Data Distribution Equivalence for Data Intensive Interconnection Networks

A Dissertation Presented

by

Junwei Zhang

to

The Graduate School

in Partial Fulfillment of the

Requirements

for the Degree of

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

August 2018

Stony Brook University

The Graduate School

Junwei Zhang

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

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

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

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

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

Wendy Tang - Member
Associate Professor, Department of Electrical and Computer Engineering

This dissertation is accepted by the Graduate School.

Charles Taber
Dean of the Graduate School

Abstract of the Dissertation

Data Distribution Equivalence for Data Intensive Interconnection Networks

by

Junwei Zhang

Doctor of Philosophy

in

Applied Mathematics and Statistics

Stony Brook University

2018

This thesis considers two problems. One problem is closed-form solutions for equivalence computation of divisible workload in mesh and toroidal networks and the other problem is scheduling divisible workloads from multiple sources in mesh and toroidal networks of processors.

We propose a flow matrix closed-form equation to present the equivalence, which allows a characterization of the nature of minimal time solution and a simple method to determine when and how much load to distribute for processors. Also, we propose the use of a reduced Manhattan distance Voronoi diagram algorithm (RMDVDA) to minimize the overall processing time of these workloads by taking advantage of the processor equivalence technique. Three case studies with 10 sources of workloads are presented to illustrate the

general approach for multiple sources of workloads. In the first phase, a Manhattan distance Voronoi diagram is used to obtain a network cluster division. In the second phase, we propose an efficient algorithm to obtain near-optimal load distribution among processors represented by equivalent processors. The algorithm minimizes the number of processors utilized. Experimental evaluation through simulations demonstrates that a task can be finished in the same suboptimal time and yet save over 30% of processor resources.

Key Words : Divisible Load Theory, Voronoi Diagram, Multi-source, Network on Chip (NOC) Interconnection Network, Mesh, Data Intensive Load

To my Parents and my loved ones

Table of Contents

List of Figures

List of Tables

Acknowledgements

Thank you to Professor Thomas G. Robertazzi who gave me strong academic and emotional support during hard times.

Thank you to Professor Esther M. Arkin, Professor Joseph S.B. Mitchell, Professor Xiaolin Li, Associate Professor Wendy Tang who give me lots of paper modification suggestions and joined my thesis defense committee. Thank you to Professor Jim Jiao, who gives me lots of suggestions about Ph.D. life time management.

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

Thanks to Professor David Gu, who taught me a great deal of optimal mass transport theory and computational conformal geometry.

Thanks to my co-authors, especially Ph.D. candidate Liang Mi, I learn a great deal of medical image knowledge from him.

Thanks to Ocoos, eBay, BNL and Uber, I found four perfect internship mentors. They gave me intensive industry project training and a good work attitude.

Chapter 1

Introduction

During the past decade, divisible load scheduling and distribution techniques have been widely studied forming the divisible load theory (DLT). The load is defined to be divisible for computation and communication if it can be partitioned arbitrarily among processors. The divisible load model is a good approximation of tasks that require large number of identical, low-granularity computations, thus has been proposed for a wide range of scientific and engineering data processing, such as image processing, matrix multiplication, fast-Fourier-transformation (FFT), video encoding/decoding, stereo matching, etc [2] [3] [4] [5] [6]. A various constrains on different interconnection topologies [7] [8] [9] scheduling problems have been studied by the researchers.

Also, Networks on Chip (NOC) have become important and even essential to today's and future's electronic technology. This paradigm has emerged as a communication backbone to enable a high degree of integration in multi-core System-on-chip (SOC) [10] [11] [12]. To aid our discussion we will consider in detail the most popular and the most basic structured interconnection network for on-chip networking: the mesh and toroidal network. It is basically a chessboard (Manhattan streets) patterns of interconnection. Intellectual property (IP) are connected to their north, south, east and west neighbors through a network interface (NI). More specifically each network interface connects

through a link to a nearby "router". Internally, each router consists of at most a 5×5 cross-bar interconnection network (for up to four neighbors and a local connection), buffers, control logic and registers. Switching on a packet basis can either be done using *store and forward* [13] or *virtual cut-through* [14] switching approaches.

In this chapter the smallest possible networks are examined: networks implemented on a single integrated circuit chip. There is a reason that such Networks on Chips (NOC) have become important and even essential to todays and the futures electronic technology.

That reason has to do with Moores law [150]. Moores law, which was valid for decades, held that through higher clock frequencies and continual miniaturization of chip circuits, computer processing power would double every 1824 months. However, in recent years the operating frequencies of chips have not increased as Moores law predicted. This is due to unsustainable power demands and constraints on thermal density. But transistor density has continued to increase. Thus the solution adopted in recent years is to make use of parallel processing by including many (multi) cores on the same chip. This development came after systems on chips (SOCs) became popular, starting in the 1990s. Systems on chips integrate processors, video processors, graphics engines, embedded memory, I/O devices, and even analog components on a single chip [102, 150].

Classic on-chip communication architectures are buses and point-to-point (P2P) interconnection networks (Fig.8.1). Both architectures have scalability issues though. Bus architectures can connect a few tens of cores with minimal cost. However, unless one is broadcasting, a bus is an inefficient medium as communication can only occur between two processors/components at one time. This inefficiency worsens as the number of processors is increased. There are also power consumption issues with buses. Finally, there is the significant

capacitative load of bus drivers, which creates appreciable delay and energy use in interconnected wires [102].

Point-to-point interconnection, at least for a complete interconnection (every core directly connected to every other core), for N nodes has an O(N2) complexity. This makes it unsustainable for a large number of cores. Point-to-point communication requires dedicated wires and specialized interfaces (which buses do not have) [102].

Further, Voronoi digram is comprehensively utilized in network optimization application [15] [16] [17]. Nonetheless, most DLT researches mainly focused on the single source load distribution because of the complexity of optimal scheduling increase exponentially with the increase of load sources. Jia [18] propose a general method to address the multi-source scheduling problem. Yet, the algorithm has three perspective need more attention. First, his algorithm need to discard the part of previous computation result and do the global re-distribution. Then, his method didn't give a quantitative model to represent the data fraction on each processor. Third, the algorithm doesn't propose a general equivalence computation power, which is very useful for user to estimate the makespan and help me to allocate the data source workload.

The concept of processor equivalence was first proposed in [19]. This inspires us to adopt this model to tackle multi-source workload scheduling problems. We propose a flow matrix closed-form equation to present the equivalence, which allows a characterization of the nature of minimal time solution and a simple method to determine when and how much load to distribute for processors.

The problem of minimizing the processing time of extensive processing loads originating from a multiplicity of sources and being processed on a multiplicity of nodes presents a challenge that, if successfully met, could foster a range of new creative applications. Inspired by this challenge, we discuss in

this paper a representative scheduling model and solutions for multi sources distributing load into a grid of a finite number of processors.

A growing literature on grid scheduling has appeared over the past several years. Some representative work is now discussed. There is work on general architectures in [20]. Prediction involving local grid scheduling [21], queue wait times [22] and variance [23] have been studied. The decoupling of computation and data scheduling is the subject of [24]. Grid scheduling research has involved such features as multiple simultaneous requests [25], memory consciousness [26], fault tolerance [27], incentives [28] and biological concepts [29]. A report on scheduling experiences with parameter sweep applications appears in [30]. A study showing a comparison of grid scheduling algorithms for coarse grain independent tasks appears in [31].

The mathematical theory of divisible loads is uniquely suited to serve as a basis for analytically modeling and solving grid scheduling problems in its ability to capture both computing and communication in a single model. Divisible load theory [32] [33] [34] is characterized by the fine granularity and large volume of loads. There are also no precedence relations among the data elements. Such a load may be arbitrarily partitioned and distributed among processors and links in a system. The approach is particularly suited to the processing of very large data files in signal processing, image processing, experimental data processing, grid computing and computer utility applications. There has been an increasing amount of study in divisible load theory since the original work of Cheng and Robertazzi [35] in 1988. The majority of these studies develop an efficient load distribution strategy and protocol in order to achieve optimal processing time in networks with a single root processor. The optimal solution is obtained by forcing the processors over a network to all stop processing simultaneously. Intuitively, this is because the solution could be improved by transferring load if some processors were idle while

other are still busy [36]. Such studies for network topologies including linear daisy chains, tree and bus networks using a set of recursive equations were presented in [35] [37] [38] respectively. There have been further studies in terms of load distribution policies for hypercubes [39] and mesh networks [40]. The concept of equivalent networks [19] was presented for complex networks such as multilevel tree networks. Work has also considered scheduling policy with multi-installment [41], multi-round algorithms [42], independent task scheduling [43], fixed communication charges [44], detailed parameterizations and solution reporting time optimization [45] and combinatorial optimization. Recently, though divisible load theory is fundamentally a deterministic theory, a study has been done to show some equivalence to Markov chain models [9].

There has been an increasing amount of study in divisible load theory since the original work of Cheng and Robertazzi [46] in 1988. The majority of these studies develop an efficient load distribution strategy and protocol in order to achieve optimal processing time in networks with a single root processor. The optimal solution is obtained by forcing the processors over a network to all stop processing simultaneously. Intuitively, this is because the solution could be improved by transferring load if some processors were idle while other are still busy [9]. Such studies for network topologies including linear daisy chains, tree and bus networks using a set of recursive equations were presented in [46] [37] [38] respectively. There have been further studies in terms of load distribution policies for hypercubes [39] and mesh networks [40]. The concept of equivalent networks [47] was presented for complex networks such as multilevel tree networks. Work has also considered scheduling policy with multi-installment [41], multi-round algorithms [42], independent task scheduling [43], fixed communication charges [44], detailed parameterizations and solution reporting time optimization [45] and combinatorial optimization. Recently, though divisible load theory is fundamentally a deterministic the-

ory, a study has been done to show some equivalence to Markov chain models [9]. There is a limited amount of literature on divisible load modeling with multiple sources. A 2002 paper on multi-source load distribution combining Markovian queuing theory and divisible load scheduling theory is by Ko and Robertazzi [48]. In 2003 Wong, Yu, Veeravalli and Robertazzi [49] examined two source grid scheduling with memory capacity constraints. Marchal, Yang, Casanova and Robert [6] in 2005 studied the use of linear programming to maximize throughput for large grids with multiple loads/sources. In 2005, Lammie and Robertazzi [50] presented a numerical solution for a linear daisy chain network with load originating at both ends of the chain. Finally, Yu and Robertazzi examined mathematical programming solutions and flow structure in multi-source problems in 2006 [51].

A word is in order on the type of load distribution examined in this paper. Two types of load distribution, sequential and concurrent, have been studied in the literature to date when there is a single nodal source of load. Sequential distribution, where a node can only distribute to one child at a time, has received the majority of study. Under concurrent load distribution, load is distributed to all children simultaneously. Sequential load distribution is implicit in bus networks and makes sense there and in tree networks when the output port hardware is capable of only sequential distribution. In fact sequential distribution leads to interesting optimization problems involving finding the best load distribution order that minimizes solution time and maximizes speedup. If it can be supported by output port hardware, concurrent load distribution [52] [53] has a higher theoretical throughput than sequential distribution. In fact concurrent load distribution makes practical sense for large grid applications, such as the new ATLAS physics experiment at CERN. With voluminous amounts of experimental data being distributed from the CERN site in Switzerland to other continents, one would like all of the links to these distant

sites to operate simultaneously to boost the utilization of facilities. One would be hard pressed trying to explain to ones manager why sequential distribution, with only one link active at a time, should be implemented in this context. Today the scientific computation problems requiring in- tense problem-solving capabilities for problems arising from complex research and industrial application has driven in all global institution and industry segments the need for dynamic collaboration of many ubiquitous computing resources to be able to work together The problem of minimizing the processing time of extensive processing loads originated from various sources presents a great challenge that, if successfully met, could foster a range of new creative applications. Inspired by this challenge, we sought to apply divisible load theory to the problem of grid computing involving multiple sources connected to multiple sinks. So far research in this area includes [1] where tasks arrive according to a basic stochastic process to multiple nodes and [2] presents a first step technique for scheduling divisible loads from multiple sources to multiple sinks, with and without buffer capacity constraints.

Divisible load theory [3,4,5] is characterized by the fine granularity and large size of loads. There are also no precedence relations among the data elements. Such a load may be arbitrarily partitioned and distributed among processors and links in a system. The approach is particularly suited to the processing of very large data files in signal processing, image processing, experimental data processing, grid computing and computer utility applications.

There has been an increasing amount of study in divisible load theory since the original work of Cheng and Robertazzi [6] in 1988. The majority of these studies develop an efficient load distribution strategy and protocol in order to achieve optimal processing time in networks with a single root processor. The optimal solution is obtained by forcing the processors over a network to all stop processing simultaneously. Intuitively, this is because the solution

could be improved by transfer- ring load if some processors were idle while other are still busy [7]. Such studies for network topologies including linear daisy chains, tree and bus networks using a set of recursive equations were presented in [6,8,9] respectively. There have been further studies in terms of of load distribution policies for hypercubes [10] and mesh networks [11]. The concept of equivalent networks [12] was presented for complex networks such as multilevel tree networks. Work has also considered scheduling policy with multi-installment [13], multi-round algorithms [14], independent task scheduling [15], fixed communication charges [16], detailed parameterization and solution reporting time optimization [17] and combinatorial optimization [18]. Recently, though divisible load theory is fundamentally a deterministic theory, a study has been done to show some equivalence to Markov chain models [19].

Most of the earlier studies in the literature so far have assumed that the processing load originates form a single root processor. In a practical scenario, this need not be always true. In this paper, we relax this assumption and consider the case in which the processing load originates from two root processors. Our recent study [20] in this area considered two root processor model where optimal processing time solutions were obtained using a linear programming approach. This paper, unlike the previous research papers, presents closed form solutions using divisible load theory to tree networks with two load originating (root) processors. Applications include computational grids, a network of large number of loads and load sources with large number of transmission, processing and storage resources as shown in in Fig. 1. Computational grids aim at exploiting synergies that result from cooperation by sharing and aggregating distributed computational capabilities and delivering them as service. Another example of grid problems is the case of high energy and nuclear physics experiments. Here large amounts of data originate from distributed

researchers who must work closely together. The analysis of data in this type of experiments requires an increased computational power and network based computing platforms such as Globus [21] and Condor [22]. In such recently emerging platforms, scheduling and performance evaluation analysis for efficient use of distributed resources are important but challenging tasks. Effective scheduling of jobs in such a system is complicated by a number of factors including data locality, network topology and latency.

Also, we propose the use of a reduced Manhattan distance Voronoi diagram algorithm (RMDVDA) to minimize the overall processing time of these workloads by taking advantage of the processor equivalence technique. Three case studies with 10 sources of workloads are presented to illustrate the general approach for multiple sources of workloads. Experimental evaluation through simulations demonstrates that a task can be finished in the same suboptimal time and yet save over 30% of processor resources.

Data flow analysis and optimization is considered for homogeneous rectangular mesh networks. Firstly, closed-form solutions are found for the equivalence computation [19] of divisible workload originating from a single data injection point in a homogeneous mesh network. We propose a flow matrix closed-form equation to present the equivalence, which allows a characterization of the nature of minimal time solution and a simple method to determine when and how much load to distribute to processors. We also propose a rigorous mathematical proof about the flow matrix optimal solution existence and that the solution is unique. Secondly, load originating at multiple data injection points is considered. A reduced Manhattan distance Voronoi diagram algorithm (RMDVDA) is used to minimize the overall processing time of these workloads by taking advantage of a processor equivalence technique. In the first phase, a Voronoi Manhattan distance diagram is used to obtain a network cluster division. In the second phase, an efficient algorithm to ob-

tain near-optimal load distribution among processors represented by equivalent processors is proposed. The algorithm minimizes the number of processors utilized. Experimental evaluation through simulations demonstrates that a load can be processed in about the same suboptimal time and yet save about 35% - 40% of processor resources. Further, the lower bound of intuitive and heuristic algorithms are also investigated. This work has important consequences for networks on chips where the demonstrated ability to reduce the number of processors needed for a computational problem has significant consequences for chip area and chip scalability.

Four chapters are organized as follows :

- A problem introduction, notation and definitions are shown in Chapter II.

- Chapter III considers mesh network. We solve the processor equivalence problem with virtual cut-through scenario and store and forward situation. Sensitivity analysis is another topic. In considering the multi-source workload assignment problem, we propose a reduced Manhattan distance Voronoi diagram algorithm. We propose two objective functions. First, we finish the load in the same makespan, yet utilizing less processors. Another one is changing the cluster partitioning to increasing the bottleneck's capacity to increase the whole cluster's equivalence computation.

- Chapter IV investigates the toroidal network situation. The virtual cut-through switching and store and forward switching assumption are discussed. Sensitivity analysis is examined and the multi-source assignment in torus network environment is also discussed. Two objective function,finishing the load with less sources and increasing the total equivalence computation, of torus network are also implemented in this chapter.

- Chapter V mentioned some general case for example, hypercube network and general network. We give two equivalence computation flow matrix in virtual cut-through switching and store and forward switching.
- Chapter VI discuss the conclusion and future work.

Chapter 2

Problem Description

2.1 Problem Background

In large-scale data intensive problems with geographically distributed resources, load is generated from multiple sources [54] for a class of problems. It is assumed that the problem representation can be arbitrarily divided amongst the processors. Thus the problem representation is said to be "divisible". The processing of massive amounts of data on distributed and parallel networks is becoming more and more common. The problem of minimizing the processing time of extensive loads originating from a multiplicity of sources and being processed on a multiplicity of nodes presents a challenge.
In this chapter, the closed-form processor equivalence [19] [**?**] problem in the grid networks of mesh network and toroidal rectangle network is discussed. Also, the multi-source workload assignment is also taken into account.

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

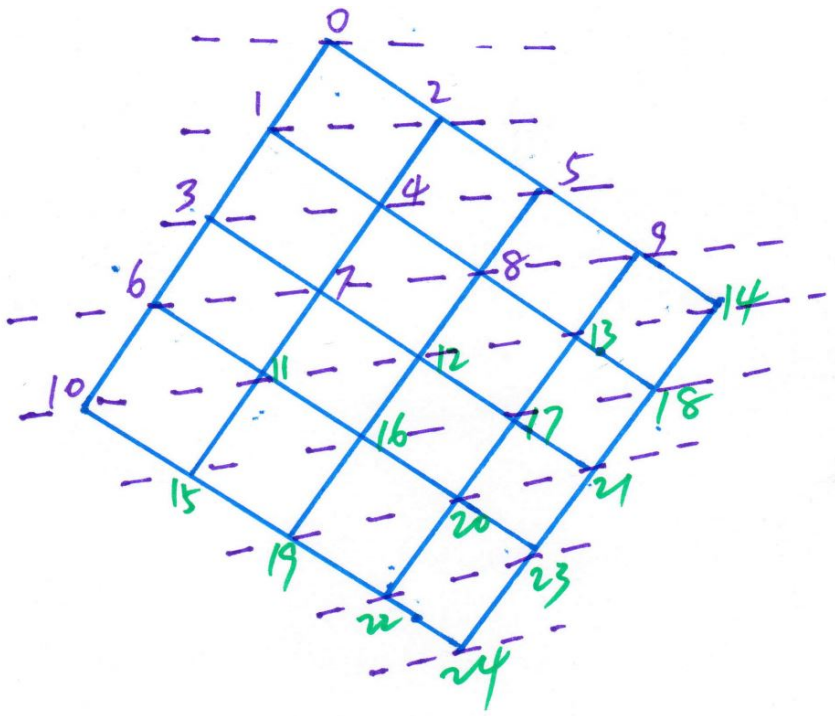


Figure 2.1: A m*n mesh network(m = 5, n = 5)

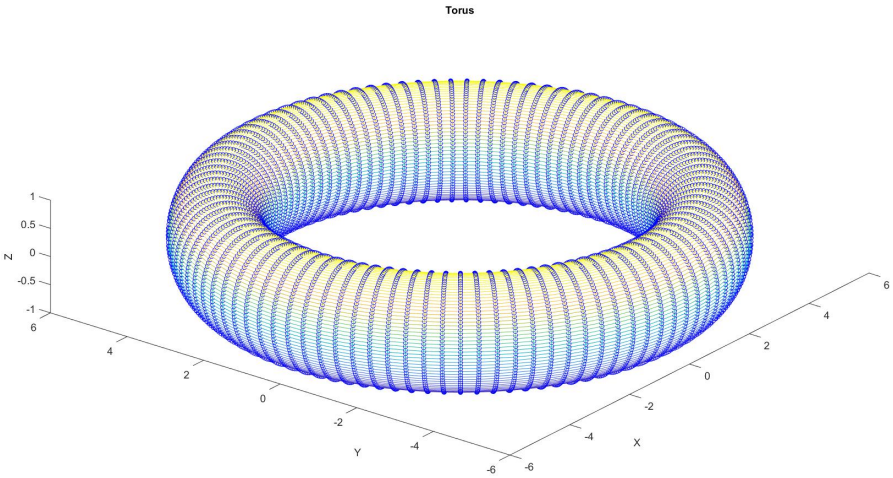


Figure 2.2: A toroidal rectangle network with grid unit cores

2.2 Definitions and Assumption

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

The following assumptions are used throughout the paper:

- Virtual cut-through [14] switching and store and forward switching is used to transmit the assigned workload between processors.
  - Under virtual cut-through switching, a node can relay the beginning bit of a message (packet) before the entire message is received.
  - Under store and forward switching, a message must be completely received by a node before it can be relayed to the next node along their transmission path.
- For simplicity, return communication is not considered.
- The communication delays are taken into consideration.
- The time costs of computation and communication are assumed to be linear function of the data size.
- The network environment is homogeneous, that is, all the processors have the same computation capacity. The link speeds between any two unit cores are identical.
- The number of outgoing ports in each processor is limited. In NOC (network on chip), the port number is fixed 4 or 5.
- The general graph's grid node's in-degree and out-degree is 5, .

The optimization objective functions are as follows :

- Single Path Communication : data transfer between two nodes follows a single path
- Homogeneous : a homogeneous (all link and processors speed are identical) is assumed.
- Equivalence computation : the problem's objective function is how to partition and schedule the workloads amongst the processors to obtain the minimum makespan (finish time).
- Multi-source assignment : how all processors can finish processing a unit 1 workload at the same time utilizing fewer processors.

To achieve the minimum time solution is obtained by forcing the processors over a network to stop processing simultaneously. Intuitively, this is because the solution could be improved by transfer load from some busy processors to idle ones [32] [36].

2.2.1 Notions

The following notations and definitions are utilized:

- P_i: The ith processor. $0 \leq i \leq m * n - 1$.
- L_i: The ith work load. $1 \leq i \leq k$.
- D_i: The minimum number of hops from the processor P_i to the data load injection site L.
- α_0: The load fraction assigned to the root processor.
- α_i: The load fraction assigned to the ith processor.
- $\hat{\alpha_i}$: The load fraction assigned to each professor on the ith layer $i \in 0 \cdots (k-1)$.

- ω_i: The inverse computing speed on the ith processor.
- ω_{eq}: The inverse computing speed on an equivalent node collapsed from a cluster of processors.
- r: The rank of flow matrix.
- z_i: The inverse link speed on the ith link.
- T_{cp}: Computing intensity constant. The entire load is processed in time $\omega_i T_{cp}$ seconds on the ith processor.
- Sp_i: The speedup of ith Voronoi cell $i \in 0 \cdots (k-1)$.
- Sp_b: The speedup of the bottleneck cell.
- T_{cm}: Communication intensity constant. The entire load is transmitted in time $z_i T_{cm}$ seconds over the ith link.
- $\hat{T_f}$: The finish time of the whole processor network. Here $\hat{T_f}$ is equal to $\omega_{eq} T_{cp}$.
- T_b: The finish time for the bottleneck cell.
- T_f: The finish time for the entire divisible load solved on the root processor. Here T_f is equal to $1 \times \omega_0 T_{cp}$, that is $\omega_0 T_{cp}$.
- α_{j,l_j}: The data fraction of each processor on the l_j layer of jth Voronoi cell.
- $T_{f,i}$: The finish time for the ith processor, $i \in 0 \cdots (m * n - 1)$.
- S_j: The processor's number of jth cell.
- $\sigma = \frac{zT_{cm}}{\omega T_{cp}}$: The ratio between the communication speed to the computation speed, $0 < \sigma < 1$ [32] [55].

- $\hat{T_{j,f}}$: The finish time of jth cell.
- In multi-source situation, $\sum_{i=1}^{k} L_i = 1$
- $\sum_{i=0}^{m*n-1} \alpha_i = 1$
- $Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

Chapter 3

Mesh Network Virtual Cut-through Switching

3.1 Virtual Cut-through Switching Scenario

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

First we consider about the $2 * 2$ mesh network, which can be generalized to a $2 * n$ mesh network. After, we analyze a more general case $m * n$ mesh network and obtain a general closed-form matrix presentation. Finally, we give a key methodology to address this type of question. In addition, different single data injection position, such as the corner, boundary and inner grid are also discussed.

3.1.1 Data Injection on The Corner Processor

2*2 Mesh Network

The load 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.

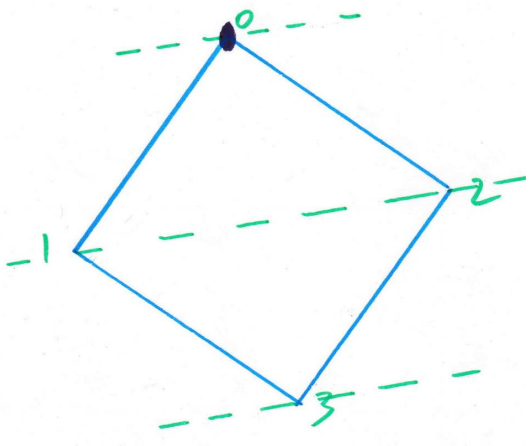


Figure 3.1: The 2*2 mesh network and the root processor is P_0

The processor P_0, P_1 and P_2 start to process its respective fraction at the same time. This includes P_1 and P_2 as they are relayed load in virtual cut-through mode at $t = 0$. The processor P_3 starts to work when the α_1 and α_2 complete transmission. That is, the link $0 - 1$ and $0 - 2$ are occupied transmitting load to processor 1 and 2, respectively and only transmission to 3 when that is finished.

According to the divisible load theory [33], we obtain the timing diagram Fig. 3.2.

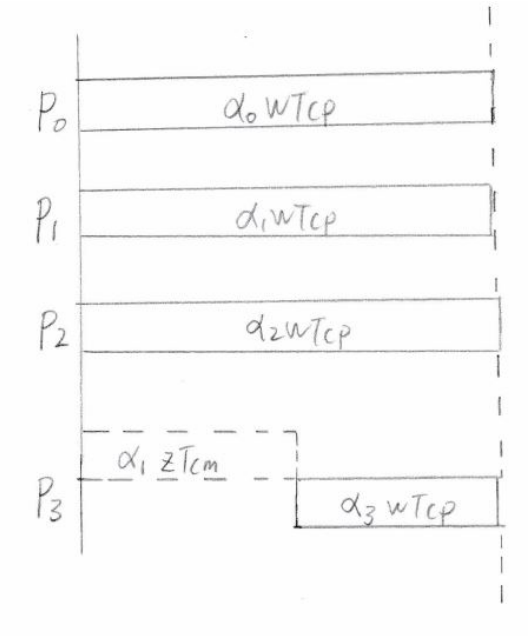


Figure 3.2: The timing diagram for 2*2 mesh network and the root processor is P_0

Here in the Gantt-like timing diagram communication appears above each axis and computations appears below the each axis. Let's assume that all processors stop computing at the same time in order to minimize the makespan [36].

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

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

The group of equations are represented by the matrix form:

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

The matrix is represented as $A \times \alpha = b$. A is named as the ***flow matrix***. Here because of symmetry $\alpha_1 = \alpha_2$, so α_2 is not listed in the matrix equations.

Finally, the explicit solution is:

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

The simulation result is illustrated:

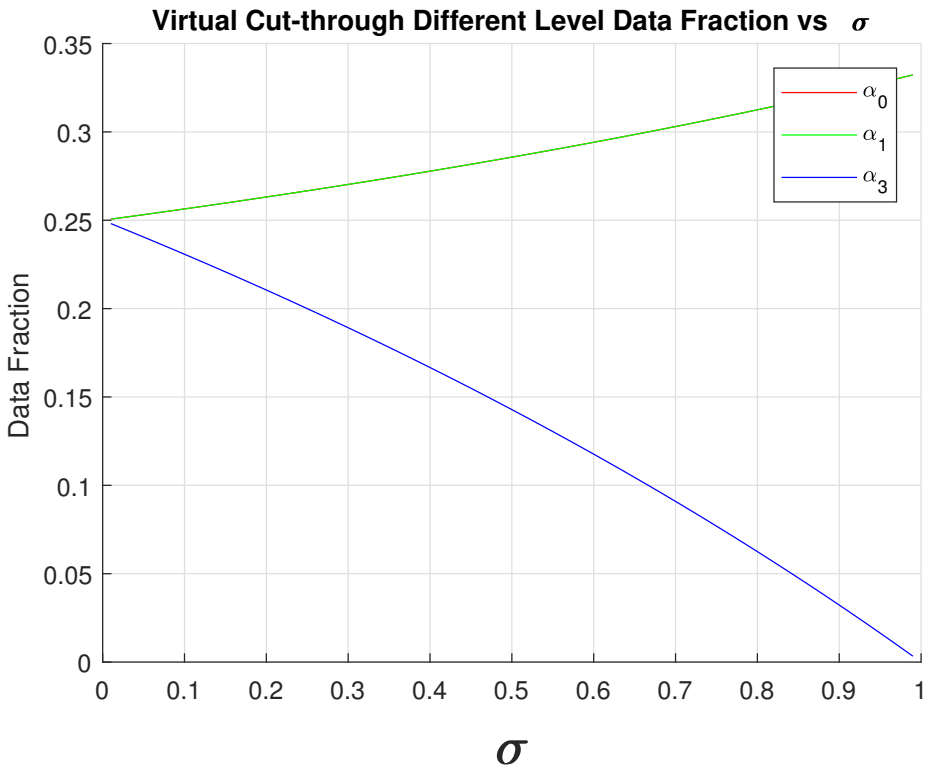


Figure 3.3: 2*2 mesh network. α_0, α_1, α_2, α_3 value curve

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

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

The equivalence inverse speed of a a single processor is w_{eq}, that can replace the original network as

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = 4 - \sigma$$

2*3 Mesh Network

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

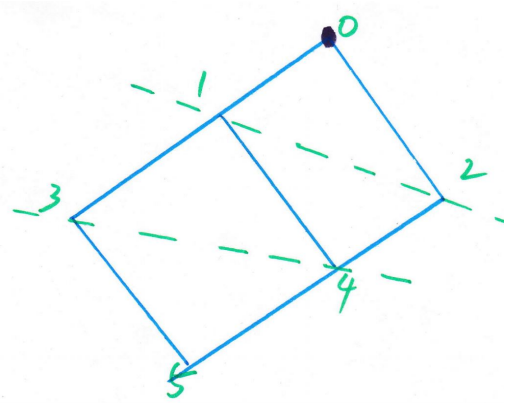


Figure 3.4: The 2*3 mesh network and the data injection happens on corner processor P_0

Here P_0, P_1 and P_2 start processing at the same time. Processor P_3 and P_4 start to work when they receive the data from processor P_1, P_2, after the $0-1$ and $0-2$ link become available to transfer load to P_3 and P_4
The last processor P_5 doesn't start to execute until the work load fraction α_0, α_1, α_2, α_3, α_4 are transmitted completed. According to the divisible load theory [33], we obtain the timing diagram Fig. 3.5.

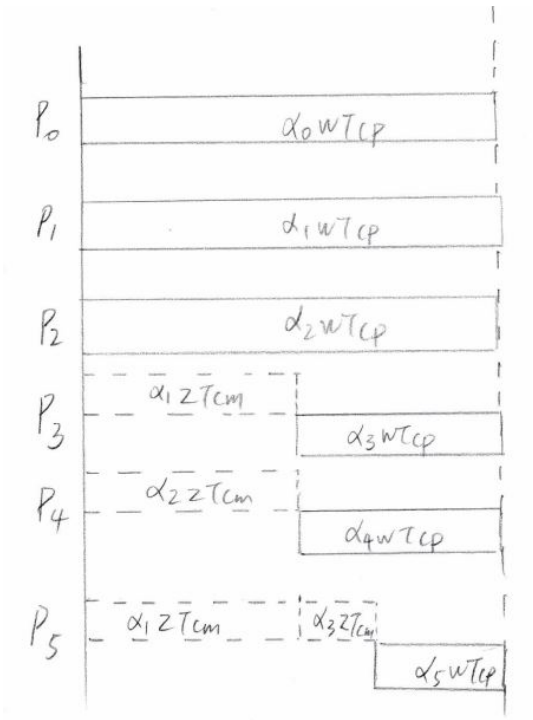


Figure 3.5: The timing diagram for a 2*3 mesh network and the data injection happens on processor P_0

The timing diagram equations as follows:

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

The flow matrix closed-form formula is:

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

The explicit solution is:

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

The α calculation result are shown in Fig. 3.6. P_0, P_1 and P_2 have the same fraction so the curve of α_0 and α_1 coincide. Because of symmetry, $\alpha_1 = \alpha_2$ and $\alpha_3 = \alpha_4$. If the communication time is zero, each of the six processors receive $\frac{1}{6}$ of the load which is in the figure. If the communication delay equals computation time $\frac{1}{3}$ of the load goes to each of P_0 P_1 and P_2. It is uneconomical to distribute load to P_3, P_4 and P_5.

One also has for the equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

For speedup:

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

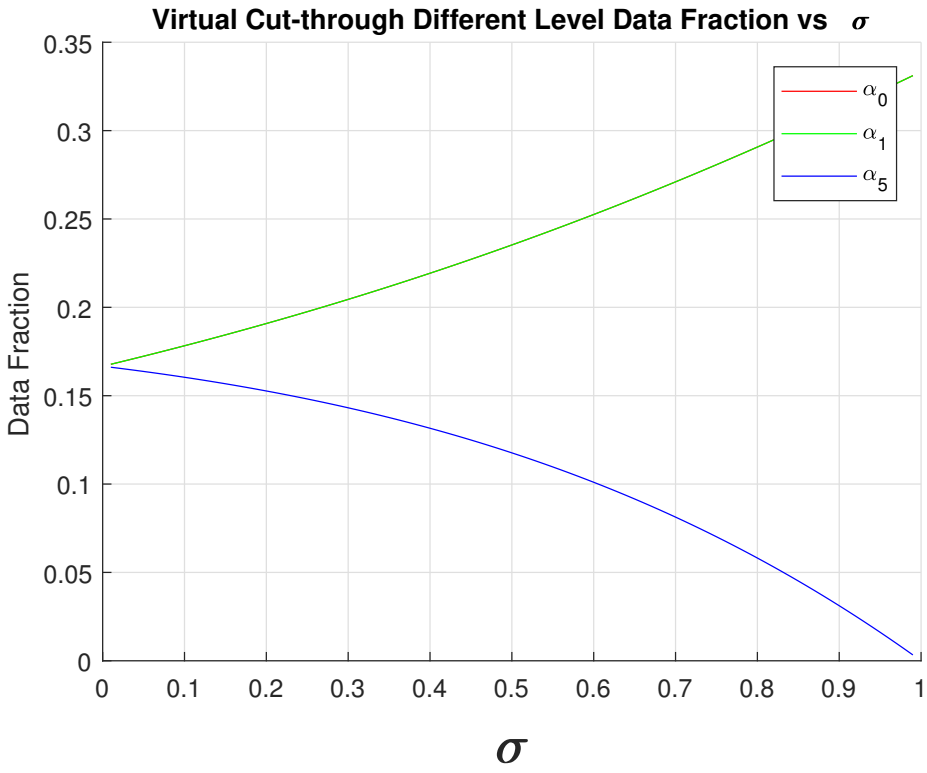


Figure 3.6: 2*3 mesh network. α_0, α_1, α_3, α_5 value curve.

2*N Mesh Network

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

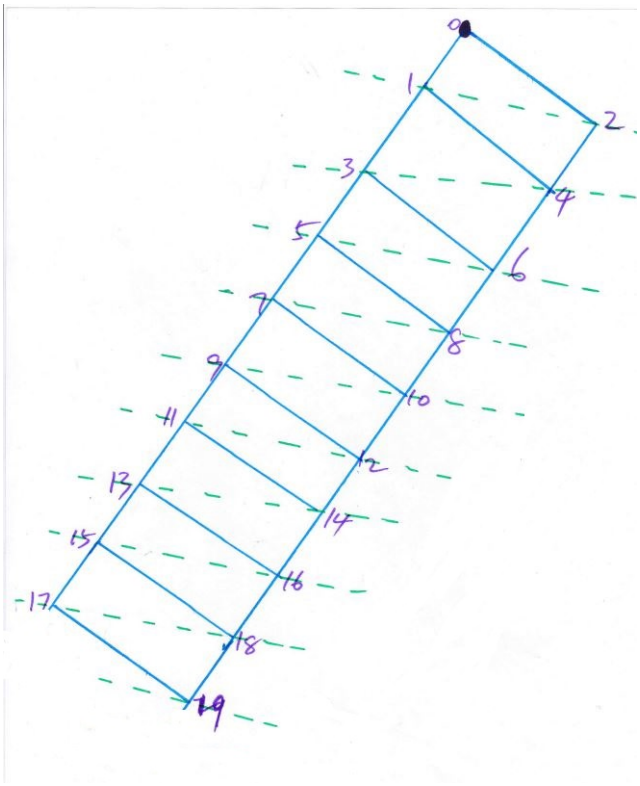


Figure 3.7: 2*n (n = 10) mesh network and the workload happens on P_0

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

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

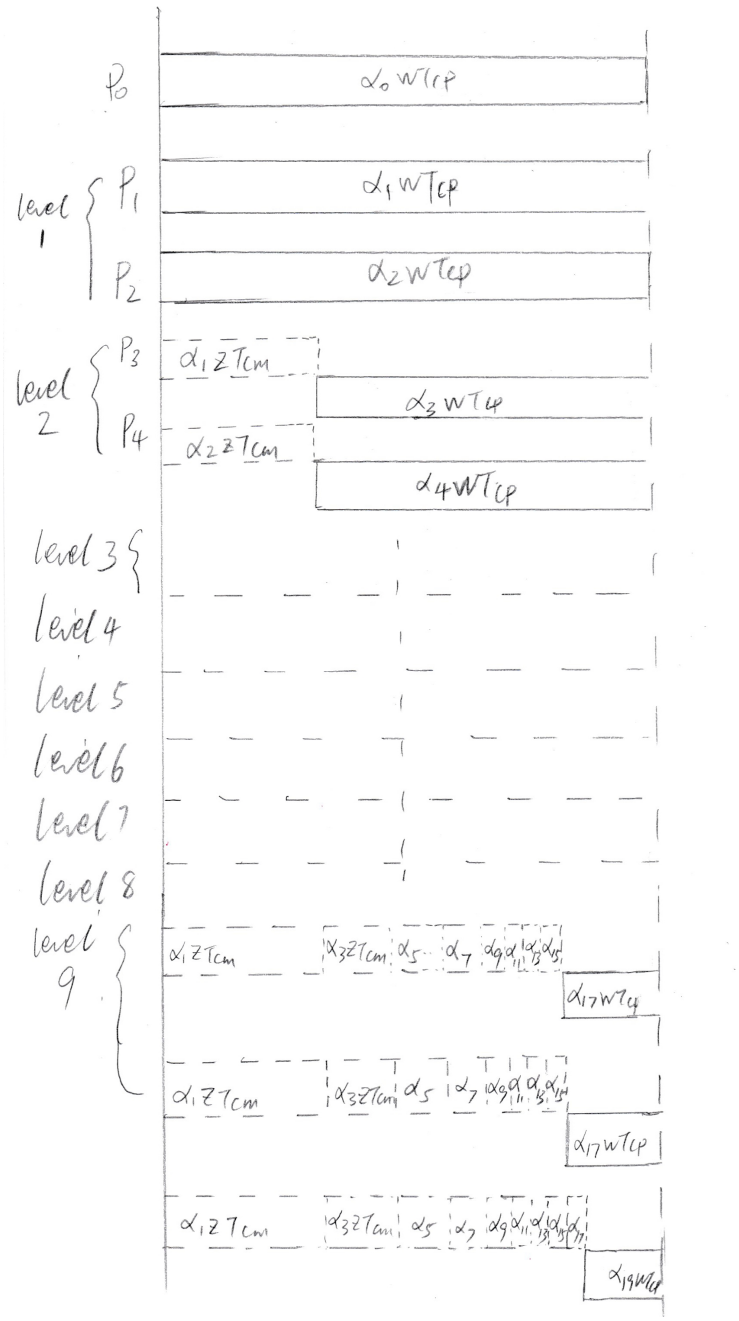


Figure 3.8: The timing diagram for 2*10 mesh network and the data injection happens on P_0

The equations are presented as:

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

The flow matrix closed-form is shown:

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

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

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

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

Especially,

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

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

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

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

Finally, the speedup is:

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

.

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

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

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

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

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

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

Considering the matrix $\hat{C}$

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

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

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

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

m*n Mesh Network

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

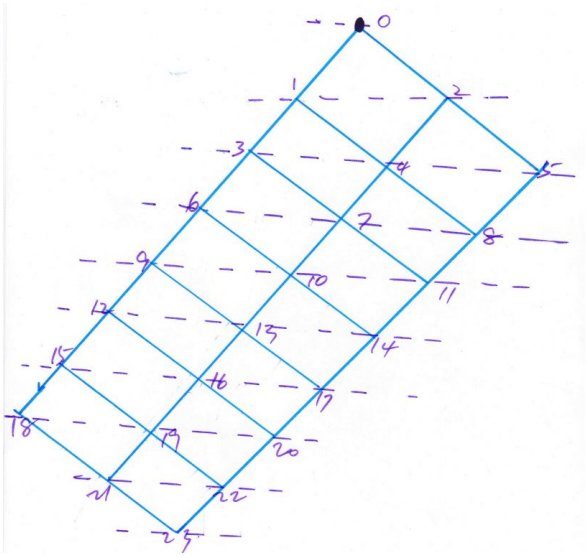


Figure 3.9: 3*8 mesh network. The data injection position is P_0

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

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

Also, the flow matrix equations for Fig. 2.1:

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

We use the similar method to prove $\det A \neq 0$. The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

so the speedup is:

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

.

The first row in flow matrix describe the number of cores on each D_i. For example, there is 1 core with 0 hop distance (D_0) with load site L. There are 2 cores with 1 hop distance (D_1) with load site L. There are 3 cores with 2 hops distance (D_2) with load site L, and so on.

The number of rows means the number of different type processor data fraction.

3.1.2 Data Injection On The Boundary Processor

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

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

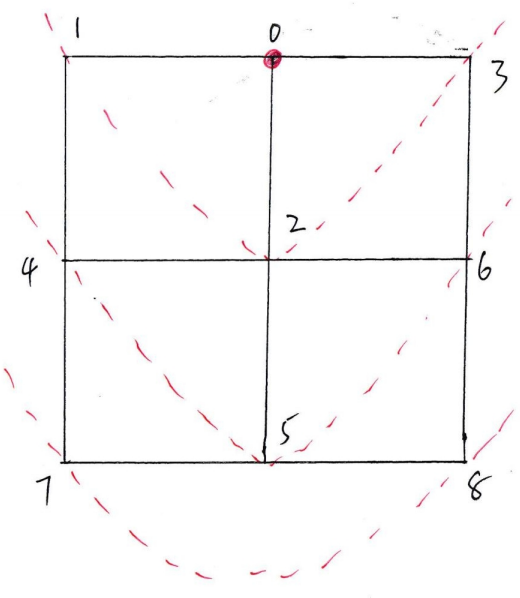


Figure 3.10: The 3*3 mesh network and the root processor is P_0

The timing diagram is Fig. 3.11:

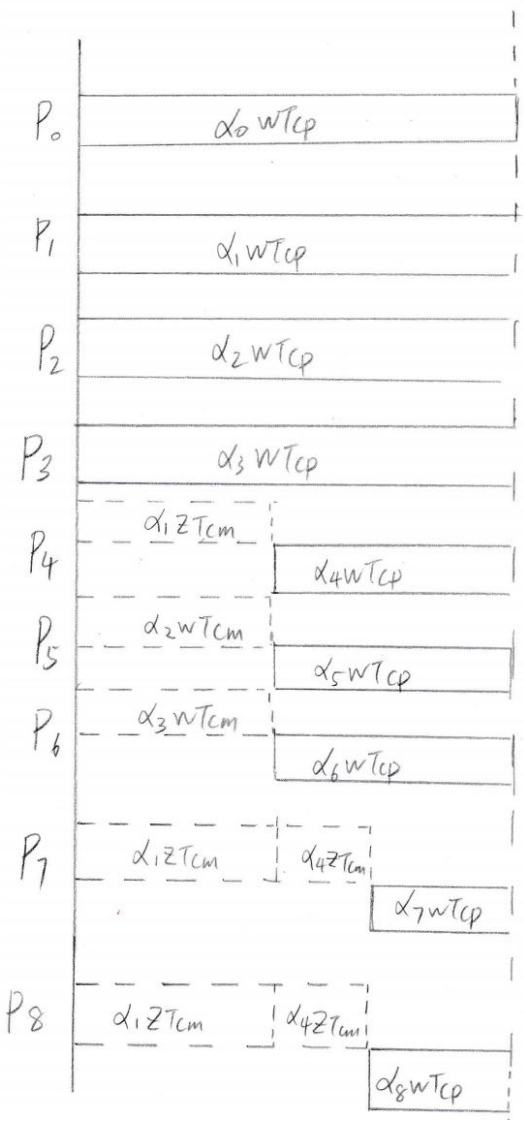


Figure 3.11: The timing diagram for 3*3 mesh network and the data injection occurs on P_0

P_0 P_1 P_2 and P_3 commence processing at $t = 0$ axis timing diagram. P_1 P_2 and P_3 receive load in virtual cut-through mode from P_0. Once that is finished, link $0-1$, $0-2$ and $0-3$ is used to forward load to nodes P_4, P_5 and P_6 in virtual cut-through mode. Once that is finished, the link that have been so far are used to forward load to P_7 and P_8 in virtual cut-through mode.
The equations are:

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

And the flow matrix form is :

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

The explicit solution is:

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

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

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

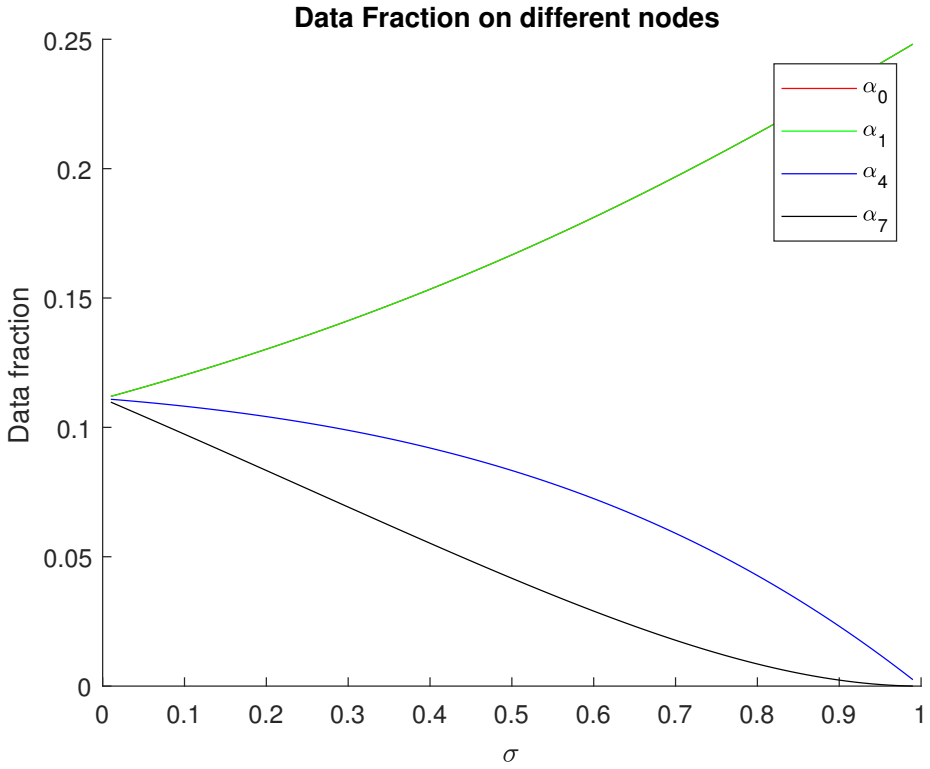


Figure 3.12: The data fraction simulation result of 3*3 mesh network and the data injection happens on the boundary P_0

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

3.1.3 Data Injection On The Inner Grid Processor

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

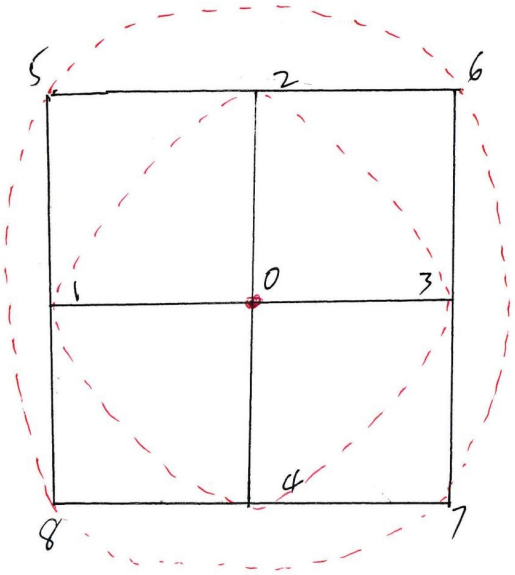


Figure 3.13: 3*3 mesh network. The data injection position is inner grid point P_0

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

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

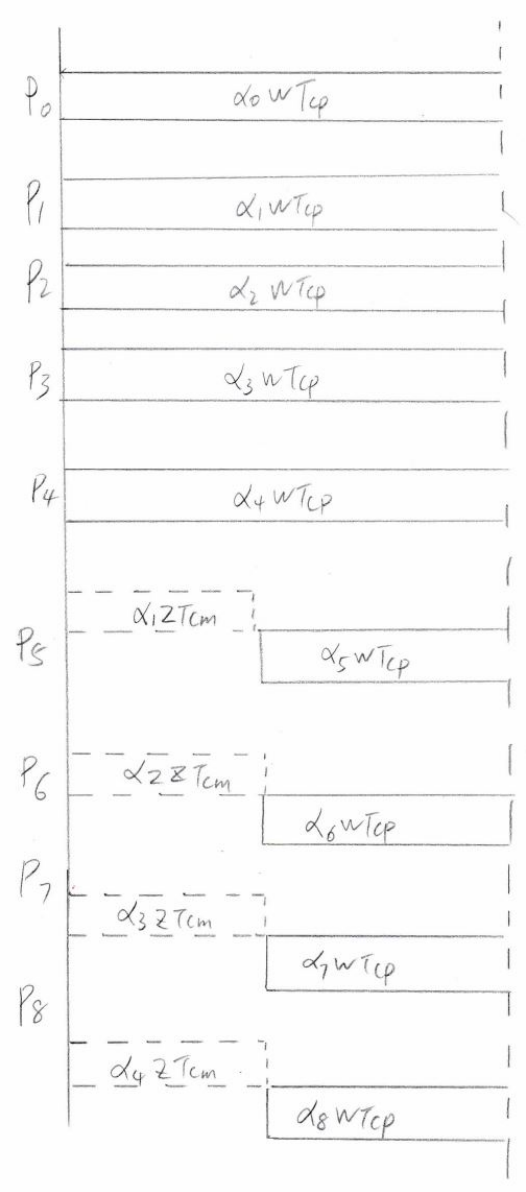


Figure 3.14: The timing diagram for 3*3 mesh network and the data injection is inner grid P_0

The group of equations are:

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

The flow matrix form is :

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

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

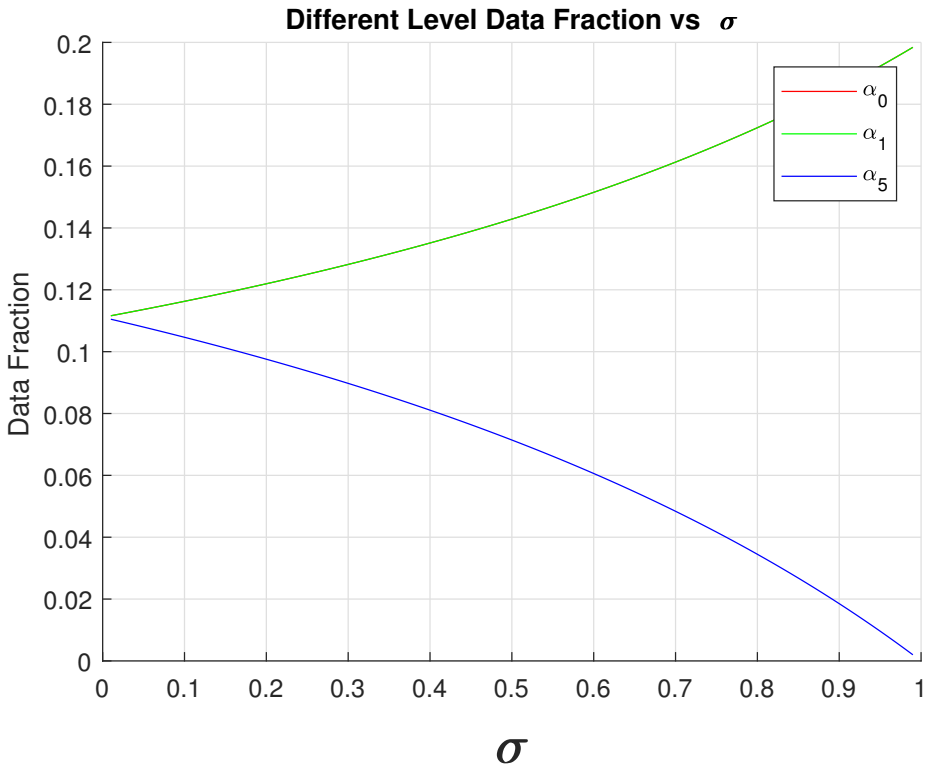


Figure 3.15: 3*3 mesh network. The data injection position is inner grid point P_0

The explicit solution is:

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

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

If the communication time is zero (very fast communication), each of nine processors get $\frac{1}{9}$ of the load. If communication time equals computation time, each of processors P_0, P_1, P_2, P_3 and P_4 get $\frac{1}{5}$ of the load.

3.1.4 Sensitivity Analysis Virtual Cut-through Switching Scenario

From Chapter 2, we know the speedup is :

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

.

Data Injection On The Corner Processor

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

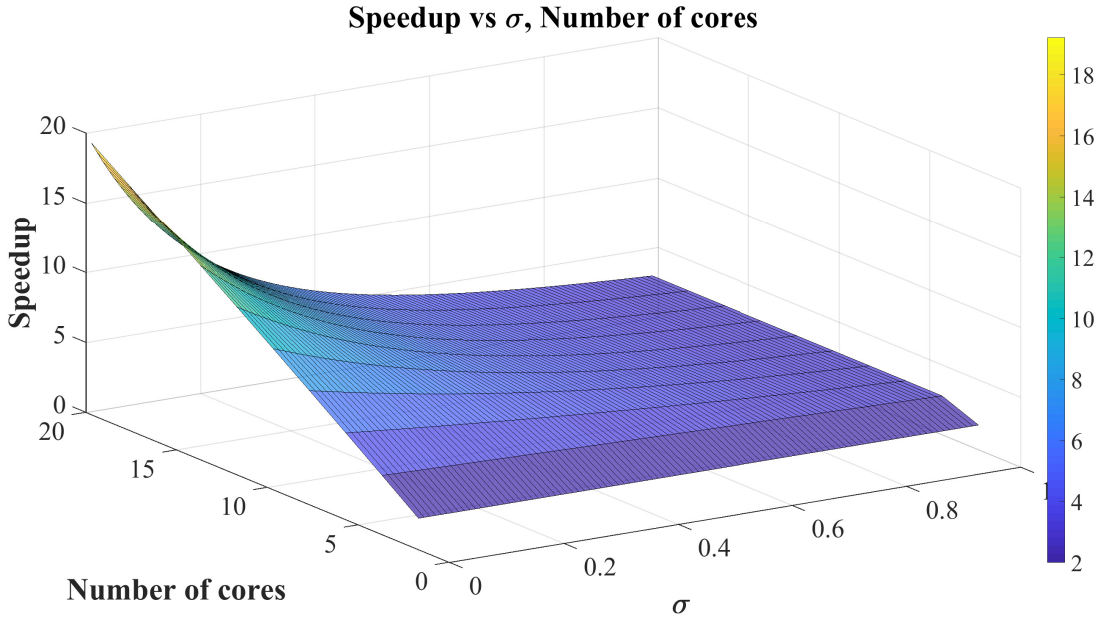


Figure 3.16: Sensitivity analysis result of 2*n mesh network (where $n = 1 \cdots 10$)

One can see speedup increases with an increasing number of core (i.e. processor) and also increases with decreasing σ (that is increasing communication speed relative to computation speed).

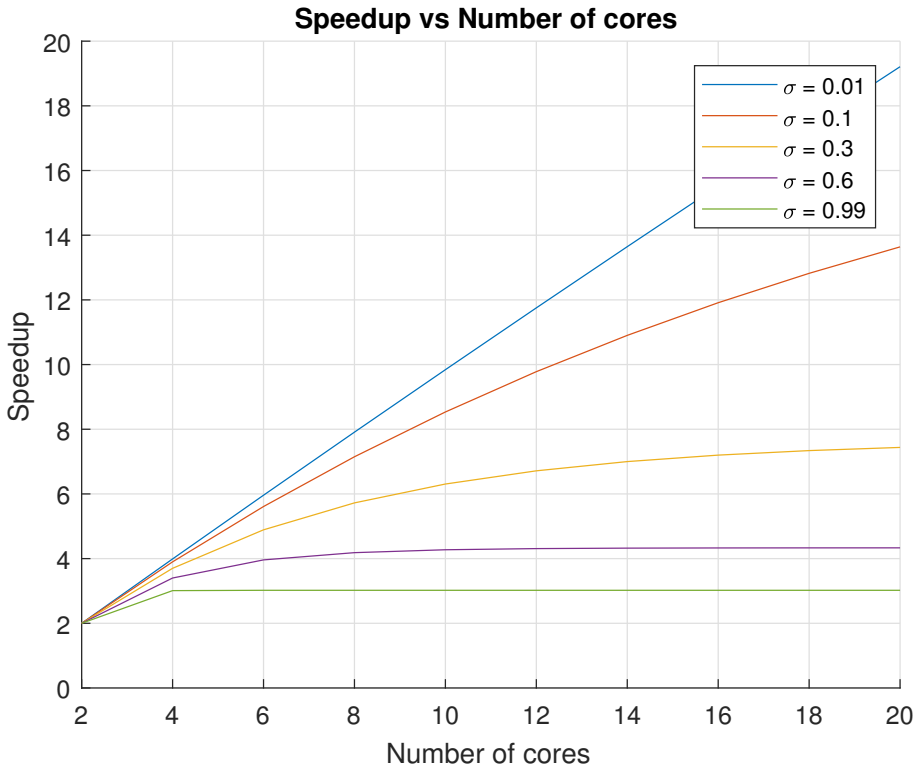


Figure 3.17: Speedup curves of 2*n mesh network (where $n = 1 \cdots 10$)

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

Data Injection On The Boundary Processor

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

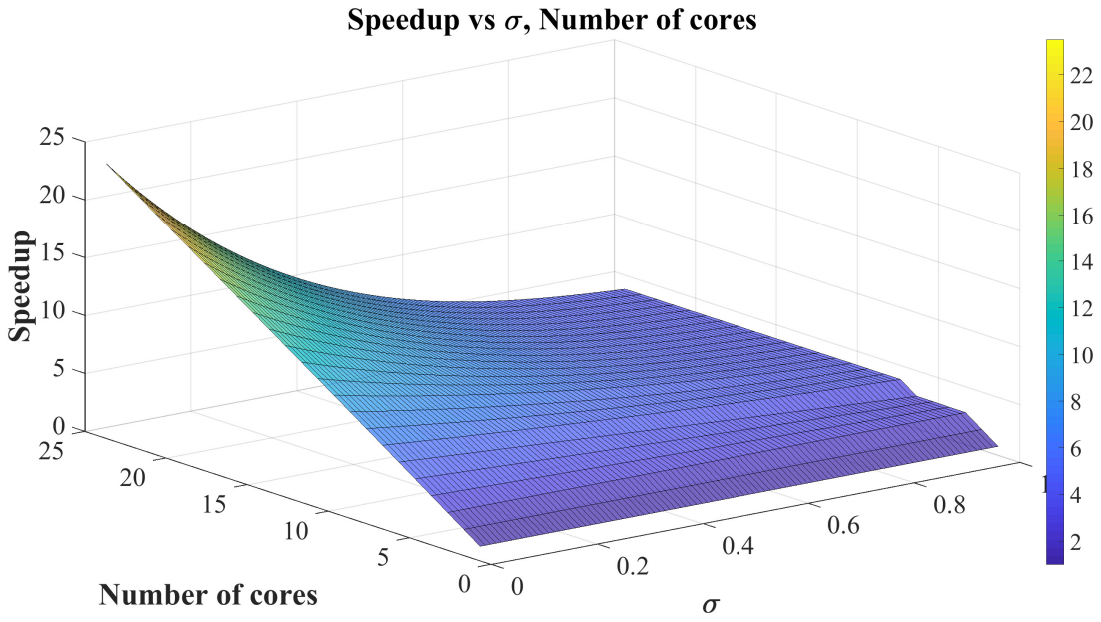


Figure 3.18: Sensitivity analysis result of $3*n$ mesh network and the injection position on boundary processor P_2 (where $n = 1 \cdots 8$)

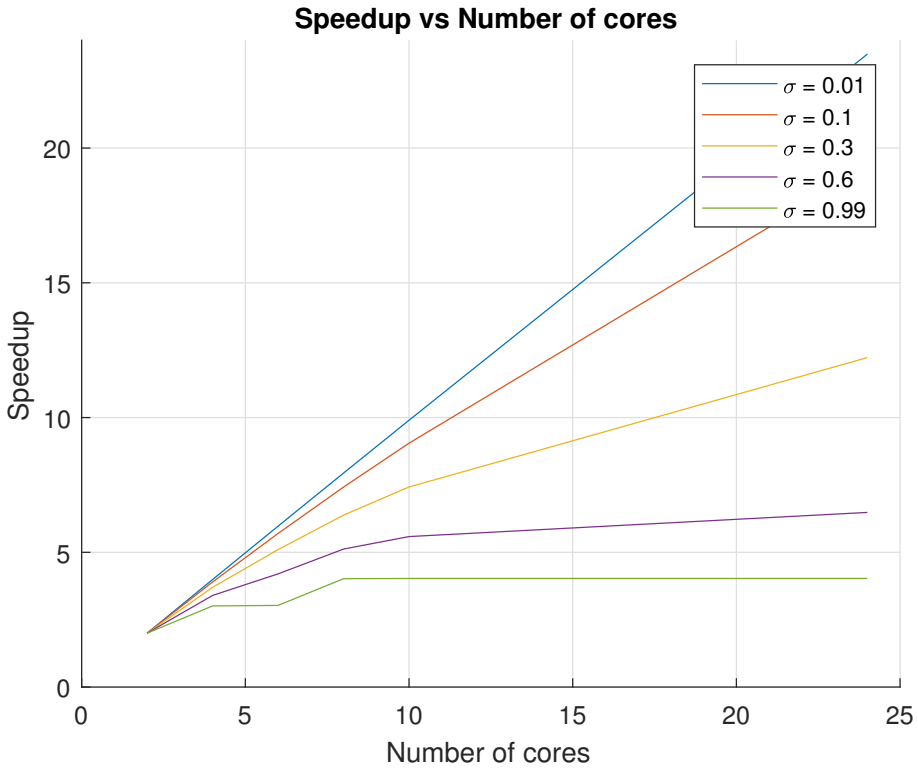


Figure 3.19: Speedup curve of 3*n mesh network (where $n = 1 \cdots 8$)

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

Data Injection On The Inner Grid Processor

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

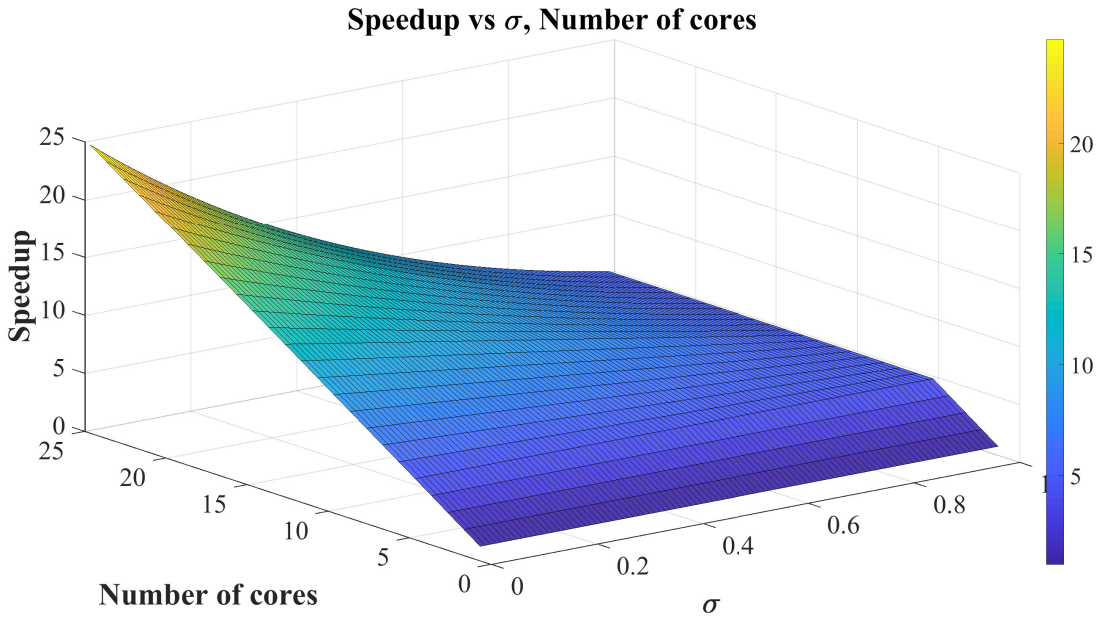


Figure 3.20: Sensitivity analysis result of $5 * n$ mesh network and the injection position on inner processor P_{12} (where $n = 1 \cdots 5$)

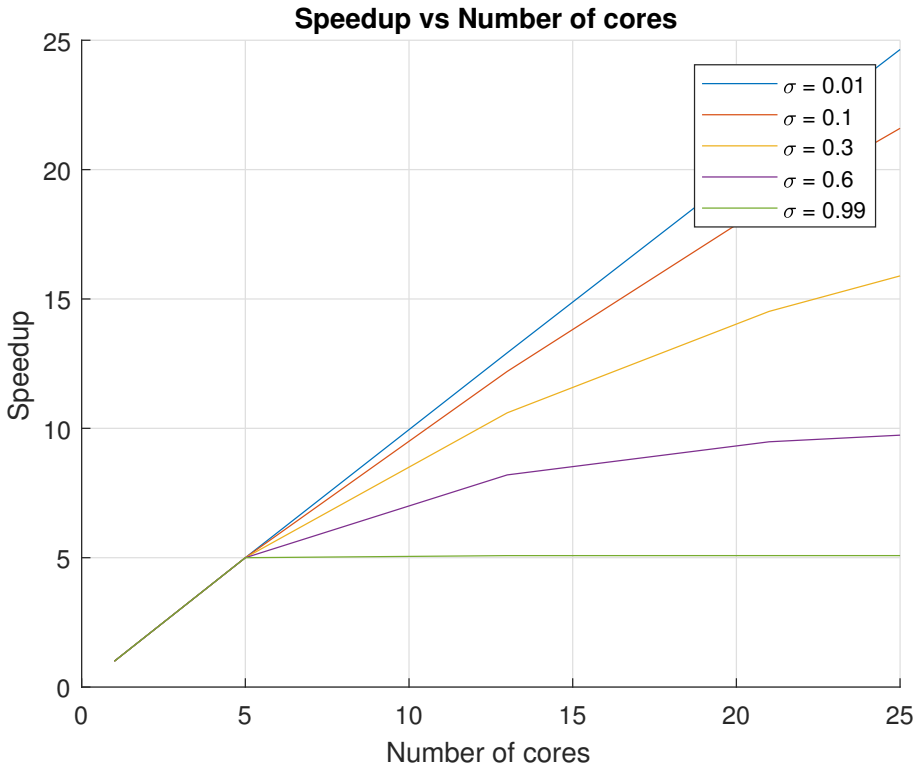


Figure 3.21: Speedup curves of 5*n mesh network Fig. 2.1 (where $n = 1 \cdots 5$)

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

3.1.5 Multi-source Uniform Data Fraction

We extend the single source assignment to the multi-source assignment problem [18] [**?**]. For each processor, we concentrate on the processors' geographical location P_i, and assigned data fraction α_i.

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

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

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

Situation I

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

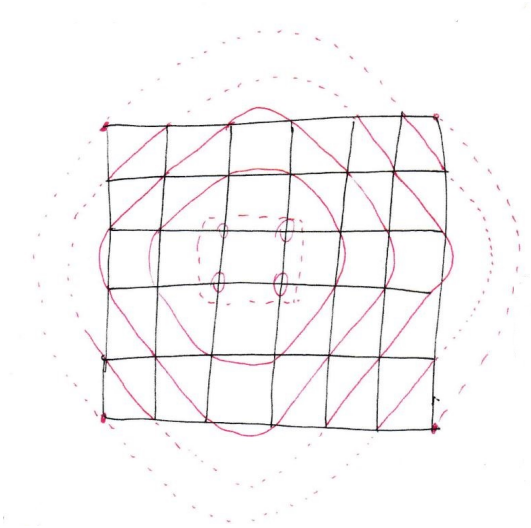


Figure 3.22: Data injection consists of a connected subgraph of G

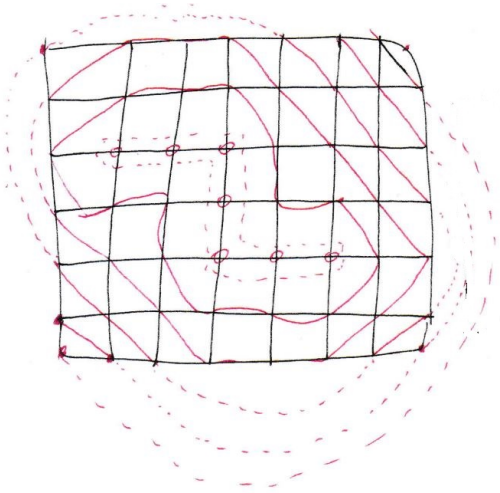


Figure 3.23: Data injection consists of a connected subgraph of G

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

Our objective is to propose a general algorithm framework to minimize the makespan and give quantitative model analysis utilizing the flow matrix.

$$\min \quad \hat{T_f} \tag{3.90}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.91}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.92}$$

$$T_{f,i} = T_{f,0}, \qquad i \in 1 \cdots (m * n - 1) \tag{3.93}$$

$$\hat{\alpha_{j,0}} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.94}$$

$$\vdots, \tag{3.95}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l_j}-2}) z T_{cm} + \alpha_{j,\hat{l_j}-1} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.96}$$

This algorithm is named as ***Equivalence Processor Scheduling Algorithm (EPSA)***.

Algorithm 1 Equivalence Processor Scheduling Algorithm (EPSA)

Input: k data injection positions

Output: $m * n$ processor data fractions α_i

Collapse the data injection processors into one "big" equivalent processor [19].

Calculate $m * n$ processor's D_i.

Obtain the flow matrix A.

Calculate $m * n$ processors data fraction α_i.

In term of the time complexity :

- The time complexity of calculating the determinant is $O(r^3)$ with Gaussian elimination or LU decomposition.

- The time complexity of calculating the flow matrix A_i is $O(k * m * n)$. k is the number of data injection location.
- The total time complexity is $O(k * r^3)$.

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

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.97}$$

The simulation result illustrates as follows:

This simulation says the best performance happens on value $\sigma \leq 0.05$, which hits about 42 times speedup as all 42 processors in the network are engaged in processing networks. There are 12 processors at the start stage. So if $\sigma \approx 1$, that is, communication time equals computation time, the network achieves about 12 times speedup performance, only those 12 processors do processing.

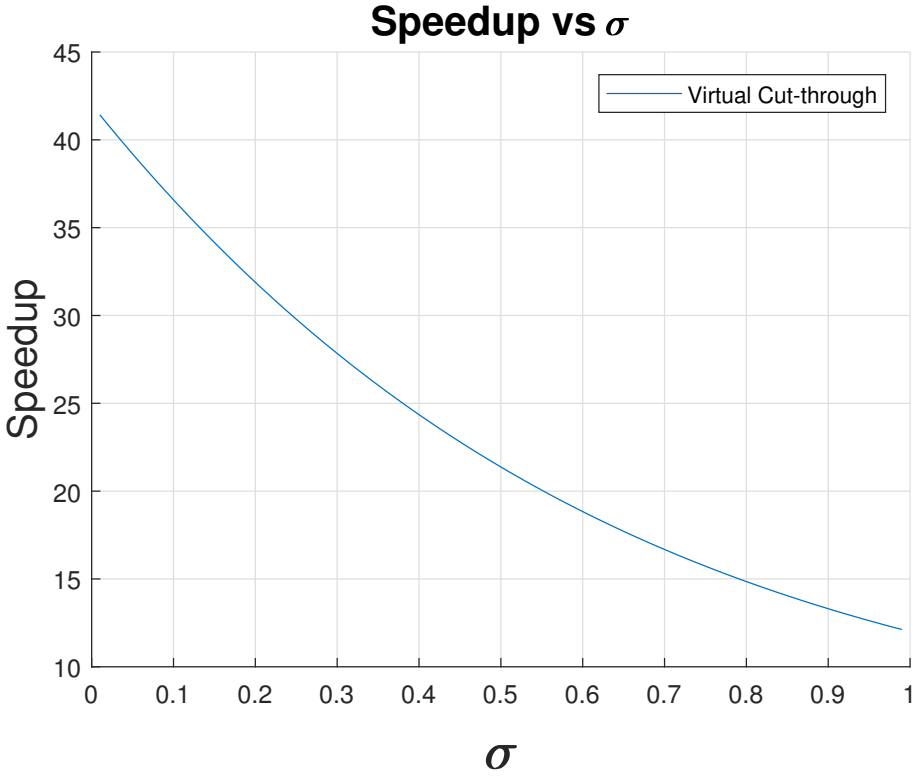


Figure 3.24: Connected subgraph Fig. 3.22 speedup vs σ

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

$$\begin{bmatrix} 7 & 14 & 15 & 10 & 6 & 3 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.98}$$

The simulation result illustrates as follows:

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

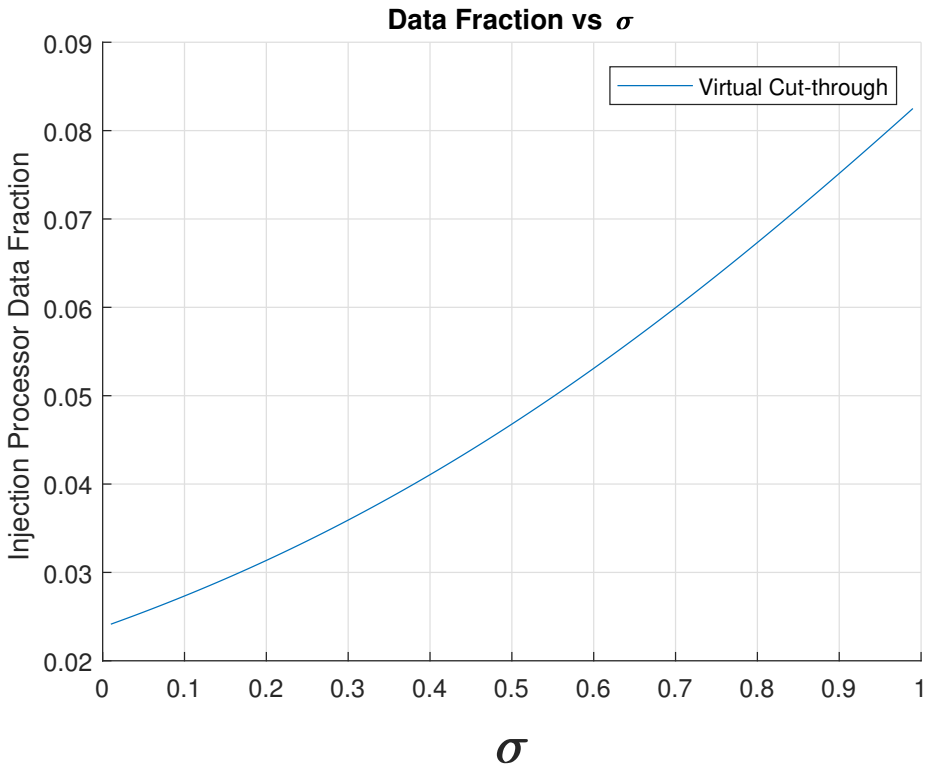


Figure 3.25: Connected subgraph Fig. 3.22's processor data fraction vs σ

which hits about 56 times speedup as there are 56 processors engaged over the entire network. There are 21 processors in the start stage. So if $\sigma \approx 1$, the network achieves 21 times speedup performance as only those 21 processors do processing.

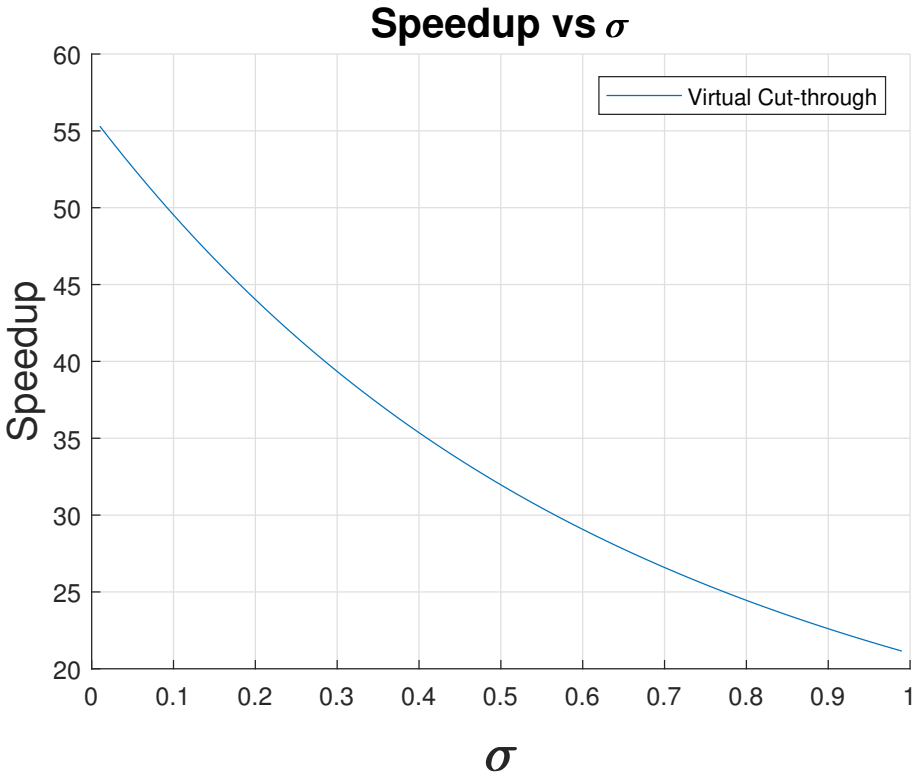


Figure 3.26: Connected subgraph Fig. 3.23 speedup vs σ

Situation II

Now the data injections don't consist of a connected subgraph of G. Jia [18] proposes a genetic algorithm, which utilize a novel Graph Partitioning (GP) scheme to partition the network such that each source in the network gains a portion of network resources and then these sources cooperate to process their loads.

Relevant divisible load applications include, for example, big file transmission, HPC scientific computation, like BlueGene [56] or Hadoop jobs.

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

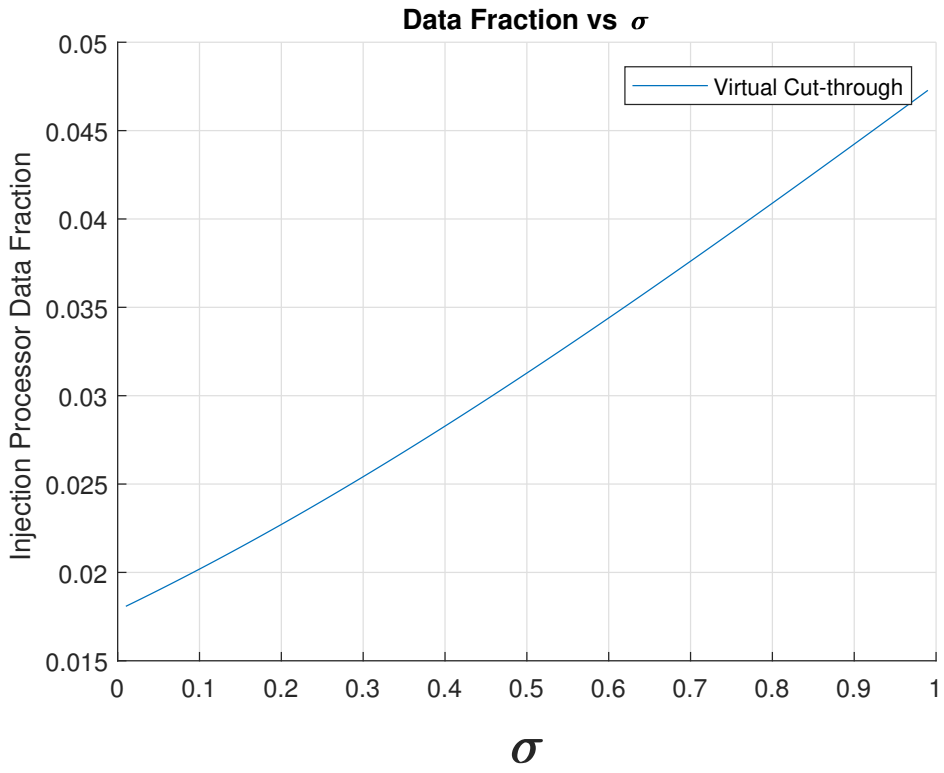


Figure 3.27: Connected subgraph Fig. 3.23's processor data fraction vs σ

the chip power computation chip. Power computation is a limiting factor to scalability in such NOCs.

Definition 3.1.1. *Voronoi diagram [57]*
Let S denote a set of n points (called sites) in the plane. For two distinct sites $p, q \in S$, the dominance of p over q is defined as the subset of the plane being at least as close to p as to q. Formally,

$$dom(p,q) = \{x \in R^2 \quad | \quad \delta(x,p) \quad \leq \quad \delta(x,q)\}$$

for δ denoting the euclidean distance function. Clearly, $dom(p,q)$ is a closed half plane bounded by the perpendicular bisector of p and q.

This bisector separates all points of the plane closer to p from those closer to q and will be termed the separator of p and q. The region of a site $p \in S$ is the portion of the plane lying in all of the dominances of p over the remaining sites in S. Formally,

$$reg(p) = \bigcap_{q \in S-p} dom(p,q)$$

For each seed there is a corresponding region consisting of all points closer to that seed than to any other.

Definition 3.1.2. *Manhattan distance Voronoi diagram Unlike most Voronoi diagrams, which use the $L2$ (Euclidean distance) metric,* ***Manhattan distance Voronoi diagrams*** *use the $L1$ metric.*

In this thesis, the intuitive algorithm is to implement the graph partitioning by the Manhattan distance Voronoi diagram.

Manhattan Distance Voronoi Diagram Algorithm

Our objective is to propose an intuitive algorithm to minimize the makespan and give quantitative model analysis utilizing the flow matrix.

$$\min \quad \hat{T_f} \tag{3.99}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.100}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.101}$$

$$T_{f,i} = \hat{T_{j,f}}, \qquad i \in 0 \cdots (S_j - 1) \tag{3.102}$$

$$\hat{\alpha_{j,0}} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.103}$$

$$\vdots, \tag{3.104}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l_j}-2}) z T_{cm} + \alpha_{j,\hat{l_j}-1} w T_{cp} = \hat{T_{j,f}},\ j \in 0 \cdots (k-1) \tag{3.105}$$

The intuitive algorithm is named as ***Manhattan Distance Voronoi Diagram Algorithm***:

Algorithm 2 Manhattan Distance Voronoi Diagram Algorithm (RMDVDA)

Input: k data injection positions

Output: $m * n$ processor data fractions

Calculate k Voronoi cells with Manhattan distance.

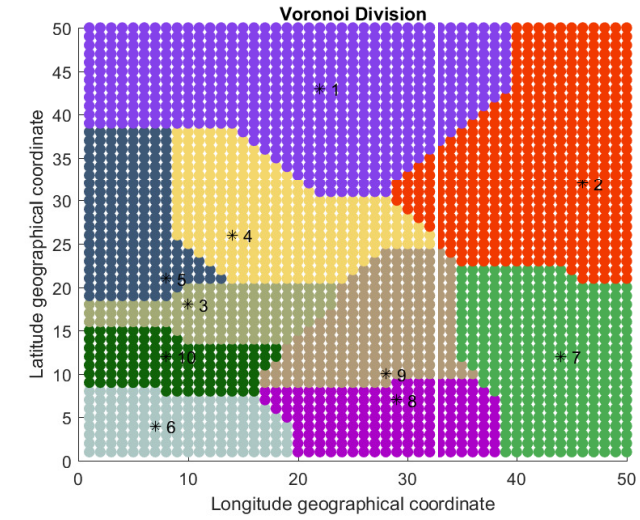
Calculate k flow matrix A_i.

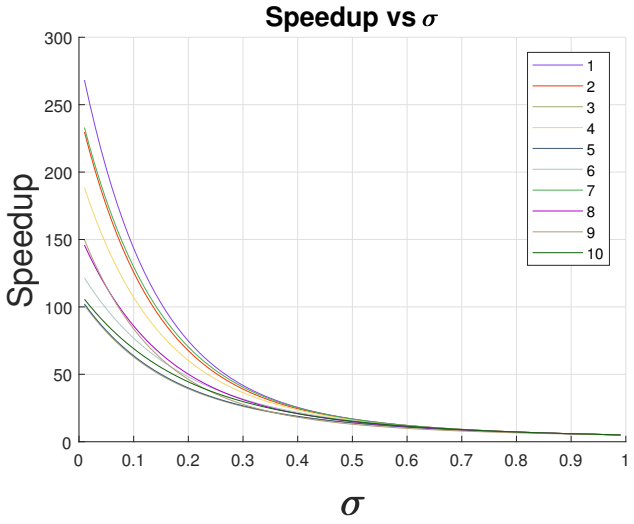
Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves.

Manhattan distance Voronoi diagram division shows 10 Voronoi cells division.

Table 3.1: The 10 Manhattan distance Voronoi Cells and speedup curves





After investigation, we find the makespan depends on the bottleneck makespan. In other words, if other cells consist of more processors than the bottleneck cell, it does not help to minimize the makespan.

Reduced Manhattan Distance Voronoi Diagram Algorithm

Our objective is to propose a heuristic algorithm to minimize the makespan and give quantitative model analysis utilizing the flow matrix.

The merits of new algorithm is finishing the task within the same makespan as MDVDA, yet utilizing less processor resources.

$$\min \quad \hat{T_f} \tag{3.106}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.107}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.108}$$

$$T_{f,i} = \hat{T_{j,f}}, \qquad i \in 0 \cdots (S_j - 1) \tag{3.109}$$

$$\hat{\alpha_{j,0}} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.110}$$

$$\vdots, \tag{3.111}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l_j}-2}) z T_{cm} + \alpha_{j,\hat{l_j}-1} w T_{cp} = \hat{T_{j,f}}, \; j \in 0 \cdots (k-1) \tag{3.112}$$

$$Sp_i = Sp_b + \epsilon, \qquad i \in 0 \cdots (k-1) \tag{3.113}$$

$$\epsilon \geq 0, \tag{3.114}$$

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

Algorithm 3 Reduced Manhattan Distance Voronoi Diagram Algorithm (RMDVDA)

Input: k data injection positions

Output: $m * n$ processor data fractions

Calculate k Voronoi cells with Manhattan distance.

Calculate k Voronoi cells' radius R_i.

Calculate k flow matrix A_i.

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

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

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

Display reduced Voronoi cells.

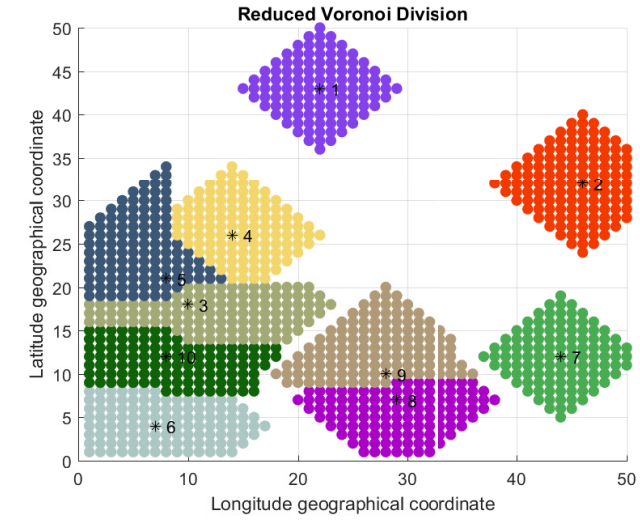
Illustrate reduced Voronoi cells' speedup curves.

"Reduced" means the Voronoi cell shrinks from the outside layer to interior direction layer by layer, which keeps the equivalence computation faster than the bottleneck's computation.

In term of the time complexity :

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

Table 3.2: The 10 Reduced Voronoi Cells and speedup curves



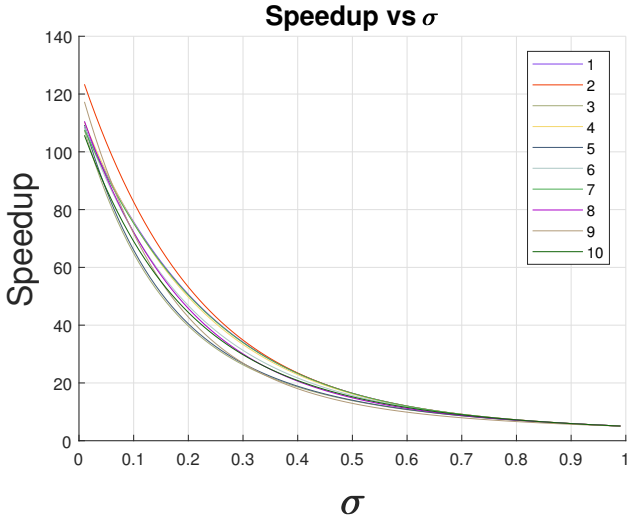


Table 3.2 reduced Voronoi cells and reduced Voronoi speedup curves show this algorithm saves about 30% processors within the same makespan.

- Voronoi speedup curves show that $\sigma < 0.2$, the ratio $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{271}{107} \approx 2.533$.
- Reduced Voronoi speedup curves show that $\sigma < 0.2$, the ratio is $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{122}{107} = 1.14$.

It displays that 10 cells' equivalence computation is more balanced than the initial setting, and the whole cluster finishes processing load within the same time by less processors. After 1000 round random sampling experiments, we obtain the average saved processors ratio in Fig. 3.28.

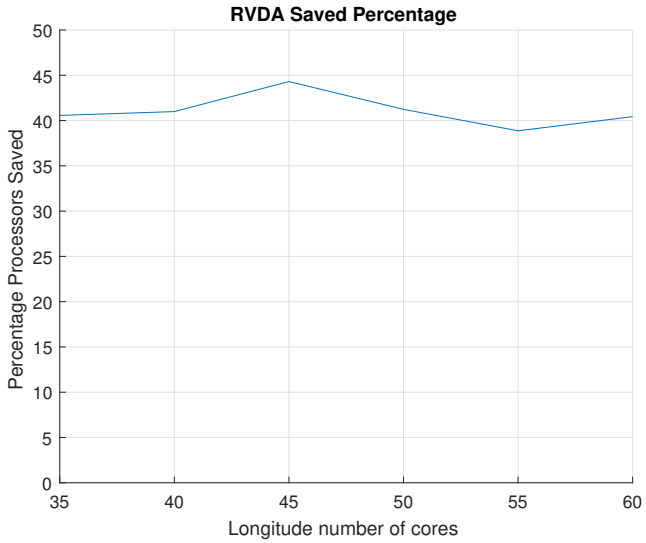


Figure 3.28: Reduced Voronoi diagram algorithm average saved processors' percentage

From the average saved processors ratio Fig. 3.28, it shows the average percentage of saved processor is about 40%.

Accelerated Reduced Manhattan Distance Voronoi Diagram Algorithm

In order to accelerate the bottleneck division computation, we propose a heuristic algorithm to minimize the makespan and give quantitative model analysis utilizing the flow matrix.

The merits of new algorithm is finishing the task within less makespan as MDVDA and also utilizing less processor resources.

$$\max \quad Sp_b \tag{3.115}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.116}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.117}$$

$$T_{f,i} = \hat{T_{j,f}}, \qquad i \in 0 \cdots (S_j - 1) \tag{3.118}$$

$$\hat{\alpha_{j,0}} wT_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.119}$$

$$\vdots, \tag{3.120}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l}_j-2}) zT_{cm} + \alpha_{j,\hat{l}_j-1} wT_{cp} = \hat{T_{j,f}},\ j \in 0 \cdots (k-1) \tag{3.121}$$

$$Sp_i = Sp_b + \epsilon, \qquad i \in 0 \cdots (k-1) \tag{3.122}$$

$$\epsilon \geq 0, \tag{3.123}$$

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

Comparing with RMDVDA, ARMDVDA accelerates the speedup about 20%.

Table 3.3: The 10 accelerated reduced Voronoi cells and speedup curves

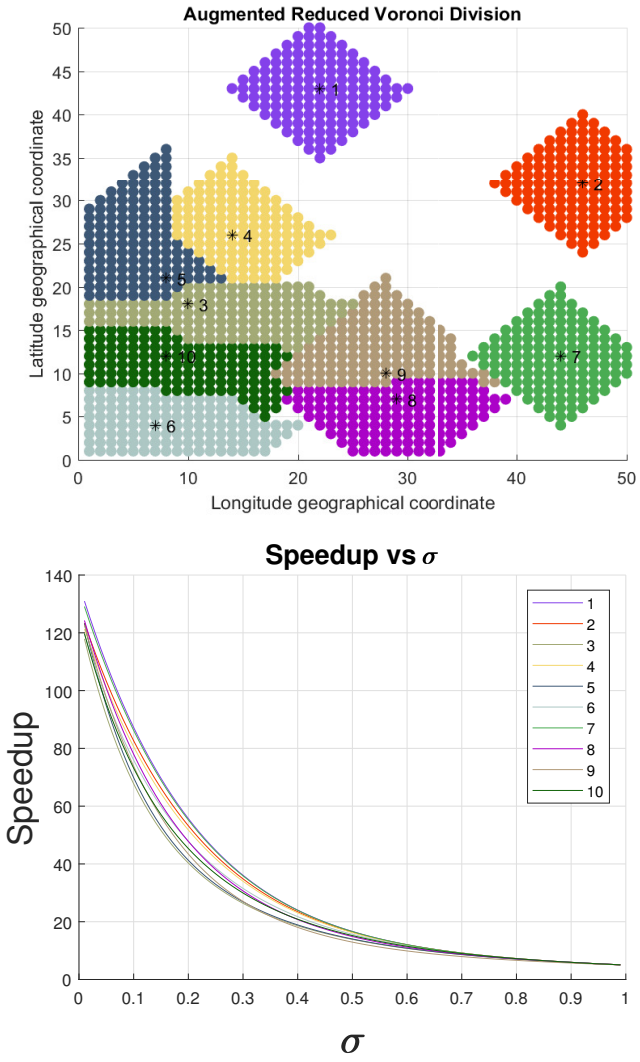


Fig. 3.29 shows the Voronoi cell 3 is the bottleneck division and there is no other economical way to enlarge this constraint cell.

Algorithm 4 Accelerated Reduced Manhattan Distance Voronoi Diagram Algorithm (ARMDVDA)

Input:

- k data injection positions
- $\sigma = \frac{zT_{cm}}{wT_{cp}}$
- processors and links' parameters
- f_T : the requirement of finish time
- r_{save} : the requirement of saved processor ratio
- tmp_{save} : the ratio of saved processor in each iteration
- Sp_i : i Voronoi cell's speedup ratio
- Sp_{min} : $\min(Sp_i) \quad i \in 1 \cdots k$
- i_{min} : the index of Sp_{min}

Output:

- $m * n$ processor data fractions α_i
- accelerated reduced Voronoi division

Calculate k Voronoi cells with Manhattan distance and k Voronoi cells' radius R_i.

Calculate k flow matrix A_i and set $depth_{min} = \min(Sp_i)$'s R_i.

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

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

while $\frac{1}{Sp_{min}} > f_T$ and $tmp_{save} > r_{save}$ **do**

merge processor to base i_{min} Voronoi cell one by one

by increasing order of D_i.

end while

Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves.

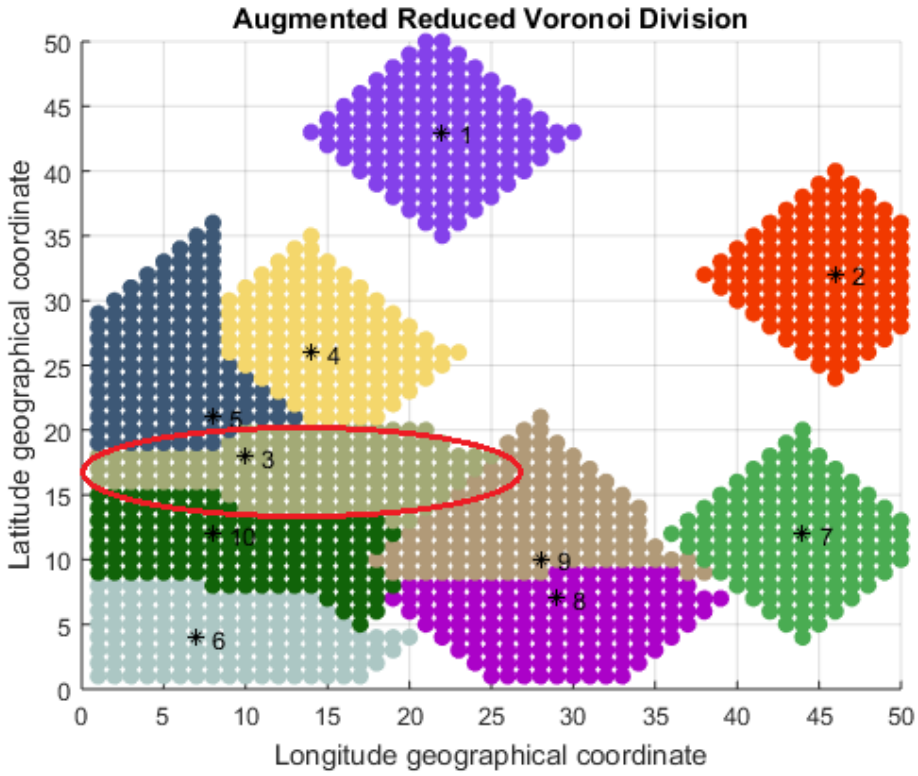


Figure 3.29: 10 reduced Voronoi cells

Situation III

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

Algorithm 5 Reduced Constrained Manhattan Distance Voronoi Diagram Algorithm (RCMDVDA)

Input: $\hat{k}$ data injection positions

Output: $m * n$ processor data fractions

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

Calculate k constrained Manhattan distance Voronoi cells [58].

Calculate k constrained Voronoi cells' radius R_i.

Calculate k constrained Voronoi cells' flow matrix A_i.

Calculate k constrained Voronoi cells' Sp_i.

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

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

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

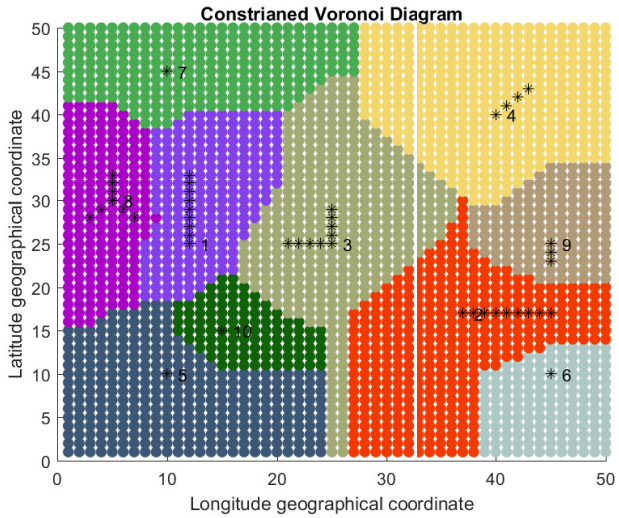
Display reduced Voronoi cells.

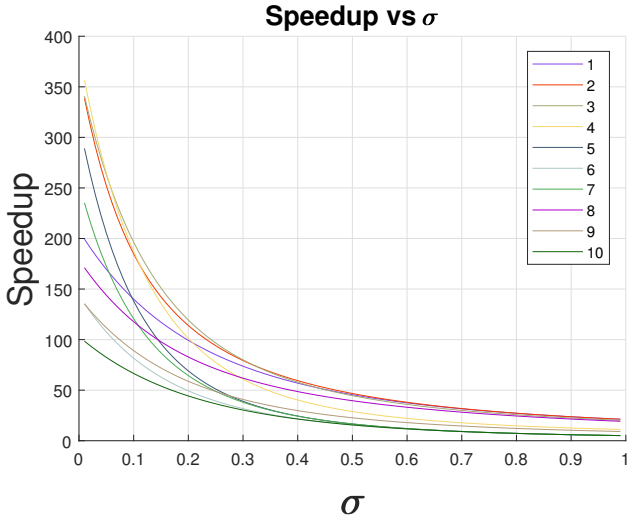
Illustrate reduced Voronoi cells' speedup curves

In term of the time complexity :

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

Table 3.4: The 10 constrained Voronoi cells and speedup curves





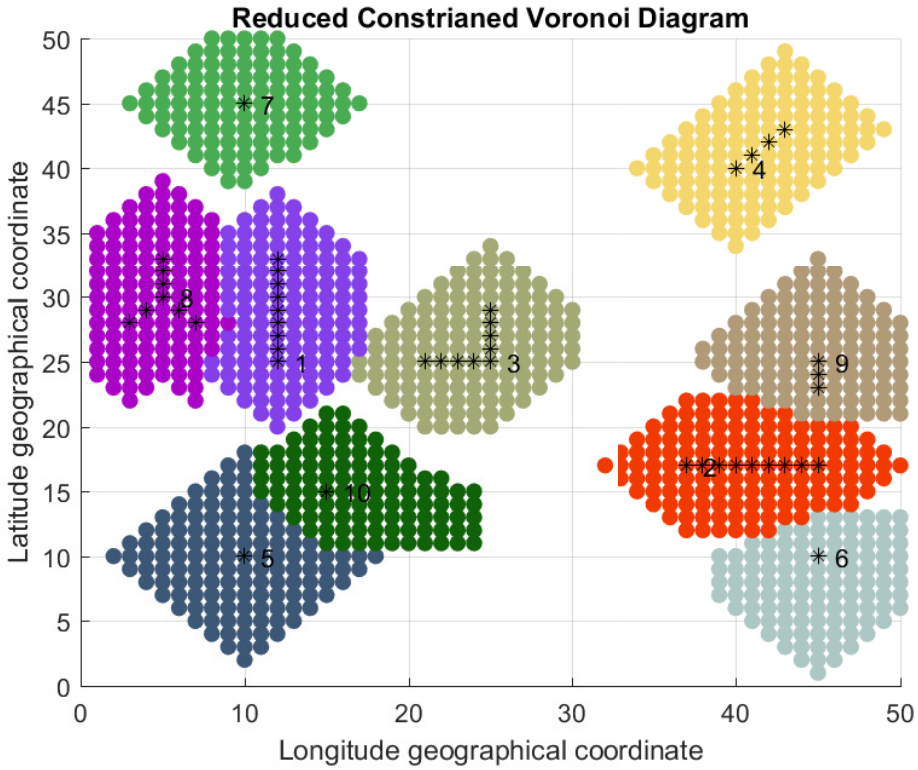


Figure 3.30: Reduced constrained Voronoi diagram for connected subgraphs and individual data injections on mesh network

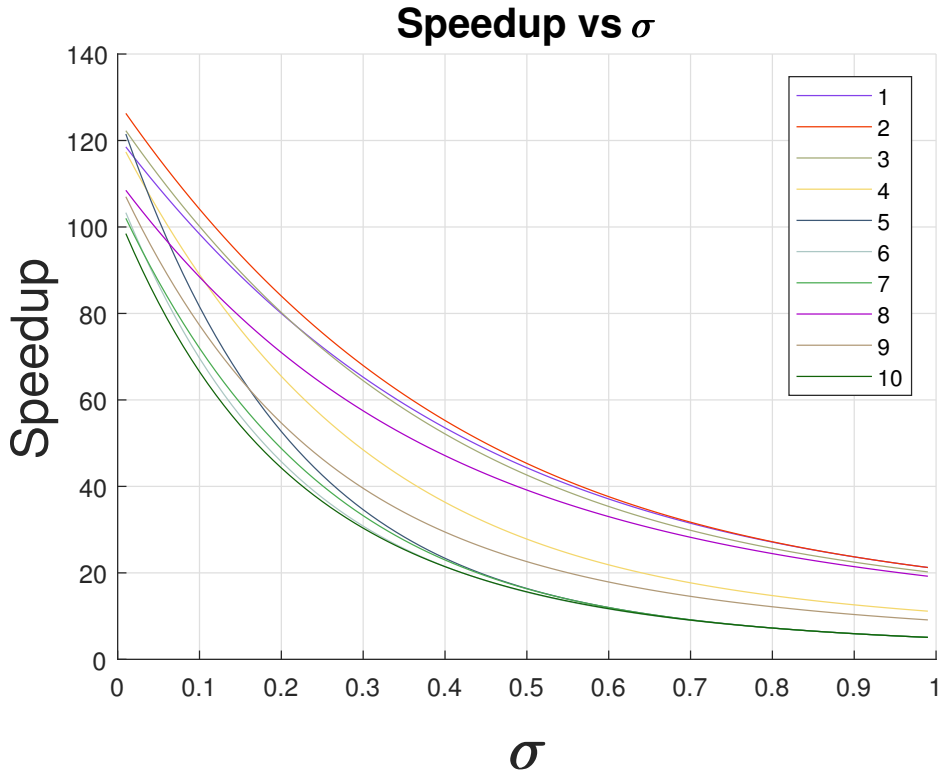


Figure 3.31: Speedup curves of connected subgraphs and individual data injection on mesh network

- Voronoi speedup curves show that $\sigma < 0.2$, the ratio $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{360}{98} \approx 3.67$

- Reduced Voronoi speedup curves show that $\sigma < 0.2$, the ratio is $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{125}{98} = 1.276$.

It displays that 10 cells' equivalence computation is more balanced than the initial setting, and the whole cluster finishes processing load within the same time by less processors.

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

Lower Band of Intuitive and Heuristic Algorithms

Considering a $n*n$ mesh network, there are l data load injections and the makespan is constrained by the makespan of bottleneck. So the best solution is that each injection deploys data to its community and the size of community is $\frac{n*n}{m}$.

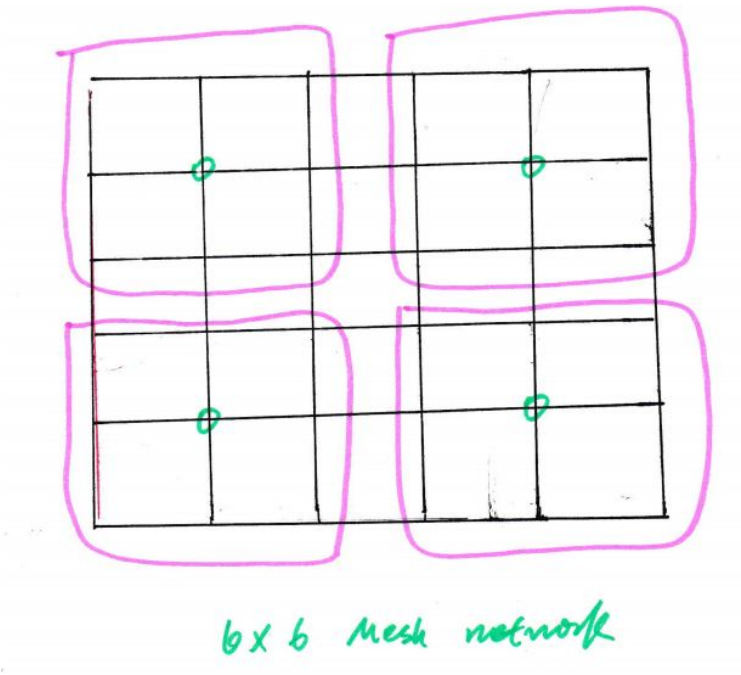


Figure 3.32: Maximal speedup for $6 * 6$ mesh network and 4 data injections

- Sp : Speedup of an equivalence computation cell.
- $optimal_p$: practical optimal speedup of a load data injection, which is NP hard [**?**].
- $optimal_i$: optimal speedup of a load data injection in ideal situation
- β : $\frac{Sp}{optimal_i}$ is the ratio between the speedup of an algorithm (i.e. EPSA or RMDVDA) over the practical ideal situation speedup.

According to the intuitive and heuristic algorithm, we can give a lower band of this algorithm.

$$\beta = \frac{Sp}{optimal_i} \leq \frac{Sp}{optimal_p}$$

$$Sp \geq \beta * optiaml_p$$

For example, we calculate a $50 * 50$ mesh network and there are $l = 10$ load data injections. Each node transmits data to $\frac{50*50}{10} = 250$ cores. The first row of flow matrix in best situation is row_i = [1 4 8 12 16 20 24 28 32 36 40 29];

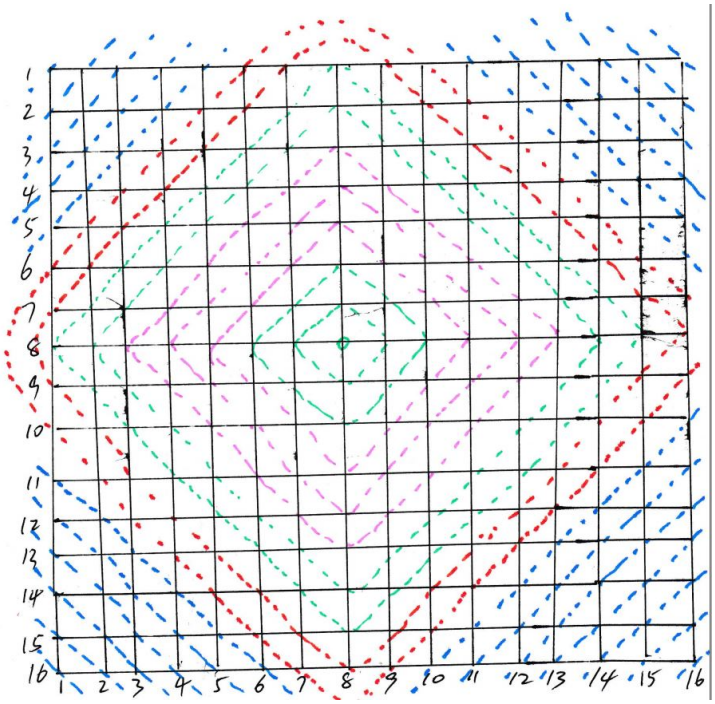


Figure 3.33: Maximal speedup for $16 * 16$ mesh network and 1 data injections

We calculate the speedup of a community and the speedup of ideal situation. In other words, we guarantee obtain a $\beta * optimal_i$ times approximation speedup of our algorithm.

3.1.6 Multi-source Non-uniform Data Injection

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

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

Situation I

We propose a suboptimal algorithm framework to minimize the makespan and give quantitative model analysis utilizing the flow matrix. The constraint comes from the divisible load theory linear equations.

$$\min \quad \hat{T_f} \tag{3.124}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1}\sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.125}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.126}$$

$$T_{f,i} = T_{f,0}, \qquad i \in 1 \cdots (m*n-1) \tag{3.127}$$

$$\hat{\alpha_{j,0}} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.128}$$

$$\vdots, \tag{3.129}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l}_j-2}) + \alpha_{j,\hat{l}_j-1} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.130}$$

The main idea is divided into two stages.

- Re-balance the data fraction between the nodes in connected subgraph first.

- Run EPSA.

Algorithm 6 Equivalence Processor Scheduling Extended Algorithm (EPSEA)

Input: k data injection positions

Output: $m * n$ processor data fractions

$local_s$:

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

$global_s$:

Collapse the data injection processors into one "big" equivalent processor [19].

Calculate $m * n$ processor's D_i

Obtain the flow matrix A_i.

Calculate $m * n$ processors data fraction α_i.

Display reduced Voronoi cells.

Illustrate reduced Voronoi cells' speedup curves.

In $local_s$ stage, our target is from a data fraction distribution to a uniform distribution and the minimum cost is called Earth Mover Distance [59].

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

Situation II

We utilize the Reduced Manhattan Distance Voronoi Diagram Algorithm (RMDVDA) and Accelerated Reduced Manhattan Distance Voronoi Diagram Algorithm (ARMDVDA) to address this problem.

Situation III

We utilize the same idea of Situation I.

The constraint comes from the divisible load theory linear equations.

$$\min \quad \hat{T_f} \tag{3.131}$$

$$\text{s.t.} \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha_{j,i}} = 1 \tag{3.132}$$

$$1 \geq \hat{\alpha_{j,i}} \geq 0, \tag{3.133}$$

$$T_{f,i} = \hat{T_{j,f}}, \qquad i \in 0 \cdots (S_j - 1) \tag{3.134}$$

$$\hat{\alpha_{j,0}} w T_{cp} = \hat{T_{j,f}}, \qquad j \in 0 \cdots (k-1) \tag{3.135}$$

$$\vdots, \tag{3.136}$$

$$(\hat{\alpha_{j,1}} + \cdots \alpha_{j,\hat{l_j}-2}) z T_{cm} + \alpha_{j,\hat{l_j}-1} w T_{cp} = \hat{T_{j,f}}, \; j \in 0 \cdots (k-1) \tag{3.137}$$

$$Sp_i = Sp_b + \epsilon, \qquad i \in 0 \cdots (k-1) \tag{3.138}$$

$$\epsilon \geq 0, \tag{3.139}$$

Algorithm 7 Reduced Constrained Manhattan Distance Voronoi Diagram Extended Algorithm (RCMDVDEA)

Input: $\hat{k}$ data injection positions

Output: $m * n$ processor data fractions

$local_s$:

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

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

$global_s$:

Calculate k constrained Manhattan distance Voronoi cells [58].

Calculate k constrained Voronoi cells' radius R_i.

Calculate k constrained Voronoi cells' flow matrix A_i.

Calculate k constrained Voronoi cells' Sp_i.

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

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

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

Display each reduced Voronoi cells.

Illustrate each reduced Voronoi cells' speedup curves

Chapter 4

Mesh Network Modified Store and Forward Switching

4.1 Modified Store and Forward Switching Scenario

In the modified store and forward scenario, the processors start to process as soon as each processor receives its entire load assignment [60]. However it is assumed that once a load assignment is received, it can serve as zero delay (virtual cut-through) relay for load to successive processors. Pure store and forward could also be modeled, though it is not discussed in this thesis.

4.1.1 Data Injection on The Corner Processor

2*2 Mesh Network

The timing diagram of Fig. 3.1 is shown:

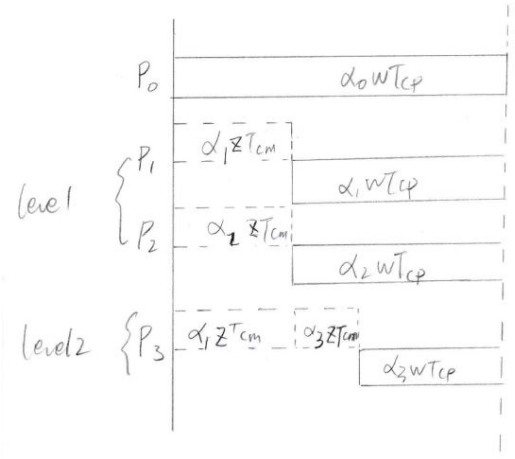


Figure 4.1: The timing diagram for 2*2 mesh network store and forward scenario

The corresponding group of equations are as follows:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (4.1) \\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (4.2) \\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (4.3) \\
(\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (4.4) \\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (4.5) \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 = 1 & (4.6) \\
0 < \sigma < 1 & (4.7) \\
0 < \alpha_0 \leq 1 & (4.8) \\
0 \leq \alpha_1, \alpha_2, \alpha_3 < 1 & (4.9)
\end{cases}
$$

The matrix closed-from is presented as:

$$\begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{4.10}$$

The explicit solution is:

$$\begin{cases} \alpha_0 = (\dfrac{\sigma+1}{\sigma+2})^2 & (4.11) \\ \alpha_1 = \dfrac{\sigma+1}{(\sigma+2)^2} & (4.12) \\ \alpha_3 = \dfrac{1}{(\sigma+2)^2} & (4.13) \end{cases}$$

The simulation result for Fig. 3.1 is provided in Fig. 4.2:

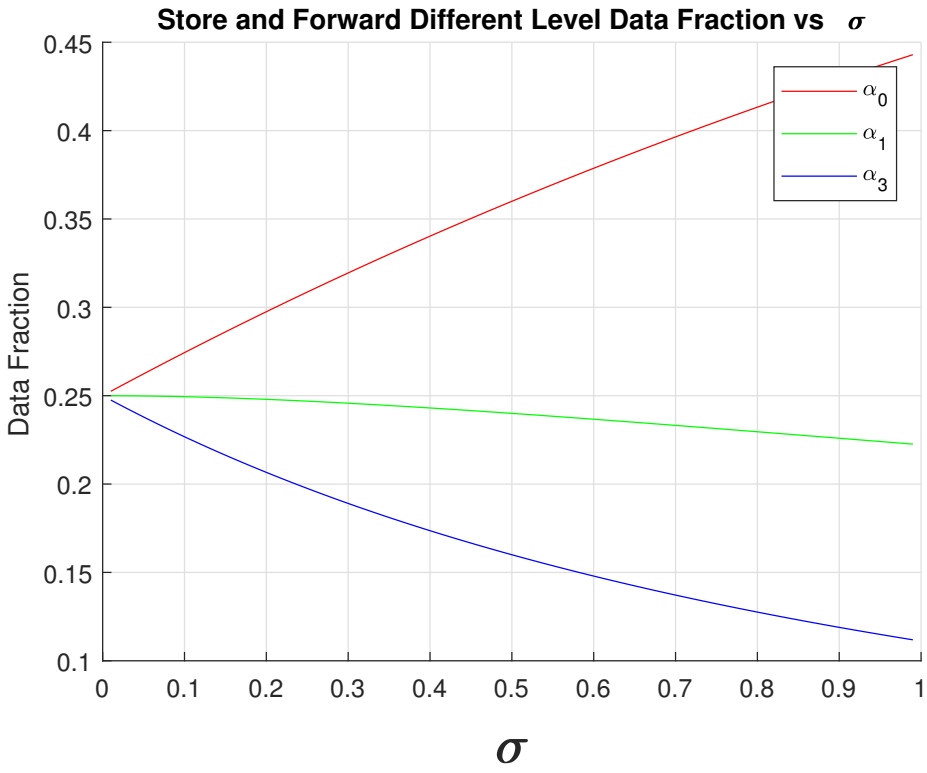


Figure 4.2: Data fraction curves of mesh network $2 * 2$ vs σ

A $2 * 2$ mesh network Fig. 3.1's data fraction curves Fig. 4.2 shows that as the value σ increases, 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 decreases, asymptotically equaling to the processor computation capacity, there is only 11% data is deployed to the $level_2$. In addition, if the $\sigma > 1$, it means that the transmitting time is greater than the processor's processing time. In this scenario, keeping the data locally is more economical than transmitting it. If σ is small (fast communication), $\alpha_0 = \alpha_1 = \alpha_2 = \alpha_3$, that is, all four processors are engaged in processing equally.

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

2*3 Mesh Network

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

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

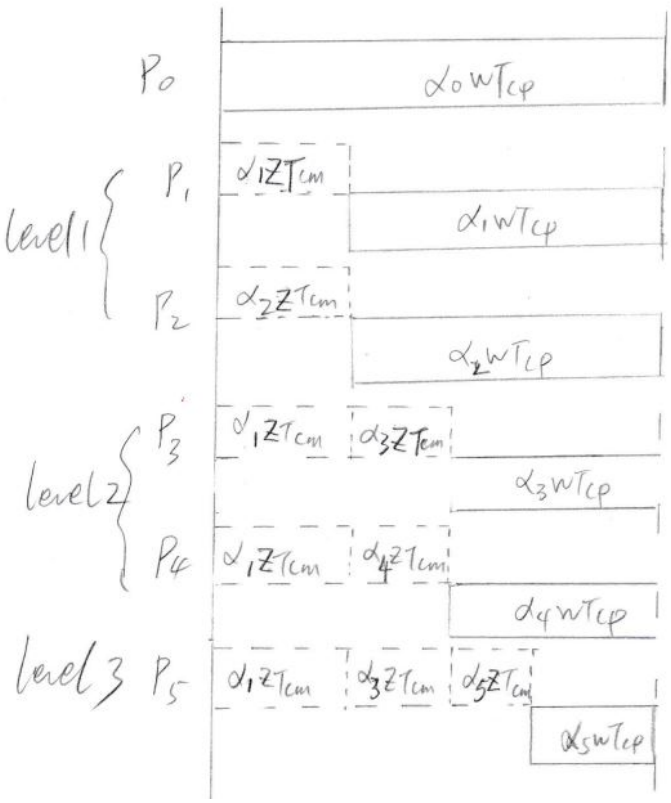


Figure 4.3: The timing diagram for store and forward switching 2*3 mesh network.

In addition, the group of equations are as follows:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (4.14) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (4.15) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (4.16) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (4.17) \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (4.18) \\ (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (4.19) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (4.20) \\ \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = 1 & (4.21) \\ 0 < \alpha_0 \leq 1 & (4.22) \\ 0 \leq \alpha_1 \quad \alpha_2 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 < 1 & (4.23) \end{cases}$$

The flow matrix is :

$$\begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.24)$$

The explicit solution is:

$$\begin{cases} \alpha_0 = (\dfrac{\sigma+1}{\sigma+2})^3 & (4.25) \\ \alpha_1 = \dfrac{(\sigma+1)^2}{(\sigma+2)^3} & (4.26) \\ \alpha_3 = \dfrac{\sigma+1}{(\sigma+2)^3} & (4.27) \\ \alpha_5 = \dfrac{1}{(\sigma+2)^3} & (4.28) \end{cases}$$

The simulation result is:

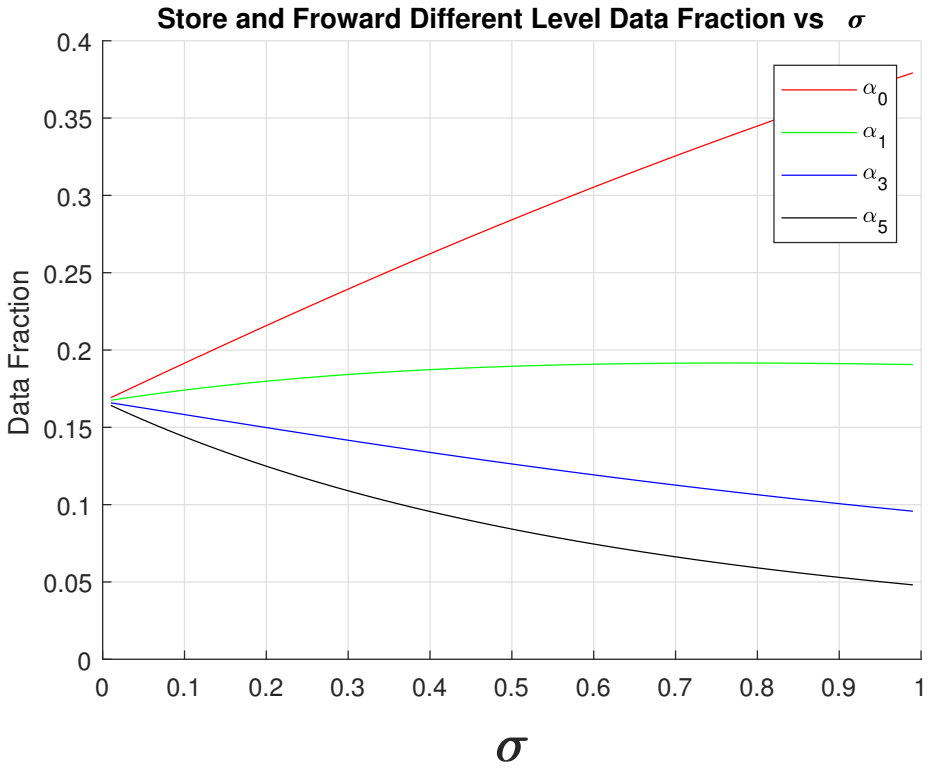


Figure 4.4: Data fraction curves of mesh network $2 * 3$ vs σ

If σ is small (fast communication), each of six processor gets $\frac{1}{6}$ of the load.

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

2*n Mesh Network

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

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (4.29) \\ \alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (4.30) \\ \alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (4.31) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (4.32) \\ (\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (4.33) \\ (\alpha_1 + \alpha_3 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (4.34) \\ \vdots & (4.35) \\ (\alpha_1 + \alpha_3 + \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (4.36) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (4.37) \\ 0 < \sigma < 1 & (4.38) \\ 0 < \alpha_0 \leq 1 & (4.39) \\ 0 \leq \quad \alpha_1 \quad \alpha_3 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (4.40) \end{cases}$$

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

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & \sigma^\star & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.41}$$

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

$$\left\{ \alpha_i = \left| \frac{\det A_i^\star}{\det A} \right| \right. \tag{4.42}$$

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

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

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

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

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

We define

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

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

Further, we define

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

$\hat{C}$ is also full rank. So the flow matrix A is full rank, that is, $\det A \neq 0$ and $\det A^* \neq 0$. The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is

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

m*n Mesh Network

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

The flow matrix closed-form is:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 4 & 3 & 2 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_{10} \\ \alpha_{15} \\ \alpha_{19} \\ \alpha_{22} \\ \alpha_{24} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4.43)$$

The first row in flow matrix describe the number of cores on each D_i. For example, there is 1 core with 0 hop distance (D_0) with load site L. There are 2 cores with 1 hop distance (D_1) with load site L. There are 3 cores with 2 hops distance (D_2) with load site L, and so on.

The number of rows means the number of different type processor data fraction.

4.1.2 Data Injection on The Boundary Processor

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

The timing diagram for Fig. 3.10 is Fig. 4.5.

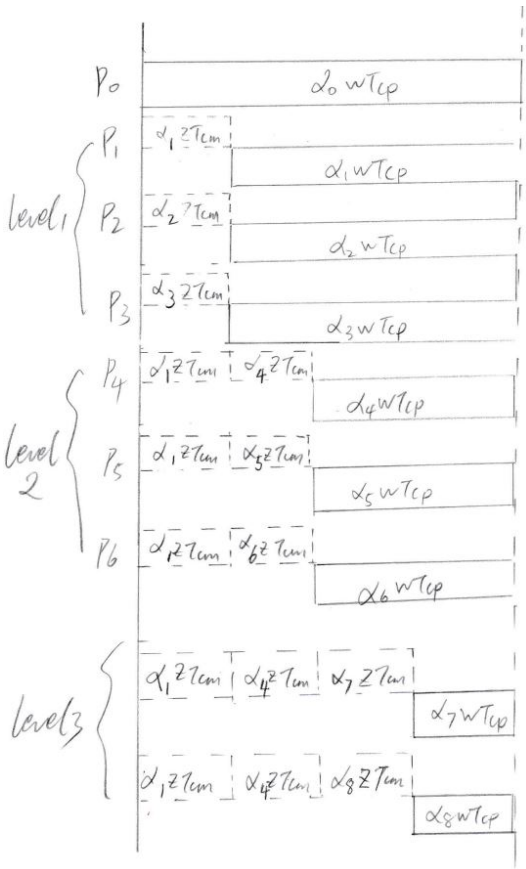


Figure 4.5: The timing diagram for 3*3 boundary data injection on P_0

The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (4.44)\\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (4.45)\\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (4.46)\\
\alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (4.47)\\
(\alpha_1 + \alpha_4) z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (4.48)\\
(\alpha_2 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (4.49)\\
(\alpha_3 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (4.50)\\
(\alpha_1 + \alpha_4 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (4.51)\\
(\alpha_1 + \alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (4.52)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (4.53)\\
0 < \sigma < 1 & (4.54)\\
0 < \alpha_0 \leq 1 & (4.55)\\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 & (4.56)
\end{cases}
$$

The flow matrix is :

$$
\begin{bmatrix}
1 & 3 & 3 & 2\\
1 & -(\sigma+1) & 0 & 0\\
1 & -\sigma & -(\sigma+1) & 0\\
1 & -\sigma & -\sigma & -(\sigma+1)
\end{bmatrix}
\times
\begin{bmatrix}
\alpha_0\\ \alpha_1\\ \alpha_4\\ \alpha_7
\end{bmatrix}
=
\begin{bmatrix}
1\\ 0\\ 0\\ 0
\end{bmatrix}
\qquad (4.57)
$$

The simulation result is shown in :

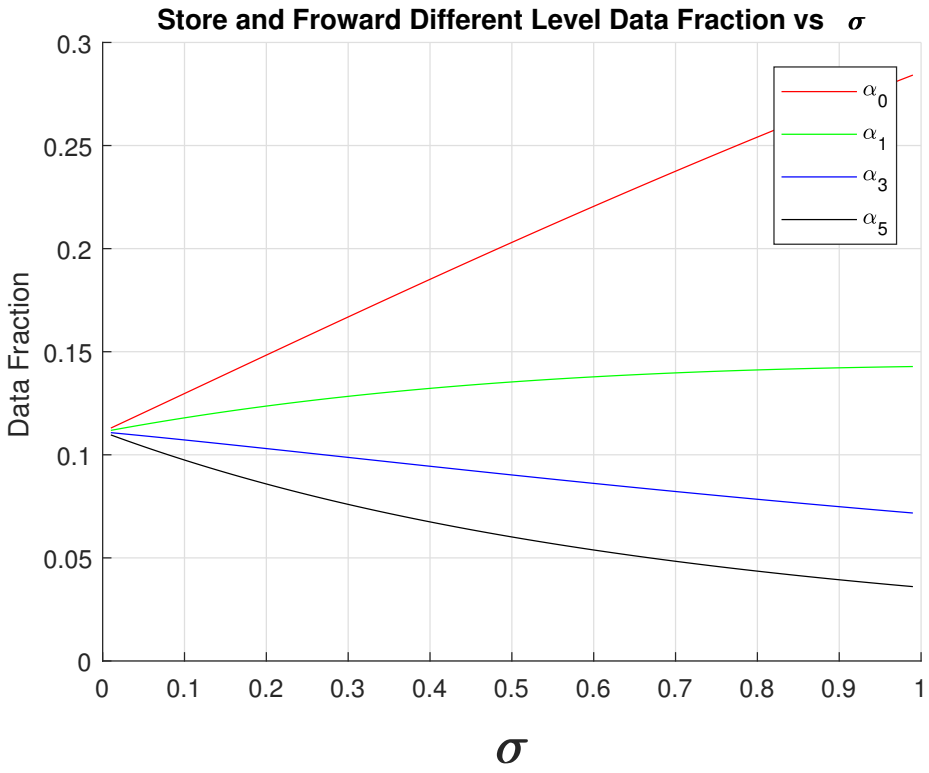


Figure 4.6: The fraction curve for 3*3 boundary data injection on P_0

A $3 * 3$ mesh network appears in Fig. 3.10 and its data fraction curves Fig. 4.6 shows that as the value σ increases, the fraction assigned to P_0 increases and the fractions distributed to $level_2$ and $level_3$ decreases. In other words, if the communication capability decreases, there are more data processed locally, which is reasonable.

If the ability of the link degrades, asymptotically equaling to the processor computation capacity, there is solely 4% data is deployed to the $level_3$. In addition, if the $\sigma > 1$, it means that the transmitting time is greater than the processor's processing time. In this scenario, keeping the data locally is more economical than transmitting it.

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

4.1.3 Data Injection on The Inner Grid Processor

The mesh network considered is $3 * 3$.

The equations are:

$$
\begin{cases}
\alpha_0 \omega T_{cp} = T_{f,m} & (4.58)\\
\alpha_1 z T_{cm} + \alpha_1 \omega T_{cp} = T_{f,m} & (4.59)\\
\alpha_2 z T_{cm} + \alpha_2 \omega T_{cp} = T_{f,m} & (4.60)\\
\alpha_3 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (4.61)\\
\alpha_4 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (4.62)\\
(\alpha_1 + \alpha_5) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (4.63)\\
(\alpha_2 + \alpha_6) z T_{cm} + \alpha_6 \omega T_{cp} = T_{f,m} & (4.64)\\
(\alpha_3 + \alpha_7) z T_{cm} + \alpha_7 \omega T_{cp} = T_{f,m} & (4.65)\\
(\alpha_4 + \alpha_8) z T_{cm} + \alpha_8 \omega T_{cp} = T_{f,m} & (4.66)\\
\sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (4.67)\\
0 < \sigma < 1 & (4.68)\\
0 < \alpha_0 \leq 1 & (4.69)\\
0 \leq \quad \alpha_1 \quad \alpha_3 \quad \alpha_4 \quad \alpha_5 \quad \alpha_6 \quad \alpha_7 \quad \alpha_8 < 1 & (4.70)
\end{cases}
$$

The flow matrix closed-form is:

$$
\begin{bmatrix} 1 & 4 & 4 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad (4.71)
$$

The simulation result shows:

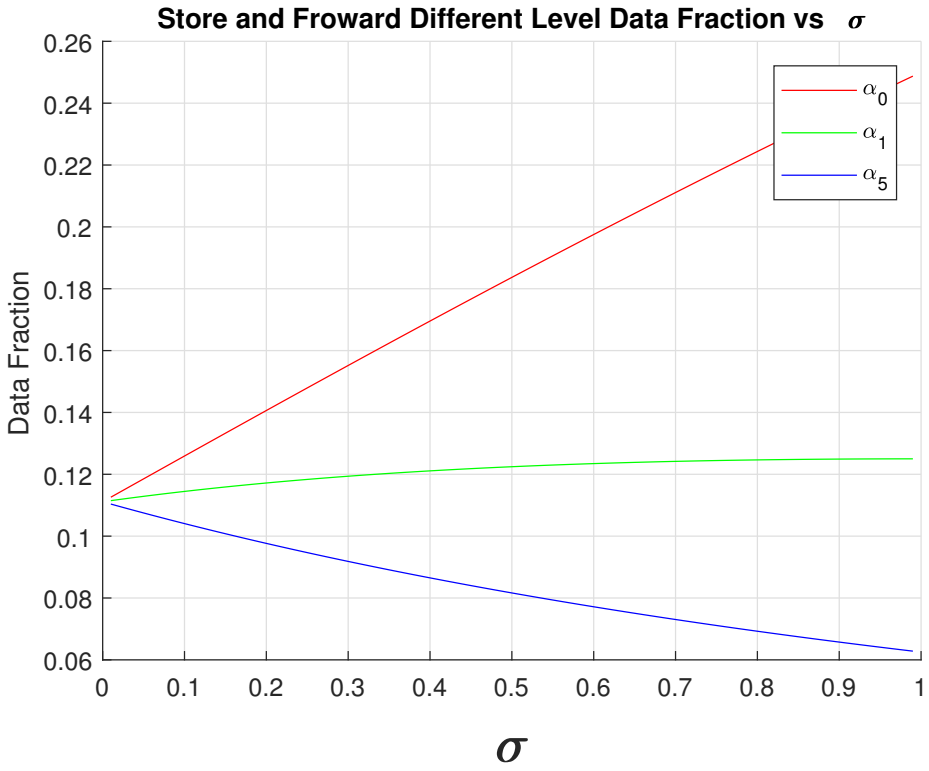


Figure 4.7: The timing diagram for 3*3 inner grid injection P_0

A $3 * 3$ mesh network appears in Fig. 3.13 and its data fraction curves Fig. 4.7 illustrates that as the value σ increases, the fraction assigned to P_0 increases and the fractions distributed to $level_2$ 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 only 6% data is deployed to the $level_2$.

If σ is very small (fast communication), each of nine processors process $\frac{1}{9}$ of the load

4.1.4 Sensitivity Analysis Modified Store and Forward Scenario

Data Injection on The Corner Processor

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

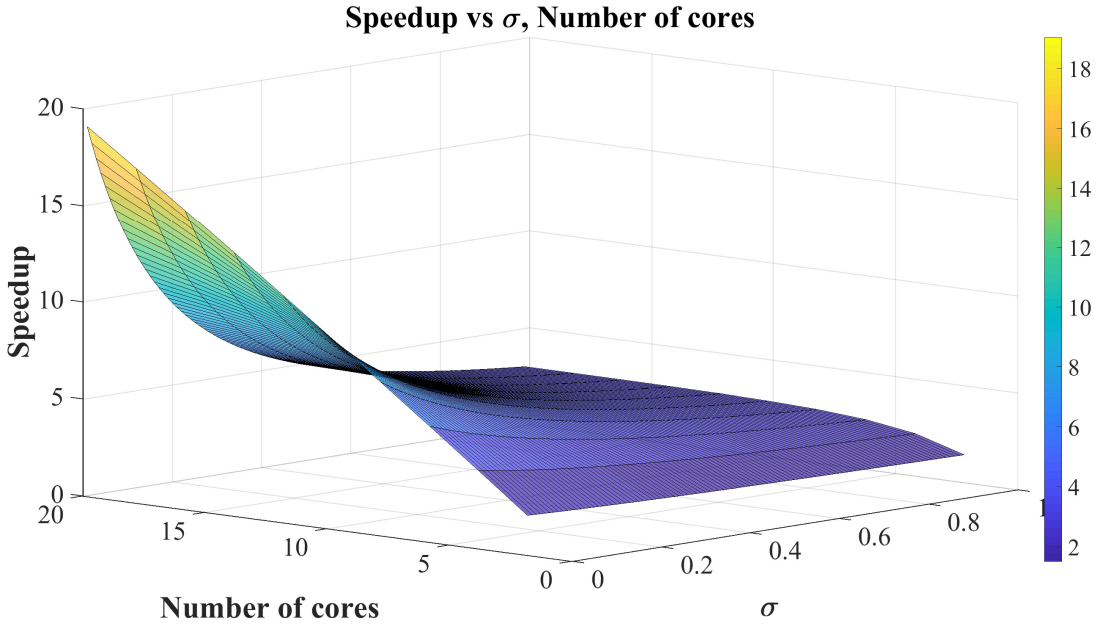


Figure 4.8: Sensitivity analysis result of 2*n mesh network result ($n = 1 \cdots 10$)

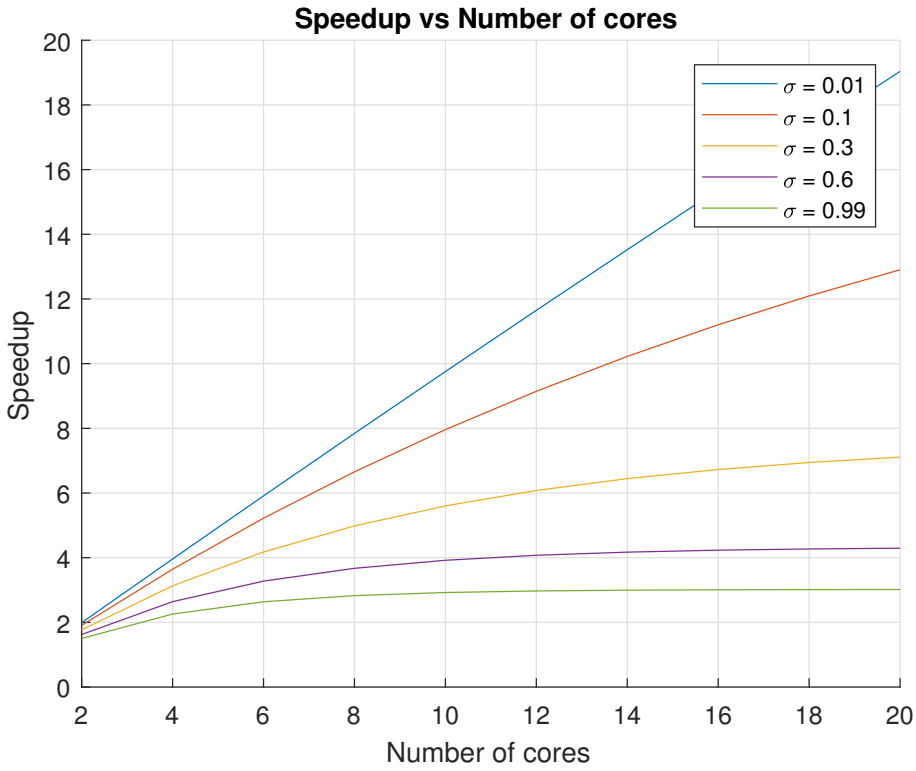


Figure 4.9: Speedup curve of 2*n mesh network result ($n = 1 \cdots 10$)

The figure illustrates that if $\sigma < 0.1$, the number of cores increases, the speedup efficiency is linearly increasing. Alternatively speaking, if $\sigma < 0.1$, the core's number is the most important factor for efficiency. If the $\sigma > 0.3$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation.

If σ is close to one (slow communication) and the number of cores is large, then speedup is three times since P_0 and P_0 two neighbors do almost all the processing.

Data Injection on The Boundary Processor

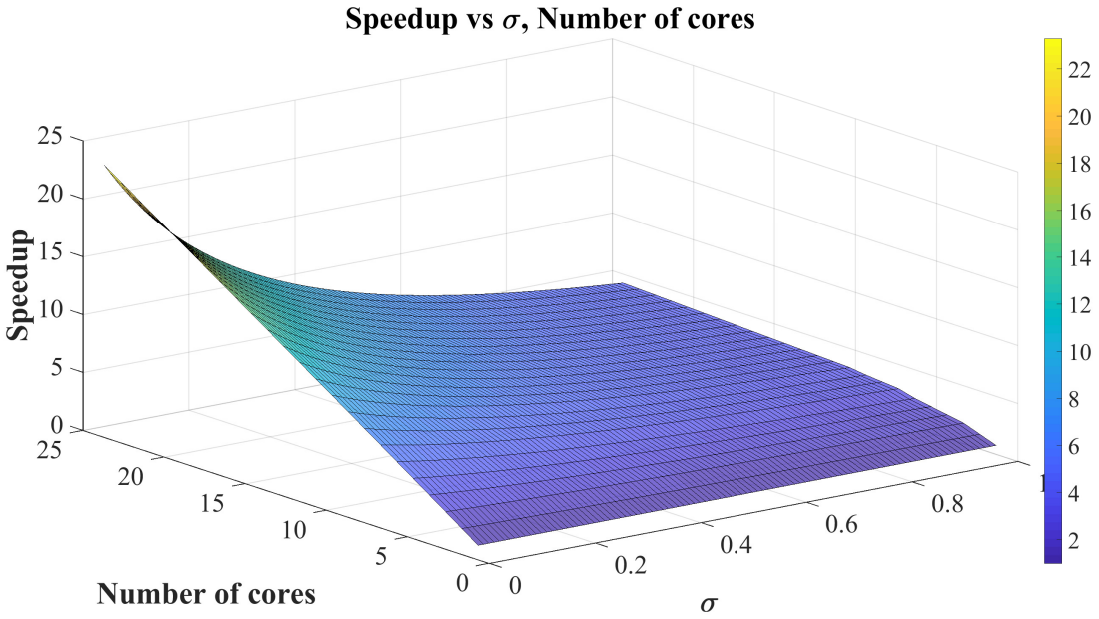


Figure 4.10: Sensitivity analysis result of 3*n mesh network result ($n = 1 \cdots 8$)

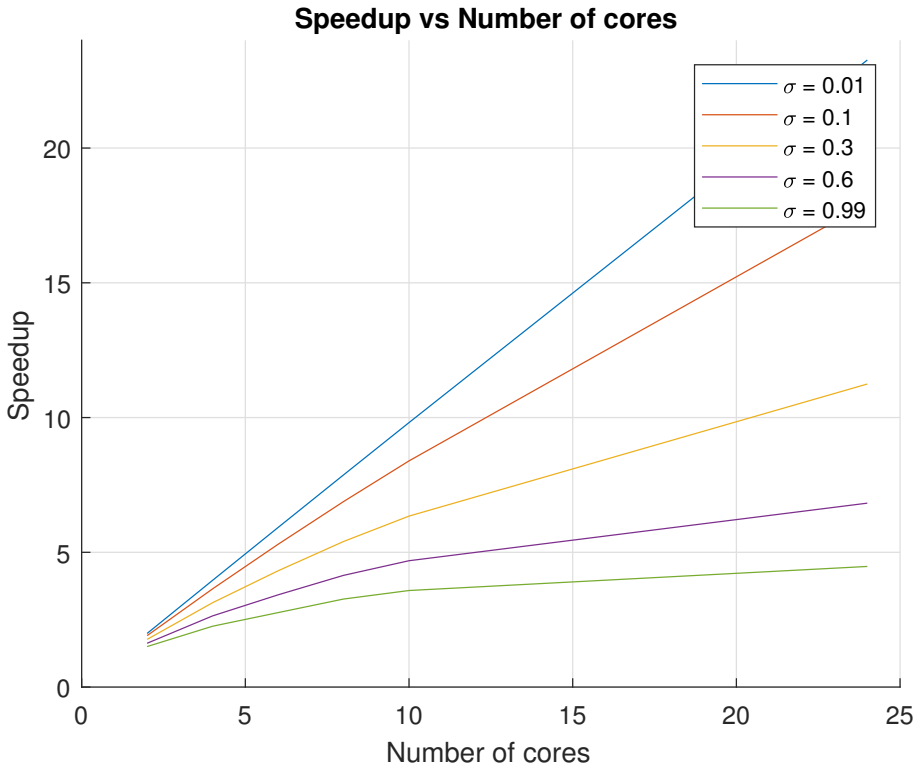


Figure 4.11: Speedup curve of 3*n mesh network result ($n = 1 \cdots 8$)

Sensitivity analysis result Fig. 4.10 and speedup curve Fig. 4.11 show the speedup efficiency for the data injection on the boundary scenario.

The figure illustrates that if $\sigma < 0.1$ and the number of cores increases, the speedup efficiency is linearly increasing. The maximum speedup is 24, the number of the cores in a $3*8$ mesh network. Alternatively speaking, if $\sigma < 0.1$, the core's number is the most important factor for efficiency. If the $\sigma > 0.3$, the efficiency drops dramatically. That is, the σ value plays more critical role in the speedup simulation.

Data Injection on The Inner Grid Processor

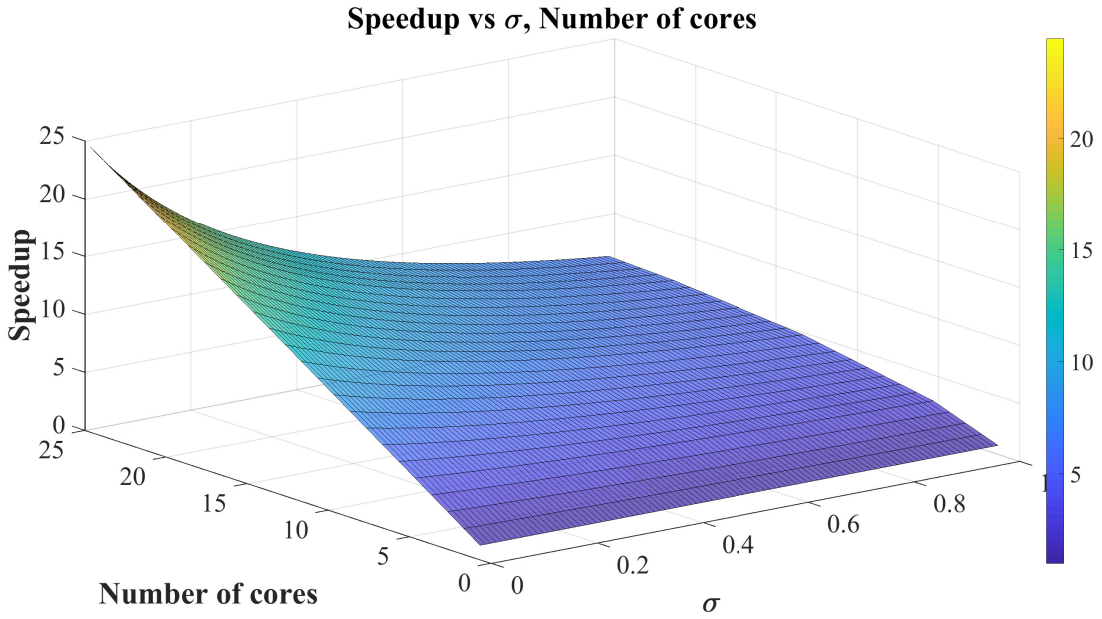


Figure 4.12: Sensitivity analysis result of data injection position on inner grid processor of $5 * 5$ mesh network

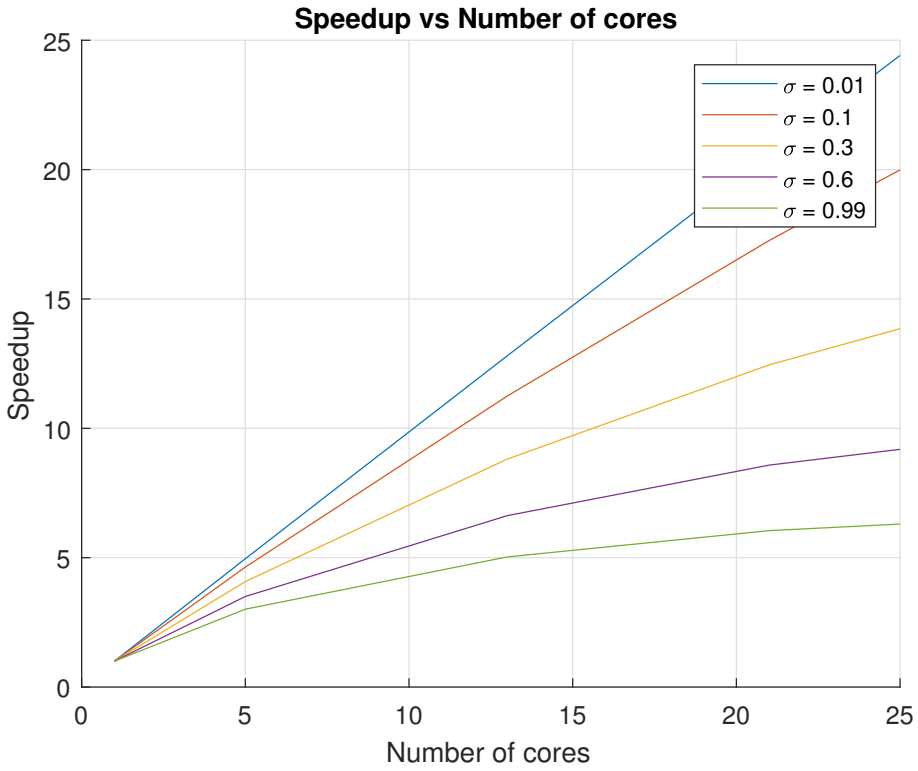


Figure 4.13: Speedup curve of $5 * n$ mesh network result ($n = 1 \cdots 5$)

Sensitivity analysis Fig. 4.12 and speedup curves Fig. 4.13 display the simulation result for the data injection inner position P_{12}. If $\sigma < 0.1$, the speedup linear increases and the maximum speedup is 24.

4.1.5 Multi-source Uniform Data Fraction

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

Situation I

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

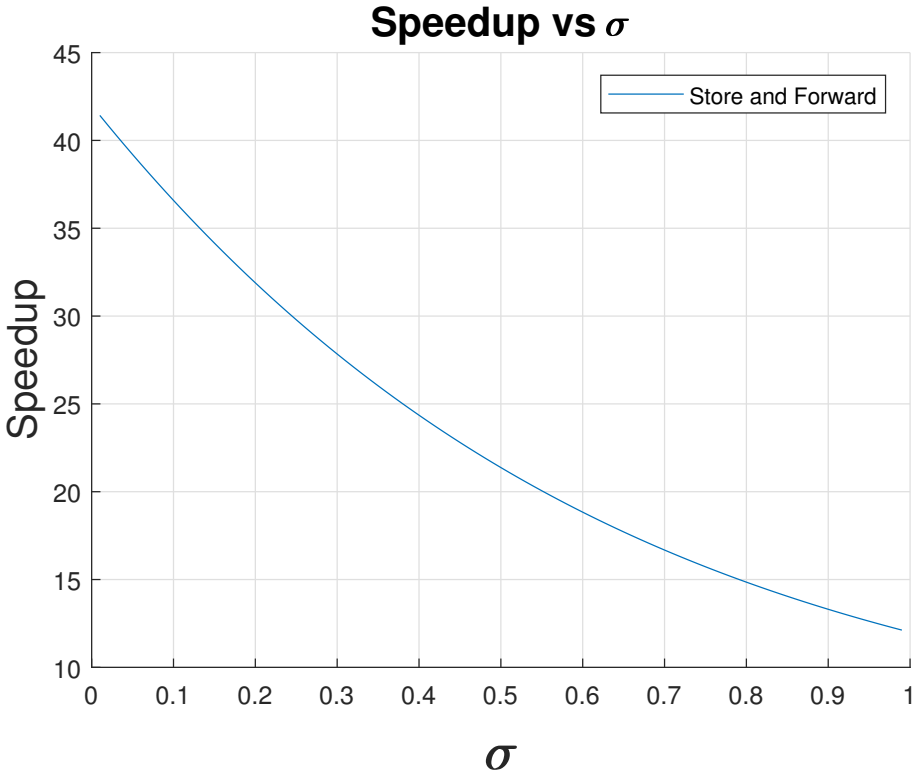


Figure 4.14: Connected subgraph Fig. 3.22 speedup vs σ

For fast communication ($\sigma \approx 0$), the maximum speedup is 42.

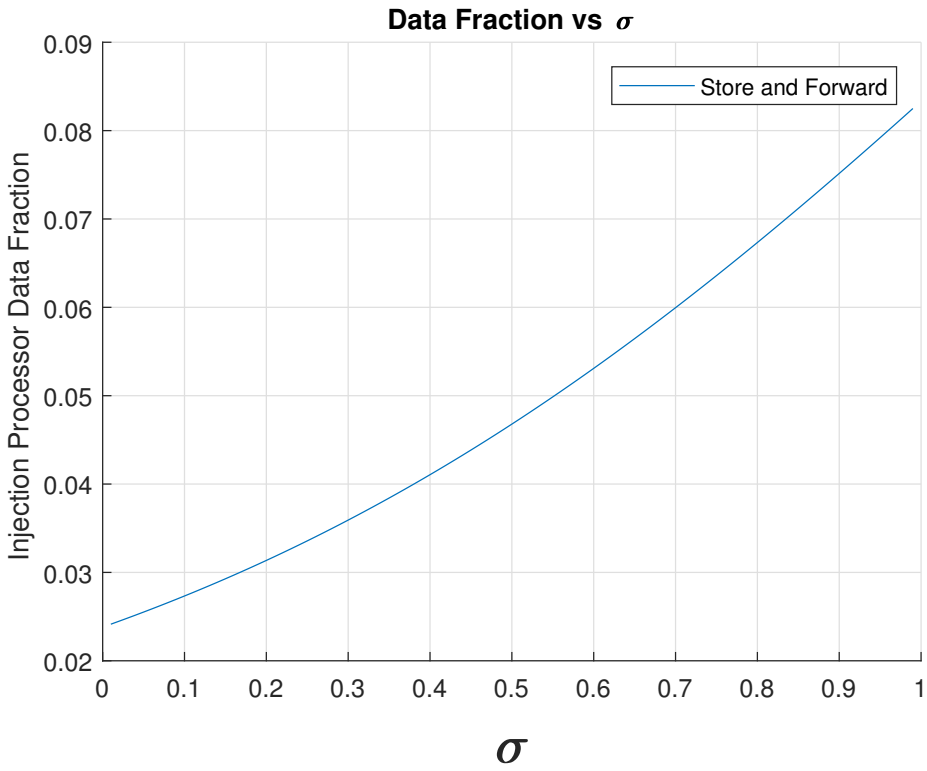


Figure 4.15: Connected subgraph Fig. 3.22's data injection processor workload fraction vs σ

For fast communication ($\sigma \approx 0$), each of 42 processors in Fig. 3.22 gets $\frac{1}{42}$ of the load since they are all equals engaged in processing.

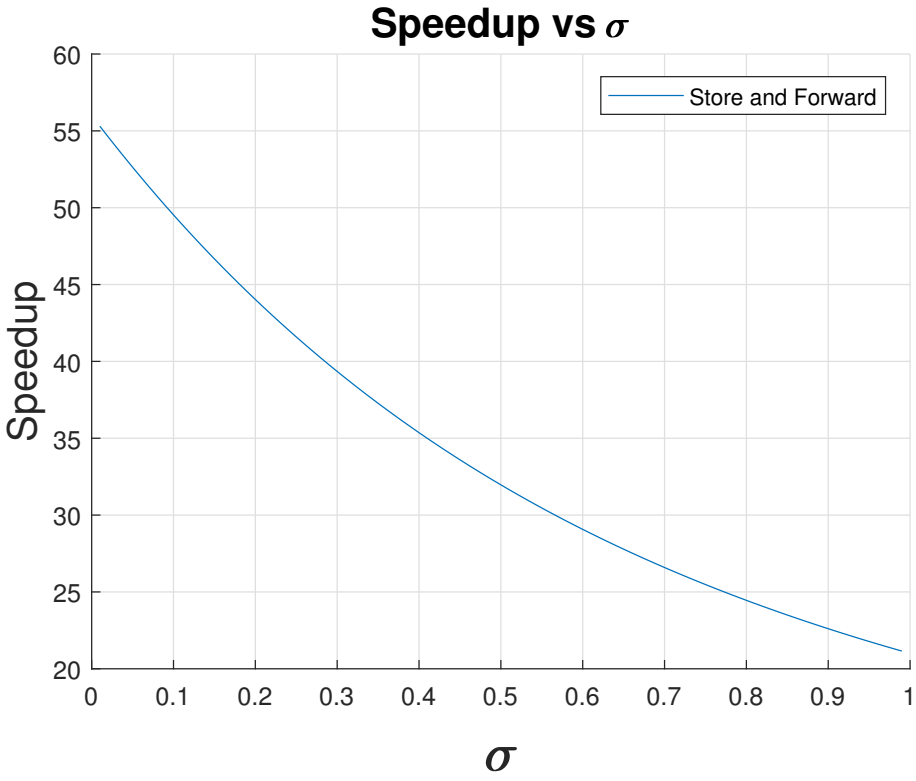


Figure 4.16: Connected subgraph Fig. 3.23 speedup vs σ

For fast communication $\sigma \approx 0$, the maximum speedup is 56 (there are $7 * 8 = 56$ processors in Fig. 3.23).

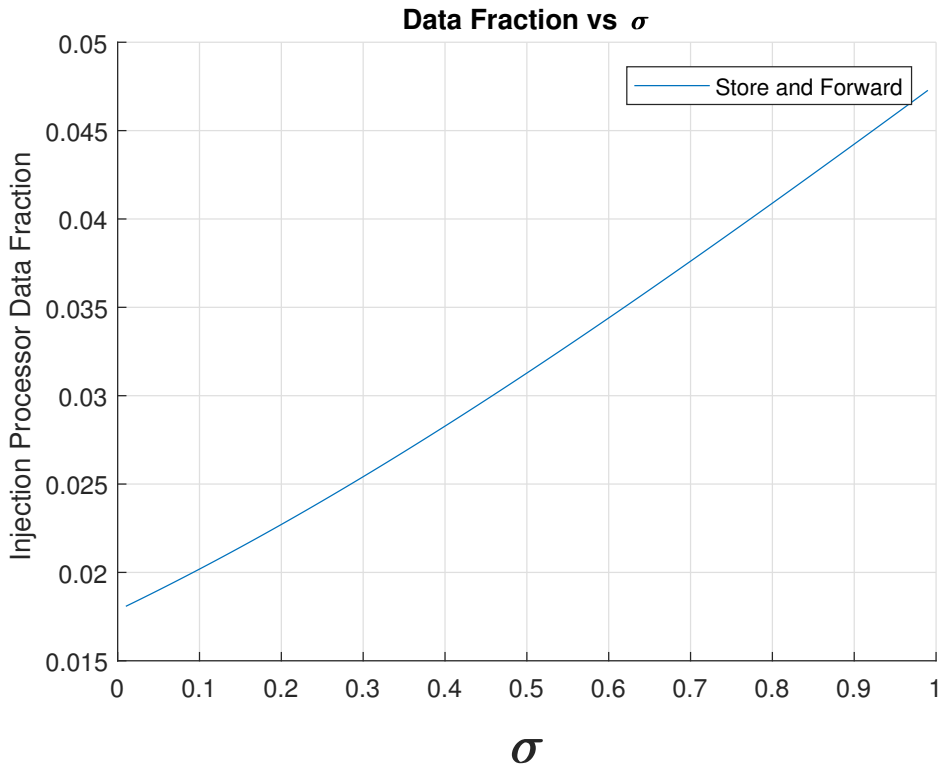


Figure 4.17: Connected subgraph Fig. 3.23's data injection processor workload fraction vs σ

For fast communication ($\sigma \approx 0$), each of 56 processors in Fig. 3.23 gets $\frac{1}{56}$ of the load.

Situation II

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

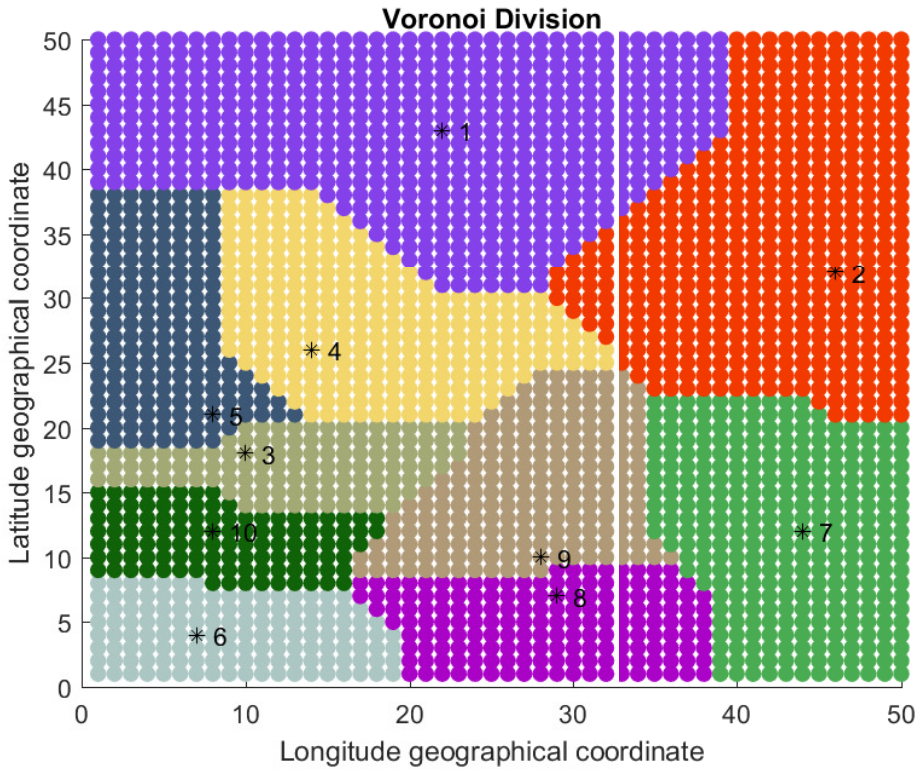


Figure 4.18: 10 Manhattan distance Voronoi Cells

Fig. 4.18 shows an intuitive algorithm to minimizing the makespan. There are 10 bases and the largest cell contains about 495 cores. There are about 107 cores in the smallest cell. So the ratio is $\frac{495}{107} \approx 4.62$.

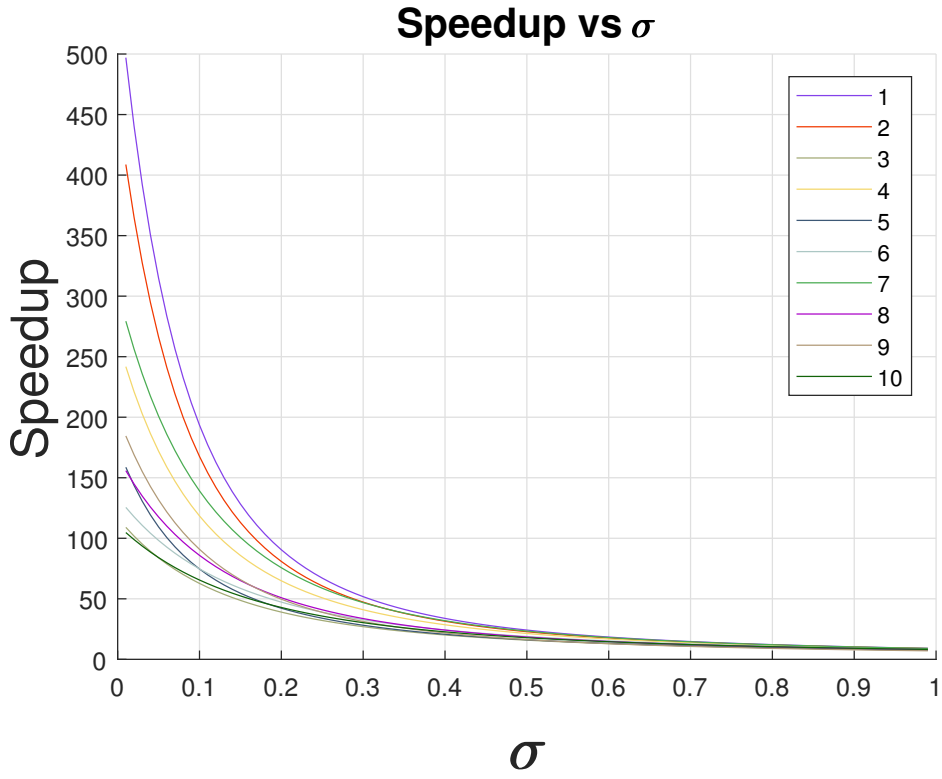


Figure 4.19: 10 Manhattan distance Voronoi cells modified store and forward switching speedup curves

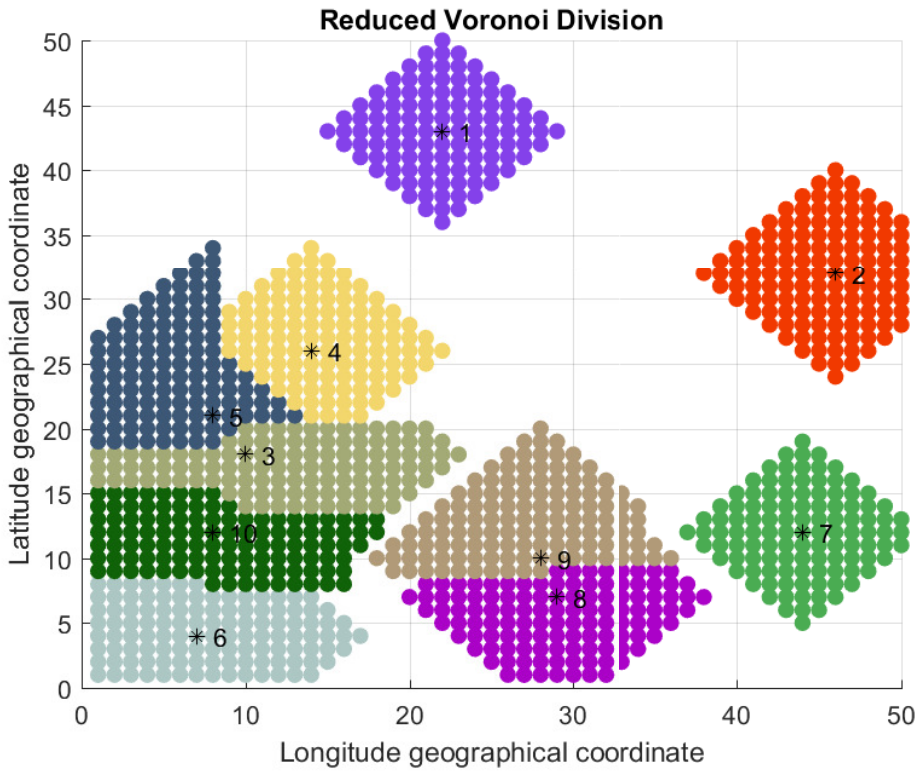


Figure 4.20: 10 reduced Manhattan distance modified store and forward switching Voronoi cells

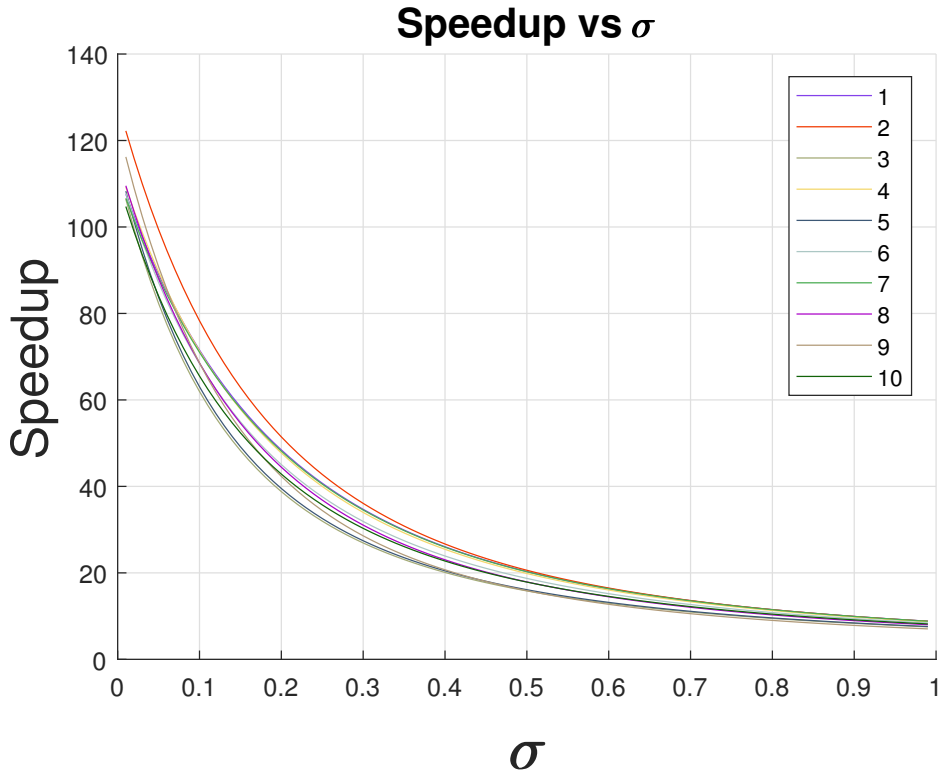


Figure 4.21: 10 reduced Voronoi cells' store and forward switching speedup curves

We save about 53.4% processors and finish the task within the same time.

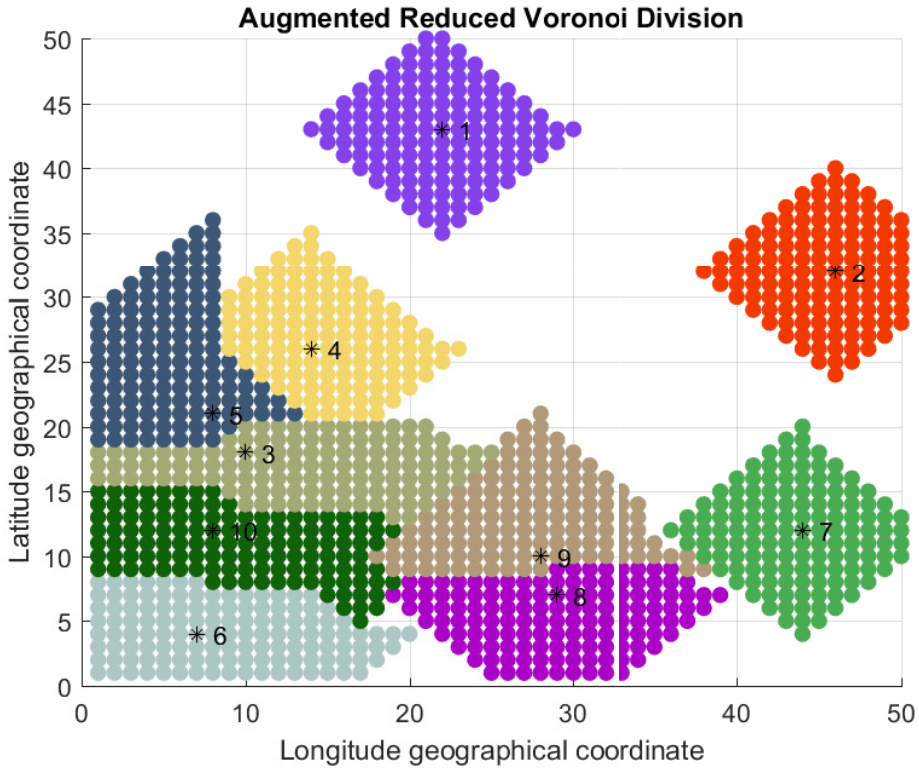


Figure 4.22: 10 reduced Manhattan distance modified store and forward switching Voronoi cells

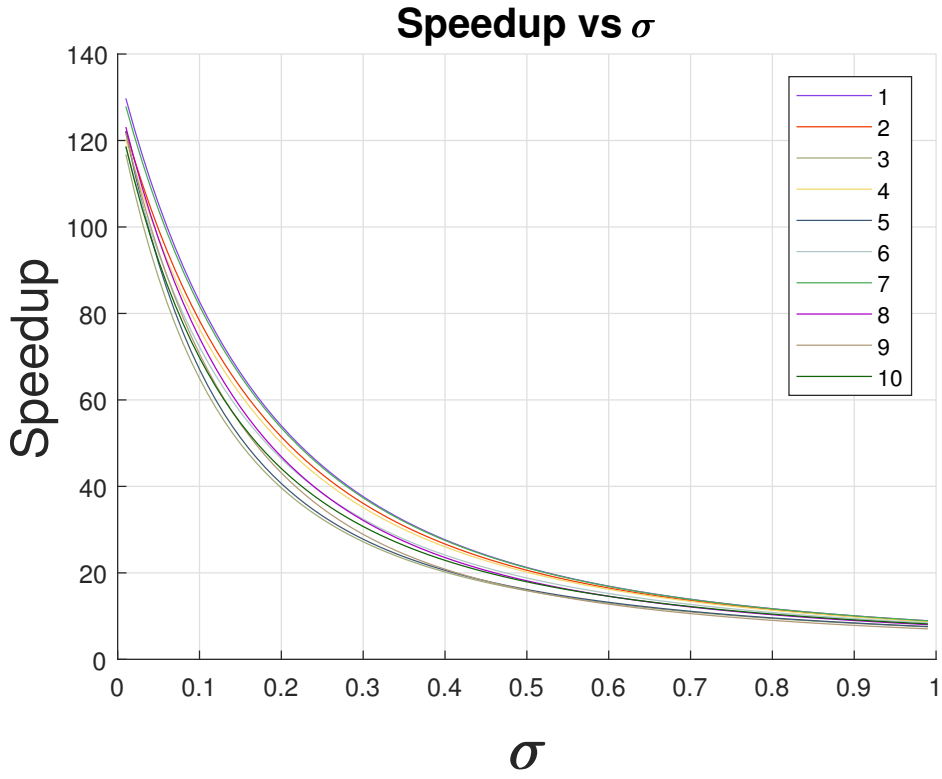


Figure 4.23: 10 reduced Voronoi cells' store and forward switching speedup curves

Fig. 4.22 and Fig. 4.23 show ARMDVDA simulation result and the new heuristic algorithm accelerates about 10% speedup.

Situation III

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

4.1.6 Multi-source Non-uniform Data Injection

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

Chapter 5

Toroidal Rectangle Network

5.1 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 [61], there are three different torus mesh networks. 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. 5.1 and Fig. 5.2.

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

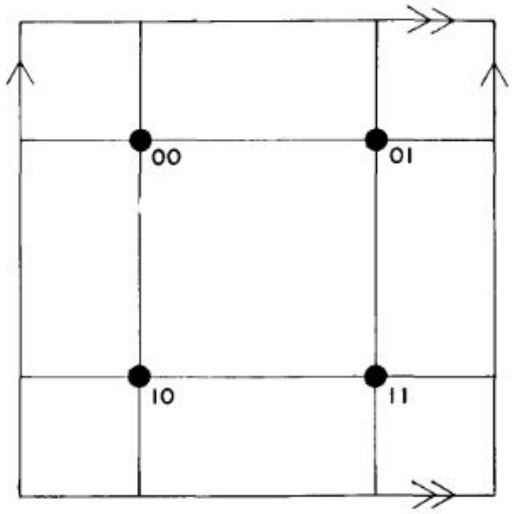


Figure 5.1: The rectangular toroidal network

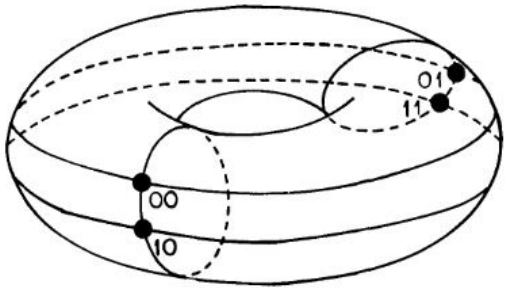


Figure 5.2: The rectangular toroidal network

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

$$\begin{cases} D_k = D_{k,x} + D_{k,y} & (5.1) \\ D_{k,x} = \min\{\|D_{k,x} - L_x\|, m - \|D_{k,x} - L_x\|\} & (5.2) \\ D_{k,y} = \min\{\|D_{k,y} - L_y\|, n - \|D_{k,y} - L_y\|\} & (5.3) \end{cases}$$

5.2 Virtual Cut-through Switching Scenario

5.2.1 Data Injection On The Grid Processor

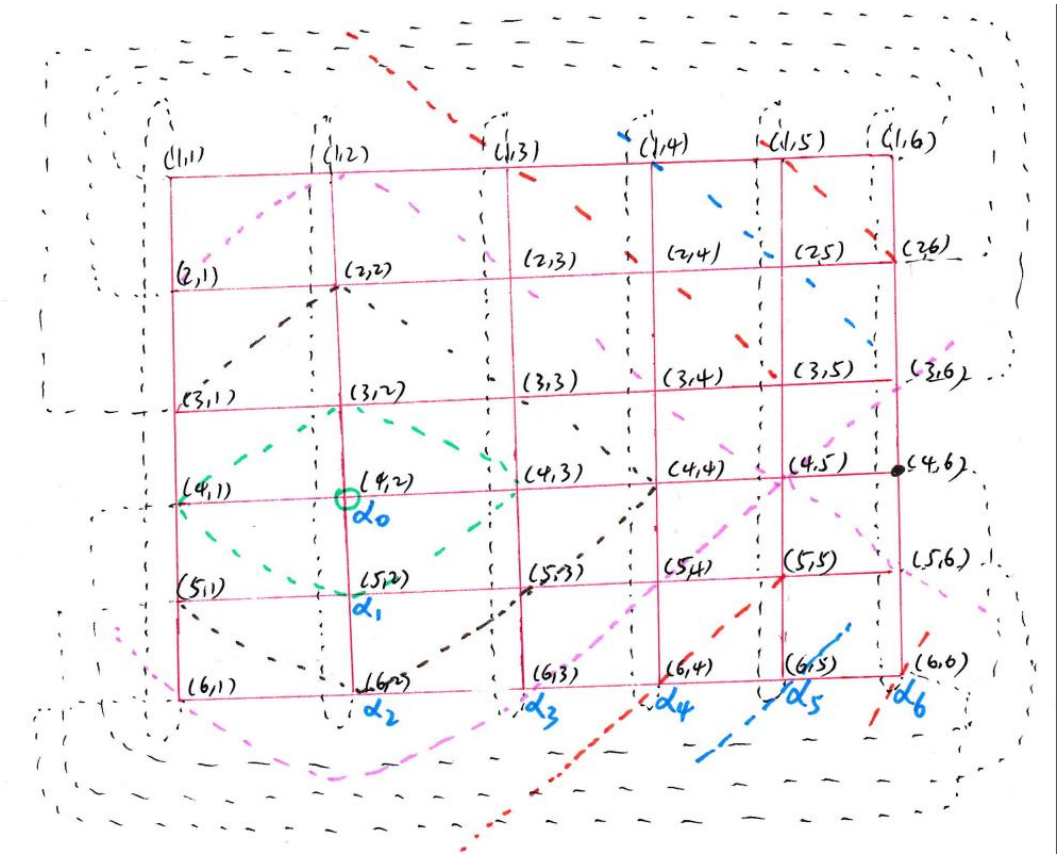


Figure 5.3: The m*n toroidal rectangle network and the data injection is $P_{4,2}$

In $m * n(m = 6, n = 6)$ Fig. 5.3 toroidal rectangle network, L happens on grid position $(4, 2)$. We calculate the $D_{k,i}$ table Table 5.1 by breadth first search(***BFS***) algorithm.

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

Table 5.1: D_i vs Number

The flow matrix closed-form is :

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.4}$$

The simulation result is :

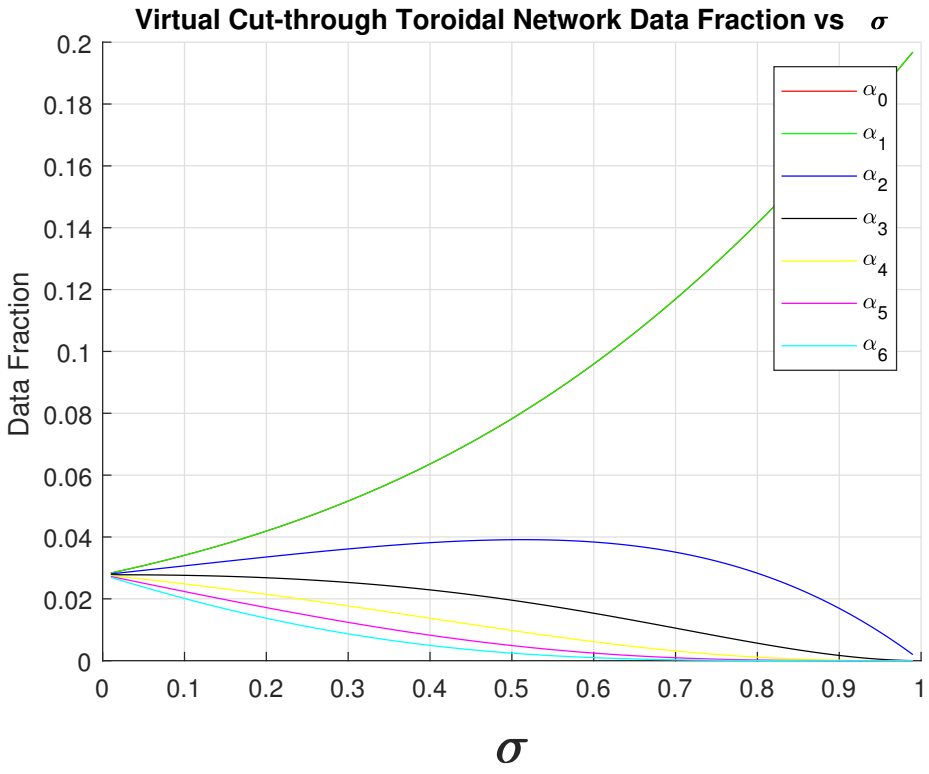


Figure 5.4: Data fraction curve of toroidal rectangle network Fig. 5.3

Here for fast communication ($\sigma \approx 0$), each of 36 processors gets $\frac{1}{36}$ of the load as all 36 processors receive a uniform amount of load. If communication is slow, each of P_0 and four P_0 neighbors receive $\frac{1}{5}$ of the load (i.e. 0.2 of the load). The load sharing by these five processors is economical.

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

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

Table 5.2: D_i vs Number of processor on each D_i

The speedup is :

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$$

5.2.2 Sensitivity Analysis of Toroidal Rectangle Network

Considering a $5 * 5$ toroidal rectangle network, the $level_i$ table shows Table 5.2.

So the simulation result illustrates in Fig. 5.5

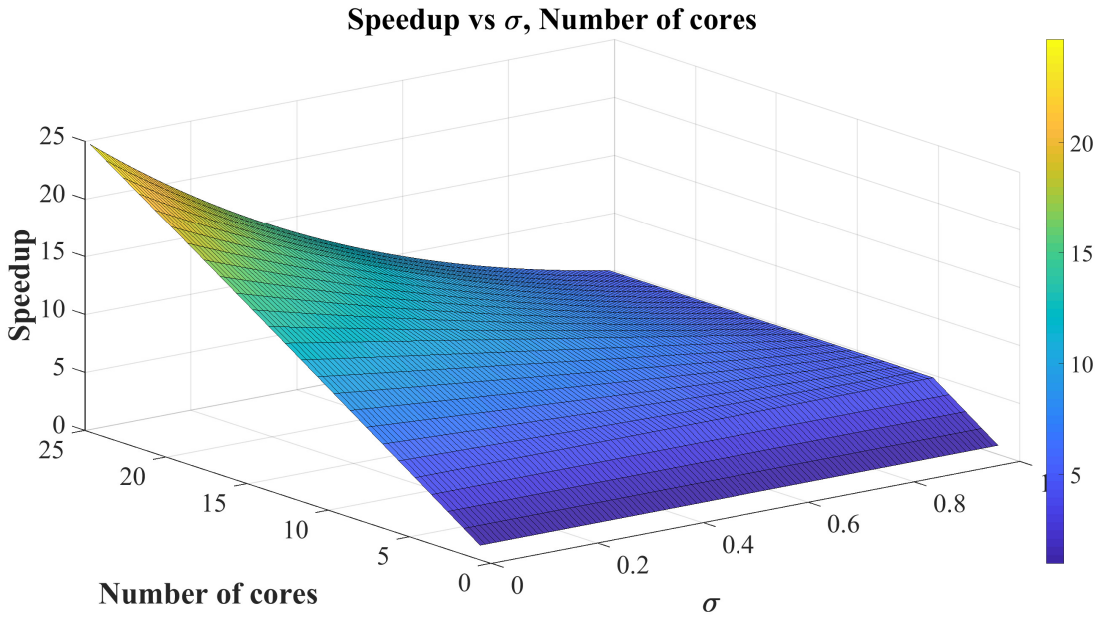


Figure 5.5: Sensitivity analysis of 5*n torus network (where $n = 1 \cdots 5$)

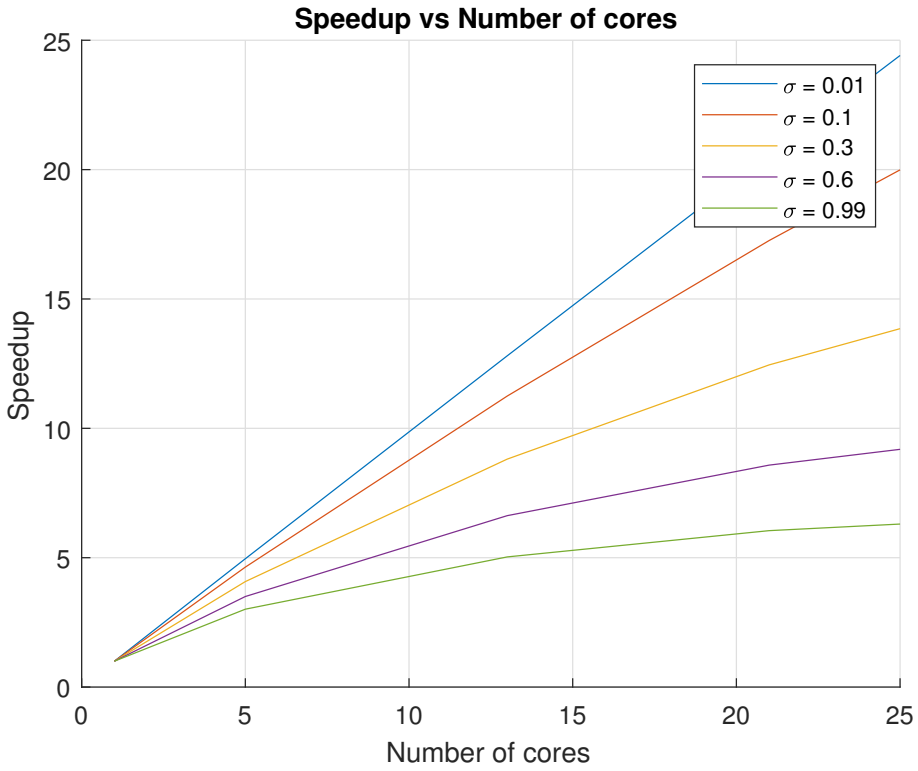


Figure 5.6: Data fraction curves of a 5*n torus network (where $n = 1 \cdots 5$) vs σ

Fig. 5.6 illustrates that as the value σ decreases, the speedup efficiency increases. For fast communication ($\sigma = 0.01$), the speedup is almost 25, as all 25 processors are uniformly engaged in processing. For slow communication ($\sigma = 0.99$), the speedup is about 5 as there are P_0 and its four neighbors are engaged into the processing load.

5.2.3 Multi-source Uniform Data Fraction

We extend the single source assignment to multi-source assignment problem [18] [**?**]. According to each processor, we concentrate on the processors' geographical location P_i, data fraction assigned α_i.

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

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

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

Situation I

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

Torus connected subgraph data injection Fig. 5.7 Fig. 5.9 show a connected subgraph data injection situation.

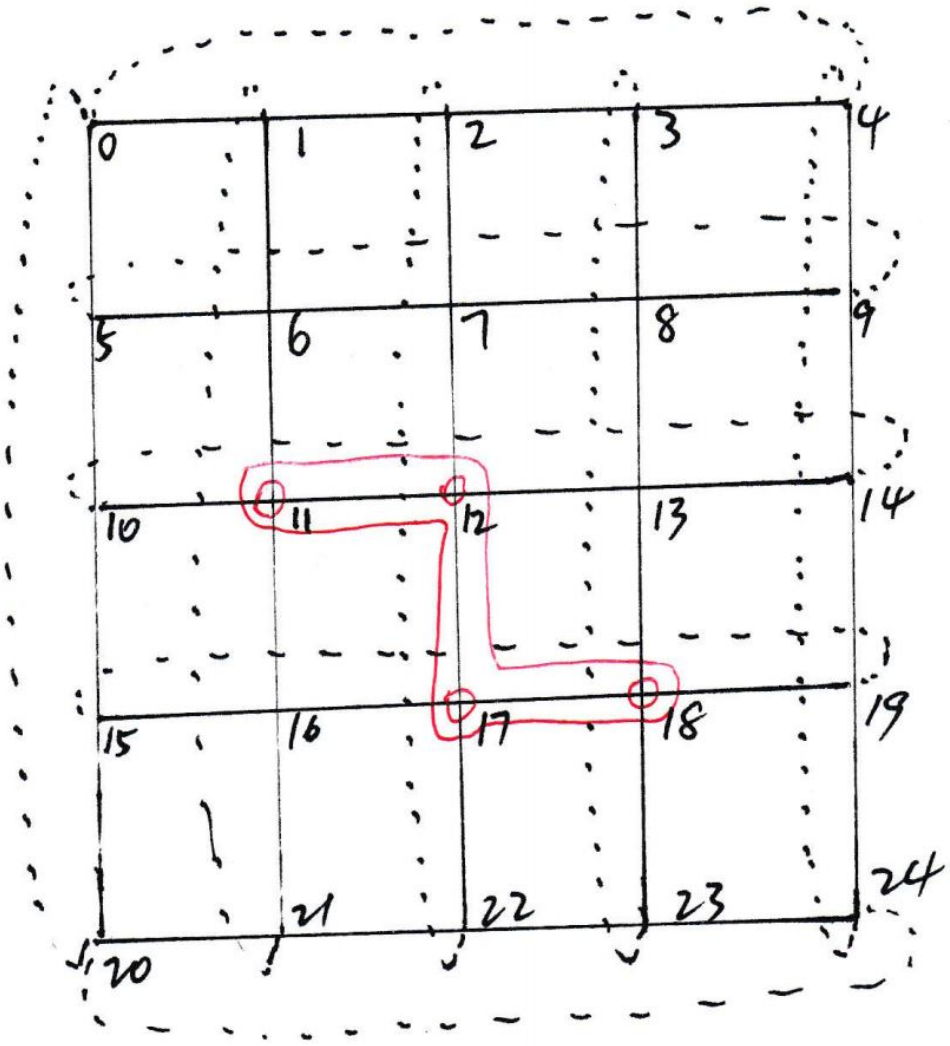


Figure 5.7: Toroidal mesh network subgraph data injection

A connected subgraph Fig. 5.7's flow matrix equation is

$$\begin{bmatrix} 4 & 8 & 9 & 4 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.5}$$

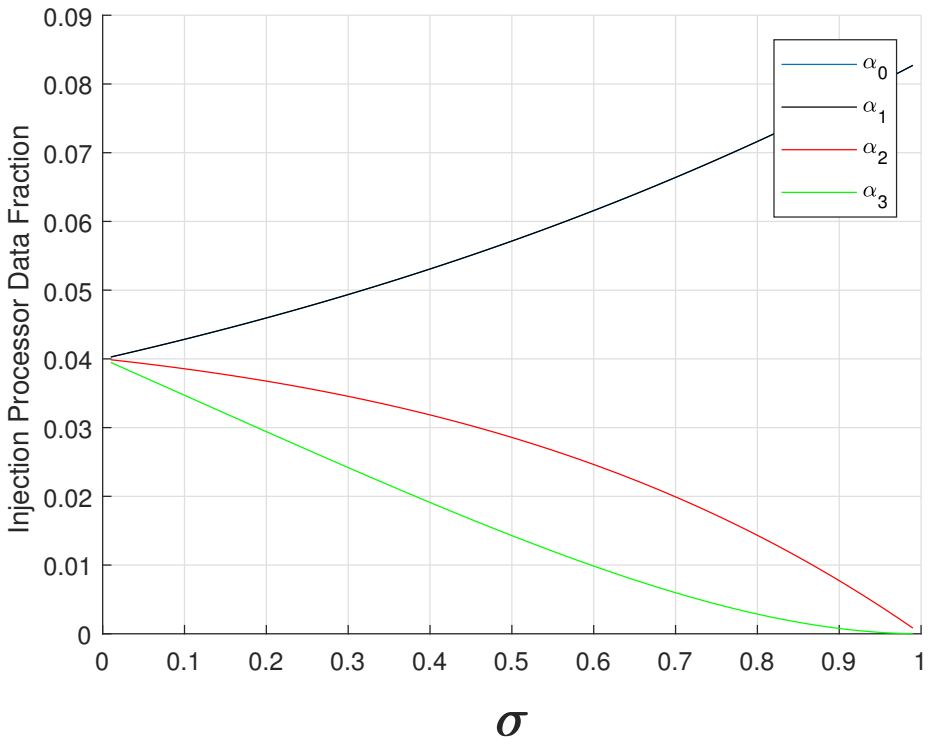


Figure 5.8: Data fraction curves of connected subgraph Fig. 5.7

The load fraction are 0.04 a small σ (fast communication) as all 25 processors are uniformly engaged processing. For slow communication, each of 12 processors (4 in the cluster and 8 in the adjacent neighbors) receives $\frac{1}{12}$ of the load as it is not economical to distribute load beyond those 12 processors.

Fig. 5.9 shows another connected subgraph data injection example.

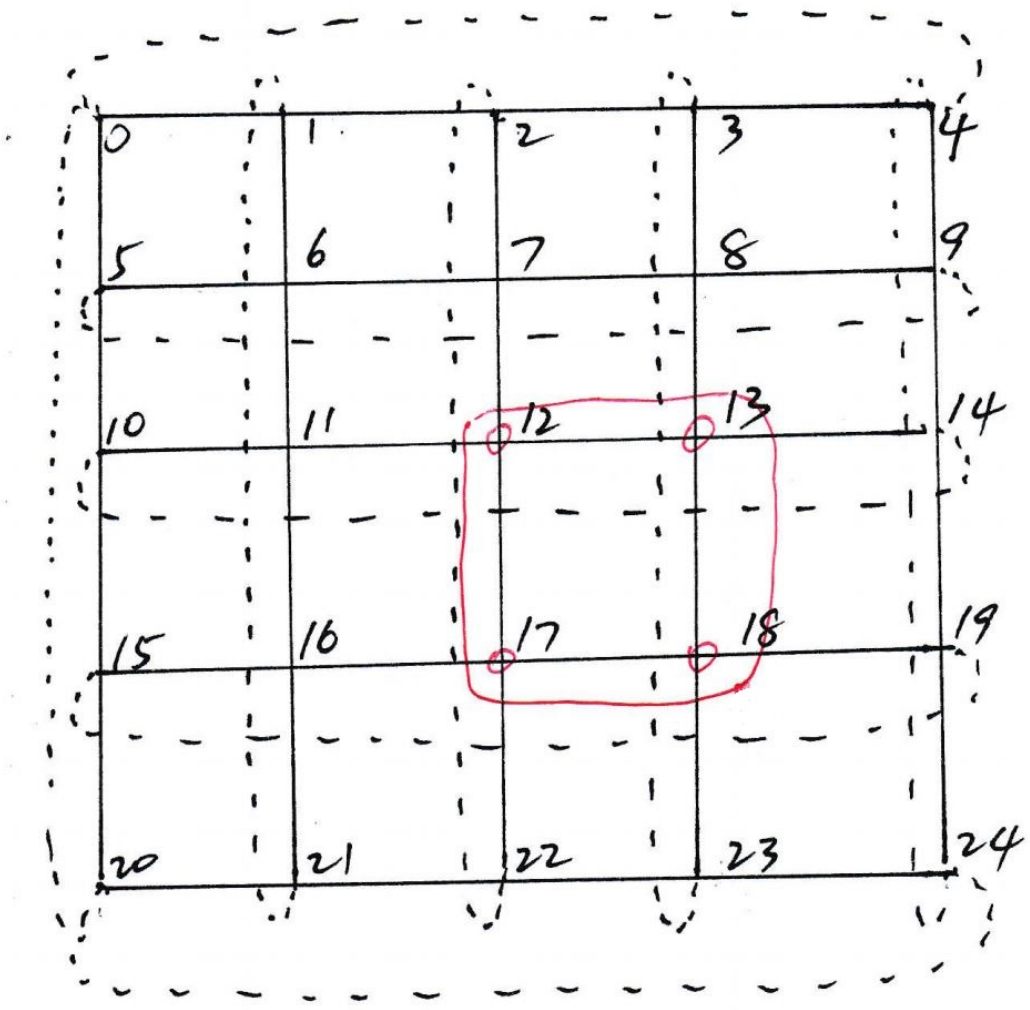


Figure 5.9: Toroidal mesh network subgraph data injection

Fig. 5.9's flow matrix equation is :

$$\begin{bmatrix} 4 & 8 & 9 & 4 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.6}$$

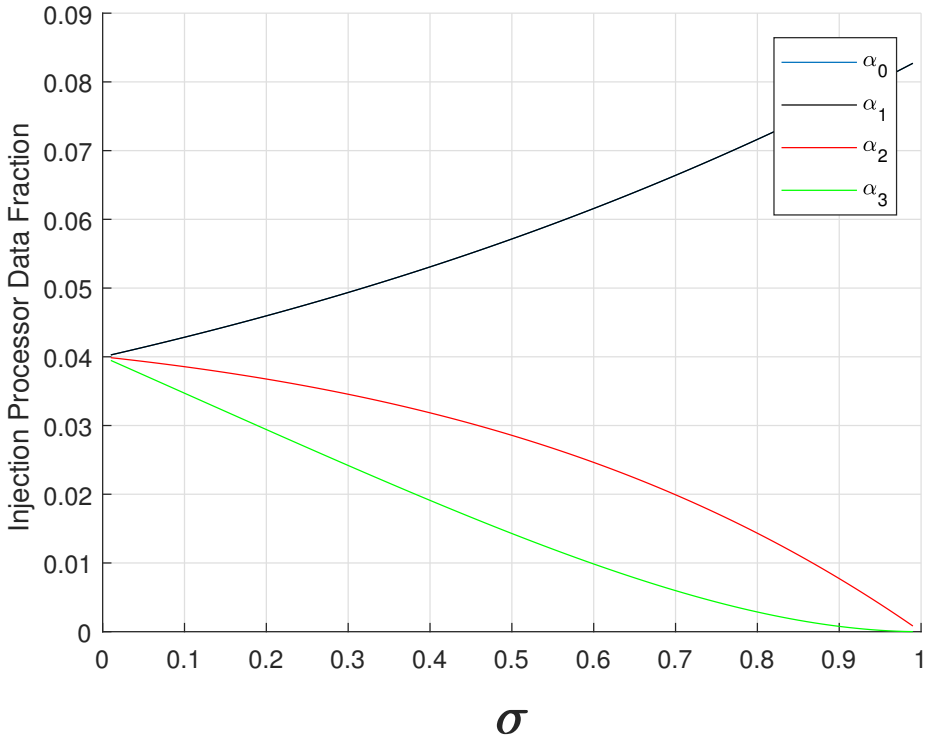


Figure 5.10: Data fraction on different type processors for Fig. 5.9

Since Fig. 5.10 also involves 25 total processors and 12 cluster processor adjacent to the source, it is identical to Fig. 5.8.

Situation II

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

Fig. 5.11 [62] provides a torus Voronoi method, which is used here. It extends the original domain to 8 copies and calculate the Voronoi Diagram as a planar algorithm [63]. Then the corner part is the torus Voronoi diagram.

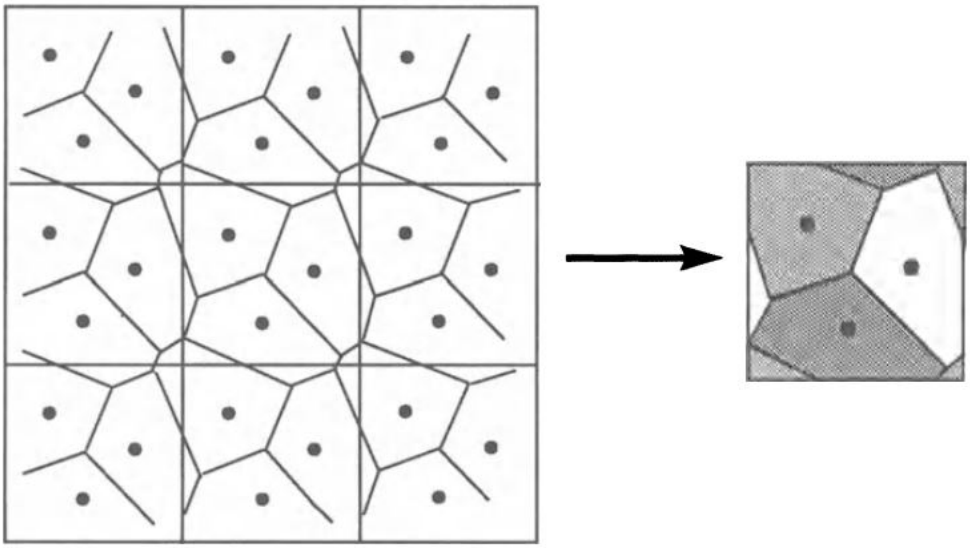


Figure 5.11: How to calculate torus Voronoi Diagram

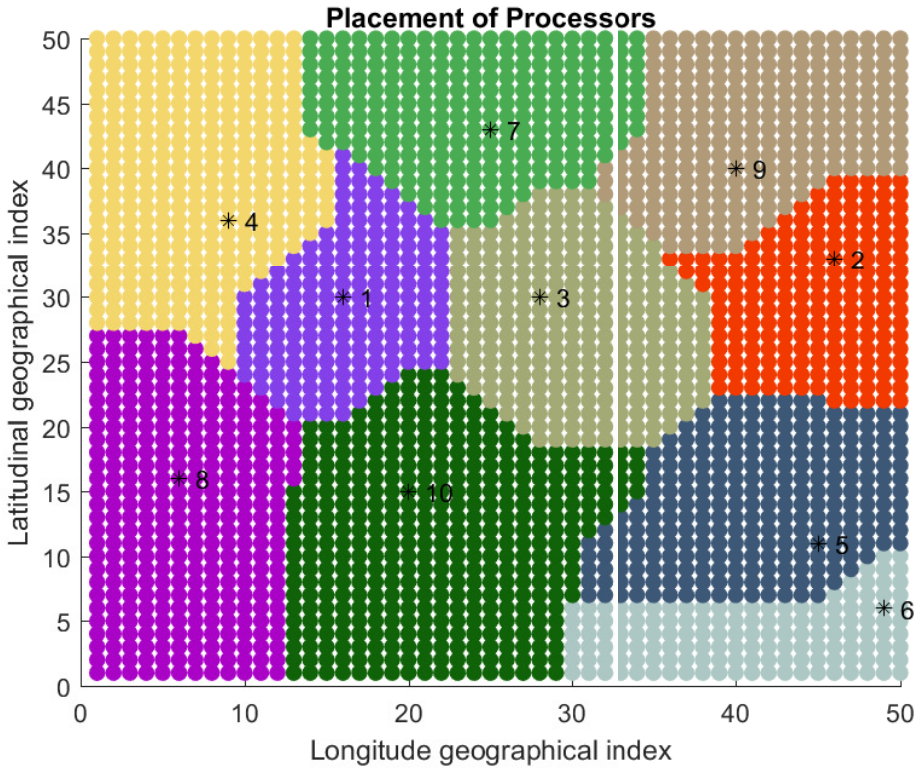


Figure 5.12: Initial Manhattan distance Voronoi Digram

Fig. 5.12 says the initial Voronoi division of a $50 * 50$ mesh network. The base is the data injection and the processor Voronoi cell tackle base's workload together. The user case contains 10 data injections and each cell's speedup curve shows in Fig. 5.15.

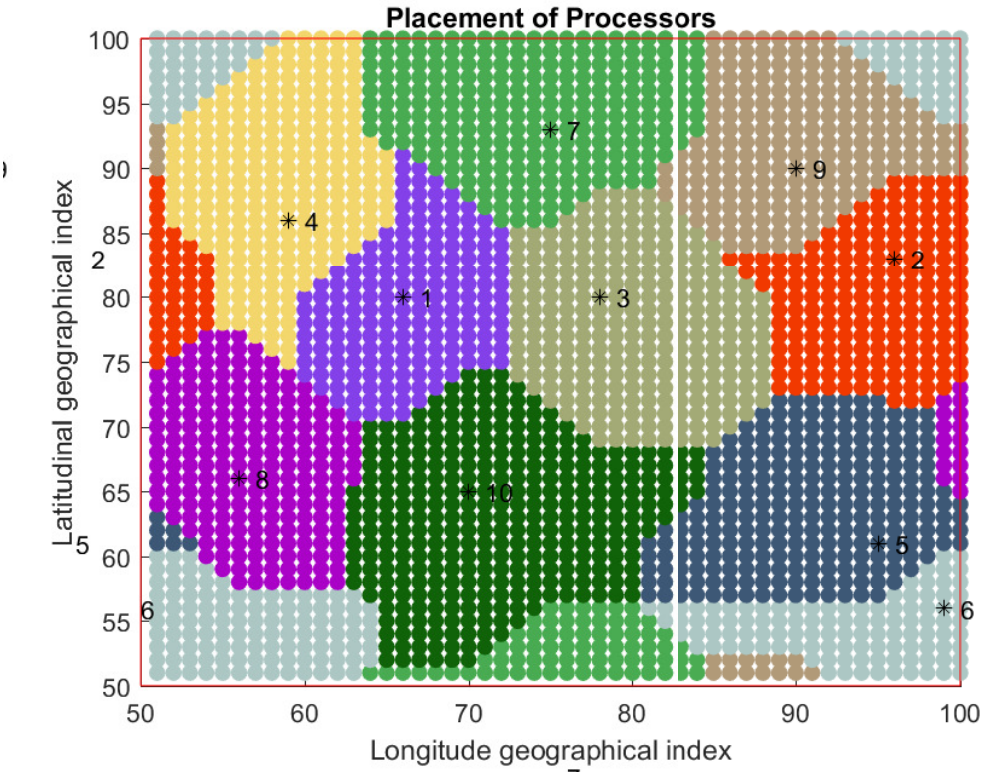


Figure 5.13: Torus Manhattan distance Voronoi diagram

Torus Manhattan distance Voronoi diagram Fig. 5.13 shows a toroidal Voronoi division result. The same color parts connect with each other and consist of a whole division in toroidal rectangle network. For example, the four gray corners consist of a whole area and the injection processor index is 6.

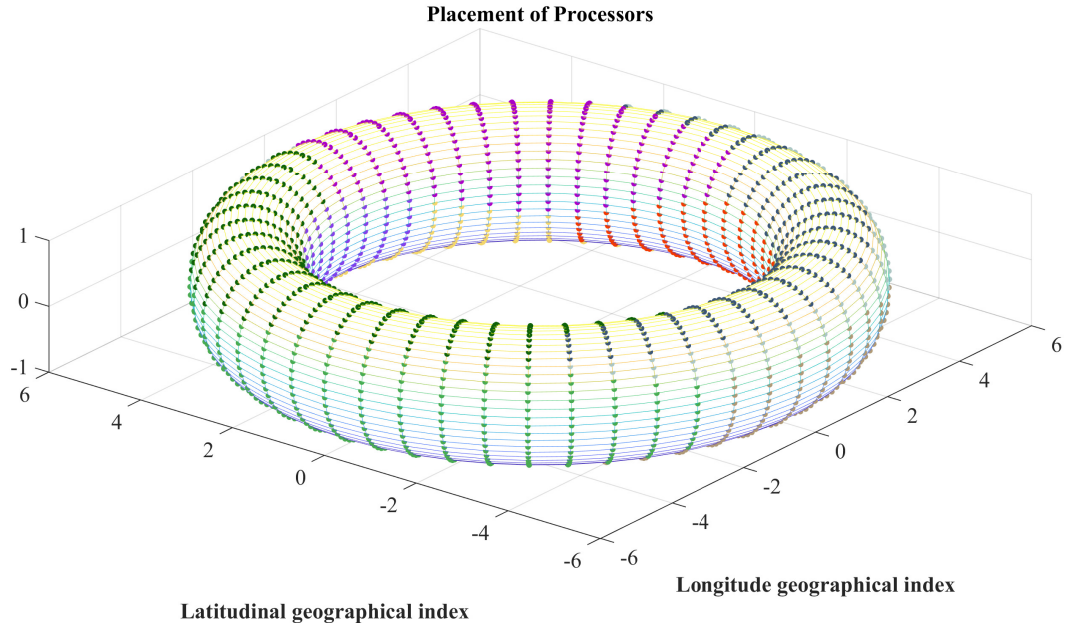


Figure 5.14: Voronoi diagram casting to the torus

Fig. 5.14 shows the Fig. 5.13 casts to the toroidal rectangle network result.

- The dot represents the processor.
- Different color represents the different cell in planar graph.
- Merge the left and right side of Fig. 5.13 to obtain a cylinder.
- Merge the top and bottom side of cylinder to obtain a torus.

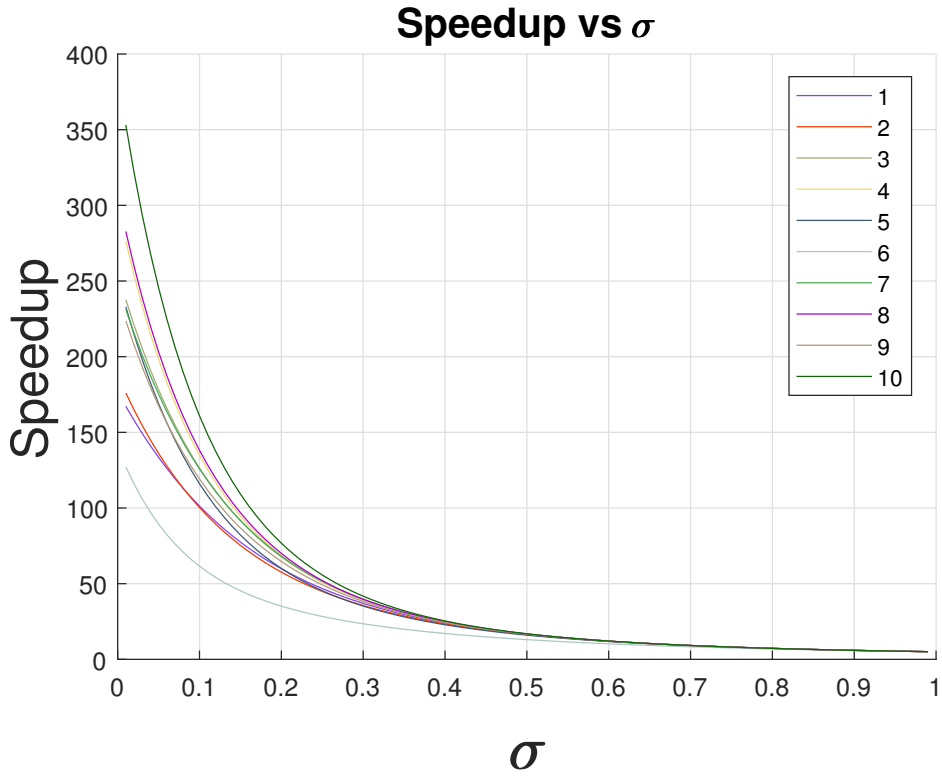


Figure 5.15: Toroidal Voronoi cells' speedup curves

- Different color represents the different Voronoi cell.
- Each curve represents the speedup efficiency of each Voronoi cell in different σ value.
- The maximum size of Voronoi cell is about 351 and the minimum size of Voronoi cell is 128.
- If $\sigma \approx 0$, the speedup ratio is similar with the number of processors in each cell.
- Even the processor number of Voronoi cell is different, $\sigma \geq 0.4$, the speedup ratio is about 1 in different cell.

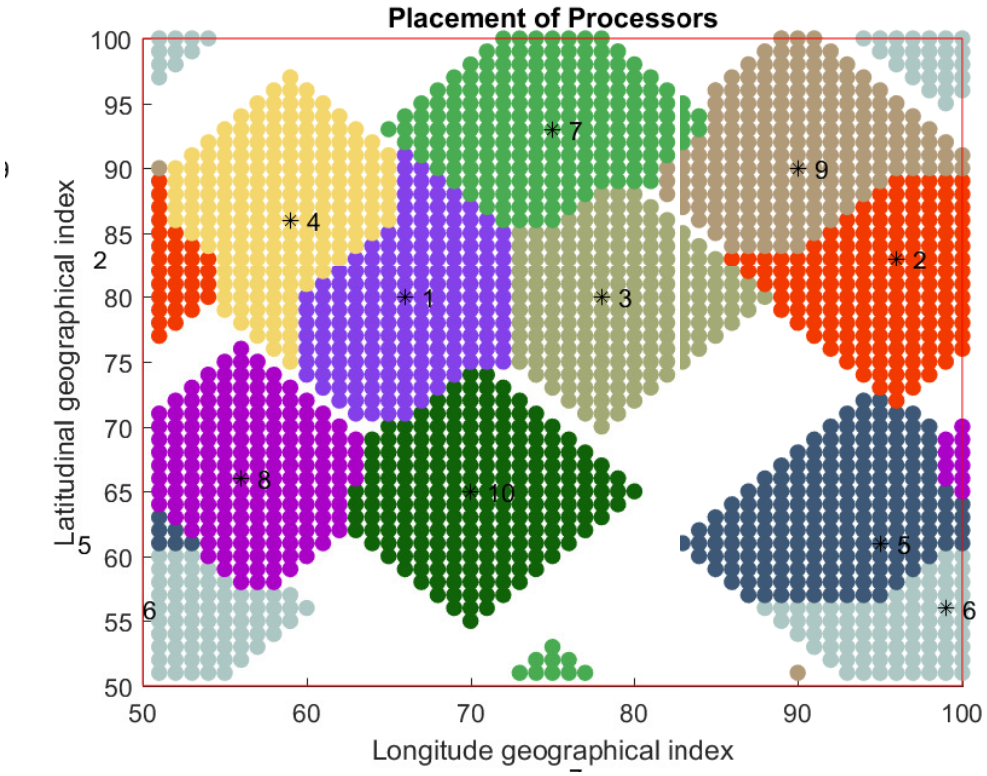


Figure 5.16: Torus reduced Voronoi diagram

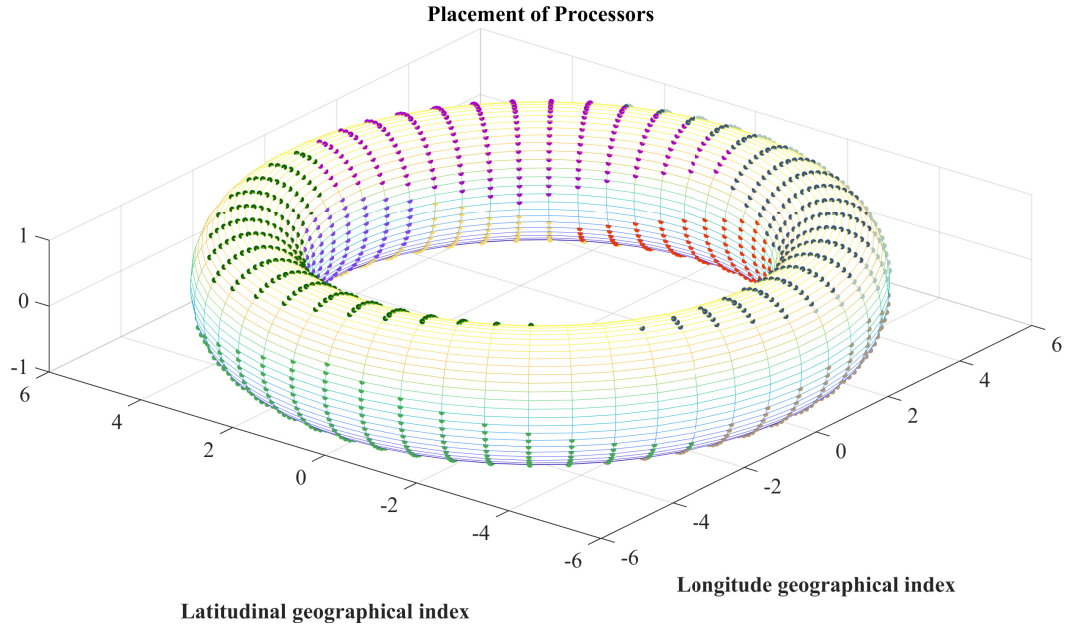


Figure 5.17: Torus reduced Voronoi diagram casting to a toroidal rectangle network

Fig. 5.17 shows the Fig. 5.16 casts to the toroidal rectangle network result.

- The dot represents the processor.
- The mesh grid without dot represents there is empty without processor.
- Different color represents the different cell in planar graph.
- Merge the left and right side of Fig. 5.13 to obtain a cylinder.
- Merge the top and bottom side of cylinder to obtain a torus.

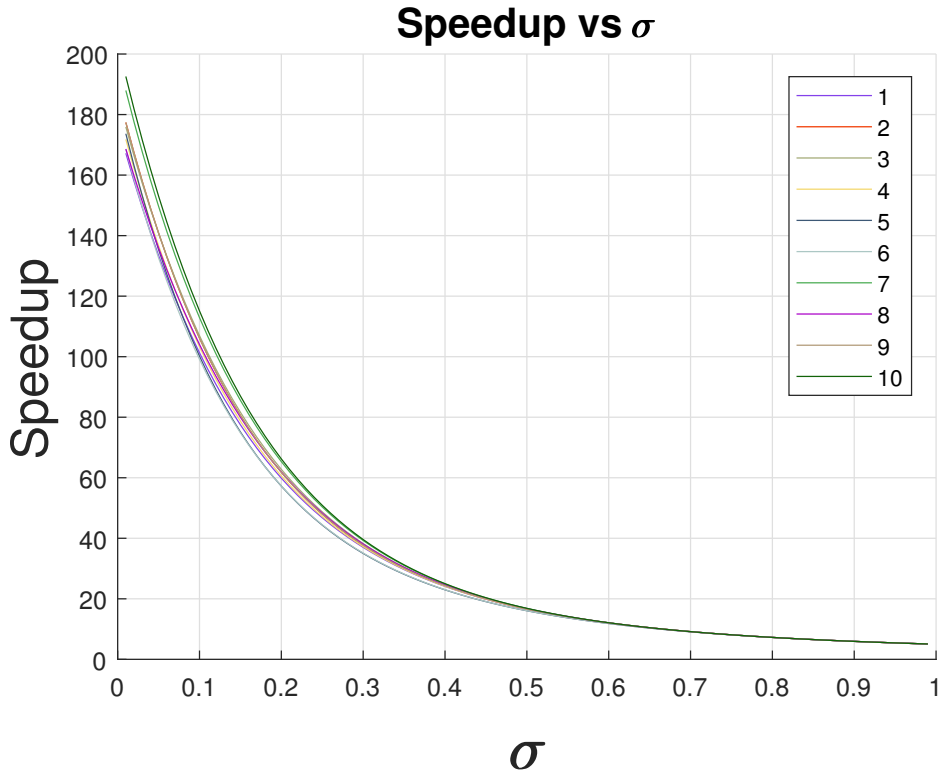


Figure 5.18: Torus reduced Voronoi diagram speedup curves

- Different color represents the different Voronoi cell.
- Each curve represents the speedup efficiency of each Voronoi cell in different σ value.
- The maximum size of Voronoi cell is about 191 and the minimum size of Voronoi cell is 168.
- Even the processor number of Voronoi cell is different, $1 > \sigma > 0$, the speedup ratio is about 1 in different Voronoi cell.

In the reduced model Fig. 5.17 and Fig. 5.18, reduced toroidal Voronoi diagram save 27% processors for the same processing capacity.

The ratio speedup is about $\frac{Sp_{max}}{Sp_{min}} = \frac{351}{125} \approx 2.816$. After the reduced stage, the ratio is $\frac{191}{168} \approx 1.137$. That is the reduced heuristic algorithm obtaining more balanced computation capacity distribution.

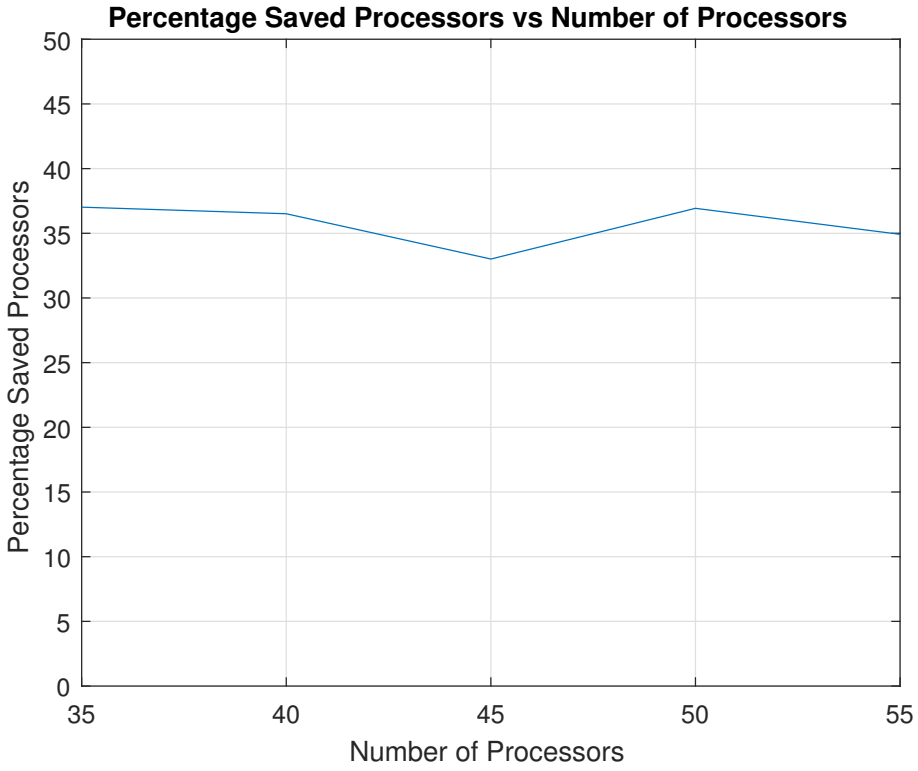
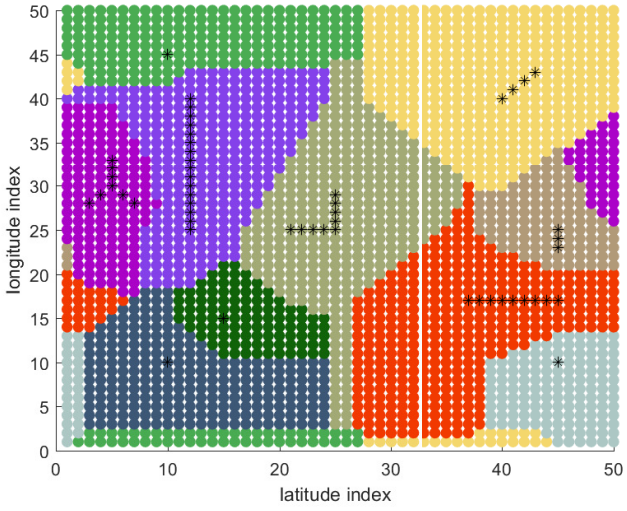


Figure 5.19: Torus reduced Voronoi diagram average saved processors

Situation III

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

Table 5.3: Constrained Voronoi Diagram and RCVD



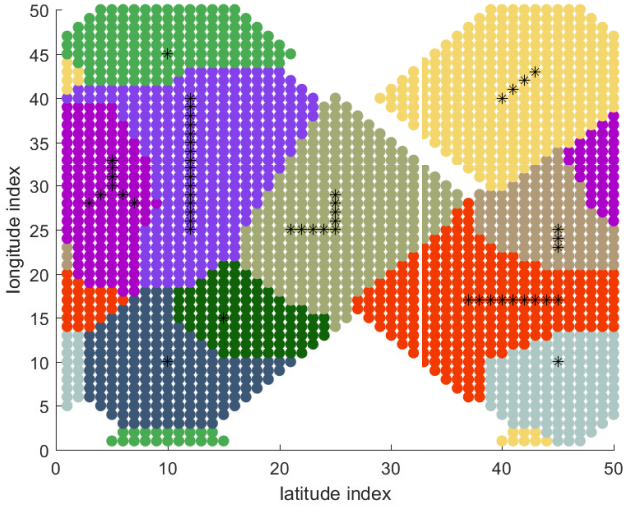
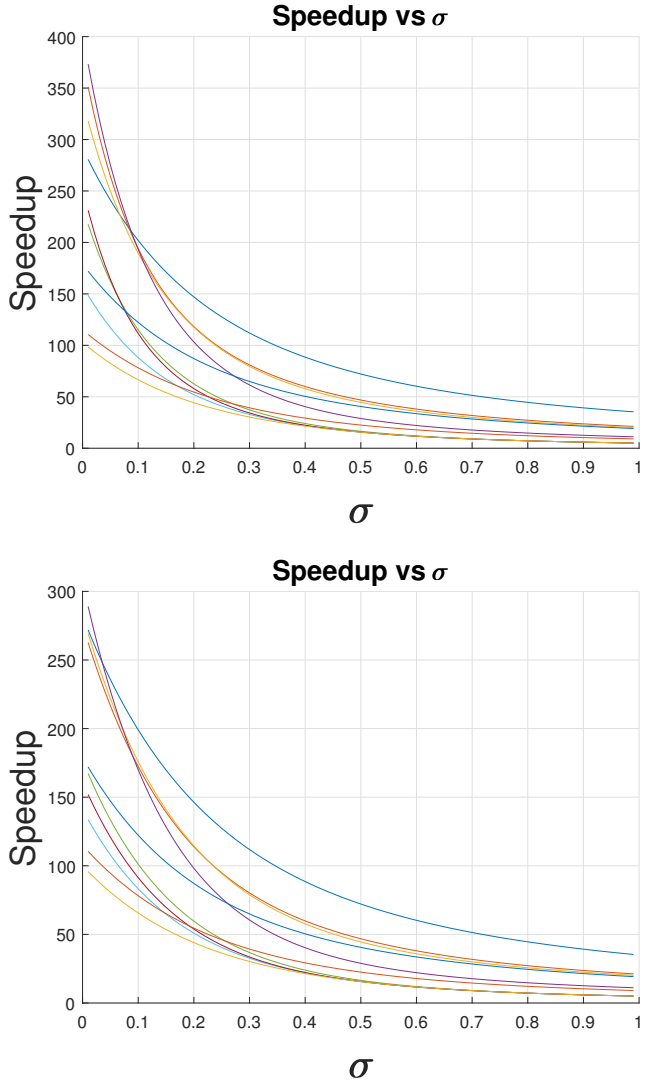


Table 5.4: Speedup curves Vs RCVDA speedup curves



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

5.2.4 Multi-source Non-uniform Data Injection

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

5.3 Modified Store and Forward Switching Scenario

5.3.1 Data Injection On The Grid Processor

We implement a modified store and forward policy where each node must receive tis entire load before it start processing. However, it is assumed once a node receive its entire load it can forward load to successive nodes with zero delay in a virtual cut-through switching like mode. Certainly other policies can be modeled and solved which is not done here for space reasons.

We utilize the $\sigma^\star$ to present $-(\sigma+1)$. The flow matrix closed-form of Fig. 5.3 is:

$$\begin{bmatrix} 1 & 4 & 8 & 10 & 8 & 4 & 1 \\ 1 & \sigma^\star & 0 & 0 & 0 & 0 & 0 \\ 1 & -\sigma & \sigma^\star & 0 & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma^\star & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -\sigma & -\sigma & \sigma^\star \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.7}$$

The fraction curve result is :

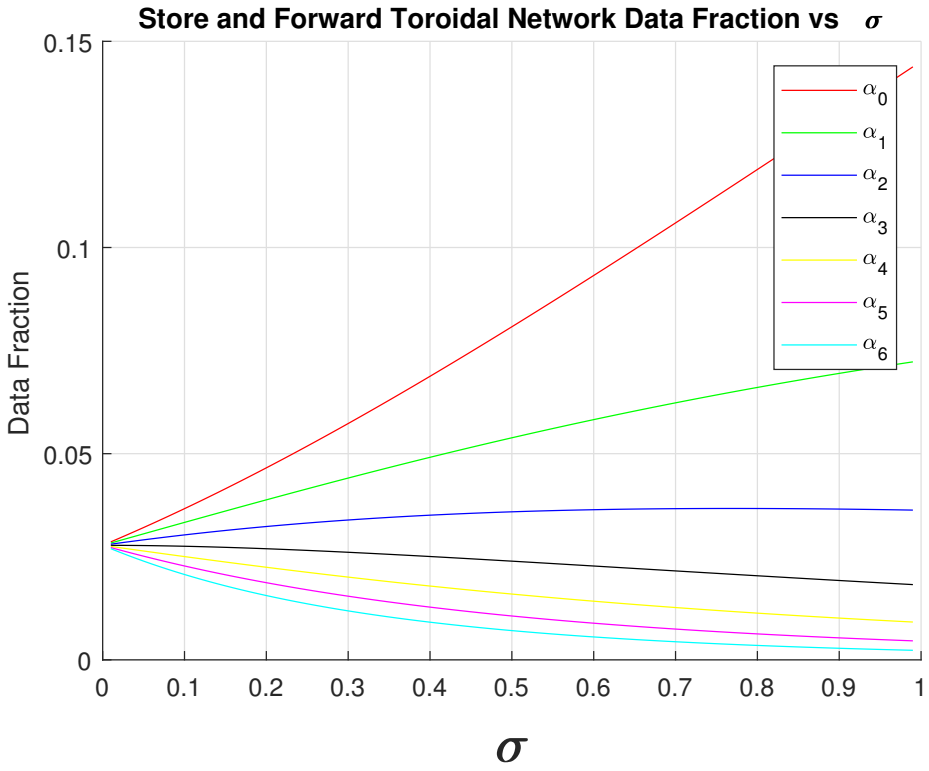


Figure 5.20: The data fraction curve of $6 * 6$ torus network Fig. 5.3

From Fig. 5.20, we see that as the value σ grows, more and more workload is assigned to the $P_{4,2}$ and its one hop neighbors. Since there are 36 processors for $\sigma \approx 0$ (fast communication), $\frac{1}{36}$ of the load is assigned uniformly to each processor. That is, as the communication ability decreases, the economical method is to locally process the job.

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

Table 5.5: D_i vs Number of processor on D_i

5.3.2 Sensitivity Analysis of Toroidal Rectangle Network

Considering a $5 * n$ $(n = 1 \cdots 5)$ toroidal rectangle network, the $level_i$ table shows Table 5.5:

So the simulation result illustrates in Fig. 5.21

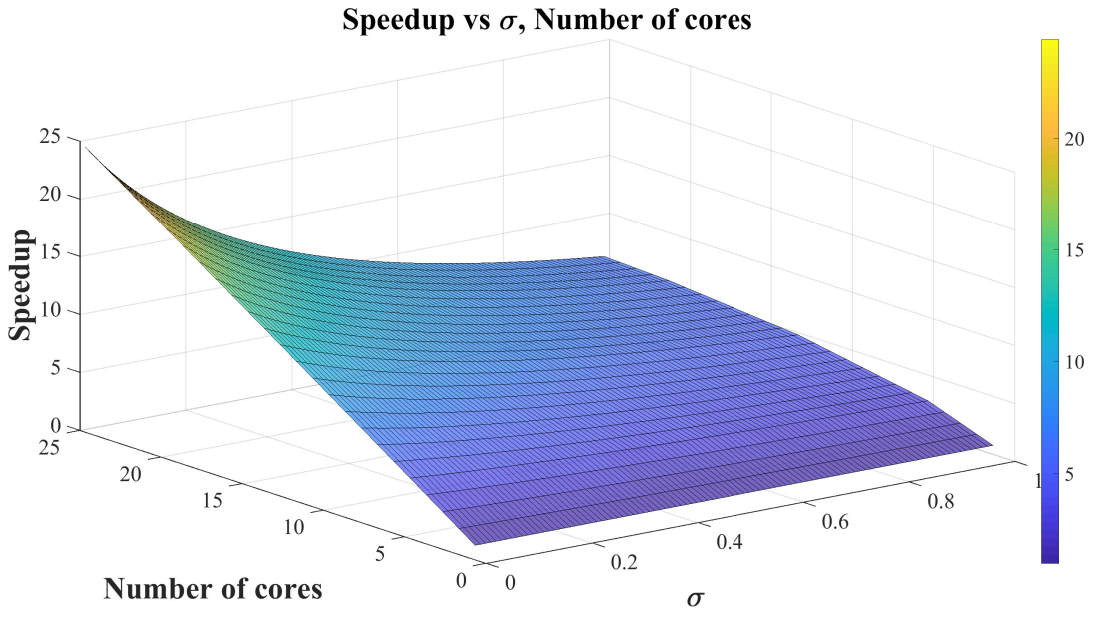


Figure 5.21: Sensitivity analysis result of $5 * n$ torus network ($n = 1 \cdots 5$)

5.3.3 Multi-source Uniform Data Injection

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

5.3.4 Multi-source Non-uniform Data Injection

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

5.4 Comparison Result

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

Considering a $5 * n$ ($n = 1 \cdots 5$) mesh network Fig. 2.1, the best position for data injection is P_{12}. Other positions, for example P_8, P_{13}, they don't have the same speedup efficiency. Yet, for a toroidal $5 * 5$ regular mesh network, each position's efficiency is equal. Fig. 5.22 explores the comparison result between the toroidal and corner scenario difference.

5.4.2 Comparison Result With Corner Processor and Inner Grid Processor

For a $5 * n$ $(n = 1 \cdots 5)$ mesh network, the inner grid position is P_{12} and the corner data injection position is P_0.

The comparing result is Fig. 5.22.

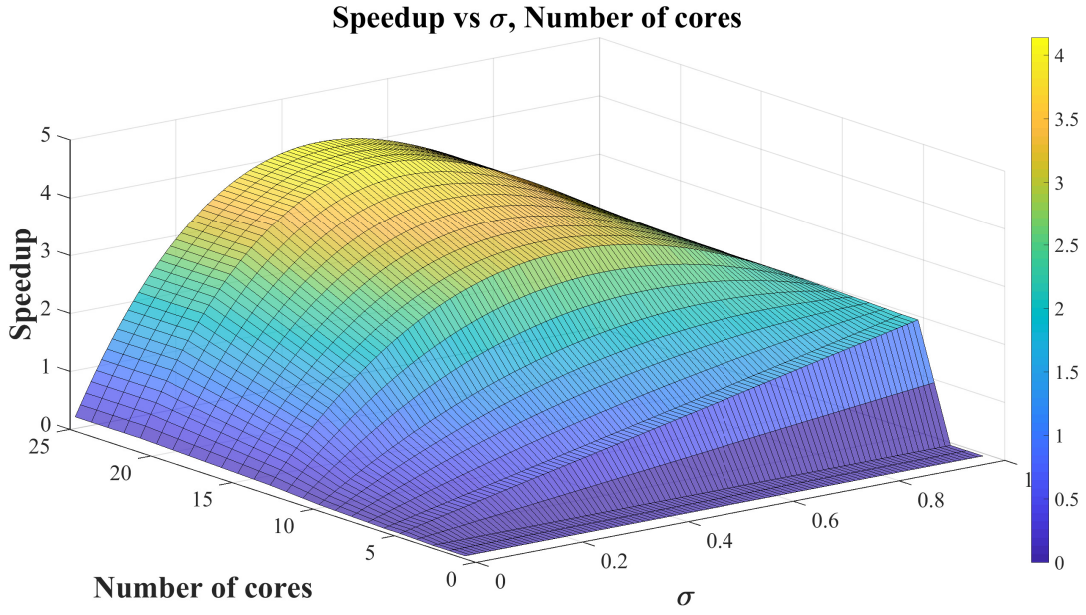


Figure 5.22: Speedup difference between inner grid injection and corner injection in $5 * n$ network $(n = 1 \cdots 5)$

Generally speaking, Fig. 5.22 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.

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

In the legend of figures, we use

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

Data Injection On the Corner Processor

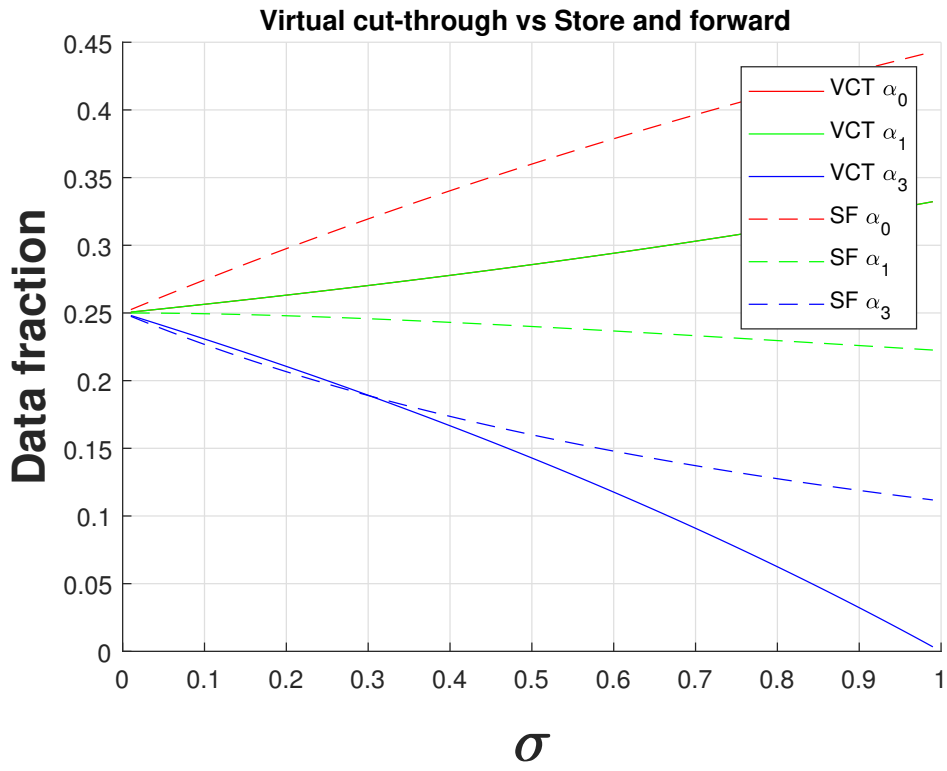


Figure 5.23: Comparing result between virtual cut-through switching and store and forward switching in $2 * 2$ mesh network

Fig. 5.23 says that P_0 takes more of the assigned load in store and forward switching scenario than virtual cut-through switching situation. As the value of σ increases, the fractions that are deployed to the deeper layers decreases in both cases. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to P_3 in the virtual cut-through switching assumption, yet in the store and forward switching situation, there is still about 10% data fraction are communicated to P_3. For fast communication ($\sigma \approx 0$), each of the four processors receive 25% of the load.

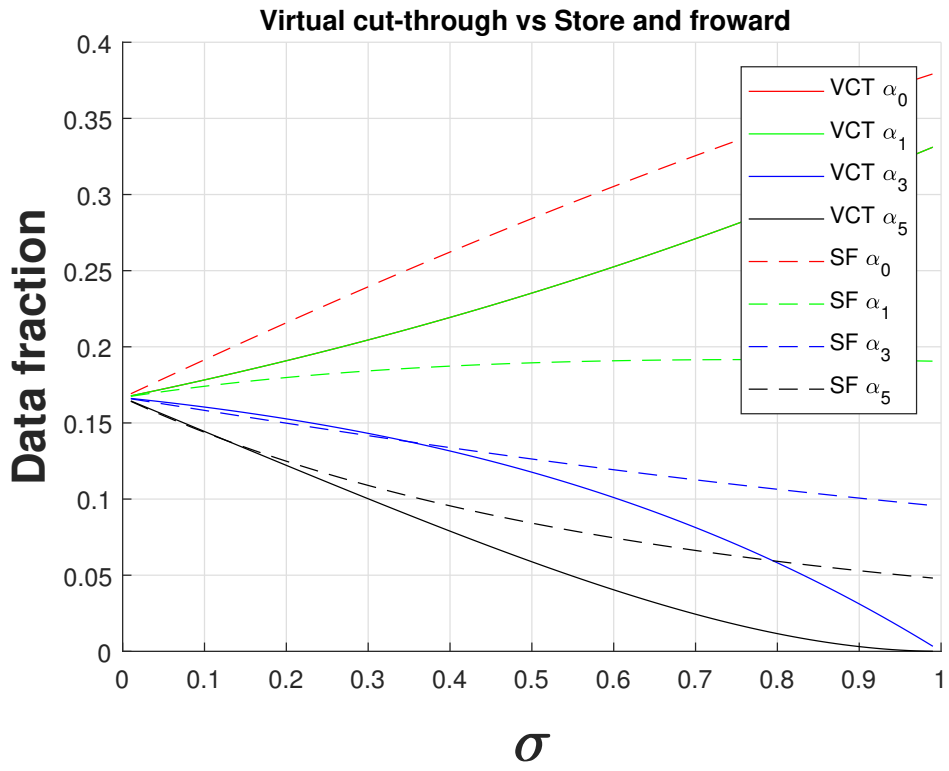


Figure 5.24: Comparing result between virtual cut-through switching and store and forward switching in $2 * 3$ mesh network

Fig. 5.24 says that P_0 takes more assigned task in store and forward switching scenario than virtual cut-through switching situation. As the value of σ increases, the fractions that 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_5 in the virtual cut-through switching assumption. Yet in the store and forward switching situation, there is still about 5% data fraction is communicated to P_5. For fast communication ($\sigma \approx 0$), each of the six processors receive $\frac{1}{6}$ of the load.

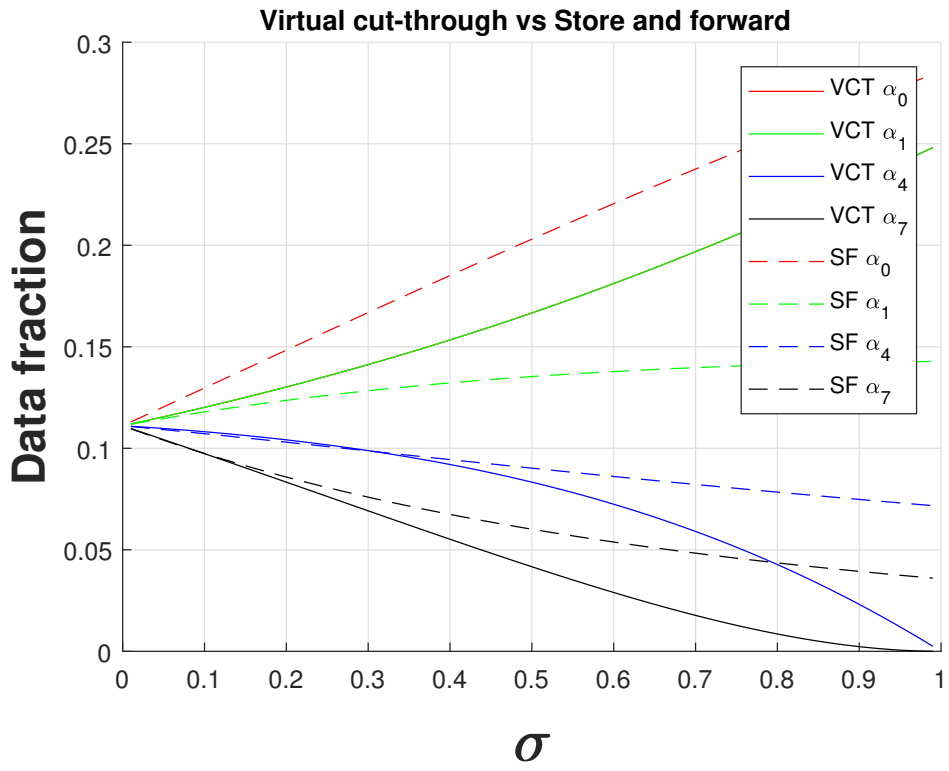


Figure 5.25: Comparing result between virtual cut-through switching scenario and store and forward switching scenario. The data injection is boundary processor of a $3 * 3$ mesh network.

Fig. 5.25 says that P_0 takes more assigned task in store and forward switching scenario than virtual cut-through switching situation. As the value of σ increases, the fractions that are deployed to the deeper levels decreases. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_3$, that is, P_7 and P_8 in the virtual cut-through switching assumption. Yet in the store and forward switching situation, there is still about 5% data fraction is communicated to P_7 and P_8. For fast communication ($\sigma \approx 0$), each of the nine processors receive $\frac{1}{9}$ of the load.

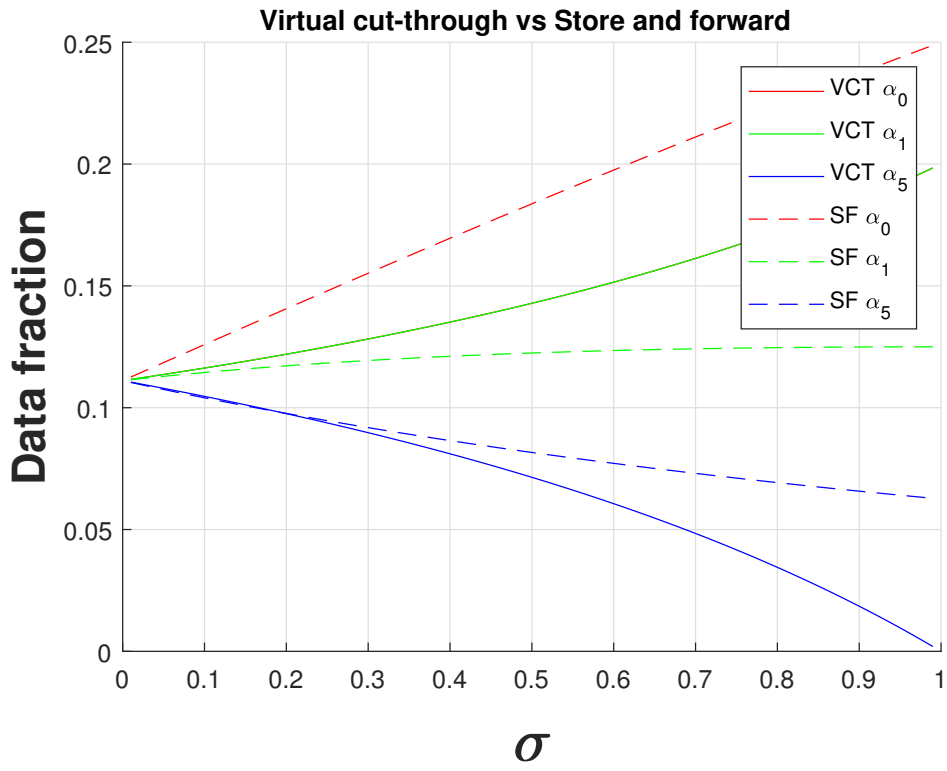


Figure 5.26: The comparing result between virtual cut-through switching processor with without virtual cut-through switching processor in 3*3 mesh network injection on inner grid processor

Fig. 5.26 says that P_0 takes more assigned task in store and forward switching scenario than virtual cut-through switching situation. As the value of σ increases, the fractions that are deployed to the deeper levels dropping down. In the limit condition, for example, $\sigma = 1$, there is no data transmitted to $level_2$, that is, P_5, P_6, P_7 and P_8 in the virtual cut-through switching assumption. Yet in the store and forward switching situation, there is still about 5% data fraction is communicated to P_5, P_6, P_7 and P_8.

Comparing with Fig. 5.25, P_0 takes less workload in inner grid position than boundary data injection. The reason is there are 4 neighbor processors on the $level_1$, yet there is only three processors on $level_1$ on the boundary.

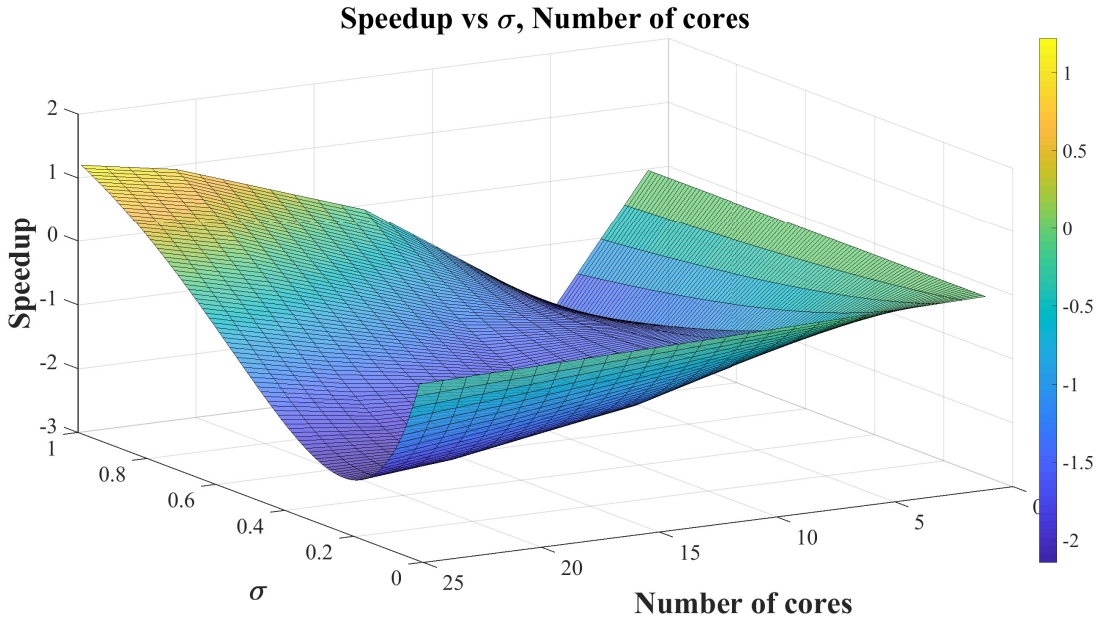


Figure 5.27: Speedup difference between store and forward switching and virtual cut-through switching in $5 * n$ mesh network ($n = 1 \cdots 5$)

Fig. 5.27 shows the speedup difference between the virtual cut-through switching situation and modified store and forward switching scenario.

Chapter 6

Hypercube Network and General Network

6.1 Hypercube Network Virtual Cut-through Switching Scenario

The hypercube topology has two nodes along each dimension and $\log_2 n$ dimensions. The construction of a hypercube goes as follows, in general a $d-$dimensional hypercube is constructed by connecting corresponding nodes of two $(d-1)$ dimensional hypercubes Fig. 6.1.

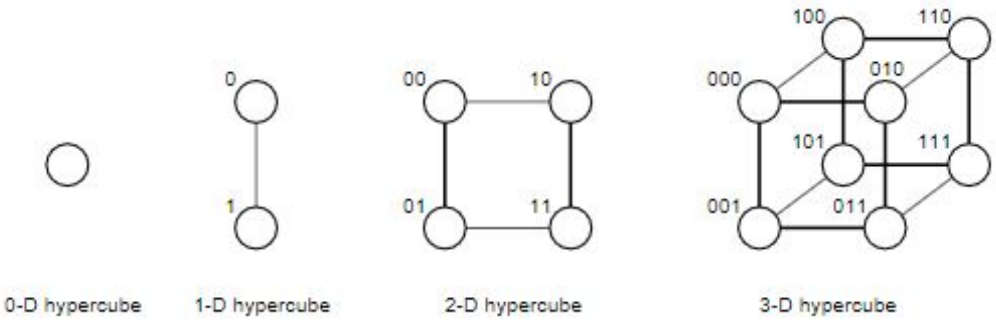


Figure 6.1: Hypercube in 0, 1, 2, 3 dimension. [1]

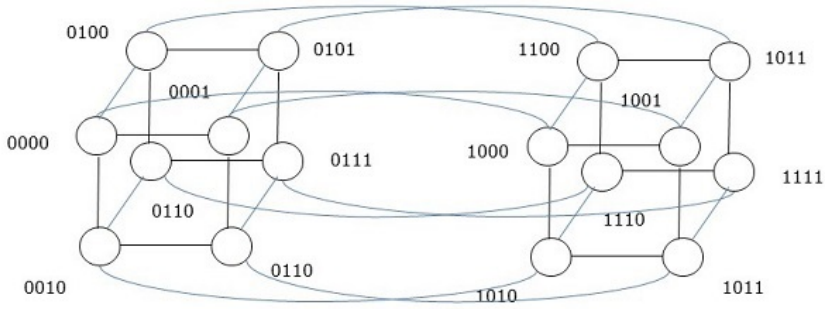


Figure 6.2: A hypercube network

6.1.1 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 [39],

Lemma 6.1.1. *In each layer i of D dimensional hypercube, there are $\binom{n}{i}$ processors each of which can be accessed through i communications links and is capable of transmitting to $d - i$ still idle processors.*

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

$$A = \begin{bmatrix} \binom{2}{0} & \binom{2}{1} & \binom{2}{2} \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 0 & \sigma - 1 & 1 \end{bmatrix} \tag{6.1}$$

, which is investigated in Regular Network Chapter.

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

$$A = \begin{bmatrix} \binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -1 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 \\ 0 & \sigma-1 & \sigma & 1 \end{bmatrix} \tag{6.2}$$

The equivalence inverse processing speed :

$$\hat{T_f} = 1 * w_{eq} * T_{cp}$$

$$w_{eq} = \alpha_0 * w$$

The speedup is :

$$Speedup = \frac{T_f}{\hat{T_f}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

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

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

6.2 General Case Virtual Cut-through Switching Scenario

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.3}$$

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

Finally, the speedup is:

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |-\det A|$$

.

6.3 Hypercube Network Modified Store and Forward Switching Scenario

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

$$A = \begin{bmatrix} \binom{2}{0} & \binom{2}{1} & \binom{2}{2} \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 1 & -(\sigma+1) & 0 \\ 1 & -\sigma & -(\sigma+1) \end{bmatrix} \quad (6.4)$$

, which is investigated in Regular Network Chapter.

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

$$A = \begin{bmatrix} \binom{3}{0} & \binom{3}{1} & \binom{3}{2} & \binom{3}{3} \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 1 & -(\sigma+1) & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \quad (6.5)$$

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

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

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

6.4 General Case Store and Forward Switching Scenario

$$\begin{bmatrix} 1 & m_1 & m_2 & \cdots & m_{n-2} & m_{n-1} & m_n \\ 1 & -(\sigma+1) & 0 & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -(\sigma+1) & \cdots & 0 & 0 & 0 \\ 1 & -\sigma & -\sigma & -(\sigma+1) & 0 & \cdots & 0 \\ 1 & -\sigma & -\sigma & -\sigma & -(\sigma+1) & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 1 & -\sigma & -\sigma & \cdots & -\sigma & -\sigma & -(\sigma+1) \end{bmatrix} \times \begin{bmatrix} \alpha_{l_0} \\ \alpha_{l_1} \\ \alpha_{l_2} \\ \alpha_{l_3} \\ \vdots \\ \alpha_{l_{n-1}} \\ \alpha_{l_n} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.6}$$

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

The speedup is

$$Speedup = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = \frac{\det A}{\det A^\star} = \left| \frac{\det A}{(\sigma^\star)^{n-1}} \right|$$

Chapter 7

Conclusion and Future work

7.1 Conclusion

In this work we investigate two problems. One problem is closed-form solutions for equivalence computation of divisible workload in mesh and toroidal networks and the other problem is scheduling divisible workloads from multiple sources in mesh and toroidal networks of processors.

We propose a flow matrix closed-form equation to present the equivalence, which allows a characterization of the nature of minimal time solution and a simple method to determine when and how much load to distribute for processors.

Also, we propose the use of a reduced Manhattan distance Voronoi diagram algorithm (RMDVDA) to minimize the overall processing time of these workloads by taking advantage of the processor equivalence technique. Three case studies with 10 sources of workloads are presented to illustrate the general approach for multiple sources of workloads.

In the first phase, a Manhattan distance Voronoi diagram is used to obtain a network cluster division. In the second phase, we propose an efficient algorithm to obtain near-optimal load distribution among processors represented by equivalent processors. The algorithm minimizes the number of processors

utilized.

Various factors, for example data injection position, data injection fraction, virtual cut-through switching and modified store and forward switching, the ratio of computation time over communication time, that influence the load distribution are shown in the numerical experiment. Experimental evaluation through simulations demonstrates that a task can be finished in the same suboptimal time and yet save over 40% of processor resources in mesh network and $11\% - 17\%$ in toroidal network.

7.2 Future Work

There are future projects that need attention.

- Given virtual cut-through and modified store and forward scenario, the hypercube network multi-source suboptimal algorithm to minimize the makespan.
- Given data injection workload, which are some dynamic workload queues, how to find the suboptimal algorithm to minimize the makespan.
- Given the workload fractions distribution, how to choose the appropriate data injection positions. The objective to minimize the cost function, which has relationship with Earth Mover Distance [59] and Wasserstein Distance [64].
- Other topology networks, for example connected cubic network graph [65], heterogeneous networks [66], Gaussian network [67] .
- Other constrain assumption, for example the front-end processor with finite-size buffer.

Bibliography

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

[2] G. N. Iyer, B. Veeravalli, and S. G. Krishnamoorthy, "On handling large-scale polynomial multiplications in compute cloud environments using divisible load paradigm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 820–831, 2012.

[3] Y. Kyong and T. G. Robertazzi, "Greedy signature processing with arbitrary location distributions: a divisible load framework," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 4, pp. 3027–3041, 2012.

[4] S. Suresh, C. Run, H. J. Kim, T. G. Robertazzi, and Y.-I. Kim, "Scheduling second-order computational load in master-slave paradigm," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 780–793, 2012.

[5] M. Hu and B. Veeravalli, "Dynamic scheduling of hybrid real-time tasks on clusters," *IEEE Transactions on Computers*, vol. 63, no. 12, pp. 2988–2997, 2014.

[6] L. Marchal, Y. Yang, H. Casanova, and Y. Robert, "A realistic network/application model for scheduling divisible loads on large-scale platforms," in *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pp. 10–pp, IEEE, 2005.

[7] H. J. Kim, G.-I. Jee, and J. G. Lee, "Optimal load distribution for tree network processors," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 32, no. 2, pp. 607–612, 1996.

[8] M. Drozdowski and W. Głazek, "Scheduling divisible loads in a three-dimensional mesh of processors," *Parallel Computing*, vol. 25, no. 4, pp. 381–404, 1999.

[9] M. Moges and T. G. Robertazzi, "Optimal divisible load scheduling and markov chain models," in *Proceedings of the 2003 Conference on Information Sciences and Systems*, 2003.

[10] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *computer*, vol. 35, no. 1, pp. 70–78, 2002.

[11] A. Ganguly, K. Chang, S. Deb, P. P. Pande, B. Belzer, and C. Teuscher, "Scalable hybrid wireless network-on-chip architectures for multicore systems," *IEEE Transactions on Computers*, vol. 60, no. 10, pp. 1485–1502, 2011.

[12] K. Tatas, K. Siozios, D. Soudris, and A. Jantsch, *Designing 2D and 3D network-on-chip architectures*. Springer, 2016.

[13] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks*, vol. 3, no. 2, pp. 97–133, 1973.

[14] 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.

[15] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial tessellations: concepts and applications of Voronoi diagrams*, vol. 501. John Wiley & Sons, 2009.

[16] I. Stojmenovic, A. P. Ruhil, and D. Lobiyal, "Voronoi diagram and convex hull based geocasting and routing in wireless networks," *Wireless communications and mobile computing*, vol. 6, no. 2, pp. 247–258, 2006.

[17] S. Meguerdichian, F. Koushanfar, G. Qu, and M. Potkonjak, "Exposure in wireless ad-hoc sensor networks," in *Proceedings of the 7th annual international conference on Mobile computing and networking*, pp. 139–150, ACM, 2001.

[18] 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.

[19] 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.

[20] J. M. Schopf, "A general architecture for scheduling on the grid," *Special issue of JPDC on Grid Computing*, vol. 4, 2002.

[21] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. Nudd, "Local grid scheduling techniques using performance prediction," *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 2, pp. 87–96, 2003.

[22] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 202–219, Springer, 1999.

[23] L. Yang, J. M. Schopf, and I. Foster, "Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, p. 31, ACM, 2003.

[24] K. Ranganathan and I. Foster, "Decoupling computation and data scheduling in distributed data-intensive applications," in *null*, p. 352, IEEE, 2002.

[25] V. Subramani, R. Kettimuthu, S. Srinivasan, and S. Sadayappan, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pp. 359–366, IEEE, 2002.

[26] M. Wu and X.-H. Sun, "Memory conscious task partition and scheduling in grid environments," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 138–145, IEEE Computer Society, 2004.

[27] J. H. Abawajy, "Fault-tolerant scheduling policy for grid computing systems," in *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, p. 238, IEEE, 2004.

[28] L. Xiao, Y. Zhu, L. M. Ni, and Z. Xu, "Gridis: An incentive-based grid scheduling," in *null*, p. 65b, IEEE, 2005.

[29] A. J. Chakravarti, G. Baumgartner, and M. Lauria, "Application-specific scheduling for the organic grid," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pp. 146–155, IEEE Computer Society, 2004.

[30] E. Huedo, R. S. Montero, and I. M. Llorente, "Experiences on adaptive grid scheduling of parameter sweep applications," in *null*, p. 28, IEEE, 2004.

[31] N. Fujimoto and K. Hagihara, "A comparison among grid scheduling algorithms for independent coarse-grained tasks," in *Applications and the Internet Workshops, 2004. SAINT 2004 Workshops. 2004 International Symposium on*, pp. 674–680, IEEE, 2004.

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

[33] 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.

[34] T. G. Robertazzi, "Ten reasons to use divisible load theory," *Computer*, vol. 36, no. 5, pp. 63–68, 2003.

[35] Y.-C. Cheng and T. G. Robertazzi, "Distributed computation with communication delay (distributed intelligent sensor networks)," *IEEE transactions on aerospace and electronic systems*, vol. 24, no. 6, pp. 700–712, 1988.

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

[37] Y.-C. Cheng and T. G. Robertazzi, "Distributed computation for a tree network with communication delays," *IEEE transactions on aerospace and electronic systems*, vol. 26, no. 3, pp. 511–516, 1990.

[38] S. Bataineh and T. G. Robertazzi, "Bus-oriented load sharing for a network of sensor driven processors," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 5, pp. 1202–1205, 1991.

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

[40] J. Błażewicz and M. Drozdowski, "The performance limits of a two dimensional network of load-sharing processors," *Foundations of Computing and Decision Sciences*, vol. 21, no. 1, pp. 3–15, 1996.

[41] V. Bharadwaj, D. Ghose, and V. Mani, "Multi-installment load distribution in tree networks with delays," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 2, pp. 555–567, 1995.

[42] Y. Yang and H. Casanova, "Umr: A multi-round algorithm for scheduling divisible workloads," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp. 9–pp, IEEE, 2003.

[43] O. Beaumont, A. Legrand, and Y. Robert, "Optimal algorithms for scheduling divisible workloads on heterogeneous systems," in *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pp. 14–pp, IEEE, 2003.

[44] J. Błażewicz and M. Drozdowski, "Distributed processing of divisible jobs with communication startup costs," *Discrete Applied Mathematics*, vol. 76, no. 1-3, pp. 21–41, 1997.

[45] A. L. Rosenberg, "Sharing partitionable workloads in heterogeneous nows: greedier is not better," in *cluster*, p. 124, IEEE, 2001.

[46] D. Ghose and V. Mani, "Distributed computation with communication delays: Asymptotic performance analysis," *Journal of Parallel and Distributed Computing*, vol. 23, no. 3, pp. 293–305, 1994.

[47] T. G. Robertazzi, "Processor equivalence for a linear daisy chain of load sharing processors," tech. rep., Stony Brook, NY: State University of New York at Stony Brook, College of Engineering., 1992.

[48] K. Ko and T. G. Robertazzi, "Scheduling in an environment of multiple job submissions," in *Proceedings of the 2002 Conference on Information Sciences and Systems*, Princeton NJ, 2002.

[49] H. M. Wong, D. Yu, B. Veeravalli, and T. G. Robertazzi, "Data intensive grid scheduling: multiple sources with capacity constraints.," in *Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems*, vol. 1, pp. 7–11, 2003.

[50] T. Lammie and G. Thomas, "A linear daisy chain with two divisible load sources," *Computer*, vol. 36, no. 5, pp. 63–68, 2003.

[51] T. G. Robertazzi and D. Yu, "Multi-source grid scheduling for divisible loads," in *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, pp. 188–191, 2006.

[52] D. A. L. Piriyakumar and C. S. R. Murthy, "Distributed computation for a hypercube network of sensor-driven processors with communication delays including setup time," *IEEE Transactions on Systems, Man, and Cybernetics-PART A: Systems and Humans*, vol. 28, no. 2, pp. 245–251, 1998.

[53] J. Hung and T. Robertazzi, "Scalable scheduling for clusters and grids using cut through switching," *International Journal of Computers and Applications*, vol. 26, no. 3, pp. 1–10, 2004.

[54] 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.

[55] 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.

[56] 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.

[57] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, vol. 23, no. 3, pp. 345–405, 1991.

[58] 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.

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

[60] 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.

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

[62] 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.

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

[64] K. Su, W. Chen, N. Lei, J. Zhang, K. Qian, and X. Gu, "Volume preserving mesh parameterization based on optimal mass transportation," *Computer-Aided Design*, vol. 82, pp. 42–56, 2017.

[65] B. Selçuk and A. Karci, "Connected cubic network graph," *Engineering science and technology, an international journal*, vol. 20, no. 3, pp. 934–943, 2017.

[66] S. M. Figueira and V. J. Reddi, "Topology-based hypercube structures for global communication in heterogeneous networks," in *European Conference on Parallel Processing*, pp. 994–1004, Springer, 2005.

[67] Z. Zhang and T. G. Robertazzi, "Scheduling divisible loads in gaussian, mesh and torus network of processors," *IEEE Transactions on Computers*, vol. 64, no. 11, pp. 3249–3264, 2015.