STONY BROOK UNIVERSITY

CEAS Technical Report 838

Modeling Load Balancing through Differential Equations:

Milton J. Jackson and Thomas G. Robertazzi

October 23, 2012

Modeling Load Balancing Through Differential Equations

Milton J. Jackson and Thomas G. Robertazzi
Department of Electrical & Computer Engineering
Stony Brook University
Stony Brook, NY 11794
mjj5000@optimum.net, tom@ece.sunysb.edu

Abstract

A technique is proposed where the load sharing of divisible loads is modeled through the use of a set of differential equations. These equations are used to describe the flow of data between processors connected in a linear daisy chain and two dimensional mesh configurations. The two dimensional mesh is reduced to nine basic structures that describe any m by n mesh. The processors have heterogeneous link speeds, processing speeds, and loads. The equations are designed to model the balancing of these loads over a period of time, by distributing the loads between the processors, allowing the amount at any processor's load to be accurately calculated at any point in time.

1. Introduction

In this paper a method is advanced that uses differential equations to model the load sharing of divisible loads. The differential equations are the decisions makers, and use the amount of load on adjacent processors, to determine if a task (load) is to be passed to another processor. The exchange of data between processors in two different topologies is demonstrated. These are the linear daisy chain and mesh topologies. Both configurations have heterogeneous link speeds, with processors that function at different rates. The loads of the linear daisy chain and mesh configurations are divisible, and have load amounts that vary. The decisions made by the differential equations balance these loads, through a process of redistribution of the loads among the processors. The load is considered processed when the load at all of the processors

has reached zero. The exact amount of load at any processor can be determined precisely at any point in time.

2 State of The Art

There is a large body of literature on load balancing. Most of this literature is dedicated to indivisible jobs. Here some representative works will be considered that also encompass divisible jobs. Load balancing techniques are often components of an application. Their purpose is to make the applications perform better. Different techniques are better suited for certain applications. The types of load balancing techniques and applications vary. However the common thread in load balancing is the redistributions of tasks. Proposed methods are diverse as in [1] where a work stealing algorithm is used that distributes workloads in a parallel system so that underutilized processors seek out work from other processors. Mitzenmacher demonstrates the usefulness of this modeling technique. When a processor is idle it tries to steal a task from a processor selected at random with an uniform distribution. If the chosen processor has more than one task, a task is stolen. This model employs differential equations. The differential equations rely on the expected change in the behavior of the system over small periods of time. That is based on arrivals or departures of tasks. A core term of these equations is similar to a term in the equations in this paper, but used differently.

Another technique is diffusive load balancing [8]. In diffusive load balancing if a processor has a quantity of tasks greater than its neighbor it moves a portion of its tasks to the neighboring processor. The decision to move some of the tasks is based solely on local information, and the amount of tasks transferred is in proportion to the differential between the number of tasks on the two processors. Furthermore only the number of tasks at each

node is subject to attempts of equalization. This method of having nodes with greater loads transfer load to nodes with lesser loads is used in this paper. An interesting load balancing technique is used in [3], where data from a high energy physics experiment is analyzed. The problem of an imbalance, occurring in queues is resolved using dynamic load balancing, resulting in the ability to detect the point of imbalance and remove it.

An application detailed in [5], where a dynamic load balancing technique is used as part of an Asynchronous Iterations-Asynchronous Communication (AIAC) model. Here the load balancing is not based on the amount of data, but the residual i.e. the max norm of the difference between a current value and two consecutive iterations. In this paper dynamic load balancing is load balancing that evolves as load arrives, and is transferred and processed. Here one task is to determine if asynchronism interferes with an effective distribution of the workload when load balancing is used. The results demonstrate that using load balancing in conjunction with asynchronism gives a greatly improved performance than when asynchronism is used by itself. In [6] a dynamic load balancing method is developed for parallel applications. That has the purpose of evenly distributing work to processors to ensure that processors are not idle while others are busy. For simple data structures the application divides the work into groups of equal size and distributes the groups among the processors. For complex data structures as in unstructured meshes, partitioning methods are used.

A similar method is used in [7], where mesh adaptation is shown to be an effective application for unstructured-grid computation. However a load imbalance is produced among the processors when used on parallel nodes. To solve this problem dynamic load balancing is used. The technique employs partitioning and remapping to balance the load. In [2] a load

balancing algorithm for a distributed system is used. The algorithm has the purpose of minimizing the expected turnaround time. This is to prevent a situation in which a task waits for one particular processor while there is another idle processor that could process the task. The performance of the system is increased, when the instantaneous load on the multiple processors network is balanced. Presented in [4] is an agent based load balancing technique that is used in a homogeneous min-grid to achieve a uniform distribution of task to nodes. This solves the scheduling problem on the grid.

In references [1-7] the loads are indivisible i.e. they cannot be arbitrarily divided, and directed to any desired node. However divisible jobs such as [8] mentioned previously are usually data files instead of programs (or files in which the information must be processed as a block of data). In [9] divisible load theory is presented with a synopsis of previous research illustrating its achievements. Basic divisible load theory is discussed in [10, 11], and linear daisy chains are covered in [19]. In [12, 13] N processors are connected by a bus, and the processors are controlled by sensors. The divisible load produced by the sensors results in a load sharing problem in which the optimum solution time is reached if all the processors stop computation at the same time. Divisible loads are also used in [14] that studies closed form solutions for large symmetric tree networks and in [11] where infinite networks are discussed.

Divisible load theory (DLT) can be applied to different topologies, a two dimensional mesh in [16, 18], and a three-dimensional mesh in [17]. Distributed linear networks that can have processors with and without front-end processors are studied in [19]. In [20] linearity in DLT is shown by comparing superposition in an electrical network to that of a network using DLT. The use of DLT in grid networks is explained in [21]. Signature searching is investigated in [22, 23]

where single and multiple files that can be arbitrary divided and examined. In [24] a process for balancing a linear daisy (LDC) chain is offered. In this process the initial load is placed on an exterior node of the LDC and the load propagates from node to node. The division of the load depends on whether or not the exterior node has front end processing or not. Contrary to [24], in this paper the loads are placed individually at each node at some point in time, and each node can be considered a root node. Each node immediately starts to process load, and is capable of transmitting load at the same. Similar to one case in [24], the nodes in this paper function as nodes with front end processing and simultaneous start. That is they transmit and receive at the same time and immediately start processing load as it is received, but only in a given interval of time. It is not required that all nodes stop working at the same time i.e. the finish time requirement for optimization does not come into play here. However as an extension of this work the algorithm is being augmented to redirect load from nodes with large loads to those with lesser loads to achieve the finish time requirement.

While Moges and Robertazzi sought to unite aspects of divisible load theory and Markov chain models, the intent of this paper is to present and elegant and scalable model in a basic form. This is to say that any network, cycle, or system that can be defined in terms of processor speed, link speed, and load can be represented by this model. Fig. 1, shows a water allegory that exemplifies the load balancing technique in this paper. Imagine containers of water with an aperture at the bottom of each container. The size of the apertures represents how fast the water will flow out of the containers. This is analogous to the processing speed. The containers are connected in a linear daisy chain of n nodes, and can be extended to a grid containing m by n nodes.

Water Analogy

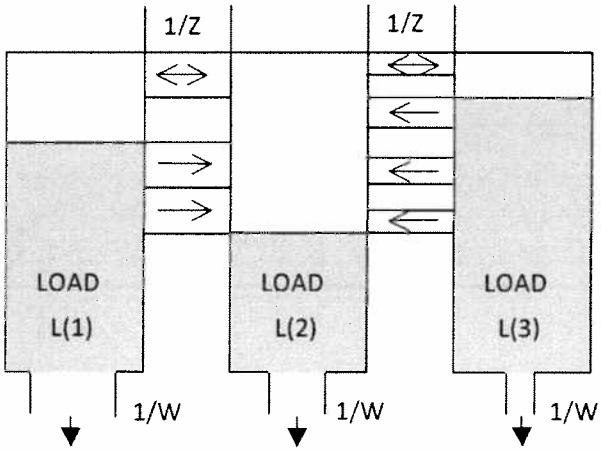

Fig. 1. Demonstrates a water analogy using three containers, where n=3. The load L(n) is water, the processor speed is 1/w, and the link speed is 1/z. The width of the apertures at the bottom of the containers indicates the processors speed, and the arrows the direction of flow. The apertures on the sides of the containers represent the link speeds, and the arrows the direct of the load transfers from container to container. There is no actual distance between the containers.

If the sides of the containers are constructed such that water flows through them from a higher potential (pressure) to a lower potential, the size of the apertures the water passes through represents the links speed. Every portion of the wall is considered to be an aperture of a certain size, the only barrier to flow is an equal potential. Larger apertures will transmit water faster. The load, water can be thought of as a divisible load. In fact any quantity that can be arbitrarily divided at any point e.g. chemical concentration or information can be regarded as a divisible load.

When water is placed in the containers processing will start, and end when the last of the water transverses the network of containers. Note the decision to transmit water depends on the local conditions at the time, and constantly changes. The algorithm put forth in this paper

can provide information on when the load has reached zero, and the amount of load at each processor at every time instant in a predetermined interval.

The approach in this paper differs from [24] in that the equations are not derived from a timing diagram. This model is a load balancing technique, and could be placed as a component in some of the applications mentioned above. With this technique underutilized or idle processors are identified. Through changes to the program that uses this technique, the information on underutilized or idle nodes could be used to increase the rate at which all nodes reach a desired state. Furthermore this technique would be extremely useful in the detection of imbalances in queues or in a system, since it can be used to find the load at every node, at every time interval.

3. Model Description

Both the linear daisy chain and two dimensional mesh configurations use the same method for distributing loads. Each node is capable of sharing its load with its adjacent neighbors. The loads move from the nodes with larger loads to those with lesser loads. This process continues until a solution is reached, in other words, the loads reach zero at all nodes.

4. Variables

w(i): the inverse processor speed

z(i): the inverse link speed in the horizontal directions

v(i): the inverse link speed in the vertical directions

L: the load

DL: derivate of the load

dt: the time interval

sgn(x): the signum function

N: the total number of processors

n: the number of columns

m: the number of rows

5. Linear Daisy Chain Configuration

In a linear daisy chain configuration, Fig. 2, the following set of differential equations describes this load balancing process.

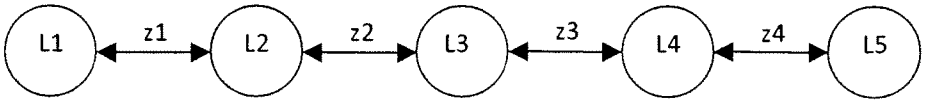

Fig. 2. Five loads connected in a linear daisy chain configuration, by four links.

For the linear daisy chain:

i: the number of processors between the first and last processor

i = 2 to N-1, in increments of 1

$$dL(1) = dt\left[-\frac{1}{w(1)} + \frac{1}{z(1)}sgn(L(2) - L(1))\right] \tag{1}$$

$$dL2(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{z(i-1)}sgn(L(i-1) - L(i))\right] \tag{2}$$

$$dL3(N) = dt\left[-\frac{1}{w(N)} + \frac{1}{z(N-1)}sgn(L(N-1) - L(N))\right] \tag{3}$$

$$L(1) = L(1) + dL1(1) \qquad L(1) \geq 0 \tag{4}$$

$$L(i) = L(i) + dL2(i) \qquad L(i) \geq 0 \tag{5}$$
$$L(N) = L(N) + dL3(N) \qquad L(N) \geq 0 \tag{6}$$

The derivative of the load (dL) with respect to time is divided into three equations. The derivative dL1(1) represents the load on the first node, and the dL3(N) represents the load on the last node. The derivative dL2(i) represents the loads on interior nodes between the first and last nodes. The processor speed (1/w) has a minus sign to insure that the act of processing always serves to decrease the load when added to the inverse link speed. The sign of the link speed (1/z) is determined by the difference between the loads. If the difference is greater than zero the sign is positive, and if it is less than zero it is negative. Load is transported from nodes with larger loads to adjacent nodes with lesser loads. In (1), the inverse link speed z(1) and the inverse processor speed w(1) are fixed.

The differential equation dL1(1) uses the difference between the first processor's load(L1), and its nearest neighbor, the second processor's load(L2), and the inverse link speed between them in its calculations. In (1) "one" is used for the load at the first node, and "two" for the load at the second node instead of i. The reason will become clear in the description of dL2(i). The derivative dL3(N) works similar to dL1(1), the only difference being that dL1(1) uses the node to its right and dL3(N) used the node to its left in the calculations. The derivative dL2(i) takes into account its neighbors on either side. The variable i start at 2 to prevent division by zero in the term 1/z(i-1) of (2). Other than this dL2(i) functions in the same sense as dL1(1) and dL3(N). Once the derivatives of the loads are found they are added to the original loads, L(1), L(i) and L(N) creating a new set of loads. The process repeats until the loads are balanced, i.e. each node's load equals zero. Loads are forced to be no less than zero. When a load reaches

zero it remains at zero unless a new load is added to that particular node. The load at a node may also be increased at anytime in the process. Furthermore, the number of nodes may be increased to any size N.

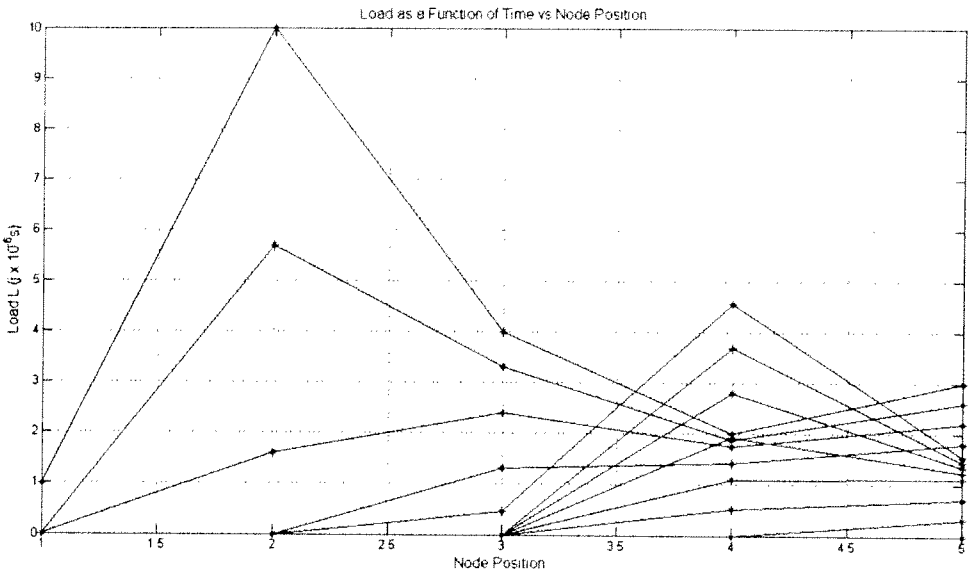

Fig. 3. Load as a function of time vs. node position. There are five nodes, and the loads on these nodes are measured at time, $j \times (1 \times 10^{-6}s)$ where j represents the number of iterations of (1) through (6).

In Fig. 3, the loads as a function of time are plotted against the node position. The amount of load on each node at a specific time is designated by the function $L(j \times dt(s))$, where j is the number of iterations of (1) through (6), and dt is the time interval, the iteration occurred in, here 1×10^{-6} s. As can be seen in Fig. 3, the initial load on nodes one through five are respectively [1 10 4 2 3]. They diminish with time in accordance with equations (1) through (6). However at a given time when j = 3500 and $dt = 1 \times 10^{-6}s$, the load at node 4 is increased by 5, L(j × 10^{-6} s) = L(0.0035 s) = 5. The increase can be seen at node 4 on the graph above. Note in Fig. 3 the increase appears to be 4.5 and not 5. The discrepancy is due to the fact that the information in the graph is plotted every 1000 iterations. The difference is the amount the load has decreased,

from the time the load is input to the time the data is updated. The increase at node 4 increases the time it will take for this node to reach zero.

6. Mesh Configuration

In the mesh configuration of Fig. 4, the nodes are extended in the vertical and horizontal directions forming a five by five mesh. In practice any m by n mesh is possible. There are 25 nodes (N), 20 inverse links speeds (z) in the horizontal directions, and 20 inverse link speeds (v) in the vertical directions. This mesh processes loads the same as the linear daisy chain, with an extra dimension added. The process is governed by nine equations that will work for any m by n mesh. However for the equations to work properly the mesh must be labeled as shown in Fig. 4. Other labeling schemes are possible, but result in somewhat different equations. In the nine equations that describe the process of transferring loads between nodes, m is the number of rows and n is the number of columns. The equations are numbered (7) through (15).

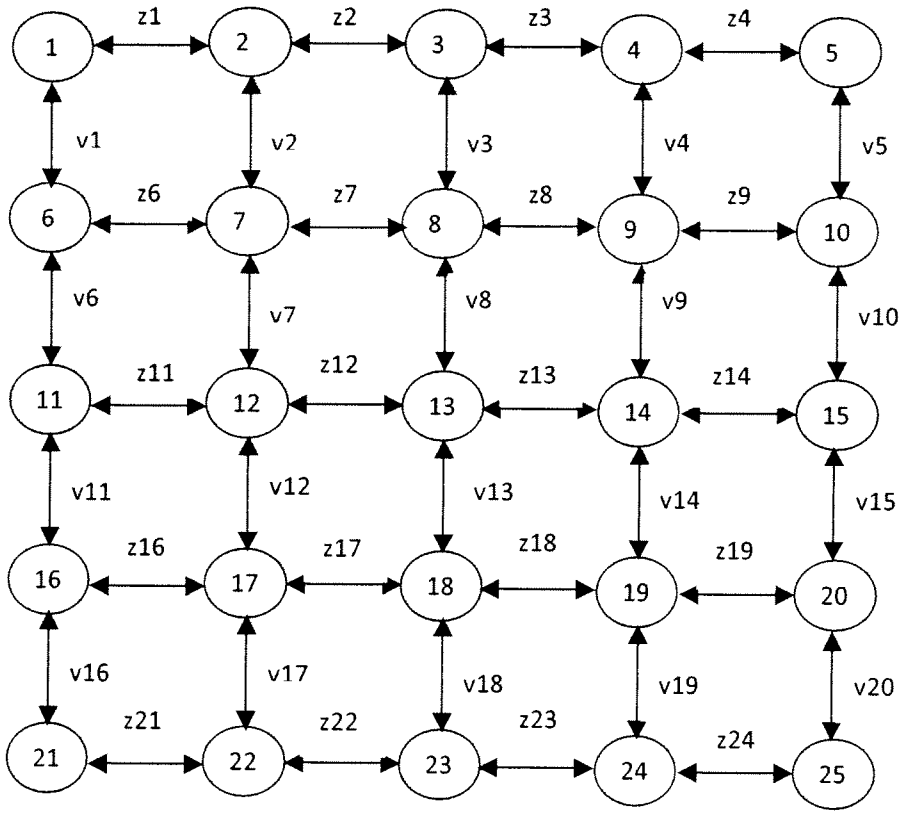

Fig. 4. A 5 by 5 mesh configuration, composed of 25 nodes, 20 horizontal links, and 20 vertical links, here N is the total number nodes, m is the number of rows, and n is the number of columns.

For the mesh configuration:

i: the number of processors from 1 to n

Where n is the number of columns and m is the number or rows

for i = i1 = 1

$$dL1(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(n))\right] \qquad (7)$$

for i = i2 = 2 to n-1, increments of 1

$$dL2(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{z(i-1)}sgn(L(i-1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(n))\right] \quad (8)$$

for i = i3 = n

$$dL3(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i-1)}sgn(L(i-1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(i))\right] \quad (9)$$

for i = i4 = n+1 to (N-n+1) – n, in increments of n

$$dL4(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(i)) + \frac{1}{v(i-n)}sgn(L(i-n) - L(i))\right] \quad (10)$$

rows = m-2
for g = 0 to rows -1
i = i5 = [(n+2) : in increments of 1: to (2n-1)] + g × n

$$dL5(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{z(i-1)}sgn(L(i-1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(i)) + \frac{1}{v(i-n)}sgn(L(i-n) - L(i))\right] \quad (11)$$

for i = i6 = 2n: in increments of n: to (N-n)

$$dL6(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i-1)}sgn(L(i-1) - L(i)) + \frac{1}{v(i)}sgn(L(i+n) - L(i)) + \frac{1}{v(i-n)}sgn(L(i-n) - L(i))\right] \quad (12)$$

for i = i7 = (N-n) +1

$$dL7(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{v(i-n)}sgn(L(i-n) - L(i))\right] \quad (13)$$

for i = i8 = (N-n) + 2: in increments of 1: to (N – 1)

$$dL8(i) = dt\left[-\frac{1}{w(i)} + \frac{1}{z(i)}sgn(L(i+1) - L(i)) + \frac{1}{z(i-1)}sgn(L(i-1) - L(i)) + \frac{1}{v(i-n)}sgn(L(i-n) - L(i))\right] \tag{14}$$

for i = i9 = N

$$dL9(i) = dt\left[-\frac{1}{w(N)} + \frac{1}{z(N-1)}sgn(L(N-1) - L(N)) + \frac{1}{v(N-n)}sgn(L(N-n) - L(N))\right] \tag{15}$$

$$L(i1) = L(i1) + dL1(i1) \qquad L(i1) \geq 0 \tag{16}$$
$$L(i2) = L(i2) + dL2(i2) \qquad L(i2) \geq 0 \tag{17}$$
$$L(i3) = L(i3) + dL3(i3) \qquad L(i3) \geq 0 \tag{18}$$
$$L(i4) = L(i4) + dL4(i4) \qquad L(i4) \geq 0 \tag{19}$$
$$L(i5) = L(i5) + dL5(i5) \qquad L(i5) \geq 0 \tag{20}$$
$$L(i6) = L(i6) + dL6(i6) \qquad L(i6) \geq 0 \tag{21}$$
$$L(i7) = L(i7) + dL7(i7) \qquad L(i7) \geq 0 \tag{22}$$
$$L(i8) = L(i8) + dL8(i8) \qquad L(i8) \geq 0 \tag{23}$$
$$L(i9) = L(i9) + dL9(i9) \qquad L(i9) \geq 0 \tag{24}$$

The mesh in Fig. 4 is described by nine structures shown individually in Fig. 5. Each equation corresponds to a structure, and each structure has one or more components. Furthermore each component has a center node. In Fig. 5 only the center node of the first component is shown when there is more than one component. In the 5 by 5 mesh of Fig. 4, there are a total of 25 components. The equation numbers (7) to (15) correspond to the structure numbers (7) to (15). The structures representing the upper left and right hand corners, (7) and (9) respectively, and the lower left and right hand corners (13) and (15) respectively each have one component, which is composed of three nodes and two links. Structure (8) has three components corresponding to the number of nodes (columns) between the first and last node

of the first row. Each component is made up of four nodes and three links. Structure (14) is the complement of (8). This structure is found in the last row of the mesh, and contains the same number of components, nodes, and links as (8), for the same reasons. Structure (10) and its complement structure, (12) contains three components, based on the number of nodes (rows) found between the first and last rows of the mesh. They have four nodes and three links. The final structure, (11) has nine components each composed of five nodes and four links.

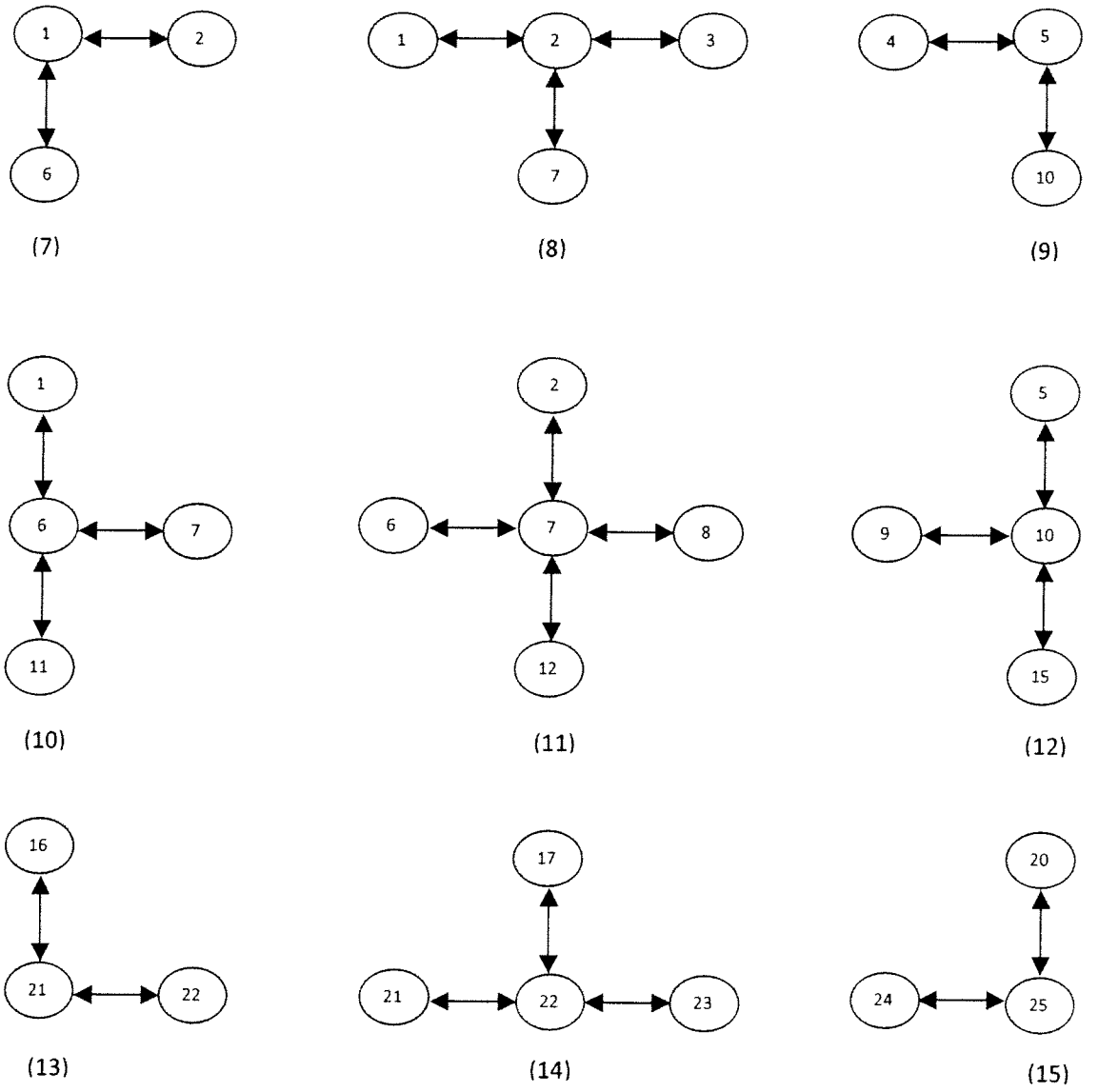

Fig. 5. The nine structures describe any m by n mesh. The structure's number corresponds to the equation's number. For a 5 by 5 mesh as in Fig. 4, the structures (7), (9), (13), and(15) each have one component. The structures (8), (10), (12), and (14) each have three components and structure (11) has nine components. The center nodes mark the position of the first component of each structure.

Once the derivative of a load at a particular node is found it is added to the current load of that node. This produces a new value for that load. This process is seen in (16) through (24).

The load at each node in the five by five mesh is plotted versus load as a function of time, versus the node position, Fig. 6.

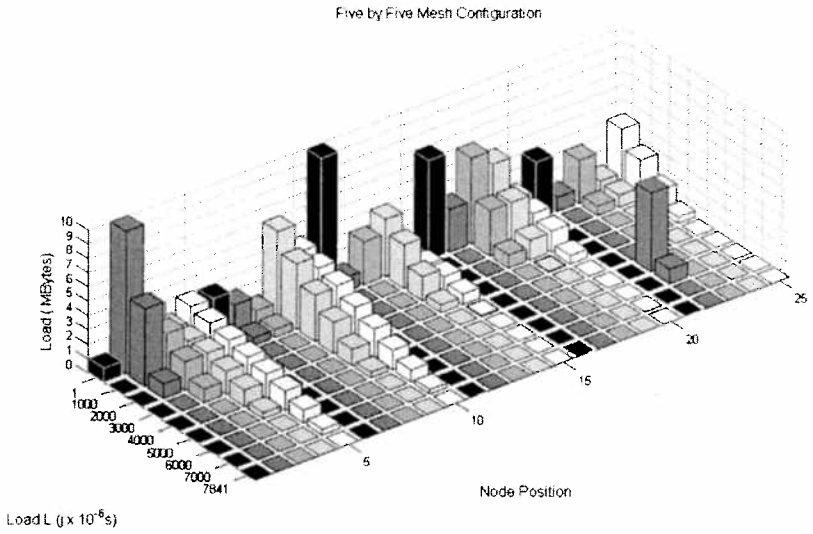

Fig. 6. Plot of the loads versus the loads as a function of time versus the node position. The x-axis gives the nodes position, the y-axis the load as a function of time, and the z-axis the load in mega bytes.

The rows in Fig. 6, starting at j equals one represents the initial input loads on the nodes of the 5 by 5 mesh in Fig. 4. The 5 by 5 mesh has been transformed into an array of length 25, and each element of the array 1 through 25 corresponds to the node number and load of the 5 by 5 mesh. Every 1000 iterations of (7) through (24) produces a new row, until all load reach zero at j = 7841. At j = 5000, the load at node position 22, $L(j \times 10^{-6}s)$ is set equal to 10. This node rapidly approaches zero because there are no load bearing nodes adjacent to it. Equation (14) governs this process.

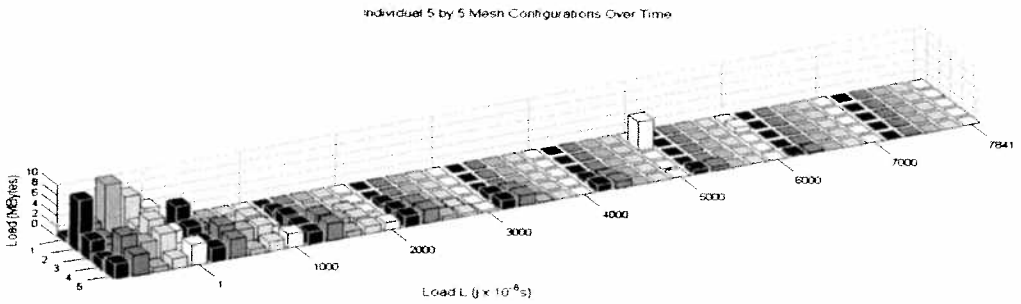

Fig. 7. Displays the 5 by 5 meshes as time increases. The y-axis shows the number of columns, and the x-axis the rows, listed every 1000 iterations.

With the nodes adjacent to node 22 being zero, the four terms of (14) are negative, generating a large negative number. When dL8(i8) is subtracted from L(i8) in (23), the load goes to zero within 2000 iterations.

A geometrically clearer picture of the surrounding nodes can be seen in Fig. 7. After the initial 5 by 5 mesh is plotted, with one on the x-axis marking the last row of the 5 by 5 mesh. The mesh is plotted again after 1000 iterations. This process continues until all nodes in the mesh are zero.

Programming the given equations has generated the previous graphs. For the one-dimensional daisy chain (1) through (6) were used, and for the two-dimensional mesh (7) through (24). The two dimensional m by n mesh can be extended into three dimensions by an arbitrary length p, producing an m by n by p mesh. A 5 by 5 by 5 mesh would require one additional parameter, the link along the z-axis h. This cube would be composed of the following structures. Eight corner structures each consisting of one component having three links and four nodes. Twelve edge structures, where each structure has three components made of four

links and five nodes. Six face structure comprised of nine components consisting of five links and six nodes. The last structure is for the internal nodes of the cube. This structure has 27 components, each having six links, and seven nodes. The center nodes of the components do not lie on the edges or surfaces of the cube.

7 Simulation Evaluation

For equal loads, processors speeds and link speeds, the methodology works as would be intuitively expected. Also, if the processor speeds and the link speeds are chosen such that the processor speed is one third that of the link speed, and the loads are chosen randomly, the equations work as expected. The number of links dictates how fast the load decreases. The structure with the most links decreases the fastest. This decrease is imperceptible in graphs such as Fig. 6 and Fig. 7. However a decrease in the second and third decimal place is readily seen in the numbers used to generate the graphs, when the parameters are changed to observe smaller steps, i.e. when the values for the loads are listed every 10 iterations, instead of in increments of 1000 iterations.

8 Conclusion And Extensions

The data derived from the equations demonstrates that the equations perform as intended. In other words the loads tend to be balanced over time.

It should be noted that this methodology can be used to model other protocols involving thresholds for allowing load transfers, different speeds for different amounts at the loads, and time varying processor and link speeds, and opening and closing multiple data pipes on each link as time evolves. Many systems and cycles not normally considered as being networks can be modeled as networks. Thus the nearest neighbor transfer heuristic used here is of interest

in other applications, such as modeling chemical diffusion through cells and across the cell walls.

REFERENCES

[1] M. Mitzenmacher, "Analyses of Load Stealing Models Based on Differential Equations," SPAA 98 Puerto Vallarta Mexico, ACM, Pages 212-221, 1998.

[2] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," ACM, Pages 47-55, 1998.

[3] J. Kasaraneni, T. Johnson, and P. Avery, "Load Balancing in a Distributed Processing System for High-Energy Physics (UFMulti)," ACM, Pages 177-181, 1995.

[4] J. Liu, X. Jin, and Y. Wang, "Agent-Based Load Balancing on Homogeneous Mini-grids: Macroscopic Modeling and Characterization," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, no.7, Pages 586-598, July 2005.

[5] J. M. Bahi, S. Contassort-Vivier, and R. Couturier, "Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, no. 4, Pages 289-299, April 2005.

[6] K. Devine, B. Hendrickson, E. Boman, M. St. John and C. Vaughan, "Design of Load-Balancing Tools for Parallel Applications," ICS 2000 Santa Fe, New Mexico, USA, Pages 110-118, 2000.

[7] L. Oliker and R. Biswas, "Efficient Load Balancing and Data Remapping for Adaptive Grid Calculations," SPAA 97 Newport, Rhode Island, USA, 1997, ACM, Pages 33- 42, 1997.

[8] R. Subramanian and I.D. Scherson, "An Analysis of Diffusive Load Balancing," SPAA 94 – 6/94 Cape May, N. J, USA, ACM, Pages 220-225, 1994.

[9] 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, Jan., 2003.

[10] T.G. Robertazzi, "Ten Reasons to Use Divisible Load Theory," Computer, vol. 36, no. 5, pp. 63-68, May, 2003.

[11] T.G. Robertazzi, Networks and Grids Technology and Theory. New York: Springer Science + Business Media LLC, 2007.

[12] S. Bataineh and T.G. Robertazzi, "Bus Oriented Load Sharing for a Network of Sensor Driven Processor," IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 5, pp.

1202-1205, Sept., 1991.

[13] J. Sohn and T.G. Robertazzi, "Optimal divisible job load sharing for bus network," IEEE Transactions on Aerospace and Electronic Systems, vol. 32, issue 1, pp. 34-39, Jan., 1996.

[14] S. Bataineh, T. Hsiung, and T.G. Robertazzi, "Closed form solutions for bus and tree network of processors load sharing divisible job," IEEE Transactions on Computers, vol. 43, issue 10, pp. 1184-1196, Oct., 1994.

[15] S. Bataineh and T.G. Robertazzi, "Performance limits for processor network with divisible jobs," IEEE Transactions on Aerospace and Electronic Systems, volume 33, issue 4, pp. 1189-1198, Oct., 1997.

[16] J. Blazewicz, M. Drozdowski, and P.J. Weglarz, "Performance limits of a two dimensional network of load sharing processors," Foundation of computing and Decision Sciences, vol. 21, no.1, pp.3-15, 1996.

[17] M. Drozdowski, and W. Glazek, "Scheduling divisible loads in a three-dimensional mesh of processors," Parallel Computing vol. 25, issue 4, pp. 381-400, April, 1999.

[18] J. Blazewicz, M. Drozdowski, F. Guinand, and D. Trystram, "Scheduling a Divisible Task in a Two-Dimensional Toroidal Mesh," Discrete Applied Mathematics vol. 94, issue 1-3, pp.35-50, May, 1999.

[19] V. Bharadwaj, D. Ghose, V. Mani, and T.G. Robertazzi, Scheduling Divisible Loads in Parallel and Distributed Systems. Alamitos, CA: IEEE Computer Society Press, 1996.

[20] T.G. Robertazzi and D. Yu, "Multi-source grid scheduling for divisible loads," Conference on Information Sciences and Systems, Princeton University, Princeton, NJ, pp. 188-191, March, 2006.

[21] D. Yu and T.G. Robertazzi, "Divisible load scheduling for grid computing," Proc. of the IASTED international conference on Parallel and Distributed Computing and System, Los Angeles, Nov., 2003.

[22] K. Ko and T.G. Robertazzi, "Signature searching time evaluation in flat file data bases," IEEE Transactions on Aerospace and Electronics Systems, vol. 44, no. 2, pp. 493-502, April 2008.

[23] M. Drozdowski and P. Wolniewiez, "Experiments with scheduling divisible tasks in clusters of workstations," in A. Bode, T. Ludwig, W. Karl, R. Wismuller, (Eds.), Proceeding of EURO-Par2000, Lecture Notes in Computer Sciences, LNCS 1900, New York: Springer-Verlag, pp. 311-319, 2000.

[24] M. Moges and T. Robertazzi, "Divisible load scheduling and markov chain models, vol. 52, issue 10-11, pp. 1529-1542, November, 2006.