Optimal Data Intensive Flows for Network on Chip Mesh Networks

Junwei Zhang[1] Esther Arkin[1] and Thomas Robertazzi[2]

Abstract— **This thesis considers two problems. One problem is closed-form solutions for equivalence computation of divisible workload in a 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 Voronoi diagram algorithm (RVDA) 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 Voronoi Manhattan distance 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. In the third phase, we propose an algorithm to accelerating the bottleneck Voronoi cell's computation. Experimental evaluation through simulations demonstrates that a task can be finished in the same suboptimal time and yet save about 40% of processor resources. Also, it can improve about $10\% - 20\%$ speedup.**

I. INTRODUCTION

Networks on chips (NOC) represent the smallest networks that have been implemented to date [refs including my recent book Introduction to Computer Networking] . A popular choice for the interconnection network on such networks on chips is the rectangular mesh. It is straightforward to implement and is a natural choice for a planar chip layout.

Data to be processed can be inserted into the chip at one or more so-called "injection points", that is node(s) in the mesh that forward the data to other nodes. Beyond NOCs, injecting data into a parallel processor's interconnection network has been done for some time, notably in IBM's Bluegene machines [ref]. In this paper it is sought to determine, for a given set of injection points how, optimally or near-optimally, to assign load to different processors in a known timed pattern so as to process a load of data in a minimal amount of time (i.e. minimize makespan). In this paper we succeed in presenting an optimal technique for single injection points in homogeneous meshes that involves no more complexity than linear equation solution. For multiple injection points we present algorithms that produce near optimal solutions using Voroni diagrams. The methodology presented here can be applied to a variety of switching/scheduling protocols besides those directly covered in this paper.

II. APPROACH

A. Divisible Load Theory

Crucial to our success in the single and multiple injection point cases, is the use of divisible load scheduling theory [refs]. Developed over the past few decades, it assumes load is a continuous variable that can be arbitrarily partitioned among processors and links in a network. Use is made of the divisible load scheduling's optimality principle [ref], which say makespan is minimized when one forces all processors to stop at the same time (intuitively otherwise one could transfer load from busy to idle processors to achieve a better solution). This leads to a series of chained linear flow and processing equations that can be solved by linear equation techniques, often yielding recursive and even closed form solutions for quantities such as makespan and speedup.

B. Voroni Diagrams

In the context of multiple injection point models, this paper represents the first use of Voroni diagrams [refs] in conjunction with divisible load scheduling for a significant applied problem. [Give a paragraph of background on Voroni diagrams].

III. EQUIVALENCE COMPUTATION

A. 2*2 mesh network

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.

The load L is assigned on the corner processor P_0 Fig 1. The whole task is tackled by four processors P_0, P_1, P_2, P_3 together.

The processor P_0, P_1 and P_2 start to process its respective fraction at the same time. This includes P_1 and P_2 as they are relayed load in virtual cut-through mode at $t = 0$. The

[1]Junwei Zhang, Ph.D. candidate of Applied Mathematics and Statistics department, Stony Brook University, stony brook, NY 11794 `junwei.zhang@stonybrook.edu`

[1]Esther Arkin, Professor of Applied Mathematics and Statistics department, Stony Brook University, stony brook, NY 11794 `esther.arkin@stonybrook.edu`

[2]Thomas Robertazzi, Professor of Electrical and Computer Engineering department, Stony Brook University, stony brook, NY 11794 `Thomas.Robertazzi@stonybrook.edu`

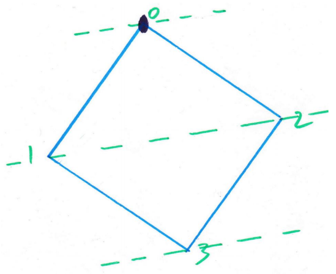

Fig. 1: The 2*2 mesh network and the root processor is P_0

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 [1], we obtain the timing diagram Fig 2.

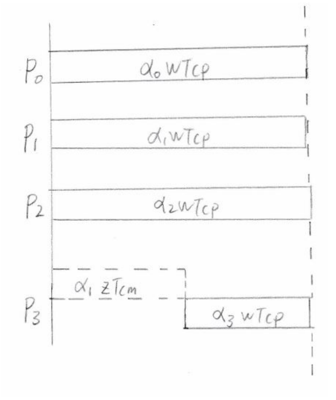

Fig. 2: The timing diagram for 2*2 mesh network and the root processor is P_0

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 [3].

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

$$\begin{cases} \alpha_0\omega T_{cp} = T_{f,m} & (1)\\ \alpha_1\omega T_{cp} = T_{f,m} & (2)\\ \alpha_2\omega T_{cp} = T_{f,m} & (3)\\ \alpha_1 zT_{cm} + \alpha_3\omega T_{cp} = T_{f,m} & (4)\\ \alpha_0+\alpha_1+\alpha_2+\alpha_3 = 1 & (5)\\ \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (6)\\ 0<\sigma<1 & (7)\\ 0<\alpha_0\le 1 & (8)\\ 0\le \alpha_1,\alpha_2,\alpha_3 < 1 & (9) \end{cases}$$

The group of equations are represented by the matrix form:

$$\begin{bmatrix} 1 & 2 & 1\\ 1 & -1 & 0\\ 0 & \sigma-1 & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0\\ \alpha_1\\ \alpha_3 \end{bmatrix} = \begin{bmatrix} 1\\ 0\\ 0 \end{bmatrix} \quad (10)$$

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:

$$\begin{cases} \sigma = \dfrac{zT_{cm}}{\omega T_{cp}} & (11)\\ \alpha_0 = \dfrac{1}{4-\sigma} & (12)\\ \alpha_1 = \dfrac{1}{4-\sigma} & (13)\\ \alpha_3 = \dfrac{1-\sigma}{4-\sigma} & (14) \end{cases}$$

The simulation result is illustrated:

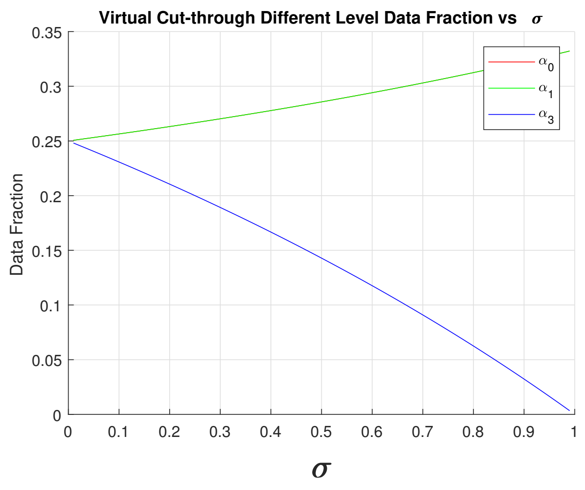

Fig. 3: 2*2 mesh network. α_0, α_1, α_2, α_3 value curve

In Fig 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

$$T_{f,n} = 1 * w_{eq} * T_{cp}$$

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

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

B. 2*n mesh network

The $2 * n$ Fig 4 homogeneous mesh network processes load L and L originates P_0.

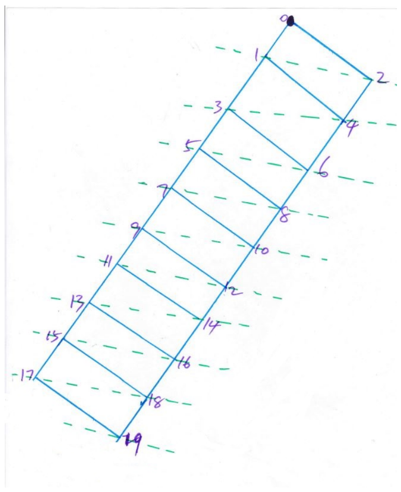

Fig. 4: 2*n (n = 10) mesh network and the workload happens on P_0

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.

The equations are presented as:

$$\begin{cases} \alpha_0 \omega T_{cp} = T_{f,m} & (15) \\ \alpha_1 \omega T_{cp} = T_{f,m} & (16) \\ \alpha_2 \omega T_{cp} = T_{f,m} & (17) \\ \alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} = T_{f,m} & (18) \\ \alpha_2 z T_{cm} + \alpha_4 \omega T_{cp} = T_{f,m} & (19) \\ (\alpha_1 + \alpha_3) z T_{cm} + \alpha_5 \omega T_{cp} = T_{f,m} & (20) \\ \vdots & (21) \\ (\alpha_1 \cdots + \alpha_{2\times n-1}) z T_{cm} + \alpha_{2\times n-1} \omega T_{cp} = T_{f,m} & (22) \\ \alpha_0 + \cdots + \alpha_{2\times n-1} = 1 & (23) \\ \sigma = \dfrac{z T_{cm}}{\omega T_{cp}} & (24) \\ 0 < \sigma < 1 & (25) \\ 0 < \alpha_0 \leq 1 & (26) \\ 0 \leq \quad \alpha_1 \quad \alpha_2 \quad \cdots \quad \alpha_{2\times n-1} < 1 & (27) \end{cases}$$

The flow matrix closed-form is shown:

$$\begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_5 \\ \vdots \\ \alpha_{2\times n-3} \\ \alpha_{2\times n-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \tag{28}$$

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

$$\begin{cases} \alpha_i = \left| \dfrac{\det A_i^\star}{\det A} \right| \end{cases} \tag{29}$$

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

$$A_0^\star = \begin{bmatrix} 1 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 0 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \\ 0 & \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{30}$$

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

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

The equivalence inverse processing speed :

$$T_{f,n} = 1 * w_{eq} * T_{cp}$$

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

Finally, the speedup is:

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

.

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

$$C = \begin{bmatrix} -1 & 0 & \cdots & 0 & 0 & 0 \\ \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{31}$$

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 $\hat{C}$

$$\hat{C} = \begin{bmatrix} -3 & -2 & \cdots & -2 & -2 & -1 \\ \sigma-1 & 1 & \cdots & 0 & 0 & 0 \\ \sigma-1 & \sigma & 1 & 0 & \cdots & 0 \\ \sigma-1 & \sigma & \sigma & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots \\ \sigma-1 & \sigma & \cdots & \sigma & \sigma & 1 \end{bmatrix} \tag{32}$$

, 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:

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

C. m*n mesh network

Considering a general $m * n$ mesh network, such as Fig 5.

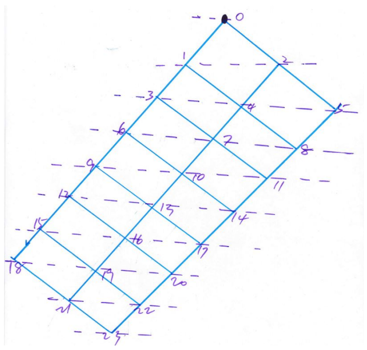

Fig. 5: 3*8 mesh network. The data injection position is P_0

Utilizing the previous methodology, we obtain the closed-form flow matrix equations for 5:

$$\begin{bmatrix} 1 & 2 & 3 & 3 & 3 & 3 & 3 & 3 & 2 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_3 \\ \alpha_6 \\ \alpha_9 \\ \alpha_{12} \\ \alpha_{15} \\ \alpha_{18} \\ \alpha_{21} \\ \alpha_{23} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{33}$$

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

$$T_{f,n} = 1 * w_{eq} * T_{cp}$$

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

so the speedup is:

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

.

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.

D. Sensitivity Analysis

From Chapter 2, we know the speedup is :

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

.

1) Data Injection On The Corner Processor: The simulation result of sensitivity analysis of $2 * n$ mesh network 4 is as follows:

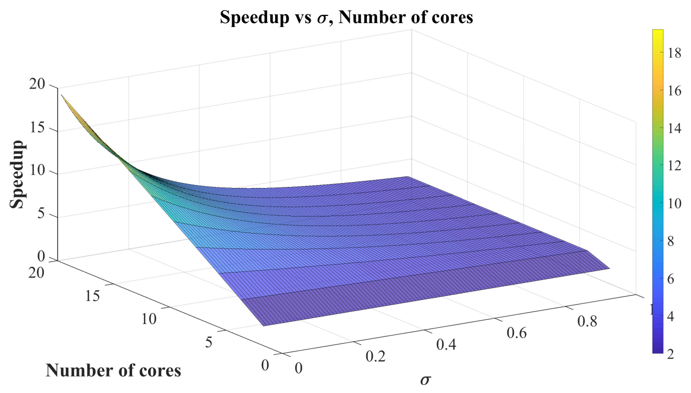

Fig. 6: Sensitivity analysis result of 2*n mesh network (where $n = 1 \cdots 10$)

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).

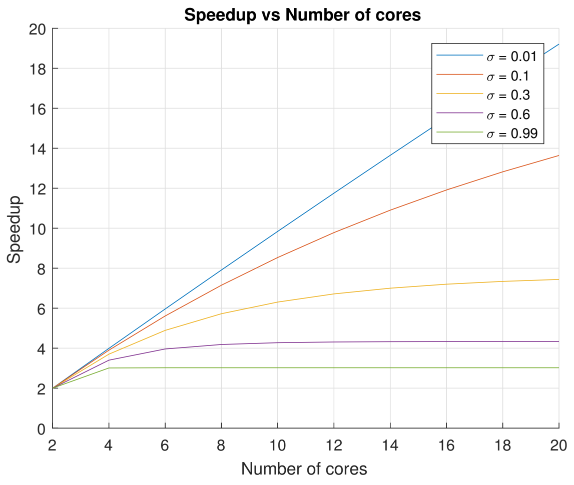

Fig. 7: Speedup curves of $2{*}n$ mesh network (where $n = 1 \cdots 10$)

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.

2) Data Injection On The Boundary Processor: For a $3{*}n$ mesh network Fig 5, the data injection happens on boundary processor P_2.

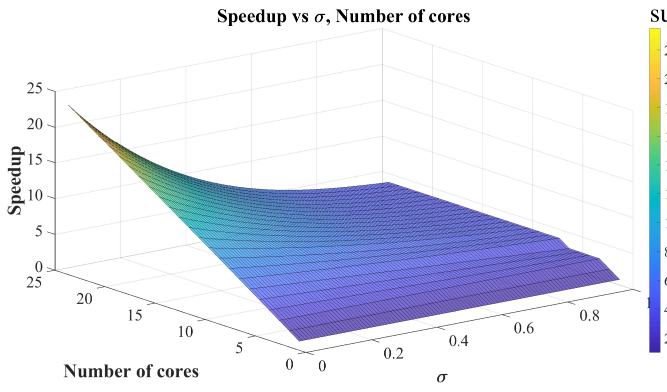

Fig. 8: Sensitivity analysis result of $3{*}n$ mesh network and the injection position on boundary processor P_2 (where $n = 1 \cdots 8$)

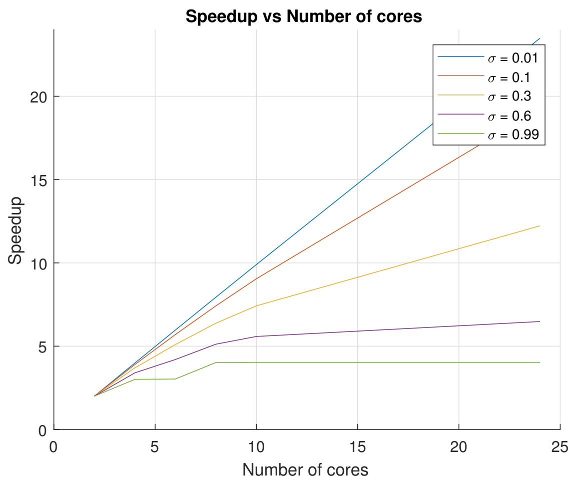

Fig. 9: Speedup curve of $3{*}n$ mesh network (where $n = 1 \cdots 8$)

Fig 8 and Fig 9 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.

IV. Multi-source Assignment Heuristic Algorithm

A. Equivalence Computation Algorithm

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

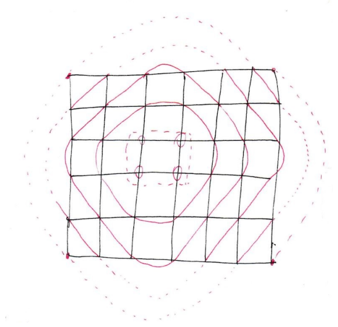

Fig. 10: Data injection consists of a connected subgraph of G

Connected subgraph 10 and connected subgraph 11 illustrate two examples that the data injections consist of connected subgraphs of G.

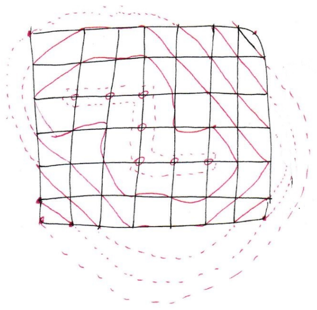

Fig. 11: Data injection consists of a connected subgraph of G

Our objective is to propose a general 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.

$$\min \quad T_{f,n} \tag{34}$$
$$\text{s.t.} \quad \sum_{i \in 0 \cdots n} \alpha_i = 1, \tag{35}$$
$$\alpha_i \geq 0, \tag{36}$$

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 [2].
Calculate $m * n$ processor's $D_i$.
Obtain the flow matrix $A$.
Calculate $m * n$ processors data fraction $\alpha_i$.
```

In term of the time complexity :

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

For example, connected subgraph 10's ***flow matrix*** is :

$$\begin{bmatrix} 4 & 8 & 12 & 10 & 6 & 2 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & \sigma-1 & 1 & 0 & 0 & 0 \\ 0 & \sigma-1 & \sigma & 1 & 0 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & 1 & 0 \\ 0 & \sigma-1 & \sigma & \sigma & \sigma & 1 \end{bmatrix} \times \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{37}$$

The simulation result illustrates as follows:

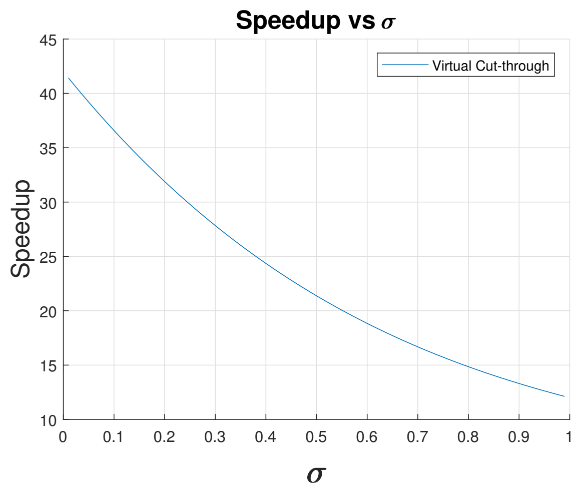

Fig. 12: Connected subgraph Fig 10 speedup vs σ

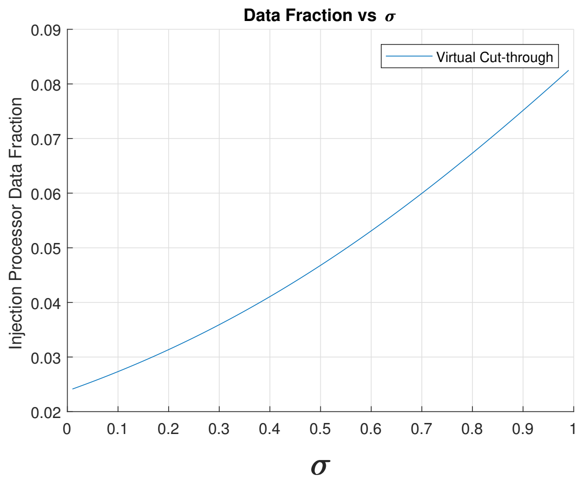

Fig. 13: Connected subgraph 10's processor data fraction vs σ

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.

B. Intuitive Voronoi Division Algorithm

Our objective is to propose an intuitive algorithm to minimize the makespan and give quantitative model analysis utilizing the flow matrix. Also, in each cell, the constraint comes from the divisible load theory linear equations.

$$\min \quad T_{f,n} \tag{38}$$
$$\text{s.t.} \quad \sum_{i \in 0 \cdots n} \alpha_i = 1, \tag{39}$$
$$\alpha_i \geq 0, \tag{40}$$

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 14 shows 10 Voronoi cells division.

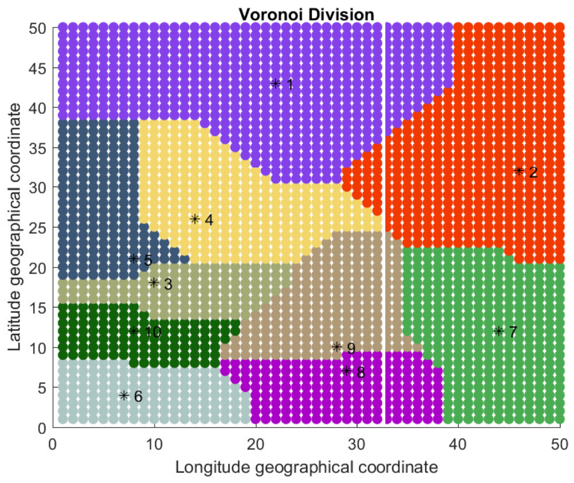

Fig. 14: 10 Manhattan distance Voronoi Cells

Voronoi diagram division speedup curves 15 specifies the speedup of each cells.

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

Our objective is to propose a heuristic algorithm to minimize the makespan and give quantitative model analysis utilizing the flow matrix. In each cell, the constraint comes from the divisible load theory linear equations.

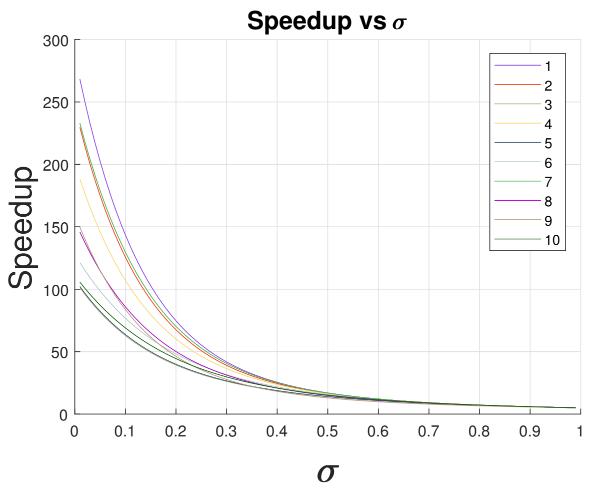

Fig. 15: 10 Manhattan distance Voronoi Cells speedup curves

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

$$\min \quad T_{f,n} \tag{41}$$
$$\text{s.t.} \quad \sum_{i \in 0 \cdots n} \alpha_i = 1, \tag{42}$$
$$\alpha_i \geq 0, \tag{43}$$

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.
```

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 $O(k * \max(m, n)^3)$.

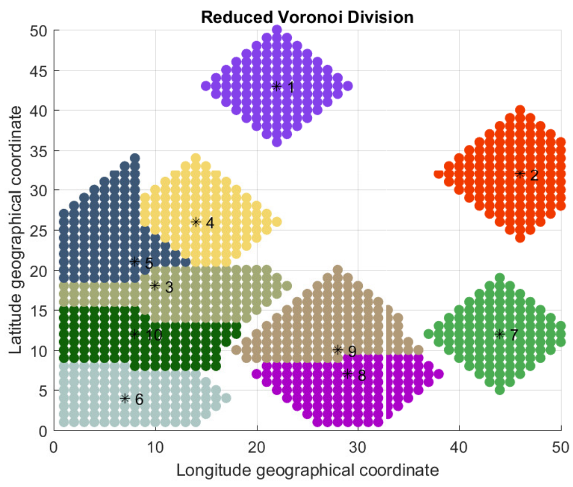

Fig. 16: 10 reduced Voronoi cells

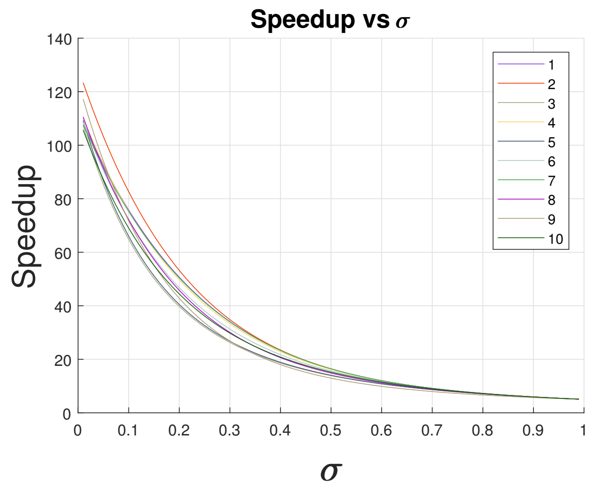

Fig. 17: 10 reduced Voronoi cells' speedup curves

Reduced Voronoi cells Fig 16 and reduced Voronoi speedup curves Fig 17 show this algorithm obtains the same running time yet saves about 30% processors.

- Voronoi speedup curves Fig 15 show that $\sigma < 0.2$, the ratio $\max(\frac{maxspeedup}{minspeedup}) = \frac{271}{107} \approx 2.533$
- Reduced Voronoi speedup curves Fig 17 show that $\sigma < 0.2$, the ratio is $\max(\frac{maxspeedup}{minspeedup}) = \frac{122}{107} = 1.14$.

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 18.

From the average saved processors ratio Fig 18, it shows the average percentage of saved processor is about 40%.

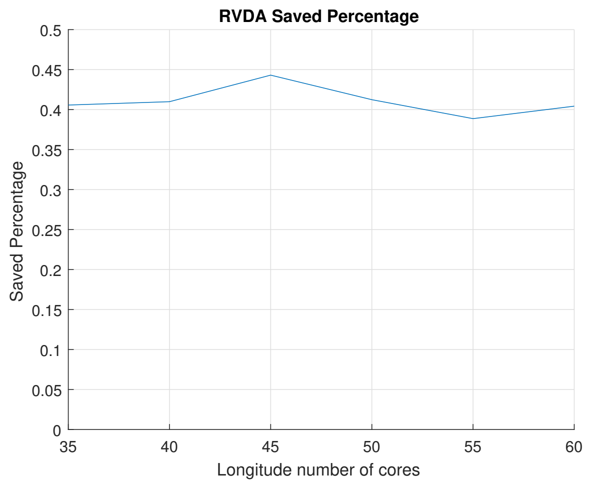

Fig. 18: Reduced Voronoi division algorithm average saved processors' percentage

C. *Reduced Voronoi Division Algorithm*

V. Conclusions

Appendix

Acknowledgment

References

[1] Veeravalli Bharadwaj, Debasish Ghose, and Thomas G Robertazzi. "Divisible load theory: A new paradigm for load scheduling in distributed systems". In: *Cluster Computing* 6.1 (2003), pp. 7–17.

[2] Thomas G Robertazzi. "Processor equivalence for daisy chain load sharing processors". In: *IEEE Transactions on Aerospace and Electronic Systems* 29.4 (1993), pp. 1216–1221.

[3] Jeeho Sohn and Thomas 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.