

## Chapter 2

### Closed-form Processor Equivalence

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

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

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

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

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

## 2.1 Problem Description

The processing of massive amounts of data on distributed and parallel networks is becoming more and more common. The problem of minimizing the processing time of extensive loads originating from a multiplicity of sources and being processed on a multiplicity of nodes presents a challenge.

The research user case involves the regular mesh Fig. 2.12 or toroidal mesh Fig. 2.2 of  $m \times n$  processors. We refer to the  $m \times n$  processor as  $P_0, P_1, P_2, \dots, P_{m \times n - 1}$ .

The workloads are denoted as  $\mathbf{L}_1, \mathbf{L}_2, \dots, \mathbf{L}_k$ . We consider two situations, such as the processor contains the front-end and without front-end. In order to simply the question model, we have the following assumptions:

- We do not consider return communications.
- Communication delays are taken into consideration.
- The time costs of computation and communication are assumed to be linear function of the data size.
- 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.

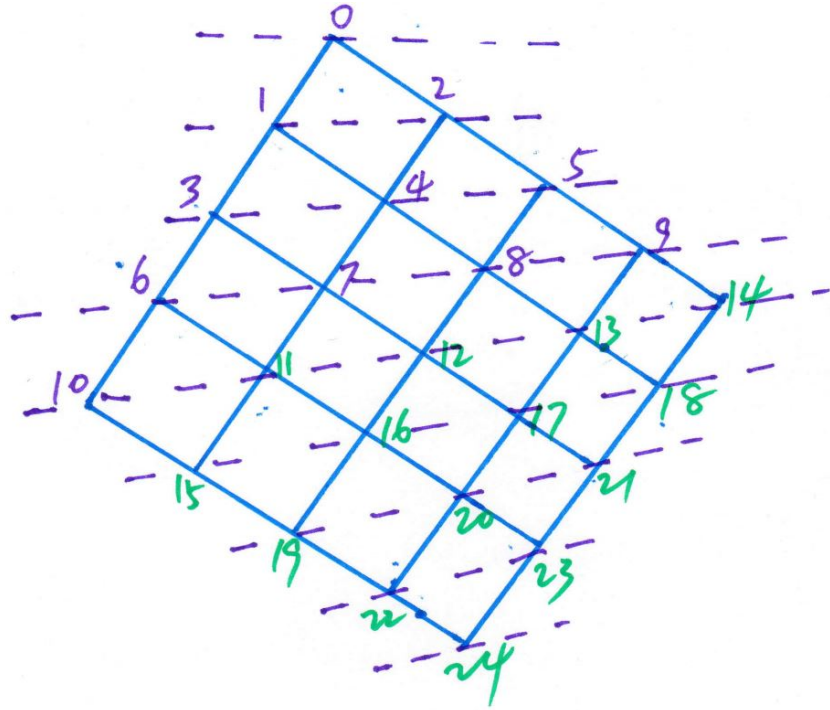


Figure 2.1: A  $m \times n$  regular mesh ( $m = 5, n = 5$ )

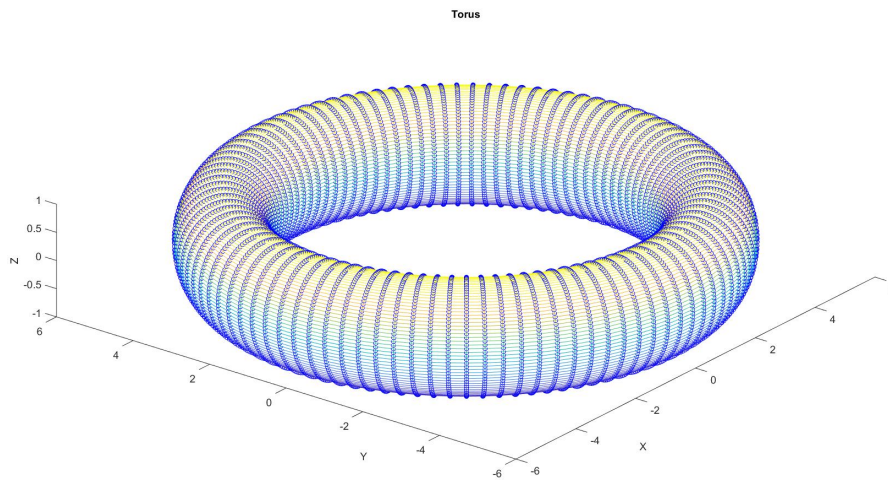


Figure 2.2: A toroidal mesh with grid unit cores

- The port of each processor is limited. In NOC (network on chip), the port number is 5.

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

In the without front-end scenario, unlike the previous protocol, the processors now simultaneously receive the data and only start to process it as soon as each processor receives its entire load assignment [4].

Thus, the problem becomes how to partition and schedule the workloads amongst the processors to get the minimum finish time. This study will focus on the speedup metric which is defined as the ratio of computation time on one processor to the computation time on the entire  $m * n$  processors.

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

### Notations and Definitions

- $D_i$ : The minimum number of hops from the node  $P_i$  to the nearest data load injection  $L$ .
- $P_i$ : The  $i$ th processor.  $i$  should be from 0 to  $m * n - 1$ .
- $L_i$ : The  $i$ th work load.  $i$  should be from 1 to  $k$ .

- $\alpha_0$ : 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 regular mesh or toroidal.
- $z_i$ : The inverse link speed on the  $i$ th link.
- $T_{cp}$ : Computing intensity constant. The entire load can be processed in  $\omega_i T_{cp}$  on the  $i$ th processor.
- $T_{cm}$ : Communication intensity constant. The entire load can be transmitted in  $z_i T_{cm}$  seconds over the  $i$ th link.
- $T_{f,n}$ : The finish time of the whole regular network. Here  $T_{f,n}$  is equal to  $\omega_{eq} T_{cp}$ .
- $T_{f,0}$ : The finish time for the entire divisible load solved on the root processor. Here  $T_{f,0}$  is equal to  $1 \times \omega_0 T_{cp}$ , that is  $\omega_0 T_{cp}$ .
- $\sigma = \frac{z T_{cm}}{\omega T_{cp}}$ : The ratio between the communication speed to the computation speed.
- $\sum_{i=1}^k L_i = 1$
- $\sum_{i=0}^{m*n-1} \alpha_i = 1$
- $Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

## 2.2 With Front End Scenario

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

### 2.2.1 Regular Mesh

#### Data Injection on The Corner Processor

##### $2 \times 2$ Regular Mesh

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

The processor  $P_0, P_1$  and  $P_2$  start to process its respective fraction at the same time. The port number of  $P_0$  is limited, so the processor  $P_3$  handles until the  $\alpha_1$  and  $\alpha_2$  are assigned completed to  $P_1$  and  $P_2$ .

According to the divisible load theory[5], we can obtain the timing diagram Fig. 2.4.

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

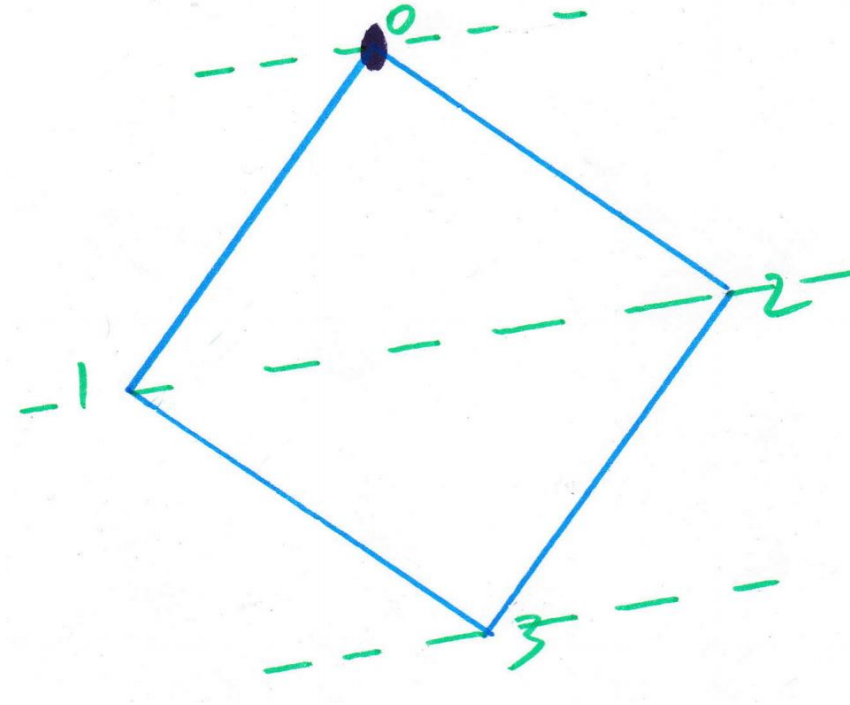


Figure 2.3: The 2\*2 regular mesh and the data injection is on  $P_0$

$$\left\{ \begin{array}{ll} \alpha_0 \omega T_{cp} = T_{f,m} & (2.1) \\ \alpha_0 \omega T_{cp} = T_{f,m} & (2.2) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (2.3) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (2.4) \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.5) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (2.6) \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (2.7) \\ 0 < \sigma < 1 & (2.8) \\ 0 < \alpha_0, \alpha_1, \alpha_2, \alpha_3 < 1 & (2.9) \end{array} \right. \quad (2.10)$$

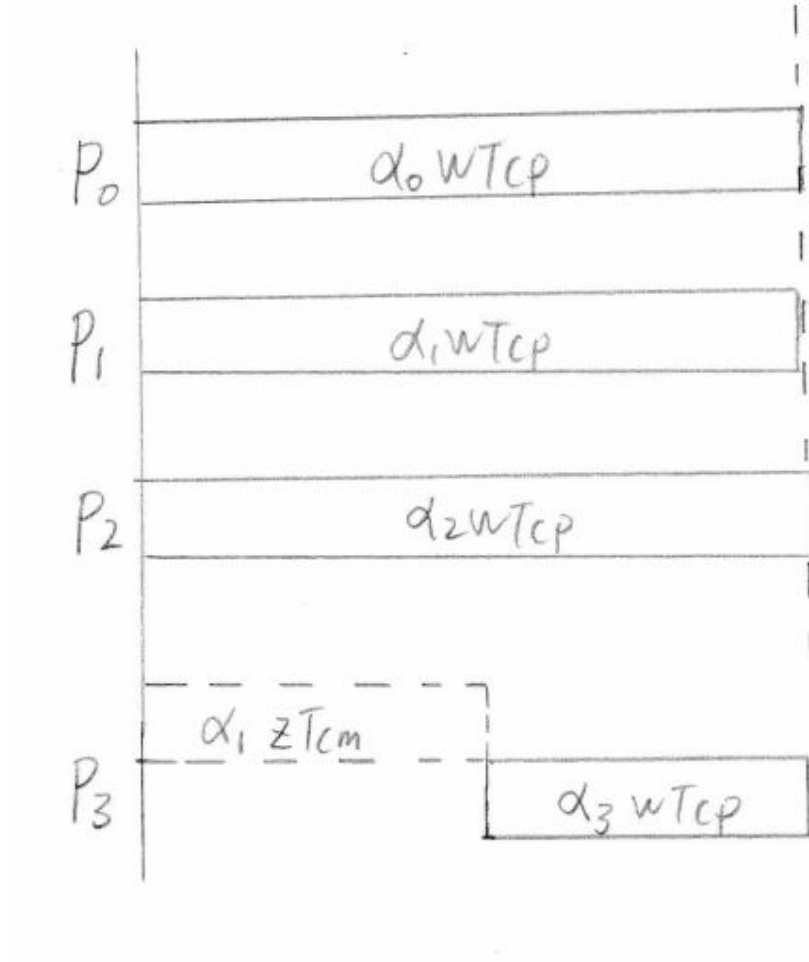


Figure 2.4: The timing diagram for 2\*2 regular mesh and the data injection position is  $P_0$

We can obtain the

$$\alpha_1 = \alpha_2$$



The group of equations are represented by the matrix form:

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

The simulation result is illustrated:

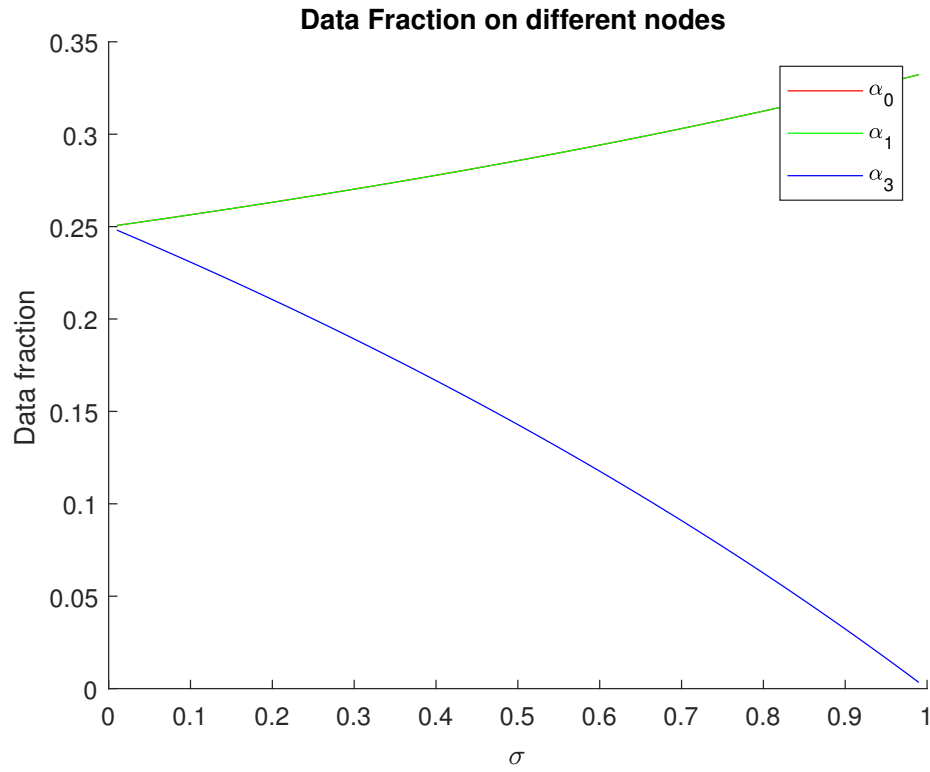


Figure 2.5: 2\*2 regular mesh.  $\alpha_0, \alpha_1, \alpha_2, \alpha_3$  value

In Fig. 2.5,  $P_0, P_1, P_2$  three processors have the same data fraction workload, so the curve of  $\alpha_0$  and  $\alpha_1$  coincide. The figure says that the  $\sigma$  value grows and the  $\alpha_3$  drops simultaneously. In other words, the communication capacity

drops down and there is less data workload assigned to  $P_3$ . Further, it means it will be economical to keep the load local nor distribute to other processor.

The speedup is:

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

### 2\*3 Regular Mesh

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

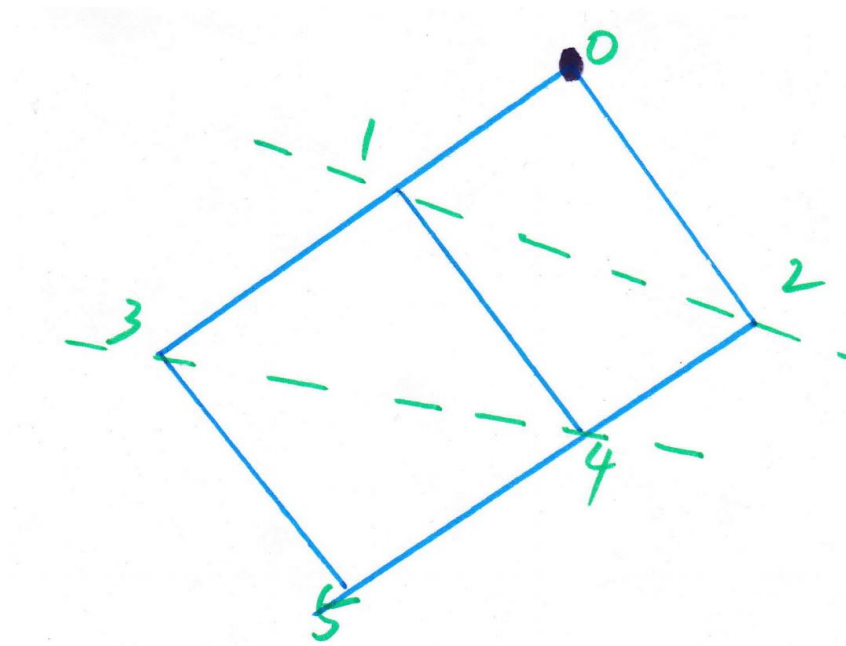


Figure 2.6: The 2\*3 regular mesh and the data injection happens on corner processor  $P_0$

$P_0, P_1$  and  $P_2$  start processing the load at the same time. According the processor  $P_3$  and  $P_4$  start to work until they get the data from their parent

processor  $P_1, P_2$ . Yet, the number of port of each processor is 4 or 5. So the fraction of  $\alpha_3, \alpha_4$  are queued in the  $L_1$ . So the  $P_3$  and  $P_4$  have to wait the fraction of  $\alpha_1$  and  $\alpha_2$  are transmitted completely and then they will obtain their respective workload task fraction.

The last unit core  $P_5$  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[5], we can obtain the timing diagram Fig. 2.7.

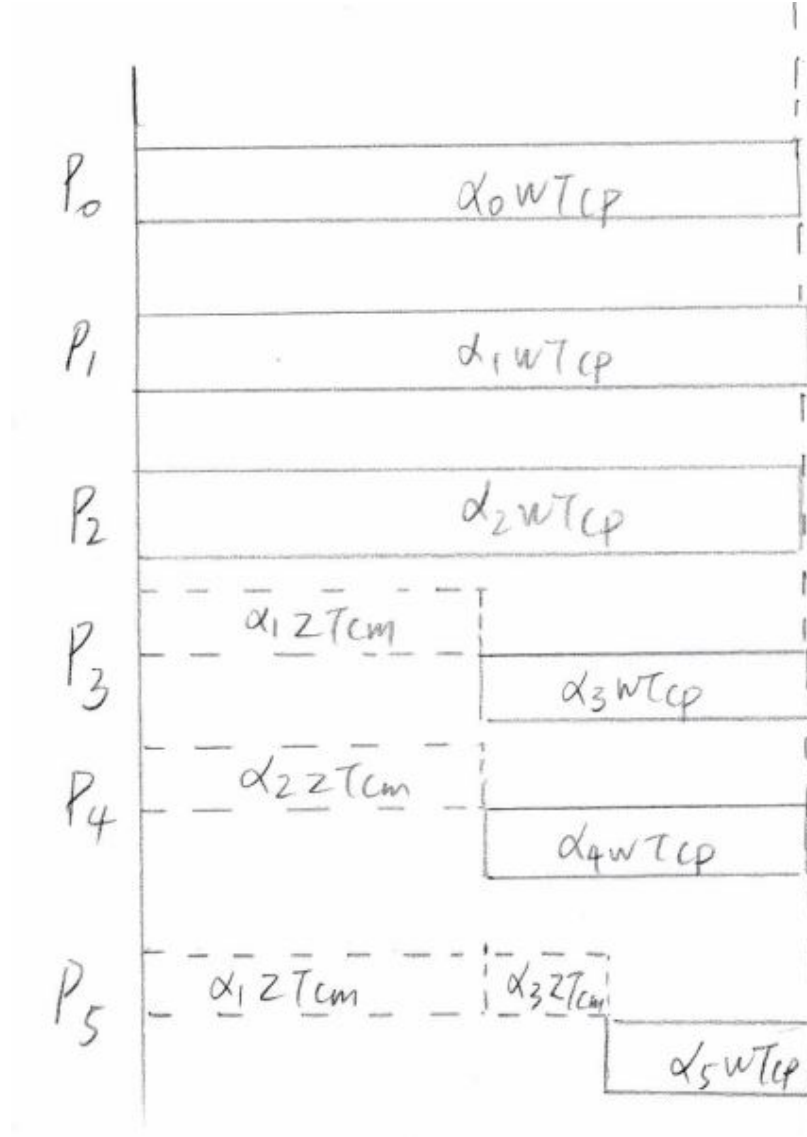


Figure 2.7: The timing diagram for a 2\*3 regular mesh and the data injection happens on processor  $P_0$

The equations as follows:

$$\left\{ \begin{array}{ll} \alpha_0 \omega T_{cp} = T_{f,m} & (2.12) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (2.13) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (2.14) \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.15) \\ \alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.16) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.17) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (2.18) \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (2.19) \\ 0 < \sigma < 1 & (2.20) \\ 0 < \alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5 < 1 & (2.21) \end{array} \right. \quad (2.22)$$

After the simplification, we found:

$$\left\{ \begin{array}{l} \alpha_1 = \alpha_2 \\ \alpha_3 = \alpha_4 \end{array} \right. \quad (2.23)$$

$$\left\{ \begin{array}{l} \alpha_3 = \alpha_4 \end{array} \right. \quad (2.24)$$

The matrix closed-form formula is:

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 \\ 0 & \sigma - 1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.25)$$

The simulation result are shown in Fig. 2.8.  $P_0, P_1$  have the same fraction so their curves of  $\alpha_0$  and  $\alpha_1$  coincide.

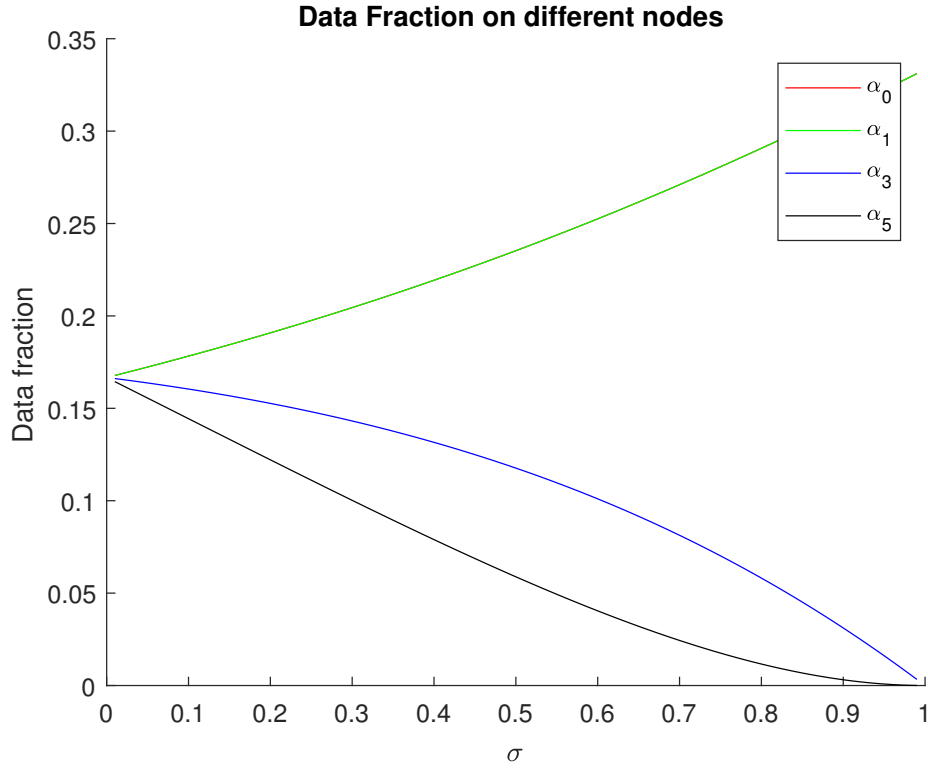


Figure 2.8: 2\*2 regular mesh.  $\alpha_0, \alpha_1, \alpha_3, \alpha_5$  data fraction value

## $2*n$ Regular Mesh

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

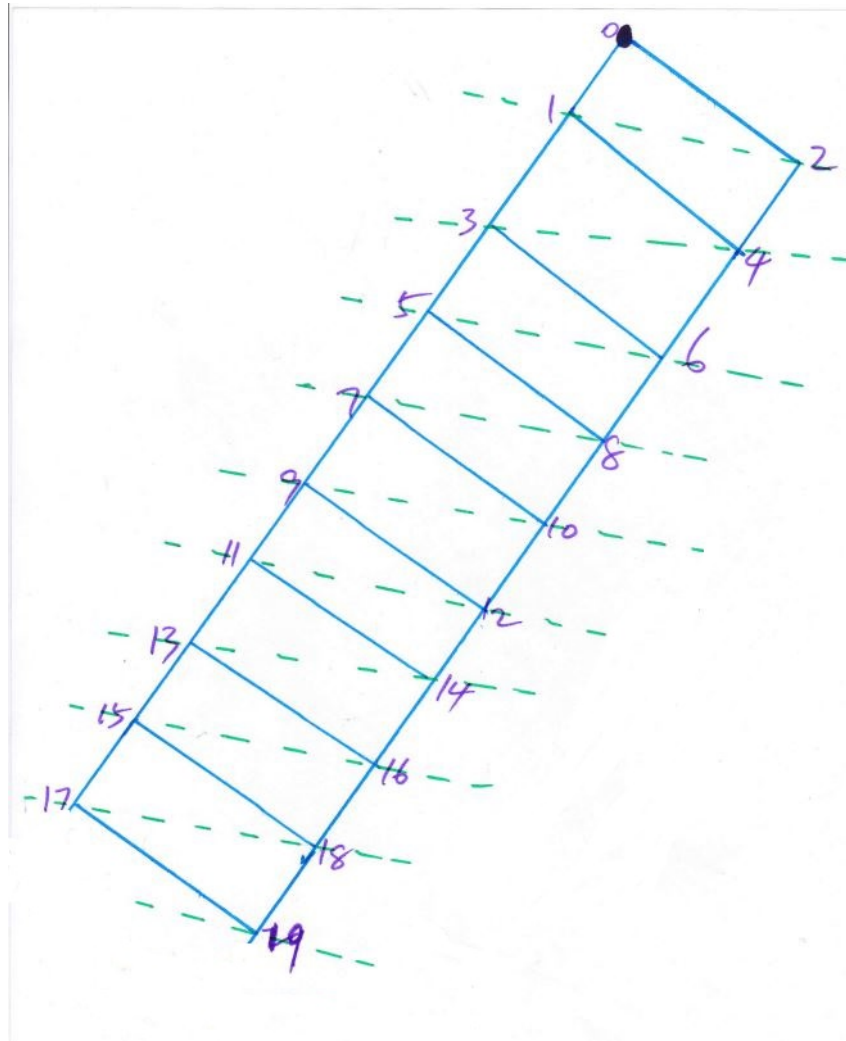


Figure 2.9: The  $2*n$  ( $n = 10$ ) regular mesh and the workload happens on  $P_0$

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

The equations are presented as:

$$\left\{ \begin{array}{ll} \alpha_0 \omega T_{cp} = T_{f,m} & (2.26) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (2.27) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (2.28) \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (2.29) \\ \alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.30) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.31) \\ \vdots & (2.32) \\ (\alpha_1 + \alpha_3 + \dots + \alpha_{2 \times n - 1}) z T_{cm} + \alpha_{2 \times n - 1} \omega T_{cp} = T_{f,m} & (2.33) \\ \alpha_0 + \dots + \alpha_{2 \times n - 1} = 1 & (2.34) \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (2.35) \\ 0 < \sigma < 1 & (2.36) \\ 0 < \alpha_0 \quad \alpha_1 \quad \alpha_3 \quad \dots \quad \alpha_{2 \times n - 1} < 1 & (2.37) \end{array} \right. \quad (2.38)$$

The matrix closed-form is shown:



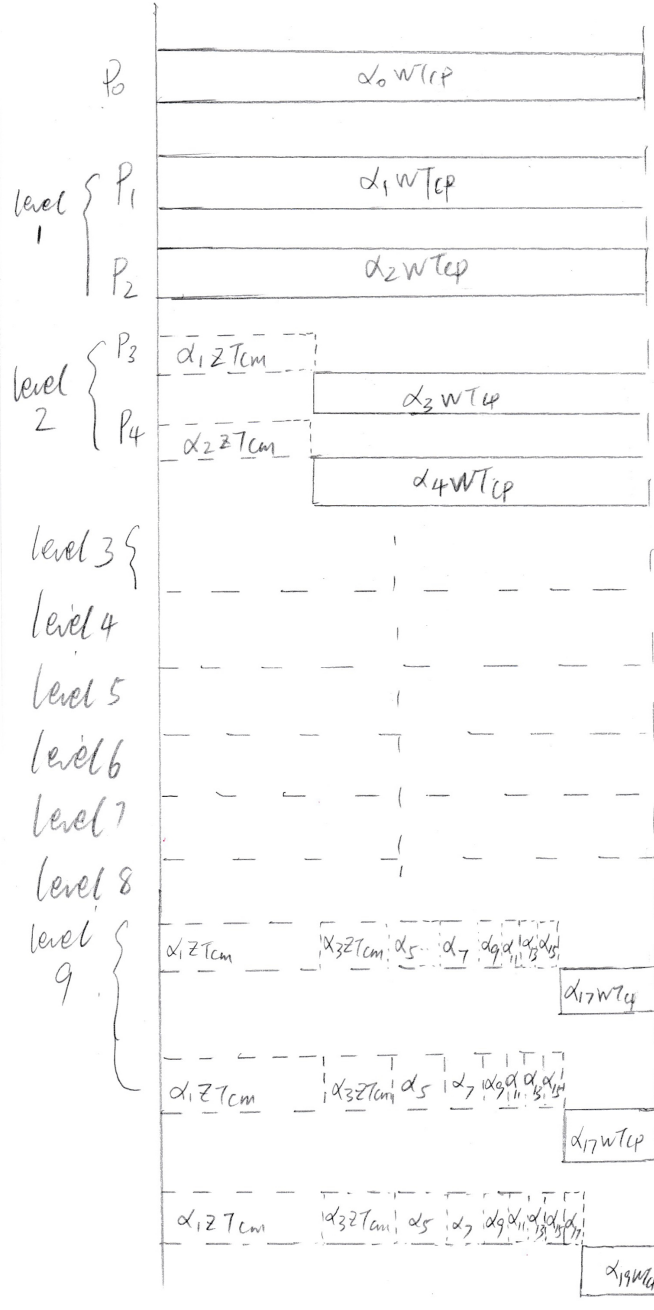


Figure 2.10: The timing diagram for 2\*10 regular mesh and the data injection happens on  $P_0$

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma - 1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2 \times n - 3} \\ \alpha_{2 \times n - 1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2.39)$$

Finally, the speedup is:

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

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

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

## $m \times n$ Regular Mesh

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

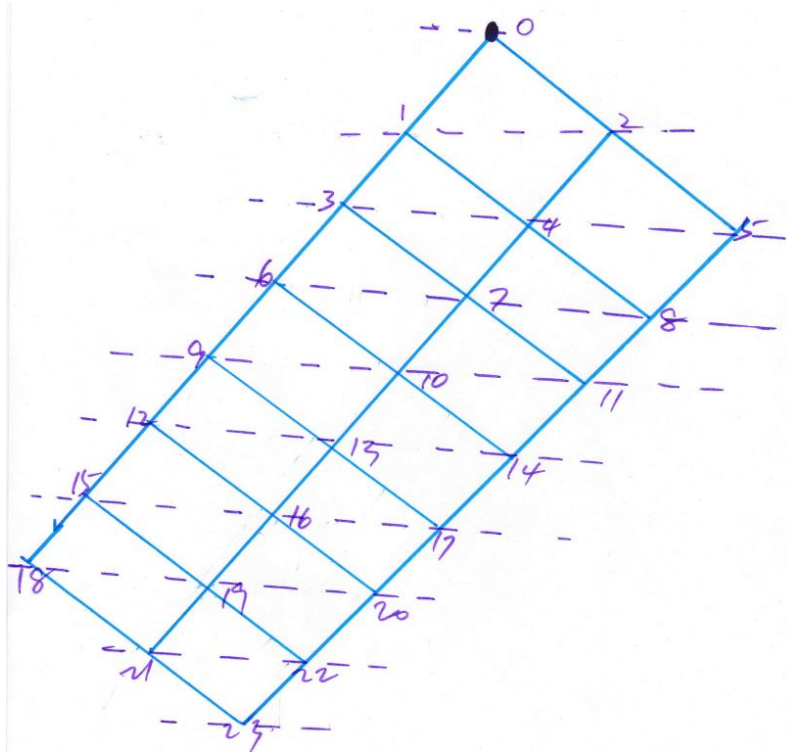


Figure 2.11:  $3 \times 8$  regular mesh. The data injection position is  $P_0$

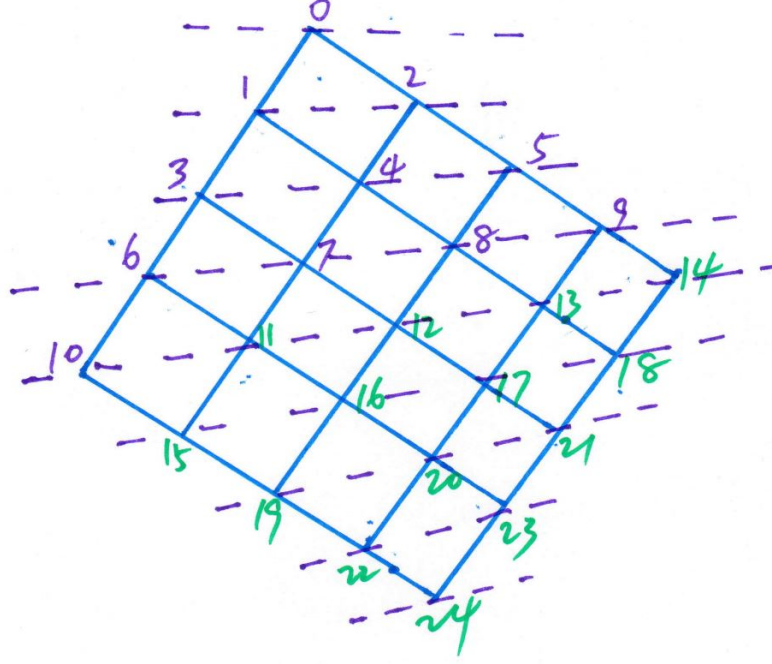


Figure 2.12: 5\*5 regular mesh. The data injection position is processor  $P_0$

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

$$\begin{bmatrix}
 1 & 2 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\
 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \alpha_0 \\
 \alpha_1 \\
 \alpha_3 \\
 \alpha_6 \\
 \alpha_9 \\
 \alpha_{12} \\
 \alpha_{15} \\
 \alpha_{18} \\
 \alpha_{21} \\
 \alpha_{23}
 \end{bmatrix}
 =
 \begin{bmatrix}
 1 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 0 \\
 \vdots \\
 0
 \end{bmatrix}
 \quad (2.40)$$

Also, the matrix equation for Fig. 2.12:

$$\begin{bmatrix}
 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\
 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\
 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1
 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (2.41)$$

## Data Injection On the Boundary Processor

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

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

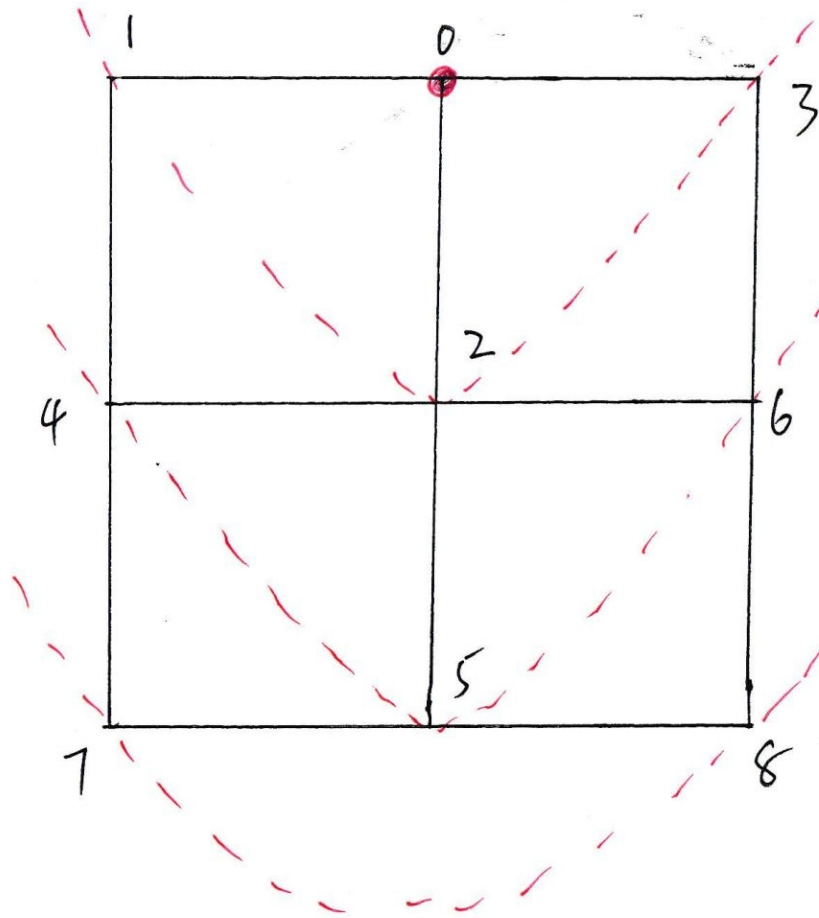


Figure 2.13: The 3\*3 regular mesh and the data injection position is  $P_0$

The timing diagram is Fig. 2.14:

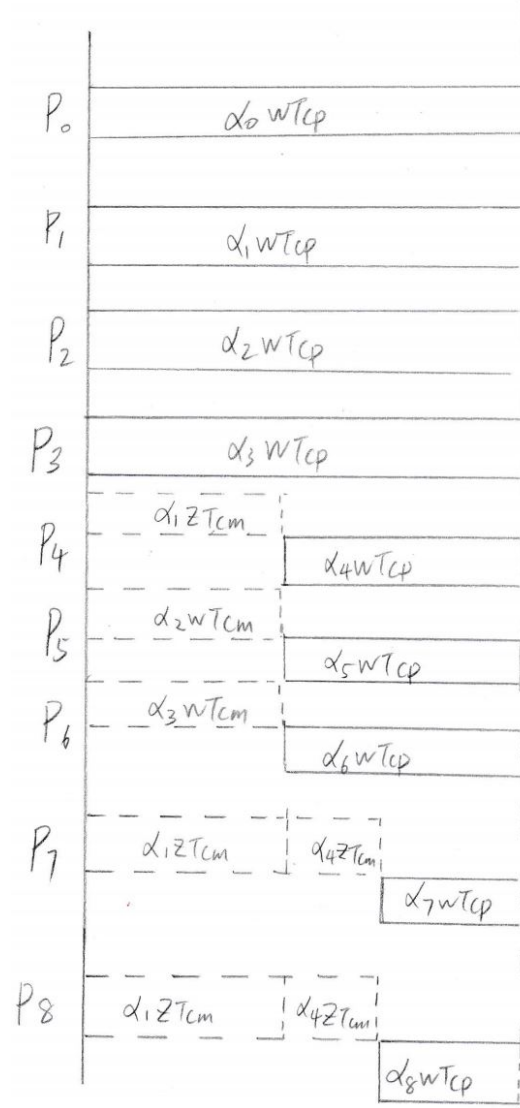


Figure 2.14: The timing diagram for 3\*3 regular mesh and the data injection occurs on the boundary processor  $P_0$

The equations are:

$$\left\{ \begin{array}{ll} \alpha_0 \omega T_{cp} = T_{f,m} & (2.42) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (2.43) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (2.44) \\ \alpha_3 \omega T_{cp} = T_{f,m} & (2.45) \\ \alpha_1 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (2.46) \\ \alpha_2 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.47) \\ \alpha_3 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.48) \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.49) \\ (\alpha_2 + \alpha_5) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.50) \\ \alpha_0 + \dots + \alpha_8 = 1 & (2.51) \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (2.52) \\ 0 < \sigma < 1 & (2.53) \\ 0 < \alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \dots \quad \alpha_8 < 1 & (2.54) \end{array} \right. \quad (2.55)$$



And the matrix form is :

$$\begin{bmatrix} 1 & 3 & 3 & 2 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 \\ 0 & \sigma - 1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_4 \\ \alpha_7 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.56)$$

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

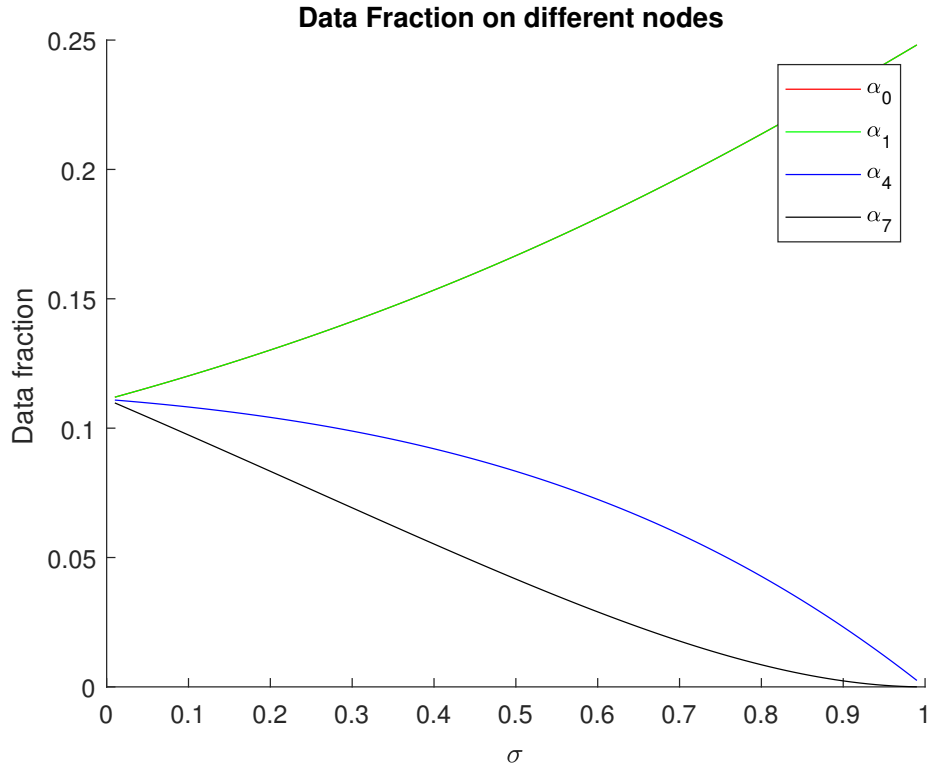


Figure 2.15: The data fraction simulation result of 3\*3 regular mesh and the data injection happens on the boundary  $P_0$

### Data Injection On The Inner Grid Processor

$L_1$  is on the inner grid processor  $P_0$  Fig. 2.16,

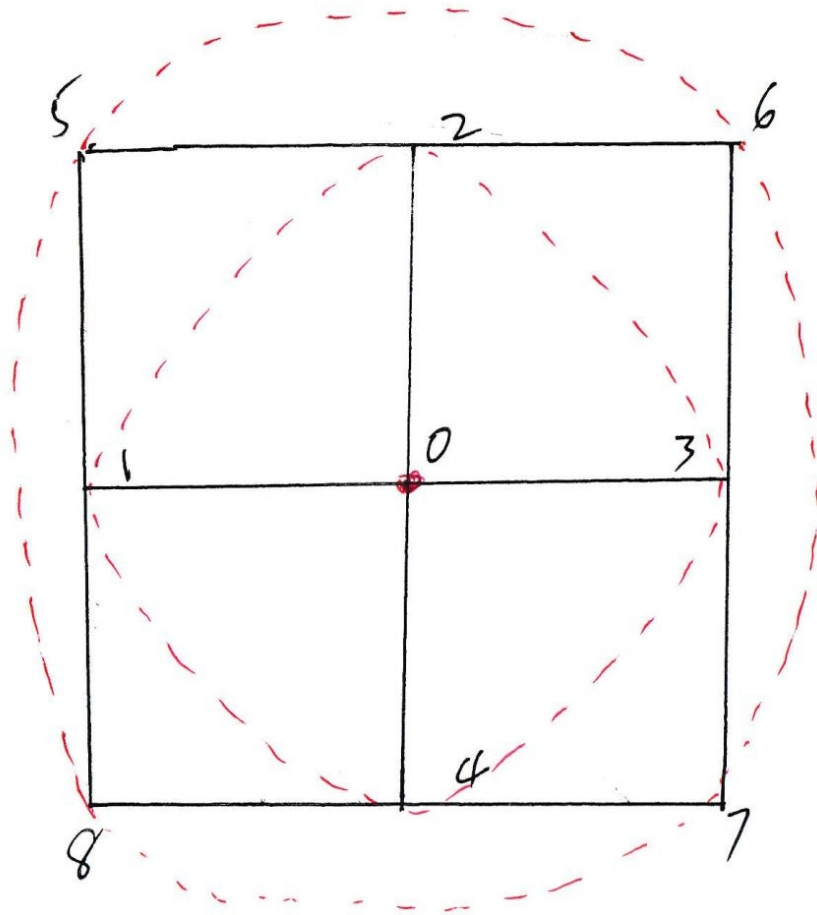


Figure 2.16: 3\*3 regular mesh. The data injection position is inner grid point  $P_0$

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

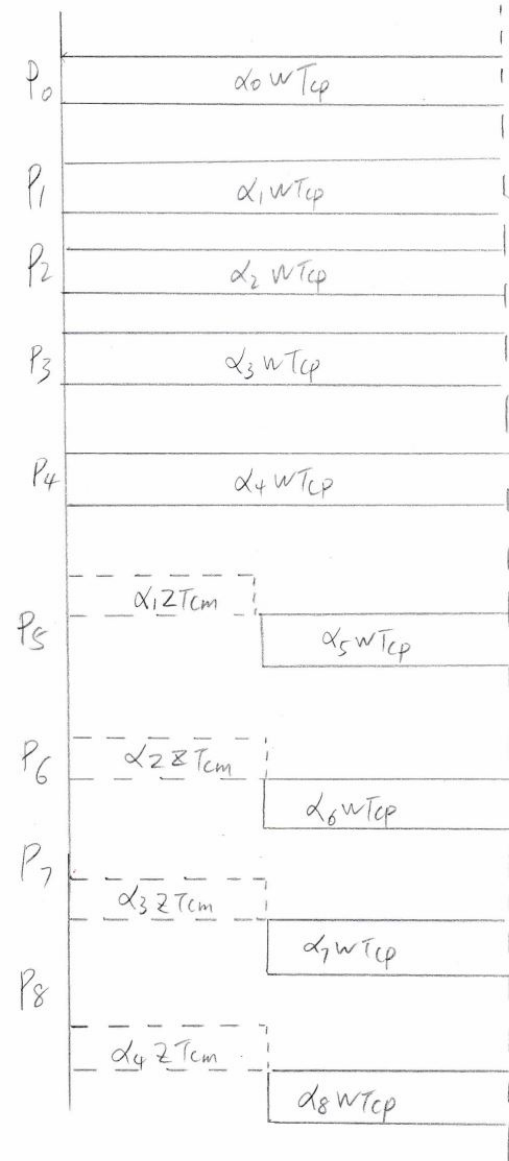


Figure 2.17: The timing diagram for 3\*3 regular mesh and the data injection is inner grid  $P_0$

The group of equations are:

$$\left\{ \begin{array}{ll} \alpha_0 \omega T_{cp} = T_{f,m} & (2.57) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (2.58) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (2.59) \\ \alpha_3 \omega T_{cp} = T_{f,m} & (2.60) \\ \alpha_4 \omega T_{cp} = T_{f,m} & (2.61) \\ \alpha_1 z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (2.62) \\ \alpha_1 z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (2.63) \\ \alpha_1 z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (2.64) \\ \alpha_1 z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (2.65) \\ \sigma = \frac{z T_{cm}}{\omega T_{cp}} & (2.66) \\ 0 < \sigma < 1 & (2.67) \\ 0 < \alpha_0 \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_8 < 1 & (2.68) \end{array} \right.$$

The matrix form is :

$$\begin{bmatrix} 1 & 4 & 4 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad (2.69)$$

The simulation result for the inner grid is Fig. 2.18:  $P_0$  and  $P_1$  have the same  $\alpha$  value,so the curve of  $\alpha_0$  and  $\alpha_1$  coincide.

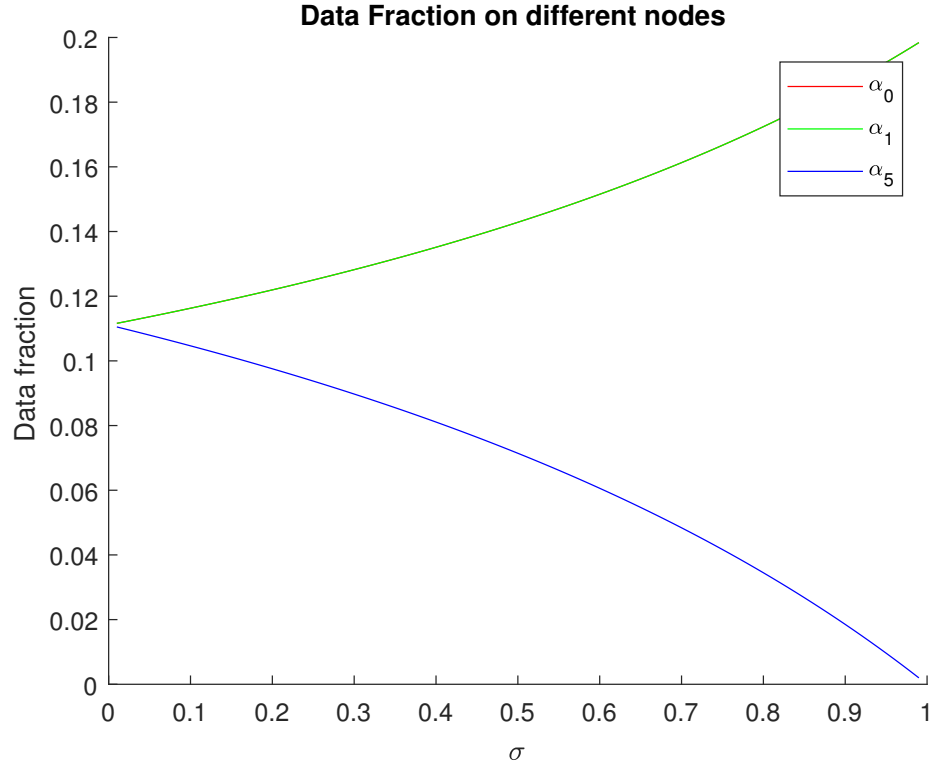


Figure 2.18: 3\*3 regular mesh. The data injection position is inner grid point  $P_0$

### 2.2.2 Torus mesh

According to the paper[6], 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. Other situation can be extended using the same rule. The toroidal network is also a regular complex in that each polygonal face has the same number of links and each node is connected to the same number of links.

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

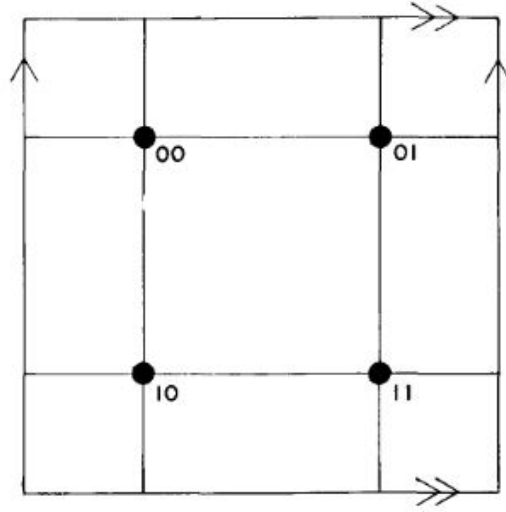


Figure 2.19: The rectangular toroidal network

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

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

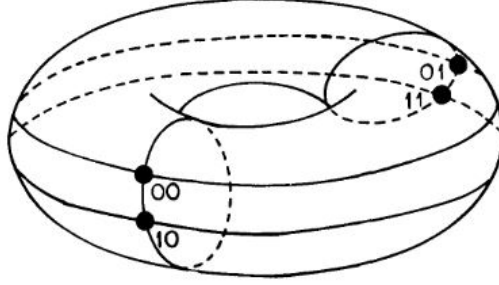


Figure 2.20: The rectangular toroidal network

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

$$\left\{ \begin{array}{l} D = D_x + D_y \\ D_x = \min\{\|N_x - L_x\|, m - \|N_x - L_x\|\} \\ D_y = \min\{\|N_y - L_y\|, n - \|N_y - L_y\|\} \end{array} \right. \quad \begin{array}{l} (2.70) \\ (2.71) \\ (2.72) \end{array}$$

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

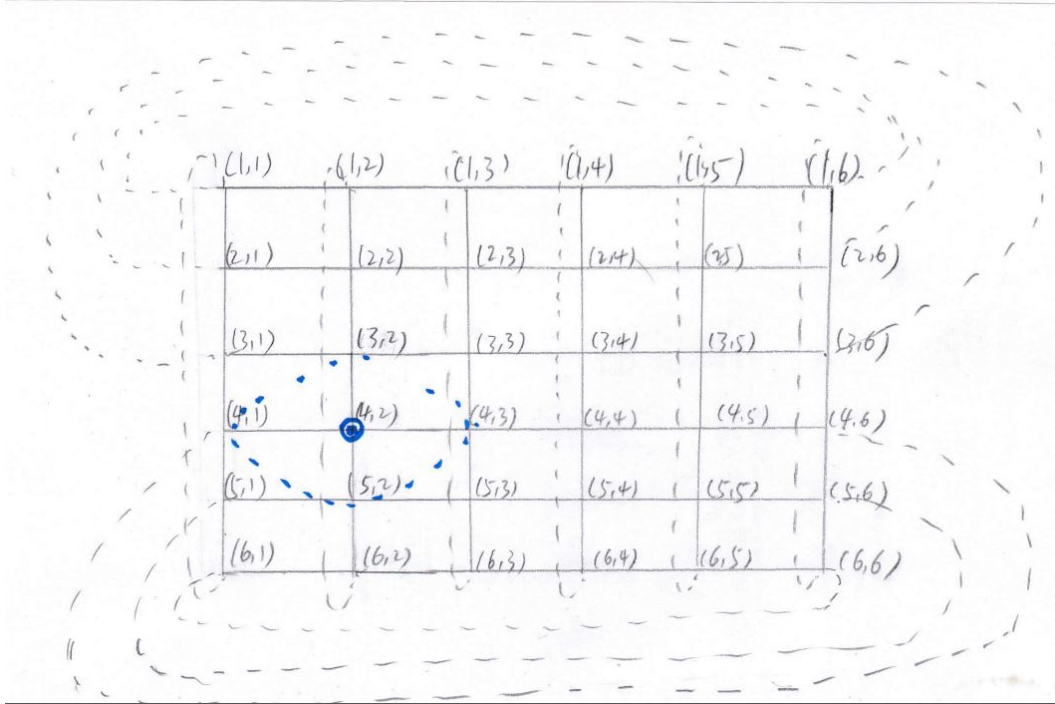


Figure 2.21: The  $m * n$  toroidal network and the data injection is  $P_{4,2}$



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

Table 2.1: The processor number of various  $D_i$

And the  $D_i$  matrix as follow in table Table 2.1

The matrix closed-form is

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma - 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma - 1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (2.73)$$

The simulation result is :

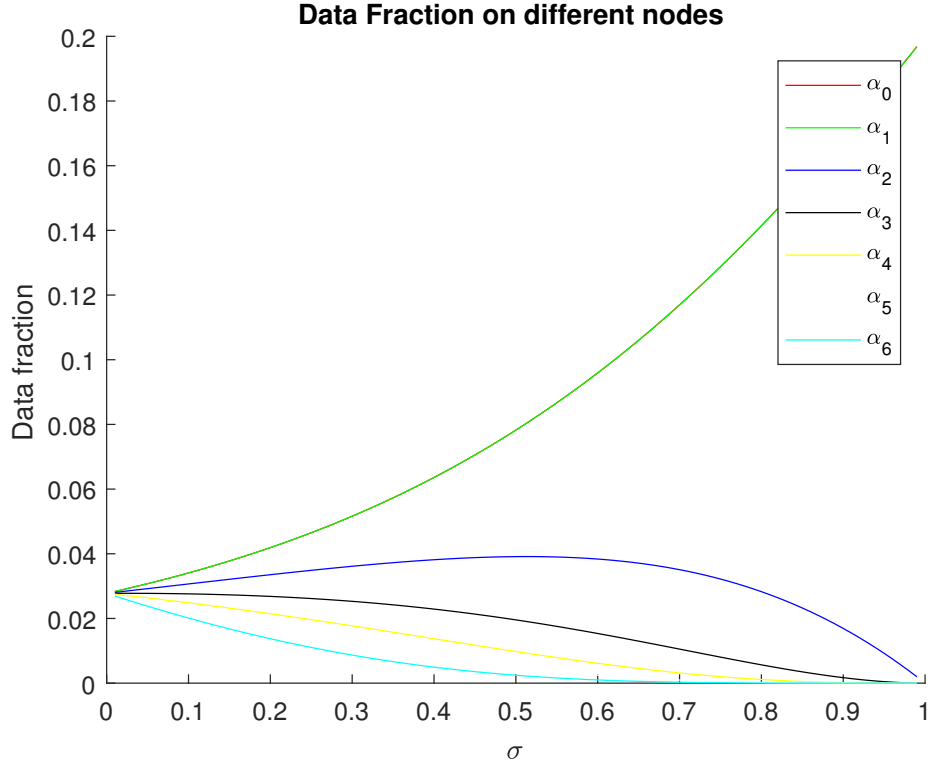


Figure 2.22: The data fraction deployed based on the radius value

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

## 2.3 Without Front End Scenario