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 [9] of divisible workload in a mesh networks and the other problem is scheduling divisible workloads from multiple sources in mesh 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. In addition, we also propose a rigorous mathematics proof about the flow matrix optimal solution existence and unique. 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. The user 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. Experimental evaluation through simulations demonstrates that a task can be finished in the same suboptimal time and yet save about 30% of processor resources. Further, the lower band of intuitive and heuristic algorithm is also investigated.**

I. INTRODUCTION

A. Related Literature

1) 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.

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

B. Problem Description

Networks on chips (NOC) represent the smallest networks that have been implemented to date [8]. 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 [7]. 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 Voronoi diagrams [3] [5]. The methodology presented here can be applied to a variety of switching/scheduling protocols besides those directly covered in this paper.

In this paper, we investigate the virtual cut-through switching [6]. 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.

Equivalence computation [9] is a technique, which consists of combining a cluster of processors as one whole equivalent processor to process a unit 1 workload.

II. ASSUMPTION AND NOTION

A. Assumption

The following assumptions are used throughout the paper:

- Under virtual cut-through switching, a node can relay the beginning bit of a message (packet) before the entire message is received.
- 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.

[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

- 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 [8].
- 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.

B. *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.
- $level_i$: The processors have i minimum Manhattan distance to the data injection node.
- α_0: The load fraction assigned to the root processor.
- α_i: The load fraction assigned to the ith processor.
- 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.
- 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.
- T_{cm}: Communication intensity constant. The entire load is transmitted in time $z_i T_{cm}$ seconds over the ith link.
- $T_{f,n}$: The finish time of the whole processor network. Here $T_{f,n}$ is equal to $\omega_{eq} T_{cp}$.
- $T_{f,0}$: The finish time for the entire divisible load solved on the root processor. Here $T_{f,0}$ is equal to $1 \times \omega_0 T_{cp}$, that is $\omega_0 T_{cp}$.
- $\sigma = \frac{zT_{cm}}{\omega T_{cp}}$: The ratio between the communication speed to the computation speed, $0 < \sigma < 1$ [2] [4].
- In multi-source situation, $\sum_{i=1}^{k} L_i = 1$
- $\sum_{i=0}^{m*n-1} \alpha_i = 1$
- $Speedup = \frac{T_{f,0}}{T_{f,n}} = \frac{\omega T_{cp}}{\alpha_0 \omega T_{cp}} = \frac{1}{\alpha_0}$

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 [2] [10].

III. Equivalence Computation

A. *2*2 mesh network*

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.

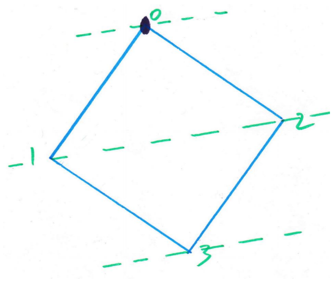

Fig. 1: The 2*2 mesh network and the root processor is P_0

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

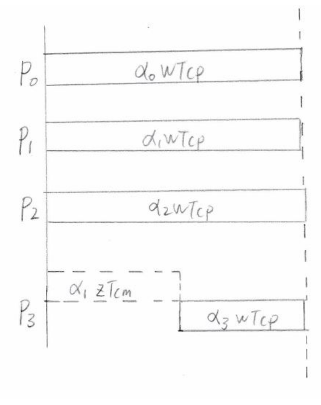

Fig. 2: The timing diagram for 2*2 mesh network and the root processor is P_0

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 [10].

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

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

The group of equations are represented by the matrix form:

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

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:

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

The equations say 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 3 homogeneous mesh network processes load L and L originates 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.

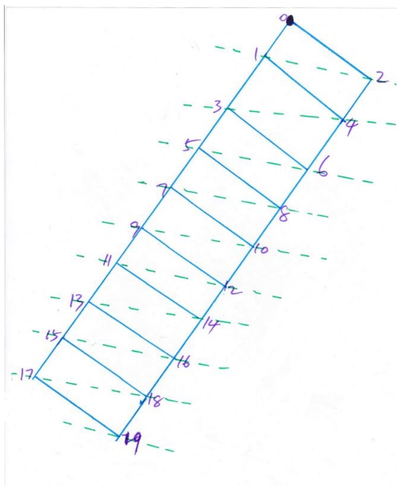

Fig. 3: 2*n (n = 10) mesh network and the workload happens on P_0

The equations are presented as:

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

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:

$$
\left\{ \alpha_i = \left| \frac{\det A_i^\star}{\det A} \right| \right. \tag{29}
$$

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:

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

. In addition, we can extend the flow matrix pattern to $m*n$ mesh network.

C. Proof of solution existence and unique

We take $2*10$ mesh network as an example, other $m*n$ mesh network is proved by the same mathematics technique.

If the solution exist and is unique, we need to prove $\det A \neq 0$.

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

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:

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

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.

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.

$$\min \quad T_{f,n} \tag{33}$$

$$\text{s.t.} \quad \sum_{i \in 0 \cdots (n-1)} \alpha_i = 1, \tag{34}$$

$$\alpha_i \geq 0, \tag{35}$$

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 $\alpha_i$
  Collapse the data injection processors into one "big" equivalent processor [9].
  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 $O(k*\max(m,n)^3)$.

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.

$$\min \quad T_{f,n} \tag{36}$$
$$\text{s.t.} \quad \sum_{i\in 0\cdots(n-1)} \alpha_i = 1, \tag{37}$$
$$\alpha_i \geq 0, \tag{38}$$

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.
```

The time complexity of algorithm consists of two parts, one is about the determinant computation of flow matrix and the other is about the Manhattan distance Voronoi diagram.

C. Reduced Voronoi Division Algorithm

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.

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

$$\min \quad T_{f,n} \tag{39}$$
$$\text{s.t.} \quad \sum_{i\in 0\cdots n} \alpha_i = 1, \tag{40}$$
$$\alpha_i \geq 0, \tag{41}$$

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

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.

From the average saved processors ratio , it shows the average percentage of saved processor is about 35%.

D. Lower Band of Intuitive and Heuristic Algorithm

Considering a $n*n$ mesh network, there are l data load injections, the makespan is constrained by the makespan of bottleneck, so the best solution is each data load injection deploys data to its community, which is $\frac{n*n}{m}$ cores. We calculate the best row

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] Veeravalli Bharadwaj et al. *Scheduling divisible loads in parallel and distributed systems*. Vol. 8. John Wiley & Sons, 1996.

[3] Steven Fortune. "A sweepline algorithm for Voronoi diagrams". In: *Algorithmica* 2.1-4 (1987), p. 153.

[4] Jui Tsun Hung and Thomas G Robertazzi. "Switching in sequential tree networks". In: *IEEE Transactions on Aerospace and Electronic Systems* 40.3 (2004), pp. 968–982.

[5] Jingxi Jia, Bharadwaj Veeravalli, and Jon Weissman. "Scheduling multisource divisible loads on arbitrary networks". In: *IEEE Transactions on Parallel and Distributed Systems* 21.4 (2010), pp. 520–531.

[6] Parviz Kermani and Leonard Kleinrock. "Virtual cut-through: A new computer communication switching technique". In: *Computer Networks (1976)* 3.4 (1979), pp. 267–286.

[7] Elie Krevat, José G Castaños, and José E Moreira. "Job scheduling for the BlueGene/L system". In: *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer. 2002, pp. 38–54.

[8] Thomas G Robertazzi. *introduction to computer networking*. Springer Science & Business Media, 2017.

[9] 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.

[10] 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.