Load Balancing Algorithms over Hexa-Hyper Cell Topology

Mohammad Fasha, Esam Y. A. Al-Nsour, Mohammad Asmaran

Table of Contents:

List of Tables

List of Figures

Load balancing on Hyper Hexa-Cell Interconnection Network

1. Introduction

1.1 Hyper Hexa-Cell Topology (HHC):

Hyper Hexa-Cell (HHC) topology proposed in [1] consists of interconnected processors (nodes) using static interconnection network. A d_h-dimensional HHC is built by connecting 2^{dh-1} Hexa cells using (d_h-1)-dimensional hyper-cube. Every one-dimensional HHC consists of six nodes arranged in Hexa corners shape. This Hexa-shape is built by connecting six processors in a shape of two triangles so that each triangle connects three nodes (processors) together. Each node in the two triangles is connected to its corresponding node in the other triangle. Figure 1-a shows 2-dimensional hyper-cube and figure 1-b shows one-dimensional HHC, while figure 2-a illustrates how each 1-dimensional HHC replaces 2 dimensional hyper-cube corners, figure 2-b shows 3-dimensional HHC.

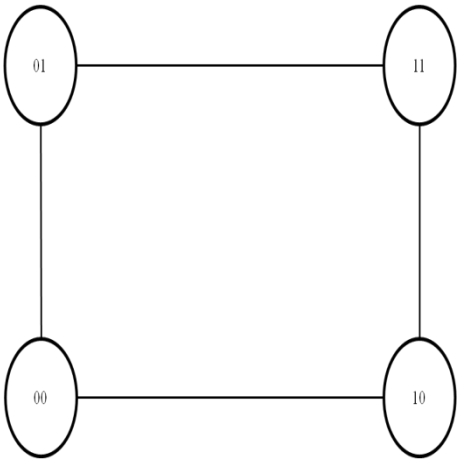

Figure 1-a: Two-dimensional hyper-cube.

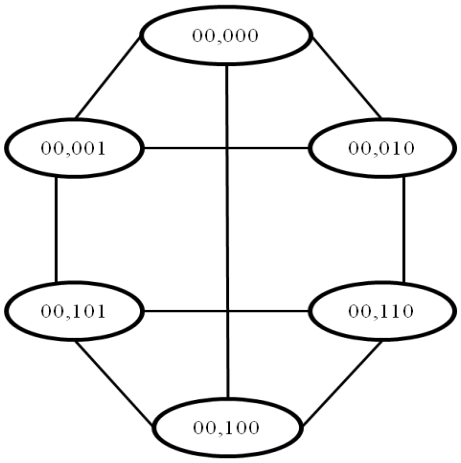

Figure 1-b: One-dimensional HHC.

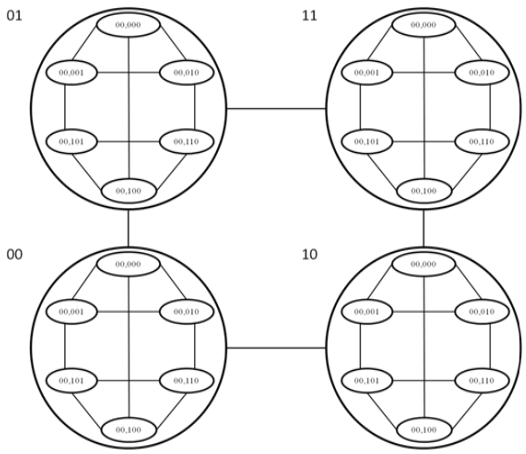

Figure 2-a: Each 1-dimensional HHC replaces the single nodes of a 2-d hyper-cube.

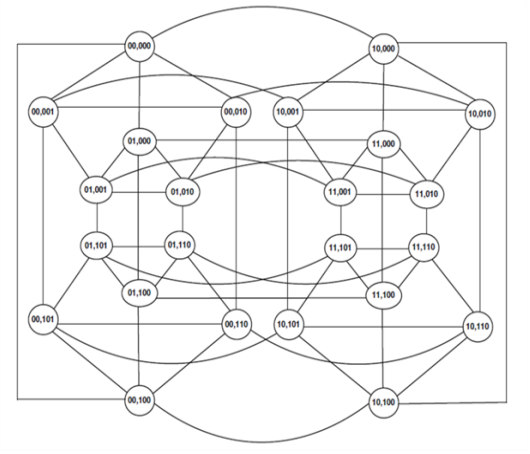

Figure 2-b: Three-dimensional HHC.

1.2 Hyper Hexa-Cell (HHC) addressing scheme.

The processor address in a Hyper Hexa-Cell (HHC) network consists of two parts P(s,g), Least significant part (the right part - g) consists of three bits and it is used for addressing interior hexa cell nodes, while most significant part (the left part - s) consists of d_h-1 bits (where d_h is the number of dimensions of the HHC) and it is used for addressing different Hexa cells in the HHC. As an example; a node with address P(10,110) is node located in the third hexa cell of the HHC (10), and its local Hexa cell address is 110. Furthermore, this node is the right node of the lower triangle of its Hexa cell.

1.3 Load balancing in interconnection networks.

Load balancing is a process that aims to insure fair distribution of load units among all the nodes in the network with minimal error possibility. Load may be defined as number of tasks waiting for run in queue, array elements, etc. [2].

Load balancing process involves reallocating of load units (tasks, jobs, etc.) from the over-loaded nodes to under-loaded ones. This increases the overall system utilization and throughput and it is necessary to measure the load of the nodes in the network and distributed system. This involves marking under-loaded/free and overloaded/busy nodes so that load is transferred from overloaded processor to an under-loaded one [3]. This would insure equal progress rate of all nodes [4].

Basically, load balancing algorithms can be categorized into two main categories, static load balancing algorithms (SLB) and dynamic load balancing algorithms (DLB). In SLB, task assignment are performed before runtime, it is not possible to re-assignment after the execution start, while in DLB the task

assignment is performed dynamically at or after a process starts to execute, depending on the changes of the load. Excess load will be transferred from heavily loaded nodes to the lightly loaded or idle ones. Complexity and communications overhead is less in SLB than DLB, while DLB is more responsive to the changes in distributed systems which yields to an overall better performance [5].

1.4 Statement of purpose:

This work presents three different HHC load-balancing algorithms and evaluates them using several metrics such as Complexity, Communication Steps, Speed, and error probability.

2. Proposed Load Balancing Algorithms for the HHC.

In this work, we introduce three different balancing algorithms, all are static load balancing algorithms that work to balance load on the nodes of HHC interconnection network. In general, all of them are balancing load in three general phases. In the beginning each Hexa-cell's triangle is balanced in parallel. Then, each hexa-cell's corresponding triangles are balanced. Finally, Dimension Exchange Method (DEM) [6] is applied on the different dimensions of the HHC. This indicates that differences in the three algorithms are located in the first phase only and the remaining phases are the same for the whole proposed algorithms. We assume that the sizes of load units at the network are equal, the maximum load at any node is M, and all connections between nodes are electrical connections. Hexa cell nodes having 0's in its right address part most significant bit are called upper-triangle, while those having 1's are called lower-triangle. In each triangle, the nodes having 00 in its address least significant two bits are called triangle-coordinators. The nodes having 01 are called left nodes (L-node), and nodes having 10 are called right nodes (R-node).

We will introduce each algorithm in a separate section that contains its pseudo code, brief description of its work, and analytical analysis of its metrics.

2.1 Algorithm A

2.1.1 Pseudo code

I. At each one-dimensional HHC (in parallel).

1. For each one-dimensional HHC of the d_h-dimension HHC interconnection network do in parallel.
   For both triangles do in parallel ,
2. Exchange L-node's weight (w_L), R-node (w_R).
3. Calculate average load: $AvgLoad_{L\text{-}R} = Floor\ [(w_L + w_R + 1) / 2]$.
4. If (Local-load > $AvgLoad_{L\text{-}R}$)
5. Send excess load (Local-load - $AvgLoad_{L\text{-}R}$ load) to neighbor node (along the Left-Right
6. connection).

Local-load = $AvgLoad_{L-R}$
7. Else If (Local-load < $AvgLoad_{L-R}$) and ($|w_L - w_R| > 1$)
8. Receive excess load ($AvgLoad_{L-R}$ load - Local load) from neighbor node (along the Left-Right
9. connection).
Local-load = Local-load + ($AvgLoad_{L-R}$ - Local-load)
10. Send (w_L), (w_R) values to coordinator.
12. Else if (node is R-node)
13. Send (w_L), (w_R) values to coordinator.
14. If (node = coordinator)
15. Receive from either L-node or R-node the value of (w_L), (w_R).
16. Calculate triangle average load: Avg_{TRI} = Floor [(w_{COR} + w_L + w_R + 2) / 3].
17. Send Avg_{TRI} to L-node and R-node.
18. Calculate local-excess-load = local-load - Avg_{TRI}
19 If (local-excess-load > 0)
20. Send to L-node or R-node or both the excess load according to their weights.
21. Else If (local-excess-load < 0)
22. Receive to L-node or R-node or both the excess load according to their weights.
23. If (node <> coordinator)
Receive from coordinator
24. Calculate the excess load: Local-excess-load = local-load - Avg_{TRI}
25. If (local-excess-load > 0)
26. Send to coordinator the excess load.
27. Else If (local-excess load < 0)
28. Receive from coordinator the excess load.
29. // By the end of this step, load is balanced among each triangle.
30. Each triangle node exchanges its weight with corresponding node at the other triangle (nodes that
31. differ only in the most significant bit of its one- dimensional HHC (in parallel).
32. Each node calculate the average weight of its load and its corresponding node load according to the formula: $AvgLoad_{TRI(0)-TRI(1)}$ = Floor [($w_{(0XX)}$ +$w_{(1XX)}$ + 1) / 2].
The node which has excess load sends its excess load to the other node.
33. // By the end of this step, load is balanced among both triangle.

34.
35.
II Between different dimensions of the d_h-dimensional HHC interconnection network (in parallel).
35. For (J=1; J<d_h ; J++)
36. For all pairs of nodes (x,y) that have the same node address and differs only in the J^{th} bit of its sub-group address, do in parallel:
37. Exchange weight between the two nodes along the dimension J.
38. Calculate average weight: Avg-load = (w_x + w_y) / 2
39. If local-load > Avg-load
40. Send excess load (local-load –Avg-load) to neighbor node along dimension J
41. local-load = Avg-load
42. Else If local-load < Avg-load
43. Receive excess load (neighbor-load - Avg-load) from neighbor node along dimension J
local-load = local-load + (neighbor-load - Avg-load)
44. // By the end of this step, load is balanced among All dimensions.
45.

2.1.2 Description of algorithm work

The algorithm balances the load of the HHC interconnection network using two main phases; phase one balances the load inside each one-dimensional HHC, and phase two to balance load between different dimensions of the HHC interconnection network. Phase one contains two parts; part one to

balance load inside each triangle first, then part two to balance load between the two triangles. Part 1 has two steps; first step to balance load between the pair of peripheral nodes of each triangle, then second step to balance load between the peripheral nodes with its coordinator at each triangle. Part2 balances load between the two triangles in each one-dimensional HHC. Phase2 applies the DEM algorithm to balance load between different HHC dimensions. In the following we give a detailed description of the algorithm work:

Phase 1: In each one dimensional HHC:

Part1, Step1: balancing load between the peripheral nodes of each triangle:

A: In the upper triangle; Node1 and Node2 exchange their weights values. The same is done in parallel in the lower triangle; Node5 and Node6 exchange their weights values.

B: Each node of pair nodes (1, 2), (5, 6) calculate the average of its weight with its corresponding node weight in the same triangle in parallel and floors the result.

C: Upon the value of the average weight, each node of pair nodes (1, 2), (5, 6) will do in parallel:

If the node load exceeds average weight, it would send its extra load to its corresponding node.

Else If the node load is less than the average weight, it would send its average weight value in addition to the average weight value of the corresponding node to the triangle coordinator node (Node0 for the upper triangle, Node4 for the lower triangle).

If the node load is equal to its corresponding node, even address nodes (Node2 and Node6) will communicate with the coordinator to send its weight value in addition to the weight value of the corresponding node (Node0 for the upper triangle, Node4 for the lower triangle).

Part1, Step 2: balancing load between the peripheral nodes and its coordinator:

Coordinator nodes (0, 4) in parallel compute the floor of average weight value of the entire weight values of its triangle (the sum of the 3 nodes weight), send the average to its peripheral nodes and do as the following cases:

A: Its load more than average load: it will send half of the excess load to every one of its peripheral nodes in two communication steps maximum.

B: its load less than average load: it will send a request to its peripheral nodes acquiring its excess load; the message would contain the value of the needed excess-load in two communication steps maximum and receiving the excess load in another two communication steps.

Part2: balancing load between the two triangles:

A: Each node of both triangles exchanges its weight value with its corresponding (directly connected) node in the other triangle ([0-4], [1-5], [2-6]).

B: Each node computes the floor average and sends its excess load to the corresponding node.

Phase 2: In all dimensions of HHC: DEM algorithm is applied to the whole hyper cube for d_h-*1* steps where d_h is the dimension of the HHC.

A: all pairs of nodes that share the same node address and differ only in the J^{th} bit of its sub-group address exchange weight along the dimension J, and calculate the average weight.

B: If the average weight is greater than the local load, calculate the excess load and send it to the neighbor node along dimension J, else receive the excess load from the neighbor node along dimension J.

2.1.3 Algorithm analytical evaluation:

A: *Execution time*: [8] measures the required time to achieve the load balancing in the network, it represents the worst case time complexity for the algorithm to balance the load in an HHC interconnection network. Assuming M as the maximum load units at any node, the execution time is calculated as in table1:

<table>
<tr><td colspan="3">Table 1: Calculating Execution time for algorithm A</td></tr>
<tr><td rowspan="3">Phase1: balances the load of each one-dimensional HHC of the HHC interconnection network</td><td rowspan="2">Part1: balances load inside each triangle</td><td>step1: Maximum of M/2</td></tr>
<tr><td>step2.a: Maximum of M/6</td></tr>
<tr><td>step2.b: Maximum of M/6</td></tr>
<tr><td colspan="2">Part2: balances load between triangles; Maximum of M/6</td></tr>
<tr><td colspan="3">Phase1 Total is: M/2 + 2M/6 + M/6 = M</td></tr>
<tr><td colspan="3">Phase2: DEM algorithm is used; for all dimensions(d_h) of HHC the maximum transferred load is:
in the first dimension (d_h=2) max of M/12
in the second dimension (d_h=3) max of M/24
in the third dimension (d_h=4) max of M/48
and so on …
M/12 + M/24 + M/48 + …
= M/12 * ($1/2^0$ + $1/2^1$ +$1/2^2$ +$1/2^3$ +….)
= (M/12) * $\sum_{i=0}^{d_h-2} 1/2^i$
= (M/12) * $\sum_{i=0}^{d_h-2} (1/2)^i$
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1 – (1/2))
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1/2))
= (M/12) * (2 * (1- $(1/2)^{d_h-1}$))
= (M/12) * ((2- 2 * $(1/2)^{d_h-1}$))</td></tr>
</table>

= ((M/12) *2) – ((M/12) * 2 * $(1/2)^{d_h-1}$)) = ((M/6) – ((M/6) * $(1/2)^{d_h-1}$)) = (M/6) * (1 – $(1/2)^{d_h-1}$))
Sum for all phases = phase1 + phase2 = M + (M/6) * (1 – $(1/2)^{d_h-1}$)) ≈ *O* (M+M/6) = ***O* (7M/6)**

B: *Load balancing accuracy*: [8] the difference between maximum workload units at any node minus the minimum workload units at any other node, it is referred to as the error in balancing load. In this algorithm there are three steps where error may accumulate; step1 when balancing load inside each triangle, step2 when balancing the load among the two triangles, and step3 when balancing the load among different dimensions. Table 2 shows the three steps error

Table 2: Calculating Load balancing accuracy for algorithm A		
Step1: balancing load inside each triangle	Maximum error = 1	Since the coordinator is adjusting the total load before dividing by three to ensure that maximum error is = 1.
Step2: balancing the load among the two triangles	Maximum error = 1	When an odd number of workload units is to be divided by two.
Step3: balancing the load among different dimensions	Maximum error = d_h-1	The DEM algorithm has d_h-1 as the Max error when balancing load among different dimensions
Total error (e) <= 1 + 1 + d_h-1 = 2 + d_h-1 = **1 + d_h.**		

C: *Number of communication steps*: [8] represents the number of steps required by the nodes in the interconnection network to communicate in order to perform load balancing, table 3 shows the communication steps needed for every phase of the algorithm, it calculate the maximum communication steps at any node and the total communication steps in the network [9].

Table 3: Calculating Number of communication steps for algorithm A: *Maximum communication steps at any node and the total communication steps in the network.*		
Phase1: balances the load inside each one dimensional HHC of the HHC interconnection network; Maximum communication steps at any node in one-dimensional HHC is: 3 + 5 + 3 = 11 Total for one-dimensional HHC is: 9 + 20 + 18 = 47. The sum of communication steps in all 1-dimensional HHC in the network is = 47 * 2^{d_h-1}	Part1: balances load inside each triangle	Step1: Three communication steps for each triangle = 3. Max for any node = 3 Total for both triangles: 3 * 3 = 9 Step2.a and Step 2.b: Maximum five communication steps for the coordinator with its nodes (3 with one of the peripheral nodes and 2 with the other one). Max for any node = 5 Total for each triangle = 5+3+2=10 Total for both triangles = 10*2 = 20

	Part2: balances load between nodes (three nodes) of the two triangles; Maximum for any node = 3 communication steps Total for 6 nodes = (6*3) = 18 communication steps
Phase2: balances load between different dimensions of the HHC interconnection network; in DEM algorithm, for each node = 3 * (d_h-1) = 3d_h - 3, where 3 is the needed communication steps at each link (2 for exchange weight and one for sending excess load, and d_h is the number of dimensions of the HHC network). Total for every 1-dimensional HHC of the network is = 6 * (3d_h-3) = 18d_h – 18, where 6 is the number of nodes in a 1-dimensional HHC. Total number of communication steps during DEM phase = number of communication steps at each 1-dimensional HHC * sum of 1-dimensional HHC = (18d_h – 18) * (2^{d_h-1})	
Maximum communication steps at any single node is in the two phases is: phase1 maximum steps + phase2 maximum steps = 11 +(3d_h - 3) = **3d_h + 8** **Total communication steps on the network is**: phase1 total steps + phase2 total steps = = (47 * 2^{d_h-1}) + (18d_h - 18) * (2^{d_h-1}) = (2^{d_h-1}) * (47 + 18d_h - 18) = **(2^{d_h-1}) * (18d_h + 29)**	

D: *Speed*: [8] represents the speed at which the load balancing algorithm achieves its work, it is calculated by multiplying the number of communication steps by the speed of the connection links in the network. Assuming that the electrical connections in the HHC network have the speed of 250 Mb/s [7], the algorithm speed can be calculated as follows:

Speed= Number of communication steps * electrical link speed

Speed = **(3d_h + 8) * 250 Mb/s**

2.2 Algorithm B

2.2.1 Pseudo code

I. At each one-dimensional HHC (in parallel).

1. For each one-dimensional HHC of the d_h-dimension HHC interconnection network do in parallel.
   For both triangles do in parallel ,
2. Send L-node's weight (w_L) to coordinator.
3. Send R-node's weight (w_R) to coordinator.
4. Coordinator calculates average load of the triangle:
5. $AvgLoad_{TRI} = Floor [(w_{COR} + w_L + w_R + 2) / 3]$.
   Coordinator calculates delta load for each node of the triangle:
6. $Delta\text{-}Load_{Nood(i)} = AvgLoad_{TRI} - w_{Node(i)}$.
   Coordinator sends to L-Node its Delta-Load to be sent/received and from/to which node.
7. Coordinator sends to R-Node its Delta-Load to be sent/received and from/to which node.
   Each node will transfer its excess load to its neighbor/s or receive load according to the
8. coordinator message in maximum two steps.
   // By the end of this step, load is balanced among each triangle.
9.
   Each triangle node exchanges its weight with corresponding node at the other triangle (nodes that
10. differ only in the most significant bit of its one- dimensional HHC (in parallel).
    Each node calculate the average weight of its load and its corresponding node load according to
12. the formula: $AvgLoad_{TRI(0)\text{-}TRI(1)} = Floor [(w_{(0XX)} + w_{(1XX)} + 1) / 2]$.
    The node which has excess load sends its excess load to the other node.
    // By the end of this step, load is balanced among both triangle.
13.

14.
15.

II Between different dimensions of the d_h-dimensional HHC interconnection network (in parallel).

16. For ($J=1$; $J<d_h$; J++)
17. For all pairs of nodes (x,y) that have the same node address and differs only in the J^{th} bit of its sub-group address, do in parallel:
18. Exchange weight between the two nodes along the dimension J.
19. Calculate average weight: Avg-load = $(w_x + w_y) / 2$
20. If Avg-load > 0
21. Send excess load (local-load –Avg-load) to neighbor node along dimension J
22. local-load = Avg-load
23. Else If Avg-load < 0
42. Receive excess load (Avg-load – neighbor-load) from neighbor node along dimension J
    local-load = local-load + (Avg-load - neighbor-load)
25. // By the end of this step, load is balanced among All dimensions.
45.

2.2.2 Description of algorithm work

The algorithm balances the load on an HHC interconnection network using two main phases; phase one balances the load inside each one-dimensional HHC, and phase two to balance load between different dimensions of the HHC interconnection network. Phase one contains two parts; part one to balance load inside each triangle first, then part two to balance load between the two triangles. Part 1 balances load between the peripheral nodes with its coordinator at each triangle. Part2 balances load

between the two triangles in each one-dimensional HHC. Phase2 applies the DEM algorithm to balance load between different HHC dimensions. In the following we give a detailed description of the algorithm work:

Phase 1: In each one dimensional HHC:

Part: balances load between the peripheral nodes with its coordinator at each triangle:

A: In the upper triangle; Node1 sends its weight to its coordinator, then Node2 sends its weight to its coordinator. The same is done in parallel in the lower triangle; Node5 then Node6 sends its weight to its coordinator.

B: Upon the value of average weight calculated at the coordinator, the coordinator first sends to Node1 then to Node2 a message contains the amount of load to send/receive and from which node. The same is done in parallel in the lower triangle; Node5 then Node6 receive a message contains the amount of load to send/receive and from which node.

C: According to the message from its coordinator, if Node1 and Node5 have excess load to be send to the triangle coordinator it will start sending while the coordinator is sending the information message to the other node. In worst case, maximum 2 steps is needed to exchange excess load (A node is to send load to the two other nodes or a node is to receive from the other two nodes)

Part2: balancing load between the two triangles:

A: Each node of both triangles exchanges its weight value with its corresponding (directly connected) node in the other triangle ([0-4], [1-5], [2-6]).

B: Each node computes the floor average and sends its excess load to the corresponding node.

Phase 2: In all dimensions of HHC: Apply DEM algorithm to the whole hyper cube for d_h-*1* steps where d_h is the dimension of the HHC.

A: each pairs of nodes that share the same node address and differs only in the J^{th} bit of its sub-group address exchange weight along the dimension J, and calculate the average weight.

B: If the average weight is greater than the local load, calculate the excess load and send it to the neighbor node along dimension J, else receive the excess load from the neighbor node along dimension J.

2.2.3 Algorithm analytical evaluation:

A: *Execution time*: measures the required time to achieve the load balancing in the network, it represents the worst case time complexity for the algorithm to balance the load in an HHC interconnection network. Assuming M as the maximum load units at any node, the execution time is calculated as in table 4:

<table>
<tr><td colspan="3">Table 4: Calculating Execution time for algorithm B</td></tr>
<tr><td rowspan="3">Phase1: balances the load of each one-dimensional HHC of the HHC interconnection network</td><td rowspan="2">Part1: balances load inside each triangle</td><td>step1: Maximum of 2M/6</td></tr>
<tr><td>step2: Maximum of 2M/6</td></tr>
<tr><td colspan="2">Part2: balances load between triangles; Maximum of M/6</td></tr>
<tr><td colspan="3">Phase1 Total is: 2*(2M/6) + M/6 = 5M/6</td></tr>
<tr><td colspan="3">Phase2: DEM algorithm is used; for all dimensions(d_h) of HHC is:
in the first dimension (d_h+1) max of M/12
in the second dimension (d_h+2) max of M/24
in the third dimension (d_h+2) max of M/48
and so on…
M/12 + M/24 + M/48 + …
= M/12 * ($1/2^0 + 1/2^1 + 1/2^2 + 1/2^3 + \ldots$)
= (M/12) * $\sum_{i=0}^{d_h-2} 1/2^i$
= (M/12) * $\sum_{i=0}^{d_h-2} (1/2)^i$
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1 – (1/2))
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1/2))
= (M/12) * (2 * (1- $(1/2)^{d_h-1}$))
= (M/12) * ((2- 2 * $(1/2)^{d_h-1}$))
= ((M/12) *2) – ((M/12) * 2 * $(1/2)^{d_h-1}$))
= ((M/6) – ((M/6) * $(1/2)^{d_h-1}$))
= (M/6) * (1 – $(1/2)^{d_h-1}$))</td></tr>
<tr><td colspan="3">Sum for all phases = phase1 + phase2 = (5M/6) + (M/6) * (1 – $(1/2)^{d_h-1}$))
≈ O(5M/6 + M/6) = O(M)</td></tr>
</table>

B: *Load balancing accuracy*: the difference between maximum workload units at any node minus the minimum workload units at any other node, it is referred to as the error in balancing load. In this algorithm there are three steps where error may accumulate; step1 when balancing load inside each triangle, step2 when balancing the load among the two triangles, and step3 when balancing the load among different dimensions. Table 5 shows the three steps error

Table 5: Calculating Load balancing accuracy for algorithm B		
Step1: balancing load inside each triangle	Maximum error = 1	Since the coordinator is calculating the triangle average and ensure that maximum error is = 1.
Step2: balancing the load among the two triangles	Maximum error = 1	When an odd number of workload units is to be divided by two.

Step3: balancing the load among different dimensions	Maximum error = 1	The DEM algorithm has d_h-1 as the Max error when balancing load among different dimensions
Total error (e) <= 1 + 1 + d_h-1 = 2 + d_h-1 = **1 + d_h.**		

C: *Number of communication steps*: represents the number of steps required by the nodes in the interconnection network to communicate in order to perform load balancing, table 6 shows the communication steps needed for every phase of the algorithm, it calculate the maximum communication steps at any node and the total communication steps in the network [9].

Table 6: : Calculating Number of communication steps for algorithm B: Maximum communication steps at any node and the total communication steps in the network.

<table>
<tr><td rowspan="4">Phase1: balances the load inside each one dimensional HHC of the HHC interconnection network;
Maximum communication steps at any node in one-dimensional HHC is:
2 + 2 + 2 + 3= 9
Total for one-dimensional HHC is:
8 + 8 + 8 + 18 = 42.
The sum of communication steps in all 1-dimensional HHC in the network is
= 42 * 2^{d_h-1}</td><td rowspan="3">Part1: balances load inside each triangle</td><td>step1: Two communication steps; one for each node sending its weight to its coordinator
Max for any node: 2 (Coordinator)
Total for one triangle: 2+1+1=4
Total for two triangles is: 4*2 = 8</td></tr>
<tr><td>step2: Two communication steps; Coordinator sends for its nodes the information message.
Max for any node: 2 (Coordinator)
Total for one triangle: 2+1+1=4
Total for two triangles is: 2*4 = 8</td></tr>
<tr><td>step3: In worst case at any node, 2 communication steps is needed to exchange excess load (A node is to send load to the two other nodes or a node is to receive from the other two nodes)
Max for any node: 2
Total for one triangle: 2+1+1=4
Total for two triangles is: 2*4 = 8</td></tr>
<tr><td colspan="2">Part2: balances load between nodes (three nodes) of the two triangles; 3 communication steps for 3 nodes
Maximum for any node = 3 communication steps
Total for two triangles = (6*3) = 18</td></tr>
<tr><td colspan="3">Phase2: balances load between different dimensions of the HHC interconnection network; in DEM algorithm, for each node = 3 * (d_h-1) = 3d_h - 3, where 3 is the needed communication steps at each link (2 for exchange weight and one for sending excess load, and d_h is the number of dimensions of the HHC network).
Total for every 1-dimensional HHC of the network is = 6 * (3d_h-3) = 18d_h – 18, where 6 is the number of nodes in a 1-dimensional HHC.
Total number of communication steps during DEM phase = number of communication steps at each 1-dimensional HHC * sum of number of 1-dimensional HHC
= (18d_h – 18) * (2^{d_h-1})</td></tr>
</table>

Maximum communication steps at any single node is in the two phases is:
phase1 maximum steps + phase2 maximum steps = 9 +($3d_h$ - 3) = **$3d_h$ + 6**
Total communication steps on the network is:
phase1 total steps + phase2 total steps =
= (42 * 2^{d_h-1}) + ($18d_h$ - 18) * (2^{d_h-1})
= (2^{d_h-1}) * (42 + $18d_h$ - 18)
= **(2^{d_h-1}) * ($18d_h$ + 24)**

C: *Speed*: represents the speed at which the load balancing algorithm achieves its work, it is calculated by multiplying the number of communication steps by the speed of the connection links in the network. Assuming that the electrical connections in the HHC network have the speed of 250 Mb/s, the algorithm speed can be calculated as follows:

Speed = Number of communication steps * electrical link speed

Speed = ($3d_h$ + 6) * 250 Mb/s

2.3 Algorithm C

2.3.1 Pseudo code

Each processor runs the following algorithm in parallel:

Phase I step 1 – Triangles Load Balancing

1. *For each connected peer in my designated triangle*
2. *Get load information from that peer*
3. *Compute the total load for the 3 triangle partners*
   Compute the average load for the 3 triangle partners as following:
4. *AverageLoad = floor((TotalLoad + 2) / 3);*
   Compute my excess load as following:
5. *MyExcessLoad = MyLoad - AverageLoad;*
6. *While (MyExcessLoad > 0)*
7. *For each MyTrianglePeer in MyTrianglePeers*
8. *Lock my peer workload-banks*
9. *Re-Poll myTrianglePeer Load Info*
   Compute myTrianglePeer NeededWorkdLoad as following:
10. *neededWorkLoad = myTrianglePeer->Load - averageLoad*
11. *If (NeededWorkdLoad is minus)*
12. *If (MyExcessLoad >= NeededWorkdLoad)*
13. *Send myTrianglePeer NeededWorkdLoad*
14. *MyExcessLoad-= NeededWorkdLoad*
15. *else*
16. *Send myTrianglePeer all MyExcessLoad*
17. *MyExcessLoad=0*
18. *UnLock my peer workload-banks*
19. *Wait for all processors in my HHC to reach this stage before proceeding to HHC two triangles load balancing (Thread Barrier)*

Phase I step 2 – Load Balancing between Opposite Triangles inside a Single HHC

1. *Select the corresponding peer in the opposite fronting triangle*
2. *Get load information for that peer*
3. *If ((MyLoad – MyPeerLoad) >= 2)*
4. *Compute TotalLoad for both peers*
5. *Compute the AverageLoad load for both peers*
6. *Compute my Excess load as following:*
7. *MyExcessLoad = MyLoad - AverageLoad;*
8. *Send MyExcessLoad to MyPeer*
9. *Wait for all processors to reach this stage before proceeding to DEM load balancing (Thread Barrier)*

Phase II – DEM load balancing between d-Dimensional HHC single HHC node elements

1. *For Each Dimension in HHC Dimension – 1*
2. *Select the connected peer in the designated dimension*
3. *Get load information for MyPeer*
4. *If ((MyLoad – MyPeerLoad) >= 2)*
5. *Compute TotalLoad for both peers*
6. *Compute the AverageLoad load for both peers*
7. *Compute MyExcess load as following:*
8. *MyExcessLoad = MyLoad - AverageLoad;*
9. *Send MyExcessLoad to MyPeer*
10. *Wait for all processors to reach this state before going into the next iteration (Thread Barrier)*

2.3.3 Description of algorithm work

Similar to the previous two algorithms, the third algorithm balances the load on an HHC interconnection network using two main phases, phase one balances the load inside each one-dimensional HHC, and phase two balances the load between different dimensions of the HHC interconnection network. Phase one consists of two parts; part one balances load inside each triangle, then part two balances load between the two triangles. Part one consists of two steps, during the first step, the load information is exchanged between triangle members, in the second step, each node computes the total load of the three nodes, the average load for the three load and any excess load it has over average, it then starts enumerating its triangle peers and sends the needed load to any peer with load lower than average. Part two of the first phase balances the load between the two triangles in each one-dimensional HHC.

In Phase two of the algorithm, DEM load balancing is applied to distribute and load balance load across the hypercube dimensions. The following part presents a detailed description about the algorithm flow:

Phase 1: Load balancing within each one-dimensional HHC:

Part 1, balancing the load inside each of the two triangles

Part 1, Step 1: Exchange load information between a given triangles nodes:

Each node will poll load information from the other two partners in a given triangle.

Part 1, Step 2: Once the load of the other two partners is polled, each node calculates the average load for the three triangle members as following:

Average Load = Floor ((Total Load of Three Nodes + 1) / 3).

Adding the (1) should assist in splitting a load of (2) over (2) nodes rather than keeping it in one node.

Part 1, Step 3: Upon the calculation of the average weight, each node in a given triangle sends excess load to the other one or both partners if they both have less than average load.

Part 2, balancing the load between the two triangles:

Part 2, Step 1: Each node of both triangles exchanges its weight value with its corresponding (directly connected) node in the other triangle ([0-4], [1-5], [2-6]).

Part 2, Step 2: Each node computes the floor average and sends its excess load to the corresponding node.

Phase 2: Balancing the load between all dimensions of the D-Dimension HHC:

Apply DEM algorithm to the whole hypercube for d_h-*1* steps where d_h is the dimension of the HHC.

A: each pairs of nodes that share the same node address and differs only in the J^{th} bit of its sub-group address exchange weight along the dimension J, and calculate the average weight.

B: If the average weight is greater than the local load, calculate the excess load and send it to the neighbor node along dimension J, else receive the excess load from the neighbor node along dimension J.

2.3.3 Algorithm analytical evaluation:

A: *Execution time*: measures the required time to achieve the load balancing in the network, it represents the worst case time complexity for the algorithm to balance the load in an HHC interconnection network. Assuming M as the maximum load units at any node, the execution time is calculated as in table 7:

Table 7: Calculating Execution time for algorithm C

<table>
<tr><td>Phase 1
Balancing the load in each one-dimensional HHC within the HHC interconnection network

The total load moved during phase 1 is:
= 2M/3 + M/6
= 4M/6 + M/6
= 5M/6

Eventually, each node in the same HHC will end up with M/6 of load.</td><td>Part 1
Balancing the load inside each triangle.

Maximum of M/3 load units are sent from the overloaded node to each of the other two nodes.

= 2 x M/3 load units are sent in total
= 2M/3</td></tr>
<tr><td></td><td>Part 2
Balancing the load between the two triangles
Each node in the first triangle already has M/3. To split that load with the other triangle, each node sends M/6 to its corresponding node in the other triangle in parallel.
Maximum of = M/6</td></tr>
<tr><td colspan="2">Phase 2
DEM algorithm is used.
For all dimensions(d_h) of HHC is:
in the first dimension (d_h+1) max of M/12
in the second dimension (d_h+2) max of M/24
in the third dimension (d_h+2) max of M/48
And so on…
M/12 + M/24 + M/48 + …
= M/12 * ($1/2^0 + 1/2^1 + 1/2^2 + 1/2^3 + \ldots$)
= (M/12) * $\sum_{i=0}^{d_h-2} 1/2^i$
= (M/12) * $\sum_{i=0}^{d_h-2} (1/2)^i$
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1 – (1/2))
= (M/12) * ((1- $(1/2)^{d_h-1}$) / (1/2))
= (M/12) * (2 * (1- $(1/2)^{d_h-1}$))
= (M/12) * ((2- 2 * $(1/2)^{d_h-1}$))
= ((M/12) *2) – ((M/12) * 2 * $(1/2)^{d_h-1}$))
= ((M/6) – ((M/6) * $(1/2)^{d_h-1}$))
= (M/6) * (1 – $(1/2)^{d_h-1}$))</td></tr>
<tr><td colspan="2">Sum for all phases = phase1 + phase2 = (5M/6) + (M/6) * (1 – $(1/2)^{d_h-1}$))
≈ O(5M/6 + M/6) = ***O*(M)**</td></tr>
</table>

B: *Load balancing accuracy*: The difference between maximum workload units at any node minus the minimum workload units at any other node, it is referred to as the error in balancing load.

In this algorithm there are three steps where error may accumulate;

Step 1 when balancing load inside each triangle.

Step 2 when balancing the load among the two triangles.

Step 3 when balancing the load among different dimensions.

Table 8 presents the three steps error.

Table 8: Calculating Load balancing accuracy for algorithm C		
Step1: balancing load inside each triangle	Maximum error = 1	Since the coordinator is adjusting the total load before dividing by three to ensure that maximum error is = 1.
Step2: balancing the load among the two triangles	Maximum error = 1	When an odd number of workload units is to be divided by two.
Step3: balancing the load among different dimensions	Maximum error = d_{h-1}	The DEM algorithm has d_{h-1} as the Max error when balancing load among different dimensions
Total error (e) <= $1 + 1 + d_{h-1} = 2 + d_{h-1} =$ **$1 + d_h$**		

C: *Number of communication steps*: Number of communications steps depicts the number steps required to finish load balancing. Table 9 below clarifies the communication detail of the algorithm, it calculate the maximum communication steps at any node and the total communication steps in the network.

Table 9: Calculating Number of communication steps for algorithm C: *Maximum communication steps at any node and the total communication steps in the network.*	
Phase1: balances the load inside each one dimensional HHC of the HHC interconnection network. Maximum communication steps at any node in one-dimensional HHC is: 20 + 9 = 29 steps. The sum of communication steps in all 1-dimensional HHC in the network is = $29 * 2^{dh-1}$	Part 1, Step1 Balancing the load inside each triangle: Each node sends 2 load info messages to its connected peers. 2 messages x 3 nodes = 6 communication steps.
	Part 1, Step2 When the data is place in one node, we will need the following communications steps to distribute the node along the HHC: 4 communication steps in each triangle to exchange load info. 2 data re-poll steps in each triangle 2 data move steps in each triangle = 10 x 2 = 20 for the two triangles.
	Part2: balances load between nodes (three nodes) of the two triangles. 3 communication steps between any two pairs (2 send load info + 1 move data). 3 x 3 node pairs = 9 communication steps.

Phase2: balances load between different dimensions of the HHC interconnection network. To calculate the worst case scenario for algorithm C, we calculate the following: The algorithm will iterate for d_{h-1} number of times (Hypercube Dimensions). In worst case, each node will send its information to one peer, and send its workload to one peer. Therefore, Number of Communications steps = = Hypercube Dimension x Number HHC groups x Number of Nodes x 2 communications steps "1 workload info send + 1 workload move" $= d_{h-1} * 2 \wedge (d_{h-1}) * 6 * 2$ $= 12 * d_{h-1} * 2 \wedge (d_{h-1})$
Total Communications Steps = Single HHC Load Balancing + DEM Load Balancing $= (29 * 2^{dh-1)} + (12 * d_{h-1} * 2 \wedge (d_{h-1}))$

C: *Speed*: represents the speed at which the load balancing algorithm achieves its work, it is calculated by multiplying the number of communication steps by the speed of the connection links in the network. Assuming that the electrical connections in the HHC network have the speed of 250 Mb/s, the algorithm speed can be calculated as follows:

Speed = Number of communication steps * electrical link speed

Speed = $(29 * 2^{dh-1}) + (12 * d_{h-1} * 2 \wedge (d_{h-1})) * 250$ Mb/s

3. Experimental results and performance evaluation.

3.1 Objective:

This section is intended to show performance comparison between different algorithms experimentally in addition to make comparison between actual experimental speed results with analytical expected speed. Experiments are conducted using Intel Processor Core-i5-3210M of 2.5 GHz with 8 GB RAM running windows 64-bit Operating system. Implementation development language is Java with multi-threading for the algorithms A and B. all experiments are done according to the worst case of each algorithm. Note that implementation is built as a library that could be used to implement any network topology.

3.2 Execution time:

3.2.1 Execution time for algorithms A and B using different load sizes.

Execution time for algorithms A and B assuming the worst case of each algorithm is compared. This experiment was conducted over 96 processors (5-dimensional HHC) and 768 processors (8-dimensional HHC). Testing several maximum load units (varies between 10 and 100,000) while keeping constant number of processors. Experiment showed that algorithm B outperforms algorithm A in execution time. Figure 3 represent the results of the execution when number of processors is 96 with different number of load units for algorithm A and algorithm B respectively. Figure 4 represents the total execution time of in case of number of processors is 96 with different number of load units for algorithm A and algorithm B note that graph represents logarithmic graph of the results. Algorithm B consumes less time than Algorithm A, which is consistent with the analytical results.

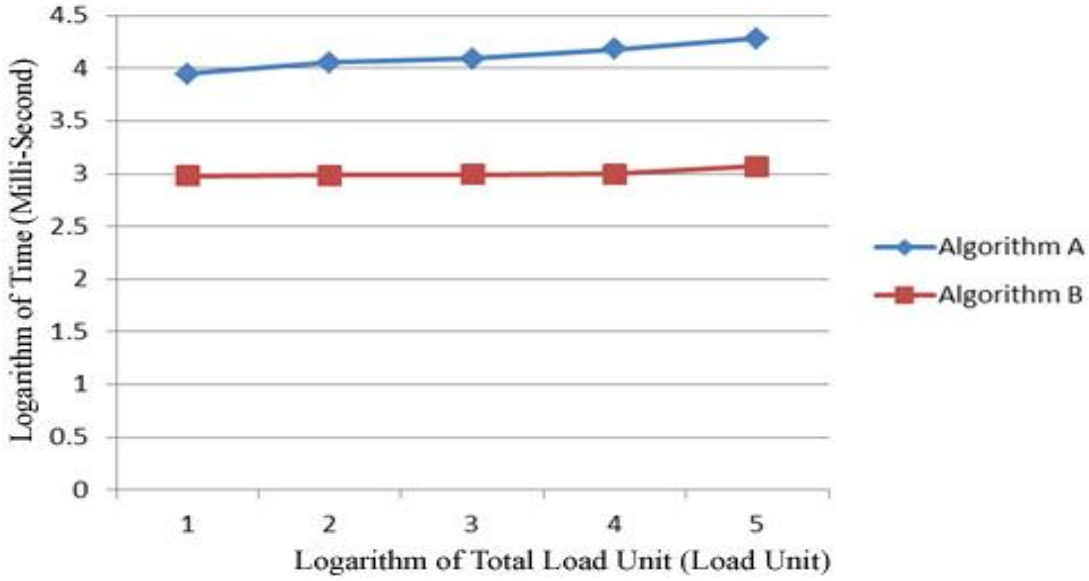

Figure 3: Execution time when number of processors is 96, different load sizes vary between 10 and 100000 for Algorithms (A, B).

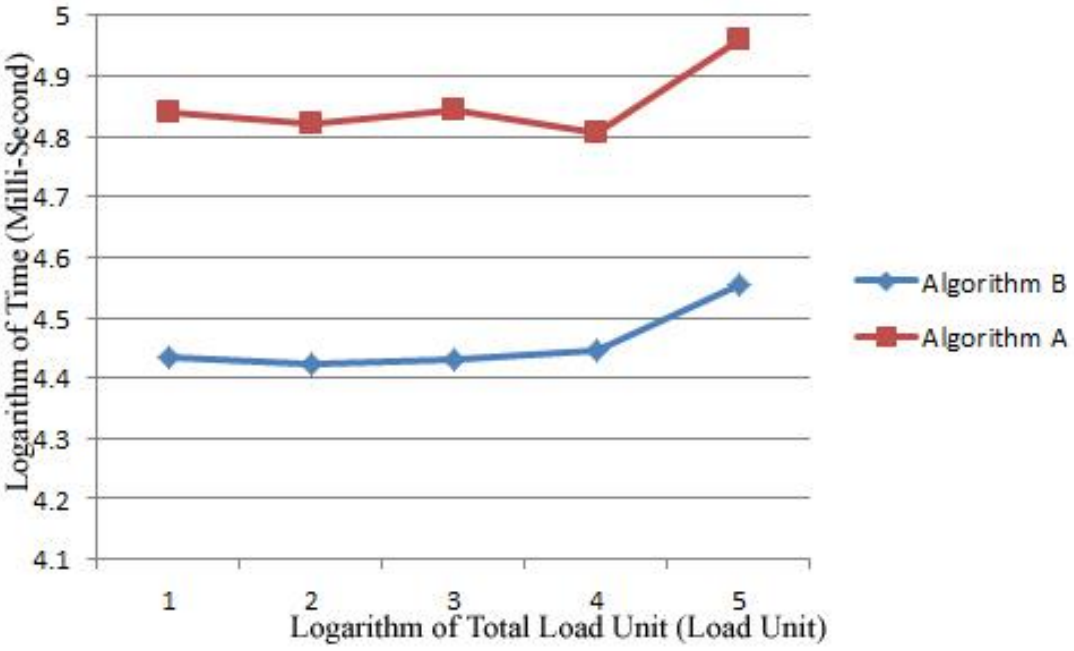

Figure 4: Execution time when number of processors is 768, different load sizes vary between 10 and 100000 for Algorithms (A, B).

3.2.2 Execution time for algorithms A and B using fixed load sizes.

In this experiment execution time is calculated for the two algorithms by testing constant maximum load units (500) across several HHC dimensions. The experiment always assumes worst case of the two algorithms in its execution. Figure 5 represents the result of executing this experiment using algorithms A and B. Results show that algorithm B outperforms algorithm A.

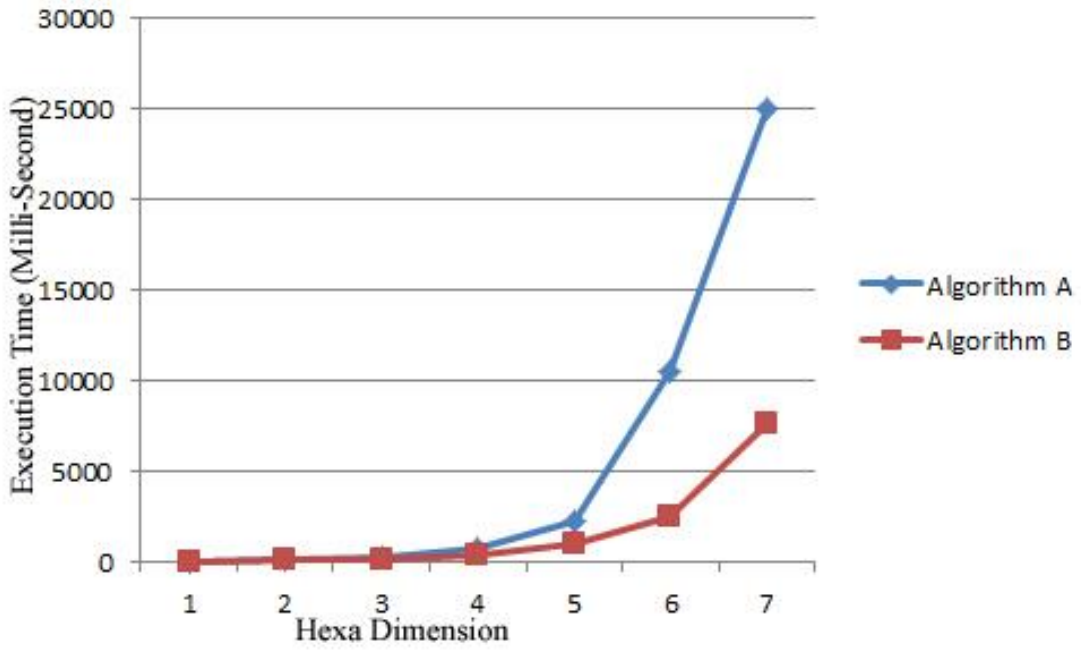

Figure 5: Execution Time of Max (500) workload units over verity of dimensions for Algorithms A, B.

3.3 speed:

In this experiment speed of the two algorithms A and B, is calculated in order to show that it is consistent with analytical expected speed, by counting the actual maximum number of steps in both algorithms and multiply the resultant number by electrical link speed (250Mb/s). Figure 6 and 7 represent the result of executing this experiment using A and algorithm B respectively with comparison to analytical results.

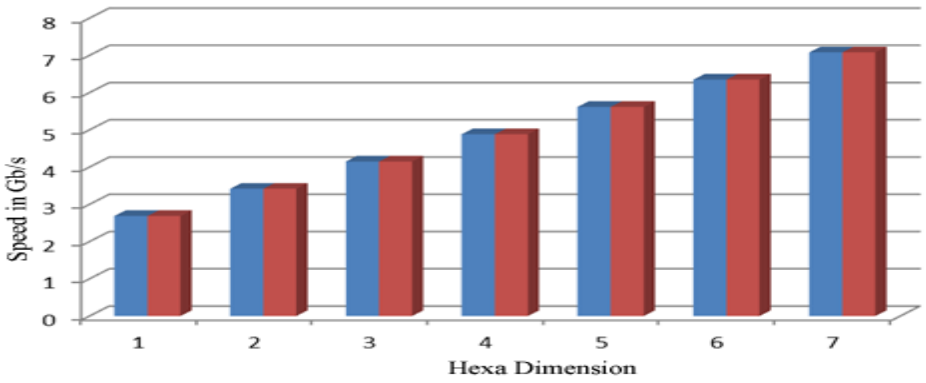

Figure 6: Experimental vs. Analytical Speed for Algorithm A

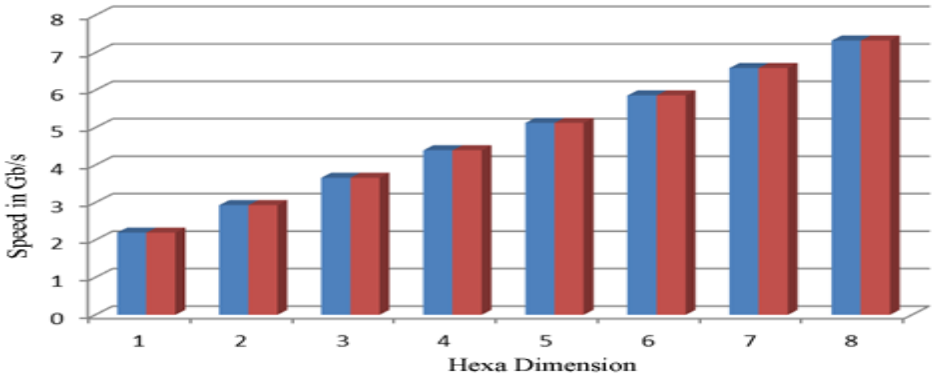

Figure 7: Experimental vs. Analytical Speed for Algorithm B.

3.4 Note:

As an observation of algorithm A execution time we conclude that it consumes more extra execution time, which is a result of performing *iReceive* operation on coordinator node to receive weights from unknown source (could be either odd or even node). This operation is done in a waiting loop that consumes extra time instead of thread wait operation that would not consume processor time.

3.5 Experimentations for Algorithm C

The third algorithm was simulated using (MinGW) GNU C version 4.7.2 implementation and pthreads library. The simulation was run on a single machine with the following specifications:

- Intel Core i7 2670 QM CPU at 2.20 GHz 2.20 GHz.
- 6 GB RAM.
- Windows 7 professional SP1 64 bit.

3.6 Execution time:

3.6.1 Execution time for algorithms C using different load sizes.

The algorithm was initiated with the worst case scenario with different load values starting from 10 to 100000 data units. The actual execution time was measured as following:

Actual execution speed = Program end time - Program start time

The following diagrams present the outcomes of the experimented algorithm.

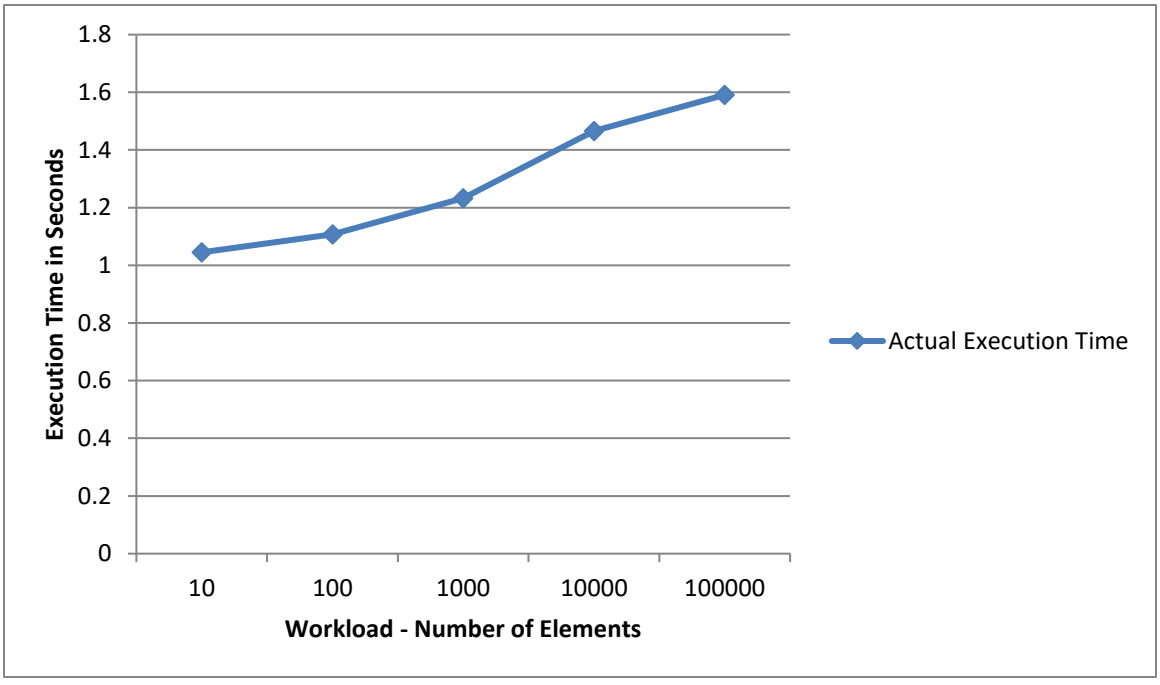

Figure 8: Execution time when the number of processors is 96, load sizes vary between 10 and 100000 for Algorithms (C).

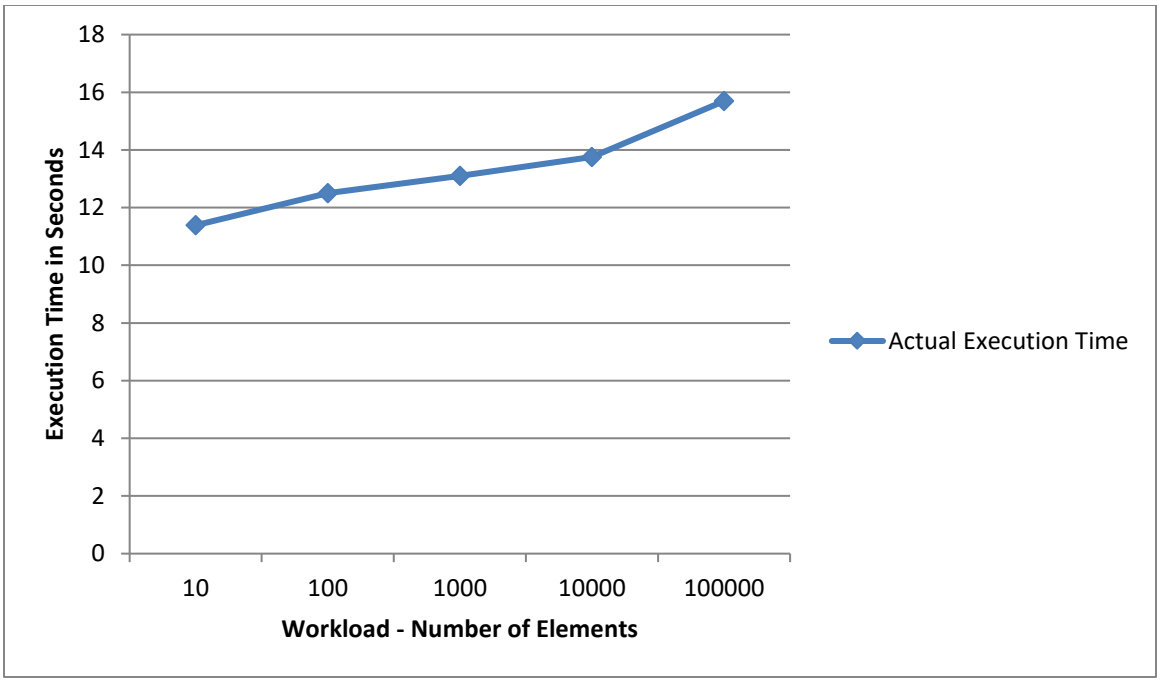

Figure 9: Execution time when the number of processors is 768, load sizes vary between 10 and 100000 for Algorithms (C).

3.6.2 Execution time for algorithm C using fixed load sizes.

In this experiment execution time is calculated for the algorithms C by testing constant maximum load units (500) across several HHC dimensions. The experiment always assumes worst case of the algorithm in its execution. Figure 10 represents the result of executing this experiment using algorithm C.

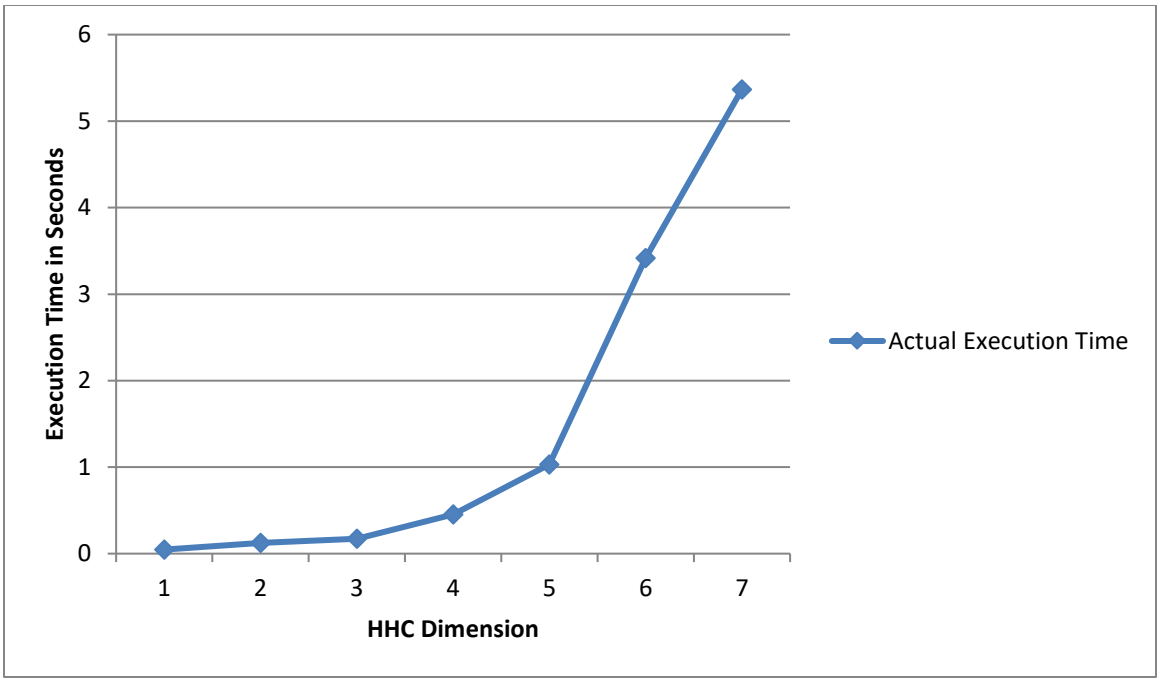

Figure 10: Execution Time for Max of (500) workload units over verity of dimensions for Algorithms (C).

3.7 speed:

In this experiment speed of algorithm C is calculated in order to show that it is consistent with analytical expected speed, by counting the actual maximum number performed by the algorithm and multiply the resultant number by electrical link speed (250Mb/s). Figure 11 represent the result of executing this experiment using C

.

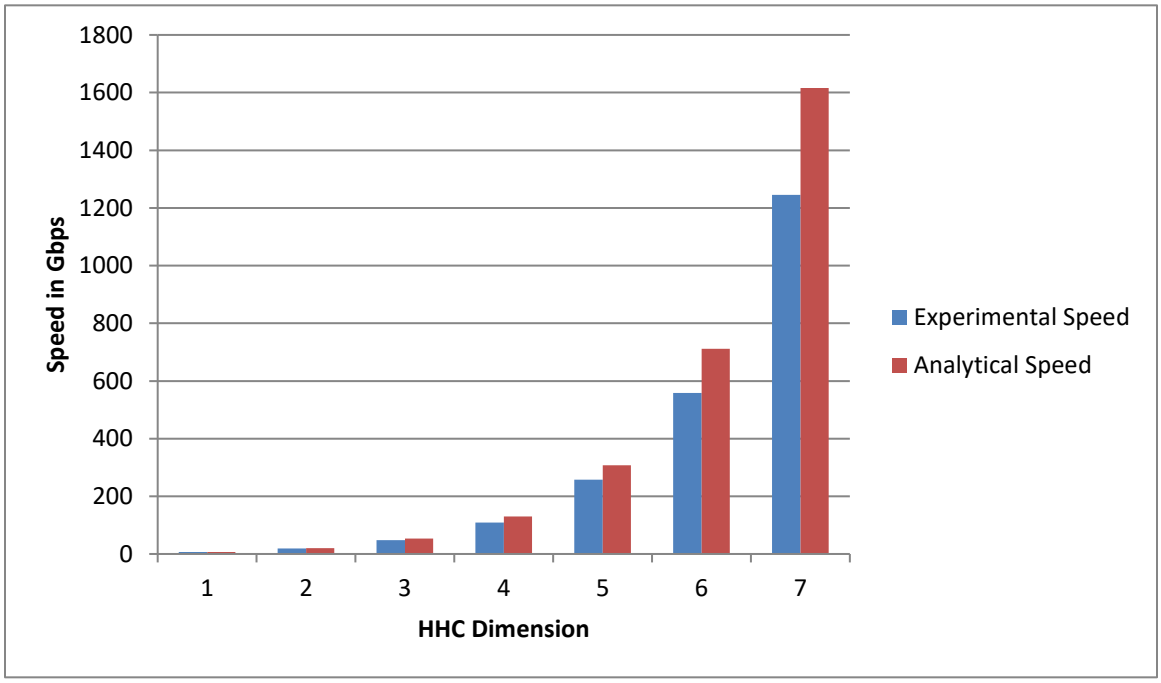

Figure 11: Experimental vs. Analytical Speed for Algorithm C

Figure (8) and figure (9) demonstrate that there is a direct linear relation between workload size and execution time, this is rational because more communication steps and data moves are needed when the workload increase, also, figure (10) clarifies that increasing the dimension of the HHC network, the execution time shall also increase due to the extra overhead that is required to transfer and balance data across more nodes in each dimension.

Finally, figure 11 demonstrates a comparison of speed between the practical experiment and the analytical model. Arrantly, a slight increase is noticed in the analytical model since it is based on the worst case scenario which is more severe than real life situations, for simplicity of calculations, the analytical model has accounted for worst case assumptions in data exchange steps and data move steps specifically in the DEM part of the load balancing algorithm. Doing that, we can easily calculate the upper limits of the algorithm which was relatively compatible with the practical findings albeit a little bit higher.

4. Summary and Conclusion.

We have introduced three load balancing algorithms for the HHC interconnection network, these algorithm share almost the same steps (two out of three steps) for achieving the goal of balancing load over the network. The algorithms differ only in its first step where the load is balanced - in parallel - at both triangles of the HHC network. We did the analytical analysis for each of the three algorithms to get some of the most important metrics. These metrics include the execution time, maximum error, maximum communication steps at any single node, total communication step, and speed. The analytical analysis showed that algorithm B has the best metrics over the other two algorithms.

We also performed experimental evaluation for the three algorithms to investigate their efficiency. The comparison between the three algorithms results proved that analytical results and showed that algorithm B has superior efficiency over the other two algorithms.

References

[1] Mahafzah, Basel A., et al. "The OTIS hyper hexa-cell optoelectronic architecture." *Computing* 94.5 (2012): 411-432.

[2] Shah, Vatsal, and Viral Kapadia. "Load Balancing by Process Migration in Distributed Operating System." *International Journal of Soft Computing and Engineering (IJSCE)* 2.1 (2012).

[3] Grama, Anath, ed. *Introduction to parallel computing*. Pearson Education, 2003.

[4] Bari, Pranit H., and B. B. Meshram. "Load Balancing In Distributed Computing." *Journal of Engineering Computers & Applied Sciences* 2.6 (2013): 43-51.

[5] Tanenbaum, Andrew S., and Maarten Van Steen. *Distributed systems*. Vol. 2. Prentice Hall, 2002.

[6] Ranka S, Won Y, Sahni S (1998) Programming a hypercube multicomputer. IEEE Softw 5(5):69–77

[7] Kibar O,Marchand P, Esener S (1998) High speed CMOS switch designs for free-space optoelectronic MINs. IEEE Trans Very Large Scale Integr (VLSI) Syst 6(3):372–386

[8] Basel A. Mahafzah · Bashira A. Jaradat, The load balancing problem in OTIS-Hypercube interconnection networks, J Supercomput (2008) 46: 276–297

[9] Willebeek-LeMairM, Reeves A (1993) Strategies for dynamic load balancing on highly parallel computers. IEEE Trans Parallel Distrib Syst 4(9):979–993