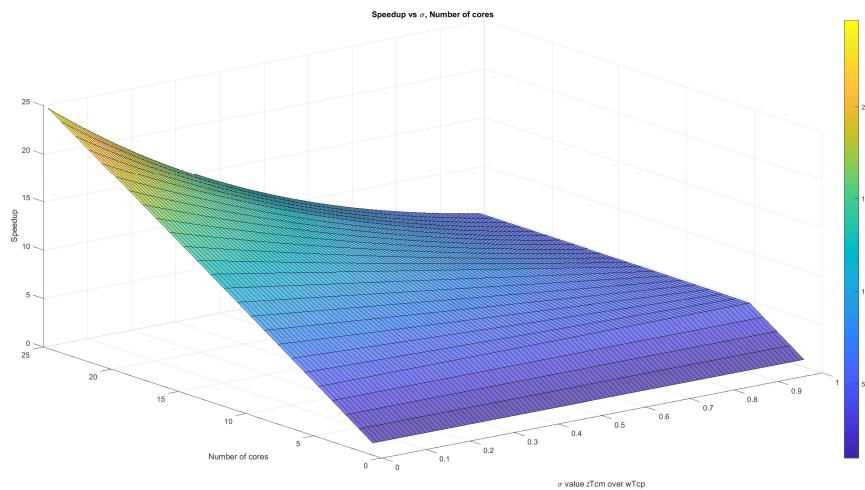


Figure 4.5: Sensitivity analysis result of data injection position on inner grid processor

Fig. 4.6 [15] provides a torus Voronoi method, which extends the original domain to 8 copy and calculate the Voronoi Diagram as planner algorithm [16]. Then the corner part is the torus Voronoi diagram.

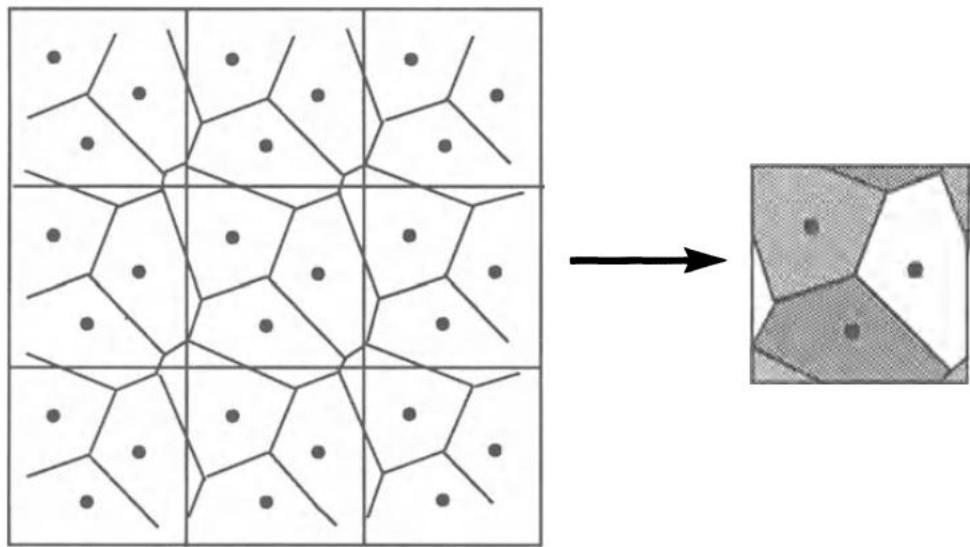


Figure 4.6: How to calculate torus Voronoi Diagram

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

#### 4.2.4 Multi-source Different Data Injection

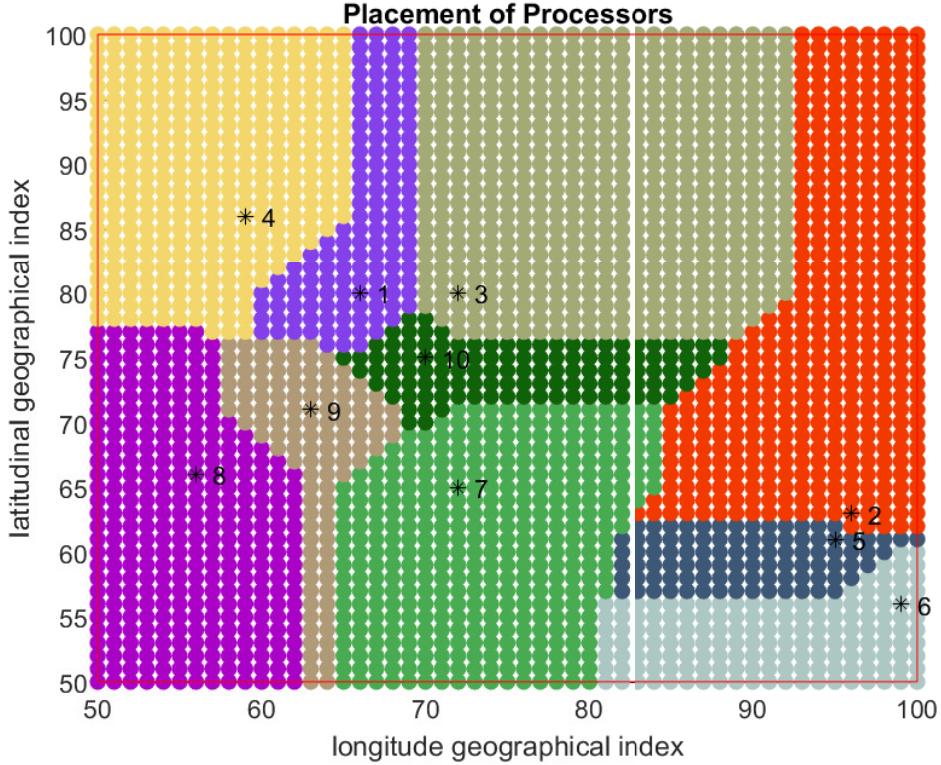


Figure 4.7: Initial Voronoi Diagram

### 4.3 Without Front-end Scenario

#### 4.3.1 Data Injection On The Grid Processor

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

$$\begin{bmatrix}
 1 & 4 & 8 & 10 & 8 & 4 & 1 \\
 1 & \sigma^* & 0 & 0 & 0 & 0 & 0 \\
 1 & -\sigma & \sigma^* & 0 & 0 & 0 & 0 \\
 1 & -\sigma & -\sigma & -\sigma^* & 0 & 0 & 0 \\
 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^* & 0 \\
 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^* & 0 \\
 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^*
 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.5)$$

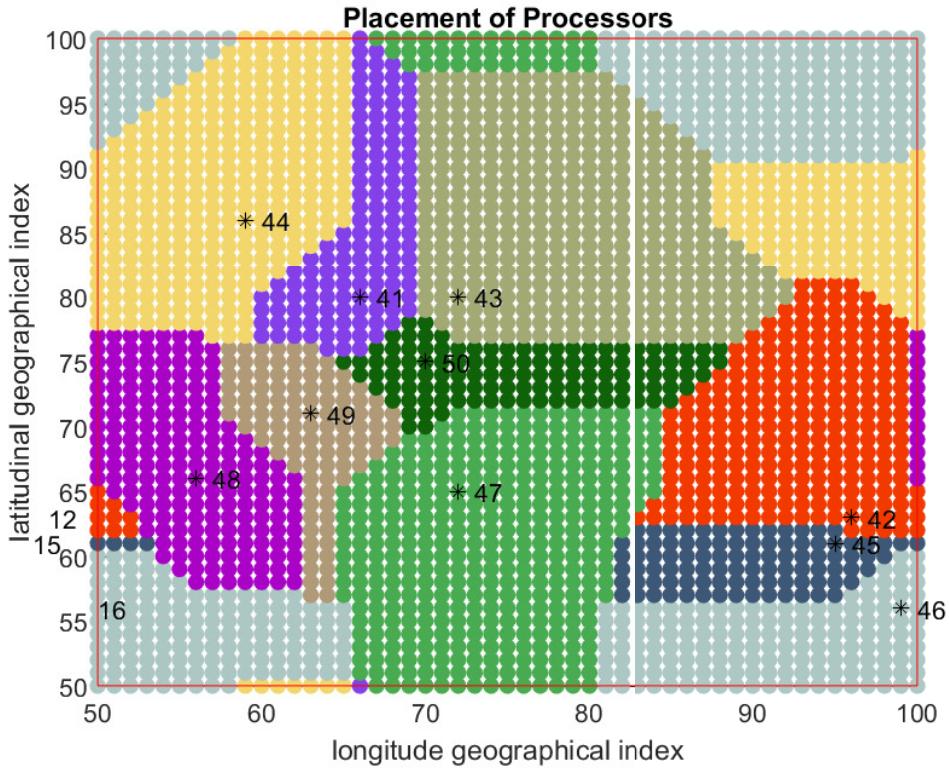


Figure 4.8: Torus Voronoi Diagram

The fraction curve result is :

From Fig. 4.13, we see that as the value  $\sigma$  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.

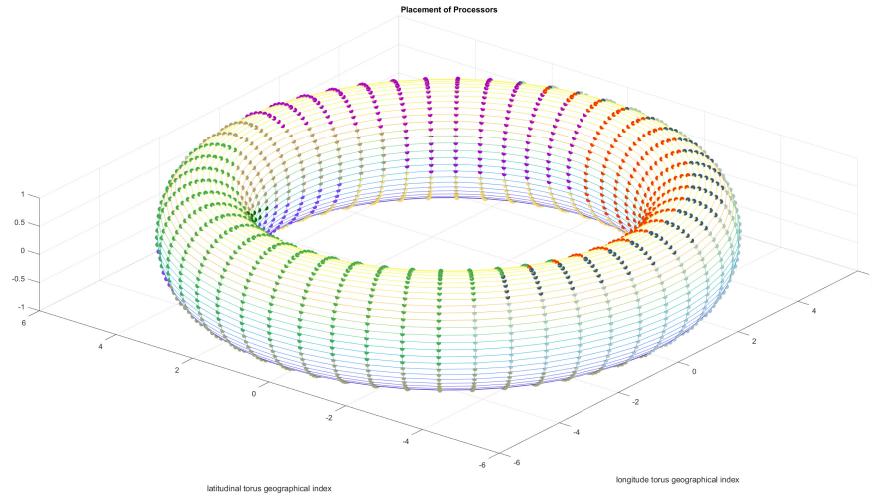


Figure 4.9: Voronoi Diagram Casting to the torus model

#### 4.3.2 Sensitivity Analysis of Toroidal Rectangle Network

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

So the simulation result illustrates in Fig. 4.14

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

Table 4.3: The processor number of various  $D_i$

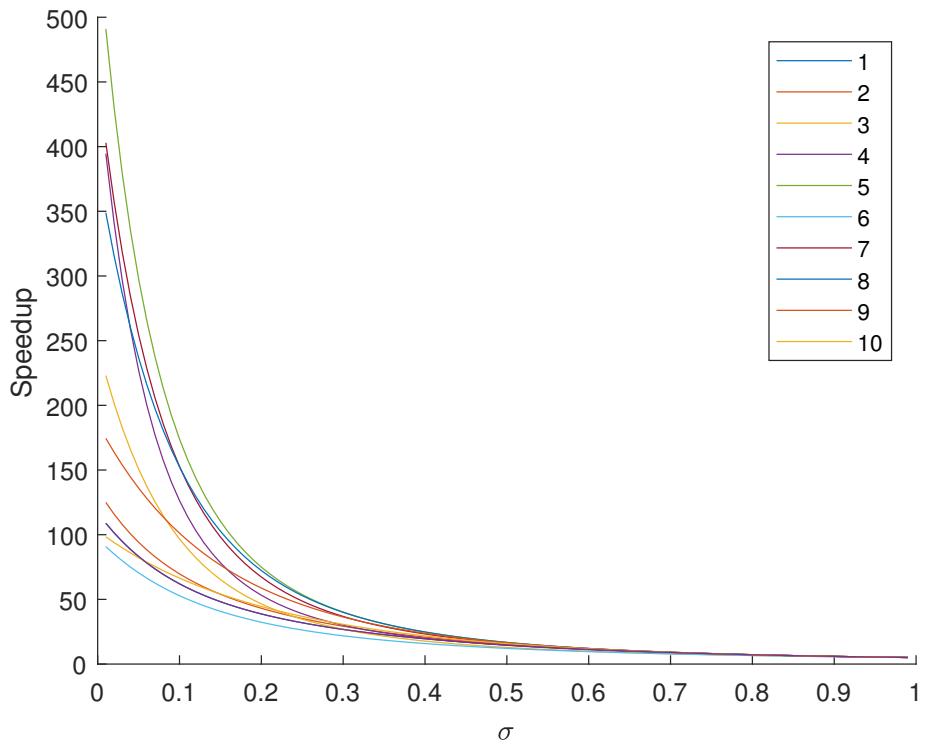


Figure 4.10: Voronoi Diagram Casting to the torus model

#### 4.3.3 Multi-source Even Data Injection

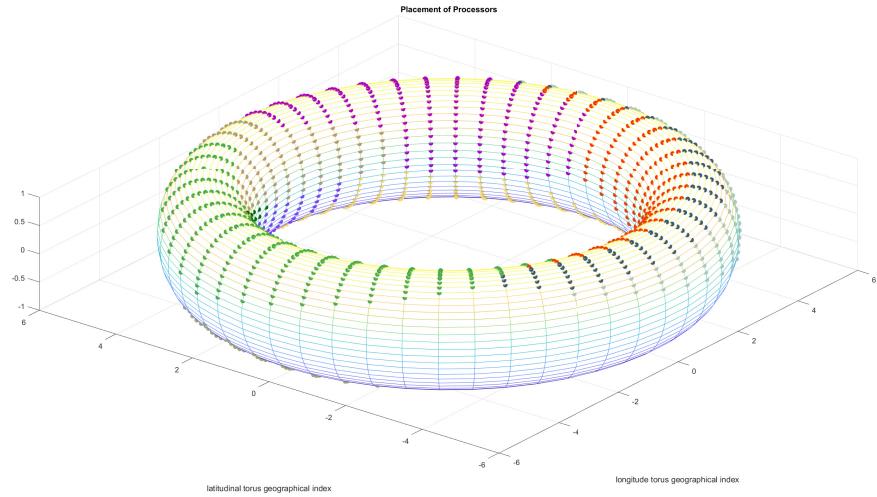


Figure 4.11: Torus Reduced Voronoi Diagram Casting to the Torus Model

#### 4.3.4 Multi-source Different Data Injection

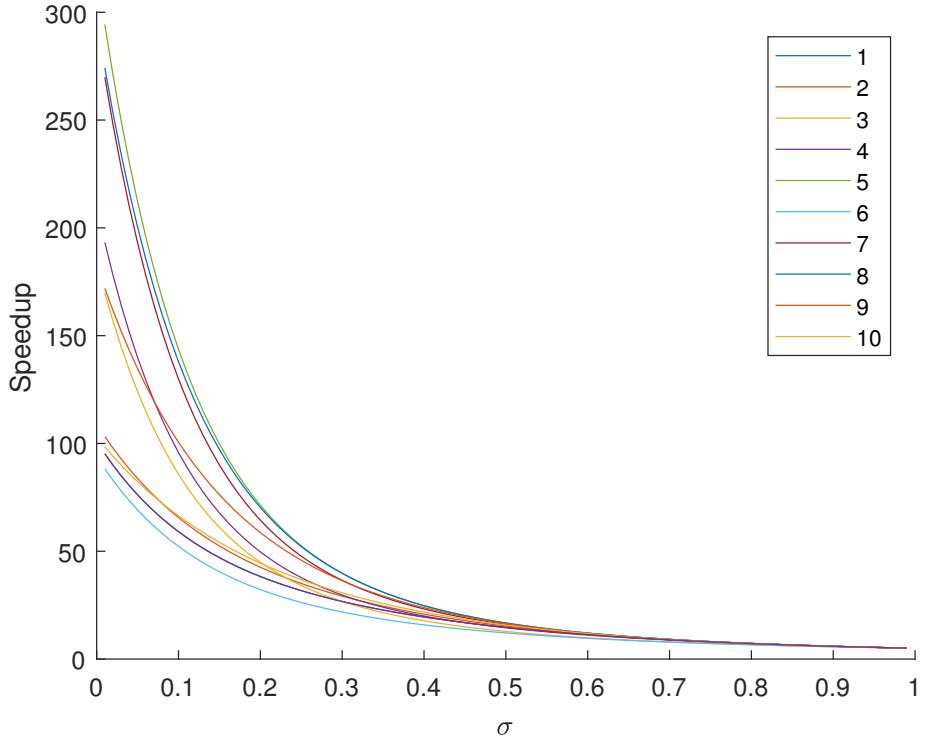


Figure 4.12: Torus Reduced Voronoi Diagram

## 4.4 Comparison Result

### 4.4.1 Comparison Result Between Regular Mesh and Toroidal With Same Processors Number

Considering a regular mesh 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  $5 \times 5$  regular mesh, each position's efficiency is equal. Fig. 4.15 explores the comparison result between the toroidal and corner scenario difference.

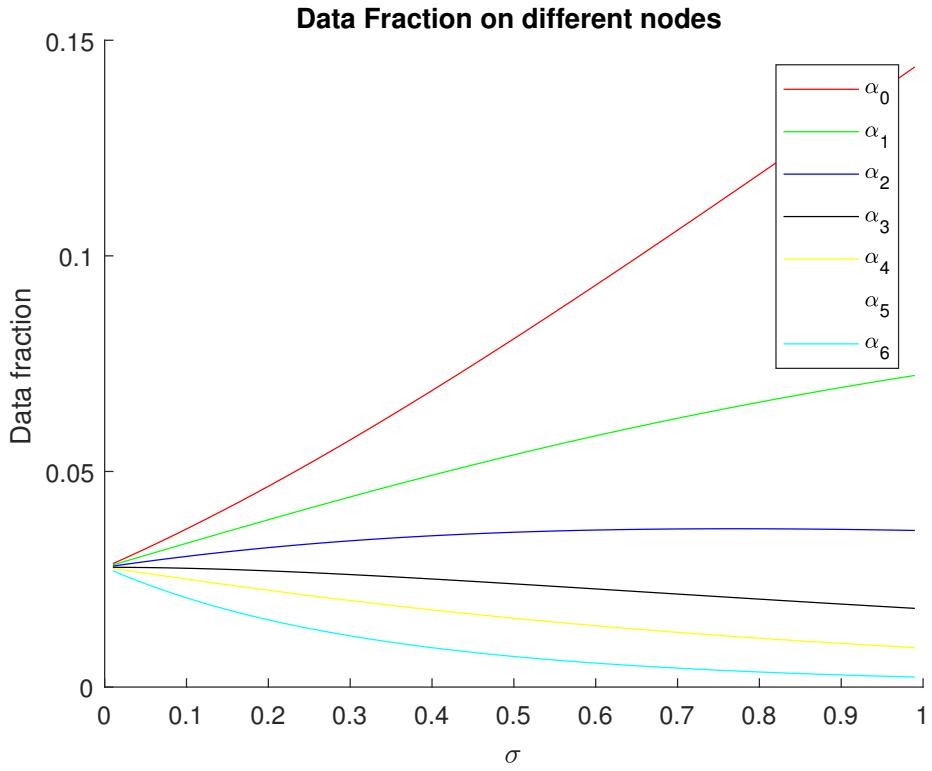


Figure 4.13: The data fraction curve of Fig. 4.3

#### 4.4.2 Comparison Result With Corner Processor and Inner Grid Processor

For a  $5 * 5$  regular network, the inner grid position is  $P_{12}$  and the corner data injection position is  $P_0$ . The comparing result is Fig. 4.15.

Generally speaking, Fig. 4.15 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.

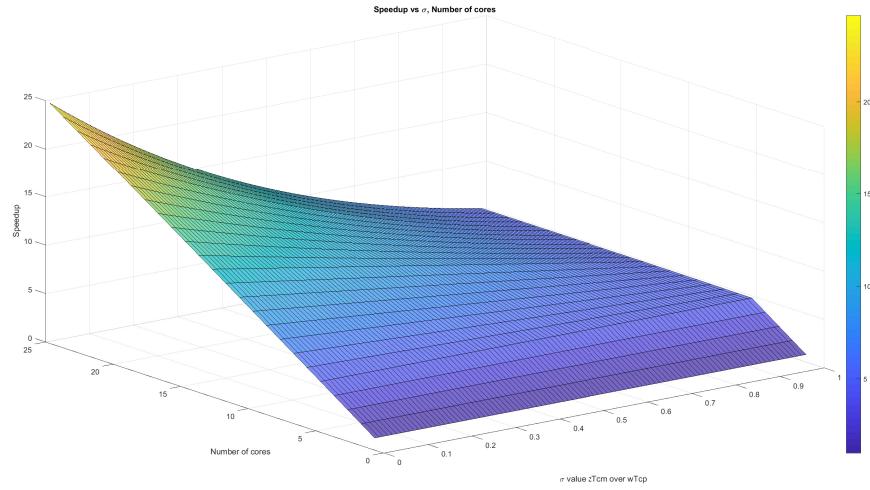


Figure 4.14: Sensitivity analysis result of data injection position on inner grid processor

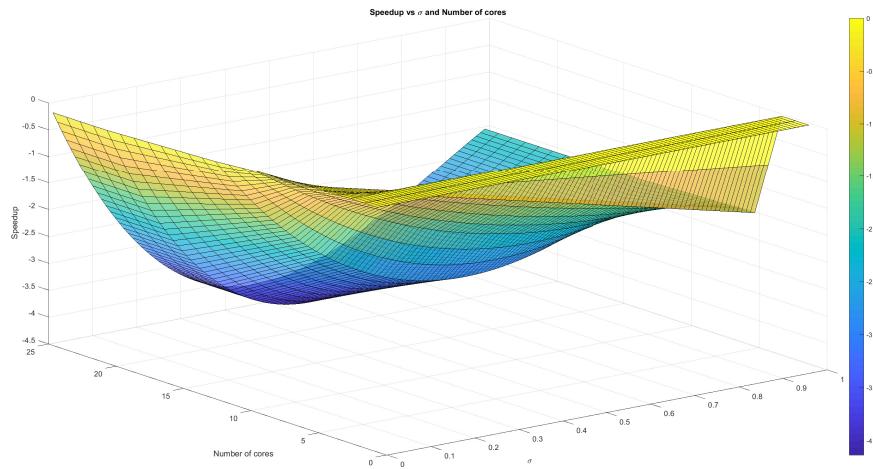


Figure 4.15: Speedup difference between corner injection and inner grid injection

### 4.4.3 Comparison Result Between Front-end Processor and Without Front-end Processor

In the legend of figures, we use

- $F$  presents the processors are with front-end situation.
- $NF$  presents the processors are without front-end situation.
- $F\alpha_0$  means the  $\alpha_0$  data fraction deployed to  $P_0$ , if the processor has front-end.
- $NF\alpha_0$  means the  $\alpha_0$  data fraction deployed to  $P_0$ , if the processor is without front-end setting.

#### Data Injection On the Corner Processor

Fig. 4.16 says that  $P_0$  takes more assigned task in without front-end scenario than front-end processor situation. As the value  $\sigma$  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 front-end assumption, yet in the without front-end situation, there is still about 10% data fraction are communicated to  $P_3$ .

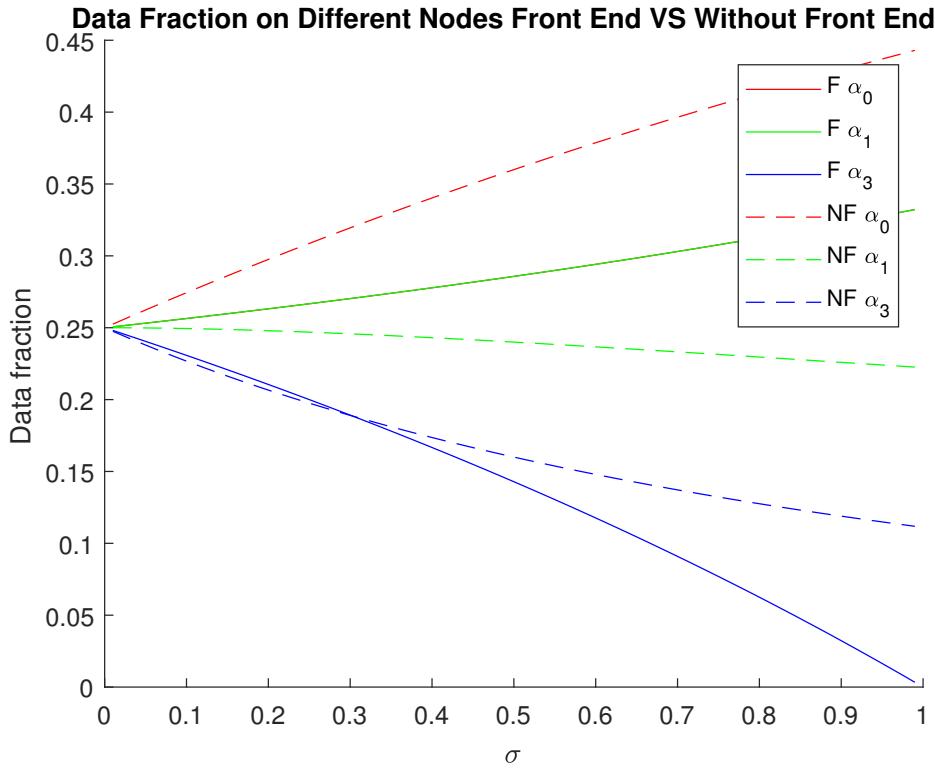


Figure 4.16: The comparing result between front-end processor with without front-end processor in  $2^*2$  regular network

Fig. 4.17 says that  $P_0$  takes more assigned task in without front-end scenario than front-end processor situation. As the  $\sigma$  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 front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to  $P_5$ .

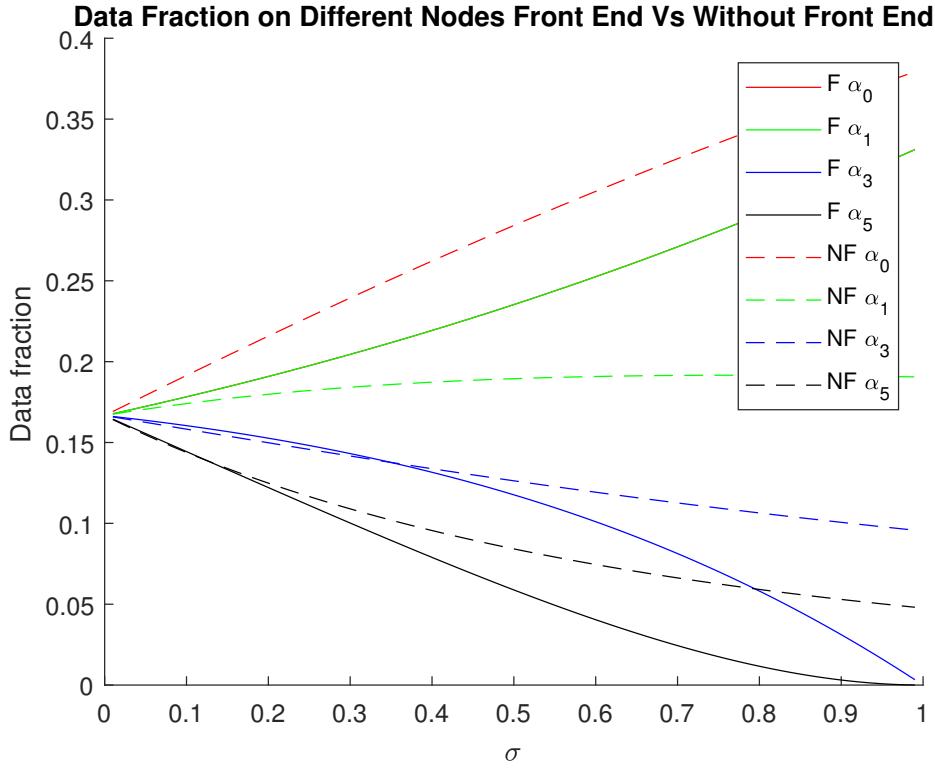


Figure 4.17: The comparing result between front-end processor with without front-end processor in  $2^*3$  regular network

Fig. 4.18 says that  $P_0$  takes more assigned task in without front-end scenario than front-end processor situation. As the  $\sigma$  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 front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to  $P_7$  and  $P_8$ .

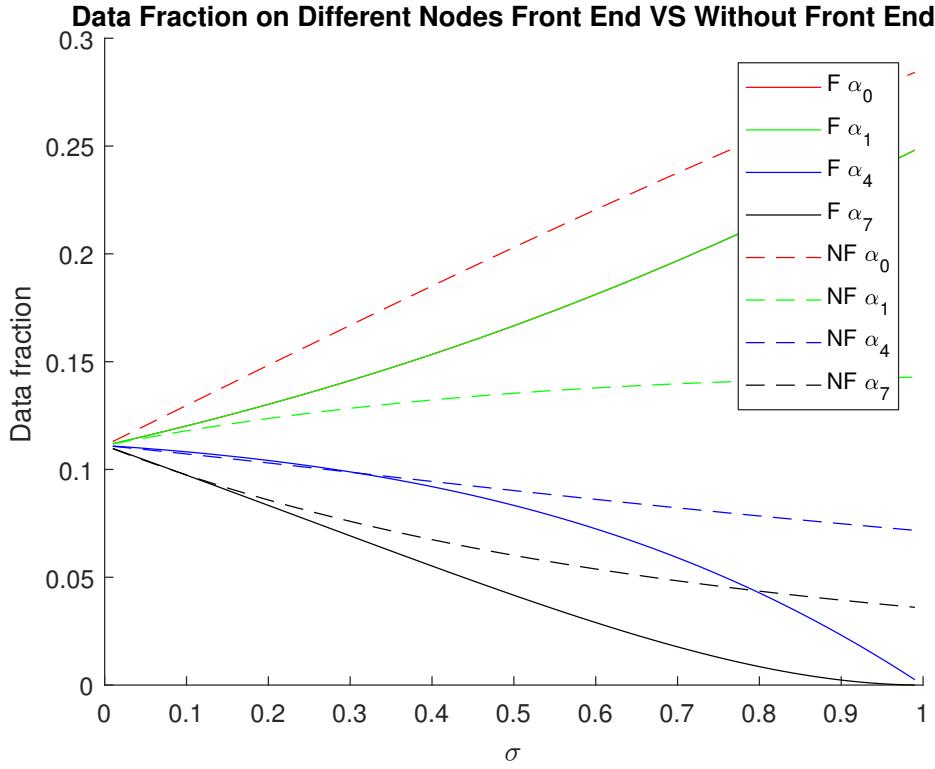


Figure 4.18: The comparing result between front-end processor with without front-end processor in  $3*3$  regular network injection on boundary processor

Fig. 4.19 says that  $P_0$  takes more assigned task in without front-end scenario than front-end processor situation. As the  $\sigma$  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<sub>2</sub>*, that is,  $P_5$ ,  $P_6$ ,  $P_7$  and  $P_8$  in the front-end assumption. Yet in the without front-end situation, there is still about 5% data fraction is communicated to  $P_5$ ,  $P_6$ ,  $P_7$  and  $P_8$ .

Comparing with Fig. 4.18,  $P_0$  takes less workload in inner grid position than boundary data injection. The reason is there are 4 neighbor processors

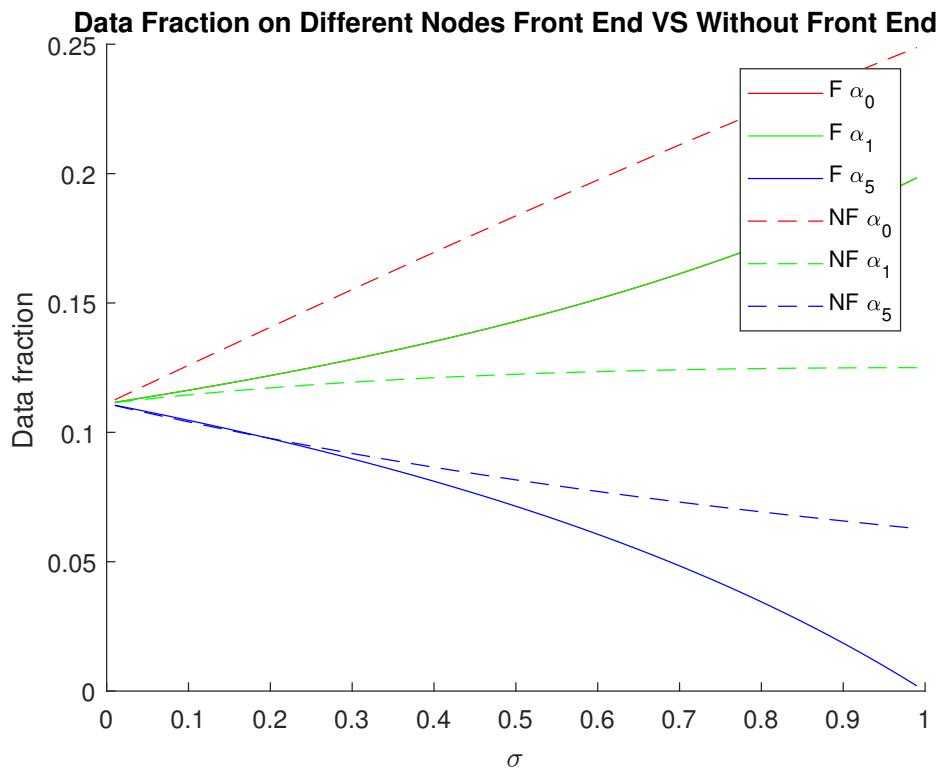


Figure 4.19: The comparing result between front-end processor with without front-end processor in  $3*3$  regular network injection on inner grid processor

on the  $level_1$ , yet there is solely three processors on  $level_1$  on the boundary.

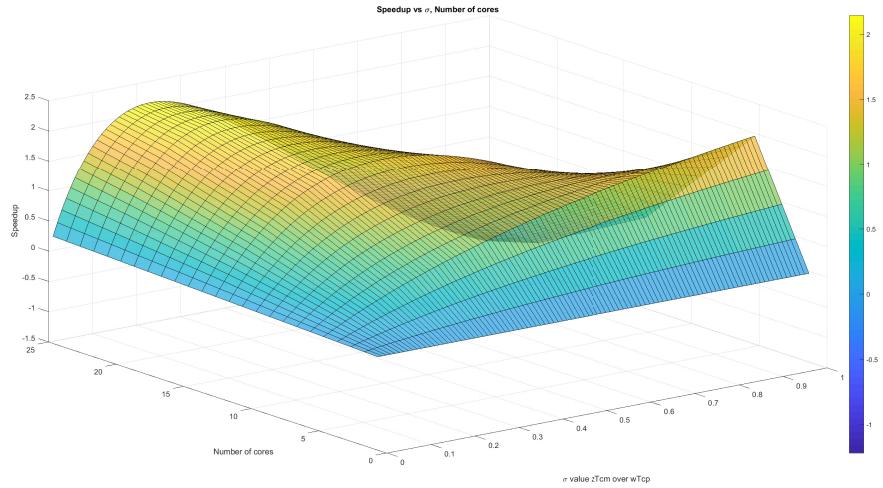


Figure 4.20: Speedup difference between front-end and without front-end in  $5 \times 5$  regular network

Fig. 4.20 shows the speedup difference between the front-end situation and without front-end scenario.

## **4.5 Store and Froward Switching**

In this chapter, we mainly discuss the virtual cut through [5] switching. In addition, the store and froward [17] schema is a mature data processing technique. In future works, we discuss it.

# Chapter 5

## Hypercube Network

### 5.1 With Front-end 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.

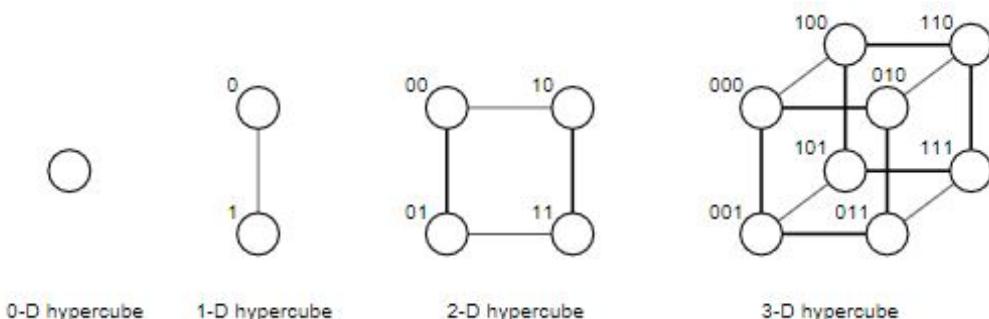


Figure 5.1: Hypercube in 0, 1, 2, 3 dimension. [1]

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 [18],

**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

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

, which is investigated in Regular Network Chapter.

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

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

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

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

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

### 5.1.2 Sensitivity Analysis With Front-end Processors

### **5.1.3 Multi-source Even Data Fraction**

#### **5.1.4 Multi-source Different Data Fraction**

## 5.2 Without Front-end 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

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

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

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

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

### 5.2.2 Sensitivity Analysis Without Front-end Processors

### **5.2.3 Multi-source Even Data Fraction**

#### **5.2.4 Multi-source Different Data Fraction**

# Chapter 6

## General Network

### 6.1 General Case With Front-end Processors

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

The  $m_1, m_2, \dots, m_n$  are the number of processors on the  $level_1, level_2, \dots, level_n$ . Also, the  $\alpha_{l_0}, \alpha_{l_1}, \dots, \alpha_{l_n}$  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 Without Front-end Processors

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

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

The speedup is

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

## **Chapter 7**

### **Conclusion and Future work**

#### **7.1 Conclusion**

## 7.2 Future Work

## Bibliography

- [1] D. R.-P. Mundan, “Parallel programming exercise 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, “Scheduling 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, *Scheduling divisible loads in parallel and distributed systems*, vol. 8. John Wiley & Sons, 1996.
- [7] J. T. Hung and T. G. Robertazzi, “Switching in sequential tree networks,” *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 968–982, 2004.
- [8] V. Bharadwaj, D. Ghose, and T. G. Robertazzi, “Divisible load theory: A new paradigm for load scheduling in distributed systems,” *Cluster Computing*, vol. 6, no. 1, pp. 7–17, 2003.
- [9] 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.

- [10] S. Fortune, “Voronoi diagrams and delaunay triangulations,” in *Computing in Euclidean geometry*, pp. 225–265, World Scientific, 1995.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] T. G. Robertazzi, “Toroidal networks,” *IEEE Communications Magazine*, vol. 26, no. 6, pp. 45–50, 1988.
- [15] 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.
- [16] S. Fortune, “A sweepline algorithm for voronoi diagrams,” *Algorithmica*, vol. 2, no. 1-4, p. 153, 1987.
- [17] G. R. Kanthraj, C. R. Srinivas, *et al.*, “Store and forward teledermatology,” *Indian Journal of Dermatology, Venereology, and Leprology*, vol. 73, no. 1, p. 5, 2007.
- [18] J. Błazewicz and M. Drozdowski, “Scheduling divisible jobs on hypercubes,” *Parallel computing*, vol. 21, no. 12, pp. 1945–1956, 1995.