OPTIMIZING DATA INTENSIVE FLOWS FOR NETWORKS ON CHIPS

Junwei Zhang
Uber Technologies Inc
1191 2nd Ave 1200, Seattle, WA 98101
junwei.zhang@stonybrook.edu

Yang Liu
Uber Technologies Inc
555 Market Street, San Francisco, CA, USA
yangliu89415@gmail.com

Li Shi
Snap Inc
Ocean Front Walk, Venice, CA
lishi.pub@gmail.com

Thomas G. Robertazzi
IEEE fellow
Department of Electrical and Computer Engineering
Stony Brook University
100 Nicolls Rd, Stony Brook, NY 11794
thomas.robertazzi@stonybrook.edu

Abstract

A novel framework is proposed to find efficient data intensive flow distributions on Networks on Chip (NoC). Voronoi diagram techniques are used to divide a NoC array of homogeneous processors and links into clusters. A new mathematical tool, named the flow matrix, is proposed to find the optimal flow distribution for individual clusters. Individual flow distributions on clusters are reconciled to be more evenly distributed. This leads to an efficient makespan and a significant savings in the number of cores actually used. The approach here is described in terms of a mesh interconnection but is suitable for other interconnection topologies.

Divisible Load Theory, Voronoi Diagram, Multi-source, Network on Chip (NOC), Mesh, Data Intensive Load, Load Injection
1 INTRODUCTION

The mapping of tasks on a network of processors heavily impacts the performance of parallel applications running on such a multi-processor system. A crucial task scheduling problem, utilizing the maximum benefits of parallel computing system, is considered. Scheduling multi-source divisible loads in a Network on Chip (NoC) is a challenging task as different sources should cooperate and share their computing power with others to balance their workloads and in a manner of minimizing total computational time (makespan).

In this paper a new approach is proposed to find efficient data flows in Networks on Chip (NoC). Voroni diagram methods are used to segment a NoC homogeneous array of processors and links into clusters of processors and links. The optimal flow distribution for individual clusters operating under a given scheduling policy and set of assumptions is found using a new linear mathematical tool called the flow matrix. Individual flow distributions for each cluster are refined to make a relatively even distribution of flows across the clusters. This minimizes the negative influence of “bottleneck” clusters. This heuristic leads to an efficient value of makepsan and a significant savings in the actual number of processors used (leading to significant chip number savings: about 30% in this study). This overall approach is described for a mesh interconnection of processors but is readily extended to other interconnection topologies (a toroidal network example is presented). Related work is described in the rest of this section. Section 2 presents definitions and assumptions. Section 3 discusses three models in order of increasing generality and together with examples. Section 4 examines thermal management issues. Section 5 discusses some load sharing aspects. Section 6 is the conclusion.

1.1 Related Work

1.1.1 Networks on Chip (NoC)

As the number of cores increase on a single chip, conventional bus technology can no longer satisfy today’s requirements of throughput and latency. Confronted with the pressing demand for extremely high bandwidth and low power consumption, network-on-chip (NoC) architectures are proposed as a new paradigm to interconnect a large number of processing cores at the chip level. In today’s and future electronic technology, Systems-on-chip (SoC) technology has become important and even essential (Robertazzi 2017), and has emerged as a communication backbone to enable a high degree of integration in multi-core Systems-on-chip (SoC) (Benini and De Micheli 2002) (Ganguly, Chang, Deb, Pande, Belzer, and Teuscher 2011) (Tatas, Siozios, Soudris, and Jantsch 2016).

Networks on chip (NoC) represents the smallest networks that have been implemented to date (Robertazzi 2017). A popular choice for the interconnection network on such networks on chip is the rectangular mesh (Fig. 1). 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 a node(s) in the mesh that forwards the data to other nodes. Beyond NoCs, injecting data into a parallel processor’s interconnection network has been done for some time, for instance in IBM’s Bluegene machines (Krevat, Castaños, and Moreira 2002). In this paper, it is sought to determine, for multi-source injection points on a homogeneous rectangular mesh, how to efficiently assign a load to different processors/links in a known timed pattern so as to process a load of data in a minimal amount of time (i.e. minimizing makespan). In this paper, we succeed in presenting an efficient technique for multi-source injection in homogeneous meshes that involves no more complexity than linear equation solution. The methodology presented here can be applied to a variety of interconnection networks and switching/scheduling protocols besides those directly covered in this paper.
1.1.2 Divisible Load Theory

There are massive divisible load scheduling theory’s applications, such as, (Bharadwaj, Ghose, and Robertazzi 2003) (Bharadwaj, Ghose, Mani, and Robertazzi 1996) (Drozdowski 2009) (Casanova, Legrand, and Robert 2008). Developed over the past few decades, divisible load theory assumes that load is a continuous variable that can be arbitrarily partitioned among processors and links in a network. We utilize the divisible load scheduling’s optimality principle (Bharadwaj, Ghose, Mani, and Robertazzi 1996) - 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 solutions for optimal load fractions and even closed form solutions for quantities such as makespan and speedup.

1.1.3 Multi-source Assignment

Wong, Yu, Veeravalli and Robertazzi (Wong, Yu, Veeravalli, and Robertazzi 2003) examined two sources grid scheduling with memory capacity constraints. Marchal, Yang, Casanova, and Robert (Marchal, Yang, Casanova, and Robert 2005) studied the use of linear programming to maximize throughput for large grids with multiple loads/sources. Lammie and Robertazzi (Lammie and Thomas 2005) 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 (Robertazzi and Yu 2006).

1.1.4 Voronoi Diagram

Voronoi diagrams are used in this paper for multi-sources assignment. Voronoi diagrams are induced by a set of points (called sites or seeds): subdivision of the plane where the faces correspond to the regions where one site is closest. Voronoi diagrams are extensively utilized in network optimization application (Okabe, Boots, Sugihara, and Chiu 2009) (Stojmenovic, Ruhil, and Lobiyal 2006) (Meguerdichian, Koushanfar, Qu, and Potkonjak 2001).

1.1.5 Processor Equivalence

The concept of processor equivalence was first proposed in (Robertazzi 1993). A linear daisy chain of processors where processor load is divisible and shared among the processors is examined. It is shown that two or more processors can be collapsed into a single equivalent processor. We propose a flow matrix closed-form equation to present the equivalence for single source assignment, 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. These works (Robertazzi 1993) (Liu, Zhao, and Li 2007) inspire us to adopt the processor equivalence model to tackle multi-source workload scheduling problems.

1.2 Our Contribution

1. With the objective of minimizing the makespan, we propose a novel algorithm framework Reduced Manhattan Distance Voronoi Diagram Algorithm (RMDVDA) to address the multi-source problem. This framework can be directly extended from typical mesh and torus networks to a general graph design of NoC patterns.
2. We propose a novel mathematics tool, the flow matrix (Zhang, Liu, Li, and Robertazzi 2018) (Zhang 2018), to calculate the data fraction allocated to each processor in each cluster and the speedup of each cluster.

3. Via the extensive random simulation experiments, we demonstrate the method reduces the computational core number by 30% while achieving the same makespan.

2 DEFINITIONS AND ASSUMPTION

A mesh network is illustrated in Fig. 1. The rainbow color bar means the Manhattan distance to data injector (0, 0). For example, the (0,0) to (0,0) Manhattan distance is 0 and the color is red and the furthest point coordinate is (50, 50) and the color is blue.

![Manhattan Distance to Data Injections](image)

Figure 1: A m*n mesh network(m = 50, n = 50). Data injector at (0,0)

The following assumptions are used throughout the paper:

- For simplicity, return communication is not considered.
- The communication delay which is proportional to data size is considered.
- The time delay of computation is proportional to the data size.
- The network environment is homogeneous, that is, all the processors have the same computation capacity and the link speed between any two connected cores is identical.
- The load of data injected from each data source is identical in size.
2.1 Notations

The following notations and definitions are utilized:

- \( m \): The number of the x-coordinate cores.
- \( n \): The number of the y-coordinate cores.
- \( k \): The number of data injection nodes.
- \( r \): The rank of flow matrix (Zhang, Liu, Li, and Robertazzi 2018).
- \( D_i \): The minimum number of hops from the processor \( P_i \) to the data injectors.
- \( \alpha_0 \): The load fraction assigned to the root processor.
- \( \alpha_i \): The load fraction assigned to the \( i \)th processor.
- \( \hat{\alpha}_i \): The load fraction assigned to the \( i \)th Voronoi cell.
- \( \hat{\alpha}_{i,l} \): The data fraction of each processor on the \( l \)th layer of \( i \)th Voronoi cell.
- \( \omega \): The inverse computing speed of a processor.
- \( \omega_{eq} \): The inverse computing speed on an equivalent node collapsed from a cluster of processors.
- \( L \): The number of layers in a cell.
- \( z \): The inverse link speed of a link.
- \( T_{cp} \): Computing intensity constant. The entire load is processed in time \( \omega \times T_{cp} \) seconds on the \( i \)th processor.
- \( T_{cm} \): Communication intensity constant. The entire load is transmitted in time \( z \times T_{cm} \) seconds over the \( i \)th link.
- \( T_f \): The makespan for the entire divisible load solved on the root processor.
- \( \hat{T}_f \): The finish time of the whole processor network. Here \( \hat{T}_f \) is equal to \( \omega_{eq}T_{cp} \).
- \( \hat{T}_{f,i} \): The finish time of \( k \)th cell.
- \( T_{f,i} \): The finish time for the \( i \)th processor, \( i \in 0 \cdots (m+n-1) \).
- \( \sigma = \frac{T_{cm}}{T_{cp}} \): The ratio between the communication speed to the computation speed, \( 0 < \sigma < 1 \) (Bharadwaj, Ghose, and Robertazzi 2003). (Hung and Robertazzi 2004).
- \( S_{p_i} \): The speedup of \( i \)th Voronoi cell \( i \in 0 \cdots (k-1) \).
- Speedup = \( \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\omega_{eq} T_{cp}} = \frac{1}{\hat{\alpha}_0} \)

2.2 Flow Matrix

The flow matrix is the matrix form of a group of divisible load theory equations, where each equation describes the relationship between the computation time, transmission time and finish time. It can be obtained via (Zhang, Liu, Li, and Robertazzi 2018) (Zhang 2018). The flow matrix concept is illustrated in terms of a simple example. The load \( L \) is assigned to the corner processor \( P_0 \), as shown in Figure 2. The whole load is processed by four processors \( P_0, P_1, P_2, P_3 \) together. The processor \( P_0 \), \( P_1 \) and \( P_2 \) start to process their respective load fraction at the same time. This includes \( P_1 \) and \( P_2 \) as they are relayed load in virtual cut-through mode at \( t = 0 \). Because we assume a homogeneous network (in processing speed and communication speed), \( \alpha_1 = \alpha_2 \) and \( P_1 \) and \( P_2 \) stop processing at the same time. The processor \( P_3 \) starts to compute when the \( \alpha_1 \) and \( \alpha_2 \) complete transmission. That is, the link \( 0-1 \) and \( 0-2 \) are occupied transmitting load to processor 1 and 2, respectively and only transmit to 3 when that is finished.

According to the divisible load theory (Bharadwaj, Ghose, and Robertazzi 2003), we obtain the Gantt-like timing diagram Figure 3. Let’s assume that all processors stop computing at the same time in order to minimize the makespan (Sohn and Robertazzi 1996). We obtain a group of linear equations to find the fraction workload assigned to each processor \( \alpha_i \):
Figure 2: The 2*2 mesh network and the root processor at $P_0(0,0)$

\[
\begin{align*}
\alpha_0 \omega T_{cp} &= T_{f,m} \\
\alpha_1 \omega T_{cp} &= T_{f,m} \\
\alpha_2 \omega T_{cp} &= T_{f,m} \\
\alpha_1 z T_{cm} + \alpha_3 \omega T_{cp} &= T_{f,m} \\
\alpha_0 + \alpha_1 + \alpha_2 + \alpha_3 &= 1 \\
\sigma &= \frac{z T_{cm}}{\omega T_{cp}} \\
0 < \sigma &< 1 \\
0 < \alpha_0 &\leq 1 \\
0 \leq \alpha_1, \alpha_2, \alpha_3 &< 1
\end{align*}
\]

The group of equations are represented by the matrix form:

\[
\begin{bmatrix}
1 & 2 & 1 \\
1 & -1 & 0 \\
0 & \sigma -1 & 1
\end{bmatrix}
\begin{bmatrix}
\alpha_0 \\
\alpha_1 \\
\alpha_3
\end{bmatrix}
= \begin{bmatrix}
1 \\
0 \\
0
\end{bmatrix}
\]
The matrix is represented as $A \times \alpha = b$. $A$ is named as the flow matrix. Because of the symmetry $\alpha_1 = \alpha_2$, the $\alpha_2$ is not listed in the matrix equations. The first row in flow matrix describes the number of cores on each hop with respect to node 0.

- There is 1 core with 0 hop distance.
- There are 2 cores with 1 hop distance.
- There is 1 core with 2 hop distance.

Finally, the explicit solution is:

$$
\begin{align*}
\sigma &= \frac{z_{cm}}{\omega T_{cp}} \\
\alpha_0 &= \frac{1}{4 - \sigma} \\
\alpha_1 &= \frac{1}{4 - \sigma} \\
\alpha_3 &= \frac{1 - \sigma}{4 - \sigma}
\end{align*}
$$

The performance result is illustrated:

In Fig 4, the three processors $P_0, P_1, P_2$ have the same data fraction workload, so the curve of $\alpha_0, \alpha_1$ and $\alpha_2$ coincide. The figure illustrates that as $\sigma$ grows, the curve of $\alpha_3$ drops. In other words, as the communication speed 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 don’t distribute it to other processors. Thus for slow communication $\alpha_0 = \alpha_1 = \alpha_2 = \frac{1}{3}$.

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

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

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

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

For a fast communication ($\sigma \approx 0$), the speedup is 4 and the data fraction is about 0.25. The flow matrix concept is applicable to larger networks. The first row holds the number of nodes $i$ hops from the source in the node $(0, i)$. The rest of the matrix has a practical structure (Zhang 2018). Since $A \times \alpha = b$, one solves the linear flow matrix equation for the fraction of load assigned to each node processor $\alpha_i$. See the next section and (Zhang, Liu, Li, and Robertazzi 2018) (Zhang 2018) for more examples.

3 MULTI-SOURCE UNIFORM DATA FRACTION

The single source assignment problem (Zhang, Liu, Li, and Robertazzi 2018) has been studied. In this section, we focus on the general multi-source assignment problem (Jia, Veeravalli, and Weissman 2010) (Liu, Zhao, and Li 2007). For each processor, we focus on the processors’ geographical location $P_i$, and the data fraction $\alpha_i$ assigned. In addition, we assume that the data fractions are distributed uniformly. For example, in the case of a one unit workload and K data injection nodes, each data injection node is assigned $\frac{1}{K}$ of the workload. Regarding the data injection position relationship, we consider three different models:

1. Data injection nodes consist of a connected induced subgraph $G_L$ of $G$.
2. Data injection nodes don’t connect with each other.
3. Some data injection nodes consist of some connected induced subgraphs and some are individual injection points.
3.1 Model I

In the scenario that the data injection positions consist of a connected induced subgraph of $G$, we use $G_L$ to present it.

Table 1 illustrates two examples where the data injectors consist of connected induced subgraph of $G$. We propose a general algorithm framework to minimize the makespan and present a quantitative model analysis utilizing the flow matrix (Zhang, Liu, Li, and Robertazzi 2018). This algorithm is an extension of the processor equivalence’s application in daisy chain (Robertazzi 1993), which is named as **Equivalence Processor Scheduling Algorithm (EPSA)**.

**Algorithm 1** Equivalence Processor Scheduling Algorithm (EPSA)

**Input:** $k$ data injection positions  

**Output:** $m \times n$ processor data fractions $\alpha_i$

1. Collapse the data injection processors into one "big" equivalent processor (Robertazzi 1993).
2. Calculate $m \times n$ processor’s $D_i$.
3. Obtain the flow matrix $A$ (Zhang, Liu, Li, and Robertazzi 2018).
4. Calculate the determinant of the flow matrix.
5. Calculate $m \times n$ processors’ data fraction $\alpha_i$.

In terms of the time complexity: according to the proof of (Zhang, Liu, Li, and Robertazzi 2018) (Zhang 2018), we know the speedup is:

$$\hat{T}_f = 1 \ast w_{eq} \ast T_{cp}$$

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

$$\text{Speedup} = \frac{T_f}{\hat{T}_f} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0} = |\det A|$$
In addition, the time complexity of calculating the determinant is $O(r^3)$ with Gaussian elimination or LU decomposition. The $r$ is the rank of matrix. The time complexity of calculating the flow matrix $A$ is $O(m*n)$. And the total time complexity is $O(r^3 + m*n)$. Note that a critical step in the EPSA is to obtain the flow matrix given a graph. For example, connected subgraph I in Table 1 (1,1)'s flow matrix is:

$$\begin{bmatrix}
4 & 8 & 12 & 8 & 4 \\
1 & -1 & 0 & 0 & 0 \\
0 & \sigma - 1 & 1 & 0 & 0 \\
0 & \sigma - 1 & \sigma & 1 & 0 \\
0 & \sigma - 1 & \sigma & \sigma & 1
\end{bmatrix} \times \begin{bmatrix}
\alpha_0 \\
\alpha_1 \\
\alpha_2 \\
\alpha_3 \\
\alpha_4
\end{bmatrix} = \begin{bmatrix}
1 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}$$ (15)

The first row in flow matrix describes the number of cores on each $D_i$.

- There are 4 cores with 0 hop distance from the cluster.
- There are 8 cores with 1 hop distance from the cluster.
- There are 12 cores with 2 hop distance, and so on from the cluster.

Since each layer processor has the same data fraction, the size of the rows in the network yields the different data fractions.

We study the performance of the EPSA algorithm numerically. As shown in Table 2, the best performance occurs for value $\sigma \leq 0.05$ ($\sigma$ is the ratio between the communication speed to the computation speed), for fast communication, which hits about 36 times speedup as all 36 processors in the network are engaged in processing networks. There are 12 processors at the initial stage, (four nodes are hop 0 and eight nodes are on hop 1). When $\sigma \approx 1$ (slow communication), in which, communication time equals computation time, the network achieves about 12 times speedup performance as only those 12 processors do processing. Note the $\sigma \approx 1$ scenario represents the slowest feasible communication. When the $\sigma > 1$, any distribution of the load yields worse performance than simply processing the load on injection nodes. In the Table 2, the $\alpha$ represents the task fraction processed by a 0 hop distance core. When the $\sigma \leq 0.05$, the fraction is about $\frac{1}{36}$ and when the $\sigma \approx 1$, the fraction is about $\frac{1}{12}$.
3.2 Model II

In the scenario that the data injection nodes don’t form a whole connected subgraph of \( G \), our objective is to propose a general heuristic algorithm framework to minimize the makespan. The seeds are induced subgraphs as well as “isolated” single injection nodes. For each Voronoi seed(s) there is a corresponding region consisting of all points closer to that seed(s) than any other. For the multi data injection nodes scenario, the intuitive algorithm is ‘divide and conquer’, which extends (Jia, Veeravalli, and Weissman 2010)’s graph partitioning algorithm to the Manhattan distance Voronoi diagram.

3.2.1 Manhattan Distance Voronoi Diagram Algorithm

Similar to the EPSA algorithm, the objective here is to find proper data fractions which minimize the makespan. This problem can be formulated into a linear programming problem whose definition is given in the following.

\[
\begin{align*}
\min & \quad \hat{T}_f \\
\text{s.t.} & \quad \sum_{j=0}^{k-1} \sum_{i=0}^{S_j-1} \hat{\alpha}_{i,j} = 1 \\
& \quad 1 \leq \hat{\alpha}_{i,j} \geq 0, \\
& \quad T_{f,j} = T_{f,0} \\
& \quad \sum_{n=1}^{j-1} \hat{\alpha}_{n,i} T_{cm} + \hat{\alpha}_{1,j-1} w T_{cp} = \hat{T}_{f,k}, 1 \leq j \leq L
\end{align*}
\] (16)

- Equation 16 is the objective function, minimizing the makespan. \( \hat{T}_f \) is the finish time of the whole processor network. Here \( \hat{T}_f \) is equal to \( w_{eq} T_{cp} \).
- Equation 17 means that all total fraction of all processors is unit 1. Here \( \hat{\alpha}_{i,j} \) is the load fraction assigned to \( i \)th processor of \( j \)th cell.
- Equation 18 means that in each cell, each processor’s data fraction is nonnegative.
- Equation 19 means that in each cell, each processor has the same finish time.
- Equation 20 means that in each cell, the \( j \)th layer’s finish time consists of previous \( (j-1) \) layers’ transmitting time and \( j \)th computation time.

We further propose an algorithm, named **Manhattan Distance Voronoi Diagram Algorithm** to solve this problem:

**Algorithm 2** Manhattan Distance Voronoi Diagram Algorithm (MDVDA)

**Input:** \( k \) data injection positions

**Output:** \( m \times n \) processor data fractions \( \alpha_j \).

Collapse the connected data injectors into some “big” equivalent processors and the number of clusters is \( k \).

Calculate \( k \) Voronoi cells with Manhattan distance. The \( k \) injectors are the Voronoi “seeds”.

Calculate the flow matrix \( A_i \).

Calculate the determinant of flow matrix \( A_i \).

Calculate \( m \times n \) processors’ data fraction \( \alpha_j \).
The time complexity is the same as with EPSA. Manhattan distance Voronoi diagram division Table 3 shows 10 and 8 Voronoi cells division scenario. The first row shows the 10 individual data injectors mesh division result and the distance heat-map which shows the core processor to its nearest data injector’s Manhattan distance. In the second row, we consider another more general scenario. The data injectors are not only individual cores but also induced sub-graphs injectors. Fig (2, 2) shows the Manhattan distance to its nearest injectors. Each division will handle the corresponding task from the data injector.

3.3 Model III

According to a general situation, if there are some nodes consisting of introduced sub-graphs and other nodes are individual seeds, our objective to minimize the makespan and save processors. After investigation, we find that the makespan depends on the bottleneck cell makespan. In other words, if other cells contain more processors than the bottleneck cell, it does not help to minimize the makespan. We propose an algorithm named Reduced Manhattan Distance Voronoi Diagram Algorithm (RMDVDA) to tackle this situation.

**Algorithm 3 Reduced Manhattan Distance Voronoi Diagram Algorithm (RMDVDA)**

**Input:** $k$ data injection positions

**Output:** $m \times n$ processor data fractions $\alpha_i$.

- Collapse the connected data injection processors into some “big” equivalent processors and the number of cluster is k.
- Calculate $k$ constrained Manhattan distance Voronoi cells (Chin and Wang 1998).
- Calculate the constrained Voronoi cells’ radius $R_i$.
- Calculate the constrained Voronoi cells’ flow matrixes $A_i$.
- Calculate the constrained Voronoi cells’ speedup $Sp_i$.
- 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 $\hat{A}_i$.
- Calculate the determinant of flow matrix $\hat{A}_i$.
- Calculate $m \times n$ processors’ data fraction $\alpha_i$.

In terms of the time complexity, it’s the same with EPSA.

- Voronoi speedup curves in Table 4(3, 1) shows that for $\sigma < 0.1$, the ratio $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{293}{133} \approx 2.2$.
- Reduced Voronoi speedup curves in Table 4(4, 1) show that for $\sigma < 0.1$, the ratio is $\max(\frac{Sp_{max}}{Sp_{min}}) = \frac{153}{133} \approx 1.13$.

To explain, the whole network makespan depends on the max makespan of each core division. So if the ratio of $\max(\frac{Sp_{max}}{Sp_{min}})$ is more balanced, in other words, the curves are closer to each other, it is better, for example, Table 4 (3, 1), (4, 1) and (3, 2), (4, 2). The divisions in Table 4(1, 1) and Table 3(1, 1) display that 10 cells’ equivalence computation is more balanced than the initial setting, and the whole cluster finishes processing load within the same time but with only $1462/2500 \approx 58.5\%$ of the processors.

Table 5 shows simulation experiments. A boxplot is a standardized way of displaying the dataset based on a five-number summary: the minimum, the maximum, the sample median, and the first and third quartile.
| Nr. | Voronoi Division | Distance Heatmap |
|-----|------------------|------------------|
|     | ![Voronoi Division](image1) | ![Distance Heatmap](image2) |
|     | ![Voronoi Division](image3) | ![Distance Heatmap](image4) |

Table 3: Model 2: Voronoi division and distance heatmap
Table 4: Model III Voronoi division and reduced Manhattan distance heatmap
Table 5 (1, 1) in Table 5 shows the initial Voronoi division. The regular mesh grid edge core number is 50 and there are 10 random data injectors. The workload fraction is uniform, so it is \( \frac{1}{10} \).

Table 5 (1, 2) describes after 2000 rounds of simulation in each \( \sigma \), as the number of data injectors grows from 10 to 20,

- In \( \sigma = 0.1 \) scenario, our algorithm is robust and it saves about 40% cores in experiments.
- In the \( \sigma > 0.5 \), it saves over 80% of the cores. It means the algorithm keeps more data to process in local area instead of broadcasting it to further cores.

Considering the edge core number is 50 and \( \sigma = 0.1 \) scenario, we adjust the data injector number from 11 to 20. Table 5 (2, 1) boxplot describes the simulation result. For example, the data injector number is 11 and we randomly place the 11 data injectors to the mesh 1000 times and summarize the simulation result in the boxplot. The minimum is about 51%, the first quartile is over 60%, the median is 75%, the third quartile is 77% and the maximum is 90%. The first quartile is over 50%, which means in 1000 times simulation experiments, the 75% tests achieve 50% percentage saved performance. In sum, the second quartile of different injectors scenario is over 50%, so the RMDVDA is robust.

Considering the edge core number is 50 and the data injector number is 10. Table 5 (2, 2) shows the simulation result as the \( 0 < \sigma < 1 \). For example, in \( \sigma = 0.1 \), after 1000 experiments, the boxplot shows the minimum percentage saved is 29%, the first quartile is 57%, the median is 66%, the third quartile is 76% and the maximum is 95%. The circles are the outliers. In sum, the median of each boxplot increases as the \( \sigma \) grows and the value is over 60%, so the RMDVDA is robust.

Considering the data injectors number is 10 and the \( \sigma = 0.1 \), Table 5 (3, 1) shows the simulation summary as the mesh grid edge from 51 to 60. For example, the edge core number is 53, we randomly place 10 data injectors in the mesh 1000 times and the boxplot shows the minimum percentage saved is 40%, the first quartile is 59%, the median is 62%, the third quartile is 70% and the maximum is 79%. In sum, the median of each setting is over 50%, so the RMDVDA is robust.

Table 5 (3, 2) shows that RMDVDA extend to irregular graph successfully, which handles the graph with holes.

We also can extend our method to torus mesh networking (Table 6).
Table 5: Simulation statistics result: Voronoi division, percentage saved vs $\sigma$, percentage saved vs number of data injectors, percentage saved vs $\sigma$
| Nr. | Original | RMDVDA |
|-----|----------|--------|
| 1   | ![Original Manhattan Distance Division](image1) | ![RMDVDA Manhattan Distance Division](image2) |
| 2   | ![Original Manhattan Distance to Data Injections](image3) | ![RMDVDA Manhattan Distance to Data Injections](image4) |
| 3   | ![Original Torus Voronoi division and distance heatmap](image5) | ![RMDVDA Torus Voronoi division and distance heatmap](image6) |

Table 6: Model 3: Torus Voronoi division and distance heatmap
4 THERMAL MANAGEMENT OF NOC

The thermal management of NoC is a complex problem. Recall power is energy per time. If these algorithms use fewer cores, the active cores’ overall power usage may not change due to increased computational load. Moreover, because the flow matrix algorithm presented takes communication and computation into account, the whole divisible load’s total execution time may have less than a proportional increase in using fewer cores (energy is power times execution time for a load). This depends on the value of $\sigma$. Using fewer cores will result in less fixed energy/power costs. Finally, using fewer cores allows more flexibility in hot spot management. Using fewer cores may also be an advantage in freeing up cores for other jobs, some of which may require a free core(s).

5 DISCUSSION ON LOAD SHARING

5.1 Question

Assume in a homogeneous network, an embedded tree is distributing load in a network. For a binary tree node, one may have a link with the data rate of $1/z$, and the outgoing total data rate is $2/z$. It, on the surface, violates the conservation of data entering and leaving a node.

5.2 Explanation

The issue is complex as for some distribution policies,

- Firstly, not all outgoing links of a node may be active at the same time (say under sequential load distribution).
- Secondly, suppose that a store and forward policy is used in a load distribution policy. In that case, all data is present at the (possibly intermediate) root node before forwarding to output links occurs, so that there is no mismatch of data transfer. What is present in the simulations of this paper is a mixture of virtual cut through (at least some of the children nodes of a parent node start computing as they receive load at time zero) and store and forward (as described in this bullet). Because of the store and forward aspect, the simulations in this paper do not have the issue of a mismatch of data rate.
- Finally, usually less data may leave a node than enters it (the node keeps, i.e., some for processing) so that the outgoing links from a node are not as active as the incoming link to a node.

Still, if one chooses an overall virtual load distribution tree technique what can we do to mitigate this problem? To mitigate the problem, in the virtual distribution links, one needs two constraints:

- The data rate of each load distribution link is less than the physical capacity of each link.
- A node’s effective virtual input data rate matches the node’s effective total virtual output data rate.

This issue was addressed in the earlier divisible load theory literature (often using heuristic methods). See (Drozdowski 2009) (Błażewicz and Drozdowski 1996) (Drozdowski and Wolniewicz 2004) (Głazek 2003).
6 CONCLUSION

This work is significant in proposing the use of established Voronoi diagram techniques for finding efficient NoC flow distributions. It is novel in the use of optimal linear solution technique based on the flow matrix for finding flow distributions within individual clusters. Savings in the number of processors used are impressive and bode well for minimizing chip power consumption.

7 ACKNOWLEDGEMENT

The authors wish to thank Prof. Emre Salman and Prof. Maciej Drozdowski for useful discussions. We also thank the anonymous reviewers’ very useful comments.

REFERENCES

Benini, L., and G. De Micheli. 2002. “Networks on chips: A new SoC paradigm”. *computer* vol. 35 (1), pp. 70–78.

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

Bharadwaj, V., D. Ghose, and T. G. Robertazzi. 2003. “Divisible load theory: A new paradigm for load scheduling in distributed systems”. *Cluster Computing* vol. 6 (1), pp. 7–17.

Błażewicz, J., and M. Drozdowski. 1996. “The performance limits of a two dimensional network of load-sharing processors”. *Foundations of Computing and Decision Sciences* vol. 21 (1), pp. 3–15.

Casanova, H., A. Legrand, and Y. Robert. 2008. *Parallel algorithms*. Chapman and Hall/CRC.

Chin, F., and C. A. Wang. 1998. “Finding the constrained Delaunay triangulation and constrained Voronoi diagram of a simple polygon in linear time”. *SIAM Journal on Computing* vol. 28 (2), pp. 471–486.

Drozdowski, M. 2009. *Scheduling for parallel processing*. Springer.

Drozdowski, M., and P. Wolniewicz. 2004. “Performance limits of divisible load processing in systems with limited communication buffers”. *Journal of Parallel and Distributed Computing* vol. 64 (8), pp. 960–973.

Ganguly, A., K. Chang, S. Deb, P. P. Pande, B. Belzer, and C. Teuscher. 2011. “Scalable hybrid wireless network-on-chip architectures for multicore systems”. *IEEE Transactions on Computers* vol. 60 (10), pp. 1485–1502.

Głazek, W. 2003. “A multistage load distribution strategy for three-dimensional meshes”. *Cluster Computing* vol. 6 (1), pp. 31–39.

Hung, J. T., and T. G. Robertazzi. 2004. “Switching in sequential tree networks”. *IEEE Transactions on Aerospace and Electronic Systems* vol. 40 (3), pp. 968–982.

Jia, J., B. Veeravalli, and J. Weissman. 2010. “Scheduling multisource divisible loads on arbitrary networks”. *IEEE Transactions on Parallel and Distributed Systems* vol. 21 (4), pp. 520–531.

Krevat, E., J. G. Castaños, and J. E. Moreira. 2002. “Job scheduling for the BlueGene/L system”. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 38–54. Springer.

Lammie, T., and G. Thomas. 2005. “A linear daisy chain with two divisible load sources”. *Conference Information Science and Systems*.

Liu, X., H. Zhao, and X. Li. 2007. “Scheduling Divisible Workloads from Multiple Sources in Linear Daisy Chain Networks.”. In *PDPTA*, pp. 528–534.
Marchal, L., Y. Yang, H. Casanova, and Y. Robert. 2005. “A realistic network/application model for scheduling divisible loads on large-scale platforms”. In 19th IEEE international parallel and distributed processing symposium, pp. 10–pp. IEEE.

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

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

Robertazzi, T. G. 1993. “Processor equivalence for daisy chain load sharing processors”. IEEE Transactions on Aerospace and Electronic Systems vol. 29 (4), pp. 1216–1221.

Robertazzi, T. G. 2017. Introduction to Computer Networking. Springer Science.

Robertazzi, T. G., and D. Yu. 2006. “Multi-source grid scheduling for divisible loads”. In Proceedings of the 40th Annual Conference on Information Sciences and Systems, pp. 188–191.

Sohn, J., and T. G. Robertazzi. 1996. “Optimal divisible job load sharing for bus networks”. IEEE Transactions on Aerospace and Electronic Systems vol. 32 (1), pp. 34–40.

Stojmenovic, I., A. P. Ruhil, and D. Lobiyal. 2006. “Voronoi diagram and convex hull based geocasting and routing in wireless networks”. Wireless communications and mobile computing vol. 6 (2), pp. 247–258.

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

Wong, H. M., D. Yu, B. Veeravalli, and T. G. Robertazzi. 2003. “Data intensive grid scheduling: multiple sources with capacity constraints.”. In Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems, Volume 1, pp. 7–11.

Zhang, J. 2018. “Data Distribution Equivalence for Data Intensive Interconnection Networks”. PhD dissertation, Dept Of Applied Mathematics and Statistics, Stony Brook University.

Zhang, J., Y. Liu, S. Li, and T. G. Robertazzi. 2018. “Optimizing Data Intensive Flows for Networks on Chips”. arXiv preprint arXiv:1812.07183.

AUTHOR BIOGRAPHIES

JUNWEI ZHANG received the PhD degree from the Applied Mathematics and Statistics Department of Stony Brook University in 2018. His research interests include parallel computing optimization, computational geometry and applied machine learning. His email address is junweizhang23@gmail.com.

YANG LIU received his PhD degree from Department of Electrical and Computer Engineering at Stony Brook University, Stony Brook, NY, in 2017. Previously, he received his B.E. degree from Department of Electrical and Computer Engineering at University of Electronic Science and Technology of China, Chengdu, China, in 2011. His research interests are in the area of distributed/parallel computing, networking, and load balancing algorithms. He is currently working on divisible load theory and heterogeneous system applications. His email address is yangliu89415@gmail.com.

LI SHI received his Ph.D. degree from Department of Electrical and Computer Engineering at Stony Brook University, Stony Brook, NY, in 2016. Previous, he received his B.E. degree in electrical and computer engineering from Shanghai Jiao Tong University, Shanghai, China, in 2010. He is working at Snap Inc, Venice, CA. His email address is lishi.pub@gmail.com.

THOMAS ROBERTAZZI is a Professor of Electrical and Computer Engineering at Stony Brook University. He is an IEEE Fellow. He received the PhD from Princeton University and the B.E.E, from the Cooper...
Union. He has published extensively in areas such as scheduling, performance evaluation and networking. His email address is thomas.Robertazzi@stonybrook.edu.